PostgreSQL 9.2.24 手册

PostgreSQL 全球开发组

翻译：冯若航[1]，Pigsty 项目组[2]

[1] https://vonng.com
[2] https://pigsty.cc

PostgreSQL 9.2.24 手册

PostgreSQL 全球开发组
翻译：冯若航[1]，Pigsty 项目组[2]
版权 © 1996-2017 PostgreSQL 全球开发组

法律声明

PostgreSQL 版权所有 © 1996-2017 PostgreSQL 全球开发组。

Postgres95 版权所有 © 1994-5 加利福尼亚大学董事会。

特此授权，可为任何目的使用、复制、修改和分发本软件及其文档，无需付费，也无需书面协议，
但条件是所有副本中都必须包含上述版权声明、本段以及后续两段内容。

在任何情况下，加利福尼亚大学都不对任何一方承担因使用本软件及其文档而产生的任何直接、间接、特殊、
附带或后果性损害赔偿责任，包括利润损失，即使加利福尼亚大学已被告知此类损害发生的可能性。

加利福尼亚大学明确不作任何担保，包括但不限于适销性和特定用途适用性的默示担保。本协议项下提供的软件按“原样”提供，
加利福尼亚大学没有义务提供维护、支持、更新、增强或修改。

[1] https://vonng.com
[2] https://pigsty.cc

目录

前言 xxxi
1. 什么是PostgreSQL? xxxi
2. PostgreSQL简史 xxxi
2.1. 伯克利 POSTGRES 项目 xxxii
2.2. Postgres95 xxxii
2.3. PostgreSQL xxxiii
3. 约定 xxxiii
4. 更多信息 xxxiii
5. 缺陷报告指南 xxxiv
5.1. 识别缺陷 xxxiv
5.2. 报告哪些内容 xxxiv
5.3. 到哪里报告缺陷 xxxvi
I. 教程 1
1. 从头开始 3
1.1. 安装 3
1.2. 架构基础 3
1.3. 创建数据库 3
1.4. 访问数据库 5
2. SQL语言 7
2.1. 简介 7
2.2. 概念 7
2.3. 创建一个新表 7
2.4. 向表中填充行 8
2.5. 查询表 9
2.6. 表之间的连接 10
2.7. 聚合函数 12
2.8. 更新 13
2.9. 删除 14
3. 高级特性 15
3.1. 简介 15
3.2. 视图 15
3.3. 外键 15
3.4. 事务 16
3.5. 窗口函数 17
3.6. 继承 20
3.7. 小结 21
II. SQL 语言 22
4. SQL语法 29
4.1. 词法结构 29
4.2. 值表达式 36
4.3. 调用函数 47
5. 数据定义 49
5.1. 表基础 49
5.2. 默认值 50
5.3. 约束 51
5.4. 系统列 57
5.5. 修改表 58
5.6. 权限 60
5.7. 模式 61
5.8. 继承 65
5.9. 分区 68
5.10. 外部数据 74
5.11. 其他数据库对象 75
5.12. 依赖跟踪 75
6. 数据操纵 77

6.1. 插入数据 77
6.2. 更新数据 78
6.3. 删除数据 78
6.4. 从被修改的行中返回数据 79
7. 查询 80
7.1. 概述 80
7.2. 表表达式 80
7.3. 选择列表 90
7.4. 组合查询 91
7.5. 行排序 92
7.6. LIMIT和OFFSET 93
7.7. VALUES列表 93
7.8. WITH查询（公共表表达式） 94
8. 数据类型 99
8.1. 数字类型 100
8.2. 货币类型 104
8.3. 字符类型 104
8.4. 二进制数据类型 106
8.5. 日期/时间类型 108
8.6. 布尔类型 116
8.7. 枚举类型 117
8.8. 几何类型 119
8.9. 网络地址类型 120
8.10. 位串类型 122
8.11. 文本搜索类型 123
8.12. UUID类型 125
8.13. XML类型 126
8.14. 数组 127
8.15. 复合类型 135
8.16. 范围类型 140
8.17. JSON 类型 145
8.18. 对象标识符类型 145
8.19. 伪类型 147
9. 函数和操作符 148
9.1. 逻辑操作符 148
9.2. 比较操作符 148
9.3. 数学函数和操作符 150
9.4. 字符串函数和操作符 154
9.5. 二进制串函数和操作符 166
9.6. 位串函数和操作符 168
9.7. 模式匹配 169
9.8. 数据类型格式化函数 182
9.9. 日期/时间函数和操作符 188
9.10. 枚举支持函数 199
9.11. 几何函数和操作符 200
9.12. 网络地址函数和操作符 203
9.13. 文本搜索函数和操作符 206
9.14. XML 函数 210
9.15. JSON 函数 219
9.16. 序列操作函数 220
9.17. 条件表达式 222
9.18. 数组函数和操作符 224
9.19. 范围函数和操作符 227
9.20. 聚合函数 229
9.21. 窗口函数 231
9.22. 子查询表达式 233
9.23. 行和数组比较 235
9.24. 集合返回函数 238

9.25. 系统信息函数 ... 240
9.26. 系统管理函数 ... 249
9.27. 触发器函数 ... 257
10. 类型转换 ... 259
10.1. 概述 ... 259
10.2. 操作符 ... 260
10.3. 函数 ... 263
10.4. 值存储 ... 266
10.5. UNION、CASE及相关结构 ... 267
11. 索引 ... 269
11.1. 简介 ... 269
11.2. 索引类型 ... 270
11.3. 多列索引 ... 271
11.4. 索引和ORDER BY ... 272
11.5. 组合多个索引 ... 273
11.6. 唯一索引 ... 273
11.7. 表达式索引 ... 274
11.8. 部分索引 ... 274
11.9. 操作符类和操作符族 ... 276
11.10. 索引和排序规则 ... 277
11.11. 检查索引使用情况 ... 278
12. 全文搜索 ... 280
12.1. 介绍 ... 280
12.2. 表和索引 ... 283
12.3. 控制文本搜索 ... 284
12.4. 附加特性 ... 289
12.5. 解析器 ... 294
12.6. 词典 ... 295
12.7. 配置示例 ... 303
12.8. 测试和调试文本搜索 ... 304
12.9. GiST 和 GIN 索引类型 ... 308
12.10. psql 支持 ... 309
12.11. 限制 ... 312
12.12. 从 8.3 之前的文本搜索迁移 ... 312
13. 并发控制 ... 314
13.1. 介绍 ... 314
13.2. 事务隔离 ... 314
13.3. 显式锁定 ... 318
13.4. 应用级别的数据一致性检查 ... 322
13.5. 注意事项 ... 324
13.6. 锁定和索引 ... 324
14. 性能提示 ... 325
14.1. 使用EXPLAIN ... 325
14.2. 规划器使用的统计信息 ... 335
14.3. 用显式JOIN子句控制规划器 ... 336
14.4. 填充一个数据库 ... 338
14.5. 非持久设置 ... 340
III. 服务器管理 ... 341
15. 从源代码安装 ... 347
15.1. 简要说明 ... 347
15.2. 需求 ... 347
15.3. 获取源代码 ... 349
15.4. 安装过程 ... 349
15.5. 安装后设置 ... 358
15.6. 受支持的平台 ... 360
15.7. 平台相关说明 ... 360
16. 在 Windows 上从源代码安装 ... 369
16.1. 使用 Visual C++ 或 Microsoft Windows SDK 构建 ... 369

16.2. 使用Visual C++或Borland C++构建libpq ... 373
17. 服务器设置和操作 ... 374
17.1. PostgreSQL用户账户 ... 374
17.2. 创建一个数据库集簇 ... 374
17.3. 启动数据库服务器 ... 375
17.4. 管理内核资源 ... 378
17.5. 关闭服务器 ... 385
17.6. 升级 PostgreSQL 集簇 ... 385
17.7. 防止服务器欺骗 ... 388
17.8. 加密选项 ... 388
17.9. 使用 SSL 的安全 TCP/IP 连接 ... 389
17.10. 通过 SSH 隧道建立安全 TCP/IP 连接 ... 391
17.11. 在Windows上注册事件日志 ... 392
18. 服务器配置 ... 393
18.1. 设置参数 ... 393
18.2. 文件位置 ... 394
18.3. 连接和认证 ... 395
18.4. 资源消耗 ... 399
18.5. 预写式日志 ... 403
18.6. 复制 ... 407
18.7. 查询规划 ... 409
18.8. 错误报告和日志 ... 413
18.9. 运行时统计数据 ... 422
18.10. 自动清理 ... 423
18.11. 客户端连接默认值 ... 424
18.12. 锁管理 ... 429
18.13. 版本和平台兼容性 ... 430
18.14. 错误处理 ... 432
18.15. 预置选项 ... 432
18.16. 自定义选项 ... 433
18.17. 开发者选项 ... 433
18.18. 短选项 ... 436
19. 客户端认证 ... 437
19.1. `pg_hba.conf` 文件 ... 437
19.2. 用户名映射 ... 443
19.3. 认证方法 ... 444
19.4. 认证问题 ... 451
20. 数据库角色 ... 452
20.1. 数据库角色 ... 452
20.2. 角色属性 ... 453
20.3. 角色成员资格 ... 454
20.4. 删除角色 ... 455
20.5. 函数和触发器安全性 ... 456
21. 管理数据库 ... 457
21.1. 概述 ... 457
21.2. 创建数据库 ... 457
21.3. 模板数据库 ... 458
21.4. 数据库配置 ... 459
21.5. 删除数据库 ... 459
21.6. 表空间 ... 460
22. 本地化 ... 462
22.1. 区域设置支持 ... 462
22.2. 排序规则支持 ... 464
22.3. 字符集支持 ... 466
23. 日常数据库维护任务 ... 472
23.1. 日常清理 ... 472
23.2. 日常重建索引 ... 477
23.3. 日志文件维护 ... 478

24. 备份和恢复 479
24.1. SQL转储 479
24.2. 文件系统级备份 481
24.3. 持续归档和时间点恢复（PITR） 482
25. 高可用、负载均衡和复制 491
25.1. 不同方案的比较 491
25.2. 日志传送备库 493
25.3. 故障切换 499
25.4. 日志传送的另一种方法 499
25.5. 热备 501
26. 恢复配置 508
26.1. 归档恢复设置 508
26.2. 恢复目标设置 509
26.3. 备库设置 509
27. 监控数据库活动 511
27.1. 标准 Unix 工具 511
27.2. 统计收集器 512
27.3. 查看锁 522
27.4. 动态追踪 522
28. 监控磁盘使用情况 530
28.1. 确定磁盘使用情况 530
28.2. 磁盘已满故障 531
29. 可靠性与预写式日志 532
29.1. 可靠性 532
29.2. 预写式日志（WAL） 533
29.3. 异步提交 534
29.4. WAL 配置 535
29.5. WAL 内部机制 536
30. 回归测试 538
30.1. 运行测试 538
30.2. 测试结果评估 540
30.3. 变体比较文件 542
30.4. 测试覆盖率检查 543
IV. 客户端接口 544
31. libpq - C 库 549
31.1. 数据库连接控制函数 549
31.2. 连接状态函数 559
31.3. 命令执行函数 563
31.4. 异步命令处理 576
31.5. 逐行检索查询结果 579
31.6. 取消进行中的查询 580
31.7. 快速路径接口 581
31.8. 异步通知 582
31.9. COPY命令相关的函数 582
31.10. 控制函数 586
31.11. 杂项函数 587
31.12. 通知处理 589
31.13. 事件系统 590
31.14. 环境变量 596
31.15. 密码文件 598
31.16. 连接服务文件 598
31.17. 连接参数的 LDAP 查找 598
31.18. SSL 支持 599
31.19. 在线程化程序中的行为 602
31.20. 编译 libpq 程序 603
31.21. 示例程序 604
32. 大对象 614
32.1. 简介 614

32.2. 实现特性 ... 614
32.3. 客户端接口 ... 614
32.4. 服务器端函数 ... 617
32.5. 示例程序 ... 617
33. ECPG - C 中的嵌入式 SQL ... 623
33.1. 概念 ... 623
33.2. 管理数据库连接 ... 623
33.3. 运行 SQL 命令 ... 626
33.4. 使用主变量 ... 628
33.5. 动态 SQL ... 640
33.6. pgtypes 库 ... 642
33.7. 使用描述符区域 ... 654
33.8. 错误处理 ... 666
33.9. 预处理器指令 ... 672
33.10. 处理嵌入式 SQL 程序 ... 674
33.11. 库函数 ... 675
33.12. 大对象 ... 675
33.13. C++ 应用 ... 677
33.14. 嵌入式 SQL 命令 ... 680
33.15. Informix兼容模式 ... 701
33.16. 内部 ... 715
34. 信息模式 ... 717
34.1. 模式 ... 717
34.2. 数据类型 ... 717
34.3. information_schema_catalog_name ... 718
34.4. administrable_role_authorizations ... 718
34.5. applicable_roles ... 718
34.6. attributes ... 719
34.7. character_sets ... 721
34.8. check_constraint_routine_usage ... 722
34.9. check_constraints ... 722
34.10. collations ... 722
34.11. collation_character_set_applicability ... 723
34.12. column_domain_usage ... 723
34.13. column_options ... 723
34.14. column_privileges ... 724
34.15. column_udt_usage ... 724
34.16. columns ... 725
34.17. constraint_column_usage ... 727
34.18. constraint_table_usage ... 728
34.19. data_type_privileges ... 728
34.20. domain_constraints ... 729
34.21. domain_udt_usage ... 729
34.22. domains ... 730
34.23. element_types ... 731
34.24. enabled_roles ... 733
34.25. foreign_data_wrapper_options ... 734
34.26. foreign_data_wrappers ... 734
34.27. foreign_server_options ... 734
34.28. foreign_servers ... 734
34.29. foreign_table_options ... 735
34.30. foreign_tables ... 735
34.31. key_column_usage ... 735
34.32. parameters ... 736
34.33. referential_constraints ... 738
34.34. role_column_grants ... 738
34.35. role_routine_grants ... 739

34.36. role_table_grants ... 739
34.37. role_udt_grants ... 740
34.38. role_usage_grants ... 740
34.39. routine_privileges ... 741
34.40. routines ... 741
34.41. schemata ... 745
34.42. sequences ... 745
34.43. sql_features ... 746
34.44. sql_implementation_info ... 746
34.45. sql_languages ... 747
34.46. sql_packages ... 747
34.47. sql_parts ... 748
34.48. sql_sizing ... 748
34.49. sql_sizing_profiles ... 748
34.50. table_constraints ... 749
34.51. table_privileges ... 749
34.52. tables ... 750
34.53. triggered_update_columns ... 750
34.54. triggers ... 751
34.55. udt_privileges ... 752
34.56. usage_privileges ... 752
34.57. user_defined_types ... 753
34.58. user_mapping_options ... 754
34.59. user_mappings ... 754
34.60. view_column_usage ... 755
34.61. view_routine_usage ... 755
34.62. view_table_usage ... 756
34.63. views ... 756
V. 服务器编程 ... 758
35. 扩展 SQL ... 763
35.1. 可扩展性如何运作 ... 763
35.2. PostgreSQL 类型系统 ... 763
35.3. User-defined Functions ... 764
35.4. Query Language (SQL) Functions ... 765
35.5. Function Overloading ... 777
35.6. Function Volatility Categories ... 778
35.7. Procedural Language Functions ... 780
35.8. Internal Functions ... 780
35.9. C-Language Functions ... 780
35.10. User-defined Aggregates ... 803
35.11. 用户定义的类型 ... 806
35.12. 用户定义的操作符 ... 809
35.13. 操作符优化信息 ... 809
35.14. 索引扩展接口 ... 813
35.15. 将相关对象打包成扩展 ... 823
35.16. 扩展构建基础设施 ... 828
36. 触发器 ... 831
36.1. 触发器行为概述 ... 831
36.2. 数据更改的可见性 ... 832
36.3. 用 C 编写触发器函数 ... 833
36.4. 一个完整的触发器示例 ... 835
37. 规则系统 ... 839
37.1. 查询树 ... 839
37.2. 视图和规则系统 ... 840
37.3. INSERT、UPDATE和DELETE规则 ... 847
37.4. 规则和权限 ... 856
37.5. 规则和命令状态 ... 858

37.6. 规则与触发器 ... 858
38. 过程语言 ... 861
38.1. 安装过程语言 ... 861
39. PL/pgSQL – SQL 过程语言 ... 863
39.1. 概述 ... 863
39.2. PL/pgSQL的结构 ... 864
39.3. 声明 ... 865
39.4. 表达式 ... 870
39.5. 基本语句 ... 871
39.6. 控制结构 ... 877
39.7. 游标 ... 888
39.8. 错误和消息 ... 893
39.9. 触发器函数 ... 894
39.10. PL/pgSQL 内部机制 ... 901
39.11. PL/pgSQL开发提示 ... 904
39.12. 从Oracle PL/SQL 移植 ... 906
40. PL/Tcl - Tcl 过程语言 ... 916
40.1. 概述 ... 916
40.2. PL/Tcl 函数和参数 ... 916
40.3. PL/Tcl 中的数据值 ... 917
40.4. PL/Tcl 中的全局数据 ... 917
40.5. 从 PL/Tcl 访问数据库 ... 918
40.6. PL/Tcl 中的触发器函数 ... 920
40.7. 模块与`unknown`命令 ... 921
40.8. Tcl 过程名 ... 922
41. PL/Perl - Perl 过程语言 ... 923
41.1. PL/Perl 函数和参数 ... 923
41.2. PL/Perl 中的数据值 ... 926
41.3. 内置函数 ... 927
41.4. PL/Perl 中的全局值 ... 931
41.5. 受信任与不受信任的 PL/Perl ... 932
41.6. PL/Perl 触发器 ... 933
41.7. PL/Perl 内部机制 ... 934
42. PL/Python - Python 过程语言 ... 936
42.1. Python 2 与 Python 3 ... 936
42.2. PL/Python 函数 ... 937
42.3. 数据值 ... 938
42.4. 共享数据 ... 942
42.5. 匿名代码块 ... 942
42.6. 触发器函数 ... 943
42.7. 数据库访问 ... 943
42.8. 显式子事务 ... 947
42.9. 辅助函数 ... 948
42.10. 环境变量 ... 949
43. 服务器编程接口 ... 950
43.1. 接口函数 ... 950
43.2. 接口支持函数 ... 982
43.3. 内存管理 ... 990
43.4. 数据更改的可见性 ... 999
43.5. 示例 ... 999
VI. Reference ... 1003
I. SQL Commands ... 1008
ABORT ... 1011
ALTER AGGREGATE ... 1012
ALTER COLLATION ... 1014
ALTER CONVERSION ... 1015
ALTER DATABASE ... 1016
ALTER DEFAULT PRIVILEGES ... 1018

ALTER DOMAIN 1021
ALTER EXTENSION 1024
ALTER FOREIGN DATA WRAPPER 1027
ALTER FOREIGN TABLE 1029
ALTER FUNCTION 1032
ALTER GROUP 1035
ALTER INDEX 1036
ALTER LANGUAGE 1038
ALTER LARGE OBJECT 1039
ALTER OPERATOR 1040
ALTER OPERATOR CLASS 1041
ALTER OPERATOR FAMILY 1042
ALTER ROLE 1046
ALTER SCHEMA 1049
ALTER SEQUENCE 1050
ALTER SERVER 1053
ALTER TABLE 1055
ALTER TABLESPACE 1066
ALTER TEXT SEARCH CONFIGURATION 1067
ALTER TEXT SEARCH DICTIONARY 1069
ALTER TEXT SEARCH PARSER 1071
ALTER TEXT SEARCH TEMPLATE 1072
ALTER TRIGGER 1073
ALTER TYPE 1074
ALTER USER 1077
ALTER USER MAPPING 1078
ALTER VIEW 1079
ANALYZE 1081
BEGIN 1083
CHECKPOINT 1085
CLOSE 1086
CLUSTER 1087
COMMENT 1089
COMMIT 1093
COMMIT PREPARED 1094
COPY 1095
CREATE AGGREGATE 1104
CREATE CAST 1107
CREATE COLLATION 1111
CREATE CONVERSION 1113
CREATE DATABASE 1115
CREATE DOMAIN 1118
CREATE EXTENSION 1120
CREATE FOREIGN DATA WRAPPER 1122
CREATE FOREIGN TABLE 1124
CREATE FUNCTION 1126
CREATE GROUP 1133
CREATE INDEX 1134
CREATE LANGUAGE 1140
CREATE OPERATOR 1143
CREATE OPERATOR CLASS 1146
CREATE OPERATOR FAMILY 1149
CREATE ROLE 1150
CREATE RULE 1154
CREATE SCHEMA 1157
CREATE SEQUENCE 1159
CREATE SERVER 1162
CREATE TABLE 1164

CREATE TABLE AS 1179
CREATE TABLESPACE 1182
CREATE TEXT SEARCH CONFIGURATION 1184
CREATE TEXT SEARCH DICTIONARY 1185
CREATE TEXT SEARCH PARSER 1187
CREATE TEXT SEARCH TEMPLATE 1189
CREATE TRIGGER 1190
CREATE TYPE 1195
CREATE USER 1203
CREATE USER MAPPING 1204
CREATE VIEW 1205
DEALLOCATE 1208
DECLARE 1209
DELETE 1212
DISCARD 1215
DO 1216
DROP AGGREGATE 1217
DROP CAST 1218
DROP COLLATION 1219
DROP CONVERSION 1220
DROP DATABASE 1221
DROP DOMAIN 1222
DROP EXTENSION 1223
DROP FOREIGN DATA WRAPPER 1224
DROP FOREIGN TABLE 1225
DROP FUNCTION 1226
DROP GROUP 1228
DROP INDEX 1229
DROP LANGUAGE 1230
DROP OPERATOR 1231
DROP OPERATOR CLASS 1233
DROP OPERATOR FAMILY 1235
DROP OWNED 1236
DROP ROLE 1237
DROP RULE 1238
DROP SCHEMA 1239
DROP SEQUENCE 1240
DROP SERVER 1241
DROP TABLE 1242
DROP TABLESPACE 1243
DROP TEXT SEARCH CONFIGURATION 1244
DROP TEXT SEARCH DICTIONARY 1245
DROP TEXT SEARCH PARSER 1246
DROP TEXT SEARCH TEMPLATE 1247
DROP TRIGGER 1248
DROP TYPE 1249
DROP USER 1250
DROP USER MAPPING 1251
DROP VIEW 1252
END 1253
EXECUTE 1254
EXPLAIN 1255
FETCH 1260
GRANT 1264
INSERT 1271
LISTEN 1274
LOAD 1275
LOCK 1276

MOVE 1278
NOTIFY 1280
PREPARE 1282
PREPARE TRANSACTION 1284
REASSIGN OWNED 1286
REINDEX 1287
RELEASE SAVEPOINT 1289
RESET 1290
REVOKE 1291
ROLLBACK 1295
ROLLBACK PREPARED 1296
ROLLBACK TO SAVEPOINT 1297
SAVEPOINT 1299
SECURITY LABEL 1301
SELECT 1304
SELECT INTO 1322
SET 1324
SET CONSTRAINTS 1327
SET ROLE 1328
SET SESSION AUTHORIZATION 1330
SET TRANSACTION 1332
SHOW 1334
START TRANSACTION 1336
TRUNCATE 1337
UNLISTEN 1339
UPDATE 1340
VACUUM 1344
VALUES 1346
II. PostgreSQL Client Applications 1349
clusterdb 1350
createdb 1353
createlang 1356
createuser 1359
dropdb 1363
droplang 1366
dropuser 1369
ecpg 1371
pg_basebackup 1373
pg_config 1378
pg_dump 1381
pg_dumpall 1392
pg_receivexlog 1397
pg_restore 1400
psql 1408
reindexdb 1441
vacuumdb 1444
III. PostgreSQL Server Applications 1447
initdb 1448
pg_controldata 1451
pg_ctl 1452
pg_resetxlog 1457
postgres 1459
postmaster 1466
VII. 内部 1467
44. PostgreSQL 内部概述 1472
44.1. 查询的路径 1472
44.2. 连接是如何建立的 1472
44.3. 解析器阶段 1473

44.4. PostgreSQL 规则系统 1474
44.5. 规划器/优化器 1474
44.6. 执行器 1475
45. 系统目录 1477
45.1. 概述 1477
45.2. pg_aggregate 1478
45.3. pg_am 1479
45.4. pg_amop 1481
45.5. pg_amproc 1482
45.6. pg_attrdef 1482
45.7. pg_attribute 1482
45.8. pg_authid 1485
45.9. pg_auth_members 1486
45.10. pg_cast 1486
45.11. pg_class 1488
45.12. pg_constraint 1490
45.13. pg_collation 1493
45.14. pg_conversion 1493
45.15. pg_database 1494
45.16. pg_db_role_setting 1495
45.17. pg_default_acl 1496
45.18. pg_depend 1496
45.19. pg_description 1497
45.20. pg_enum 1498
45.21. pg_extension 1498
45.22. pg_foreign_data_wrapper 1499
45.23. pg_foreign_server 1500
45.24. pg_foreign_table 1500
45.25. pg_index 1500
45.26. pg_inherits 1502
45.27. pg_language 1503
45.28. pg_largeobject 1504
45.29. pg_largeobject_metadata 1504
45.30. pg_namespace 1505
45.31. pg_opclass 1505
45.32. pg_operator 1506
45.33. pg_opfamily 1506
45.34. pg_pltemplate 1507
45.35. pg_proc 1508
45.36. pg_range 1511
45.37. pg_rewrite 1511
45.38. pg_seclabel 1512
45.39. pg_shdepend 1513
45.40. pg_shdescription 1514
45.41. pg_shseclabel 1514
45.42. pg_statistic 1515
45.43. pg_tablespace 1516
45.44. pg_trigger 1517
45.45. pg_ts_config 1518
45.46. pg_ts_config_map 1519
45.47. pg_ts_dict 1519
45.48. pg_ts_parser 1520
45.49. pg_ts_template 1520
45.50. pg_type 1521
45.51. pg_user_mapping 1525
45.52. 系统视图 1525

45.53. pg_available_extensions 1526
45.54. pg_available_extension_versions 1527
45.55. pg_cursors 1527
45.56. pg_group 1528
45.57. pg_indexes 1528
45.58. pg_locks 1529
45.59. pg_prepared_statements 1531
45.60. pg_prepared_xacts 1532
45.61. pg_roles 1532
45.62. pg_rules 1533
45.63. pg_seclabels 1534
45.64. pg_settings 1534
45.65. pg_shadow 1536
45.66. pg_stats 1537
45.67. pg_tables 1538
45.68. pg_timezone_abbrevs 1539
45.69. pg_timezone_names 1539
45.70. pg_user 1540
45.71. pg_user_mappings 1540
45.72. pg_views 1541
46. Frontend/Backend Protocol 1542
46.1. Overview 1542
46.2. Message Flow 1544
46.3. Streaming Replication Protocol 1555
46.4. Message Data Types 1558
46.5. Message Formats 1559
46.6. Error and Notice Message Fields 1574
46.7. Summary of Changes since Protocol 2.0 1575
47. PostgreSQL 编码约定 1577
47.1. 格式 1577
47.2. 在服务器内部报告错误 1577
47.3. 错误消息风格指南 1579
48. 本地语言支持 1584
48.1. 给翻译者 1584
48.2. 给程序员 1586
49. 编写过程语言调用处理器 1589
50. Writing A Foreign Data Wrapper 1592
50.1. Foreign Data Wrapper Functions 1592
50.2. Foreign Data Wrapper Callback Routines 1592
50.3. Foreign Data Wrapper Helper Functions 1595
50.4. Foreign Data Wrapper Query Planning 1596
51. 遗传查询优化器 1598
51.1. 将查询处理看成是一个复杂的优化问题 1598
51.2. 遗传算法 1598
51.3. PostgreSQL 中的遗传查询优化（GEQO） 1599
51.4. 进一步阅读 1600
52. 索引访问方法接口定义 1601
52.1. 索引的目录条目 1601
52.2. 索引访问方法函数 1602
52.3. 索引扫描 1605
52.4. 索引锁定注意事项 1606
52.5. 索引唯一性检查 1607
52.6. 索引代价估算函数 1608
53. GiST 索引 1611
53.1. 简介 1611
53.2. 可扩展性 1611
53.3. 实现 1617

53.4. 示例 1618
54. SP-GiST 索引 1619
54.1. 简介 1619
54.2. 可扩展性 1619
54.3. 实现 1625
54.4. 示例 1626
55. GIN Indexes 1627
55.1. Introduction 1627
55.2. Extensibility 1627
55.3. Implementation 1629
55.4. GIN Tips and Tricks 1630
55.5. Limitations 1631
55.6. Examples 1631
56. Database Physical Storage 1633
56.1. Database File Layout 1633
56.2. TOAST 1635
56.3. Free Space Map 1637
56.4. Visibility Map 1637
56.5. The Initialization Fork 1637
56.6. Database Page Layout 1638
57. BKI 后端接口 1641
57.1. BKI 文件格式 1641
57.2. BKI 命令 1641
57.3. 引导 BKI 文件的结构 1642
57.4. 示例 1643
58. 规划器如何使用统计信息 1644
58.1. 行估计示例 1644
58.2. 规划器统计信息与安全 1648
VIII. 附录 1650
A. PostgreSQL错误代码 1670
B. 日期/时间支持 1678
B.1. 日期/时间输入解释 1678
B.2. 日期/时间关键字 1679
B.3. 日期/时间配置文件 1680
B.4. 历法的历史 1681
C. SQL 关键字 1683
D. SQL 符合性 1703
D.1. 支持的特性 1704
D.2. 不支持的特性 1717
E. 发布说明 1730
E.1. Release 9.2.24 1730
E.2. Release 9.2.23 1731
E.3. Release 9.2.22 1733
E.4. Release 9.2.21 1737
E.5. Release 9.2.20 1740
E.6. Release 9.2.19 1743
E.7. Release 9.2.18 1745
E.8. Release 9.2.17 1748
E.9. Release 9.2.16 1749
E.10. Release 9.2.15 1750
E.11. Release 9.2.14 1754
E.12. Release 9.2.13 1758
E.13. Release 9.2.12 1759
E.14. Release 9.2.11 1760
E.15. Release 9.2.10 1764
E.16. Release 9.2.9 1771
E.17. Release 9.2.8 1775
E.18. Release 9.2.7 1776

E.19. Release 9.2.6 1781
E.20. Release 9.2.5 1783
E.21. Release 9.2.4 1786
E.22. Release 9.2.3 1789
E.23. Release 9.2.2 1792
E.24. Release 9.2.1 1797
E.25. Release 9.2 1799
E.26. Release 9.1.24 1817
E.27. Release 9.1.23 1818
E.28. Release 9.1.22 1821
E.29. Release 9.1.21 1822
E.30. Release 9.1.20 1824
E.31. Release 9.1.19 1827
E.32. Release 9.1.18 1831
E.33. Release 9.1.17 1832
E.34. Release 9.1.16 1832
E.35. Release 9.1.15 1837
E.36. Release 9.1.14 1843
E.37. Release 9.1.13 1846
E.38. Release 9.1.12 1847
E.39. Release 9.1.11 1851
E.40. Release 9.1.10 1853
E.41. Release 9.1.9 1855
E.42. Release 9.1.8 1858
E.43. Release 9.1.7 1860
E.44. Release 9.1.6 1863
E.45. Release 9.1.5 1865
E.46. Release 9.1.4 1868
E.47. Release 9.1.3 1872
E.48. Release 9.1.2 1877
E.49. Release 9.1.1 1881
E.50. Release 9.1 1882
E.51. Release 9.0.23 1900
E.52. Release 9.0.22 1903
E.53. Release 9.0.21 1904
E.54. Release 9.0.20 1905
E.55. Release 9.0.19 1908
E.56. Release 9.0.18 1914
E.57. Release 9.0.17 1916
E.58. Release 9.0.16 1917
E.59. Release 9.0.15 1921
E.60. Release 9.0.14 1923
E.61. Release 9.0.13 1925
E.62. Release 9.0.12 1927
E.63. Release 9.0.11 1929
E.64. Release 9.0.10 1932
E.65. Release 9.0.9 1933
E.66. Release 9.0.8 1935
E.67. Release 9.0.7 1937
E.68. Release 9.0.6 1941
E.69. Release 9.0.5 1944
E.70. Release 9.0.4 1948
E.71. Release 9.0.3 1951
E.72. Release 9.0.2 1952
E.73. Release 9.0.1 1955
E.74. Release 9.0 1956
E.75. Release 8.4.22 1977
E.76. Release 8.4.21 1980

E.77. Release 8.4.20 1981
E.78. Release 8.4.19 1984
E.79. Release 8.4.18 1986
E.80. Release 8.4.17 1987
E.81. Release 8.4.16 1989
E.82. Release 8.4.15 1990
E.83. Release 8.4.14 1992
E.84. Release 8.4.13 1993
E.85. Release 8.4.12 1995
E.86. Release 8.4.11 1997
E.87. Release 8.4.10 2000
E.88. Release 8.4.9 2002
E.89. Release 8.4.8 2006
E.90. Release 8.4.7 2007
E.91. Release 8.4.6 2009
E.92. Release 8.4.5 2011
E.93. Release 8.4.4 2014
E.94. Release 8.4.3 2017
E.95. Release 8.4.2 2020
E.96. Release 8.4.1 2023
E.97. Release 8.4 2025
E.98. Release 8.3.23 2046
E.99. Release 8.3.22 2047
E.100. Release 8.3.21 2049
E.101. Release 8.3.20 2050
E.102. Release 8.3.19 2052
E.103. Release 8.3.18 2054
E.104. Release 8.3.17 2056
E.105. Release 8.3.16 2058
E.106. Release 8.3.15 2060
E.107. Release 8.3.14 2061
E.108. Release 8.3.13 2063
E.109. Release 8.3.12 2064
E.110. Release 8.3.11 2067
E.111. Release 8.3.10 2069
E.112. Release 8.3.9 2071
E.113. Release 8.3.8 2073
E.114. Release 8.3.7 2075
E.115. Release 8.3.6 2077
E.116. Release 8.3.5 2079
E.117. Release 8.3.4 2081
E.118. Release 8.3.3 2083
E.119. Release 8.3.2 2084
E.120. Release 8.3.1 2086
E.121. Release 8.3 2088
E.122. Release 8.2.23 2105
E.123. Release 8.2.22 2107
E.124. Release 8.2.21 2109
E.125. Release 8.2.20 2110
E.126. Release 8.2.19 2111
E.127. Release 8.2.18 2113
E.128. Release 8.2.17 2115
E.129. Release 8.2.16 2116
E.130. Release 8.2.15 2118
E.131. Release 8.2.14 2120
E.132. Release 8.2.13 2121
E.133. Release 8.2.12 2123
E.134. Release 8.2.11 2124

E.135. Release 8.2.10 2125
E.136. Release 8.2.9 2127
E.137. Release 8.2.8 2127
E.138. Release 8.2.7 2129
E.139. Release 8.2.6 2131
E.140. Release 8.2.5 2133
E.141. Release 8.2.4 2134
E.142. Release 8.2.3 2135
E.143. Release 8.2.2 2135
E.144. Release 8.2.1 2136
E.145. Release 8.2 2137
E.146. Release 8.1.23 2152
E.147. Release 8.1.22 2154
E.148. Release 8.1.21 2155
E.149. Release 8.1.20 2156
E.150. Release 8.1.19 2158
E.151. Release 8.1.18 2159
E.152. Release 8.1.17 2160
E.153. Release 8.1.16 2161
E.154. Release 8.1.15 2162
E.155. Release 8.1.14 2163
E.156. Release 8.1.13 2164
E.157. Release 8.1.12 2165
E.158. Release 8.1.11 2167
E.159. Release 8.1.10 2169
E.160. Release 8.1.9 2169
E.161. Release 8.1.8 2170
E.162. Release 8.1.7 2170
E.163. Release 8.1.6 2171
E.164. Release 8.1.5 2172
E.165. Release 8.1.4 2173
E.166. Release 8.1.3 2175
E.167. Release 8.1.2 2176
E.168. Release 8.1.1 2177
E.169. Release 8.1 2178
E.170. Release 8.0.26 2193
E.171. Release 8.0.25 2195
E.172. Release 8.0.24 2196
E.173. Release 8.0.23 2197
E.174. Release 8.0.22 2199
E.175. Release 8.0.21 2200
E.176. Release 8.0.20 2200
E.177. Release 8.0.19 2201
E.178. Release 8.0.18 2202
E.179. Release 8.0.17 2203
E.180. Release 8.0.16 2203
E.181. Release 8.0.15 2205
E.182. Release 8.0.14 2206
E.183. Release 8.0.13 2207
E.184. Release 8.0.12 2208
E.185. Release 8.0.11 2208
E.186. Release 8.0.10 2209
E.187. Release 8.0.9 2209
E.188. Release 8.0.8 2210
E.189. Release 8.0.7 2211
E.190. Release 8.0.6 2212
E.191. Release 8.0.5 2213
E.192. Release 8.0.4 2214

E.193. Release 8.0.3 2216
E.194. Release 8.0.2 2217
E.195. Release 8.0.1 2219
E.196. Release 8.0 2220
E.197. Release 7.4.30 2236
E.198. Release 7.4.29 2237
E.199. Release 7.4.28 2238
E.200. Release 7.4.27 2239
E.201. Release 7.4.26 2240
E.202. Release 7.4.25 2241
E.203. Release 7.4.24 2242
E.204. Release 7.4.23 2242
E.205. Release 7.4.22 2243
E.206. Release 7.4.21 2243
E.207. Release 7.4.20 2244
E.208. Release 7.4.19 2245
E.209. Release 7.4.18 2246
E.210. Release 7.4.17 2246
E.211. Release 7.4.16 2247
E.212. Release 7.4.15 2247
E.213. Release 7.4.14 2248
E.214. Release 7.4.13 2249
E.215. Release 7.4.12 2250
E.216. Release 7.4.11 2250
E.217. Release 7.4.10 2251
E.218. Release 7.4.9 2252
E.219. Release 7.4.8 2253
E.220. Release 7.4.7 2255
E.221. Release 7.4.6 2256
E.222. Release 7.4.5 2257
E.223. Release 7.4.4 2257
E.224. Release 7.4.3 2258
E.225. Release 7.4.2 2259
E.226. Release 7.4.1 2261
E.227. Release 7.4 2262
E.228. Release 7.3.21 2278
E.229. Release 7.3.20 2279
E.230. Release 7.3.19 2279
E.231. Release 7.3.18 2280
E.232. Release 7.3.17 2280
E.233. Release 7.3.16 2280
E.234. Release 7.3.15 2281
E.235. Release 7.3.14 2282
E.236. Release 7.3.13 2282
E.237. Release 7.3.12 2283
E.238. Release 7.3.11 2284
E.239. Release 7.3.10 2284
E.240. Release 7.3.9 2286
E.241. Release 7.3.8 2287
E.242. Release 7.3.7 2287
E.243. Release 7.3.6 2287
E.244. Release 7.3.5 2288
E.245. Release 7.3.4 2289
E.246. Release 7.3.3 2290
E.247. Release 7.3.2 2292
E.248. Release 7.3.1 2293
E.249. Release 7.3 2294
E.250. Release 7.2.8 2306

E.251. Release 7.2.7 2307
E.252. Release 7.2.6 2307
E.253. Release 7.2.5 2308
E.254. Release 7.2.4 2308
E.255. Release 7.2.3 2309
E.256. Release 7.2.2 2309
E.257. Release 7.2.1 2310
E.258. Release 7.2 2311
E.259. Release 7.1.3 2321
E.260. Release 7.1.2 2321
E.261. Release 7.1.1 2321
E.262. Release 7.1 2322
E.263. Release 7.0.3 2326
E.264. Release 7.0.2 2327
E.265. Release 7.0.1 2327
E.266. Release 7.0 2328
E.267. Release 6.5.3 2336
E.268. Release 6.5.2 2336
E.269. Release 6.5.1 2337
E.270. Release 6.5 2337
E.271. Release 6.4.2 2342
E.272. Release 6.4.1 2343
E.273. Release 6.4 2344
E.274. Release 6.3.2 2348
E.275. Release 6.3.1 2349
E.276. Release 6.3 2350
E.277. Release 6.2.1 2355
E.278. Release 6.2 2355
E.279. Release 6.1.1 2358
E.280. Release 6.1 2359
E.281. Release 6.0 2361
E.282. Release 1.09 2364
E.283. Release 1.02 2364
E.284. Release 1.01 2365
E.285. Release 1.0 2368
E.286. Postgres95 Release 0.03 2369
E.287. Postgres95 Release 0.02 2372
E.288. Postgres95 Release 0.01 2372
F. 额外提供的模块与扩展 2373
F.1. adminpack 2374
F.2. auth_delay 2376
F.3. auto_explain 2377
F.4. btree_gin 2380
F.5. btree_gist 2381
F.6. chkpass 2383
F.7. citext – 大小写不敏感的字符串类型 2384
F.8. cube – 多维立方体数据类型 2386
F.9. dblink 2389
F.10. dict_int – 用于整数的示例全文搜索词典 2421
F.11. dict_xsyn – 示例同义词全文检索词典 2422
F.12. dummy_seclabel 2424
F.13. earthdistance – 计算大圆距离 2425
F.14. file_fdw 2427
F.15. fuzzystrmatch – 确定字符串的相似性和距离 2429
F.16. hstore 2432
F.17. intagg – 整数聚合器和枚举器 2438
F.18. intarray 2440
F.19. isn 2443

F.20. lo – 管理大对象 2447
F.21. ltree 2448
F.22. pageinspect 2454
F.23. passwordcheck 2456
F.24. pg_buffercache 2457
F.25. pgcrypto 2459
F.26. pg_freespacemap 2470
F.27. pgrowlocks – 显示表的行锁信息 2472
F.28. pg_stat_statements 2473
F.29. pgstattuple – 获取元组级统计信息 2477
F.30. pg_trgm 2479
F.31. seg – 用于线段或浮点区间的数据类型 2482
F.32. sepgsql 2485
F.33. spi 2493
F.34. sslinfo 2495
F.35. tablefunc 2497
F.36. tcn 2506
F.37. test_parser 2507
F.38. tsearch2 2509
F.39. unaccent 2511
F.40. uuid-ossp 2513
F.41. xml2 2514
G. 额外提供的程序 2518
G.1. 客户端应用 2518
G.2. 服务器应用 2532
H. 外部项目 2548
H.1. 客户端接口 2548
H.2. 管理工具 2548
H.3. 过程语言 2548
H.4. 扩展 2549
I. 源代码仓库 2550
I.1. 通过 Git 获取源代码 2550
J. Documentation 2551
J.1. DocBook 2551
J.2. Tool Sets 2551
J.3. Building The Documentation 2556
J.4. Documentation Authoring 2559
J.5. Style Guide 2560
K. 缩略词 2563
参考书目 2569
索引 2571

插图清单

9.1. 转换 SQL/XML 输出到 HTML 的 XSLT 样式表 219
51.1. 遗传算法的结构图 1599

表格清单

4.1. 反斜线转义序列 31
4.2. 操作符优先级（递减） 36
8.1. 数据类型 99
8.2. 数字类型 100
8.3. 货币类型 104
8.4. 字符类型 104
8.5. 特殊字符类型 106
8.6. 二进制数据类型 106
8.7. `bytea`字面量中需要转义的字节 107
8.8. `bytea`输出转义字节 107
8.9. 日期/时间类型 108
8.10. 日期输入 110
8.11. 时间输入 110
8.12. 时区输入 111
8.13. 特殊日期/时间输入 112
8.14. 日期/时间输出风格 112
8.15. 日期顺序习惯 113
8.16. ISO 8601 间隔单位缩写 115
8.17. 间隔输入 115
8.18. 间隔输出风格示例 116
8.19. 布尔数据类型 116
8.20. 几何类型 119
8.21. 网络地址类型 121
8.22. `cidr`类型输入示例 121
8.23. 对象标识符类型 146
8.24. 伪类型 147
9.1. 比较操作符 148
9.2. 数学操作符 150
9.3. 数学函数 151
9.4. 随机函数 154
9.5. 三角函数 154
9.6. SQL字符串函数和操作符 155
9.7. 其他字符串函数 157
9.8. 内置转换 163
9.9. SQL二进制串函数和操作符 166
9.10. 其他二进制串函数 167
9.11. 位串操作符 169
9.12. 正则表达式匹配操作符 172
9.13. 正则表达式原子 175
9.14. 正则表达式量词 175
9.15. 正则表达式约束 176
9.16. 正则表达式字符输入转义 177
9.17. 正则表达式字符类简写转义 178
9.18. 正则表达式约束转义 178
9.19. 正则表达式反向引用 179
9.20. ARE 嵌入选项字母 179
9.21. 格式化函数 182
9.22. 用于日期/时间格式化的模板模式 183
9.23. 用于日期/时间格式化的模板模式修饰符 184
9.24. 用于数值格式化的模板模式 186
9.25. 用于数值格式化的模板模式修饰符 187
9.26. `to_char`示例 187
9.27. 日期/时间操作符 188
9.28. 日期/时间函数 189
9.29. `AT TIME ZONE` 变体 197

9.30. 枚举支持函数 199
9.31. 几何操作符 200
9.32. 几何函数 201
9.33. 几何类型转换函数 202
9.34. cidr 和 inet 操作符 203
9.35. cidr 和 inet 函数 204
9.36. macaddr 函数 206
9.37. 文本搜索操作符 206
9.38. 文本搜索函数 207
9.39. 文本搜索调试函数 209
9.40. JSON 支持函数 220
9.41. 序列函数 220
9.42. 数组操作符 224
9.43. 数组函数 225
9.44. 范围操作符 227
9.45. 范围函数 228
9.46. 通用聚合函数 229
9.47. 用于统计的聚合函数 231
9.48. 通用窗口函数 232
9.49. 序列生成函数 238
9.50. 下标生成函数 239
9.51. 会话信息函数 240
9.52. 访问权限查询函数 242
9.53. 模式可见性查询函数 244
9.54. 系统目录信息函数 245
9.55. 注释信息函数 248
9.56. 事务 ID 和快照 248
9.57. 快照组件 249
9.58. 配置设定函数 249
9.59. 服务器信号函数 250
9.60. 备份控制函数 250
9.61. 恢复信息函数 252
9.62. 恢复控制函数 252
9.63. 快照同步函数 253
9.64. 数据库对象尺寸函数 253
9.65. 数据库对象位置函数 255
9.66. 通用文件访问函数 255
9.67. 咨询锁函数 256
12.1. 默认解析器的词元类型 294
13.1. 标准 SQL 事务隔离级别 314
13.2. 冲突的锁模式 320
17.1. System V IPC参数 378
17.2. PostgreSQL 共享内存用量 383
17.3. SSL 服务器文件用法 390
18.1. 消息严重级别 417
18.2. 短选项对照 436
22.1. PostgreSQL 字符集 466
22.2. 客户端/服务器字符集转换 469
25.1. 高可用、负载均衡和复制特性矩阵 493
27.1. 标准统计视图 513
27.2. pg_stat_activity 视图 514
27.3. pg_stat_bgwriter 视图 515
27.4. pg_stat_database 视图 516
27.5. pg_stat_all_tables 视图 517
27.6. pg_stat_all_indexes 视图 517
27.7. pg_statio_all_tables 视图 518
27.8. pg_statio_all_indexes 视图 519

27.9. pg_statio_all_sequences 视图 519
27.10. pg_stat_user_functions 视图 519
27.11. pg_stat_replication 视图 519
27.12. pg_stat_database_conflicts 视图 520
27.13. 附加统计函数 521
27.14. 按后端统计函数 521
27.15. 内置 DTrace 探针 523
27.16. 探针参数中使用的已定义类型 527
31.1. SSL 模式描述 601
31.2. libpq/客户端 SSL 文件用法 601
33.1. 在 PostgreSQL 数据类型和 C 变量类型之间映射 630
33.2. PGTYPESdate_from_asc的合法输入格式 645
33.3. PGTYPESdate_fmt_asc的合法输入格式 647
33.4. rdefmtdate的合法输入格式 648
33.5. PGTYPEStimestamp_from_asc的合法输入格式 649
34.1. information_schema_catalog_name 列 718
34.2. administrable_role_authorizations 列 718
34.3. applicable_roles 列 718
34.4. attributes 列 719
34.5. character_sets 列 721
34.6. check_constraint_routine_usage 列 722
34.7. check_constraints 列 722
34.8. collations 列 722
34.9. collation_character_set_applicability 列 723
34.10. column_domain_usage 列 723
34.11. column_options 列 723
34.12. column_privileges 列 724
34.13. column_udt_usage 列 724
34.14. columns 列 725
34.15. constraint_column_usage 列 728
34.16. constraint_table_usage 列 728
34.17. data_type_privileges 列 728
34.18. domain_constraints 列 729
34.19. domain_udt_usage 列 729
34.20. domains 列 730
34.21. element_types 列 732
34.22. enabled_roles 列 733
34.23. foreign_data_wrapper_options 列 734
34.24. foreign_data_wrappers 列 734
34.25. foreign_server_options 列 734
34.26. foreign_servers 列 735
34.27. foreign_table_options 列 735
34.28. foreign_tables 列 735
34.29. key_column_usage 列 736
34.30. parameters 列 736
34.31. referential_constraints 列 738
34.32. role_column_grants 列 738
34.33. role_routine_grants 列 739
34.34. role_table_grants 列 739
34.35. role_udt_grants 列 740
34.36. role_usage_grants 列 740
34.37. routine_privileges 列 741
34.38. routines 列 741
34.39. schemata 列 745
34.40. sequences 列 745
34.41. sql_features 列 746

34.42. sql_implementation_info 列 746
34.43. sql_languages 列 747
34.44. sql_packages 列 747
34.45. sql_parts 列 748
34.46. sql_sizing 列 748
34.47. sql_sizing_profiles 列 748
34.48. table_constraints 列 749
34.49. table_privileges 列 749
34.50. tables 列 750
34.51. triggered_update_columns 列 751
34.52. triggers 列 751
34.53. udt_privileges 列 752
34.54. usage_privileges 列 753
34.55. user_defined_types 列 753
34.56. user_mapping_options 列 754
34.57. user_mappings 列 755
34.58. view_column_usage 列 755
34.59. view_routine_usage 列 755
34.60. view_table_usage 列 756
34.61. views 列 756
35.1. Equivalent C Types for Built-in SQL Types 784
35.2. B-树策略 814
35.3. 哈希策略 814
35.4. GiST 二维 “R 树” 策略 814
35.5. SP-GiST 点策略 815
35.6. GIN 数组策略 815
35.7. B-树支持函数 815
35.8. 哈希支持函数 815
35.9. GiST 支持函数 816
35.10. SP-GiST 支持函数 816
35.11. GIN 支持函数 816
39.1. 错误诊断项 888
45.1. 系统目录 1477
45.2. pg_aggregate 列 1478
45.3. pg_am 列 1479
45.4. pg_amop 列 1481
45.5. pg_amproc 列 1482
45.6. pg_attrdef 列 1482
45.7. pg_attribute 列 1483
45.8. pg_authid 列 1485
45.9. pg_auth_members 列 1486
45.10. pg_cast 列 1487
45.11. pg_class 列 1488
45.12. pg_constraint 列 1491
45.13. pg_collation 列 1493
45.14. pg_conversion 列 1493
45.15. pg_database 列 1494
45.16. pg_db_role_setting 列 1495
45.17. pg_default_acl 列 1496
45.18. pg_depend 列 1496
45.19. pg_description 列 1498
45.20. pg_enum 列 1498
45.21. pg_extension 列 1498
45.22. pg_foreign_data_wrapper 列 1499
45.23. pg_foreign_server 列 1500
45.24. pg_foreign_table 列 1500

45.25. pg_index 列 1501
45.26. pg_inherits 列 1503
45.27. pg_language 列 1503
45.28. pg_largeobject 列 1504
45.29. pg_largeobject_metadata 列 1505
45.30. pg_namespace 列 1505
45.31. pg_opclass 列 1505
45.32. pg_operator 列 1506
45.33. pg_opfamily 列 1507
45.34. pg_pltemplate列 1507
45.35. pg_proc 列 1508
45.36. pg_range 列 1511
45.37. pg_rewrite 列 1512
45.38. pg_seclabel 列 1512
45.39. pg_shdepend 列 1513
45.40. pg_shdescription 列 1514
45.41. pg_shseclabel 列 1514
45.42. pg_statistic 列 1515
45.43. pg_tablespace 列 1517
45.44. pg_trigger 列 1517
45.45. pg_ts_config 列 1518
45.46. pg_ts_config_map 列 1519
45.47. pg_ts_dict 列 1519
45.48. pg_ts_parser 列 1520
45.49. pg_ts_template 列 1520
45.50. pg_type 列 1521
45.51. typcategory Codes 1525
45.52. pg_user_mapping 列 1525
45.53. 系统视图 1526
45.54. pg_available_extensions 列 1526
45.55. pg_available_extension_versions 列 1527
45.56. pg_cursors 列 1527
45.57. pg_group 列 1528
45.58. pg_indexes 列 1528
45.59. pg_locks 列 1529
45.60. pg_prepared_statements 列 1531
45.61. pg_prepared_xacts 列 1532
45.62. pg_roles 列 1532
45.63. pg_rules 列 1533
45.64. pg_seclabels 列 1534
45.65. pg_settings 列 1535
45.66. pg_shadow 列 1536
45.67. pg_stats 列 1537
45.68. pg_tables 列 1539
45.69. pg_timezone_abbrevs 列 1539
45.70. pg_timezone_names 列 1539
45.71. pg_user 列 1540
45.72. pg_user_mappings 列 1540
45.73. pg_views 列 1541
56.1. Contents of PGDATA 1633
56.2. Page Layout 1638
56.3. PageHeaderData Layout 1638
56.4. HeapTupleHeaderData Layout 1640
A.1. PostgreSQL错误代码 1670
B.1. 月份名称 1679
B.2. 星期名称 1679

B.3. 日期/时间字段修饰符 1679
C.1. SQL 关键字 1683
F.1. adminpack 函数 2374
F.2. 已弃用的adminpack函数 2374
F.3. cube 外部表示 2386
F.4. cube GiST 操作符 2386
F.5. cube 函数 2387
F.6. 基于立方体的地球距离函数 2425
F.7. 基于点的地球距离操作符 2426
F.8. hstore Operators 2433
F.9. hstore Functions 2434
F.10. intarray 函数 2440
F.11. intarray 操作符 2440
F.12. isn 数据类型 2443
F.13. isn 函数 2444
F.14. ltree 操作符 2449
F.15. ltree 函数 2450
F.16. pg_buffercache 列 2457
F.17. Supported Algorithms for crypt() 2460
F.18. Iteration Counts for crypt() 2460
F.19. Hash Algorithm Speeds 2461
F.20. Summary of Functionality with and without OpenSSL 2467
F.21. pgrowlocks 输出列 2472
F.22. pg_stat_statements Columns 2473
F.23. pgstattuple 输出列 2477
F.24. pg_trgm函数 2479
F.25. pg_trgm操作符 2479
F.26. seg 外部表示 2482
F.27. 合法 seg 输入示例 2483
F.28. seg GiST 操作符 2483
F.29. Sepgsql Functions 2491
F.30. tablefunc函数 2497
F.31. connectby参数 2503
F.32. 用于生成 UUID 的函数 2513
F.33. 返回 UUID 常量的函数 2513
F.34. 函数 2514
F.35. xpath_table 参数 2515
H.1. 由外部维护的客户端接口 2548
H.2. 由外部维护的过程语言 2549

范例清单

8.1. 使用字符类型 105
8.2. 使用boolean类型 117
8.3. 使用位串类型 122
10.1. 阶乘操作符类型解析 261
10.2. 字符串连接操作符类型解析 261
10.3. 绝对值与取反操作符类型解析 262
10.4. 数组包含操作符类型解析 263
10.5. 域类型上的自定义操作符 263
10.6. round 函数参数类型解析 265
10.7. substr 函数类型解析 265
10.8. character 的存储类型转换 266
10.9. 未充分指定类型的 UNION 中的类型解析 267
10.10. 简单 UNION 中的类型解析 267
10.11. 次序对调的 UNION 中的类型解析 268
11.1. 建立一个部分索引以排除常见值 275
11.2. 建立一个部分索引以排除不感兴趣的值 275
11.3. 建立一个部分唯一索引 276
19.1. 示例 pg_hba.conf 项 441
19.2. 示例 pg_ident.conf 文件 444
31.1. libpq 示例程序 1 604
31.2. libpq示例程序 2 606
31.3. libpq示例程序 3 609
32.1. 用libpq操作大对象的示例程序 618
33.1. 示例 SQLDA 程序 663
33.2. 访问大对象的 ECPG 程序 676
38.1. 手工安装PL/Perl 862
39.1. 在动态查询中为值加引号 874
39.2. UPDATE/INSERT的异常 887
39.3. 一个 PL/pgSQL 触发器函数 896
39.4. 一个用于审计的 PL/pgSQL 触发器函数 897
39.5. 一个用于审计的 PL/pgSQL 视图触发器函数 898
39.6. 一个用于维护汇总表的 PL/pgSQL 触发器函数 899
39.7. 把一个简单函数从 PL/SQL 移植到 PL/pgSQL 907
39.8. 把一个创建其它函数的函数从 PL/SQL 移植到 PL/pgSQL 908
39.9. 把一个带字符串处理和 OUT 参数的过程从 PL/SQL 移植到 PL/pgSQL 909
39.10. 把一个过程从 PL/SQL 移植到 PL/pgSQL 910
F.1. 为 PostgreSQL CSV 日志创建外部表 2427

前言

本书是PostgreSQL的官方文档。它由PostgreSQL开发人员及其他志愿者在 PostgreSQL软件开发过程中同步编写。它描述了当前版本的PostgreSQL正式支持的全部功能。

为了便于管理有关PostgreSQL的大量信息，本书被组织为若干部分。
每一部分都面向不同类型的用户，或者面向处于不同PostgreSQL使用阶段的用户：

- 第 I 部分是面向新用户的非正式介绍。
- 第 II 部分记述了SQL查询语言环境，包括数据类型、函数以及用户级性能调优。每个PostgreSQL用户都应阅读这一部分。
- 第 III 部分描述了服务器的安装和管理。无论是为自己使用还是为他人提供服务，凡是运行PostgreSQL服务器的人都应阅读这一部分。
- 第 IV 部分描述PostgreSQL客户端程序的编程接口。
- 第 V 部分包含面向高级用户的服务器可扩展能力相关信息，其中的主题包括用户定义数据类型和函数。
- 第 VI 部分包含有关 SQL 命令、客户端程序和服务器程序的参考信息。该部分以按命令或程序组织的结构化信息为其他部分提供支持。
- 第 VII 部分包含可能对PostgreSQL开发人员有用的各类信息。

1. 什么是PostgreSQL?

PostgreSQL是一种基于 POSTGRES, Version 4.2[1] 的对象关系数据库管理系统（ORDBMS），后者由加州大学伯克利分校计算机科学系开发。POSTGRES 首创了许多概念，而这些概念直到很久之后才在某些商业数据库系统中出现。

PostgreSQL是这套原始伯克利代码的开源后继版本。它支持 SQL 标准中的很大一部分，并提供了许多现代特性：

- 复杂查询
- 外键
- 触发器
- 视图
- 事务完整性
- 多版本并发控制

此外，PostgreSQL还允许用户以多种方式扩展，例如添加新的：

- 数据类型
- 函数
- 操作符
- 聚合函数
- 索引方法
- 过程语言

由于采用宽松的许可证，任何人都可以出于任何目的免费使用、修改和分发PostgreSQL，
无论是私人、商业还是学术用途。

2. PostgreSQL简史

[1] http://db.cs.berkeley.edu/postgres.html

如今名为PostgreSQL的对象关系数据库管理系统，源自加州大学伯克利分校编写的POSTGRES软件包。经过二十多年的发展，
PostgreSQL如今已成为当今最先进的开源数据库。

2.1. 伯克利 POSTGRES 项目

由 Michael Stonebraker 教授领导的 POSTGRES 项目，得到了国防高级研究计划局（DARPA）、陆军研究办公室（ARO）、国家科学基金会（NSF）以及 ESL, Inc. 的资助。POSTGRES 的实现始于 1986 年。该系统的最初概念见于[ston86]，而最初数据模型的定义见于[rowe87]。当时规则系统的设计见于[ston87a]。
存储管理器的设计动机和体系结构详见[ston87b]。

自那以后，POSTGRES 经历了几次重大版本发布。第一个“演示版”系统于 1987 年开始运行，并在 1988 年的 ACM-SIGMOD 大会上进行了展示。版本 1 在[ston90a]中有描述，并于 1989 年 6 月发布给少数外部用户。针对第一代规则系统所受到的批评（[ston89]），规则系统被重新设计（[ston90b]），版本 2 连同新的规则系统于 1990 年 6 月发布。版本 3 于 1991 年问世，增加了对多个存储管理器的支持、改进后的查询执行器以及重写后的规则系统。此后直到 Postgres95（见下文）之前的后续版本，主要聚焦于可移植性和可靠性。

POSTGRES 被用于实现许多不同的研究和生产应用。这些应用包括金融数据分析系统、喷气发动机性能监测软件包、小行星跟踪数据库、医疗信息数据库以及若干地理信息系统。POSTGRES 还在多所大学中被用作教学工具。最后，Illustra Information Technologies（后来并入 Informix[2]，该公司现归 IBM[3] 所有）接手了这套代码并将其商业化。在 1992 年末，POSTGRES 成为Sequoia 2000 科学计算项目[4]的主要数据管理器。

1993 年期间，外部用户社区的规模几乎翻了一番。人们越来越清楚地意识到，对原型代码的维护和支持工作占用了大量本应用于数据库研究的时间。为了减轻这一支持负担，伯克利 POSTGRES 项目在版本 4.2 时正式结束。

2.2. Postgres95

1994 年，Andrew Yu 和 Jolly Chen 为 POSTGRES 添加了一个 SQL 语言解释器。随后，它以 Postgres95 这一新名称发布到网络上，作为最初的 POSTGRES 伯克利代码的开源后继者自行发展。

Postgres95的代码完全采用 ANSI C 编写，并将体积缩减了 25%。许多内部更改提升了性能和可维护性。Postgres951.0.x 版本在 Wisconsin Benchmark 上快了大约 30-50%，比较对象是POSTGRES4.2 版。除了错误修复之外，主要增强还包括：

- 查询语言 PostQUEL 被 SQL 所取代（在服务器中实现）。直到 PostgreSQL（见下文）才支持子查询，但在 Postgres95 中可以用用户定义的 SQL 函数来模拟。聚合函数也得到了重新实现，并增加了对 `GROUP BY` 查询子句的支持。
- 新增了一个用于交互式 SQL 查询的程序（psql），它使用了 GNU Readline。这在很大程度上取代了旧的 monitor 程序。
- 新增的前端库 `libpgtcl` 支持基于 Tcl 的客户端。示例 shell `pgtclsh` 提供了新的 Tcl 命令，用于让 Tcl 程序与 Postgres95 服务器交互。
- 大对象接口经过了全面改造。反转大对象成为存储大对象的唯一机制。
（反转文件系统已被移除。）
- 实例级规则系统被移除。规则仍然可以作为重写规则使用。
- 源代码中附带了一份简短教程，介绍常规 SQL 特性以及 Postgres95 的特性。

[2] http://www.informix.com/
[3] http://www.ibm.com/
[4] http://meteora.ucsd.edu/s2k/s2k_home.html

- 构建时使用的是 GNU make（而不是 BSD make）。此外，Postgres95 可以用未经修改的 GCC 编译（双精度浮点数的数据对齐问题已修复）。

2.3. PostgreSQL

到了 1996 年，很明显“Postgres95”这个名字经不起时间的考验。我们选择了一个新名字 PostgreSQL，以体现最初的 POSTGRES 与后来具备 SQL 能力的版本之间的关系。同时，我们将版本号从 6.0 开始，使编号重新回到最初由伯克利 POSTGRES 项目开启的序列中。

许多人出于传统，或因为更容易发音，仍然将PostgreSQL称为“Postgres”（如今很少全部大写）。这一用法作为昵称或别名已被广泛接受。

在开发 Postgres95 期间，重点是识别并理解服务器代码中已有的问题。到了 PostgreSQL 阶段，重点则转向增强特性和能力，不过各个方面的工作仍在继续。

自那以后 PostgreSQL 发生了哪些变化，可见附录 E。

3. 约定

本书使用以下排版约定来标记某些文本段落：新的术语、外来语和其他重要的段落以斜体强调。所有表示计算机输入或输出的内容，特别是命令、程序代码和屏幕输出，都以等宽字体（`example`）显示。在这类段落中，斜体（*`example`*）表示占位符；你必须用实际值来替换占位符。有时，程序代码的一部分会以粗体（**`example`**）强调，表示它们是相对于前一个示例新增或修改的内容。

以下约定用于命令概要：方括号（`[`和`]`）表示可选部分。（在 Tcl 命令的概要中，改用问号（`?`）表示，这是 Tcl 的惯例。）大括号（`{`和`}`）和竖线（`|`）表示必须选择其中一个替代项。点（`...`）表示前面的元素可以重复。

在有助于增强清晰度时，SQL 命令前会加上提示符`=>`，shell 命令前会加上提示符`$`。不过，通常并不显示提示符。

一般来说，管理员是负责安装和运行服务器的人。用户则可以是任何正在使用或希望使用PostgreSQL系统任一部分的人。不应把这些术语理解得过于狭隘；本书并不对系统管理流程作固定预设。

4. 更多信息

除了本文档，也就是本书之外，还有其他关于PostgreSQL的资源：

Wiki

PostgreSQL wiki[5] 收录了该项目的FAQ[6]（常见问题解答）列表、TODO[7] 列表，以及许多其他主题的详细信息。

网站

PostgreSQL 网站[8]提供最新版本的详细信息，以及其他能让你在工作中或日常使用PostgreSQL时更加高效的信息。

邮件列表

邮件列表是一个让你的问题得到解答、与其他用户分享经验以及联系开发者的好地方。详情请参阅PostgreSQL网站。

[5] https://wiki.postgresql.org
[6] https://wiki.postgresql.org/wiki/Frequently_Asked_Questions
[7] https://wiki.postgresql.org/wiki/Todo
[8] https://www.postgresql.org

你自己！

PostgreSQL是一个开源项目。因此，它依赖用户社区提供持续支持。随着你开始使用PostgreSQL，无论是通过本文档还是通过邮件列表，你都会依赖他人的帮助。
请考虑把你的知识回馈给社区。阅读邮件列表并回答问题。
如果你学到了一些文档中尚未包含的内容，就把它写出来并贡献出去。
如果你为代码增加了功能，也请将其贡献出来。

5. 缺陷报告指南

当你在PostgreSQL中发现缺陷时，我们希望得知此事。你的缺陷报告对于提高 PostgreSQL的可靠性十分重要，因为即使再怎么小心，也无法保证 PostgreSQL的每一部分都能在每个平台、任何情况下正常工作。

以下建议旨在帮助你组织缺陷报告，以便能够高效处理。没有人必须遵循这些建议，但这样做通常对各方都有好处。

我们无法承诺立刻修复每一个缺陷。如果该缺陷明显、严重，或者影响大量用户，很可能会有人去调查它。也可能我们会让你升级到更新的版本，看看在新版本中是否也会出现同样的缺陷。或者，我们可能会认定，在某些计划中的重大重写完成之前，这个缺陷无法修复。再或者，它只是过于棘手，而当前还有更重要的事情要处理。如果你需要立即获得帮助，请考虑购买商业支持合同。

5.1. 识别缺陷

在报告缺陷之前，请反复阅读文档，以确认你确实可以完成正在尝试做的事情。
如果从文档中无法明确看出某件事是否可行，也请报告；那就是文档中的缺陷。
如果程序实际做的事情与文档描述不同，那就是缺陷。这可能包括但不限于以下情况：

- 程序因致命信号而终止，或者给出一条表明程序本身存在问题的操作系统错误消息。（反例可能是“磁盘满”之类的消息，因为那得由你自己解决。）
- 程序对给定输入产生了错误的输出。
- 程序拒绝接受有效输入（按文档定义）。
- 程序接受了无效输入，却没有给出提示或错误消息。但请记住，你认为的无效输入，可能正是我们认为的扩展功能，或者是对传统做法的兼容。
- 在受支持的平台上，PostgreSQL无法按照说明完成编译、构建或安装。

这里的“程序”指的是任何可执行程序，不只是后端进程。

运行缓慢或消耗大量资源，并不一定就是缺陷。请阅读文档，或在某个邮件列表中寻求调优应用程序的帮助。不符合 SQL标准，也不一定是缺陷，除非明确声称该特定特性符合标准。

在继续之前，请查看 TODO 列表和 FAQ，确认你的缺陷是否已经是已知问题。如果你看不懂 TODO 列表中的信息，也请报告你的问题。至少我们可以把 TODO 列表写得更清楚。

5.2. 报告哪些内容

关于缺陷报告，最重要的一点是陈述全部事实，而且只陈述事实。
不要猜测你觉得哪里出了问题、“它看起来像是在做什么”，或者程序的哪一部分有错。
如果你不熟悉实现，十有八九会猜错，对我们毫无帮助。即使你熟悉实现，有根据的解释也只能是事实的有益补充，而不能替代事实。如果我们要修复这个缺陷，仍然必须先亲自重现它。
陈述这些基本事实并不难做（你大概可以直接从屏幕上复制粘贴），但太多时候，

重要细节会被遗漏，因为有人觉得它们不重要，或者认为即使不说清楚，报告也一样能被理解。

以下各项应包含在每一个缺陷报告中：

- 重现问题所需的精确步骤序列，而且必须是从程序启动开始。这些步骤应当是自包含的；如果输出依赖表中的数据，那么仅仅发来一条 `SELECT`语句，而不附带前面的`CREATE TABLE`和 `INSERT`语句，是不够的。我们没有时间去逆向推导你的数据库模式；如果要靠我们自己编造数据，多半会错过问题。

  对于 SQL 相关问题，测试用例的最佳形式是一个可由psql前端运行、并能展示问题的文件。（请务必确保你的`~/.psqlrc`启动文件中没有任何内容。）创建这个文件的一个简便方法，是用pg_dump导出搭建场景所需的表声明和数据，再附上触发问题的查询。我们鼓励你尽量缩小示例规模，但这并非绝对必要。如果缺陷可重现，我们无论如何都能找出来。

  如果你的应用使用的是别的客户端接口，例如PHP，那么请尽量隔离出触发问题的查询。我们大概不会为了重现你的问题去搭建一个 Web 服务器。无论如何，都请记得提供精确的输入文件；不要笼统地猜测问题发生在“大文件”或“中等大小的数据库”等情况下，因为这些信息不够精确，派不上用场。

- 你实际得到的输出。请不要只说它“没起作用”或“崩溃了”。如果有错误消息，请把它贴出来，即使你看不懂。如果程序因操作系统错误而终止，请说明是哪一种错误。如果什么都没发生，也请说清楚。即使你的测试用例导致程序崩溃，或出现其他显而易见的问题，这种情况在我们的平台上也未必会发生。如果可以，最简单的做法就是直接从终端复制输出。

  > **注意**
  >
  > 如果你要报告错误消息，请尽量获取其最详细的形式。在psql中，请预先执行`\set VERBOSITY verbose`。如果你是从服务器日志中提取消息，请将运行时参数log_error_verbosity设为`verbose`，以便记录全部细节。

  > **注意**
  >
  > 在致命错误的情况下，客户端报告的错误消息可能不包含全部可用信息。也请查看数据库服务器的日志输出。如果你平时没有保留服务器日志，现在正是开始这样做的好时机。

- 你期望得到的输出，也非常重要。如果你只是写“这个命令给了我那个输出。”或“这不是我期望的。”，我们可能会自己运行一遍，扫一眼输出，然后觉得它看起来没问题，也正是我们所期望的。我们不应该花时间去揣摩你的命令背后的确切语义。尤其不要只是说“这不是 SQL 规定的/Oracle 所做的。”从SQL标准中找出正确行为并不是一件轻松的事，而且我们也不都了解其他所有关系数据库的行为。（如果你的问题是程序崩溃，显然可以省略这一项。）

- 任何命令行选项和其他启动选项，包括所有被你改动过、且与问题相关的环境变量或配置文件。同样，请提供精确的信息。如果你使用的是一种预打包发行版，并且它会在系统启动时自动启动数据库服务器，那么你应当尽量弄清楚它是怎么做到的。

- 任何偏离安装说明的做法。

- PostgreSQL的版本。你可以运行命令 `SELECT version();`来查明所连接服务器的版本。大多数可执行程序也支持 `--version`选项；至少`postgres --version`和

psql --version应当可以工作。如果这个函数或这些选项都不存在，那就说明你的版本已经老到足以该升级了。如果你运行的是预打包版本，例如 RPM，请说明这一点，并附上该软件包可能带有的子版本号。如果你说的是一个 Git 快照，也请说明，并给出提交哈希值。

如果你的版本早于 9.2.24，我们几乎肯定会建议你升级。每个新版本都会包含大量缺陷修复和改进，因此你在旧版 PostgreSQL中遇到的缺陷，很可能已经被修复。对于使用旧版 PostgreSQL的站点，我们只能提供有限支持；如果你需要超出这一范围的帮助，请考虑购买商业支持合同。

- 平台信息。这包括内核名称和版本、C 库、处理器、内存信息等。在大多数情况下，报告供应商和版本就足够了，但不要想当然地认为所有人都知道“Debian”具体包含什么，或者所有人都运行在 i386 上。如果你遇到的是安装问题，那么你机器上的工具链信息（编译器、make等）也是必需的。

不要担心你的缺陷报告会变得很长；这很正常。第一次就把所有内容都报告出来，总比让我们反复追问细节要好。另一方面，如果你的输入文件非常大，先问问是否有人愿意看一看，也是合理的。这里有一篇文章[9]，概述了更多关于报告缺陷的建议。

不要把全部时间都花在找出输入中的哪些变化会让问题消失上。这多半无助于解决问题。如果最后发现这个缺陷无法立刻修复，你仍然有时间去寻找并分享变通办法。还有，再强调一次，不要浪费时间去猜测这个缺陷为什么存在。我们迟早会查明原因。

在编写缺陷报告时，请避免使用含混的术语。整个软件包叫作“PostgreSQL”，有时简称“Postgres”。如果你特指后端进程，请明确说出来，不要只是说“PostgreSQL 崩溃了”。单个后端进程崩溃，与父“postgres”进程崩溃，是完全不同的事情；当你指的是某个单独的后端进程挂掉时，请不要说“服务器崩溃了”，反过来也一样。此外，客户端程序，例如交互式前端“psql”，与后端完全是分开的。请尽量明确问题是在客户端还是服务器端。

5.3. 到哪里报告缺陷

一般来说，请将缺陷报告发送到缺陷报告邮件列表 <pgsql-bugs@postgresql.org>。请为电子邮件使用描述性的主题，必要时可以摘取部分错误消息。

另一种方法是填写项目网站[10]上提供的缺陷报告网页表单。通过这种方式提交的缺陷报告，也会被发送到 <pgsql-bugs@postgresql.org>邮件列表。

如果你的缺陷报告涉及安全影响，而且你不希望它立刻出现在公开归档中，请不要发送到 pgsql-bugs。安全问题可以私下报告给 <security@postgresql.org>。

不要将缺陷报告发送到任何用户邮件列表，例如 <pgsql-sql@postgresql.org>或 <pgsql-general@postgresql.org>。这些邮件列表用于回答用户问题，其订阅者通常并不希望收到缺陷报告。更重要的是，他们也不太可能去修复这些缺陷。

此外，请不要把报告发送到开发者邮件列表 <pgsql-hackers@postgresql.org>。这个列表是用来讨论 PostgreSQL开发的，最好把缺陷报告和这类讨论分开。如果这个问题需要更深入的审查，我们可能会选择在 pgsql-hackers上继续讨论你的缺陷报告。

如果你遇到的是文档问题，报告它的最佳地点是文档邮件列表 <pgsql-docs@postgresql.org>。请明确指出你对文档的哪一部分不满意。

如果你的缺陷是在某个不受支持平台上出现的可移植性问题，请发送邮件到 <pgsql-hackers@postgresql.org>，这样我们（以及你）就可以共同推进 PostgreSQL在该平台上的移植工作。

[9] http://www.chiark.greenend.org.uk/~sgtatham/bugs.html
[10] https://www.postgresql.org/

注意

这些邮件地址都是封闭邮件列表。也就是说，你必须订阅某个列表才能向它发帖。（不过，使用错误报告网页表单无需订阅。）
如果你想发邮件但又不想接收列表流量，可以订阅并把订阅选项设为 `nomail`。要获取更多信息，可向 `<majordomo@postgresql.org>` 发送邮件，并在正文里写上单词 `help`。

部分 I. 教程

欢迎阅读PostgreSQL教程。本教程旨在介绍PostgreSQL、关系数据库概念以及 SQL 语言。我们假定读者具备基本的计算机使用常识，不要求具备特定的 Unix 或编程经验。本教程旨在让读者通过动手实践了解PostgreSQL系统的一些重要方面。它并不试图对所涵盖的主题作全面论述。

在成功完成本教程后，你可能会希望阅读第 II 部分部分，以更深入地理解 SQL 语言；或者阅读第 IV 部分，了解如何使用PostgreSQL开发应用程序。自行部署并管理 PostgreSQL 安装的读者也应阅读第 III 部分。

目录

1. 从头开始 3
1.1. 安装 3
1.2. 架构基础 3
1.3. 创建数据库 3
1.4. 访问数据库 5
2. SQL语言 7
2.1. 简介 7
2.2. 概念 7
2.3. 创建一个新表 7
2.4. 向表中填充行 8
2.5. 查询表 9
2.6. 表之间的连接 10
2.7. 聚合函数 12
2.8. 更新 13
2.9. 删除 14
3. 高级特性 15
3.1. 简介 15
3.2. 视图 15
3.3. 外键 15
3.4. 事务 16
3.5. 窗口函数 17
3.6. 继承 20
3.7. 小结 21

第 1 章 从头开始

1.1. 安装

当然，在你开始使用PostgreSQL之前，必须先安装它。
PostgreSQL可能已经安装在你的站点上，可能是因为它随操作系统发行版一起提供，
也可能是因为系统管理员已经安装了它。如果是这样，
你应该从操作系统文档或系统管理员那里了解如何访问PostgreSQL。

如果你不确定PostgreSQL是否已经可用，或者是否可以用它来做实验，
那么你可以自行安装。这并不难，而且也是一个不错的练习。
PostgreSQL可由任何非特权用户安装，不需要超级用户（root）权限。

如果你要自己安装PostgreSQL，请参阅第 15 章中的安装说明，待安装完成后再回到本指南。务必仔细遵循关于设置适当环境变量的小节。

如果你的站点管理员没有按默认方式完成设置，你可能还需要做一些额外工作。例如，
如果数据库服务器所在的机器是远程机器，你就需要把`PGHOST`
环境变量设置为数据库服务器机器的名称。环境变量`PGPORT`也可能需要设置。
归根结底就是：如果你试着启动某个应用程序，而它抱怨无法连接到数据库，
那么你应该去咨询站点管理员；如果你自己就是管理员，
那就查阅文档以确保你的环境已经正确设置。如果你没有理解上一段的意思，
请阅读下一节。

1.2. 架构基础

在继续之前，你应该先了解PostgreSQL的基本系统架构。
了解PostgreSQL各部分如何交互，会让本章更容易理解。

用数据库术语来说，PostgreSQL采用客户端/服务器模型。
一次PostgreSQL会话由以下协同工作的进程（程序）组成：

- 一个服务器进程，负责管理数据库文件、接受客户端应用到数据库的连接，
  并代表客户端执行数据库操作。数据库服务器程序叫做`postgres`。

- 用户用来执行数据库操作的客户端（前端）应用。
  客户端应用的形式可以非常多样：可以是一个文本工具，也可以是一个图形应用程序，
  或者是一个通过访问数据库来显示网页的 Web 服务器，还可以是专门的数据库维护工具。
  有些客户端应用随PostgreSQL发行版一起提供；大多数则由用户开发。

和典型的客户端/服务器应用一样，客户端和服务器可以位于不同的主机上。在这种情况下，
它们通过 TCP/IP 网络连接通信。你应该记住这一点，因为在客户端机器上可访问的文件，在
数据库服务器机器上可能不可访问（或者只能使用不同的文件名访问）。

PostgreSQL服务器可以处理来自客户端的多个并发连接。为此，
它会为每个连接启动（“forks”）一个新进程。从那时起，
客户端和新的服务器进程就无需原始 `postgres`进程介入而直接通信。因此，这个主服务器
进程始终在运行，等待客户端连接，而客户端及其关联的服务器进程则不断创建和退出。
（当然，这一切对用户都是不可见的。这里只是为求完整而提及。）

1.3. 创建数据库

检验你能否访问数据库服务器的第一步，就是试着创建一个数据库。一个正在运行的
PostgreSQL服务器可以管理许多数据库。
通常会为每个项目或每个用户使用一个独立的数据库。

你的站点管理员可能已经为你创建了一个可用的数据库。
他应该已经告诉你你的数据库名是什么。如果是这样，你就可以省略这一步，
直接跳到下一节。

要创建一个新数据库（本例中命名为 mydb），请使用以下命令：

```
$ createdb mydb
```

如果这条命令没有任何输出，就说明这一步成功了，你可以跳过本节余下的内容。

如果你看到类似下面这样的信息：

```
createdb: command not found
```

那么说明PostgreSQL没有正确安装。要么根本没有安装，要么就是你的 shell 搜索路径没有把它包含进去。试着改用绝对路径调用该命令：

```
$ /usr/local/pgsql/bin/createdb mydb
```

你所在站点上的路径可能不同。请联系站点管理员，或者查看安装说明来修正这个问题。

另一种响应可能是这样：

```
createdb: could not connect to database postgres: could not connect
 to server: No such file or directory
        Is the server running locally and accepting
        connections on Unix domain socket "/tmp/.s.PGSQL.5432"?
```

这意味着服务器尚未启动，或者它没有在 createdb 期望的位置启动。同样，请查看安装说明或咨询管理员。

另一种响应可能是这样：

```
createdb: could not connect to database postgres: FATAL:  role
 "joe" does not exist
```

其中提到了你自己的登录名。如果管理员没有为你创建 PostgreSQL 用户帐号，就会发生这种情况。（PostgreSQL用户帐号与操作系统用户帐号是分开的。）如果你是管理员，请参阅第 20 章了解如何创建帐号。你需要切换成安装 PostgreSQL 时所用的操作系统用户（通常是 postgres），才能创建第一个用户帐号。也可能分配给你的 PostgreSQL 用户名与操作系统用户名不同；在这种情况下，你需要使用 -U 选项，或者设置 PGUSER 环境变量，来指定你的 PostgreSQL 用户名。

如果你有用户帐号，但它没有创建数据库所需的权限，那么你会看到下面的信息：

```
createdb: database creation failed: ERROR:  permission denied to
 create database
```

并非每个用户都被授权创建新数据库。如果 PostgreSQL 拒绝为你创建数据库，那么站点管理员就需要授予你创建数据库的权限。遇到这种情况时，请咨询站点管理员。如果 PostgreSQL 是你自己安装的，那么在本教程中，你应当以启动服务器时所用的用户帐号登录。[1]

你也可以创建其他名称的数据库。
PostgreSQL允许你在一个给定站点上创建任意数量的数据库。
数据库名的首字符必须是字母，并且长度限制为 63 字节。一种方便的做法是创建一个与当前用户名同名的数据库。许多工具都将该数据库名视为默认值，因此这样可以少打一些字。
要创建这个数据库，只需输入：

[1] 这之所以可行，是因为PostgreSQL用户名与操作系统用户帐号是分开的。当你连接到数据库时，可以选择以哪个PostgreSQL用户名连接；如果不指定，默认就会使用你当前操作系统帐号的名称。恰好总会有一个PostgreSQL用户帐号，
其名称与启动服务器的操作系统用户相同，而该用户也总是有权创建数据库。除了直接以该用户身份登录之外，
你还可以在各处使用-U选项来选择连接时使用的PostgreSQL用户名。

```
$ createdb
```

如果你不再想使用某个数据库，可以删除它。例如，如果你是数据库mydb的所有者（创建者），就可以使用下面的命令销毁它：

```
$ dropdb mydb
```

（对于这条命令，数据库名不会默认成用户帐号名，你始终需要显式指定它。）这个操作会从物理上删除与该数据库关联的所有文件，而且无法撤销，因此务必三思而后行。

关于createdb和dropdb的更多信息，可分别参见createdb和dropdb。

1.4. 访问数据库

一旦你创建了数据库，你就可以通过以下方式访问它：

- 运行PostgreSQL的交互式终端程序 psql，它允许你以交互方式输入、编辑并执行SQL命令。
- 使用现有的图形前端工具，例如pgAdmin，或者使用具备ODBC或JDBC支持的办公套件来创建和操作数据库。这些方式不在本教程讨论范围内。
- 使用若干可用语言绑定之一编写自定义应用程序。关于这些方式的进一步讨论，请见第 IV 部分。

你可能需要启动psql来试验本教程中的示例。可以通过输入下面的命令连接到mydb数据库：

```
$ psql mydb
```

如果你不提供数据库名，那么它默认就是你的用户帐号名。在上一节使用createdb时，你已经见识过这种机制了。

在psql中，你将看到下面的欢迎信息：

```
psql (9.2.24)
Type "help" for help.

mydb=>
```

最后一行也可能是：

```
mydb=#
```

这表示你是数据库超级用户；如果这个 PostgreSQL实例是你自己安装的，那很可能就是这种情况。超级用户意味着你不受访问控制限制。对本教程而言，这一点并不重要。

如果你在启动psql时遇到问题，请回到前一节。createdb和psql的诊断信息很相似，如果前者能工作，后者也应该能工作。

由 psql 打印出的最后一行就是提示符，它表示 psql正在等待你的输入，你可以将 SQL 查询输入到由 psql 维护的工作区中。试试这些命令：

```
mydb=> SELECT version();
                                 version
 ----------------------------------------------------------------
-----
 PostgreSQL 9.2.24 on i586-pc-linux-gnu, compiled by GCC 2.96, 32-
bit
(1 row)

mydb=> SELECT current_date;
```

```
    date
------------
 2002-08-31
(1 row)

mydb=> SELECT 2 + 2;
 ?column?
----------
        4
(1 row)
```

psql程序有一些不属于 SQL 命令的内部命令。它们以反斜线“\”开头。例如，你可以输入下面的命令来查看各种PostgreSQL SQL命令的语法帮助：

```
mydb=> \h
```

要退出psql，输入：

```
mydb=> \q
```

psql将退出并返回到命令 shell。（要了解更多内部命令，可输入\?，在psql提示符下执行。）psql的完整功能见psql。在本教程中，我们不会明确使用这些功能，
但在有帮助时你可以自行使用它们。

第 2 章 SQL语言

2.1. 简介

本章概述如何使用SQL执行简单操作。本教程仅供你入门之用，绝不是一份完整的SQL教程。已有许多关于SQL的书籍，包括[melt93]和[date97]。你应当知道，PostgreSQL的某些语言特性是对标准的扩展。

在下面的示例中，我们假定你已经按照前一章所述创建了名为mydb的数据库，并且已经能够启动psql。

本手册中的示例也可以在PostgreSQL源代码发行包的 src/tutorial/目录中找到。（PostgreSQL 的二进制发行包可能不包含这些源文件。）要使用这些源文件，请先切换到该目录并运行 make：

```
$ cd ..../src/tutorial
$ make
```

这会创建脚本并编译包含用户定义函数和类型的 C 文件。然后，要开始本教程，请执行以下操作：

```
$ cd ..../tutorial
$ psql -s mydb
...
mydb=> \i basics.sql
```

\i命令会从指定文件中读取命令。psql的 -s选项会让你进入单步模式，在将每条语句发送给服务器之前都会暂停。本节使用的命令都位于文件basics.sql中。

2.2. 概念

PostgreSQL是一种关系数据库管理系统（RDBMS）。这意味着它是一个用于管理存储在关系中的数据的系统。关系本质上是表的数学术语。如今，把数据存储在表中的观念已经十分常见，似乎显而易见，但组织数据库的方式还有很多。类Unix 操作系统中的文件和目录就是层次数据库的一个例子。较新的发展则是面向对象数据库。

每个表都是一个有名字的行集合。给定表中的每一行都有相同的一组具名列，并且每一列都具有特定的数据类型。虽然列在每一行中的顺序是固定的，但一定要记住，SQL 并不以任何方式保证表内行的顺序（虽然可以为了显示而对它们显式排序）。

表被归入数据库，而由单个PostgreSQL服务器实例管理的数据库集合构成一个数据库集簇。

2.3. 创建一个新表

你可以通过指定表名以及各列的列名和类型来创建一个新表：

```
CREATE TABLE weather (
    city            varchar(80),
    temp_lo         int,           -- low temperature
    temp_hi         int,           -- high temperature
    prcp            real,          -- precipitation
    date            date
);
```

你可以在psql中按这种换行格式输入该命令。psql会识别出该命令在分号出现之前尚未结束。

在 SQL 命令中可以随意使用空白字符（即空格、制表符和换行符）。这意味着你可以按与上面不同的方式对齐该命令，甚至把它全部写在一行上。两个连字符（“--”）表示注释。从它们开始直到行尾的内容都会被忽略。SQL 对关键字和标识符不区分大小写，除非用双引号括起标识符以保留大小写（上面的例子没有这么做）。

varchar(80)指定一种数据类型，它可以存储长度不超过 80 个字符的任意字符串。int是普通的整数类型。real是一种用于存储单精度浮点数的类型。date应当不言自明。（是的，类型为date的列也叫 date。这可能方便，也可能令人困惑 – 由你自己决定。）

PostgreSQL支持标准的SQL类型 int、smallint、real、double precision、char(*N*)、varchar(*N*)、date、time、timestamp和interval，还支持其他一些通用类型以及丰富的几何类型。PostgreSQL 可以通过任意数量的用户定义数据类型进行定制。因此，类型名在语法中并不是关键字，只有在为了支持SQL标准中的特殊情况而必须如此时才是例外。

第二个示例将存储城市及其关联的地理位置：

```
CREATE TABLE cities (
    name            varchar(80),
    location        point
);
```

point类型就是PostgreSQL特有数据类型的一个例子。

最后还要提一句，如果你不再需要某个表，或者想用不同的定义重新创建它，可以使用下面的命令把它移除：

```
DROP TABLE tablename;
```

2.4. 向表中填充行

INSERT语句用于向表中填充行：

```
INSERT INTO weather VALUES ('San Francisco', 46, 50, 0.25, '1994-11
-27');
```

请注意，所有数据类型的输入格式都相当直观。那些并非简单数值的常量通常必须用单引号（'）括起，就像这个例子一样。date类型实际上可接受的格式相当灵活，不过在本教程中，我们将坚持使用这里展示的这种无歧义格式。

point类型要求一个坐标对作为输入，如下所示：

```
INSERT INTO cities VALUES ('San Francisco', '(-194.0, 53.0)');
```

到目前为止所用的语法要求你记住列的顺序。另一种语法允许你显式列出列：

```
INSERT INTO weather (city, temp_lo, temp_hi, prcp, date)
    VALUES ('San Francisco', 43, 57, 0.0, '1994-11-29');
```

如果愿意，你可以按不同顺序列出列，甚至省略某些列，例如降水量未知时：

```
INSERT INTO weather (date, city, temp_hi, temp_lo)
    VALUES ('1994-11-29', 'Hayward', 54, 37);
```

许多开发者认为，与依赖隐含顺序相比，显式列出列是一种更好的风格。

请把上面展示的所有命令都输入一遍，这样你在下面各节中才有一些数据可供操作。

也可以使用COPY从纯文本文件装载大量数据。这通常更快，因为COPY命令针对这种用途进行了优化，不过灵活性不如INSERT。例如：

```
COPY weather FROM '/home/user/weather.txt';
```

其中，源文件必须可以在运行后端进程的机器上访问，而不是客户端机器，
因为后端进程直接读取该文件。有关COPY命令的更多信息，请参见COPY。

2.5. 查询表

要从表中检索数据，就要查询该表。SQL的SELECT语句就是用来做这件事的。该语句分为选择列表（列出要返回的列）、表列表（列出从哪些表中检索数据）
以及可选的限定条件（指定限制条件的部分）。例如，要检索表 weather的所有行，键入：

```
SELECT * FROM weather;
```

这里*是“所有列”的缩写。[1]因此，使用下面的语句也会得到相同结果：

```
SELECT city, temp_lo, temp_hi, prcp, date FROM weather;
```

输出应该是：

```
     city      | temp_lo | temp_hi | prcp |    date
---------------+---------+---------+------+------------
 San Francisco |      46 |      50 | 0.25 | 1994-11-27
 San Francisco |      43 |      57 |    0 | 1994-11-29
 Hayward       |      37 |      54 |      | 1994-11-29
(3 rows)
```

你可以在选择列表中编写表达式，而不仅仅是简单的列引用。例如，你可以这样做：

```
SELECT city, (temp_hi+temp_lo)/2 AS temp_avg, date FROM weather;
```

这样应该得到：

```
     city      | temp_avg |    date
---------------+----------+------------
 San Francisco |       48 | 1994-11-27
 San Francisco |       50 | 1994-11-29
 Hayward       |       45 | 1994-11-29
(3 rows)
```

注意这里AS子句如何用于重新命名输出列。（AS子句是可选的。）

查询可以被“限定”，方法是添加一个指定所需行的WHERE子句。WHERE
子句包含一个布尔（真值）表达式，只有使该布尔表达式为真的行才会被返回。
在限定条件中可以使用常见的布尔操作符（AND、OR和NOT）。例如，下面的查询检索 San Francisco 在雨天的天气记录：

```
SELECT * FROM weather
    WHERE city = 'San Francisco' AND prcp > 0.0;
```

结果：

```
     city      | temp_lo | temp_hi | prcp |    date
---------------+---------+---------+------+------------
 San Francisco |      46 |      50 | 0.25 | 1994-11-27
(1 row)
```

你可以要求查询结果按排序后的顺序返回：

[1] 虽然SELECT *对于即席查询很有用，但在生产代码中通常认为这是糟糕的风格，因为给表增加一个列就会改变结果。

```
SELECT * FROM weather
    ORDER BY city;

     city      | temp_lo | temp_hi | prcp |    date
---------------+---------+---------+------+------------
 Hayward       |      37 |      54 |      | 1994-11-29
 San Francisco |      43 |      57 |    0 | 1994-11-29
 San Francisco |      46 |      50 | 0.25 | 1994-11-27
```

在这个例子中，排序顺序没有完全指定，因此两个 San Francisco 行可能以任意次序出现。但如果你使用下面的语句，就总会得到上面显示的结果：

```
SELECT * FROM weather
    ORDER BY city, temp_lo;
```

你可以要求从查询结果中去除重复行：

```
SELECT DISTINCT city
    FROM weather;

     city
---------------
 Hayward
 San Francisco
(2 rows)
```

这里同样要说明，结果行的顺序可能发生变化。你可以把`DISTINCT`和`ORDER BY`一起使用，以确保结果一致：[2]

```
SELECT DISTINCT city
    FROM weather
    ORDER BY city;
```

2.6. 表之间的连接

到目前为止，我们的查询每次只访问一个表。查询可以同时访问多个表，也可以访问同一个表并同时处理其中的多行。一次访问同一个表或不同表中多行的查询称为连接查询。例如，假设要列出所有天气记录及其对应城市的位置。为此，需要比较`city`列在`weather`表各行中的值与`name`列在`cities`表所有行中的值，选出这些值匹配的行对。

> **注意**
>
> 这只是一个概念模型。
> 连接通常会以比真正比较每一个可能的行对更高效的方式执行，
> 不过这对用户是不可见的。

下面的查询可以完成这一点：

```
SELECT *
    FROM weather, cities
    WHERE city = name;

     city      | temp_lo | temp_hi | prcp |    date    |     name
    | location
```

[2] 在一些数据库系统中，包括较老版本的PostgreSQL，`DISTINCT`的实现会自动对行排序，因此`ORDER BY`并非必需。但这并不是SQL 标准的要求，而且当前的 PostgreSQL并不保证`DISTINCT`会使行按顺序排列。

```
---------------+---------+---------+------+------------+-----------
----+-----------
 San Francisco |      46 |      50 | 0.25 | 1994-11-27 | San
 Francisco | (-194,53)
 San Francisco |      43 |      57 |    0 | 1994-11-29 | San
 Francisco | (-194,53)
(2 rows)
```

注意结果集的两点：

- 没有与城市 Hayward 对应的结果行。这是因为`cities`表中没有与 Hayward 匹配的项，所以连接会忽略`weather`表中的不匹配行。我们很快就会看到如何修正这一点。
- 有两列包含城市名称。这是正确的，因为来自`weather`和`cities`表的列列表被拼接在一起。不过，实际中通常不希望这样，因此可能更希望显式列出输出列，而不使用`*`：

  ```
  SELECT city, temp_lo, temp_hi, prcp, date, location
      FROM weather, cities
      WHERE city = name;
  ```

练习：. 试着确定省略`WHERE`子句时，这条查询的含义。

由于所有列的名称各不相同，解析器自动确定了它们所属的表。
如果两个表中有重复的列名，就需要限定列名，以说明所指的是哪一列，例如：

```
SELECT weather.city, weather.temp_lo, weather.temp_hi,
       weather.prcp, weather.date, cities.location
    FROM weather, cities
    WHERE cities.name = weather.city;
```

通常认为，限定连接查询中的所有列名是一种良好风格，
这样即使以后向某个表中添加了重复列名，查询也不会失败。

目前所见的连接查询，也可以采用下面这种形式编写：

```
SELECT *
    FROM weather INNER JOIN cities ON (weather.city = cities.name);
```

这种语法不如上面的语法常用，不过这里展示它是为了帮助理解后续主题。

现在来看看如何把 Hayward 的记录找回来。希望查询扫描`weather`表，并为其中每一行查找匹配的`cities`行。如果找不到匹配行，希望用一些“空值”代替`cities`表中的列。
这种查询称为外连接。（此前所见的连接都是内连接。）命令如下：

```
SELECT *
    FROM weather LEFT OUTER JOIN cities ON (weather.city = cities.
name);

     city      | temp_lo | temp_hi | prcp |    date    |     name
    | location
---------------+---------+---------+------+------------+-----------
----+-----------
 Hayward       |      37 |      54 |      | 1994-11-29 |
    |
 San Francisco |      46 |      50 | 0.25 | 1994-11-27 | San
 Francisco | (-194,53)
 San Francisco |      43 |      57 |    0 | 1994-11-29 | San
 Francisco | (-194,53)
(3 rows)
```

这条查询称为左外连接，因为连接操作符左边的表，其每一行都至少在输出中出现一次，而右边的表只有与左表某行匹配的行才会输出。在输出某个没有右表匹配行的左表行时，右表的各列会用空值（null）代替。

练习：. 还有右外连接和全外连接。试着找出它们能做什么。

也可以把一个表与它自身连接，这称为自连接。例如，假设要找出处于其他天气记录温度范围内的所有天气记录。为此，需要比较temp_lo和temp_hi两列在每个weather行中的值与temp_lo和temp_hi两列在其他所有weather行中的值。可以使用下面的查询：

```
SELECT W1.city, W1.temp_lo AS low, W1.temp_hi AS high,
    W2.city, W2.temp_lo AS low, W2.temp_hi AS high
    FROM weather W1, weather W2
    WHERE W1.temp_lo < W2.temp_lo
    AND W1.temp_hi > W2.temp_hi;

     city      | low | high |     city      | low | high
---------------+-----+------+---------------+-----+------
 San Francisco |  43 |   57 | San Francisco |  46 |   50
 Hayward       |  37 |   54 | San Francisco |  46 |   50
(2 rows)
```

这里把 weather 表重新标记为W1和W2，以区分连接的左侧和右侧。在其他查询中也可以使用这种别名来减少输入，例如：

```
SELECT *
    FROM weather w, cities c
    WHERE w.city = c.name;
```

你会经常遇到这种缩写方式。

2.7. 聚合函数

与大多数其他关系数据库产品一样，PostgreSQL支持聚合函数。
聚合函数从多行输入中计算出单个结果。例如，有一些聚合函数可在一组行上计算count（计数）、sum（和）、avg（平均值）、max（最大值）和min（最小值）。

例如，我们可以用下面的语句找出所有记录中最低温度读数的最高值：

```
SELECT max(temp_lo) FROM weather;

 max
-----
  46
(1 row)
```

如果我们想知道该读数出现在哪个城市（或哪些城市），可能会尝试：

```
SELECT city FROM weather WHERE temp_lo = max(temp_lo);     错误
```

但这样行不通，因为聚合函数max不能用于 WHERE子句中（存在这个限制是因为WHERE 子句决定哪些行会被纳入聚合计算；因此显然它必须在聚合函数计算之前求值。）不过，这类情况下通常可以把查询改写成能够得到所需结果的形式，这里我们通过使用子查询来做到这一点：

```
SELECT city FROM weather
    WHERE temp_lo = (SELECT max(temp_lo) FROM weather);

     city
---------------
```

```
 San Francisco
(1 row)
```

这样做是可以的，因为子查询是一次独立的计算，它会与外层查询分开计算自己的聚合。

聚合与GROUP BY子句结合使用也很有用。例如，可以通过下面的查询，得到每个城市所观测到的最低温度的最大值：

```
SELECT city, max(temp_lo)
    FROM weather
    GROUP BY city;

     city      | max
---------------+-----
 Hayward       |  37
 San Francisco |  46
(2 rows)
```

这会为每个城市输出一行。每个聚合结果都是针对与该城市匹配的表行计算的。
这些分组行可以用以下子句过滤：HAVING：

```
SELECT city, max(temp_lo)
    FROM weather
    GROUP BY city
    HAVING max(temp_lo) < 40;

  city   | max
---------+-----
 Hayward |  37
(1 row)
```

这样就只会为所有temp_lo值均小于 40 的城市返回相同的结果。最后，如果只关心名称以“S”开头的城市，可以这样做：

```
SELECT city, max(temp_lo)
    FROM weather
    WHERE city LIKE 'S%'❶
    GROUP BY city
    HAVING max(temp_lo) < 40;
```

❶ LIKE操作符执行模式匹配，详细说明见第 9.7 节。

理解聚合与SQL的WHERE和 HAVING子句之间的相互作用非常重要。WHERE和HAVING的根本区别在于：WHERE在分组和聚合计算之前选择输入行（因此它控制哪些行进入聚合计算），而HAVING在分组和聚合计算之后选择分组行。因此，WHERE子句中不能包含聚合函数；试图用聚合来决定哪些行应该成为聚合的输入是没有意义的。另一方面，HAVING子句总是包含聚合函数。（严格地说，也允许编写不使用聚合的HAVING子句，但那很少有用。同样的条件在WHERE阶段使用会更高效。）

在前面的例子中，我们可以在WHERE中应用城市名限制，因为它不需要聚合。这样比把限制放到HAVING中更高效，因为我们避免了对所有未通过WHERE检查的行进行分组和聚合计算。

2.8. 更新

你可以使用UPDATE命令更新已有的行。假设你发现 11 月 28 日之后的温度读数全部偏差了 2 度，那么就可以按下面的方式改正数据：

```
UPDATE weather
```

```
    SET temp_hi = temp_hi - 2,  temp_lo = temp_lo - 2
    WHERE date > '1994-11-28';
```

看看数据的新状态：

```
SELECT * FROM weather;

     city      | temp_lo | temp_hi | prcp |    date
---------------+---------+---------+------+------------
 San Francisco |      46 |      50 | 0.25 | 1994-11-27
 San Francisco |      41 |      55 |    0 | 1994-11-29
 Hayward       |      35 |      52 |      | 1994-11-29
(3 rows)
```

2.9. 删除

可以使用`DELETE`命令从表中移除行。假设你不再关心 Hayward 的天气，那么可以用下面的方法把这些行从表中删除：

```
DELETE FROM weather WHERE city = 'Hayward';
```

所有属于 Hayward 的天气记录都会被删除。

```
SELECT * FROM weather;

     city      | temp_lo | temp_hi | prcp |    date
---------------+---------+---------+------+------------
 San Francisco |      46 |      50 | 0.25 | 1994-11-27
 San Francisco |      41 |      55 |    0 | 1994-11-29
(2 rows)
```

对如下形式的语句必须保持警惕：

```
DELETE FROM tablename;
```

如果没有限定条件，`DELETE`将从指定表中删除所有行，使其变为空表。
系统在执行前不会请求你确认！

第 3 章 高级特性

3.1. 简介

在前一章中，我们已经介绍了如何使用SQL在PostgreSQL中存储和访问数据的基础知识。现在我们将讨论SQL的一些更高级特性，它们能够简化管理，并防止数据丢失或损坏。最后，我们还将看看一些PostgreSQL扩展。

本章有时会引用第 2 章中的示例，对其加以修改或改进，因此事先读过那一章会很有帮助。本章中的一些示例也可以在教程目录中的`advanced.sql`文件中找到。该文件还包含一些要装载的样例数据，这里不再重复。（关于如何使用该文件，参见第 2.1 节。）

3.2. 视图

回过头看第 2.6 节中的查询。假设天气记录和城市位置的组合清单对你的应用特别有用，但你又不想每次需要它时都键入这条查询。你可以在该查询之上创建一个视图，为该查询命名，以后就可以像引用普通表一样引用它：

```
CREATE VIEW myview AS
    SELECT city, temp_lo, temp_hi, prcp, date, location
        FROM weather, cities
        WHERE city = name;

SELECT * FROM myview;
```

大量使用视图是良好的 SQL 数据库设计的一个关键方面。视图允许你通过一致的接口封装表结构的细节，而这些细节可能会随着应用演进而变化。

几乎凡是真实表可以使用的地方，都可以使用视图。在其他视图之上再构建视图也很常见。

3.3. 外键

回想第 2 章中的`weather`和`cities`表。考虑这样一个问题：你希望确保没有人能向`weather`表中插入在`cities`表里没有匹配项的行。这称为维护数据的引用完整性。在较为简陋的数据库系统中，这通常是通过先查看`cities`表、检查是否存在匹配记录，然后再插入新的`weather`记录或拒绝插入来实现的（如果系统支持这种做法的话）。这种做法有很多问题，而且很不方便，所以PostgreSQL可以替你完成这件事。

新的表声明如下：

```
CREATE TABLE cities (
        city     varchar(80) primary key,
        location point
);

CREATE TABLE weather (
        city      varchar(80) references cities(city),
        temp_lo   int,
        temp_hi   int,
        prcp      real,
        date      date
);
```

现在试着插入一条无效记录：

```
INSERT INTO weather VALUES ('Berkeley', 45, 53, 0.0, '1994-11-28');
```

```
ERROR:  insert or update on table "weather" violates foreign key
 constraint "weather_city_fkey"
DETAIL:  Key (city)=(Berkeley) is not present in table "cities".
```

外键的行为可以针对应用进行精细调整。本教程不再超出这个简单例子继续展开，
更多信息请参见第 5 章。正确使用外键肯定能提高数据库应用的质量，因此强烈建议你了解它们。

3.4. 事务

事务是所有数据库系统中的一个基本概念。事务的要点在于，
它把多个步骤打包成一个单一的、要么全部成功要么全部失败的操作。
步骤之间的中间状态对其他并发事务不可见；如果发生某种故障使事务无法完成，
那么其中任何一步都不会对数据库产生影响。

例如，考虑一个银行数据库，其中保存着各个客户账户的余额，
以及各营业网点的总存款余额。假设我们要记录一笔从Alice的账户向Bob的账户支付100.00美元的款项。大幅简化后，相应的 SQL 命令可能是：

```
UPDATE accounts SET balance = balance - 100.00
    WHERE name = 'Alice';
UPDATE branches SET balance = balance - 100.00
    WHERE name = (SELECT branch_name FROM accounts WHERE name =
 'Alice');
UPDATE accounts SET balance = balance + 100.00
    WHERE name = 'Bob';
UPDATE branches SET balance = balance + 100.00
    WHERE name = (SELECT branch_name FROM accounts WHERE name =
 'Bob');
```

这些命令的细节在这里并不重要；重要的是，为了完成这个相当简单的操作，
涉及了若干次彼此独立的更新。银行管理人员会希望确信这些更新要么全部发生，
要么一个也不发生。显然，不能因为系统故障导致Bob收到了100.00美元，
而Alice那边却没有被扣款。反过来，如果Alice被扣了款而Bob没有入账，她也不会满意。
我们需要保证：如果操作进行到一半出了问题，到目前为止已执行的步骤都不会生效。
把这些更新归为一个事务，就能提供这种保证。事务被称为原子的：从其他事务的角度看，
它要么完整发生，要么根本不发生。

我们还希望保证，一旦事务完成并得到数据库系统确认，它确实已经被永久记录下来，
即便紧接着发生崩溃也不会丢失。例如，如果我们正在记录Bob的一次现金提款，
我们当然不希望他刚走出银行大门，对他账户的扣款就在崩溃后消失。事务型数据库保证，
在把事务报告为完成之前，会先将该事务所做的全部更新记入永久存储（即磁盘）。

事务型数据库的另一个重要性质与原子更新的概念密切相关：当多个事务并发运行时，
每个事务都不应该看到其他事务尚未完成的更改。例如，
如果某个事务正在统计所有营业网点的余额，就不能让它只算进Alice所在网点的扣款，
却没有算进Bob所在网点的入账，反过来也不行。因此，
事务不仅在对数据库的持久影响上必须是全有或全无的，
在其发生过程中的可见性上也必须如此。一个未结束事务到目前为止所做的更新，
对其他事务不可见；等到该事务完成时，所有更新会同时变得可见。

在PostgreSQL中，可以通过将事务中的 SQL 命令置于`BEGIN`和`COMMIT`命令之间来建立一个事务。因此，我们的银行事务实际上会是这样：

```
BEGIN;
UPDATE accounts SET balance = balance - 100.00
    WHERE name = 'Alice';
-- etc etc
COMMIT;
```

如果事务执行到一半时，我们决定不想提交了（例如刚刚发现Alice的余额变成了负数），就可以发出ROLLBACK而不是COMMIT，这样到目前为止的全部更新都会被取消。

PostgreSQL实际上把每条 SQL 语句都视为在一个事务中执行。如果你没有显式发出BEGIN，那么每条独立语句外围都会隐式包上一对BEGIN和（若成功）COMMIT。由BEGIN和COMMIT包围起来的一组语句，有时称为事务块。

> **注意**
>
> 某些客户端库会自动发出BEGIN和COMMIT命令，
> 因此你可能在没有主动要求的情况下，也得到了事务块的效果。
> 请查阅你所使用接口的文档。

也可以借助保存点，以更细粒度控制事务中的语句。
保存点允许你有选择地丢弃事务的一部分，同时提交其余部分。使用SAVEPOINT定义保存点之后，如有需要，可以用ROLLBACK TO回滚到该保存点。事务中从定义保存点到回滚到它之间所做的数据库修改都会被丢弃，而早于该保存点的修改会被保留。

回滚到某个保存点之后，该保存点仍然保持定义状态，因此你可以多次回滚到它。反过来，如果你确定不再需要回滚到某个保存点，可以将其释放，以便系统回收一些资源。请记住，无论是释放某个保存点，还是回滚到某个保存点，
都会自动释放在它之后定义的所有保存点。

所有这些都发生在事务块内部，因此其他数据库会话都看不到。等到你提交整个事务块时，被提交的动作才会作为一个整体对其他会话可见，而被回滚的动作则永远不会可见。

回到银行数据库，假设我们从Alice的账户扣除了100.00美元，并记入Bob的账户，后来却发现其实应该记入Wally的账户。我们可以像下面这样利用保存点来处理：

```
BEGIN;
UPDATE accounts SET balance = balance - 100.00
    WHERE name = 'Alice';
SAVEPOINT my_savepoint;
UPDATE accounts SET balance = balance + 100.00
    WHERE name = 'Bob';
-- oops ... forget that and use Wally's account
ROLLBACK TO my_savepoint;
UPDATE accounts SET balance = balance + 100.00
    WHERE name = 'Wally';
COMMIT;
```

当然，这个例子是过度简化的，但在事务块中借助保存点可以进行大量控制。此外，对于一个由于错误而被系统置为中止状态的事务块，要重新取得控制权，
唯一的办法就是使用ROLLBACK TO；否则就只能把整个事务完全回滚并重新开始。

3.5. 窗口函数

窗口函数会在一组与当前行存在某种关联的表行上执行计算。
这与聚合函数能够完成的计算类型类似。但与普通聚合函数不同，
使用窗口函数不会把多行分组为一条输出行 – 各行仍然保留各自的独立身份。在幕后，窗口函数能够访问的不仅仅是查询结果中的当前行。

下面的例子说明如何将每个雇员的工资与其所在部门的平均工资进行比较：

```
SELECT depname, empno, salary, avg(salary) OVER (PARTITION BY
 depname) FROM empsalary;
```

```
  depname   | empno | salary |          avg
-----------+-------+--------+----------------------
 develop   |    11 |   5200 | 5020.0000000000000000
 develop   |     7 |   4200 | 5020.0000000000000000
 develop   |     9 |   4500 | 5020.0000000000000000
 develop   |     8 |   6000 | 5020.0000000000000000
 develop   |    10 |   5200 | 5020.0000000000000000
 personnel |     5 |   3500 | 3700.0000000000000000
 personnel |     2 |   3900 | 3700.0000000000000000
 sales     |     3 |   4800 | 4866.6666666666666667
 sales     |     1 |   5000 | 4866.6666666666666667
 sales     |     4 |   4800 | 4866.6666666666666667
(10 rows)
```

前三个输出列直接来自表empsalary，并且该表中的每一行对应一条输出行。
第四列表示在所有depname值与当前行相同的表行上求得的平均值。（这其实与普通的avg聚合函数是同一个函数，但OVER子句使它被当作窗口函数处理，
并在一组适当的行上计算。）

窗口函数调用总是包含一个紧跟在窗口函数名和参数之后的OVER子句。
这正是它在语法上区别于普通函数或聚合函数的地方。OVER
子句精确决定查询中的哪些行会被拆分出来供窗口函数处理。OVER内部的PARTITION BY列表指定将具有相同PARTITION BY表达式值的行划分为组，也就是分区。对于每一行，窗口函数都是在与当前行处于同一分区的那些行上计算的。

你也可以用以下子句来控制窗口函数处理行的顺序：ORDER BY，将其放在OVER内部。（窗口中的ORDER BY甚至不必与行的输出顺序一致。）下面是一个例子：

```
SELECT depname, empno, salary, rank() OVER (PARTITION BY depname
 ORDER BY salary DESC) FROM empsalary;

  depname   | empno | salary | rank
-----------+-------+--------+------
 develop   |     8 |   6000 |    1
 develop   |    10 |   5200 |    2
 develop   |    11 |   5200 |    2
 develop   |     9 |   4500 |    4
 develop   |     7 |   4200 |    5
 personnel |     2 |   3900 |    1
 personnel |     5 |   3500 |    2
 sales     |     1 |   5000 |    1
 sales     |     4 |   4800 |    2
 sales     |     3 |   4800 |    2
(10 rows)
```

如上所示，rank函数会为当前行所在分区中每个不同的ORDER BY值，按ORDER BY子句定义的顺序产生一个数值排名。rank不需要显式参数，因为它的行为完全由OVER子句定义。

窗口函数所考虑的行，是查询的FROM子句产生并经WHERE、GROUP BY和HAVING子句（如果有）过滤后的那个“虚拟表”中的行。例如，由于不满足WHERE条件而被删除的行，
不会被任何窗口函数看到。一个查询可以包含多个窗口函数，它们可以通过不同的OVER
子句以不同方式划分数据，但它们都作用于这个虚拟表所定义的同一组行。

我们已经看到，如果行的顺序并不重要，就可以省略ORDER BY。PARTITION BY也可以省略，这时所有行构成一个分区。

与窗口函数相关的另一个重要概念是：对于每一行，在其所在分区内有一组行，称为它的
窗口帧。许多（但不是全部）窗口函数只作用于窗口帧中的行，
而不是整个分区中的所有行。默认情况下，如果提供了ORDER BY，则窗口帧包含从分区起

始处直到当前行的所有行，再加上根据ORDER BY子句与当前行相等的所有后续行。如果省略ORDER BY，默认窗口帧则包含该分区中的所有行。[1]下面是一个使用sum的例子：

```
SELECT salary, sum(salary) OVER () FROM empsalary;

 salary |  sum
--------+-------
   5200 | 47100
   5000 | 47100
   3500 | 47100
   4800 | 47100
   3900 | 47100
   4200 | 47100
   4500 | 47100
   4800 | 47100
   6000 | 47100
   5200 | 47100
(10 rows)
```

上面由于OVER子句中没有ORDER BY，窗口帧与分区相同；而在没有PARTITION BY的情况下，这个分区就是整张表。换言之，每个求和都是在整张表上计算的，
因此每条输出行得到的结果都相同。但是如果加上ORDER BY子句，结果就会大不一样：

```
SELECT salary, sum(salary) OVER (ORDER BY salary) FROM empsalary;

 salary |  sum
--------+-------
   3500 |  3500
   3900 |  7400
   4200 | 11600
   4500 | 16100
   4800 | 25700
   4800 | 25700
   5000 | 30700
   5200 | 41100
   5200 | 41100
   6000 | 47100
(10 rows)
```

这里的求和是从第一条（最低）工资一直累加到当前行，
包括与当前行工资相同的所有重复值（注意那些重复工资对应的结果）。

窗口函数只允许出现在查询的SELECT列表和ORDER BY子句中。它们不允许出现在其他地方，例如GROUP BY、HAVING和WHERE子句中。这是因为从逻辑上讲，它们在这些子句处理完成之后才执行。另外，窗口函数在普通聚合函数之后执行。
这意味着在窗口函数的参数中包含聚合函数调用是合法的，反过来则不行。

如果需要在窗口计算执行之后再过滤或分组行，可以使用子查询。例如：

```
SELECT depname, empno, salary, enroll_date
FROM
  (SELECT depname, empno, salary, enroll_date,
     rank() OVER (PARTITION BY depname ORDER BY salary DESC, empno)
 AS pos
     FROM empsalary
  ) AS ss
WHERE pos < 3;
```

[1] 还可以用其他方式定义窗口帧，但本教程不涉及这些选项。细节见第 4.2.8 节。

上面的查询只显示内部查询中rank小于3的那些行（也就是每个部门的前两行）。

当查询涉及多个窗口函数时，可以为每个函数分别写一个独立的OVER子句；
但如果多个函数都需要相同的窗口行为，这样写既重复，又容易出错。替代方法是，在
WINDOW子句中给每一种窗口行为命名，然后在OVER中引用它。例如：

```
SELECT sum(salary) OVER w, avg(salary) OVER w
  FROM empsalary
  WINDOW w AS (PARTITION BY depname ORDER BY salary DESC);
```

关于窗口函数的更多细节可以在第 4.2.8 节、第 9.21 节、第 7.2.4 节以及SELECT参考页中找到。

3.6. 继承

继承是面向对象数据库中的一个概念。它为数据库设计打开了一些有趣的新可能性。

让我们创建两个表：表cities和表capitals。很自然，首都也是城市，
因此当你列出所有城市时，应该有某种办法能隐式地把首都也显示出来。如果你很有巧思，
也许会想出这样的方案：

```
CREATE TABLE capitals (
  name       text,
  population real,
  altitude  int,    -- (in ft)
  state      char(2)
);

CREATE TABLE non_capitals (
  name       text,
  population real,
  altitude  int     -- (in ft)
);

CREATE VIEW cities AS
  SELECT name, population, altitude FROM capitals
    UNION
  SELECT name, population, altitude FROM non_capitals;
```

就查询而言，这样做还行，但一旦需要更新多行，它就会变得很麻烦。

更好的解决办法是：

```
CREATE TABLE cities (
  name       text,
  population real,
  altitude  int     -- (in ft)
);

CREATE TABLE capitals (
  state      char(2)
) INHERITS (cities);
```

在这种情况下，capitals的一行会从它的父表cities继承全部列（name、population
和altitude）。列name的类型是text，这是PostgreSQL内置的一种变长字符串类型。
州首府有一个额外的列state，用来表示所在的州。在PostgreSQL中，
一个表可以从零个或多个其他表继承。

例如，下面的查询会找出所有海拔超过500英尺的城市名称，其中也包括州首府：

```
SELECT name, altitude
  FROM cities
  WHERE altitude > 500;
```

返回：

```
   name    | altitude
-----------+-----------
 Las Vegas |      2174
 Mariposa  |      1953
 Madison   |       845
(3 rows)
```

另一方面，下面的查询会找出所有海拔500英尺或更高且不是州首府的城市：

```
SELECT name, altitude
    FROM ONLY cities
    WHERE altitude > 500;
```

```
   name    | altitude
-----------+-----------
 Las Vegas |      2174
 Mariposa  |      1953
(2 rows)
```

这里，放在cities前面的ONLY表示该查询只针对cities表执行，而不包括继承层次中位于cities之下的表。我们前面已经讨论过的许多命令 – SELECT、UPDATE和DELETE – 都支持这种ONLY记法。

> **注意**
>
> 尽管继承经常很有用，但它尚未与唯一约束或外键集成，这限制了它的实用性。
> 更多细节见第 5.8 节。

3.7. 小结

PostgreSQL还有许多特性在这个面向SQL新用户的入门教程中没有涉及。
本书余下部分会更详细地讨论这些特性。

如果你觉得还需要更多入门材料，请访问 PostgreSQL 官方网站[2]，其中有指向更多资源的链接。

[2] https://www.postgresql.org

部分 II. SQL 语言

本部分描述SQL语言在PostgreSQL中的用法。我们首先介绍SQL的一般语法，然后说明如何创建表、如何填充数据库以及如何查询数据库。中间部分列出可在SQL命令中使用的数据类型和函数。最后，我们讨论数据库调优中的若干重要方面。

本部分的内容安排方式使初学者能够从头到尾依次阅读，
并在不必过多向后查阅的情况下充分理解这些主题。各章力求自成体系，
因此高级用户也可以按需单独阅读。相关内容按主题单元以叙述方式展开。
希望获得某个命令完整描述的读者，建议查阅第 VI 部分。

读者应当知道如何连接到PostgreSQL数据库并发出SQL命令。不熟悉这些内容的读者，
建议先阅读第 I 部分。SQL命令通常通过PostgreSQL的交互式终端程序psql输入，但也可以使用具有类似功能的其他程序。

目录

4. SQL语法 29
4.1. 词法结构 29
4.1.1. 标识符和关键词 29
4.1.2. 常量 30
4.1.3. 操作符 34
4.1.4. 特殊字符 35
4.1.5. 注释 35
4.1.6. 操作符优先级 35
4.2. 值表达式 36
4.2.1. 列引用 37
4.2.2. 位置参数 37
4.2.3. 下标 38
4.2.4. 字段选择 38
4.2.5. 操作符调用 38
4.2.6. 函数调用 39
4.2.7. 聚合表达式 39
4.2.8. 窗口函数调用 40
4.2.9. 类型转换 41
4.2.10. 排序规则表达式 42
4.2.11. 标量子查询 43
4.2.12. 数组构造器 43
4.2.13. 行构造器 44
4.2.14. 表达式计算规则 46
4.3. 调用函数 47
4.3.1. 使用位置记法 47
4.3.2. 使用命名记法 48
4.3.3. 使用混合记法 48
5. 数据定义 49
5.1. 表基础 49
5.2. 默认值 50
5.3. 约束 51
5.3.1. 检查约束 51
5.3.2. 非空约束 52
5.3.3. 唯一约束 53
5.3.4. 主键 54
5.3.5. 外键 54
5.3.6. 排他约束 57
5.4. 系统列 57
5.5. 修改表 58
5.5.1. 增加列 58
5.5.2. 移除列 59
5.5.3. 增加约束 59
5.5.4. 移除约束 59
5.5.5. 更改列的默认值 60
5.5.6. 修改列的数据类型 60
5.5.7. 重命名列 60
5.5.8. 重命名表 60
5.6. 权限 60
5.7. 模式 61
5.7.1. 创建模式 62
5.7.2. 公共模式 62
5.7.3. 模式搜索路径 63
5.7.4. 模式和权限 64
5.7.5. 系统目录模式 64
5.7.6. 使用方式 64

5.7.7. 可移植性 ... 65
5.8. 继承 ... 65
5.8.1. 注意事项 ... 67
5.9. 分区 ... 68
5.9.1. 概述 ... 68
5.9.2. 实现分区 ... 68
5.9.3. 分区管理 ... 71
5.9.4. 分区和约束排除 ... 72
5.9.5. 备选分区方法 ... 73
5.9.6. 注意事项 ... 74
5.10. 外部数据 ... 74
5.11. 其他数据库对象 ... 75
5.12. 依赖跟踪 ... 75
6. 数据操纵 ... 77
6.1. 插入数据 ... 77
6.2. 更新数据 ... 78
6.3. 删除数据 ... 78
6.4. 从被修改的行中返回数据 ... 79
7. 查询 ... 80
7.1. 概述 ... 80
7.2. 表表达式 ... 80
7.2.1. FROM子句 ... 80
7.2.2. WHERE子句 ... 86
7.2.3. GROUP BY和HAVING子句 ... 87
7.2.4. 窗口函数处理 ... 89
7.3. 选择列表 ... 90
7.3.1. 选择列表项 ... 90
7.3.2. 列标签 ... 90
7.3.3. DISTINCT ... 91
7.4. 组合查询 ... 91
7.5. 行排序 ... 92
7.6. LIMIT和OFFSET ... 93
7.7. VALUES列表 ... 93
7.8. WITH查询（公共表表达式） ... 94
7.8.1. WITH中的SELECT ... 94
7.8.2. WITH中的数据修改语句 ... 97
8. 数据类型 ... 99
8.1. 数字类型 ... 100
8.1.1. 整数类型 ... 101
8.1.2. 任意精度数值 ... 101
8.1.3. 浮点类型 ... 102
8.1.4. serial 类型 ... 103
8.2. 货币类型 ... 104
8.3. 字符类型 ... 104
8.4. 二进制数据类型 ... 106
8.4.1. bytea的十六进制格式 ... 106
8.4.2. bytea的转义格式 ... 107
8.5. 日期/时间类型 ... 108
8.5.1. 日期/时间输入 ... 109
8.5.2. 日期/时间输出 ... 112
8.5.3. 时区 ... 113
8.5.4. 间隔输入 ... 114
8.5.5. 间隔输出 ... 115
8.6. 布尔类型 ... 116
8.7. 枚举类型 ... 117
8.7.1. 枚举类型的声明 ... 117
8.7.2. 排序 ... 117

8.7.3. 类型安全性 118
8.7.4. 实现细节 118
8.8. 几何类型 119
8.8.1. 点 119
8.8.2. 线段 119
8.8.3. 方框 119
8.8.4. 路径 120
8.8.5. 多边形 120
8.8.6. 圆 120
8.9. 网络地址类型 120
8.9.1. `inet` 121
8.9.2. `cidr` 121
8.9.3. `inet` 与 `cidr` 122
8.9.4. `macaddr` 122
8.10. 位串类型 122
8.11. 文本搜索类型 123
8.11.1. `tsvector` 123
8.11.2. `tsquery` 124
8.12. UUID类型 125
8.13. XML类型 126
8.13.1. 创建XML值 126
8.13.2. 编码处理 127
8.13.3. 访问XML值 127
8.14. 数组 127
8.14.1. 数组类型的声明 127
8.14.2. 数组值输入 128
8.14.3. 访问数组 129
8.14.4. 修改数组 131
8.14.5. 在数组中搜索 133
8.14.6. 数组输入和输出语法 134
8.15. 复合类型 135
8.15.1. 复合类型的声明 135
8.15.2. 构造组合值 136
8.15.3. 访问复合类型 137
8.15.4. 修改组合值 137
8.15.5. 在查询中使用复合类型 137
8.15.6. 复合类型的输入和输出语法 139
8.16. 范围类型 140
8.16.1. 内置范围类型 141
8.16.2. 示例 141
8.16.3. 包含界限与排除界限 141
8.16.4. 无限（无界）范围 141
8.16.5. 范围输入/输出 142
8.16.6. 构造范围 143
8.16.7. 离散范围类型 143
8.16.8. 定义新的范围类型 143
8.16.9. 索引 144
8.16.10. 范围上的约束 144
8.17. JSON 类型 145
8.18. 对象标识符类型 145
8.19. 伪类型 147
9. 函数和操作符 148
9.1. 逻辑操作符 148
9.2. 比较操作符 148
9.3. 数学函数和操作符 150
9.4. 字符串函数和操作符 154
9.5. 二进制串函数和操作符 166

9.6. 位串函数和操作符 168
9.7. 模式匹配 169
9.7.1. LIKE 170
9.7.2. SIMILAR TO正则表达式 171
9.7.3. POSIX正则表达式 171
9.8. 数据类型格式化函数 182
9.9. 日期/时间函数和操作符 188
9.9.1. EXTRACT, date_part 193
9.9.2. date_trunc 196
9.9.3. AT TIME ZONE 197
9.9.4. 当前日期/时间 197
9.9.5. 延时执行 199
9.10. 枚举支持函数 199
9.11. 几何函数和操作符 200
9.12. 网络地址函数和操作符 203
9.13. 文本搜索函数和操作符 206
9.14. XML 函数 210
9.14.1. 产生 XML 内容 210
9.14.2. XML 谓词 214
9.14.3. 处理 XML 215
9.14.4. 将表映射到 XML 216
9.15. JSON 函数 219
9.16. 序列操作函数 220
9.17. 条件表达式 222
9.17.1. CASE 222
9.17.2. COALESCE 223
9.17.3. NULLIF 223
9.17.4. GREATEST和LEAST 224
9.18. 数组函数和操作符 224
9.19. 范围函数和操作符 227
9.20. 聚合函数 229
9.21. 窗口函数 231
9.22. 子查询表达式 233
9.22.1. EXISTS 233
9.22.2. IN 233
9.22.3. NOT IN 234
9.22.4. ANY/SOME 234
9.22.5. ALL 235
9.22.6. 单一行比较 235
9.23. 行和数组比较 235
9.23.1. IN 236
9.23.2. NOT IN 236
9.23.3. ANY/SOME（数组） 236
9.23.4. ALL（数组） 237
9.23.5. 按行比较 237
9.24. 集合返回函数 238
9.25. 系统信息函数 240
9.26. 系统管理函数 249
9.26.1. 配置设定函数 249
9.26.2. 服务器信号函数 250
9.26.3. 备份控制函数 250
9.26.4. 恢复控制函数 252
9.26.5. 快照同步函数 253
9.26.6. 数据库对象管理函数 253
9.26.7. 通用文件访问函数 255
9.26.8. 咨询锁函数 256
9.27. 触发器函数 257

10. 类型转换 259
10.1. 概述 259
10.2. 操作符 260
10.3. 函数 263
10.4. 值存储 266
10.5. UNION、CASE及相关结构 267
11. 索引 269
11.1. 简介 269
11.2. 索引类型 270
11.3. 多列索引 271
11.4. 索引和ORDER BY 272
11.5. 组合多个索引 273
11.6. 唯一索引 273
11.7. 表达式索引 274
11.8. 部分索引 274
11.9. 操作符类和操作符族 276
11.10. 索引和排序规则 277
11.11. 检查索引使用情况 278
12. 全文搜索 280
12.1. 介绍 280
12.1.1. 什么是文档? 280
12.1.2. 基本文本匹配 281
12.1.3. 配置 282
12.2. 表和索引 283
12.2.1. 搜索表 283
12.2.2. 创建索引 283
12.3. 控制文本搜索 284
12.3.1. 解析文档 284
12.3.2. 解析查询 285
12.3.3. 搜索结果排名 286
12.3.4. 高亮结果 288
12.4. 附加特性 289
12.4.1. 操纵文档 290
12.4.2. 操纵查询 290
12.4.3. 用于自动更新的触发器 292
12.4.4. 收集文档统计数据 293
12.5. 解析器 294
12.6. 词典 295
12.6.1. 停用词 296
12.6.2. 简单词典 297
12.6.3. 同义词词典 298
12.6.4. 分类词典 300
12.6.5. Ispell 词典 302
12.6.6. Snowball 词典 302
12.7. 配置示例 303
12.8. 测试和调试文本搜索 304
12.8.1. 配置测试 304
12.8.2. 解析器测试 307
12.8.3. 词典测试 308
12.9. GiST 和 GIN 索引类型 308
12.10. psql 支持 309
12.11. 限制 312
12.12. 从 8.3 之前的文本搜索迁移 312
13. 并发控制 314
13.1. 介绍 314
13.2. 事务隔离 314
13.2.1. 读已提交隔离级别 315
13.2.2. 可重复读隔离级别 316

13.2.3. 可串行化隔离级别 317
13.3. 显式锁定 318
13.3.1. 表级锁 318
13.3.2. 行级锁 321
13.3.3. 死锁 321
13.3.4. 咨询锁 322
13.4. 应用级别的数据一致性检查 322
13.4.1. 用可串行化事务强制一致性 323
13.4.2. 使用显式阻塞锁强制一致性 323
13.5. 注意事项 324
13.6. 锁定和索引 324
14. 性能提示 325
14.1. 使用EXPLAIN 325
14.1.1. EXPLAIN基础 325
14.1.2. EXPLAIN ANALYZE 330
14.1.3. 注意事项 334
14.2. 规划器使用的统计信息 335
14.3. 用显式JOIN子句控制规划器 336
14.4. 填充一个数据库 338
14.4.1. 禁用自动提交 338
14.4.2. 使用COPY 338
14.4.3. 移除索引 338
14.4.4. 移除外键约束 339
14.4.5. 增加maintenance_work_mem 339
14.4.6. 增加checkpoint_segments 339
14.4.7. 禁用 WAL 归档和流复制 339
14.4.8. 事后运行ANALYZE 339
14.4.9. 关于pg_dump的一些注记 340
14.5. 非持久设置 340

第 4 章 SQL语法

本章描述 SQL 的语法。它是理解后续各章的基础，后续各章将详细介绍如何使用 SQL 命令定义和修改数据。

我们也建议已经熟悉 SQL 的用户仔细阅读本章，因为其中包含一些在各 SQL 数据库之间实现并不一致的规则和概念，以及一些 PostgreSQL 特有的规则和概念。

4.1. 词法结构

SQL 输入由一系列命令组成。一个命令由一系列词元组成，并以分号（“;”）终止。输入流的结束也会终止一个命令。哪些词元是合法的，取决于具体命令的语法。

一个词元可以是关键词、标识符、带引号的标识符、字面量（或常量），
或者特殊字符符号。词元通常由空白（空格、制表符、换行）分隔，但如果不存在歧义，
则不必如此（一般只有特殊字符与其他类型的词元相邻时才会这样）。

例如，下面是一个（语法上）合法的SQL输入：

```
SELECT * FROM MY_TABLE;
UPDATE MY_TABLE SET A = 5;
INSERT INTO MY_TABLE VALUES (3, 'hi there');
```

这是一个由三个命令组成的序列，每行一个命令（虽然这并非必须；
一行中可以包含多个命令，命令也可以有意义地跨多行书写）。

此外，注释也可以出现在 SQL 输入中。它们不是词元，实际上等价于空白。

在哪些词元标识命令、哪些词元是操作数或参数这一点上，SQL 语法并不十分一致。最前面的几个词元通常是命令名，因此在上面的示例中，我们通常会说是“SELECT”、“UPDATE”和“INSERT”命令。但是例如 `UPDATE` 命令总是要求在特定位置出现一个 `SET` 词元，而这种形式的 `INSERT` 也要求有一个 `VALUES` 才算完整。每个命令的精确语法规则见第 VI 部分。

4.1.1. 标识符和关键词

上例中的 `SELECT`、`UPDATE` 或 `VALUES` 这类词元都是关键词，也就是在 SQL 语言中具有固定含义的词。词元 `MY_TABLE` 和 `A` 则是标识符的例子。它们标识表、列或其他数据库对象的名称，这取决于它们所在的命令。因此，它们有时也简称为“名称”。
关键词和标识符具有相同的词法结构，这意味着如果不了解该语言，
就无法判断一个词元究竟是标识符还是关键词。完整的关键词列表可以在附录 C中找到。

SQL 标识符和关键词必须以字母（`a-z`，也包括带变音符的字母和非拉丁字母）或下划线（`_`）开头。标识符或关键词中的后续字符可以是字母、下划线、数字（`0-9`）
或美元符号（`$`）。注意，按照 SQL 标准的字面规定，标识符中不允许使用美元符号，因此它可能会降低应用的可移植性。SQL 标准不会定义包含数字、以下划线开头或以下划线结尾的关键词，因此这种形式的标识符可以安全地避免与未来标准扩展发生冲突。

系统对标识符最多只使用 `NAMEDATALEN`-1 个字节；在命令中可以写更长的名称，但它们会被截断。默认情况下，`NAMEDATALEN` 为 64，因此标识符的最大长度是 63 字节。如果这个限制有问题，可以通过修改 `src/include/pg_config_manual.h` 中的 `NAMEDATALEN` 常量来提高。

关键词和未加引号的标识符不区分大小写。因此：

```
UPDATE MY_TABLE SET A = 5;
```

也可以等价地写成：

```
uPDaTE my_TabLE SeT a = 5;
```

一种常见约定是关键词使用大写、名称使用小写，例如：

```
UPDATE my_table SET a = 5;
```

还有第二类标识符：定界标识符或带引号的标识符。它通过把任意字符序列括在双引号中形成（"）。定界标识符始终是标识符，绝不会是关键词。因此，`"select"` 可以用来引用名为“select”的列或表，而不加引号的 `select` 会被视为关键词，因此在期望表名或列名的位置使用时会导致解析错误。这个示例可以用带引号的标识符写成：

```
UPDATE "my_table" SET "a" = 5;
```

带引号的标识符可以包含除编码为零的字符之外的任意字符。（如果要包含双引号，
就写两个双引号。）这使得可以构造原本不可能的表名或列名，
例如包含空格或和号的名称。长度限制仍然适用。

带引号标识符的一种变体允许包含按代码点标识的转义 Unicode 字符。这种变体在开头双引号之前紧接着写 `U&`（大写或小写的字母 U 后接和号），中间没有空格，例如 `U&"foo"`。（注意这会与操作符 `&` 产生歧义。请在该操作符两侧加空格以避免这个问题。）在引号内，可以通过写一个反斜线后跟 4 位十六进制代码点编号，或者一个反斜线后跟加号再跟 6 位十六进制代码点编号，来指定 Unicode 字符。例如，标识符 `"data"` 可以写成：

```
U&"d\0061t\+000061"
```

下面这个稍微复杂一些的示例，用西里尔字母写出了俄语单词“slon”（大象）：

```
U&"\0441\043B\043E\043D"
```

如果想使用反斜线以外的转义字符，可以在字符串后用 `UESCAPE` 子句指定，例如：

```
U&"d!0061t!+000061" UESCAPE '!'
```

转义字符可以是除十六进制数字、加号、单引号、双引号或空白字符之外的任何单个字符。
注意，转义字符要写在单引号内，而不是双引号内。

如果要在标识符中按字面包含转义字符本身，把它写两次即可。

Unicode 转义语法只有在服务器编码为 `UTF8` 时才有效。使用其他服务器编码时，只能指定 ASCII 范围内的代码点（不超过 `\007F`）。4 位或 6 位转义形式都可以用来指定 UTF-16 代理对，以组合出代码点大于 U+FFFF 的字符，尽管从技术上说，由于有 6 位形式，这样做并无必要。（代理对不会被直接存储，而是会合并成一个代码点，然后以 UTF-8 编码。）

给标识符加上引号也会使其区分大小写，而不加引号的名称总是会被折叠成小写。例如，
标识符 `FOO`、`foo` 和 `"foo"` 在 PostgreSQL 中被视为相同，但 `"Foo"` 和 `"FOO"` 与这三个都不同，彼此之间也不同。（PostgreSQL 将不加引号的名称折叠为小写，这与 SQL 标准不兼容；标准规定，不加引号的名称应折叠为大写。因此，按照标准，`foo` 应当等价于 `"FOO"`，而不是 `"foo"`。如果你想编写可移植的应用，建议对某个特定名称要么始终加引号，要么始终不加引号。）

4.1.2. 常量

在PostgreSQL中有三种隐式类型常量：字符串、位串和数字。也可以为常量指定显式类型，这样系统就能更准确地表示并更高效地处理它们。这些方式将在后续小节中讨论。

4.1.2.1. 字符串常量

SQL 中的字符串常量是由单引号（'）括起的任意字符序列，例如 `'This is a string'`。
要在字符串常量中包含单引号字符，请写两个相邻的单引号，例如 `'Dianne''s horse'`。
注意，这不同于双引号字符（"）。

两个只由空白及至少一个新行分隔的字符串常量会被连接在一起，
并且将作为一个写在一起的字符串常量来对待。例如：

```
SELECT 'foo'
'bar';
```

等同于：

```
SELECT 'foobar';
```

但是：

```
SELECT 'foo'      'bar';
```

则不是合法的语法（这种有些奇怪的行为是SQL指定的，PostgreSQL遵循了该标准）。

4.1.2.2. C风格转义的字符串常量

PostgreSQL也接受“转义”字符串常量，这也是SQL标准的一个扩展。
一个转义字符串常量可以通过在开单引号前面写一个字母`E`（大写或小写形式）来指定，例如`E'foo'`（当一个转义字符串常量跨行时，只在第一个开引号之前写`E`）。
在一个转义字符串内部，一个反斜线字符（`\`）会开始一个 C 风格的反斜线转义序列，在其中反斜线和后续字符的组合表示一个特殊的字节值（如表 4.1中所示）。

表 4.1. 反斜线转义序列

反斜线转义序列	解释
`\b`	退格
`\f`	换页
`\n`	换行
`\r`	回车
`\t`	制表符
`\o`, `\oo`, `\ooo` (*o* = 0 - 7)	八进制字节值
`\xh`, `\xhh` (*h* = 0 - 9, A - F)	十六进制字节值
`\uxxxx`, `\Uxxxxxxxx` (*x* = 0 - 9, A - F)	16 或 32 位十六进制 Unicode 字符值

跟随在一个反斜线后面的任何其他字符被当做其字面意思。因此，要包括一个反斜线字符，
请写两个反斜线（`\\`）。在一个转义字符串中包括一个单引号除了普通方法`''`之外，
还可以写成`\'`。

你有责任确保所创建的字节序列能够构成服务器字符集编码中的有效字符，
尤其是在使用八进制或十六进制转义时。当服务器编码为 UTF-8 时，应改用 Unicode 转义或者第 4.1.2.3 节中说明的另一种 Unicode 转义语法。（另一种做法是手工进行 UTF-8 编码并写出各字节，但那会非常繁琐。）

Unicode 转义语法只有在服务器编码为 `UTF8` 时才能完全有效。使用其他服务器编码时，只能指定 ASCII 范围内的代码点（不超过 `\u007F`）。4 位和 8 位形式都可以用来指定 UTF-16 代理对，以组合出代码点大于 U+FFFF 的字符，尽管从技术上说，由于有 8 位形式，这样做并无必要。（在服务器编码为 `UTF8` 时使用代理对，它们会先合并成一个代码点，然后以 UTF-8 编码。）

小心

如果配置参数standard_conforming_strings为`off`，那么
PostgreSQL对常规字符串常量和转义字符串常量中的反斜线转义都识别。不过，从
PostgreSQL 9.1 开始，该参数的默认值为`on`，意味着只在转义字符串常量中识别反斜

> 线转义。这种行为更兼容标准，但是可能使依赖历史行为（始终识别反斜线转义）的应用无法正常工作。作为一种变通，你可以设置该参数为`off`，但最好修改应用，不再使用反斜线转义。如果你需要使用一个反斜线转义来表示一个特殊字符，为该字符串常量写上一个`E`。
>
> 在`standard_conforming_strings`之外，配置参数escape_string_warning和backslash_quote也决定了如何对待字符串常量中的反斜线。

编码为零的字符不能出现在字符串常量中。

4.1.2.3. 带有 Unicode 转义的字符串常量

PostgreSQL 还支持另一种字符串转义语法，它允许用代码点指定任意 Unicode 字符。Unicode 转义字符串常量在开引号前紧接着写 `U&`（大写或小写字母 U 后跟和号），中间没有任何空白，例如 `U&'foo'`。（注意这会与操作符 `&` 产生歧义。请在该操作符两侧加空格以避免这个问题。）在引号内，可以通过写一个反斜线后跟 4 位十六进制代码点编号，或者一个反斜线后跟加号再跟 6 位十六进制代码点编号，来指定 Unicode 字符。例如，字符串 `'data'` 可以写成

```
U&'d\0061t\+000061'
```

下面这个稍微复杂一些的示例，用西里尔字母写出了俄语单词“slon”（大象）：

```
U&'\0441\043B\043E\043D'
```

如果想要一个不是反斜线的转义字符，可以在字符串之后使用`UESCAPE`子句来指定，例如：

```
U&'d!0061t!+000061' UESCAPE '!'
```

转义字符可以是除十六进制数字、加号、单引号、双引号或空白字符之外的任意单个字符。

Unicode 转义语法只有在服务器编码为 `UTF8` 时才有效。使用其他服务器编码时，只能指定 ASCII 范围内的代码点（不超过 `\007F`）。4 位和 6 位形式都可以用来指定 UTF-16 代理对，以组合出代码点大于 U+FFFF 的字符，尽管从技术上说，由于有 6 位形式，这样做并无必要。（在服务器编码为 `UTF8` 时使用代理对，它们会先合并成一个代码点，然后以 UTF-8 编码。）

此外，字符串常量的 Unicode 转义语法只有在配置参数standard_conforming_strings开启时才有效。这是因为否则这种语法可能会混淆解析 SQL 语句的客户端，从而导致 SQL 注入以及类似的安全问题。如果该参数被设置为 off，这种语法将被拒绝并报错。

如果要在字符串中按字面包含转义字符本身，把它写两次即可。

4.1.2.4. 美元引用的字符串常量

指定字符串常量的标准语法通常很方便，但当所需字符串包含很多单引号或反斜线时，就可能难以理解，因为每个这样的字符都必须写成双份。为了在这种情况下让查询更易读，PostgreSQL提供了另一种编写字符串常量的方法，称为“美元引用”。
美元引用的字符串常量由一个美元符号（`$`）、一个由零个或多个字符组成的可选“标签”、另一个美元符号、构成字符串内容的任意字符序列、一个美元符号、与该美元引用开头相同的标签，以及一个美元符号组成。例如，对于字符串“Dianne's horse”，下面是使用美元引用指定它的两种不同方法：

```
$$Dianne's horse$$
$SomeTag$Dianne's horse$SomeTag$
```

注意，在美元引用的字符串中，单引号不需要转义就可以使用。事实上，美元引用字符串内的任何字符都不会被转义：字符串内容始终按字面书写。
反斜线没有特殊含义，美元符号也没有，除非它是与开头标签相匹配的序列的一部分。

可以通过在每个嵌套层级选择不同的标签，来嵌套美元引用字符串常量。
这种做法最常见于编写函数定义时。例如：

```
$function$
BEGIN
    RETURN ($1 ~ $q$[\t\r\n\v\\]$q$);
END;
$function$
```

这里，序列 `$q$[\t\r\n\v\\]$q$` 表示一个美元引用的字面字符串 `[\t\r\n\v\\]`，当函数体被 PostgreSQL 执行时，它会被识别。但由于该序列与外层美元引用定界符 `$function$` 不匹配，就外层字符串而言，它只不过是常量中的一部分字符。

美元引用字符串的标签（如果有）遵循与未加引号标识符相同的规则，
但不能包含美元符号。标签区分大小写，因此 `$tag$String content$tag$` 是正确的，而 `$TAG$String content$tag$` 则不正确。

如果美元引用字符串位于某个关键词或标识符之后，必须用空白把它们分隔开，
否则美元引用定界符会被当作前一个标识符的一部分。

美元引用不是 SQL 标准的一部分，但在书写复杂字符串字面量时，它往往比符合标准的单引号语法更方便。当要表示的字符串常量位于其他常量内部时，它尤其有用，
这种情况在过程函数定义中很常见。如果使用单引号语法，
上一个示例中的每个反斜线都必须写成四个反斜线；
在解析原始字符串常量时它们会缩减为两个反斜线，
而在函数执行期间重新解析内层字符串常量时又会变成一个。

4.1.2.5. 位串常量

位串常量看起来像常规字符串常量在开引号之前（中间无空白）加了一个`B`（大写或小写形式），例如`B'1001'`。位串常量中允许的字符只有`0`和`1`。

或者，也可以用十六进制记法指定位串常量，即在开头写一个 `X`（大写或小写形式），例如`X'1FF'`。这种记法等价于把每个十六进制位替换成四个二进制位得到的位串常量。

两种形式的位串常量可以以常规字符串常量相同的方式跨行继续。
美元引用不能被用在位串常量中。

4.1.2.6. 数字常量

数字常量接受下列一般形式：

digits
digits.[*digits*][e[+-]*digits*]
[*digits*].*digits*[e[+-]*digits*]
*digits*e[+-]*digits*

其中 *digits* 是一个或多个十进制数字（0 到 9）。若使用小数点，则小数点前后至少有一侧必须有数字。若使用指数标记（e），其后至少必须有一位数字。
常量中不能嵌入空格或其他字符。请注意，前导正号或负号实际上不属于常量本身，
而是作用于常量的操作符。

这些是合法数字常量的示例：

42
3.5
4.
.001
5e2
1.925e-3

如果一个不包含小数点和指数的数字常量的值适合类型integer（32 位），它首先被假定为类型integer。否则如果它的值适合类型bigint（64 位），它被假定为类型bigint。再否则它会被取做类型numeric。
包含小数点和/或指数的常量总是首先被假定为类型numeric。

为数字常量最初指定的数据类型，只是类型解析算法的起点。在大多数情况下，
常量会根据上下文被自动强制转换成最合适的类型。必要时，
你可以通过类型转换强制把一个数值解释为特定的数据类型。例如，
可以通过下面的写法强制把一个数值当作类型 real（float4）处理：

```
REAL '1.23'  -- 字符串形式
1.23::REAL   -- PostgreSQL（历史）形式
```

这些实际上只是下面将要讨论的一般类型转换记法的特例。

4.1.2.7. 其他类型的常量

任意类型的常量都可以用下列记法之一输入：

```
type 'string'
'string'::type
CAST ( 'string' AS type )
```

字符串常量的文本会被传递给名为 *type* 的类型的输入转换例程。其结果是一个指定类型的常量。如果该常量必须具有的类型不存在歧义（例如，它被直接赋给某个表列时），
就可以省略显式类型转换，在这种情况下它会被自动强制转换。

字符串常量可以使用常规 SQL 记法或美元引用书写。

也可以用类似函数调用的语法来指定类型强制转换：

```
typename ( 'string' )
```

但并非所有类型名都可以这样使用，详见第 4.2.9 节。

如第 4.2.9 节中所述，::、CAST() 以及函数调用语法也可以用来指定任意表达式的运行时类型转换。为避免语法歧义，*type* '*string*' 语法只能用来指定简单字面常量的类型。*type* '*string*' 语法的另一个限制是不能用于数组类型；要指定数组常量的类型，请使用 :: 或 CAST()。

CAST()语法符合 SQL。*type* '*string*'语法是该标准的一般化：SQL 指定这种语法只用于一些数据类型，但是PostgreSQL允许它用于所有类型。带有::
的语法是PostgreSQL的历史用法，就像函数调用语法一样。

4.1.3. 操作符

操作符名称由下列列表中的字符组成，长度最多为 NAMEDATALEN-1 个字符（默认是 63）：

+ - * / < > = ~ ! @ # % ^ & | ` ?

不过，操作符名称也有一些限制：

- -- 和 /* 不能出现在操作符名称中的任何位置，因为它们会被视为注释的开始。
- 多字符操作符名称不能以 + 或 - 结尾，除非该名称中还至少包含下列字符中的一个：

  ~ ! @ # % ^ & | ` ?

  例如，@- 是合法的操作符名称，但 *- 不是。这个限制使 PostgreSQL 能够在不要求词元之间必须有空格的情况下解析符合 SQL 的查询。

当使用非 SQL 标准的操作符名时，你通常需要用空格分隔相邻的操作符来避免歧义。例如，如果你定义了一个名为`@`的左一元操作符，你不能写`X*@Y`，你必须写`X* @Y`来确保PostgreSQL把它读作两个操作符名而不是一个。

4.1.4. 特殊字符

某些非字母数字字符具有不同于操作符的特殊含义。关于它们的详细用法，
可以在描述相应语法元素的位置找到。本节只是为了提示它们的存在，
并概述这些字符的用途。

- 美元符号（`$`）后跟数字时，用来表示函数定义体或预备语句中的位置参数。
  在其他上下文中，美元符号可以是标识符的一部分，
  也可以是美元引用字符串常量的一部分。
- 圆括号（`()`）具有它们通常的含义，用来对表达式分组并确定运算优先级。
  在某些情况中，圆括号被要求作为一个特定 SQL 命令的固定语法的一部分。
- 方括号（`[]`）被用来选择一个数组中的元素。更多关于数组的信息见第 8.14 节。
- 逗号（`,`）被用在某些语法结构中来分割一个列表的元素。
- 分号（`;`）结束一个 SQL 命令。它不能出现在一个命令中间的任何位置，除了在一个字符串常量中或者一个带引号的标识符中。
- 冒号（`:`）被用来从数组中选择“切片”（见第 8.14 节）。在某些 SQL 的“方言”（例如嵌入式 SQL）中，冒号被用来作为变量名的前缀。
- 星号（`*`）在某些上下文中用来表示表行或复合值的所有字段。
  当它被用作聚合函数的参数时，还有一种特殊含义，即该聚合不需要任何显式参数。
- 句点（`.`）被用在数字常量中，并且被用来分割模式、表和列名。

4.1.5. 注释

注释是一串以双连字符开始并延伸到行尾的字符，例如：

```
-- 这是一条标准 SQL 注释
```

另外，也可以使用 C 风格注释块：

```
/* 多行注释
 * 包含嵌套：/* 嵌套块注释 */
 */
```

这里该注释开始于`/*`并且延伸到匹配出现的`*/`。这些注释块可按照 SQL 标准中指定的方式嵌套，但和 C 中不同。这样我们可以注释掉一大段可能包含注释块的代码。

在进一步进行语法分析之前，注释会从输入流中移除，并实际被替换为空白。

4.1.6. 操作符优先级

表 4.2展示了 PostgreSQL 中操作符的优先级和结合性。大多数操作符具有相同的优先级，并且是左结合的。操作符的优先级和结合性是固定写在解析器中的。
这可能导致不直观的行为；例如，布尔操作符 < 和 > 与布尔操作符 <= 和 >= 的优先级不同。
此外，在组合使用二元和一元操作符时，有时需要添加圆括号。例如：

```
SELECT 5 ! - 6;
```

将被解析为：

```
SELECT 5 ! (- 6);
```

因为解析器并不知道 – 等到知道时已经太迟了 – !被定义为后缀操作符，而不是中缀操作符。要在这种情况下得到所需的行为，必须写成：

```
SELECT (5 !) - 6;
```

这是为可扩展性付出的代价。

表 4.2. 操作符优先级（递减）

操作符/元素	结合性	描述
.	左	表/列名分隔符
::	左	PostgreSQL-风格的类型转换
[ ]	左	数组元素选择
+ -	右	一元正号、一元负号
^	左	求幂
* / %	左	乘、除、取模
+ -	左	加、减
IS		IS TRUE、IS FALSE、IS NULL等
ISNULL		空值测试
NOTNULL		非空测试
（任意其他）	左	所有其他操作符
IN		集合成员关系
BETWEEN		范围包含
OVERLAPS		时间区间重叠
LIKE ILIKE SIMILAR		字符串模式匹配
< >		小于、大于
=	右	相等、赋值
NOT	右	逻辑否定
AND	左	逻辑合取
OR	左	逻辑析取

注意，这些操作符优先级规则也适用于名称与上述内置操作符相同的用户定义操作符。例如，如果你为某种自定义数据类型定义了一个“+”操作符，那么无论你的操作符做什么，它都将具有与内置“+”操作符相同的优先级。

当一个模式限定的操作符名被用在OPERATOR语法中时，如下面的示例：

```
SELECT 3 OPERATOR(pg_catalog.+) 4;
```

OPERATOR 结构在优先级上会被视为表 4.2中为“任意其他”操作符所示的优先级。无论OPERATOR()中指定的是哪个具体操作符，这一点都成立。

4.2. 值表达式

值表达式被用于各种各样的环境中，例如在SELECT命令的目标列表中，作为INSERT或UPDATE中的新列值，或者作为若干命令中的搜索条件。
为了与表表达式（其结果是一张表）区分开来，值表达式的结果有时被称为标量。因此，值表达式也称为标量表达式（甚至简称为表达式）。表达式语法允许使用算术、逻辑、集合以及其他操作，从基本部分计算出值。

一个值表达式是下列之一：

- 一个常量或字面值
- 一个列引用
- 在一个函数定义体或预备语句中的一个位置参数引用
- 一个下标表达式
- 一个字段选择表达式
- 一个操作符调用
- 一个函数调用
- 一个聚合表达式
- 一个窗口函数调用
- 一个类型转换
- 一个排序规则表达式
- 一个标量子查询
- 一个数组构造器
- 一个行构造器
- 另一个圆括号中的值表达式（用于对子表达式分组并覆盖优先级）

在这个列表之外，还有一些结构可以被分类为一个表达式，
但是它们不遵循任何一般语法规则。这些通常具有一个函数或操作符的语义并且在第 9 章中的合适位置解释。一个示例是`IS NULL`子句。

我们已经在第 4.1.2 节中讨论过常量。下面的小节会讨论剩下的选项。

4.2.1. 列引用

一个列可以以下面的形式被引用：

correlation.*columnname*

correlation 是一个表的名字（可能带有模式名限定），或者是在 `FROM` 子句中为某个表定义的别名。如果该列名在当前查询所用的所有表中都是唯一的，
则关联名称和分隔用的句点都可以省略（另见第 7 章）。

4.2.2. 位置参数

位置参数引用用来表示一个由 SQL 语句外部提供的值。参数可用于 SQL 函数定义和预备语句中。某些客户端库还支持把数据值与 SQL 命令字符串分开指定，在这种情况下，参数就用来引用这些命令字符串之外的数据值。参数引用的形式是：

$*number*

例如，考虑一个函数`dept`的定义：

```
CREATE FUNCTION dept(text) RETURNS dept
    AS $$ SELECT * FROM dept WHERE name = $1 $$
    LANGUAGE SQL;
```

这里$1引用函数被调用时第一个函数参数的值。

4.2.3. 下标

如果一个表达式得到了一个数组类型的值，那么可以抽取出该数组值的一个特定元素：

expression[*subscript*]

或者抽取出多个相邻元素（一个“数组切片”）：

expression[*lower_subscript*:*upper_subscript*]

（这里，方括号[ ]表示其字面意思）。每一个下标自身是一个表达式，它必须得到一个整数值。

通常，数组表达式必须加上圆括号，但如果要加下标的表达式只是列引用或位置参数，则圆括号可以省略。另外，当原始数组是多维数组时，多个下标可以连写。例如：

```
mytable.arraycolumn[4]
mytable.two_d_column[17][34]
$1[10:42]
(arrayfunction(a,b))[42]
```

最后一个示例中的圆括号是必需的。详见第 8.14 节。

4.2.4. 字段选择

如果一个表达式得到一个复合类型（行类型）的值，那么可以抽取该行的指定字段：

expression.*fieldname*

通常，行表达式必须加上圆括号，但如果要取字段的表达式只是表引用或位置参数，则圆括号可以省略。例如：

```
mytable.mycolumn
$1.somecolumn
(rowfunction(a,b)).col3
```

（因此，限定列引用实际上只是字段选择语法的一种特例。）
一种重要的特例是从复合类型的表列中抽取字段：

```
(compositecol).somefield
(mytable.compositecol).somefield
```

这里需要圆括号来显示compositecol是一个列名而不是一个表名，在第二种情况中则是显示mytable是一个表名而不是一个模式名。

你可以通过书写.*来请求一个组合值的所有字段：

```
(compositecol).*
```

这种记法的行为根据上下文会有不同，详见第 8.15.5 节。

4.2.5. 操作符调用

对于一次操作符调用，有三种可能的语法：

expression operator expression（二元中缀操作符）
operator expression（一元前缀操作符）

expression operator（一元后缀操作符）

其中 *operator* 词元遵循第 4.1.3 节的语法规则，或者是关键词 AND，OR 和 NOT 之一，或者是一个如下形式的模式限定操作符名：

OPERATOR(*schema.operatorname*)

具体存在哪些操作符以及它们是一元还是二元，取决于系统或用户定义了哪些操作符。
第 9 章描述了内置操作符。

4.2.6. 函数调用

函数调用的语法是：函数名（可能带有模式名限定）后跟一组放在圆括号内的参数列表：

function_name ([*expression* [, *expression* ...]])

例如，下面会计算 2 的平方根：

sqrt(2)

内置函数的列表在第 9 章中。其他函数可以由用户增加。

参数也可以选择附带名称。详见第 4.3 节。

> **注意**
>
> 接受单个复合类型参数的函数，可以选择使用字段选择语法来调用；反过来，
> 字段选择也可以写成函数风格。也就是说，col(table) 与 table.col 这两种记法可以互换。这种行为不是 SQL 标准的一部分，但 PostgreSQL 提供了它，因为这允许使用函数来模拟“计算字段”。详见第 8.15.5 节。

4.2.7. 聚合表达式

一个聚合表达式表示将聚合函数应用于查询选中的各行。
聚合函数将多个输入归约为单个输出值，例如输入的和或平均值。
聚合表达式的语法可以是以下形式之一：

aggregate_name (*expression* [, ...] [*order_by_clause*])
aggregate_name (ALL *expression* [, ...] [*order_by_clause*])
aggregate_name (DISTINCT *expression* [, ...] [*order_by_clause*])
aggregate_name (*)

其中，*aggregate_name* 是先前定义的聚合（可以带模式名限定），*expression* 是任何本身不包含聚合表达式或窗口函数调用的值表达式，而 *order_by_clause* 是可选的 ORDER BY 子句，在下文说明。

第一种形式的聚合表达式为每一个输入行调用一次聚合。第二种形式和第一种相同，因为ALL是默认选项。
第三种形式为输入行中表达式的每一个可区分值（或者对于多个表达式是值的可区分集合）调用一次聚合。最后一种形式为每一个输入行调用一次聚合，
因为没有特定的输入值被指定，它通常只对于count(*)聚合函数有用。

大部分聚合函数忽略空输入，这样其中一个或多个表达式得到空值的行将被丢弃。
除非另有说明，对于所有内置聚合都是这样。

例如，count(*)得到输入行的总数。count(f1)得到输入行中f1为非空的数量，因为count忽略空值。而count(distinct f1)得到f1的非空可区分值的数量。

通常，输入行以未指定的顺序传递给聚合函数。在许多情况下这无关紧要；例如，min 无论以什么顺序接收输入都会产生相同的结果。但是，有些聚合函数（例如 array_agg 和 string_agg）的结果取决于输入行的顺序。使用这类聚合时，可选的 *order_by_clause* 可以用来指定所需的顺序。*order_by_clause*的语法与查询级 ORDER BY 子句相同，详见第 7.5 节，但其中的表达式只能是表达式，不能是输出列的名称或编号。例如：

```
SELECT array_agg(a ORDER BY b DESC) FROM table;
```

使用多参数聚合函数时，请注意 ORDER BY 子句位于所有聚合参数之后。例如，应写成：

```
SELECT string_agg(a, ',' ORDER BY a) FROM table;
```

而不是：

```
SELECT string_agg(a ORDER BY a, ',') FROM table;  -- incorrect
```

后者在语法上是有效的，但它表示调用一个带有两个 ORDER BY 排序键的单参数聚合函数（第二个排序键是常量，因此没什么用处）。

如果在 DISTINCT 之外还指定了 *order_by_clause*，那么所有 ORDER BY 表达式都必须与聚合的常规参数相匹配；也就是说，不能根据未包含在 DISTINCT 列表中的表达式排序。

> **注意**
>
> 在聚合函数中同时指定 DISTINCT 和 ORDER BY 的能力是 PostgreSQL 的扩展。

预定义的聚合函数在第 9.20 节中描述。其他聚合函数可以由用户增加。

一个聚合表达式只能出现在SELECT命令的结果列表或是HAVING子句中。
在其他子句（如WHERE）中禁止使用它，
因为那些子句的计算在逻辑上是在聚合的结果被形成之前。

当一个聚合表达式出现在一个子查询中（见第 4.2.11 节和第 9.22 节），聚合通常在该子查询的行上被计算。
但是如果该聚合的参数只包含外层变量则会产生一个异常：该聚合则属于最近的那个外层，
并且会在那个查询的行上被计算。
该聚合表达式从整体上则是对其所出现于的子查询的一种外层引用，
并且在那个子查询的任意一次计算中都作为一个常量。只出现在结果列表或HAVING
子句的限制适用于该聚合所属的查询层次。

4.2.8. 窗口函数调用

一次窗口函数调用表示在查询选出的部分行上应用一个类似聚合的函数。
与普通聚合函数调用不同，这并不意味着把选出的行分组为单个输出行 – 每一行在查询输出中仍然独立存在。不过，窗口函数能够扫描根据该窗口函数调用的分组说明（PARTITION BY 列表）属于当前行所在组的所有行。窗口函数调用的语法是下列之一：

```
function_name ([expression [, expression ... ]]) OVER window_name
function_name ([expression [, expression ... ]]) OVER ( window_
definition )
function_name ( * ) OVER window_name
function_name ( * ) OVER ( window_definition )
```

其中 *window_definition* 的语法为：

```
[ existing_window_name ]
[ PARTITION BY expression [, ...] ]
```

```
[ ORDER BY expression [ ASC | DESC | USING operator ] [ NULLS {
 FIRST | LAST } ] [, ...] ]
[ frame_clause ]
```

而可选的 *frame_clause* 可以是以下形式之一：

```
{ RANGE | ROWS } frame_start
{ RANGE | ROWS } BETWEEN frame_start AND frame_end
```

其中 *frame_start* 和 *frame_end* 可以是以下形式之一：

```
UNBOUNDED PRECEDING
value PRECEDING
CURRENT ROW
value FOLLOWING
UNBOUNDED FOLLOWING
```

这里，*expression*表示任何自身不含有窗口函数调用的值表达式。PARTITION BY和ORDER BY列表与整个查询的GROUP BY和ORDER BY子句具有本质上相同的语法和语义，唯一例外是它们的表达式始终只能是表达式，不能是输出列的名称或编号。*window_name*是对定义在查询的WINDOW子句中的一个命名窗口声明的引用。还可以使用在WINDOW子句中定义命名窗口的相同语法在圆括号内给定一个完整的*window_definition*，详见SELECT参考页。值得指出的是，OVER wname并不严格地等价于OVER (wname)，后者表示复制并修改窗口定义，并且在被引用窗口声明包括一个帧子句时会被拒绝。

*frame_clause*指定构成窗口帧的行集合，供那些作用于帧而不是整个分区的窗口函数使用。如果*frame_end*被省略，则默认为CURRENT ROW。限制是*frame_start*不能为UNBOUNDED FOLLOWING，*frame_end*不能为UNBOUNDED PRECEDING，并且在上述列表中，*frame_end*选项不能出现在*frame_start*选项之前 – 例如，不允许RANGE BETWEEN CURRENT ROW AND *value* PRECEDING。默认的帧选项是RANGE UNBOUNDED PRECEDING，它和RANGE BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW相同；这会把该帧设置为从分区开始一直到当前行的最后一个ORDER BY同等行的所有行（如果没有ORDER BY，则是所有行）。一般来说，UNBOUNDED PRECEDING表示该帧开始于分区的第一行，类似地UNBOUNDED FOLLOWING表示该帧结束于分区的最后一行（不管是RANGE还是ROWS模式）。在ROWS模式中，CURRENT ROW表示该帧开始或结束于当前行；而在RANGE模式中，它表示该帧开始或结束于当前行在ORDER BY次序中的第一个或最后一个同等行。*value* PRECEDING和*value* FOLLOWING目前只允许用于ROWS模式。它们表示帧开始或结束于当前行之前或之后那么多行的位置。*value*必须是一个不包含任何变量、聚合函数或窗口函数的整数表达式。其值不能为空值或负数；但可以为零，这会只选择当前行本身。

内置的窗口函数在表 9.48中介绍。用户可以加入其他窗口函数。此外，任何内置的或者用户定义的聚合函数都可以被用作窗口函数。

使用*的语法被用来把参数较少的聚合函数当作窗口函数调用，例如count(*) OVER (PARTITION BY x ORDER BY y)。星号（*）通常不被用于非聚合的窗口函数。与普通聚合函数不同，聚合窗口函数不允许在函数参数列表中使用DISTINCT或ORDER BY。

只有在SELECT列表和查询的ORDER BY子句中才允许窗口函数调用。

更多关于窗口函数的信息可以在第 3.5 节、第 9.21 节和第 7.2.4 节中找到。

4.2.9. 类型转换

一个类型转换指定从一种数据类型到另一种数据类型的转换。PostgreSQL接受两种等价的类型转换语法：

```
CAST ( expression AS type )
```

expression::*type*

CAST语法遵从 SQL，而用::的语法是PostgreSQL的历史用法。

当类型转换被应用于一种已知类型的值表达式时，它表示一次运行时类型转换。
只有在已经定义了合适的类型转换操作时，该类型转换才会成功。注意，
这与常量上的类型转换（如第 4.1.2.7 节中所示）略有不同。应用于未修饰字符串字面量的类型转换，表示为一个字面常量值赋予初始类型，
因此它对任意类型都能成功（前提是该字符串字面量的内容符合该数据类型的输入语法）。

如果一个值表达式必须产生的类型没有歧义（例如当它被赋给一个表列时），
通常可以省略显式类型转换，在这种情况下系统会自动应用类型转换。但是，
自动类型转换只会对那些在系统目录中被标记为“可以隐式应用”的类型转换执行。
其他类型转换必须使用显式类型转换语法来调用。
施加这一限制的目的是防止出人意料的转换被悄无声息地应用。

还可以用类似函数调用的语法来指定类型转换：

typename (*expression*)

不过，这只对那些名称本身也可作为函数名使用的类型有效。例如，double precision 不能用这种方式，但等效的 float8 可以。另外，由于语法冲突，名称 interval、time 和 timestamp 只有在使用双引号引用时才能采用这种写法。因此，函数风格的类型转换语法会带来不一致性，通常应尽量避免。

> **注意**
>
> 函数风格的语法事实上只是一次函数调用。
> 当两种标准类型转换语法之一被用来做一次运行时转换时，
> 它将在内部调用一个已注册的函数来执行该转换。简而言之，
> 这些转换函数具有和它们的输出类型相同的名字，并且因此“函数风格的语法”
> 无非是对底层转换函数的一次直接调用。显然，一个可移植的应用不应当依赖于它。
> 详见CREATE CAST。

4.2.10. 排序规则表达式

COLLATE 子句会覆盖表达式的排序规则。它附加在所作用的表达式之后：

expr COLLATE *collation*

这里 *collation* 可以是一个带模式名限定的标识符。COLLATE 子句比操作符绑定得更紧，必要时可以使用圆括号。

如果没有显式指定排序规则，数据库系统会从表达式所涉及的列中得到一个排序规则，
如果该表达式没有涉及列，则会默认采用数据库的默认排序规则。

COLLATE 子句的两种常见用途是覆盖 ORDER BY 子句中的排序顺序，例如：

```
SELECT a, b, c FROM tbl WHERE ... ORDER BY a COLLATE "C";
```

以及覆盖具有区域敏感结果的函数或操作符调用的排序规则，例如：

```
SELECT * FROM tbl WHERE a > 'foo' COLLATE "C";
```

注意，在后一种情况中，COLLATE 子句是附加到我们希望影响的那个操作符的某个输入参数上的。把 COLLATE 子句附加到该操作符或函数调用的哪个参数上并不重要，因为操作符或函数实际采用的排序规则是综合所有参数得出的，而显式的 COLLATE 子句会覆盖其他所有

参数的排序规则。（不过，若把不匹配的 COLLATE 子句附加到多个参数上，则会报错。详见第 22.2 节。）因此，这会得到与前一个示例相同的结果：

```
SELECT * FROM tbl WHERE a COLLATE "C" > 'foo';
```

但是这是一个错误：

```
SELECT * FROM tbl WHERE (a > 'foo') COLLATE "C";
```

因为它尝试把一个排序规则应用到>操作符的结果，
而它的数据类型是不支持排序规则的数据类型boolean。

4.2.11. 标量子查询

标量子查询是一个放在圆括号中的普通 SELECT 查询，它恰好返回一行一列（关于如何编写查询，见第 7 章）。该 SELECT 查询会被执行，其返回的单个值会用在外围值表达式中。把一个返回多于一行或多于一列的查询当作标量子查询来用是错误的。
（但如果在某次具体执行中该子查询没有返回任何行，则不算错误；
其标量结果会被视为空值。）该子查询可以引用外围查询中的变量，
这些变量在该子查询的每一次求值期间都会作为常量。有关其他涉及子查询的表达式，
另见第 9.22 节。

例如，下列语句会寻找每个州中最大的城市人口：

```
SELECT name, (SELECT max(pop) FROM cities WHERE cities.state =
 states.name)
    FROM states;
```

4.2.12. 数组构造器

数组构造器是一个表达式，它使用其成员元素的值来构造一个数组值。
简单的数组构造器由关键字 ARRAY、左方括号 [、一个用于指定数组元素值的表达式列表（以逗号分隔）以及最后的右方括号] 组成。例如：

```
SELECT ARRAY[1,2,3+4];
  array
---------
 {1,2,7}
(1 row)
```

默认情况下，数组元素类型是成员表达式的公共类型，其确定规则与 UNION 或 CASE 结构相同（见第 10.5 节）。你可以通过把数组构造器显式转换为所需类型来覆盖这一点，例如：

```
SELECT ARRAY[1,2,22.7]::integer[];
  array
----------
 {1,2,23}
(1 row)
```

这与把每个表达式分别转换为数组元素类型具有相同的效果。关于类型转换的更多信息，
见第 4.2.9 节。

多维数组值可以通过嵌套数组构造器来构造。在内部构造器中，关键字 ARRAY 可以省略。例如，下面两种写法产生相同的结果：

```
SELECT ARRAY[ARRAY[1,2], ARRAY[3,4]];
     array
---------------
 {{1,2},{3,4}}
```

```
(1 row)

SELECT ARRAY[[1,2],[3,4]];
     array
---------------
 {{1,2},{3,4}}
(1 row)
```

由于多维数组必须是长方形的，
因此同一级别的内部构造器必须生成维度完全相同的子数组。施加到外层 ARRAY 构造器上的任何类型转换，都会自动传播到所有内层构造器。

多维数组构造器中的元素可以是任何能够生成适当类型数组的表达式，而不仅仅是子 ARRAY 构造。例如：

```
CREATE TABLE arr(f1 int[], f2 int[]);

INSERT INTO arr VALUES (ARRAY[[1,2],[3,4]], ARRAY[[5,6],[7,8]]);

SELECT ARRAY[f1, f2, '{{9,10},{11,12}}'::int[]] FROM arr;
                     array
------------------------------------------------
 {{{1,2},{3,4}},{{5,6},{7,8}},{{9,10},{11,12}}}
(1 row)
```

你也可以构造空数组，但由于不可能存在没有类型的数组，
因此必须把空数组显式转换为所需类型。例如：

```
SELECT ARRAY[]::integer[];
 array
-------
 {}
(1 row)
```

也可以根据子查询的结果构造数组。在这种形式中，数组构造器写为关键字 ARRAY 后跟一个用圆括号（而不是方括号）括起的子查询。例如：

```
SELECT ARRAY(SELECT oid FROM pg_proc WHERE proname LIKE 'bytea%');
                                 array
-----------------------------------------------------------------
----
 {2011,1954,1948,1952,1951,1244,1950,2005,1949,1953,2006,31,2412,
2413}
(1 row)
```

子查询必须返回单列。生成的数组为子查询结果中的每一行包含一个元素，
元素类型与子查询输出列的类型相匹配。

用 ARRAY 构造出来的数组值，其下标总是从 1 开始。关于数组的更多信息，见第 8.14 节。

4.2.13. 行构造器

行构造器是一种表达式，它使用成员字段的值来构造一个行值（也称为组合值）。
行构造器由关键字 ROW、左圆括号、零个或多个用于指定行各字段值的表达式（以逗号分隔），以及最后的右圆括号组成。例如：

```
SELECT ROW(1,2.5,'this is a test');
```

当列表中有多个表达式时，关键字 ROW 是可选的。

一个行构造器可以包括语法*rowvalue*.*，它将被扩展为该行值的元素的一个列表，就像在一个顶层SELECT列表（见第 8.15.5 节）中使用.*时发生的事情一样。例如，如果表t有列f1和f2，那么这些是相同的：

```
SELECT ROW(t.*, 42) FROM t;
SELECT ROW(t.f1, t.f2, 42) FROM t;
```

> **注意**
>
> 在PostgreSQL 8.2 以前，.* 语法不会在行构造器中展开，因此写成 ROW(t.*, 42) 会创建一个具有两个字段的行，其中第一个字段是另一个行值。新的行为通常更有用。如果你需要旧版本那种嵌套行值的行为，写内层行值时不要使用 .*，例如 ROW(t, 42)。

默认情况下，ROW 表达式创建的值具有匿名记录类型。如有需要，可以把它转换为具名复合类型，也就是某个表的行类型，或者使用 CREATE TYPE AS 创建的复合类型。为避免歧义，可能需要显式类型转换。例如：

```
CREATE TABLE mytable(f1 int, f2 float, f3 text);

CREATE FUNCTION getf1(mytable) RETURNS int AS 'SELECT $1.f1'
 LANGUAGE SQL;

-- No cast needed since only one getf1() exists
SELECT getf1(ROW(1,2.5,'this is a test'));
 getf1
-------
     1
(1 row)

CREATE TYPE myrowtype AS (f1 int, f2 text, f3 numeric);

CREATE FUNCTION getf1(myrowtype) RETURNS int AS 'SELECT $1.f1'
 LANGUAGE SQL;

-- Now we need a cast to indicate which function to call:
SELECT getf1(ROW(1,2.5,'this is a test'));
ERROR:  function getf1(record) is not unique

SELECT getf1(ROW(1,2.5,'this is a test')::mytable);
 getf1
-------
     1
(1 row)

SELECT getf1(CAST(ROW(11,'this is a test',2.5) AS myrowtype));
 getf1
-------
    11
(1 row)
```

行构造器可以用来构造复合值，以存储在复合类型的表列中，或者传递给接受复合参数的函数。此外，还可以比较两个行值，或者用 IS NULL 或 IS NOT NULL 来测试一个行，例如：

```
SELECT ROW(1,2.5,'this is a test') = ROW(1, 3, 'not the same');
```

```
SELECT ROW(table.*) IS NULL FROM table;  -- detect all-null rows
```

更多详情见第 9.23 节。行构造器也可以与子查询结合使用，详见第 9.22 节。

4.2.14. 表达式计算规则

子表达式的计算顺序是未定义的。特别是，
操作符或函数的输入不一定会按照从左到右或任何其他固定顺序计算。

此外，如果一个表达式的结果可以通过只计算其一部分来决定，
那么其他子表达式可能完全不需要被计算。例如，如果我们写：

```
SELECT true OR somefunc();
```

那么somefunc()将（可能）完全不被调用。如果我们写成下面这样也是一样：

```
SELECT somefunc() OR true;
```

注意这和一些编程语言中布尔操作符从左至右的“短路”不同。

因此，在复杂表达式中使用带有副作用的函数是不明智的。在WHERE和HAVING
子句中依赖副作用或计算顺序尤其危险，
因为在建立一个执行计划时这些子句会被广泛地重新处理。这些子句中布尔表达式（AND/OR
/NOT的组合）可能会以布尔代数定律所允许的任何方式被重组。

当有必要强制计算顺序时，可以使用一个CASE结构（见第 9.17 节）。例如，在一个WHERE子句中使用下面的方法尝试避免除零是不可靠的：

```
SELECT ... WHERE x > 0 AND y/x > 1.5;
```

但是这是安全的：

```
SELECT ... WHERE CASE WHEN x > 0 THEN y/x > 1.5 ELSE false END;
```

一个以这种风格使用的CASE结构将使得优化尝试失败，
因此只有必要时才这样做（在这个特别的示例中，最好通过写y > 1.5*x来回避这个问题）。

不过，CASE 并不是这类问题的万灵药。上述技术的一个限制是，它无法阻止常量子表达式被提前计算。如第 35.6 节中所述，当查询被规划而不是被执行时，被标记成 IMMUTABLE的函数和操作符可以被计算。因此

```
SELECT CASE WHEN x > 0 THEN x ELSE 1/0 END FROM tab;
```

很可能会导致一次除零失败，因为规划器尝试简化常量子表达式。即便是表中的每一行都有x > 0（这样运行时永远不会进入到 ELSE分支）也是这样。

虽然这个例子看起来有些可笑，但在函数内部执行的查询中，
也可能出现一些看上去并未明显涉及常量的相关情况，
因为函数参数值和局部变量值为了规划目的，可能会作为常量插入到查询中。例如，在 PL/pgSQL 函数中，使用 IF-THEN-ELSE 语句来保护一项有风险的计算，要比仅仅把它嵌套进一个 CASE 表达式安全得多。

另一个同类型的限制是，一个CASE无法阻止其所包含的聚合表达式的计算，因为在考虑
SELECT列表或HAVING子句中的其他表达式之前，会先计算聚合表达式。例如，
下面的查询可能会导致除零错误，尽管看起来似乎已经对这种情况做了防护：

```
SELECT CASE WHEN min(employees) > 0
            THEN avg(expenses / employees)
       END
```

```
    FROM departments;
```

min()和avg()聚合会在所有输入行上并行地计算，因此如果任何行有employees等于零，在有机会测试 min()的结果之前，就会发生除零错误。取而代之的是，可以使用一个WHERE子句来首先阻止有问题的输入行到达一个聚合函数。

4.3. 调用函数

PostgreSQL 允许对带命名参数的函数使用位置记法或命名记法来调用。命名记法对于拥有大量参数的函数尤其有用，因为它能让形参与实参之间的对应关系更明确、更可靠。
在位置记法中，函数调用中的参数值按照它们在函数声明中定义的顺序书写；
在命名记法中，实参与函数参数按名称匹配，因此可以按任意顺序书写。

在任意一种记法中，在函数声明中给出了默认值的参数都可以在调用中省略。
不过这在命名记法中特别有用，因为任意组合的参数都可以被省略；而在位置记法中，
参数只能从右向左省略。

PostgreSQL 也支持混合记法，它结合了位置记法和命名记法。在这种情况下，位置参数先写，命名参数写在后面。

下面的示例将展示这三种记法的用法，所用函数定义如下：

```
CREATE FUNCTION concat_lower_or_upper(a text, b text, uppercase
 boolean DEFAULT false)
RETURNS text
AS
$$
 SELECT CASE
        WHEN $3 THEN UPPER($1 || ' ' || $2)
        ELSE LOWER($1 || ' ' || $2)
        END;
$$
LANGUAGE SQL IMMUTABLE STRICT;
```

函数 concat_lower_or_upper 有两个必需参数 a 和 b。此外还有一个可选参数 uppercase，其默认值为 false。a 和 b 的输入将被连接起来，并根据 uppercase 参数的值转换成大写或小写。该函数定义的其余细节在这里并不重要（详见第 35 章）。

4.3.1. 使用位置记法

位置记法是在 PostgreSQL 中向函数传递参数的传统机制。例如：

```
SELECT concat_lower_or_upper('Hello', 'World', true);
 concat_lower_or_upper
-----------------------
 HELLO WORLD
(1 row)
```

所有参数都按顺序给出。由于 uppercase 被指定为 true，因此结果是大写。另一个例子是：

```
SELECT concat_lower_or_upper('Hello', 'World');
 concat_lower_or_upper
-----------------------
 hello world
(1 row)
```

这里省略了 uppercase 参数，因此它接收默认值 false，结果是小写输出。在位置记法中，只要参数具有默认值，就可以从右向左依次省略。

4.3.2. 使用命名记法

在命名记法中，每个参数名都使用 `:=` 与其参数表达式分隔。例如：

```
SELECT concat_lower_or_upper(a := 'Hello', b := 'World');
 concat_lower_or_upper
-----------------------
 hello world
(1 row)
```

同样，`uppercase` 参数被省略，因此它被隐式设置为 `false`。使用命名记法的一个优点是，参数可以按任意顺序指定，例如：

```
SELECT concat_lower_or_upper(a := 'Hello', b := 'World',
 uppercase := true);
 concat_lower_or_upper
-----------------------
 HELLO WORLD
(1 row)

SELECT concat_lower_or_upper(a := 'Hello', uppercase := true, b :=
 'World');
 concat_lower_or_upper
-----------------------
 HELLO WORLD
(1 row)
```

4.3.3. 使用混合记法

混合记法结合了位置记法和命名记法。不过，如前所述，
命名参数不能出现在位置参数之前。例如：

```
SELECT concat_lower_or_upper('Hello', 'World', uppercase := true);
 concat_lower_or_upper
-----------------------
 HELLO WORLD
(1 row)
```

在上面的查询中，参数 `a` 和 `b` 采用位置方式指定，而 `uppercase` 采用命名方式指定。在这个示例中，这样做除了起到一定说明作用外并没有太多额外价值。但对于拥有大量带默认值参数的更复杂函数，命名记法或混合记法可以大幅减少书写量，并降低出错机会。

> **注意**
>
> 命名调用记法和混合调用记法目前不能用于调用聚合函数（但当聚合函数被用作窗口函数时，它们是可以使用的）。

第 5 章 数据定义

本章介绍如何创建用于保存数据的数据库结构。在关系数据库中，原始数据存储在表中，因此本章的大部分内容都在说明如何创建和修改表，以及有哪些特性可用于控制表中存储的数据。随后，我们将讨论如何把表组织到模式中，以及如何向表分配权限。最后，我们会简要介绍一些同样会影响数据存储的特性，例如继承、视图、函数和触发器。

5.1. 表基础

关系数据库中的表很像纸面上的表格：它由行和列组成。列的数量和顺序是固定的，每一列都有名称。行的数量则是可变的，它反映了某一时刻存储了多少数据。SQL 不对表中行的顺序作任何保证。读取表时，除非显式要求排序，否则行会以未指定的顺序出现，参见第 7 章。此外，SQL 不会为行分配唯一标识符，因此一个表中可能出现多行完全相同的记录。这是 SQL 所依据的数学模型带来的结果，但通常并不理想。本章稍后将看到如何处理这个问题。

每一列都有一种数据类型。数据类型限制了可以赋给该列的可能值集合，并为列中存储的数据赋予语义，从而使其能够用于计算。例如，声明为数值类型的列不能接受任意文本字符串，存储在这类列中的数据可以参与数学运算。相比之下，声明为字符串类型的列几乎能接受任何种类的数据，但并不适合用于数学计算，尽管仍可进行字符串拼接等其他操作。

PostgreSQL提供了相当丰富的一组内置数据类型，适用于许多应用场景。用户也可以定义自己的数据类型。大多数内置数据类型的名称和语义都很直观，因此我们把详细说明留到第 8 章。一些常用的数据类型包括：表示整数的`integer`、表示可能带小数部分数值的`numeric`、表示字符串的`text`、表示日期的`date`、表示一天中时间值的`time`，以及同时包含日期和时间的`timestamp`。

要创建一个表，可使用名副其实的CREATE TABLE命令。在该命令中，至少要指定新表的名称、各列的名称以及每列的数据类型。例如：

```
CREATE TABLE my_first_table (
    first_column text,
    second_column integer
);
```

这会创建一个名为`my_first_table`的表，它有两列。第一列名为`first_column`，数据类型为`text`；第二列名为`second_column`，类型为`integer`。表名和列名遵循第 4.1.1 节中解释的标识符语法。类型名通常也是标识符，但也有少数例外。注意，列列表以逗号分隔，并用圆括号括起来。

当然，前面的例子相当刻意。通常，你会为表和列起能体现其所存储数据含义的名称。下面看一个更实际的例子：

```
CREATE TABLE products (
    product_no integer,
    name text,
    price numeric
);
```

（`numeric`类型可以存储小数部分，这对金额很常见。）

> **提示**
>
> 在创建许多相互关联的表时，最好为表和列选择一致的命名模式。例如，
> 表名可以使用单数名词或复数名词，两种做法都各有支持者。

一个表能包含的列数是有限的。根据列类型不同，这个上限在 250 到 1600 之间。不过，定义一个接近这个数量的表极不常见，而且其设计往往值得商榷。

如果不再需要某个表，可以使用DROP TABLE命令删除它。例如：

```
DROP TABLE my_first_table;
DROP TABLE products;
```

尝试删除不存在的表会报错。不过，在 SQL 脚本文件中，人们常常在创建每个表之前无条件地尝试删除它，并忽略可能出现的错误消息，这样脚本无论表是否存在都能工作。
（如果愿意，也可以使用`DROP TABLE IF EXISTS`变体来避免错误消息，但这不是标准SQL。）

如果我们需要修改一个已经存在的表，请参考本章稍后的第 5.5 节。

利用到目前为止所讨论的工具，我们已经可以创建功能完备的表。
本章其余内容关注如何向表定义添加特性，以确保数据完整性、安全性或使用便利。
如果你现在就急着向表中填充数据，可以先跳到第 6 章，稍后再回来阅读本章其余内容。

5.2. 默认值

列可以被赋予默认值。当创建新行时，如果没有为某些列指定值，
这些列就会填入各自的默认值。数据操作命令也可以显式要求把某列设为其默认值，
而不必知道默认值究竟是什么。（数据操作命令详见第 6 章。）

如果没有显式指定默认值，则默认值是空值。这是合理的，因为空值表示未知数据。

在表定义中，默认值写在列的数据类型之后。例如：

```
CREATE TABLE products (
    product_no integer,
    name text,
    price numeric DEFAULT 9.99
);
```

默认值也可以是表达式。每次需要插入默认值时都会对其求值（不是在创建表时）。
一个常见例子是让`timestamp`列的默认值为`CURRENT_TIMESTAMP`，
这样它会被设为插入该行时的时间。另一个常见例子是为每一行生成一个“序列号”。
在PostgreSQL中，这通常可以这样实现：

```
CREATE TABLE products (
    product_no integer DEFAULT nextval('products_product_no_seq'),
    ...
);
```

其中`nextval()`函数从序列对象（见第 9.16 节）中依次取值。这种写法非常常见，因此还有一种专门的简写形式：

```
CREATE TABLE products (
    product_no SERIAL,
    ...
```

```
);
```

SERIAL简写会在第 8.1.4 节进一步讨论。

5.3. 约束

数据类型是一种限制能够存储在表中的数据类别的方法。但是对于很多应用来说，它们提供的约束太粗糙。例如，一个包含产品价格的列应该只接受正值。但是没有任何一种标准数据类型只接受正值。另一个问题是我们可能需要根据其他列或行来约束一个列中的数据。例如，在一个包含产品信息的表中，对于每个产品编号应该只有一行。

为此，SQL 允许我们在列和表上定义约束。约束让我们能够按照需要控制表中的数据。如果用户试图在列中存储违反约束的数据，就会报错。即使该值来自默认值定义，这条规则也同样适用。

5.3.1. 检查约束

检查约束是最通用的约束类型。它允许我们指定某一列中的值必须满足一个布尔（真值）表达式。例如，要要求产品价格为正值，可以使用：

```
CREATE TABLE products (
    product_no integer,
    name text,
    price numeric CHECK (price > 0)
);
```

如你所见，约束定义和默认值定义一样都写在数据类型之后。默认值和约束的先后顺序没有影响。检查约束由关键字CHECK以及其后放在圆括号中的表达式组成。检查约束表达式应当涉及被约束的列，否则这个约束就没有太大意义。

我们也可以为约束单独指定一个名称。这样可以让错误消息更清晰，也便于在需要修改约束时引用它。语法如下：

```
CREATE TABLE products (
    product_no integer,
    name text,
    price numeric CONSTRAINT positive_price CHECK (price > 0)
);
```

要指定一个命名约束，在约束名标识符前写关键字CONSTRAINT，再在其后写约束定义即可。（如果没有用这种方式指定约束名，系统会为你选择一个。）

一个检查约束也可以引用多个列。例如我们存储一个普通价格和一个打折后的价格，而我们希望保证打折后的价格低于普通价格：

```
CREATE TABLE products (
    product_no integer,
    name text,
    price numeric CHECK (price > 0),
    discounted_price numeric CHECK (discounted_price > 0),
    CHECK (price > discounted_price)
);
```

前两个约束看起来很相似。第三个则使用了一种新语法。它并没有依附在一个特定的列，而是作为一个独立的项出现在逗号分隔的列列表中。列定义和这种约束定义可以以混合的顺序出现在列表中。

我们将前两个约束称为列约束，而第三个约束为表约束，因为它独立于任何一个列定义。
列约束也可以写成表约束，但反过来不行，因为一个列约束只能引用它所依附的那一个列（PostgreSQL并不强制要求这个规则，
但是如果我们希望表定义能够在其他数据库系统中工作，那就应该遵循它）。
上述示例也可以写成：

```
CREATE TABLE products (
    product_no integer,
    name text,
    price numeric,
    CHECK (price > 0),
    discounted_price numeric,
    CHECK (discounted_price > 0),
    CHECK (price > discounted_price)
);
```

甚至是：

```
CREATE TABLE products (
    product_no integer,
    name text,
    price numeric CHECK (price > 0),
    discounted_price numeric,
    CHECK (discounted_price > 0 AND price > discounted_price)
);
```

这只是口味的问题。

表约束也可以用与列约束相同的方法来指定名称：

```
CREATE TABLE products (
    product_no integer,
    name text,
    price numeric,
    CHECK (price > 0),
    discounted_price numeric,
    CHECK (discounted_price > 0),
    CONSTRAINT valid_discount CHECK (price > discounted_price)
);
```

需要注意的是，当检查表达式的值为真或空值时，检查约束就被视为满足。
由于当任一操作数为空时，大多数表达式都会计算为空值，
所以检查约束不会阻止受约束列中出现空值。要确保某列不包含空值，
可以使用下一节介绍的非空约束。

5.3.2. 非空约束

非空约束只是指定某列不能取空值。语法示例如下：

```
CREATE TABLE products (
    product_no integer NOT NULL,
    name text NOT NULL,
    price numeric
);
```

非空约束总是写成列约束。从功能上看，非空约束等价于创建检查约束`CHECK (column_name IS NOT NULL)`，但在PostgreSQL中创建显式的非空约束效率更高。缺点是无法为以这种方式创建的非空约束显式指定名称。

当然，一列可以有多个约束。只需将约束逐个写出：

```
CREATE TABLE products (
    product_no integer NOT NULL,
    name text NOT NULL,
    price numeric NOT NULL CHECK (price > 0)
);
```

顺序并不重要，也不一定决定检查约束的顺序。

该NOT NULL约束有一个反面形式：NULL约束。这并不表示该列必须为空值，那显然毫无用处。它只是显式选择列可以为空的默认行为。NULL约束不属于 SQL 标准，因此不应在需要可移植性的应用中使用。（它之所以被加入PostgreSQL，
只是为了兼容某些其他数据库系统。）不过，有些用户喜欢它，
因为它让在脚本文件中切换该约束变得比较容易。例如，可以从下面的定义开始：

```
CREATE TABLE products (
    product_no integer NULL,
    name text NULL,
    price numeric NULL
);
```

然后在需要的地方插入NOT关键字。

> **提示**
>
> 在大多数数据库设计中，多数列都应标记为非空。

5.3.3. 唯一约束

唯一约束保证某一列或某一组列中保存的数据在整个表的所有行之间都是唯一的。
写成列约束时的语法是：

```
CREATE TABLE products (
    product_no integer UNIQUE,
    name text,
    price numeric
);
```

写成表约束时则是：

```
CREATE TABLE products (
    product_no integer,
    name text,
    price numeric,
    UNIQUE (product_no)
);
```

要为一组列定义一个唯一约束，把它写作一个表级约束，列名用逗号分隔：

```
CREATE TABLE example (
    a integer,
    b integer,
    c integer,
    UNIQUE (a, c)
);
```

这指定这些列的组合值在整个表的范围内是唯一的，
但其中任意一列的值并不需要是（一般也不是）唯一的。

我们可以按通常的方式为唯一约束指定名称：

```
CREATE TABLE products (
    product_no integer CONSTRAINT must_be_different UNIQUE,
    name text,
    price numeric
);
```

增加一个唯一约束会在约束中列出的列或列组上自动创建一个唯一 B-树索引。只覆盖某些行的唯一性限制不能写成唯一约束，但可以通过创建唯一的部分索引来强制这种限制。

通常情况下，如果表中有多行，且约束所包含的所有列的值都相等，就违反了唯一约束。
不过，在这种比较中，两个空值从不被视为相等。这意味着即使存在唯一约束，
也可以存储重复行，只要其中至少一个受约束列含有空值。此行为符合 SQL 标准，但我们听说其他 SQL 数据库可能不遵循此规则。因此，开发需要可移植性的应用时要小心。

5.3.4. 主键

一个主键约束表示可以用作表中行的唯一标识符的一个列或者一组列。
这要求那些值都是唯一的并且非空。因此，下面的两个表定义接受相同的数据：

```
CREATE TABLE products (
    product_no integer UNIQUE NOT NULL,
    name text,
    price numeric
);

CREATE TABLE products (
    product_no integer PRIMARY KEY,
    name text,
    price numeric
);
```

主键也可以包含多于一个列，其语法和唯一约束相似：

```
CREATE TABLE example (
    a integer,
    b integer,
    c integer,
    PRIMARY KEY (a, c)
);
```

增加一个主键会自动在主键中列出的列或列组上创建一个唯一 B-树索引，并强制这些列被标记为`NOT NULL`。

一个表最多只能有一个主键（可以有任意数量的唯一和非空约束，
它们可以达到和主键几乎一样的功能，但只能有一个被标识为主键）。
关系数据库理论要求每一个表都要有一个主键。PostgreSQL 并未强制要求这一点，但通常最好遵循它。

主键对于文档和客户端应用都是有用的。例如，一个允许修改行值的 GUI 应用可能需要知道一个表的主键，以便能唯一地标识行。如果定义了主键，
数据库系统也有多种方法来利用主键。例如，主键定义了外键要引用的默认目标列。

5.3.5. 外键

一个外键约束指定一列（或一组列）中的值必须匹配出现在另一个表中某些行的值。
我们说这维持了两个关联表之间的引用完整性。

例如我们有一个使用过多次的产品表：

```
CREATE TABLE products (
    product_no integer PRIMARY KEY,
    name text,
    price numeric
);
```

让我们假设我们还有一个存储这些产品订单的表。
我们希望保证订单表中只包含真正存在的产品的订单。
因此我们在订单表中定义一个引用产品表的外键约束：

```
CREATE TABLE orders (
    order_id integer PRIMARY KEY,
    product_no integer REFERENCES products (product_no),
    quantity integer
);
```

现在就不可能创建包含不存在于产品表中的`product_no`值（非空）的订单。

我们说在这种情况下，订单表是引用表而产品表是被引用表。相应地，
也有引用和被引用列的说法。

我们也可以把上述命令简写为：

```
CREATE TABLE orders (
    order_id integer PRIMARY KEY,
    product_no integer REFERENCES products,
    quantity integer
);
```

因为如果缺少列的列表，则被引用表的主键将被用作被引用列。

一个外键也可以约束和引用一组列。照例，它需要被写成表约束的形式。下面是一个示例：

```
CREATE TABLE t1 (
  a integer PRIMARY KEY,
  b integer,
  c integer,
  FOREIGN KEY (b, c) REFERENCES other_table (c1, c2)
);
```

当然，被约束列的数量和类型应该匹配被引用列的数量和类型。

你可以按常规方式为外键约束指定自己的名称。

一个表可以含有多个外键约束。这可用于实现表之间的多对多关系。
假设有产品表和订单表，
现在希望一个订单可以包含多个产品（上面的结构不允许这样做）。可以使用如下表结构：

```
CREATE TABLE products (
    product_no integer PRIMARY KEY,
    name text,
    price numeric
);

CREATE TABLE orders (
```

```
    order_id integer PRIMARY KEY,
    shipping_address text,
    ...
);

CREATE TABLE order_items (
    product_no integer REFERENCES products,
    order_id integer REFERENCES orders,
    quantity integer,
    PRIMARY KEY (product_no, order_id)
);
```

注意，最后一个表中的主键与外键有重叠。

我们知道，外键不允许创建与任何产品都无关的订单。但如果先创建了引用某产品的订单，随后又删除该产品，会怎样呢？SQL 也允许处理这种情况。直观来看，有以下几种选择：

- 不允许删除被引用的产品
- 同时删除订单
- 其他处理方式？

为说明这一点，
在上面的多对多关系示例中实行如下策略：如果有人要删除仍被订单引用（通过`order_items`）的产品，则不允许删除。如果有人删除订单，则同时删除订单项：

```
CREATE TABLE products (
    product_no integer PRIMARY KEY,
    name text,
    price numeric
);

CREATE TABLE orders (
    order_id integer PRIMARY KEY,
    shipping_address text,
    ...
);

CREATE TABLE order_items (
    product_no integer REFERENCES products ON DELETE RESTRICT,
    order_id integer REFERENCES orders ON DELETE CASCADE,
    quantity integer,
    PRIMARY KEY (product_no, order_id)
);
```

限制删除和级联删除是最常见的两种选项。`RESTRICT`阻止删除被引用的行。`NO ACTION`表示检查约束时若仍存在引用行，就会报错；未指定任何选项时，这是默认行为。
（这两种选项的根本区别在于，`NO ACTION`允许将检查延迟到事务稍后进行，而`RESTRICT`不允许。）`CASCADE`指定当被引用行被删除时，引用它的行也应自动删除。另有两个选项：`SET NULL`和`SET DEFAULT`。它们会在被引用行删除时，分别把引用列设为空值或默认值。注意，这并不能让你免于遵守其他约束。例如，如果某个动作指定了`SET DEFAULT`，但默认值本身不满足外键，则该操作仍会失败。

与`ON DELETE`类似，还有`ON UPDATE`，当被引用列发生变化（更新）时就会触发。可用动作相同。

由于从被引用表中`DELETE`行或`UPDATE`被引用列时，需要扫描引用表以查找匹配旧值的行，因此通常建议为引用列建立索引。由于这并不总是必需，而且索引方式也有很多种，因此声明外键约束时不会自动在引用列上创建索引。

更多关于更新和删除数据的信息请见第 6 章。

最后，我们要提到外键必须引用作为主键或组成唯一约束的列。
如果外键引用的是一个唯一约束，那么在如何匹配空值方面还有一些额外的可能性。这些在CREATE TABLE的参考文档中有解释。

5.3.6. 排他约束

排他约束保证：对任意两行，若对指定列或表达式使用指定操作符进行比较，
则这些操作符比较中至少有一个会返回假或空值。语法如下：

```
CREATE TABLE circles (
    c circle,
    EXCLUDE USING gist (c WITH &&)
);
```

详见`CREATE TABLE ... CONSTRAINT ... EXCLUDE`。

增加一个排他约束会自动创建约束声明中指定类型的索引。

5.4. 系统列

每个表都有几个由系统隐式定义的系统列。因此，这些名称不能用作用户定义列的名称。
（注意，这些限制与名称是否为关键字无关；给名称加引号也不能绕过这些限制。）
你其实不必关心这些列的细节，只需要知道它们存在即可。

`oid`

行的对象标识符（对象 ID）。只有使用`WITH OIDS`创建表，或创建时设置了default_with_oids配置变量，才存在此列。该列的类型是`oid`（与列同名）；
关于该类型的更多信息，参见第 8.18 节。

`tableoid`

包含该行的表的 OID。对于从继承层次（见第 5.8 节）中查询数据的场景，这一列尤其方便，因为如果没有它，就很难判断一行究竟来自哪个具体表。`tableoid`可以与`pg_class`的`oid`列连接，以取得表名。

`xmin`

插入该行版本的事务 ID。（行版本是某一行的一个具体状态；对同一逻辑行的每次更新都会创建一个新的行版本。）

`cmin`

插入事务中的命令标识符（从0开始）。

`xmax`

删除事务的标识（事务 ID）；对于未删除的行版本则为 0。对于一个可见的行版本，该列也可能是非零值。这通常表示删除事务尚未提交，或者一次删除尝试被回滚了。

`cmax`

删除事务中的命令标识符，或者为0。

`ctid`

行版本在其表中的物理位置。注意尽管ctid可以被用来非常快速地定位行版本，但是一个行的ctid会在被更新或者被VACUUM FULL移动时改变。因此，ctid不适合用作长期行标识符。应使用 OID，或者更好地使用用户定义的序列号，来标识逻辑行。

OID 是 32 位的量，由整个集簇共用的单个计数器分配。在大型或长期运行的数据库中，该计数器可能发生回卷。因此，除非采取了确保唯一性的措施，否则不应假定 OID 唯一。如果需要标识表中的行，强烈建议使用序列生成器。不过，只要采取一些额外的预防措施，也可以使用 OID：

- 对于使用 OID 标识行的每个表，都应在其 OID 列上创建唯一约束。有这样的唯一约束（或唯一索引）时，系统会确保不生成与已有行相同的 OID。（当然，这只有在表的行数小于 2^{32}（40 亿）时才有可能；实际上，表的大小最好远小于此值，否则性能可能受影响。）
- 绝不能假定 OID 在不同表之间也唯一；如果需要数据库范围的标识符，应将tableoid与行 OID 组合使用。
- 当然，相关表必须以WITH OIDS创建。从PostgreSQL 8.1 开始，默认值是WITHOUT OIDS。

事务 ID也是 32 位的量。在一个长期运行的数据库中，事务 ID 可能会回卷。只要采取适当的维护措施，这并不是致命问题，详见第 23 章。不过，从长期来看（超过十亿个事务）依赖事务 ID 的唯一性是不明智的。

命令标识符也是 32 位的量。这为单个事务中的SQL命令数设置了一个硬上限：2^{32}（40 亿）。在实践中，这个限制并不是问题 – 注意，这里限制的是SQL命令的数量，而不是处理的行数。另外，从PostgreSQL 8.3 开始，只有真正修改数据库内容的命令才会消耗命令标识符。

5.5. 修改表

创建表之后，如果你意识到自己犯了错误，或者应用需求发生了变化，可以删掉表再重新创建。但如果表中已经有数据，或者该表已被其他数据库对象引用（例如被外键约束引用），这样做就不方便了。因此，PostgreSQL提供了一组命令来修改现有表。注意，这在概念上不同于修改表中保存的数据：这里关注的是修改表的定义，也就是表的结构。

利用这些命令，我们可以：

- 增加列
- 移除列
- 增加约束
- 移除约束
- 修改默认值
- 修改列数据类型
- 重命名列
- 重命名表

所有这些动作都由ALTER TABLE命令执行，其参考页面中包含更详细的信息。

5.5.1. 增加列

要添加一列，可以使用这样的命令：

```
ALTER TABLE products ADD COLUMN description text;
```

新列最初会填入给定的默认值（如果没有指定DEFAULT子句，则填入空值）。

也可以同时为该列定义约束，使用常规语法：

```
ALTER TABLE products ADD COLUMN description text CHECK (description
 <> '');
```

实际上，凡是在CREATE TABLE中可用于列描述的选项，在这里都可以使用。不过要记住，默认值必须满足给定约束，否则ADD就会失败。另一种做法是先把新列正确填好，再在之后添加约束（见下文）。

> **提示**
>
> 添加带默认值的列需要更新表中的每一行（以存储新列值）。不过，如果没有指定默认值，PostgreSQL就可以避免物理更新。因此，如果打算为该列填入的大多数值都不是默认值，最好先添加不带默认值的列，用UPDATE填入正确的值，再按下面所述添加所需的默认值。

5.5.2. 移除列

要删除一列，使用如下命令：

```
ALTER TABLE products DROP COLUMN description;
```

该列中的数据会消失，涉及该列的表约束也会被删除。不过，如果该列被其他表的外键约束引用，PostgreSQL不会静默删除该约束。你可以通过添加CASCADE来授权删除所有依赖该列的对象：

```
ALTER TABLE products DROP COLUMN description CASCADE;
```

关于这个操作背后的一般性机制请见第 5.12 节。

5.5.3. 增加约束

添加约束时使用表约束语法。例如：

```
ALTER TABLE products ADD CHECK (name <> '');
ALTER TABLE products ADD CONSTRAINT some_name UNIQUE (product_no);
ALTER TABLE products ADD FOREIGN KEY (product_group_id) REFERENCES
 product_groups;
```

非空约束不能写成表约束，添加时应使用以下语法：

```
ALTER TABLE products ALTER COLUMN product_no SET NOT NULL;
```

系统会立即检查约束，因此只有表中的数据满足约束，才能将其添加。

5.5.4. 移除约束

要删除约束，需要知道它的名称。如果曾为它指定名称，这很容易；否则系统会分配一个自动生成的名称，需要将其查出来。psql的命令\d *tablename*对此很有帮助；其他接口也可能提供检查表细节的方式。然后执行以下命令：

```
ALTER TABLE products DROP CONSTRAINT some_name;
```

（如果处理的是自动生成的约束名，例如$2，别忘了加上双引号，使其成为有效标识符。）

和删除列一样，如果要删除某些其他对象所依赖的约束，也需要加上CASCADE。例如，外键约束就依赖于被引用列上的唯一约束或主键约束。

除非空约束之外，所有类型的约束都可以用相同方式删除。要删除非空约束，使用：

```
ALTER TABLE products ALTER COLUMN product_no DROP NOT NULL;
```

（请记住，非空约束没有名称。）

5.5.5. 更改列的默认值

要为一个列设置一个新默认值，使用命令：

```
ALTER TABLE products ALTER COLUMN price SET DEFAULT 7.77;
```

注意这不会影响表中任何已经存在的行，它只是为未来的`INSERT`命令改变了默认值。

要移除任何默认值，使用：

```
ALTER TABLE products ALTER COLUMN price DROP DEFAULT;
```

这等同于将默认值设置为空值。相应地，试图删除一个未被定义的默认值并不会引发错误，因为默认值已经被隐式地设置为空值。

5.5.6. 修改列的数据类型

为了将一个列转换为一种不同的数据类型，使用如下命令：

```
ALTER TABLE products ALTER COLUMN price TYPE numeric(10,2);
```

只有当列中的每一个项都能通过一个隐式类型转换为新的类型时该操作才能成功。
如果需要一种更复杂的转换，应该加上一个`USING`子句来指定应该如何把旧值转换为新值。

PostgreSQL将尝试把列的默认值转换为新类型，其他涉及到该列的任何约束也是一样。
但是这些转换可能失败或者产生奇特的结果。
因此最好在修改类型之前先删除该列上所有的约束，
然后在修改完类型后重新加上相应修改过的约束。

5.5.7. 重命名列

要重命名一个列：

```
ALTER TABLE products RENAME COLUMN product_no TO product_number;
```

5.5.8. 重命名表

要重命名一个表：

```
ALTER TABLE products RENAME TO items;
```

5.6. 权限

一旦一个对象被创建，它会被分配一个所有者。所有者通常是执行创建语句的角色。
对于大部分类型的对象，初始状态下只有所有者（或者超级用户）
能够对该对象做任何事情。为了允许其他角色使用它，必须分配权限。

有不同种类的权限：`SELECT`、`INSERT`、`UPDATE`、`DELETE`、`TRUNCATE`、`REFERENCES`、`TRIGGER`、`CREATE`、`CONNECT`、`TEMPORARY`、`EXECUTE`和`USAGE`。
适用于特定对象的权限取决于对象的类型（表、函数等）。关于
PostgreSQL支持的不同权限类型的完整信息，请参阅GRANT参考页。
后续的各节和各章还将展示这些权限是如何使用的。

修改或销毁对象的权利始终只属于该对象的拥有者。

可以使用适合对象类型的ALTER命令，将对象分配给新的拥有者，例如ALTER TABLE。超级用户始终可以这样做；普通角色只有同时是对象的当前拥有者（或拥有者角色的成员），并且是新拥有者角色的成员，才能这样做。

分配权限使用GRANT命令。例如，如果joe是一个已有角色，并且accounts是一个已有表，则可以用以下命令授予更新该表的权限：

```
GRANT UPDATE ON accounts TO joe;
```

用ALL代替某个具体权限，会授予与该对象类型相关的全部权限。

特殊的“用户”名PUBLIC可用于向系统中的每个用户授予权限。此外，当数据库用户很多时，可以设置“组”角色来协助管理权限 – 详见第 20 章。

要撤销权限，使用名称贴切的REVOKE命令：

```
REVOKE ALL ON accounts FROM PUBLIC;
```

对象拥有者的特殊权限（即执行DROP，GRANT，REVOKE等操作的权利）始终隐含在拥有者身份中，不能授予或撤销。不过，对象拥有者可以选择撤销自己的普通权限，例如将表设为对自己和其他人都只读。

通常，只有对象的拥有者（或超级用户）可以授予或撤销对象上的权限。不过，也可以在授予权限时“附带授权选项”，使接收者有权进一步将该权限授予其他人。如果之后撤销了授权选项，所有从该接收者直接或经过授权链取得此权限的用户都会失去权限。详见GRANT和REVOKE参考页。

5.7. 模式

一个PostgreSQL数据库集簇中包含一个或更多命名的数据库。用户和用户组被整个集簇共享，但数据库之间不共享其他任何数据。任何一条到服务器的客户端连接都只能访问单个数据库中的数据，也就是连接请求中指定的那个数据库。

> **注意**
>
> 一个集簇的用户并不必拥有访问集簇中每一个数据库的权限。用户名的共享意味着不可能在同一个集簇中出现重名的不同用户，例如两个数据库中都有叫joe的用户。但系统可以被配置为只允许joe访问某些数据库。

一个数据库包含一个或多个命名的模式，模式中又包含表。模式还包含其他类型的命名对象，包括数据类型、函数和操作符。在同一个模式中，同一类型的两个对象不能有相同的名称。此外，表、序列、索引、视图、物化视图和外部表共享同一个名字空间，因此例如当它们位于同一模式中时，索引和表就必须具有不同的名称。相同的对象名可以在不同模式中重复使用而不发生冲突；例如，schema1和myschema都可以包含名为mytable的表。与数据库不同，模式并不是被严格隔离的：只要拥有相应权限，用户就可以访问其所连接数据库中任意模式里的对象。

下面是一些使用模式的原因：

- 允许多个用户使用一个数据库并且不会互相干扰。

- 将数据库对象组织成逻辑组以便更容易管理。
- 第三方应用的对象可以放在独立的模式中，这样它们就不会与其他对象的名称发生冲突。

模式类似于操作系统层的目录，但是模式不能嵌套。

5.7.1. 创建模式

要创建一个模式，可使用CREATE SCHEMA命令，并且给出选择的模式名称。例如：

```
CREATE SCHEMA myschema;
```

在一个模式中创建或访问对象，需要使用由模式名和表名构成的限定名，
模式名和表名之间以点号分隔：

```
schema.table
```

在任何需要一个表名的地方都可以这样用，
包括表修改命令和后续章节要讨论的数据访问命令（为了简洁我们在这里只谈到表，
但是这种方式对其他类型的命名对象同样有效，例如类型和函数）。

实际上，也可以使用更通用的语法

```
database.schema.table
```

，但目前这只是为了形式上符合 SQL 标准。如果写出数据库名称，它必须与当前连接的数据库相同。

因此，如果要在一个新模式中创建一个表，可用：

```
CREATE TABLE myschema.mytable (
 ...
);
```

要删除一个为空的模式（其中的所有对象已经被删除），可用：

```
DROP SCHEMA myschema;
```

要删除一个模式以及其中包含的所有对象，可用：

```
DROP SCHEMA myschema CASCADE;
```

有关于此的更一般的机制请参见第 5.12 节。

我们常常希望创建一个由其他人所拥有的模式（因为这是将用户动作限制在良定义的名字空间中的方法之一）。其语法是：

```
CREATE SCHEMA schemaname AUTHORIZATION username;
```

我们甚至可以省略模式名称，在此种情况下模式名称将会使用用户名，参见第 5.7.6 节。

以pg_开头的模式名被保留用于系统目的，所以不能被用户所创建。

5.7.2. 公共模式

在前面的小节中，我们创建表时都没有指定模式名。默认情况下，这些表（以及其他对象）会自动放入名为“public”的模式中。每个新数据库都包含这样一个模式。因此，
下面两条命令是等效的：

```
CREATE TABLE products ( ... );
```

以及：

```
CREATE TABLE public.products ( ... );
```

5.7.3. 模式搜索路径

限定名写起来很繁琐，而且通常最好不要把某个特定模式名硬编码到应用中。因此，表通常通过非限定名来引用，也就是只写表名。系统会沿着一条搜索路径来决定该名称指的是哪个表；搜索路径就是要查找的一组模式列表。搜索路径中第一个匹配的表会被视为目标表。如果搜索路径中没有任何匹配，就会报错，即使数据库的其他模式中存在同名表也是如此。

搜索路径中的第一个模式被称为当前模式。除了是第一个被搜索的模式外，如果`CREATE TABLE`命令没有指定模式名，它将是新创建表所在的模式。

要显示当前搜索路径，可使用下面的命令：

```
SHOW search_path;
```

在默认设置下这将返回：

```
 search_path
--------------
 "$user",public
```

第一个元素说明一个和当前用户同名的模式会被搜索。如果不存在这个模式，该项将被忽略。第二个元素指向我们已经见过的公共模式。

搜索路径中第一个存在的模式，是创建新对象时的默认位置。这就是默认情况下对象会被创建在公共模式中的原因。当对象在其他任何未限定模式的上下文中被引用时（表修改、数据修改或查询命令），系统会沿搜索路径查找，直到找到匹配对象为止。因此，在默认配置中，任何非限定访问仍然只能指向公共模式。

要把新模式放在搜索路径中，我们可以使用：

```
SET search_path TO myschema,public;
```

（我们在这里省略了`$user`，因为我们并不立即需要它）。然后我们可以删除该表而无需使用模式进行限定：

```
DROP TABLE mytable;
```

同样，由于`myschema`是路径中的第一个元素，新对象默认也会创建在其中。

我们也可以这样写：

```
SET search_path TO myschema;
```

这样一来，在没有显式限定时，我们就不再能访问公共模式了。公共模式本身并无特殊之处，只是默认存在而已；它同样可以被删除。

其他操作模式搜索路径的方法请见第 9.25 节。

搜索路径对于数据类型名称、函数名称和操作符名称的作用与表名一样。数据类型和函数名称可以使用和表名完全相同的限定方式。如果我们需要在一个表达式中写一个限定的操作符名称，我们必须写成一种特殊的形式：

```
OPERATOR(schema.operator)
```

这是为了避免句法歧义。例如：

```
SELECT 3 OPERATOR(pg_catalog.+) 4;
```

实际上我们通常都会依赖于搜索路径来查找操作符，因此没有必要去写如此“丑陋”的东西。

5.7.4. 模式和权限

默认情况下，用户无法访问他们不拥有的模式中的任何对象。要允许这样做，
模式的所有者必须授予该模式上的USAGE权限。要允许用户使用模式中的对象，
可能需要授予其他权限，适用于该对象。

也可以允许用户在他人的模式中创建对象。为此，需要授予该模式上的CREATE权限。注意，
默认情况下，每个人都具有CREATE和USAGE权限，它们作用于模式public。
这使所有能够连接到某个数据库的用户都可以在它的public模式中创建对象。
如果你不希望允许这样做，可以撤销该权限：

```
REVOKE CREATE ON SCHEMA public FROM PUBLIC;
```

（第一个“public”是模式名，第二个“public”表示“每个用户”。前一种用法是标识符，
后一种是关键字，因此大小写不同；请回顾第 4.1.1 节。）

5.7.5. 系统目录模式

除public和用户创建的模式之外，每一个数据库还包括一个pg_catalog模式，
它包含了系统表和所有内置的数据类型、函数以及操作符。pg_catalog
总是搜索路径的一个有效部分。如果没有在路径中显式地包括该模式，它将在路径中的模式
之前被搜索。这保证了内置的名称总是能被找到。然而，
如果我们希望用用户定义的名称重载内置的名称，可以显式的将pg_catalog
放在搜索路径的末尾。

在PostgreSQL 7.3 之前的版本中，以pg_开头的表名是被保留的。现在不再如此：如果你愿
意，可以在任何非系统模式中创建这样名字的表。不过，
最好还是继续避免使用这样的名称，以确保将来某个版本定义出与你的表同名的系统表时，
你不会遇到冲突。系统表会继续遵循以pg_开头的约定，因此只要用户避免使用pg_前缀，
它们就不会与未经限定的用户表名发生冲突。

5.7.6. 使用方式

模式可以用多种方式组织数据。下面几种使用方式是推荐的做法，
并且很容易得到默认配置的支持：

- 如果你不创建任何模式，那么所有用户都将隐式地访问 public 模式。这模拟了根本没有模式可用的情况。
  这种设置主要推荐用于数据库中只有单个用户或少数几个相互合作的用户的场景。
  这种设置也允许从不支持模式的环境平滑过渡。

- 你可以为每个用户创建一个与其同名的模式。请记住，默认搜索路径以$user开头，
  它会解析为用户名。因此，如果每个用户都有单独的模式，
  他们默认访问的就是自己的模式。

  如果采用这种设置，你可能还想撤销对 public 模式的访问（或者干脆删除它），使用户真正被限制在他们自己的模式中。

- 如果要安装共享应用（所有人都要使用的表、第三方提供的附加函数等），
  可以把它们放进单独的模式中。记得授予适当的权限，以便其他用户能够访问它们。

这样，用户既可以通过带模式名的限定名来引用这些附加对象，
也可以按自己的需要把这些附加模式加入搜索路径。

5.7.7. 可移植性

在 SQL 标准中，不存在同一模式中的对象由不同用户拥有这一概念。此外，有些实现不允许创建与其所有者名称不同的模式。事实上，
在那些只实现了标准中基本模式支持的数据库系统中，
模式和用户这两个概念几乎是等价的。因此，很多用户认为限定名实际上就是 *username*.*tablename*。如果你为每个用户都创建一个独立模式，
PostgreSQL的行为实际上也会是这样。

同样，SQL 标准中也没有public模式这一概念。为了尽可能符合标准，你不应使用（甚至可以移除）public模式。

当然，某些SQL数据库系统可能根本没有实现模式，或者提供（很可能是有限制地）
允许跨数据库访问的命名空间。如果需要使用这样的系统，为了获得最好的可移植性，
最好不要使用模式。

5.8. 继承

PostgreSQL实现了表继承，这对数据库设计者来说是一种有用的工具（SQL:
1999及其后的版本定义了一种类型继承特性，但和这里介绍的继承有很大的不同）。

让我们从一个示例开始：假设我们要为城市建立一个数据模型。每个州有很多城市，
但只有一个首府。我们希望能够快速检索任意特定州的首府城市。
这可以通过创建两个表来实现：一个用于州首府，另一个用于非首府城市。然而，
当我们想要查询某个城市的数据，而不关心它是不是首府时，
会发生什么？继承特性将有助于解决这个问题。我们可以将capitals
表定义为继承自cities表：

```
CREATE TABLE cities (
    name            text,
    population      float,
    altitude       int     -- in feet
);

CREATE TABLE capitals (
    state           char(2)
) INHERITS (cities);
```

在这种情况下，capitals表继承了它的父表cities的所有列。
州首府还有一个额外的列state用来表示它所属的州。

在PostgreSQL中，一个表可以从0个或者多个其他表继承，
而对一个表的查询则可以引用一个表的所有行或者该表的所有行加上它所有的后代表。
默认情况是后一种行为。例如，下面的查询将查找所有高度高于500尺的城市的名称，
包括州首府：

```
SELECT name, altitude
    FROM cities
    WHERE altitude > 500;
```

对于来自PostgreSQL教程（见第 2.1 节）的示例数据，它将返回：

```
   name    | altitude
-----------+-----------
 Las Vegas |      2174
```

```
 Mariposa  |      1953
 Madison   |       845
```

另一方面，下面的查询将找到所有高度超过 500 尺且不是州首府的城市：

```
SELECT name, altitude
    FROM ONLY cities
    WHERE altitude > 500;

   name    | altitude
-----------+-----------
 Las Vegas |      2174
 Mariposa  |      1953
```

这里的ONLY关键词指示查询只被应用于cities上，而其他在继承层次中位于cities之下的其他表都不会被该查询涉及。很多我们已经讨论过的命令（如SELECT、UPDATE和DELETE）都支持ONLY关键词。

我们也可以在表名后写上一个*来显式地将后代表包括在查询范围内：

```
SELECT name, altitude
    FROM cities*
    WHERE altitude > 500;
```

写*并非必需，因为这种行为就是默认的（除非你更改了sql_inheritance配置选项的设置）。不过，写*或许有助于强调将会搜索额外的表。

在某些情况下，我们可能希望知道一个特定行来自于哪个表。每个表中的系统列tableoid可以告诉我们行来自于哪个表：

```
SELECT c.tableoid, c.name, c.altitude
FROM cities c
WHERE c.altitude > 500;
```

将会返回：

```
 tableoid |   name    | altitude
----------+-----------+-----------
   139793 | Las Vegas |      2174
   139793 | Mariposa  |      1953
   139798 | Madison   |       845
```

（如果重新生成这个结果，可能会得到不同的OID数字。）通过与pg_class进行连接可以看到实际的表名：

```
SELECT p.relname, c.name, c.altitude
FROM cities c, pg_class p
WHERE c.altitude > 500 AND c.tableoid = p.oid;
```

将会返回：

```
 relname  |   name    | altitude
----------+-----------+-----------
 cities   | Las Vegas |      2174
 cities   | Mariposa  |      1953
 capitals | Madison   |       845
```

继承不会自动地将来自INSERT或COPY命令的数据传播到继承层次中的其他表中。在我们的示例中，下面的INSERT语句将会失败：

```
INSERT INTO cities (name, population, altitude, state)
VALUES ('New York', NULL, NULL, 'NY');
```

我们也许会希望数据能以某种方式被路由到capitals表中，但这不会发生：INSERT 总是向指定的表中插入。在某些情况下，可以通过使用一个规则（见第 37 章）将插入动作重定向。但是这对上面的情况并没有帮助，因为cities表根本就不包含state列，因而这个命令会在触发规则之前就被拒绝。

父表上的所有检查约束和非空约束都将自动被它的后代所继承。其他类型的约束（唯一、主键和外键约束）则不会被继承。

一个表可以从超过一个的父表继承，在这种情况下它拥有父表们所定义的列的并集。任何定义在子表上的列也会被加入到其中。如果在这个集合中出现重名列，那么这些列将被“合并”，这样在子表中只会有一个这样的列。重名列能被合并的前提是这些列必须具有相同的数据类型，否则会导致错误。合并后的列将带有它所来自的任何一个列定义中的全部检查约束的拷贝，并且只要其中任何一个被标记为非空，它就会被标记为非空。

表继承通常在创建子表时建立，即通过CREATE TABLE语句中的INHERITS子句。已经创建好的表也可以通过ALTER TABLE的INHERIT变体再增加一个新的父表关系。要这么做，新子表必须已经包含与父表同名且数据类型相同的列。子表还必须包含与父表相同的检查约束和检查表达式。类似地，也可以使用ALTER TABLE的NO INHERIT变体，从子表中移除一条继承链接。动态添加和移除继承链接可用于实现表分区（见第 5.9 节）。

一种创建将来要用作子表的新表的方法，是在CREATE TABLE中使用LIKE子句。这样会创建一个与源表具有相同列的新表。如果源表上定义了任何CHECK约束，可以使用LIKE的INCLUDING CONSTRAINTS选项，让新表也包含与父表相同的约束。

当有任何一个子表存在时，父表不能被删除。当子表的列或者检查约束继承于父表时，它们也不能被删除或修改。如果希望移除一个表和它的所有后代，一种简单的方法是使用CASCADE选项删除父表。

ALTER TABLE会把列的数据定义或检查约束上的任何变化沿着继承层次向下传播。同样，删除被其他表依赖的列只能使用CASCADE选项。对于同名列的合并与拒绝，ALTER TABLE遵循与CREATE TABLE相同的规则。

注意表访问权限是如何处理的。查询父表可以自动访问子表中的数据，而无需进一步的访问权限检查。这保持了数据（也）在父表中的外观。但是，直接访问子表并不会被自动允许，需要授予进一步的权限。

5.8.1. 注意事项

注意，并非所有 SQL 命令都能作用于继承层次。用于数据查询、数据修改或模式修改的命令（例如SELECT、UPDATE、DELETE、大多数ALTER TABLE变体，但不包括INSERT或ALTER TABLE ... RENAME）通常默认包含子表，并支持使用ONLY记法将其排除。用于数据库维护和调优的命令（例如REINDEX、VACUUM）通常只作用于独立的物理表，不支持沿继承层次递归。各条命令的具体行为都记录在相应参考页中（SQL Commands）。

继承特性的一个严重限制是，索引（包括唯一约束）和外键约束只作用于单个表，而不作用于其继承子表。对于外键约束的引用端和被引用端，这一点都成立。因此，沿用上面的示例：

- 如果我们把cities.name声明为UNIQUE或PRIMARY KEY，这并不能阻止capitals表中出现与cities中城市同名的行。而且这些重复行默认还会出现在针对cities的查询结果中。事实上，默认情况下capitals根本没有唯一约束，因此它可以包含多行同名记录。你当然可以给capitals添加唯一约束，但这仍无法阻止相对于cities的重复。

- 相似地，如果我们指定cities.name REFERENCES某个其他表，该约束不会自动地传播到capitals。在此种情况下，我们可以变通地在capitals
  上手工创建一个相同的REFERENCES约束。

- 如果让另一个表的某列REFERENCES cities(name)，那么该表可以包含城市名称，但不能包含首府名称。对于这种情况，并没有什么好的变通办法。

这些不足未来可能会在某个发行版中修复，但在此期间，在决定继承是否适合你的应用时，
仍需要非常小心。

5.9. 分区

PostgreSQL支持基本的表分区。
本节介绍为什么以及如何把分区作为数据库设计的一部分来实现。

5.9.1. 概述

分区是指将逻辑上的一个大表拆分成较小的物理部分。分区可以带来以下好处：

- 在某些情况下查询性能能够显著提升，
  特别是当那些访问压力大的行在一个分区或者少数几个分区时。
  分区替代了索引的前导列，从而减小索引大小，
  使索引中被大量使用的部分更有可能容纳在内存中。

- 当查询或更新访问单个分区的很大一部分时，
  可以通过使用该分区的顺序扫描来提高性能，而不是使用索引，
  这将需要分散在整个表中的随机访问读取。

- 如果在分区设计中考虑到了这种使用模式，
  就可以通过添加或移除分区来完成批量加载和删除。ALTER TABLE NO INHERIT和DROP TABLE都比批量操作快得多。这些命令还完全避免了批量DELETE所导致的VACUUM开销。

- 很少使用的数据可以被迁移到便宜且较慢的存储介质上。

通常只有表本身非常大时，这些好处才值得考虑。表从多大开始受益于分区，
取决于具体应用；不过，一条经验法则是表的大小应超过数据库服务器的物理内存。

目前，PostgreSQL通过表继承来支持分区。每个分区都必须作为单个父表的子表创建。
父表本身通常为空；它的存在只是为了表示整个数据集。在尝试设置分区之前，
你应当先熟悉继承（见第 5.8 节）。

在PostgreSQL中可以实现以下几种分区形式：

范围分区

按照某个键列或一组列定义的“范围”对表进行分区，
不同分区所分配的值范围互不重叠。例如，可以按日期范围，
或特定业务对象的标识符范围分区。

列表分区

通过明确列出每个分区包含哪些键值来对表进行分区。

5.9.2. 实现分区

要设置一个分区表，可以按以下步骤操作：

1. 创建所有分区都将继承的“主”表。

这个表将不包含数据。不要在这个表上定义任何检查约束，
除非想让它们等同地应用到所有分区上。
在这个表上定义索引或者唯一约束也没有意义。

2. 创建若干“子”表，每个都从主表继承。通常，
这些表不会在从主表继承的列集合之外增加任何列。

   我们将这些子表称为分区，尽管它们在各方面都是普通的PostgreSQL表。

3. 为各个分区表添加表约束，定义每个分区中允许的键值。

   典型示例如下：

```
CHECK ( x = 1 )
CHECK ( county IN ( 'Oxfordshire', 'Buckinghamshire',
 'Warwickshire' ))
CHECK ( outletID >= 100 AND outletID < 200 )
```

   确保约束保证不同分区所允许的键值互不重叠。常见错误是设置如下范围约束：

```
CHECK ( outletID BETWEEN 100 AND 200 )
CHECK ( outletID BETWEEN 200 AND 300 )
```

   这是错误的，因为无法明确键值 200 属于哪个分区。

   注意，范围分区和列表分区在语法上没有区别；这些术语只是描述性的。

4. 对于每个分区，在键列上创建索引，以及其他所需的索引。（键索引不是严格必需的，
但在大多数场景中是有帮助的。如果希望键值唯一，
则应始终为每个分区创建唯一或主键约束。）
5. 可以选择定义一个触发器或规则，把插入到主表的数据重定向到适当的分区。
6. 确认constraint_exclusion配置参数在`postgresql.conf`中没有被禁用，
否则查询将无法按预期得到优化。

例如，假设正在为一家大型冰淇淋公司构建数据库。该公司每天测量最高温度，
并统计各个地区的冰淇淋销量。从概念上看，我们需要如下表：

```
CREATE TABLE measurement (
    city_id         int not null,
    logdate         date not null,
    peaktemp        int,
    unitsales       int
);
```

我们知道，大多数查询只会访问最近一周、一个月或一个季度的数据，
因为该表主要用于为管理层生成在线报表。为减少需要存储的旧数据量，决定只保留最近 3 年的数据，并在每月月初删除最早一个月的数据。

在这种情况下，可以利用分区来帮助满足测量表的各种不同需求。按照上面概述的步骤，
可以这样设置分区：

1. 主表就是`measurement`表，完全按上面的方式声明。
2. 然后，为每个活动月份创建一个分区：

```
CREATE TABLE measurement_y2006m02 ( ) INHERITS (measurement);
CREATE TABLE measurement_y2006m03 ( ) INHERITS (measurement);
...
CREATE TABLE measurement_y2007m11 ( ) INHERITS (measurement);
CREATE TABLE measurement_y2007m12 ( ) INHERITS (measurement);
CREATE TABLE measurement_y2008m01 ( ) INHERITS (measurement);
```

   每个分区本身都是完整的表，但它们从`measurement`表继承其定义。

这解决了我们的一个问题：删除旧数据。每个月，我们只需对最旧的子表执行DROP TABLE，并为新月份的数据创建一个新的子表。

3. 我们必须提供互不重叠的表约束。与其像上面那样只创建分区表，
表创建脚本其实应该是：

```
CREATE TABLE measurement_y2006m02 (
    CHECK ( logdate >= DATE '2006-02-01' AND logdate < DATE
'2006-03-01' )
) INHERITS (measurement);
CREATE TABLE measurement_y2006m03 (
    CHECK ( logdate >= DATE '2006-03-01' AND logdate < DATE
'2006-04-01' )
) INHERITS (measurement);
...
CREATE TABLE measurement_y2007m11 (
    CHECK ( logdate >= DATE '2007-11-01' AND logdate < DATE
'2007-12-01' )
) INHERITS (measurement);
CREATE TABLE measurement_y2007m12 (
    CHECK ( logdate >= DATE '2007-12-01' AND logdate < DATE
'2008-01-01' )
) INHERITS (measurement);
CREATE TABLE measurement_y2008m01 (
    CHECK ( logdate >= DATE '2008-01-01' AND logdate < DATE
'2008-02-01' )
) INHERITS (measurement);
```

4. 我们可能还需要在键列上创建索引：

```
CREATE INDEX measurement_y2006m02_logdate ON measurement_
y2006m02 (logdate);
CREATE INDEX measurement_y2006m03_logdate ON measurement_
y2006m03 (logdate);
...
CREATE INDEX measurement_y2007m11_logdate ON measurement_
y2007m11 (logdate);
CREATE INDEX measurement_y2007m12_logdate ON measurement_
y2007m12 (logdate);
CREATE INDEX measurement_y2008m01_logdate ON measurement_
y2008m01 (logdate);
```

我们此时选择不添加更多索引。

5. 我们希望应用程序能够执行INSERT INTO measurement ...，并使数据被重定向到适当的分区表。可以通过在主表上附加合适的触发器函数来实现这一点。
如果数据只添加到最新的分区，可以使用非常简单的触发器函数：

```
CREATE OR REPLACE FUNCTION measurement_insert_trigger()
RETURNS TRIGGER AS $$
BEGIN
    INSERT INTO measurement_y2008m01 VALUES (NEW.*);
    RETURN NULL;
END;
$$
LANGUAGE plpgsql;
```

创建函数后，创建一个调用该触发器函数的触发器：

```
CREATE TRIGGER insert_measurement_trigger
    BEFORE INSERT ON measurement
```

```
    FOR EACH ROW EXECUTE PROCEDURE measurement_insert_trigger()
;
```

必须每月重新定义触发器函数，使其始终指向当前分区。不过，触发器定义无需更新。

我们可能希望插入数据时，由服务器自动定位应添加该行的分区。
这可以通过更复杂的触发器函数实现，例如：

```
CREATE OR REPLACE FUNCTION measurement_insert_trigger()
RETURNS TRIGGER AS $$
BEGIN
    IF ( NEW.logdate >= DATE '2006-02-01' AND
         NEW.logdate < DATE '2006-03-01' ) THEN
        INSERT INTO measurement_y2006m02 VALUES (NEW.*);
    ELSIF ( NEW.logdate >= DATE '2006-03-01' AND
            NEW.logdate < DATE '2006-04-01' ) THEN
        INSERT INTO measurement_y2006m03 VALUES (NEW.*);
    ...
    ELSIF ( NEW.logdate >= DATE '2008-01-01' AND
            NEW.logdate < DATE '2008-02-01' ) THEN
        INSERT INTO measurement_y2008m01 VALUES (NEW.*);
    ELSE
        RAISE EXCEPTION 'Date out of range.  Fix the
 measurement_insert_trigger() function!';
    END IF;
    RETURN NULL;
END;
$$
LANGUAGE plpgsql;
```

触发器定义与之前相同。注意，每个`IF`测试必须与其子表的`CHECK`约束完全一致。

当该函数比单月形式更加复杂时，并不需要频繁地更新它，
因为可以在需要的时候提前加入分支。

> **注意**
>
> 在实践中，如果大部分插入都会进入最新的分区，最好先检查它。为了简洁，
> 我们为触发器的检查采用了和本例中其他部分一致的顺序。

如我们所见，一个复杂的分区方案可能需要大量的DDL。在上面的示例中，
我们可能每个月创建一个新分区，因此编写一个脚本来自动生成所需要的DDL可能会更好。

5.9.3. 分区管理

通常，最初定义表时建立的分区集合并不会保持不变。常见的需求是删除旧数据分区，
并定期为新数据添加新分区。分区最重要的优势之一，恰恰在于它允许通过操纵分区结构，
而不是物理地移动大量数据，来近乎瞬时地完成这种原本非常痛苦的任务。

移除旧数据最简单的选择，是直接删除不再需要的分区：

```
DROP TABLE measurement_y2006m02;
```

由于不必逐条删除每条记录，这可以非常快速地删除数百万条记录。

另一种通常更可取的选择，是将该分区从分区表中移除，但保留它作为独立表的可访问性：

```
ALTER TABLE measurement_y2006m02 NO INHERIT measurement;
```

这样在删除数据之前，还可以对其执行进一步的操作。例如，这常常是使用COPY、pg_dump或类似工具备份数据的好时机。也可能是把数据聚合成更小格式、执行其他数据处理或运行报告的好时机。

类似地，我们可以添加一个新分区来处理新数据。可以像上面创建初始分区那样，在分区表中创建一个空分区：

```
CREATE TABLE measurement_y2008m02 (
    CHECK ( logdate >= DATE '2008-02-01' AND logdate < DATE '2008
-03-01' )
) INHERITS (measurement);
```

作为替代，有时更方便的做法是在分区结构之外创建新表，之后再把它变成正式的分区。这允许数据在出现在分区表中之前先被载入、检查和转换：

```
CREATE TABLE measurement_y2008m02
  (LIKE measurement INCLUDING DEFAULTS INCLUDING CONSTRAINTS);
ALTER TABLE measurement_y2008m02 ADD CONSTRAINT y2008m02
   CHECK ( logdate >= DATE '2008-02-01' AND logdate < DATE '2008-03
-01' );
\copy measurement_y2008m02 from 'measurement_y2008m02'
-- possibly some other data preparation work
ALTER TABLE measurement_y2008m02 INHERIT measurement;
```

5.9.4. 分区和约束排除

约束排除是一种查询优化技术，可提高按上述方式定义的分区表的性能。例如：

```
SET constraint_exclusion = on;
SELECT count(*) FROM measurement WHERE logdate >= DATE '2008-01
-01';
```

如果没有约束排除，上述查询会扫描measurement表的每个分区。启用约束排除后，规划器会检查每个分区的约束，并尝试证明该分区不需要扫描，因为它不可能包含满足查询WHERE子句的行。如果规划器能够证明这一点，就会将该分区排除在查询计划之外。

可以使用EXPLAIN命令，显示启用constraint_exclusion时的计划与禁用时的计划之间的差异。对于这种表结构，典型的未优化计划如下：

```
SET constraint_exclusion = off;
EXPLAIN SELECT count(*) FROM measurement WHERE logdate >= DATE
 '2008-01-01';

                                           QUERY PLAN
-----------------------------------------------------------------
----------------------------
 Aggregate  (cost=158.66..158.68 rows=1 width=0)
   ->  Append  (cost=0.00..151.88 rows=2715 width=0)
         ->  Seq Scan on measurement  (cost=0.00..30.38 rows=543
 width=0)
               Filter: (logdate >= '2008-01-01'::date)
         ->  Seq Scan on measurement_y2006m02 measurement  (cost=0.
00..30.38 rows=543 width=0)
               Filter: (logdate >= '2008-01-01'::date)
         ->  Seq Scan on measurement_y2006m03 measurement  (cost=0.
00..30.38 rows=543 width=0)
               Filter: (logdate >= '2008-01-01'::date)
...
         ->  Seq Scan on measurement_y2007m12 measurement  (cost=0.
00..30.38 rows=543 width=0)
```

```
               Filter: (logdate >= '2008-01-01'::date)
         ->  Seq Scan on measurement_y2008m01 measurement  (cost=0.
00..30.38 rows=543 width=0)
               Filter: (logdate >= '2008-01-01'::date)
```

某些或全部分区可能使用索引扫描，而不是全表顺序扫描，但这里的重点是，
为回答该查询，根本不需要扫描较旧的分区。启用约束排除后，可以得到代价明显更低、
结果却相同的计划：

```
SET constraint_exclusion = on;
EXPLAIN SELECT count(*) FROM measurement WHERE logdate >= DATE
 '2008-01-01';
                                          QUERY PLAN
-----------------------------------------------------------------
----------------------------
 Aggregate  (cost=63.47..63.48 rows=1 width=0)
   ->  Append  (cost=0.00..60.75 rows=1086 width=0)
         ->  Seq Scan on measurement  (cost=0.00..30.38 rows=543
 width=0)
               Filter: (logdate >= '2008-01-01'::date)
         ->  Seq Scan on measurement_y2008m01 measurement  (cost=0.
00..30.38 rows=543 width=0)
               Filter: (logdate >= '2008-01-01'::date)
```

注意，约束排除只由CHECK约束驱动，与是否存在索引无关。因此，
不必在键列上定义索引。是否需要为某个分区创建索引，
取决于预期查询扫描该分区时通常会扫描其中的大部分，还是仅扫描一小部分。
后一种情况下索引会有帮助，前一种情况下则不会。

constraint_exclusion的默认（也是推荐的）设置不是on也不是off，而是一种被称为
partition的中间设置，这会导致该技术仅被应用于可能访问分区表的查询。on
设置导致规划器检查所有查询中的CHECK约束，甚至是那些不太可能受益的简单查询。

5.9.5. 备选分区方法

将插入重定向到适当分区表的另一种方法，是在根表上设置规则来代替触发器。例如：

```
CREATE RULE measurement_insert_y2006m02 AS
ON INSERT TO measurement WHERE
    ( logdate >= DATE '2006-02-01' AND logdate < DATE '2006-03
-01' )
DO INSTEAD
    INSERT INTO measurement_y2006m02 VALUES (NEW.*);
...
CREATE RULE measurement_insert_y2008m01 AS
ON INSERT TO measurement WHERE
    ( logdate >= DATE '2008-01-01' AND logdate < DATE '2008-02
-01' )
DO INSTEAD
    INSERT INTO measurement_y2008m01 VALUES (NEW.*);
```

规则的开销明显高于触发器，但这种开销每个查询只支付一次，而不是每行一次，
因此在批量插入时，这种方法可能有优势。不过，在大多数情况下，
触发器方法的性能更好。

注意COPY会忽略规则。如果想要使用COPY插入数据，
则需要拷贝到正确的分区表而不是根表中。COPY会引发触发器，
因此在使用触发器方法时可以正常使用它。

规则方法的另一个缺点是，如果规则集合无法覆盖插入日期，
则没有简单的方法能够强制产生错误，数据将会无声无息地进入到根表中。

也可以使用UNION ALL视图来安排分区，而不是使用表继承。例如：

```
CREATE VIEW measurement AS
          SELECT * FROM measurement_y2006m02
UNION ALL SELECT * FROM measurement_y2006m03
...
UNION ALL SELECT * FROM measurement_y2007m11
UNION ALL SELECT * FROM measurement_y2007m12
UNION ALL SELECT * FROM measurement_y2008m01;
```

但是，重新创建视图的需要给数据集分区的添加和删除增加了一个额外步骤。实践中，
与使用继承相比，这种方法没有什么可取之处。

5.9.6. 注意事项

分区表有以下注意事项：

- 没有自动方式验证所有CHECK约束是否互斥。与手工逐个编写相比，
  通过代码生成分区并创建和/或修改相关对象更安全。

- 这里展示的方案假定行的分区键列值永不改变，
  或者至少不会改变到必须把该行移入另一个分区的程度。由于CHECK约束的存在，
  试图那样做的UPDATE将会失败。如果需要处理这种情况，
  可以在分区表上放置合适的更新触发器，但这会让整个结构的管理复杂得多。

- 如果手动执行VACUUM或ANALYZE命令，不要忘记需要在每个分区上分别运行。例如，
  以下命令：

  ```
  ANALYZE measurement;
  ```

  只会处理根表。

约束排除有以下注意事项：

- 只有查询的WHERE子句包含常量（或者外部提供的参数）时，约束排除才能有效果。
  例如，针对一个非不可变函数（如CURRENT_TIMESTAMP）的比较不能被优化，
  因为规划器不知道该函数的值在运行时会落到哪个分区中。

- 保持分区约束尽量简单，否则规划器可能无法证明哪些分区不需要访问。
  如前面的示例所示，对列表分区使用简单的等值条件，对范围分区使用简单的范围测试。
  一条很好的经验法则是：分区约束应只包含分区列与常量之间使用 B-树可索引操作符的比较。

- 约束排除期间会检查根表的所有分区上的全部约束，
  因此大量分区很可能显著增加查询规划时间。使用这些技术进行分区，
  大概在不超过一百个分区时效果较好；不要尝试使用成千上万个分区。

5.10. 外部数据

PostgreSQL实现了部分的SQL/MED规定，允许我们使用普通SQL查询来访问位于PostgreSQL之外的数据。这种数据被称为外部数据（注意这种用法不要和外键混淆，
后者是数据库中的一种约束）。

外部数据可以借助外部数据包装器来访问。外部数据包装器是一个库，
它可以与外部数据源通信，并隐藏连接数据源以及从中获取数据的细节。在contrib
模块中有一个外部数据包装器，它可以读取服务器上存储的普通数据文件。

其他类型的外部数据包装器可以在第三方产品中找到。
如果这些现有的外部数据包装器都不能满足需要，也可以自行编写，参见第 50 章。

要访问外部数据，我们需要创建一个外部服务器对象，
它根据某个特定外部数据包装器所使用的一组选项，
定义如何连接到某个特定的外部数据源。然后还需要创建一个或多个外部表，
用于定义远程数据的结构。外部表在查询中可以像普通表一样使用，但它在 PostgreSQL 服务器中并不存储数据。每次使用外部表时，
PostgreSQL都会要求外部数据包装器从外部数据源获取数据。

访问远程数据可能需要在外部数据源进行认证。这些信息可以通过一个用户映射提供，
它会基于当前的PostgreSQL角色提供附加选项。

目前，外部表是只读的。这一限制可能会在将来的版本中解决。

5.11. 其他数据库对象

表是关系数据库结构中的核心对象，因为它们承载着数据。
但它们并不是数据库中唯一存在的对象。还可以创建许多其他种类的对象，
以便让数据的使用和管理更加高效或方便。本章不会讨论这些对象，但这里给出一个列表，
让你了解有哪些可能性：

- 视图
- 函数和操作符
- 数据类型和域
- 触发器和重写规则

这些主题的详细信息请见第 V 部分。

5.12. 依赖跟踪

当我们创建一个涉及到很多具有外键约束、视图、触发器、
函数等的表的复杂数据库结构时，我们隐式地创建了一张对象之间的依赖关系网。例如，
具有一个外键约束的表依赖于它所引用的表。

为了保证整个数据库结构的完整性，
PostgreSQL确保我们无法删除仍然被其他对象依赖的对象。例如，尝试删除第 5.3.5 节中的产品表会导致一个如下的错误消息，因为有订单表依赖于产品表：

```
DROP TABLE products;

ERROR:  cannot drop table products because other objects depend on
 it
DETAIL:  constraint orders_product_no_fkey on table orders depends
 on table products
HINT:  Use DROP ... CASCADE to drop the dependent objects too.
```

该错误消息包含了一个有用的提示：如果我们不想一个一个去删除所有的依赖对象，
我们可以执行：

```
DROP TABLE products CASCADE;
```

这样所有的依赖对象将被移除。在这种情况下，订单表不会被移除，
但是它的外键约束会被移除。（如果希望检查`DROP ... CASCADE`会干什么，运行不带`CASCADE`的`DROP`并阅读`DETAIL`输出。）

PostgreSQL中所有DROP命令都支持指定CASCADE。当然，
可能出现的依赖关系形态会随着对象类型不同而变化。你也可以写RESTRICT
来代替CASCADE，从而得到默认行为，也就是阻止删除任何被其他对象依赖的对象。

> **注意**
>
> 根据 SQL 标准，DROP命令必须指定RESTRICT或CASCADE。实际上，没有数据库系统强制执行这条规则，但不同系统的默认行为是RESTRICT还是CASCADE，各有不同。

对于用户定义函数，PostgreSQL会跟踪与函数外部可见属性有关的依赖，
例如参数和结果类型，但不会跟踪那些只有检查函数体才能得知的依赖。例如，
考虑以下情况：

```
CREATE TYPE rainbow AS ENUM ('red', 'orange', 'yellow',
                             'green', 'blue', 'purple');

CREATE TABLE my_colors (color rainbow, note text);

CREATE FUNCTION get_color_note (rainbow) RETURNS text AS
  'SELECT note FROM my_colors WHERE color = $1'
  LANGUAGE SQL;
```

（关于 SQL 语言函数的说明，请参阅第 35.4 节。）PostgreSQL会知道get_color_note函数依赖于rainbow类型：删除该类型将迫使系统删除函数，
因为它的参数类型将不再有定义。但是，PostgreSQL不会认为get_color_note依赖于my_colors表，因此不会在删除该表时删除函数。这种做法有缺点，也有好处。即使表不存在，
函数在某种意义上仍然有效，尽管执行它会报错；创建一个同名新表后，
函数就可以重新工作。

第 6 章 数据操纵

前一章讨论了如何创建用于保存数据的表和其他结构。现在该把数据填入表中了。
本章介绍如何插入、更新和删除表数据。
下一章则终于会讲到如何从数据库中取回那些久寻不得的数据。

6.1. 插入数据

创建表时，其中不包含任何数据。在数据库真正派上用场之前，首先要做的就是插入数据。
从概念上说，数据一次插入一行。当然也可以插入多行，但不能插入少于一行的数据。
即使只知道部分列值，也必须创建完整的一行。

要创建新行，可使用INSERT命令。该命令需要提供表名和列值。例如，考虑第 5 章中的 products 表：

```
CREATE TABLE products (
    product_no integer,
    name text,
    price numeric
);
```

插入一行的示例命令如下：

```
INSERT INTO products VALUES (1, 'Cheese', 9.99);
```

数据值按列在表中出现的顺序列出，并以逗号分隔。通常，数据值会是字面量（常量），
但也允许使用标量表达式。

上述语法的缺点在于你必须知道表中列的顺序。为了避免这一点，你也可以显式列出列。
例如，下面两条命令的效果都与上面的命令相同：

```
INSERT INTO products (product_no, name, price) VALUES (1, 'Cheese',
 9.99);
INSERT INTO products (name, price, product_no) VALUES ('Cheese', 9.
99, 1);
```

许多用户认为，总是列出列名是一种好习惯。

如果你没有所有列的值，可以省略其中一些列。在这种情况下，
这些列将填入各自的默认值。例如：

```
INSERT INTO products (product_no, name) VALUES (1, 'Cheese');
INSERT INTO products VALUES (1, 'Cheese');
```

第二种形式是PostgreSQL 的一个扩展。它从左到右用给出的值填充各列，其余列则取默认值。

为求清晰，你也可以显式请求默认值，既可以针对单个列，也可以针对整行：

```
INSERT INTO products (product_no, name, price) VALUES (1, 'Cheese',
 DEFAULT);
INSERT INTO products DEFAULT VALUES;
```

你可以在一条命令中插入多行：

```
INSERT INTO products (product_no, name, price) VALUES
    (1, 'Cheese', 9.99),
    (2, 'Bread', 1.99),
```

```
    (3, 'Milk', 2.99);
```

也可以插入一个查询的结果（它可能返回零行、一行或多行）：

```
INSERT INTO products (product_no, name, price)
  SELECT product_no, name, price FROM new_products
    WHERE release_date = 'today';
```

这使你能够利用 SQL 查询机制的全部能力（见第 7 章）来计算要插入的行。

> **提示**
>
> 当需要一次插入大量数据时，可以考虑使用COPY命令。
> 它不像INSERT命令那样灵活，但效率更高。关于如何改善批量装载性能的更多信息，请参阅第 14.4 节。

6.2. 更新数据

修改数据库中已有的数据称为更新。你可以更新单独的行、表中的所有行，
或者所有行中的一个子集。每一列都可以单独更新；其他列不受影响。

要更新现有的行，可使用UPDATE命令。这需要三部分信息：

1. 要更新的表名和列名
2. 列的新值
3. 要更新哪一行（或哪些行）

回想一下第 5 章中的内容，SQL 通常并不为行提供唯一标识符。因此，并不总能直接指定要更新哪一行。取而代之的是，你要指定一行必须满足哪些条件才会被更新。
只有当表中存在主键时（不论你是否声明了它），
才能通过选择一个匹配主键的条件来可靠地定位单独的行。
图形化数据库访问工具正是依赖这一事实来允许你逐行更新。

例如，这条命令把所有价格为 5 的产品的价格更新为 10：

```
UPDATE products SET price = 10 WHERE price = 5;
```

这可能更新零行、一行或多行。尝试执行一个匹配不到任何行的更新并不算错误。

让我们仔细看看这条命令。首先是关键字 `UPDATE`，后面跟着表名。和平常一样，表名可以用模式限定；否则就在搜索路径中查找。接着是关键字 `SET`，后面跟着列名、一个等号以及新的列值。新列值可以是任意标量表达式，而不仅仅是常量。例如，
如果你想把所有产品的价格提高 10%，可以使用：

```
UPDATE products SET price = price * 1.10;
```

如你所见，用于新值的表达式也可以引用该行中的现有值。我们还省略了`WHERE`子句。
如果省略它，就意味着更新表中的所有行。如果给出了它，则只有匹配 `WHERE` 条件的那些行才会被更新。请注意，`SET` 子句中的等号表示赋值，而 `WHERE` 子句中的等号表示比较，但这不会造成歧义。当然，`WHERE` 条件不必是等值测试。还有许多其他操作符可用（见第 9 章）。不过，该表达式必须计算出一个布尔结果。

你可以在一条`UPDATE`命令中更新多个列，只需在 `SET`子句中列出多个赋值。例如：

```
UPDATE mytable SET a = 5, b = 3, c = 1 WHERE a > 0;
```

6.3. 删除数据

到目前为止，我们已经解释了如何向表中添加数据以及如何修改数据。
接下来要讨论的是如何删除不再需要的数据。正如添加数据只能按整行进行一样，
从表中删除数据也只能按整行进行。前一节已经说明，SQL 不提供直接寻址单独行的方法。
因此，只能通过指定待删除行必须满足的条件来删除它们。如果表中有主键，
那么你可以指定精确的一行；但你也可以删除匹配某个条件的一组行，
或者一次删除表中的所有行。

可以使用DELETE命令删除行；其语法与UPDATE命令非常相似。例如，要删除 products 表中所有价格为 10 的行，可以使用：

```
DELETE FROM products WHERE price = 10;
```

如果你只是写：

```
DELETE FROM products;
```

那么表中的所有行都会被删除！程序员须当心。

6.4. 从被修改的行中返回数据

有时候，在修改行的同时取得这些行中的数据会很有用。INSERT、UPDATE、DELETE 命令都支持可选的 RETURNING 子句。使用 RETURNING 可以避免额外执行一次数据库查询来收集这些数据；当原本难以可靠地识别被修改的行时，这一点尤其有价值。

RETURNING 子句允许的内容与 SELECT 命令的输出列表相同（见第 7.3 节）。它可以包含命令目标表的列名，也可以包含使用这些列的值表达式。一个常见的简写是 RETURNING *，它按顺序选择目标表的全部列。

在 INSERT 中，RETURNING可用的数据是插入后的行。对于简单的插入，这并没有太大用处，因为它只是重复客户端提供的数据。但在依赖计算得出的默认值时，它就会非常方便。例如，当使用 serial 列提供唯一标识符时，RETURNING 可以返回分配给新行的 ID：

```
CREATE TABLE users (firstname text, lastname text, id serial
 primary key);

INSERT INTO users (firstname, lastname) VALUES ('Joe', 'Cool')
 RETURNING id;
```

RETURNING 子句配合 INSERT ... SELECT 也非常有用。

在 UPDATE 中，RETURNING可用的数据是被修改行的新内容。例如：

```
UPDATE products SET price = price * 1.10
  WHERE price <= 99.99
  RETURNING name, price AS new_price;
```

在 DELETE 中，RETURNING可用的数据是被删除行的内容。例如：

```
DELETE FROM products
  WHERE obsoletion_date = 'today'
  RETURNING *;
```

如果目标表上有触发器（第 36 章），RETURNING可取得的数据就是触发器修改后的行。因此，检查由触发器计算的列，也是RETURNING的另一个常见用途。

第 7 章 查询

前面的章节解释了如何创建表、如何用数据填充它们，以及如何操纵这些数据。
现在我们终于可以讨论如何从数据库中检索数据了。

7.1. 概述

从数据库检索数据的过程或用于检索数据的命令称为查询。在 SQL 中，SELECT命令用于指定查询。SELECT命令的一般语法为：

```
[WITH with_queries] SELECT select_list FROM table_expression [sort_specification]
```

以下各节将详细描述选择列表、表表达式和排序说明。WITH查询是一项高级特性，
因此放在最后介绍。

一种简单查询的形式如下：

```
SELECT * FROM table1;
```

假设存在一个名为table1的表，则这条命令会从table1
中检索所有行以及所有用户定义列。（具体的检索方式取决于客户端应用。例如，
psql程序会在屏幕上显示一个 ASCII 形式的表格，而客户端库则会提供从查询结果中提取单个值的函数。）选择列表说明*表示表表达式所提供的全部列。
选择列表也可以只选择部分可用列，或者利用这些列进行计算。例如，如果table1
有名为a、b和c的列（还可能有其他列），那么可以写出下面的查询：

```
SELECT a, b + c FROM table1;
```

（假设b和c都是数值数据类型）。有关更多细节请参见第 7.3 节。

FROM table1是一种简单的表表达式：它只读取一个表。通常，表表达式可以是由基本表、连接和子查询组成的复杂结构。不过，你也可以完全省略表表达式，把SELECT
命令当成计算器来使用：

```
SELECT 3 * 4;
```

如果选择列表中的表达式会返回变化的结果，这种用法就更有用了。例如，
你可以这样调用函数：

```
SELECT random();
```

7.2. 表表达式

表表达式用于计算出一个表。表表达式包含一个FROM子句，后面可以根据需要跟上WHERE、GROUP BY和HAVING子句。最简单的表表达式只是引用磁盘上的一个表，即所谓的基本表；但也可以使用更复杂的表达式，以多种方式修改或组合基本表。

表表达式中可选的WHERE、GROUP BY和HAVING子句指定了一个连续转换的流水线，这些转换作用于由FROM子句派生出的表。所有这些转换都会生成一个虚拟表，
该表提供的各行会传递给选择列表，以计算查询的输出行。

7.2.1. FROM子句

此FROM Clause从逗号分隔的表引用列表所给出的一个或多个其他表推导出一个表。

```
FROM table_reference [, table_reference [, ...]]
```

表引用可以是表名（可能带模式限定），也可以是子查询、JOIN结构等推导表，或它们的复杂组合。如果FROM子句中列出了多个表引用，这些表会进行交叉连接（即形成各表行的笛卡尔积，见下文）。FROM列表的结果是一个中间虚拟表，随后可以用WHERE，GROUP BY和HAVING子句对它进行变换，最终得到整个表表达式的结果。

当表引用命名的是表继承层次中的父表时，除非在表名前加上关键字ONLY，否则该表引用不仅会产生该表中的行，还会产生其所有后代表中的行。不过，这种引用只会产生该命名表中出现的列 – 子表中新增的列会被忽略。

也可以不在表名前写ONLY，而是在表名后面写上*，显式指定要包含后代表。由于这种行为是默认行为（除非你更改了sql_inheritance配置选项的设置），写*并无必要。不过，写*或许有助于强调将会搜索额外的表。

7.2.1.1. 连接表

一个连接表是根据特定的连接类型的规则从两个其它表（真实表或生成表）中派生的表。目前支持内连接、外连接和交叉连接。一个连接表的一般语法是：

```
T1 join_type T2 [ join_condition ]
```

所有类型的连接都可以被链在一起或者嵌套：*T1*和*T2*都可以是连接表。在JOIN子句周围可以使用圆括号来控制连接顺序。如果不使用圆括号，JOIN子句会从左至右嵌套。

连接类型

交叉连接

```
T1 CROSS JOIN T2
```

对来自于*T1*和*T2*的行的每一种可能的组合（即笛卡尔积），连接表将包含这样一行：它由所有*T1*里面的列后面跟着所有*T2*里面的列构成。如果两个表分别有 N 和 M 行，连接表将有 N * M 行。

FROM *T1* CROSS JOIN *T2*等效于FROM *T1* INNER JOIN *T2* ON TRUE（见下文）。它也等效于FROM *T1*,*T2*。

> **注意**
>
> 当出现两个以上的表时，后一种等价关系并不严格成立，因为JOIN的绑定强于逗号。例如FROM *T1* CROSS JOIN *T2* INNER JOIN *T3* ON *condition*和FROM *T1*,*T2* INNER JOIN *T3* ON *condition*并不完全相同，因为第一种情况中的*condition*可以引用*T1*，而第二种情况中则不行。

限定连接

```
T1 { [INNER] | { LEFT | RIGHT | FULL } [OUTER] } JOIN T2
 ON boolean_expression
T1 { [INNER] | { LEFT | RIGHT | FULL } [OUTER] } JOIN T2 USING (
 join column list )
T1 NATURAL { [INNER] | { LEFT | RIGHT | FULL } [OUTER] } JOIN T2
```

INNER和OUTER在所有形式中都是可选的。INNER是默认值；LEFT、RIGHT和FULL表示外连接。

连接条件在ON或USING子句中指定，或者用关键字NATURAL隐含地指定。
连接条件决定来自两个源表中的哪些行是“匹配”的，这些我们将在后文详细解释。

限定连接有以下几种类型：

INNER JOIN

对于 T1 的每一行 R1，生成的连接表都有一行对应 T2 中的每一个满足和 R1 的连接条件的行。

LEFT OUTER JOIN

首先，执行一次内连接。然后，为 T1 中每一个无法在连接条件上匹配 T2 里任何一行的行返回一个连接行，该连接行中 T2 的列用空值补齐。因此，生成的连接表里为来自 T1 的每一行都至少包含一行。

RIGHT OUTER JOIN

首先，执行一次内连接。然后，为 T2 中每一个无法在连接条件上匹配 T1 里任何一行的行返回一个连接行，该连接行中 T1 的列用空值补齐。因此，生成的连接表里为来自 T2 的每一行都至少包含一行。

FULL OUTER JOIN

首先，执行一次内连接。然后，为 T1 中每一个无法在连接条件上匹配 T2 里任何一行的行返回一个连接行，该连接行中 T2 的列用空值补齐。同样，为 T2 中每一个无法在连接条件上匹配 T1 里任何一行的行返回一个连接行，该连接行中 T1 的列用空值补齐。

ON子句是最常见的连接条件的形式：它接收一个和WHERE
子句里用的一样的布尔值表达式。如果两个分别来自*T1*和*T2*的行在ON
表达式上运算的结果为真，那么它们就算是匹配的行。

USING是个缩写符号，它允许你利用特殊的情况：连接的两端都具有相同的连接列名。
它接受共享列名的一个逗号分隔列表，
并且为其中每一个共享列构造一个包含等值比较的连接条件。例如用USING (a, b)连接*T1*和*T2*会产生连接条件ON *T1*.a = *T2*.a AND *T1*.b = *T2*.b。

更进一步，JOIN USING的输出会废除冗余列：不需要把匹配上的列都打印出来，因为它们必须具有相等的值。不过JOIN ON会先产生来自*T1*的所有列，后面跟上所有来自*T2*的列；而JOIN USING会先为列出的每一个列对产生一个输出列，然后先跟上来自*T1*的剩余列，最后跟上来自*T2*的剩余列。

最后，NATURAL是USING的一种缩写形式：它会形成一个USING列表，其中包含两个输入表中共同出现的所有列名。和USING一样，这些列在输出表中只出现一次。
如果没有公共列名，NATURAL JOIN的行为就与JOIN ... ON TRUE相同，产生一个叉积连接。

注意

对于被连接的关系发生列变化的情况，USING相当安全，
因为只有列出的列才会被合并。NATURAL的风险则大得多，
因为只要任一关系的模式变更导致出现新的同名列，
连接就会把这个新列也纳入合并。

综合起来看，假设我们有表t1：

```
 num | name
-----+------
   1 | a
   2 | b
   3 | c
```

和t2:

```
 num | value
-----+-------
   1 | xxx
   3 | yyy
   5 | zzz
```

然后我们用不同的连接方式可以获得各种结果:

```
=> SELECT * FROM t1 CROSS JOIN t2;
 num | name | num | value
-----+------+-----+-------
   1 | a    |   1 | xxx
   1 | a    |   3 | yyy
   1 | a    |   5 | zzz
   2 | b    |   1 | xxx
   2 | b    |   3 | yyy
   2 | b    |   5 | zzz
   3 | c    |   1 | xxx
   3 | c    |   3 | yyy
   3 | c    |   5 | zzz
(9 rows)

=> SELECT * FROM t1 INNER JOIN t2 ON t1.num = t2.num;
 num | name | num | value
-----+------+-----+-------
   1 | a    |   1 | xxx
   3 | c    |   3 | yyy
(2 rows)

=> SELECT * FROM t1 INNER JOIN t2 USING (num);
 num | name | value
-----+------+-------
   1 | a    | xxx
   3 | c    | yyy
(2 rows)

=> SELECT * FROM t1 NATURAL INNER JOIN t2;
 num | name | value
-----+------+-------
   1 | a    | xxx
   3 | c    | yyy
(2 rows)

=> SELECT * FROM t1 LEFT JOIN t2 ON t1.num = t2.num;
 num | name | num | value
-----+------+-----+-------
   1 | a    |   1 | xxx
   2 | b    |     |
   3 | c    |   3 | yyy
(3 rows)
```

```
=> SELECT * FROM t1 LEFT JOIN t2 USING (num);
 num | name | value
-----+------+-------
   1 | a    | xxx
   2 | b    |
   3 | c    | yyy
(3 rows)

=> SELECT * FROM t1 RIGHT JOIN t2 ON t1.num = t2.num;
 num | name | num | value
-----+------+-----+-------
   1 | a    |   1 | xxx
   3 | c    |   3 | yyy
     |      |   5 | zzz
(3 rows)

=> SELECT * FROM t1 FULL JOIN t2 ON t1.num = t2.num;
 num | name | num | value
-----+------+-----+-------
   1 | a    |   1 | xxx
   2 | b    |     |
   3 | c    |   3 | yyy
     |      |   5 | zzz
(4 rows)
```

用ON指定的连接条件也可以包含与连接不直接相关的条件。
这种功能可能对某些查询很有用，但是需要我们仔细想清楚。例如：

```
=> SELECT * FROM t1 LEFT JOIN t2 ON t1.num = t2.num AND t2.value =
 'xxx';
 num | name | num | value
-----+------+-----+-------
   1 | a    |   1 | xxx
   2 | b    |     |
   3 | c    |     |
(3 rows)
```

注意把限制放在WHERE子句中会产生不同的结果：

```
=> SELECT * FROM t1 LEFT JOIN t2 ON t1.num = t2.num WHERE t2.value
 = 'xxx';
 num | name | num | value
-----+------+-----+-------
   1 | a    |   1 | xxx
(1 row)
```

这是因为放在ON子句中的约束会在连接之前处理，而放在WHERE
子句中的约束则会在连接之后处理。这对内连接无关紧要，但对外连接影响很大。

7.2.1.2. 表和列别名

你可以给表以及复杂的表引用指定一个临时名字，以便在查询的其余部分引用该派生表。
这被称为表别名。

要创建一个表别名，我们可以写：

FROM *table_reference* AS *alias*

或者

```
FROM table_reference alias
```

AS关键字只是可选的语法噪音。*alias*可以是任意标识符。

表别名的典型应用是给长表名赋予比较短的标识符，好让连接子句更易读。例如：

```
SELECT * FROM some_very_long_table_name s JOIN another_fairly_long_
name a ON s.id = a.num;
```

对于当前查询而言，
别名成为该表引用的新名称–不允许在查询的其他地方通过原名称引用该表。因此，
以下写法无效：

```
SELECT * FROM my_table AS m WHERE my_table.a > 5;    -- wrong
```

表别名主要用于简化符号，但是当把一个表连接到它自身时必须使用别名，例如：

```
SELECT * FROM people AS mother JOIN people AS child ON mother.id =
 child.mother_id;
```

此外，当表引用是子查询时也必须使用别名（参见第 7.2.1.3 节）。

圆括弧用于解决歧义。在下面的示例中，第一个语句将把别名b赋给my_table
的第二个实例，但是第二个语句把别名赋给连接的结果：

```
SELECT * FROM my_table AS a CROSS JOIN my_table AS b ...
SELECT * FROM (my_table AS a CROSS JOIN my_table) AS b ...
```

另外一种给表指定别名的形式是给表的列赋予临时名字，就像给表本身指定别名一样：

```
FROM table_reference [AS] alias ( column1 [, column2 [, ...]] )
```

如果指定的列别名比表里实际的列少，那么剩下的列就没有被重命名。
这种语法对于自连接或子查询特别有用。

如果用这些形式中的任何一种给一个JOIN子句的输出附加了一个别名，那么该别名就在JOIN
的作用下隐去了其原始的名字。例如：

```
SELECT a.* FROM my_table AS a JOIN your_table AS b ON ...
```

是合法 SQL，但是：

```
SELECT a.* FROM (my_table AS a JOIN your_table AS b ON ...) AS c
```

是不合法的：表别名a在别名c外面是看不到的。

7.2.1.3. 子查询

指定派生表的子查询必须用圆括号括起来，并且必须赋予一个表别名。（参见第 7.2.1.2 节。）例如：

```
FROM (SELECT * FROM table1) AS alias_name
```

这个示例等效于FROM table1 AS alias_name。更有趣的情况是在子查询里面有分组或聚合的时候，子查询不能被简化为一个简单的连接。

一个子查询也可以是一个VALUES列表：

```
FROM (VALUES ('anne', 'smith'), ('bob', 'jones'), ('joe', 'blow'))
     AS names(first, last)
```

同样，表别名是必须的。为VALUES列表中的列分配别名是可选的，但这是一个好习惯。更多信息可参见第 7.7 节。

7.2.1.4. 表函数

表函数是那些生成行集合的函数，这些行可以由基础类型（标量类型）组成，也可以由复合数据类型（表行）组成。它们在查询的FROM子句中用法类似于表、视图或子查询。表函数返回的列可以像表、视图或子查询的列一样，出现在SELECT、JOIN或WHERE子句中。

如果一个表函数返回基础数据类型，唯一的结果列名与该函数名相同。如果该函数返回一个复合类型，结果列会得到与该类型的各个属性相同的名称。

表函数可以在FROM子句中被赋予别名，但也可以不使用别名。如果一个函数在FROM子句中使用时没有别名，该函数名将被用作结果表的表名。

示例：

```
CREATE TABLE foo (fooid int, foosubid int, fooname text);

CREATE FUNCTION getfoo(int) RETURNS SETOF foo AS $$
    SELECT * FROM foo WHERE fooid = $1;
$$ LANGUAGE SQL;

SELECT * FROM getfoo(1) AS t1;

SELECT * FROM foo
    WHERE foosubid IN (
                        SELECT foosubid
                        FROM getfoo(foo.fooid) z
                        WHERE z.fooid = foo.fooid
                      );

CREATE VIEW vw_getfoo AS SELECT * FROM getfoo(1);

SELECT * FROM vw_getfoo;
```

有时候，定义一个能够根据调用方式返回不同列集合的表函数很有用。为支持这一点，表函数可以被声明为返回伪类型record。当此类函数在查询中使用时，必须在查询本身中指定预期的行结构，这样系统才能知道如何分析和规划该查询。考虑下面的示例：

```
SELECT *
    FROM dblink('dbname=mydb', 'SELECT proname, prosrc FROM pg_
proc')
      AS t1(proname name, prosrc text)
    WHERE proname LIKE 'bytea%';
```

dblink函数（dblink模块的一部分）执行远程查询。它被声明为返回record，因为它可能用于任意类型的查询。实际的列集必须在调用它的查询中指定，这样分析器才知道像*这样的写法应当扩展成什么。

7.2.2. WHERE子句

此WHERE Clause的语法是：

```
WHERE search_condition
```

其中，*search_condition*可以是任意值表达式（见第 4.2 节），只要其返回值的类型为boolean。

在完成FROM子句的处理之后，生成的虚拟表中的每一行都会根据搜索条件进行检查。如果条件结果为真，该行就会保留在输出表中；否则（也就是说，如果结果为假或空）就会被丢弃。搜索条件通常至少会引用FROM子句生成的表中的某一列；这并非必须，但若不引用这些列，WHERE子句通常就没什么用了。

> **注意**
>
> 内连接的连接条件既可以写在WHERE子句也可以写在JOIN子句里。例如，这些表表达式是等效的：
>
> ```
> FROM a, b WHERE a.id = b.id AND b.val > 5
> ```
>
> 和：
>
> ```
> FROM a INNER JOIN b ON (a.id = b.id) WHERE b.val > 5
> ```
>
> 或者可能还有：
>
> ```
> FROM a NATURAL JOIN b WHERE b.val > 5
> ```
>
> 选择其中哪一种主要只是风格问题。FROM子句中的JOIN语法虽然属于 SQL 标准，但在其他 SQL 数据库管理系统中可能没有那么容易移植。对于外连接则没有选择：它们必须放在FROM子句中完成。外连接的ON或USING子句与WHERE条件并不等价，因为它除了会在最终结果中移除行之外，还会为不匹配的输入行增加结果行。

下面是一些WHERE子句的例子：

```
SELECT ... FROM fdt WHERE c1 > 5

SELECT ... FROM fdt WHERE c1 IN (1, 2, 3)

SELECT ... FROM fdt WHERE c1 IN (SELECT c1 FROM t2)

SELECT ... FROM fdt WHERE c1 IN (SELECT c3 FROM t2 WHERE c2 = fdt.
c1 + 10)

SELECT ... FROM fdt WHERE c1 BETWEEN (SELECT c3 FROM t2 WHERE c2 =
 fdt.c1 + 10) AND 100

SELECT ... FROM fdt WHERE EXISTS (SELECT c1 FROM t2 WHERE c2 > fdt.
c1)
```

fdt是由FROM子句推导得到的表。不满足WHERE子句搜索条件的行会从fdt中消除。注意，这里将标量子查询用作值表达式。与其他查询一样，子查询也可以使用复杂的表表达式。还要注意子查询中对fdt的引用方式。将c1限定为fdt.c1仅在c1也是子查询推导输入表中的列名时才有必要。不过，即使不必限定，限定列名仍能使含义更清晰。本例说明了外层查询的列命名作用域如何扩展到其内部查询。

7.2.3. GROUP BY和HAVING子句

通过WHERE过滤之后，派生出来的输入表还可以使用GROUP BY子句进行分组，并通过HAVING子句排除某些分组行。

```
SELECT select_list
    FROM ...
    [WHERE ...]
```

```
    GROUP BY grouping_column_reference [, grouping_column_
reference]...
```

此GROUP BY Clause用于把表中所有列出列的值均相同的行分到同一组。列的列出顺序没有影响。其效果是把具有相同值的每一组行合并为一个分组行，由它代表组内的所有行。这样可以消除输出中的冗余，或者计算应用于各组的聚合。例如：

```
=> SELECT * FROM test1;
 x | y
---+---
 a | 3
 c | 2
 b | 5
 a | 1
(4 rows)

=> SELECT x FROM test1 GROUP BY x;
 x
---
 a
 b
 c
(3 rows)
```

在第二个查询里，我们不能写成SELECT * FROM test1 GROUP BY x，因为列y没有一个单一值可以与每个分组对应。被分组的列可以在选择列表中引用是因为它们在每个组都有单一的值。

通常，如果一个表被分了组，那么没有在GROUP BY中列出的列都不能被引用，除非在聚合表达式中被引用。一个用聚合表达式的示例是：

```
=> SELECT x, sum(y) FROM test1 GROUP BY x;
 x | sum
---+-----
 a |   4
 b |   5
 c |   2
(3 rows)
```

这里的sum是一个聚合函数，它在整个组上计算出一个单一值。
有关可用的聚合函数的更多信息可以在第 9.20 节中找到。

> **提示**
>
> 不带聚合表达式的分组实际上是在计算某一列的非重复值集合。
> 这也可以用DISTINCT子句实现（参阅第 7.3.3 节）。

这里是另外一个示例：它计算每种产品的总销售额（而不是所有产品的总销售额）：

```
SELECT product_id, p.name, (sum(s.units) * p.price) AS sales
    FROM products p LEFT JOIN sales s USING (product_id)
    GROUP BY product_id, p.name, p.price;
```

在这个示例里，列product_id、p.name和p.price必须在GROUP BY子句里，因为它们都在查询的选择列表里被引用到（但见下文）。列s.units不必在GROUP BY列表里，因为它只是在一个聚合表达式（sum(...)）里使用，它代表一种产品的销售额。对于每种产品，这个查询都返回一行关于该产品所有销售额的汇总。

如果 products 表的设计使得product_id是主键，那么在上面的示例中只按product_id分组就足够了，因为名称和价格都函数依赖于产品 ID，因此对于每个产品 ID 分组该返回哪个名称和价格值不会有歧义。

在严格的 SQL 里，GROUP BY只能对源表的列进行分组，但PostgreSQL把这个扩展为也允许GROUP BY去根据选择列表中的列分组。也允许对值表达式进行分组，而不仅是简单的列名。

如果一个表已经用GROUP BY子句分了组，而你只对其中某些组感兴趣，那么就可以使用HAVING子句。它很像WHERE子句，用于从结果中删除某些组。其语法是：

```
SELECT select_list FROM ... [WHERE ...] GROUP BY ...
 HAVING boolean_expression
```

在HAVING
子句中的表达式可以引用分组的表达式和未分组的表达式（后者必须涉及一个聚合函数）。

示例：

```
=> SELECT x, sum(y) FROM test1 GROUP BY x HAVING sum(y) > 3;
 x | sum
---+-----
 a |   4
 b |   5
(2 rows)

=> SELECT x, sum(y) FROM test1 GROUP BY x HAVING x < 'c';
 x | sum
---+-----
 a |   4
 b |   5
(2 rows)
```

再次，一个更现实的示例：

```
SELECT product_id, p.name, (sum(s.units) * (p.price - p.cost)) AS
 profit
    FROM products p LEFT JOIN sales s USING (product_id)
    WHERE s.date > CURRENT_DATE - INTERVAL '4 weeks'
    GROUP BY product_id, p.name, p.price, p.cost
    HAVING sum(p.price * s.units) > 5000;
```

在上面的示例里，WHERE
子句通过一个未分组的列来选择数据行（该表达式仅对最近四周内发生的销售为真），而HAVING子句则把输出限制为总销售收入超过 5000 的组。请注意，聚合表达式不必在查询的所有部分都完全相同。

如果一个查询包含聚合函数调用，但没有GROUP BY子句，分组仍然会发生：结果是一个单一组行（如果该单一组行随后又被HAVING消除，则可能根本没有结果行）。如果查询包含HAVING子句，即使没有任何聚合函数调用或GROUP BY子句，情况也是如此。

7.2.4. 窗口函数处理

如果查询包含任何窗口函数（见第 3.5 节、第 9.21 节和第 4.2.8 节），这些函数将在任何分组、聚合和HAVING过滤被执行之后被计算。也就是说如果查询使用了任何聚合、GROUP BY或HAVING，则窗口函数看到的行是分组行而不是来自于FROM/WHERE的原始表行。

当使用多个窗口函数时，窗口定义中具有语法等价的PARTITION BY和ORDER BY子句的所有窗口函数，都保证在数据的一次遍历中求值。因此，即使ORDER BY不能唯一确定排序顺序，它们看到的排序顺序也相同。不过，对于具有不同PARTITION BY或ORDER BY定义的函数，其求值不作保证。（在这种情况下，窗口函数的各次求值遍历之间通常需要排序步骤，而排序不保证保留那些在其ORDER BY看来等价的行的顺序。）

目前，窗口函数总是要求输入数据预先排好序，因此查询输出总会按照某个窗口函数的PARTITION BY/ORDER BY子句排序。不过，不建议依赖这一点。如果你希望确保结果按特定方式排序，请显式使用顶层的ORDER BY子句。

7.3. 选择列表

如前面几节所示，SELECT命令中的表表达式会构造出一个中间虚拟表，其中可能涉及组合表和视图、消除行、分组等操作。这个表随后交由选择列表处理。选择列表决定中间表中的哪些列会实际输出。

7.3.1. 选择列表项

最简单的选择列表是*，它会输出表表达式生成的所有列。否则，选择列表就是一个逗号分隔的值表达式列表（定义见第 4.2 节）。例如，它可以是一个列名列表：

```
SELECT a, b, c FROM ...
```

列名a、b和c要么是FROM子句中所引用表的实际列名，要么是像第 7.2.1.2 节所述为它们指定的别名。选择列表中可用的名字空间与WHERE子句相同；如果使用了分组，则与HAVING子句相同。

如果超过一个表有同样的列名，那么你还必须给出表名字，如：

```
SELECT tbl1.a, tbl2.a, tbl1.b FROM ...
```

在使用多个表时，要求一个特定表的所有列也是有用的：

```
SELECT tbl1.*, tbl2.a FROM ...
```

更多有关*table_name*.*记号的内容请参考第 8.15.5 节。

如果在选择列表中使用任意值表达式，那么从概念上说，它会在返回表中增加一个新的虚拟列。该值表达式会对每个结果行计算一次，并用该行中的值替换其中的列引用。不过，选择列表中的表达式并不一定要引用FROM子句表表达式中的列；例如，它也可以是任意常量算术表达式。

7.3.2. 列标签

选择列表中的项可以被赋予名字，以供后续处理使用，例如在ORDER BY子句中引用，或者供客户端应用显示。例如：

```
SELECT a AS value, b + c AS sum FROM ...
```

如果没有使用AS指定输出列名，系统就会分配一个默认列名。对于简单的列引用，它就是被引用列的名字；对于函数调用，它就是函数的名字；对于复杂表达式，系统则会生成一个通用名称。

此AS关键字是可选的，但前提是新列名不能与任何PostgreSQL关键字相同（见附录 C）。为了避免意外使用关键字，可以用双引号括住列名。例如，VALUE是关键字，因此下面的写法无效：

```
SELECT a value, b + c AS sum FROM ...
```

但下面的写法有效：

```
SELECT a "value", b + c AS sum FROM ...
```

为防范将来可能新增的关键字，建议始终写出AS，或者用双引号括住输出列名。

> **注意**
>
> 输出列的命名和在FROM子句里的命名是不一样的（参阅第 7.2.1.2 节）。它实际上允许你对同一个列命名两次，但是在选择列表中分配的名字是要传递下去的名字。

7.3.3. DISTINCT

在处理完选择列表之后，结果表还可以选择性地删除重复行。我们可以直接在SELECT后面写上DISTINCT关键字来指定：

```
SELECT DISTINCT select_list ...
```

（如果不用DISTINCT，则可以用ALL关键字来显式指定保留所有行这一默认行为。）

显然，如果两行至少在一个列值上不同，就认为它们是不同的。
空值在这种比较中被认为是相等的。

另外，我们还可以用任意表达式来判断什么行可以被认为是可区分的：

```
SELECT DISTINCT ON (expression [, expression ...]) select_list ...
```

这里的*expression*是一个对所有行求值的任意值表达式。
如果某一组行在这些表达式上的值都相等，那么它们就会被视为重复行，
因此只保留该组中的第一行。请注意，某一组里的“第一行”是不可预测的，
除非查询在足够多的列上进行了排序，以保证到达DISTINCT过滤器的行顺序是唯一的。
（DISTINCT ON的处理发生在ORDER BY排序之后。）

DISTINCT ON子句不是 SQL 标准的一部分，有时会被认为风格不佳，因为它的结果可能具有不确定性。通过审慎使用GROUP BY以及FROM中的子查询，可以避免使用这种结构，但它往往是最方便的替代方案。

7.4. 组合查询

两个查询的结果可以用集合操作并、交、差进行组合。语法是

```
query1 UNION [ALL] query2
query1 INTERSECT [ALL] query2
query1 EXCEPT [ALL] query2
```

*query1*和*query2*都是可以使用以上所有特性的查询。集合操作也可以嵌套和链式使用，
例如

```
query1 UNION query2 UNION query3
```

它会被当作：

```
(query1 UNION query2) UNION query3
```

来执行。

UNION有效地把*query2*的结果附加到*query1*
的结果上（不过我们不能保证这就是这些行实际被返回的顺序）。此外，
它将删除结果中所有重复的行，就像DISTINCT做的那样，除非你使用了UNION ALL。

INTERSECT返回那些同时存在于*query1*和*query2*的结果中的行，除非声明了INTERSECT ALL，否则所有重复行都被消除。

EXCEPT返回所有在*query1*结果中但不在*query2*结果中的行。
（这有时被称为两个查询的差。）同样，除非使用了EXCEPT ALL，否则重复行都会被消除。

为了计算两个查询的并、交、差，这两个查询必须是“并操作兼容的”，
也就意味着它们都返回同样数量的列，并且对应的列有兼容的数据类型，如第 10.5 节中描述的那样。

7.5. 行排序

当一个查询生成输出表之后（也就是处理完选择列表之后），还可以选择对其排序。
如果没有选择排序，行将按未指定的顺序返回。
这时的实际顺序取决于扫描和连接计划类型以及磁盘上的物理顺序，
但绝不能依赖这些因素。只有显式选择排序步骤，才能保证特定的输出顺序。

ORDER BY子句指定了排序顺序：

```
SELECT select_list
    FROM table_expression
    ORDER BY sort_expression1 [ASC | DESC] [NULLS { FIRST | LAST }]
             [, sort_expression2 [ASC | DESC] [NULLS { FIRST |
 LAST }] ...]
```

排序表达式可以是任何在查询选择列表中合法的表达式。例如：

```
SELECT a, b FROM table1 ORDER BY a + b, c;
```

当指定了多个表达式时，后面的值将用于对那些在前面值上相等的行继续排序。
每个表达式后面都可以选择性地写上ASC或DESC关键字，以指定升序或降序排序方向。ASC是默认值。升序会把较小的值排在前面，而“较小”是由<操作符定义的。类似地，降序则由>操作符定义。[1]

NULLS FIRST和NULLS LAST选项可用于决定空值在排序顺序中是位于非空值之前还是之后。默认情况下，空值排序时被认为大于任何非空值；也就是说，在DESC
顺序下默认是NULLS FIRST，否则默认是NULLS LAST。

注意顺序选项是对每一个排序列独立考虑的。例如ORDER BY x, y DESC表示ORDER BY x ASC, y DESC，而和ORDER BY x DESC, y DESC不同。

一个*sort_expression*也可以是输出列的列标签或编号，例如：

```
SELECT a + b AS sum, c FROM table1 ORDER BY sum;
SELECT a, max(b) FROM table1 GROUP BY a ORDER BY 1;
```

这两条查询都按第一个输出列排序。注意，输出列名必须单独出现，
不能用于表达式中–例如，下面的写法是不正确的：

```
SELECT a + b AS sum, c FROM table1 ORDER BY sum + c;          --
 wrong
```

[1] 事实上，PostgreSQL会使用该表达式数据类型的默认 B-树操作符类来决定ASC和DESC的排序顺序。按惯例，数据类型会被设置成由<和>操作符来对应这种排序顺序，不过用户定义数据类型的设计者也可以选择不同的做法。

这一限制是为了减少歧义。如果ORDER BY项是一个简单名称，既可能匹配输出列名，也可能匹配表表达式中的列，那么仍会有歧义。这种情况下使用输出列。只有在使用AS将输出列重命名为其他表列的名称时，才会造成这种混淆。

ORDER BY可以被应用于UNION、INTERSECT或EXCEPT组合的结果，但是在这种情况中它只被允许根据输出列名或编号排序，而不能根据表达式排序。

7.6. LIMIT和OFFSET

LIMIT和OFFSET允许仅检索查询其余部分所生成行的一部分：

```
SELECT select_list
    FROM table_expression
    [ ORDER BY ... ]
    [ LIMIT { number | ALL } ] [ OFFSET number ]
```

如果给出了一个限制计数，那么会返回数量不超过该限制的行（但可能更少些，因为查询本身可能生成的行数就比较少）。LIMIT ALL的效果和省略LIMIT子句一样。

OFFSET说明在开始返回结果行之前要跳过多少行。OFFSET 0的效果与省略OFFSET子句相同，而LIMIT NULL与省略LIMIT子句相同。如果OFFSET和LIMIT都出现，那么在开始返回LIMIT行之前，会先跳过OFFSET行。

如果使用LIMIT，那么用一个ORDER BY子句把结果行约束成一个唯一的顺序是很重要的。否则你就会拿到一个不可预料的该查询的行的子集。你要的可能是第十到第二十行，但以什么顺序的第十到第二十？除非你指定了ORDER BY，否则顺序是不知道的。

查询优化器在生成查询计划时会考虑LIMIT，因此如果你给定LIMIT和OFFSET，那么你很可能收到不同的规划（产生不同的行顺序）。因此，使用不同的LIMIT/OFFSET值选择查询结果的不同子集将生成不一致的结果，除非你用ORDER BY强制一个可预测的顺序。这并非bug，这是一个很自然的结果，因为 SQL 没有许诺把查询的结果按照任何特定的顺序发出，除非用了ORDER BY来约束顺序。

被OFFSET子句跳过的行仍然需要在服务器内部计算；因此，较大的OFFSET可能效率不高。

7.7. VALUES列表

VALUES提供了一种生成“常量表”的方法，可以在查询中使用，而不必在磁盘上实际创建并填充表。语法为：

```
VALUES ( expression [, ...] ) [, ...]
```

每个括号括住的表达式列表都会生成表中的一行。所有列表必须具有相同数量的元素（即表中的列数），并且各列表中对应项的数据类型必须兼容。为结果中各列分配的实际数据类型，使用与UNION相同的规则确定（见第 10.5 节）。

一个示例：

```
VALUES (1, 'one'), (2, 'two'), (3, 'three');
```

将会返回一个有两列三行的表。它实际上等效于：

```
SELECT 1 AS column1, 'one' AS column2
UNION ALL
SELECT 2, 'two'
UNION ALL
SELECT 3, 'three';
```

在默认情况下，PostgreSQL将column1、column2等名字分配给一个VALUES表的列。这些列名不是由SQL标准指定的，并且不同的数据库系统的做法也不同，因此通常最好使用表别名列表来重写这些默认的名字。

从语法上说，后面跟有表达式列表的VALUES列表被视为等同于

SELECT *select_list* FROM *table_expression*

，并且可以出现在SELECT能出现的任何地方。例如，你可以把它用作UNION的一部分，或者附加一个*sort_specification*（ORDER BY、LIMIT和/或OFFSET）给它。VALUES最常见的用途是作为一个INSERT命令的数据源，其次常见的用途是作为子查询。

更多信息请见VALUES。

7.8. WITH查询（公共表表达式）

WITH提供了一种为更大查询编写辅助语句的方法。这些语句通常被称为公共表表达式或CTE，可以视为仅在单个查询期间存在的临时表定义。WITH子句中的每个辅助语句都可以是SELECT、INSERT、UPDATE或DELETE；而WITH子句本身则附加到一个主语句上，该主语句同样可以是SELECT、INSERT、UPDATE或DELETE。

7.8.1. WITH中的SELECT

将SELECT用于WITH的基本价值是把复杂查询分解成更简单的部分。例如：

```
WITH regional_sales AS (
    SELECT region, SUM(amount) AS total_sales
    FROM orders
    GROUP BY region
), top_regions AS (
    SELECT region
    FROM regional_sales
    WHERE total_sales > (SELECT SUM(total_sales)/10 FROM regional_
sales)
)
SELECT region,
       product,
       SUM(quantity) AS product_units,
       SUM(amount) AS product_sales
FROM orders
WHERE region IN (SELECT region FROM top_regions)
GROUP BY region, product;
```

这条查询只显示销售额最高的几个地区中各产品的销售总额。WITH子句定义了两个辅助语句，名为regional_sales和top_regions，其中，regional_sales的输出用于top_regions，而top_regions的输出用于主SELECT查询。本例也可以不用WITH来编写，但那样就需要两层嵌套的子SELECT。采用这种写法更容易理解一些。

可选的RECURSIVE修饰符把WITH从单纯的语法便利变成一项能够完成标准 SQL 中原本无法完成之事的特性。使用RECURSIVE后，WITH查询可以引用自己的输出。一个非常简单的例子是以下查询，它计算 1 到 100 的整数之和：

```
WITH RECURSIVE t(n) AS (
    VALUES (1)
  UNION ALL
    SELECT n+1 FROM t WHERE n < 100
```

```
)
SELECT sum(n) FROM t;
```

递归WITH查询的一般形式始终是一个非递归项，接着是UNION（或UNION ALL），然后是一个递归项，其中只有递归项才能包含对查询自身输出的引用。这样的查询按以下方式执行：

递归查询求值

1. 计算非递归项。对UNION（但不对UNION ALL），抛弃重复行。把所有剩余的行包括在递归查询的结果中，并且也把它们放在一个临时的工作表中。
2. 只要工作表不为空，重复下列步骤：
   a. 计算递归项，用当前工作表的内容替换递归自引用。对UNION（不是UNION ALL），抛弃重复行以及那些与之前结果行重复的行。
      将剩下的所有行包括在递归查询的结果中，并且也把它们放在一个临时的中间表中。
   b. 用中间表的内容替换工作表的内容，然后清空中间表。

> **注意**
>
> 严格地说，这个过程是迭代而非递归，但RECURSIVE是 SQL 标准委员会选定的术语。

在上面的示例中，工作表在每一步都只有一行，并且在连续步骤中依次取值 1 到 100。到了第 100 步，由于WHERE子句不再产生输出，因此查询终止。

递归查询通常用于处理层次结构或树形结构数据。
以下是一个有用的例子：只给定一个表示直接包含关系的表，
查询找出产品的所有直接和间接子部件：

```
WITH RECURSIVE included_parts(sub_part, part, quantity) AS (
    SELECT sub_part, part, quantity FROM parts WHERE part = 'our_
product'
  UNION ALL
    SELECT p.sub_part, p.part, p.quantity
    FROM included_parts pr, parts p
    WHERE p.part = pr.sub_part
)
SELECT sub_part, SUM(quantity) as total_quantity
FROM included_parts
GROUP BY sub_part
```

使用递归查询时，必须确保查询的递归部分最终会不返回任何元组，否则查询会无限循环。
有时，可以用UNION代替UNION ALL，通过丢弃与先前输出行重复的行来做到这一点。不过，循环往往并不涉及完全重复的输出行：可能需要只检查一个或几个字段，
以判断是否曾到达同一点。处理这种情况的标准方法是计算一个包含已访问值的数组。
例如，考虑以下查询，它搜索表graph，使用其中的link字段：

```
WITH RECURSIVE search_graph(id, link, data, depth) AS (
    SELECT g.id, g.link, g.data, 1
    FROM graph g
  UNION ALL
    SELECT g.id, g.link, g.data, sg.depth + 1
    FROM graph g, search_graph sg
    WHERE g.id = sg.link
)
SELECT * FROM search_graph;
```

如果link关系包含循环，此查询就会循环。由于需要输出“depth”，仅把UNION ALL改为UNION并不能消除循环。我们需要判断在沿特定链接路径前进时，是否再次到达了同一行。为这个容易循环的查询添加path和cycle两列：

```
WITH RECURSIVE search_graph(id, link, data, depth, path, cycle) AS
 (
        SELECT g.id, g.link, g.data, 1,
          ARRAY[g.id],
          false
        FROM graph g
      UNION ALL
        SELECT g.id, g.link, g.data, sg.depth + 1,
          path || g.id,
          g.id = ANY(path)
        FROM graph g, search_graph sg
        WHERE g.id = sg.link AND NOT cycle
)
SELECT * FROM search_graph;
```

除了防止循环，数组值本身也常常很有用，因为它表示到达某一行所经过的“路径”。

在需要检查多个字段以识别循环的一般情况下，使用行数组。例如，如果需要比较字段f1和f2：

```
WITH RECURSIVE search_graph(id, link, data, depth, path, cycle) AS
 (
        SELECT g.id, g.link, g.data, 1,
          ARRAY[ROW(g.f1, g.f2)],
          false
        FROM graph g
      UNION ALL
        SELECT g.id, g.link, g.data, sg.depth + 1,
          path || ROW(g.f1, g.f2),
          ROW(g.f1, g.f2) = ANY(path)
        FROM graph g, search_graph sg
        WHERE g.id = sg.link AND NOT cycle
)
SELECT * FROM search_graph;
```

提示

在通常只有一个字段需要检查即可识别环的情况下，可以省略ROW()语法。这样就能使用简单数组而不是复合类型数组，从而提高效率。

提示

递归查询求值算法按广度优先搜索顺序产生输出。可以让外层查询按这样构造的“路径”列进行ORDER BY排序，以深度优先搜索顺序显示结果。

当不确定查询是否会循环时，一个有用的测试技巧是在父查询中加入LIMIT。例如，下面的查询将会无限循环，除非加上LIMIT：

```
WITH RECURSIVE t(n) AS (
    SELECT 1
```

```
  UNION ALL
    SELECT n+1 FROM t
)
SELECT n FROM t LIMIT 100;
```

这一技巧之所以有效，是因为PostgreSQL的实现只会对WITH查询中父查询实际取出的那些行求值。不推荐在生产中使用这一技巧，因为其他系统可能有不同的行为。另外，如果让外层查询对递归查询结果排序或将其与其他表连接，这一技巧通常就不起作用，因为在这种情况下，外层查询通常还是会尝试取出WITH查询的全部输出。

WITH查询的一个有用特性是，在父查询每次执行时，它只求值一次，即使父查询或同级WITH查询对它有多次引用也是如此。因此，在多处需要的昂贵计算可以放在一个WITH查询中，以避免重复工作。另一个用途是防止带副作用的函数被意外求值多次。不过，另一方面，与普通子查询相比，优化器更难把父查询中的限制下推到WITH查询中。WITH查询通常会按书写的样子求值，不会抑制父查询稍后可能丢弃的行。（不过，如上所述，如果对该查询的引用只需要有限数量的行，求值可能会提前停止。）

上面的示例仅显示了WITH与SELECT一起使用，但它可以以相同的方式附加到INSERT、UPDATE或DELETE。在每种情况下，它都等效于提供了可在主命令中引用的临时表。

7.8.2. WITH中的数据修改语句

你可以在WITH中使用数据修改语句（INSERT、UPDATE或DELETE）。这样就可以在同一查询中执行多个不同的操作。例如：

```
WITH moved_rows AS (
    DELETE FROM products
    WHERE
        "date" >= '2010-10-01' AND
        "date" < '2010-11-01'
    RETURNING *
)
INSERT INTO products_log
SELECT * FROM moved_rows;
```

这个查询实际上将行从products移动到 products_log。在WITH中的DELETE 从products中删除指定的行，通过其RETURNING子句返回它们的内容；然后主查询读取该输出并将其插入到 products_log中。

上述示例中有一个细节值得注意：WITH子句附加在INSERT上，而不是附加在INSERT内部的子SELECT上。这是必要的，因为数据修改语句只允许出现在附着于顶层语句的WITH子句中。不过，普通WITH的可见性规则仍然适用，因此仍然可以在该子SELECT中引用WITH语句的输出。

如上面的例子所示，WITH中的数据修改语句通常带有RETURNING子句（见第 6.4 节）。构成其他查询可引用临时表的，是RETURNING子句的输出，而不是数据修改语句的目标表。如果WITH中的数据修改语句缺少RETURNING子句，那么它就不会形成临时表，也无法被查询的其他部分引用。尽管如此，这样的语句仍然会被执行。下面是一个并不特别有用的示例：

```
WITH t AS (
    DELETE FROM foo
)
DELETE FROM bar;
```

这个示例将从表foo和bar中移除所有行。被报告给客户端的受影响行的数目可能只包括从bar中移除的行。

数据修改语句不允许递归自引用。有时，可以通过引用递归WITH的输出来绕过这一限制，例如：

```
WITH RECURSIVE included_parts(sub_part, part) AS (
    SELECT sub_part, part FROM parts WHERE part = 'our_product'
  UNION ALL
    SELECT p.sub_part, p.part
    FROM included_parts pr, parts p
    WHERE p.part = pr.sub_part
)
DELETE FROM parts
  WHERE part IN (SELECT part FROM included_parts);
```

此查询会删除某产品的所有直接和间接子部件。

WITH中的数据修改语句只会执行一次，并且总会执行到完成，而不管主查询是否读取了它们的全部输出，甚至是否读取了任何输出。注意这与WITH中SELECT的规则不同：正如前一小节所述，SELECT只会执行到主查询实际需要其输出为止。

WITH中的子语句彼此之间以及与主查询是并发执行的。因此，在WITH中使用数据修改语句时，实际更新发生的顺序是不可预知的。所有语句都使用同一个快照执行（参见第 13 章），因此它们无法“看见”彼此对目标表产生的影响。这减轻了实际行更新顺序不可预知所带来的影响，也意味着RETURNING数据是在不同WITH子语句与主查询之间传递更改的唯一方式。例如，在

```
WITH t AS (
    UPDATE products SET price = price * 1.05
    RETURNING *
)
SELECT * FROM products;
```

外层SELECT返回的将是UPDATE操作发生之前的原始价格，而在

```
WITH t AS (
    UPDATE products SET price = price * 1.05
    RETURNING *
)
SELECT * FROM t;
```

外层SELECT将返回更新后的数据。

在单个语句中试图更新同一行两次是不受支持的。只会发生一次修改，但很难（有时根本无法）可靠地预测究竟是哪一次。这同样适用于删除在同一语句中已经被更新过的行：只有更新会生效。因此，通常应避免在一个语句中两次修改同一行。特别要避免编写那些可能影响主语句或同级子语句所修改行的WITH子语句，因为这类语句的效果将不可预测。

当前，用作WITH中数据修改语句目标的任何表不能有条件规则、ALSO规则或扩展到多个语句的INSTEAD规则。

第 8 章 数据类型

PostgreSQL 提供了丰富的原生数据类型。用户可以使用CREATE TYPE命令向 PostgreSQL添加新类型。

表 8.1展示了所有内置的通用数据类型。“别名”列中列出的多数可选名称，都是 PostgreSQL出于历史原因在内部使用的名称。此外，还有一些内部使用或已废弃的类型也可用，但未在此列出。

表 8.1. 数据类型

名字	别名	描述
bigint	int8	有符号的8字节整数
bigserial	serial8	自动递增的8字节整数
bit [(*n*)]		定长位串
bit varying [(*n*)]	varbit	变长位串
boolean	bool	逻辑布尔值（真/假）
box		平面上的矩形框
bytea		二进制数据（“字节数组”）
character [(*n*)]	char [(*n*)]	定长字符串
character varying [(*n*)]	varchar [(*n*)]	变长字符串
cidr		IPv4或IPv6网络地址
circle		平面上的圆
date		日历日期（年、月、日）
double precision	float8	双精度浮点数（8 字节）
inet		IPv4或IPv6主机地址
integer	int, int4	有符号4字节整数
interval [*fields*] [(*p*)]		时间段
json		文本 JSON 数据
line		平面上的无限长的线
lseg		平面上的线段
macaddr		MAC（Media Access Control）地址
money		货币数量
numeric [(*p*, *s*)]	decimal [(*p*, *s*)]	可选择精度的精确数字
path		平面上的几何路径
point		平面上的几何点
polygon		平面上的封闭几何路径
real	float4	单精度浮点数（4字节）
smallint	int2	有符号2字节整数
smallserial	serial2	自动递增的2字节整数
serial	serial4	自动递增的4字节整数
text		变长字符串
time [(*p*)] [without time zone]		一天中的时间（无时区）

名字	别名	描述
time [(*p*)] with time zone	timetz	一天中的时间，包括时区
timestamp [(*p*)] [without time zone]		日期和时间（无时区）
timestamp [(*p*)] with time zone	timestamptz	日期和时间，包括时区
tsquery		文本搜索查询
tsvector		文本搜索文档
txid_snapshot		用户级事务 ID 快照
uuid		通用唯一标识符
xml		XML 数据

兼容性

下列类型（或它们的这些拼写形式）由SQL规定：bigint、bit、bit varying、boolean、char、character varying、character、varchar、date、double precision、integer、interval、numeric、decimal、real、smallint、time（有时区或无时区）、timestamp（有时区或无时区）、xml。

每种数据类型都有一种由其输入和输出函数决定的外部表示。
许多内置类型的外部格式都很直观。不过，也有一些类型是 PostgreSQL所特有的，例如几何路径；还有一些类型可能存在多种可能的格式，例如日期/时间类型。
有些输入和输出函数并不可逆，也就是说，
输出函数的结果与原始输入相比可能会丢失精度。

8.1. 数字类型

数字类型包括 2、4、8 字节的整数，4、8 字节的浮点数，以及可选精度的小数。表 8.2列出了所有可用类型。

表 8.2. 数字类型

名字	存储尺寸	描述	范围
smallint	2字节	小范围整数	-32768 到 +32767
integer	4字节	整数的典型选择	-2147483648 到 +2147483647
bigint	8字节	大范围整数	-9223372036854775808 到 9223372036854775807
decimal	可变	用户指定精度，精确	最高小数点前131072位，以及小数点后16383位
numeric	可变	用户指定精度，精确	最高小数点前131072位，以及小数点后16383位
real	4字节	可变精度，不精确	6位十进制精度
double precision	8字节	可变精度，不精确	15位十进制精度
smallserial	2字节	自动递增的小整数	1到32767
serial	4字节	自动递增的整数	1到2147483647
bigserial	8字节	自动递增的大整数	1到 9223372036854775807

数字类型常量的语法在第 4.1.2 节里描述。数字类型有一整套对应的算术操作符和函数。相关信息请参考第 9 章。下面的几节详细描述这些类型。

8.1.1. 整数类型

类型 smallint、integer 和 bigint 用于存储不同范围的整数，也就是没有小数部分的数。试图存储超出允许范围的值会导致错误。

常用的类型是integer，因为它提供了在范围、存储空间和性能之间的最佳平衡。
一般只有在磁盘空间紧张的时候才使用 smallint类型。只有当integer类型的范围不够时才应使用bigint类型，因为integer类型的速度绝对更快。

在非常精简的操作系统上，bigint类型可能无法正确工作，
因为它依赖编译器对八字节整数的支持。在这样的机器上，bigint的行为与integer相同，
但仍然占用八字节的存储空间。（我们不知道任何现代平台存在这种情况。）

SQL只规定了整数类型integer（或int）、smallint和bigint。类型int2、int4
和int8都是扩展，也在某些其他SQL数据库系统中使用。

8.1.2. 任意精度数值

类型numeric可以存储非常多位的数字并精确地执行计算。
我们特别建议将它用于货币金额和其它要求精确的数量。不过，numeric
类型上的算术运算比整数类型或者下一节描述的浮点数类型要慢很多。

下面使用如下术语：numeric的精度是整个数中有效数字的总数，
即小数点两侧的数字位数之和。numeric的小数位数是小数部分中十进制数字的数量，
即小数点右侧的位数。因此，数值 23.5141 的精度为 6，小数位数为 4。整数可以视为小数位数为零。

可以配置numeric列的最大精度和最大小数位数。要声明numeric类型的列，
使用以下语法：

NUMERIC(*precision*, *scale*)

精度必须为正数，小数位数必须为零或正数。也可以使用：

NUMERIC(*precision*)

这会选择小数位数为 0。指定：

NUMERIC

而不指定任何精度或小数位数，会创建一个可存储任意精度和小数位数数值的列，
直到达到实现对精度的限制。这种列不会把输入值强制转换为某个特定的小数位数，
而声明了小数位数的numeric列会将输入值强制转换为该小数位数。
（SQL标准要求默认小数位数为 0，即强制转换为整数精度。我们认为这没有多少用处。如果你关心可移植性，请始终显式指定精度和小数位数。）

> **注意**
>
> 在类型声明中显式指定精度时，所允许的最大值为 1000；未指定精度的NUMERIC受表 8.2中所述限制的约束。

如果要存储的值的小数位数大于该列声明的小数位数，
系统会把该值舍入到指定的小数位数。然后，
如果小数点左侧的位数超过了声明的精度减去声明的小数位数，就会报错。

数值在物理上存储时不会保留多余的前导零或尾随零。因此，
列上声明的精度和小数位数只是最大值，而不是固定分配的空间（从这个意义上说，numeric更像varchar(*n*)，而不像char(*n*)）。实际存储需求是每四个十进制数字组占两个字节，再加上 3 到 8 字节的开销。

除了普通数值外，numeric类型还允许特殊值NaN，表示“非数”。对NaN
的任何运算都会产生另一个NaN。在 SQL 命令中将该值写为常量时，必须加上引号，例如UPDATE table SET x = 'NaN'。输入时，字符串NaN的识别不区分大小写。

> **注意**
>
> 在大多数“非数字”概念的实现中，NaN 都被认为不等于任何其他数值（包括 NaN 本身）。为了让 numeric 值能够排序并用于基于树的索引，PostgreSQL 将 NaN 值视为彼此相等，并且大于所有非 NaN 值。

类型decimal和numeric是等效的。两种类型都是SQL标准的一部分。

8.1.3. 浮点类型

数据类型real和double precision是不精确的、变精度的数字类型。实际上，在底层处理器、操作系统和编译器支持的范围内，这两种类型通常实现了IEEE 754 二进制浮点算术标准（分别对应单精度和双精度）。

所谓不精确，是指某些值无法被精确转换为内部格式，只能以近似值存储，
因此存储并再取出一个值时，可能会看到轻微差异。
如何处理这类误差以及它们在计算中如何传播，是数学和计算机科学中的一个独立领域，
这里不再展开，只强调以下几点：

- 如果你要求准确的存储和计算（例如计算货币金额），应使用numeric类型。
- 如果你想用这些类型进行任何重要的复杂计算，尤其是依赖边界情形（无穷大、下溢）特定行为的计算，那么应该仔细评估其实现。
- 比较两个浮点值是否相等，未必总能得到符合预期的结果。

在大多数平台上，real类型的范围至少为 1E-37 到 1E+37，精度至少为 6 位十进制数字。double precision类型的范围通常约为 1E-307 到 1E+308，精度至少为 15 位数字。过大或过小的值会导致错误。如果输入数字的精度过高，可能会发生舍入。过于接近零且无法表示为与零不同的数值，会导致下溢错误。

> **注意**
>
> extra_float_digits设置控制浮点值转换为文本输出时所包含的额外有效数字位数。使用默认值0时，输出在 PostgreSQL 支持的每个平台上都相同。增大该值会使输出更精确地表示所存储的值，但可能不具有可移植性。

除了普通数值外，浮点类型还有几个特殊值：

```
Infinity
-Infinity
NaN
```

它们分别表示 IEEE 754 的特殊值“无穷大”，“负无穷大”和“非数字”。（在浮点运算不遵循 IEEE 754 的机器上，这些值可能无法按预期工作。）在 SQL 命令中将这些值写为常量时，必须加上引号，例如UPDATE table SET x = 'Infinity'。输入时，这些字符串的识别不区分大小写。

> **注意**
>
> IEEE 754 规定，NaN 不应与任何其他浮点值（包括NaN）相等。为了允许浮点值被排序，并可用于基于树的索引，PostgreSQL 将NaN视为彼此相等，并且大于所有非 NaN 值。

PostgreSQL 也支持 SQL 标准记法 float 和 float(*p*) 来指定不精确数值类型。这里，*p* 指定可接受的最小二进制精度位数。PostgreSQL 将 float(1) 到 float(24) 视为选择 real 类型，而 float(25) 到 float(53) 则选择 double precision。*p* 超出允许范围会报错。未指定精度的 float 视为 double precision。

> **注意**
>
> 在PostgreSQL 7.4 之前，float(*p*)中的精度被理解为这么多位十进制数字。这一点已被修正，以符合 SQL 标准，即精度以二进制位来度量。假定real和double precision的尾数分别恰好有 24 位和 53 位，对于 IEEE 标准浮点实现而言是正确的。在非 IEEE 平台上可能会有少许偏差，但为简单起见，所有平台都使用相同的*p*取值范围。

8.1.4. serial 类型

smallserial、serial 和 bigserial 并不是真正的数据类型，它们只是为了创建唯一标识符列而提供的记法便利（类似于其他一些数据库支持的 AUTO_INCREMENT 属性）。在当前实现中，下列语句：

```
CREATE TABLE tablename (
    colname SERIAL
);
```

等价于以下语句：

```
CREATE SEQUENCE tablename_colname_seq;
CREATE TABLE tablename (
    colname integer NOT NULL DEFAULT nextval('tablename_colname_
seq')
);
ALTER SEQUENCE tablename_colname_seq OWNED BY tablename.colname;
```

这样就创建了一个整数列，并把它的默认值设置为从序列生成器中取值。同时还会加上NOT NULL约束，以确保不能插入空值。（在大多数情况下，你可能还会希望再加上UNIQUE 或PRIMARY KEY约束，以防意外插入重复值，但这不会自动发生。）最后，该序列会被标记为“属于”该列，这样当列或表被删除时，序列也会随之删除。

> **注意**
>
> 因为 smallserial、serial 和 bigserial 是用序列实现的，所以即使没有删除任何行，列中出现的值序列也可能存在"空洞"或缺口。
> 即使包含该值的行从未成功插入表中，从序列分配出去的值仍会被视为"已使用"。
> 例如，如果插入事务回滚，就会发生这种情况。详细信息见第 9.16 节中的 nextval()。

要向 serial 列插入序列中的下一个值，应指定让serial列使用其默认值。这既可以通过在 INSERT 语句的列表中省略该列来实现，也可以通过使用 DEFAULT 关键字来实现。

类型名 serial 和 serial4 是等价的：二者都会创建 integer 列。类型名 bigserial 和 serial8 的工作方式相同，只是它们创建的是 bigint 列。如果预计表在其生命周期内会使用超过 2^{31} 个标识符，就应使用 bigserial。类型名 smallserial 和 serial2 也同理，只是它们创建的是 smallint 列。

为 serial 列创建的序列会在其所属列被删除时自动删除。你也可以在不删除该列的情况下删除该序列，但这会强制移除该列的默认值表达式。

> **注意**
>
> 在PostgreSQL 7.3 之前，serial 隐含 UNIQUE。现在这不再是自动的了。如果希望一个 serial 列拥有唯一约束或作为主键，现在必须像其他数据类型一样显式指定。

8.2. 货币类型

money 类型以固定的小数精度存储货币金额；参见表 8.3。小数精度由数据库的lc_monetary设置决定。表中显示的范围假定有两位小数。输入支持多种格式，包括整数字面量和浮点数字面量，以及典型的货币格式，例如 '$1,000.00'。输出通常也采用后一种形式，但会受到区域设置影响。

表 8.3. 货币类型

名字	存储尺寸	描述	范围
money	8 字节	货币额	-92233720368547758.08 到 +92233720368547758.07

由于该数据类型的输出依赖区域设置，因此在 lc_monetary 设置不同的数据库之间装入 money 数据时，可能无法正常工作。为了避免这类问题，在将转储恢复到新数据库之前，应确保新数据库的 lc_monetary 设置与原数据库相同或等效。

数据类型numeric、int和bigint的值可以转换为money。从数据类型real和double precision的转换可以通过先转换为numeric来实现，例如：

```
SELECT '12.34'::float8::numeric::money;
```

但是，我们不推荐这样做。浮点数不应该被用来处理货币，因为浮点数可能会有舍入误差。

一个money值可以在不损失精度的情况下转换为numeric。转换到其他类型可能会丢失精度，并且必须采用两个阶段完成：

```
SELECT '52093.89'::money::numeric::float8;
```

一个 money 值除以一个整数值时，会朝零方向截去小数部分。要得到舍入后的结果，可以除以一个浮点值，或者在除法前先把 money 转换为 numeric，再在除法后转换回 money（如果要避免精度丢失风险，后一种做法更好）。当一个 money 值被另一个 money 值除时，结果是 double precision（即一个纯数字，而不是金额），因为在除法中货币单位被约掉了。

8.3. 字符类型

表 8.4. 字符类型

名字	描述
character varying(*n*),varchar(*n*)	有长度限制的变长
character(*n*),char(*n*)	定长，空白填充
text	无限长度的变长

表 8.4显示了在PostgreSQL里可用的一般用途的字符类型。

SQL 定义了两种主要字符类型：character varying(*n*) 和 character(*n*)，其中 *n* 是正整数。这两种类型都可以存储长度最多为 *n* 个字符（而不是字节）的字符串。试图把更长的字符串存入这些类型的列时会报错，除非超出的字符全部都是空格，在这种情况下字符串会被截断到最大长度。（这一多少有些奇怪的例外是 SQL 标准要求的。）如果要存储的字符串短于声明长度，character 类型的值会用空格填充；character varying 类型则只是简单地存储较短的字符串。

如果显式把一个值转换为 character varying(*n*) 或 character(*n*)，那么超长的值会被截断为 *n* 个字符而不会报错。（这同样是 SQL 标准要求的。）

varchar(*n*) 和 char(*n*) 分别是 character varying(*n*) 和 character(*n*) 的别名。character 若不带长度说明则等同于 character(1)。如果 character varying 不带长度说明，则该类型接受任意长度的字符串。这是 PostgreSQL 的扩展。

此外，PostgreSQL 还提供 text 类型，用于存储任意长度的字符串。虽然 text 类型不在 SQL 标准中，但其他若干 SQL 数据库管理系统也提供了它。

character 类型的值在物理存储时会在右侧用空格填充到指定宽度 *n*，并且也会按这种形式存储和显示。但是，填充的空格在语义上被视为不重要。在比较两个 character 值时，尾随空格会被忽略，把 character 值转换为其他字符串类型之一时，尾随空格会被移除。请注意，在 character varying 和 text 值中，以及进行模式匹配（例如 LIKE 和正则表达式）时，尾随空格在语义上是有意义的。

短字符串（最长 126 字节）的存储需求是 1 个字节，再加上实际字符串本身；对 character 而言，这其中还包括填充的空格。更长的字符串则需要 4 个字节的额外开销，而不是 1 个字节。长字符串会被系统自动压缩，因此在磁盘上的实际占用可能更小。非常长的值还会被存储在后台表中，以免影响对较短列值的快速访问。无论如何，可存储的最长字符串大约为 1 GB。（数据类型声明中 *n* 允许的最大值比这还小。更改这个限制并没有意义，因为在多字节字符编码下，字符数和字节数可能差异很大。如果你想存储没有明确上限的长字符串，应使用 text 或未指定长度的 character varying，而不是随意给出一个长度上限。）

> **提示**
>
> 这三种类型之间没有性能差别，除了使用空格填充类型时会占用更多存储空间，以及在写入带长度约束的列时需要少量额外 CPU 周期来检查长度。虽然在某些其他数据库系统中，character(*n*) 可能有一定性能优势，但在 PostgreSQL 中并不存在这种优势。事实上，由于额外的存储开销，character(*n*) 通常是三种类型中最不常用的一种。在大多数情况下，应优先使用 text 或 character varying。

关于字符串常量的语法，请参见第 4.1.2.1 节；关于可用的操作符和函数，请参见第 9 章。数据库字符集决定了用于存储文本值的字符集；有关字符集支持的更多信息，请参阅第 22.3 节。

例 8.1. 使用字符类型

```
CREATE TABLE test1 (a character(4));
INSERT INTO test1 VALUES ('ok');
SELECT a, char_length(a) FROM test1; -- 1
  a   | char_length
------+-------------
 ok   |           2

CREATE TABLE test2 (b varchar(5));
```

```
INSERT INTO test2 VALUES ('ok');
INSERT INTO test2 VALUES ('good      ');
INSERT INTO test2 VALUES ('too long');
ERROR:  value too long for type character varying(5)
INSERT INTO test2 VALUES ('too long'::varchar(5)); -- explicit
 truncation
SELECT b, char_length(b) FROM test2;
   b   | char_length
-------+-------------
 ok    |           2
 good  |           5
 too l |           5
```

1 函数char_length在第 9.4 节中讨论。

PostgreSQL 中还有两种固定长度字符类型，如表 8.5所示。name 类型仅用于在内部系统目录中存储标识符，并非供一般用户使用。它的长度目前定义为 64 字节（63 个可用字符加结束符），但在 C 源代码中应使用常量 NAMEDATALEN 来引用。这个长度是在编译时设定的（因此可以针对特殊用途调整）；默认最大长度在未来版本中可能会变化。类型 "char"（注意带引号）不同于 char(1)，因为它只使用 1 个字节存储。它在系统目录中被用作一种简单的枚举类型。

表 8.5. 特殊字符类型

名字	存储尺寸	描述
"char"	1字节	单字节内部类型
name	64字节	用于对象名的内部类型

8.4. 二进制数据类型

bytea数据类型允许存储二进制串，参见表 8.6。

表 8.6. 二进制数据类型

名字	存储尺寸	描述
bytea	1或4字节外加真正的二进制串	变长二进制串

二进制串是八位组（或字节）的序列。二进制串与字符串有两点区别。首先，
二进制串明确允许存储值为零的字节以及其他“不可打印”的字节（通常指 32 到 126 范围之外的字节）。而字符串不允许零字节，
也不允许那些按数据库所选字符集编码看属于非法的其他字节值或字节序列。其次，
对二进制串的操作处理的是实际字节，而字符串的处理则取决于区域设置。简单地说，
二进制串适合存储程序员视为“裸字节”的数据，而字符串适合存储文本。

bytea类型支持两种用于输入和输出的外部格式：PostgreSQL历史的“转义”
格式和“十六进制”格式。在输入时这两种格式总是会被接受。
输出格式则取决于配置参数bytea_output，其默认值为十六进制（注意十六进制格式是在PostgreSQL 9.0中被引入的，早期的版本和某些工具无法理解它）。

SQL标准定义了一种不同的二进制串类型，叫做BLOB或者BINARY LARGE OBJECT。其输入格式和bytea不同，但是提供的函数和操作符大多一样。

8.4.1. bytea的十六进制格式

“十六进制”格式把二进制数据编码为每字节两个十六进制数字，最高有效半字节在前。
整个串以前缀 \x 开头（以便与转义格式区分）。在某些上下文中，这个开头的反斜线可能需

要通过双写进行转义。作为输入时，十六进制数字可以使用大写或小写，
并且每两个数字组成一组，组与组之间允许出现空白（但组内以及起始的 `\x` 序列中不能有空白）。十六进制格式与大量外部应用和协议兼容，并且通常比转义格式转换得更快，
因此更推荐使用。

例如：

```
SELECT '\xDEADBEEF';
```

8.4.2. bytea的转义格式

“转义”格式是 `bytea` 类型在 PostgreSQL 中的传统格式。它采用把二进制串表示为 ASCII 字符序列的方式，同时把那些不能表示为 ASCII 字符的字节转换为特殊的转义序列。如果从应用角度看，把字节当作字符表示是合理的，那么这种表示法会比较方便。
但在实际中它往往让人困惑，因为它模糊了二进制串和字符串之间的区别，
而且所选用的转义机制也比较笨拙。所以对大多数新应用来说，最好避免使用这种格式。

在转义格式中输入 `bytea` 值时，某些字节值必须转义，而所有字节值都可以转义。通常，转义一个字节的方法是把它写成三位八进制值，并在前面加一个反斜线。反斜线本身（字节值 92）也可以写成双反斜线。表 8.7展示了必须转义的字符，并给出了可用的替代转义序列。

表 8.7. bytea字面量中需要转义的字节

十进制字节值	描述	转义输入表示	示例	输出表示
0	0字节	`E'\\000'`	`SELECT E'\\000'::bytea;`	`\000`
39	单引号	`''''`或`E'\\047'`	`SELECT E'\''::bytea;`	`'`
92	反斜线	`E'\\\\'`或`E'\\134'`	`SELECT E'\\\\'::bytea;`	`\\`
0到31和127到255	“不可打印的”字节	`E'\\`*xxx*`'`（八进制值）	`SELECT E'\\001'::bytea;`	`\001`

是否必须转义这些不可打印字节，会因区域设置不同而有所差异。在某些情况下，
你可以不转义它们。请注意，表 8.7中每个示例的结果都恰好是一个字节长，尽管其输出表示有时多于一个字符。

如表 8.7所示之所以需要多个反斜线，原因在于以字符串字面量形式书写的输入串必须经过 PostgreSQL 服务器的两个解析阶段。每对反斜线中的第一个会被字符串常量解析器解释为转义字符（假定使用转义字符串语法）而被消耗，留下该对中的第二个反斜线。
（可以使用美元引用的字符串来避免这一层转义。）剩下的反斜线随后被 `bytea` 输入函数识别为某个三位八进制值的开始，或者是对另一个反斜线的转义。例如，
以字符串字面量形式传给服务器的 `E'\\001'` 在经过转义字符串解析器之后变为 `\001`。然后 `\001` 被送到 `bytea` 输入函数，在那里被转换为十进制值为 1 的单个八位组。注意，单引号字符不会被 `bytea` 特殊对待，因此它遵循字符串常量的常规规则。（另见第 4.1.2.1 节。）

`bytea` 的字节在输出时有时会被转义。一般而言，每个“不可打印”字节会被转换成等价的三位八进制值，并在前面加一个反斜线。大多数“可打印”
字节则会按其在客户端字符集中的标准表示输出。十进制值 92（反斜线）的字节在输出中会被双写。详情见表 8.8。

表 8.8. bytea输出转义字节

十进制字节值	描述	转义的输出表示	示例	输出结果
92	反斜线	`\\`	`SELECT E'\\134'::bytea;`	`\\`

十进制字节值	描述	转义的输出表示	示例	输出结果
0到31和127到255	“不可打印的”字节	`\xxx`（八进制值）	`SELECT E'\\001'::bytea;`	`\001`
32到126	“可打印的”字节	客户端字符集表示	`SELECT E'\\176'::bytea;`	~

取决于你所使用的 PostgreSQL 前端，在转义和反转义 `bytea` 串时可能还需要做额外工作。例如，如果你的接口会自动转换换行和回车，那么你可能还需要对它们进行转义。

8.5. 日期/时间类型

PostgreSQL 支持完整的一组 SQL 日期和时间类型，如表 8.9所示。这些数据类型上可用的操作见第 9.9 节。日期按格里高利历计算，即使是在该历法启用之前的年份也是如此（更多信息见第 B.4 节）。

表 8.9. 日期/时间类型

名字	存储尺寸	描述	最小值	最大值	解析度
`timestamp [ (p) ] [ without time zone ]`	8字节	包括日期和时间（无时区）	4713 BC	294276 AD	1微秒 / 14位
`timestamp [ (p) ] with time zone`	8字节	包括日期和时间，有时区	4713 BC	294276 AD	1微秒 / 14位
`date`	4字节	日期（没有一天中的时间）	4713 BC	5874897 AD	1日
`time [ (p) ] [ without time zone ]`	8字节	一天中的时间（无日期）	00:00:00	24:00:00	1微秒 / 14位
`time [ (p) ] with time zone`	12字节	仅一天中的时间，带时区	00:00:00+1459	24:00:00-1459	1微秒 / 14位
`interval [ fields ] [ (p) ]`	16字节	时间间隔	-178000000年	178000000年	1微秒 / 14位

注意

SQL 标准要求仅写 `timestamp` 时，应等效于 `timestamp without time zone`，而 PostgreSQL 也遵循这种行为。（7.3 之前的版本将其视为 `timestamp with time zone`。）`timestamptz` 被接受为 `timestamp with time zone` 的简写，这是 PostgreSQL 的扩展。

`time`、`timestamp` 和 `interval` 都接受一个可选精度值 *p*，用来指定在秒字段中保留多少位小数。默认情况下，对精度没有显式上界。对于 `timestamp` 和 `interval` 类型，*p* 的允许范围是 0 到 6。

注意

当 `timestamp` 值存储为八字节整数（目前的默认方式）时，在整个取值范围内都可以得到微秒精度。而当 `timestamp` 值改为存储为双精度浮点数（一个已废弃的编译期选项）时，有效精度上界可能小于 6。`timestamp` 值存储为相对于 2000-01-01

> 午夜之前或之后的秒数。当 timestamp 值采用浮点数实现时，对于距 2000-01-01 几年之内的日期可以达到微秒精度，但对更远的日期精度会下降。注意，使用浮点日期时间可以表示比上表所列更大的 timestamp 取值范围：从公元前 4713 年直到公元 5874897 年。
>
> 同一个编译期选项还决定 time 和 interval 值存储为浮点数还是八字节整数。在浮点存储的情况下，大的 interval 值会随着间隔长度的增大而降低精度。

对于 time 类型，采用八字节整数存储时，*p* 的允许范围是 0 到 6；采用浮点存储时，允许范围是 0 到 10。

interval 类型还有一个附加选项，可以通过写出下面这些短语之一来限制所存储字段的集合：

```
YEAR
MONTH
DAY
HOUR
MINUTE
SECOND
YEAR TO MONTH
DAY TO HOUR
DAY TO MINUTE
DAY TO SECOND
HOUR TO MINUTE
HOUR TO SECOND
MINUTE TO SECOND
```

注意，如果同时指定了 *fields* 和 *p*，那么 *fields* 必须包含 SECOND，因为精度只作用于秒。

time with time zone 类型由 SQL 标准定义，但其定义具有一些会让人怀疑其实用性的特性。在大多数情况下，date、time、timestamp without time zone 和 timestamp with time zone 的组合，就足以提供任何应用所需的完整日期/时间功能。

类型abstime和reltime是内部使用的低精度类型。不推荐在应用中使用这些类型；这些内部类型可能在将来的版本中消失。

8.5.1. 日期/时间输入

日期和时间输入几乎接受任何合理的格式，包括 ISO 8601、与 SQL 兼容的格式、传统 POSTGRES 格式等。对于某些格式，日期输入中日、月、年的顺序可能存在歧义，因此支持指定这些字段的预期顺序。将DateStyle参数设置为 MDY，表示采用月-日-年解释；设置为 DMY 表示采用日-月-年解释；而 YMD 表示采用年-月-日解释。

PostgreSQL 在处理日期/时间输入方面，比 SQL 标准要求的更灵活。有关日期/时间输入的精确解析规则，以及可识别的文本字段（包括月份、星期几和时区），请参见附录 B。

请记住，任何日期或时间字面值输入都必须像文本字符串一样用单引号括起来。
更多信息请参见第 4.1.2.7 节。SQL 要求使用下列语法：

type [(*p*)] '*value*'

其中 *p* 是可选的精度说明，给出秒字段中保留的小数位数。精度可用于 time、timestamp 和 interval 类型，允许的取值范围见上文。如果在常量声明中没有指定精度，则默认采用该字面值本身的精度。

8.5.1.1. 日期

表 8.10显示了date类型可能的输入方式。

表 8.10. 日期输入

示例	描述
1999-01-08	ISO 8601；任何模式下的1月8日（推荐格式）
January 8, 1999	在任何datestyle输入模式下都无歧义
1/8/1999	MDY模式中的1月8日；DMY模式中的8月1日
1/18/1999	MDY模式中的1月18日；在其他模式中被拒绝
01/02/03	MDY模式中的2003年1月2日；DMY模式中的2003年2月1日；YMD模式中的2001年2月3日
1999-Jan-08	任何模式下的1月8日
Jan-08-1999	任何模式下的1月8日
08-Jan-1999	任何模式下的1月8日
99-Jan-08	YMD模式中的1月8日，否则错误
08-Jan-99	1月8日，除了在YMD模式中错误
Jan-08-99	1月8日，除了在YMD模式中错误
19990108	ISO 8601；任何模式中的1999年1月8日
990108	ISO 8601；任何模式中的1999年1月8日
1999.008	年和一年中的日子
J2451187	儒略日期
January 8, 99 BC	公元前99年

8.5.1.2. 时间

一天中的时间类型包括 `time [ (p) ] without time zone` 和 `time [ (p) ] with time zone`。单独写 `time` 等效于 `time without time zone`。

这些类型的有效输入由一个一天中的时间，加上一个可选时区组成（见表 8.11和表 8.12）。如果在 `time without time zone` 的输入中指定了时区，它会被静默忽略。你也可以指定一个日期，但它同样会被忽略，除非你使用了涉及夏令时规则的时区名称，例如 `America/New_York`。在这种情况下，必须指定日期，以便确定应适用标准时间还是夏令时。
相应的时区偏移会被记录到 `time with time zone` 值中。

表 8.11. 时间输入

示例	描述
04:05:06.789	ISO 8601
04:05:06	ISO 8601
04:05	ISO 8601
040506	ISO 8601
04:05 AM	和04:05一样，AM并不影响值
04:05 PM	和16:05一样，输入的小时必须为 <= 12
04:05:06.789-8	ISO 8601
04:05:06-08:00	ISO 8601
04:05-08:00	ISO 8601
040506-08	ISO 8601
04:05:06 PST	缩写指定的时区
2003-04-12 04:05:06 America/New_York	全名指定的时区

表 8.12. 时区输入

示例	描述
PST	缩写（太平洋标准时间）
America/New_York	完整时区名
PST8PDT	POSIX风格的时区声明
-8:00	PST的ISO-8601偏移
-800	PST的ISO-8601偏移
-8	PST的ISO-8601偏移
zulu	UTC的军方缩写
z	zulu的缩写形式

关于如何指定时区，参见第 8.5.3 节。

8.5.1.3. 时间戳

时间戳类型的有效输入由一个日期和时间的串接组成，后面跟着一个可选时区，以及一个可选的 AD 或 BC（另外，AD/BC 也可以出现在时区前面，但这种顺序并不推荐）。因此：

```
1999-01-08 04:05:06
```

和：

```
1999-01-08 04:05:06 -8:00
```

都是遵循 ISO 8601 标准的有效值。另外，广泛使用的下列格式：

```
January 8 04:05:06 1999 PST
```

也被支持。

按照SQL标准，timestamp without time zone和timestamp with time zone字面量的区别在于，时间后是否有 “+” 或 “-” 符号及其后的时区偏移。因此，按照该标准，

```
TIMESTAMP '2004-10-19 10:23:54'
```

是timestamp without time zone，而

```
TIMESTAMP '2004-10-19 10:23:54+02'
```

是timestamp with time zone。PostgreSQL在确定字符串字面量的类型之前，从不检查其内容，因此会把上述两者都视为timestamp without time zone。为确保字面量被视为timestamp with time zone，应为它显式指定正确类型：

```
TIMESTAMP WITH TIME ZONE '2004-10-19 10:23:54+02'
```

若字面量已经被确定为timestamp without time zone，PostgreSQL会静默忽略任何时区标记。也就是说，所得值来自输入值中的日期/时间字段，不会根据时区调整。

对于timestamp with time zone，内部存储的值始终使用 UTC（协调世界时，传统上称为格林尼治标准时间，GMT）。对于显式指定时区的输入值，会使用该时区适当的偏移将它转换为 UTC。如果输入字符串中没有说明时区，则假定它属于系统TimeZone参数指定的时区，并使用timezone时区的偏移将它转换为 UTC。

当输出一个 timestamp with time zone 值时，它总会从 UTC 转换到当前 timezone 时区，并显示为该时区的本地时间。若要查看其他时区的时间，可以修改 timezone，或者使用 AT TIME ZONE 构造（见第 9.9.3 节）。

在 timestamp without time zone 和 timestamp with time zone 之间转换时，通常假定 timestamp without time zone 值应被解释为，或输出为，timezone 本地时间。要为该转换指定不同的时区，可以使用 AT TIME ZONE。

8.5.1.4. 特殊值

为了方便起见，PostgreSQL 支持若干特殊的日期/时间输入值，如表 8.13所示。infinity 和 -infinity 在系统内部有特殊表示，并且输出时会保持不变；其余值则只是记法上的简写，在读取时会被转换成普通日期/时间值。（特别是，now 及相关字符串在被读取后会立刻转换成某个特定的时间值。）这些值在 SQL 命令中作为常量使用时，都必须用单引号括起来。

表 8.13. 特殊日期/时间输入

输入串	合法类型	描述
epoch	date, timestamp	1970-01-01 00:00:00 +00（Unix系统时间0）
infinity	date, timestamp	晚于所有其他时间戳
-infinity	date, timestamp	早于所有其他时间戳
now	date, time, timestamp	当前事务的开始时间
today	date, timestamp	今天午夜
tomorrow	date, timestamp	明天午夜
yesterday	date, timestamp	昨天午夜
allballs	time	00:00:00.00 UTC

以下与 SQL 兼容的函数也可用于获取相应数据类型的当前时间值：CURRENT_DATE、CURRENT_TIME、CURRENT_TIMESTAMP、LOCALTIME 和 LOCALTIMESTAMP。后四个函数接受可选的亚秒级精度说明。（参见第 9.9.4 节。）请注意，这些是 SQL 函数，不会在日期/时间输入字符串中被识别。

8.5.2. 日期/时间输出

日期/时间类型的输出格式可以设为四种样式之一：ISO 8601、SQL（Ingres）、传统的 POSTGRES（Unix date 格式）或 German。默认是 ISO 格式。（SQL 标准要求使用 ISO 8601 格式。之所以存在名为 “SQL” 的输出格式，只是历史原因。）表 8.14展示了各种输出样式的示例。date 和 time 类型的输出当然只包含示例中的日期部分或时间部分。

表 8.14. 日期/时间输出风格

风格声明	描述	示例
ISO	ISO 8601, SQL标准	1997-12-17 07:37:16-08
SQL	传统样式	12/17/1997 07:37:16.00 PST
Postgres	原始样式	Wed Dec 17 07:37:16 1997 PST
German	地区样式	17.12.1997 07:37:16.00 PST

注意

ISO 8601规定使用大写字母T来分隔日期和时间。PostgreSQL在输入时接受该格式，但在输出时使用空格而不是T，如上所示。这是为了可读性和与RFC 3339以及其他一些数据库系统的一致性。

SQL和POSTGRES风格中，如果指定了 DMY 字段顺序，“日”将出现在“月”之前，否则“月”出现在“日”之前（有关该设置如何影响输入值的解释，请参考第 8.5.1 节）。表 8.15给出了示例。

表 8.15. 日期顺序习惯

datestyle设置	输入顺序	示例输出
SQL, DMY	日/月/年	17/12/1997 15:37:16.00 CET
SQL, MDY	月/日/年	12/17/1997 07:37:16.00 PST
Postgres, DMY	日/月/年	Wed 17 Dec 07:37:16 1997 PST

用户可以通过 SET datestyle 命令、postgresql.conf 配置文件中的DateStyle参数，或者服务器端或客户端上的 PGDATESTYLE 环境变量来选择日期/时间样式。

格式化函数to_char（见第 9.8 节）也可以作为一个更灵活的方式来格式化日期/时间输出。

8.5.3. 时区

时区及其约定不仅受地球几何形状影响，也受政治决定影响。世界各地的时区在 20 世纪逐渐趋于标准化，但仍然容易发生任意变化，尤其是夏令时规则方面。PostgreSQL 使用广泛采用的 IANA（Olson）时区数据库来获取历史时区规则信息。对于未来时间，则假定某个时区最新已知的规则会无限期地持续下去。

PostgreSQL 努力在典型用法上与 SQL 标准定义保持兼容。不过，SQL 标准在日期和时间类型及其能力方面存在一些奇怪的混搭。两个显而易见的问题是：

- 尽管 date 类型不能有关联的时区，time 类型却可以。但现实世界中的时区如果不同时关联日期和时间，几乎没有意义，因为偏移量可能会随着夏令时切换而在一年中发生变化。
- 默认时区被指定为相对于 UTC 的一个固定数值偏移。因此，在跨越 DST 边界做日期/时间算术时，根本无法适应夏令时变化。

为了克服这些困难，我们建议在使用时区时采用同时包含日期和时间的日期/时间类型。我们不建议使用 time with time zone 类型（尽管 PostgreSQL 出于兼容旧应用以及遵循 SQL 标准的考虑而支持它）。PostgreSQL 对于任何只包含日期或时间的类型，都会假定其使用本地时区。

在系统内部，所有带时区的日期和时间都以 UTC 存储。显示给客户端之前，它们会被转换为由TimeZone配置参数指定的本地时间。

PostgreSQL 允许使用三种不同形式来指定时区：

- 完整时区名称，例如 America/New_York。识别到的时区名称列在 pg_timezone_names 视图中（见第 45.69 节）。PostgreSQL 为此使用广泛采用的 IANA 时区数据，因此同样的时区名称通常也会被其他软件识别。
- 时区缩写，例如 PST。与完整时区名称不同，这种指定方式只定义了一个特定的 UTC 偏移，而完整时区名称还可能隐含一套夏令时切换规则。识别到的缩写列在 pg_timezone_abbrevs 视图中（见第 45.68 节）。你不能把TimeZone或log_timezone配置参数设置为时区缩写，但可以在日期/时间输入值中以及与 AT TIME ZONE 操作符一起使用缩写。
- 除了时区名称和缩写之外，PostgreSQL 还接受形如 *STDoffset* 或 *STDoffsetDST* 的 POSIX 风格时区说明，其中 *STD* 是时区缩写，*offset* 是以小时计、相对 UTC 向西的数字偏移，而 *DST* 是可选的夏令时缩写，假定它表示比给定偏移早一小时。例如，如果 EST5EDT 还不是一个已被识别的时区名称，它也会被接受，并且在功能上等同于美国东海岸时间。当给出夏令时时区名称时，假定它按照 IANA 时区数据库 posixrules 条目中所

用的相同夏令时转换日期规则使用。在标准的 PostgreSQL 安装中，posixrules 与 US/Eastern 相同，因此 POSIX 风格的时区说明遵循美国的夏令时规则。如果需要，可以通过替换 posixrules 文件来调整这一行为。

简而言之，缩写和完整名称的区别是：缩写表示特定的 UTC 偏移，而许多完整名称隐含本地夏令时规则，因此有两个可能的 UTC 偏移。例如，2014-06-04 12:00 America/New_York表示纽约当地时间的中午，在该日期使用的是东部夏令时间（UTC-4）。因此，2014-06-04 12:00 EDT指定相同的时刻。但是，2014-06-04 12:00 EST指定东部标准时间（UTC-5）的中午，而不管该日期是否名义上实行夏令时。

应当当心，POSIX 风格的时区特性可能导致静默接受伪劣输入，因为对时区缩写的合理性没有任何检查。例如，SET TIMEZONE TO FOOBAR0 也能工作，使系统实际上使用一个相当古怪的 UTC 缩写。另一个需要记住的问题是，在 POSIX 时区名中，正偏移用于格林尼治以西的地点；而在其他所有地方，PostgreSQL 都遵循 ISO-8601 约定，正的时区偏移位于格林尼治以东。

更复杂的是，一些司法辖区在不同时间使用同一时区缩写来表示不同的 UTC 偏移；例如在莫斯科，MSK 在某些年份表示 UTC+3，在另一些年份则表示 UTC+4。PostgreSQL 会按照该缩写在所给日期上的含义（或最近一次的含义）来解释这类缩写；但与上面的 EST 例子一样，这并不一定等同于该日期的当地民用时间。

无论哪种形式，时区名称及其缩写都不区分大小写。（这是对 PostgreSQL 8.2 之前版本的一项改动；在那些版本中，时区名在某些环境下区分大小写，而在另一些环境下则不区分。）

时区名称和缩写并不是硬编码在服务器中的；它们来自安装目录下 .../share/timezone/ 和 .../share/timezonesets/ 子目录中的配置文件（见第 B.3 节）。

TimeZone配置参数可以在 postgresql.conf 文件中设置，也可以通过第 18 章中说明的其他标准方式设置。另外，还有一些特殊的设置方法：

- SQL 命令 SET TIME ZONE 用于设置会话的时区。它是 SET TIMEZONE TO 的另一种写法，语法上更符合 SQL 规范。
- PGTZ 环境变量会被 libpq 客户端用于在连接到服务器时发送一条 SET TIME ZONE 命令。

8.5.4. 间隔输入

interval值可以使用下列详细语法书写：

[@] *quantity unit* [*quantity unit*...] [*direction*]

其中*quantity*是一个数字（可以带有符号）；*unit*是microsecond、millisecond、second、minute、hour、day、week、month、year、decade、century、millennium 或它们的缩写或复数形式；*direction* 可以是 ago 或为空。at 符号（@）只是可选的噪声。不同单位的数量会按适当的符号规则隐式相加。ago 会将所有字段取反。如果IntervalStyle被设置为 postgres_verbose，该语法也会用于间隔输出。

日、小时、分钟和秒的数量也可以不写显式单位标记。例如，'1 12:59:10' 会被读作 '1 day 12 hours 59 min 10 sec'。同样，年和月的组合也可以用一个连字符表示，例如 '200-10' 会被读作 '200 years 10 months'。（事实上，这些较短形式正是 SQL 标准唯一允许的形式，并且在 IntervalStyle 被设置为 sql_standard 时也用于输出。）

间隔值也可以写成 ISO 8601 时间间隔，使用标准第 4.4.3.2 节的“带标志符的格式”，或第 4.4.3.3 节的“替代格式”。带标志符的格式如下：

P *quantity unit* [*quantity unit* ...] [T [*quantity unit* ...]]

字符串必须以 P 开头，并且可以包含一个 T 来引出一天中的时间单位。可用的单位缩写见表 8.16。单位可以省略，也可以按任意顺序出现，但小于一天的单位必须出现在 T 之后。特别是，M 的含义取决于它是在 T 之前还是之后。

表 8.16. ISO 8601 间隔单位缩写

缩写	含义
Y	年
M	月（在日期部分中）
W	周
D	日
H	小时
M	分钟 (在时间部分中)
S	秒

如果使用替代格式：

```
P [ years-months-days ] [ T hours:minutes:seconds ]
```

串必须以`P`开始，并且一个`T`分隔间隔的日期和时间部分。其值按照类似于 ISO 8601日期的数字给出。

当编写带有 *fields* 说明的间隔常量，或者把字符串赋给一个定义时带有 *fields* 说明的间隔列时，未标记数量的解释方式取决于 *fields*。例如，`INTERVAL '1' YEAR` 会被读作 1 年，而 `INTERVAL '1'` 表示 1 秒。此外，位于 *fields* 说明所允许的最小字段“右侧”的字段值会被静默丢弃。例如，写 `INTERVAL '1 day 2:03:04' HOUR TO MINUTE` 会导致秒字段被丢弃，而不是日字段。

根据 SQL 标准，一个间隔值的所有字段都必须带有相同符号，因此一个前导负号会作用于所有字段；例如间隔字面值 `'-1 2:03:04'` 中的负号会同时作用于日、小时、分钟和秒。PostgreSQL 允许各字段具有不同符号，并且传统上认为文本表示中的每个字段都带有各自独立的符号，因而在这个例子里，小时、分钟和秒部分会被视为正值。如果 `IntervalStyle` 被设置为 `sql_standard`，则会把前导符号视为作用于所有字段（但前提是没有出现额外符号）；否则将采用传统的 PostgreSQL 解释。为了避免混淆，我们建议只要有任何字段为负值，就为每个字段都显式写出符号。

在内部，`interval` 值以月、日和秒存储。这是因为一个月的天数各不相同，而涉及夏令时调整时，一天可能有 23 或 25 小时。月和日字段为整数，秒字段则可以存储小数秒。由于时间间隔通常由常量字符串或`timestamp`相减创建，这种存储方式在大多数情况下都很有效。函数`justify_days`和 `justify_hours`可用于调整超出正常范围的日和小时。

在冗长输入格式中，以及在一些更紧凑输入格式的某些字段中，字段值可以带小数部分；例如`'1.5 week'`或`'01:02:03.45'`。这样的输入会被转换为适当数量的月、日和秒来存储。当转换结果出现小数的月数或天数时，小数部分会按 1 月 = 30 天、1 天 = 24 小时的换算系数计入更低阶的字段。例如，`'1.5 month'`会变成 1 个月零 15 天。输出时，只有秒会显示小数部分。

表 8.17展示了一些有效`interval`输入的示例。

表 8.17. 间隔输入

示例	描述
1-2	SQL标准格式：1年2个月
3 4:05:06	SQL标准格式：3日4小时5分钟6秒
1 year 2 months 3 days 4 hours 5 minutes 6 seconds	传统Postgres格式：1年2个月3日4小时5分钟6秒钟
P1Y2M3DT4H5M6S	“带标志符的” ISO 8601 格式：含义同上
P0001-02-03T04:05:06	ISO 8601 的“替代格式”：含义同上

8.5.5. 间隔输出

间隔类型的输出格式可以通过 `SET intervalstyle` 命令设置为以下四种风格之一：`sql_standard`、`postgres`、`postgres_verbose` 或 `iso_8601`。默认值是 `postgres` 格式。表 8.18展示了每种输出风格的示例。

如果间隔值满足 SQL 标准的限制条件（仅有年-月或仅有日-时间，且不混合正负分量），`sql_standard` 风格会生成符合 SQL 标准的间隔字面值输出。否则，输出将表现为一个标准的年-月字面值串后跟一个日-时间字面值串，并显式加上符号，以消除正负混合间隔的歧义。

`postgres` 样式的输出与 PostgreSQL 8.4 之前版本在DateStyle参数设为 `ISO` 时的输出一致。

`postgres_verbose` 样式的输出与 PostgreSQL 8.4 之前版本在 `DateStyle` 参数设为非 `ISO` 输出时的结果一致。

`iso_8601` 风格的输出符合 ISO 8601 标准 4.4.3.2 节描述的“带标志符的格式”。

表 8.18. 间隔输出风格示例

风格声明	年-月间隔	日-时间间隔	混合间隔
`sql_standard`	1-2	3 4:05:06	-1-2 +3 -4:05:06
`postgres`	1 year 2 mons	3 days 04:05:06	-1 year -2 mons +3 days -04:05:06
`postgres_verbose`	@ 1 year 2 mons	@ 3 days 4 hours 5 mins 6 secs	@ 1 year 2 mons -3 days 4 hours 5 mins 6 secs ago
`iso_8601`	P1Y2M	P3DT4H5M6S	P-1Y-2M3DT-4H-5M-6S

8.6. 布尔类型

PostgreSQL提供标准的SQL类型`boolean`，参见表 8.19。`boolean`可以有多个状态：“true（真）”、“false（假）”和第三种状态“unknown（未知）”，未知状态由SQL空值表示。

表 8.19. 布尔数据类型

名字	存储尺寸	描述
`boolean`	1字节	状态为真或假

“真”状态的有效字面值有：

```
TRUE
't'
'true'
'y'
'yes'
'on'
1
```

对于“假”状态，可以使用下列值：

```
FALSE
'f'
'false'
'n'
'no'
'off'
```

```
'0'
```

前导或尾随空白会被忽略，并且大小写不敏感。关键字 `TRUE` 和 `FALSE` 是首选的（符合 SQL 标准的）用法。

`boolean` 类型的输出函数总是产生 `t` 或 `f`，如例 8.2所示。

例 8.2. 使用boolean类型

```
CREATE TABLE test1 (a boolean, b text);
INSERT INTO test1 VALUES (TRUE, 'sic est');
INSERT INTO test1 VALUES (FALSE, 'non est');
SELECT * FROM test1;
 a |    b
---+---------
 t | sic est
 f | non est

SELECT * FROM test1 WHERE a;
 a |    b
---+---------
 t | sic est
```

8.7. 枚举类型

枚举（enum）类型是由一个静态的有序值集合构成的数据类型。
它们等效于很多编程语言所支持的`enum`类型。枚举类型的一个示例可以是一周中的各天，或者一个数据的状态值集合。

8.7.1. 枚举类型的声明

可以使用CREATE TYPE命令创建枚举类型，例如：

```
CREATE TYPE mood AS ENUM ('sad', 'ok', 'happy');
```

创建之后，该 enum 类型就可以像任何其他类型一样在表定义和函数定义中使用：

```
CREATE TYPE mood AS ENUM ('sad', 'ok', 'happy');
CREATE TABLE person (
    name text,
    current_mood mood
);
INSERT INTO person VALUES ('Moe', 'happy');
SELECT * FROM person WHERE current_mood = 'happy';
 name | current_mood
------+--------------
 Moe  | happy
(1 row)
```

8.7.2. 排序

enum 类型中各值的顺序，就是创建该类型时列出的顺序。枚举支持所有标准比较操作符以及相关聚合函数。例如：

```
INSERT INTO person VALUES ('Larry', 'sad');
INSERT INTO person VALUES ('Curly', 'ok');
SELECT * FROM person WHERE current_mood > 'sad';
 name  | current_mood
```

```
-------+--------------
 Moe   | happy
 Curly | ok
(2 rows)

SELECT * FROM person WHERE current_mood > 'sad' ORDER BY current_
mood;
 name  | current_mood
-------+--------------
 Curly | ok
 Moe   | happy
(2 rows)

SELECT name
FROM person
WHERE current_mood = (SELECT MIN(current_mood) FROM person);
 name
-------
 Larry
(1 row)
```

8.7.3. 类型安全性

每一种枚举数据类型都是独立的并且不能和其他枚举类型相比较。看这样一个示例：

```
CREATE TYPE happiness AS ENUM ('happy', 'very happy', 'ecstatic');
CREATE TABLE holidays (
    num_weeks integer,
    happiness happiness
);
INSERT INTO holidays(num_weeks,happiness) VALUES (4, 'happy');
INSERT INTO holidays(num_weeks,happiness) VALUES (6, 'very happy');
INSERT INTO holidays(num_weeks,happiness) VALUES (8, 'ecstatic');
INSERT INTO holidays(num_weeks,happiness) VALUES (2, 'sad');
ERROR:  invalid input value for enum happiness: "sad"
SELECT person.name, holidays.num_weeks FROM person, holidays
  WHERE person.current_mood = holidays.happiness;
ERROR:  operator does not exist: mood = happiness
```

如果你确实需要做这类事情，可以编写自定义操作符，或者在查询中添加显式类型转换：

```
SELECT person.name, holidays.num_weeks FROM person, holidays
  WHERE person.current_mood::text = holidays.happiness::text;
 name | num_weeks
------+-----------
 Moe  |         4
(1 row)
```

8.7.4. 实现细节

一个枚举值在磁盘上占据4个字节。一个枚举值的文本标签的长度受限于`NAMEDATALEN`设置，该设置被编译在PostgreSQL中，在标准编译下它表示最多63字节。

枚举标签是大小写敏感的，因此`'happy'`与`'HAPPY'`是不同的。
标签中的空白也是有意义的。

从内部枚举值到文本标签的转换关系被保存在系统目录`pg_enum`中。
直接查询该目录可能会有所帮助。

8.8. 几何类型

几何数据类型表示二维的空间物体。表 8.20展示了PostgreSQL中可以用的几何类型。

表 8.20. 几何类型

名字	存储尺寸	描述	表示
point	16字节	平面上的点	(x,y)
line	32字节	无限直线（未完全实现）	((x1,y1),(x2,y2))
lseg	32字节	有限线段	((x1,y1),(x2,y2))
box	32字节	矩形框	((x1,y1),(x2,y2))
path	16+16n字节	封闭路径（类似于多边形）	((x1,y1),...)
path	16+16n字节	开放路径	[(x1,y1),...]
polygon	40+16n字节	多边形（类似于封闭路径）	((x1,y1),...)
circle	24字节	圆	<(x,y),r>（中心点和半径）

我们提供了丰富的函数和操作符来进行各种几何操作，例如缩放、平移、旋转以及求交等，详见第 9.11 节。

8.8.1. 点

点是几何类型的基本二维构造块。用下面任一种语法描述point类型的值：

```
( x , y )
  x , y
```

其中*x*和*y*分别是坐标，都是浮点数。

点使用第一种语法输出。

8.8.2. 线段

线段（lseg）用成对的点来表示。lseg类型的值用下面任一种语法指定：

```
[ ( x1 , y1 ) , ( x2 , y2 ) ]
( ( x1 , y1 ) , ( x2 , y2 ) )
  ( x1 , y1 ) , ( x2 , y2 )
    x1 , y1   ,   x2 , y2
```

其中(*x1*,*y1*) 和 (*x2*,*y2*) 是线段的端点。

线段使用第一种语法输出。

8.8.3. 方框

方框用其对角的点对表示。box类型的值使用下面任一种语法指定：

```
( ( x1 , y1 ) , ( x2 , y2 ) )
  ( x1 , y1 ) , ( x2 , y2 )
    x1 , y1   ,   x2 , y2
```

其中(*x1*,*y1*) 和 (*x2*,*y2*) 是方框的对角点。

方框使用第二种语法输出。

在输入时可以提供任意两个对角，但是值将根据需要被重新排列，
并依次存储为右上角和左下角。

8.8.4. 路径

路径由一系列连接的点组成。路径可能是开放的，
也就是认为列表中第一个点和最后一个点没有被连接起来；也可能是封闭的，
这时认为第一个和最后一个点被连接起来。

`path`类型的值用下面任一种语法指定：

```
[ ( x1 , y1 ) , ... , ( xn , yn ) ]
( ( x1 , y1 ) , ... , ( xn , yn ) )
  ( x1 , y1 ) , ... , ( xn , yn )
  ( x1 , y1   , ... ,   xn , yn )
    x1 , y1   , ... ,   xn , yn
```

其中的点是组成路径的线段的端点。方括弧（`[]`）表示一个开放的路径，圆括弧（`()`）
表示一个封闭的路径。如第三种到第五种语法所示，当最外面的圆括号被省略时，
路径将被假定为封闭。

路径的输出视情况使用第一种或第二种语法。

8.8.5. 多边形

多边形由点的列表（多边形的顶点）表示。多边形与封闭路径非常相似，但存储方式不同，
并且有自己的支持例程。

`polygon`类型的值用下列任一种语法指定：

```
( ( x1 , y1 ) , ... , ( xn , yn ) )
  ( x1 , y1 ) , ... , ( xn , yn )
  ( x1 , y1   , ... ,   xn , yn )
    x1 , y1   , ... ,   xn , yn
```

其中的点是组成多边形边界的线段的端点。

多边形的输出使用第一种语法。

8.8.6. 圆

圆由一个圆心和一个半径代表。`circle`类型的值用下面任一种语法指定：

```
< ( x , y ) , r >
( ( x , y ) , r )
  ( x , y ) , r
    x , y   , r
```

其中`(x,y)`是圆心，而`r`是圆的半径。

圆的输出用第一种语法。

8.9. 网络地址类型

PostgreSQL 提供了用于存储 IPv4、IPv6 和 MAC 地址的数据类型，如表 8.21所示。使用这些数据类型来存储网络地址，比使用纯文本类型更好，因为它们提供了输入错误检查以及专门的操作符和函数（见第 9.12 节）。

表 8.21. 网络地址类型

名字	存储尺寸	描述
cidr	7或19字节	IPv4和IPv6网络
inet	7或19字节	IPv4和IPv6主机以及网络
macaddr	6字节	MAC地址

在对 inet 或 cidr 数据类型排序时，IPv4 地址总是排在 IPv6 地址之前，即使后者是封装或映射 IPv4 地址所得的 IPv6 地址，例如 ::10.2.3.4 或 ::ffff:10.4.3.2。

8.9.1. inet

inet 在一个字段中保存 IPv4 或 IPv6 主机地址，以及可选的子网信息。子网由主机地址中包含的网络地址位数（即“网络掩码”）表示。如果网络掩码为 32 且地址是 IPv4，那么该值表示的不是子网，而只是单个主机。在 IPv6 中，地址长度为 128 位，因此 128 位表示一个唯一的主机地址。请注意，如果你只想接受网络地址，应使用 cidr 类型，而不是 inet。

该类型的输入格式为 *address/y*，其中 *address* 是 IPv4 或 IPv6 地址，而 *y* 是网络掩码中的位数。如果省略 */y* 部分，则 IPv4 的网络掩码取 32，IPv6 的网络掩码取 128，因此该值表示单个主机。显示时，如果 */y* 部分表示的是单个主机，则它不会被显示出来。

8.9.2. cidr

cidr 类型保存 IPv4 或 IPv6 网络说明。输入和输出格式遵循无类别域间路由（CIDR）约定。指定网络的格式为 *address/y*，其中 *address* 是网络的 IPv4 或 IPv6 表示，而 *y* 是网络掩码中的位数。如果省略 *y*，则会按照旧式的分类网络编号系统假设来计算，但至少会大到足以覆盖输入中写出的所有八位组。若指定的网络地址在所给网络掩码右侧仍有置位比特，则会报错。

表 8.22展示了一些示例。

表 8.22. cidr类型输入示例

cidr输入	cidr输出	abbrev(cidr)
192.168.100.128/25	192.168.100.128/25	192.168.100.128/25
192.168/24	192.168.0.0/24	192.168.0/24
192.168/25	192.168.0.0/25	192.168.0.0/25
192.168.1	192.168.1.0/24	192.168.1/24
192.168	192.168.0.0/24	192.168.0/24
128.1	128.1.0.0/16	128.1/16
128	128.0.0.0/16	128.0/16
128.1.2	128.1.2.0/24	128.1.2/24
10.1.2	10.1.2.0/24	10.1.2/24
10.1	10.1.0.0/16	10.1/16
10	10.0.0.0/8	10/8
10.1.2.3/32	10.1.2.3/32	10.1.2.3/32
2001:4f8:3:ba::/64	2001:4f8:3:ba::/64	2001:4f8:3:ba::/64
2001:4f8:3:ba:2e0:81ff:fe22:d1f1/128	2001:4f8:3:ba:2e0:81ff:fe22:d1f1/128	2001:4f8:3:ba:2e0:81ff:fe22:d1f1
::ffff:1.2.3.0/120	::ffff:1.2.3.0/120	::ffff:1.2.3/120
::ffff:1.2.3.0/128	::ffff:1.2.3.0/128	::ffff:1.2.3.0/128

8.9.3. inet 与 cidr

inet 和 cidr 两种数据类型的本质区别在于：inet 接受网络掩码右侧仍含有非零位的值，而 cidr 不接受。

> **提示**
>
> 如果你不喜欢 inet 或 cidr 值的输出格式，可以试试 host、text 和 abbrev 函数。

8.9.4. macaddr

macaddr类型存储 MAC 地址，例如以太网卡硬件地址（尽管 MAC 地址还用于其它用途）。可以接受下列格式的输入：

```
'08:00:2b:01:02:03'
'08-00-2b-01-02-03'
'08002b:010203'
'08002b-010203'
'0800.2b01.0203'
'08002b010203'
```

这些示例指定的都是同一个地址。十六进制数字 a 到 f 可使用大小写任意形式。输出总是采用上面展示的第一种形式。

IEEE标准802-2001指定所示的第二种形式（带连字符）作为MAC地址的规范形式，并指定第一种形式（带冒号）作为位反转表示法，因此 08-00-2b-01-02-03 = 01:00:4D:08:04:0C。这种约定现在被广泛忽略，仅适用于过时的网络协议（如Token Ring）。PostgreSQL不提供位反转的功能，所有接受的格式都使用规范的LSB顺序。

剩下的四种输入格式不属于任何标准。

8.10. 位串类型

位串是由 1 和 0 组成的字符串。它们可用于存储或展示位掩码。SQL 中有两种位类型：bit(*n*) 和 bit varying(*n*)，其中 *n* 是正整数。

bit 类型的数据长度必须与 *n* 完全一致；试图存储更短或更长的位串都会报错。bit varying 数据则是长度上限为 *n* 的变长类型，更长的串会被拒绝。不带长度的 bit 等效于 bit(1)，而不带长度说明的 bit varying 表示长度不受限。

> **注意**
>
> 如果显式地将一个位串值转换为 bit(*n*)，它会在右侧被截断或补零，直到长度恰好为 *n* 位，且不会报错。类似地，如果显式地将一个位串值转换为 bit varying(*n*)，而其长度超过 *n* 位，则会在右侧被截断。

关于位串常量的语法信息，请参见第 4.1.2.5 节。位逻辑操作符和字符串操作函数也可用；参见第 9.6 节。

例 8.3. 使用位串类型

```
CREATE TABLE test (a BIT(3), b BIT VARYING(5));
INSERT INTO test VALUES (B'101', B'00');
INSERT INTO test VALUES (B'10', B'101');
```

```
ERROR:  bit string length 2 does not match type bit(3)
INSERT INTO test VALUES (B'10'::bit(3), B'101');
SELECT * FROM test;
  a  |  b
-----+-----
 101 | 00
 100 | 101
```

一个位串值每 8 位需要 1 个字节，再加上 5 或 8 个字节的额外开销，具体取决于串的长度。（不过，长值可能会被压缩或移到行外存储，与第 8.3 节中对字符串的说明相同。）

8.11. 文本搜索类型

PostgreSQL 提供了两种专为支持全文检索而设计的数据类型。所谓全文检索，是指在一组自然语言文档中查找最匹配某个查询的文档。`tsvector` 类型以适合文本检索的优化形式表示文档，`tsquery` 类型则表示文本查询。关于这一功能的详细解释见第 12 章；相关函数和操作符的概览见第 9.13 节。

8.11.1. tsvector

`tsvector` 值是一个排好序且互不重复的词位（lexeme）列表，这些词已经过正规化，以便把同一单词的不同变体合并起来（详见第 12 章）。排序和去重会在输入时自动完成，如下例所示：

```
SELECT 'a fat cat sat on a mat and ate a fat rat'::tsvector;
                      tsvector
-----------------------------------------------------
 'a' 'and' 'ate' 'cat' 'fat' 'mat' 'on' 'rat' 'sat'
```

若要表示包含空白或标点的词位，请用引号将它们括起来：

```
SELECT $$the lexeme '    ' contains spaces$$::tsvector;
                 tsvector
-------------------------------------------
 '    ' 'contains' 'lexeme' 'spaces' 'the'
```

（本例及下例使用美元引用的字符串常量，
以避免在字符串内部必须双写引号所造成的混淆。）嵌入的引号和反斜线必须双写：

```
SELECT $$the lexeme 'Joe''s' contains a quote$$::tsvector;
                    tsvector
--------------------------------------------------
 'Joe''s' 'a' 'contains' 'lexeme' 'quote' 'the'
```

还可以为词位附加整数形式的位置：

```
SELECT 'a:1 fat:2 cat:3 sat:4 on:5 a:6 mat:7 and:8 ate:9 a:10 fat:
11 rat:12'::tsvector;
                                  tsvector
-------------------------------------------------------------------
------------
 'a':1,6,10 'and':8 'ate':9 'cat':3 'fat':2,11 'mat':7 'on':5
 'rat':12 'sat':4
```

一个位置通常表示源词在文档中的位置。位置信息可用于邻近度排序。位置值可以位于 1 到 16383 之间；更大的数字会被静默设为 16383。同一词位的重复位置会被丢弃。

带有位置的词位还可以进一步附加一个权重标签，其值可以是 `A`、`B`、`C` 或 `D`。`D` 是默认值，因此在输出中不会显示：

```
SELECT 'a:1A fat:2B,4C cat:5D'::tsvector;
          tsvector
----------------------------
 'a':1A 'cat':5 'fat':2B,4C
```

权重通常用于反映文档结构，例如把标题中的词和正文中的词区分开来。
文本检索排序函数可以为不同的权重标记分配不同优先级。

必须认识到，tsvector 类型本身并不会执行任何词语正规化；它假定输入的词已经按照应用需求完成正规化。例如：

```
select 'The Fat Rats'::tsvector;
      tsvector
--------------------
 'Fat' 'Rats' 'The'
```

对于大多数英文全文检索应用来说，上述词会被视为尚未正规化，但 tsvector 并不在意。原始文档文本通常应先经过 to_tsvector，以按检索需要对词语进行正规化：

```
SELECT to_tsvector('english', 'The Fat Rats');
   to_tsvector
-----------------
 'fat':2 'rat':3
```

更多细节仍请参见第 12 章。

8.11.2. tsquery

tsquery 值存储要搜索的词位，并按照布尔操作符 &（AND）、|（OR）和 !（NOT）把它们组合起来。可以使用圆括号强制指定这些操作符的分组方式：

```
SELECT 'fat & rat'::tsquery;
    tsquery
---------------
 'fat' & 'rat'

SELECT 'fat & (rat | cat)'::tsquery;
          tsquery
----------------------------
 'fat' & ( 'rat' | 'cat' )

SELECT 'fat & rat & ! cat'::tsquery;
        tsquery
------------------------
 'fat' & 'rat' & !'cat'
```

若没有圆括号，!（NOT）的绑定最紧，而 &（AND）比 |（OR）绑定得更紧。

可选地，tsquery 中的词位可以用一个或多个权重字母标注，这会限制它们只匹配 tsvector 中带有相应权重的词位：

```
SELECT 'fat:ab & cat'::tsquery;
    tsquery
------------------
 'fat':AB & 'cat'
```

此外，tsquery 中的词位还可以带上 * 标签来指定前缀匹配：

```
SELECT 'super:*'::tsquery;
  tsquery
```

```
-----------
 'super':*
```

这个查询将匹配 tsvector 中任何以“super”开头的词位。请注意，前缀会先由文本搜索配置处理，这意味着下面的比较会返回真：

```
SELECT to_tsvector( 'postgraduate' ) @@ to_tsquery( 'postgres:*' );
 ?column?
----------
 t
(1 row)
```

因为 postgres 会被词干化为 postgr：

```
SELECT to_tsquery('postgres:*');
 to_tsquery
------------
 'postgr':*
(1 row)
```

它随后就能匹配 postgraduate。

引号的使用规则与前面介绍 tsvector 时相同；同样，与 tsvector 一样，任何需要的词语正规化都必须在转换为 tsquery 类型之前完成。to_tsquery 函数很适合用来实现这种正规化：

```
SELECT to_tsquery('Fat:ab & Cats');
    to_tsquery
------------------
 'fat':AB & 'cat'
```

8.12. UUID类型

uuid 数据类型用于存储由 RFC 4122、ISO/IEC 9834-8:2005 及相关标准定义的通用唯一标识符（UUID）。（有些系统把这种数据类型称为全局唯一标识符，或 GUID。）这种标识符是一个 128 位的量，由某种算法生成，该算法的设计目标是：使已知宇宙中其他人用同一算法生成出相同标识符的概率极低。因此，对于分布式系统而言，
这类标识符能比序列生成器提供更好的唯一性保证，
因为序列生成器仅在单个数据库内唯一。

UUID 写作一串小写十六进制数字，并用连字符分隔成若干组：先是 8 位一组，接着是三个 4 位组，最后是一个 12 位组。总共 32 个十六进制数字表示 128 个二进制位。标准形式的 UUID 例如：

```
a0eebc99-9c0b-4ef8-bb6d-6bb9bd380a11
```

PostgreSQL 也接受其他输入形式：可以使用大写字母、用花括号包围标准格式、省略部分或全部连字符，或者在任意 4 位分组后额外加上连字符。例如：

```
A0EEBC99-9C0B-4EF8-BB6D-6BB9BD380A11
{a0eebc99-9c0b-4ef8-bb6d-6bb9bd380a11}
a0eebc999c0b4ef8bb6d6bb9bd380a11
a0ee-bc99-9c0b-4ef8-bb6d-6bb9-bd38-0a11
{a0eebc99-9c0b4ef8-bb6d6bb9-bd380a11}
```

输出总是采用标准形式。

PostgreSQL为 UUID 提供存储和比较函数，但核心数据库不包含生成 UUID 的函数，因为没有一种算法适合所有应用。uuid-ossp模块提供实现多种标准算法的函数。或者，UUID 也可以由客户端应用或通过服务器端函数调用的其他库来生成 UUID。

8.13. XML类型

xml 数据类型可用于存储 XML 数据。与把 XML 数据存储在 text 字段中相比，它的优势在于会检查输入值的良构性，并且提供了可执行类型安全操作的支持函数；见第 9.14 节。使用该数据类型要求构建时使用 configure --with-libxml。

xml 类型既可以存储 XML 标准所定义的良构“文档”，也可以存储“内容”片段，后者由 XML 标准中的产生式 XMLDecl? content 定义。粗略地说，这意味着内容片段可以拥有多个顶层元素或字符节点。表达式 *xmlvalue* IS DOCUMENT 可用来判定某个特定的 xml 值是完整文档还是仅仅一个内容片段。

8.13.1. 创建XML值

要从字符数据生成 xml 类型的值，可以使用函数 xmlparse：

```
XMLPARSE ( { DOCUMENT | CONTENT } value)
```

示例：

```
XMLPARSE (DOCUMENT '<?xml version="1.0"?><book><title>Manual</
title><chapter>...</chapter></book>')
XMLPARSE (CONTENT 'abc<foo>bar</foo><bar>foo</bar>')
```

按照 SQL 标准，这是把字符串转换为 XML 值的唯一方法；但 PostgreSQL 特有的下列语法：

```
xml '<foo>bar</foo>'
'<foo>bar</foo>'::xml
```

也可以使用。

即使输入值指定了文档类型声明（DTD），xml 类型也不会根据 DTD 来验证输入值。目前也没有内置支持可依据其他 XML 模式语言（如 XML Schema）来执行验证。

反向操作，也就是从 xml 生成字符串值，则使用函数 xmlserialize：

```
XMLSERIALIZE ( { DOCUMENT | CONTENT } value AS type )
```

type 可以是 character、character varying 或 text（或它们的别名）。同样，根据 SQL 标准，这是在 xml 与字符类型之间进行转换的唯一方式，但 PostgreSQL 也允许你直接进行类型转换。

当字符串值在不经过 XMLPARSE 或 XMLSERIALIZE 的情况下与 xml 类型互相转换时，选择 DOCUMENT 还是 CONTENT 由会话配置参数“XML 选项”决定，可以使用标准命令设置：

```
SET XML OPTION { DOCUMENT | CONTENT };
```

或使用更接近 PostgreSQL 风格的语法

```
SET xmloption TO { DOCUMENT | CONTENT };
```

默认值是 CONTENT，因此允许所有形式的 XML 数据。

> **注意**
>
> 在使用默认的 XML 选项设置时，如果字符串中包含文档类型声明，则不能直接把它转换为 xml 类型，因为 XML 内容片段的定义不接受文档类型声明。如果需要这样做，可以使用 XMLPARSE，或者更改 XML 选项。

8.13.2. 编码处理

在客户端、服务器以及其间传输的 XML 数据上处理多种字符编码时，必须格外小心。使用文本模式向服务器发送查询并把查询结果返回给客户端时（这是通常使用的模式），PostgreSQL 会将客户端与服务器之间传输的所有字符数据转换为目标端的字符编码，参见第 22.3 节。这也包括表示 XML 值的字符串，如上例所示。这通常意味着，由于字符数据在客户端和服务器之间传输时可能被转换为其他编码，XML 数据中包含的编码声明可能会失效，因为内嵌的编码声明本身并不会被修改。为处理这种情况，表示 `xml` 类型输入值的字符串中所包含的编码声明会被忽略，其内容被假定为当前服务器编码。因此，为了正确处理，客户端发出的 XML 数据字符串必须采用当前客户端编码。客户端负责在将文档发送给服务器之前把它们转换为当前客户端编码，或适当调整客户端编码。输出时，`xml` 类型值不会带有编码声明，而客户端应假定所有数据都采用当前客户端编码。

若使用二进制模式把查询参数传给服务器并把查询结果返回客户端，则不会执行字符集转换，因此情况有所不同。在这种情况下，XML 数据中的编码声明会被识别；如果缺少编码声明，则该数据会被假定为 UTF-8（这是 XML 标准的要求；另请注意 PostgreSQL 不支持 UTF-16）。输出时，除非客户端编码是 UTF-8（此时编码声明会被省略），否则数据会带有一个指明客户端编码的编码声明。

不言而喻，在 PostgreSQL 中处理 XML 数据时，如果 XML 数据编码、客户端编码和服务器编码三者相同，不仅更高效，也更不容易出错。由于 XML 数据在内部以 UTF-8 处理，如果服务器编码也是 UTF-8，则处理效率最高。

> **小心**
>
> 当服务器编码不是 UTF-8 时，某些与 XML 相关的函数在非 ASCII 数据上可能完全无法工作。尤其是 `xpath()`，这是一个已知问题。

8.13.3. 访问XML值

`xml` 数据类型有些特殊，因为它不提供任何比较操作符。这是因为对 XML 数据并不存在良定义且通用的比较算法。其结果是，你无法通过把某个 `xml` 列与搜索值比较来检索行。因此，XML 值通常应伴随一个独立的键字段，例如 ID。另一种比较 XML 值的办法，是先把它们转换成字符串；但请注意，字符串比较对 XML 的比较需求通常帮助不大。

由于 `xml` 数据类型没有可用的比较操作符，因此无法直接在这种类型的列上创建索引。如果需要在 XML 中快速搜索，可行方案包括：将表达式转换为字符串类型后为其建立索引，或者为某个 XPath 表达式建立索引。当然，实际查询也必须相应调整为使用该被索引的表达式。

PostgreSQL 的文本检索功能也可用于加速 XML 数据的全文检索。不过，目前 PostgreSQL 发行版中仍缺少所需的预处理支持。

8.14. 数组

PostgreSQL 允许将表列定义为变长多维数组。可以创建任何内置或用户定义的基础类型、枚举类型或复合类型的数组。尚不支持域的数组。

8.14.1. 数组类型的声明

为了说明数组类型的用法，我们创建下面这个表：

```
CREATE TABLE sal_emp (
    name            text,
    pay_by_quarter  integer[],
    schedule        text[][]
```

```
);
```

如上所示，数组数据类型通过在数组元素的数据类型名后附加方括号（[]）来命名。
上述命令将创建一个名为 sal_emp 的表，其中有一个 text 类型的列（name）、一个表示员工各季度薪资的一维 integer 数组（pay_by_quarter），以及一个表示员工每周日程安排的二维 text 数组（schedule）。

CREATE TABLE 的语法允许指定数组的确切大小，例如：

```
CREATE TABLE tictactoe (
    squares   integer[3][3]
);
```

不过，当前实现会忽略给出的任何数组大小限制，也就是说，
其行为与未指定长度的数组相同。

当前实现也不会强制执行所声明的维度数。对于某一特定元素类型的数组，
无论大小或维度数如何，都会被视为同一种类型。因此，在 CREATE TABLE 中声明数组大小或维度数仅仅是文档说明；它不会影响运行时行为。

另一种使用关键字 ARRAY、并且符合 SQL 标准的语法可用于一维数组。pay_by_quarter 也可以定义为：

```
    pay_by_quarter  integer ARRAY[4],
```

或者，如果不指定数组大小：

```
    pay_by_quarter  integer ARRAY,
```

不过，与前面一样，PostgreSQL 在任何情况下都不会强制执行大小限制。

8.14.2. 数组值输入

要把数组值写成字面常量，请将元素值放在花括号内，并用逗号分隔。（如果你了解 C，这与 C 中初始化结构体的语法有些类似。）你可以给任意元素值加上双引号；如果它包含逗号或花括号，则必须这样做。（更多细节见下文。）因此，数组常量的一般格式如下：

```
'{ val1 delim val2 delim ... }'
```

其中 *delim* 是该类型的分隔符字符，它记录在其 pg_type 条目中。在 PostgreSQL 发行版提供的标准数据类型里，除类型 box 使用分号（;）外，其余都使用逗号（,）。每个 *val* 要么是数组元素类型的常量，要么是一个子数组。数组常量的一个示例是：

```
'{{1,2,3},{4,5,6},{7,8,9}}'
```

这个常量是一个 3 x 3 的二维数组，由三个整数子数组构成。

要把数组常量中的某个元素设为 NULL，请把该元素值写成 NULL。（NULL 的任意大小写变体都可以。）如果你想要实际的字符串值“NULL”，就必须为它加上双引号。

（这类数组常量实际上只是第 4.1.2.7 节中讨论的通用类型常量的一种特例。该常量最初会被当作字符串处理，然后传递给数组输入转换例程。可能需要显式指定类型。）

现在我们来看几个 INSERT 语句：

```
INSERT INTO sal_emp
    VALUES ('Bill',
    '{10000, 10000, 10000, 10000}',
    '{{"meeting", "lunch"}, {"training", "presentation"}}');

INSERT INTO sal_emp
```

```
    VALUES ('Carol',
    '{20000, 25000, 25000, 25000}',
    '{{"breakfast", "consulting"}, {"meeting", "lunch"}}');
```

前两条插入语句的结果如下：

```
SELECT * FROM sal_emp;
 name  |      pay_by_quarter       |                  schedule
-------+---------------------------+-------------------------------
------------
 Bill  | {10000,10000,10000,10000} | {{meeting,lunch},{training,
presentation}}
 Carol | {20000,25000,25000,25000} | {{breakfast,consulting},{
meeting,lunch}}
(2 rows)
```

多维数组的各个子数组在每个维度上的长度必须匹配。不匹配会导致错误，例如：

```
INSERT INTO sal_emp
    VALUES ('Bill',
    '{10000, 10000, 10000, 10000}',
    '{{"meeting", "lunch"}, {"meeting"}}');
ERROR:  multidimensional arrays must have array expressions with
 matching dimensions
```

> **注意**
>
> 请记住，你在 SQL 命令中写的内容会先被解释为字符串字面量，然后才被解释为数组。这会使你所需的反斜杠数量加倍。例如，
> 要插入一个包含一个反斜杠和一个双引号的`text`数组值，你需要写：
>
> ```
> INSERT ... VALUES (E'{"\\\\","\\""}');
> ```
>
> 转义字符串处理器会去掉一层反斜杠，因此传到数组值解析器时看起来是`{"\\","\""}`。进而，送入`text`数据类型输入例程的字符串分别变成`\`和`"`。
> （如果我们使用的数据类型的输入例程也会对反斜杠做特殊处理，例如`bytea`，那么为了在存储的数组元素中得到一个反斜杠，命令里可能需要多达八个反斜杠。）
> 美元引用（见第 4.1.2.4 节）可用于避免反斜杠加倍的需要。

也可以使用 `ARRAY` 构造器语法：

```
INSERT INTO sal_emp
    VALUES ('Bill',
    ARRAY[10000, 10000, 10000, 10000],
    ARRAY[['meeting', 'lunch'], ['training', 'presentation']]);

INSERT INTO sal_emp
    VALUES ('Carol',
    ARRAY[20000, 25000, 25000, 25000],
    ARRAY[['breakfast', 'consulting'], ['meeting', 'lunch']]);
```

注意，数组元素是普通的 SQL 常量或表达式；例如，字符串字面值用单引号而不是双引号包围，而在数组字面量中则会用双引号。关于 `ARRAY` 构造器语法的更多细节见第 4.2.12 节。

8.14.3. 访问数组

现在，我们可以对该表执行一些查询。首先，演示如何访问数组中的单个元素。
下面的查询取回第二季度薪资发生变化的员工姓名：

```
SELECT name FROM sal_emp WHERE pay_by_quarter[1] <> pay_by_quarter[
2];

 name
-------
 Carol
(1 row)
```

数组下标写在方括号内。默认情况下，PostgreSQL 对数组采用从 1 开始的编号约定，也就是说，一个有 *n* 个元素的数组从 `array[1]` 开始，到 `array[n]` 结束。

下面这个查询取回所有员工第三季度的薪资：

```
SELECT pay_by_quarter[3] FROM sal_emp;

 pay_by_quarter
----------------
          10000
          25000
(2 rows)
```

我们还可以访问数组的任意矩形切片，即子数组。
数组切片通过在一个或多个数组维度上写成 *lower-bound*:*upper-bound* 的形式来表示。例如，下面这个查询取回 Bill 在一周中前两天里每天的首项日程：

```
SELECT schedule[1:2][1:1] FROM sal_emp WHERE name = 'Bill';

        schedule
------------------------
 {{meeting},{training}}
(1 row)
```

如果任何一个维度写成切片形式，也就是包含冒号，那么所有维度都会被当作切片处理。
任何只有单个数字（没有冒号）的维度都会被视为从 1 到该数字指定的范围。例如，`[2]` 会被当作 `[1:2]`，如下例所示：

```
SELECT schedule[1:2][2] FROM sal_emp WHERE name = 'Bill';

                 schedule
--------------------------------------------
 {{meeting,lunch},{training,presentation}}
(1 row)
```

为了避免与非切片情况混淆，最好对所有维度都使用切片语法，例如写成 `[1:2][1:1]`，而不是 `[2][1:1]`。

如果数组本身或任一下标表达式为 NULL，则数组下标表达式将返回 NULL。此外，如果下标超出数组边界，也会返回 NULL（这种情况不会报错）。例如，如果 `schedule` 当前的维度是 `[1:3][1:2]`，那么引用 `schedule[3][3]` 会得到 NULL。类似地，使用错误数量的下标访问数组，得到的也是 NULL 而不是错误。

同样地，如果数组本身或任一下标表达式为 NULL，数组切片表达式也会返回 NULL。不过，在其他情况下，例如选择一个完全位于当前数组边界之外的数组切片时，
切片表达式返回的是空（零维）数组而不是 NULL。（这与非切片行为不一致，是出于历史原因。）如果所请求的切片与数组边界仅部分重叠，那么它会被静默缩减为重叠区域，
而不是返回 NULL。

任意数组值的当前维度都可以用 `array_dims` 函数取回：

```
SELECT array_dims(schedule) FROM sal_emp WHERE name = 'Carol';
```

```
 array_dims
------------
 [1:2][1:2]
(1 row)
```

array_dims 产生的是一个 text 结果，这对人工阅读比较方便，但对程序来说可能不太方便。也可以用 array_upper 和 array_lower 取回维度信息，它们分别返回指定数组维度的上界和下界：

```
SELECT array_upper(schedule, 1) FROM sal_emp WHERE name = 'Carol';

 array_upper
-------------
           2
(1 row)
```

array_length 会返回指定数组维度的长度：

```
SELECT array_length(schedule, 1) FROM sal_emp WHERE name = 'Carol';

 array_length
--------------
            2
(1 row)
```

8.14.4. 修改数组

数组值可以被整体替换：

```
UPDATE sal_emp SET pay_by_quarter = '{25000,25000,27000,27000}'
    WHERE name = 'Carol';
```

或者使用 ARRAY 表达式语法：

```
UPDATE sal_emp SET pay_by_quarter = ARRAY[25000,25000,27000,27000]
    WHERE name = 'Carol';
```

也可以更新数组中的单个元素：

```
UPDATE sal_emp SET pay_by_quarter[4] = 15000
    WHERE name = 'Bill';
```

或者更新其中一个切片：

```
UPDATE sal_emp SET pay_by_quarter[1:2] = '{27000,27000}'
    WHERE name = 'Carol';
```

已存储的数组值可以通过给尚不存在的元素赋值来扩展。
原有元素与新赋值元素之间的任何位置都将用 NULL 填充。例如，如果数组 myarray 当前有 4 个元素，那么在一次更新把值赋给 myarray[6] 之后，它将有 6 个元素；myarray[5] 将包含 NULL。目前，以这种方式扩展只允许用于一维数组，不允许用于多维数组。

带下标的赋值也允许创建不使用从 1 开始下标的数组。例如，可以给 myarray[-2:7] 赋值，从而创建一个下标值范围为 -2 到 7 的数组。

新的数组值也可以使用连接操作符 || 构造：

```
SELECT ARRAY[1,2] || ARRAY[3,4];
 ?column?
-----------
```

```
 {1,2,3,4}
(1 row)

SELECT ARRAY[5,6] || ARRAY[[1,2],[3,4]];
      ?column?
---------------------
 {{5,6},{1,2},{3,4}}
(1 row)
```

连接操作符允许把单个元素添加到一维数组的开头或末尾。它也接受两个 *N* 维数组，或者一个 *N* 维数组与一个 *N+1* 维数组。

当把单个元素添加到一维数组的开头或末尾时，结果数组会保留该数组操作数的下界下标。例如：

```
SELECT array_dims(1 || '[0:1]={2,3}'::int[]);
 array_dims
------------
 [0:2]
(1 row)

SELECT array_dims(ARRAY[1,2] || 3);
 array_dims
------------
 [1:3]
(1 row)
```

当连接两个维度数相同的数组时，结果会保留左侧操作数外层维度的下界下标。结果数组由左侧操作数的所有元素后跟右侧操作数的所有元素组成。例如：

```
SELECT array_dims(ARRAY[1,2] || ARRAY[3,4,5]);
 array_dims
------------
 [1:5]
(1 row)

SELECT array_dims(ARRAY[[1,2],[3,4]] || ARRAY[[5,6],[7,8],[9,0]]);
 array_dims
------------
 [1:5][1:2]
(1 row)
```

当把一个 *N* 维数组添加到一个 *N+1* 维数组的开头或末尾时，其结果与前面“单个元素与数组相连”的情况类似。每个 *N* 维子数组本质上都是该 *N+1* 维数组外层维度中的一个元素。例如：

```
SELECT array_dims(ARRAY[1,2] || ARRAY[[3,4],[5,6]]);
 array_dims
------------
 [1:3][1:2]
(1 row)
```

也可以使用 `array_prepend`、`array_append` 或 `array_cat` 函数来构造数组。前两个只支持一维数组，而 `array_cat` 支持多维数组。示例如下：

```
SELECT array_prepend(1, ARRAY[2,3]);
 array_prepend
---------------
 {1,2,3}
(1 row)
```

```
SELECT array_append(ARRAY[1,2], 3);
 array_append
--------------
 {1,2,3}
(1 row)

SELECT array_cat(ARRAY[1,2], ARRAY[3,4]);
 array_cat
-----------
 {1,2,3,4}
(1 row)

SELECT array_cat(ARRAY[[1,2],[3,4]], ARRAY[5,6]);
      array_cat
---------------------
 {{1,2},{3,4},{5,6}}
(1 row)

SELECT array_cat(ARRAY[5,6], ARRAY[[1,2],[3,4]]);
      array_cat
---------------------
 {{5,6},{1,2},{3,4}}
```

在简单情况下，优先使用上面讨论的连接操作符，而不是直接调用这些函数。不过，由于连接操作符被重载以同时服务于这三种情形，所以在某些场景下使用这些函数之一有助于避免歧义。例如，考虑：

```
SELECT ARRAY[1, 2] || '{3, 4}';  -- 未指定类型的字面量被当作数组
 ?column?
-----------
 {1,2,3,4}

SELECT ARRAY[1, 2] || '7';                  -- 这个字面量也一样
ERROR:  malformed array literal: "7"

SELECT ARRAY[1, 2] || NULL;                 -- 未加类型修饰的 NULL 也一样
 ?column?
----------
 {1,2}
(1 row)

SELECT array_append(ARRAY[1, 2], NULL);     -- 这可能才是原本想表达的意思
 array_append
--------------
 {1,2,NULL}
```

在上面的示例中，解析器看到连接操作符的一侧是整数数组，另一侧是类型未定的常量。它用来解析该常量类型的启发式规则是假定其类型与该操作符另一侧的输入相同，在这里就是整数数组。因此，连接操作符会被假定为表示 `array_cat`，而不是 `array_append`。如果这是错误的选择，可以通过把常量显式转换为数组的元素类型来修正；但显式使用 `array_append` 也许是更好的解决方案。

8.14.5. 在数组中搜索

要在数组中搜索某个值，就必须检查每一个值。如果你知道数组的大小，这可以手工完成。例如：

```
SELECT * FROM sal_emp WHERE pay_by_quarter[1] = 10000 OR
                            pay_by_quarter[2] = 10000 OR
                            pay_by_quarter[3] = 10000 OR
                            pay_by_quarter[4] = 10000;
```

不过，对于大型数组，这样很快就会变得繁琐；而在数组大小未知时，这也没有帮助。
另一种方法见第 9.23 节。上面的查询可以改写为：

```
SELECT * FROM sal_emp WHERE 10000 = ANY (pay_by_quarter);
```

此外，如果要查找数组中所有值都等于 10000 的行，可以使用：

```
SELECT * FROM sal_emp WHERE 10000 = ALL (pay_by_quarter);
```

另外，也可以使用 `generate_subscripts` 函数。例如：

```
SELECT * FROM
   (SELECT pay_by_quarter,
           generate_subscripts(pay_by_quarter, 1) AS s
      FROM sal_emp) AS foo
 WHERE pay_by_quarter[s] = 10000;
```

关于该函数的说明见表 9.50。

> **提示**
>
> 数组不是集合；搜索特定数组元素可能是数据库设计不当的信号。
> 可以考虑使用一个独立的表，让原本会成为数组元素的每个项各占一行。
> 这样会更容易搜索，而且在元素数量很多时通常也有更好的可伸缩性。

8.14.6. 数组输入和输出语法

数组值的外部文本表示由若干项构成，这些项会按照数组元素类型的 I/O 转换规则进行解释，再加上一些表示数组结构的修饰。修饰部分包括数组值外层的花括号（`{` 和 `}`），以及相邻项之间的分隔符字符。分隔符字符通常是逗号（`,`），
但也可能是别的字符：它由数组元素类型的 `typdelim` 设置决定。在 PostgreSQL 发行版提供的标准数据类型中，除类型 `box` 使用分号（`;`）外，其余都使用逗号。在多维数组中，每个维度（行、平面、立方体等）都有自己的一层花括号，并且同一级别相邻的、
由花括号括起的各项之间必须写出分隔符。

如果元素值是空字符串、包含花括号、分隔符字符、双引号、反斜线或空白，或者与单词 `NULL` 匹配，数组输出例程就会在元素值外面加上双引号。元素值中嵌入的双引号和反斜线会用反斜线转义。对于数字数据类型，可以安全地假定不会出现双引号；
但对于文本数据类型，则应准备好处理带引号和不带引号两种情况。

默认情况下，数组各维度的下界下标值都设为 1。要表示具有其他下界的数组，可以在写出数组内容之前显式指定数组下标范围。这种修饰由包围每个数组维度上下界的方括号（`[]`）构成，中间以冒号（`:`）作为分隔符字符。数组维度修饰后面再跟一个等号（`=`）。例如：

```
SELECT f1[1][-2][3] AS e1, f1[1][-1][5] AS e2
 FROM (SELECT '[1:1][-2:-1][3:5]={{{1,2,3},{4,5,6}}}'::int[] AS f1)
 AS ss;

 e1 | e2
----+----
  1 |  6
(1 row)
```

只有当一个或多个下界不等于 1 时，数组输出例程才会在结果中包含显式维度信息。

如果某个元素写入的值是 NULL（任何大小写变体都可以），该元素就会被视为 NULL。若存在任何引号或反斜线，则会禁用这种行为，从而可以输入字面字符串值“NULL”。另外，为了与 PostgreSQL 8.2 之前的版本保持向后兼容，可以将array_nulls配置参数设为 off，从而禁止把 NULL 识别为 NULL。

如前所述，在写数组值时，可以给任意单个数组元素加上双引号。如果元素值不加引号就会让数组值解析器产生混淆，那么你必须这样做。例如，包含花括号、逗号（或者该数据类型的分隔符字符）、双引号、反斜线，或者前后带有空白的元素值，必须使用双引号。空字符串以及与单词 NULL 匹配的字符串也必须加引号。要在一个带双引号的数组元素值中放入双引号或反斜线，需要使用转义字符串语法并在它前面加反斜线。或者，你也可以避免使用引号，而改用反斜线转义来保护所有原本会被当作数组语法的数据字符。

你可以在左花括号之前或右花括号之后添加空白。也可以在任意单个项字符串之前或之后添加空白。在所有这些情况下，这些空白都会被忽略。不过，双引号元素内部的空白，或者元素中夹在两个非空白字符之间的空白，不会被忽略。

> **提示**
>
> 在 SQL 命令中写数组值时，ARRAY 构造器语法（见第 4.2.12 节）通常比数组字面量语法更容易使用。在 ARRAY 中，各个元素值的写法与它们不作为数组成员时完全相同。

8.15. 复合类型

复合类型表示一行或一条记录的结构；本质上它就是字段名及其数据类型的列表。PostgreSQL允许像使用简单类型那样，在许多相同的场合使用复合类型。例如，表中的一列可以声明为复合类型。

8.15.1. 复合类型的声明

下面是两个定义复合类型的简单示例：

```
CREATE TYPE complex AS (
    r       double precision,
    i       double precision
);

CREATE TYPE inventory_item AS (
    name            text,
    supplier_id     integer,
    price           numeric
);
```

这种语法与CREATE TABLE类似，但只能指定字段名和类型；目前还不能包含约束（例如 NOT NULL）。注意，AS关键字必不可少；如果没有它，系统会认为你想使用另一种CREATE TYPE命令，并产生令人费解的语法错误。

定义了这些类型之后，我们可以用它们来创建表：

```
CREATE TABLE on_hand (
    item      inventory_item,
    count     integer
);
```

```
INSERT INTO on_hand VALUES (ROW('fuzzy dice', 42, 1.99), 1000);
```

或者创建函数：

```
CREATE FUNCTION price_extension(inventory_item, integer) RETURNS
 numeric
AS 'SELECT $1.price * $2' LANGUAGE SQL;

SELECT price_extension(item, 10) FROM on_hand;
```

每当创建一张表时，也会自动创建一个与该表同名的复合类型，用来表示表的行类型。
例如，假如我们执行了：

```
CREATE TABLE inventory_item (
    name            text,
    supplier_id     integer REFERENCES suppliers,
    price           numeric CHECK (price > 0)
);
```

那么，与上文相同的`inventory_item`复合类型就会随之产生，而且可以像上文那样使用。
不过，请注意当前实现的一个重要限制：由于复合类型本身不关联任何约束，
表定义中的约束并不适用于表之外的复合类型值。
（一种部分解决办法是使用域类型作为复合类型的成员。）

8.15.2. 构造组合值

要把组合值写成字面常量，请将各字段值放在圆括号内，并用逗号分隔。
你可以给任意字段值加双引号；如果它包含逗号或圆括号，则必须这样做。（更多细节见下文。）因此，组合常量的一般格式如下：

```
'( val1 , val2 , ... )'
```

一个示例是：

```
'("fuzzy dice",42,1.99)'
```

这就是上文定义的`inventory_item`类型的一个合法值。要让某个字段为 NULL，就在列表中对应的位置什么也不写。例如，这个常量指定第三个字段为 NULL：

```
'("fuzzy dice",42,)'
```

如果想写空字符串而不是 NULL，请写双引号：

```
'("",42,)'
```

这里第一个字段是非 NULL 的空字符串，第三个字段是 NULL。

（这些常量实际上只是第 4.1.2.7 节中讨论的通用类型常量的一种特例。该常量最初会被当作字符串处理，然后传递给复合类型输入转换例程。必要时可能需要显式指定类型。）

这种`ROW`表达式语法也可以用于构造复合值。在大多数情况下，
它比字符串字面量语法简单得多，因为你不必担心多层引号。
我们在上文已经用过这种方法：

```
ROW('fuzzy dice', 42, 1.99)
ROW('', 42, NULL)
```

只要表达式中有多个字段，ROW 关键字实际上是可选的，因此这些可以简写为：

```
('fuzzy dice', 42, 1.99)
```

```
('', 42, NULL)
```

这种ROW表达式语法的更多细节见第 4.2.13 节。

8.15.3. 访问复合类型

要访问组合列中的某个字段，可以写一个点号再加字段名，这很像通过表名选取字段。实际上，它与通过表名选取字段太像了，以至于你通常必须使用圆括号，以免让解析器混淆。例如，你可能尝试从示例表on_hand中选取一些子字段：

```
SELECT item.name FROM on_hand WHERE item.price > 9.99;
```

这不会起作用，因为根据 SQL 语法规则，名称item会被当成表名，而不是on_hand的列名。你必须写成这样：

```
SELECT (item).name FROM on_hand WHERE (item).price > 9.99;
```

或者，如果你还需要使用表名（例如在多表查询中），可以这样写：

```
SELECT (on_hand.item).name FROM on_hand WHERE (on_hand.item).price
 > 9.99;
```

现在，加上括号的对象就会被正确解释为对item列的引用，然后就可以从中选出子字段。

无论何时从组合值中选择字段，都会遇到类似的语法问题。例如，要从一个返回组合值的函数结果中只选取一个字段，你需要这样写：

```
SELECT (my_func(...)).field FROM ...
```

如果没有额外的圆括号，这将生成一个语法错误。

特殊字段名*表示“所有字段”，其进一步解释见第 8.15.5 节。

8.15.4. 修改组合值

下面是一些插入和更新组合列时正确语法的示例。先看插入或更新整个列值的情况：

```
INSERT INTO mytab (complex_col) VALUES((1.1,2.2));

UPDATE mytab SET complex_col = ROW(1.1,2.2) WHERE ...;
```

第一个示例省略了ROW，第二个示例使用了它；两种写法都可以。

我们也可以更新组合列中的单个子字段：

```
UPDATE mytab SET complex_col.r = (complex_col).r + 1 WHERE ...;
```

注意，这里不需要（事实上也不能）给紧跟在SET后面的列名加圆括号，但在等号右侧的表达式中引用同一列时，则需要加圆括号。

我们也可以把子字段指定为INSERT的目标：

```
INSERT INTO mytab (complex_col.r, complex_col.i) VALUES(1.1, 2.2);
```

如果我们没有为该列的所有子字段提供值，其余子字段就会填充为 NULL 值。

8.15.5. 在查询中使用复合类型

在查询中，复合类型有多种特殊的语法规则和行为。这些规则提供了有用的简写形式，但如果不了解背后的逻辑，也可能让人困惑。

在PostgreSQL中，查询中对表名（或别名）的引用，
实际上就是对该表当前行的组合值的引用。例如，如果我们有一个如
上文所示的表inventory_item，就可以写：

```
SELECT c FROM inventory_item c;
```

这个查询会产生一个单独的组合值列，因此我们可能得到如下输出：

```
           c
------------------------
 ("fuzzy dice",42,1.99)
(1 row)
```

不过要注意，简单名称会先与列名匹配，再与表名匹配，因此这个示例之所以可行，
只是因为该查询涉及的表中没有名为c的列。

普通的限定列名语法*table_name.column_name*
可以理解为对该表当前行的组合值进行字段选择。（出于效率原因，
实际上并不是这样实现的。）

当我们写

```
SELECT c.* FROM inventory_item c;
```

时，根据 SQL 标准，应该得到把该表内容展开为独立列后的结果：

```
    name     | supplier_id | price
-------------+-------------+-------
 fuzzy dice |           42 |  1.99
(1 row)
```

就好像查询写成了

```
SELECT c.name, c.supplier_id, c.price FROM inventory_item c;
```

PostgreSQL会对任何结果为复合类型的表达式应用这种展开行为，不过正如上文所示，只要
.*所作用的值不是简单表名，就需要给该值加圆括号。例如，如果myfunc()
是一个返回复合类型的函数，该复合类型有a、b和c三列，那么下面两个查询的结果相同：

```
SELECT (myfunc(x)).* FROM some_table;
SELECT (myfunc(x)).a, (myfunc(x)).b, (myfunc(x)).c FROM some_table;
```

> **提示**
>
> PostgreSQL处理列展开时，实际上会把第一种形式转换成第二种形式。因此，
> 在这个示例中，myfunc()每行都会被调用三次，无论采用哪种语法。
> 如果它是一个开销较大的函数，你可能希望避免这种情况，可以使用如下查询：
>
> ```
> SELECT (m).* FROM (SELECT myfunc(x) AS m FROM some_table
> OFFSET 0) ss;
> ```
>
> 这里的OFFSET 0子句可以防止优化器“展平”子查询，从而避免形成会多次调用以下
> 函数的形式：myfunc()。

这里的*composite_value*.*语法在以下结构的顶层出现时会产生这类列展开：（SELECT
输出列表）、RETURNING列表（位于INSERT/UPDATE/DELETE）、VALUES子句，或
行构造器。在所有其他上下文中（包括嵌套在上述结构之内时），将.*
附加到复合值上不会改变该值，因为它表示“所有列”，因此结果仍然是同一个复合值。
例如，如果somefunc()接受一个复合值参数，这些查询就是等价的：

```
SELECT somefunc(c.*) FROM inventory_item c;
SELECT somefunc(c) FROM inventory_item c;
```

在这两种情况下，inventory_item的当前行都会作为单个复合值参数传递给该函数。
即使.*在这种情况下不起作用，使用它仍是良好的风格，
因为它明确表示这里需要的是复合值。特别是，解析器会将c（位于c.*）
解释为表名或别名，而不是列名，因此不存在歧义；但如果没有.*，就不能明确判断c
表示表名还是列名，而且会优先采用列名解释，只要存在一列名为c。

另一个说明这些概念的例子是，下面这些查询的含义都相同：

```
SELECT * FROM inventory_item c ORDER BY c;
SELECT * FROM inventory_item c ORDER BY c.*;
SELECT * FROM inventory_item c ORDER BY ROW(c.*);
```

所有这些ORDER BY子句都指定了该行的组合值。不过，如果inventory_item包含一个名为c的列，第一种情况就会不同于其他情况，因为它表示只按那一列排序。
按照前面展示的列名，下面这些查询也与上述查询等效：

```
SELECT * FROM inventory_item c ORDER BY ROW(c.name, c.supplier_id,
 c.price);
SELECT * FROM inventory_item c ORDER BY (c.name, c.supplier_id, c.
price);
```

（最后一种情况使用的是省略了关键字ROW的行构造器。）

另一种与组合值有关的特殊语法行为是，我们可以使用函数记法来提取组合值中的字段。
简单来说，记法*field*(*table*)和*table*.*field*可以互换。例如，这些查询是等价的：

```
SELECT c.name FROM inventory_item c WHERE c.price > 1000;
SELECT name(c) FROM inventory_item c WHERE price(c) > 1000;
```

此外，如果我们有一个接受单个复合类型参数的函数，也可以用这两种记法来调用它。
这些查询都等价：

```
SELECT somefunc(c) FROM inventory_item c;
SELECT somefunc(c.*) FROM inventory_item c;
SELECT c.somefunc FROM inventory_item c;
```

函数记法与字段记法之间的这种等价性，
使得我们可以通过在复合类型上使用函数来实现“计算字段”。
使用上面最后一种查询形式的应用程序，无需直接知道somefunc并不是该表中的真实列。

> **提示**
>
> 由于这种行为，
> 不宜让接受单个复合类型参数的函数与该复合类型中的任何字段同名。
> 如果存在歧义，会优先选择字段名解释，
> 因此不采用特殊办法就无法调用这样的函数。
> 强制按函数解释的一种方法是为函数名加模式限定，也就是写成 *schema*.*func*(*compositevalue*)。

8.15.6. 复合类型的输入和输出语法

组合值的外部文本表示由两部分组成：一部分是按照各字段类型的 I/O 转换规则解释的项，另一部分是表明组合结构的附加符号。这些附加符号包括包围整个值的圆括号（(和)），

以及相邻项之间的逗号（,）。圆括号外部的空白会被忽略；但在圆括号内部，空白会被视为字段值的一部分，其是否有意义取决于该字段数据类型的输入转换规则。例如，在

```
'(  42)'
```

中，如果字段类型是 integer，则空白会被忽略；如果是 text，则不会被忽略。

如前所示，在写组合值时，你可以给任意单个字段值加双引号。如果字段值本身可能让组合值解析器混淆，则必须这样做。特别是，包含圆括号、逗号、双引号或反斜杠的字段必须用双引号括起来。要在带引号的组合字段值中写入双引号或反斜杠，需要在其前面加一个反斜杠。（另外，带双引号的字段值内部成对出现的双引号会被视为一个双引号字符，这与 SQL 字面字符串中单引号的规则类似。）或者，你也可以完全不使用引号，而改用反斜杠转义，保护所有原本会被当作组合语法的数据字符。

完全空的字段值（即逗号或圆括号之间一个字符也没有）表示 NULL。要写一个空字符串值而不是 NULL，可以写成""。

如果字段值是空字符串，或者包含圆括号、逗号、双引号、反斜杠或空白字符，复合类型输出例程就会在其周围加上双引号。（对空白字符这样做并非必需，但有助于提高可读性。）嵌入字段值中的双引号和反斜杠会被加倍。

注意

记住，你在 SQL 命令中写的内容会先被解释为字符串字面量，然后才会被解释为组合值。这会使所需的反斜杠数量翻倍（假定使用的是转义字符串语法）。例如，要在组合值中插入一个包含双引号和反斜杠的`text`字段，需要写成：

```
INSERT ... VALUES (E'("\\"\\\\")');
```

字符串字面量处理器会去掉一层反斜杠，因此传到组合值解析器时看起来是`("\"\\")`。随后，送入`text`数据类型输入例程的字符串就变成了`"\`。（如果我们使用的数据类型的输入例程也会把反斜杠当作特殊字符处理，例如`bytea`，那么为了在存储的组合字段中得到一个反斜杠，命令里可能需要多达八个反斜杠。）美元引用（见第 4.1.2.4 节）可用于避免反斜杠加倍的需要。

提示

在 SQL 命令中编写组合值时，`ROW`构造器语法通常比组合字面量语法更容易使用。在`ROW`中，各个字段值的写法与它们不是组合成员时完全相同。

8.16. 范围类型

范围类型是表示某种元素类型的值范围的数据类型（该元素类型称为范围的子类型（subtype））。例如，`timestamp`的范围可用于表示会议室被预订的时间范围。在这种情况下，数据类型为`tsrange`（“timestamp range”的缩写），而`timestamp`就是其子类型。子类型必须具有全序，这样才能明确定义元素值是位于某个值范围之内、之前还是之后。

范围类型很有用，因为它们能用单个范围值表示许多元素值，也能清晰地表达诸如范围重叠之类的概念。将时间和日期范围用于日程安排，是最清晰的例子；但价格区间、仪器的测量范围等场景也同样有用。

8.16.1. 内置范围类型

PostgreSQL 提供以下内置范围类型：

- `int4range` — `integer` 的范围
- `int8range` — `bigint` 的范围
- `numrange` — `numeric` 的范围
- `tsrange` — `timestamp without time zone` 的范围
- `tstzrange` — `timestamp with time zone` 的范围
- `daterange` — `date` 的范围

此外，还可以定义自己的范围类型；参见CREATE TYPE了解更多信息。

8.16.2. 示例

```
CREATE TABLE reservation (room int, during tsrange);
INSERT INTO reservation VALUES
    (1108, '[2010-01-01 14:30, 2010-01-01 15:30)');

-- 包含关系
SELECT int4range(10, 20) @> 3;

-- 重叠
SELECT numrange(11.1, 22.2) && numrange(20.0, 30.0);

-- 提取上界
SELECT upper(int8range(15, 25));

-- 计算交集
SELECT int4range(10, 20) * int4range(15, 25);

-- 范围是否为空?
SELECT isempty(numrange(1, 5));
```

范围类型上的操作符和函数的完整列表见表 9.44和表 9.45。

8.16.3. 包含界限与排除界限

每个非空范围都有两个界限：下界和上界。位于这两个值之间的所有点都包含在该范围内。包含界限表示边界点本身也包含在范围内，而排除界限则表示边界点不包含在范围内。

在范围的文本形式中，包含下界用“`[`”表示，排除下界用“`(`”表示。同样，包含上界用“`]`”表示，排除上界用“`)`”表示。更多细节见第 8.16.5 节。

函数`lower_inc`和`upper_inc`分别测试范围值的下界和上界是否包含在内。

8.16.4. 无限（无界）范围

范围的下界可以省略，这意味着所有小于上界的点都包含在范围内。同样，
如果省略范围的上界，则所有大于下界的点都包含在范围内。如果上下界都被省略，
则该元素类型的所有值都被认为处于该范围内。

这等价于把下界看作“负无穷”，或把上界看作“正无穷”。但请注意，
这些无限值永远不是范围元素类型的值，也永远不可能成为范围的一部分。

（因此不存在包含式无限界限这样的东西--如果你试图写一个，
它会被自动转换为排除式界限。）

另外，某些元素类型具有“infinity”的概念，但就范围类型的机制而言，
那只是又一个普通的值。例如，对于时间戳范围，`[today,]`与`[today,)`
不同--前者包括特殊的`timestamp`值`infinity`，而后者不包括。`[today,infinity]`
和`[today,infinity)`的含义与此相同。

函数`lower_inf`和`upper_inf`分别测试范围的下界和上界是否为无限。

8.16.5. 范围输入/输出

范围值的输入必须遵循下列模式之一：

```
(lower-bound,upper-bound)
(lower-bound,upper-bound]
[lower-bound,upper-bound)
[lower-bound,upper-bound]
empty
```

如前所述，圆括号或方括号指示上下界是排除还是包含。注意最后一种模式是`empty`，
它表示空范围（即不包含任何点的范围）。

*lower-bound*可以是子类型的合法输入字符串，也可以留空以表示没有下界。同样，
*upper-bound*也可以是子类型的合法输入字符串，或者留空以表示没有上界。

每个界限值都可以用"（双引号）字符括起来。如果界限值包含圆括号、方括号、逗号、
双引号或反斜线，这样做就是必须的，因为否则这些字符会被视为范围语法的一部分。
要在带引号的界限值中写入双引号或反斜线，需要在前面加一个反斜线。（另外，
在双引号括起来的界限值中，两个连续的双引号表示一个双引号字符，这与 SQL 字符串字面
量中单引号的规则类似。）或者，你也可以不使用引号，
而是用反斜线转义所有本来会被当作范围语法的字符。另外，
如果要把空字符串写成界限值，应写成""，因为什么都不写表示无限界限。

范围值前后允许有空白，
但圆括号或方括号之间的任何空白都会被视为下界或上界值的一部分。（取决于元素类型，
这些空白可能有意义，也可能没有意义。）

> **注意**
>
> 这些规则与在复合类型字面量中书写字段值的规则非常相似。更多说明见第 8.15.6 节。

示例：

```
-- 包含 3，不包含 7，并包含中间的所有点
SELECT '[3,7)'::int4range;

-- 既不包含 3，也不包含 7，但包含中间的所有点
SELECT '(3,7)'::int4range;

-- 只包含单个点 4
SELECT '[4,4]'::int4range;

-- 不包含任何点（并将被规范化为 'empty'）
SELECT '[4,4)'::int4range;
```

8.16.6. 构造范围

每种范围类型都有一个与范围类型同名的构造函数。
使用构造函数通常比书写范围字面量更方便，因为这样无需为界限值额外加引号。
构造函数接受两个或三个参数。两个参数的形式构造标准形式的范围（下界包含，
上界排除），而三个参数的形式则按第三个参数指定的界限形式构造范围。
第三个参数必须是下列字符串之一：“()”、“(]”、“[)”或者“[]”。例如：

```
-- 完整形式为：下界、上界，以及指示界限包含性/排除性的文本参数。
SELECT numrange(1.0, 14.0, '(]');

-- 如果省略第三个参数，则假定为 '[)'。
SELECT numrange(1.0, 14.0);

-- 虽然这里指定的是 '(]'，但显示时该值会转换为规范形式，因为 int8range 是离散
范围类型（见下文）。
SELECT int8range(1, 14, '(]');

-- 任一界限使用 NULL 都会使该侧无界。
SELECT numrange(NULL, 2.2);
```

8.16.7. 离散范围类型

离散范围是指其元素类型具有明确定义的“步长”，例如integer或date。在这类类型中，
如果两个元素之间不存在合法值，就可以说它们是相邻的。
这与连续范围形成对比：在连续范围中，两个给定值之间总是（或几乎总是）
可以识别出其他元素值。例如，基于numeric的范围是连续的，基于timestamp
的范围也是如此。（尽管timestamp的精度有限，因此理论上可以视为离散的，
但通常最好仍将其视为连续的，因为步长通常并不是关注点。）

理解离散范围类型的另一种方式是：对每个元素值，都有一个明确的“下一个”或“上一个”
值。知道这一点后，就可以把原先给定的元素值替换成其下一个或上一个值，
从而在范围界限的包含表示和排除表示之间相互转换。例如，在整数范围类型中，[4,8]和(
3,9)表示相同的值集合；但对基于 numeric 的范围则并非如此。

离散范围类型应当具有一个规范化函数，该函数知道元素类型所期望的步长。
规范化函数负责将范围类型中的等价值转换成完全一致的表示形式，
特别是对包含或排除界限保持一致。如果未指定规范化函数，
那么格式不同的范围即使实际上表示的是同一组值，也总会被视为不相等。

内置范围类型int4range、int8range和daterange
都使用一种规范形式：包含下界而排除上界，也就是[)。不过，
用户定义的范围类型也可以采用其他约定。

8.16.8. 定义新的范围类型

用户可以定义自己的范围类型。最常见的原因，
是希望对内置范围类型中未提供的子类型使用范围。例如，要定义一个子类型为float8
的新范围类型：

```
CREATE TYPE floatrange AS RANGE (
    subtype = float8,
    subtype_diff = float8mi
);

SELECT '[1.234, 5.678]'::floatrange;
```

因为float8没有有意义的“步长”，所以在这个示例中我们没有定义规范化函数。

定义自己的范围类型还允许你指定不同的子类型 B-树操作符类或排序规则，从而改变用于判定哪些值落入给定范围的排序顺序。

如果认为子类型的值是离散而非连续的，则CREATE TYPE命令应指定canonical函数。规范化函数接受一个输入的范围值，并必须返回一个等价的范围值，该值的界限和格式可能不同。对于表示同一组值的两个范围，例如整数范围[1, 7]和[1, 8)，规范化输出必须相同。选择哪一种表示作为规范形式并不重要，只要不同格式的两个等价值总是会被映射为同一种格式的相同值即可。除了调整界限的包含/排除格式之外，如果期望的步长大于子类型能够存储的精度，规范化函数还可能对界限值进行舍入。例如，基于timestamp的范围类型可以定义为步长为一小时，在这种情况下，规范化函数需要把不是一小时整数倍的界限值舍入，或者直接抛出错误。

此外，任何打算与 GiST 索引一起使用的范围类型都应定义子类型差值函数，即subtype_diff函数。（没有subtype_diff时 GiST 索引仍可工作，但效率很可能明显低于提供了差值函数的情况。）子类型差值函数接受两个子类型输入值，并返回它们的差值（即*X*减*Y*），结果表示为一个float8值。在上面的示例中，可以使用常规float8减法操作符的底层函数；；但对于其他子类型，则需要某种类型转换。此外，可能还需要仔细考虑如何把差异表示为数字。尽可能地，subtype_diff函数应与所选操作符类和排序规则所隐含的排序顺序一致；也就是说，只要其第一个参数按该排序顺序大于第二个参数，它的结果就应该为正。

关于创建范围类型的更多信息，见CREATE TYPE。

8.16.9. 索引

可以为范围类型的表列创建 GiST 索引。例如：

```
CREATE INDEX reservation_idx ON reservation USING gist (during);
```

GiST 索引可以加速涉及以下范围操作符的查询：=, &&, <@, @>, <<, >>, -|-, &<和&>（参见表 9.44了解更多信息）。

此外，也可以为范围类型的表列创建 B-树和哈希索引。对于这些索引类型，基本上唯一有用的范围操作就是等值。系统为范围值定义了对应<和>操作符的 B-树排序顺序，但这种顺序相当任意，在现实中通常并没有什么用处。范围类型的 B-树和哈希支持主要是为了允许在查询内部进行排序和哈希，而不是用于创建实际的索引。

8.16.10. 范围上的约束

虽然UNIQUE是标量值的一种自然约束，但它通常并不适用于范围类型。相反，排他约束往往更合适（见CREATE TABLE ... CONSTRAINT ... EXCLUDE）。排他约束允许在范围类型上指定诸如“不重叠”之类的约束。例如：

```
CREATE TABLE reservation (
    during tsrange,
    EXCLUDE USING gist (during WITH &&)
);
```

该约束会阻止表中同时存在任何重叠值：

```
INSERT INTO reservation VALUES
    ('[2010-01-01 11:30, 2010-01-01 15:00)');
INSERT 0 1

INSERT INTO reservation VALUES
    ('[2010-01-01 14:45, 2010-01-01 15:45)');
ERROR:  conflicting key value violates exclusion constraint
 "reservation_during_excl"
```

```
DETAIL:  Key (during)=(["2010-01-01 14:45:00","2010-01-01 15:45:
00")) conflicts
with existing key (during)=(["2010-01-01 11:30:00","2010-01-01 15:
00:00")).
```

你可以使用btree_gist扩展在普通标量数据类型上定义排他约束，
然后再将其与范围上的排他约束结合起来，以获得最大的灵活性。例如，在安装了btree_gist之后，下面的约束仅在会议室编号相等时才会拒绝重叠的范围：

```
CREATE EXTENSION btree_gist;
CREATE TABLE room_reservation (
    room text,
    during tsrange,
    EXCLUDE USING gist (room WITH =, during WITH &&)
);

INSERT INTO room_reservation VALUES
    ('123A', '[2010-01-01 14:00, 2010-01-01 15:00)');
INSERT 0 1

INSERT INTO room_reservation VALUES
    ('123A', '[2010-01-01 14:30, 2010-01-01 15:30)');
ERROR:  conflicting key value violates exclusion constraint "room_
reservation_room_during_excl"
DETAIL:  Key (room, during)=(123A, ["2010-01-01 14:30:00","2010-01
-01 15:30:00")) conflicts
with existing key (room, during)=(123A, ["2010-01-01 14:00:00",
"2010-01-01 15:00:00")).

INSERT INTO room_reservation VALUES
    ('123B', '[2010-01-01 14:30, 2010-01-01 15:30)');
INSERT 0 1
```

8.17. JSON 类型

`json` 数据类型可用于存储 RFC 4627[1] 规定的 JSON（JavaScript Object Notation）数据。此类数据也可以存储为 `text`，但 `json` 数据类型的优点是能检查每个存储的值是否为有效的 JSON 值。此外还有相关的支持函数可用；参见第 9.15 节。

PostgreSQL 的每个数据库只允许一种服务器编码。因此，除非服务器编码为 UTF-8，否则 JSON 不可能严格遵循规范。试图直接包含无法在服务器编码中表示的字符将会失败；反过来，可以在服务器编码中表示但不能在 UTF-8 中表示的字符则是允许的。无论服务器编码如何，都允许 `\uXXXX` 转义，并且只检查其语法正确性。

8.18. 对象标识符类型

对象标识符（OID）在PostgreSQL内部被用作若干系统表的主键。除非建表时指定`WITH OIDS`，或启用配置变量default_with_oids，否则不会向用户创建的表添加 OID。类型`oid`表示一个对象标识符。还有若干`oid`的别名类型：`regproc`、`regprocedure`、`regoper`、`regoperator`、`regclass`、`regtype`、`regconfig`和`regdictionary`。表 8.23给出了概要说明。

`oid`类型目前实现为无符号 4 字节整数。因此，在大型数据库中，它不足以提供数据库范围内的唯一性，甚至在大型的单个表中也无法保证唯一性。因此，不推荐把用户创建表的 OID 列用作主键。OID 最好仅用于引用系统表。

[1] http://www.ietf.org/rfc/rfc4627.txt

oid 类型本身除比较之外几乎没有其他操作。不过，它可以转换为整数，然后再用标准整数操作符进行处理。（这样做时要注意有符号与无符号可能造成的混淆。）

OID 别名类型除了专门的输入和输出例程外，没有自己的操作。这些例程能够接受并显示系统对象的符号名称，而不是 oid 类型所使用的原始数值。别名类型简化了对象 OID 值的查找。例如，要查看与表 mytable 相关的 pg_attribute 行，可以写成：

```
SELECT * FROM pg_attribute WHERE attrelid = 'mytable'::regclass;
```

而不是：

```
SELECT * FROM pg_attribute
  WHERE attrelid = (SELECT oid FROM pg_class WHERE relname =
 'mytable');
```

虽然这样看起来也不算太糟，但其实仍然过于简化。如果不同模式中有多个名为 mytable 的表，就必须写一个复杂得多的子查询来选出正确的 OID。regclass 的输入转换器会按照模式路径设置来处理表查找，因此能够自动完成“正确的事情”。同样，把表的 OID 转换成 regclass，也很适合用来以符号形式显示数值 OID。

表 8.23. 对象标识符类型

名字	引用	描述	值示例
oid	任意	数字形式的对象标识符	564182
regproc	pg_proc	函数名称	sum
regprocedure	pg_proc	函数与参数类型	sum(int4)
regoper	pg_operator	操作符名称	+
regoperator	pg_operator	带参数类型的操作符	*(integer,integer)或-(NONE,integer)
regclass	pg_class	关系名称	pg_type
regtype	pg_type	数据类型名称	integer
regconfig	pg_ts_config	文本搜索配置	english
regdictionary	pg_ts_dict	文本搜索字典	simple

所有 OID 别名类型都接受模式限定名称；如果不加限定就无法在当前搜索路径中找到对象，输出时也会显示模式限定名称。regproc和regoper别名类型只接受唯一的（未重载的）输入名称，因此用途有限；对于大多数用途，regprocedure或regoperator更合适。对于regoperator，通过将未使用的操作数写为NONE来标识一元操作符。

大多数 OID 别名类型还有一个附加特性，就是会创建依赖关系。如果这些类型的常量出现在存储的表达式中（例如列默认表达式或视图），它就会对被引用对象建立依赖。例如，如果某列的默认表达式是 nextval('my_seq'::regclass)，PostgreSQL 就会知道该默认表达式依赖于序列 my_seq，因此系统不允许在尚未移除该默认表达式时删除序列。

系统使用的另一种标识符类型是xid，即事务（缩写为xact）标识符。这是系统列xmin和xmax的数据类型。事务标识符是 32 位的量。

系统使用的第三种标识符类型是 cid，也就是命令标识符。这是系统列 cmin 和 cmax 的数据类型。命令标识符同样是 32 位量。

系统使用的最后一种标识符类型是 tid，也就是元组标识符（行标识符）。这是系统列 ctid 的数据类型。一个元组 ID 是一对值（块号、块内元组索引），用来标识该行在其所属表中的物理位置。

（系统列的更多说明见第 5.4 节。）

8.19. 伪类型

PostgreSQL 的类型系统中包含一些特殊用途的条目，统称为伪类型。伪类型不能作为列数据类型使用，但可以用于声明函数的参数类型或结果类型。
每一种伪类型都适用于这样的场景：函数的行为并不对应于简单地接受或返回某个特定 SQL 数据类型的值。表 8.24列出了现有的伪类型。

表 8.24. 伪类型

名字	描述
any	表示一个函数可以接受任意输入数据类型。
anyelement	表示一个函数可以接受任意数据类型（参见第 35.2.5 节）。
anyarray	表示一个函数可以接受任意数组数据类型（参见第 35.2.5 节）
anynonarray	表示一个函数可以接受任意非数组数据类型（参见第 35.2.5 节）。
anyenum	表示一个函数可以接受任意枚举数据类型（参见第 35.2.5 节和第 8.7 节）。
anyrange	表示一个函数可以接受任意范围数据类型（参见第 35.2.5 节和第 8.16 节）。
cstring	表示一个函数接受或返回一个以空字符结尾的 C 字符串。
internal	表示一个函数接受或返回一个服务器内部数据类型。
language_handler	表示过程语言调用处理器被声明为返回 language_handler。
fdw_handler	表示外部数据包装器处理器被声明为返回 fdw_handler。
record	标识一个返回未指定行类型的函数。
trigger	触发器函数被声明为返回trigger.
void	表示一个函数不返回值。
opaque	一个已废弃的类型名称，以前用于上述所有用途。

用 C 编写的函数（无论是内置的还是动态装载的）都可以声明为接受或返回这些伪类型中的任意一种。函数作者必须自行确保，当伪类型被用作参数类型时，该函数的行为仍然安全。

用过程语言编写的函数，只有在其实现语言允许的情况下才能使用伪类型。目前，
各过程语言都禁止把伪类型用作参数类型，而只允许把void和record
用作结果类型（如果函数被用作触发器，则也允许trigger）。
有些语言还支持使用anyarray、anyelement、anyenum和anynonarray实现多态函数。

internal 伪类型用于声明那些只应由数据库系统内部调用、而不应被 SQL 查询直接调用的函数。如果一个函数至少有一个 internal 类型参数，那么它就不能从 SQL 中调用。为了保持这一限制的类型安全性，必须遵守这样一条编码规则：除非函数至少有一个 internal 类型参数，否则不要声明任何返回 internal 的函数。

第 9 章 函数和操作符

PostgreSQL为内置数据类型提供了大量函数和操作符。用户也可以按第 V 部分中的说明定义自己的函数和操作符。psql命令`\df`和`\do`可分别用于列出所有可用的函数和操作符。

如果你关心可移植性，请注意，本章描述的大多数函数和操作符，
除了最简单的算术和比较操作符以及少数明确标记的函数之外，都不是SQL标准规定的。
不过，其中一些扩展功能也存在于其他SQL数据库管理系统中，而且在很多情况下，
各实现之间的这类功能是兼容且一致的。本章也并非详尽无遗；
手册的相关章节中还有其他函数。

9.1. 逻辑操作符

常用的逻辑操作符有：

```
AND
OR
NOT
```

SQL使用三值的逻辑系统，包括真、假和`null`，它表示“未知”。观察下面的真值表：

a	*b*	*a* AND *b*	*a* OR *b*
TRUE	TRUE	TRUE	TRUE
TRUE	FALSE	FALSE	TRUE
TRUE	NULL	NULL	TRUE
FALSE	FALSE	FALSE	FALSE
FALSE	NULL	FALSE	NULL
NULL	NULL	NULL	NULL

a	NOT *a*
TRUE	FALSE
FALSE	TRUE
NULL	NULL

操作符`AND`和`OR`是可交换的，也就是说，你可以交换左右操作数而不影响结果。但请参见第 4.2.14 节获取有关子表达式计算顺序的更多信息。

9.2. 比较操作符

常见的比较操作符都可用，如表 9.1所示。

表 9.1. 比较操作符

操作符	描述
<	小于
>	大于
<=	小于或等于
>=	大于或等于
=	等于
<> 或 !=	不等于

> **注意**
>
> != 操作符在解析器阶段会转换为 <>。不可能实现行为不同的 != 和 <> 操作符。

比较操作符适用于所有相关的数据类型。所有比较操作符都是返回 boolean 类型值的二元操作符；类似 1 < 2 < 3 的表达式是无效的（因为没有用于比较布尔值和 3 的 < 操作符）。

除比较操作符之外，还可以使用特殊的BETWEEN结构：

```
a BETWEEN x AND y
```

等价于

```
a >= x AND a <= y
```

注意，BETWEEN将两个端点值都视为包含在范围内。NOT BETWEEN执行相反的比较：

```
a NOT BETWEEN x AND y
```

等价于

```
a < x OR a > y
```

BETWEEN SYMMETRIC与BETWEEN相同，但不要求AND左侧的参数小于或等于右侧的参数。如果不是这样，这两个参数会自动交换，以确保表示的范围始终非空。

当任一输入为 null 时，普通的比较操作符会得到 null（表示“未知”），而不是真或假。例如，7 = NULL得到 null，7 <> NULL也一样。如果这种行为不合适，可以使用IS [NOT] DISTINCT FROM结构：

```
a IS DISTINCT FROM b
a IS NOT DISTINCT FROM b
```

对于非 null 输入，IS DISTINCT FROM和<>操作符一样。不过，如果两个输入都为 null，它会返回假。而如果只有一个输入为 null，它会返回真。类似地，IS NOT DISTINCT FROM对于非 null 输入的行为与=相同，但是当两个输入都为 null 时它返回真，并且当只有一个输入为 null 时返回假。因此，这些结构实际上把 null 当作一种普通数据值，而不是“未知”。

要检查一个值是否为 null，使用下面的结构：

```
expression IS NULL
expression IS NOT NULL
```

或者等效但非标准的结构：

```
expression ISNULL
expression NOTNULL
```

不要写*expression* = NULL，因为NULL并不“等于”NULL。（null 值表示未知值，而我们并不知道两个未知值是否相等。）

> **提示**
>
> 有些应用可能期望表达式*expression* = NULL在*expression*求值为 null 时返回真。我们强烈建议此类应用修改为遵循 SQL 标准。但是，如果无法这样修改，那么可以使用配置变量transform_null_equals。如果将其打开，PostgreSQL会把x = NULL子句转换成x IS NULL。

如果*expression*是行值，那么当行表达式本身为 null 或其所有字段都为 null 时，`IS NULL` 为真；而当行表达式本身非 null 且其所有字段都非 null 时，`IS NOT NULL` 为真。由于这种行为，`IS NULL`和`IS NOT NULL`并不总是对行值表达式返回相反的结果；特别是，一个同时包含 null 和非 null 字段的行值表达式会对这两种测试都返回假。在某些情况下，写成*row* `IS DISTINCT FROM NULL`或者*row* `IS NOT DISTINCT FROM NULL`可能更合适，因为它们只会检查整个行值是否为 null，而不会再对行字段做额外测试。

布尔值也可以使用下列结构进行测试：

```
expression IS TRUE
expression IS NOT TRUE
expression IS FALSE
expression IS NOT FALSE
expression IS UNKNOWN
expression IS NOT UNKNOWN
```

这些谓词总是返回真或假，从不返回空值，即使操作数为 null 也是如此。null 输入被当作逻辑值“未知”。请注意，`IS UNKNOWN`和`IS NOT UNKNOWN`实际上分别等同于`IS NULL`和`IS NOT NULL`，只是输入表达式必须是布尔类型。

9.3. 数学函数和操作符

PostgreSQL为很多类型提供了数学操作符。对于那些没有标准数学惯例的类型（如日期/时间类型），我们将在后续小节中描述实际的行为。

表 9.2列出了可用的数学操作符。

表 9.2. 数学操作符

操作符	描述	示例	结果
+	加法	2 + 3	5
-	减法	2 - 3	-1
*	乘法	2 * 3	6
/	除法（整数除法会截断结果）	4 / 2	2
%	取模（余数）	5 % 4	1
^	求幂（从左向右结合）	2.0 ^ 3.0	8
|/	平方根	|/ 25.0	5
||/	立方根	||/ 27.0	3
!	阶乘	5 !	120
!!	阶乘（前缀操作符）	!! 5	120
@	绝对值	@ -5.0	5
&	按位与	91 & 15	11
|	按位或	32 | 3	35
#	按位异或	17 # 5	20
~	按位非	~1	-2
<<	按位左移	1 << 4	16
>>	按位右移	8 >> 2	2

按位操作符仅适用于整数数据类型，也可用于位串类型 `bit` 和 `bit varying`，如表 9.11所示。

表 9.3列出了可用的数学函数。表中的 `dp` 表示 `double precision`。许多函数提供了参数类型不同的多种形式。除非另有说明，函数的每种形式都返回与其参数相同的数据类型。处理

`double precision` 数据的函数大多基于主机系统的 C 库实现；因此，其精度和边界情况下的行为可能因主机系统而异。

表 9.3. 数学函数

函数	`abs(x)`
返回类型	（与输入相同）
描述	绝对值
示例	`abs(-17.4)`
结果	`17.4`
函数	`cbrt(dp)`
返回类型	`dp`
描述	立方根
示例	`cbrt(27.0)`
结果	`3`
函数	`ceil(dp or numeric)`
返回类型	（与输入相同）
描述	大于或等于参数的最小整数
示例	`ceil(-42.8)`
结果	`-42`
函数	`ceiling(dp or numeric)`
返回类型	（与输入相同）
描述	大于或等于参数的最小整数（与 `ceil` 相同）
示例	`ceiling(-95.3)`
结果	`-95`
函数	`degrees(dp)`
返回类型	`dp`
描述	将弧度转换为角度
示例	`degrees(0.5)`
结果	`28.6478897565412`
函数	`div(y numeric, x numeric)`
返回类型	`numeric`
描述	y/x 的整数商
示例	`div(9,4)`
结果	`2`
函数	`exp(dp or numeric)`
返回类型	（与输入相同）
描述	指数函数
示例	`exp(1.0)`
结果	`2.71828182845905`

函数	floor(dp or numeric)
返回类型	（与输入相同）
描述	小于或等于参数的最大整数
示例	floor(-42.8)
结果	-43
函数	ln(dp or numeric)
返回类型	（与输入相同）
描述	自然对数
示例	ln(2.0)
结果	0.693147180559945
函数	log(dp or numeric)
返回类型	（与输入相同）
描述	以 10 为底的对数
示例	log(100.0)
结果	2
函数	log(*b* numeric, *x* numeric)
返回类型	numeric
描述	以 *b* 为底的对数
示例	log(2.0, 64.0)
结果	6.0000000000
函数	mod(*y*, *x*)
返回类型	（与参数类型相同）
描述	*y*/*x* 的余数
示例	mod(9,4)
结果	1
函数	pi()
返回类型	dp
描述	常数“π”
示例	pi()
结果	3.14159265358979
函数	power(*a* dp, *b* dp)
返回类型	dp
描述	*a*的*b*次幂
示例	power(9.0, 3.0)
结果	729
函数	power(*a* numeric, *b* numeric)
返回类型	numeric

描述	*a*的*b*次幂
示例	power(9.0, 3.0)
结果	729
函数	radians(dp)
返回类型	dp
描述	将角度转换为弧度
示例	radians(45.0)
结果	0.785398163397448
函数	round(dp or numeric)
返回类型	（与输入相同）
描述	舍入到最接近的整数
示例	round(42.4)
结果	42
函数	round(*v* numeric, *s* int)
返回类型	numeric
描述	舍入到 *s* 位小数
示例	round(42.4382, 2)
结果	42.44
函数	sign(dp or numeric)
返回类型	（与输入相同）
描述	参数的符号（-1、0、+1）
示例	sign(-8.4)
结果	-1
函数	sqrt(dp or numeric)
返回类型	（与输入相同）
描述	平方根
示例	sqrt(2.0)
结果	1.4142135623731
函数	trunc(dp or numeric)
返回类型	（与输入相同）
描述	向零截断
示例	trunc(42.8)
结果	42
函数	trunc(*v* numeric, *s* int)
返回类型	numeric
描述	截断到 *s* 位小数
示例	trunc(42.4382, 2)

结果	42.43
函数	width_bucket(*op* numeric, *b1* numeric, *b2* numeric, *count* int)
返回类型	int
描述	返回 *operand* 在等宽直方图（在 *b1* 到 *b2* 的范围内有 *count* 个桶）中所属的桶编号
示例	width_bucket(5.35, 0.024, 10.06, 5)
结果	3
函数	width_bucket(*op* dp, *b1* dp, *b2* dp, *count* int)
返回类型	int
描述	返回 *operand* 在等宽直方图（在 *b1* 到 *b2* 的范围内有 *count* 个桶）中所属的桶编号
示例	width_bucket(5.35, 0.024, 10.06, 5)
结果	3

表 9.4展示了用于产生随机数的函数。

表 9.4. 随机函数

函数	返回类型	描述
random()	dp	范围 0.0 <= x < 1.0 内的随机值
setseed(dp)	void	为后续 random() 调用设置种子（值在 -1.0 与 1.0 之间，包含端点）

random() 返回值的特性取决于系统实现。它不适用于加密应用；替代方案见pgcrypto模块。

最后，表 9.5列出了可用的三角函数。所有三角函数的参数和返回值都是 double precision 类型。三角函数的参数以弧度表示。反三角函数的返回值以弧度表示。可使用上面的单位换算函数radians()和degrees()。

表 9.5. 三角函数

函数	描述
acos(*x*)	反余弦
asin(*x*)	反正弦
atan(*x*)	反正切
atan2(*y*, *x*)	*y*/*x* 的反正切
cos(*x*)	余弦
cot(*x*)	余切
sin(*x*)	正弦
tan(*x*)	正切

9.4. 字符串函数和操作符

本节描述用于检查和操作字符串值的函数和操作符。这里的字符串包括 character、character varying 和 text 类型的值。除非另有说明，下面列出的所有函数都适用于这些类型，但使用 character 类型时，应注意自动填充空格可能带来的影响。一些函数也为位串类型提供了原生实现。

SQL定义了一些字符串函数，它们使用关键字，而不是逗号来分隔参数。详情请见表 9.6，PostgreSQL也提供了这些函数使用正常函数调用语法的版本（见表 9.7）。

> **注意**
>
> 在 PostgreSQL 8.3 之前，由于某些非字符串数据类型到 `text` 的隐式强制转换，这些函数也会悄然接受那些类型的值。由于这些强制转换经常产生出人意料的行为，它们已被移除。不过，字符串串接操作符（`||`）仍接受非字符串输入，只要至少一个输入属于字符串类型，如表 9.6所示。对于其他情况，如果需要重现以前的行为，可添加到 `text` 的显式强制转换。

表 9.6. SQL字符串函数和操作符

函数	*string* `||` *string*
返回类型	`text`
描述	字符串串接
示例	`'Post' || 'greSQL'`
结果	`PostgreSQL`
函数	*string* `||` *non-string* 或 *non-string* `||` *string*
返回类型	`text`
描述	一个输入为非字符串的字符串串接
示例	`'Value: ' || 42`
结果	`Value: 42`
函数	`bit_length(`*string*`)`
返回类型	`int`
描述	字符串中的位数
示例	`bit_length('jose')`
结果	`32`
函数	`char_length(`*string*`)` 或 `character_length(`*string*`)`
返回类型	`int`
描述	字符串中的字符数
示例	`char_length('jose')`
结果	`4`
函数	`lower(`*string*`)`
返回类型	`text`
描述	将字符串转换为小写
示例	`lower('TOM')`
结果	`tom`
函数	`octet_length(`*string*`)`
返回类型	`int`
描述	字符串中的字节数

示例	octet_length('jose')
结果	4
函数	overlay(*string* placing *string* from int [for int])
返回类型	text
描述	替换子字符串
示例	overlay('Txxxxas' placing 'hom' from 2 for 4)
结果	Thomas
函数	position(*substring* in *string*)
返回类型	int
描述	指定子字符串的位置
示例	position('om' in 'Thomas')
结果	3
函数	substring(*string* [from int] [for int])
返回类型	text
描述	提取子字符串
示例	substring('Thomas' from 2 for 3)
结果	hom
函数	substring(*string* from *pattern*)
返回类型	text
描述	提取匹配 POSIX 正则表达式的子字符串。有关模式匹配的更多信息，参见第 9.7 节。
示例	substring('Thomas' from '...$')
结果	mas
函数	substring(*string* from *pattern* for *escape*)
返回类型	text
描述	提取匹配 SQL 正则表达式的子字符串。有关模式匹配的更多信息，参见第 9.7 节。
示例	substring('Thomas' from '%#"o_a#"_' for '#')
结果	oma
函数	trim([leading | trailing | both] [*characters*] from *string*)
返回类型	text
描述	从 *string* 的开头、结尾或两端（默认为 both）移除仅由 *characters* 中字符（默认为空格）组成的最长字符串
示例	trim(both 'x' from 'xTomxx')
结果	Tom
函数	upper(*string*)
返回类型	text
描述	将字符串转换为大写
示例	upper('tom')
结果	TOM

还有其他字符串操作函数，列在表 9.7中。其中一些用于内部实现表 9.6中列出的 SQL 标准字符串函数。

表 9.7. 其他字符串函数

函数	`ascii(string)`
返回类型	`int`
描述	参数首字符的 ASCII 码。对于 UTF8，返回该字符的 Unicode 码点。对于其他多字节编码，参数必须是 ASCII 字符。
示例	`ascii('x')`
结果	`120`
函数	`btrim(string text [, characters text])`
返回类型	`text`
描述	从 *string* 的开头和结尾移除仅由 *characters* 中字符（默认为空格）组成的最长字符串
示例	`btrim('xyxtrimyyx', 'xy')`
结果	`trim`
函数	`chr(int)`
返回类型	`text`
描述	指定编码值对应的字符。对于 UTF8，参数被视为 Unicode 码点。对于其他多字节编码，参数必须指定一个 ASCII 字符。不允许 NULL（0）字符，因为文本数据类型无法存储这样的字节。
示例	`chr(65)`
结果	`A`
函数	`concat(str "any" [, str "any" [, ...] ])`
返回类型	`text`
描述	串接所有参数的文本表示。忽略 NULL 参数。
示例	`concat('abcde', 2, NULL, 22)`
结果	`abcde222`
函数	`concat_ws(sep text, str "any" [, str "any" [, ...] ])`
返回类型	`text`
描述	使用分隔符串接除第一个参数以外的所有参数。第一个参数用作分隔字符串。忽略 NULL 参数。
示例	`concat_ws(',', 'abcde', 2, NULL, 22)`
结果	`abcde,2,22`
函数	`convert(string bytea, src_encoding name, dest_encoding name)`
返回类型	`bytea`
描述	将字符串转换为 *dest_encoding*。原编码由 *src_encoding* 指定。*string* 必须是该编码下的有效字符串。可以用 `CREATE CONVERSION` 定义转换。此外，也有一些预定义的转换。可用的转换见表 9.8。
示例	`convert('text_in_utf8', 'UTF8', 'LATIN1')`
结果	以 Latin-1 编码（ISO 8859-1）表示的 `text_in_utf8`

函数	convert_from(*string* bytea, *src_encoding* name)
返回类型	text
描述	将字符串转换为数据库编码。原编码由 *src_encoding* 指定。*string* 必须是该编码下的有效字符串。
示例	convert_from('text_in_utf8', 'UTF8')
结果	以当前数据库编码表示的 text_in_utf8
函数	convert_to(*string* text, *dest_encoding* name)
返回类型	bytea
描述	将字符串转换为 *dest_encoding*。
示例	convert_to('some text', 'UTF8')
结果	以 UTF8 编码表示的 some text
函数	decode(*string* text, *format* text)
返回类型	bytea
描述	从 *string* 中的文本表示解码二进制数据。*format* 的选项与 encode 相同。
示例	decode('MTIzAAE=', 'base64')
结果	\x3132330001
函数	encode(*data* bytea, *format* text)
返回类型	text
描述	将二进制数据编码为文本表示。支持的格式为：base64、hex、escape。escape 将零字节和最高位为 1 的字节转换为八进制序列（*nnn*），并将反斜杠双写。
示例	encode(E'123\\000\\001', 'base64')
结果	MTIzAAE=
函数	format(*formatstr* text [, *formatarg* "any" [, ...]])
返回类型	text
描述	格式化字符串。此函数类似于 C 函数 sprintf，但只识别下列转换说明：%s将对应参数作为字符串插入；%I将其参数转义为 SQL 标识符；%L将其参数转义为 SQL 字面量；%%输出一个字面 %。转换可以通过在转换说明符前加上*n*$来引用显式的参数位置，其中*n*是参数位置。另见例 39.1。
示例	format('Hello %s, %1$s', 'World')
结果	Hello World, World
函数	initcap(*string*)
返回类型	text
描述	将每个单词的首字母转换为大写，其余字母转换为小写。单词是由非字母数字字符分隔的字母数字字符序列。
示例	initcap('hi THOMAS')
结果	Hi Thomas
函数	left(*str* text, *n* int)
返回类型	text
描述	返回字符串中的前 *n* 个字符。当 *n* 为负数时，返回除最后 |*n*| 个字符以外的所有字符。

示例	`left('abcde', 2)`
结果	`ab`
函数	`length(string)`
返回类型	`int`
描述	*`string`* 中的字符数
示例	`length('jose')`
结果	`4`
函数	`length(string bytea, encoding name )`
返回类型	`int`
描述	采用给定 *`encoding`* 的 *`string`* 中的字符数。*`string`* 必须是该编码下的有效字符串。
示例	`length('jose', 'UTF8')`
结果	`4`
函数	`lpad(string text, length int [, fill text])`
返回类型	`text`
描述	在 *`string`* 前面添加字符 *`fill`*（默认为空格），将其填充到长度 *`length`*。如果 *`string`* 已经长于 *`length`*，则从右侧截断。
示例	`lpad('hi', 5, 'xy')`
结果	`xyxhi`
函数	`ltrim(string text [, characters text])`
返回类型	`text`
描述	从 *`string`* 的开头移除仅由 *`characters`* 中字符（默认为空格）组成的最长字符串
示例	`ltrim('zzzytrim', 'xyz')`
结果	`test`
函数	`md5(string)`
返回类型	`text`
描述	计算 *`string`* 的 MD5 hash，并以十六进制返回结果
示例	`md5('abc')`
结果	`900150983cd24fb0 d6963f7d28e17f72`
函数	`pg_client_encoding()`
返回类型	`name`
描述	当前客户端编码名称
示例	`pg_client_encoding()`
结果	`SQL_ASCII`
函数	`quote_ident(string text)`
返回类型	`text`
描述	为给定字符串添加适当的引号并返回，使其可用作 SQL 语句字符串中的标识符。仅在必要时添加引号（即字符串包含不能用于标识符的字符，或会被大小写折叠时）。内嵌引号会适当地双写。另见例 39.1。

示例	`quote_ident('Foo bar')`
结果	`"Foo bar"`
函数	`quote_literal(string text)`
返回类型	`text`
描述	为给定字符串添加适当的引号并返回，使其可用作 SQL 语句字符串中的字符串字面量。内嵌单引号和反斜杠会被适当地双写。注意，`quote_literal` 在输入为 null 时返回 null；如果参数可能为 null，`quote_nullable` 通常更合适。另见例 39.1。
示例	`quote_literal(E'O\'Reilly')`
结果	`'O''Reilly'`
函数	`quote_literal(value anyelement)`
返回类型	`text`
描述	将给定值强制转换为文本，然后作为字面量加引号。内嵌单引号和反斜杠会被适当地双写。
示例	`quote_literal(42.5)`
结果	`'42.5'`
函数	`quote_nullable(string text)`
返回类型	`text`
描述	为给定字符串添加适当的引号并返回，使其可用作 SQL 语句字符串中的字符串字面量；如果参数为 null，则返回 `NULL`。内嵌单引号和反斜杠会被适当地双写。另见例 39.1。
示例	`quote_nullable(NULL)`
结果	`NULL`
函数	`quote_nullable(value anyelement)`
返回类型	`text`
描述	将给定值强制转换为文本，然后作为字面量加引号；如果参数为 null，则返回 `NULL`。内嵌单引号和反斜杠会被适当地双写。
示例	`quote_nullable(42.5)`
结果	`'42.5'`
函数	`regexp_matches(string text, pattern text [, flags text])`
返回类型	`setof text[]`
描述	返回 POSIX 正则表达式与 *string* 匹配所得到的所有捕获子串。更多信息参见第 9.7.3 节。
示例	`regexp_matches('foobarbequebaz', '(bar)(beque)')`
结果	`{bar,beque}`
函数	`regexp_replace(string text, pattern text, replacement text [, flags text])`
返回类型	`text`
描述	替换匹配 POSIX 正则表达式的子字符串。更多信息参见第 9.7.3 节。
示例	`regexp_replace('Thomas', '.[mN]a.', 'M')`
结果	`ThM`

函数	`regexp_split_to_array(string text, pattern text [, flags text ])`
返回类型	`text[]`
描述	使用 POSIX 正则表达式作为分隔符拆分 *string*。更多信息参见第 9.7.3 节。
示例	`regexp_split_to_array('hello world', '\s+')`
结果	`{hello,world}`
函数	`regexp_split_to_table(string text, pattern text [, flags text])`
返回类型	`setof text`
描述	使用 POSIX 正则表达式作为分隔符拆分 *string*。更多信息参见第 9.7.3 节。
示例	`regexp_split_to_table('hello world', '\s+')`

结果	`hello`
	`world`
	（2 行）

函数	`repeat(string text, number int)`
返回类型	`text`
描述	将 *string* 重复指定的 *number* 次
示例	`repeat('Pg', 4)`
结果	`PgPgPgPg`
函数	`replace(string text, from text, to text)`
返回类型	`text`
描述	将 *string* 中每次出现的子字符串 *from* 替换为子字符串 *to*
示例	`replace('abcdefabcdef', 'cd', 'XX')`
结果	`abXXefabXXef`
函数	`reverse(str)`
返回类型	`text`
描述	返回反转后的字符串。
示例	`reverse('abcde')`
结果	`edcba`
函数	`right(str text, n int)`
返回类型	`text`
描述	返回字符串中的最后 *n* 个字符。当 *n* 为负数时，返回除前 |*n*| 个字符以外的所有字符。
示例	`right('abcde', 2)`
结果	`de`
函数	`rpad(string text, length int [, fill text])`
返回类型	`text`
描述	在 *string* 后面追加字符 *fill*（默认为空格），将其填充到长度 *length*。如果 *string* 已经长于 *length*，则截断它。

示例	rpad('hi', 5, 'xy')
结果	hixyx
函数	rtrim(*string* text [, *characters* text])
返回类型	text
描述	从 *string* 的结尾移除仅由 *characters* 中字符（默认为空格）组成的最长字符串
示例	rtrim('testim', 'xyz')
结果	test
函数	split_part(*string* text, *delimiter* text, *field* int)
返回类型	text
描述	按 *delimiter* 拆分 *string*，并返回指定字段（从一开始计数）
示例	split_part('abc~@~def~@~ghi', '~@~', 2)
结果	def
函数	strpos(*string*, *substring*)
返回类型	int
描述	指定子字符串的位置（与 position(*substring* in *string*) 相同，但注意参数顺序相反）
示例	strpos('high', 'ig')
结果	2
函数	substr(*string*, *from* [, *count*])
返回类型	text
描述	提取子字符串（与 substring(*string* from *from* for *count*) 相同）
示例	substr('alphabet', 3, 2)
结果	ph
函数	to_ascii(*string* text [, *encoding* text])
返回类型	text
描述	将 *string* 从其他编码转换为 ASCII（只支持从 LATIN1、LATIN2、LATIN9 和 WIN1250 编码转换）
示例	to_ascii('Karel')
结果	Karel
函数	to_hex(*number* int or bigint)
返回类型	text
描述	将 *number* 转换为等价的十六进制表示
示例	to_hex(2147483647)
结果	7fffffff
函数	translate(*string* text, *from* text, *to* text)
返回类型	text
描述	将 *string* 中与 *from* 集合中某字符匹配的每个字符替换为 *to* 集合中的对应字符。如果 *from* 比 *to* 长，则移除输入中出现的 *from* 中的多余字符。

示例	`translate('12345', '143', 'ax')`
结果	`a2x5`

另见第 9.20 节中的聚合函数 `string_agg`。

表 9.8. 内置转换

转换名称 [a]	源编码	目标编码
ascii_to_mic	SQL_ASCII	MULE_INTERNAL
ascii_to_utf8	SQL_ASCII	UTF8
big5_to_euc_tw	BIG5	EUC_TW
big5_to_mic	BIG5	MULE_INTERNAL
big5_to_utf8	BIG5	UTF8
euc_cn_to_mic	EUC_CN	MULE_INTERNAL
euc_cn_to_utf8	EUC_CN	UTF8
euc_jp_to_mic	EUC_JP	MULE_INTERNAL
euc_jp_to_sjis	EUC_JP	SJIS
euc_jp_to_utf8	EUC_JP	UTF8
euc_kr_to_mic	EUC_KR	MULE_INTERNAL
euc_kr_to_utf8	EUC_KR	UTF8
euc_tw_to_big5	EUC_TW	BIG5
euc_tw_to_mic	EUC_TW	MULE_INTERNAL
euc_tw_to_utf8	EUC_TW	UTF8
gb18030_to_utf8	GB18030	UTF8
gbk_to_utf8	GBK	UTF8
iso_8859_10_to_utf8	LATIN6	UTF8
iso_8859_13_to_utf8	LATIN7	UTF8
iso_8859_14_to_utf8	LATIN8	UTF8
iso_8859_15_to_utf8	LATIN9	UTF8
iso_8859_16_to_utf8	LATIN10	UTF8
iso_8859_1_to_mic	LATIN1	MULE_INTERNAL
iso_8859_1_to_utf8	LATIN1	UTF8
iso_8859_2_to_mic	LATIN2	MULE_INTERNAL
iso_8859_2_to_utf8	LATIN2	UTF8
iso_8859_2_to_windows_1250	LATIN2	WIN1250
iso_8859_3_to_mic	LATIN3	MULE_INTERNAL
iso_8859_3_to_utf8	LATIN3	UTF8
iso_8859_4_to_mic	LATIN4	MULE_INTERNAL
iso_8859_4_to_utf8	LATIN4	UTF8
iso_8859_5_to_koi8_r	ISO_8859_5	KOI8R
iso_8859_5_to_mic	ISO_8859_5	MULE_INTERNAL
iso_8859_5_to_utf8	ISO_8859_5	UTF8
iso_8859_5_to_windows_1251	ISO_8859_5	WIN1251

转换名称 [a]	源编码	目标编码
iso_8859_5_to_windows_866	ISO_8859_5	WIN866
iso_8859_6_to_utf8	ISO_8859_6	UTF8
iso_8859_7_to_utf8	ISO_8859_7	UTF8
iso_8859_8_to_utf8	ISO_8859_8	UTF8
iso_8859_9_to_utf8	LATIN5	UTF8
johab_to_utf8	JOHAB	UTF8
koi8_r_to_iso_8859_5	KOI8R	ISO_8859_5
koi8_r_to_mic	KOI8R	MULE_INTERNAL
koi8_r_to_utf8	KOI8R	UTF8
koi8_r_to_windows_1251	KOI8R	WIN1251
koi8_r_to_windows_866	KOI8R	WIN866
koi8_u_to_utf8	KOI8U	UTF8
mic_to_ascii	MULE_INTERNAL	SQL_ASCII
mic_to_big5	MULE_INTERNAL	BIG5
mic_to_euc_cn	MULE_INTERNAL	EUC_CN
mic_to_euc_jp	MULE_INTERNAL	EUC_JP
mic_to_euc_kr	MULE_INTERNAL	EUC_KR
mic_to_euc_tw	MULE_INTERNAL	EUC_TW
mic_to_iso_8859_1	MULE_INTERNAL	LATIN1
mic_to_iso_8859_2	MULE_INTERNAL	LATIN2
mic_to_iso_8859_3	MULE_INTERNAL	LATIN3
mic_to_iso_8859_4	MULE_INTERNAL	LATIN4
mic_to_iso_8859_5	MULE_INTERNAL	ISO_8859_5
mic_to_koi8_r	MULE_INTERNAL	KOI8R
mic_to_sjis	MULE_INTERNAL	SJIS
mic_to_windows_1250	MULE_INTERNAL	WIN1250
mic_to_windows_1251	MULE_INTERNAL	WIN1251
mic_to_windows_866	MULE_INTERNAL	WIN866
sjis_to_euc_jp	SJIS	EUC_JP
sjis_to_mic	SJIS	MULE_INTERNAL
sjis_to_utf8	SJIS	UTF8
windows_1258_to_utf8	WIN1258	UTF8
uhc_to_utf8	UHC	UTF8
utf8_to_ascii	UTF8	SQL_ASCII
utf8_to_big5	UTF8	BIG5
utf8_to_euc_cn	UTF8	EUC_CN
utf8_to_euc_jp	UTF8	EUC_JP
utf8_to_euc_kr	UTF8	EUC_KR
utf8_to_euc_tw	UTF8	EUC_TW
utf8_to_gb18030	UTF8	GB18030
utf8_to_gbk	UTF8	GBK
utf8_to_iso_8859_1	UTF8	LATIN1

转换名称 [a]	源编码	目标编码
utf8_to_iso_8859_10	UTF8	LATIN6
utf8_to_iso_8859_13	UTF8	LATIN7
utf8_to_iso_8859_14	UTF8	LATIN8
utf8_to_iso_8859_15	UTF8	LATIN9
utf8_to_iso_8859_16	UTF8	LATIN10
utf8_to_iso_8859_2	UTF8	LATIN2
utf8_to_iso_8859_3	UTF8	LATIN3
utf8_to_iso_8859_4	UTF8	LATIN4
utf8_to_iso_8859_5	UTF8	ISO_8859_5
utf8_to_iso_8859_6	UTF8	ISO_8859_6
utf8_to_iso_8859_7	UTF8	ISO_8859_7
utf8_to_iso_8859_8	UTF8	ISO_8859_8
utf8_to_iso_8859_9	UTF8	LATIN5
utf8_to_johab	UTF8	JOHAB
utf8_to_koi8_r	UTF8	KOI8R
utf8_to_koi8_u	UTF8	KOI8U
utf8_to_sjis	UTF8	SJIS
utf8_to_windows_1258	UTF8	WIN1258
utf8_to_uhc	UTF8	UHC
utf8_to_windows_1250	UTF8	WIN1250
utf8_to_windows_1251	UTF8	WIN1251
utf8_to_windows_1252	UTF8	WIN1252
utf8_to_windows_1253	UTF8	WIN1253
utf8_to_windows_1254	UTF8	WIN1254
utf8_to_windows_1255	UTF8	WIN1255
utf8_to_windows_1256	UTF8	WIN1256
utf8_to_windows_1257	UTF8	WIN1257
utf8_to_windows_866	UTF8	WIN866
utf8_to_windows_874	UTF8	WIN874
windows_1250_to_iso_8859_2	WIN1250	LATIN2
windows_1250_to_mic	WIN1250	MULE_INTERNAL
windows_1250_to_utf8	WIN1250	UTF8
windows_1251_to_iso_8859_5	WIN1251	ISO_8859_5
windows_1251_to_koi8_r	WIN1251	KOI8R
windows_1251_to_mic	WIN1251	MULE_INTERNAL
windows_1251_to_utf8	WIN1251	UTF8
windows_1251_to_windows_866	WIN1251	WIN866
windows_1252_to_utf8	WIN1252	UTF8
windows_1256_to_utf8	WIN1256	UTF8
windows_866_to_iso_8859_5	WIN866	ISO_8859_5

转换名称 [a]	源编码	目标编码
windows_866_to_koi8_r	WIN866	KOI8R
windows_866_to_mic	WIN866	MULE_INTERNAL
windows_866_to_utf8	WIN866	UTF8
windows_866_to_windows_1251	WIN866	WIN
windows_874_to_utf8	WIN874	UTF8
euc_jis_2004_to_utf8	EUC_JIS_2004	UTF8
utf8_to_euc_jis_2004	UTF8	EUC_JIS_2004
shift_jis_2004_to_utf8	SHIFT_JIS_2004	UTF8
utf8_to_shift_jis_2004	UTF8	SHIFT_JIS_2004
euc_jis_2004_to_shift_jis_2004	EUC_JIS_2004	SHIFT_JIS_2004
shift_jis_2004_to_euc_jis_2004	SHIFT_JIS_2004	EUC_JIS_2004

[a] 转换名称遵循标准命名规则：将源编码正式名称中的所有非字母数字字符替换为下划线，再接上 _to_，然后接上按相同方式处理的目标编码名称。因此，这些名称可能与惯用的编码名称不同。

9.5. 二进制串函数和操作符

本节描述用于检查和操作 bytea 类型值的函数和操作符。

SQL定义了一些使用关键字而不是逗号来分隔参数的字符串函数。详情请见表 9.9。PostgreSQL也提供了这些函数使用常规函数调用语法的版本（参阅表 9.10）。

> **注意**
>
> 本页显示的示例结果假定服务器参数 bytea_output 设为 escape（传统的 PostgreSQL 格式）。

表 9.9. SQL二进制串函数和操作符

函数	*string* || *string*
返回类型	bytea
描述	字符串串接
示例	E'\\\\Post'::bytea || E'\\047gres\\000'::bytea
结果	\\Post'gres\000
函数	octet_length(*string*)
返回类型	int
描述	二进制串中的字节数
示例	octet_length(E'jo\\000se'::bytea)
结果	5
函数	overlay(*string* placing *string* from int [for int])
返回类型	bytea
描述	替换子字符串

示例	overlay(E'Th\\000omas'::bytea placing E'\\002\\003'::bytea from 2 for 3)
结果	T\\002\\003mas
函数	position(*substring* in *string*)
返回类型	int
描述	指定子字符串的位置
示例	position(E'\\000om'::bytea in E'Th\\000omas'::bytea)
结果	3
函数	substring(*string* [from int] [for int])
返回类型	bytea
描述	提取子字符串
示例	substring(E'Th\\000omas'::bytea from 2 for 3)
结果	h\000o
函数	trim([both] *bytes* from *string*)
返回类型	bytea
描述	从 *string* 的开头和结尾移除仅由 *bytes* 中字节组成的最长字符串
示例	trim(E'\\000'::bytea from E'\\000Tom\\000'::bytea)
结果	Tom

还有一些二进制串处理函数可以使用，在表 9.10列出。其中有一些是在内部使用，用于实现表 9.9列出的 SQL 标准串函数。

表 9.10. 其他二进制串函数

函数	btrim(*string* bytea, *bytes* bytea)
返回类型	bytea
描述	从 *string* 的开头和结尾移除仅由 *bytes* 中字节组成的最长字符串
示例	btrim(E'\\000trim\\000'::bytea, E'\\000'::bytea)
结果	trim
函数	decode(*string* text, *format* text)
返回类型	bytea
描述	从 *string* 中的文本表示解码二进制数据。*format* 的选项与 encode 相同。
示例	decode(E'123\\000456', 'escape')
结果	123\000456
函数	encode(*data* bytea, *format* text)
返回类型	text
描述	将二进制数据编码为文本表示。支持的格式为：base64、hex、escape。escape 将零字节和最高位为 1 的字节转换为八进制序列（*\nnn*），并将反斜杠双写。
示例	encode(E'123\\000456'::bytea, 'escape')
结果	123\000456

函数	get_bit(*string*, *offset*)
返回类型	int
描述	从字符串中提取位
示例	get_bit(E'Th\\000omas'::bytea, 45)
结果	1
函数	get_byte(*string*, *offset*)
返回类型	int
描述	从字符串中提取字节
示例	get_byte(E'Th\\000omas'::bytea, 4)
结果	109
函数	length(*string*)
返回类型	int
描述	二进制串的长度
示例	length('jo\000se'::bytea)
结果	5
函数	md5(*string*)
返回类型	text
描述	计算 *string* 的 MD5 hash，并以十六进制返回结果
示例	md5('Th\000omas'::bytea)
结果	8ab2d3c9689aaf18 b4958c334c82d8b1
函数	set_bit(*string*, *offset*, *newvalue*)
返回类型	bytea
描述	设置字符串中的位
示例	set_bit('Th\000omas'::bytea, 45, 0)
结果	Th\000omAs
函数	set_byte(*string*, *offset*, *newvalue*)
返回类型	bytea
描述	设置字符串中的字节
示例	set_byte('Th\000omas'::bytea, 4, 64)
结果	Th\000o@as

函数get_byte和set_byte把二进制串中的第一个字节编号为字节 0。函数get_bit和set_bit在每一个字节中从右边起计数位；例如位 0 是第一个字节的最低有效位，而位 15 是第二个字节的最高有效位。

另见第 9.20 节中的聚合函数string_agg。

9.6. 位串函数和操作符

本节描述用于检查和操作位串的函数和操作符，也就是 bit 和 bit varying 类型的值。除常用比较操作符外，还可使用表 9.11中的操作符。&、| 和 # 的位串操作数长度必须相同。移位时会保留字符串的原始长度，如示例所示。

表 9.11. 位串操作符

操作符	描述	示例	结果
||	串接	B'10001' || B'011'	10001011
&	按位与	B'10001' & B'01101'	00001
|	按位或	B'10001' | B'01101'	11101
#	按位异或	B'10001' # B'01101'	11100
~	按位非	~ B'10001'	01110
<<	按位左移	B'10001' << 3	01000
>>	按位右移	B'10001' >> 2	00100

以下 SQL 标准函数既适用于位串，也适用于字符串：length、bit_length、octet_length、position、substring、overlay。

以下函数既适用于位串，也适用于二进制串：get_bit、set_bit。处理位串时，这些函数将字符串的第一位（最左侧的位）编号为位 0。

此外，还可以在整数值与bit类型之间进行类型转换。例如：

```
44::bit(10)                    0000101100
44::bit(3)                     100
cast(-44 as bit(12))           111111010100
'1110'::bit(4)::integer        14
```

注意，仅转换为“bit”意味着转换为bit(1)，因此只会得到该整数的最低有效位。

> **注意**
>
> 在 PostgreSQL 8.0 之前，将整数转换为 bit(n) 会复制整数最左侧的 n 位，而现在复制的是最右侧的 n 位。此外，将整数转换为宽度大于该整数自身的位串时，会在左侧进行符号扩展。

9.7. 模式匹配

PostgreSQL提供了三种独立的模式匹配方法：传统的SQL LIKE操作符、较新的SIMILAR TO操作符（在 SQL:1999 中加入），以及POSIX风格的正则表达式。除了用于判断“这个字符串是否匹配这个模式？”的基本操作符外，还提供了提取或替换匹配子字符串、在匹配位置分割字符串的函数。

> **提示**
>
> 如果你的模式匹配的要求超出了这些，请考虑用 Perl 或 Tcl 写一个用户定义的函数。

> **小心**
>
> 虽然大多数正则表达式搜索都能很快完成，
> 但特意构造的正则表达式可能需要任意长的处理时间和任意多的内存。
> 接受来自恶意来源的正则表达式搜索模式时应当谨慎。如果必须这样做，
> 建议设置语句超时。
>
> 使用SIMILAR TO模式进行搜索具有同样的安全风险，因为SIMILAR TO提供了许多与POSIX风格正则表达式相同的能力。
>
> LIKE搜索比另外两种方法简单得多，因此，当模式可能来自恶意来源时，
> 使用它更安全。

9.7.1. LIKE

string LIKE *pattern* [ESCAPE *escape-character*]
string NOT LIKE *pattern* [ESCAPE *escape-character*]

如果该*string*匹配了提供的*pattern*，那么LIKE表达式返回真（和预期的一样，如果LIKE返回真，那么NOT LIKE表达式返回假，反之亦然。一个等效的表达式是NOT (*string* LIKE *pattern*)）。

如果*pattern*不包含百分号或下划线，那么该模式只表示它本身的字符串；在这种情况下，LIKE的行为就像等号操作符。*pattern*中的下划线（_）代表（匹配）任意单个字符；
百分号（%）匹配任意由零个或多个字符组成的序列。

一些示例：

```
'abc' LIKE 'abc'    true
'abc' LIKE 'a%'     true
'abc' LIKE '_b_'    true
'abc' LIKE 'c'      false
```

LIKE模式匹配总是覆盖整个字符串。因此，如果想要匹配字符串内任意位置上的一个序列，该模式就必须以百分号开头并以百分号结尾。

要匹配字面量下划线或百分号而不是把它们当作通配符，*pattern*
中相应的字符前面必须带有转义字符。默认的转义字符是反斜线，但也可以使用ESCAPE
子句选择其他转义字符。要匹配转义字符本身，请写两个转义字符。

> **注意**
>
> 如果你关掉了standard_conforming_strings，
> 你在字符串常量中写的任何反斜线都需要被双写。详见第 4.1.2.1 节。

也可以通过写 ESCAPE '' 来选择不使用转义字符。这会禁用转义机制，从而无法关闭模式中下划线和百分号的特殊含义。

可以用关键字 ILIKE 代替 LIKE，使匹配根据当前区域设置忽略大小写。这不属于 SQL 标准，而是 PostgreSQL 的扩展。

操作符~~等效于LIKE，而~~*对应ILIKE。还有 !~~和!~~*操作符分别代表NOT LIKE和NOT ILIKE。所有这些操作符都是PostgreSQL特有的。

9.7.2. SIMILAR TO正则表达式

```
string SIMILAR TO pattern [ESCAPE escape-character]
string NOT SIMILAR TO pattern [ESCAPE escape-character]
```

SIMILAR TO 操作符根据其模式是否匹配给定字符串返回真或假。它与 LIKE 类似，但按照 SQL 标准定义的正则表达式来解释模式。SQL 正则表达式是 LIKE 表示法和常见正则表达式表示法的一种奇特结合。

与LIKE类似，SIMILAR TO操作符只有在其模式匹配整个字符串时才算成功；这一点不同于普通正则表达式，后者可以匹配字符串的任意部分。与LIKE相同，SIMILAR TO也使用_和%作为通配符，分别匹配任意单个字符和任意字符串（分别类似于 POSIX 正则表达式中的.和.*）。

除了这些从LIKE借用的功能之外，SIMILAR TO支持下面这些从 POSIX 正则表达式借用的模式匹配元字符：

- |表示选择（两个候选之一）。
- *表示重复前面的项零次或更多次。
- +表示重复前面的项一次或更多次。
- ?表示重复前面的项零次或一次。
- {*m*}表示重复前面的项刚好*m*次。
- {*m*,}表示重复前面的项*m*次或更多次。
- {*m*,*n*}表示重复前面的项至少*m*次并且不超过*n*次。
- 可以使用圆括号()把多个项组合成一个逻辑项。
- 一个方括号表达式[...]声明一个字符类，就像 POSIX 正则表达式一样。

注意点号（.）不是SIMILAR TO的一个元字符。

与LIKE一样，反斜杠会禁用这些元字符的特殊含义；也可以用 ESCAPE 指定其他转义字符。

下面是一些示例：

```
'abc' SIMILAR TO 'abc'      true
'abc' SIMILAR TO 'a'        false
'abc' SIMILAR TO '%(b|d)%'  true
'abc' SIMILAR TO '(b|c)%'   false
```

带三个参数的 substring 函数，即 substring(*string* from *pattern* for *escape-character*)，用于提取匹配 SQL 正则表达式模式的子串。与 SIMILAR TO 一样，指定模式必须匹配整个数据字符串，否则函数失败并返回空值。要指出成功时应返回的模式部分，模式中必须出现两次后跟双引号（"）的转义字符。
函数返回匹配这两个标记之间的模式部分的文本。

下面是一些示例，其中使用#"界定返回字符串：

```
substring('foobar' from '%#"o_b#"%' for '#')   oob
substring('foobar' from '#"o_b#"%' for '#')    NULL
```

9.7.3. POSIX正则表达式

表 9.12列出了所有可用于 POSIX 正则表达式模式匹配的操作符。

表 9.12. 正则表达式匹配操作符

操作符	描述	示例
~	匹配正则表达式，区分大小写	'thomas' ~ '.*thomas.*'
~*	匹配正则表达式，不区分大小写	'thomas' ~* '.*Thomas.*'
!~	不匹配正则表达式，区分大小写	'thomas' !~ '.*Thomas.*'
!~*	不匹配正则表达式，不区分大小写	'thomas' !~* '.*vadim.*'

POSIX正则表达式提供了比LIKE和SIMILAR TO操作符更强大的模式匹配方式。许多 Unix 工具，例如egrep、sed或awk，都使用与这里描述的模式匹配语言相似的语言。

正则表达式是一个字符序列，是定义一组字符串（一个正则集）的简写。
如果字符串属于正则表达式描述的正则集，就称该字符串匹配此正则表达式。与LIKE一样，模式中的字符精确匹配字符串中的字符，除非该模式字符在正则表达式语言中有特殊含义 – 但正则表达式使用的特殊字符与LIKE不同。与LIKE模式不同，
正则表达式可以匹配字符串中的任意位置，除非显式将其锚定到字符串开头或末尾。

下面是一些示例：

```
'abc' ~ 'abc'    true
'abc' ~ '^a'     true
'abc' ~ '(b|d)'  true
'abc' ~ '^(b|c)' false
```

POSIX模式语言的详细描述见下文。

带两个参数的substring函数，即substring(*string* from *pattern*)，用于提取匹配 POSIX 正则表达式模式的子串。如果没有匹配，它返回空值；否则返回匹配模式的那部分文本。但是，如果模式包含圆括号，
则返回匹配第一个括号子表达式（左圆括号最先出现的那个）的文本。
如果要在表达式内部使用圆括号而不触发这个例外，可以在整个表达式外再加一对圆括号。
如果需要在想要提取的子表达式之前使用圆括号，请参见下文介绍的非捕获圆括号。

下面是一些示例：

```
substring('foobar' from 'o.b')     oob
substring('foobar' from 'o(.)b')   o
```

regexp_replace函数用新文本替换匹配 POSIX 正则表达式模式的子字符串。它的语法为regexp_replace(*source*, *pattern*, *replacement* [, *flags*])。如果没有与 *pattern* 匹配的内容，则原样返回 *source* 字符串。如果存在匹配，则返回将匹配子字符串替换为 *replacement* 字符串后的 *source* 字符串。*replacement* 字符串可以包含 *n*，其中 *n* 为 1 至 9，表示应插入与模式中第 *n* 个圆括号子表达式匹配的源子字符串；它也可以包含 \&，表示应插入与整个模式匹配的子字符串。如果需要在替换文本中放置字面的反斜杠，应写成 \\。*flags* 参数是可选的文本字符串，其中包含零个或多个改变函数行为的单字母标志。标志 i 指定不区分大小写的匹配，而标志 g 指定替换每个匹配的子字符串，而不仅是第一个。其他支持的标志在表 9.20中描述。

下面是一些示例：

```
regexp_replace('foobarbaz', 'b..', 'X')
                                   fooXbaz
regexp_replace('foobarbaz', 'b..', 'X', 'g')
                                   fooXX
regexp_replace('foobarbaz', 'b(..)', E'X\\1Y', 'g')
                                   fooXarYXazY
```

regexp_matches函数返回一个文本数组，其中包含与 POSIX 正则表达式模式匹配所得到的所有捕获子串。它的语法为 regexp_matches(*string*, *pattern* [, *flags*])。该函数可以不返回行、返回一行或多行（见下面的g标志）。如果*pattern*不匹配，该函数不返回任何行。如果模式不包含圆括号子表达式，
则返回的每一行都是包含与整个模式匹配的子字符串的单元素文本数组。
如果模式包含圆括号子表达式，该函数返回一个文本数组，其中第*n*个元素是匹配模式的第*n*个圆括号子表达式的子字符串（不包括“非捕获”括号；详情见下文）。*flags*
参数是一个可选的文本字符串，其中包含零个或多个单个字母标志，用于更改函数的行为。
标志g使该函数查找字符串中的每个匹配项，而不仅仅是第一个，
并为每个这样的匹配返回一行。其他支持的标志在表 9.20中描述。

下面是一些示例：

```
SELECT regexp_matches('foobarbequebaz', '(bar)(beque)');
 regexp_matches
----------------
 {bar,beque}
(1 row)

SELECT regexp_matches('foobarbequebazilbarfbonk', '(b[^b]+)(b[^b]+)
', 'g');
 regexp_matches
----------------
 {bar,beque}
 {bazil,barf}
(2 rows)

SELECT regexp_matches('foobarbequebaz', 'barbeque');
 regexp_matches
----------------
 {barbeque}
(1 row)
```

可以通过使用子查询强制regexp_matches()总是返回一行；
当希望返回所有行（包括不匹配的行）时，这在SELECT目标列表中特别有用：

```
SELECT col1, (SELECT regexp_matches(col2, '(bar)(beque)')) FROM
 tab;
```

regexp_split_to_table把一个 POSIX 正则表达式模式当作一个定界符来分割字符串。它的语法形式是regexp_split_to_table(*string*, *pattern* [, *flags*])。如果没有与*pattern*的匹配，该函数返回*string*。如果至少有一个匹配，对每一个匹配它都返回从上一个匹配的末尾（或者串的开头）到这次匹配开头之间的文本。
当没有更多匹配时，它返回从上一次匹配的末尾到串末尾之间的文本。*flags*
参数是一个可选的文本串，它包含零个或更多单字母标志，
这些标志可以改变该函数的行为。regexp_split_to_table能支持的标志在表 9.20中描述。

regexp_split_to_array函数的行为和regexp_split_to_table相同，不过regexp_split_to_array会把它的结果以一个text数组的形式返回。它的语法是regexp_split_to_array(*string*, *pattern* [, *flags*])。这些参数和regexp_split_to_table的相同。

下面是一些示例：

```
SELECT foo FROM regexp_split_to_table('the quick brown fox jumped
 over the lazy dog', E'\\s+') AS foo;
  foo
```

```
--------
 the
 quick
 brown
 fox
 jumped
 over
 the
 lazy
 dog
(9 rows)

SELECT regexp_split_to_array('the quick brown fox jumped over the
 lazy dog', E'\\s+');
              regexp_split_to_array
-----------------------------------------------
 {the,quick,brown,fox,jumped,over,the,lazy,dog}
(1 row)

SELECT foo FROM regexp_split_to_table('the quick brown fox', E'\
\s*') AS foo;
 foo
-----
 t
 h
 e
 q
 u
 i
 c
 k
 b
 r
 o
 w
 n
 f
 o
 x
(16 rows)
```

正如最后一个示例所示，
正则表达式分割函数会忽略出现在字符串开头或结尾或紧跟在前一个匹配项之后的零长度匹配。这与`regexp_matches`实现的严格的正则表达式匹配定义相矛盾，
但在实践中通常是最方便的行为。其他软件系统如Perl使用类似的定义。

9.7.3.1. 正则表达式细节

PostgreSQL的正则表达式是使用 Henry Spencer 写的一个包来实现的。下面的正则表达式的大部分描述都是从他的手册页中逐字拷贝过来的。

正则表达式（RE），在POSIX 1003.2 中定义，它有两种形式：扩展的RE或者是ERE（大概地说就是那些在`egrep`里的），基本的RE或者是BRE（大概地说就是那些在`ed`里的）。PostgreSQL支持两种形式，并且还实现了一些POSIX标准中没有但是在类似 Perl 或者 Tcl 这样的语言中得到广泛应用的一些扩展。使用了那些非POSIX扩展的RE叫高级RE，或者本文档里说的ARE。ARE 几乎完全是 ERE 的超集，但是 BRE 有几个符号上的不兼容（以及更多的限制）。我们首先描述 ARE 和 ERE 形式，描述那些只适用于 ARE 的特性，然后描述 BRE 的区别是什么。

> **注意**
>
> PostgreSQL最初总是假定正则表达式遵循 ARE 规则。不过，也可以像第 9.7.3.4 节中所述那样，在 RE 模式前加上一个嵌入选项，从而改用限制更多的 ERE 或 BRE 规则。这对于需要精确遵循POSIX 1003.2 规则的应用有助于保持兼容性。

一个正则表达式被定义为一个或更多分支，它们之间被 | 分隔。
只要能匹配其中一个分支的东西都能匹配正则表达式。

一个分支是零个或多个量化原子或约束，连接在一起。它匹配第一个的匹配项，
然后是第二个的匹配项，依此类推；一个空分支匹配空字符串。

一个量化原子是一个原子，后面可以跟一个量词。没有量词时，
匹配一次原子所匹配的内容；有量词时，按量词指定的次数匹配原子所匹配的内容。
原子可以是表 9.13列出的任何一种形式。可用量词及其含义见表 9.14。

一个约束匹配一个空串，但只是在满足特定条件下才匹配。
约束可以在能够使用原子的地方使用，只是它不能跟着量词。简单的约束在表 9.15里显示；
更多的约束稍后描述。

表 9.13. 正则表达式原子

原子	描述
(*re*)	（其中*re*是任意正则表达式）匹配*re*所匹配的内容，并记录该匹配，以备输出结果
(?:*re*)	同上，但不记录匹配结果（“非捕获”圆括号；仅适用于 ARE）
.	匹配任意单个字符
[*chars*]	一个方括号表达式，匹配*chars*中的任意一个（详见第 9.7.3.2 节）
k	（其中*k*既不是字母也不是数字）把该字符视为普通字符并匹配它，例如，\\匹配反斜线字符
c	其中*c*是字母或数字（后面可能还有其他字符），这是一个转义，参见第 9.7.3.3 节（仅适用于 ARE；在 ERE 和 BRE 中，它匹配*c*）
{	如果后面跟着非数字字符，则匹配左花括号{；如果后面跟着数字，则是*bound*的开头（见下文）
x	其中*x*是一个没有其它意义的单个字符，则匹配该字符

RE 不能以反斜线（\）结尾。

> **注意**
>
> 如果你关掉了standard_conforming_strings，
> 你在字符串常量中写的任何反斜线都需要被双写。详见第 4.1.2.1 节。

表 9.14. 正则表达式量词

量词	匹配
*	一个由原子的 0 次或更多次匹配组成的序列
+	一个由原子的 1 次或更多次匹配组成的序列
?	一个由原子的 0 次或 1 次匹配组成的序列

量词	匹配
{*m*}	一个由原子的正好*m*次匹配组成的序列
{*m*,}	一个由原子的*m*次或更多次匹配组成的序列
{*m*,*n*}	一个由原子的从*m*次到*n*次（包括）匹配组成的序列；*m*不能超过*n*
*?	*的非贪婪版本
+?	+的非贪婪版本
??	?的非贪婪版本
{*m*}?	{*m*}的非贪婪版本
{*m*,}?	{*m*,}的非贪婪版本
{*m*,*n*}?	{*m*,*n*}的非贪婪版本

使用{...}的形式被称作范围。一个范围内的数字*m*和*n*都是无符号十进制整数，允许的数值从 0 到 255（包含）。

非贪婪量词（仅适用于 ARE）与对应的普通（贪婪）量词匹配相同的可能内容，但优先选择最少的匹配次数，而不是最多的匹配次数。详见第 9.7.3.5 节。

> **注意**
>
> 一个量词不能紧跟在另外一个量词后面，例如**是非法的。
> 量词不能作为表达式或者子表达式的开头，也不能跟在^或者|后面。

表 9.15. 正则表达式约束

约束	描述
^	在字符串开头匹配
$	在字符串末尾匹配
(?=*re*)	正向先行断言在这样的位置匹配：存在从该位置开始且匹配*re*的子字符串（仅适用于 ARE）
(?!*re*)	负向先行断言在这样的位置匹配：不存在任何从该位置开始且匹配*re*的子字符串（仅适用于 ARE）

先行约束不能包含反向引用（参见第 9.7.3.3 节），其中的所有圆括号都视为非捕获圆括号。

9.7.3.2. 方括号表达式

方括号表达式是用[]括起来的字符列表。通常，它匹配列表中的任意单个字符（但请参见下文）。如果列表以^开头，则匹配任意不在列表剩余部分中的单个字符。如果列表中的两个字符用-分隔，则表示排序序列中这两个字符之间的完整字符范围（包含两个端点）；例如，ASCII中的[0-9]匹配任意十进制数字。两个范围共享一个端点是不合法的，例如a-c-e。范围高度依赖排序序列，因此可移植程序应避免依赖它们。

想在列表中包含字面字符]，可以让它做列表的首字符（如果使用了^，需要放在其后）。想在列表中包含字面字符-，可以让它做列表的首字符或者尾字符，或者一个范围的第二个端点。想在列表中把字面字符-当做范围的起点，把它用[.和.]包围起来，这样它就成为一个排序元素（见下文）。除了这些字符本身、一些用[的组合（见下段）以及转义（只在 ARE 中有效）以外，所有其它特殊字符在方括号表达式里都失去它们的特殊含义。特别是，在 ERE 和 BRE 规则下\不是特殊的，但在 ARE 里，它是特殊的（引入一个转义）。

在一个方括号表达式里，一个排序元素（一个字符、一个被当做一个单一字符排序的多字符序列或者一个表示上面两种情况的排序序列名称）

包含在`[.`和`.]`里面的时候表示该排序元素的字符序列。
该序列被当做该方括号列表的一个单一元素。
这允许一个包含多字符排序元素的方括号表达式去匹配多于一个字符，例如，
如果排序序列包含一个`ch`排序元素，那么 RE `[[.ch.]]*c`匹配`chchcc`的头五个字符。

> **注意**
>
> PostgreSQL当前不支持多字符排序元素。这些信息描述了将来可能有的行为。

在方括号表达式里，包围在`[=`和`=]`里的排序元素是一个等价类，
代表等效于那一个的所有排序元素的字符序列，包括它本身（如果没有其它等效排序元素，
那么就好像封装定界符是`[.`和 `.]`）。例如，如果`o`和`^`是一个等价类的成员，那么`[[=o=]]`、`[[=^=]]`和`[o^]`都是同义的。一个等价类不能是一个范围的端点。

在方括号表达式中，用 `[:` 和 `:]` 括起的字符类名表示属于该类的所有字符的列表。标准字符类名包括：`alnum`、`alpha`、`blank`、`cntrl`、`digit`、`graph`、`lower`、`print`、`punct`、`space`、`upper`、`xdigit`。它们表示 ctype 中定义的字符类。区域设置可以提供其他字符类。字符类不能用作范围的端点。

方括号表达式有两个特例：`[[:<:]]` 和 `[[:>:]]` 是约束，分别匹配单词开头和结尾的空字符串。单词定义为一个前后都没有单词字符的单词字符序列。单词字符是 `alnum` 字符（由 ctype 定义）或下划线。这是与 POSIX 1003.2 兼容但未由其规定的扩展，在打算移植到其他系统的软件中应谨慎使用。通常更适合使用下文介绍的约束转义；它们并不更标准，
但更容易键入。

9.7.3.3. 正则表达式转义

转义是以`\`开头，后面跟着一个字母或数字字符的特殊序列。
转义有好几种变体：字符输入转义、字符类简写转义、约束转义以及反向引用。在 ARE 里，如果一个`\`后面跟着一个字母或数字，但是并未组成一个合法的转义，那么它是非法的。在 ERE 中没有转义：在方括号表达式之外，一个后面跟着字母或数字字符的`\`只是表示该字符是一个普通的字符，而且在一个方括号表达式里，`\`是一个普通的字符（后者是 ERE 与 ARE 之间唯一的实际不兼容之处）。

字符输入转义便于在 RE 中指定不可打印或其他不便输入的字符，见表 9.16。

字符类简写转义用来提供一些常用的字符类简写。它们显示在表 9.17中。

约束转义是以转义形式书写的约束，在满足特定条件时匹配空字符串，见表 9.18。

反向引用（`\`*n*）匹配数字*n*指定的被前面的圆括号子表达式匹配的同一个串（参阅表 9.19）。例如，`([bc])\1`匹配`bb`或者`cc`，但是不匹配`bc`或者`cb`。RE 中子表达式必须完全在反向引用前面。子表达式以它们的左圆括号的顺序编号。非捕获圆括号并不定义子表达式。

> **注意**
>
> 请记住，把模式作为 SQL 字符串常量输入时，转义的开头`\`需要双写。例如：
>
> `'123' ~ E'^\\d{3}'` *true*

表 9.16. 正则表达式字符输入转义

转义	描述
`\a`	警告（响铃）字符，和 C 中一样
`\b`	退格，和 C 中一样

转义	描述
\B	反斜线（\）的同义词，用来减少双写反斜线
\c*X*	（其中*X*是任意字符）低序5位和*X*相同的字符，它的其他位都是零
\e	排序序列名称为ESC的字符；若不存在这样的字符，则使用八进制值为033的字符
\f	换页，和 C 中一样
\n	换行符，与 C 中相同
\r	回车，和 C 中一样
\t	水平制表符，和 C 中一样
\u*wxyz*	（其中*wxyz*正好是四个十六进制位）十六进制值为0x*wxyz*的字符
\U*stuvwxyz*	（其中*stuvwxyz*正好是八个十六进制位）十六进制值为0x*stuvwxyz*的字符
\v	垂直制表符，和 C 中一样
\x*hhh*	（其中*hhh*是十六进制位的任意序列）十六进制值为0x*hhh*的字符（一个单一字符，不管用了多少个十六进制位）
\0	值为0（空字节）的字符
xy	（其中*xy*正好是两个八进制位，并且不是一个反向引用）八进制值为0*xy*的字符
xyz	（其中*xyz*正好是三个八进制位，并且不是一个反向引用）八进制值为0*xyz*的字符

十六进制位是0-9、a-f和A-F。八进制位是0-7。

指定 ASCII 范围（0–127）之外的值的数字字符输入转义的含义取决于数据库编码。当编码是 UTF-8 时，转义值等价于 Unicode 代码点，例如 \u1234表示字符U+1234。对于其他多字节编码，字符输入转义通常只是指定该字符的字节值的串接。
如果该转义值不对应数据库编码中的任何合法字符，将不会发生错误，
但是它不会匹配任何数据。

字符输入转义总是被当作普通字符。例如，\135是 ASCII 中的]，但\135并不终止一个方括号表达式。

表 9.17. 正则表达式字符类简写转义

转义	描述
\d	[[:digit:]]
\s	[[:space:]]
\w	[[:alnum:]_]（注意包括下划线）
\D	[^[:digit:]]
\S	[^[:space:]]
\W	[^[:alnum:]_]（注意包括下划线）

在方括号表达式中，\d、\s 和 \w 会失去其外层方括号，而 \D、\S 和 \W 则是非法的。（因此，例如 [a-c\d] 等同于 [a-c[:digit:]]。此外，等同于 [a-c^[:digit:]] 的 [a-c\D] 是非法的。）

表 9.18. 正则表达式约束转义

转义	描述
\A	只在串开头匹配（与^的不同请参见第 9.7.3.5 节）

转义	描述
\m	只在一个词的开头匹配
\M	只在一个词的末尾匹配
\y	只在一个词的开头或末尾匹配
\Y	只在不属于单词开头或末尾的位置匹配
\Z	只在串的末尾匹配（与$的不同请参见第 9.7.3.5 节）

单词的定义与上文[[:<:]]和[[:>:]]的说明相同。方括号表达式中不允许使用约束转义。

表 9.19. 正则表达式反向引用

转义	描述
m	（其中*m*是一个非零数字）一个到第*m*个子表达式的反向引用
mnn	（其中*m*是一个非零数字，并且*nn*是后续的若干数字，并且十进制值*mnn*不大于此前已出现的捕获右圆括号数）一个到第*mnn*个子表达式的反向引用

> **注意**
>
> 八进制字符输入转义与反向引用之间存在固有歧义，
> 按上文提到的启发式规则解决：前导零始终表示八进制转义。单个非零数字，
> 如果后面没有其他数字，始终视为反向引用。不以零开头的多位数字序列，
> 如果前面已有相应的子表达式（即该数字在反向引用的合法范围内），
> 则视为反向引用，否则视为八进制转义。

9.7.3.4. 正则表达式元语法

除了上面描述的主要语法之外，还有几种特殊形式和杂项语法。

RE 可以以两种特殊的引导前缀之一开头。如果 RE 以***:开头，余下部分就被视为 ARE。（这在PostgreSQL中通常没有影响，因为 RE 默认被视为 ARE；但如果通过正则表达式函数的*flags*参数指定了 ERE 或 BRE 模式，它就会产生影响。）如果 RE 以***=开头，余下部分就作为按字面解释的字符串处理，所有字符都视为普通字符。

一个 ARE 可以以嵌入选项开头：一个序列(?*xyz*)（这里的*xyz*是一个或多个字母字符）声明影响剩余 RE 的选项。这些选项覆盖任何先前确定的选项 – 特别地，它们可以覆盖一个正则表达式操作符隐含的大小写敏感的行为，或者覆盖正则表达式函数的*flags*参数所指定的选项。可用的选项字母在表 9.20中显示。注意这些同样的选项字母也被用在正则表达式函数的*flags*参数中。

表 9.20. ARE 嵌入选项字母

选项	描述
b	RE的剩余部分是一个BRE
c	大小写敏感的匹配（覆盖操作符类型）
e	RE的剩余部分是一个ERE
i	大小写不敏感的匹配（见第 9.7.3.5 节）（覆盖操作符类型）
m	n的历史原因的同义词
n	换行敏感的匹配（见第 9.7.3.5 节）
p	部分换行敏感的匹配（见第 9.7.3.5 节）

选项	描述
q	RE 的剩余部分按字面（“加引号”）解释，所有字符都视为普通字符
s	非换行敏感的匹配（默认）
t	紧凑语法（默认；见下文）
w	逆部分换行敏感（“怪异”）的匹配（见第 9.7.3.5 节）
x	扩展语法（见下文）

嵌入选项从结束该序列的)处开始生效。它们只能出现在 ARE 的开头（如果存在***:引导前缀，则位于该前缀之后）。

除了通常的紧凑 RE 语法（其中所有字符都有意义）之外，还有一种扩展语法，可以通过指定嵌入的x选项来使用。在扩展语法中，RE 中的空白字符会被忽略，同样被忽略的还有#与其后的换行符（或 RE 末尾）之间的所有字符。这使得复杂的 RE 可以分段并添加注释。此基本规则有三个例外：

- 空白字符或 # 的前面若有 \，该字符就会被保留。
- 方括号表达式里的空白或者#将被保留
- 在多字符符号里面不能出现空白和注释，例如(?:

在这里，空白字符包括空格、制表符、换行符，以及属于*space*字符类的任何字符。

最后，在 ARE 里，方括号表达式外面，序列(?#*ttt*)（其中*ttt*是任意不包含一个)的文本）是一个注释，它被完全忽略。同样，这样的东西是不允许出现在多字符符号的字符中间的，例如 (?:。这种注释更像是一种历史产物而不是一种有用的设施，并且它们的使用已经被废弃；请使用扩展语法来替代。

如果指定了开头的***=引导前缀，那么这些元语法扩展都不能使用，因为这表示把用户输入作为按字面解释的字符串，而非 RE 处理。

9.7.3.5. 正则表达式匹配规则

在 RE 可以在给定串中匹配多于一个子串的情况下，RE 匹配串中最靠前的那个子串。如果 RE 可以匹配在那个位置开始的多个子串，要么是取最长的子串，要么是最短的，具体哪种，取决于 RE 是贪婪的还是非贪婪的。

一个 RE 是否贪婪取决于下面规则：

- 大多数原子以及所有约束，都没有贪婪属性（因为它们毕竟无法匹配长度不定的文本）。
- 在一个 RE 周围加上圆括号并不会改变其贪婪性。
- 带一个固定重复次数量词（{*m*}或者{*m*}?）的量化原子和原子自身具有同样的贪婪性（可能是没有）。
- 一个带其他普通的量词（包括{*m*,*n*}中*m*等于*n*的情况）的量化原子是贪婪的（首选最长匹配）。
- 一个带非贪婪量词（包括{*m*,*n*}?中*m*等于 *n*的情况）的量化原子是非贪婪的（首选最短匹配）。
- 一个分支 – 也就是说，一个没有顶级|操作符的 RE – 和它里面的第一个有贪婪属性的量化原子有着同样的贪婪性。
- 一个由|操作符连接起来的两个或者更多分支组成的 RE 总是贪婪的。

上面的规则所描述的贪婪属性不仅仅适用于独立的量化原子，而且也适用于包含量化原子的分支和整个 RE。这里的意思是，匹配是按照分支或者整个

RE 作为一个整体匹配最长或者最短的可能子串。一旦整个匹配的长度确定，那么匹配任意特定子表达式的部分就基于该子表达式的贪婪属性进行判断，在 RE 里面靠前的子表达式的优先级高于靠后的子表达式。

一个相应的示例：

```
SELECT SUBSTRING('XY1234Z', 'Y*([0-9]{1,3})');
结果：123
SELECT SUBSTRING('XY1234Z', 'Y*?([0-9]{1,3})');
结果：1
```

在第一个示例里，RE 作为整体是贪婪的，因为`Y*`是贪婪的。它可以匹配从`Y`开始的东西，并且它匹配从这个位置开始的最长的串，也就是，`Y123`。输出是这里的圆括号包围的部分，或者说是`123`。在第二个示例里，RE 总体上是一个非贪婪的 RE，因为`Y*?`是非贪婪的。它可以匹配从`Y`开始的最短的子串，也就是说`Y1`。子表达式`[0-9]{1,3}`是贪婪的，但是它不能修改总体匹配长度的决定；因此它被迫只匹配`1`。

简而言之，如果一个 RE 同时包含贪婪和非贪婪的子表达式，那么总的匹配长度要么是尽可能长，要么是尽可能短，这取决于给整个 RE 赋予的属性。给子表达式赋予的属性只影响在这个匹配里，各个子表达式相对于其他子表达式能“吃掉”多少内容。

量词`{1,1}`和`{1,1}?`可以分别用于在一个子表达式或者整个 RE 上强制贪婪或者非贪婪。当需要整个 RE 具有不同于从其元素中推导出的贪婪属性时，这很有用。例如，假设我们尝试将一个包含一些数字的字符串分隔成数字以及在它们之前和之后的部分，我们可能会尝试这样做：

```
SELECT regexp_matches('abc01234xyz', '(.*)(\d+)(.*)');
结果：{abc0123,4,xyz}
```

这不会有用：第一个`.*`是贪婪的，因此它会“吃掉”尽可能多的字符而留下`\d+`去匹配在最后一个可能位置上的最后一个数字。我们可能会通过让它变成非贪婪来修复：

```
SELECT regexp_matches('abc01234xyz', '(.*?)(\d+)(.*)');
结果：{abc,0,""}
```

这也不会有用：因为现在 RE 作为整体来说是非贪婪的，因此它会尽快结束全部的匹配。我们可以通过强制 RE 整体是贪婪的来得到我们想要的：

```
SELECT regexp_matches('abc01234xyz', '(?:(.*?)(\d+)(.*)){1,1}');
结果：{abc,01234,xyz}
```

将 RE 的整体贪婪性与各组件自身的贪婪性分开控制，为处理变长模式提供了很大的灵活性。

在决定更长或者更短的匹配时，匹配长度是以字符衡量的，而不是排序元素。一个空串会被认为比什么都不匹配长。例如：`bb*`匹配`abbbc`的中间三个字符；`(week|wee)(night|knights)`匹配`weeknights`的所有十个字符；而`(.*).*`匹配 `abc`的时候，圆括号包围的子表达式匹配所有三个字符；当`(a*)*`被拿来匹配`bc`时，整个 RE 和圆括号子表达式都匹配一个空串。

如果指定不区分大小写的匹配，其效果近似于字母表中的所有大小写差别都消失了。当存在大小写形式的字母作为普通字符出现在方括号表达式之外时，实际上会转换为包含其大小写形式的方括号表达式，例如`x`变成`[xX]`。当它出现在方括号表达式内部时，其所有大小写形式都会加入该表达式，例如`[x]`变成`[xX]`，`[^x]`变成`[^xX]`。

如果指定了换行敏感的匹配，`.`和使用`^`的方括号表达式将永远不会匹配换行字符（这样，匹配就不会跨越行，除非 RE 显式安排了跨行匹配）并且`^`和`$`除了分别匹配串开头和结尾之外，还将分别匹配换行后面和前面的空串。但是 ARE 转义`\A`和`\Z`仍然只匹配串的开头和结尾。

如果指定了部分换行敏感的匹配，那么它影响`.`和方括号表达式，这个时候和换行敏感的匹配一样，但是不影响`^`和`$`。

如果指定了逆部分换行敏感匹配，那么它影响^和$，其作用和在换行敏感的匹配里一样，但是不影响.和方括号表达式。这个并不是很有用，只是为了满足对称性而提供的。

9.7.3.6. 限制和兼容性

在这个实现里，对 RE 的长度没有特别的限制。但是，那些希望高移植性的程序应该避免使用长度超过 256 字节的 RE，因为 POSIX 兼容的实现可以拒绝接受这样的 RE。

ARE 与 POSIX ERE 实际不兼容的唯一特性是：\在方括号表达式中不会失去特殊含义。其他所有 ARE 特性所使用的语法，在 POSIX ERE 中都是非法的，或其效果未定义或未指定；引导前缀的***语法同样不属于 POSIX 的 BRE 或 ERE 语法。

许多 ARE 扩展借鉴自 Perl，但其中一些经过了整理和修改，也未实现少数 Perl 扩展。需要注意的不兼容之处包括\b、\B、不对末尾换行符作特殊处理、
取反的方括号表达式也受换行敏感匹配影响、先行约束中对圆括号及反向引用的限制，
以及采用最长或最短匹配而非首次匹配的语义。

ARE 与旧版 ERE 语法存在两点重要的不兼容，这里比较的是 7.4 之前的PostgreSQL：

- 在 ARE 中，\ 后跟字母或数字字符时，要么表示转义，要么是错误；而在以前的版本中，它只是书写该字母或数字字符的另一种方式。这应该不是什么大问题，
  因为以前的版本没有理由使用这样的序列。
- 在 ARE 中，\ 在 [] 内仍是特殊字符，因此方括号表达式内的字面 \ 必须写成 \\。

9.7.3.7. 基本正则表达式

BRE 在几个方面和 ERE 不太一样。在 BRE 中，|、+和?都是普通字符并且没有与它们功能等价的东西。范围的定界符是\{和\}，而{和}本身是普通字符。嵌套的子表达式的圆括号是\(和\)，而(和)自身是普通字符。除非在 RE 开头或者是圆括号子表达式开头，^都是一个普通字符。除非在 RE 结尾或者是圆括号子表达式的结尾，$是一个普通字符。如果*出现在 RE 开头或者是圆括号封装的子表达式开头（前面可能有^），那么它是个普通字符。最后，可以用单数字的反向引用，\<和\>分别是[[:<:]]和[[:>:]]的同义词；在 BRE 中没有其它可用的转义。

9.8. 数据类型格式化函数

PostgreSQL格式化函数提供一套强大的工具用于把各种数据类型（日期/时间、整数、浮点数、数值）
转换成格式化的字符串以及反过来从格式化的字符串转换成指定的数据类型。表 9.21列出了这些函数。这些函数都遵循一个公共的调用规范：第一个参数是待格式化的值，
而第二个是一个定义输出或输入格式的模板。

还有一个单参数的to_timestamp函数；它接受一个 double precision参数，并把 Unix 纪元（自 1970-01-01 00:00:00+00 起的秒数）转换为timestamp with time zone。（Integer Unix 纪元会被隐式转换为double precision。）

表 9.21. 格式化函数

函数	返回类型	描述	示例
to_char(timestamp, text)	text	将时间戳转换为字符串	to_char(current_timestamp, 'HH12:MI:SS')
to_char(interval, text)	text	将时间间隔转换为字符串	to_char(interval '15h 2m 12s', 'HH24:MI:SS')
to_char(int, text)	text	将整数转换为字符串	to_char(125, '999')

函数	返回类型	描述	示例
to_char(double precision, text)	text	将 real/double precision 转换为字符串	to_char(125.8::real, '999D9')
to_char(numeric, text)	text	将 numeric 转换为字符串	to_char(-125.8, '999D99S')
to_date(text, text)	date	将字符串转换为日期	to_date('05 Dec 2000', 'DD Mon YYYY')
to_number(text, text)	numeric	将字符串转换为 numeric	to_number('12,454.8-', '99G999D9S')
to_timestamp(text, text)	timestamp with time zone	将字符串转换为时间戳	to_timestamp('05 Dec 2000', 'DD Mon YYYY')
to_timestamp(double precision)	timestamp with time zone	把 Unix 纪元转换为时间戳	to_timestamp(1284352323)

在一个to_char输出模板串中，
一些特定的模式可以被识别并且被替换成基于给定值的被恰当地格式化的数据。
任何不属于模板模式的文本都简单地照字面拷贝。同样，
在一个输入模板串里（对其他函数），模板模式标识由输入数据串提供的值。

表 9.22展示了可以用于格式化日期和时间值的模板模式。

表 9.22. 用于日期/时间格式化的模板模式

模式	描述
HH	一天中的小时（01-12）
HH12	一天中的小时（01-12）
HH24	一天中的小时（00-23）
MI	分钟（00-59）
SS	秒（00-59）
MS	毫秒（000-999）
US	微秒（000000-999999）
SSSS	自午夜起的秒数（0-86399）
AM, am, PM 或 pm	上午/下午标记（不带句点）
A.M., a.m., P.M. 或 p.m.	上午/下午标记（带句点）
Y,YYY	带逗号的年（4 位或者更多位）
YYYY	年（4 位或者更多位）
YYY	年的最后 3 位数字
YY	年的最后 2 位数字
Y	年的最后 1 位数字
IYYY	ISO 8601 周编号方式的年（4 位或更多位）
IYY	ISO 8601 周编号方式的年的最后 3 位数字
IY	ISO 8601 周编号方式的年的最后 2 位数字
I	ISO 8601 周编号方式的年的最后 1 位数字
BC, bc, AD 或 ad	纪元指示器（不带句号）
B.C., b.c., A.D. 或 a.d.	纪元指示器（带句号）
MONTH	大写的月份全称（空格补齐到 9 字符）
Month	首字母大写的月份全称（空格补齐到 9 字符）

模式	描述
month	小写的月份全称（空格补齐到 9 字符）
MON	简写的大写形式的月名（英文 3 字符，本地化长度可变）
Mon	简写的首字母大写形式的月名（英文 3 字符，本地化长度可变）
mon	简写的小写形式的月名（英文 3 字符，本地化长度可变）
MM	月份编号（01-12）
DAY	大写的星期全称（空格补齐到 9 字符）
Day	首字母大写的星期全称（空格补齐到 9 字符）
day	小写的星期全称（空格补齐到 9 字符）
DY	大写的星期简称（英语 3 字符，本地化长度可变）
Dy	首字母大写的星期简称（英语 3 字符，本地化长度可变）
dy	小写的星期简称（英语 3 字符，本地化长度可变）
DDD	一年中的第几天（001-366）
IDDD	ISO 8601 周编号年中的第几天（001-371；一年的第 1 天是第一个 ISO 周的星期一）
DD	月内日序数（01-31）
D	星期几，周日 (1) 到周六 (7)
ID	ISO 8601 星期几，周一 (1) 到周日 (7)
W	月内周序数（1-5）（第一周从该月第一天开始）
WW	一年中的第几周（1-53）（第一周从该年第一天开始）
IW	ISO 8601 周编号年中的第几周（01-53；该年的第一个星期四位于第 1 周）
CC	世纪（2 位数）（21 世纪开始于 2001-01-01）
J	儒略日（从公元前 4714 年 11 月 24 日 UTC 午夜开始的整数日数）
Q	季度（被to_date和to_timestamp忽略）
RM	以大写罗马数字表示的月份（I-XII；I=一月）
rm	以小写罗马数字表示的月份（i-xii；i=一月）
TZ	大写时区缩写（仅在to_char中支持）
tz	小写时区缩写（仅在to_char中支持）

修饰符可以被应用于模板模式来修改它们的行为。例如，FMMonth就是带着FM修饰符的Month模式。表 9.23展示了可用于日期/时间格式化的修饰符模式。

表 9.23. 用于日期/时间格式化的模板模式修饰符

修饰符	描述	示例
FM 前缀	填充模式（抑制前导零和填充的空格）	FMMonth
TH 后缀	大写形式的序数后缀	DDTH，例如，12TH
th 后缀	小写形式的序数后缀	DDth，例如，12th
FX 前缀	固定格式全局选项（见使用须知）	FX Month DD Day
TM 前缀	翻译模式（根据lc_time输出本地化的星期名和月份名）	TMMonth

修饰符	描述	示例
SP 后缀	拼写模式（未实现）	DDSP

日期/时间格式化的使用注意事项：

- FM抑制了在模式输出中添加前导零和尾随空格的行为，这些前导零和尾随空格本来会被添加以使输出成为固定宽度。在PostgreSQL中，FM仅修改下一个格式说明，而在Oracle中FM影响所有后续格式说明，并且重复的FM修饰符切换填充模式的开启和关闭。
- TM不包含尾随空格。
- to_timestamp 和 to_date 会跳过输入字符串中的多个空格，除非使用 FX 选项。例如，to_timestamp('2000 JUN', 'YYYY MON') 可以工作，但 to_timestamp('2000 JUN', 'FXYYYY MON') 会报错，因为 to_timestamp 只接受一个空格。FX 必须指定为模板中的第一项。
- to_char 模板中允许普通文本，并会按字面输出。可以用双引号括起子串，使其即使包含模式关键字也强制按字面文本解释。例如，在 '"Hello Year "YYYY' 中，YYYY 会被年份数据替换，但 Year 中单独的 Y 不会被替换。在 to_date、to_number 和 to_timestamp 中，双引号字符串会跳过与该字符串所含字符数相同数量的输入字符，例如 "XX" 跳过两个输入字符。
- 如果要在输出中包含双引号，必须在它前面加上反斜杠，例如 '\"YYYY Month\"'。
- 如果年份格式规范少于四位数字，例如YYY，并且提供的年份少于四位数字，年份将被调整为最接近2020年的年份，例如95变为1995年。
- 在从字符串到timestamp或date的转换中，YYYY转换在处理超过4位数字的年份时有限制。您必须在YYYY后使用一些非数字字符或模板，否则年份总是被解释为4位数字。例如（使用年份20000）：to_date('200001131', 'YYYYMMDD')将被解释为4位年份；而应该在年份后使用非数字分隔符，如 to_date('20000-1131', 'YYYY-MMDD')或 to_date('20000Nov31', 'YYYYMonDD')。
- 在从字符串到timestamp或date的转换中，如果存在YYY、YYYY或Y,YYY字段，则忽略CC（世纪）字段。如果CC与 YY或Y一起使用，则年份按(CC-1)*100+YY计算。如果指定了世纪但未指定年份，则假定为该世纪的第一年。
- 可以向to_timestamp和to_date以下列两种方式之一指定 ISO 8601 周编号日期（与公历日期不同）：
  - 年份、周编号和星期几：例如to_date('2006-42-4', 'IYYY-IW-ID')返回日期2006-10-19。如果省略星期几，则假定它为 1（星期一）。
  - 年份和年内日序数：例如to_date('2006-291', 'IYYY-IDDD')也返回2006-10-19。

  尝试使用ISO 8601周编号字段和公历日期字段的混合输入日期是荒谬的，并将导致错误。在ISO 8601周编号年的背景下，“月份”或“月内日序数”的概念没有意义。在公历年的背景下，ISO周没有意义。

> **小心**
>
> 虽然to_date会拒绝混合使用公历和ISO周编号日期字段，但to_char不会，因为输出格式规范如YYYY-MM-DD (IYYY-IDDD)可能很有用。但要避免编写类似IYYY-MM-DD的内容；那会在年初附近产生令人惊讶的结果。（有关更多信息，请参见第 9.9.1 节。）

- 在从字符串到timestamp的转换中，毫秒（MS）或微秒（US）值被用作小数点后的秒数位。例如to_timestamp('12:3', 'SS:MS')不是3毫秒，而是300，因为转换将其计为12 + 0.3秒。这意味着对于格式SS:MS，输入值12:3、12:30和12:300指定相同数量的毫秒。要获得三毫秒，必须写成12:003，转换将其计为12 + 0.003 = 12.003秒。

  这是一个更复杂的示例：to_timestamp('15:12:02.020.001230', 'HH:MI:SS.MS.US') 为15小时12分钟，秒数为2秒 + 20毫秒 + 1230微秒 = 2.021230秒。

- to_char(..., 'ID')的星期几编号与extract(isodow from ...)函数匹配，但to_char(..., 'D') 的星期几编号与 extract(dow from ...) 的不一致。

- to_char(interval) 按照如12小时制时钟上显示的方式格式化HH和HH12，即零小时和36小时都输出为12，而HH24输出完整的小时值，对于时间间隔它可以超过23。

表 9.24展示了可以用于格式化数值的模板模式。

表 9.24. 用于数值格式化的模板模式

模式	描述
9	数位（非有效位可以被省略）
0	数位（即便是非有效位也不会被省略）
.（句点）	小数点
,（逗号）	分组（千）分隔符
PR	尖括号内的负值
S	紧贴数值的正负号（使用区域设置）
L	货币符号（使用区域设置）
D	小数点（使用区域设置）
G	分组分隔符（使用区域设置）
MI	在指定位置的负号（如果数字 < 0）
PL	在指定位置的正号（如果数字 > 0）
SG	在指定位置的正/负号
RN	罗马数字（输入在 1 和 3999 之间）
TH 或 th	序数后缀
V	移动指定位数（参阅注解）
EEEE	科学记数的指数

数值格式化的使用注意事项：

- 0指定一个数字位置，即使它包含前导/尾随零，也将始终打印出来。9也指定一个数字位置，但如果它是一个前导零，则将被替换为一个空格，而如果它是一个尾随零并且指定了填充模式，则将被删除。（对于to_number()，这两个模式字符是等效的。）

- 模式字符S、L、D和G表示当前区域设置定义的正负号、货币符号、小数点和千位分隔符字符（参见lc_monetary和lc_numeric）。模式字符句点和逗号表示这些确切字符，具有小数点和千位分隔符的含义，不受区域设置影响。

- 如果to_char()的模式中没有明确指定正负号的位置，就会为正负号保留一列，并使其紧贴数值（紧靠数值左侧）。如果S紧邻若干个9的左侧，它同样会紧贴数值。

- 使用SG、PL或MI格式化的正负号不紧贴数值；例如，to_char(-12, 'MI9999')会产生'- 12'，但to_char(-12, 'S9999')会产生' -12'。（Oracle实现不允许在9之前使用MI，而是要求9在MI之前。）

- TH不会转换小于零的值，也不会转换小数。
- PL，SG和 TH是PostgreSQL 的扩展。
- V实际上将输入值乘以10^*n*，其中*n*是跟在V后面的数字位数。to_char 不支持与小数点结合使用的V（例如，不允许使用99.9V99）。
- EEEE（科学计数法）不能与任何其他格式模式或修饰符结合使用，除了数字和小数点模式之外，必须位于格式字符串的末尾（例如，9.99EEEE 是一个有效模式）。

某些修饰符可以被应用到任何模板来改变其行为。例如，FM99.99是带有FM修饰符的99.99模式。表 9.25中展示了用于数值格式化的模式修饰符。

表 9.25. 用于数值格式化的模板模式修饰符

修饰符	描述	示例
FM 前缀	填充模式（抑制尾随零和填充的空白）	FM99.99
TH 后缀	大写序数后缀	999TH
th 后缀	小写序数后缀	999th

表 9.26展示了一些使用to_char函数的示例。

表 9.26. to_char示例

表达式	结果
to_char(current_timestamp, 'Day, DD HH12:MI:SS')	'Tuesday , 06 05:39:18'
to_char(current_timestamp, 'FMDay, FMDD HH12:MI:SS')	'Tuesday, 6 05:39:18'
to_char(-0.1, '99.99')	' -.10'
to_char(-0.1, 'FM9.99')	'-.1'
to_char(-0.1, 'FM90.99')	'-0.1'
to_char(0.1, '0.9')	' 0.1'
to_char(12, '9990999.9')	' 0012.0'
to_char(12, 'FM9990999.9')	'0012.'
to_char(485, '999')	' 485'
to_char(-485, '999')	'-485'
to_char(485, '9 9 9')	' 4 8 5'
to_char(1485, '9,999')	' 1,485'
to_char(1485, '9G999')	' 1 485'
to_char(148.5, '999.999')	' 148.500'
to_char(148.5, 'FM999.999')	'148.5'
to_char(148.5, 'FM999.990')	'148.500'
to_char(148.5, '999D999')	' 148,500'
to_char(3148.5, '9G999D999')	' 3 148,500'
to_char(-485, '999S')	'485-'
to_char(-485, '999MI')	'485-'
to_char(485, '999MI')	'485 '
to_char(485, 'FM999MI')	'485'
to_char(485, 'PL999')	'+485'

表达式	结果
to_char(485, 'SG999')	'+485'
to_char(-485, 'SG999')	'-485'
to_char(-485, '9SG99')	'4-85'
to_char(-485, '999PR')	'<485>'
to_char(485, 'L999')	'DM 485'
to_char(485, 'RN')	' CDLXXXV'
to_char(485, 'FMRN')	'CDLXXXV'
to_char(5.2, 'FMRN')	'V'
to_char(482, '999th')	' 482nd'
to_char(485, '"Good number:"999')	'Good number: 485'
to_char(485.8, '"Pre:"999" Post:" .999')	'Pre: 485 Post: .800'
to_char(12, '99V999')	' 12000'
to_char(12.4, '99V999')	' 12400'
to_char(12.45, '99V9')	' 125'
to_char(0.0004859, '9.99EEEE')	' 4.86e-04'

9.9. 日期/时间函数和操作符

表 9.28展示了可用于处理日期/时间值的函数，其细节在随后的小节中描述。表 9.27演示了基本算术操作符（+、*等）的行为。而与格式化相关的函数，可以参考第 9.8 节。你应当熟悉第 8.5 节中的日期/时间数据类型的背景知识。

所有下文描述的接受time或timestamp
输入的函数和操作符实际上都有两种变体：一种接收time with time zone或timestamp with time zone，另外一种接受time without time zone或者timestamp without time zone。为了简化，这些变种没有被独立地展示。此外，+和*操作符都是可交换的操作符对（例如，date + integer 和 integer + date）；我们只显示每一对中的一个。

表 9.27. 日期/时间操作符

操作符	示例	结果
+	date '2001-09-28' + integer '7'	date '2001-10-05'
+	date '2001-09-28' + interval '1 hour'	timestamp '2001-09-28 01:00:00'
+	date '2001-09-28' + time '03:00'	timestamp '2001-09-28 03:00:00'
+	interval '1 day' + interval '1 hour'	interval '1 day 01:00:00'
+	timestamp '2001-09-28 01:00' + interval '23 hours'	timestamp '2001-09-29 00:00:00'
+	time '01:00' + interval '3 hours'	time '04:00:00'
-	- interval '23 hours'	interval '-23:00:00'
-	date '2001-10-01' - date '2001-09-28'	integer '3'（天）
-	date '2001-10-01' - integer '7'	date '2001-09-24'

操作符	示例	结果
-	date '2001-09-28' - interval '1 hour'	timestamp '2001-09-27 23:00:00'
-	time '05:00' - time '03:00'	interval '02:00:00'
-	time '05:00' - interval '2 hours'	time '03:00:00'
-	timestamp '2001-09-28 23:00' - interval '23 hours'	timestamp '2001-09-28 00:00:00'
-	interval '1 day' - interval '1 hour'	interval '1 day -01:00:00'
-	timestamp '2001-09-29 03:00' - timestamp '2001-09-27 12:00'	interval '1 day 15:00:00'
*	900 * interval '1 second'	interval '00:15:00'
*	21 * interval '1 day'	interval '21 days'
*	double precision '3.5' * interval '1 hour'	interval '03:30:00'
/	interval '1 hour' / double precision '1.5'	interval '00:40:00'

表 9.28. 日期/时间函数

函数	age(timestamp, timestamp)
返回类型	interval
描述	将两个参数相减，生成一个使用年和月，而不是只用日的“符号化”的结果
示例	age(timestamp '2001-04-10', timestamp '1957-06-13')
结果	43 years 9 mons 27 days
函数	age(timestamp)
返回类型	interval
描述	从 current_date（午夜）减去
示例	age(timestamp '1957-06-13')
结果	43 years 8 mons 3 days
函数	clock_timestamp()
返回类型	timestamp with time zone
描述	当前日期和时间（在语句执行期间变化）；参见第 9.9.4 节
示例	
结果	
函数	current_date
返回类型	date
描述	当前日期；参见第 9.9.4 节
示例	
结果	

函数	`current_time`
返回类型	`time with time zone`
描述	一天中的当前时刻；参见第 9.9.4 节
示例	
结果	
函数	`current_timestamp`
返回类型	`timestamp with time zone`
描述	当前日期和时间（当前事务开始时）；参见第 9.9.4 节
示例	
结果	
函数	`date_part(text, timestamp)`
返回类型	`double precision`
描述	获取子字段（等价于 `extract`）；参见第 9.9.1 节
示例	`date_part('hour', timestamp '2001-02-16 20:38:40')`
结果	`20`
函数	`date_part(text, interval)`
返回类型	`double precision`
描述	获取子字段（等价于 `extract`）；参见第 9.9.1 节
示例	`date_part('month', interval '2 years 3 months')`
结果	`3`
函数	`date_trunc(text, timestamp)`
返回类型	`timestamp`
描述	截断到指定精度；另见第 9.9.2 节
示例	`date_trunc('hour', timestamp '2001-02-16 20:38:40')`
结果	`2001-02-16 20:00:00`
函数	`extract(`*`field`*` from timestamp)`
返回类型	`double precision`
描述	获取子字段；参见第 9.9.1 节
示例	`extract(hour from timestamp '2001-02-16 20:38:40')`
结果	`20`
函数	`extract(`*`field`*` from interval)`
返回类型	`double precision`
描述	获取子字段；参见第 9.9.1 节
示例	`extract(month from interval '2 years 3 months')`
结果	`3`
函数	`isfinite(date)`
返回类型	`boolean`

描述	测试日期是否有限（不是正负无穷）
示例	`isfinite(date '2001-02-16')`
结果	`true`
函数	`isfinite(timestamp)`
返回类型	`boolean`
描述	测试时间戳是否有限（不是正负无穷）
示例	`isfinite(timestamp '2001-02-16 21:28:30')`
结果	`true`
函数	`isfinite(interval)`
返回类型	`boolean`
描述	测试时间间隔是否有限
示例	`isfinite(interval '4 hours')`
结果	`true`
函数	`justify_days(interval)`
返回类型	`interval`
描述	调整时间间隔，使 30 天的时段表示为月
示例	`justify_days(interval '35 days')`
结果	`1 mon 5 days`
函数	`justify_hours(interval)`
返回类型	`interval`
描述	调整时间间隔，使 24 小时的时段表示为天
示例	`justify_hours(interval '27 hours')`
结果	`1 day 03:00:00`
函数	`justify_interval(interval)`
返回类型	`interval`
描述	使用 `justify_days` 和 `justify_hours`调整时间间隔，并额外调整符号
示例	`justify_interval(interval '1 mon -1 hour')`
结果	`29 days 23:00:00`
函数	`localtime`
返回类型	`time`
描述	一天中的当前时刻；参见第 9.9.4 节
示例	
结果	
函数	`localtimestamp`
返回类型	`timestamp`
描述	当前日期和时间（当前事务开始时）；参见第 9.9.4 节
示例	

结果	
函数	`now()`
返回类型	`timestamp with time zone`
描述	当前日期和时间（当前事务开始时）；参见第 9.9.4 节
示例	
结果	
函数	`statement_timestamp()`
返回类型	`timestamp with time zone`
描述	当前日期和时间（当前语句开始时）；参见第 9.9.4 节
示例	
结果	
函数	`timeofday()`
返回类型	`text`
描述	当前的日期和时间（类似 `clock_timestamp`，但是采用 `text` 字符串）；参见第 9.9.4 节
示例	
结果	
函数	`transaction_timestamp()`
返回类型	`timestamp with time zone`
描述	当前日期和时间（当前事务开始时）；参见第 9.9.4 节
示例	
结果	

除了这些函数以外，还支持 SQL 操作符`OVERLAPS`：

```
(start1, end1) OVERLAPS (start2, end2)
(start1, length1) OVERLAPS (start2, length2)
```

这个表达式在两个时间段（用它们的端点定义）重叠的时候得到真，
当它们不重叠时得到假。端点可以用一对日期、时间或者时间戳来指定；
或者是用一个后面跟着一个间隔的日期、时间或时间戳来指定。当一对值被提供时，
起点或终点都可以被写在前面，`OVERLAPS`会自动地把较早的值作为起点。
每一个时间段被认为是表示半开区间*start* <= *time* < *end*，除非*start*和*end*相等，这种情况下它表示单个时刻。例如这表示两个只有一个共同端点的时间段不重叠。

```
SELECT (DATE '2001-02-16', DATE '2001-12-21') OVERLAPS
       (DATE '2001-10-30', DATE '2002-10-30');
结果: true
SELECT (DATE '2001-02-16', INTERVAL '100 days') OVERLAPS
       (DATE '2001-10-30', DATE '2002-10-30');
结果: false
SELECT (DATE '2001-10-29', DATE '2001-10-30') OVERLAPS
       (DATE '2001-10-30', DATE '2001-10-31');
结果: false
SELECT (DATE '2001-10-30', DATE '2001-10-30') OVERLAPS
       (DATE '2001-10-30', DATE '2001-10-31');
结果: true
```

当把一个interval值加到某个timestamp with time zone值上（或从中减去一个interval值）时，天数部分会按指定的天数推进或退回timestamp with time zone的日期。当跨越夏令时变化时（会话时区设置为识别夏令时的时区），这意味着interval '1 day'不一定等于interval '24 hours'。例如，将会话时区设置为CST7CDT时，timestamp with time zone '2005-04-02 12:00-07' + interval '1 day'会得到timestamp with time zone '2005-04-03 12:00-06'，而向同一个初始timestamp with time zone加上interval '24 hours'会得到timestamp with time zone '2005-04-03 13:00-06'，这是因为在时区CST7CDT中，夏令时变化发生在2005-04-03 02:00。

注意，age返回的months字段可能存在歧义，因为不同月份的天数不同。PostgreSQL在计算不足整月的部分时，会采用两个日期中较早的那个日期所在的月份。例如：age('2004-06-01', '2004-04-30')使用 4 月得到1 mon 1 day，而如果使用 5 月则会得到1 mon 2 days，因为 5 月有 31 天，而 4 月只有 30 天。

9.9.1. EXTRACT, date_part

EXTRACT(*field* FROM *source*)

该extract函数从日期/时间值中提取年份、小时等子字段。*source*必须是以下类型的值表达式：timestamp，time，或interval。（date类型的表达式会转换为timestamp，因此也可以使用。）*field*是一个标识符或字符串，用于选择从源值中提取的字段。extract函数返回的值的类型为double precision。以下是有效的字段名称：

century

世纪

```
SELECT EXTRACT(CENTURY FROM TIMESTAMP '2000-12-16 12:21:13');
```
结果：20
```
SELECT EXTRACT(CENTURY FROM TIMESTAMP '2001-02-16 20:38:40');
```
结果：21

第一个世纪始于公元 0001-01-01 00:00:00，尽管当时的人们并不知道。这一定义适用于所有采用格里高利历的国家。世纪编号没有 0，而是从 -1 世纪直接跳到 1 世纪。如果你对此有异议，请向梵蒂冈罗马圣彼得大教堂的教皇投诉。

PostgreSQL 8.0 之前的版本不遵循传统的世纪编号方式，而是直接返回年份字段除以 100 的结果。

day

对于 timestamp 值，表示月份中的日期字段（1 - 31）；对于 interval 值，表示天数

```
SELECT EXTRACT(DAY FROM TIMESTAMP '2001-02-16 20:38:40');
```
结果：16
```
SELECT EXTRACT(DAY FROM INTERVAL '40 days 1 minute');
```
结果：40

decade

年份字段除以10

```
SELECT EXTRACT(DECADE FROM TIMESTAMP '2001-02-16 20:38:40');
```
结果：200

dow

一周的日子从星期天（0）到星期六（6）

```
SELECT EXTRACT(DOW FROM TIMESTAMP '2001-02-16 20:38:40');
```

结果：5

请注意extract函数的星期几编号与to_char(..., 'D')函数不同。

doy

一年中的第几天（1 - 365/366）

```
SELECT EXTRACT(DOY FROM TIMESTAMP '2001-02-16 20:38:40');
```

结果：47

epoch

对于timestamp with time zone值，自1970-01-01 00:00:00 UTC 以来的秒数（可以为负）；对于date和timestamp值，是自1970-01-01 00:00:00 本地时间以来的秒数；对于interval值，是间隔的总秒数

```
SELECT EXTRACT(EPOCH FROM TIMESTAMP WITH TIME ZONE '2001-02-16
 20:38:40.12-08');
```

结果：982384720.12

```
SELECT EXTRACT(EPOCH FROM INTERVAL '5 days 3 hours');
```

结果：442800

下面介绍如何把一个 epoch 值转换回时间戳：

```
SELECT TIMESTAMP WITH TIME ZONE 'epoch' + 982384720.12 *
 INTERVAL '1 second';
```

（to_timestamp函数封装了上述转换。）

hour

小时字段（0 - 23）

```
SELECT EXTRACT(HOUR FROM TIMESTAMP '2001-02-16 20:38:40');
```

结果：20

isodow

一周的日子从星期一（1）到星期日（7）

```
SELECT EXTRACT(ISODOW FROM TIMESTAMP '2001-02-18 20:38:40');
```

结果：7

这与dow相同，除了星期天。这匹配ISO 8601的星期几编号。

isoyear

日期所属的 ISO 8601 周编号年份（不适用于间隔）

```
SELECT EXTRACT(ISOYEAR FROM DATE '2006-01-01');
```

结果：2005

```
SELECT EXTRACT(ISOYEAR FROM DATE '2006-01-02');
```

结果：2006

每个 ISO 8601 周编号年都从包含 1 月 4 日的那一周的星期一开始，因此在 1 月初或 12 月末，ISO 年可能与格里高利年不同。更多信息请参见 week 字段。

PostgreSQL 8.3 之前的版本不支持此字段。

microseconds

秒字段，包括小数部分，乘以1 000 000；注意这包括完整的秒数

```
SELECT EXTRACT(MICROSECONDS FROM TIME '17:12:28.5');
```
结果：28500000

`millennium`

千年

```
SELECT EXTRACT(MILLENNIUM FROM TIMESTAMP '2001-02-16 20:38:40');
```
结果：3

1900 年代的年份属于第二个千年。第三个千年始于 2001 年 1 月 1 日。

PostgreSQL 8.0 之前的版本不遵循传统的千年编号方式，而是直接返回年份字段除以 1000 的结果。

`milliseconds`

秒字段，包括小数部分，乘以1000。请注意，这包括完整的秒数。

```
SELECT EXTRACT(MILLISECONDS FROM TIME '17:12:28.5');
```
结果：28500

`minute`

分钟字段（0 - 59）

```
SELECT EXTRACT(MINUTE FROM TIMESTAMP '2001-02-16 20:38:40');
```
结果：38

`month`

对于 `timestamp` 值，表示一年中的月份编号（1 - 12）；对于 `interval` 值，表示月数对 12 取模（0 - 11）

```
SELECT EXTRACT(MONTH FROM TIMESTAMP '2001-02-16 20:38:40');
```
结果：2
```
SELECT EXTRACT(MONTH FROM INTERVAL '2 years 3 months');
```
结果：3
```
SELECT EXTRACT(MONTH FROM INTERVAL '2 years 13 months');
```
结果：1

`quarter`

日期所在的一年中的季度（1 - 4）

```
SELECT EXTRACT(QUARTER FROM TIMESTAMP '2001-02-16 20:38:40');
```
结果：1

`second`

秒字段，包括小数部分（0 - 59[1]）

```
SELECT EXTRACT(SECOND FROM TIMESTAMP '2001-02-16 20:38:40');
```
结果：40
```
SELECT EXTRACT(SECOND FROM TIME '17:12:28.5');
```
结果：28.5

`timezone`

与UTC的时区偏移量，以秒为单位。正值对应于UTC东部的时区，
负值对应于UTC西部的时区。（从技术上讲，PostgreSQL使用 UT1 而不是 UTC，因为不处理闰秒。）

[1]如果操作系统实现了闰秒，则为 60

timezone_hour

时区偏移的小时部分

timezone_minute

时区偏移的分钟部分

week

一年中按ISO 8601 周编号体系计算的周序号。根据定义，ISO 周从周一开始，一年的第一周包含该年的 1 月 4 日。换句话说，一年的第一个星期四在该年的第 1 周。

在 ISO 周编号体系中，1 月初的日期可能属于前一年的第 52 周或第 53 周，而 12 月末的日期可能属于下一年的第一周。例如，2005-01-01属于 2004 年的第 53 周，2006-01-01属于 2005 年的第 52 周，而2012-12-31属于 2013 年的第一周。建议将isoyear字段与week一起使用，以获得一致的结果。

```
SELECT EXTRACT(WEEK FROM TIMESTAMP '2001-02-16 20:38:40');
```
结果：7

year

年份字段。请记住，没有0 AD，所以把BC年份从AD年份中减去时需要小心。

```
SELECT EXTRACT(YEAR FROM TIMESTAMP '2001-02-16 20:38:40');
```
结果：2001

extract函数主要的用途是做计算性处理。对于用于显示的日期/时间值格式化，参阅第 9.8 节。

date_part函数仿照传统的Ingres实现，后者对应SQL标准的extract函数：

date_part('*field*', *source*)

注意，此处的*field*参数必须是字符串值，而不能是名称。date_part的有效字段名与extract相同。

```
SELECT date_part('day', TIMESTAMP '2001-02-16 20:38:40');
```
结果：16
```
SELECT date_part('hour', INTERVAL '4 hours 3 minutes');
```
结果：4

9.9.2. date_trunc

date_trunc函数在概念上和用于数字的trunc函数类似。

date_trunc('*field*', *source*)

*source*是timestamp或interval类型的值表达式。（类型为date和time的值会自动转换为timestamp或interval，分别对应这两种输入类型。）*field*选择输入值的截断精度。返回值的类型是timestamp或interval，所有小于所选精度的字段都设为零（日和月则设为一）。

*field*的有效值是：

microseconds
milliseconds
second
minute
hour
day
week

```
month
quarter
year
decade
century
millennium
```

示例：

```
SELECT date_trunc('hour', TIMESTAMP '2001-02-16 20:38:40');
```
结果：`2001-02-16 20:00:00`

```
SELECT date_trunc('year', TIMESTAMP '2001-02-16 20:38:40');
```
结果：`2001-01-01 00:00:00`

9.9.3. AT TIME ZONE

`AT TIME ZONE` 结构允许把时间戳转换到不同的时区。表 9.29展示了它的各种变体。

表 9.29. AT TIME ZONE 变体

表达式	返回类型	描述
`timestamp without time zone AT TIME ZONE zone`	`timestamp with time zone`	将给定的不带时区时间戳视为指定时区中的时间
`timestamp with time zone AT TIME ZONE zone`	`timestamp without time zone`	将给定的带时区时间戳转换为新时区的时间，结果不带时区标识
`time with time zone AT TIME ZONE zone`	`time with time zone`	将给定的带时区时间转换为新时区的时间

在这些表达式里，所需的时区 *zone* 可以指定为文本值（例如 `'PST'`），也可以指定为一个间隔值（例如 `INTERVAL '-08:00'`）。在文本情况下，时区名称可以按第 8.5.3 节中描述的任意方式指定。

示例（假设本地时区为`PST8PDT`）：

```
SELECT TIMESTAMP '2001-02-16 20:38:40' AT TIME ZONE 'MST';
```
结果：`2001-02-16 19:38:40-08`

```
SELECT TIMESTAMP WITH TIME ZONE '2001-02-16 20:38:40-05' AT TIME
 ZONE 'MST';
```
结果：`2001-02-16 18:38:40`

第一个示例取一个不带时区的时间戳，把它解释为 MST 时间（UTC-7），然后转换为 PST（UTC-8）用于显示。第二个示例取一个以 EST（UTC-5）指定的时间戳，把它转换为 MST（UTC-7）的本地时间。

函数`timezone(zone, timestamp)`等效于符合 SQL 标准的结构`timestamp AT TIME ZONE zone`。

9.9.4. 当前日期/时间

PostgreSQL提供了许多返回当前日期和时间的函数。这些 SQL 标准的函数全部都按照当前事务的开始时刻返回值：

```
CURRENT_DATE
CURRENT_TIME
CURRENT_TIMESTAMP
CURRENT_TIME(precision)
CURRENT_TIMESTAMP(precision)
```

```
LOCALTIME
LOCALTIMESTAMP
LOCALTIME(precision)
LOCALTIMESTAMP(precision)
```

CURRENT_TIME和CURRENT_TIMESTAMP返回带时区的值；LOCALTIME和LOCALTIMESTAMP返回不带时区的值。

CURRENT_TIME、CURRENT_TIMESTAMP、LOCALTIME和 LOCALTIMESTAMP可以有选择地接受一个精度参数，该精度会使结果的秒字段舍入到指定的小数位数。如果没有精度参数，结果将给出可用的全部精度。

下面是一些示例：

```
SELECT CURRENT_TIME;
结果： 14:39:53.662522-05
SELECT CURRENT_DATE;
结果： 2001-12-23
SELECT CURRENT_TIMESTAMP;
结果： 2001-12-23 14:39:53.662522-05
SELECT CURRENT_TIMESTAMP(2);
结果： 2001-12-23 14:39:53.66-05
SELECT LOCALTIMESTAMP;
结果： 2001-12-23 14:39:53.662522
```

因为这些函数全部都按照当前事务的开始时刻返回结果，所以它们的值在事务运行的整个期间内都不改变。我们认为这是一个特性：目的是为了允许一个事务在“当前”时间上有一致的概念，这样在同一个事务里的多个修改可以保持同样的时间戳。

> **注意**
>
> 其他数据库系统可能会更频繁地推进这些值。

PostgreSQL还提供了返回当前语句的开始时间以及调用该函数时的实际当前时间的函数。这些非 SQL 标准的函数列表如下：

```
transaction_timestamp()
statement_timestamp()
clock_timestamp()
timeofday()
now()
```

transaction_timestamp()等价于CURRENT_TIMESTAMP，但是其命名清楚地反映了它的返回值。statement_timestamp()返回当前语句的开始时刻（更准确地说，是接收到客户端最近一条命令消息的时间）。statement_timestamp()和transaction_timestamp()在一个事务的第一条命令期间返回值相同，但是在随后的命令中却不一定相同。clock_timestamp()返回真正的当前时间，因此它的值甚至在同一条 SQL 命令中都会变化。timeofday()是一个有历史原因的PostgreSQL函数。和clock_timestamp()相似，它也返回真实的当前时间，但是它的结果是一个格式化的text串，而不是timestamp with time zone值。now()是PostgreSQL中与transaction_timestamp()等价的传统函数。

所有日期/时间数据类型也都接受特殊字面值now来指定当前日期和时间（同样解释为事务开始时间）。因此，下面三种写法都返回相同的结果：

```
SELECT CURRENT_TIMESTAMP;
SELECT now();
```

```
SELECT TIMESTAMP 'now';  -- incorrect for use with DEFAULT
```

> **提示**
>
> 在创建表时指定DEFAULT子句的情况下，你不希望使用第三种形式。
> 系统将在分析这个常量的时候把now转换为一个timestamp，
> 这样需要默认值时就会得到创建表的时间！而前两种形式要到实际使用默认值的时候才被计算，因为它们是函数调用。因此它们可以给出每次插入行的时刻。

9.9.5. 延时执行

以下函数可用于延迟服务器进程的执行：

pg_sleep(*seconds*)

pg_sleep使当前会话的进程休眠*seconds*秒。*seconds*的类型为double precision，因此可以指定带小数部分的秒数作为延迟时间。例如：

```
SELECT pg_sleep(1.5);
```

> **注意**
>
> 有效的休眠时间间隔精度与平台有关，0.01 秒是常见值。休眠延迟将至少持续指定的时长，也有可能由于服务器负荷而比指定的时间长。特别地，
> 休眠并不保证能刚好在指定的时刻结束，但它不会在比指定时刻早的时候醒来。

> **警告**
>
> 请确保在调用pg_sleep或者其变体时，你的会话没有持有不必要的锁。
> 否则其它会话可能必须等待你的休眠会话，因而减慢整个系统速度。

9.10. 枚举支持函数

对于枚举类型（见第 8.7 节），有些函数可以避免硬编码枚举类型中的特定值，使程序更简洁。这些函数列在表 9.30中。以下示例假定枚举类型按如下方式创建：

```
CREATE TYPE rainbow AS ENUM ('red', 'orange', 'yellow', 'green',
 'blue', 'purple');
```

表 9.30. 枚举支持函数

函数	描述	示例	示例结果
enum_first(anyenum)	返回输入枚举类型的第一个值。	enum_first(null::rainbow)	red
enum_last(anyenum)	返回输入枚举类型的最后一个值。	enum_last(null::rainbow)	purple
enum_range(anyenum)	将输入枚举类型的所有值作为一个有序的数组返回。	enum_range(null::rainbow)	{red,orange,yellow,green,blue,purple}
enum_range(anyenum, anyenum)	以有序数组返回两个给定枚举值之间的范围。两个值必须来自同一枚举类型。如果第一个参	enum_range('orange'::rainbow, 'green'::rainbow)	{orange,yellow,green}

函数	描述	示例	示例结果
	数为 null，结果从该枚举类型的第一个值开始；如果第二个参数为 null，结果以该枚举类型的最后一个值结束。	enum_range(NULL, 'green'::rainbow)	{red,orange,yellow,green}
		enum_range('orange'::rainbow, NULL)	{orange,yellow,green,blue,purple}

请注意，除了enum_range的双参数形式外，这些函数都忽略传入的具体值，只关心其声明的数据类型。传入 null 或该类型的某个具体值，结果都相同。通常会将这些函数用于表列或函数参数，而不是像示例那样使用硬编码的类型名。

9.11. 几何函数和操作符

几何类型point、box、lseg、line、path、polygon和circle有大量内置支持函数和操作符，如表 9.31、表 9.32和表 9.33中所示。

> **小心**
>
> 注意，“相同”操作符 ~= 表示 point、box、polygon 和 circle 类型通常意义上的相等。其中一些类型也有 = 操作符，但 = 仅比较面积是否相等。这些类型的其他标量比较操作符（<= 等）同样比较面积。

表 9.31. 几何操作符

操作符	描述	示例
+	平移	box '((0,0),(1,1))' + point '(2.0,0)'
-	平移	box '((0,0),(1,1))' - point '(2.0,0)'
*	缩放/旋转	box '((0,0),(1,1))' * point '(2.0,0)'
/	缩放/旋转	box '((0,0),(2,2))' / point '(2.0,0)'
#	相交的点或矩形框	box '((1,-1),(-1,1))' # box '((1,1),(-2,-2))'
#	路径或多边形中的点数	# path '((1,0),(0,1),(-1,0))'
@-@	长度或周长	@-@ path '((0,0),(1,0))'
@@	中心	@@ circle '((0,0),10)'
##	第二个操作数上距离第一个操作数最近的点	point '(0,0)' ## lseg '((2,0),(0,2))'
<->	两者之间的距离	circle '((0,0),1)' <-> circle '((5,0),1)'
&&	是否重叠？（有一个公共点即为真。）	box '((0,0),(1,1))' && box '((0,0),(2,2))'
<<	是否严格位于左侧？	circle '((0,0),1)' << circle '((5,0),1)'
>>	是否严格位于右侧？	circle '((5,0),1)' >> circle '((0,0),1)'
&<	是否未超出对方的右边界？	box '((0,0),(1,1))' &< box '((0,0),(2,2))'

操作符	描述	示例
&>	是否未超出对方的左边界?	box '((0,0),(3,3))' &> box '((0,0),(2,2))'
<<|	是否严格位于下方?	box '((0,0),(3,3))' <<| box '((3,4),(5,5))'
|>>	是否严格位于上方?	box '((3,4),(5,5))' |>> box '((0,0),(3,3))'
&<|	是否未超出对方的上边界?	box '((0,0),(1,1))' &<| box '((0,0),(2,2))'
|&>	是否未超出对方的下边界?	box '((0,0),(3,3))' |&> box '((0,0),(2,2))'
<^	是否位于下方（允许接触）?	circle '((0,0),1)' <^ circle '((0,5),1)'
>^	是否位于上方（允许接触）?	circle '((0,5),1)' >^ circle '((0,0),1)'
?#	是否相交?	lseg '((-1,0),(1,0))' ?# box '((-2,-2),(2,2))'
?-	是否水平?	?- lseg '((-1,0),(1,0))'
?-	是否水平对齐?	point '(1,0)' ?- point '(0,0)'
?|	是否竖直?	?| lseg '((-1,0),(1,0))'
?|	是否竖直对齐?	point '(0,1)' ?| point '(0,0)'
?-|	是否互相垂直?	lseg '((0,0),(0,1))' ?-| lseg '((0,0),(1,0))'
?||	是否平行?	lseg '((-1,0),(1,0))' ?|| lseg '((-1,2),(1,2))'
@>	是否包含?	circle '((0,0),2)' @> point '(1,1)'
<@	是否位于内部或边界上?	point '(1,1)' <@ circle '((0,0),2)'
~=	是否相同?	polygon '((0,0),(1,1))' ~= polygon '((1,1),(0,0))'

注意

在 PostgreSQL 8.2 之前，包含操作符 @> 和 <@ 分别称为 ~ 和 @。这些名称仍然可用，但已被弃用，最终将被移除。

表 9.32. 几何函数

函数	返回类型	描述	示例
area(*object*)	double precision	面积	area(box '((0,0),(1,1))')
center(*object*)	point	中心	center(box '((0,0),(1,2))')
diameter(circle)	double precision	圆的直径	diameter(circle '((0,0),2.0)')
height(box)	double precision	矩形框的竖直尺寸	height(box '((0,0),(1,1))')

函数	返回类型	描述	示例
isclosed(path)	boolean	是否为闭合路径?	isclosed(path '((0,0),(1,1),(2,0))')
isopen(path)	boolean	是否为开放路径?	isopen(path '[(0,0),(1,1),(2,0)]')
length(*object*)	double precision	长度	length(path '((-1,0),(1,0))')
npoints(path)	int	点数	npoints(path '[(0,0),(1,1),(2,0)]')
npoints(polygon)	int	点数	npoints(polygon '((1,1),(0,0))')
pclose(path)	path	将路径转换为闭合路径	pclose(path '[(0,0),(1,1),(2,0)]')
popen(path)	path	将路径转换为开放路径	popen(path '((0,0),(1,1),(2,0))')
radius(circle)	double precision	圆的半径	radius(circle '((0,0),2.0)')
width(box)	double precision	矩形框的水平尺寸	width(box '((0,0),(1,1))')

表 9.33. 几何类型转换函数

函数	返回类型	描述	示例
box(circle)	box	将圆转换为矩形框	box(circle '((0,0),2.0)')
box(point, point)	box	将点转换为矩形框	box(point '(0,0)', point '(1,1)')
box(polygon)	box	将多边形转换为矩形框	box(polygon '((0,0),(1,1),(2,0))')
circle(box)	circle	将矩形框转换为圆	circle(box '((0,0),(1,1))')
circle(point, double precision)	circle	由圆心和半径构造圆	circle(point '(0,0)', 2.0)
circle(polygon)	circle	将多边形转换为圆	circle(polygon '((0,0),(1,1),(2,0))')
lseg(box)	lseg	将矩形框的对角线转换为线段	lseg(box '((-1,0),(1,0))')
lseg(point, point)	lseg	由点构造线段	lseg(point '(-1,0)', point '(1,0)')
path(polygon)	path	将多边形转换为路径	path(polygon '((0,0),(1,1),(2,0))')
point(double precision, double precision)	point	构造点	point(23.4, -44.5)
point(box)	point	矩形框的中心	point(box '((-1,0),(1,0))')
point(circle)	point	圆的中心	point(circle '((0,0),2.0)')

函数	返回类型	描述	示例
point(lseg)	point	线段的中心	point(lseg '((-1,0),(1,0))')
point(polygon)	point	多边形的中心	point(polygon '((0,0),(1,1),(2,0))')
polygon(box)	polygon	将矩形框转换为 4 个顶点的多边形	polygon(box '((0,0),(1,1))')
polygon(circle)	polygon	将圆转换为 12 个顶点的多边形	polygon(circle '((0,0),2.0)')
polygon(*npts*, circle)	polygon	将圆转换为 *npts* 个顶点的多边形	polygon(12, circle '((0,0),2.0)')
polygon(path)	polygon	将路径转换为多边形	polygon(path '((0,0),(1,1),(2,0))')

可以把一个point当作下标为 0 和 1 的数组，访问它的两个数值分量。例如，如果t.p是一个point列，那么SELECT p[0] FROM t检索 X 坐标而 UPDATE t SET p[1] = ...改变 Y 坐标。同样，box或者lseg类型的值可以当作两个point值组成的数组看待。

area 函数适用于 box、circle 和 path 类型。对于 path 数据类型，只有当 path 中的点构成的路径不自相交时，area 函数才能工作。例如，path '((0,0),(0,1),(2,1),(2,2),(1,2),(1,0),(0,0))'::PATH 无法使用；但下面这个视觉上相同的 path '((0,0),(0,1),(1,1),(1,2),(2,2),(2,1),(1,1),(1,0),(0,0))'::PATH 可以使用。如果难以理解自相交与非自相交 path 的区别，可以把上述两条 path 并排画在方格纸上。

9.12. 网络地址函数和操作符

表 9.34列出了可用于 cidr 和 inet 类型的操作符。操作符 <<、<<=、>> 和 >>= 测试子网包含关系。它们只考虑两个地址的网络部分（忽略主机部分），并判断一个网络是否与另一个网络相同，或是另一个网络的子网。

表 9.34. cidr 和 inet 操作符

操作符	描述	示例
<	小于	inet '192.168.1.5' < inet '192.168.1.6'
<=	小于或等于	inet '192.168.1.5' <= inet '192.168.1.5'
=	等于	inet '192.168.1.5' = inet '192.168.1.5'
>=	大于或等于	inet '192.168.1.5' >= inet '192.168.1.5'
>	大于	inet '192.168.1.5' > inet '192.168.1.4'
<>	不等于	inet '192.168.1.5' <> inet '192.168.1.4'
<<	被包含	inet '192.168.1.5' << inet '192.168.1/24'
<<=	被包含在内或等于	inet '192.168.1/24' <<= inet '192.168.1/24'
>>	包含	inet '192.168.1/24' >> inet '192.168.1.5'

操作符	描述	示例
>>=	包含或等于	inet '192.168.1/24' >>= inet '192.168.1/24'
~	按位非	~ inet '192.168.1.6'
&	按位与	inet '192.168.1.6' & inet '0.0.0.255'
|	按位或	inet '192.168.1.6' | inet '0.0.0.255'
+	加法	inet '192.168.1.6' + 25
-	减法	inet '192.168.1.43' - 36
-	减法	inet '192.168.1.43' - inet '192.168.1.19'

表 9.35列出了可用于 cidr 和 inet 类型的函数。abbrev、host 和 text 函数主要用于提供其他显示格式。

表 9.35. cidr 和 inet 函数

函数	abbrev(inet)
返回类型	text
描述	以文本表示的缩略显示格式
示例	abbrev(inet '10.1.0.0/16')
结果	10.1.0.0/16
函数	abbrev(cidr)
返回类型	text
描述	以文本表示的缩略显示格式
示例	abbrev(cidr '10.1.0.0/16')
结果	10.1/16
函数	broadcast(inet)
返回类型	inet
描述	网络的广播地址
示例	broadcast('192.168.1.5/24')
结果	192.168.1.255/24
函数	family(inet)
返回类型	int
描述	提取地址族；IPv4 为 4，IPv6 为 6
示例	family('::1')
结果	6
函数	host(inet)
返回类型	text
描述	以文本形式提取 IP 地址
示例	host('192.168.1.5/24')
结果	192.168.1.5

函数	hostmask(inet)
返回类型	inet
描述	构造网络的主机掩码
示例	hostmask('192.168.23.20/30')
结果	0.0.0.3
函数	masklen(inet)
返回类型	int
描述	提取网络掩码长度
示例	masklen('192.168.1.5/24')
结果	24
函数	netmask(inet)
返回类型	inet
描述	构造网络的网络掩码
示例	netmask('192.168.1.5/24')
结果	255.255.255.0
函数	network(inet)
返回类型	cidr
描述	提取地址的网络部分
示例	network('192.168.1.5/24')
结果	192.168.1.0/24
函数	set_masklen(inet, int)
返回类型	inet
描述	设置 inet 值的网络掩码长度
示例	set_masklen('192.168.1.5/24', 16)
结果	192.168.1.5/16
函数	set_masklen(cidr, int)
返回类型	cidr
描述	设置 cidr 值的网络掩码长度
示例	set_masklen('192.168.1.0/24'::cidr, 16)
结果	192.168.0.0/16
函数	text(inet)
返回类型	text
描述	以文本形式提取 IP 地址和网络掩码长度
示例	text(inet '192.168.1.5')
结果	192.168.1.5/32

任何 cidr 值都可以隐式或显式地转换为 inet；因此，上述处理 inet 的函数也适用于 cidr 值。（如果 inet 和 cidr 各有单独的函数，是因为这两种情况下的行为应当不同。）也允许将 inet 值转换为 cidr。这样做时，网络掩码右侧的所有位都会被直接置零，以创建有效的

cidr 值。此外，还可以用普通的类型转换语法将文本值转换为 inet 或 cidr：例如 inet(*expression*) 或 *colname*::cidr。

表 9.36列出了可用于 macaddr 类型的函数。函数 trunc(macaddr) 返回将最后 3 个字节置零的 MAC 地址。可以用余下的前缀确定制造商。

表 9.36. macaddr 函数

函数	trunc(macaddr)
返回类型	macaddr
描述	将最后 3 个字节设为零
示例	trunc(macaddr '12:34:56:78:90:ab')
结果	12:34:56:00:00:00

macaddr 类型还支持按字典顺序比较的标准关系操作符（>、<= 等），以及用于按位非、与、或的位运算操作符（~、& 和 |）。

9.13. 文本搜索函数和操作符

表 9.37、表 9.38以及表 9.39总结了为全文检索提供的函数和操作符。PostgreSQL的文本检索功能的详细解释可参考第 12 章。

表 9.37. 文本搜索操作符

操作符	描述	示例	结果
@@	tsvector 是否匹配 tsquery?	to_tsvector('fat cats ate rats') @@ to_tsquery('cat & rat')	t
@@@	@@ 的已弃用同义写法	to_tsvector('fat cats ate rats') @@@ to_tsquery('cat & rat')	t
||	串接 tsvector	'a:1 b:2'::tsvector || 'c:1 d:2 b:3'::tsvector	'a':1 'b':2,5 'c':3 'd':4
&&	将 tsquery 进行 AND 组合	'fat | rat'::tsquery && 'cat'::tsquery	('fat' | 'rat') & 'cat'
||	将 tsquery 进行 OR 组合	'fat | rat'::tsquery || 'cat'::tsquery	('fat' | 'rat') | 'cat'
!!	对 tsquery 取反	!! 'cat'::tsquery	!'cat'
@>	tsquery 是否包含另一个查询?	'cat'::tsquery @> 'cat & rat'::tsquery	f
<@	tsquery 是否被包含于另一个查询?	'cat'::tsquery <@ 'cat & rat'::tsquery	t

注意

tsquery 包含操作符只考虑两个查询中列出的词位，忽略组合操作符。

除了表中列出的操作符，tsvector 和 tsquery 类型还定义了普通的 B-树比较操作符（=、< 等）。这些操作符对文本检索用处不大，但可以用于其他用途，例如在这些类型的列上建立唯一索引。

表 9.38. 文本搜索函数

函数	get_current_ts_config()
返回类型	regconfig
描述	获取默认的文本搜索配置
示例	get_current_ts_config()
结果	english
函数	length(tsvector)
返回类型	integer
描述	tsvector 中的词位数
示例	length('fat:2,4 cat:3 rat:5A'::tsvector)
结果	3
函数	numnode(tsquery)
返回类型	integer
描述	tsquery 中的词位数与操作符数之和
示例	numnode('(fat & rat) | cat'::tsquery)
结果	5
函数	plainto_tsquery([*config* regconfig ,] *query* text)
返回类型	tsquery
描述	生成 tsquery，忽略标点符号
示例	plainto_tsquery('english', 'The Fat Rats')
结果	'fat' & 'rat'
函数	querytree(*query* tsquery)
返回类型	text
描述	获取 tsquery 中可索引的部分
示例	querytree('foo & ! bar'::tsquery)
结果	'foo'
函数	setweight(tsvector, "char")
返回类型	tsvector
描述	为tsvector中的每个元素赋予权重
示例	setweight('fat:2,4 cat:3 rat:5B'::tsvector, 'A')
结果	'cat':3A 'fat':2A,4A 'rat':5A
函数	strip(tsvector)
返回类型	tsvector
描述	从 tsvector 中移除位置和权重
示例	strip('fat:2,4 cat:3 rat:5A'::tsvector)

结果	'cat' 'fat' 'rat'
函数	to_tsquery([*config* regconfig ,] *query* text)
返回类型	tsquery
描述	正规化单词并转换为 tsquery
示例	to_tsquery('english', 'The & Fat & Rats')
结果	'fat' & 'rat'
函数	to_tsvector([*config* regconfig ,] *document* text)
返回类型	tsvector
描述	将文档文本转换为 tsvector
示例	to_tsvector('english', 'The Fat Rats')
结果	'fat':2 'rat':3
函数	ts_headline([*config* regconfig,] *document* text, *query* tsquery [, *options* text])
返回类型	text
描述	显示查询匹配内容
示例	ts_headline('x y z', 'z'::tsquery)
结果	x y z
函数	ts_rank([*weights* float4[],] *vector* tsvector, *query* tsquery [, *normalization* integer])
返回类型	float4
描述	计算文档相对于查询的排名分值
示例	ts_rank(textsearch, query)
结果	0.818
函数	ts_rank_cd([*weights* float4[],] *vector* tsvector, *query* tsquery [, *normalization* integer])
返回类型	float4
描述	使用覆盖密度计算文档相对于查询的排名分值
示例	ts_rank_cd('{0.1, 0.2, 0.4, 1.0}', textsearch, query)
结果	2.01317
函数	ts_rewrite(*query* tsquery, *target* tsquery, *substitute* tsquery)
返回类型	tsquery
描述	将查询中的target替换为substitute
示例	ts_rewrite('a & b'::tsquery, 'a'::tsquery, 'foo|bar'::tsquery)
结果	'b' & ('foo' | 'bar')
函数	ts_rewrite(*query* tsquery, *select* text)
返回类型	tsquery
描述	使用 SELECT 命令提供的目标和替代内容进行替换

示例	`SELECT ts_rewrite('a & b'::tsquery, 'SELECT t,s FROM aliases')`
结果	`'b' & ( 'foo' | 'bar' )`
函数	`tsvector_update_trigger()`
返回类型	`trigger`
描述	自动更新 `tsvector` 列的触发器函数
示例	`CREATE TRIGGER ... tsvector_update_trigger(tsvcol, 'pg_catalog.swedish', title, body)`
结果	
函数	`tsvector_update_trigger_column()`
返回类型	`trigger`
描述	自动更新 `tsvector` 列的触发器函数
示例	`CREATE TRIGGER ... tsvector_update_trigger_column(tsvcol, configcol, title, body)`
结果	

注意

所有接受一个可选的`regconfig`参数的文本检索函数在省略该参数时，会使用由default_text_search_config指定的配置。

表 9.39中的函数被单独列出，因为它们通常不被用于日常的文本检索操作。它们有助于开发和调试新的文本检索配置。

表 9.39. 文本搜索调试函数

函数	`ts_debug([ config regconfig, ] document text, OUT alias text, OUT description text, OUT token text, OUT dictionaries regdictionary[], OUT dictionary regdictionary, OUT lexemes text[])`
返回类型	`setof record`
描述	测试配置
示例	`ts_debug('english', 'The Brightest supernovaes')`
结果	`(asciiword,"Word, all ASCII",The,{english_stem},english_stem,{}) ...`
函数	`ts_lexize(dict regdictionary, token text)`
返回类型	`text[]`
描述	测试词典
示例	`ts_lexize('english_stem', 'stars')`
结果	`{star}`
函数	`ts_parse(parser_name text, document text, OUT tokid integer, OUT token text)`
返回类型	`setof record`
描述	测试解析器

示例	ts_parse('default', 'foo - bar')
结果	(1,foo) ...
函数	ts_parse(*parser_oid* oid, *document* text, OUT *tokid* integer, OUT *token* text)
返回类型	setof record
描述	测试解析器
示例	ts_parse(3722, 'foo - bar')
结果	(1,foo) ...
函数	ts_token_type(*parser_name* text, OUT *tokid* integer, OUT *alias* text, OUT *description* text)
返回类型	setof record
描述	获取解析器定义的词元类型
示例	ts_token_type('default')
结果	(1,asciiword,"Word, all ASCII") ...
函数	ts_token_type(*parser_oid* oid, OUT *tokid* integer, OUT *alias* text, OUT *description* text)
返回类型	setof record
描述	获取解析器定义的词元类型
示例	ts_token_type(3722)
结果	(1,asciiword,"Word, all ASCII") ...
函数	ts_stat(*sqlquery* text, [*weights* text,] OUT *word* text, OUT *ndoc* integer, OUT *nentry* integer)
返回类型	setof record
描述	获取 tsvector 列的统计信息
示例	ts_stat('SELECT vector from apod')
结果	(foo,10,15) ...

9.14. XML 函数

本节中描述的函数以及类函数表达式都作用于类型 xml 的值。关于 xml 类型的详细信息请查阅第 8.13 节。用于在值与类型 xml 之间转换的类函数表达式 xmlparse 和 xmlserialize 记录在该节中，这里不再重复。使用这些函数中的大多数要求安装时使用了configure --with-libxml编译。

9.14.1. 产生 XML 内容

有一组函数和类函数的表达式可以用来从 SQL 数据产生 XML 内容。它们特别适合于将查询结果格式化成 XML 文档以便于在客户端应用中处理。

9.14.1.1. xmlcomment

xmlcomment(*text*)

函数 xmlcomment 创建一个 XML 值，其中包含以指定文本为内容的 XML 注释。该文本不能包含“--”，也不能以“-”结尾，以确保构造出的 XML 注释有效。如果参数为空值，结果也为空值。

示例：

```
SELECT xmlcomment('hello');

  xmlcomment
--------------
 <!--hello-->
```

9.14.1.2. xmlconcat

```
xmlconcat(xml[, ...])
```

函数xmlconcat将由各个独立 XML 值组成的列表串接成一个单独的值，这个值包含一个 XML 内容片断。空值会被忽略，只有当没有参数为非空时结果才为空。

示例：

```
SELECT xmlconcat('<abc/>', '<bar>foo</bar>');

      xmlconcat
----------------------
 <abc/><bar>foo</bar>
```

如果 XML 声明存在，它们会按照下面的方式被组合。如果所有的参数值都有相同的 XML 版本声明，该版本将被用在结果中，否则将不使用版本。如果所有参数值有独立声明值“yes”，那么该值将被用在结果中。
如果所有参数值都有一个独立声明值并且至少有一个为“no”，则该值被用在结果中。
否则结果中将没有独立声明。如果结果被决定要要求一个独立声明但是没有版本声明，
将会使用一个版本 1.0 的版本声明，因为 XML 要求一个 XML 声明要包含一个版本声明。编码声明会被忽略并且在所有情况中都会被移除。

示例：

```
SELECT xmlconcat('<?xml version="1.1"?><foo/>', '<?xml version="1.
1" standalone="no"?><bar/>');

             xmlconcat
-----------------------------------
 <?xml version="1.1"?><foo/><bar/>
```

9.14.1.3. xmlelement

```
xmlelement(name name [, xmlattributes(value [AS attname] [, ... ])]
 [, content, ...])
```

表达式 xmlelement 使用给定的名称、属性和内容生成一个 XML 元素。

示例：

```
SELECT xmlelement(name foo);

 xmlelement
------------
 <foo/>

SELECT xmlelement(name foo, xmlattributes('xyz' as bar));
```

```
    xmlelement
------------------
 <foo bar="xyz"/>

SELECT xmlelement(name foo, xmlattributes(current_date as bar),
 'cont', 'ent');

             xmlelement
-------------------------------------
 <foo bar="2007-01-26">content</foo>
```

不是合法 XML 名字的元素名和属性名将被转义，转义的方法是将不合法的字符用序列_x*HHHH*_替换，其中*HHHH*是被替换字符的 Unicode 代码点的十六进制表示。例如：

```
SELECT xmlelement(name "foo$bar", xmlattributes('xyz' as "a&b"));

            xmlelement
----------------------------------
 <foo_x0024_bar a_x0026_b="xyz"/>
```

如果属性值是一个列引用，则不需要指定一个显式的属性名，
在这种情况下列的名字将被默认用于属性的名字。在其他情况下，
属性必须被给定一个显式名称。因此这个示例是合法的：

```
CREATE TABLE test (a xml, b xml);
SELECT xmlelement(name test, xmlattributes(a, b)) FROM test;
```

但是下面这些不合法：

```
SELECT xmlelement(name test, xmlattributes('constant'), a, b) FROM
 test;
SELECT xmlelement(name test, xmlattributes(func(a, b))) FROM test;
```

如果指定了元素内容，它们将被根据其数据类型格式化。如果内容本身也是类型`xml`，
就可以构建复杂的 XML 文档。例如：

```
SELECT xmlelement(name foo, xmlattributes('xyz' as bar),
                            xmlelement(name abc),
                            xmlcomment('test'),
                            xmlelement(name xyz));

                  xmlelement
----------------------------------------------
 <foo bar="xyz"><abc/><!--test--><xyz/></foo>
```

其他类型的内容将被格式化为合法的 XML 字符数据。这意味着字符 <、> 和 & 将被转换为实体。二进制数据（数据类型`bytea`）将被表示成 base64 或十六进制编码，具体取决于配置参数xmlbinary的设置。个别数据类型的特殊行为将不断发展，以使SQL和Postgre
SQL数据类型与 XML Schema 规范对齐，届时将给出更精确的描述。

9.14.1.4. xmlforest

```
xmlforest(content [AS name] [, ...])
```

表达式 `xmlforest` 使用给定的名称和内容生成由元素构成的 XML 森林（序列）。

示例：

```
SELECT xmlforest('abc' AS foo, 123 AS bar);

          xmlforest
------------------------------
```

```
 <foo>abc</foo><bar>123</bar>

SELECT xmlforest(table_name, column_name)
FROM information_schema.columns
WHERE table_schema = 'pg_catalog';

                                          xmlforest
-----------------------------------------------------------------
------------------------
 <table_name>pg_authid</table_name><column_name>rolname</column_
name>
 <table_name>pg_authid</table_name><column_name>rolsuper</column_
name>
 ...
```

如第二个示例所示，如果内容值是一个列引用，可以省略元素名称，此时默认使用列名。否则，必须指定名称。

如上文`xmlelement`所示，非法 XML 名字的元素名会被转义。相似地，内容数据也会被转义来产生合法的 XML 内容，除非它已经是一个`xml`类型。

注意如果 XML 森林由多于一个元素组成，那么它不是合法的 XML 文档，因此在`xmlelement`中包装`xmlforest`表达式会有用处。

9.14.1.5. xmlpi

```
xmlpi(name target [, content])
```

表达式 `xmlpi` 创建一个 XML 处理指令。如果提供了内容，其中不得包含字符序列 `?>`。

示例：

```
SELECT xmlpi(name php, 'echo "hello world";');

            xmlpi
-----------------------------
 <?php echo "hello world";?>
```

9.14.1.6. xmlroot

```
xmlroot(xml, version text | no value [, standalone yes|no|no
 value])
```

表达式`xmlroot`修改一个 XML 值的根结点的属性。如果指定了一个版本，它会替换根节点的版本声明中的值；如果指定了一个独立设置，它会替换根节点的独立声明中的值。

```
SELECT xmlroot(xmlparse(document '<?xml version="1.1"?
><content>abc</content>'),
               version '1.0', standalone yes);

                xmlroot
----------------------------------------
 <?xml version="1.0" standalone="yes"?>
 <content>abc</content>
```

9.14.1.7. xmlagg

```
xmlagg(xml)
```

和这里描述的其他函数不同，函数`xmlagg`是一个聚合函数。
它将聚合函数调用的输入值串接起来，非常像`xmlconcat`所做的事情，

除了串接是跨行发生的而不是在单一行的多个表达式上发生。
聚合表达式的更多信息请见第 9.20 节。

示例：

```
CREATE TABLE test (y int, x xml);
INSERT INTO test VALUES (1, '<foo>abc</foo>');
INSERT INTO test VALUES (2, '<bar/>');
SELECT xmlagg(x) FROM test;
        xmlagg
----------------------
 <foo>abc</foo><bar/>
```

为了决定串接的顺序，可以为聚合调用增加一个`ORDER BY`子句，如第 4.2.7 节中所述。例如：

```
SELECT xmlagg(x ORDER BY y DESC) FROM test;
        xmlagg
----------------------
 <bar/><foo>abc</foo>
```

我们推荐在以前的版本中使用下列非标准方法，并且它们在特定情况下仍然有用：

```
SELECT xmlagg(x) FROM (SELECT * FROM test ORDER BY y DESC) AS tab;
        xmlagg
----------------------
 <bar/><foo>abc</foo>
```

9.14.2. XML 谓词

这一节描述的表达式检查`xml`值的属性。

9.14.2.1. IS DOCUMENT

```
xml IS DOCUMENT
```

如果参数 XML 值是一个正确的 XML 文档，则`IS DOCUMENT`返回真，如果不是则返回假（即它是一个内容片断），或者是参数为空时返回空。文档和内容片断之间的区别请见第 8.13 节。

9.14.2.2. XMLEXISTS

```
XMLEXISTS(text PASSING [BY REF] xml [BY REF])
```

如果第一个参数中的XPath表达式返回任何节点，函数`xmlexists`返回true，
否则返回false。（如果任一参数为空，则结果为空。）

示例：

```
SELECT xmlexists('//town[text() = ''Toronto'']' PASSING BY REF
 '<towns><town>Toronto</town><town>Ottawa</town></towns>');

 xmlexists
------------
 t
(1 row)
```

`BY REF` 子句在 PostgreSQL 中没有效果，但为了 SQL 一致性以及与其他实现的兼容性，允许使用它们。按照 SQL 标准，第一个 `BY REF` 是必需的，第二个是可选的。另请注意，SQL 标准规定 `xmlexists` 结构接受 XQuery 表达式作为第一个参数，但 PostgreSQL 目前只支持 XPath，它是 XQuery 的一个子集。

9.14.2.3. xml_is_well_formed

```
xml_is_well_formed(text)
xml_is_well_formed_document(text)
xml_is_well_formed_content(text)
```

这些函数检查一个text串是不是一个良构的 XML，返回一个布尔结果。xml_is_well_formed_document检查一个良构的文档，而xml_is_well_formed_content检查良构的内容。如果xmloption配置参数被设置为DOCUMENT，xml_is_well_formed会做第一个函数的工作；如果配置参数被设置为CONTENT，它会做第二个函数的工作。这意味着xml_is_well_formed对于检查一个到类型xml的简单类型转换是否会成功非常有用，而其他两个函数对于检查XMLPARSE的对应变体是否会成功有用。

示例：

```
SET xmloption TO DOCUMENT;
SELECT xml_is_well_formed('<>');
 xml_is_well_formed
--------------------
 f
(1 row)

SELECT xml_is_well_formed('<abc/>');
 xml_is_well_formed
--------------------
 t
(1 row)

SET xmloption TO CONTENT;
SELECT xml_is_well_formed('abc');
 xml_is_well_formed
--------------------
 t
(1 row)

SELECT xml_is_well_formed_document('<pg:foo xmlns:pg="http://
postgresql.org/stuff">bar</pg:foo>');
 xml_is_well_formed_document
-----------------------------
 t
(1 row)

SELECT xml_is_well_formed_document('<pg:foo xmlns:pg="http://
postgresql.org/stuff">bar</my:foo>');
 xml_is_well_formed_document
-----------------------------
 f
(1 row)
```

最后一个示例显示了检查是否正确匹配命名空间。

9.14.3. 处理 XML

为了处理xml数据类型的值，PostgreSQL 提供了xpath和xpath_exists函数，它们计算XPath 1.0 表达式。

```
xpath(xpath, xml [, nsarray])
```

函数xpath针对 XML 值*xml*计算 XPath 表达式*xpath*（一个text值）。它返回一个由 XPath 表达式产生的节点集所对应的 XML 值组成的数组。如果 XPath 表达式返回的是标量值而不是节点集，则返回一个单元素数组。

第二个参数必须是一个良构的 XML 文档。特殊地，它必须有一个单一根结点元素。

该函数可选的第三个参数是一个名字空间映射的数组。这个数组应该是一个二维text数组，其第二轴长度等于2（即它应该是一个数组的数组，其中每一个都由刚好 2 个元素组成）。每个数组项的第一个元素是名字空间的名称（别名），第二个元素是名字空间的 URI。并不要求在这个数组中提供的别名和在 XML 文档本身中使用的那些名字空间相同（换句话说，在 XML 文档中和在xpath函数环境中，别名都是本地的）。

示例：

```
SELECT xpath('/my:a/text()', '<my:a xmlns:my="http://example.
com">test</my:a>',
             ARRAY[ARRAY['my', 'http://example.com']]);

 xpath
--------
 {test}
(1 row)
```

要处理默认（匿名）命名空间，做这样的事情：

```
SELECT xpath('//mydefns:b/text()', '<a xmlns="http://example.
com"><b>test</b></a>',
             ARRAY[ARRAY['mydefns', 'http://example.com']]);

 xpath
--------
 {test}
(1 row)
```

xpath_exists(*xpath*, *xml* [, *nsarray*])

函数xpath_exists是xpath函数的一个特殊形式。它不返回满足 XPath 的各个 XML 值，而是返回一个布尔值，指示查询是否被满足。这个函数等价于标准的XMLEXISTS谓词，只是它还支持名称空间映射参数。

示例：

```
SELECT xpath_exists('/my:a/text()', '<my:a xmlns:my="http://
example.com">test</my:a>',
                    ARRAY[ARRAY['my', 'http://example.com']]);
 xpath_exists
--------------
 t
(1 row)
```

9.14.4. 将表映射到 XML

以下函数将关系表的内容映射为 XML 值，可以将它们视为 XML 导出功能：

```
table_to_xml(tbl regclass, nulls boolean, tableforest boolean,
 targetns text)
query_to_xml(query text, nulls boolean, tableforest boolean,
 targetns text)
cursor_to_xml(cursor refcursor, count int, nulls boolean,
              tableforest boolean, targetns text)
```

每个函数的返回类型都是xml。

table_to_xml映射由参数*tbl*传递的命名表的内容。regclass类型接受使用常见标记标识表的字符串，包括可选的模式限定和双引号。query_to_xml执行由参数*query*传递的查询并且映射结果集。cursor_to_xml从*cursor*指定的游标中取出指定数量的行。如果需要映射一个大型的表，我们推荐这种变体，因为每一个函数都是在内存中构建结果值的。

如果*tableforest*为假，则结果的 XML 文档看起来像这样：

```
<tablename>
  <row>
    <columnname1>data</columnname1>
    <columnname2>data</columnname2>
  </row>

  <row>
    ...
  </row>

  ...
</tablename>
```

如果*tableforest*为真，结果是一个看起来像这样的 XML 内容片断：

```
<tablename>
  <columnname1>data</columnname1>
  <columnname2>data</columnname2>
</tablename>

<tablename>
  ...
</tablename>

...
```

如果没有表名可用，在映射一个查询或一个游标时，在第一种格式中使用串table，在第二种格式中使用row。

这几种格式的选择由用户决定。第一种格式是一个正确的 XML 文档，它在很多应用中都很重要。如果结果值要被重组为一个文档，第二种格式在cursor_to_xml函数中更有用。前文讨论的产生 XML 内容的函数（特别是xmlelement）可以被用来把结果修改成符合用户的要求。

数据值会被以前文的函数xmlelement中描述的相同方法映射。

参数*nulls*决定空值是否会被包含在输出中。如果为真，列中的空值被表示为：

```
<columnname xsi:nil="true"/>
```

其中xsi是 XML 模式实例的 XML 名字空间前缀。一个合适的名字空间声明将被加入到结果值中。如果为假，包含空值的列将被从输出中忽略掉。

参数*targetns*指定想要的结果的 XML 名字空间。如果没有想要的特定名字空间，将会传递一个空串。

以下函数返回 XML Schema 文档，描述上面相应函数所执行的映射：

```
table_to_xmlschema(tbl regclass, nulls boolean, tableforest
 boolean, targetns text)
query_to_xmlschema(query text, nulls boolean, tableforest boolean,
 targetns text)
```

```
cursor_to_xmlschema(cursor refcursor, nulls boolean, tableforest
 boolean, targetns text)
```

必须传入相同的参数，才能得到彼此匹配的 XML 数据映射和 XML Schema 文档。

以下函数在同一个文档（或森林）中生成 XML 数据映射和对应的 XML Schema，并将两者链接在一起。当需要自包含且自描述的结果时，这些函数很有用：

```
table_to_xml_and_xmlschema(tbl regclass, nulls boolean, tableforest
 boolean, targetns text)
query_to_xml_and_xmlschema(query text, nulls boolean, tableforest
 boolean, targetns text)
```

此外，还可以使用以下函数，为整个模式或整个当前数据库生成类似的映射：

```
schema_to_xml(schema name, nulls boolean, tableforest boolean,
 targetns text)
schema_to_xmlschema(schema name, nulls boolean, tableforest
 boolean, targetns text)
schema_to_xml_and_xmlschema(schema name, nulls boolean, tableforest
 boolean, targetns text)

database_to_xml(nulls boolean, tableforest boolean, targetns text)
database_to_xmlschema(nulls boolean, tableforest boolean, targetns
 text)
database_to_xml_and_xmlschema(nulls boolean, tableforest boolean,
 targetns text)
```

注意，这些函数可能生成大量数据，需要在内存中构建。
请求大型模式或数据库的内容映射时，可以考虑分别映射各个表，甚至通过游标来完成。

一个模式内容映射的结果看起来像这样：

```
<schemaname>

table1-mapping

table2-mapping

...

</schemaname>
```

其中一个表映射的格式取决于上文解释的*tableforest*参数。

一个数据库内容映射的结果看起来像这样：

```
<dbname>

<schema1name>
  ...
</schema1name>

<schema2name>
  ...
</schema2name>

...

</dbname>
```

其中的模式映射如上所述。

作为一个使用这些函数产生的输出的示例，图 9.1展示了一个 XSLT 样式表，它将`table_to_xml_and_xmlschema`的输出转换为一个包含表数据的扁平转印的 HTML 文档。以一种相似的方式，这些函数的结果可以被转换成其他基于 XML 的格式。

图 9.1. 转换 SQL/XML 输出到 HTML 的 XSLT 样式表

```
<?xml version="1.0"?>
<xsl:stylesheet version="1.0"
    xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema"
    xmlns="http://www.w3.org/1999/xhtml"
>

  <xsl:output method="xml"
      doctype-system="http://www.w3.org/TR/xhtml1/DTD/xhtml1-
strict.dtd"
      doctype-public="-//W3C/DTD XHTML 1.0 Strict//EN"
      indent="yes"/>

  <xsl:template match="/*">
    <xsl:variable name="schema" select="//xsd:schema"/>
    <xsl:variable name="tabletypename"
                  select="$schema/xsd:element[@name=name(current())
]/@type"/>
    <xsl:variable name="rowtypename"
                  select="$schema/xsd:complexType[@name=
$tabletypename]/xsd:sequence/xsd:element[@name='row']/@type"/>

    <html>
      <head>
        <title><xsl:value-of select="name(current())"/></title>
      </head>
      <body>
        <table>
          <tr>
            <xsl:for-each select="$schema/xsd:complexType[@name=
$rowtypename]/xsd:sequence/xsd:element/@name">
              <th><xsl:value-of select="."/></th>
            </xsl:for-each>
          </tr>

          <xsl:for-each select="row">
            <tr>
              <xsl:for-each select="*">
                <td><xsl:value-of select="."/></td>
              </xsl:for-each>
            </tr>
          </xsl:for-each>
        </table>
      </body>
    </html>
  </xsl:template>

</xsl:stylesheet>
```

9.15. JSON 函数

表 9.40展示了可用于创建 JSON（见第 8.17 节）数据的函数。

表 9.40. JSON 支持函数

函数	描述	示例	示例结果
`array_to_json(anyarray [, pretty_bool])`	把数组作为 JSON 返回。PostgreSQL 多维数组会变成 JSON 数组的数组。如果 *pretty_bool* 为真，将在第 1 维元素之间添加换行符。	`array_to_json('{{1,5},{99,100}}'::int[])`	`[[1,5],[99,100]]`
`row_to_json(record [, pretty_bool])`	把行作为 JSON 返回。如果 *pretty_bool* 为真，将在第 1 层元素之间添加换行符。	`row_to_json(row(1,'foo'))`	`{"f1":1,"f2":"foo"}`

9.16. 序列操作函数

本节描述用于操作序列对象（也称为序列生成器，或简称序列）的函数。序列对象是使用CREATE SEQUENCE创建的特殊单行表。序列对象通常用于为表中的行生成惟一标识符。在表 9.41中列出的序列函数，提供了简单的、多用户安全方法，用于从序列对象中获取连续的序列值。

表 9.41. 序列函数

函数	返回类型	描述
`currval(regclass)`	`bigint`	返回最近对指定序列调用 `nextval` 得到的值
`lastval()`	`bigint`	返回最近对任意序列调用 `nextval` 得到的值
`nextval(regclass)`	`bigint`	推进序列并返回新值
`setval(regclass, bigint)`	`bigint`	设置序列的当前值
`setval(regclass, bigint, boolean)`	`bigint`	设置序列的当前值和 `is_called` 标志

序列函数要操作的序列由一个`regclass`参数指定，该参数就是序列在`pg_class`系统目录中的 OID。不过，不必手动查找 OID，因为`regclass`数据类型的输入转换函数会替你完成这项工作。只需将序列名用单引号括起来，使其看上去像一个字面常量。为兼容普通SQL名称的处理方式，除非序列名被双引号括起，否则字符串会转换为小写。因此：

`nextval('foo')`	操作序列 *foo*
`nextval('FOO')`	操作序列 *foo*
`nextval('"Foo"')`	操作序列 *Foo*

必要时，序列名可以带模式限定：

`nextval('myschema.foo')`	操作 *myschema.foo*
`nextval('"myschema".foo')`	同上
`nextval('foo')`	在搜索路径中查找 *foo*

参见第 8.18 节以获取更多有关以下类型的信息：`regclass`。

> **注意**
>
> 在 PostgreSQL 8.1 之前，序列函数的参数类型是 `text`，而不是 `regclass`；上述从文本字符串到 OID 值的转换会在每次调用时于运行期间执行。为保持向后兼容，这种能力仍然存在，但内部现在会在调用函数前，通过从 `text` 到 `regclass` 的隐式强制转换来处理。

> 将序列函数的参数写成不加修饰的字符串字面量时，它会成为以下类型的常量：regclass。由于它实际上只是一个 OID，即使后来发生重命名、模式变更等情况，它仍会指向最初标识的序列。这种“早绑定”
> 行为通常适用于列默认值和视图中的序列引用。但有时你可能希望采用“后绑定”，在运行时解析序列引用。要获得后绑定行为，应强制将常量存储为text常量，而不是regclass：
>
> `nextval('foo'::text)` *foo*运行时被查找
>
> 请注意，后绑定是下列旧版本唯一支持的行为：PostgreSQL8.1 之前的版本，因此可能需要这样做来保留旧应用的语义。
>
> 当然，序列函数的参数既可以是常量，也可以是表达式。如果它是文本表达式，隐式强制转换就会导致运行时查找。

可用的序列函数如下：

`nextval`

将序列对象推进到下一个值并返回该值。这个操作是原子的：即使多个会话并发执行 nextval，每个会话也会安全地获得一个不同的序列值。

如果使用默认参数创建序列对象，连续调用 nextval 会返回从 1 开始的连续值。可以在CREATE SEQUENCE命令中使用特殊参数获得其他行为；更多信息请参见该命令的参考页。

> **重要**
>
> 为了避免阻塞从同一序列获取数值的并发事务，nextval操作永远不会回滚；也就是说，一旦一个值被取出，它就被视为已使用，即使执行nextval的事务随后中止也是如此。因此，
> 中止的事务可能在序列的已分配值中留下未使用的“空洞”。

`currval`

返回当前会话中最近一次针对该序列调用 nextval 所获得的值。（如果当前会话从未针对该序列调用过 nextval，则会报错。）由于返回的是会话局部值，无论其他会话是否在当前会话调用之后执行过 nextval，结果都是可预测的。

`lastval`

返回当前会话中最近一次调用 nextval 所返回的值。此函数与 currval 相同，但不接受序列名参数，而是引用当前会话中最近一次调用 nextval 时所操作的序列。如果当前会话尚未调用过 nextval，调用 lastval 会报错。

`setval`

重置序列对象的计数器值。双参数形式将序列的last_value字段设为指定值，并将其is_called字段设为true，这意味着下次调用nextval会在返回值之前推进序列。currval报告的值也会设为指定值。在三参数形式中，is_called可以设为true或false。true与双参数形式具有相同的效果。如果将它设为false，下次调用nextval将恰好返回指定值，并从再下一次调用nextval时开始推进序列。此外，在这种情况下，currval报告的值不会改变。例如：

`SELECT setval('foo', 42);`	下一次 *nextval* 将返回 *43*
`SELECT setval('foo', 42, true);`	同上
`SELECT setval('foo', 42, false);`	下一次 *nextval* 将返回 *42*

函数setval返回的结果就是其第二个参数的值。

> **重要**
>
> 由于序列是非事务性的，如果事务回滚，`setval`所做的更改不会被撤销。

9.17. 条件表达式

本节描述在PostgreSQL中可用的SQL兼容的条件表达式。

> **提示**
>
> 如果你的需求超过这些条件表达式的能力，
> 你可能会希望用一种更富表现力的编程语言写一个存储过程。

9.17.1. CASE

SQL `CASE`表达式是一种通用的条件表达式，类似于其它编程语言中的 if/else 语句：

```
CASE WHEN condition THEN result
     [WHEN ...]
     [ELSE result]
END
```

`CASE`子句可以用于任何表达式可以出现的地方。每一个*condition*是一个返回boolean结果的表达式。如果结果为真，那么`CASE`表达式的结果就是紧随该条件后的*result*，并且剩下的`CASE`表达式不会被处理。如果条件的结果不为真，那么以相同方式搜寻任何随后的`WHEN`子句。如果没有`WHEN` *condition*为真，那么`CASE`表达式的值就是在`ELSE`子句里的*result*。如果省略了`ELSE`子句而且没有条件为真，结果为null。

示例：

```
SELECT * FROM test;

 a
---
 1
 2
 3

SELECT a,
       CASE WHEN a=1 THEN 'one'
            WHEN a=2 THEN 'two'
            ELSE 'other'
       END
    FROM test;

 a | case
---+-------
 1 | one
 2 | two
 3 | other
```

所有*result*表达式的数据类型都必须可以转换成单一的输出类型。参阅第 10.5 节获取细节。

下面这个“简单”形式的CASE表达式是上述通用形式的一个变种：

```
CASE expression
    WHEN value THEN result
    [WHEN ...]
    [ELSE result]
END
```

第一个*expression*会被计算，然后与所有在WHEN子句中的每一个*value*对比，直到找到一个相等的。如果没有找到匹配的，则返回在ELSE子句中的*result*（或者 null 值）。这类似于 C 里的switch语句。

上面的示例可以用简单CASE语法来写：

```
SELECT a,
       CASE a WHEN 1 THEN 'one'
              WHEN 2 THEN 'two'
              ELSE 'other'
       END
    FROM test;

 a | case
---+-------
 1 | one
 2 | two
 3 | other
```

CASE表达式并不计算任何无助于判断结果的子表达式。例如，下面是一个可以避免被零除错误的方法：

```
SELECT ... WHERE CASE WHEN x <> 0 THEN y/x > 1.5 ELSE false END;
```

> **注意**
>
> 如第 4.2.14 节所述，在多种情况下，表达式中的子表达式会在不同阶段求值，因此“CASE只计算必要的子表达式”这一原则并非绝对成立。例如，常量子表达式1/0通常会在规划时导致除零错误，即便它位于一个运行时永远不会进入的CASE分支中也是如此。

9.17.2. COALESCE

```
COALESCE(value [, ...])
```

COALESCE函数返回其参数中第一个非空的值。只有当所有参数都为空时才返回空。它常用于在为显示而检索数据时为空值替换一个默认值，例如：

```
SELECT COALESCE(description, short_description, '(none)') ...
```

这会在 description 非空时返回它，否则在 short_description 非空时返回它，否则返回 (none)。

和CASE表达式一样，COALESCE只计算确定结果所需的参数；也就是说，在第一个不为 null 的参数右边的参数不会被计算。这个 SQL 标准函数提供了类似于NVL和IFNULL的能力，它们被用在某些其他数据库系统中。

9.17.3. NULLIF

```
NULLIF(value1, value2)
```

当*value1*和*value2*相等时，NULLIF返回 null。否则它返回*value1*。这些可以用于执行前文给出的COALESCE示例的逆操作：

```
SELECT NULLIF(value, '(none)') ...
```

在这个示例中，如果value是(none)，将返回 null，否则返回value的值。

9.17.4. GREATEST和LEAST

GREATEST(*value* [, ...])

LEAST(*value* [, ...])

GREATEST和LEAST函数从一个任意数量的表达式列表中选出最大或最小的值。
这些表达式必须都能转换为一个公共数据类型，该类型也就是结果的类型（细节见第 10.5 节）。列表中的空值会被忽略。只有当所有表达式都计算为空时，结果才为空。

注意，GREATEST和LEAST不在 SQL 标准中，但它们是一种常见的扩展。有些其他数据库让它们在任一参数为空时就返回空，而不是只在全部参数都为空时才返回空。

9.18. 数组函数和操作符

表 9.42列出了可用于数组类型的操作符。

表 9.42. 数组操作符

操作符	描述	示例	结果
=	等于	ARRAY[1.1,2.1,3.1]::int[] = ARRAY[1,2,3]	t
<>	不等于	ARRAY[1,2,3] <> ARRAY[1,2,4]	t
<	小于	ARRAY[1,2,3] < ARRAY[1,2,4]	t
>	大于	ARRAY[1,4,3] > ARRAY[1,2,4]	t
<=	小于等于	ARRAY[1,2,3] <= ARRAY[1,2,3]	t
>=	大于等于	ARRAY[1,4,3] >= ARRAY[1,4,3]	t
@>	包含	ARRAY[1,4,3] @> ARRAY[3,1,3]	t
<@	被包含	ARRAY[2,2,7] <@ ARRAY[1,7,4,2,6]	t
&&	重叠（有公共元素）	ARRAY[1,4,3] && ARRAY[2,1]	t
||	数组与数组串接	ARRAY[1,2,3] || ARRAY[4,5,6]	{1,2,3,4,5,6}
||	数组与数组串接	ARRAY[1,2,3] || ARRAY[[4,5,6],[7,8,9]]	{{1,2,3},{4,5,6},{7,8,9}}
||	元素与数组串接	3 || ARRAY[4,5,6]	{3,4,5,6}
||	数组与元素串接	ARRAY[4,5,6] || 7	{4,5,6,7}

数组比较操作符逐个元素地比较数组内容，使用元素数据类型的默认 B-树比较函数。多维数组的元素按行优先顺序访问（最后一个下标变化最快）。如果两个数组的内容相同，

但维度信息不同，则由维度信息中的第一个差异决定排序顺序。（这与 PostgreSQL 8.2 之前的版本不同：旧版本会将内容相同的两个数组视为相等，即使它们的维数或下标范围不同。）

参阅第 8.14 节获取有关数组操作符行为的更多细节。

表 9.43展示了可以用于数组类型的函数。参阅第 8.14 节获取更多信息以及使用这些函数的示例。

表 9.43. 数组函数

函数	`array_append(anyarray, anyelement)`
返回类型	`anyarray`
描述	向数组末尾追加一个元素
示例	`array_append(ARRAY[1,2], 3)`
结果	`{1,2,3}`
函数	`array_cat(anyarray, anyarray)`
返回类型	`anyarray`
描述	串接两个数组
示例	`array_cat(ARRAY[1,2,3], ARRAY[4,5])`
结果	`{1,2,3,4,5}`
函数	`array_ndims(anyarray)`
返回类型	`int`
描述	返回数组的维数
示例	`array_ndims(ARRAY[[1,2,3], [4,5,6]])`
结果	`2`
函数	`array_dims(anyarray)`
返回类型	`text`
描述	返回数组维度的文本表示
示例	`array_dims(ARRAY[[1,2,3], [4,5,6]])`
结果	`[1:2][1:3]`
函数	`array_fill(anyelement, int[] [, int[]])`
返回类型	`anyarray`
描述	返回按给定值和维度初始化的数组，可选地使用不为 1 的下界
示例	`array_fill(7, ARRAY[3], ARRAY[2])`
结果	`[2:4]={7,7,7}`
函数	`array_length(anyarray, int)`
返回类型	`int`
描述	返回数组指定维度的长度
示例	`array_length(array[1,2,3], 1)`
结果	`3`

函数	`array_lower(anyarray, int)`
返回类型	`int`
描述	返回数组指定维度的下界
示例	`array_lower('[0:2]={1,2,3}'::int[], 1)`
结果	`0`
函数	`array_prepend(anyelement, anyarray)`
返回类型	`anyarray`
描述	向数组开头添加一个元素
示例	`array_prepend(1, ARRAY[2,3])`
结果	`{1,2,3}`
函数	`array_to_string(anyarray, text [, text])`
返回类型	`text`
描述	使用给定分隔符和可选的 null 替代字符串串接数组元素
示例	`array_to_string(ARRAY[1, 2, 3, NULL, 5], ',', '*')`
结果	`1,2,3,*,5`
函数	`array_upper(anyarray, int)`
返回类型	`int`
描述	返回数组指定维度的上界
示例	`array_upper(ARRAY[1,8,3,7], 1)`
结果	`4`
函数	`string_to_array(text, text [, text])`
返回类型	`text[]`
描述	使用给定分隔符和可选的 null 替代字符串将字符串拆分为数组元素
示例	`string_to_array('xx~^~yy~^~zz', '~^~', 'yy')`
结果	`{xx,NULL,zz}`
函数	`unnest(anyarray)`
返回类型	`setof anyelement`
描述	将数组展开为一组行
示例	`unnest(ARRAY[1,2])`

结果	`1`
	`2`
	(2 rows)

在 `string_to_array` 中，如果分隔符参数为 NULL，输入字符串中的每个字符都会成为结果数组中的独立元素。如果分隔符为空字符串，则将整个输入字符串作为单元素数组返回。否则，在分隔符字符串的每次出现处拆分输入字符串。

在 `string_to_array` 中，如果省略 null 替代字符串参数，或该参数为 NULL，则不会将任何输入子串替换为 NULL。在 `array_to_string` 中，如果省略 null 替代字符串参数，或该参数为 NULL，则直接跳过数组中的所有 null 元素，不在输出字符串中表示它们。

> **注意**
>
> `string_to_array` 的行为与 PostgreSQL 9.1 之前的版本有两点不同。首先，当输入字符串长度为零时，它返回空数组（零元素），而不是 NULL。其次，如果分隔符字符串为 NULL，该函数会将输入拆分为单独的字符，而不是像以前那样返回 NULL。

也可参见第 9.20 节了解用于数组的聚合函数`array_agg`。

9.19. 范围函数和操作符

范围类型的概述可参见第 8.16 节。

表 9.44列出了可用于范围类型的操作符。

表 9.44. 范围操作符

操作符	描述	示例	结果
=	等于	`int4range(1, 5) = '[1,4]'::int4range`	t
<>	不等于	`numrange(1.1,2.2) <> numrange(1.1, 2.3)`	t
<	小于	`int4range(1,10) < int4range(2,3)`	t
>	大于	`int4range(1,10) > int4range(1,5)`	t
<=	小于等于	`numrange(1.1,2.2) <= numrange(1.1, 2.2)`	t
>=	大于等于	`numrange(1.1,2.2) >= numrange(1.1, 2.0)`	t
@>	包含范围	`int4range(2,4) @> int4range(2,3)`	t
@>	包含元素	`'[2011-01-01,2011-03-01)'::tsrange @> '2011-01-10'::timestamp`	t
<@	范围被包含于	`int4range(2,4) <@ int4range(1,7)`	t
<@	元素被包含于	`42 <@ int4range(1,7)`	f
&&	重叠（有公共点）	`int8range(3,7) && int8range(4,12)`	t
<<	严格位于左侧	`int8range(1,10) << int8range(100, 110)`	t
>>	严格位于右侧	`int8range(50,60) >> int8range(20, 30)`	t
&<	不超出其右边界	`int8range(1,20) &< int8range(18, 20)`	t

操作符	描述	示例	结果
&>	不超出其左边界	int8range(7,20) &> int8range(5,10)	t
-|-	与之相邻	numrange(1.1,2.2) -|- numrange(2.2,3.3)	t
+	并集	numrange(5,15) + numrange(10,20)	[5,20)
*	交集	int8range(5,15) * int8range(10,20)	[10,15)
-	差集	int8range(5,15) - int8range(10,20)	[5,10)

简单比较操作符 <、>、<= 和 >= 首先比较下界，只有下界相等时才比较上界。这些比较对范围通常用处不大，但提供这些操作符可以在范围上建立 B-树索引。

涉及空范围时，左侧、右侧和相邻操作符总是返回假；也就是说，空范围不被视为位于任何其他范围之前或之后。

如果结果范围需要包含两个不相交的子范围，并集和差集操作符会失败，因为这样的范围无法表示。

表 9.45列出了可用于范围类型的函数。

表 9.45. 范围函数

函数	lower(anyrange)
返回类型	范围的元素类型
描述	范围的下界
示例	lower(numrange(1.1,2.2))
结果	1.1
函数	upper(anyrange)
返回类型	范围的元素类型
描述	范围的上界
示例	upper(numrange(1.1,2.2))
结果	2.2
函数	isempty(anyrange)
返回类型	boolean
描述	范围是否为空?
示例	isempty(numrange(1.1,2.2))
结果	false
函数	lower_inc(anyrange)
返回类型	boolean
描述	是否包含下界?
示例	lower_inc(numrange(1.1,2.2))
结果	true

函数	upper_inc(anyrange)
返回类型	boolean
描述	是否包含上界？
示例	upper_inc(numrange(1.1,2.2))
结果	false
函数	lower_inf(anyrange)
返回类型	boolean
描述	下界是否无穷？
示例	lower_inf('(,)'::daterange)
结果	true
函数	upper_inf(anyrange)
返回类型	boolean
描述	上界是否无穷？
示例	upper_inf('(,)'::daterange)
结果	true

如果范围为空，或请求的边界是无限的，lower 和 upper 函数返回 null。对于空范围，lower_inc、upper_inc、lower_inf 和 upper_inf 函数都返回假。

9.20. 聚合函数

聚合函数根据一组输入值计算出单个结果。内置聚合函数列于表 9.46和表 9.47。聚合函数的特殊语法说明见第 4.2.7 节。更多入门信息请参见第 2.7 节。

表 9.46. 通用聚合函数

函数	参数类型	返回类型	描述
array_agg(*expression*)	任意非数组类型	参数类型的数组	将输入值（包括 null）串接为数组
avg(*expression*)	smallint、int、bigint、real、double precision、numeric 或 interval	整数类型参数返回 numeric，浮点参数返回 double precision，其他情况与参数数据类型相同	所有输入值的平均值（算术平均值）
bit_and(*expression*)	smallint、int、bigint 或 bit	与参数数据类型相同	所有非空输入值的按位与；没有非空输入时为 null
bit_or(*expression*)	smallint、int、bigint 或 bit	与参数数据类型相同	所有非空输入值的按位或；没有非空输入时为 null
bool_and(*expression*)	bool	bool	所有输入值都为真时返回真，否则返回假
bool_or(*expression*)	bool	bool	至少一个输入值为真时返回真，否则返回假
count(*)		bigint	输入行数
count(*expression*)	任意	bigint	*expression* 的值不为 null 的输入行数

函数	参数类型	返回类型	描述
every(*expression*)	bool	bool	等价于 bool_and
max(*expression*)	任意数值、字符串、日期/时间、网络或枚举类型，或这些类型的数组	与参数类型相同	所有输入值中 *expression* 的最大值
min(*expression*)	任意数值、字符串、日期/时间、网络或枚举类型，或这些类型的数组	与参数类型相同	所有输入值中 *expression* 的最小值
string_agg(*expression*, *delimiter*)	(text, text) 或 (bytea, bytea)	与参数类型相同	将输入值串接为字符串，以分隔符分隔
sum(*expression*)	smallint、int、bigint、real、double precision、numeric、interval 或 money	smallint 或 int 参数返回 bigint，bigint 参数返回 numeric，其他情况与参数数据类型相同	对所有输入值的 *expression* 求和
xmlagg(*expression*)	xml	xml	串接 XML 值（另见第 9.14.1.7 节）

需要注意，除了count之外，这些函数在没有选中任何行时都会返回空值。特别地，sum在没有输入行时返回空值，而不是预期中的零；array_agg在没有输入行时返回空值，而不是空数组。必要时，可以用coalesce函数把空值替换成零或空数组。

> **注意**
>
> 布尔聚合bool_and和bool_or对应于标准 SQL 聚合every和any或some。至于any和some，标准语法似乎存在歧义：
>
> ```
> SELECT b1 = ANY((SELECT b2 FROM t2 ...)) FROM t1 ...;
> ```
>
> 此处的ANY既可以被视为引入一个子查询，也可以在该子查询返回一行布尔值时被视为聚合函数。因此，不能将标准名称用于这些聚合。

> **注意**
>
> 习惯于其他 SQL 数据库管理系统的用户，可能会对count聚合用于整个表时的性能感到失望。如下查询：
>
> ```
> SELECT count(*) FROM sometable;
> ```
>
> 将由PostgreSQL通过顺序扫描整个表来执行。

聚合函数array_agg、string_agg和xmlagg，以及类似的用户定义聚合函数，其结果值会随输入值的顺序发生实质性变化。默认情况下，输入顺序未指定，但可以在聚合调用中写入ORDER BY子句来控制，如第 4.2.7 节所示。也可以用已排序的子查询提供输入值，这通常也能奏效。例如：

```
SELECT xmlagg(x) FROM (SELECT x FROM test ORDER BY y DESC) AS tab;
```

但这种语法在 SQL 标准中是不允许的，并且不可移植到其他数据库系统。

表 9.47列出了统计分析常用的聚合函数。（将它们单独列出只是为了避免让更常用的聚合函数列表过于拥挤。）说明中提到的 *N* 表示所有输入表达式均非空的输入行数。在所有情况下，如果计算没有意义，例如 *N* 为零，都会返回空值。

表 9.47. 用于统计的聚合函数

函数	参数类型	返回类型	描述
corr(*Y*, *X*)	double precision	double precision	相关系数
covar_pop(*Y*, *X*)	double precision	double precision	总体协方差
covar_samp(*Y*, *X*)	double precision	double precision	样本协方差
regr_avgx(*Y*, *X*)	double precision	double precision	自变量的平均值（sum(*X*)/*N*）
regr_avgy(*Y*, *X*)	double precision	double precision	因变量的平均值（sum(*Y*)/*N*）
regr_count(*Y*, *X*)	double precision	bigint	两个表达式都非空的输入行数
regr_intercept(*Y*, *X*)	double precision	double precision	由（*X*, *Y*）数值对确定的最小二乘拟合线性方程的 y 轴截距
regr_r2(*Y*, *X*)	double precision	double precision	相关系数的平方
regr_slope(*Y*, *X*)	double precision	double precision	由（*X*, *Y*）数值对确定的最小二乘拟合线性方程的斜率
regr_sxx(*Y*, *X*)	double precision	double precision	sum(*X*^2) - sum(*X*)^2/*N*（自变量的“平方和”）
regr_sxy(*Y*, *X*)	double precision	double precision	sum(*X**Y) - sum(*X*) * sum(*Y*)/*N*（自变量与因变量的“乘积和”）
regr_syy(*Y*, *X*)	double precision	double precision	sum(*Y*^2) - sum(*Y*)^2/*N*（因变量的“平方和”）
stddev(*expression*)	smallint、int、bigint、real、double precision 或 numeric	浮点参数返回 double precision，其他情况返回 numeric	stddev_samp 的历史别名
stddev_pop(*expression*)	smallint、int、bigint、real、double precision 或 numeric	浮点参数返回 double precision，其他情况返回 numeric	输入值的总体标准差
stddev_samp(*expression*)	smallint、int、bigint、real、double precision 或 numeric	浮点参数返回 double precision，其他情况返回 numeric	输入值的样本标准差
variance(*expression*)	smallint、int、bigint、real、double precision 或 numeric	浮点参数返回 double precision，其他情况返回 numeric	var_samp 的历史别名
var_pop(*expression*)	smallint、int、bigint、real、double precision 或 numeric	浮点参数返回 double precision，其他情况返回 numeric	输入值的总体方差（总体标准差的平方）
var_samp(*expression*)	smallint、int、bigint、real、double precision 或 numeric	浮点参数返回 double precision，其他情况返回 numeric	输入值的样本方差（样本标准差的平方）

9.21. 窗口函数

窗口函数提供了对与当前查询行相关的一组行执行计算的能力。该特性的介绍请参见第 3.5 节。

内置的窗口函数列在表 9.48中。注意，这些函数必须使用窗口函数语法调用，也就是说，需要一个OVER子句。

除了这些函数外，
任何内置或用户定义的聚合函数都可以用作窗口函数（内置聚合函数列表见第 9.20 节）。聚合函数只有在调用后跟随 OVER 子句时才作为窗口函数；否则作为常规聚合。

表 9.48. 通用窗口函数

函数	返回类型	描述
row_number()	bigint	当前行在其分区内的编号，从 1 开始计数
rank()	bigint	当前行的排名，允许空缺；等于其第一个同等行的 row_number
dense_rank()	bigint	当前行的排名，没有空缺；此函数对同等行组计数
percent_rank()	double precision	当前行的相对排名：（rank - 1）/（总行数 - 1）
cume_dist()	double precision	当前行的相对排名：（当前行之前或与当前行同等的行数）/（总行数）
ntile(*num_buckets* integer)	integer	范围为 1 到参数值的整数，将分区尽可能均等地划分
lag(*value* anyelement [, *offset* integer [, *default* anyelement]])	与 *value* 相同的类型	返回在分区内当前行之前 *offset* 行处计算的 *value*；如果没有这样的行，则返回 *default*（其类型必须与 *value* 相同）。*offset* 和 *default* 都针对当前行求值。如果省略，*offset* 默认为 1，*default* 默认为 null
lead(*value* anyelement [, *offset* integer [, *default* anyelement]])	与 *value* 相同的类型	返回在分区内当前行之后 *offset* 行处计算的 *value*；如果没有这样的行，则返回 *default*（其类型必须与 *value* 相同）。*offset* 和 *default* 都针对当前行求值。如果省略，*offset* 默认为 1，*default* 默认为 null
first_value(*value* any)	与 *value* 相同的类型	返回在窗口帧第一行处计算的 *value*
last_value(*value* any)	与 *value* 相同的类型	返回在窗口帧最后一行处计算的 *value*
nth_value(*value* any, *nth* integer)	与 *value* 相同的类型	返回在窗口帧第 *nth* 行处计算的 *value*（从 1 开始计数）；没有这样的行时返回 null

在表 9.48中列出的所有函数都依赖于相关窗口定义的ORDER BY子句指定的排序顺序。在ORDER BY排序中不能区分的行被称为是同等行；这四个排名函数的定义使它们对任何两个同等行都给出相同的答案。

注意first_value、last_value和nth_value只考虑“窗口帧”内的行，
它默认情况下包含从分区的开始行直到当前行的最后一个同等行。这对last_value
可能不会给出有用的结果，有时对nth_value也一样。你可以通过向OVER

子句增加一个合适的帧声明（RANGE或ROWS）来重定义帧。
关于帧声明的更多信息请参考第 4.2.8 节。

当一个聚合函数被用作窗口函数时，它将在当前行的窗口帧内的行上聚合。一个使用ORDER BY和默认窗口帧定义的聚合产生一种“累计求和”类型的行为，
这可能是或者不是想要的结果。为了获取在整个分区上的聚合，省略ORDER BY或者使用ROWS BETWEEN UNBOUNDED PRECEDING AND UNBOUNDED FOLLOWING。其它窗口帧声明可以用来获得其它的效果。

注意

SQL 标准为lead、lag、first_value、last_value和nth_value定义了一个RESPECT NULLS或IGNORE NULLS选项。这在PostgreSQL中没有实现：行为总是与标准的默认相同，即RESPECT NULLS。同样，标准中用于nth_value的FROM FIRST或FROM LAST选项没有实现：只支持默认的FROM FIRST行为（你可以通过反转ORDER BY的排序达到FROM LAST的结果）。

9.22. 子查询表达式

本节描述PostgreSQL中可用的SQL兼容的子查询表达式。
本节介绍的所有表达式形式都返回布尔值（真/假）结果。

9.22.1. EXISTS

EXISTS (*subquery*)

EXISTS的参数是一个任意的SELECT语句，或者说子查询。
系统对子查询进行运算以判断它是否返回行。如果它至少返回一行，那么
EXISTS的结果就为“真”；如果子查询没有返回行，那么EXISTS的结果是“假”。

子查询可以引用外层查询的变量，这些变量在该子查询的任何一次计算中都起常量的作用。

这个子查询通常只是运行到能判断它是否可以返回至少一行为止，而不是等到全部结束。
在这里写任何有副作用的子查询都是不明智的（例如调用序列函数）；
这些副作用是否发生是很难判断的。

因为结果只取决于是否会返回行，而不取决于这些行的内容，
所以这个子查询的输出列表通常是无关紧要的。一个常用的编码习惯是用EXISTS(SELECT 1 WHERE ...)的形式写所有的EXISTS测试。不过这条规则有例外，例如那些使用INTERSECT的子查询。

下面这个简单的示例类似在col2上的一次内连接，但是它为每个 tab1的行最多生成一行输出，即使存在多个匹配tab2的行也如此：

```
SELECT col1
FROM tab1
WHERE EXISTS (SELECT 1 FROM tab2 WHERE col2 = tab1.col2);
```

9.22.2. IN

expression IN (*subquery*)

右手边是一个圆括号括起来的子查询，它必须恰好返回一列。左手边表达式会被求值，
并与子查询结果逐行比较。如果找到任意相等的子查询行，那么IN的结果就是“真”。
如果没有找到相等行，那么结果就是“假”（包括子查询不返回任何行的情况）。

请注意，如果左侧表达式得到空值，或者右侧没有相等的值且至少有一行得到空值，则IN结构的结果将是空值，而不是假。这符合 SQL 对空值布尔组合的一般规则。

和EXISTS一样，假定子查询一定会完整运行并不明智。

row_constructor IN (*subquery*)

这种形式的IN左手边是一个行构造器，如第 4.2.13 节中所述。右手边是一个圆括号括起来的子查询，它必须返回与左手边行中表达式数量完全相同的列数。左手边表达式会被求值，并与子查询结果逐行比较。如果找到任意相等的子查询行，则IN的结果为“真”。如果没有找到相等行，那么结果为“假”（包括子查询不返回任何行的情况）。

通常，表达式或者子查询行里的空值是按照 SQL 布尔表达式的一般规则进行组合的。如果两个行对应的成员都非空并且相等，那么认为这两行相等；如果任意对应成员为非空且不等，那么这两行不等；否则这样的行比较的结果是未知（空值）。如果所有行的结果要么是不等，要么是空值，并且至少有一个空值，那么IN的结果是空值。

9.22.3. NOT IN

expression NOT IN (*subquery*)

右手边是一个圆括号括起来的子查询，它必须恰好返回一列。左手边表达式会被求值，并与子查询结果逐行比较。如果只找到不相等的子查询行（包括子查询不返回任何行的情况），那么NOT IN的结果是“真”。如果找到任何相等行，则结果为“假”。

请注意，如果左侧表达式得到空值，或者右侧没有相等的值且至少有一行得到空值，则NOT IN结构的结果将是空值，而不是真。这符合 SQL 对空值布尔组合的一般规则。

和EXISTS一样，假定子查询一定会完整运行并不明智。

row_constructor NOT IN (*subquery*)

这种形式的NOT IN左侧是一个行构造器，如第 4.2.13 节中所述。右侧是一个用圆括号括起来的子查询，必须返回与左侧行中表达式数量完全相同的列数。对左侧表达式求值后，将其按行与子查询结果的每一行比较。如果只找到不相等的子查询行（包括子查询不返回任何行的情况），则NOT IN的结果为“真”。如果找到任何相等行，则结果为“假”。

通常，表达式或者子查询行里的空值是按照 SQL 布尔表达式的一般规则进行组合的。如果两个行对应的成员都非空并且相等，那么认为这两行相等；如果任意对应成员为非空且不等，那么这两行不等；否则这样的行比较的结果是未知（空值）。如果所有行的结果要么是不等，要么是空值，并且至少有一个空值，那么NOT IN的结果是空值。

9.22.4. ANY/SOME

expression operator ANY (*subquery*)
expression operator SOME (*subquery*)

这种形式的右侧是一个用括号括起来的子查询，它必须恰好返回一列。左侧表达式会被求值，并使用给定的 *operator* 与子查询结果的每一行进行比较。该操作符必须产生布尔结果。如果得到任何真值结果，那么ANY的结果就是“真”。如果没有找到真值结果，那么结果是“假”（包括子查询没有返回任何行的情况）。

SOME是ANY的同义词。IN等价于= ANY。

请注意，如果没有任何比较返回真，并且至少有一个右侧行的操作符结果为空值，则ANY结构的结果将是空值，而不是假。这符合 SQL 对空值布尔组合的一般规则。

和EXISTS一样，假定子查询一定会完整运行并不明智。

row_constructor operator ANY (*subquery*)
row_constructor operator SOME (*subquery*)

这种形式的ANY左侧是一个行构造器，如第 4.2.13 节所述。右侧是一个用括号括起来的子查询，它必须返回与左侧行中表达式数量完全相同的列数。左侧表达式会被求值，并使用给定的*operator*与子查询结果的每一行逐行比较。如果比较对任何子查询行返回真，则ANY的结果为“真”。如果比较对每一个子查询行都返回假，则结果为“假”（包括子查询不返回行的情况）。如果比较对任何行都不返回真，并且至少有一次比较返回 NULL，则结果为 NULL。

关于按行比较的详细含义请见第 9.23.5 节。

9.22.5. ALL

expression operator ALL (*subquery*)

右侧是一个用圆括号括起来的子查询，必须恰好返回一列。左侧表达式会被求值，并使用给定的*operator*与子查询结果的每一行进行比较。该操作符必须产生布尔结果。如果所有行都得到真（包括子查询不返回任何行的情况），则ALL的结果为“真”。如果得到任何假值结果，则结果为“假”。如果与任何子查询行的比较都不返回假，并且至少有一次比较返回 NULL，则结果为 NULL。

NOT IN等价于<> ALL。

和EXISTS一样，假定子查询一定会完整运行并不明智。

row_constructor operator ALL (*subquery*)

这种形式的 ALL 左侧是一个行构造器，如第 4.2.13 节所述。右侧是一个用括号括起来的子查询，它必须返回与左侧行中表达式一样多的列。左侧表达式会被求值，并使用给定的*operator* 与子查询结果逐行比较。如果该比较对所有子查询行都返回真，那么ALL的结果就是“真”（包括子查询没有返回任何行的情况）。如果对任何子查询行的比较返回假，则结果为“假”。如果比较对任何子查询行都不返回假，并且至少有一次比较返回 NULL，则结果为 NULL。

关于按行比较的详细含义请见第 9.23.5 节。

9.22.6. 单一行比较

row_constructor operator (*subquery*)

左侧是一个行构造器，如第 4.2.13 节所述。右侧是一个用圆括号括起来的子查询，必须返回与左侧行中表达式数量完全相同的列数。此外，该子查询不能返回超过一行；如果它返回零行，则结果为空值。对左侧求值后，将所得的行与子查询返回的唯一一行进行比较。

关于按行比较的详细含义请见第 9.23.5 节。

9.23. 行和数组比较

本节描述几个特殊的结构，用于在值的组之间进行多重比较。这些形式语法上和前面一节的子查询形式相关，但是不涉及子查询。涉及数组子表达式的形式是PostgreSQL的扩展；其余形式是SQL兼容的。本节介绍的所有表达式形式都返回布尔（Boolean）结果（真/假）。

9.23.1. IN

```
expression IN (value [, ...])
```

右侧是一个用圆括号括起来的标量表达式列表。
如果左侧表达式的结果等于右侧任一表达式的结果，则结果为“真”。这等价于以下写法：

```
expression = value1
OR
expression = value2
OR
...
```

请注意如果左手边表达式得到空值，
或者没有相等的右手边值并且至少有一个右手边的表达式得到空值，那么
IN结构的结果将为空值，而不是假。这符合 SQL 处理空值的布尔组合的一般规则。

9.23.2. NOT IN

```
expression NOT IN (value [, ...])
```

右侧是一个用圆括号括起来的标量表达式列表。
如果左侧表达式的结果与右侧所有表达式的结果都不相等，则结果为“真”。
这等价于以下写法：

```
expression <> value1
AND
expression <> value2
AND
...
```

请注意如果左手边表达式得到空值，
或者没有相等的右手边值并且至少有一个右手边的表达式得到空值，那么NOT IN结构的结果将为空值，而不是我们可能天真地认为的真值。这符合 SQL 处理空值的布尔组合的一般规则。

> **提示**
>
> x NOT IN y在所有情况下都等效于NOT (x IN y)。但是，在处理空值的时候，用NOT IN比用IN更可能迷惑新手。最好尽可能用正逻辑来表达你的条件。

9.23.3. ANY/SOME（数组）

```
expression operator ANY (array expression)
expression operator SOME (array expression)
```

右侧是一个用括号括起来的表达式，它必须产生一个数组值。左侧表达式会被求值，
并使用给定的*operator*与数组的每个元素进行比较，该操作符必须产生布尔结果。
如果得到了任何真值结果，那么ANY的结果是“真”。
如果没有找到真值结果（包括数组有零个元素的情况），那么结果是“假”。

如果数组表达式得到的是 null 数组，那么ANY的结果将为 null。如果左手边的表达式得到null，ANY通常也为 null（尽管非严格比较操作符可能得到不同结果）。另外，如果右手边数组包含任何 null 元素，并且没有得到真值比较结果，那么ANY的结果将为 null 而不是假（同样假设这里使用的是严格比较操作符）。这符合 SQL 处理 null 值布尔组合的一般规则。

SOME是ANY的同义词。

9.23.4. ALL（数组）

expression operator ALL (*array expression*)

右侧是一个用括号括起来的表达式，它必须产生一个数组值。左侧表达式会被求值，并使用给定的*operator*与数组的每个元素进行比较，该操作符必须产生布尔结果。如果所有比较都得到真值结果，那么ALL的结果是“真”（包括数组有零个元素的情况）。如果有任何假值结果，那么结果是“假”。

如果数组表达式得到的是 null 数组，那么ALL的结果将为 null。如果左手边的表达式得到 null，ALL通常也为 null（尽管非严格比较操作符可能得到不同结果）。另外，如果右手边数组包含任何 null 元素，并且没有得到假值比较结果，那么ALL的结果将为 null 而不是真（同样假设这里使用的是严格比较操作符）。这符合 SQL 处理 null 值布尔组合的一般规则。

9.23.5. 按行比较

row_constructor operator row_constructor

两侧都是行构造器，如第 4.2.13 节所述。两个行值必须具有相同数量的字段。对两侧分别求值后，按行进行比较。当 *operator* 为 =、<>、<、<=、>、>=，或具有与其中之一类似语义的操作符时，允许进行行比较。（具体而言，如果某个操作符是某个 B-树操作符类的成员，或者是某个 B-树操作符类中 = 成员的否定子，它就可以作为行比较操作符。）

=和<>情况略有不同。如果两行的所有对应成员都是非空且相等则这两行被认为相等；如果任何对应成员是非空但是不相等则这两行不相等；否则行比较的结果为未知（空值）。

对于<、<=、>和>=这几种情况，会从左到右比较各行元素，一旦找到一对不相等或含有 null 的元素就立即停止。如果这对元素中的任意一个为 null，那么行比较的结果就是未知（null）；否则，这对元素的比较结果决定整个行比较的结果。例如，ROW(1,2,NULL) < ROW(1,3,0)的结果为真，而不是 null，因为第三对元素并不会被考虑。

> **注意**
>
> 在PostgreSQL 8.2 之前，<、<=、>和>=这几种情况并不是按照 SQL 规范处理的。像ROW(a,b) < ROW(c,d)这样的比较会被实现为a < c AND b < d，而正确行为应当等价于a < c OR (a = c AND b < d)。

row_constructor IS DISTINCT FROM *row_constructor*

这个结构与<>行比较相似，但是它对于空值输入不会得到空值。任何空值被认为和任何非空值不相等（有区别），并且任意两个空值被认为相等（无区别）。因此结果将总是为真或为假，永远不会是空值。

row_constructor IS NOT DISTINCT FROM *row_constructor*

这个结构与=行比较相似，但是它对于空值输入不会得到空值。任何空值被认为和任何非空值不相等（有区别），并且任意两个空值被认为相等（无区别）。因此结果将总是为真或为假，永远不会是空值。

> **注意**
>
> SQL 规范要求在结果依赖于比较两个 NULL 值或者一个 NULL 与一个非 NULL 时行比较返回 NULL。PostgreSQL只有在比较两个行构造器的结果或者比较一个行构造器与一

> 个子查询的输出时才这样做（如第 9.22 节中所述）。在其他比较两个复合类型值的环境中，两个 NULL 字段值被认为相等，并且一个 NULL 被认为大于一个非 NULL。为了得到复合类型的一致的排序和索引行为，这样做是必要的。

9.24. 集合返回函数

本节描述那些可能返回多于一行的函数。目前这个类中仅有的函数是序列生成函数，如表 9.49和表 9.50所述。

表 9.49. 序列生成函数

函数	参数类型	返回类型	描述
generate_series(*start*, *stop*)	int或bigint	setof int或setof bigint（与参数类型相同）	以 1 为步长，从 *start* 到 *stop* 生成一系列值
generate_series(*start*, *stop*, *step*)	int或bigint	setof int或setof bigint（与参数类型相同）	以 *step* 为步长，从 *start* 到 *stop* 生成一系列值
generate_series(*start*, *stop*, *step* interval)	timestamp 或 timestamp with time zone	setof timestamp 或 setof timestamp with time zone（与参数类型相同）	以 *step* 为步长，从 *start* 到 *stop* 生成一系列值

当*step*为正数时，如果*start*大于*stop*，则返回零行。反之，当*step*为负数时，如果*start*小于*stop*则返回零行。如果输入为NULL，也返回零行。如果*step*为零，则会报错。下面是一些示例：

```
SELECT * FROM generate_series(2,4);
 generate_series
-----------------
               2
               3
               4
(3 rows)

SELECT * FROM generate_series(5,1,-2);
 generate_series
-----------------
               5
               3
               1
(3 rows)

SELECT * FROM generate_series(4,3);
 generate_series
-----------------
(0 rows)

-- 此示例使用日期加整数的操作符
SELECT current_date + s.a AS dates FROM generate_series(0,14,7) AS
 s(a);
   dates
------------
 2004-02-05
 2004-02-12
 2004-02-19
```

```
(3 rows)

SELECT * FROM generate_series('2008-03-01 00:00'::timestamp,
                              '2008-03-04 12:00', '10 hours');
   generate_series
---------------------
 2008-03-01 00:00:00
 2008-03-01 10:00:00
 2008-03-01 20:00:00
 2008-03-02 06:00:00
 2008-03-02 16:00:00
 2008-03-03 02:00:00
 2008-03-03 12:00:00
 2008-03-03 22:00:00
 2008-03-04 08:00:00
(9 rows)
```

表 9.50. 下标生成函数

函数	返回类型	描述
generate_subscripts(*array anyarray*, *dim int*)	setof int	生成由给定数组下标组成的序列。
generate_subscripts(*array anyarray*, *dim int*, *reverse boolean*)	setof int	生成由给定数组下标组成的序列。当 *reverse* 为真时，按逆序返回该序列。

generate_subscripts 是一个便利函数，它为给定数组的指定维度生成有效下标的集合。对于没有所请求维度的数组或空值数组，返回零行（但对空值数组元素会返回有效下标）。下面是一些示例：

```
-- 基本用法
SELECT generate_subscripts('{NULL,1,NULL,2}'::int[], 1) AS s;
 s
---
 1
 2
 3
 4
(4 rows)

-- 展示一个数组、其下标以及下标对应的值需要用到子查询
SELECT * FROM arrays;
         a
--------------------
 {-1,-2}
 {100,200,300}
(2 rows)

SELECT a AS array, s AS subscript, a[s] AS value
FROM (SELECT generate_subscripts(a, 1) AS s, a FROM arrays) foo;
     array     | subscript | value
---------------+-----------+-------
 {-1,-2}       |         1 |    -1
 {-1,-2}       |         2 |    -2
 {100,200,300} |         1 |   100
 {100,200,300} |         2 |   200
 {100,200,300} |         3 |   300
```

```
(5 rows)

-- 展开一个二维数组
CREATE OR REPLACE FUNCTION unnest2(anyarray)
RETURNS SETOF anyelement AS $$
select $1[i][j]
   from generate_subscripts($1,1) g1(i),
        generate_subscripts($1,2) g2(j);
$$ LANGUAGE sql IMMUTABLE;
CREATE FUNCTION
postgres=# SELECT * FROM unnest2(ARRAY[[1,2],[3,4]]);
 unnest2
---------
       1
       2
       3
       4
(4 rows)
```

9.25. 系统信息函数

表 9.51列出了多个用于提取会话和系统信息的函数。

除了本节列出的函数外，还有许多与统计系统相关的函数，这些函数也提供系统信息。有关更多信息，请参见第 27.2.2 节。

表 9.51. 会话信息函数

名称	返回类型	描述
current_catalog	name	当前数据库名称（在 SQL 标准中称为“目录”）
current_database()	name	当前数据库名称
current_query()	text	客户端提交的当前执行查询文本（可能包含多条语句）
current_role	name	等价于 current_user
current_schema[()]	name	当前模式名称
current_schemas(boolean)	name[]	搜索路径中的模式名称，可选地包含隐式模式
current_user	name	当前执行上下文的用户名
inet_client_addr()	inet	远端连接地址
inet_client_port()	int	远端连接端口
inet_server_addr()	inet	本地连接地址
inet_server_port()	int	本地连接端口
pg_backend_pid()	int	服务当前会话的服务器进程的进程 ID
pg_conf_load_time()	timestamp with time zone	配置加载时间
pg_is_other_temp_schema(oid)	boolean	该模式是否为另一个会话的临时模式？
pg_listening_channels()	setof text	会话当前监听的通道名称
pg_my_temp_schema()	oid	会话临时模式的 OID；不存在时为 0
pg_postmaster_start_time()	timestamp with time zone	服务器启动时间

名称	返回类型	描述
pg_trigger_depth()	int	PostgreSQL 触发器的当前嵌套层级（如果不是从触发器内部直接或间接调用，则为 0）
session_user	name	会话用户名
user	name	等价于 current_user
version()	text	PostgreSQL版本信息

注意

current_catalog、current_role、current_schema、current_user、session_user和user在SQL中具有特殊语法：调用时不得在后面加圆括号。在PostgreSQL 中，current_schema可以选择加圆括号，其他函数则不可以。

session_user 通常是发起当前数据库连接的用户；但超级用户可以用SET SESSION AUTHORIZATION更改这个设置。current_user 是适用于权限检查的用户标识符。通常它等于会话用户，但可以用SET ROLE更改。在执行带有SECURITY DEFINER属性的函数期间，它也会发生改变。用 Unix 术语来说，会话用户是“真实用户”，而当前用户是“有效用户”。current_role和user是current_user的同义词。（SQL 标准区分current_role和current_user，但 PostgreSQL 不作区分，因为它把用户和角色统一为单一类型的实体。）

current_schema 返回搜索路径中第一个模式的名称（如果搜索路径为空，则返回空值）。在创建表或其他命名对象时，如果未指定目标模式，就会使用该模式。current_schemas(boolean) 返回当前搜索路径中所有模式名称的数组。布尔选项决定是否在返回的搜索路径中包含 pg_catalog 等隐式包含的系统模式。

注意

可以在运行时修改搜索路径，命令如下：

```
SET search_path TO schema [, schema, ...]
```

pg_listening_channels返回当前会话正在侦听的信道名称集合。更多信息见LISTEN。

inet_client_addr 返回当前客户端的 IP 地址，inet_client_port 返回端口号。inet_server_addr 返回服务器接受当前连接所用的 IP 地址，inet_server_port 返回端口号。如果当前连接通过 Unix 域套接字建立，这些函数都返回 NULL。

pg_conf_load_time 返回服务器配置文件最近一次加载的时间，类型为 timestamp with time zone。（如果当时当前会话已经存在，则返回该会话自身重新读取配置文件的时间，因此不同会话中的时间会略有不同。否则，返回 postmaster 进程重新读取配置文件的时间。）

pg_my_temp_schema 返回当前会话的临时模式的 OID；如果没有临时模式（因为尚未创建任何临时表），则返回零。如果给定 OID 是另一个会话的临时模式的 OID，pg_is_other_temp_schema 返回真。（这可以用于从系统目录显示结果中排除其他会话的临时表等场景。）

pg_postmaster_start_time 返回服务器启动的时间，类型为 timestamp with time zone。

version返回一个描述PostgreSQL服务器版本的字符串。

has_type_privilege 检查用户是否可以以某种方式访问类型。其参数形式与 has_table_privilege 类似。通过文本字符串而不是 OID 指定类型时，允许的输入与 regtype 数据类型相同（参见第 8.18 节）。所需访问权限类型必须为 USAGE。

表 9.52列出了允许用户以编程方式查询对象访问权限的函数。关于权限的更多信息，请参见第 5.6 节。

表 9.52. 访问权限查询函数

名称	返回类型	描述
has_any_column_privilege(*user*, *table*, *privilege*)	boolean	用户是否对表的至少一列具有权限
has_any_column_privilege(*table*, *privilege*)	boolean	当前用户是否对表的至少一列具有权限
has_column_privilege(*user*, *table*, *column*, *privilege*)	boolean	用户是否具有列权限
has_column_privilege(*table*, *column*, *privilege*)	boolean	当前用户是否具有列权限
has_database_privilege(*user*, *database*, *privilege*)	boolean	用户是否具有数据库权限
has_database_privilege(*database*, *privilege*)	boolean	当前用户是否具有数据库权限
has_foreign_data_wrapper_privilege(*user*, *fdw*, *privilege*)	boolean	用户是否具有外部数据包装器权限
has_foreign_data_wrapper_privilege(*fdw*, *privilege*)	boolean	当前用户是否具有外部数据包装器权限
has_function_privilege(*user*, *function*, *privilege*)	boolean	用户是否具有函数权限
has_function_privilege(*function*, *privilege*)	boolean	当前用户是否具有函数权限
has_language_privilege(*user*, *language*, *privilege*)	boolean	用户是否具有语言权限
has_language_privilege(*language*, *privilege*)	boolean	当前用户是否具有语言权限
has_schema_privilege(*user*, *schema*, *privilege*)	boolean	用户是否具有模式权限
has_schema_privilege(*schema*, *privilege*)	boolean	当前用户是否具有模式权限
has_sequence_privilege(*user*, *sequence*, *privilege*)	boolean	用户是否具有序列权限
has_sequence_privilege(*sequence*, *privilege*)	boolean	当前用户是否具有序列权限
has_server_privilege(*user*, *server*, *privilege*)	boolean	用户是否具有外部服务器权限

名称	返回类型	描述
has_server_privilege(*server*, *privilege*)	boolean	当前用户是否具有外部服务器权限
has_table_privilege(*user*, *table*, *privilege*)	boolean	用户是否具有表权限
has_table_privilege(*table*, *privilege*)	boolean	当前用户是否具有表权限
has_tablespace_privilege(*user*, *tablespace*, *privilege*)	boolean	用户是否具有表空间权限
has_tablespace_privilege(*tablespace*, *privilege*)	boolean	当前用户是否具有表空间权限
has_type_privilege(*user*, *type*, *privilege*)	boolean	用户是否具有类型权限
has_type_privilege(*type*, *privilege*)	boolean	当前用户是否具有类型权限
pg_has_role(*user*, *role*, *privilege*)	boolean	用户是否具有角色权限
pg_has_role(*role*, *privilege*)	boolean	当前用户是否具有角色权限

has_table_privilege检查用户是否可以以某种方式访问表。用户参数可以是名称、OID（pg_authid.oid），public（表示 PUBLIC 伪角色）；如果省略该参数，则使用current_user。表可以通过名称或 OID 指定。（因此，has_table_privilege实际上有六种变体，可以通过参数的数量和类型来区分。）通过名称指定时，如有需要，可以用模式限定名称。所需访问权限类型由文本字符串指定，其值必须为以下值之一：SELECT，INSERT，UPDATE，DELETE，TRUNCATE，REFERENCES，或TRIGGER。还可以添加WITH GRANT OPTION到权限类型后，以检查该权限是否附带授予选项。也可以用逗号分隔列出多个权限类型；只要拥有列出的任一权限，结果就为true。（权限字符串不区分大小写，权限名之间允许有额外的空白，但权限名内部不允许。）下面是一些示例：

```
SELECT has_table_privilege('myschema.mytable', 'select');
SELECT has_table_privilege('joe', 'mytable', 'INSERT, SELECT WITH
 GRANT OPTION');
```

has_sequence_privilege 检查用户是否可以以某种方式访问序列。其参数形式与 has_table_privilege 类似。所需访问权限类型必须为 USAGE、SELECT 或 UPDATE 之一。

has_any_column_privilege 检查用户是否可以以某种方式访问表的至少一列。其参数形式与 has_table_privilege 类似，但所需访问权限类型必须为 SELECT、INSERT、UPDATE 或 REFERENCES 的某种组合。注意，在表级别拥有这些权限中的任意一种，就隐式地对表的每一列拥有该权限，因此对于相同参数，如果 has_table_privilege 返回 true，has_any_column_privilege 也总是返回真。但是，只要至少有一列获得该权限的列级授权，has_any_column_privilege 也会成功。

has_column_privilege 检查用户是否可以以某种方式访问列。其参数形式与 has_table_privilege 类似，但还可以通过列名或属性编号指定列。所需访问权限类型必须为 SELECT、INSERT、UPDATE 或 REFERENCES 的某种组合。注意，在表级别拥有这些权限中的任意一种，就隐式地对表的每一列拥有该权限。

has_database_privilege 检查用户是否可以以某种方式访问数据库。其参数形式与 has_table_privilege 类似。所需访问权限类型必须为 CREATE、CONNECT、TEMPORARY 或 TEMP（等同于 TEMPORARY）的某种组合。

has_function_privilege检查用户是否可以以某种方式访问函数。
其参数形式类似于has_table_privilege。通过文本字符串而不是 OID 指定函数时，允许的输入与regprocedure数据类型相同（参见第 8.18 节）。所需访问权限类型必须为EXECUTE。例如：

```
SELECT has_function_privilege('joeuser', 'myfunc(int, text)',
 'execute');
```

has_foreign_data_wrapper_privilege 检查用户是否可以以某种方式访问外部数据包装器。其参数形式与 has_table_privilege 类似。所需访问权限类型必须为 USAGE。

has_language_privilege 检查用户是否可以以某种方式访问过程语言。其参数形式与 has_table_privilege 类似。所需访问权限类型必须为 USAGE。

has_schema_privilege 检查用户是否可以以某种方式访问模式。其参数形式与 has_table_privilege 类似。所需访问权限类型必须为 CREATE 或 USAGE 的某种组合。

has_server_privilege 检查用户是否可以以某种方式访问外部服务器。其参数形式与 has_table_privilege 类似。所需访问权限类型必须为 USAGE。

has_tablespace_privilege 检查用户是否可以以某种方式访问表空间。其参数形式与 has_table_privilege 类似。所需访问权限类型必须为 CREATE。

pg_has_role检查用户是否可以以某种方式访问角色。它的参数可能性与has_table_privilege类似，但不允许使用public作为用户名。所需访问权限类型必须为MEMBER或USAGE的某种组合。MEMBER表示直接或间接地属于该角色（即有权执行SET ROLE），而USAGE表示无需执行SET ROLE就能立即使用该角色的权限。

表 9.53列出了判断某个特定对象是否可见的函数，其判断依据是当前模式搜索路径。例如，如果表所在的模式位于搜索路径中，并且在搜索路径的更前面没有同名表，就称该表可见。这等价于说，可以只通过表名引用该表，而不必显式地用模式限定。
要列出所有可见表的名称：

```
SELECT relname FROM pg_class WHERE pg_table_is_visible(oid);
```

表 9.53. 模式可见性查询函数

名称	返回类型	描述
pg_collation_is_visible(*collation_oid*)	boolean	排序规则是否在搜索路径中可见
pg_conversion_is_visible(*conversion_oid*)	boolean	转换是否在搜索路径中可见
pg_function_is_visible(*function_oid*)	boolean	函数是否在搜索路径中可见
pg_opclass_is_visible(*opclass_oid*)	boolean	操作符类是否在搜索路径中可见
pg_operator_is_visible(*operator_oid*)	boolean	操作符是否在搜索路径中可见
pg_opfamily_is_visible(*opclass_oid*)	boolean	操作符族是否在搜索路径中可见
pg_table_is_visible(*table_oid*)	boolean	表是否在搜索路径中可见
pg_ts_config_is_visible(*config_oid*)	boolean	全文检索配置是否在搜索路径中可见
pg_ts_dict_is_visible(*dict_oid*)	boolean	全文检索词典是否在搜索路径中可见

名称	返回类型	描述
pg_ts_parser_is_visible(*parser_oid*)	boolean	全文检索解析器是否在搜索路径中可见
pg_ts_template_is_visible(*template_oid*)	boolean	全文检索模板是否在搜索路径中可见
pg_type_is_visible(*type_oid*)	boolean	类型（或域）是否在搜索路径中可见

每个函数检查一种数据库对象的可见性。注意，pg_table_is_visible 也可用于视图、物化视图、索引、序列和外部表；pg_type_is_visible 也可用于域。对于函数和操作符，如果搜索路径更前面没有名称和参数数据类型都相同的对象，
那么路径中的该对象就是可见的。对于操作符类，会同时考虑名称和关联的索引访问方法。

所有这些函数都需要用对象 OID 标识要检查的对象。如果想按名称测试对象，使用 OID 别名类型会很方便（regclass，regtype，regprocedure，regoperator，regconfig，或regdictionary），例如：

```
SELECT pg_type_is_visible('myschema.widget'::regtype);
```

注意，用这种方式测试不带模式限定的类型名并没有太大意义：只要该名称能够被识别，
它就必然可见。

表 9.54列出从系统目录中提取信息的函数。

pg_describe_object返回由目录 OID、对象 OID 和子对象 ID（可为零）指定的数据库对象的文本描述。该描述供人阅读，并且可能会根据服务器配置被翻译。这有助于确定存储在pg_depend目录中的对象标识。

表 9.54. 系统目录信息函数

名称	返回类型	描述
format_type(*type_oid*, *typemod*)	text	获取数据类型的 SQL 名称
pg_describe_object(*catalog_id*, *object_id*, *object_sub_id*)	text	获取数据库对象的描述
pg_get_constraintdef(*constraint_oid*)	text	获取约束定义
pg_get_constraintdef(*constraint_oid*, *pretty_bool*)	text	获取约束定义
pg_get_expr(*pg_node_tree*, *relation_oid*)	text	反编译表达式的内部形式，假定其中所有 Var 节点都引用第二个参数指定的关系
pg_get_expr(*pg_node_tree*, *relation_oid*, *pretty_bool*)	text	反编译表达式的内部形式，假定其中所有 Var 节点都引用第二个参数指定的关系
pg_get_functiondef(*func_oid*)	text	获取函数定义
pg_get_function_arguments(*func_oid*)	text	获取函数定义的参数列表（包含默认值）
pg_get_function_identity_arguments(*func_oid*)	text	获取用于标识函数的参数列表（不含默认值）
pg_get_function_result(*func_oid*)	text	获取函数的 RETURNS 子句

名称	返回类型	描述
pg_get_indexdef(*index_oid*)	text	获取索引的 CREATE INDEX 命令
pg_get_indexdef(*index_oid*, *column_no*, *pretty_bool*)	text	获取索引的 CREATE INDEX 命令；当 *column_no* 非零时，仅获取一个索引列的定义
pg_get_keywords()	setof record	获取 SQL 关键字及其类别的列表
pg_get_ruledef(*rule_oid*)	text	获取规则的 CREATE RULE 命令
pg_get_ruledef(*rule_oid*, *pretty_bool*)	text	获取规则的 CREATE RULE 命令
pg_get_serial_sequence(*table_name*, *column_name*)	text	获取 serial、smallserial 或 bigserial 列使用的序列名称
pg_get_triggerdef(*trigger_oid*)	text	获取触发器的 CREATE [CONSTRAINT] TRIGGER 命令
pg_get_triggerdef(*trigger_oid*,*pretty_bool*)	text	获取触发器的 CREATE [CONSTRAINT] TRIGGER 命令
pg_get_userbyid(*role_oid*)	name	获取具有给定 OID 的角色名称
pg_get_viewdef(*view_name*)	text	获取视图或物化视图的底层 SELECT 命令（已弃用）
pg_get_viewdef(*view_name*, *pretty_bool*)	text	获取视图或物化视图的底层 SELECT 命令；如果 *pretty_bool* 为真，带字段的行会换行到 80 列（已弃用）
pg_get_viewdef(*view_oid*)	text	获取视图或物化视图的底层 SELECT 命令
pg_get_viewdef(*view_oid*, *pretty_bool*)	text	获取视图或物化视图的底层 SELECT 命令；如果 *pretty_bool* 为真，带字段的行会换行到 80 列
pg_get_viewdef(*view_oid*, *wrap_column_int*)	text	获取视图或物化视图的底层 SELECT 命令；包含字段的行按指定列数折行，并隐含启用美化输出
pg_options_to_table(*reloptions*)	setof record	获取存储选项的名称/值对集合
pg_tablespace_databases(*tablespace_oid*)	setof oid	获取在该表空间中具有对象的数据库 OID 集合
pg_tablespace_location(*tablespace_oid*)	text	获取该表空间在文件系统中的路径
pg_typeof(*any*)	regtype	获取任意值的数据类型
collation for (*any*)	text	获取参数的排序规则

format_type 返回数据类型的 SQL 名称，该类型由其类型 OID 和可能存在的类型修饰符标识。如果不知道具体的类型修饰符，请为类型修饰符参数传入 NULL。

pg_get_keywords 返回一组记录，描述服务器识别的 SQL 关键字。word 列包含关键字。catcode 列包含类别代码：U 表示非保留关键字，C 表示列名，T 表示类型名或函数名，R 表示保留关键字。catdesc 列包含描述该类别的字符串，该字符串可能已经过本地化。

pg_get_constraintdef、pg_get_indexdef、pg_get_ruledef和pg_get_triggerdef分别重建约束、索引、规则或触发器的创建命令。（注意，这是通过反编译重建的，并非命令的原始文本。）pg_get_expr 反编译单个表达式的内部形

式，例如列的默认值。这在检查系统目录内容时可能很有用。如果表达式可能包含 Var 节点，请将它们所引用的关系的 OID 指定为第二个参数；如果预计不含 Var 节点，传入零即可。pg_get_viewdef 重建定义视图的 SELECT 查询。这些函数中的大多数有两种变体，其中一种可以选择“美化输出”结果。美化格式更易读，但默认格式更可能被未来版本的 PostgreSQL 以相同方式解释；转储时应避免使用美化输出。向美化输出参数传入 false，所得结果与完全没有该参数的变体相同。

pg_get_functiondef 返回一个函数的完整 CREATE OR REPLACE FUNCTION 语句。pg_get_function_arguments 以 CREATE FUNCTION 中所需的形式返回函数参数列表。pg_get_function_result 类似地返回该函数相应的 RETURNS 子句。pg_get_function_identity_arguments 返回标识函数所需的参数列表，例如以 ALTER FUNCTION 中所需的形式返回。此形式省略默认值。

pg_get_serial_sequence返回与某一列关联的序列名称；如果该列没有关联的序列，则返回 NULL。第一个输入参数是可以带有模式名的表名，第二个参数是列名。由于第一个参数可能包含模式和表，它不会被当作双引号括起的标识符处理，因此默认转换为小写；第二个参数只包含列名，会被当作带双引号的标识符处理并保留大小写。
函数返回的值采用适于传递给序列函数的格式（参见第 9.16 节）。这一关联可以使用ALTER SEQUENCE OWNED BY修改或移除。（该函数也许应该叫作pg_get_owned_sequence；它当前的名称反映了它通常用于serial或bigserial列这一事实。）

pg_get_userbyid 根据给定 OID 获取角色名称。

向 pg_options_to_table 传入 pg_class.reloptions 或 pg_attribute.attoptions 时，它返回存储选项名称/值对（option_name/option_value）的集合。

pg_tablespace_databases 用于检查表空间。它返回在该表空间中存储了对象的数据库 OID 集合。如果此函数返回任何行，则说明该表空间不为空，不能删除。要显示存放在该表空间中的具体对象，需要连接到 pg_tablespace_databases 标识的数据库，并查询它们的 pg_class 系统目录。

表达式collation for返回所传入值的排序规则。例如：

```
SELECT collation for (description) FROM pg_description LIMIT 1;
 pg_collation_for
------------------
 "default"
(1 row)

SELECT collation for ('foo' COLLATE "de_DE");
 pg_collation_for
------------------
 "de_DE"
(1 row)
```

返回值可能带有引号和模式限定。如果不能为参数表达式推导出排序规则，则返回空值。
如果参数的数据类型不支持排序规则，则会报错。

col_description 返回表列的注释，列由其所属表的 OID 和列号指定。（obj_description 不能用于表列，因为列没有自身的 OID。）

pg_typeof返回所传入值的数据类型的 OID。这有助于排查问题或动态构造 SQL 查询。函数声明的返回类型是regtype，它是一种 OID 别名类型（参见第 8.18 节）；这意味着它在比较时与 OID 相同，但显示为类型名。例如：

```
SELECT pg_typeof(33);

 pg_typeof
-----------
```

```
 integer
(1 row)

SELECT typlen FROM pg_type WHERE oid = pg_typeof(33);
 typlen
--------
      4
(1 row)
```

表 9.55中的函数用于提取此前通过COMMENT命令存储的注释。如果找不到与指定参数对应的注释，则返回空值。

表 9.55. 注释信息函数

名称	返回类型	描述
col_description(*table_oid*, *column_number*)	text	获取表列的注释
obj_description(*object_oid*, *catalog_name*)	text	获取数据库对象的注释
obj_description(*object_oid*)	text	获取数据库对象的注释（已弃用）
shobj_description(*object_oid*, *catalog_name*)	text	获取共享数据库对象的注释

obj_description 的双参数形式返回数据库对象的注释，对象由其 OID 和所在系统目录的名称指定。例如，obj_description(123456,'pg_class') 会获取 OID 为 123456 的表的注释。obj_description 的单参数形式只需要对象 OID。该形式已被弃用，因为无法保证 OID 在不同系统目录之间唯一，因而可能返回错误的注释。

shobj_description 的用法与 obj_description 相同，但用于获取共享对象的注释。有些系统目录由数据库集簇中的所有数据库全局共享，其中对象的描述也全局存储。

表 9.56中的函数以可导出的形式提供服务器事务信息。这些函数主要用于确定两个快照之间有哪些事务提交。

表 9.56. 事务 ID 和快照

名称	返回类型	描述
txid_current()	bigint	获取当前事务 ID；如果当前事务尚无 ID，则分配一个新的 ID
txid_current_snapshot()	txid_snapshot	获取当前快照
txid_snapshot_xip(*txid_snapshot*)	setof bigint	获取快照中正在进行的事务 ID
txid_snapshot_xmax(*txid_snapshot*)	bigint	获取快照的 xmax
txid_snapshot_xmin(*txid_snapshot*)	bigint	获取快照的 xmin
txid_visible_in_snapshot(*bigint*, *txid_snapshot*)	boolean	该事务 ID 是否在快照中可见？（不要用于子事务 ID）

内部事务 ID 类型（xid）为 32 位，每经过约 40 亿个事务就会回卷一次。不过，这些函数导出的是用“纪元”计数器扩展的 64 位格式，在一次安装的整个使用期间都不会回卷。这些函

数使用的 `txid_snapshot` 数据类型存储某一时刻的事务 ID 可见性信息。其组成部分见表 9.57。

表 9.57. 快照组件

名称	描述
`xmin`	仍然活动的最早事务 ID（txid）。所有更早的事务要么已经提交且可见，要么已经回滚而失效。
`xmax`	第一个尚未分配的 txid。所有大于或等于此值的 txid 在快照时尚未开始，因此不可见。
`xip_list`	快照时活动的 txid。该列表仅包含 `xmin` 与 `xmax` 之间的活动 txid；可能存在大于 `xmax` 的活动 txid。满足 `xmin <= txid < xmax` 且不在此列表中的 txid 在快照时已经完成，因此根据其提交状态，要么可见，要么失效。该列表不包含子事务的 txid。

`txid_snapshot` 的文本表示为 *`xmin`*`:`*`xmax`*`:`*`xip_list`*。例如，`10:20:10,14,15` 表示 `xmin=10, xmax=20, xip_list=10, 14, 15`。

9.26. 系统管理函数

这一节描述的函数被用来控制和监视一个PostgreSQL安装。

9.26.1. 配置设定函数

表 9.58展示了那些可以用于查询以及修改运行时配置参数的函数。

表 9.58. 配置设定函数

名称	返回类型	描述
`current_setting(`*`setting_name`*`)`	`text`	获取配置项的当前值
`set_config(`*`setting_name`*`, `*`new_value`*`, `*`is_local`*`)`	`text`	设置参数并返回新值

函数`current_setting`返回设置*`setting_name`*的当前值。它对应于SQL命令`SHOW`。例如：

```
SELECT current_setting('datestyle');

 current_setting
-----------------
 ISO, MDY
(1 row)
```

`set_config`将参数*`setting_name`*设置为*`new_value`*。如果*`is_local`*为`true`，新值仅适用于当前事务。如果要让新值适用于当前会话，请改用`false`。该函数对应于 SQL 命令`SET`。例如：

```
SELECT set_config('log_statement_stats', 'off', false);

 set_config
------------
 off
(1 row)
```

9.26.2. 服务器信号函数

在表 9.59中展示的函数向其它服务器进程发送控制信号。这些函数的使用通常仅限超级用户，例外情况已单独注明。

表 9.59. 服务器信号函数

名称	返回类型	描述
pg_cancel_backend(*pid* int)	boolean	取消后端的当前查询。如果调用角色是被取消后端所属角色的成员，也允许执行此操作；但只有超级用户才能取消超级用户的后端。
pg_reload_conf()	boolean	使服务器进程重新加载配置文件
pg_rotate_logfile()	boolean	轮转服务器日志文件
pg_terminate_backend(*pid* int)	boolean	终止后端。如果调用角色是被终止后端所属角色的成员，也允许执行此操作；但只有超级用户才能终止超级用户的后端。

这些函数在成功时都返回 true，否则返回 false。

pg_cancel_backend和pg_terminate_backend向由进程 ID 标识的后端进程发送信号（分别是SIGINT或SIGTERM）。一个活动后端的进程 ID可以从pg_stat_activity视图的pid列中找到，或者通过在服务器上列出postgres进程（在 Unix 上使用ps或者在Windows上使用任务管理器）得到。一个活动后端的角色可以在pg_stat_activity视图的usename列中找到。

pg_reload_conf 向服务器发送 SIGHUP 信号，使所有服务器进程重新加载配置文件。

pg_rotate_logfile 向日志文件管理器发送信号，使其立即切换到新的输出文件。只有内置日志收集器正在运行时才有效，否则不存在日志文件管理器子进程。

9.26.3. 备份控制函数

表 9.60中列出的函数有助于进行在线备份。这些函数不能在恢复期间执行（pg_xlog_location_diff除外）。

表 9.60. 备份控制函数

名称	返回类型	描述
pg_create_restore_point(*name* text)	text	创建用于恢复的命名点（仅限超级用户）
pg_current_xlog_insert_location()	text	获取当前预写式日志插入位置
pg_current_xlog_location()	text	获取当前预写式日志写入位置
pg_start_backup(*label* text [, *fast* boolean])	text	准备执行在线备份（仅限超级用户或复制角色）
pg_stop_backup()	text	结束在线备份（仅限超级用户或复制角色）
pg_switch_xlog()	text	强制切换到新的预写式日志文件（仅限超级用户）
pg_xlogfile_name(*location* text)	text	将事务日志位置字符串转换为文件名

名称	返回类型	描述
pg_xlogfile_name_offset(*location* text)	text,integer	将事务日志位置字符串转换为文件名和文件内的十进制字节偏移量
pg_xlog_location_diff(*location* text, *location* text)	numeric	计算两个事务日志位置之差

pg_start_backup接受任意用户定义的备份标签。
（通常是备份转储文件将要保存的名称。）该函数将备份标签文件（backup_label）
写入数据库集簇的数据目录，执行检查点，然后以文本形式返回备份的起始事务日志位置。
用户可以忽略该返回值，提供它只是因为它可能有用。

```
postgres=# select pg_start_backup('label_goes_here');
 pg_start_backup
-----------------
 0/D4445B8
(1 row)
```

还可以提供第二个可选参数，其类型为boolean。如果该参数为true，则尽快执行pg_start_backup。这会强制立即执行检查点，使 I/O 操作量陡增，并降低并发执行的查询的速度。

pg_stop_backup 删除 pg_start_backup 创建的标签文件，并在事务日志归档区域创建一个备份历史文件。历史文件包括传给 pg_start_backup 的标签、备份的起止事务日志位置以及备份的起止时间。返回值是备份的结束事务日志位置（同样可以忽略）。
记录结束位置后，当前事务日志插入点会自动推进到下一个事务日志文件的起始处，
使包含结束位置的事务日志文件可以立即归档，从而完成备份。

pg_switch_xlog 切换到下一个预写式日志文件，使当前文件可以归档（假设正在使用连续归档）。返回值是刚完成的预写式日志文件中的结束预写式日志位置加 1。如果自上次切换预写式日志以来没有发生任何预写式日志活动，pg_switch_xlog 不执行任何操作，并返回当前使用的预写式日志文件的起始位置。

pg_create_restore_point 创建一条可用作恢复目标的命名预写式日志记录，并返回相应的预写式日志位置。随后可以在recovery_target_name中使用该名称，
指定恢复到哪一点。应避免创建多个同名恢复点，
因为恢复会在第一个名称匹配恢复目标的恢复点停止。

pg_current_xlog_location 显示当前预写式日志写入位置，格式与上述函数相同。类似地，pg_current_xlog_insert_location 显示当前预写式日志插入点。插入位置是预写式日志在任意时刻的“逻辑”末尾，
而写入位置是实际从服务器内部缓冲区写出的内容的末尾。
写入位置是能从服务器外部检查到的内容的末尾，如果要归档尚未写满的预写式日志文件，
通常需要的是这个位置。提供插入位置主要是为了服务器调试。这些都是只读操作，
不需要超级用户权限。

可以使用pg_xlogfile_name_offset
从上述任一函数的结果中提取相应的预写式日志文件名和字节偏移。例如：

```
postgres=# SELECT * FROM pg_xlogfile_name_offset(pg_stop_backup());
        file_name         | file_offset
--------------------------+-------------
 00000001000000000000000D |     4039624
(1 row)
```

类似地，pg_xlogfile_name仅提取预写式日志文件名。
当指定的预写式日志位置恰好位于预写式日志文件边界时，

这两个函数都返回前一个预写式日志文件的名称。
这通常正是管理预写式日志归档时所需的行为，
因为前一个文件是当前需要归档的最后一个文件。

pg_xlog_location_diff 计算两个预写式日志位置之间的字节差。可以将它与 pg_stat_replication 或表 9.60中的某些函数配合使用，以获取复制延迟。

有关正确使用这些函数的详细信息，参见第 24.3 节。

9.26.4. 恢复控制函数

表 9.61中展示的函数提供有关备库当前状态的信息。这些函数可以在恢复或普通运行过程中被执行。

表 9.61. 恢复信息函数

名称	返回类型	描述
pg_is_in_recovery()	bool	如果恢复仍在进行，则为真。
pg_last_xlog_receive_location()	text	获取流复制最近接收并同步到磁盘的预写式日志位置。在流复制进行期间，该值单调增加。如果恢复已完成，该值将保持为恢复期间最后接收并同步到磁盘的 WAL 记录的位置。如果禁用了流复制，或者流复制尚未开始，此函数返回 NULL。
pg_last_xlog_replay_location()	text	获取恢复期间最近重放的预写式日志位置。如果恢复仍在进行，该值单调增加。如果恢复已完成，该值将保持为该次恢复期间最后应用的 WAL 记录的位置。如果服务器未经恢复而正常启动，此函数返回 NULL。
pg_last_xact_replay_timestamp()	timestamp with time zone	获取恢复期间最近重放事务的时间戳，即该事务的提交或中止 WAL 记录在主库上生成的时间。如果恢复期间尚未重放任何事务，此函数返回 NULL。否则，如果恢复仍在进行，该值单调增加。如果恢复已完成，该值将保持为该次恢复期间最后应用的事务的时间戳。如果服务器未经恢复而正常启动，此函数返回 NULL。

表 9.62列出的函数用于控制恢复进度。这些函数只能在恢复期间执行。

表 9.62. 恢复控制函数

名称	返回类型	描述
pg_is_xlog_replay_paused()	bool	如果恢复已暂停，则为真。
pg_xlog_replay_pause()	void	立即暂停恢复（默认仅限超级用户，但可以向其他用户授予EXECUTE权限以运行此函数）。
pg_xlog_replay_resume()	void	如果恢复已暂停，则重新开始恢复（默认仅限超级用户，但可以

名称	返回类型	描述
		向其他用户授予EXECUTE权限以运行此函数）。

恢复暂停时，不再应用数据库更改。如果处于热备状态，
所有新查询都会看到同一个一致的数据库快照，并且在恢复继续之前不会再产生查询冲突。

如果禁用了流复制，则暂停状态可能会无限期地持续下去，不会出现问题。
如果正在进行流复制，那么将继续接收WAL记录，这将最终填满可用磁盘空间，
这取决于暂停持续时间、WAL生成速度和可用磁盘空间。

9.26.5. 快照同步函数

PostgreSQL允许数据库会话同步它们的快照。
一个快照决定对于正在使用该快照的事务哪些数据是可见的。
当两个或者更多个会话需要看到数据库中的相同内容时，就需要同步快照。
如果两个会话独立开始其事务，就总是有可能有某个第三事务在两个`START TRANSACTION`命令的执行之间提交，这样其中一个会话就可以看到该事务的效果而另一个则看不到。

为了解决这个问题，PostgreSQL允许一个事务导出它正在使用的快照。
只要导出快照的事务仍然保持打开，其他事务可以导入它的快照，
并且因此可以保证它们可以看到和第一个事务看到的完全一样的数据库视图。
但是注意这些事务中的任何一个对数据库所作的更改对其他事务仍然保持不可见，
和未提交事务所作的修改一样。因此这些事务是针对以前存在的数据同步，
而对由它们自己所作的更改则采取正常的动作。

如表 9.63中所示，快照通过`pg_export_snapshot`函数导出，并且通过SET TRANSACTION命令导入。

表 9.63. 快照同步函数

名称	返回类型	描述
`pg_export_snapshot()`	`text`	保存当前快照并返回其标识符

函数 `pg_export_snapshot` 保存当前快照，并返回标识该快照的 `text` 字符串。必须在数据库之外将此字符串传递给要导入该快照的客户端。
快照只能在导出它的事务结束之前导入。如有需要，一个事务可以导出多个快照。注意，这只对 `READ COMMITTED` 事务有用，因为在 `REPEATABLE READ` 及更高隔离级别中，事务在整个存续期间都使用同一个快照。事务一旦导出过快照，就不能再通过PREPARE TRANSACTION进行预备。

关于如何使用导出快照的详细说明，请参见SET TRANSACTION。

9.26.6. 数据库对象管理函数

表 9.64中的函数计算数据库对象的磁盘空间用量。

表 9.64. 数据库对象尺寸函数

名称	返回类型	描述
`pg_column_size(any)`	`int`	存储某个值所使用的字节数（可能已压缩）
`pg_database_size(oid)`	`bigint`	指定 OID 的数据库占用的磁盘空间
`pg_database_size(name)`	`bigint`	指定名称的数据库占用的磁盘空间

名称	返回类型	描述
pg_indexes_size(regclass)	bigint	指定表的索引占用的磁盘空间总量
pg_relation_size(*relation* regclass, *fork* text)	bigint	指定表或索引的指定分支（'main'、'fsm'、'vm' 或 'init'）占用的磁盘空间
pg_relation_size(*relation* regclass)	bigint	pg_relation_size(..., 'main') 的简写
pg_size_pretty(bigint)	text	将以 64 位整数表示的字节大小转换为带有大小单位的人类可读格式
pg_size_pretty(numeric)	text	将以 numeric 值表示的字节大小转换为带有大小单位的人类可读格式
pg_table_size(regclass)	bigint	指定表占用的磁盘空间，不包括索引（但包括 TOAST、空闲空间映射和可见性映射）
pg_tablespace_size(oid)	bigint	指定 OID 的表空间占用的磁盘空间
pg_tablespace_size(name)	bigint	指定名称的表空间占用的磁盘空间
pg_total_relation_size(regclass)	bigint	指定表占用的磁盘空间总量，包括所有索引和 TOAST 数据

pg_column_size 显示存储任意单个数据值所用的空间。

pg_total_relation_size 接受表或 TOAST 表的 OID 或名称，返回该表占用的磁盘总空间，包括所有关联索引。此函数等同于 pg_table_size + pg_indexes_size。

pg_table_size 接受表的 OID 或名称，返回该表所需的磁盘空间，不包括索引。（包括 TOAST 空间、空闲空间映射和可见性映射。）

pg_indexes_size 接受表的 OID 或名称，返回该表所有附属索引占用的磁盘总空间。

pg_database_size 和 pg_tablespace_size 接受数据库或表空间的 OID 或名称，返回其中占用的磁盘总空间。使用 pg_database_size 必须对指定数据库拥有 CONNECT 权限（默认授予）。使用 pg_tablespace_size 必须对指定表空间拥有 CREATE 权限；但如果该表空间是当前数据库的默认表空间，则不受此限制。

pg_relation_size接受表、索引或 TOAST 表的 OID 或名称，并以字节为单位返回该关系某个分支的磁盘大小。（注意，大多数情况下使用更高层的函数会更方便，例如pg_total_relation_size或pg_table_size，它们会将所有分支的大小相加。）只带一个参数时，返回关系的主数据分支大小。可以提供第二个参数来指定要检查的分支：

- 'main' 返回关系的主数据分支大小。
- 'fsm' 返回关系所关联的空闲空间映射的大小（参见第 56.3 节）。
- 'vm' 返回关系所关联的可见性映射的大小（参见第 56.4 节）。
- 'init' 返回关系所关联的初始化分支（如有，见第 56.5 节）的大小。

pg_size_pretty 可以将其他函数的结果格式化为便于阅读的形式，适当地使用 kB、MB、GB 或 TB 单位。

上述操作表和索引的函数接受一个regclass参数，它是该表或索引在pg_class系统目录中的 OID。你不必手工去查找该 OID，因为regclass数据类型的输入转换器会为你代劳。只写包围在单引号内的表名，这样它看起来像一个字面量。为了与普通SQL名称的处理相兼容，该字符串将被转换为小写形式，除非其中在表名周围包含双引号。

如果向上述某个函数传入的 OID 不代表现有对象，则返回 NULL。

表 9.65中展示的函数帮助标识数据库对象相关的磁盘文件。

表 9.65. 数据库对象位置函数

名称	返回类型	描述
pg_relation_filenode(*relation* regclass)	oid	指定关系的文件结点编号
pg_relation_filepath(*relation* regclass)	text	指定关系的文件路径名

pg_relation_filenode 接受表、索引、序列或 TOAST 表的 OID 或名称，返回当前分配给它的“文件结点”编号。文件结点是关系文件名的基本组成部分（更多信息见第 56.1 节）。对于大多数表，结果与 pg_class.relfilenode 相同；但某些系统目录的 relfilenode 为零，必须使用此函数获取正确值。如果传入的关系没有存储空间（例如视图），函数返回 NULL。

pg_relation_filepath 与 pg_relation_filenode 类似，但返回关系的完整文件路径名（相对于数据库集簇的数据目录 PGDATA）。

9.26.7. 通用文件访问函数

表 9.66中的函数提供对数据库服务器所在机器上的文件的本地访问。只能访问数据库集簇目录和 log_directory 中的文件。集簇目录中的文件使用相对路径，日志文件使用与 log_directory 配置设置相匹配的路径。除非另有说明，这些函数仅限超级用户使用。

表 9.66. 通用文件访问函数

名称	返回类型	描述
pg_ls_dir(*dirname* text)	setof text	列出目录内容。
pg_read_file(*filename* text [, *offset* bigint, *length* bigint])	text	返回文本文件的内容。
pg_read_binary_file(*filename* text [, *offset* bigint, *length* bigint])	bytea	返回文件内容。
pg_stat_file(*filename* text)	record	返回文件信息。

pg_ls_dir 返回指定目录中的所有名称，但不包括特殊项“.”和“..”。

pg_read_file 从给定的 *offset* 开始返回文本文件的一部分，最多返回 *length* 字节（如果先到达文件末尾，则返回更少的字节）。如果 *offset* 为负，则它相对于文件末尾。如果省略 *offset* 和 *length*，则返回整个文件。从文件读取的字节按服务器编码解释为字符串；如果它们在该编码下无效，则会报错。

pg_read_binary_file类似于pg_read_file，但结果是bytea值，因此不执行编码检查。与convert_from函数配合使用，可以按指定编码读取文件：

```
SELECT convert_from(pg_read_binary_file('file_in_utf8.txt'),
 'UTF8');
```

pg_stat_file返回一条记录，包含文件大小、最后访问时间戳、最后修改时间戳、最后文件状态更改时间戳（仅限 Unix 平台）、文件创建时间戳（仅限 Windows），以及一个boolean值，表示它是否是目录。典型用法包括：

```
SELECT * FROM pg_stat_file('filename');
SELECT (pg_stat_file('filename')).modification;
```

9.26.8. 咨询锁函数

表 9.67中展示的函数管理咨询锁。有关正确使用这些函数的细节请参考第 13.3.4 节。

表 9.67. 咨询锁函数

名称	返回类型	描述
pg_advisory_lock(*key* bigint)	void	获取排他会话级咨询锁
pg_advisory_lock(*key1* int, *key2* int)	void	获取排他会话级咨询锁
pg_advisory_lock_shared(*key* bigint)	void	获取共享会话级咨询锁
pg_advisory_lock_shared(*key1* int, *key2* int)	void	获取共享会话级咨询锁
pg_advisory_unlock(*key* bigint)	boolean	释放排他会话级咨询锁
pg_advisory_unlock(*key1* int, *key2* int)	boolean	释放排他会话级咨询锁
pg_advisory_unlock_all()	void	释放当前会话持有的所有会话级咨询锁
pg_advisory_unlock_shared(*key* bigint)	boolean	释放共享会话级咨询锁
pg_advisory_unlock_shared(*key1* int, *key2* int)	boolean	释放共享会话级咨询锁
pg_advisory_xact_lock(*key* bigint)	void	获取排他事务级咨询锁
pg_advisory_xact_lock(*key1* int, *key2* int)	void	获取排他事务级咨询锁
pg_advisory_xact_lock_shared(*key* bigint)	void	获取共享事务级咨询锁
pg_advisory_xact_lock_shared(*key1* int, *key2* int)	void	获取共享事务级咨询锁
pg_try_advisory_lock(*key* bigint)	boolean	如果可用，则获取排他会话级咨询锁
pg_try_advisory_lock(*key1* int, *key2* int)	boolean	如果可用，则获取排他会话级咨询锁
pg_try_advisory_lock_shared(*key* bigint)	boolean	如果可用，则获取共享会话级咨询锁
pg_try_advisory_lock_shared(*key1* int, *key2* int)	boolean	如果可用，则获取共享会话级咨询锁
pg_try_advisory_xact_lock(*key* bigint)	boolean	如果可用，则获取排他事务级咨询锁
pg_try_advisory_xact_lock(*key1* int, *key2* int)	boolean	如果可用，则获取排他事务级咨询锁
pg_try_advisory_xact_lock_shared(*key* bigint)	boolean	如果可用，则获取共享事务级咨询锁

名称	返回类型	描述
pg_try_advisory_xact_lock_shared(*key1* int, *key2* int)	boolean	如果可用，则获取共享事务级咨询锁

pg_advisory_lock 锁定应用定义的资源，可以用一个 64 位键值或两个 32 位键值标识资源（注意，这两个键空间不重叠）。如果另一个会话已持有相同资源标识符上的锁，此函数会等待，直到该资源可用。该锁是排他的。多个锁请求会叠加，因此如果同一资源被锁定了三次，就必须解锁三次，才能释放给其他会话使用。

pg_advisory_lock_shared 的行为与 pg_advisory_lock 相同，但该锁可以与其他请求共享锁的会话共享。只有试图获取排他锁的会话会被阻挡。

pg_try_advisory_lock 与 pg_advisory_lock 类似，但不会等待锁变得可用。它要么立即获得锁并返回 true，要么在无法立即获得锁时返回 false。

pg_try_advisory_lock_shared 的行为与 pg_try_advisory_lock 相同，但尝试获取的是共享锁，而不是排他锁。

pg_advisory_unlock 释放先前获得的会话级排他咨询锁。如果成功释放锁，则返回 true。如果未持有该锁，则返回 false，服务器还会报告一条 SQL 警告。

pg_advisory_unlock_shared 的行为与 pg_advisory_unlock 相同，但释放的是会话级共享咨询锁。

pg_advisory_unlock_all 释放当前会话持有的所有会话级咨询锁。（会话结束时会隐式调用此函数，即使客户端非正常断开连接也是如此。）

pg_advisory_xact_lock 的行为与 pg_advisory_lock 相同，但锁会在当前事务结束时自动释放，不能显式释放。

pg_advisory_xact_lock_shared 的行为与 pg_advisory_lock_shared 相同，但锁会在当前事务结束时自动释放，不能显式释放。

pg_try_advisory_xact_lock 的行为与 pg_try_advisory_lock 相同，但如果获得了锁，该锁会在当前事务结束时自动释放，不能显式释放。

pg_try_advisory_xact_lock_shared 的行为与 pg_try_advisory_lock_shared 相同，但如果获得了锁，该锁会在当前事务结束时自动释放，不能显式释放。

9.27. 触发器函数

目前 PostgreSQL 提供一个内置触发器函数 suppress_redundant_updates_trigger，它会阻止那些实际上不会改变行中数据的更新。通常的行为则是无论数据是否改变都执行更新。（这种通常行为不需要检查，因此更新运行得更快，并且在某些情况下也有用。）

理想情况下，应尽量避免执行实际上不会改变记录中数据的更新。冗余更新会耗费大量不必要的时间，尤其是在需要改动许多索引时；

它们还会使死行占用空间，最终需要通过清理回收。但是，
在客户端代码中检测这种情况并不总是容易，甚至可能无法做到，
而编写用于检测的表达式也容易出错。另一种办法是使用 `suppress_redundant_updates_trigger`，跳过不改变数据的更新。但使用时应当谨慎。
此触发器处理每条记录的耗时虽小，却不可忽略，
因此如果一次更新涉及的大多数记录确实会改变，使用此触发器反而会使更新更慢。

该`suppress_redundant_updates_trigger`函数可以按如下方式添加到表中：

```
CREATE TRIGGER z_min_update
BEFORE UPDATE ON tablename
FOR EACH ROW EXECUTE PROCEDURE suppress_redundant_updates_trigger()
;
```

在大多数情况下，应让该触发器对每一行最后触发。由于触发器按名称顺序触发，
应选择一个排序位于表上其他所有触发器名称之后的名称。

关于创建触发器的更多信息，请参见CREATE TRIGGER。

第 10 章 类型转换

SQL语句可能有意或无意地要求在同一表达式中混用不同的数据类型。
PostgreSQL提供了丰富的机制来求值这类混合类型表达式。

在很多情况下，用户并不需要理解类型转换机制的细节。不过，
PostgreSQL执行的隐式类型转换会影响查询结果。必要时，可以使用
显式类型转换来控制这些结果。

本章介绍PostgreSQL的类型转换机制和约定。
有关特定数据类型以及允许使用的函数和操作符的更多信息，请参阅第 8 章和第 9 章中的相关章节。

10.1. 概述

SQL是一种强类型语言。也就是说，
每个数据项都关联着一个决定其行为和允许用法的数据类型。
PostgreSQL拥有可扩展的类型系统，比其他SQL实现更通用也更灵活。因此，
PostgreSQL中的大多数类型转换行为都由通用规则控制，而不是依赖临时性的启发式规则。
这使得即使表达式中包含用户定义类型，也能够使用混合类型表达式。

PostgreSQL扫描器/解析器将词法元素分成五个基本类别：整数、非整数数字、字符串、
标识符和关键字。大多数非数值类型的常量会先被归类为字符串。
SQL语言定义允许给字符串指定类型名，而
PostgreSQL可以利用这一机制让解析器从一开始就沿着正确的路径处理。例如，查询：

```
SELECT text 'Origin' AS "label", point '(0,0)' AS "value";

 label  | value
--------+-------
 Origin | (0,0)
(1 row)
```

包含两个字面量，它们的类型分别是`text`和`point`。
如果没有为某个字符串字面量指定类型，那么它最初会被赋予占位类型`unknown`，
并在后文所述的后续阶段解析。

在PostgreSQL解析器中，有四种基本的SQL结构需要采用不同的类型转换规则：

函数调用

PostgreSQL类型系统的很大一部分建立在一套丰富的函数之上。
函数可以有一个或多个参数。由于PostgreSQL支持函数重载，
单凭函数名无法唯一确定要调用哪个函数；
解析器必须根据所提供参数的数据类型选择正确的函数。

操作符

PostgreSQL允许表达式使用前缀和后缀一元（单参数）操作符，以及二元（双参数）
操作符。与函数一样，操作符也可以重载，因此同样存在选择正确操作符的问题。

值存储

SQL的`INSERT`和`UPDATE`语句会把表达式的结果放入表中。
语句中的表达式必须与目标列的类型匹配，并且在必要时转换为目标列类型。

`UNION`、`CASE`和相关结构

由于经UNION合并的`SELECT`语句的所有查询结果都必须出现在同一组列中，因此每个
`SELECT`子句的结果类型必须彼此匹配，并转换为统一的一组类型。类似地，`CASE`

结构中的结果表达式必须转换为某种公共类型，这样整个CASE
表达式才有确定的输出类型。对于ARRAY构造以及GREATEST和LEAST函数，
情况也是如此。

系统目录保存了哪些数据类型之间存在哪些类型转换，以及如何执行这些转换的信息。
用户还可以使用CREATE CAST命令添加额外的类型转换。（这通常与定义新数据类型一并完成。内置类型之间的类型转换集合经过精心设计，最好不要改动。）

解析器还提供了一种额外的启发式规则，
可在存在隐式类型转换的一组类型之间更准确地判定适当的转换行为。
数据类型被划分为若干基本的类型分类，包括boolean、numeric、string、bitstring、datetime、timespan、geometric、network以及用户定义类型。
（列表见表 45.51；不过也可以创建自定义类型分类。）每个分类中可以有一个或多个首选类型，当存在多种可能类型可选时，会优先选择它们。
通过仔细选择首选类型和可用的隐式类型转换，
可以确保有歧义的表达式（即存在多个候选解析方案的表达式）能够以有用的方式被解析。

所有类型转换规则都以以下几个原则为基础：

- 隐式转换绝不应产生令人意外或无法预测的结果。
- 如果查询不需要隐式类型转换，解析器和执行器就不应承担额外开销。也就是说，
  如果查询本身书写正确且类型已经匹配，那么执行时不应让解析器额外耗时，
  也不应在查询中引入不必要的隐式类型转换调用。
- 此外，如果某个查询通常需要先对参数做隐式类型转换才能调用某个函数，
  而此后用户又定义了一个参数类型正确的新函数，那么解析器应改为使用这个新函数，
  而不再通过隐式转换去调用旧函数。

10.2. 操作符

操作符表达式所引用的具体操作符按照下述过程确定。注意，
该过程还会间接受到相关操作符优先级的影响，
因为优先级决定了哪些子表达式会被视为哪些操作符的输入。详见第 4.1.6 节。

操作符类型解析

1. 从系统目录pg_operator中选取要考虑的操作符。
   如果使用的是未带模式限定的操作符名（通常如此），
   则考虑当前搜索路径中可见且名称和参数个数匹配的操作符（见第 5.7.3 节）。如果给出了限定名，则只考虑指定模式中的操作符。
   - (可选的) 如果搜索路径中找到多个参数类型完全相同的操作符，只考虑路径中最早出现的那一个。参数类型不同的操作符则不受搜索路径位置影响，处于同等地位。
2. 检查是否存在一个恰好接受输入参数类型的操作符。
   如果存在（在被考虑的操作符集合中至多只有一个精确匹配），就使用它。
   a. (可选的) 如果二元操作符调用的一个参数是unknown类型，则在本次检查中假定它与另一个参数类型相同。涉及两个unknown输入的调用，或带有unknown
      输入的一元操作符，在这一步都不可能找到匹配。
   b. (可选的) 如果二元操作符调用的一个参数是unknown类型，而另一个是域类型，则接着检查是否存在两边都恰好接受该域基础类型的操作符；如果有，就使用它。
3. 寻找最佳匹配。
   a. 丢弃那些输入类型不匹配，且也无法通过隐式类型转换实现匹配的候选操作符。
      为此，假定unknown字面量可以转换为任何类型。如果只剩一个候选操作符，
      就使用它；否则继续下一步。

b. 如果任何输入参数是域类型，则在后续所有步骤中都把它当作该域的基础类型。
这样可以确保在解决操作符歧义时，域的行为与其基础类型一致。

c. 遍历所有候选操作符，保留那些在输入类型上精确匹配数最多的候选项。
如果没有任何候选项存在精确匹配，则保留全部候选项。
如果只剩一个候选操作符，就使用它；否则继续下一步。

d. 遍历所有候选操作符，在需要类型转换的位置上，
保留接受首选类型（属于输入数据类型的类型分类）的位置数最多的候选项。
如果没有候选项接受首选类型，则保留全部候选项。如果只剩一个候选操作符，
就使用它；否则继续下一步。

e. 如果有任何输入参数是`unknown`，
就检查剩余候选操作符在这些参数位置上接受哪些类型分类。在每个位置上，
如果有任何候选项接受`string`分类，就选择该分类。
（这种偏向字符串的做法是合适的，
因为unknown类型的字面量看起来像字符串。）否则，
如果所有剩余候选项都接受同一种类型分类，就选择该分类；否则失败，
因为在缺少更多线索的情况下无法推断出正确选择。
然后丢弃不接受所选类型分类的候选项。此外，
如果有任何候选项接受该分类中的首选类型，
就丢弃在该参数上接受非首选类型的候选项。如果没有候选项通过这些测试，
则保留全部候选项。如果只剩一个候选操作符，就使用它；否则继续下一步。

f. 如果同时存在`unknown`参数和已知类型参数，
且所有已知类型参数都具有相同类型，则假定`unknown`参数也属于该类型，
并检查哪些候选项能在`unknown`参数位置接受该类型。
如果恰好只有一个候选项通过测试，就使用它；否则失败。

下面是一些示例。

例 10.1. 阶乘操作符类型解析

标准目录中只定义了一个阶乘操作符（后缀 `!`），它接受一个类型为 `bigint` 的参数。扫描器为以下查询表达式中的参数赋予初始类型 `integer`：

```
SELECT 40 ! AS "40 factorial";

                   40 factorial
----------------------------------------------------
 815915283247897734345611269596115894272000000000
(1 row)
```

因此，解析器会对操作数进行类型转换，该查询等价于：

```
SELECT CAST(40 AS bigint) ! AS "40 factorial";
```

例 10.2. 字符串连接操作符类型解析

处理字符串类型以及复杂的扩展类型时，常常会使用类似字符串的语法。
未指定类型的字符串会与可能的操作符候选项进行匹配。

一个参数未指定类型的例子：

```
SELECT text 'abc' || 'def' AS "text and unknown";

 text and unknown
------------------
```

```
 abcdef
(1 row)
```

在这种情况下，解析器会查看是否存在一个两边参数都接受text的操作符。既然存在，它就会假定第二个参数应解释为text类型。

下面是两个未指定类型值的连接：

```
SELECT 'abc' || 'def' AS "unspecified";

 unspecified
-------------
 abcdef
(1 row)
```

这里对使用哪种类型没有初始提示，因为查询中没有指定任何类型。因此，解析器会查找所有候选操作符，并发现既有接受字符串分类输入的候选项，也有接受位串分类输入的候选项。由于在可用时优先选择字符串分类，所以会选中该分类，再使用字符串的首选类型text作为解析这些unknown类型字面量的具体类型。

例 10.3. 绝对值与取反操作符类型解析

PostgreSQL操作符目录中为前缀操作符@提供了多个条目，它们分别实现各种数值数据类型的绝对值操作。其中一个条目对应float8，它是数值分类中的首选类型。因此，当遇到一个unknown输入时，PostgreSQL会使用该条目：

```
SELECT @ '-4.5' AS "abs";
 abs
-----
 4.5
(1 row)
```

这里，系统在应用所选操作符之前，已经把unknown类型字面量隐式解析为float8。我们可以验证使用的确实是float8，而不是其他类型：

```
SELECT @ '-4.5e500' AS "abs";

ERROR:  "-4.5e500" is out of range for type double precision
```

另一方面，前缀操作符~（按位取反）只对整数数据类型定义，并没有为float8定义。因此，如果对~试一个类似的例子，会得到：

```
SELECT ~ '20' AS "negation";

ERROR:  operator is not unique: ~ "unknown"
HINT:  Could not choose a best candidate operator. You might need
 to add
explicit type casts.
```

这是因为系统无法决定几个可能的~操作符中应优先选择哪一个。我们可以通过显式类型转换来帮助它：

```
SELECT ~ CAST('20' AS int8) AS "negation";

 negation
----------
      -21
(1 row)
```

例 10.4. 数组包含操作符类型解析

下面是另一个解析带有一个已知类型输入和一个未知类型输入的操作符的例子：

```
SELECT array[1,2] <@ '{1,2,3}' as "is subset";

 is subset
-----------
 t
(1 row)
```

PostgreSQL操作符目录中为中缀操作符<@定义了多个条目，
但左侧能够接受整数数组的只有两种：数组包含（anyarray <@ anyarray）和范围包含（anyelement <@ anyrange）。由于这些多态伪类型（见第 8.19 节）都不被视为首选类型，解析器无法据此消除歧义。不过，步骤 3.f要求它假定unknown类型字面量与另一输入具有相同类型，也就是整数数组。这样两个操作符中只有一个能够匹配，
因此会选择数组包含。（如果选中范围包含，就会报错，
因为该字符串的格式并不是合法的范围字面量。）

例 10.5. 域类型上的自定义操作符

用户有时会尝试声明只适用于某个域类型的操作符。这是可行的，
但远没有看上去那样有用，
因为操作符解析规则被设计成优先选择作用于域基础类型的操作符。考虑下面的例子：

```
CREATE DOMAIN mytext AS text CHECK(...);
CREATE FUNCTION mytext_eq_text (mytext, text) RETURNS boolean AS ..
.;
CREATE OPERATOR = (procedure=mytext_eq_text, leftarg=mytext,
 rightarg=text);
CREATE TABLE mytable (val mytext);

SELECT * FROM mytable WHERE val = 'foo';
```

这个查询不会使用自定义操作符。解析器首先会检查是否存在mytext = mytext操作符（步骤 2.a），但并不存在；然后它会考虑该域的基础类型text，并检查是否存在text = text操作符（步骤 2.b），而这是存在的；于是它会把unknown类型的字面量解析为text，并使用text = text操作符。要让自定义操作符被使用，唯一的方法是显式地对该字面量做类型转换：

```
SELECT * FROM mytable WHERE val = text 'foo';
```

这样就会根据精确匹配规则立即找到mytext = text操作符。如果解析过程进入最佳匹配规则，它们会主动排斥作用于域类型的操作符。如果不是这样，
这类操作符会导致过多的操作符歧义失败，
因为类型转换规则总是把域视为可以转换到其基础类型或从其基础类型转换而来，
因此域操作符会在所有与基础类型上同名操作符相同的场景中都被视为可用。

10.3. 函数

函数调用所引用的具体函数按照下述过程确定。

函数类型解析

1. 从系统目录pg_proc中选取要考虑的函数。如果使用的是未带模式限定的函数名，
   则考虑当前搜索路径中可见且名称和参数个数匹配的函数（见第 5.7.3 节）。如果给出了限定名，则只考虑指定模式中的函数。

a. (可选的) 如果搜索路径中找到多个参数类型完全相同的函数，只考虑路径中最早出现的那一个。参数类型不同的函数则不受搜索路径位置影响，处于同等地位。

b. (可选的) 如果函数声明了一个`VARIADIC`数组参数，而调用时没有使用`VARIADIC`关键字，则把数组参数视为一个或多个其元素类型的参数，个数按匹配该调用的需要确定。扩展之后，该函数的有效参数类型可能与某个非variadic函数完全相同。在这种情况下，使用搜索路径中较早出现的函数；如果两个函数位于同一模式，则优先选择非variadic的那个。

c. (可选的) 带有参数默认值的函数，被认为可以匹配任何省略了零个或多个可使用默认值的参数位置的调用。如果不止一个这样的函数能匹配某个调用，则使用搜索路径中最早出现的那一个。如果同一模式中存在两个或更多这样的函数，并且它们在未使用默认值的参数位置上的参数类型相同（如果它们具有不同的带有默认值的参数集合，这种情况是可能的），系统就无法判断该偏好哪一个；如果找不到更好的匹配，就会报“有歧义的函数调用”错误。

2. 检查是否存在一个恰好接受输入参数类型的函数。如果存在（在被考虑的函数集合中至多只有一个精确匹配），就使用它。在这种情况下，应通过对参数进行类型转换来强制获得精确匹配。（涉及`unknown`的情况在这一步永远找不到匹配。）

3. 如果没有找到精确匹配，就检查该函数调用是否看起来像一种特殊的类型转换请求。这种情况发生在：函数调用只有一个参数，且函数名与某个数据类型的（内部）名称相同。此外，函数参数必须是unknown类型字面量，或者是一种可以二进制强制转换为该命名数据类型的类型，或者是一种可以通过应用其I/O 函数转换为该命名数据类型的类型（也就是说，该转换要么转到某种标准字符串类型，要么从某种标准字符串类型转来）。满足这些条件时，该函数调用会被当作一种`CAST`形式。[1]

4. 寻找最佳匹配。

   a. 丢弃那些输入类型不匹配，且也无法通过隐式类型转换实现匹配的候选函数。为此，假定`unknown`字面量可以转换为任何类型。如果只剩一个候选函数，就使用它；否则继续下一步。

   b. 如果任何输入参数是域类型，则在后续所有步骤中都把它当作该域的基础类型。这样可以确保在解决函数歧义时，域的行为与其基础类型一致。

   c. 遍历所有候选函数，保留那些在输入类型上精确匹配数最多的候选项。如果没有任何候选项存在精确匹配，则保留全部候选项。如果只剩一个候选函数，就使用它；否则继续下一步。

   d. 遍历所有候选函数，在需要类型转换的位置上，保留接受首选类型（属于输入数据类型的类型分类）的位置数最多的候选项。如果没有候选项接受首选类型，则保留全部候选项。如果只剩一个候选函数，就使用它；否则继续下一步。

   e. 如果有任何输入参数是`unknown`，就检查剩余候选函数在这些参数位置上接受哪些类型分类。在每个位置上，如果有任何候选项接受`string`分类，就选择该分类。（这种偏向字符串的做法是合适的，因为unknown类型的字面量看起来像字符串。）否则，如果所有剩余候选项都接受同一种类型分类，就选择该分类；否则失败，因为在缺少更多线索的情况下无法推断出正确选择。

[1] 之所以有这一步，是为了在不存在实际类型转换函数的情况下仍支持函数风格的类型转换写法。如果存在类型转换函数，按惯例它会以其输出类型命名，因此无需为此设置特殊情况。更多说明见CREATE CAST。

然后丢弃不接受所选类型分类的候选项。此外，
如果有任何候选项接受该分类中的首选类型，
就丢弃在该参数上接受非首选类型的候选项。如果没有候选项通过这些测试，
则保留全部候选项。如果只剩一个候选函数，就使用它；否则继续下一步。

f. 如果同时存在unknown参数和已知类型参数，
且所有已知类型参数都具有相同类型，则假定unknown参数也属于该类型，
并检查哪些候选项能在unknown参数位置接受该类型。
如果恰好只有一个候选项通过测试，就使用它；否则失败。

注意，操作符类型解析和函数类型解析的“最佳匹配”规则是完全相同的。下面是一些示例。

例 10.6. round 函数参数类型解析

只有一个接受两个参数的round函数；它的第一个参数类型是numeric，第二个参数类型是integer。因此下面的查询会自动把类型为integer的第一个参数转换为numeric：

```
SELECT round(4, 4);

 round
--------
 4.0000
(1 row)
```

该查询实际上会被解析器改写成：

```
SELECT round(CAST (4 AS numeric), 4);
```

由于带小数点的数字常量最初会被赋予numeric类型，下面的查询不需要类型转换，
因此可能稍微更高效一些：

```
SELECT round(4.0, 4);
```

例 10.7. substr 函数类型解析

存在多个substr函数，其中一个接受text和integer。
如果以未指定类型的字符串常量调用，系统会选择接受首选类型分类string
参数的候选函数（也就是text类型的那个）。

```
SELECT substr('1234', 3);

 substr
--------
     34
(1 row)
```

如果字符串被声明为varchar类型，例如它来自表中的某一列，
那么解析器会尝试把它转换为text：

```
SELECT substr(varchar '1234', 3);

 substr
--------
     34
(1 row)
```

解析器会把它改写成如下形式：

```
SELECT substr(CAST (varchar '1234' AS text), 3);
```

> **注意**
>
> 解析器从pg_cast目录得知text与varchar二进制兼容，
> 这意味着把其中一种类型传给接受另一种类型的函数时，无需做任何物理转换。
> 因此，这种情况下实际上不会插入类型转换调用。

如果用integer类型参数调用该函数，解析器会尝试把它转换为text：

```
SELECT substr(1234, 3);
ERROR:  function substr(integer, integer) does not exist
HINT:  No function matches the given name and argument types. You
 might need
to add explicit type casts.
```

这行不通，因为integer并没有到text的隐式类型转换。不过，显式类型转换可以：

```
SELECT substr(CAST (1234 AS text), 3);

 substr
--------
     34
(1 row)
```

10.4. 值存储

要插入表中的值会按照以下步骤转换为目标列的数据类型。

值存储的类型转换

1. 检查是否与目标类型精确匹配。

2. 否则，尝试把表达式转换为目标类型。如果这两种类型之间存在已登记的类型转换，这就会成功。如果表达式是unknown类型字面量，则会把字符串字面量的内容送入目标类型的输入转换例程。

3. 检查目标类型是否有尺寸调整类型转换。
   尺寸调整类型转换是该类型到其自身的类型转换。如果在pg_cast目录中找到了这种转换，就在把表达式存入目标列之前先应用它。
   此类类型转换的实现函数总是额外接受一个integer参数，用来接收目标列的atttypmod值（通常是其声明长度，尽管不同数据类型对atttypmod的解释各不相同）；它还可能接受第三个boolean参数，用来说明该类型转换是显式还是隐式的。
   类型转换函数负责应用所有与长度相关的语义，例如长度检查或截断。

例 10.8. character 的存储类型转换

对于声明为character(20)的目标列，下面的语句表明存储值会被调整到正确的长度：

```
CREATE TABLE vv (v character(20));
INSERT INTO vv SELECT 'abc' || 'def';
SELECT v, octet_length(v) FROM vv;

          v           | octet_length
----------------------+--------------
 abcdef               |           20
(1 row)
```

这里实际发生的是，两个unknown类型字面量默认都被解析为text，从而使||操作符被解析为text的连接操作。然后，操作符得到的text结果被转换为bpchar（“用空格填充的字符”，blank-padded char，即character数据类型的内部名称），以匹配目标列类型。（由于从text到bpchar的转换是二进制可强制转换的，这一步不会插入任何真实的函数调用。）最后，系统从系统目录中找到尺寸调整函数bpchar(bpchar, integer, boolean)，并把它应用到操作符结果以及存储列长度上。这个特定于类型的函数会执行所需的长度检查并补齐填充空格。

10.5. UNION、CASE及相关结构

SQL 的UNION结构必须让可能不同的类型彼此匹配，以形成单一结果集。该解析算法会分别应用到联合查询的每个输出列。INTERSECT和EXCEPT结构以与UNION相同的方式解析不同类型。CASE、ARRAY、VALUES以及GREATEST和LEAST构造使用完全相同的算法来匹配其组成表达式并选择结果数据类型。

UNION、CASE及相关结构的类型解析

1. 如果所有输入都是同一类型，且该类型不是unknown，就解析为该类型。
2. 如果任何输入是域类型，则在后续所有步骤中都把它当作该域的基础类型。[2]
3. 如果所有输入都是unknown类型，则解析为text类型（字符串分类的首选类型）。否则，忽略unknown输入。
4. 如果非unknown输入并非全都属于同一类型分类，则失败。
5. 选择该分类中的首选类型（如果第一个非unknown输入类型恰好是首选类型，则选它）。
6. 否则，选择最后一个能让所有在它之前的非unknown输入都隐式转换为它的非unknown输入类型。（这样的类型总是存在的，因为列表中至少第一个类型就满足这一条件。）
7. 把所有输入都转换为所选类型。如果某个输入到所选类型不存在转换，则失败。

下面是一些示例。

例 10.9. 未充分指定类型的 UNION 中的类型解析

```
SELECT text 'a' AS "text" UNION SELECT 'b';

 text
------
 a
 b
(2 rows)
```

这里，unknown类型字面量'b'会被解析为text类型。

例 10.10. 简单 UNION 中的类型解析

```
SELECT 1.2 AS "numeric" UNION SELECT 1;

 numeric
---------
```

[2] 这在某种程度上类似于操作符和函数对域输入的处理方式；只要用户确保所有输入都隐式或显式地正好属于该域类型，这种行为就能让域类型在UNION或类似结构中得以保留。否则，将优先使用该域的基础类型。

```
       1
     1.2
(2 rows)
```

字面量1.2的类型是numeric，而integer值1可以隐式转换为numeric，因此使用该类型。

例 10.11. 次序对调的 UNION 中的类型解析

```
SELECT 1 AS "real" UNION SELECT CAST('2.2' AS REAL);

 real
------
    1
  2.2
(2 rows)
```

这里，由于real类型不能隐式转换为integer，而integer可以隐式转换为real，因此UNION结果类型被解析为real。

第 11 章 索引

索引是增强数据库性能的一种常见方式。
索引允许数据库服务器比没有索引时更快地找到并检索特定行。
但索引也会给整个数据库系统带来额外开销，因此应当合理使用。

11.1. 简介

假设我们有一个类似如下的表：

```
CREATE TABLE test1 (
    id integer,
    content varchar
);
```

并且应用会发出很多如下形式的查询：

```
SELECT content FROM test1 WHERE id = constant;
```

如果没有预先准备，系统就必须逐行扫描整个test1表，以找出所有匹配项。如果test1中有很多行，而这种查询只会返回很少几行（甚至可能一行也没有，或者只有一行），这显然是低效的办法。但是如果系统被要求维护一个基于id列的索引，它就可以用更高效的方法定位匹配行。例如，它可能只需要在一棵搜索树中向下走几层。

大多数非虚构类书籍也采用了类似的方法：读者经常查找的术语和概念会被收集到书末按字母顺序排列的索引中。感兴趣的读者可以相对快速地浏览索引并翻到相应页面，而不必通读整本书才能找到自己感兴趣的内容。
正如作者需要预判读者可能查找哪些条目一样，数据库程序员也需要预见哪些索引会有用。

如前所述，下列命令可用于在id列上创建索引：

```
CREATE INDEX test1_id_index ON test1 (id);
```

test1_id_index这个名字可以自由选择，但最好选一个以后仍能让你记住该索引用途的名字。

要删除索引，使用DROP INDEX命令。索引可以在任何时候添加到表上，也可以在任何时候从表上删除。

一旦创建了索引，就无需再进行额外干预：系统会在表被修改时更新索引，并且会在它认为这样做比顺序扫描表更高效时，在查询中使用该索引。不过，你可能仍需要定期运行ANALYZE命令来更新统计信息，以便查询规划器作出更有根据的决策。关于如何判断某个索引是否被使用，以及规划器何时和为何可能选择不使用索引，可参见第 14 章。

索引还可以让带有搜索条件的UPDATE和DELETE命令受益。索引也可以用于连接搜索。因此，定义在连接条件组成列上的索引也能显著加速带连接的查询。

在大表上创建索引可能需要很长时间。默认情况下，PostgreSQL允许在创建索引期间并行执行读取（SELECT语句），但写入（INSERT、UPDATE、DELETE）会被阻塞，直到索引构建完成。在生产环境中，这往往不可接受。可以允许在创建索引时并行写入，但有若干注意事项需要了解，详见并发构建索引。

索引创建之后，系统还必须使其与表保持同步。这会给数据操作增加开销。因此，那些在查询中很少使用或从不使用的索引应当被移除。

11.2. 索引类型

PostgreSQL提供了多种索引类型：B-树、Hash、GiST、SP-GiST 和 GIN。每种索引类型都采用不同的算法，分别最适合不同类型的查询。默认情况下，`CREATE INDEX`命令创建的是 B-树索引，它适用于最常见的场景。

B-树能够处理可以按某种顺序排序的数据上的等值查询和范围查询。特别是，只要已索引列参与了下列任一操作符的比较，PostgreSQL查询规划器就会考虑使用 B-树索引：

```
<
<=
=
>=
>
```

与这些操作符组合等价的构造，例如`BETWEEN`和`IN`，也可以通过 B-树索引搜索来实现。此外，索引列上的`IS NULL`或`IS NOT NULL`条件也可以配合 B-树索引使用。

如果模式是常量，并且锚定在字符串起始位置，优化器也可以对涉及模式匹配操作符`LIKE`和`~`的查询使用 B-树索引，例如`col LIKE 'foo%'`或`col ~ '^foo'`，但不能用于`col LIKE '%bar'`。不过，如果你的数据库没有使用 C 区域设置，就需要以一个特殊操作符类来创建该索引，才能支持模式匹配查询的索引化，详见下文第 11.9 节。B-树索引也可以用于`ILIKE`和`~*`，但前提是模式以非字母字符开头，也就是不会受到大小写转换影响的字符。

B-树索引还可以用于按排序顺序取回数据。这并不总是比简单扫描再排序更快，
但通常会很有帮助。

Hash 索引只能处理简单的等值比较。只要已索引列参与的是使用=操作符的比较，查询规划器就会考虑使用 Hash 索引。使用以下命令创建 Hash 索引：

```
CREATE INDEX name ON table USING hash (column);
```

> **小心**
>
> 哈希索引操作目前不会写入 WAL 日志，因此如果发生过未写入的更改，数据库崩溃后可能需要用`REINDEX`重建哈希索引。此外，在初始基础备份之后，
> 哈希索引的更改不会通过流复制或基于文件的复制进行复制，
> 因此它们会对随后使用它们的查询给出错误的答案。由于这些原因，
> 目前不建议使用哈希索引。

GiST 索引并不是某一种单独的索引，而是一种基础设施，可在其中实现许多不同的索引策略。因此，GiST 索引可使用哪些具体操作符，取决于所采用的索引策略（即操作符类）。例如，PostgreSQL标准发布版中包含了若干二维几何数据类型的 GiST 操作符类，它们支持使用下列操作符的索引化查询：

```
<<
&<
&>
>>
<<|
&<|
|&>
|>>
@>
<@
```

```
~=
&&
```

（这些操作符的含义见第 9.11 节。）还有许多其他 GiST 操作符类可在`contrib`集合或独立项目中获得。更多信息见第 53 章。

GiST 索引还能够优化“最近邻”搜索，例如：

```
SELECT * FROM places ORDER BY location <-> point '(101,456)' LIMIT
 10;
```

这会找出距离给定目标点最近的十个地点。是否能做到这一点，
同样取决于所使用的具体操作符类。

与 GiST 一样，SP-GiST 索引也提供一种支持多种搜索的基础设施。SP-GiST 允许实现大量不同的、非平衡的、基于磁盘的数据结构，例如四叉树、k-d 树和后缀树（trie）。例如，PostgreSQL标准发布版中包含了用于二维点的 SP-GiST 操作符类，它支持使用下列操作符的索引化查询：

```
<<
>>
~=
<@
<^
>^
```

（这些操作符的含义见第 9.11 节。）更多信息见第 54 章。

GIN 索引是倒排索引，可以处理包含多个键的值，例如数组。与 GiST 和 SP-GiST 一样，GIN 也能支持多种不同的用户定义索引策略，GIN 索引可使用哪些具体操作符同样取决于索引策略。例如，PostgreSQL标准发布版中包含用于一维数组的 GIN 操作符类，它们支持使用下列操作符的索引化查询：

```
<@
@>
=
&&
```

（这些操作符的含义见第 9.18 节。）还有许多其他 GIN 操作符类可在`contrib`集合或独立项目中获得。更多信息见第 55 章。

11.3. 多列索引

一个索引可以定义在表的多个列上。例如，如果你有一个如下形式的表：

```
CREATE TABLE test2 (
  major int,
  minor int,
  name varchar
);
```

（假设你把`/dev`目录保存在数据库里.....）并且经常发出如下查询：

```
SELECT name FROM test2 WHERE major = constant AND minor = constant;
```

那么将`major`和`minor`两列一起建一个索引可能是合适的，例如：

```
CREATE INDEX test2_mm_idx ON test2 (major, minor);
```

当前，只有 B-树、GiST 和 GIN 索引类型支持多列索引。最多可以指定 32 列。（这个限制可以在构建 PostgreSQL 时修改；参见文件 `pg_config_manual.h`。）

多列 B-树索引可以用于涉及索引任意列子集的查询条件，但当对前导（最左）列存在约束时，索引效率最高。精确的规则是：对前导列的等值约束，
再加上第一个没有等值约束的列上的任何不等约束，会被用来限制索引扫描的范围。
这些列右侧列上的约束会在索引中进行检查，因此可以减少访问表本体的次数，
但不会缩小必须扫描的索引范围。例如，给定一个 `(a, b, c)` 上的索引和查询条件 `WHERE a = 5 AND b >= 42 AND c < 77`，索引必须从第一个 `a` = 5 且 `b` = 42 的条目扫描到最后一个 `a` = 5 的条目。`c` >= 77 的索引项会被排除，但仍然必须扫描这些项。原则上，这个索引也可以用于只对 `b` 和/或 `c` 有约束、而对 `a` 没有约束的查询，但必须扫描整个索引，因此在大多数情况下，规划器会倾向于顺序扫描表，而不是使用该索引。

多列 GiST 索引可以用于涉及索引任意列子集的查询条件。附加列上的条件会限制索引返回的项，但决定索引需要扫描多少内容的，最重要的仍是第一列上的条件。
如果第一列只有很少几个非重复值，即使其他列有很多非重复值，GiST 索引也会相对低效。

多列 GIN 索引可以用于涉及索引任意列子集的查询条件。与 B-树或 GiST 不同，无论查询条件使用的是哪些索引列，GIN 的索引搜索效果都一样。

当然，每一列都必须配合适合该索引类型的操作符使用；
涉及其他操作符的子句不会被考虑。

多列索引应谨慎使用。在大多数情况下，单列索引已经足够，而且更省空间、也更省时间。
除非表的使用方式极其程式化，否则超过三列的索引通常不会有帮助。
关于不同索引配置优缺点的讨论，还可参见第 11.5 节。

11.4. 索引和ORDER BY

除了简单地找到查询要返回的行之外，索引还可能能够以某种特定排序顺序交付这些行。
这使得查询中的`ORDER BY`要求无需额外排序步骤即可满足。在PostgreSQL当前支持的索引类型中，只有 B-树能产生有序输出，其他索引类型返回匹配行的顺序则未指定，并且依赖具体实现。

规划器在满足`ORDER BY`要求时，会考虑两种方案：扫描一个与该要求匹配的可用索引，或者按物理顺序扫描表再显式排序。对于需要扫描表中很大一部分内容的查询，
显式排序通常会比使用索引更快，因为它遵循顺序访问模式，需要的磁盘 I/O 更少。只需取出少量行时，索引更有价值。一个重要的特例是`ORDER BY`与`LIMIT` *n*组合使用：显式排序必须处理所有数据才能找出最前面的*n*行，而如果有一个与`ORDER BY`匹配的索引，就可以直接取回前*n*行，而完全不必扫描剩余数据。

默认情况下，B-树索引按升序存储其项，并将空值放在最后。这意味着，对列`x`
上的索引进行前向扫描，会产生满足`ORDER BY x`的输出（更完整地说，是`ORDER BY x ASC NULLS LAST`）。索引也可以反向扫描，从而产生满足`ORDER BY x DESC`的输出（更完整地说，是`ORDER BY x DESC NULLS FIRST`，因为`NULLS FIRST`是`ORDER BY DESC`的默认行为）。

你可以在创建 B-树索引时通过指定`ASC`、`DESC`、`NULLS FIRST`和/或`NULLS LAST`选项来调整索引的排序方式，例如：

```
CREATE INDEX test2_info_nulls_low ON test2 (info NULLS FIRST);
CREATE INDEX test3_desc_index ON test3 (id DESC NULLS LAST);
```

一个按升序且空值在前存储的索引，根据扫描方向不同，可以满足`ORDER BY x ASC NULLS FIRST`或`ORDER BY x DESC NULLS LAST`。

你可能会疑惑，既然借助两个选项再加上反向扫描的可能性就能覆盖所有`ORDER BY`变体，为什么还要提供四个选项。对于单列索引，这些选项确实是冗余的，
但对于多列索引它们就可能很有用。考虑一个`(x, y)`上的两列索引：前向扫描时它能满

足ORDER BY x, y，反向扫描时能满足ORDER BY x DESC, y DESC。但应用也可能经常需要ORDER BY x ASC, y DESC。普通索引无法产生这种顺序，而如果把索引定义成(x ASC, y DESC)或(x DESC, y ASC)，就可以做到。

显然，具有非默认排序方式的索引是一项相当专门的特性，
但在某些查询上它们有时能带来巨大的加速。是否值得维护这样的索引，
取决于你需要这种特殊排序的查询出现得有多频繁。

11.5. 组合多个索引

单次索引扫描只能利用那些在索引列上使用了该索引操作符类中操作符、并且彼此以AND连接的查询子句。例如，给定一个(a, b)上的索引，查询条件WHERE a = 5 AND b = 6可以使用该索引，而WHERE a = 5 OR b = 6这样的查询就不能直接使用该索引。

幸运的是，PostgreSQL能够组合多个索引（包括同一个索引的多次使用），
以处理单次索引扫描无法实现的情况。系统可以在多个索引扫描之间形成AND和OR条件。
例如，WHERE x = 42 OR x = 47 OR x = 53 OR x = 99这样的查询，可以拆分成对x上的一个索引进行四次独立扫描，每次扫描使用一个查询子句。然后将这些扫描的结果做 OR 运算以得到最终结果。另一个例子是，如果我们在x和y上分别有索引，那么WHERE x = 5 AND y = 6这样的查询，其一种可能实现方式就是分别用两个索引处理相应条件，然后将索引结果做 AND 运算，以确定结果行。

为了组合多个索引，系统会扫描每个所需索引，并在内存中准备一个位图，
给出满足该索引条件的表行位置。然后按查询需要对这些位图进行 AND 和 OR 运算。最后，
再访问并返回实际的表行。表行是按物理顺序访问的，因为位图就是这样布局的；
这意味着原始索引中的任何排序都会丢失，所以如果查询带有ORDER BY子句，就需要单独的排序步骤。也正因为如此，再加上每多一次索引扫描都会增加额外时间，
规划器有时会选择使用简单的单一索引扫描，即使还有其他本来也可以利用的索引。

除了最简单的应用之外，通常会有多种可能有用的索引组合，
数据库开发者必须权衡决定提供哪些索引。有时多列索引最好，
但有时创建独立索引并依赖索引组合功能会更合适。例如，
如果你的工作负载包含一组查询：有时只涉及列x，有时只涉及列y，有时同时涉及两列，
那么你可以选择分别在x和y上创建两个独立索引，
并依赖索引组合来处理同时使用两列的查询。你也可以创建一个(x, y)上的多列索引。对于同时涉及两列的查询，这个索引通常会比索引组合更高效，但正如第 11.3 节中所讨论的，它对于只涉及y的查询几乎没有用处，因此不应只创建这一个索引。
将这个多列索引与一个单独的y索引组合起来，能够取得相当好的效果。对于只涉及x
的查询，多列索引也可以使用，但它会比单独的x索引更大，因此更慢。
最后一种选择是同时创建这三个索引，
但这大概只有在该表被搜索的频率远高于被更新的频率，并且三类查询都很常见时才合理。
如果其中一种查询远没有另外两种常见，
那么你大概只需创建最适合常见查询类型的两个索引即可。

11.6. 唯一索引

索引还可以用于强制列值唯一，或者强制多个列组合值唯一。

```
CREATE UNIQUE INDEX name ON table (column [, ...]);
```

当前，只有 B-树索引可以声明为唯一。

当一个索引被声明为唯一时，就不允许存在多个具有相同索引值的表行。
空值不被视为相等。多列唯一索引只会拒绝那种在多行中所有索引列都相等的情况。

当为表定义唯一约束或主键时，PostgreSQL会自动创建一个唯一索引。
该索引覆盖构成主键或唯一约束的列（必要时是多列索引），
并且正是它在强制执行该约束。

> **注意**
>
> 向表添加唯一约束的首选方式是`ALTER TABLE ... ADD CONSTRAINT`。用索引来强制唯一约束可以被视为一种不应直接访问的实现细节。不过应当知道，无需手工在唯一列上创建索引；那样做只会重复自动创建的索引。

11.7. 表达式索引

索引列不一定非得是底层表中的某一列，也可以是一个由表中一列或多列计算得到的函数或标量表达式。这个特性有助于基于计算结果来快速访问表。

例如，进行大小写不敏感比较的一种常见方式，是使用`lower`函数：

```
SELECT * FROM test1 WHERE lower(col1) = 'value';
```

如果为`lower(col1)`函数的结果定义了索引，这个查询就可以使用该索引：

```
CREATE INDEX test1_lower_col1_idx ON test1 (lower(col1));
```

如果我们把这个索引声明为`UNIQUE`，那么它不仅会阻止创建那些`col1`值仅在大小写上不同的行，也会阻止创建`col1`值完全相同的行。因此，表达式索引可用于强制那些无法定义为简单唯一约束的约束。

再例如，如果你经常执行如下查询：

```
SELECT * FROM people WHERE (first_name || ' ' || last_name) = 'John
 Smith';
```

那么创建一个如下索引可能是值得的：

```
CREATE INDEX people_names ON people ((first_name || ' ' || last_
name));
```

正如第二个示例所示，`CREATE INDEX`命令的语法通常要求对索引表达式写上圆括号。当表达式只是一个函数调用时，则可以像第一个示例那样省略这些圆括号。

维护索引表达式的代价相对较高，因为在插入每一行时以及每次更新时都必须计算派生表达式。不过，在使用索引搜索时，索引表达式不会被重新计算，因为它们已经存储在索引中。在上面的两个示例里，系统看到的查询只是`WHERE indexedcolumn = 'constant'`，因此搜索速度与其他简单索引查询相同。因此，当检索速度比插入和更新速度更重要时，表达式索引会很有用。

11.8. 部分索引

部分索引是建立在表的一个子集上的索引；这个子集由一个条件表达式定义（称为部分索引的谓词）。索引中只包含满足该谓词的表行的项。部分索引是一种专门特性，但有几种场景下它会很有用。

使用部分索引的一个主要原因是避免索引常见值。由于搜索常见值（即占全部表行百分之几以上的值）的查询反正也不会使用索引，因此完全没有必要把这些行保留在索引里。这会减小索引尺寸，从而加快那些确实会使用该索引的查询。它也会加快很多表更新操作，因为索引并不需要在所有情况下都更新。例 11.1展示了这种思路的一种可能应用。

例 11.1. 建立一个部分索引以排除常见值

假设你把 Web 服务器访问日志存储在数据库中。大多数访问来自你所在组织的 IP 地址范围，但也有一些来自其他地方（例如使用拨号连接的员工）。如果你按 IP 搜索时主要关心外部访问，那么你大概没有必要为对应于组织内子网的 IP 范围建索引。

假设有一个如下表：

```
CREATE TABLE access_log (
    url varchar,
    client_ip inet,
    ...
);
```

要创建一个适合这个例子的部分索引，可使用如下命令：

```
CREATE INDEX access_log_client_ip_ix ON access_log (client_ip)
WHERE NOT (client_ip > inet '192.168.100.0' AND
           client_ip < inet '192.168.100.255');
```

一个能够使用此索引的典型查询如下：

```
SELECT *
FROM access_log
WHERE url = '/index.html' AND client_ip = inet '212.78.10.32';
```

以下查询无法使用此索引：

```
SELECT *
FROM access_log
WHERE client_ip = inet '192.168.100.23';
```

请注意，这类部分索引要求常见值事先可知，因此最适合用于数据分布不会变化的场景。也可以偶尔重建这类索引以适应新的数据分布，但那会增加维护工作量。

部分索引的另一种可能用途，是把典型查询工作负载不感兴趣的值排除在索引之外，如例 11.2所示。这样会得到与上面相同的好处，但也意味着这些“不感兴趣”的值无法通过该索引访问，即使在那种情况下索引扫描可能是有利的。显然，为这种场景建立部分索引需要大量的谨慎和实验。

例 11.2. 建立一个部分索引以排除不感兴趣的值

如果你有一张表，其中同时包含已开账单和未开账单的订单，而未开账单订单只占整张表的一小部分，却是最常被访问的那些行，那么你可以通过只为未开账单的行创建索引来提升性能。创建该索引的命令如下：

```
CREATE INDEX orders_unbilled_index ON orders (order_nr)
    WHERE billed is not true;
```

一个可能会使用该索引的查询是：

```
SELECT * FROM orders WHERE billed is not true AND order_nr < 10000;
```

不过，这个索引也可以用于那些完全不涉及`order_nr`的查询，例如：

```
SELECT * FROM orders WHERE billed is not true AND amount > 5000.00;
```

这不像在`amount`列上建立部分索引那样高效，因为系统必须扫描整个索引。不过，如果未开账单的订单相对较少，那么即使只是用这个部分索引来找出未开账单订单，也可能是值得的。

请注意，下面这个查询不能使用该索引：

```
SELECT * FROM orders WHERE order_nr = 3501;
```

因为订单 3501 可能属于已开账单订单，也可能属于未开账单订单。

例 11.2也说明了：索引列和谓词中使用的列不必一致。PostgreSQL支持带任意谓词的部分索引，只要其中只涉及正在建立索引的那张表的列。不过要记住，
谓词必须与那些希望从该索引受益的查询中使用的条件相匹配。更准确地说，
只有当系统能够识别出查询的WHERE条件在数学上蕴含该索引的谓词时，
部分索引才能用于该查询。PostgreSQL并没有一个复杂的定理证明器，
来识别那些写法不同但数学上等价的表达式。
（不仅构建这样一个通用定理证明器极其困难，而且它很可能也会慢到失去实际用途。）
系统可以识别简单的不等式蕴含，例如“x < 1”蕴含“x < 2”；否则，谓词条件必须与查询WHERE条件的某一部分完全匹配，否则索引不会被识别为可用。匹配发生在查询规划阶段，
而不是运行时。因此，参数化查询子句无法与部分索引配合工作。例如，
一个带参数的预备查询可能写成“x < ?”，而它无法保证在参数的所有可能取值下都蕴含“x < 2”。

部分索引的第三种可能用途，甚至不要求索引被查询使用。这里的思路是像例 11.3那样，在表的一个子集上创建唯一索引。这样就能在满足索引谓词的那些行之间强制唯一性，
而不会约束不满足谓词的行。

例 11.3. 建立一个部分唯一索引

假设我们有一张描述测试结果的表。我们希望确保对于给定的测试对象和目标组合，
只有一条“成功”记录，但可以有任意多条“失败”记录。实现方法之一如下：

```
CREATE TABLE tests (
    subject text,
    target text,
    success boolean,
    ...
);

CREATE UNIQUE INDEX tests_success_constraint ON tests (subject,
 target)
    WHERE success;
```

当成功测试很少而失败测试很多时，这是一种特别高效的方法。

最后，部分索引还可以用来影响系统的查询计划选择。同样，
分布异常的数据集可能导致系统在实际上不应该使用索引的时候选择使用它。
在这种情况下，可以把索引建成对那个有问题的查询不可用。通常，
PostgreSQL会对索引使用作出合理选择（例如，它会在检索常见值时避免使用索引，
因此前面的例子实际上只是节省索引空间，而不是为了避免使用索引所必需的），
如果出现明显错误的计划选择，那就应当提交 bug 报告。

请记住，建立部分索引意味着你至少与查询规划器一样了解情况，
尤其是你知道索引何时可能是有利的。形成这种认识需要经验，以及对
PostgreSQL中索引工作方式的理解。在大多数情况下，
部分索引相比普通索引的优势都很小。

关于部分索引的更多信息可参见[ston89b]、[olson93]和[seshadri95]。

11.9. 操作符类和操作符族

索引定义可以为索引的每一列指定一个操作符类。

```
CREATE INDEX name ON table (column opclass [sort options] [, ...]);
```

操作符类确定索引针对该列使用哪些操作符。例如，建立在类型int4上的 B-树索引会使用int4_ops类；该操作符类包含针对以下类型的值的比较函数：int4。实际中，列数据类型的默认操作符类通常就足够了。之所以需要操作符类，主要是因为对于某些数据类型，可能存在不止一种有意义的索引行为。例如，可能希望按绝对值或实部对复数数据类型排序。可以为该数据类型定义两个操作符类，并在创建索引时选择合适的类。
操作符类决定基本排序顺序（随后可以通过添加排序选项进行调整：COLLATE，ASC/DESC和/或NULLS FIRST/NULLS LAST）。

除了默认操作符类之外，还有一些内置操作符类：

- 操作符类text_pattern_ops、varchar_pattern_ops和bpchar_pattern_ops分别支持类型text、varchar和char上的 B-树索引。它们与默认操作符类的区别在于，值是严格按字符逐个比较，而不是按照区域设置相关的排序规则比较。因此，当数据库未使用标准“C”区域设置时，这些操作符类适用于涉及模式匹配表达式（LIKE或POSIX 正则表达式）的查询。例如，你可以像这样为一个varchar列建立索引：

  ```
  CREATE INDEX test_index ON test_table (col varchar_pattern_ops);
  ```

  注意，如果你希望涉及普通<、<=、>或>=比较的查询也能使用索引，那么还应该再创建一个使用默认操作符类的索引。这类查询不能使用*xxx*_pattern_ops操作符类。（不过普通等值比较是可以使用这些操作符类的。）
  同一列上可以创建多个使用不同操作符类的索引。如果你使用的是 C 区域设置，则不需要*xxx*_pattern_ops操作符类，因为带默认操作符类的索引在 C 区域设置下就可以用于模式匹配查询。

下面的查询会显示所有已定义的操作符类：

```
SELECT am.amname AS index_method,
       opc.opcname AS opclass_name
    FROM pg_am am, pg_opclass opc
    WHERE opc.opcmethod = am.oid
    ORDER BY index_method, opclass_name;
```

操作符类其实只是一个更大结构的子集，这个结构称为操作符族。当若干数据类型具有相似行为时，定义跨数据类型操作符并让索引支持它们，往往会很有用。为此，每种类型对应的操作符类都必须归入同一个操作符族。跨类型操作符属于该族，但不与该族中任何单独一个类关联。

下面这个查询会显示所有已定义的操作符族，以及每个族中包含的全部操作符：

```
SELECT am.amname AS index_method,
       opf.opfname AS opfamily_name,
       amop.amopopr::regoperator AS opfamily_operator
    FROM pg_am am, pg_opfamily opf, pg_amop amop
    WHERE opf.opfmethod = am.oid AND
          amop.amopfamily = opf.oid
    ORDER BY index_method, opfamily_name, opfamily_operator;
```

11.10. 索引和排序规则

每个索引列只能支持一种排序规则。如果你需要关心多种排序规则，就可能需要多个索引。

考虑以下语句：

```
CREATE TABLE test1c (
    id integer,
```

```
    content varchar COLLATE "x"
);

CREATE INDEX test1c_content_index ON test1c (content);
```

该索引会自动使用底层列的排序规则。因此，下面这种形式的查询：

```
SELECT * FROM test1c WHERE content > constant;
```

可以使用该索引，因为比较默认会使用该列的排序规则。不过，
这个索引不能加速涉及其他排序规则的查询。因此，如果你也关心如下形式的查询：

```
SELECT * FROM test1c WHERE content > constant COLLATE "y";
```

那么可以额外创建一个支持"y"排序规则的索引，例如：

```
CREATE INDEX test1c_content_y_index ON test1c (content COLLATE "y")
;
```

11.11. 检查索引使用情况

尽管PostgreSQL中的索引不需要维护或调优，
但检查真实查询工作负载到底实际使用了哪些索引，仍然很重要。
检查某个单独查询的索引使用情况，可以使用EXPLAIN命令；它在这方面的应用见第 14.1 节。也可以像第 27.2 节中描述的那样，在运行中的服务器上收集索引使用的总体统计信息。

很难给出一个确定应该创建哪些索引的通用流程。
前面各节的示例已经展示了若干典型情形。通常需要做大量实验。
下面这部分会给出一些相关提示：

- 一定要先运行ANALYZE。这个命令会收集表中值分布的统计信息。
  估计查询会返回多少行，需要这些信息；
  而规划器要为每种可能的查询计划分配现实的代价，也需要这些信息。
  如果没有真实统计信息，系统就会假定一些默认值，而这些默认值几乎肯定并不准确。
  因此，在没有运行ANALYZE的情况下去检查应用的索引使用情况，基本上是徒劳的。
  更多信息见第 23.1.3 节和第 23.1.6 节。

- 用真实数据做实验。使用测试数据来设置索引，只会告诉你测试数据需要什么索引，
  仅此而已。

  特别要避免使用非常小的测试数据集。虽然在 100000 行里选 1000 行可能适合用索引，但在 100 行里选 1 行几乎不适合，因为这 100 行大概都能放进一个磁盘页，而没有任何计划能比顺序取一个磁盘页更快。

  另外，在编造测试数据时也要小心；如果应用尚未投入生产，这往往无法避免。
  非常相似、完全随机、或者按排序顺序插入的值，
  都会让统计信息偏离真实数据应有的分布。

- 当索引没有被使用时，强制使用它们对测试会很有帮助。
  有一些运行时参数可以关闭不同的计划类型（见第 18.7.1 节）。例如，关闭顺序扫描（enable_seqscan）和嵌套循环连接（enable_nestloop）这两种最基本的计划，
  就会迫使系统使用不同的计划。如果系统仍然选择顺序扫描或嵌套循环连接，
  那么索引未被使用很可能有更根本的原因，例如查询条件并不匹配该索引。
  （前面各节已经解释了什么样的查询可以使用什么样的索引。）

- 如果强制使用索引后确实使用了索引，那么就有两种可能：要么系统是对的，
  使用索引确实不合适；要么查询计划的代价估计并未反映现实。因此，
  你应该分别在有索引和无索引的情况下对查询计时。EXPLAIN ANALYZE命令在这里会很有帮助。

- 如果发现代价估计是错的，那么又有两种可能。
  总代价是根据每个计划节点的逐行代价乘以该计划节点的选择率估计计算出来的。
  计划节点的估计代价可以通过运行时参数来调整（见第 18.7.2 节）。而选择率估计不准确，则是因为统计信息不足。可以尝试通过调整统计信息收集参数来改进这一点（见ALTER TABLE）。

  如果你无法把这些代价调得更合适，那么可能就不得不退而求其次，显式强制使用索引。
  你也可能希望联系PostgreSQL开发者来研究这个问题。

第 12 章 全文搜索

12.1. 介绍

全文检索（或简称文本检索）提供了识别满足查询条件的自然语言文档的能力，并且可按它们与查询的相关度进行排序。最常见的搜索类型，是找出所有包含给定查询词的文档，并按它们与查询的相似性排序返回。查询和相似性的概念都很灵活，取决于具体应用。最简单的搜索把查询视为一组词，把相似性视为查询词在文档中的出现频率。

文本检索操作符在数据库中已经存在很多年了。PostgreSQL为文本数据类型提供了 `~`、`~*`、`LIKE` 和 `ILIKE` 操作符，但它们缺少现代信息系统所要求的许多关键特性：

- 缺少语言学支持，即便对英语也是如此。正则表达式并不足够，因为它们难以轻松处理词形变化，例如 `satisfies` 和 `satisfy`。搜索 `satisfy` 时，你大概也希望找到包含 `satisfies` 的文档，但实际上可能会漏掉它们。虽然可以用 `OR` 搜索多个派生形式，但这样既繁琐又容易出错（有些词甚至可能有数千种派生形式）。
- 它们不会对搜索结果排序（排名），因此当找到成千上万条匹配文档时就难以实用。
- 它们往往很慢，因为没有索引支持，所以每次搜索都必须处理全部文档。

全文索引允许先对文档进行预处理，并保存索引以供后续快速搜索。预处理包括：

将文档解析成词元。识别出不同类别的词元很有帮助，例如数字、单词、复合词、电子邮件地址等，这样就可以分别处理。原则上，词元类别取决于具体应用，但在大多数场景中，一套预定义的类别已经足够。PostgreSQL使用解析器执行这一步。系统提供了标准解析器，也可以根据特定需求创建自定义解析器。

将词元转换成词位。词位和词元一样都是字符串，但它已经过正规化，使同一个词的不同形式归为一致。例如，正规化几乎总是包含把大写字母折叠为小写，也常常会去掉后缀（如英语中的 `s` 或 `es`）。这样，搜索时无需繁琐地输入所有可能的变体，就能匹配同一个词的不同形式。此外，这一步通常还会去除停用词，也就是那些过于常见、对搜索没有帮助的词。（简言之，词元是文档文本的原始片段，而词位则是被认为适合用于索引和搜索的词。）PostgreSQL使用词典执行这一步。系统提供了多种标准词典，也可以按需创建自定义词典。

以适合搜索的形式存储预处理后的文档。例如，每个文档都可以表示为由正规化词位组成的有序数组。除词位之外，通常还希望存储位置信息，以便用于邻近排名；这样，查询词分布更“密集”的文档会比查询词分散出现的文档得到更高的排名。

词典允许对词元的正规化方式进行细粒度控制。借助合适的词典，你可以：

- 定义不应该被索引的停用词。
- 使用 Ispell 把同义词映射为同一个词。
- 使用分类词典把短语映射为同一个词。
- 使用 Ispell 词典把一个词的不同变体映射到一种规范形式。
- 使用 Snowball 词干分析规则把一个词的不同变体映射到一种规范形式。

系统提供了数据类型 `tsvector` 用于存储预处理后的文档，也提供了数据类型 `tsquery` 用于表示处理过的查询（第 8.11 节）。围绕这两种数据类型还有许多函数和操作符（第 9.13 节），其中最重要的是匹配操作符 `@@`，我们将在第 12.1.2 节中介绍。全文检索还可以借助索引加速（第 12.9 节）。

12.1.1. 什么是文档？

文档是全文检索系统中的搜索单位，例如一篇杂志文章或一封电子邮件。文本检索引擎必须能够解析文档，并保存词位（关键字）与其所属文档之间的关联。随后，就可以利用这些关联来搜索包含查询词的文档。

在 PostgreSQL 中进行搜索时，文档通常是数据库表某一行中的一个文本字段，或者是这类字段的组合（串接），这些字段可能分布在多个表里，也可能是动态取得的。换句话说，文档可以由不同部分拼接而成以供索引，而且未必会以一个整体存储在任何地方。例如：

```
SELECT title || ' ' ||  author || ' ' ||  abstract || ' ' || body
 AS document
FROM messages
WHERE mid = 12;

SELECT m.title || ' ' || m.author || ' ' || m.abstract || ' ' || d.
body AS document
FROM messages m, docs d
WHERE mid = did AND mid = 12;
```

> **注意**
>
> 实际上，这些示例查询中应使用 `coalesce`，以避免某个单独的 `NULL` 属性导致整个文档结果变成 `NULL`。

另一种做法是把文档存储为文件系统中的普通文本文件。在这种情况下，数据库可以用于保存全文索引并执行搜索，而文档本身则借助某个唯一标识符从文件系统中取回。不过，从数据库外部检索文件需要超级用户权限或特殊函数支持，因此这种方法通常不如把所有数据都保存在 PostgreSQL 内部方便。此外，把所有内容都放在数据库中，也便于访问文档元数据来辅助索引和展示。

为了进行文本检索，每个文档都必须被化简为预处理后的 `tsvector` 格式。搜索和排名完全基于文档的 `tsvector` 表示来执行 – 只有当文档被选中并准备展示给用户时，才需要取回原始文本。因此，我们常常把 `tsvector` 直接称作文档，但它当然只是完整文档的一种紧凑表示。

12.1.2. 基本文本匹配

在PostgreSQL中，全文检索基于匹配操作符`@@`。如果一个`tsvector`（文档）匹配一个`tsquery`（查询），它就返回`true`。哪一种数据类型写在前面并不重要：

```
SELECT 'a fat cat sat on a mat and ate a fat rat'::tsvector @@ 'cat
 & rat'::tsquery;
 ?column?
----------
 t

SELECT 'fat & cow'::tsquery @@ 'a fat cat sat on a mat and ate a
 fat rat'::tsvector;
 ?column?
----------
 f
```

正如上例所示，`tsquery`并不只是原始文本，`tsvector`也不是。`tsquery`包含搜索词，这些搜索词必须已经是正规化后的词位，并且可以用 AND、OR 和 NOT 操作符把多个搜索词组合起来。（详情见第 8.11 节。）`to_tsquery`和`plainto_tsquery`有助于把用户输入的文本转换为合适的 `tsquery`，例如对文本中的词做正规化。类似地，`to_tsvector` 用于解析并正规化文档字符串。因此在实践中，文本检索匹配更像是这样：

```
SELECT to_tsvector('fat cats ate fat rats') @@ to_tsquery('fat &
 rat');
```

```
 ?column?
----------
 t
```

请注意，如果写成

```
SELECT 'fat cats ate fat rats'::tsvector @@ to_tsquery('fat & rat')
;
 ?column?
----------
 f
```

就不会匹配成功，因为这里不会对单词 `rats` 做正规化。`tsvector` 的元素是词位，被假定为已经正规化，因此 `rats` 不会匹配 `rat`。

`@@`操作符也支持`text`输入，因此在简单场景下可以跳过把文本字符串显式转换为`tsvector`或`tsquery`。可用的形式有：

```
tsvector @@ tsquery
tsquery  @@ tsvector
text @@ tsquery
text @@ text
```

前两种形式我们已经见过。`text @@ tsquery` 等价于 `to_tsvector(x) @@ y`。`text @@ text` 等价于 `to_tsvector(x) @@ plainto_tsquery(y)`。

12.1.3. 配置

前述的都是简单的文本检索示例。正如前面所提到的，
全文检索功能包括做更多事情的能力：跳过索引特定词（停用词）、
处理同义词并使用更高级的解析，例如不局限于按空白划分的解析。这个功能由
文本检索配置控制。PostgreSQL中有多种语言的预定义配置，
并且你可以很容易地创建你自己的配置（psql的`\dF`命令显示所有可用的配置）。

在安装期间会选择一个合适的配置，并据此在`postgresql.conf`中设置default_text_search_config。如果整个集簇都使用同一种文本检索配置，你可以直接使用`postgresql.conf`中的这个值。若要在整个集簇中使用不同配置，
但保证每个数据库内部使用同一种配置，可以使用`ALTER DATABASE ... SET`。否则，你也可以在每个会话中设置`default_text_search_config`。

依赖一个配置的每一个文本检索函数都有一个可选的`regconfig`参数，
因此要使用的配置可以被显式指定。只有当省略这个参数时，`default_text_search_config`才被使用。

为了便于构建自定义文本检索配置，配置由更简单的数据库对象组合而成。PostgreSQL 的文本检索功能提供了四类与配置相关的数据库对象：

- 文本检索解析器将文档拆分成词元并对每个词元分类（例如，作为词或者数字）。
- 文本检索词典将词元转变成正规化的形式并拒绝停用词。
- 文本检索模板提供位于词典底层的函数（词典只需指定一个模板及其一组参数）。
- 文本检索配置选择一个解析器和一组用于将解析器产生的词元正规化的词典。

文本检索解析器和模板由底层 C 函数构成，因此开发新的解析器和模板需要 C 编程能力，将它们安装到数据库中则需要超级用户权限。（PostgreSQL 发行版的 `contrib/` 目录中有附加解析器和模板的示例。）词典和配置只是为底层解析器和模板设置参数并将它们连接起来，因此创建新词典或配置不需要特殊权限。
本章后面会给出创建自定义词典和配置的示例。

12.2. 表和索引

在前一节中的示例演示了使用简单常量字符串进行全文匹配。本节展示如何搜索表数据，以及可选择地使用索引。

12.2.1. 搜索表

即使没有索引，也可以执行全文检索。下面这个简单查询会输出每一行的title，其中对应的body字段包含单词friend：

```
SELECT title
FROM pgweb
WHERE to_tsvector('english', body) @@ to_tsquery('english',
 'friend');
```

这还会找到相关词，例如friends和friendly，因为这些词都会被约简为同一个正规化词位。

上述查询指定使用 english 配置来解析并正规化字符串。我们也可以省略配置参数：

```
SELECT title
FROM pgweb
WHERE to_tsvector(body) @@ to_tsquery('friend');
```

该查询将使用由default_text_search_config设置的配置。

更复杂一点的例子，是选出最近的 10 个文档，它们的 title 和 body 合在一起包含 create 和 table：

```
SELECT title
FROM pgweb
WHERE to_tsvector(title || ' ' || body) @@ to_tsquery('create &
 table')
ORDER BY last_mod_date DESC
LIMIT 10;
```

为简洁起见，这里省略了 coalesce 调用；如果希望在这两个字段之一为 NULL 时，另一字段仍能参与搜索，就需要加上它。

虽然这些查询在没有索引的情况下也能工作，但除偶尔的临时搜索外，大多数应用都会觉得这种方式太慢。文本检索在实际使用中通常都需要建立索引。

12.2.2. 创建索引

我们可以创建一个GIN索引（第 12.9 节）来加速文本检索：

```
CREATE INDEX pgweb_idx ON pgweb USING gin(to_tsvector('english',
 body));
```

注意这里使用的是to_tsvector的双参数版本。只有显式指定配置名称的文本检索函数，才能用于表达式索引（第 11.7 节）。这是因为索引内容必须不受default_text_search_config的影响。否则，索引内容就可能不一致，因为不同的索引项可能包含使用不同文本检索配置创建的tsvector，而且无法判断各自使用了哪一种配置。这样的索引也不可能被正确地转储和恢复。

由于上面的索引使用了 to_tsvector 的双参数版本，因此只有同样使用相同配置名的双参数版 to_tsvector 查询，才能使用该索引。也就是说，WHERE to_tsvector('english', body) @@ 'a & b' 可以使用该索引，而 WHERE to_tsvector(body) @@ 'a & b' 则不能。这样可以保证索引只会和创建索引项时所用的同一配置配合使用。

还可以建立更复杂的表达式索引，其中配置名由另一个列指定，例如：

```
CREATE INDEX pgweb_idx ON pgweb USING gin(to_tsvector(config_name,
 body));
```

这里config_name是pgweb表中的一个列。这样就允许在同一个索引中混合使用不同配置，同时记录每个索引项使用的是哪一种配置。例如，如果文档集合中包含不同语言的文档，这就会很有用。同样，打算使用该索引的查询也必须写成对应的形式，例如WHERE to_tsvector(config_name, body) @@ 'a & b'。

索引甚至可以串接多个列：

```
CREATE INDEX pgweb_idx ON pgweb USING gin(to_tsvector('english',
 title || ' ' || body));
```

另一种方法是创建一个单独的tsvector列来保存to_tsvector的输出。下面的示例把title和body串接起来，并用coalesce保证一个字段仍然可以被建立索引，即使另一个字段为NULL：

```
ALTER TABLE pgweb ADD COLUMN textsearchable_index_col tsvector;
UPDATE pgweb SET textsearchable_index_col =
     to_tsvector('english', coalesce(title,'') || ' ' || coalesce(
body,''));
```

然后我们创建一个GIN索引来加速搜索：

```
CREATE INDEX textsearch_idx ON pgweb USING gin(textsearchable_
index_col);
```

现在可以执行快速全文检索了：

```
SELECT title
FROM pgweb
WHERE textsearchable_index_col @@ to_tsquery('create & table')
ORDER BY last_mod_date DESC
LIMIT 10;
```

当使用一个单独的列来存储 tsvector 表示时，需要创建一个触发器来使 tsvector 列保持最新，以应对 title 或 body 的任何更改。第 12.4.3 节说明了如何做到这一点。

与表达式索引相比，单独列方法的一个优点是，为了利用索引，查询中不必显式指定文本检索配置。正如上面的例子所示，查询可以依赖default_text_search_config。另一个优点是搜索会更快，因为它不必重新执行to_tsvector调用来验证索引匹配（使用 GiST 索引时这一点比使用 GIN 索引时更重要；见第 12.9 节）。不过，表达式索引方法更容易设置，而且占用更少磁盘空间，因为tsvector表示并没有被显式存储。

12.3. 控制文本搜索

要实现全文检索，必须有函数能够从文档创建 tsvector，并从用户查询创建 tsquery。此外，我们还希望结果能按有意义的顺序返回，因此还需要函数根据文档与查询的相关性进行比较。同样重要的，是把结果良好地展示出来。PostgreSQL为这些能力都提供了支持。

12.3.1. 解析文档

PostgreSQL 提供了 to_tsvector 函数，用于把文档转换成 tsvector 数据类型。

```
to_tsvector([ config regconfig, ] document text) returns tsvector
```

to_tsvector 会把文本文档解析为词元，将词元归约为词位，并返回一个 tsvector，其中列出各词位及其在文档中的位置。文档会按指定的或默认的文本检索配置进行处理。
下面是一个简单示例：

```
SELECT to_tsvector('english', 'a fat  cat sat on a mat - it ate a
 fat rats');
                   to_tsvector
-----------------------------------------------------
 'ate':9 'cat':3 'fat':2,11 'mat':7 'rat':12 'sat':4
```

在上面的示例中可以看到，结果 tsvector 不包含 a、on 和 it，单词 rats 变成了 rat，标点符号 - 也被忽略了。

to_tsvector 在内部会调用解析器，把文档文本拆分成词元并为每个词元分配类型。对于每个词元，系统都会查询一个词典列表（第 12.6 节），而这个列表会随词元类型而变化。第一个能够识别该词元的词典，会输出一个或多个正规化后的词位来表示它。例如，rats 之所以变成 rat，是因为某个词典识别出 rats 是 rat 的复数形式。某些词会被识别为停用词（第 12.6.1 节），于是被忽略，因为它们出现得过于频繁，对搜索没有帮助。在这个示例中，a、on 和 it 就是停用词。如果列表中的词典都无法识别某个词元，它也会被忽略。示例里的标点符号 - 就属于这种情况，因为其词元类型（Space symbols）实际上没有分配任何词典，也就是说，空白类词元永远不会被索引。解析器、词典以及需要索引哪些词元类型，都由所选的文本检索配置（第 12.7 节）决定。一个数据库里可以同时存在多种不同配置，并且系统已经为多种语言提供了预定义配置。在本例中，我们使用的是英语的默认配置 english。

setweight 函数可用于给 tsvector 中的项打上指定的权重标签，权重可以是四个字母之一：A、B、C 或 D。这通常用于标记来自文档不同部分的项，例如标题和正文。稍后，这些信息可以用于搜索结果排名。

由于 to_tsvector(NULL) 会返回 NULL，因此只要字段可能为 NULL，就建议使用 coalesce。下面是从结构化文档创建 tsvector 的推荐方法：

```
UPDATE tt SET ti =
    setweight(to_tsvector(coalesce(title,'')), 'A')    ||
    setweight(to_tsvector(coalesce(keyword,'')), 'B')  ||
    setweight(to_tsvector(coalesce(abstract,'')), 'C') ||
    setweight(to_tsvector(coalesce(body,'')), 'D');
```

这里我们使用 setweight 为最终 tsvector 中每个词位标注来源，然后再用 tsvector 连接操作符 || 把这些已标注的 tsvector 值合并起来（关于这些操作的细节，见第 12.4.1 节）。

12.3.2. 解析查询

PostgreSQL 提供了 to_tsquery和 plainto_tsquery，用于把查询转换成 tsquery 数据类型。to_tsquery 提供的特性比 plainto_tsquery 更丰富，但对输入也更严格。

```
to_tsquery([ config regconfig, ] querytext text) returns tsquery
```

to_tsquery 根据 querytext 创建一个 tsquery 值。输入必须由单个词元组成，这些词元之间使用布尔操作符 &（AND）、|（OR）和 !（NOT）分隔。这些操作符可使用圆括号分组。换句话说，传给 to_tsquery 的输入，必须已经遵循第 8.11 节中描述的 tsquery 一般输入规则。不同之处在于，基础 tsquery 输入是按字面接受词元，而 to_tsquery 会使用指定或默认配置，把每个词元正规化为词位，并丢弃那些按该配置被判定为停用词的词元。例如：

```
SELECT to_tsquery('english', 'The & Fat & Rats');
```

```
  to_tsquery
---------------
 'fat' & 'rat'
```

与基础tsquery输入一样，可以为每个词位附加权重，以限制它只匹配具有这些权重的tsvector词位。例如：

```
SELECT to_tsquery('english', 'Fat | Rats:AB');
    to_tsquery
------------------
 'fat' | 'rat':AB
```

此外，还可以在词位后附加*来指定前缀匹配：

```
SELECT to_tsquery('supern:*A & star:A*B');
        to_tsquery
--------------------------
 'supern':*A & 'star':*AB
```

这样的词位将匹配tsvector中任何以给定字符串开头的单词。

to_tsquery也可以接受单引号括起来的短语。
当配置中包含可能在这类短语上触发的分类词典时，这一点尤其有用。在下面的例子中，一个分类词典包含规则 supernovae stars : sn：

```
SELECT to_tsquery('''supernovae stars'' & !crab');
  to_tsquery
---------------
 'sn' & !'crab'
```

如果不加引号，对于那些未被 AND 或 OR 操作符分隔的词元，to_tsquery会产生语法错误。

```
plainto_tsquery([ config regconfig, ] querytext text)
 returns tsquery
```

plainto_tsquery把未格式化的文本*querytext*转换成tsquery。该文本会像to_tsvector那样被解析并正规化，然后在保留下来的词之间插入&（AND）布尔操作符。

示例：

```
SELECT plainto_tsquery('english', 'The Fat Rats');
 plainto_tsquery
-----------------
 'fat' & 'rat'
```

注意，plainto_tsquery不能识别其输入中的布尔操作符、权重标签或前缀匹配标签：

```
SELECT plainto_tsquery('english', 'The Fat & Rats:C');
   plainto_tsquery
---------------------
 'fat' & 'rat' & 'c'
```

这里，输入中的所有标点符号都被当作空白符丢弃。

12.3.3. 搜索结果排名

排名旨在衡量文档与特定查询的相关程度，以便在匹配很多时优先显示最相关的结果。PostgreSQL提供了两种预定义的排名函数，它们会综合考虑词法信息、

邻近关系和结构信息；也就是说，会考虑查询词在文档中出现的频率、
这些词彼此之间的距离，以及这些词所处文档部分的重要程度。不过，
相关性这一概念本身就比较模糊，而且高度依赖具体应用。
不同应用可能还需要额外信息参与排名，例如文档修改时间。内置排名函数仅仅是示例。
你可以编写自己的排名函数，或者把它们的结果与其他因素结合起来，以满足特定需求。

目前可用的两种排名函数是：

```
ts_rank([ weights float4[], ] vector tsvector,
 query tsquery [, normalization integer ]) returns float4
```

根据向量中匹配词位的频率对向量进行排名。

```
ts_rank_cd([ weights float4[], ] vector tsvector,
 query tsquery [, normalization integer ]) returns float4
```

该函数为给定的文档向量和查询计算覆盖密度排名。该方法见 Clarke、Cormack 和 Tudhope 于 1999 年发表于期刊 "Information Processing and Management" 的文章 "Relevance Ranking for One to Three Term Queries"。

该函数需要输入中包含位置信息。因此，它对“被剥离的”`tsvector`值不起作用 — 结果总是为零。

对这两个函数来说，可选的 *weights* 参数允许根据词实例的标注情况赋予它们不同权重。权重数组按如下顺序指定各类词的权重：

`{D-权重，C-权重，B-权重，A-权重}`

如果没有提供 *weights*，则使用如下默认值：

`{0.1, 0.2, 0.4, 1.0}`

通常，权重用于标记来自文档特定区域的词，例如标题或开头摘要中的词，
从而使它们相较于正文中的词被赋予更高或更低的重要性。

由于较长的文档更有机会包含查询词，因此把文档大小纳入考量是合理的。例如，
一个一百词的文档里某个搜索词出现五次，
通常会比一个一千词的文档里同一搜索词也只出现五次更相关。
两种排名函数都接受一个整数 *normalization* 选项，用于指定文档长度是否影响排名，以及具体如何影响。该整数选项控制多种行为，因此它是一个位掩码：你可以使用 `|` 指定一种或多种行为（例如 `2|4`）。

- 0（默认值）忽略文档长度
- 1 用 1 + 文档长度的对数除排名
- 2 用文档长度除排名
- 4 用匹配范围之间的平均调和距离除排名（仅由 `ts_rank_cd` 实现）
- 8 用文档中唯一词的数量除排名
- 16 用 1 + 文档中不同词数量的对数除排名
- 32 用排名 + 1 除排名

如果指定了多个标志位，这些变换将按上面列出的顺序依次应用。

需要注意的是，排名函数不会使用任何全局信息，因此不可能像有时人们希望的那样，
给出一种能够公平归一化到 1% 或 100% 的结果。归一化选项 32（`rank/(rank+1)`）可以把所有排名缩放到 0 到 1 的范围内，但这当然只是表面上的变化，不会影响搜索结果的顺序。

下面是只选择排名最高的十个匹配的示例：

```
SELECT title, ts_rank_cd(textsearch, query) AS rank
FROM apod, to_tsquery('neutrino|(dark & matter)') query
```

```
WHERE query @@ textsearch
ORDER BY rank DESC
LIMIT 10;
                     title                     |   rank
-----------------------------------------------+----------
 Neutrinos in the Sun                          |      3.1
 The Sudbury Neutrino Detector                 |      2.4
 A MACHO View of Galactic Dark Matter          |  2.01317
 Hot Gas and Dark Matter                       |  1.91171
 The Virgo Cluster: Hot Plasma and Dark Matter |  1.90953
 Rafting for Solar Neutrinos                   |      1.9
 NGC 4650A: Strange Galaxy and Dark Matter     |  1.85774
 Hot Gas and Dark Matter                       |   1.6123
 Ice Fishing for Cosmic Neutrinos              |      1.6
 Weak Lensing Distorts the Universe            | 0.818218
```

下面是使用归一化排名的同一示例：

```
SELECT title, ts_rank_cd(textsearch, query, 32 /* rank/(rank+1) */
 ) AS rank
FROM apod, to_tsquery('neutrino|(dark & matter)') query
WHERE  query @@ textsearch
ORDER BY rank DESC
LIMIT 10;
                     title                     |        rank
-----------------------------------------------+--------------------
 Neutrinos in the Sun                          | 0.756097569485493
 The Sudbury Neutrino Detector                 | 0.705882361190954
 A MACHO View of Galactic Dark Matter          | 0.668123210574724
 Hot Gas and Dark Matter                       |  0.65655958650282
 The Virgo Cluster: Hot Plasma and Dark Matter | 0.656301290640973
 Rafting for Solar Neutrinos                   | 0.655172410958162
 NGC 4650A: Strange Galaxy and Dark Matter     | 0.650072921219637
 Hot Gas and Dark Matter                       | 0.617195790024749
 Ice Fishing for Cosmic Neutrinos              | 0.615384618911517
 Weak Lensing Distorts the Universe            | 0.450010798361481
```

排名计算可能非常昂贵，因为它需要访问每个匹配文档的 `tsvector`，这往往会受 I/O 限制而变慢。不幸的是，这几乎无法避免，因为实际查询常常会产生大量匹配。

12.3.4. 高亮结果

在呈现搜索结果时，理想的方式是展示每个文档的一段摘录，并说明它与查询的关系。通常，搜索引擎会在文档片段中标出搜索词。PostgreSQL 提供了 `ts_headline` 函数来实现这一功能。

```
ts_headline([ config regconfig, ] document text, query tsquery [,
 options text ]) returns text
```

`ts_headline` 接收文档和查询，并返回文档中一段高亮查询词的摘录。用于解析文档的配置可通过 *`config`* 指定；若省略 *`config`*，则使用 `default_text_search_config` 配置。

如果指定了*`options`*字符串，它必须由一个或多个用逗号分隔的*`option=value`*对组成。可用选项有：

- `StartSel`、`StopSel`：用于界定文档中查询词的字符串，以便与摘录中的其他词区分。如果这些字符串包含空格或逗号，则必须用双引号括起来。
- `MaxWords`、`MinWords`：这两个数值决定输出摘要的最长和最短长度。

- ShortWord：长度不超过该值的词就会从摘要的开头和结尾处被丢弃。默认值 3 会去掉常见的英语冠词。
- HighlightAll：布尔标志；如果为 true，则整个文档都会被用作摘要，而忽略前面三个参数。
- MaxFragments：要显示的最大文本摘录或片段数。默认值 0 选择一种非面向片段的摘要生成方法。大于 0 的值选择基于片段的摘要生成方法。这种方法会寻找包含尽可能多查询词的文本片段，并把这些片段围绕查询词扩展。这样，查询词就接近每个片段的中间，两侧都有词语。每个片段至多包含 MaxWords 个词，并且长度不超过 ShortWord 的词会在每个片段的开头和结尾处被丢弃。如果并非所有查询词都在文档中找到，则显示由文档前 MinWords 个词组成的单个片段。
- FragmentDelimiter：当显示多个片段时，片段之间将以该字符串分隔。

未指定的选项将采用以下默认值：

```
StartSel=<b>, StopSel=</b>,
MaxWords=35, MinWords=15, ShortWord=3, HighlightAll=FALSE,
MaxFragments=0, FragmentDelimiter=" ... "
```

例如：

```
SELECT ts_headline('english',
  'The most common type of search
is to find all documents containing given query terms
and return them in order of their similarity to the
query.',
  to_tsquery('query & similarity'));
                        ts_headline
------------------------------------------------------------
 containing given <b>query</b> terms
 and return them in order of their <b>similarity</b> to the
 <b>query</b>.

SELECT ts_headline('english',
  'The most common type of search
is to find all documents containing given query terms
and return them in order of their similarity to the
query.',
  to_tsquery('query & similarity'),
  'StartSel = <, StopSel = >');
                       ts_headline
-------------------------------------------------------
 containing given <query> terms
 and return them in order of their <similarity> to the
 <query>.
```

ts_headline 使用的是原始文档，而不是 tsvector 摘要，因此它可能较慢，使用时应当谨慎。一个典型的错误是，当只需要显示十个文档时，却对每一个匹配文档都调用 ts_headline。SQL 子查询可以提供帮助，下面是一个例子：

```
SELECT id, ts_headline(body, q), rank
FROM (SELECT id, body, q, ts_rank_cd(ti, q) AS rank
      FROM apod, to_tsquery('stars') q
      WHERE ti @@ q
      ORDER BY rank DESC
      LIMIT 10) AS foo;
```

12.4. 附加特性

本节介绍一些在文本检索中很有用的附加函数和操作符。

12.4.1. 操纵文档

第 12.3.1 节展示了原始文本文档如何被转换成 `tsvector` 值。PostgreSQL 也提供了用于操纵已经处于 `tsvector` 形式文档的函数和操作符。

```
tsvector || tsvector
```

`tsvector` 连接操作符返回一个向量，它结合了两个参数向量中的词位和位置信息。位置和权重标签在连接过程中会被保留。
右侧向量中的位置会按左侧向量中出现的最大位置做偏移，
因此结果几乎等同于对两个原始文档字符串连接后的结果执行 `to_tsvector`。（这种等价并不完全成立，因为从左侧参数末尾移除的停用词不会影响结果；
而如果直接做文本连接，它们本来会影响右侧参数中词位的位置。）

相比先连接文本再应用 `to_tsvector`，直接连接向量的一个优点是，你可以用不同配置来解析文档的不同部分。此外，由于 `setweight` 会以相同方式标记给定向量中的全部词位，如果希望给文档不同部分打上不同权重，就必须先解析文本并调用 `setweight`，然后再做连接。

```
setweight(vector tsvector, weight "char") returns tsvector
```

`setweight` 返回输入向量的一个副本，其中每个位置都被标注为给定的 *`weight`*：`A`、`B`、`C` 或 `D`（`D` 是新向量的默认值，因此不会在输出中显示）。这些标签在向量连接时会被保留，从而允许排名函数对来自文档不同部分的词赋予不同权重。

注意权重标签是应用到位置而不是词位。如果输入向量已经被剥离了位置，则 `setweight`什么也不会做。

```
length(vector tsvector) returns integer
```

返回存储在向量中的词位数。

```
strip(vector tsvector) returns tsvector
```

返回一个向量，其中列出与给定向量相同的词位，但不包含任何位置或权重信息。
虽然返回的向量在相关度排名方面远不如未剥离的向量有用，但通常会比它小得多。

12.4.2. 操纵查询

第 12.3.2 节展示了原始文本查询如何被转换成 `tsquery` 值。PostgreSQL 也提供了用于操纵已经处于 `tsquery` 形式查询的函数和操作符。

```
tsquery && tsquery
```

返回用 AND 结合的两个给定查询。

```
tsquery || tsquery
```

返回用 OR 结合的两个给定查询。

```
!! tsquery
```

返回给定查询的否定（NOT）。

```
numnode(query tsquery) returns integer
```

返回`tsquery`中的节点数（词位加操作符）。这个函数可用于判断*`query`*
是否有意义（返回值 > 0），或者是否只包含停用词（返回 0）。例如：

```
SELECT numnode(plainto_tsquery('the any'));
```

```
NOTICE:  query contains only stopword(s) or doesn't contain
 lexeme(s), ignored
 numnode
---------
       0

SELECT numnode('foo & bar'::tsquery);
 numnode
---------
       3
```

querytree(*query* tsquery) returns text

返回一个tsquery中可用于搜索索引的部分。此函数可用于检测无法使用索引的查询，例如只包含停用词或只包含否定词项的查询。例如：

```
SELECT querytree(to_tsquery('!defined'));
 querytree
-----------
```

12.4.2.1. 查询重写

ts_rewrite 函数族会在给定的 tsquery 中搜索目标子查询的出现，并把每一次出现都替换为替换子查询。本质上，这就是面向 tsquery 的一种子串替换。目标与替换的组合可以看作一条查询重写规则。一组这样的重写规则可以成为非常强大的搜索辅助工具。例如，你可以利用同义词扩展搜索（如 new york、big apple、nyc、gotham），或者收窄搜索，把用户引向某个热门主题。该特性与分类词典（第 12.6.4 节）在功能上有一定重叠。不过，重写规则可以随时修改而无需重建索引，而更新分类词典则必须重建索引后才能生效。

ts_rewrite (*query* tsquery, *target* tsquery, *substitute* tsquery) returns tsquery

这种形式的 ts_rewrite 只应用一条重写规则：无论 *target* 在 *query* 中的何处出现，都用 *substitute* 替换它。例如：

```
SELECT ts_rewrite('a & b'::tsquery, 'a'::tsquery, 'c'::tsquery);
 ts_rewrite
------------
 'b' & 'c'
```

ts_rewrite (*query* tsquery, *select* text) returns tsquery

这种形式的 ts_rewrite 接受一个起始 *query* 和一个以文本字符串给出的 SQL *select* 命令。该 *select* 必须产生两列 tsquery 类型的值。对于 *select* 结果中的每一行，第一列值（目标）在当前 *query* 值中的所有出现都会被第二列值（替换）取代。例如：

```
CREATE TABLE aliases (t tsquery PRIMARY KEY, s tsquery);
INSERT INTO aliases VALUES('a', 'c');

SELECT ts_rewrite('a & b'::tsquery, 'SELECT t,s FROM aliases');
 ts_rewrite
------------
 'b' & 'c'
```

注意，当以这种方式应用多个重写规则时，应用顺序可能很重要；因此在实际使用中，你通常会要求源查询按某个排序键 ORDER BY。

我们来看一个现实中的天文示例。我们将使用表驱动的重写规则来扩展查询 supernovae：

```
CREATE TABLE aliases (t tsquery primary key, s tsquery);
```

```
INSERT INTO aliases VALUES(to_tsquery('supernovae'), to_tsquery(
'supernovae|sn'));

SELECT ts_rewrite(to_tsquery('supernovae & crab'), 'SELECT * FROM
 aliases');
           ts_rewrite
---------------------------------
 'crab' & ( 'supernova' | 'sn' )
```

我们只需更新表，就可以修改这些重写规则：

```
UPDATE aliases
SET s = to_tsquery('supernovae|sn & !nebulae')
WHERE t = to_tsquery('supernovae');

SELECT ts_rewrite(to_tsquery('supernovae & crab'), 'SELECT * FROM
 aliases');
                 ts_rewrite
---------------------------------------------
 'crab' & ( 'supernova' | 'sn' & !'nebula' )
```

当重写规则很多时，重写过程可能会很慢，因为它需要检查每一条规则是否可能匹配。为了过滤掉显然不可能命中的规则，可以使用 `tsquery` 类型的包含操作符。在下面的例子中，我们只选择那些可能匹配原始查询的规则：

```
SELECT ts_rewrite('a & b'::tsquery,
                  'SELECT t,s FROM aliases WHERE ''a & b''::tsquery
 @> t');
 ts_rewrite
------------
 'b' & 'c'
```

12.4.3. 用于自动更新的触发器

当使用单独一列来存储文档的 `tsvector` 表示时，就需要创建触发器，以便在文档内容列发生变化时更新该 `tsvector` 列。系统为此提供了两个内置触发器函数，你也可以自行编写触发器函数。

```
tsvector_update_trigger(tsvector_column_name, config_name, text_
column_name [, ... ])
tsvector_update_trigger_column(tsvector_column_name, config_column_
name, text_column_name [, ... ])
```

这些触发器函数会在 `CREATE TRIGGER` 命令中给定参数的控制下，从一个或多个文本列自动计算出一个 `tsvector` 列。下面是它们的一个使用示例：

```
CREATE TABLE messages (
    title       text,
    body        text,
    tsv         tsvector
);

CREATE TRIGGER tsvectorupdate BEFORE INSERT OR UPDATE
ON messages FOR EACH ROW EXECUTE PROCEDURE
tsvector_update_trigger(tsv, 'pg_catalog.english', title, body);

INSERT INTO messages VALUES('title here', 'the body text is here');
```

```
SELECT * FROM messages;
   title    |         body          |            tsv
------------+-----------------------+----------------------------
 title here | the body text is here | 'bodi':4 'text':5 'titl':1

SELECT title, body FROM messages WHERE tsv @@ to_tsquery('title &
 body');
   title    |         body
------------+-----------------------
 title here | the body text is here
```

创建该触发器之后，对title或body的任何修改都会自动反映到tsv中，
而应用程序无需关心这一点。

第一个触发器参数必须是要更新的 tsvector 列名。第二个参数指定执行转换时要使用的文本检索配置。对于 tsvector_update_trigger，配置名直接作为第二个触发器参数给出。如上所示，它必须带模式限定，这样触发器行为就不会随着 search_path 的变化而变化。对于 tsvector_update_trigger_column，第二个触发器参数则是另一个表列的名称，该列必须是 regconfig 类型。这样就可以按行选择配置。其余参数是文本列的名称（类型为 text、varchar 或 char），它们会按给定顺序并入文档。NULL 值会被跳过（但其他列仍会被索引）。

这些内置触发器的一个限制是，它们会以相同方式处理所有输入列。若要对列做不同处理 — 例如让标题和正文具有不同权重 — 就需要编写自定义触发器。下面是一个使用 PL/pgSQL 作为触发器语言的示例：

```
CREATE FUNCTION messages_trigger() RETURNS trigger AS $$
begin
  new.tsv :=
     setweight(to_tsvector('pg_catalog.english', coalesce(new.
title,'')), 'A') ||
     setweight(to_tsvector('pg_catalog.english', coalesce(new.body,
'')), 'D');
  return new;
end
$$ LANGUAGE plpgsql;

CREATE TRIGGER tsvectorupdate BEFORE INSERT OR UPDATE
    ON messages FOR EACH ROW EXECUTE PROCEDURE messages_trigger();
```

请记住，在触发器中创建 tsvector 值时，明确指定配置名称非常重要，这样列的内容才不会受到 default_text_search_config 变化的影响。否则，很可能会导致问题，例如在转储并恢复之后搜索结果发生变化。

12.4.4. 收集文档统计数据

ts_stat 可用于检查配置，并寻找候选停用词。

```
ts_stat(sqlquery text, [ weights text, ]
        OUT word text, OUT ndoc integer,
        OUT nentry integer) returns setof record
```

sqlquery 是一个文本值，其中包含一条必须返回单一 tsvector 列的 SQL 查询。ts_stat 会执行该查询，并返回该 tsvector 数据中每个不同词位（单词）的统计信息。返回的列如下：

- *word* text — 一个词位的值
- *ndoc* integer — 词出现过的文档（tsvector）的数量

- *nentry* integer – 词出现的总次数

如果提供了 *weights*，则只统计具有这些权重之一的出现。

例如，要在一个文档集合中查找十个最频繁的词：

```
SELECT * FROM ts_stat('SELECT vector FROM apod')
ORDER BY nentry DESC, ndoc DESC, word
LIMIT 10;
```

同样的查询，但只统计权重为 A 或 B 的出现次数：

```
SELECT * FROM ts_stat('SELECT vector FROM apod', 'ab')
ORDER BY nentry DESC, ndoc DESC, word
LIMIT 10;
```

12.5. 解析器

文本检索解析器负责把未处理的文档文本划分成词元并标识每个词元的类型，而可能的类型集合由解析器本身定义。注意，解析器完全不会修改文本 – 它只是识别看似合理的词边界。由于作用范围有限，相比自定义词典，对应用相关的自定义解析器的需求没有那么强烈。目前PostgreSQL只提供一种内置解析器，而它已经被证明对广泛的应用都很有用。

内置解析器被称为pg_catalog.default。它识别 23 种词元类型，如表 12.1所示。

表 12.1. 默认解析器的词元类型

别名	描述	示例
asciiword	单词，所有 ASCII 字母	elephant
word	单词，所有字母	mañana
numword	单词，字母和数字	beta1
asciihword	带连字符的单词，所有 ASCII	up-to-date
hword	带连字符的单词，所有字母	lógico-matemática
numhword	带连字符的单词，字母和数字	postgresql-beta1
hword_asciipart	带连字符的单词部分，所有 ASCII	在 postgresql-beta1 上下文中的 postgresql
hword_part	带连字符的单词部分，所有字母	在 lógico-matemática 上下文中的 lógico 或 matemática
hword_numpart	带连字符的单词部分，字母和数字	在 postgresql-beta1 上下文中的 beta1
email	电子邮件地址	foo@example.com
protocol	协议头部	http://
url	URL	example.com/stuff/index.html
host	主机	example.com
url_path	URL 路径	/stuff/index.html，在 URL 上下文中
file	文件或路径名	/usr/local/foo.txt，如果不在 URL 中
sfloat	科学记数法	-1.234e56
float	十进制记数法	-1.234
int	有符号整数	-1234

别名	描述	示例
uint	无符号整数	1234
version	版本号	8.3.0
tag	XML 标签	<a href="dictionaries.html">
entity	XML 实体	&
blank	空格符号	（其他不识别的任意空白或标点符号）

> **注意**
>
> 解析器对“字母”的界定由数据库的区域设置决定，具体是lc_ctype。只包含基本 ASCII 字母的词被报告为一个单独的词元类型，因为有时区分它们会很有用。在大部分欧洲语言中，词元类型word和asciiword应该被同样对待。
>
> email 不支持 RFC 5322 定义的所有有效电子邮件字符。具体来说，电子邮件用户名中支持的非字母数字字符只有句点、短横线和下划线。

解析器有可能从同一段文本中产生重叠的词元。举例来说，
一个带连字符的词既会被报告为整个单词，也会被报告为各个组成部分：

```
SELECT alias, description, token FROM ts_debug('foo-bar-beta1');
      alias      |               description                |
 token
-----------------+------------------------------------------+------
---------
 numhword        | Hyphenated word, letters and digits      | foo-
bar-beta1
 hword_asciipart | Hyphenated word part, all ASCII          | foo
 blank           | Space symbols                            | -
 hword_asciipart | Hyphenated word part, all ASCII          | bar
 blank           | Space symbols                            | -
 hword_numpart   | Hyphenated word part, letters and digits | beta1
```

这种行为是可取的，因为它既允许针对整个复合词搜索，也允许针对各个组成部分搜索。
下面是另一个有启发性的示例：

```
SELECT alias, description, token FROM ts_debug('http://example.com/
stuff/index.html');
  alias   |  description  |            token
----------+---------------+------------------------------
 protocol | Protocol head | http://
 url      | URL           | example.com/stuff/index.html
 host     | Host          | example.com
 url_path | URL path      | /stuff/index.html
```

12.6. 词典

词典用于消除不应参与搜索的词（停用词），并用于对词进行正规化，
以便同一个词的不同派生形式能够匹配。成功完成正规化的词被称为词位。
除了改善搜索质量，正规化和移除停用词还会减小文档的tsvector表示，从而提高性能。
正规化并不总是具有语言学意义，而且通常依赖于应用的语义。

一些正规化的示例：

- 语言学上的例子 – Ispell 词典尝试把输入词约简为一种正规化形式；词干分析器词典则去掉词尾
- URL 地址可以规范化，以便让等价的 URL 匹配：
  - http://www.pgsql.ru/db/mw/index.html
  - http://www.pgsql.ru/db/mw/
  - http://www.pgsql.ru/db/../db/mw/index.html
- 颜色名可以被它们的十六进制值替换，例如`red`，`green`，`blue`，`magenta -> FF0000`，`00FF00`，`0000FF`，`FF00FF`
- 如果要索引数字，我们可以去掉某些小数位，以缩小可能数值的范围。因此，如果只保留小数点后两位，那么例如 3.14159265359、3.1415926 和 3.14 在正规化后就会变成相同的值。

一个词典是一个程序，它接受一个词元作为输入，并返回：

- 如果输入的词元对词典是已知的，则返回一个词位数组（注意一个词元可能产生多于一个词位）
- 一个带有 `TSL_FILTER` 标志的单个词位，它会用一个新词元替换原始词元，并把它传递给后续词典（执行这种工作的词典称为过滤词典）
- 如果词典知道该词元但它是一个停用词，则返回一个空数组
- 如果词典不识别该输入词元，则返回`NULL`

PostgreSQL为许多语言提供了预定义的词典。
也有多种预定义模板可以被用于创建带自定义参数的新词典。
每一种预定义词典模板在下面描述。如果没有合适的现有模板，可以创建新的；示例见PostgreSQL发行版的`contrib/`目录。

文本检索配置把一个解析器与一组用于处理解析器输出词元的词典绑定在一起。
对于解析器可能返回的每一种词元类型，配置都会指定一个单独的词典列表。
当解析器找到该类型的词元时，会按顺序依次查询列表中的每个词典，
直到有某个词典把它识别为已知词。如果它被识别为停用词，或者没有任何词典识别它，
那么该词元就会被丢弃，既不会建立索引，也不会参与搜索。通常，第一个返回非 `NULL` 输出的词典就决定结果，后续词典不会再被查询；但过滤词典可以把给定单词替换为一个修改后的单词，再传递给后续词典。

配置词典列表的一般规则是，把适用范围最窄、最专门的词典放在最前面，
然后是更通用的词典，最后以一个非常通用的词典收尾，例如 Snowball 词干分析器，或能识别所有内容的 `simple`。例如，对于一个天文学相关搜索（配置名为 `astro_en`），可以把词元类型 `asciiword`（ASCII 词）绑定到一个天文学术语同义词词典、一个通用英语词典，以及一个 Snowball 英语词干分析器：

```
ALTER TEXT SEARCH CONFIGURATION astro_en
    ADD MAPPING FOR asciiword WITH astrosyn, english_ispell,
 english_stem;
```

过滤词典可以放在列表中的任何位置，但放在末尾没有用处。
过滤词典可用于先对词做部分正规化，以简化后续词典的工作。例如，
可以用过滤词典去掉带重音字母中的重音符号，就像unaccent模块所做的那样。

12.6.1. 停用词

停用词是非常常见、几乎出现在每个文档中的词，没有区分价值。因此，
在全文检索中可以忽略它们。例如，每篇英文文本都包含像 `a` 和 `the` 这样的词，所以把它们存储在索引中没有用处。不过，停用词确实会影响 `tsvector` 中的位置，而这又会影响排名：

```
SELECT to_tsvector('english','in the list of stop words');
        to_tsvector
----------------------------
```

```
 'list':3 'stop':5 'word':6
```

缺失的位置 1、2、4 就是由停用词造成的。对包含停用词和不包含停用词的文档，计算出来的排名会明显不同：

```
SELECT ts_rank_cd (to_tsvector('english','in the list of stop
 words'), to_tsquery('list & stop'));
 ts_rank_cd
------------
       0.05

SELECT ts_rank_cd (to_tsvector('english','list stop words'), to_
tsquery('list & stop'));
 ts_rank_cd
------------
        0.1
```

如何处理停用词取决于具体词典。例如，ispell词典会先正规化单词，再查看停用词列表，而Snowball词干分析器则先检查停用词列表。这种不同行为是为了尽量减少噪声。

12.6.2. 简单词典

simple 词典模板的工作方式是先把输入词元转换为小写，再根据停用词文件检查它。如果在文件中找到该词元，就返回一个空数组，从而丢弃该词元；否则，
返回其小写形式作为正规化后的词位。或者，
也可以把该词典配置为把非停用词报告为未识别，
从而允许它们传递给列表中的下一个词典。

下面是一个使用 simple 模板的词典定义示例：

```
CREATE TEXT SEARCH DICTIONARY public.simple_dict (
    TEMPLATE = pg_catalog.simple,
    STOPWORDS = english
);
```

其中，english 是停用词文件的基名。文件完整名称将是 $SHAREDIR/tsearch_data/english.stop，其中 $SHAREDIR 表示 PostgreSQL 安装的共享数据目录，通常是 /usr/local/share/postgresql（如果不确定，可用 pg_config --sharedir 来确定）。文件格式只是一个单词列表，每行一个单词。空行和行尾空格会被忽略，大写会折叠为小写，但不会对文件内容做其他处理。

现在我们可以测试这个词典：

```
SELECT ts_lexize('public.simple_dict','YeS');
 ts_lexize
-----------
 {yes}

SELECT ts_lexize('public.simple_dict','The');
 ts_lexize
-----------
 {}
```

如果在停用词文件中找不到该词，我们也可以选择返回 NULL，而不是返回它的小写形式。这种行为可以通过把词典的 Accept 参数设置为 false 来启用。继续上面的例子：

```
ALTER TEXT SEARCH DICTIONARY public.simple_dict ( Accept = false );

SELECT ts_lexize('public.simple_dict','YeS');
 ts_lexize
```

```
-----------


SELECT ts_lexize('public.simple_dict','The');
 ts_lexize
-----------
 {}
```

在默认设置 `Accept = true` 下，只有把 `simple` 词典放在词典列表末尾才有意义，因为它不会把任何词元传递给后续词典。相反，`Accept = false` 只有在后面至少还有一个词典时才有用。

> **小心**
>
> 大多数词典类型都依赖配置文件，例如停用词文件。这些文件必须采用 UTF-8 编码保存。当它们被读入服务器时，如果数据库编码不同，
> 就会被转换为实际的数据库编码。

> **小心**
>
> 通常，词典配置文件在某个数据库会话中第一次被使用时，只会被读取一次。
> 如果你修改了配置文件，并希望强制现有会话加载新内容，可以对该词典执行一条 `ALTER TEXT SEARCH DICTIONARY` 命令。这可以是一条“假”更新，也就是并不实际修改任何参数值的更新。

12.6.3. 同义词词典

这个词典模板用于创建把一个单词替换为同义词的词典。
不支持短语（对此请使用分类词典模板第 12.6.4 节）。同义词词典可以用来克服语言学上的问题，例如防止英语词干分析词典把“Paris”约简成“pari”。只要在同义词词典中加入一行 `Paris paris`，并把该词典放在 `english_stem` 词典之前即可。例如：

```
SELECT * FROM ts_debug('english', 'Paris');
   alias   |   description   | token |  dictionaries  |  dictionary
  | lexemes
-----------+-----------------+-------+----------------+------------
--+---------
 asciiword | Word, all ASCII | Paris | {english_stem} | english_
stem | {pari}

CREATE TEXT SEARCH DICTIONARY my_synonym (
    TEMPLATE = synonym,
    SYNONYMS = my_synonyms
);

ALTER TEXT SEARCH CONFIGURATION english
    ALTER MAPPING FOR asciiword
    WITH my_synonym, english_stem;

SELECT * FROM ts_debug('english', 'Paris');
   alias   |   description   | token |       dictionaries        |
 dictionary | lexemes
-----------+-----------------+-------+---------------------------+-
-----------+---------
```

```
 asciiword | Word, all ASCII | Paris | {my_synonym,english_stem} |
 my_synonym | {paris}
```

synonym模板要求的唯一参数是SYNONYMS，它是其配置文件的基名 – 上例中的my_synonyms。该文件的完整名称将是$SHAREDIR/tsearch_data/my_synonyms.syn（其中$SHAREDIR表示PostgreSQL安装的共享数据目录）。
该文件格式是每行一个要被替换的词，后面跟着它的同义词，用空白分隔。
空行和结尾的空格会被忽略。

synonym模板还有一个可选参数CaseSensitive，默认值为false。当CaseSensitive为false时，同义词文件中的词会像输入词元一样折叠为小写。当它为true时，
词和词元都不会被折叠为小写，而是按原样比较。

可以在配置文件中把星号（*）放在同义词末尾，表示该同义词是一个前缀。当该条目用于 to_tsvector() 时，星号会被忽略；但当它用于 to_tsquery() 时，结果将是一个带有前缀匹配标记的查询项（见第 12.3.2 节）。例如，假设在 $SHAREDIR/tsearch_data/synonym_sample.syn 中有以下条目：

```
postgres        pgsql
postgresql      pgsql
postgre pgsql
gogle   googl
indices index*
```

那么将得到如下结果：

```
mydb=# CREATE TEXT SEARCH DICTIONARY syn (template=synonym,
 synonyms='synonym_sample');
mydb=# SELECT ts_lexize('syn','indices');
 ts_lexize
-----------
 {index}
(1 row)

mydb=# CREATE TEXT SEARCH CONFIGURATION tst (copy=simple);
mydb=# ALTER TEXT SEARCH CONFIGURATION tst ALTER MAPPING FOR
 asciiword WITH syn;
mydb=# SELECT to_tsvector('tst','indices');
 to_tsvector
-------------
 'index':1
(1 row)

mydb=# SELECT to_tsquery('tst','indices');
 to_tsquery
------------
 'index':*
(1 row)

mydb=# SELECT 'indexes are very useful'::tsvector;
            tsvector
---------------------------------
 'are' 'indexes' 'useful' 'very'
(1 row)

mydb=# SELECT 'indexes are very useful'::tsvector @@ to_tsquery(
'tst','indices');
 ?column?
----------
```

```
 t
(1 row)
```

12.6.4. 分类词典

一个分类词典（有时被简写成TZ）是一个词的集合，其中包括了词与短语之间的联系，
即广义词（BT）、狭义词（NT）、首选词、非首选词、相关词等。

分类词典基本上会用一个首选词替换所有非首选词，并且也可以选择保留原始词用于索引。
PostgreSQL的分类词典的当前实现是同义词词典的一个扩展，并增加了短语支持。
一个分类词典需要一个采用以下格式的配置文件：

```
# 这是一条注释
示例词（一个或多个） : 索引词（一个或多个）
更多示例词（一个或多个） : 更多索引词（一个或多个）
...
```

其中冒号（:）符号用作短语及其替换词之间的分隔符。

分类词典会使用一个子词典（在词典配置中指定）在检查短语匹配之前正规化输入文本。
只能选择一个子词典。如果子词典无法识别某个词，就会报错。在这种情况下，
你应当避免使用该词，或者让子词典学会它。你可以在某个被索引词的开头放置一个星号（*），以跳过将子词典应用于该词，但所有样例词都必须能被子词典识别。

如果有多个短语匹配输入，分类词典会选择最长的匹配；长度相同时使用最后的定义。

由子词典识别的特定停用词不能够被指定；改用?标记可出现任意停用词的位置。例如，
假定根据子词典a和the是停用词：

```
? one ? two : swsw
```

匹配a one the two和the one a two；两者都将被swsw替换。

由于分类词典具备识别短语的能力，因此它必须记住自身状态并与解析器交互。
分类词典会利用这些映射关系来判断自己是应当处理下一个词，还是停止累积。
分类词典必须经过仔细配置。例如，如果分类词典只被映射到 asciiword 词元，那么像 one 7 这样的分类词典定义就无法工作，因为词元类型 uint 并没有映射给该分类词典。

> **小心**
>
> 分类词典会在建立索引时使用，因此其参数发生任何变化都要求重新建立索引。
> 对于大多数其他词典类型，
> 像增加或移除停用词这样的细小改动则不会强制重新建立索引。

12.6.4.1. 分类词典配置

要定义一个新的分类词典，可使用thesaurus模板。例如：

```
CREATE TEXT SEARCH DICTIONARY thesaurus_simple (
    TEMPLATE = thesaurus,
    DictFile = mythesaurus,
    Dictionary = pg_catalog.english_stem
);
```

这里：

- thesaurus_simple是新词典的名称

- mythesaurus是分类词典配置文件的基名（它的全名将是$SHAREDIR/tsearch_data/mythesaurus.ths，其中$SHAREDIR表示安装的共享数据目录）。
- pg_catalog.english_stem是要用于分类词典正规化的子词典（这里是一个 Snowball 英语词干分析器）。注意子词典将拥有它自己的配置（例如停用词），但这里没有展示。

现在可以在配置中把分类词典thesaurus_simple绑定到想要的词元类型上，例如：

```
ALTER TEXT SEARCH CONFIGURATION russian
    ALTER MAPPING FOR asciiword, asciihword, hword_asciipart
    WITH thesaurus_simple;
```

12.6.4.2. 分类词典示例

考虑一个简单的天文分类词典 thesaurus_astro，其中包含一些天文单词组合：

```
supernovae stars : sn
crab nebulae : crab
```

下面我们创建一个词典，并把一些词元类型绑定到天文分类词典和英语词干分析器上：

```
CREATE TEXT SEARCH DICTIONARY thesaurus_astro (
    TEMPLATE = thesaurus,
    DictFile = thesaurus_astro,
    Dictionary = english_stem
);

ALTER TEXT SEARCH CONFIGURATION russian
    ALTER MAPPING FOR asciiword, asciihword, hword_asciipart
    WITH thesaurus_astro, english_stem;
```

现在我们可以看看它是如何工作的。ts_lexize 对测试分类词典并不十分有用，因为它把输入当作一个单独的词元。相反，我们可以使用 plainto_tsquery 和 to_tsvector，它们会把输入字符串拆分成多个词元：

```
SELECT plainto_tsquery('supernova star');
 plainto_tsquery
-----------------
 'sn'

SELECT to_tsvector('supernova star');
 to_tsvector
-------------
 'sn':1
```

原则上，如果把参数用引号括起来，也可以使用 to_tsquery：

```
SELECT to_tsquery('''supernova star''');
 to_tsquery
------------
 'sn'
```

请注意，supernova star 能匹配 thesaurus_astro 中的 supernovae stars，因为我们在分类词典定义中指定了 english_stem 词干分析器。它去掉了词尾的 e 和 s。

如果既要为替换词建立索引，也要为原始短语建立索引，
只需把原始短语也写到定义右侧即可：

```
supernovae stars : sn supernovae stars

SELECT plainto_tsquery('supernova star');
```

```
       plainto_tsquery
-----------------------------
 'sn' & 'supernova' & 'star'
```

12.6.5. Ispell 词典

Ispell词典模板支持形态词典，它可以把一个词的许多不同语言学形式正规化为同一个词位。例如，一个英语 Ispell 词典可以匹配搜索词 `bank` 的所有变格和变位形式，例如 `banking`、`banked`、`banks`、`banks'` 和 `bank's`。

标准PostgreSQL发行版不包含任何Ispell配置文件。大量语言的词典可从Ispell[1]获取。此外，还支持一些更现代的词典文件格式 – MySpell[2] (OO < 2.0.1) 和Hunspell[3](OO >= 2.0.2)。在OpenOffice Wiki[4]上有大量词典可用。

要创建一个Ispell词典，可使用内置的`ispell`模板并指定几个参数：

```
CREATE TEXT SEARCH DICTIONARY english_ispell (
    TEMPLATE = ispell,
    DictFile = english,
    AffFile = english,
    StopWords = english
);
```

这里，`DictFile`、`AffFile`和`StopWords`指定词典、词缀和停用词文件的基名。停用词文件的格式和前面解释的`simple`词典类型相同。其他文件的格式在这里没有指定，但是也可以从上面提到的网站获得。

Ispell 词典通常只能识别有限的词，因此其后应跟着另一个覆盖范围更广的词典；例如，一个 Snowball 词典，它可以识别所有输入。

Ispell 词典支持划分复合词，这是一个有用的特性。注意词缀文件应该用`compoundwords controlled`语句指定一个特殊标志，用于标记词典中可以参与构成复合词的词：

```
compoundwords  controlled z
```

下面是挪威语的一些示例：

```
SELECT ts_lexize('norwegian_ispell',
 'overbuljongterningpakkmesterassistent');
   {over,buljong,terning,pakk,mester,assistent}
SELECT ts_lexize('norwegian_ispell', 'sjokoladefabrikk');
   {sjokoladefabrikk,sjokolade,fabrikk}
```

> **注意**
>
> MySpell 不支持复合词。Hunspell则对复合词有更好的支持。当前，PostgreSQL只实现了 Hunspell 中基本的复合词操作。

12.6.6. Snowball 词典

Snowball词典模板基于 Martin Porter 的一个项目，他是流行的英语 Porter 词干分析算法的发明者。Snowball 现在对许多语言提供词干分析算法（详见Snowball 站点[5]）。每种算法都

[1] http://ficus-www.cs.ucla.edu/geoff/ispell.html
[2] http://en.wikipedia.org/wiki/MySpell
[3] http://sourceforge.net/projects/hunspell/
[4] http://wiki.services.openoffice.org/wiki/Dictionaries
[5] http://snowballstem.org/

知道如何把其语言中词的常见变体约简为基本拼写形式，也就是词干。一个 Snowball 词典要求一个language参数来标识要用哪种词干分析器，并且可以选择指定一个stopword文件名来给出一个要被消除的词列表（PostgreSQL的标准停用词列表也是由 Snowball 项目提供的）。例如，有一个内置的定义等效于

```
CREATE TEXT SEARCH DICTIONARY english_stem (
    TEMPLATE = snowball,
    Language = english,
    StopWords = english
);
```

停用词文件格式和已经解释的一样。

一个Snowball词典识别所有输入，不管它能不能简化该词，因此它应当被放置在词典列表的最后。把它放在任何其他词典前面是没有用处的，因为一个词元永远不会穿过它而进入到下一个词典。

12.7. 配置示例

一个文本检索配置指定了将一个文档转换成一个tsvector所需的所有选项：用于把文本分解成词元的解析器，以及用于将每一个词元转换成词位的词典。每一次to_tsvector或to_tsquery的调用都需要一个文本检索配置来执行其处理。配置参数default_text_search_config指定了默认配置的名称，如果省略了显式的配置参数，文本检索函数将会使用它。它可以在postgresql.conf中设置，或者使用SET命令为一个单独的会话设置。

有一些预定义的文本检索配置可用，并且你可以容易地创建自定义的配置。为了便于管理文本检索对象，可以使用一组SQL命令，并且有多个psql命令可以显示有关文本检索对象（第 12.10 节）的信息。

作为一个示例，我们将创建一个配置pg，从复制内置的english配置开始：

```
CREATE TEXT SEARCH CONFIGURATION public.pg ( COPY = pg_catalog.
english );
```

我们将使用一个 PostgreSQL 相关的同义词列表，并将它存储在$SHAREDIR/tsearch_data/pg_dict.syn中。文件内容看起来像：

```
postgres    pg
pgsql       pg
postgresql  pg
```

我们定义同义词词典如下：

```
CREATE TEXT SEARCH DICTIONARY pg_dict (
    TEMPLATE = synonym,
    SYNONYMS = pg_dict
);
```

接下来我们注册Ispell词典english_ispell，它有其自己的配置文件：

```
CREATE TEXT SEARCH DICTIONARY english_ispell (
    TEMPLATE = ispell,
    DictFile = english,
    AffFile = english,
    StopWords = english
);
```

现在我们可以在配置pg中建立词的映射：

```
ALTER TEXT SEARCH CONFIGURATION pg
    ALTER MAPPING FOR asciiword, asciihword, hword_asciipart,
                      word, hword, hword_part
    WITH pg_dict, english_ispell, english_stem;
```

我们选择不索引或搜索某些内置配置确实处理的词元类型：

```
ALTER TEXT SEARCH CONFIGURATION pg
    DROP MAPPING FOR email, url, url_path, sfloat, float;
```

现在我们可以测试我们的配置：

```
SELECT * FROM ts_debug('public.pg', '
PostgreSQL, the highly scalable, SQL compliant, open source object-
relational
database management system, is now undergoing beta testing of the
 next
version of our software.
');
```

下一步是把会话设置为使用这个新配置，该配置是在 `public` 模式中创建的：

```
=> \dF
   List of text search configurations
 Schema  | Name | Description
---------+------+-------------
 public  | pg   |

SET default_text_search_config = 'public.pg';
SET

SHOW default_text_search_config;
 default_text_search_config
----------------------------
 public.pg
```

12.8. 测试和调试文本搜索

一个自定义文本检索配置的行为很容易让人困惑。
本节中描述的函数对于测试文本检索对象有用。你可以测试一个完整的配置，
或者独立测试解析器和词典。

12.8.1. 配置测试

函数`ts_debug`可方便地测试一个文本检索配置。

```
ts_debug([ config regconfig, ] document text,
         OUT alias text,
         OUT description text,
         OUT token text,
         OUT dictionaries regdictionary[],
         OUT dictionary regdictionary,
         OUT lexemes text[])
         returns setof record
```

`ts_debug`显示*`document`*的每一个词元的信息，词元由解析器产生并由配置的词典处理过。该函数使用由*`config`*指定的配置，如果省略该参数则使用`default_text_search_config`指定的配置。

ts_debug为解析器在文本中标识的每一个词元返回一行。被返回的列是：

- *alias* text – 词元类型的短名称
- *description* text – 词元类型的描述
- *token* text – 词元的文本
- *dictionaries* regdictionary[] – 配置为这种词元类型选择的词典
- *dictionary* regdictionary – 识别该词元的词典，如果没有词典能识别则为NULL
- *lexemes* text[] – 识别该词元的词典产生的词位，如果没有词典能识别则为NULL；一个空数组（{}）表示该词元被识别为一个停用词

以下是一个简单的示例：

```
SELECT * FROM ts_debug('english','a fat  cat sat on a mat - it ate
 a fat rats');
   alias   |   description   | token |  dictionaries  |  dictionary
  | lexemes
-----------+-----------------+-------+----------------+------------
--+---------
 asciiword | Word, all ASCII | a     | {english_stem} | english_
stem | {}
 blank     | Space symbols   |       | {}             |
  |
 asciiword | Word, all ASCII | fat   | {english_stem} | english_
stem | {fat}
 blank     | Space symbols   |       | {}             |
  |
 asciiword | Word, all ASCII | cat   | {english_stem} | english_
stem | {cat}
 blank     | Space symbols   |       | {}             |
  |
 asciiword | Word, all ASCII | sat   | {english_stem} | english_
stem | {sat}
 blank     | Space symbols   |       | {}             |
  |
 asciiword | Word, all ASCII | on    | {english_stem} | english_
stem | {}
 blank     | Space symbols   |       | {}             |
  |
 asciiword | Word, all ASCII | a     | {english_stem} | english_
stem | {}
 blank     | Space symbols   |       | {}             |
  |
 asciiword | Word, all ASCII | mat   | {english_stem} | english_
stem | {mat}
 blank     | Space symbols   |       | {}             |
  |
 blank     | Space symbols   | -     | {}             |
  |
 asciiword | Word, all ASCII | it    | {english_stem} | english_
stem | {}
 blank     | Space symbols   |       | {}             |
  |
 asciiword | Word, all ASCII | ate   | {english_stem} | english_
stem | {ate}
 blank     | Space symbols   |       | {}             |
  |
 asciiword | Word, all ASCII | a     | {english_stem} | english_
stem | {}
```

```
 blank     | Space symbols   |       | {}             |
  |
 asciiword | Word, all ASCII | fat   | {english_stem} | english_
stem | {fat}
 blank     | Space symbols   |       | {}             |
  |
 asciiword | Word, all ASCII | rats  | {english_stem} | english_
stem | {rat}
```

为了做更完整的演示，我们先为英语创建一个public.english配置和 Ispell 词典：

```
CREATE TEXT SEARCH CONFIGURATION public.english ( COPY = pg_
catalog.english );

CREATE TEXT SEARCH DICTIONARY english_ispell (
    TEMPLATE = ispell,
    DictFile = english,
    AffFile = english,
    StopWords = english
);

ALTER TEXT SEARCH CONFIGURATION public.english
   ALTER MAPPING FOR asciiword WITH english_ispell, english_stem;

SELECT * FROM ts_debug('public.english','The Brightest
 supernovaes');
   alias   |   description   |    token    |         dictionaries
        |   dictionary   |   lexemes
-----------+-----------------+-------------+-----------------------
--------+----------------+-------------
 asciiword | Word, all ASCII | The         | {english_ispell,
english_stem} | english_ispell | {}
 blank     | Space symbols   |             | {}
        |                |
 asciiword | Word, all ASCII | Brightest   | {english_ispell,
english_stem} | english_ispell | {bright}
 blank     | Space symbols   |             | {}
        |                |
 asciiword | Word, all ASCII | supernovaes | {english_ispell,
english_stem} | english_stem   | {supernova}
```

在这个示例中，词Brightest被解析器识别为一个ASCII 词（别名asciiword）。对于这种词元类型，词典列表是english_ispell和english_stem。该词被english_ispell识别，并被这个词典归约为词位bright。词supernovaes对english_ispell词典来说是未知的，因此它会被传递给下一个词典；幸运的是，它随后被识别了。（实际上，english_stem是一个 Snowball 词典，它能够识别所有输入；这也是为什么它被放在词典列表末尾。）

词The被english_ispell词典识别为一个停用词（第 12.6.1 节）并且将不会被索引。空格也被丢弃，因为该配置没有为它们提供词典。

你可以通过明确指定要查看的列来减少输出宽度：

```
SELECT alias, token, dictionary, lexemes
FROM ts_debug('public.english','The Brightest supernovaes');
   alias   |    token    |   dictionary   |   lexemes
-----------+-------------+----------------+-------------
 asciiword | The         | english_ispell | {}
 blank     |             |                |
```

```
 asciiword | Brightest   | english_ispell | {bright}
 blank     |             |                |
 asciiword | supernovaes | english_stem   | {supernova}
```

12.8.2. 解析器测试

下列函数允许直接测试一个文本检索解析器。

```
ts_parse(parser_name text, document text,
         OUT tokid integer, OUT token text) returns setof record
ts_parse(parser_oid oid, document text,
         OUT tokid integer, OUT token text) returns setof record
```

`ts_parse`解析给定的*`document`*，并返回一组记录，每个由解析过程产生的词元对应一条记录。每条记录都包含一个 `tokid`，显示所分配的词元类型，以及一个 `token`，即该词元的文本。例如：

```
SELECT * FROM ts_parse('default', '123 - a number');
 tokid | token
-------+--------
    22 | 123
    12 |
    12 | -
     1 | a
    12 |
     1 | number
```

```
ts_token_type(parser_name text, OUT tokid integer,
              OUT alias text, OUT description text) returns setof
 record
ts_token_type(parser_oid oid, OUT tokid integer,
              OUT alias text, OUT description text) returns setof
 record
```

`ts_token_type`返回一个表，描述指定解析器能够识别的每一种词元类型。对于每种词元类型，该表给出解析器用来标记该类词元的整数 `tokid`、在配置命令中命名该词元类型的 `alias`，以及简短的 `description`。例如：

```
SELECT * FROM ts_token_type('default');
 tokid |      alias      |               description
-------+-----------------+-----------------------------------------
-
     1 | asciiword       | Word, all ASCII
     2 | word            | Word, all letters
     3 | numword         | Word, letters and digits
     4 | email           | Email address
     5 | url             | URL
     6 | host            | Host
     7 | sfloat          | Scientific notation
     8 | version         | Version number
     9 | hword_numpart   | Hyphenated word part, letters and digits
    10 | hword_part      | Hyphenated word part, all letters
    11 | hword_asciipart | Hyphenated word part, all ASCII
    12 | blank           | Space symbols
    13 | tag             | XML tag
    14 | protocol        | Protocol head
    15 | numhword        | Hyphenated word, letters and digits
```

```
 16 | asciihword      | Hyphenated word, all ASCII
 17 | hword           | Hyphenated word, all letters
 18 | url_path        | URL path
 19 | file            | File or path name
 20 | float           | Decimal notation
 21 | int             | Signed integer
 22 | uint            | Unsigned integer
 23 | entity          | XML entity
```

12.8.3. 词典测试

`ts_lexize`函数便于测试词典。

```
ts_lexize(dict regdictionary, token text) returns text[]
```

如果输入的*token*是该词典已知的，则`ts_lexize`返回一个词位数组；
如果词元是词典已知的但是它是一个停用词，则返回一个空数组；
或者如果它对词典是未知词，则返回`NULL`。

示例：

```
SELECT ts_lexize('english_stem', 'stars');
 ts_lexize
-----------
 {star}

SELECT ts_lexize('english_stem', 'a');
 ts_lexize
-----------
 {}
```

注意

`ts_lexize`函数期望的是单个词元，而不是文本。下面这种情况可能会让人困惑：

```
SELECT ts_lexize('thesaurus_astro','supernovae stars') is
 null;
 ?column?
----------
 t
```

分类词典 `thesaurus_astro` 确实知道短语 `supernovae stars`，但 `ts_lexize` 会失败，因为它不会解析输入文本，而是把它当作一个单独的词元。测试分类词典时应使用 `plainto_tsquery` 或 `to_tsvector`，例如：

```
SELECT plainto_tsquery('supernovae stars');
 plainto_tsquery
-----------------
 'sn'
```

12.9. GiST 和 GIN 索引类型

有两种索引可以用来加速全文搜索。请注意，索引对于全文搜索并非强制要求，
但在定期搜索某一列的情况下，通常是可取的。

```
CREATE INDEX name ON table USING gist(column);
```

创建基于 GiST（广义搜索树）的索引。*column*可以是`tsvector`或`tsquery`类型。

```
CREATE INDEX name ON table USING gin(column);
```

创建基于 GIN（广义倒排索引）的索引。*column*必须是`tsvector`类型。

这两种索引类型之间存在显著的性能差异，因此理解它们的特性很重要。

一个 GiST 索引是有损的，这表示索引可能产生假匹配，并且有必要检查真实的表行来消除这种假匹配（PostgreSQL在需要时会自动做这一步）。GiST 索引之所以是有损的，是因为每一个文档在索引中被表示为一个定长的签名。该签名通过将每个词 hash 到一个 n 位串中的一位，再将所有这些位进行 OR 运算来生成，结果是一个 n 位的文档签名。当两个词 hash 到同一个位位置时就会产生假匹配。如果查询中所有词都有匹配（真或假），则必须检索表行查看匹配是否正确。

有损性导致的性能下降归因于不必要的表记录（即被证实为假匹配的记录）获取。
因为表记录的随机访问是较慢的，这限制了 GiST 索引的实用性。假匹配的可能性取决于几个因素，特别是不同词的数量，因此推荐使用词典来缩减这个数量。

GIN 索引对标准查询不是有损的，但其性能以对数方式依赖于不同词的数量。（不过，GIN 索引只存储`tsvector`值的词（词位），而不存储它们的权重标签。因此，
在使用涉及权重的查询时需要一次表行重新检查。）

在选择使用哪种索引类型（GiST 还是 GIN）时，请考虑这些性能差异：

- GIN 索引的查找大约比 GiST 快三倍
- GIN 索引的构建大约比 GiST 慢三倍
- GIN 索引的更新比 GiST 索引略慢，但如果禁用了快速更新支持则要慢大约 10 倍（详见第 55.3.1 节）
- GIN 索引比 GiST 索引大两到三倍

根据经验法则，GIN 索引最适合静态数据，因为查找更快。对于动态数据，GiST 索引的更新更快。具体来说，当不同词（词位）的数量在 100,000 以下时，GiST 索引对动态数据非常好而且快；而 GIN 索引能更好地处理 100,000 以上的词位，但更新较慢。

注意GIN索引的构建时间常常可以通过增加maintenance_work_mem来缩短，而
GiST索引的构建时间则对该参数不敏感。

对大集合分区并正确使用 GIN 和 GiST 索引允许实现带在线更新的快速搜索。分区可以在数据库层面上使用表继承来完成，或者通过将文档分布在服务器上，并使用
dblink模块收集搜索结果来完成。后者是可能的，因为排名函数只使用本地信息。

12.10. psql 支持

关于文本检索配置对象的信息可以在psql中使用一组命令获得：

```
\dF{d,p,t}[+] [PATTERN]
```

可选的`+`能产生更多细节。

可选参数 `PATTERN` 可以是文本检索对象的名称，也可以可选地带上模式限定。如果省略`PATTERN`，则会显示所有可见对象的信息。`PATTERN` 还可以是正则表达式，并且可以为模式名和对象名分别提供独立的匹配模式。下面的示例说明了这一点：

```
=> \dF *fulltext*
       List of text search configurations
 Schema |  Name        | Description
```

```
--------+--------------+-------------
 public | fulltext_cfg |

=> \dF *.fulltext*
       List of text search configurations
 Schema   |  Name        | Description
----------+----------------------------
 fulltext | fulltext_cfg |
 public   | fulltext_cfg |
```

可用命令如下：

```
\dF[+] [PATTERN]
```

列出文本检索配置（加上 + 可显示更多细节）。

```
=> \dF russian
            List of text search configurations
   Schema   |  Name   |            Description
------------+---------+-----------------------------------
 pg_catalog | russian | configuration for russian language

=> \dF+ russian
Text search configuration "pg_catalog.russian"
Parser: "pg_catalog.default"
      Token      | Dictionaries
-----------------+--------------
 asciihword      | english_stem
 asciiword       | english_stem
 email           | simple
 file            | simple
 float           | simple
 host            | simple
 hword           | russian_stem
 hword_asciipart | english_stem
 hword_numpart   | simple
 hword_part      | russian_stem
 int             | simple
 numhword        | simple
 numword         | simple
 sfloat          | simple
 uint            | simple
 url             | simple
 url_path        | simple
 version         | simple
 word            | russian_stem
```

```
\dFd[+] [PATTERN]
```

列出文本检索词典（加上 + 可显示更多细节）。

```
=> \dFd
                            List of text search dictionaries
   Schema   |      Name       |
 Description
------------+-----------------+-------------------------------
--------------------------
 pg_catalog | danish_stem     | snowball stemmer for danish
 language
```

```
 pg_catalog | dutch_stem      | snowball stemmer for dutch
 language
 pg_catalog | english_stem    | snowball stemmer for english
 language
 pg_catalog | finnish_stem    | snowball stemmer for finnish
 language
 pg_catalog | french_stem     | snowball stemmer for french
 language
 pg_catalog | german_stem     | snowball stemmer for german
 language
 pg_catalog | hungarian_stem  | snowball stemmer for hungarian
 language
 pg_catalog | italian_stem    | snowball stemmer for italian
 language
 pg_catalog | norwegian_stem  | snowball stemmer for norwegian
 language
 pg_catalog | portuguese_stem | snowball stemmer for portuguese
 language
 pg_catalog | romanian_stem   | snowball stemmer for romanian
 language
 pg_catalog | russian_stem    | snowball stemmer for russian
 language
 pg_catalog | simple          | simple dictionary: just lower
 case and check for stopword
 pg_catalog | spanish_stem    | snowball stemmer for spanish
 language
 pg_catalog | swedish_stem    | snowball stemmer for swedish
 language
 pg_catalog | turkish_stem    | snowball stemmer for turkish
 language
```

```
    \dFp[+] [PATTERN]
```

列出文本检索解析器（加上 + 可显示更多细节）。

```
=> \dFp
        List of text search parsers
   Schema   |  Name   |     Description
------------+---------+---------------------
 pg_catalog | default | default word parser
=> \dFp+
    Text search parser "pg_catalog.default"
     Method      |    Function    | Description
-----------------+----------------+-------------
 Start parse     | prsd_start     |
 Get next token  | prsd_nexttoken |
 End parse       | prsd_end       |
 Get headline    | prsd_headline  |
 Get token types | prsd_lextype   |

        Token types for parser "pg_catalog.default"
   Token name    |               Description
-----------------+------------------------------------------
 asciihword      | Hyphenated word, all ASCII
 asciiword       | Word, all ASCII
 blank           | Space symbols
 email           | Email address
 entity          | XML entity
 file            | File or path name
```

```
 float           | Decimal notation
 host            | Host
 hword           | Hyphenated word, all letters
 hword_asciipart | Hyphenated word part, all ASCII
 hword_numpart   | Hyphenated word part, letters and digits
 hword_part      | Hyphenated word part, all letters
 int             | Signed integer
 numhword        | Hyphenated word, letters and digits
 numword         | Word, letters and digits
 protocol        | Protocol head
 sfloat          | Scientific notation
 tag             | XML tag
 uint            | Unsigned integer
 url             | URL
 url_path        | URL path
 version         | Version number
 word            | Word, all letters
(23 rows)
```

```
     \dFt[+] [PATTERN]
```

列出文本检索模板（加上 + 可显示更多细节）。

```
=> \dFt
                            List of text search templates
   Schema   |   Name    |                        Description
------------+-----------+----------------------------------------
--------------------
 pg_catalog | ispell    | ispell dictionary
 pg_catalog | simple    | simple dictionary: just lower case and
 check for stopword
 pg_catalog | snowball  | snowball stemmer
 pg_catalog | synonym   | synonym dictionary: replace word by
 its synonym
 pg_catalog | thesaurus | thesaurus dictionary: phrase by phrase
 substitution
```

12.11. 限制

目前，PostgreSQL的文本检索功能存在以下限制：

- 每个词位的长度必须小于 2K 字节
- `tsvector` 的长度（词位加位置）必须小于 1 兆字节
- 词位数量必须小于 2^{64}
- `tsvector` 中的位置值必须大于 0 且不超过 16,383
- 每个词位的位置数不能超过 256 个
- `tsquery` 中的节点数（词位加操作符）必须小于 32,768

为了对比，PostgreSQL 8.1 的文档包含 10,441 个不同词，总数 335,420 个词，并且最频繁的词 “postgresql” 在 655 个文档中被提到 6,127 次。

另一个示例 – PostgreSQL的邮件列表归档在 461,020 条消息的 57,491,343 个词位中包含 910,989 个不同词。

12.12. 从 8.3 之前的文本搜索迁移

使用tsearch2模块进行文本搜索的应用需要进行一些调整，才能使用内置功能：

- 一些函数已被重命名，或在其参数列表上做了小调整，而且它们现在都位于 `pg_catalog` 模式中，而在以前的安装中它们可能位于`public`或其他非系统模式中。新版本的tsearch2提供了一个兼容层，可解决这一领域的大多数问题。

- 从 8.3 之前的数据库加载pg_dump输出时，必须屏蔽旧的tsearch2函数和其他对象。虽然其中很多本来就无法加载，但少数可以加载并随后引发问题。
  处理这个问题的一种简单方法是在恢复转储之前先加载新的 tsearch2模块；这样它就会阻止旧对象被加载。

- 文本搜索配置的设置方式现在完全不同了。不再需要手动向配置表中插入行，
  而是通过本章前面介绍的专用 SQL 命令来配置搜索。没有将现有自定义配置转换为 8.3 形式的自动化支持；这一步需要你自己完成。

- 大多数类型的词典依赖于数据库之外的一些配置文件。这些文件与 8.3 之前的用法大体兼容，但请注意以下差异：
  - 配置文件现在必须放在一个指定的目录中（`$SHAREDIR/tsearch_data`），
    并且根据文件类型必须有特定的扩展名，如前面各种词典类型的描述中所述。
    添加这一限制是为了防止安全问题。
  - 无论使用什么数据库编码，配置文件都必须以 UTF-8 编码。
  - 在分类词典配置文件中，停用词必须用`?`标记。

第 13 章 并发控制

本章描述 PostgreSQL 数据库系统在两个或更多会话试图同时访问同一数据时的行为。在这种情况下，目标是在保持严格数据完整性的同时，为所有会话提供高效访问。
每一个数据库应用开发者都应该熟悉本章涉及的主题。

13.1. 介绍

PostgreSQL 为开发者提供了一组丰富的工具来管理对数据的并发访问。在内部，数据一致性是通过使用多版本模型（多版本并发控制，MVCC）来维护的。这意味着，在查询数据库时，每个事务看到的都是某个较早时刻的数据快照（即一个数据库版本），
而不受底层数据当前状态的影响。这样可以保护事务不看到由（其他）
并发事务在同一数据行上执行更新所可能导致的不一致数据，从而为每个数据库会话提供事务隔离。MVCC 摒弃了传统数据库系统的加锁方法，尽量减少锁争用，以便在多用户环境中获得合理的性能。

使用 MVCC 并发控制模型而不是加锁的主要优点在于，在 MVCC 中，为查询（读取）数据而获得的锁不会与为写入数据而获得的锁冲突，因此读取不会阻塞写入，
写入也不会阻塞读取。PostgreSQL 甚至在通过创新性的可串行化快照隔离（SSI）级别提供最严格的事务隔离时，也保持了这一保证。

PostgreSQL 也提供表级和行级锁设施，供那些通常不需要完整事务隔离、并且倾向于显式管理特定冲突点的应用使用。不过，正确使用 MVCC 通常会比加锁提供更好的性能。此外，由应用定义的咨询锁还提供了一种获取不依附于单个事务的锁的机制。

13.2. 事务隔离

SQL 标准定义了四个事务隔离级别。最严格的是可串行化，标准将它定义为：一组可串行化事务的任意并发执行，都保证与按照某种顺序一次只运行一个事务的效果相同。
其他三个级别则按现象定义，这些现象源自并发事务之间的相互作用，
而在各自级别上这些现象不得发生。标准指出，由于可串行化的定义，
这些现象在该级别下都不可能出现。（这并不奇怪，
如果事务的效果必须与一次只运行一个事务保持一致，
又怎么可能看到由相互作用引起的任何现象呢？）

在各个级别上被禁止的现象有：

脏读

一个事务读取了由另一个并发未提交事务写入的数据。

不可重复读

一个事务重新读取它之前读过的数据，
发现这些数据已经被另一个事务修改（且该事务已在初次读取之后提交）。

幻读

一个事务重新执行一个返回满足某个搜索条件的一组行的查询，
却发现由于另一个最近提交的事务，满足该条件的行集已经发生了变化。

四种事务隔离级别及其对应行为见表 13.1。

表 13.1. 标准 SQL 事务隔离级别

隔离级别	脏读	不可重复读	幻读
读未提交	可能	可能	可能
读已提交	不可能	可能	可能

隔离级别	脏读	不可重复读	幻读
可重复读	不可能	不可能	可能
可串行化	不可能	不可能	不可能

在 PostgreSQL 中，你可以请求四种标准事务隔离级别中的任意一种。但在内部，实际上只有三种不同的隔离级别，分别对应读已提交、可重复读和可串行化这几个级别。
当你选择读未提交级别时，实际得到的是读已提交；而在 PostgreSQL 的可重复读实现中，幻读是不可能发生的，因此实际的隔离级别可能比你选择的更严格。这是 SQL 标准所允许的：这四个隔离级别只定义了哪些现象不允许发生，并没有定义哪些现象必须发生。PostgreSQL 之所以只提供三种隔离级别，是因为要把标准隔离级别映射到多版本并发控制架构上，这是唯一合理的做法。可用隔离级别的行为将在下面各小节中详细说明。

要设置事务的事务隔离级别，请使用命令SET TRANSACTION。

13.2.1. 读已提交隔离级别

读已提交是 PostgreSQL 中默认的隔离级别。当事务使用这一隔离级别时，`SELECT` 查询（不带 `FOR UPDATE/SHARE` 子句）只能看到查询开始之前已经提交的数据；它既看不到未提交的数据，也看不到查询执行期间并发事务提交的更改。实际上，`SELECT` 查询看到的是该查询开始运行瞬间的数据库快照。不过，`SELECT` 能看到其自身事务中先前执行的更新效果，即使这些更新尚未提交。还要注意，即使两个连续的 `SELECT` 命令位于同一个事务中，如果其他事务在第一个 `SELECT` 执行期间提交了更改，这两个命令也可能看到不同的数据。

`UPDATE`、`DELETE`、`SELECT FOR UPDATE` 和 `SELECT FOR SHARE` 命令在搜索目标行时的行为与 `SELECT` 相同：它们只会找到在命令开始时已经提交的目标行。不过，当找到这样的目标行时，它可能已经被其他并发事务更新、删除或者锁定。在这种情况下，即将执行更新的事务会等待第一个更新事务提交或回滚（如果它仍在进行中）。如果第一个更新事务回滚，那么它的作用将被撤销，第二个更新事务就可以继续更新最初找到的行。如果第一个更新事务提交了，那么若该行已被第一个更新者删除，第二个更新事务就会忽略该行；否则，第二个更新者将尝试在该行的已更新版本上应用自己的操作。该命令的搜索条件（`WHERE` 子句）会被重新计算，以判断该行的已更新版本是否仍然符合搜索条件。如果符合，则第二个更新者会基于该行的已更新版本继续执行操作。对于 `SELECT FOR UPDATE` 和 `SELECT FOR SHARE`，这意味着被锁住并返回给客户端的是该行的已更新版本。

由于上述规则，更新命令可能看到一个不一致的快照：它能够看到并发更新命令在它试图更新的同一行上的效果，却看不到这些命令对数据库中其他行的影响。这种行为使得读已提交模式不适合涉及复杂搜索条件的命令；不过，对于较简单的场景它恰到好处。例如，考虑以如下事务更新银行余额：

```
BEGIN;
UPDATE accounts SET balance = balance + 100.00 WHERE acctnum =
 12345;
UPDATE accounts SET balance = balance - 100.00 WHERE acctnum =
 7534;
COMMIT;
```

如果两个这样的事务并发地尝试更改账户 12345 的余额，我们显然希望第二个事务从该账户行的已更新版本开始。因为每个命令只影响一个预先确定的行，让它看到该行的已更新版本不会造成任何麻烦的不一致。

在读已提交模式下，更复杂的用法可能产生不理想的结果。例如，考虑一个 `DELETE` 命令，另一个命令正在修改数据，使某些行开始满足其筛选条件、另一些行不再满足。假设 `website` 是一个有两行的表，其中 `website.hits` 分别等于 `9` 和 `10`：

```
BEGIN;
UPDATE website SET hits = hits + 1;
-- 从另一个会话运行： DELETE FROM website WHERE hits = 10;
```

```
COMMIT;
```

该 DELETE 将不会产生任何效果，即使在 UPDATE 之前和之后都存在一行 website.hits = 10。这是因为更新前值为 9 的那一行被跳过了，而当 UPDATE 完成并且 DELETE 获得锁时，新行值已经不再是 10 而是 11，因此不再匹配条件。

因为在读已提交模式中，每个命令都从一个新快照开始，
而该快照包含截至当时已提交的所有事务，
所以同一事务中的后续命令无论如何都会看到并发事务已提交的效果。
上面真正的问题在于，单个命令是否能看到数据库的绝对一致视图。

读已提交模式所提供的部分事务隔离对于许多应用来说已经足够，
而且这种模式既快速又容易使用。不过，它并不能满足所有场景。
执行复杂查询和更新的应用，可能需要比读已提交模式所提供的更严格一致的数据库视图。

13.2.2. 可重复读隔离级别

可重复读隔离级别只能看到事务开始前已提交的数据；它看不到未提交的数据，
也看不到事务执行期间并发事务提交的更改。（不过，
每个查询都能看到其自身事务中先前执行的更新效果，即使这些更新尚未提交。）这比 SQL 标准对该隔离级别的要求更强，能够防止表 13.1中描述的所有现象。正如前面提到的，这正是标准明确允许的，因为标准只描述每个隔离级别必须提供的最低保护。

这一点与读已提交不同：在可重复读事务中，查询看到的是该事务开始时的快照，
而不是事务中当前查询开始时的快照。因此，在单个事务中的连续 SELECT 命令会看到相同的数据，也就是说，它们不会看到在自身事务开始后由其他事务提交的更改。

使用这一隔离级别的应用必须准备好在发生串行化失败时重试事务。

UPDATE、DELETE、SELECT FOR UPDATE 和 SELECT FOR SHARE 命令在搜索目标行方面与SELECT相同：它们只会找到在事务开始时已提交的目标行。然而，
这样的目标行可能在找到时已被另一个并发事务更新（或删除或锁定）。在这种情况下，
可重复读事务将等待第一个更新事务提交或回滚（如果仍在进行）。
如果第一个更新者回滚，则其效果将被取消，可重复读事务可以继续更新最初找到的行。
但如果第一个更新者提交了（并实际更新或删除了行，而不仅仅是锁定了它），
那么可重复读事务将被回滚，并显示以下消息：

```
ERROR:  could not serialize access due to concurrent update
```

因为可重复读事务不能修改或锁定在其开始之后已被其他事务更改的行。

当应用收到这条错误消息时，应当中止当前事务，并从头重试整个事务。第二次执行时，
该事务会把先前已提交的更改视为其初始数据库视图的一部分，
因此以该行的新版本作为新事务更新的起点时，就不存在逻辑冲突。

注意，只有更新事务才可能需要重试；只读事务永远不会发生串行化冲突。

可重复读模式提供了严格的保证，即每个事务都看到数据库的一个完全稳定的视图。不过，
这个视图并不一定总能与同一级别并发事务的某种串行（一次一个）执行保持一致。例如，
即使是该级别的只读事务，也可能看到一条表明某批次已完成的控制记录，却
看不到逻辑上属于该批次的某条明细记录，
因为创建该明细记录的事务读取了控制记录的较早版本。
如果不仔细使用显式锁来阻塞冲突事务，
试图依靠运行在这一隔离级别的事务来强制业务规则，往往无法正确工作。

> **注意**
>
> 在 PostgreSQL 9.1 之前，请求可串行化事务隔离级别会得到与这里描述完全相同的行为。若要保留旧式的可串行化行为，现在应请求可重复读。

13.2.3. 可串行化隔离级别

可串行化隔离级别提供最严格的事务隔离。该级别模拟串行事务执行；也就是说，
效果就像这些事务不是并发执行，而是一个接一个地串行执行一样。不过，
与可重复读级别一样，使用该级别的应用必须准备好在发生串行化失败时重试事务。
实际上，这个隔离级别的工作方式与可重复读完全相同，
只是它还会监视那些可能使一组并发可串行化事务的执行结果，
与这些事务任何可能的串行（一次一个）执行都不一致的条件。
这种监视不会引入超出可重复读已有阻塞之外的任何阻塞，但会带来一定开销；
一旦检测到可能导致串行化异常的条件，就会触发串行化失败。

例如，考虑表mytab，它最初包含以下内容：

```
 class | value
-------+-------
     1 |    10
     1 |    20
     2 |   100
     2 |   200
```

假设可串行化事务 A 执行以下计算：

```
SELECT SUM(value) FROM mytab WHERE class = 1;
```

然后将结果（30）作为新行的value插入，且该行满足class = 2。与此同时，可串行化事务 B 执行以下计算：

```
SELECT SUM(value) FROM mytab WHERE class = 2;
```

得到结果 300，并将它插入一个新行，该行满足class = 1。然后两个事务都尝试提交。如果任一事务运行在可重复读隔离级别，两个事务都可以提交；
但由于不存在与该结果一致的串行执行顺序，使用可串行化事务时，
只会允许一个事务提交，另一个则会被回滚，并收到以下消息：

```
ERROR:  could not serialize access due to read/write dependencies
 among transactions
```

这是因为，如果 A 在 B 之前执行，B 算出的总和应该是 330，而不是 300；同样，按另一种顺序执行，A 算出的总和也会不同。

为了保证真正的可串行性，PostgreSQL 使用了谓词锁，也就是说，系统会保留一些锁，以便判断某个写操作如果先发生，是否会影响并发事务先前读取的结果。在 PostgreSQL 中，这些锁不会造成任何阻塞，因此不会参与形成死锁。
它们用于识别并标记并发可串行化事务之间的依赖关系，
而这些依赖在某些组合下可能导致串行化异常。相比之下，
想要保证数据一致性的读已提交或可重复读事务，可能需要在整张表上加锁，
这会阻塞其他试图使用该表的用户；或者它可能需要使用 SELECT FOR UPDATE 或 SELECT FOR SHARE，而这些做法不仅可能阻塞其他事务，还会造成磁盘访问。

与大多数其他数据库系统一样，PostgreSQL 中的谓词锁以事务实际访问的数据为基础。这些锁会出现在pg_locks系统视图中，其 mode 为 SIReadLock。查询执行期间获取的具体锁取决于查询所使用的计划，并且在事务过程中，多个更细粒度的锁（如元组锁）
可能会合并成较少的粗粒度锁（如页锁），以防耗尽用于跟踪锁的内存。如果某个 READ ONLY 事务检测到不再可能发生会导致串行化异常的冲突，它可以在完成前释放其 SIRead 锁。事实上，READ ONLY 事务通常在启动时就能确定这一点，从而避免获取任何谓词锁。如果你显式请求一个 SERIALIZABLE READ ONLY DEFERRABLE 事务，它会阻塞，直到能够确立这一事实为止。（这是唯一一种可串行化事务会阻塞而可重复读事务不会阻塞的情况。）另一方面，SIRead 锁往往需要保留到事务提交之后，直到与之重叠的读写事务完成。

统一采用可串行化事务可以简化开发。任何一组成功提交的并发可串行化事务，都保证与把它们一次一个运行的效果相同；这意味着，如果你能够证明某个事务在单独运行时会做正确的事，那么即使不了解其他事务会做什么，也可以相信它在任何可串行化事务的混合执行中也会做正确的事，否则它就不会成功提交。重要的是，采用这种技术的环境应当有一套通用机制来处理串行化失败（其 SQLSTATE 值总是 '40001'），因为很难准确预测究竟哪些事务会对读/写依赖关系有贡献，并因此需要被回滚来防止串行化异常。监视读/写依赖关系会带来开销，被串行化失败中止的事务重新启动也会带来开销；但在把这些开销与显式锁以及 `SELECT FOR UPDATE` 或 `SELECT FOR SHARE` 所涉及的成本和阻塞进行权衡之后，可串行化事务在某些环境中仍然是性能最佳的选择。

虽然 PostgreSQL 的可串行化事务隔离级别只允许那些能够证明存在等价串行执行顺序的并发事务提交，但它并不总能阻止某些在真正串行执行中不会出现的错误被报告。特别是，即使在尝试插入某个键之前已经显式检查过该键不存在，仍然可能看到由于重叠执行的可串行化事务引起的唯一约束违例。要避免这种问题，必须确保所有插入潜在冲突键的可串行化事务，都先显式检查自己是否可以这样做。例如，设想一个要求用户输入新键的应用，它会先尝试查询用户给定的键以检查其是否已经存在，或者通过选取当前最大键再加一来生成新键。如果某些可串行化事务不遵循这一协议而直接插入新键，那么即使在这些并发事务的串行执行中本不会发生唯一约束违例，系统仍可能报告唯一约束违例。

当依赖可串行化事务进行并发控制时，为了获得最佳性能，应考虑以下问题：

- 在可能时声明事务为`READ ONLY`。
- 控制活动连接的数量，并在需要时使用连接池。这始终是重要的性能考量，但在使用可串行化事务的繁忙系统中尤为重要。
- 单个事务中只包含为保证完整性所必需的内容。
- 不要让连接不必要地“闲置在事务中”。
- 在那些已经由可串行化事务自动提供保护的地方，消除不再需要的显式锁、`SELECT FOR UPDATE` 和 `SELECT FOR SHARE`。
- 当系统因谓词锁表内存不足而被迫把多个页级谓词锁合并为单个关系级谓词锁时，串行化失败的比例可能会上升。你可以通过增加max_pred_locks_per_transaction来避免这种情况。
- 顺序扫描总是需要关系级谓词锁。这可能导致串行化失败的比例上升。通过降低random_page_cost和/或提高cpu_tuple_cost来鼓励使用索引扫描，可能对此有所帮助。务必在事务回滚和重启次数减少所带来的收益，与查询执行时间整体变化之间进行权衡。

13.3. 显式锁定

PostgreSQL 提供了多种锁模式，用于控制对表中数据的并发访问。在 MVCC 无法给出期望行为的场景中，这些模式可用于由应用自行控制的加锁。此外，大多数 PostgreSQL 命令也会自动获取适当模式的锁，以确保在命令执行期间，被引用的表不会以不兼容的方式被删除或修改。（例如，`TRUNCATE` 无法安全地与同一张表上的其他操作并发执行，因此它会在该表上获取独占锁来强制实现这一点。）

要查看数据库服务器中当前尚未释放的锁列表，可以使用 `pg_locks` 系统视图。有关监控锁管理器子系统状态的更多信息，请参见第 27 章。

13.3.1. 表级锁

下面的列表给出了可用的锁模式，以及它们在 PostgreSQL 中被自动使用的场景。你也可以通过命令LOCK显式获取其中任意一种锁。请记住，这些锁模式全部都是表级锁，

即使名称中包含“row”一词；这些名称只是历史遗留。在某种程度上，
这些名称反映了各锁模式的典型用途 – 但它们的语义是完全相同的。一个锁模式与另一个锁模式真正的区别，只在于它与哪些锁模式冲突（见表 13.2）。两个事务不能在同一时刻在同一张表上持有相互冲突模式的锁。（不过，事务永远不会与自己冲突。例如，它可能先获取 ACCESS EXCLUSIVE 锁，随后又在同一张表上获取 ACCESS SHARE 锁。）不冲突的锁模式可以被多个事务同时持有。特别要注意，有些锁模式与自身冲突（例如，一次只能有一个事务持有 ACCESS EXCLUSIVE 锁），而另一些锁模式并不与自身冲突（例如，可以有多个事务持有 ACCESS SHARE 锁）。

表级锁模式

ACCESS SHARE

只与 ACCESS EXCLUSIVE 锁模式冲突。

SELECT 命令会在被引用的表上获取这种模式的锁。通常，任何只读取表而不修改它的查询都会获取这种锁模式。

ROW SHARE

与 EXCLUSIVE 和 ACCESS EXCLUSIVE 锁模式冲突。

SELECT FOR UPDATE 和 SELECT FOR SHARE 命令会在目标表上获取这种模式的锁（此外，对于引用到但没有用 FOR UPDATE/FOR SHARE 选取的其他表，还会获取 ACCESS SHARE 锁）。

ROW EXCLUSIVE

与SHARE、SHARE ROW EXCLUSIVE、EXCLUSIVE和ACCESS EXCLUSIVE锁模式冲突。

UPDATE、DELETE 和 INSERT 命令会在目标表上获取这种锁模式（此外，对任何其他被引用的表还会获取 ACCESS SHARE 锁）。通常，任何修改表中数据的命令都会获取这种锁模式。

SHARE UPDATE EXCLUSIVE

与 SHARE UPDATE EXCLUSIVE、SHARE、SHARE ROW EXCLUSIVE、EXCLUSIVE 和 ACCESS EXCLUSIVE 锁模式冲突。这种模式用于保护表不受并发模式更改和 VACUUM 运行的影响。

由 VACUUM（不带 FULL）、ANALYZE、CREATE INDEX CONCURRENTLY 以及某些形式的 ALTER TABLE 获取。

SHARE

与 ROW EXCLUSIVE、SHARE UPDATE EXCLUSIVE、SHARE ROW EXCLUSIVE、EXCLUSIVE 和 ACCESS EXCLUSIVE 锁模式冲突。这种模式用于保护表不受并发数据更改的影响。

由 CREATE INDEX（不带 CONCURRENTLY）获取。

SHARE ROW EXCLUSIVE

与 ROW EXCLUSIVE、SHARE UPDATE EXCLUSIVE、SHARE、SHARE ROW EXCLUSIVE、EXCLUSIVE 和 ACCESS EXCLUSIVE 锁模式冲突。这种模式用于保护表不受并发数据更改的影响，并且是自排他的，因此同一时刻只能有一个会话持有它。

这种锁模式不会被任何 PostgreSQL 命令自动获取。

EXCLUSIVE

与 ROW SHARE、ROW EXCLUSIVE、SHARE UPDATE EXCLUSIVE、SHARE、SHARE ROW EXCLUSIVE、EXCLUSIVE 和 ACCESS EXCLUSIVE 锁模式冲突。这种模式只允许并发的 ACCESS SHARE 锁，也就是说，只有对该表的读操作可以与持有这种锁模式的事务并行执行。

没有任何 PostgreSQL 命令会自动在表上获取这种锁模式。

ACCESS EXCLUSIVE

与所有模式的锁冲突（ACCESS SHARE、ROW SHARE、ROW EXCLUSIVE、SHARE UPDATE EXCLUSIVE、SHARE、SHARE ROW EXCLUSIVE、EXCLUSIVE 和 ACCESS EXCLUSIVE）。这种模式保证持有者是以任何方式访问该表的唯一事务。

由 ALTER TABLE、DROP TABLE、TRUNCATE、REINDEX、CLUSTER 和 VACUUM FULL 命令获取。这也是未显式指定模式的 LOCK TABLE 语句的默认锁模式。

> **提示**
>
> 只有 ACCESS EXCLUSIVE 锁才会阻塞 SELECT（不带 FOR UPDATE/SHARE）语句。

锁一旦被获取，通常会一直持有到事务结束。但是，如果锁是在建立保存点之后才获取的，那么在回滚到该保存点时，这个锁会立即释放。这与 ROLLBACK 会取消保存点之后所有命令效果的原则是一致的。相同的原则也适用于在 PL/pgSQL 异常块中获取的锁：从该块中因错误跳出时，会释放在块中获取的锁。

表 13.2. 冲突的锁模式

请求的锁模式	当前的锁模式							
	ACCESS SHARE	ROW SHARE	ROW EXCLUSIVE	SHARE UPDATE EXCLUSIVE	SHARE	SHARE ROW EXCLUSIVE	EXCLUSIVE	ACCESS EXCLUSIVE
ACCESS SHARE								X
ROW SHARE							X	X
ROW EXCLUSIVE					X	X	X	X
SHARE UPDATE EXCLUSIVE				X	X	X	X	X
SHARE			X	X		X	X	X
SHARE ROW EXCLUSIVE			X	X	X	X	X	X
EXCLUSIVE		X	X	X	X	X	X	X
ACCESS EXCLUSIVE	X	X	X	X	X	X	X	X

13.3.2. 行级锁

除了表级锁之外，还有行级锁，它可以是排他锁或共享锁。当某一行被更新或删除时，会自动获取该行上的排他行级锁。与表级锁一样，该锁会被持有到事务提交或回滚为止。行级锁不会影响数据查询；它们只会阻塞对同一行的写入者。

要在不实际修改某一行的情况下获取该行上的排他行级锁，可以使用`SELECT FOR UPDATE`选择该行。请注意，一旦获取了行级锁，事务就可以多次更新该行而不必担心冲突。

要获取某一行上的共享行级锁，可以使用`SELECT FOR SHARE`选择该行。共享锁不会阻止其他事务获取相同的共享锁。但是，任何事务都不允许更新、删除或排他锁定其他事务持有共享锁的行。任何这样的尝试都会被阻塞，直到共享锁被释放。

PostgreSQL 不会在内存中保存任何关于已修改行的信息，因此一次加锁的行数没有限制。不过，锁定一行可能会导致一次磁盘写；例如，`SELECT FOR UPDATE` 会修改被选中的行以标记它们已被锁定，因此会产生磁盘写入。

除了表级锁和行级锁之外，还使用页级共享/排他锁来控制对共享缓冲池中表页面的读写访问。这些锁会在取出或更新某一行后立即释放。应用开发者通常不需要关心页级锁，这里提到它们只是为了完整性。

13.3.3. 死锁

使用显式锁可能会增加死锁的发生概率。所谓死锁，是指两个（或更多）事务各自持有对方想要的锁。例如，如果事务 1 在表 A 上获得了排他锁，然后试图再获取表 B 上的排他锁，而事务 2 已经持有表 B 上的排他锁，现在又想获取表 A 上的排他锁，那么两者都无法继续进行。PostgreSQL 会自动检测死锁，并通过中止其中一个事务来解决这个问题，使其他事务得以继续完成。（具体会中止哪个事务很难预测，也不应依赖这种预测。）

还要注意，死锁也可能由于行级锁而发生（因此，即使没有使用显式锁，它们也可能出现）。考虑下面这种情况：两个并发事务都在修改一个表。第一个事务执行：

```
UPDATE accounts SET balance = balance + 100.00 WHERE acctnum =
 11111;
```

这会在指定账号的那一行上获取一个行级锁。然后第二个事务执行：

```
UPDATE accounts SET balance = balance + 100.00 WHERE acctnum =
 22222;
UPDATE accounts SET balance = balance - 100.00 WHERE acctnum =
 11111;
```

第一条 `UPDATE` 语句成功地在指定行上获取了一个行级锁，因此它成功更新了那一行。然而，第二条 `UPDATE` 语句发现它试图更新的行已经被锁住了，于是它等待持有该锁的事务结束。此时，事务二正在等待事务一结束之后才能继续执行。现在，事务一执行：

```
UPDATE accounts SET balance = balance - 100.00 WHERE acctnum =
 22222;
```

事务一试图在指定行上获取一个行级锁，但它做不到，因为事务二已经持有了这个锁。于是它必须等待事务二完成。这样一来，事务一被事务二阻塞，而事务二又被事务一阻塞：死锁条件形成了。PostgreSQL 会检测到这种情况，并中止其中一个事务。

防止死锁的最好办法通常是避免它们出现，也就是确保所有使用同一数据库的应用都以一致的顺序在多个对象上获取锁。

在上面的例子中，如果两个事务都是按相同顺序更新这些行，就不会发生死锁。
还应确保事务在某个对象上获取的第一个锁，就是该对象所需的最严格锁模式。
如果事先无法验证这一点，那么就应通过重试因死锁而中止的事务来动态处理死锁。

只要没有检测到死锁，寻求表级锁或行级锁的事务就会无限期等待冲突锁被释放。
这意味着让应用长时间保持事务打开并不是好主意（例如等待用户输入时）。

13.3.4. 咨询锁

PostgreSQL 提供了一种创建由应用自行定义含义的锁的方法。这类锁被称为咨询锁，因为系统并不强制要求使用它们 – 是否正确使用完全取决于应用。咨询锁对于那些与 MVCC 模型不太契合的加锁策略非常有用。例如，咨询锁的一个常见用途是模拟所谓“平面文件”数据管理系统中典型的悲观锁策略。
虽然也可以通过在表中存储一个标志位来达到相同目的，但咨询锁速度更快，
可以避免表膨胀，并且会在会话结束时由服务器自动清理。

在 PostgreSQL 中获取咨询锁有两种方式：会话级和事务级。会话级咨询锁一旦获取，就会一直保持到显式释放或会话结束。不同于标准锁请求，
会话级咨询锁请求不遵守事务语义：在随后回滚的事务中获取的锁，回滚后仍会保持；
同样，即使发出解锁请求的事务后来失败，解锁依然有效。
一个进程可以多次获取同一个锁；每成功一次加锁请求，都必须有对应的解锁请求，
锁才会真正释放。另一方面，
事务级锁请求更像普通锁请求：它们会在事务结束时自动释放，而且没有显式解锁操作。
对于短期使用咨询锁，这种行为通常比会话级行为更方便。
针对同一咨询锁标识符的会话级和事务级锁请求，会按预期相互阻塞。
如果某个会话已经持有给定的咨询锁，那么它发出的附加请求总会成功，
即使其他会话正在等待该锁；无论现有锁和新请求属于会话级还是事务级，都是如此。

和 PostgreSQL 中的所有锁一样，任何会话当前持有的咨询锁完整列表，都可以在`pg_locks`系统视图中找到。

咨询锁和普通锁都存储在一个共享内存池中，其大小由配置变量max_locks_per_transaction和max_connections定义。必须小心不要耗尽这部分内存，
否则服务器将根本无法再授予任何锁。这也为服务器可授予的咨询锁数量设定了上限，
具体通常在数万到数十万个之间，取决于服务器的配置。

在某些使用咨询锁的方法中，特别是涉及显式排序和 `LIMIT` 子句的查询，必须注意控制由于 SQL 表达式求值顺序而获取的锁。例如：

```
SELECT pg_advisory_lock(id) FROM foo WHERE id = 12345; -- 可以
SELECT pg_advisory_lock(id) FROM foo WHERE id > 12345 LIMIT 100; --
 危险!
SELECT pg_advisory_lock(q.id) FROM
(
  SELECT id FROM foo WHERE id > 12345 LIMIT 100
) q; -- 可以
```

在上面的查询中，第二种形式是危险的，因为不能保证在执行加锁函数之前先应用 `LIMIT`。
这可能导致获取了一些应用并未预期的锁，因此应用也不会去释放它们（直到会话结束）。
从应用的角度看，这类锁会变成悬而未清的锁，尽管仍然可以在 `pg_locks` 中看到。

用于操作咨询锁的函数见第 9.26.8 节。

13.4. 应用级别的数据一致性检查

用读已提交事务强制执行有关数据完整性的业务规则非常困难，
因为数据视图会随每条语句而变化，而且一旦发生写冲突，
即使是单条语句也未必局限于该语句自己的快照。

虽然可重复读事务在整个执行期间都具有稳定的数据视图，但在使用 MVCC 快照进行数据一致性检查时，还存在一个微妙问题，即所谓的读/写冲突。如果一个事务写入数据，而并发事务试图读取相同的数据（无论是在写入之前还是之后），它都看不到另一个事务的工作。于是，读取者看起来就像是先执行的一方，而不管究竟是谁先启动，也不管谁先提交。如果事情只停留在这一步，就没有问题；但如果读取者还写入了数据，而这些数据又被另一个并发事务读取，那么现在就会有一个事务看起来像是在前面提到的任一事务之前执行。如果看起来最后执行的事务实际上最先提交，就很容易在事务执行顺序图中形成一个环。一旦出现这样的环，不借助额外机制，完整性检查就无法正确工作。

正如第 13.2.3 节中提到的，可串行化事务其实就是在可重复读事务的基础上，增加了对危险读/写冲突模式的非阻塞监控。当检测到某种模式可能在表面执行顺序中形成一个环时，其中一个相关事务就会被回滚，以打破这个环。

13.4.1. 用可串行化事务强制一致性

如果对所有写操作，以及所有需要一致数据视图的读操作都使用可串行化事务隔离级别，那么不需要额外工作就能确保一致性。从其他环境移植而来、按使用可串行化事务来保证一致性而编写的软件，在这方面应当能够在 PostgreSQL 中“正常工作”。

使用这种技术时，如果应用软件通过某个框架运行，并由该框架自动重试因串行化失败而回滚的事务，就可以避免给应用程序员带来不必要的负担。把 `default_transaction_isolation` 设置为 `serializable` 可能是个好主意。通过在触发器中检查事务隔离级别来采取某些措施，以确保不会因为疏忽或为了绕过完整性检查而使用其他事务隔离级别，也是明智的。

性能方面的建议见第 13.2.3 节。

> **警告**
>
> 利用可串行化事务提供的这一层完整性保护，尚未扩展到热备模式（第 25.5 节）。因此，使用热备的用户可能希望在主库上使用可重复读和显式锁定。

13.4.2. 使用显式阻塞锁强制一致性

当存在非可串行化写入时，要确保某一行当前仍然有效，并保护它不受并发更新影响，就必须使用 `SELECT FOR UPDATE`、`SELECT FOR SHARE` 或适当的 `LOCK TABLE` 语句。（`SELECT FOR UPDATE` 和 `SELECT FOR SHARE` 只锁定返回的行以防止并发更新，而 `LOCK TABLE` 会锁住整张表。）从其他环境向 PostgreSQL 迁移应用时，应当考虑这一点。

对于从其他环境迁移而来的用户，还需要注意：`SELECT FOR UPDATE` 并不能保证并发事务不会更新或删除被选中的行。要在 PostgreSQL 中做到这一点，必须真正去更新该行，即使没有任何值需要改变。`SELECT FOR UPDATE` 只是临时阻塞其他事务，使它们不能获取同样的锁，也不能执行会影响被锁定行的 `UPDATE` 或 `DELETE`；但是一旦持有该锁的事务提交或回滚，被阻塞的事务就会继续执行冲突操作，除非在持锁期间已经对该行执行了实际的 `UPDATE`。

在非可串行化的 MVCC 环境下，全局有效性检查需要额外考虑。例如，一个银行应用可能希望检查一个表中的贷方金额总和等于另一个表中的借方金额总和，而这两个表都在被活跃更新。在读已提交模式下，比较两个连续的 `SELECT sum(...)` 命令的结果并不可靠，因为第二个查询很可能会包含第一个查询没有统计到的事务提交结果。在单个可重复读事务中完成这两次求和，只能准确反映在该可重复读事务开始之前已提交事务的效果 – 但等到结果交付时，人们完全可能合理地怀疑这个答案是否仍然相关。如果可重复读事务本身在尝试进行一致性检查之前已经应用了某些更改，

那么这种检查的意义就更加值得商榷，
因为此时它包含了事务开始后的一部分而不是全部更改。在这种情况下，
谨慎的做法可能是锁定执行检查所需的所有表，以获得当前真实状态的无可争议图景。
`SHARE` 模式（或更高）的锁能够保证，在被锁定的表中，除了当前事务自身的更改之外，没有其他未提交更改。

注意，如果依赖显式锁定来防止并发更改，就应当使用读已提交模式，
或者在可重复读模式下小心地在执行查询之前先获取锁。
可重复读事务获得的锁能够保证没有其他修改该表的事务仍在运行，
但如果该事务看到的快照早于获取锁的时刻，
那么它看到的快照也可能早于表中某些现在已经提交的更改。
可重复读事务的快照实际上是在其第一条查询或数据修改命令（`SELECT`、`INSERT`、`UPDATE`、`DELETE`）开始时被冻结的，因此可以在快照冻结之前显式获取锁。

13.5. 注意事项

一些 DDL 命令（目前只有TRUNCATE和会重写表的ALTER TABLE形式）不是 MVCC 安全的。这意味着在截断或重写提交之后，如果并发事务使用的是在该 DDL 命令提交之前取得的快照，那么该表对它们来说会表现为空表。这只会对那些在 DDL 命令开始前没有访问过相关表的事务造成问题 – 任何在 DDL 命令开始前访问过该表的事务，都会持有至少一个 `ACCESS SHARE` 表锁，从而阻塞该 DDL 命令直到该事务完成。因此，这些命令不会在针对目标表的连续查询中造成明显的表内容不一致，但它们可能会导致目标表内容与数据库中其他表内容之间出现可见的不一致。

对可串行化事务隔离级别的支持尚未添加到热备复制目标（见第 25.5 节）。当前热备模式下支持的最严格隔离级别是可重复读。
虽然在主库上把所有永久数据库写操作都放在可串行化事务中执行，
能够确保所有备库最终达到一致状态，但在备库上运行的可重复读事务，
有时仍可能看到一种瞬态状态，而这种状态与主库上事务的任何串行执行都不一致。

13.6. 锁定和索引

尽管 PostgreSQL 为表数据提供了非阻塞的读写访问，但 PostgreSQL 当前实现的索引访问方法并不是每一种都能提供非阻塞的读写访问。各种索引类型的处理方式如下：

B-树、GiST 和 SP-GiST 索引

读写访问使用短期的页级共享/排他锁。每个索引行在被取出或插入之后，
锁都会立即释放。这些索引类型提供最高的并发性，并且不会产生死锁。

Hash 索引

读写访问使用的是 Hash 桶级共享/排他锁。锁会在整个 Hash 桶处理完成后释放。桶级锁比索引级锁具有更好的并发性，但也可能产生死锁，
因为它们的持有时间比一次索引操作更长。

GIN 索引

读写访问使用短期的页级共享/排他锁。每个索引行在被取出或插入之后，
锁都会立即释放。但要注意，插入一个使用 GIN 索引的值通常会导致每行产生多个索引键插入，因此 GIN 可能为了插入单个值而做大量工作。

目前，B-树索引为并发应用提供了最佳性能；由于它们还比 Hash 索引拥有更多特性，因此对于需要为标量数据建立索引的并发应用，推荐使用 B-树索引类型。处理非标量数据时，B-树就不再适用，此时应改用 GiST、SP-GiST 或 GIN 索引。

第 14 章 性能提示

查询性能可能受许多因素影响。其中一些因素可以由用户控制，另一些则属于系统底层设计的基本特性。本章提供一些帮助理解和调优PostgreSQL性能的提示。

14.1. 使用EXPLAIN

PostgreSQL会为收到的每个查询制定一个查询计划。选择与查询结构和数据特性相匹配的正确计划，对获得良好的性能至关重要，因此系统内置了一个复杂的规划器来尽量选出好的计划。可以使用EXPLAIN命令查看规划器为任意查询生成的查询计划。读懂计划是一门需要经验积累的技能，本节将尝试介绍其中的基础知识。

本节中的示例取自执行过VACUUM ANALYZE的回归测试数据库，使用的是 9.3 开发版本源码。如果自行尝试这些示例，通常应当能得到相近的结果，但估计代价和行计数可能略有差异，因为ANALYZE生成的统计信息来自随机采样而非精确计数，而且代价本身在某种程度上也依赖于平台。

这些示例使用EXPLAIN默认的“text”输出格式，它紧凑且便于人工阅读。如果希望将EXPLAIN的输出交给程序做进一步分析，则应改用机器可读的输出格式（XML、JSON 或 YAML）。

14.1.1. EXPLAIN基础

查询计划的结构是一棵由计划节点组成的树。树的最底层是扫描节点，它们从表中返回原始行。不同的表访问方法对应不同类型的扫描节点，例如顺序扫描、索引扫描和位图索引扫描。也有一些并非表的行来源，例如VALUES子句和FROM中的返回集合函数，它们也各自有对应的扫描节点类型。如果查询需要对原始行执行连接、聚合、排序或其他操作，那么扫描节点之上还会出现额外的节点来完成这些操作。同样，这些操作通常不止一种实现方式，因此这里也会出现不同的节点类型。EXPLAIN会为计划树中的每个节点输出一行，显示基本节点类型以及规划器对该计划节点执行代价的估计值。还可能出现相对节点摘要行缩进的附加行，用来显示该节点的更多属性。第一行，也就是最顶层节点的摘要行，给出了整个计划的估计总执行代价；规划器力图最小化的正是这个数字。

下面是一个简单的示例，仅用于展示输出的样子：

```
EXPLAIN SELECT * FROM tenk1;

                          QUERY PLAN
-------------------------------------------------------------
 Seq Scan on tenk1  (cost=0.00..458.00 rows=10000 width=244)
```

由于这个查询没有WHERE子句，它必须扫描表中的所有行，因此规划器选择了一个简单的顺序扫描计划。圆括号中的数字从左到右依次表示：

- 估计启动开销。这是输出阶段开始前需要消耗的时间，例如排序节点执行排序所需的时间。
- 估计总开销。这里假定该计划节点会运行到结束，也就是取回所有可用的行。实际中某个节点的父节点可能会在尚未读完所有可用行之前提前停止（见后文的LIMIT示例）。
- 该计划节点输出行数的估计值。同样，也是假定该节点会运行到结束。

- 预计该计划节点输出行的平均宽度，以字节计。

这些开销使用由规划器代价参数决定的任意单位来衡量（见第 18.7.2 节）。传统上通常以磁盘页读取作为代价单位；也就是说，惯例上将seq_page_cost设为1.0，
其他代价参数都相对它来设定。本节中的示例都使用默认代价参数。

需要理解的一点是，上层节点的开销包含了其所有子节点的开销。还要注意，
这个开销只反映规划器关心的内容。特别是，
它没有考虑将输出值转换为文本形式或传输给客户端所消耗的时间，
而这些在实际耗时中可能是重要因素；但规划器会忽略这些代价，
因为它无法通过改变计划来影响它们。（我们相信每个正确的计划都会输出相同的行集。）

rows值有些容易误解，因为它不是计划节点处理或扫描过的行数，而是该节点输出的行数。
由于在该节点上应用的WHERE条件会过滤掉一部分扫描到的行，
这个数字通常小于扫描行数。理想情况下，顶层的行数估计应当接近查询实际返回、
更新或删除的行数。

回到我们的示例：

```
EXPLAIN SELECT * FROM tenk1;

                          QUERY PLAN
-------------------------------------------------------------
 Seq Scan on tenk1  (cost=0.00..458.00 rows=10000 width=244)
```

这些数字的来源非常直接。如果执行：

```
SELECT relpages, reltuples FROM pg_class WHERE relname = 'tenk1';
```

可以看到tenk1有 358 个磁盘页和 10000 行。估计代价按如下公式计算：（读取的磁盘页数 * seq_page_cost）+（扫描的行数 * cpu_tuple_cost）。默认情况下，seq_page_cost为 1.0，cpu_tuple_cost为 0.01，因此估计代价就是 (358 * 1.0) + (10000 * 0.01) = 458。

现在修改查询，添加一个WHERE条件：

```
EXPLAIN SELECT * FROM tenk1 WHERE unique1 < 7000;

                          QUERY PLAN
------------------------------------------------------------
 Seq Scan on tenk1  (cost=0.00..483.00 rows=7001 width=244)
   Filter: (unique1 < 7000)
```

注意，EXPLAIN的输出表明，WHERE子句被作为一个“过滤器”条件附加到 Seq Scan 计划节点上。这表示计划节点会对扫描到的每一行检查该条件，并且只输出满足条件的行。
由于有了WHERE子句，估计输出行数减少了。但扫描仍需访问全部 10000 行，因此代价没有下降；实际上还略有上升（精确地说，增加了 10000 *cpu_operator_cost），以反映检查WHERE条件所消耗的额外 CPU 时间。

这条查询实际选出的行数是 7000，但估计的rows只是近似值。如果重复这个实验，很可能会得到略有不同的估计值。此外，由于ANALYZE生成的统计信息来自该表的随机采样，
这个估计值可能在每次执行ANALYZE之后发生变化。

现在让条件更严格一些：

```
EXPLAIN SELECT * FROM tenk1 WHERE unique1 < 100;

                                  QUERY PLAN
-------------------------------------------------------------------
-----------
```

```
 Bitmap Heap Scan on tenk1  (cost=5.03..229.17 rows=101 width=244)
   Recheck Cond: (unique1 < 100)
   ->  Bitmap Index Scan on tenk1_unique1  (cost=0.00..5.01 rows=
101 width=0)
         Index Cond: (unique1 < 100)
```

这里规划器决定使用两步计划：子计划节点访问索引，找出匹配索引条件的行的位置，然后上层计划节点再从表本身取出这些行。分别取出各行的代价远高于顺序读取，但由于不必访问表的所有页，这仍然比顺序扫描便宜。（使用两层计划的原因是，上层计划节点在读取行之前，会先把索引确定的行位置按物理顺序排列，以尽量降低逐行读取的代价。节点名称中的“位图”就是执行这种排序的机制。）

现在向WHERE子句添加另一个条件：

```
EXPLAIN SELECT * FROM tenk1 WHERE unique1 < 100 AND stringu1 =
 'xxx';

                                  QUERY PLAN
-----------------------------------------------------------------
-----------
 Bitmap Heap Scan on tenk1  (cost=5.01..229.40 rows=1 width=244)
   Recheck Cond: (unique1 < 100)
   Filter: (stringu1 = 'xxx'::name)
   ->  Bitmap Index Scan on tenk1_unique1  (cost=0.00..5.01 rows=
101 width=0)
         Index Cond: (unique1 < 100)
```

新增的条件stringu1 = 'xxx'降低了估计输出行数，却没有降低代价，因为仍然必须访问同一组行。注意，stringu1子句不能用作索引条件，因为该索引只建立在unique1列上。它只能作为过滤条件，应用到通过索引取出的行上。因此，代价实际上略有增加，以反映这项额外检查。

在某些情况下，规划器更倾向于一个“简单”索引扫描计划：

```
EXPLAIN SELECT * FROM tenk1 WHERE unique1 = 42;

                                 QUERY PLAN
-----------------------------------------------------------------
----------
 Index Scan using tenk1_unique1 on tenk1  (cost=0.00..8.27 rows=1
 width=244)
   Index Cond: (unique1 = 42)
```

在这种计划中，表行按索引顺序取出，因此读取代价更高，但行数很少，不值得为排序行位置付出额外代价。对于仅取出一行的查询，最常见到这种计划。它也经常用于带有以下条件的查询：ORDER BY与索引顺序匹配，因为这时无需额外排序步骤就能满足ORDER BY。

如果在WHERE引用的多个列上建有索引，规划器可能会选择将这些索引做 AND 或 OR 组合使用：

```
EXPLAIN SELECT * FROM tenk1 WHERE unique1 < 100 AND unique2 > 9000;

                                     QUERY PLAN
-----------------------------------------------------------------
------------------
 Bitmap Heap Scan on tenk1  (cost=25.01..60.14 rows=10 width=244)
   Recheck Cond: ((unique1 < 100) AND (unique2 > 9000))
   ->  BitmapAnd  (cost=25.01..25.01 rows=10 width=0)
```

```
        ->  Bitmap Index Scan on tenk1_unique1  (cost=0.00..5.01
 rows=101 width=0)
              Index Cond: (unique1 < 100)
        ->  Bitmap Index Scan on tenk1_unique2  (cost=0.00..19.74
 rows=999 width=0)
              Index Cond: (unique2 > 9000)
```

但这需要访问两个索引，因此与只使用一个索引、把另一个条件作为过滤条件相比，并不一定更划算。如果调整条件中的范围，就会看到计划相应改变。

下面用一个示例说明LIMIT：

```
EXPLAIN SELECT * FROM tenk1 WHERE unique1 < 100 AND unique2 > 9000
 LIMIT 2;

                                     QUERY PLAN
-----------------------------------------------------------------
------------------
 Limit  (cost=0.00..14.25 rows=2 width=244)
   ->  Index Scan using tenk1_unique2 on tenk1  (cost=0.00..71.23
 rows=10 width=244)
        Index Cond: (unique2 > 9000)
        Filter: (unique1 < 100)
```

这与上面的查询相同，只是加上了LIMIT，因此不必检索全部行，规划器也就改变了选择。注意，Index Scan 节点的总代价和行计数显示得像是它会运行到结束一样；但 Limit 节点预计在只取到其中五分之一的行后就会停止，因此它的总代价也只有前者的五分之一，这才是该查询真正的估计代价。之所以更偏好这个计划，而不是在前一个计划之上再加一个 Limit 节点，是因为后者仍然无法避免位图扫描的启动代价，那样总代价仍会高于 25。

使用之前讨论的列，尝试连接两个表：

```
EXPLAIN SELECT *
FROM tenk1 t1, tenk2 t2
WHERE t1.unique1 < 10 AND t1.unique2 = t2.unique2;

                                      QUERY PLAN
-------------------------------------------------------------------
-------------------
 Nested Loop  (cost=4.33..118.25 rows=10 width=488)
   ->  Bitmap Heap Scan on tenk1 t1  (cost=4.33..39.44 rows=10
 width=244)
        Recheck Cond: (unique1 < 10)
        ->  Bitmap Index Scan on tenk1_unique1  (cost=0.00..4.33
 rows=10 width=0)
              Index Cond: (unique1 < 10)
   ->  Index Scan using tenk2_unique2 on tenk2 t2  (cost=0.00..7.87
 rows=1 width=244)
        Index Cond: (unique2 = t1.unique2)
```

在这个计划中，有一个嵌套循环连接节点，它的两个输入，也就是两个子节点，都是表扫描。节点摘要行的缩进反映了计划树结构。连接的第一个子节点，也就是“外侧”子节点，是一个与前面见过的位图扫描类似的节点。它的代价和行计数与SELECT ... WHERE unique1 < 10得到的结果相同，因为WHERE子句unique1 < 10正是在该节点上应用的。t1.unique2 = t2.unique2子句此时还无关，因此不会影响外侧扫描的行计数。嵌套循环连接节点会对从外侧子节点得到的每一行执行一次第二个，也就是“内侧”子节点。当前外侧行中的列值可以代入内侧扫描；这里外侧行的t1.unique2值可用，因此得到的计划和代价与前面看到的简单SELECT ... WHERE t2.unique2 =

*constant*情形类似。（由于预期在对t2反复执行索引扫描期间会发生缓存命中，估计代价实际上比前面看到的略低一些。）随后，循环节点的代价建立在外侧扫描代价之上，再加上每个外侧行都要执行一次内侧扫描的代价（这里是 10 * 7.91），以及少量连接处理的 CPU 时间。

本例中，连接的输出行数等于两个扫描行数的乘积，但并非所有情况都如此，因为可能还有额外的WHERE子句同时涉及两个表，因此只能在连接处应用，不能应用到任一输入扫描。下面是一个示例：

```
EXPLAIN SELECT *
FROM tenk1 t1, tenk2 t2
WHERE t1.unique1 < 10 AND t1.unique2 = t2.unique2 AND t1.hundred <
 t2.hundred;

                                       QUERY PLAN
-----------------------------------------------------------------
-------------------
 Nested Loop  (cost=4.33..118.28 rows=3 width=488)
   Join Filter: (t1.hundred < t2.hundred)
   ->  Bitmap Heap Scan on tenk1 t1  (cost=4.33..39.44 rows=10
 width=244)
         Recheck Cond: (unique1 < 10)
         ->  Bitmap Index Scan on tenk1_unique1  (cost=0.00..4.33
 rows=10 width=0)
               Index Cond: (unique1 < 10)
   ->  Index Scan using tenk2_unique2 on tenk2 t2  (cost=0.00..7.87
 rows=1 width=244)
         Index Cond: (unique2 = t1.unique2)
```

条件t1.hundred < t2.hundred不能在tenk2_unique2索引中检验，因此应用在连接节点上。这会减少连接节点的估计输出行数，但不改变任一输入扫描。

在处理外连接时，可能会看到连接计划节点同时带有 “Join Filter” 和普通 “Filter” 条件。Join Filter 条件来自外连接的ON子句，因此某一行即使未通过 Join Filter，仍可能作为一条补齐空值的行被输出。但普通 Filter 条件是在外连接规则应用之后再执行的，因此会无条件移除行。在内连接中，这两类过滤条件在语义上没有区别。

如果稍微改变查询的选择率，可能得到完全不同的连接计划：

```
EXPLAIN SELECT *
FROM tenk1 t1, tenk2 t2
WHERE t1.unique1 < 100 AND t1.unique2 = t2.unique2;

                                         QUERY PLAN
-----------------------------------------------------------------
-----------------------
 Hash Join  (cost=230.43..713.94 rows=101 width=488)
   Hash Cond: (t2.unique2 = t1.unique2)
   ->  Seq Scan on tenk2 t2  (cost=0.00..445.00 rows=10000 width=
244)
   ->  Hash  (cost=229.17..229.17 rows=101 width=244)
         ->  Bitmap Heap Scan on tenk1 t1  (cost=5.03..229.17 rows=
101 width=244)
               Recheck Cond: (unique1 < 100)
               ->  Bitmap Index Scan on tenk1_unique1  (cost=0.00..
5.01 rows=101 width=0)
                     Index Cond: (unique1 < 100)
```

这里规划器选择了哈希连接：先把一个表的行放入内存中的哈希表，然后扫描另一个表，并对其中每一行到哈希表中查找匹配。再次注意缩进如何反映计划结构：tenk1

上的位图扫描是 Hash 节点的输入，Hash 节点据此构造哈希表；随后该哈希表被返回给 Hash Join 节点，后者从其外侧子计划读取行，并对每一行在哈希表中进行查找。

另一种可能的连接类型是归并连接，如下所示：

```
EXPLAIN SELECT *
FROM tenk1 t1, onek t2
WHERE t1.unique1 < 100 AND t1.unique2 = t2.unique2;

                                        QUERY PLAN
-------------------------------------------------------------------
-----------------------
 Merge Join  (cost=197.83..267.93 rows=10 width=488)
   Merge Cond: (t1.unique2 = t2.unique2)
   ->  Index Scan using tenk1_unique2 on tenk1 t1  (cost=0.00..656.
25 rows=101 width=244)
         Filter: (unique1 < 100)
   ->  Sort  (cost=197.83..200.33 rows=1000 width=244)
         Sort Key: t2.unique2
         ->  Seq Scan on onek t2  (cost=0.00..148.00 rows=1000
 width=244)
```

归并连接要求输入数据按连接键排序。在这个计划中，tenk1 的数据通过索引扫描按正确顺序访问行来完成排序，而 onek 则更适合顺序扫描后再排序，因为需要访问该表中的更多行。（对大量行进行排序时，顺序扫描加排序往往优于索引扫描，
因为索引扫描需要非顺序的磁盘访问。）

查看其他候选计划的一种方法，是使用第 18.7.1 节中介绍的启用/禁用标志，强制规划器忽略它认为代价最低的策略。（这是一种粗略但有用的工具。另请参见第 14.3 节。）例如，如果不确信顺序扫描加排序是处理上例中onek表的最佳方法，可以尝试：

```
SET enable_sort = off;

EXPLAIN SELECT *
FROM tenk1 t1, onek t2
WHERE t1.unique1 < 100 AND t1.unique2 = t2.unique2;

                                        QUERY PLAN
-------------------------------------------------------------------
-----------------------
 Merge Join  (cost=0.00..292.36 rows=10 width=488)
   Merge Cond: (t1.unique2 = t2.unique2)
   ->  Index Scan using tenk1_unique2 on tenk1 t1  (cost=0.00..656.
25 rows=101 width=244)
         Filter: (unique1 < 100)
   ->  Index Scan using onek_unique2 on onek t2  (cost=0.00..224.76
 rows=1000 width=244)
```

这表明规划器认为，通过索引扫描来排序onek，代价比顺序扫描加排序大约高 12%。当然，接下来的问题是它的判断是否正确。可以使用EXPLAIN ANALYZE来调查，下文将详细说明。

14.1.2. EXPLAIN ANALYZE

可以使用EXPLAIN的ANALYZE选项检查规划器估计的准确性。启用此选项后，EXPLAIN 会实际执行查询，然后显示各个计划节点内累计的实际行数和实际运行时间，
同时也显示普通EXPLAIN提供的估计值。例如，可能得到如下结果：

```
EXPLAIN ANALYZE SELECT *
```

```
FROM tenk1 t1, tenk2 t2
WHERE t1.unique1 < 10 AND t1.unique2 = t2.unique2;

                                                                QUERY
 PLAN
-----------------------------------------------------------------------
-------------------------------------------------------------
 Nested Loop  (cost=4.33..118.25 rows=10 width=488) (actual time=0.
370..1.126 rows=10 loops=1)
   ->  Bitmap Heap Scan on tenk1 t1  (cost=4.33..39.44 rows=10
 width=244) (actual time=0.254..0.380 rows=10 loops=1)
         Recheck Cond: (unique1 < 10)
         ->  Bitmap Index Scan on tenk1_unique1  (cost=0.00..4.33
 rows=10 width=0) (actual time=0.164..0.164 rows=10 loops=1)
               Index Cond: (unique1 < 10)
   ->  Index Scan using tenk2_unique2 on tenk2 t2  (cost=0.00..7.87
 rows=1 width=244) (actual time=0.041..0.048 rows=1 loops=10)
         Index Cond: (unique2 = t1.unique2)
 Total runtime: 2.414 ms
```

注意，“actual time”的值以实际时间的毫秒数表示，而cost估计值使用任意单位，因此两者不太可能相等。通常最重要的是检查估计行数是否足够接近实际行数。在本例中，所有估计都完全准确，但实际中很少如此。

在某些查询计划中，一个子计划节点可能会执行多次。例如，上面那个嵌套循环计划中的内侧索引扫描会对外侧的每一行执行一次。在这种情况下，loops值报告的是该节点的总执行次数，而 actual time 和 rows 显示的是每次执行的平均值。这样做是为了让这些数字更容易与代价估计的展示方式相比较。将它们乘以loops值，就能得到该节点实际消耗的总时间。在上面的示例中，执行tenk2上的索引扫描总共花费了0.220 毫秒。

在某些情况下，EXPLAIN ANALYZE除了计划节点的执行时间和行数之外，还会显示额外的执行统计信息。例如，Sort 和 Hash 节点会提供附加信息：

```
EXPLAIN ANALYZE SELECT *
FROM tenk1 t1, tenk2 t2
WHERE t1.unique1 < 100 AND t1.unique2 = t2.unique2 ORDER BY t1.
fivethous;


 QUERY PLAN
-----------------------------------------------------------------------
-----------------------------------------------------------------------
------
 Sort  (cost=717.30..717.56 rows=101 width=488) (actual time=104.
950..105.327 rows=100 loops=1)
   Sort Key: t1.fivethous
   Sort Method: quicksort  Memory: 68kB
   ->  Hash Join  (cost=230.43..713.94 rows=101 width=488) (actual
 time=3.680..102.396 rows=100 loops=1)
         Hash Cond: (t2.unique2 = t1.unique2)
         ->  Seq Scan on tenk2 t2  (cost=0.00..445.00 rows=10000
 width=244) (actual time=0.046..46.219 rows=10000 loops=1)
         ->  Hash  (cost=229.17..229.17 rows=101 width=244) (actual
 time=3.184..3.184 rows=100 loops=1)
               Buckets: 1024  Batches: 1  Memory Usage: 27kB
               ->  Bitmap Heap Scan on tenk1 t1  (cost=5.03..229.17
 rows=101 width=244) (actual time=0.612..1.959 rows=100 loops=1)
                     Recheck Cond: (unique1 < 100)
```

```
                     ->  Bitmap Index Scan on tenk1_unique1  (cost=
0.00..5.01 rows=101 width=0) (actual time=0.390..0.390 rows=100
 loops=1)
                           Index Cond: (unique1 < 100)
 Total runtime: 107.392 ms
```

Sort 节点显示所用的排序方法（尤其是排序在内存中还是磁盘上进行），以及所需的内存或磁盘空间。Hash 节点显示哈希桶数、批次数，以及哈希表的内存用量峰值。（如果批次数超过一，还会使用磁盘空间，但此处不显示。）

另一种附加信息是被过滤条件排除的行数：

```
EXPLAIN ANALYZE SELECT * FROM tenk1 WHERE ten < 7;

                                                   QUERY PLAN
-------------------------------------------------------------------
---------------------------------------
 Seq Scan on tenk1  (cost=0.00..483.00 rows=7000 width=244) (actual
 time=0.111..59.249 rows=7000 loops=1)
   Filter: (ten < 7)
   Rows Removed by Filter: 3000
 Total runtime: 85.340 ms
```

对于应用在连接节点上的过滤条件，这些计数尤其有价值。
只有当至少一个扫描行（对于连接节点，则是一个潜在连接行对）被过滤条件排除时，
才会出现“Rows Removed”这一行。

在“有损”索引扫描中，也会出现类似于过滤条件的情况。例如，
考虑下面这个查找包含指定点的多边形的查询：

```
EXPLAIN ANALYZE SELECT * FROM polygon_tbl WHERE f1 @> polygon '(0.
5,2.0)';

                                                 QUERY PLAN
-------------------------------------------------------------------
-----------------------------------
 Seq Scan on polygon_tbl  (cost=0.00..1.05 rows=1 width=32) (actual
 time=0.251..0.251 rows=0 loops=1)
   Filter: (f1 @> '((0.5,2))'::polygon)
   Rows Removed by Filter: 4
 Total runtime: 0.517 ms
```

规划器认为（而且完全正确）这个示例表太小，不值得使用索引扫描，
因此得到的是普通顺序扫描，其中所有行都被过滤条件排除。但如果强制使用索引扫描，
就会看到：

```
SET enable_seqscan TO off;

EXPLAIN ANALYZE SELECT * FROM polygon_tbl WHERE f1 @> polygon '(0.
5,2.0)';

                                                       QUERY PLAN
-------------------------------------------------------------------
-------------------------------------------------------
 Index Scan using gpolygonind on polygon_tbl  (cost=0.00..8.27
 rows=1 width=32) (actual time=0.293..0.293 rows=0 loops=1)
   Index Cond: (f1 @> '((0.5,2))'::polygon)
   Rows Removed by Index Recheck: 1
 Total runtime: 1.054 ms
```

这里可以看到，索引返回了一个候选行，但随后在重新检查索引条件时被排除。这是因为 GiST 索引对于多边形包含关系测试是“有损”的：它实际返回的是包含与目标重叠的多边形的行，然后必须对这些行进行精确的包含关系测试。

EXPLAIN有一个BUFFERS选项，可以与ANALYZE一起使用，以获取更多运行时统计信息：

```
EXPLAIN (ANALYZE, BUFFERS) SELECT * FROM tenk1 WHERE unique1 < 100
 AND unique2 > 9000;

                                                              QUERY
 PLAN
-----------------------------------------------------------------
-------------------------------------------------------------
 Bitmap Heap Scan on tenk1  (cost=25.07..60.23 rows=10 width=244) (
actual time=3.069..3.213 rows=10 loops=1)
   Recheck Cond: ((unique1 < 100) AND (unique2 > 9000))
   Buffers: shared hit=16
   ->  BitmapAnd  (cost=25.07..25.07 rows=10 width=0) (actual time=
2.967..2.967 rows=0 loops=1)
         Buffers: shared hit=7
         ->  Bitmap Index Scan on tenk1_unique1  (cost=0.00..5.02
 rows=102 width=0) (actual time=0.732..0.732 rows=200 loops=1)
               Index Cond: (unique1 < 100)
               Buffers: shared hit=2
         ->  Bitmap Index Scan on tenk1_unique2  (cost=0.00..19.80
 rows=1007 width=0) (actual time=2.015..2.015 rows=1009 loops=1)
               Index Cond: (unique2 > 9000)
               Buffers: shared hit=5
 Total runtime: 3.917 ms
```

由BUFFERS提供的数值有助于找出查询中 I/O 最密集的部分。

请记住，由于EXPLAIN ANALYZE会实际执行查询，任何副作用都会照常发生，即使查询原本输出的结果被丢弃，改为打印EXPLAIN数据。如果想分析会修改数据的查询而不改变表，可以在之后回滚命令，例如：

```
BEGIN;

EXPLAIN ANALYZE UPDATE tenk1 SET hundred = hundred + 1 WHERE
 unique1 < 100;

                                                              QUERY
 PLAN
-----------------------------------------------------------------
-------------------------------------------------------------
 Update on tenk1  (cost=5.03..229.42 rows=101 width=250) (actual
 time=81.055..81.055 rows=0 loops=1)
   ->  Bitmap Heap Scan on tenk1  (cost=5.03..229.42 rows=101
 width=250) (actual time=0.766..3.396 rows=100 loops=1)
         Recheck Cond: (unique1 < 100)
         ->  Bitmap Index Scan on tenk1_unique1  (cost=0.00..5.01
 rows=101 width=0) (actual time=0.461..0.461 rows=100 loops=1)
               Index Cond: (unique1 < 100)
 Total runtime: 81.922 ms

ROLLBACK;
```

如本例所示，当查询是INSERT、UPDATE或DELETE命令时，真正执行表修改的工作由顶层的 Insert、Update或 Delete 计划节点完成。该节点下面的计划节点负责定位旧行和/或计算新

数据。因此，上面看到的是前文已经介绍过的同类位图表扫描，
它的输出被送入一个负责存储更新后行的 Update 节点。值得注意的是，虽然数据修改节点可能占用相当可观的运行时间（这里它消耗了大部分时间），
但规划器目前不会为这部分工作额外增加任何代价估计。
这是因为这项工作对每个正确的查询计划都相同，因此不会影响规划决策。

EXPLAIN ANALYZE显示的Total runtime包括执行器启动和关闭所需的时间，以及运行所有已触发触发器所花费的时间，但它不包括解析、重写或规划时间。若存在BEFORE触发器，其执行时间会计入相关的 Insert、Update 或 Delete 节点；而AFTER触发器的时间不会计入那里，因为AFTER触发器是在整个计划完成之后才触发的。每个触发器，无论是BEFORE还是AFTER，其总耗时也会单独显示。注意，延迟约束触发器要到事务结束时才会执行，
因此EXPLAIN ANALYZE完全不会把它们计入其中。

14.1.3. 注意事项

EXPLAIN ANALYZE测得的运行时间，可能以两种重要方式偏离同一查询的正常执行时间。第一，由于不会向客户端传递任何输出行，因此网络传输开销和 I/O 转换开销均不会被计入。第二，EXPLAIN ANALYZE附加的测量开销本身可能相当可观，尤其是在操作系统调用gettimeofday()较慢的机器上。可以使用pg_test_timing工具来测量系统上的计时开销。

不应将EXPLAIN结果外推到与实际测试场景差异很大的情况。例如，
在一个很小的表上得到的结果，不能假定也适用于大型表。规划器的代价估计不是线性的，
因此它可能会为更大或更小的表选择不同的计划。
一个极端示例是：对于只占用一个磁盘页的表，无论索引是否可用，
几乎总会得到顺序扫描计划。规划器认识到，无论如何处理该表都需要一次磁盘页读取，
因此再额外读取页面去查看索引并没有价值。（前面的polygon_tbl
示例已经展示过这种情况。）

有时实际值与估计值不太一致，但实际上并没有问题。其中一种情况是，计划节点的执行因LIMIT或类似效果而提前停止。例如，在前面使用的LIMIT查询中：

```
EXPLAIN ANALYZE SELECT * FROM tenk1 WHERE unique1 < 100 AND unique2
 > 9000 LIMIT 2;

                                                            QUERY
 PLAN
-----------------------------------------------------------------
-------------------------------------------------------------
 Limit  (cost=0.00..14.25 rows=2 width=244) (actual time=1.652..2.
293 rows=2 loops=1)
   ->  Index Scan using tenk1_unique2 on tenk1  (cost=0.00..71.23
 rows=10 width=244) (actual time=1.631..2.259 rows=2 loops=1)
         Index Cond: (unique2 > 9000)
         Filter: (unique1 < 100)
         Rows Removed by Filter: 287
 Total runtime: 2.857 ms
```

Index Scan 节点的估计代价和行数按该节点执行到完成的情况显示。但实际上，Limit 节点取得两行后就停止请求行，因此实际行数只有 2，运行时间也小于估计代价所暗示的时间。这不是估计错误，只是估计值与实际值的显示方式存在差别。

归并连接也会产生一些容易误导人的计量现象。如果归并连接已经耗尽其中一个输入，
而另一个输入中的下一个键值又大于前一个输入的最后一个键值，
那么它就会停止继续读取前者；在这种情况下，不可能再有更多匹配，
因此也就不需要扫描另一个输入的剩余部分。这会导致某个子节点没有被完整读取，
其结果与前面提到的LIMIT情况类似。此外，如果外侧（第一个）
子节点包含带有重复键值的行，内侧（第二个）子节点会被回退并重新扫描，
以查找能够匹配该键值的行。EXPLAIN ANALYZE会把这些对同一内侧行的重复输出，统计得

像是真实的额外行一样。当外侧存在大量重复值时，内侧子计划节点报告出的实际行计数，可能会明显大于内侧关系中真实存在的行数。

由于实现上的限制，BitmapAnd 和 BitmapOr 节点总是报告其实际行计数为零。

14.2. 规划器使用的统计信息

如上一节所见，查询规划器需要估计查询将检索多少行，才能对查询计划做出良好选择。本节简要介绍系统用于这些估计的统计信息。

统计信息的一部分是各个表和索引的条目总数，以及各个表和索引占用的磁盘块数。这些信息保存在表pg_class的reltuples和relpages列中。可以使用类似下面的查询来查看：

```
SELECT relname, relkind, reltuples, relpages
FROM pg_class
WHERE relname LIKE 'tenk1%';

       relname        | relkind | reltuples | relpages
----------------------+---------+-----------+----------
 tenk1                | r       |     10000 |      358
 tenk1_hundred        | i       |     10000 |       30
 tenk1_thous_tenthous | i       |     10000 |       30
 tenk1_unique1        | i       |     10000 |       30
 tenk1_unique2        | i       |     10000 |       30
(5 rows)
```

这里可以看到，tenk1包含 10000 行，它的索引也一样，但索引比表小得多（这并不意外）。

出于效率考虑，reltuples和relpages不会实时更新，因此它们通常包含有些过时的值。它们会在VACUUM、ANALYZE以及少数 DDL 命令（如CREATE INDEX）执行时被更新。不扫描全表的VACUUM或ANALYZE操作（这很常见）会根据其实际扫描到的那一部分增量更新reltuples计数，因此得到的是近似值。无论如何，规划器都会将它在pg_class中找到的值按当前物理表大小进行缩放，从而得到更接近实际情况的近似值。

大多数查询只会检索表中一部分行，因为它们通过WHERE子句限制了需要检查的行。因此，规划器需要估算WHERE子句的选择度，也就是满足WHERE子句中各个条件的行所占比例。完成这项任务所需的信息存储在pg_statistic系统目录中。pg_statistic中的条目由ANALYZE和VACUUM ANALYZE命令更新，而且即使刚更新完，也始终只是近似值。

手动检查统计信息时，与其直接查看pg_statistic，不如查看它的视图pg_stats。pg_stats旨在让统计信息更易读。此外，pg_stats所有用户都可以读取，而pg_statistic只有超级用户可以读取。（这可以防止没有相应权限的用户通过统计信息获知其他用户表中的内容。pg_stats视图被限制为只显示当前用户有权读取的表的相关行。）例如，可以执行：

```
SELECT attname, inherited, n_distinct,
       array_to_string(most_common_vals, E'\n') as most_common_vals
FROM pg_stats
WHERE tablename = 'road';

 attname | inherited | n_distinct |          most_common_vals
---------+-----------+------------+--------------------------------
----
```

```
 name     | f          |  -0.363388 | I- 580
 Ramp+
          |            |            | I- 880
 Ramp+
          |            |            | Sp Railroad
   +
          |            |            | I- 580
   +
          |            |            | I- 680
 Ramp
 name     | t          |  -0.284859 | I- 880
 Ramp+
          |            |            | I- 580
 Ramp+
          |            |            | I- 680
 Ramp+
          |            |            | I- 580
   +
          |            |            | State Hwy 13
 Ramp
(2 rows)
```

注意，同一列显示了两行，其中一行对应从road表开始的整个继承层次结构（inherited =t），另一行只包含road表本身（inherited=f）。

ANALYZE在pg_statistic中存储的信息量，特别是每列most_common_vals中的最大项数和histogram_bounds数组的大小，可以使用ALTER TABLE SET STATISTICS命令按列设置，也可以通过设置配置变量default_statistics_target进行全局设置。目前默认上限是 100 项。提高这一上限可能让规划器做出更准确的估计，尤其是对于数据分布不规则的列；代价则是pg_statistic占用更多空间，并且计算估计值所需时间也会略有增加。相反，对于数据分布较简单的列，较低的上限可能已经足够。

更多规划器对统计信息的使用可参阅第 58 章。

14.3. 用显式JOIN子句控制规划器

可以在一定程度上用显式JOIN语法控制查询规划器。要理解这一点为何重要，先需要一些背景知识。

在一个简单的连接查询中，例如：

```
SELECT * FROM a, b, c WHERE a.id = b.id AND b.ref = c.id;
```

规划器可以自由地按任意顺序连接这些表。例如，它可以生成一个查询计划，先利用WHERE条件a.id = b.id将 A 连接到 B，然后再利用另一个WHERE条件把 C 连接到这个结果上。也可以先连接 B 和 C，再把 A 连接到所得结果上。甚至还可以先连接 A 和 C，再与 B 连接，但这样效率会很低，因为必须先形成 A 和 C 的完整笛卡尔积，而WHERE子句中并没有可用于优化这一连接的条件。（PostgreSQL执行器中的所有连接都发生在两个输入表之间，因此结果必须以这些形式之一逐步构造出来。）关键在于，这些不同的连接可能性在语义上是等价的，但执行代价可能相差极大。因此，规划器会探索它们，力图找出最高效的查询计划。

当查询只涉及两个或三个表时，需要考虑的连接顺序并不多。但可能的连接顺序数量会随着表数增加而呈指数增长。输入表超过十个左右之后，对所有可能性做穷举搜索实际上就不再可行，甚至六七个表也可能让规划耗时长得令人厌烦。输入表过多时，PostgreSQL规划器会从穷举搜索切换到一种遗传概率搜索，只考虑有限数量的可能性。

（切换阈值由运行时参数geqo_threshold控制。）遗传搜索耗时更少，
但并不一定能找到最优计划。

当查询涉及外连接时，规划器比处理普通（内）连接时拥有更小的自由度。例如，考虑：

```
SELECT * FROM a LEFT JOIN (b JOIN c ON (b.ref = c.id)) ON (a.id =
 b.id);
```

尽管这个查询的约束表面上与前一个非常相似，但它们的语义不同，因为如果 A 中有某一行无法匹配 B 和 C 连接结果中的任何行，该行仍然必须被输出。因此这里规划器对连接顺序没有选择：它必须先连接 B 和 C，再把 A 连接到该结果上。相应地，这个查询比前一个查询需要更少的规划时间。在其他情况下，规划器可能会判断多种连接顺序都是安全的。例如：

```
SELECT * FROM a LEFT JOIN b ON (a.bid = b.id) LEFT JOIN c ON (a.cid
 = c.id);
```

将 A 首先连接到 B 或 C 都是有效的。当前，只有`FULL JOIN`完全约束连接顺序。大多数涉及`LEFT JOIN`或`RIGHT JOIN`的实际情况都在某种程度上可以被重新排列。

显式连接语法（`INNER JOIN`、`CROSS JOIN`或无修饰的`JOIN`）在语义上和`FROM`中列出输入关系是一样的，因此它不约束连接顺序。

即使大多数类型的`JOIN`并不会完全约束连接顺序，仍然可以指示
PostgreSQL查询规划器将所有`JOIN`子句都当作带有连接顺序约束来处理。例如，
下面三个查询在逻辑上是等价的：

```
SELECT * FROM a, b, c WHERE a.id = b.id AND b.ref = c.id;
SELECT * FROM a CROSS JOIN b CROSS JOIN c WHERE a.id = b.id AND b.
ref = c.id;
SELECT * FROM a JOIN (b JOIN c ON (b.ref = c.id)) ON (a.id = b.id);
```

但如果告诉规划器遵循`JOIN`顺序，
那么第二个和第三个查询在规划上会比第一个花费更少时间。对于只有三个表的连接来说，
这种效果微不足道；但当表很多时，它可能非常关键。

要强制规划器遵循显式`JOIN`给出的连接顺序，可以将运行时参数join_collapse_limit设置为1。（其他可能值见下文讨论。）

不必为了缩短搜索时间而完全约束连接顺序，因为可以在普通`FROM`列表的某一项中使用`JOIN`操作符。例如：

```
SELECT * FROM a CROSS JOIN b, c, d, e WHERE ...;
```

如果设置`join_collapse_limit = 1`，就会强制规划器先将 A 连接到 B，然后再与其他表连接，但不会进一步约束它的选择。在这个示例中，可能的连接顺序数量减少了 5 倍。

以这种方式约束规划器的搜索，是一种既可减少规划时间、
又可引导规划器生成更好查询计划的实用技巧。如果规划器默认选择了糟糕的连接顺序，
可以通过`JOIN`语法强制它采用更好的顺序，前提当然是确实知道哪个顺序更好。
建议进行实验。

与此密切相关、同样会影响规划时间的另一个问题，是将子查询折叠进父查询。例如：

```
SELECT *
FROM x, y,
    (SELECT * FROM a, b, c WHERE something) AS ss
WHERE somethingelse;
```

这种情况可能出现在使用包含连接的视图时；该视图的`SELECT`
规则会被插入到引用视图的位置，从而得到一个与上面非常相似的查询。通常，
规划器会尝试把子查询折叠进父查询，得到：

```
SELECT * FROM x, y, a, b, c WHERE something AND somethingelse;
```

这通常会生成比单独规划子查询更好的计划。（例如，外层WHERE条件可能会先把 X 连接到A，从而消除 A 中的大量行，也就避免了形成子查询完整逻辑输出的需要。）但与此同时，规划时间也增加了；这里把两个彼此独立的三路连接问题，替换成了一个五路连接问题。由于可能性数量呈指数增长，这种差别可能非常大。为了避免陷入巨大的连接搜索问题，如果折叠子查询会使父查询产生超过from_collapse_limit个FROM项，规划器就会尝试通过停止提升子查询来规避这一点。可以通过调高或调低这个运行时参数，在规划时间和计划质量之间做权衡。

from_collapse_limit和join_collapse_limit名称相似，因为它们做的几乎是同一件事：一个控制规划器何时将子查询“平面化”，另一个控制何时将显式连接平面化。通常，要么把join_collapse_limit设置为与from_collapse_limit相同，这样显式连接与子查询的行为类似；要么把join_collapse_limit设置为 1，如果想使用显式连接控制连接顺序。不过，也可以把它们设置为不同的值，以更细致地调节规划时间与运行时间之间的平衡。

14.4. 填充一个数据库

初次填充数据库时，可能需要插入大量数据。本节给出一些使这一过程尽可能高效的建议。

14.4.1. 禁用自动提交

使用多个INSERT时，应关闭自动提交，只在最后提交一次。（在普通 SQL 中，这意味着开始时发出BEGIN，结束时发出COMMIT。某些客户端库可能会替调用方完成这件事，在这种情况下需要确认它们确实会在所需的时候这样做。）如果允许每次插入都单独提交，PostgreSQL就必须为每一行的加入执行大量额外工作。把所有插入都放在一个事务中的另一个好处是：如果其中某一行插入失败，那么此前插入的所有行都会被回滚，这样就不会留下部分装载的数据。

14.4.2. 使用COPY

使用COPY在一条命令中装载所有记录，而不是使用一系列INSERT命令。COPY命令针对装载大量行做了优化；它不如INSERT灵活，但在大规模数据装载时开销显著更小。由于COPY是一条单独的命令，因此采用这种方法填充表时无须关闭自动提交。

如果不能使用COPY，那么用PREPARE创建一个预备INSERT语句，再按需多次执行EXECUTE也会有所帮助。这样可以避免重复解析和规划INSERT的开销。不同接口以不同方式提供这一功能，可参阅接口文档中关于“预备语句”的说明。

请注意，在装载大量行时，使用COPY几乎总是比使用INSERT更快，即使已经使用了PREPARE，并把多次插入批量放入同一个事务中也是如此。

当COPY与更早的CREATE TABLE或TRUNCATE命令处于同一事务中时，速度最快。在这种情况下，不需要写 WAL，因为一旦出错，包含新装载数据的文件反正也会被移除。不过，这一点只适用于wal_level为minimal的非分区表；否则所有命令都必须写 WAL。

14.4.3. 移除索引

如果正在装载一个新创建的表，最快的方法是先创建表，用COPY批量装载数据，然后再创建该表所需的索引。在已有数据的表上创建索引，要比在每行装载时对索引做增量更新更快。

如果正在向现有表加入大量数据，那么删除索引、装载数据、再重建索引可能是更好的方案。当然，在索引缺失期间，其他数据库用户的性能可能会下降。删除唯一索引之前也必须慎重，因为唯一约束提供的错误检查会在索引缺失期间丧失。

14.4.4. 移除外键约束

与索引类似，“批量”检查外键约束比逐行检查更高效。因此，先删除外键约束、装载数据、再重建约束可能很有用。同样，
这里也需要在装载速度与约束缺失期间失去错误检查之间做权衡。

更重要的是，当在已有外键约束的情况下向表中装载数据时，
每一行新数据都需要在服务器待处理的触发器事件列表中占一个条目，
因为外键约束检查是通过触发器触发完成的。
装载数百万行可能导致触发器事件队列溢出可用内存，造成无法接受的交换，
甚至让命令直接失败。因此在装载大量数据时，删除并重新应用外键可能是必须的，
而不仅仅是期望如此。如果不能临时移除约束，
唯一的替代办法可能就是把装载操作拆分成更小的事务。

14.4.5. 增加maintenance_work_mem

在装载大量数据时，临时增大maintenance_work_mem配置变量可以提升性能。
这个参数也有助于加速`CREATE INDEX`和`ALTER TABLE ADD FOREIGN KEY`命令。它对`COPY`本身帮助不大，因此这个建议只有在采用前述一种或两种技巧时才有意义。

14.4.6. 增加checkpoint_segments

临时增大checkpoint_segments配置变量，也可以让大规模数据装载更快。这是因为向PostgreSQL中装载大量数据，会导致检查点比平常更频繁地发生，而正常频率由`checkpoint_timeout`配置变量指定。每次发生检查点时，所有脏页都必须刷写到磁盘。
通过在批量装载期间临时增大`checkpoint_segments`，可以减少所需的检查点次数。

14.4.7. 禁用 WAL 归档和流复制

当在使用 WAL 归档或流复制的安装中载入大量数据时，完成装载后重新做一次基础备份，可能比处理大量增量 WAL 数据更快。为了避免在装载期间记录这些增量 WAL，可以通过将wal_level设为`minimal`、将archive_mode设为`off`，并将max_wal_senders设为零，
来禁用归档和流复制。但请注意，修改这些设置需要重启服务器。

这样做除了省去归档进程或 WAL 发送进程处理 WAL 数据的时间外，还能使某些命令执行得更快，因为这些命令被设计为在`wal_level`为`minimal`时完全不写 WAL。（它们可以在结束时执行一次`fsync`，以比写 WAL 更低的代价保证崩溃安全性。）这适用于以下命令：

- `CREATE TABLE AS SELECT`
- `CREATE INDEX`（以及 `ALTER TABLE ADD PRIMARY KEY` 等变体）
- `ALTER TABLE SET TABLESPACE`
- `CLUSTER`
- `COPY FROM`，前提是目标表在同一事务中已被创建或截断

14.4.8. 事后运行ANALYZE

每当显著改变了表中数据的分布，都强烈建议运行ANALYZE。
这也包括向表中批量装载大量数据。运行`ANALYZE`（或`VACUUM ANALYZE`）可以确保规划器掌握该表的最新统计信息。如果没有统计信息，或者统计信息已经过时，
规划器在生成查询计划时就可能做出糟糕决定，从而导致相关表性能不佳。注意，
如果启用了自动清理守护进程，它可能会自动运行`ANALYZE`；详见第 23.1.3 节和第 23.1.6 节。

14.4.9. 关于pg_dump的一些注记

由pg_dump生成的转储脚本会自动应用上面若干条指导原则，但并非全部。若要尽可能快速地还原pg_dump的转储，仍需手动做一些额外操作。（注意，这些要点适用于还原转储，而不是创建转储。无论是使用psql加载文本转储，还是使用pg_restore从pg_dump归档文件加载，相关要点都是一样的。）

默认情况下，pg_dump使用`COPY`；而当它生成完整的模式加数据转储时，也会小心地先装载数据，再创建索引和外键。因此在这种情况下，上述若干指导原则已经被自动处理。剩下需要做的是：

- 为`maintenance_work_mem`和`checkpoint_segments`设置适当的（即比正常值大的）值。
- 如果使用 WAL 归档或流复制，可以考虑在恢复期间禁用它们。为此，请在载入转储之前将`archive_mode`设为`off`、将`wal_level`设为`minimal`，并将`max_wal_senders`设为零。恢复完成后，再把这些设置改回正确的值，并重新做一次基础备份。
- 考虑是否应当把整个转储作为单个事务来恢复。要这样做，请把`-1`或`--single-transaction`命令行选项传给psql或pg_restore。使用这种模式时，即使是很小的错误也会回滚整个恢复过程，可能丢掉数小时的处理成果。视数据之间的关联程度而定，这种做法未必一定比手工清理更可取。如果使用单个事务并关闭 WAL 归档，`COPY`命令会运行得最快。
- 如果在数据库服务器上有多个 CPU 可用，可以考虑使用pg_restore的`--jobs`选项。这允许并行数据载入和索引创建。
- 之后运行`ANALYZE`。

仅包含数据的转储仍然会使用`COPY`，但它不会删除或重建索引，通常也不会处理外键。[1]因此，在装载纯数据转储时，如果想采用这些技术，就需要自行负责删除并重建索引与外键。装载数据期间增大`checkpoint_segments`仍然有益，但没有必要同时增大`maintenance_work_mem`；后者更适合留到之后手工重建索引和外键时再调大。完成后也别忘了执行`ANALYZE`；详见第 23.1.3 节和第 23.1.6 节。

14.5. 非持久设置

持久性是一种数据库特性，保证即使服务器崩溃或断电，已提交的事务也会被记录下来。但持久性会带来显著的数据库开销，因此如果你的应用场景不需要这种保证，可以将PostgreSQL配置为以快得多的速度运行。以下配置更改可以在这些情况下提高性能。除下文特别注明的情况外，数据库软件崩溃时仍然保证持久性；使用这些设置时，只有操作系统突然停止运行才会造成数据丢失或损坏的风险。

- 将数据库集簇的数据目录放在一个内存支持的文件系统上（即RAM磁盘）。这消除了所有的数据库磁盘 I/O，但将数据存储限制到可用的内存量（可能有交换区）。
- 关闭fsync；不需要将数据刷入磁盘。
- 关闭synchronous_commit；这样可能无需在每次提交时强制将WAL写入磁盘。这种设置会在数据库崩溃时带来事务丢失的风险（但不会造成数据损坏）。
- 关闭full_page_writes；不需要警惕部分页面写入。
- 增加checkpoint_segments和checkpoint_timeout；这会降低检查点频率，但会增加`/pg_wal`的存储需求。

[1] 可以通过使用`--disable-triggers`选项达到禁用外键的效果 — 但要注意，这样做是取消外键验证，而不仅仅是推迟它。因此如果使用该选项，就有可能插入坏数据。

部分 III. 服务器管理

本部分涵盖PostgreSQL管理员关心的主题，包括软件安装、服务器配置、
用户和数据库管理以及维护任务。任何运行PostgreSQL服务器的人，即使仅供个人使用，
尤其是在生产环境中，都应熟悉这些主题。

本部分的内容大体按照新用户应当阅读的顺序来安排。各章自成体系，也可按需单独阅读。
相关内容按主题单元以叙述方式展开。希望获得某个命令完整描述的读者，建议查阅第 VI 部分。

开头几章写得无需预备知识即可理解，因此需要自行搭建服务器的新用户可以从这里开始探索。
本部分其余内容涉及调优和管理；这些材料假定读者已经熟悉PostgreSQL数据库系统的一般用法。
建议读者再参阅第 I 部分和第 II 部分部分，以获取更多信息。

目录

15. 从源代码安装 347
15.1. 简要说明 347
15.2. 需求 347
15.3. 获取源代码 349
15.4. 安装过程 349
15.5. 安装后设置 358
15.5.1. 共享库 358
15.5.2. 环境变量 359
15.6. 受支持的平台 360
15.7. 平台相关说明 360
15.7.1. AIX 360
15.7.2. Cygwin 363
15.7.3. HP-UX 364
15.7.4. IRIX 364
15.7.5. MinGW/原生 Windows 365
15.7.6. SCO OpenServer 和 SCO UnixWare 365
15.7.7. Solaris 367
16. 在 Windows 上从源代码安装 369
16.1. 使用 Visual C++ 或 Microsoft Windows SDK 构建 369
16.1.1. 需求 370
16.1.2. 64 位 Windows 的特殊注意事项 371
16.1.3. 构建 371
16.1.4. 清理和安装 371
16.1.5. 运行回归测试 372
16.1.6. 构建文档 372
16.2. 使用Visual C++或Borland C++构建libpq 373
16.2.1. 生成的文件 373
17. 服务器设置和操作 374
17.1. PostgreSQL用户账户 374
17.2. 创建一个数据库集簇 374
17.2.1. 二级文件系统的使用 375
17.2.2. 使用网络文件系统 375
17.3. 启动数据库服务器 375
17.3.1. 服务器启动失败 377
17.3.2. 客户端连接问题 377
17.4. 管理内核资源 378
17.4.1. 共享内存和信号量 378
17.4.2. 资源限制 383
17.4.3. Linux 内存过量分配 384
17.5. 关闭服务器 385
17.6. 升级 PostgreSQL 集簇 385
17.6.1. 通过pg_dump升级数据 386
17.6.2. 非转储升级方法 387
17.7. 防止服务器欺骗 388
17.8. 加密选项 388
17.9. 使用 SSL 的安全 TCP/IP 连接 389
17.9.1. 使用客户端证书 390
17.9.2. SSL 服务器文件用法 390
17.9.3. 创建自签名证书 390
17.10. 通过 SSH 隧道建立安全 TCP/IP 连接 391
17.11. 在Windows上注册事件日志 392
18. 服务器配置 393
18.1. 设置参数 393
18.1.1. 参数名称和值 393
18.1.2. 通过配置文件影响参数 393

18.1.3. 设置参数的其他方法 394
18.1.4. 查看参数设置 394
18.2. 文件位置 394
18.3. 连接和认证 395
18.3.1. 连接设置 395
18.3.2. 安全和认证 397
18.4. 资源消耗 399
18.4.1. 内存 399
18.4.2. 磁盘 400
18.4.3. 内核资源使用 401
18.4.4. 基于代价的清理延迟 401
18.4.5. 后台写入器 402
18.4.6. 异步行为 403
18.5. 预写式日志 403
18.5.1. 设置 403
18.5.2. 检查点 406
18.5.3. 归档 406
18.6. 复制 407
18.6.1. 发送服务器 407
18.6.2. 主库 408
18.6.3. 备库 408
18.7. 查询规划 409
18.7.1. 规划器方法配置 409
18.7.2. 规划器代价常量 410
18.7.3. 遗传查询优化器 412
18.7.4. 其他规划器选项 412
18.8. 错误报告和日志 413
18.8.1. 日志记录到哪里 414
18.8.2. 什么时候记录日志 416
18.8.3. 记录哪些内容 417
18.8.4. 使用 CSV 格式的日志输出 421
18.9. 运行时统计数据 422
18.9.1. 查询和索引统计信息收集器 422
18.9.2. 统计监控 423
18.10. 自动清理 423
18.11. 客户端连接默认值 424
18.11.1. 语句行为 424
18.11.2. 区域设置和格式化 427
18.11.3. 其他默认值 428
18.12. 锁管理 429
18.13. 版本和平台兼容性 430
18.13.1. 以前的 PostgreSQL 版本 430
18.13.2. 平台和客户端兼容性 431
18.14. 错误处理 432
18.15. 预置选项 432
18.16. 自定义选项 433
18.17. 开发者选项 433
18.18. 短选项 436
19. 客户端认证 437
19.1. `pg_hba.conf` 文件 437
19.2. 用户名映射 443
19.3. 认证方法 444
19.3.1. 信任认证 444
19.3.2. 密码认证 445
19.3.3. GSSAPI 认证 445
19.3.4. SSPI 认证 445
19.3.5. Kerberos 认证 446
19.3.6. Ident 认证 447

19.3.7. Peer 认证 448
19.3.8. LDAP 认证 448
19.3.9. RADIUS 认证 449
19.3.10. 证书认证 450
19.3.11. PAM 认证 450
19.4. 认证问题 451
20. 数据库角色 452
20.1. 数据库角色 452
20.2. 角色属性 453
20.3. 角色成员资格 454
20.4. 删除角色 455
20.5. 函数和触发器安全性 456
21. 管理数据库 457
21.1. 概述 457
21.2. 创建数据库 457
21.3. 模板数据库 458
21.4. 数据库配置 459
21.5. 删除数据库 459
21.6. 表空间 460
22. 本地化 462
22.1. 区域设置支持 462
22.1.1. 概述 462
22.1.2. 行为 463
22.1.3. 问题 463
22.2. 排序规则支持 464
22.2.1. 概念 464
22.2.2. 管理排序规则 465
22.3. 字符集支持 466
22.3.1. 支持的字符集 466
22.3.2. 设置字符集 468
22.3.3. 服务器和客户端之间的自动字符集转换 469
22.3.4. 进一步阅读 471
23. 日常数据库维护任务 472
23.1. 日常清理 472
23.1.1. 清理基础 472
23.1.2. 回收磁盘空间 473
23.1.3. 更新规划器统计信息 473
23.1.4. 更新可见性映射 474
23.1.5. 防止事务 ID 回卷失败 474
23.1.6. 自动清理守护进程 476
23.2. 日常重建索引 477
23.3. 日志文件维护 478
24. 备份和恢复 479
24.1. SQL转储 479
24.1.1. 恢复转储 480
24.1.2. 使用pg_dumpall 480
24.1.3. 处理大型数据库 481
24.2. 文件系统级备份 481
24.3. 持续归档和时间点恢复（PITR） 482
24.3.1. 设置 WAL 归档 483
24.3.2. 进行基础备份 485
24.3.3. 使用低级 API 进行基础备份 485
24.3.4. 使用持续归档备份进行恢复 486
24.3.5. 时间线 488
24.3.6. 建议和示例 488
24.3.7. 注意事项 490
25. 高可用、负载均衡和复制 491
25.1. 不同方案的比较 491

25.2. 日志传送备库 493
25.2.1. 规划 494
25.2.2. 备库操作 494
25.2.3. 为备库准备主库 494
25.2.4. 设置备库 495
25.2.5. 流复制 495
25.2.6. 级联复制 497
25.2.7. 同步复制 497
25.3. 故障切换 499
25.4. 日志传送的另一种方法 499
25.4.1. 实现 500
25.4.2. 基于记录的日志传送 501
25.5. 热备 501
25.5.1. 用户概览 501
25.5.2. 处理查询冲突 502
25.5.3. 管理员概览 504
25.5.4. 热备参数参考 506
25.5.5. 注意事项 506
26. 恢复配置 508
26.1. 归档恢复设置 508
26.2. 恢复目标设置 509
26.3. 备库设置 509
27. 监控数据库活动 511
27.1. 标准 Unix 工具 511
27.2. 统计收集器 512
27.2.1. 统计信息收集配置 512
27.2.2. 查看收集到的统计信息 512
27.2.3. 统计函数 521
27.3. 查看锁 522
27.4. 动态追踪 522
27.4.1. 为动态追踪编译 522
27.4.2. 内置探针 523
27.4.3. 使用探针 528
27.4.4. 定义新探针 528
28. 监控磁盘使用情况 530
28.1. 确定磁盘使用情况 530
28.2. 磁盘已满故障 531
29. 可靠性与预写式日志 532
29.1. 可靠性 532
29.2. 预写式日志（WAL） 533
29.3. 异步提交 534
29.4. WAL 配置 535
29.5. WAL 内部机制 536
30. 回归测试 538
30.1. 运行测试 538
30.1.1. 针对临时安装运行测试 538
30.1.2. 针对现有安装运行测试 538
30.1.3. 附加测试套件 539
30.1.4. 区域设置和编码 539
30.1.5. 额外测试 540
30.1.6. 测试热备 540
30.2. 测试结果评估 540
30.2.1. 错误消息差异 541
30.2.2. 区域设置差异 541
30.2.3. 日期和时间差异 541
30.2.4. 浮点差异 541
30.2.5. 行顺序差异 542
30.2.6. 栈深度不足 542

30.2.7. “random” 测试 542
30.2.8. 配置参数 542
30.3. 变体比较文件 542
30.4. 测试覆盖率检查 543

第 15 章 从源代码安装

本章介绍如何使用源代码发布包安装 PostgreSQL。（如果你安装的是预打包的发行版，例如 RPM 或 Debian 软件包，请忽略本章，改为阅读打包者提供的说明。）

15.1. 简要说明

```
./configure
make
su
make install
adduser postgres
mkdir /usr/local/pgsql/data
chown postgres /usr/local/pgsql/data
su - postgres
/usr/local/pgsql/bin/initdb -D /usr/local/pgsql/data
/usr/local/pgsql/bin/postgres -D /usr/local/pgsql/data >logfile
 2>&1 &
/usr/local/pgsql/bin/createdb test
/usr/local/pgsql/bin/psql test
```

详细说明见本章的其余部分。

15.2. 需求

一般来说，现代的 Unix 兼容平台应该能够运行 PostgreSQL。发布时经过具体测试的平台列于下方第 15.6 节。发行包的 `doc` 子目录中有若干平台特定的 FAQ 文档，遇到问题时可以查阅。

构建 PostgreSQL 需要下列软件包：

- 需要 GNU make 3.80 或更高版本；其他 make 程序或较旧版本的 GNU make 都无法工作。GNU make 通常以 `gmake` 这个名字安装；本文档将始终以该名称称呼它。（在某些系统上，GNU make 是名为 `make` 的默认工具。）要检查是否为 GNU make，请输入：

  ```
  gmake --version
  ```

- 你需要一个 ISO/ANSI C 编译器（至少符合 C89）。推荐使用较新的 GCC 版本，不过已知 PostgreSQL 也可以使用来自不同厂商的多种编译器构建。

- 需要 tar 来解开源代码发布包，此外还需要 gzip 或 bzip2 之一。

- 默认使用 GNU Readline 库。它可以让 psql（PostgreSQL 命令行 SQL 解释器）记住你输入的每一条命令，并允许你使用方向键回忆和编辑此前的命令。这非常有用，强烈推荐。如果你不想使用它，则必须给 `configure` 指定 `--without-readline` 选项。作为替代方案，你通常也可以使用 BSD 许可的 `libedit` 库，它最初是在 NetBSD 上开发的。`libedit` 库与 GNU Readline 兼容；如果找不到 `libreadline`，或者在 `configure` 中使用了 `--with-libedit-preferred` 选项，就会使用它。如果你使用的是基于软件包的 Linux 发行版，请注意，如果你的发行版把它们拆成单独的软件包，那么需要同时安装 `readline` 和 `readline-devel`。

- 默认使用 zlib 压缩库。如果你不想使用它，则必须给 `configure` 指定 `--without-zlib` 选项。使用这个选项将禁用 pg_dump 和 pg_restore 对压缩归档的支持。

下列软件包是可选的。默认配置不需要它们，但在启用某些构建选项时会用到，如下所述：

- 要构建服务器端编程语言 PL/Perl，你需要一个完整的 Perl 安装，包括 `libperl` 库和头文件。最低要求版本是 Perl 5.8.3。由于 PL/Perl 是一个共享库，在大多数平台上，`libperl`

也必须是共享库。这在较新的 Perl 版本中似乎是默认行为，但在更早版本中并非如此；无论如何，这取决于你所在环境中安装 Perl 的人是如何选择的。如果你打算不仅仅是偶尔使用 PL/Perl，应确保 Perl 安装是在启用了 `usemultiplicity` 选项的情况下构建的（`perl -V` 会显示这一点）。

如果你没有共享库但需要一个，PostgreSQL 构建期间会出现类似下面的消息来指出这一事实：

```
*** Cannot build PL/Perl because libperl is not a shared library.
*** You might have to rebuild your Perl installation.  Refer to
*** the documentation for details.
```

（如果你没有留意屏幕输出，你只会注意到 PL/Perl 库对象 `plperl.so` 或类似文件没有被安装。）如果看到这条消息，你就必须手动重新构建并安装 Perl，才能构建 PL/Perl。在配置 Perl 时，请要求生成共享库。

- 要构建 PL/Python 服务器端编程语言，你需要带有头文件以及 distutils 模块的 Python 安装。最低要求版本是 Python 2.3。支持 3.1 及更高版本的 Python 3；但使用 Python 3 时请参见第 42.1 节。

  由于 PL/Python 是一个共享库，在大多数平台上，`libpython` 也必须是共享库。默认的 Python 安装并非如此。如果在构建并安装 PostgreSQL 之后，你得到了一个名为 `plpython.so` 的文件（扩展名可能不同），那么一切正常。否则，你应该会看到类似下面的一闪而过的通知：

  ```
  *** Cannot build PL/Python because libpython is not a shared
   library.
  *** You might have to rebuild your Python installation.  Refer to
  *** the documentation for details.
  ```

  这意味着你必须重新构建（部分）Python 安装来创建这个共享库。

  如果遇到问题，请在 Python 2.3 或更高版本的 configure 中使用 `--enable-shared` 标志。在某些操作系统上，你不必构建共享库，但你必须让 PostgreSQL 构建系统相信这一点。详情请查阅 `src/pl/plpython` 目录中的 `Makefile`。

- 要构建 PL/Tcl 过程语言，你当然需要安装 Tcl。如果你使用的是 8.4 之前的 Tcl 版本，请确保它构建时没有启用多线程支持。

- 要启用本地语言支持（NLS），也就是以非英语语言显示程序消息的能力，你需要一个 Gettext API 的实现。某些操作系统已内置此功能（例如 Linux、NetBSD、Solaris）；对于其他系统，你可以从http://www.gnu.org/software/gettext/下载附加软件包。
  如果你使用的是 GNU C 库中的 Gettext 实现，那么还需要 GNU Gettext 软件包来提供某些实用程序。如果使用其他实现，则不需要它。

- 如果你希望支持使用这些服务进行认证或加密，就需要 Kerberos、OpenSSL、OpenLDAP 和/或 PAM。

如果你从 Git 树构建，而不是使用已发布的源码包，或者你要进行服务器开发，还需要以下软件包：

- 从 Git 检出构建，或者修改了扫描器和解析器的实际定义文件时，需要 GNU Flex 和 Bison。如果需要它们，请确保使用 Flex 2.5.31 或更高版本，以及 Bison 1.875 或更高版本。不能使用其他 lex 和 yacc 程序。

- 从 Git 检出构建，或者修改了使用 Perl 脚本的任何构建步骤的输入文件时，需要 Perl 5.8.3 或更高版本。在 Windows 上构建时，无论如何都需要 Perl。

如果需要获取 GNU 软件包，可以从本地的 GNU 镜像站点获取（站点列表见http://www.gnu.org/order/ftp.html），或者从ftp://ftp.gnu.org/gnu/获取。

还要检查磁盘空间是否充足。编译期间，源码树需要约 100 MB，安装目录需要约 20 MB。空数据库集簇约占 35 MB；数据库所需空间大约是存储相同数据的纯文本文件的五倍。如果要运行回归测试，还会临时额外需要最多 150 MB。使用 `df` 命令检查可用磁盘空间。

15.3. 获取源代码

要获取 PostgreSQL 9.2.24 的源码，请访问网站的下载区：https://www.postgresql.org/download/。你应该取得一个名为 `postgresql-9.2.24.tar.gz` 或 `postgresql-9.2.24.tar.bz2` 的文件。取得文件后，将其解包：

```
gunzip postgresql-9.2.24.tar.gz
tar xf postgresql-9.2.24.tar
```

（可使用 `bunzip2` 代替 `gunzip` 来解压 `.bz2` 文件。）这会在当前目录下创建目录 `postgresql-9.2.24`，其中包含 PostgreSQL 源码。切换到该目录，继续完成其余安装过程。

你也可以直接从版本控制仓库获取源码，参见附录 I。

15.4. 安装过程

1. **配置**

   安装过程的第一步是为你的系统配置源码树，并选择所需选项。
   这通过运行以下脚本完成：`configure`。对于默认安装，只需输入：

   ```
   ./configure
   ```

   此脚本会运行一系列测试，确定各个依赖系统的变量的值，
   并检测操作系统的特殊之处，最后在构建树中创建若干文件来记录结果。你也可以运行 `configure` 时使用源码树之外的目录，以将构建目录单独存放。这种过程也称为 VPATH 构建。方法如下：

   ```
   mkdir build_dir
   cd build_dir
   /path/to/source/tree/configure [options go here]
   make
   ```

   默认配置将构建服务器和实用程序，以及所有只需要 C 编译器的客户端应用程序和接口。默认情况下，所有文件都会安装到 `/usr/local/pgsql` 之下。

   你可以通过提供以下一个或多个命令行选项来定制构建和安装过程，这些选项传给 `configure`：

   `--prefix=`*`PREFIX`*

   把所有文件安装到目录 *`PREFIX`* 下，而不是 `/usr/local/pgsql` 下。实际文件会安装到各个子目录中；不会有任何文件直接安装到 *`PREFIX`* 目录中。

   如果有特殊需求，还可以通过以下选项分别定制各个子目录。不过，
   如果保留它们的默认值，安装就可以重定位，也就是说，安装完成后可以移动目录。（`man` 和 `doc` 的位置不受此影响。）

   对于可重定位的安装，可以考虑使用 `configure` 的 `--disable-rpath` 选项。此外，还需要告知操作系统如何查找共享库。

`--exec-prefix=EXEC-PREFIX`

你可以把与体系结构相关的文件安装到与 *PREFIX* 不同的前缀 *EXEC-PREFIX* 下。这对于在多台主机之间共享与体系结构无关的文件很有用。如果省略该选项，那么 *EXEC-PREFIX* 会被设为与 *PREFIX* 相同，与体系结构相关和无关的文件都会安装到同一棵目录树下，这通常正是你想要的。

`--bindir=DIRECTORY`

指定可执行程序的目录。默认是 *EXEC-PREFIX*/bin，通常也就是 /usr/local/pgsql/bin。

`--sysconfdir=DIRECTORY`

设置各种配置文件的目录，默认是 *PREFIX*/etc。

`--libdir=DIRECTORY`

设置安装库和动态可加载模块的位置。默认是 *EXEC-PREFIX*/lib。

`--includedir=DIRECTORY`

设置安装 C 和 C++ 头文件的目录。默认是 *PREFIX*/include。

`--datarootdir=DIRECTORY`

设置多种只读数据文件的根目录。这只会为后续某些选项设置默认值。默认是 *PREFIX*/share。

`--datadir=DIRECTORY`

设置已安装程序使用的只读数据文件目录。默认是 *DATAROOTDIR*。注意，这与数据库文件实际存放的位置无关。

`--localedir=DIRECTORY`

设置安装区域设置数据的目录，尤其是消息翻译目录文件。默认是 *DATAROOTDIR*/locale。

`--mandir=DIRECTORY`

PostgreSQL 附带的手册页将安装到这个目录下各自对应的 man*x* 子目录中。默认是 *DATAROOTDIR*/man。

`--docdir=DIRECTORY`

设置安装文档文件（不包括“man”页）的根目录。这只会为后续选项设置默认值。该选项的默认值是 *DATAROOTDIR*/doc/postgresql。

`--htmldir=DIRECTORY`

PostgreSQL 的 HTML 格式文档将安装到这个目录。默认是 *DATAROOTDIR*。

注意

为了能把 PostgreSQL 安装到共享安装位置（例如 /usr/local/include），同时又不干扰系统其他部分的名字空间，我们做了特别处理。首先，除非完整展开后的目录名已经包含字符串“postgres”或“pgsql”，否则会自动把字符串“/postgresql”追加到 datadir、sysconfdir 和

> docdir 上。例如，如果你选择 /usr/local 作为前缀，那么文档会安装到 /usr/local/doc/postgresql；但如果前缀是 /opt/postgres，那么它会安装到 /opt/postgres/doc。客户端接口的公共 C 头文件会安装到 includedir 中，并且不会污染名字空间。内部头文件和服务器头文件则会安装到 includedir 下的私有目录中。关于如何访问这些头文件，请参见各接口自己的文档。最后，如果有需要，也会在 libdir 下创建私有子目录，用于存放动态可加载模块。

`--with-includes=DIRECTORIES`

DIRECTORIES 是一个以冒号分隔的目录列表，这些目录会被加入编译器搜索头文件的路径中。如果你把可选软件包（如 GNU Readline）安装在非标准位置，就必须使用此选项，并且通常还要使用对应的 `--with-libraries` 选项。

例如：`--with-includes=/opt/gnu/include:/usr/sup/include`。

`--with-libraries=DIRECTORIES`

DIRECTORIES 是一个以冒号分隔的目录列表，用于搜索库文件。如果你把软件包安装在非标准位置，很可能需要使用这个选项（以及对应的 `--with-includes`）。

例如：`--with-libraries=/opt/gnu/lib:/usr/sup/lib`。

`--enable-nls[=LANGUAGES]`

启用本地语言支持（NLS），也就是以非英语语言显示程序消息的能力。*LANGUAGES* 是可选的、以空格分隔的语言代码列表，例如 `--enable-nls='de fr'`。（你的列表与实际提供的翻译集合之间的交集会自动计算。）如果你不指定列表，则会安装所有可用的翻译。

要使用此选项，需要一个 Gettext API 实现；参见上文。

`--with-pgport=NUMBER`

将 *NUMBER* 设为服务器和客户端的默认端口号。默认值是 5432。端口号以后始终都可以修改，但如果在这里指定，那么服务器和客户端都会编译进同一个默认值，这可能很方便。通常选择非默认值的唯一合理原因，是你打算在同一台机器上运行多个 PostgreSQL 服务器。

`--with-perl`

构建 PL/Perl 服务器端语言。

`--with-python`

构建 PL/Python 服务器端语言。

`--with-tcl`

构建 PL/Tcl 服务器端语言。

`--with-tclconfig=DIRECTORY`

Tcl 会安装文件 `tclConfig.sh`，其中包含构建与 Tcl 交互的模块所需的配置信息。通常会在已知位置自动找到此文件，但如果要使用其他版本的 Tcl，可以指定查找它的目录。

`--with-gssapi`

构建时支持 GSSAPI 认证。在许多系统上，GSSAPI 系统（通常是 Kerberos 安装的一部分）并不安装在默认搜索的位置（如 /usr/include、/usr/lib），因此除了此选项，还必须使用 `--with-includes` 和 `--with-libraries`。`configure` 会检查所需的头文件和库，确保 GSSAPI 安装满足要求，然后才会继续。

`--with-krb5`

构建时支持 Kerberos 5 认证。在许多系统上，Kerberos 系统并未安装在默认搜索的位置（如 /usr/include、/usr/lib），因此除了此选项，还必须使用 `--with-includes` 和 `--with-libraries`。`configure` 会检查所需的头文件和库，确保 Kerberos 安装满足要求，然后才会继续。

`--with-krb-srvnam=`*`NAME`*

Kerberos 服务主体的默认名称（GSSAPI 也会使用）。默认值为 `postgres`。通常没有理由更改，除非使用 Windows 环境，此时必须将其设为大写的 `POSTGRES`。

`--with-openssl`

构建时支持 SSL（加密）连接。这要求已安装 OpenSSL 软件包。`configure` 会检查所需的头文件和库，确保 OpenSSL 安装满足要求，然后才会继续。

`--with-pam`

构建时支持 PAM（可插拔认证模块）。

`--with-ldap`

构建时支持 LDAP 认证和连接参数查找（更多信息参见第 31.17 节和第 19.3.8 节）。在 Unix 上，这要求已安装 OpenLDAP 软件包。在 Windows 上，使用默认的 WinLDAP 库。`configure` 会检查所需的头文件和库，确保 OpenLDAP 安装满足要求，然后才会继续。

`--without-readline`

禁止使用 Readline 库（也禁止 libedit）。此选项会禁用 psql 中的命令行编辑和历史记录，因此不建议使用。

`--with-libedit-preferred`

优先使用 BSD 许可的 libedit 库，而不是 GPL 许可的 Readline。只有在两个库都已安装时，该选项才有意义；在那种情况下，默认行为是使用 Readline。

`--with-bonjour`

构建时支持 Bonjour。这要求操作系统支持 Bonjour。建议在 OS X 上启用。

`--with-ossp-uuid`

使用 OSSP UUID 库[1]构建相关组件。具体来说，构建 uuid-ossp模块，它提供生成 UUID 的函数。

`--with-libxml`

构建时支持 libxml（以启用 SQL/XML 支持）。这一特性要求 libxml 2.6.23 或更高版本。

[1] http://www.ossp.org/pkg/lib/uuid/

libxml 会安装一个 `xml2-config` 程序，可用于检测所需的编译器和链接器选项。如果找到该程序，PostgreSQL 会自动使用它。要指定位于不常见位置的 libxml 安装，可以把环境变量 `XML2_CONFIG` 设为指向该安装对应的 `xml2-config` 程序，或使用 `--with-includes` 和 `--with-libraries` 选项。

`--with-libxslt`

构建 xml2模块时使用 libxslt。xml2 依赖此库对 XML 执行 XSL 转换。

`--disable-integer-datetimes`

禁用对时间戳和时间间隔的 64 位整数存储支持，改为以浮点数存储日期时间值。浮点日期时间存储是 8.4 之前PostgreSQL发行版的默认方式，但现在已被弃用，因为它无法在完整`timestamp`取值范围内支持微秒精度。不过，
基于整数的日期时间存储需要 64 位整数类型。因此，在没有这种类型的平台上，或为了与为早期PostgreSQL版本编写的应用保持兼容，可以使用此选项。
更多信息参见第 8.5 节。

`--disable-float4-byval`

禁用 float4 值的“按值”传递，改为“按引用”传递。此选项会降低性能，但为了兼容用 C 编写并使用“版本 0”调用约定的旧用户定义函数，可能需要使用它。更好的长期解决办法是更新这些函数，使其使用“版本 1”调用约定。

`--disable-float8-byval`

禁用 float8 值的“按值”传递，改为“按引用”传递。此选项会降低性能，但为了兼容用 C 编写并使用“版本 0”调用约定的旧用户定义函数，可能需要使用它。更好的长期解决办法是更新这些函数，使其使用“版本 1”调用约定。注意，此选项不仅影响 float8，还影响 int8 和 timestamp 等相关类型。在 32 位平台上，默认使用 `--disable-float8-byval`，且不允许选择 `--enable-float8-byval`。

`--with-segsize=`*`SEGSIZE`*

设置段大小，单位为 GB。大型表会拆分为多个操作系统文件，每个文件的大小等于段大小。这避免了许多平台上的文件大小限制问题。默认段大小为 1 GB，在所有支持的平台上都是安全的。如果操作系统支持“大文件”（如今大多数系统都支持），可以使用更大的段大小。这有助于减少处理超大表时消耗的文件描述符数量。
但应注意，不要选择超出平台和拟用文件系统支持范围的值。
你可能要使用的其他工具（如 tar）也可能限制可用的文件大小。建议将此值设为 2 的幂，但这并非硬性要求。注意，更改此值需要执行 initdb。

`--with-blocksize=`*`BLOCKSIZE`*

设置块大小，单位为 KB。这是表内存储和 I/O 的单位。默认值 8 KB 适用于大多数情况；在特殊情况下，其他值可能有用。该值必须是 1 到 32（KB）之间的 2 的幂。注意，更改此值需要执行 initdb。

`--with-wal-segsize=`*`SEGSIZE`*

设置 WAL 段大小，单位为 MB。这是 WAL 日志中每个文件的大小。调整此大小可能有助于控制 WAL 日志传送的粒度。默认大小为 16 MB。该值必须是 1 到 64（MB）之间的 2 的幂。注意，更改此值需要执行 initdb。

`--with-wal-blocksize=`*`BLOCKSIZE`*

设置 WAL 块大小，单位为 KB。这是 WAL 日志内存储和 I/O 的单位。默认值 8 KB 适用于大多数情况；在特殊情况下，其他值可能有用。该值必须是 1 到 64（KB）之间的 2 的幂。注意，更改此值需要执行 initdb。

`--disable-spinlocks`

即使 PostgreSQL 不支持该平台的 CPU 自旋锁，也允许构建成功。缺少自旋锁支持会导致性能不佳；因此，仅应在构建中止并告知平台缺少自旋锁支持时使用此选项。如果在你的平台上构建 PostgreSQL 必须使用此选项，请向 PostgreSQL 开发者报告该问题。

`--disable-thread-safety`

禁用客户端库的线程安全性。这会使 libpq 和 ECPG 程序中的并发线程无法安全地控制各自私有的连接句柄。

`--with-system-tzdata=`*`DIRECTORY`*

PostgreSQL 自带了日期和时间操作所需的时区数据库。这个时区数据库实际上与很多操作系统（如 FreeBSD、Linux 和 Solaris）提供的 IANA 时区数据库兼容，因此再次安装它是多余的。使用此选项时，将使用位于 *`DIRECTORY`* 的、由系统提供的时区数据库，而不是 PostgreSQL 源码发布中自带的那一份。*`DIRECTORY`* 必须是绝对路径。在某些操作系统上，`/usr/share/zoneinfo` 是一个可能的目录。请注意，安装过程不会检测时区数据是否不匹配或有误。如果你使用该选项，建议运行回归测试，以验证你指定的时区数据能够与 PostgreSQL 正常配合工作。

这个选项主要面向那些非常了解其目标操作系统的二进制包发布者。
使用该选项的主要优点是，当本地众多夏令时规则中的任何一条发生变化时，PostgreSQL 软件包都无需升级。另一个优点是，如果安装时不需要构建时区数据库文件，那么 PostgreSQL 的交叉编译也会更直接。

`--without-zlib`

禁止使用 Zlib 库。这会禁用 pg_dump 和 pg_restore 对压缩归档的支持。此选项仅用于没有该库的罕见系统。

`--enable-debug`

将所有程序和库编译为带调试符号的版本。这意味着你可以在调试器中运行程序，以分析问题。这会显著增大安装后的可执行文件大小，而且在非 GCC 编译器上，通常还会禁用编译器优化，从而导致变慢。不过，
保留这些符号对于处理可能出现的各种问题极其有帮助。目前，只有在你使用 GCC 的情况下，才建议在生产安装中使用该选项。但如果你在做开发工作或运行测试版，就应始终启用它。

`--enable-coverage`

如果使用 GCC，所有程序和库都会在编译时插入代码覆盖率测试所需的检测代码。运行时，它们会在构建目录中生成包含代码覆盖率指标的文件。更多信息参见第 30.4 节。此选项仅用于使用 GCC 进行开发工作时。

`--enable-profiling`

如果使用 GCC，所有程序和库都会编译为可进行性能分析的形式。后端退出时，会创建一个子目录，其中包含用于性能分析的 `gmon.out` 文件。此选项仅用于使用 GCC 进行开发工作时。

`--enable-cassert`

在服务器中启用断言检查，用于测试许多“不可能发生”的条件。
这对代码开发非常有价值，但这些测试可能会显著拖慢服务器速度。此外，

启用这些测试并不一定会增强服务器稳定性！断言检查并未按严重程度分类，因此即使某个 bug 相对无害，只要触发了断言失败，仍会导致服务器重启。该选项不建议用于生产环境，但如果你在做开发工作或运行测试版，就应当启用它。

`--enable-depend`

启用自动依赖跟踪。启用后，makefile 会在任何头文件被修改时，重新构建所有受影响的目标文件。如果你在做开发工作，这很有用；但如果你只是打算编译一次并安装，这只是额外的开销。目前该选项只在 GCC 下有效。

`--enable-dtrace`

编译 PostgreSQL 时支持动态跟踪工具 DTrace。更多信息参见第 27.4 节。

可以设置环境变量 `DTRACE` 来指定 `dtrace` 程序。这通常是必要的，因为 `dtrace` 一般安装在 `/usr/sbin` 下，而该目录可能不在搜索路径中。

对于 `dtrace` 程序，可在以下环境变量中指定额外命令行选项：`DTRACEFLAGS`。在 Solaris 上，要在 64 位二进制文件中包含 DTrace 支持，必须指定 `DTRACEFLAGS="-64"` 给 configure。例如，使用 GCC 编译器时：

```
./configure CC='gcc -m64' --enable-dtrace DTRACEFLAGS='
-64' ...
```

使用 Sun 的编译器时：

```
./configure CC='/opt/SUNWspro/bin/cc -xtarget=native64' --
enable-dtrace DTRACEFLAGS='-64' ...
```

如果你希望使用不同于 `configure` 所选的 C 编译器，可以将环境变量 `CC` 设为所选程序。默认情况下，`configure` 会在 `gcc` 可用时选择它，否则选择平台默认编译器（通常为 `cc`）。类似地，如有需要，可以使用 `CFLAGS` 变量覆盖默认编译器标志。

可以在 `configure` 命令行上指定环境变量，例如：

```
./configure CC=/opt/bin/gcc CFLAGS='-O2 -pipe'
```

下面列出可以按这种方式设置的重要变量：

`BISON`

Bison 程序

`CC`

C 编译器

`CFLAGS`

传给 C 编译器的选项

`CPP`

C 预处理器

`CPPFLAGS`

传给 C 预处理器的选项

DTRACE

dtrace 程序的位置

DTRACEFLAGS

传给 dtrace 程序的选项

FLEX

Flex 程序

LDFLAGS

用于链接可执行文件或共享库的选项

LDFLAGS_EX

仅用于链接可执行文件的附加选项

LDFLAGS_SL

仅用于链接共享库的附加选项

MSGFMT

用于本地语言支持的 msgfmt 程序

PERL

Perl 解释器的完整路径名。它将用于确定构建 PL/Perl 所需的依赖。

PYTHON

Python 解释器的完整路径名。它将用于确定构建 PL/Python 所需的依赖。此外，这里指定（或以其他方式隐式选择）Python 2 还是 3，决定了可用的 PL/Python 语言变体。更多信息参见第 42.1 节。

TCLSH

Tcl 解释器的完整路径名。它将用于确定构建 PL/Tcl 所需的依赖，并会被代入 Tcl 脚本中。

XML2_CONFIG

用于定位 libxml 安装的 xml2-config 程序。

有时，事后补充编译器标志会很有用；原有标志由 configure 选定。一个重要的例子是 gcc 的 -Werror 选项不能包含在以下变量中：CFLAGS（该变量会传给 configure），因为它会破坏 configure 的许多内置测试。要添加此类标志，请将其放入 COPT 环境变量，再运行 make。COPT 的内容会被加入以下两组选项：CFLAGS 和 LDFLAGS；这两组选项原本由以下程序设置：configure。例如，你可以执行：

```
make COPT='-Werror'
```

或者：

```
export COPT='-Werror'
make
```

> **注意**
>
> 开发服务器内部代码时，建议使用 configure 选项 `--enable-cassert`（开启许多运行时错误检查）和 `--enable-debug`（提高调试工具的实用性）。
>
> 如果使用 GCC，最好使用至少 `-O1` 的优化级别进行构建，因为不使用优化（`-O0`）会关闭一些重要的编译器警告（例如未初始化变量的使用）。不过，非零优化级别会让调试变得更复杂，因为单步执行编译后的代码时，通常无法与源代码行一一对应。如果你在调试优化后的代码时感到困惑，可以把感兴趣的特定文件用 `-O0` 重新编译。一个简单方法是向 make 传递选项：`make PROFILE=-O0 file.o`。
>
> `COPT` 和 `PROFILE` 环境变量实际上会被 PostgreSQL 的 makefile 以完全相同的方式处理。用哪个只是个人偏好问题，但开发者常见的习惯是把 `PROFILE` 用于一次性的标志调整，而 `COPT` 则可能长期保持设置。

2. **构建**

   要开始构建，请输入：

   `make`

   （记得使用 GNU make。）构建需要几分钟，具体取决于硬件。显示的最后一行应该是：

   ```
   All of PostgreSQL successfully made. Ready to install.
   ```

   如果要构建所有可构建的内容，包括文档（HTML 和手册页）及附加模块（`contrib`），请改为输入：

   `make world`

   显示的最后一行应该是：

   ```
   PostgreSQL, contrib, and documentation successfully made. Ready
    to install.
   ```

3. **回归测试**

   如果希望在安装前测试刚构建的服务器，可以在此时运行回归测试。
   回归测试是一套测试程序，用于验证 PostgreSQL 是否在你的机器上按开发者预期的方式运行。输入：

   `make check`

   （不能以 root 身份运行；请使用非特权用户。）第 30 章包含解读测试结果的详细信息。以后任何时候都可以通过执行同一命令来重复此测试。

4. **安装文件**

> **注意**
>
> 如果正在升级现有系统，请务必阅读第 17.6 节其中包含升级数据库集簇的说明。

   要安装 PostgreSQL，请输入：

```
make install
```

这会把文件安装到步骤 1中指定的目录。请确保你有权限写入该区域。通常这一步需要以 root 身份执行。另一种做法是预先创建目标目录，并安排授予适当权限。

要安装文档（HTML 和手册页），请输入：

```
make install-docs
```

如果你上面执行的是 world 构建，则改为输入：

```
make install-world
```

这也会安装文档。

你可以用 `make install-strip` 代替 `make install`，在安装时剥离可执行文件和库中的符号。这会节省一些空间。如果你在构建时启用了调试支持，那么剥离操作实际上会移除调试支持，因此只有在不再需要调试时才应该这样做。`install-strip` 会尽量合理地节省空间，但它并不知道如何从每个可执行文件中去掉所有不需要的字节，所以如果你希望尽可能节省所有磁盘空间，仍需要手工处理。

标准安装会提供客户端应用开发以及服务器端程序开发（例如用 C 编写自定义函数或数据类型）所需的全部头文件。（在 PostgreSQL 8.0 之前，后者需要单独执行 `make install-all-headers` 命令，但此步骤现已并入标准安装。）

仅客户端安装：. 如果你只想安装客户端应用程序和接口库，可以使用以下命令：

```
make -C src/bin install
make -C src/include install
make -C src/interfaces install
make -C doc install
```

`src/bin` 中有少量仅供服务器使用的二进制文件，不过它们都很小。

卸载：. 要撤销安装，请使用命令 `make uninstall`。不过，这不会删除任何已创建的目录。

清理：. 安装完成后，你可以使用命令 `make clean` 从源码树中删除构建产生的文件，以释放磁盘空间。这会保留 `configure` 程序生成的文件，以便你之后可以用 `make` 重新构建全部内容。要把源码树重置到发布时的状态，请使用 `make distclean`。如果你打算在同一个源码树中为多个平台构建，就必须为每个平台执行这一步并重新配置。（另一种做法是为每个平台使用单独的构建树，这样源码树就不会被修改。）

如果你已经执行了一次构建，随后发现 `configure` 选项有误，或者修改了任何会被 `configure` 检查的内容（例如软件升级），那么在重新配置和重新构建之前先执行 `make distclean` 是一个好主意。否则，你对配置选项所做的更改可能不会传播到所有需要它们的地方。

15.5. 安装后设置

15.5.1. 共享库

在某些使用共享库的系统上，你需要告诉系统如何找到新安装的共享库。不需要这样做的系统包括 FreeBSD、HP-UX、IRIX、Linux、NetBSD、OpenBSD、Tru64 UNIX（前身为 Digital UNIX）和 Solaris。

设置共享库搜索路径的方法因平台而异，但最常见的方法是设置环境变量 LD_LIBRARY_PATH，方法如下。在 Bourne shell（sh，ksh，bash，zsh）：

```
LD_LIBRARY_PATH=/usr/local/pgsql/lib
export LD_LIBRARY_PATH
```

或者在 csh 或 tcsh：

```
setenv LD_LIBRARY_PATH /usr/local/pgsql/lib
```

请将 /usr/local/pgsql/lib 替换为你为 --libdir 指定的值，设置过程见步骤 1。应将这些命令放入 shell 启动文件，例如 /etc/profile 或 ~/.bash_profile。关于此方法相关注意事项的一些有用信息，参见http://xahlee.org/UnixResource_dir/_/ldpath.html。

在某些系统上，更好的做法可能是在构建之前设置环境变量 LD_RUN_PATH。

在 Cygwin 上，请把库目录加入 PATH，或者把 .dll 文件移到 bin 目录中。

如果不确定，请参考你系统的手册页（可能是 ld.so 或 rld）。如果你随后收到类似下面这样的消息：

```
psql: error in loading shared libraries
libpq.so.2.1: cannot open shared object file: No such file or
 directory
```

那就说明这一步确实是必需的。到那时再处理即可。

如果你使用的是 Linux 且拥有 root 权限，可以在安装后运行：

```
/sbin/ldconfig /usr/local/pgsql/lib
```

（或者等效目录），这样可以让运行时链接器更快地找到共享库。更多信息请参见 ldconfig 的手册页。在 FreeBSD、NetBSD 和 OpenBSD 上，相应命令是：

```
/sbin/ldconfig -m /usr/local/pgsql/lib
```

其他系统尚不清楚是否有等效命令。

15.5.2. 环境变量

如果你安装到了 /usr/local/pgsql，或者安装到了其他默认不会搜索程序的目录，那么应该把 /usr/local/pgsql/bin（或者你在步骤 1中通过 --bindir 设置的值）加入你的 PATH。严格来说，这不是必需的，但它会让 PostgreSQL 的使用方便得多。

为此，请把下面内容加到你的 shell 启动文件中，例如 ~/.bash_profile（如果你希望影响所有用户，则用 /etc/profile）：

```
PATH=/usr/local/pgsql/bin:$PATH
export PATH
```

如果你使用的是 csh 或 tcsh，则使用这条命令：

```
set path = ( /usr/local/pgsql/bin $path )
```

要让系统能够找到 man 文档，除非你安装到了默认会搜索的位置，否则需要把类似下面的内容加到 shell 启动文件中：

```
MANPATH=/usr/local/pgsql/share/man:$MANPATH
export MANPATH
```

环境变量 PGHOST 和 PGPORT 会告诉客户端应用数据库服务器的主机和端口，从而覆盖编译时内置的默认值。如果你准备从远端运行客户端应用，

那么让每个打算使用数据库的用户都设置 `PGHOST` 会比较方便。不过这不是必需的；对于大多数客户端程序，也可以通过命令行选项传递这些设置。

15.6. 受支持的平台

如果代码中包含了使某个平台（也就是一种 CPU 架构和操作系统的组合）工作的相关处理，并且最近已经在该平台上验证过能够成功构建并通过回归测试，那么 PostgreSQL 开发社区就认为该平台受支持。目前，大多数平台兼容性测试都是由 PostgreSQL 构建农场[2]中的测试机器自动完成的。如果你有兴趣在某个平台上使用 PostgreSQL，而该平台尚未出现在构建农场中，但代码已经能够工作或可以被修改得可工作，
我们强烈鼓励你设置一个构建农场成员机器，以便持续确保兼容性。

一般来说，可以预期 PostgreSQL 能在以下 CPU 架构上工作：x86、x86_64、IA64、Power PC、PowerPC 64、S/390、S/390x、Sparc、Sparc 64、Alpha、ARM、MIPS、MIPSEL、M68K 和 PA-RISC。代码中包含对 M32R 和 VAX 的支持，但尚不清楚这些架构近期是否经过测试。对于不受支持的 CPU 类型，通常可以通过配置 `--disable-spinlocks` 来构建，但性能会很差。

可以预期 PostgreSQL 能在以下操作系统上工作：Linux（所有近期发行版）、Windows（Win2000 SP4 及更高版本）、FreeBSD、OpenBSD、NetBSD、OS X、AIX、HP/UX、Solaris、Tru64 Unix 和 UnixWare。其他类 Unix 系统也可能可用，但目前未在测试。大多数情况下，某个操作系统所支持的全部 CPU 架构也都可以工作。尤其是在使用较旧系统时，请查看下面的第 15.7 节，看看是否有针对你的操作系统的特别说明。

如果你在某个平台上遇到安装问题，而根据近期构建农场的结果该平台是受支持的，
请把问题报告到 `<pgsql-bugs@lists.postgresql.org>`。如果你有兴趣把 PostgreSQL 移植到一个新平台，那么 `<pgsql-hackers@lists.postgresql.org>` 是合适的讨论地点。

15.7. 平台相关说明

本节记录 PostgreSQL 安装和设置时的一些平台相关附加问题。请务必阅读安装说明，尤其是第 15.2 节。此外，也请参阅第 30 章中关于如何解释回归测试结果的说明。

这里未列出的平台，目前没有已知的平台特定安装问题。

15.7.1. AIX

PostgreSQL 可以在 AIX 上运行，但正确安装可能比较困难。AIX 4.3.3 至 6.1 被视为受支持版本。你可以使用 GCC 或原生 IBM 编译器 `xlc`。一般来说，使用较新的 AIX 和 PostgreSQL 版本会有所帮助。关于哪些 AIX 版本已知可用的最新信息，请查看构建农场。

对于受支持的 AIX 版本，建议至少达到以下修复级别：

AIX 4.3.3

维护级别 11，加上 ML11 后续补丁包

AIX 5.1

维护级别 9，加上 ML9 后续补丁包

AIX 5.2

技术级别 10，服务包 3

[2] https://buildfarm.postgresql.org/

AIX 5.3

技术级别 7

AIX 6.1

基础级别

要检查当前修复级别，在 AIX 4.3.3 至 AIX 5.2 ML 7 上使用 `oslevel -r`，在更高版本上使用 `oslevel -s`。

如果将 Readline 或 libz 安装在 `/usr/local`，除了自己的选项外，还应使用以下 `configure` 标志：`--with-includes=/usr/local/include --with-libraries=/usr/local/lib`。

15.7.1.1. GCC 问题

在 AIX 5.3 上，使用 GCC 编译和运行 PostgreSQL 曾出现一些问题。

应使用晚于 3.3.2 的 GCC 版本，尤其是在使用预打包版本时。我们使用 4.0.1 的效果很好。较早版本的问题似乎更多与 IBM 打包 GCC 的方式有关，而非 GCC 本身的问题，因此如果自行编译 GCC，使用较早版本的 GCC 也很可能成功。

15.7.1.2. Unix 域套接字不可用

AIX 5.3 存在一个问题：`sockaddr_storage` 的大小定义不足。在 5.3 版中，IBM 增大了 Unix 域套接字地址结构 `sockaddr_un` 的大小，却未相应增大 `sockaddr_storage`。结果是，尝试通过 Unix 域套接字使用 PostgreSQL 时，libpq 会写出该数据结构的边界。TCP/IP 连接正常，但 Unix 域套接字无法正常工作，导致回归测试无法运行。

此问题已报告给 IBM，缺陷报告编号为 PMR29657。升级到维护级别 5300-03 或更高版本即可包含此修复。一种快速解决办法是将 `/usr/include/sys/socket.h` 中的 `_SS_MAXSIZE` 改为 1025。无论采用哪种方法，在取得修正后的头文件后，都应重新编译 PostgreSQL。

15.7.1.3. 互联网地址问题

PostgreSQL 依赖系统的 `getaddrinfo` 函数解析 `listen_addresses`、`pg_hba.conf` 等位置中的 IP 地址。较旧的 AIX 版本在此函数中存在多种缺陷。如果遇到与这些设置有关的问题，更新到上面所列的适当 AIX 修复级别应该能够解决。

一位用户报告：

在 AIX 5.3 上部署 PostgreSQL 8.1 时，我们不时遇到统计信息收集器“莫名其妙地”无法成功启动的问题。这似乎是 IPv6 实现中意外行为所致。看起来在 AIX 5.3 上，PostgreSQL 与 IPv6 配合得不太好。

以下任意操作都能“修复”此问题。

- 删除 localhost 的 IPv6 地址：

  ```
  (as root)
  # ifconfig lo0 inet6 ::1/0 delete
  ```

- 从网络服务中移除 IPv6。在 AIX 上，文件 `/etc/netsvc.conf` 大致相当于 Solaris/Linux 上的 `/etc/nsswitch.conf`。AIX 上的默认设置如下：

  ```
  hosts=local,bind
  ```

  将其替换为：

  ```
  hosts=local4,bind4
  ```

以停用对 IPv6 地址的搜索。

> **警告**
>
> 这实际上只是针对 IPv6 支持不成熟所引发问题的临时解决办法，而 IPv6 支持在 AIX 5.3 的各次发布中已有明显改进。此方法在 AIX 5.3 上有效，但并不是优雅的解决方案。据报告，在 IPv6 支持更成熟的 AIX 6.1 上，这种做法不仅没有必要，还会导致问题。

15.7.1.4. 内存管理

AIX 的内存管理方式有些特殊。服务器可能有数 GB 乃至更多的空闲 RAM，但运行应用程序时仍会出现内存不足或地址空间错误。例如，`createlang`可能因不寻常的错误而失败。
以下是以 PostgreSQL 安装所有者身份运行的例子：

```
-bash-3.00$ createlang plperl template1
createlang: language installation failed: ERROR:  could not load
 library "/opt/dbs/pgsql748/lib/plperl.so": A memory address is not
 in the address space for the process.
```

以 PostgreSQL 安装所属组中非所有者的成员身份运行：

```
-bash-3.00$ createlang plperl template1
createlang: language installation failed: ERROR:  could not load
 library "/opt/dbs/pgsql748/lib/plperl.so": Bad address
```

另一个例子是 PostgreSQL 服务器日志中的内存不足错误，此时每次接近或超过 256 MB 的内存分配都会失败。

这些问题的总体原因是服务器进程使用的默认位数和内存模型。默认情况下，在 AIX 上构建的所有二进制文件都是 32 位的。这与硬件类型或所用内核无关。这些 32 位进程的内存上限为 4 GB，按几种模型之一以 256 MB 的段布局。默认情况下，堆与栈共享一个段，因此堆可用空间不足 256 MB。

对于上面的 `createlang` 示例，请检查你的 umask 和 PostgreSQL 安装中二进制文件的权限。该示例涉及的二进制文件是 32 位的，安装时使用模式 750 而非 755。由于权限以这种方式设置，只有所有者或所属组的成员可以加载该库。因为它不是所有用户可读，加载器会将该对象放入进程的堆中，而不是通常应放置的共享库段。

这个问题的“理想”解决办法是使用 64 位构建的 PostgreSQL，但这并非总是可行，因为采用 32 位处理器的系统可以构建 64 位二进制文件，却无法运行它们。

如果需要 32 位二进制文件，请在启动 PostgreSQL 服务器之前，将 `LDR_CNTRL` 设为 `MAXDATA=0x*n*0000000`，其中 1 <= n <= 8，并尝试不同的值及 `postgresql.conf` 设置，找到能令人满意地工作的配置。这样使用 `LDR_CNTRL` 会告知 AIX，为服务器的堆预留 `MAXDATA` 字节，按 256 MB 的段分配。找到可用配置后，可以使用 `ldedit` 修改二进制文件，使其默认使用所需的堆大小。也可以重新构建 PostgreSQL，传入 `configure LDFLAGS="-Wl,-bmaxdata:0x*n*0000000"` 来达到相同效果。

对于 64 位构建，将 `OBJECT_MODE` 设为 64，并向 `configure` 传入 `CC="gcc -maix64"` 和 `LDFLAGS="-Wl,-bbigtoc"`。（`xlc` 的选项可能不同。）如果未导出 `OBJECT_MODE`，构建可能因链接器错误而失败。设置 `OBJECT_MODE` 后，它会告知 AIX 的 `ar`、`as` 和 `ld` 等构建工具默认处理哪种对象。

默认情况下，可能发生分页空间过量分配。虽然我们尚未见过这种情况，
但当内存耗尽且访问了过量分配的空间时，AIX 会终止进程。我们遇到过最接近的情况是，系统认为没有足够内存容纳另一个进程，导致 fork 失败。与 AIX 的许多其他部分一样，如果这成为问题，可以在系统或进程级别配置分页空间分配方式和内存不足时终止进程的行为。

References and Resources

"Large Program Support[1]" . AIX Documentation: General Programming Concepts: Writing and Debugging Programs.

"Program Address Space Overview[2]" . AIX Documentation: General Programming Concepts: Writing and Debugging Programs.

"Performance Overview of the Virtual Memory Manager (VMM)[3]" . AIX Documentation: Performance Management Guide.

"Page Space Allocation[4]" . AIX Documentation: Performance Management Guide.

"Paging-space thresholds tuning[5]" . AIX Documentation: Performance Management Guide.

Developing and Porting C and C++ Applications on AIX[6]. IBM Redbook.

15.7.2. Cygwin

可以使用 Cygwin 这个 Windows 上的类 Linux 环境来构建 PostgreSQL，但这种方式不如原生 Windows 构建（见第 16 章），且如今已不再推荐在 Cygwin 下运行服务器。

从源码构建时，请按照常规安装过程操作（即 `./configure; make` 等），同时注意以下 Cygwin 特有的差异：

- 请把路径设置成优先使用 Cygwin 的 bin 目录，而不是 Windows 工具目录。这有助于避免编译问题。
- GNU make 命令叫 `make`，而不是 `gmake`。
- 不支持 `adduser` 命令；请使用 Windows NT、2000 或 XP 中相应的用户管理应用。如果不是这些系统，则跳过此步骤。
- 不支持 `su` 命令；请在 Windows NT、2000 或 XP 上使用 ssh 模拟 su。如果不是这些系统，则跳过此步骤。
- 不支持 OpenSSL。
- 请启动 `cygserver` 以支持共享内存。为此，请输入命令 `/usr/sbin/cygserver &`。每次启动 PostgreSQL 服务器或初始化数据库集簇（`initdb`）时，该程序都必须在运行中。默认的 `cygserver` 配置可能需要修改（例如增大 `SEMMNS`），以防止 PostgreSQL 因系统资源不足而失败。
- 在某些使用非 C 区域设置的系统上，构建可能会失败。要修复这一点，请在构建前执行 `export LANG=C.utf8` 把区域设置改为 C，安装完 PostgreSQL 后再把它恢复为之前的设置。
- 并行回归测试（`make check`）可能因 `listen()` 的待处理连接队列溢出而误报回归测试失败；队列溢出会导致连接被拒绝的错误或挂起。可以使用 make 变量 `MAX_CONNECTIONS` 限制连接数，方法如下：

  ```
  make MAX_CONNECTIONS=5 check
  ```

  （在某些系统上，并发连接数最高可达约 10 个。）

[1] http://publib.boulder.ibm.com/infocenter/pseries/topic/com.ibm.aix.doc/aixprggd/genprogc/lrg_prg_support.htm
[2] http://publib.boulder.ibm.com/infocenter/pseries/topic/com.ibm.aix.doc/aixprggd/genprogc/address_space.htm
[3] http://publib.boulder.ibm.com/infocenter/pseries/v5r3/topic/com.ibm.aix.doc/aixbman/prftungd/resmgmt2.htm
[4] http://publib.boulder.ibm.com/infocenter/pseries/v5r3/topic/com.ibm.aix.doc/aixbman/prftungd/memperf7.htm
[5] http://publib.boulder.ibm.com/infocenter/pseries/v5r3/topic/com.ibm.aix.doc/aixbman/prftungd/memperf6.htm
[6] http://www.redbooks.ibm.com/abstracts/sg245674.html?Open

可以把 `cygserver` 和 PostgreSQL 服务器安装为 Windows NT 服务。关于具体做法，请参阅 Cygwin 上 PostgreSQL 二进制包附带的 `README` 文档。它安装在 `/usr/share/doc/Cygwin` 目录中。

15.7.3. HP-UX

如果系统补丁级别和构建工具合适，PostgreSQL 7.3+ 应能在运行 HP-UX 10.X 或 11.X 的 Series 700/800 PA-RISC 机器上工作。至少有一位开发者经常在 HP-UX 10.20 上测试，我们也收到过在 HP-UX 11.00 和 11.11 上成功安装的报告。

除了 PostgreSQL 源码发行包，还需要 GNU make（HP 的 make 不可用），以及 GCC 或 HP 的完整 ANSI C 编译器。如果打算从 Git 源码而非发行 tar 包构建，还需要 Flex（GNU lex）和 Bison（GNU yacc）。我们也建议确保 HP 补丁相对较新。至少，在 HP-UX 11.11 上构建 64 位二进制文件时，可能需要 PHSS_30966（11.11）或其后续补丁，否则 `initdb` 可能挂起：

PHSS_30966 s700_800 ld(1) and linker tools cumulative patch

一般而言，应安装最新的 libc 和 ld/dld 补丁；如果使用 HP 的 C 编译器，也应安装最新的编译器补丁。请访问 HP 的支持站点，例如http://itrc.hp.com和ftp://us-ffs.external.hp.com/，免费获取最新补丁。

如果在 PA-RISC 2.0 机器上构建，并希望使用 GCC 生成 64 位二进制文件，则必须使用 64 位版本的 GCC。HP-UX PA-RISC 和 Itanium 的 GCC 二进制文件可从http://www.hp.com/go/gcc获取。别忘了同时获取并安装 binutils。

如果在 PA-RISC 2.0 机器上构建，并希望编译后的二进制文件能在 PA-RISC 1.1 机器上运行，需要在 `CFLAGS` 中指定 `+DAportable`。

如果在 HP-UX Itanium 机器上构建，需要最新的 HP ANSI C 编译器及其依赖补丁或后续补丁：

PHSS_30848 s700_800 HP C Compiler (A.05.57)
PHSS_30849 s700_800 u2comp/be/plugin library Patch

如果同时安装了 HP 的 C 编译器和 GCC，可以在运行 `configure` 时显式选择要使用的编译器：

```
./configure CC=cc
```

这会选择 HP 的 C 编译器；或者执行：

```
./configure CC=gcc
```

这会选择 GCC。如果省略此设置，configure 在可以选择时会选用 `gcc` 作为编译器。

默认安装目标位置为 `/usr/local/pgsql`，你可能希望将其改为 `/opt` 下的某个位置。如果是这样，请为 `configure` 使用 `--prefix` 选项。

在回归测试中，几何测试可能存在一些低位数字差异，
具体取决于所用编译器和数学库的版本。任何其他错误都值得怀疑。

15.7.4. IRIX

据报告，PostgreSQL 可以在 MIPS r8000、r10000（ip25 和 ip27）以及 r12000（ip35）处理器上成功运行，操作系统为 IRIX 6.5.5m、6.5.12、6.5.13 和 6.5.26，所用 MIPSPro 编译器版本为 7.30、7.3.1.2m、7.3 和 7.4.4m。

你需要完整的 MIPSPro ANSI C 编译器。使用 GCC 构建会存在问题。这是一个已知的 GCC 缺陷（截至 3.0 版仍未修复），与使用返回某些类型结构的函数有关。此缺陷影响诸如`inet_`

ntoa、inet_lnaof、inet_netof、inet_makeaddr和semctl之类的函数。据称可以通过强制将这些函数与 libgcc 链接来修复，但尚未经过测试。

已知 MIPSPro 编译器 7.4.1m 版会生成不正确的代码。症状是在尝试启动数据库时出现“invalid primary checkpoint record”（无效的主检查点记录）。7.4.4m 版没有问题；中间版本的状况不确定。

可能会出现如下编译问题：

```
cc-1020 cc: ERROR File = pqcomm.c, Line = 427
  The identifier "TCP_NODELAY" is undefined.

                if (setsockopt(port->sock, IPPROTO_TCP, TCP_
NODELAY,
```

某些版本的 TCP 定义包含在sys/xti.h中，因此需要在src/backend/libpq/pqcomm.c和src/interfaces/libpq/fe-connect.c中添加#include <sys/xti.h>。如果你遇到此问题，请告知我们，以便我们开发适当的修复。

在回归测试中，几何测试可能会出现一些低位数字的差异，具体取决于你所使用的 FPU。任何其他错误都是值得怀疑的。

15.7.5. MinGW/原生 Windows

Windows 版 PostgreSQL 可以使用 MinGW（用于 Microsoft 操作系统的类 Unix 构建环境）构建，也可以使用 Microsoft 的 Visual C++ 编译器套件构建。MinGW 构建方式使用本章介绍的常规构建系统；Visual C++ 构建方式完全不同，详见第 16 章。后者是完全原生的构建，不使用 MinGW 等附加软件。PostgreSQL 主网站上提供现成的安装程序。

原生 Windows 移植版本要求 32 位或 64 位的 Windows 2000 或更高版本。更早的操作系统没有足够的基础设施（但可以在其上使用 Cygwin）。可以从http://www.mingw.org/下载 MinGW 这一类 Unix 构建工具，以及 MSYS 这一运行 configure 等 shell 脚本所需的 Unix 工具集。运行生成的二进制文件不需要它们；只有创建二进制文件时才需要。

要使用 MinGW 构建 64 位二进制文件，请从http://mingw-w64.sourceforge.net/安装 64 位工具集，将其 bin 目录放入 PATH，并使用 --host=x86_64-w64-mingw32 选项运行 configure。

安装好所有组件后，建议在 CMD.EXE 下运行 psql，因为 MSYS 控制台存在缓冲问题。

15.7.5.1. 在 Windows 上收集崩溃转储

如果 PostgreSQL 在 Windows 上崩溃，它能够生成 minidumps，可用于追踪崩溃原因，类似于 Unix 上的核心转储。这些转储可以使用 Windows Debugger Tools 或 Visual Studio 读取。要在 Windows 上启用转储生成，请在集簇数据目录中创建一个名为 crashdumps 的子目录。随后，转储会以唯一名称写入该目录，该名称基于崩溃进程的标识符以及崩溃发生时的当前时间。

15.7.6. SCO OpenServer 和 SCO UnixWare

PostgreSQL 可以在 SCO UnixWare 7 和 SCO OpenServer 5 上构建。在 OpenServer 上，你可以使用 OpenServer Development Kit，也可以使用 Universal Development Kit。不过，可能需要按下面的说明做一些调整。

15.7.6.1. Skunkware

你应当找到你的 SCO Skunkware CD 副本。Skunkware CD 随 UnixWare 7 和当前版本的 OpenServer 5 一起提供。Skunkware 包含 Internet 上许多流行程序的即装即用版本。例如，gzip、gunzip、GNU Make、Flex 和 Bison 都包含在内。对于 UnixWare 7.1，这张 CD

现在标为 "Open License Software Supplement"。如果你没有这张 CD，其上的软件可以从 http://www.sco.com/skunkware/获取。

Skunkware 对 UnixWare 和 OpenServer 有不同的版本。请确保为你的操作系统安装正确的版本，下面注明的情况除外。

在 UnixWare 7.1.3 及更高版本中，UDK CD 上附带有 GCC 编译器，GNU Make 也是如此。

15.7.6.2. GNU Make

你需要使用 GNU Make 程序，它在 Skunkware CD 上。默认情况下，它会安装为 `/usr/local/bin/make`。为避免与 SCO 的 `make` 程序混淆，你可能想把 GNU `make` 重命名为 `gmake`。

对于 UnixWare 7.1.3 及更高版本，GNU Make 程序位于 UDK CD 的 OSTK 部分，路径为 `/usr/gnu/bin/gmake`。

15.7.6.3. Readline

Readline 库在 Skunkware CD 上。但 UnixWare 7.1 的 Skunkware CD 并不包含它。如果你有 UnixWare 7.0.0 或 7.0.1 的 Skunkware CD，可以从那里安装。否则，请尝试http://www.sco.com/skunkware/。

默认情况下，Readline 会安装到 `/usr/local/lib` 和 `/usr/local/include`。但是，PostgreSQL 的 `configure` 程序在没有帮助的情况下无法在那里找到它。如果你安装了 Readline，请对 `configure` 使用以下选项：

```
./configure --with-libraries=/usr/local/lib --with-includes=/usr/
local/include
```

15.7.6.4. 在OpenServer上使用UDK

如果你在 OpenServer 上使用新的 Universal Development Kit（UDK）编译器，需要指定 UDK 库的位置：

```
./configure --with-libraries=/udk/usr/lib --with-includes=/udk/usr/
include
```

把这些与上面的 Readline 选项结合起来：

```
./configure --with-libraries="/udk/usr/lib /usr/local/lib" --with-
includes="/udk/usr/include /usr/local/include"
```

15.7.6.5. 阅读PostgreSQL手册页

默认情况下，PostgreSQL 的手册页会安装到 `/usr/local/pgsql/share/man`。默认情况下，UnixWare 不会到该目录查找手册页。要能阅读它们，你需要修改 `/etc/default/man` 中的 `MANPATH` 变量，例如：

```
MANPATH=/usr/lib/scohelp/%L/man:/usr/dt/man:/usr/man:/usr/share/
man:scohelp:/usr/local/man:/usr/local/pgsql/share/man
```

在 OpenServer 上，要让手册页可用还需要额外花些功夫研究，因为其手册系统与其他平台有些不同。目前，PostgreSQL 完全不会安装它们。

15.7.6.6. 7.1.1b特性补充中的 C99 问题

对于早于 OpenUNIX 8.0.0（UnixWare 7.1.2）随附编译器的编译器，包括 7.1.1b 特性补充，你可能需要在 `CFLAGS` 或 `CC` 环境变量中指定 `-Xb`。其表现是在编译 `tuplesort.c` 时出现引用内联函数的错误。显然，7.1.2(8.0.0) 及之后的编译器发生了变化。

15.7.6.7. UnixWare上的线程

对于线程，你必须在所有使用 libpq 的程序上使用 `-Kpthread`。libpq 使用 `pthread_*` 调用，而这些调用只有在使用 `-Kpthread`/`-Kthread` 标志时才可用。

15.7.7. Solaris

PostgreSQL 在 Solaris 上有良好支持。你的操作系统越新，遇到的问题越少；详情见下文。

15.7.7.1. 所需工具

你可以使用 GCC 或 Sun 编译器套件来构建。为了获得更好的代码优化，在 SPARC 架构上强烈推荐使用 Sun 编译器。我们收到过使用 GCC 2.95.1 时出现问题的报告；建议使用 GCC 2.95.3 或更高版本。如果使用 Sun 编译器，请注意不要选择 `/usr/ucb/cc`；应使用 `/opt/SUNWspro/bin/cc`。

你可以从http://developers.sun.com/sunstudio/downloads/下载 Sun Studio。许多 GNU 工具已经集成到 Solaris 10 中，或者包含在 Solaris 配套光盘中。如果你需要适用于较旧 Solaris 版本的软件包，可以到http://www.sunfreeware.com或http://www.blastwave.org查找这些工具。如果你更想要源码，请看http://www.gnu.org/order/ftp.html。

15.7.7.2. OpenSSL 的问题

在以 OpenSSL 支持构建 PostgreSQL 时，你可能会在以下文件中遇到编译错误：

- `src/backend/libpq/crypt.c`
- `src/backend/libpq/password.c`
- `src/interfaces/libpq/fe-auth.c`
- `src/interfaces/libpq/fe-connect.c`

这是因为标准的 `/usr/include/crypt.h` 头文件与 OpenSSL 提供的头文件之间存在命名空间冲突。

将你的 OpenSSL 安装升级到 0.9.6a 版可解决此问题。Solaris 9 及更高版本带有较新版本的 OpenSSL。

15.7.7.3. configure 报告测试程序失败

如果 `configure` 报告某个测试程序失败，这多半是因为运行时链接器找不到某些库，通常是 libz、libreadline，或其他非标准库如 libssl。要把它指向正确位置，请在 `configure` 命令行中设置环境变量 `LDFLAGS`，例如：

```
configure ... LDFLAGS="-R /usr/sfw/lib:/opt/sfw/lib:/usr/local/lib"
```

更多信息请参见 ld 手册页。

15.7.7.4. 64 位构建有时会崩溃

在 Solaris 7 及更早版本上，64 位 libc 的 `vsnprintf` 例程存在缺陷，导致 PostgreSQL 不定期发生核心转储。已知最简单的解决办法是强制 PostgreSQL 使用自带的 `vsnprintf`，而不是库中的版本。为此，在运行 `configure` 后，编辑由 `configure` 生成的文件：在 `src/Makefile.global` 中，将以下行：

```
LIBOBJS =
```

改为：

```
LIBOBJS = snprintf.o
```

（此变量中可能已经列有其他文件，顺序无关紧要。）然后照常构建。

15.7.7.5. 为获得最佳性能而编译

在 SPARC 架构上，强烈推荐使用 Sun Studio 进行编译。可以尝试使用 `-xO5` 优化标志，以生成显著更快的二进制文件。不要使用任何会改变浮点运算行为以及 `errno` 处理行为的标志（例如 `-fast`）。这些标志可能导致 PostgreSQL 出现不符合标准的行为，例如在日期和时间计算中。

如果你没有理由在 SPARC 上使用 64 位二进制文件，那么优先选择 32 位版本。64 位运算更慢，而且 64 位二进制文件也比 32 位变体慢。另一方面，在 AMD64 CPU 家族上，32 位代码并不是原生模式，因此 32 位代码在该 CPU 家族上会明显更慢。

关于 PostgreSQL 与 Solaris 性能调优的一些技巧，可以参见http://www.sun.com/servers/coolthreads/tnb/applications_postgresql.jsp。这篇文章主要针对 T2000 平台，但其中的许多建议对运行 Solaris 的其他硬件同样有用。

15.7.7.6. 使用 DTrace 跟踪 PostgreSQL

是的，可以使用 DTrace。更多信息见第 27.4 节。你还可以在这篇文章中找到更多信息：http://blogs.sun.com/robertlor/entry/user_level_dtrace_probes_in。

如果发现链接 `postgres` 可执行文件时中止，并出现类似以下错误消息：

```
Undefined                       first referenced
 symbol                             in file
AbortTransaction                    utils/probes.o
CommitTransaction                   utils/probes.o
ld: fatal: Symbol referencing errors. No output written to postgres
collect2: ld returned 1 exit status
make: *** [postgres] Error 1
```

则说明安装的 DTrace 太旧，无法处理静态函数中的探针。需要 Solaris 10u4 或更高版本。

第 16 章 在 Windows 上从源代码安装

建议大多数用户下载适用于 Windows 的二进制发行版，该发行版以图形化安装程序包的形式提供，可从PostgreSQL网站获取。从源码构建仅面向开发PostgreSQL或扩展的人员。

在Windows上构建 PostgreSQL 有几种不同的方法。使用 Microsoft 工具进行构建的最简单方式，是安装Visual Studio Express 2012 for Windows Desktop并使用其自带的编译器。也可以使用完整的Microsoft Visual C++ 2005、2008 或 2010进行构建。在某些情况下，除编译器之外还需要安装Windows SDK。

也可以使用MinGW提供的 GNU 编译器工具构建 PostgreSQL，或者在较旧版本的Windows上使用Cygwin。

最后，客户端访问库（libpq）可以使用 Visual C++ 7.1或Borland C++构建，以便与使用这些工具静态链接构建的应用保持兼容。

使用MinGW或Cygwin构建时，采用的是常规构建系统，见第 15 章以及第 15.7.5 节和第 15.7.2 节中的专门说明。要在这些环境中生成原生 64 位二进制文件，请使用MinGW-w64工具。这些工具也可用于在其他主机（例如Linux和Darwin）上为 32 位和 64 位Windows目标进行交叉编译。不建议使用Cygwin运行生产服务器，它只应用于原生构建无法工作的较旧版本Windows，例如Windows 98。官方二进制文件是使用Visual Studio构建的。

原生构建的psql不支持命令行编辑。Cygwin构建支持命令行编辑，因此如果需要在Windows上交互式使用 psql，应当使用它。

16.1. 使用 Visual C++ 或 Microsoft Windows SDK 构建

PostgreSQL 可以使用 Microsoft 的 Visual C++ 编译器套件构建。这些编译器既可以来自Visual Studio、Visual Studio Express，也可以来自某些版本的Microsoft Windows SDK。如果你还没有配置好Visual Studio环境，最简单的办法是使用Windows SDK 7.1或Visual Studio Express 2012 for Windows Desktop中的编译器，它们都可以从 Microsoft 免费下载。

已知 PostgreSQL 支持使用Visual Studio 2005到 Visual Studio 2012（包括 Express 版本）自带的编译器，以及独立的 Windows SDK 6.0 到 7.1 版本进行编译。64 位 PostgreSQL 构建只支持 Microsoft Windows SDK 6.0a 到 7.1 版本，或 Visual Studio 2008及更高版本。

使用Visual C++构建所需的工具位于`src/tools/msvc`目录中。构建时，请确保系统 PATH 中没有来自MinGW或Cygwin的工具。同时，也要确保所需的 Visual C++ 工具都已在 PATH 中可用。在Visual Studio中，启动Visual Studio Command Prompt。在Microsoft Windows SDK中，启动开始菜单中 SDK 项下列出的CMD shell。在较新的 SDK 版本中，可以使用`setenv`命令来更改目标 CPU 架构。如果你想构建 64 位版本，就必须使用 64 位版本的命令提示符，反之亦然。所有命令都应在`src\tools\msvc`目录中运行。

在构建之前，你可能需要编辑`config.pl`文件，以反映你想更改的任何配置选项，或要使用的任何第三方库的路径。完整配置是先读取并解析`config_default.pl`文件，再应用来自`config.pl`的任何更改而得到的。例如，要指定Python安装位置，可在`config.pl`中写入以下内容：

```
$config->{python} = 'c:\python26';
```

你只需要指定那些与`config_default.pl`中不同的参数。

如果你需要设置其他环境变量，请创建一个名为`buildenv.pl`的文件，并将所需命令放入其中。例如，如果 bison 不在 PATH 中，可以创建一个包含以下内容的文件，以将其路径加入 PATH：

```
$ENV{PATH}=$ENV{PATH} . ';c:\some\where\bison\bin';
```

16.1.1. 需求

构建PostgreSQL需要以下附加产品。请使用`config.pl`文件指定这些库所在的目录。

Microsoft Windows SDK

建议升级到最新版本（目前为版本 7.1），可从https://www.microsoft.com/download下载。

你必须始终包含 SDK 中的Windows Headers and Libraries部分。如果你安装的Windows SDK包含Visual C++ Compilers，则无需Visual Studio即可构建。请注意，从 8.0a 版开始，Windows SDK 不再附带完整的命令行构建环境。

ActiveState Perl

运行构建生成脚本需要 ActiveState Perl。MinGW 或 Cygwin Perl 都无法工作。它还必须出现在 PATH 中。二进制文件可从http://www.activestate.com下载（注意：需要 5.8.3 或更高版本，免费的标准发行版即可）。

以下附加产品并非入门所必需，但若要构建完整的软件包则需要它们。请使用`config.pl`文件指定这些库所在的目录。

ActiveState TCL

构建 PL/TCL 需要它（注意：需要 8.4 版本，免费的标准发行版即可）。

Bison和Flex

从 Git 构建需要 Bison 和 Flex，但从发布文件构建则不需要。注意只有 Bison 1.875 或 2.2 及以上版本可以工作。此外，需要 Flex 2.5.31 或更高版本。Bison 可从http://gnuwin32.sourceforge.net下载。Flex 可从http://www.postgresql.org/ftp/misc/winflex/下载。如果你使用 msysGit 访问 PostgreSQL 的 Git 仓库，那么你的 Git 二进制目录中可能已经有较新版本的 bison 和 flex。

> **注意**
>
> 来自 GnuWin32 的 Bison 发行版似乎存在一个 bug：如果安装在名称中含有空格的目录中，Bison 就会工作异常，例如英文安装中的默认位置`C:\Program Files\GnuWin32`。请考虑安装到`C:\GnuWin32`。

Diff

运行回归测试需要 Diff，可从http://gnuwin32.sourceforge.net下载。

Gettext

构建带有 NLS 支持的版本需要 Gettext，可从http://gnuwin32.sourceforge.net下载。请注意，二进制文件、依赖项以及开发文件都是必需的。

MIT Kerberos

支持 Kerberos 认证需要 MIT Kerberos。MIT Kerberos 可从http://web.mit.edu/Kerberos/dist/index.html下载。

libxml2和libxslt

XML 支持需要它们。二进制文件可从http://zlatkovic.com/pub/libxml下载，源代码可从http://xmlsoft.org获取。请注意，libxml2 需要 iconv，而它可从相同的下载位置获得。

openssl

SSL 支持需要它。二进制文件可从http://www.slproweb.com/products/Win32OpenSSL.html下载，源码可从http://www.openssl.org下载。

ossp-uuid

支持 UUID-OSSP 需要它（仅 contrib）。源代码可从http://www.ossp.org/pkg/lib/uuid/下载。

Python

构建PL/Python需要它。二进制文件可从http://www.python.org下载。

zlib

pg_dump和pg_restore中的压缩支持需要它。二进制文件可从http://www.zlib.net下载。

16.1.2. 64 位 Windows 的特殊注意事项

PostgreSQL 在 64 位 Windows 上只会为 x64 架构构建，不支持 Itanium 处理器。

不支持在同一个构建树中混用 32 位和 64 位版本。构建系统会自动检测自己是在 32 位还是 64 位环境中运行，并据此构建 PostgreSQL。因此，在开始构建之前启动正确的命令提示符非常重要。

若要使用服务器端第三方库，例如python或openssl，该库也必须是 64 位的。不支持在 64 位服务器中加载 32 位库。PostgreSQL 支持的若干第三方库可能仅提供 32 位版本，在这种情况下，它们不能与 64 位 PostgreSQL 一起使用。

16.1.3. 构建

要以发布配置（默认值）构建 PostgreSQL 的全部内容，请运行以下命令：

```
build
```

要以调试配置构建 PostgreSQL 的全部内容，请运行以下命令：

```
build DEBUG
```

若只构建单个项目，例如 psql，请运行以下命令：

```
build psql
build DEBUG psql
```

若要将默认构建配置改为调试模式，请在`buildenv.pl`文件中加入以下内容：

```
$ENV{CONFIG}="Debug";
```

也可以在 Visual Studio 图形界面中进行构建。在这种情况下，你需要先从命令提示符运行：

```
perl mkvcbuild.pl
```

然后在 Visual Studio 中打开生成的`pgsql.sln`（位于源码树根目录）。

16.1.4. 清理和安装

大多数时候，Visual Studio 的自动依赖跟踪都会处理好变更过的文件。但如果变更较大，你可能需要清理安装。要执行此操作，只需运行`clean.bat`命令，

它会自动清除所有生成的文件。你也可以带上*dist*参数运行它，此时它的行为类似于**make distclean**，并且也会删除 flex/bison 的输出文件。

默认情况下，所有文件都会写入debug或release目录下的某个子目录。
若要按标准布局安装这些文件，并同时生成初始化和使用数据库所需的文件，
请运行以下命令：

```
install c:\destination\directory
```

16.1.5. 运行回归测试

要运行回归测试，请先确保已完成所有必需部分的构建。此外，
还要确保加载系统各部分所需的 DLL（例如过程语言所用的 Perl 和 Python DLL）都能在系统路径中找到。如果不能，请通过 `buildenv.pl` 文件设置路径。要运行测试，请在 `src\tools\msvc` 目录中执行以下命令之一：

```
vcregress check
vcregress installcheck
vcregress plcheck
vcregress contribcheck
```

要更改所用的测试调度方式（默认为 parallel），请将其追加到命令行，例如：

```
vcregress check serial
```

有关回归测试的更多信息，参见第 30 章。

16.1.6. 构建文档

将 PostgreSQL 文档构建为 HTML 格式，需要若干工具和文件。请为这些文件创建一个根目录，并将它们存放在下列子目录中。

OpenJade 1.3.1-2

从http://sourceforge.net/projects/openjade/files/openjade/1.3.1/openjade-1_3_1-2-bin.zip/download下载，并解压到子目录 `openjade-1.3.1` 中。

DocBook DTD 4.2

从http://www.oasis-open.org/docbook/sgml/4.2/docbook-4.2.zip下载，
并解压到子目录 `docbook` 中。

DocBook DSSSL 1.79

从http://sourceforge.net/projects/docbook/files/docbook-dsssl/1.79/docbook-dsssl-1.79.zip/download下载，并解压到子目录 `docbook-dsssl-1.79` 中。

ISO 字符实体

从http://www.oasis-open.org/cover/ISOEnts.zip下载，并解压到子目录 `docbook` 中。

编辑 `buildenv.pl` 文件，并添加一个变量来指定根目录的位置，例如：

```
$ENV{DOCROOT}='c:\docbook';
```

要构建文档，请运行命令 `builddoc.bat`。注意，为了生成索引，实际上会构建两次。生成的 HTML 文件位于 `doc\src\sgml`。

16.2. 使用Visual C++或Borland C++构建libpq

只有当你需要带有不同调试/发布标志的版本，或需要静态库以链接到应用中时，才建议使用Visual C++ 7.1-9.0或Borland C++构建 libpq。一般用途推荐使用MinGW、Visual Studio或Windows SDK方法。

要使用Visual Studio 7.1 或更高版本构建libpq客户端库，请切换到`src`目录并输入命令：

```
nmake /f win32.mak
```

要使用Visual Studio 8.0 或更高版本构建 64 位版本的libpq客户端库，请切换到`src`目录并输入命令：

```
nmake /f win32.mak CPU=AMD64
```

关于所支持变量的更多细节，参见`win32.mak`文件。

要使用Borland C++构建libpq客户端库，请切换到`src`目录并输入命令：

```
make -N -DCFG=Release /f bcc32.mak
```

16.2.1. 生成的文件

将构建以下文件：

`interfaces\libpq\Release\libpq.dll`

动态可链接的前端库

`interfaces\libpq\Release\libpqdll.lib`

用于把你的程序链接到`libpq.dll`的导入库

`interfaces\libpq\Release\libpq.lib`

前端库的静态版本

通常不需要安装任何客户端文件。你应当把`libpq.dll`文件放在与应用程序可执行文件相同的目录中。除非绝对必要，不要把`libpq.dll`安装到你的`Windows`、`System`或`System32`目录。如果使用安装程序安装该文件，则应利用文件中包含的`VERSIONINFO`资源进行带版本检查的安装，以确保不会覆盖较新版本的库。

如果你计划在这台机器上使用libpq进行开发，则需要把源码树的`src\include`和`src\interfaces\libpq`子目录加入编译器设置中的包含路径。

要使用该库，你必须把`libpqdll.lib`文件加入你的项目。（在 Visual C++ 中，只需右键单击项目并选择添加它。）

第 17 章 服务器设置和操作

本章讨论如何设置和运行数据库服务器，以及它与操作系统的交互。

17.1. PostgreSQL用户账户

与任何可从外部访问的服务器守护进程一样，建议使用单独的用户账户运行 PostgreSQL。该用户账户应该只拥有服务器管理的数据，不应与其他守护进程共用。（例如，使用 nobody 用户就不是好主意。）不建议将可执行文件安装为由该用户所有，因为系统一旦被攻破，就可能修改自己的二进制文件。

要在系统中添加一个 Unix 用户账户，请查找useradd或adduser命令。通常使用的用户名是postgres，本书也都按此假定，不过你也可以使用其他名称。

17.2. 创建一个数据库集簇

在做任何事情之前，必须先在磁盘上初始化一个数据库存储区域。我们把它称为数据库集簇。（SQL 标准使用术语目录集簇（catalog cluster）。）数据库集簇是由一个正在运行的数据库服务器实例管理的一组数据库。初始化之后，数据库集簇中会包含一个名为postgres的数据库，作为供工具、用户和第三方应用使用的默认数据库。数据库服务器本身并不要求该postgres数据库存在，但许多外部工具程序都假定它存在。初始化期间，集簇中还会创建另一个数据库，名为template1。顾名思义，它会作为后续新建数据库的模板；不应将它用于实际工作。（关于在集簇中创建新数据库的信息，见第 21 章。）

从文件系统角度看，数据库集簇就是一个目录，所有数据都存储在其下。我们将它称为数据目录或数据区域。数据存放位置完全由你决定，没有默认值；常用位置包括 /usr/local/pgsql/data或/var/lib/pgsql/data。要初始化数据库集簇，请使用命令initdb，该命令随以下产品安装：PostgreSQL。数据库集簇在文件系统中的目标位置通过 -D 选项指定，例如：

```
$ initdb -D /usr/local/pgsql/data
```

注意，必须先登录 PostgreSQL 用户账户，再执行此命令。该账户已在上一节介绍。

> **提示**
>
> 除了使用 -D 选项，也可以设置环境变量 PGDATA。

或者，也可以通过pg_ctl程序运行initdb，如下所示：

```
$ pg_ctl -D /usr/local/pgsql/data initdb
```

如果你打算用pg_ctl来启动和停止服务器（见第 17.3 节），这种方式可能更直观，因为这样pg_ctl就会成为你管理数据库服务器实例时使用的唯一命令。

如果指定的目录尚不存在，initdb会尝试创建它。当然，如果initdb对其父目录没有写权限，这会失败。通常建议让PostgreSQL用户不仅拥有数据目录，也拥有其父目录，这样就不会有这个问题。如果目标父目录也不存在，则需要先创建它；如果祖父目录不可写，就需要使用 root 权限。因此，过程可能如下所示：

```
root# mkdir /usr/local/pgsql
root# chown postgres /usr/local/pgsql
root# su postgres
```

```
postgres$ initdb -D /usr/local/pgsql/data
```

如果数据目录已存在且其中已有文件，initdb会拒绝运行；
这是为了防止意外覆盖现有安装。

由于数据目录包含存储在数据库中的全部数据，必须保护它免遭未经授权的访问。因此，initdb 会撤销除 PostgreSQL 用户以外所有人的访问权限。

不过，尽管目录内容是安全的，默认的客户端认证设置却允许任何本地用户连接到数据库，甚至成为数据库超级用户。如果你不信任其他本地用户，我们建议在运行 initdb 时使用 -W、--pwprompt 或 --pwfile 之一，为数据库超级用户指定密码。另外，指定 -A md5 或 -A password，以避免使用默认的 trust 认证方式；或者在运行 initdb 之后、首次启动服务器之前修改生成的 pg_hba.conf 文件。（其他合理方法包括使用 peer 认证，或使用文件系统权限限制连接。更多信息见第 19 章。）

initdb 还会初始化数据库集簇的默认区域设置。通常，它会直接采用环境中的区域设置，并将其应用于初始化的数据库。可以为数据库指定不同的区域设置；更多信息见第 22.1 节。特定数据库集簇中使用的默认排序顺序由 initdb 设置，虽然你可以使用不同的排序顺序创建新数据库，但 initdb 创建的模板数据库所使用的顺序，只有删除并重建这些数据库才能更改。使用 C 或 POSIX 以外的区域设置还会影响性能。因此，一开始就作出正确选择很重要。

initdb还会为数据库集簇设置默认字符集编码。
通常应选择与区域设置相匹配的字符集编码。详见第 22.3 节。

非C和非POSIX的区域设置依赖操作系统的排序规则库来决定字符集排序。
这会控制索引中存储键值的顺序。因此，无论是通过快照恢复、二进制流复制、
切换到不同操作系统，还是升级操作系统，
都不能让一个集簇切换到不兼容版本的排序规则库。

17.2.1. 二级文件系统的使用

很多安装会把数据库集簇建在机器“根”卷以外的文件系统（卷）上。如果这样做，
不建议把二级卷的最顶层目录（挂载点）直接用作数据目录。
最佳实践是在挂载点目录下创建一个由PostgreSQL用户拥有的目录，
再在其下创建数据目录。这样可以避免权限问题，尤其是在执行pg_upgrade之类的操作时；
同时也能确保二级卷脱机时以干净的方式失败。

17.2.2. 使用网络文件系统

许多安装会在网络文件系统上创建数据库集簇。有时通过 NFS 完成，有时则使用内部采用 NFS 的网络附加存储（NAS）设备。PostgreSQL 不会对 NFS 文件系统做特殊处理，也就是说，它假定 NFS 的行为与本地连接的驱动器完全一致。如果客户端或服务器端的 NFS 实现不提供标准文件系统语义，就可能导致可靠性问题（参见http://www.time-travellers.org/shane/papers/NFS_considered_harmful.html）。具体而言，对 NFS 服务器的延迟（异步）写入可能导致数据损坏。如果可能，应以同步方式（不使用缓存）挂载 NFS 文件系统，以避免这一风险。此外，不建议对 NFS 文件系统使用软挂载。

存储区域网络（SAN）通常使用 NFS 以外的通信协议，可能存在也可能不存在此类风险。建议查阅供应商文档中有关数据一致性保证的说明。PostgreSQL 的可靠性不可能高于它所使用的文件系统。

17.3. 启动数据库服务器

在任何人访问数据库之前，必须先启动数据库服务器。数据库服务器程序名为 postgres。其中，postgres 程序必须知道应到哪里查找它要使用的数据。这通过 -D 选项指定。因此，启动服务器最简单的方法是：

```
$ postgres -D /usr/local/pgsql/data
```

这会让服务器在前台运行。必须先登录 PostgreSQL 用户账户再执行此操作。如果不使用 -D，服务器会尝试使用环境变量所指定的数据目录，该变量为 PGDATA。如果也未提供该变量，则会失败。

通常最好在后台启动postgres。做法是使用常见的 Unix shell 语法：

```
$ postgres -D /usr/local/pgsql/data >logfile 2>&1 &
```

如上所示，把服务器的stdout和stderr输出保存到某个地方非常重要。这有助于审计，也有助于诊断问题。（关于日志文件处理的更深入讨论，见第 23.3 节。）

postgres还接受许多其他命令行选项。更多信息请见postgres参考页以及下面的第 18 章。

这些 shell 语法很快就会让人觉得繁琐。因此提供了包装器程序pg_ctl来简化一些任务。例如：

```
pg_ctl start -l logfile
```

这会在后台启动服务器，并把输出写入指定的日志文件。这里的 -D 选项含义与在 postgres 中相同。pg_ctl 还可用于停止服务器。

通常，你会希望在计算机启动时就启动数据库服务器。自动启动脚本依赖于操作系统。PostgreSQL 在 contrib/start-scripts 目录中附带了一些示例脚本。安装这些脚本需要 root 权限。

不同系统在引导时启动守护进程的惯例各不相同。许多系统有/etc/rc.local或/etc/rc.d/rc.local文件，其他系统则使用init.d或rc.d目录。无论采用哪种方式，服务器都必须由PostgreSQL用户账户而不是 root或其他用户来启动。因此，你大概应该在命令中使用su postgres -c '...'这种形式。例如：

```
su postgres -c 'pg_ctl start -D /usr/local/pgsql/data -l serverlog'
```

下面是一些与操作系统相关的补充建议。（在每种情况下，请确保使用正确的安装目录和用户名；这里展示的是通用值。）

- 对于FreeBSD，请查看PostgreSQL源码发布包中的contrib/start-scripts/freebsd文件。
- 在OpenBSD上，把以下内容加入/etc/rc.local：

  ```
  if [ -x /usr/local/pgsql/bin/pg_ctl -a -x /usr/local/pgsql/bin/postgres ]; then
      su -l postgres -c '/usr/local/pgsql/bin/pg_ctl start -s -l /var/postgresql/log -D /usr/local/pgsql/data'
      echo -n ' postgresql'
  fi
  ```

- 在Linux系统上，可以把

  ```
  /usr/local/pgsql/bin/pg_ctl start -l logfile -D /usr/local/pgsql/data
  ```

  加入/etc/rc.d/rc.local或/etc/rc.local，或者查看PostgreSQL源码发布包中的contrib/start-scripts/linux文件。

- 在NetBSD上，可根据偏好使用FreeBSD或Linux的启动脚本。
- 在Solaris上，创建一个名为/etc/init.d/postgresql的文件，其中包含以下内容：

  ```
  su - postgres -c "/usr/local/pgsql/bin/pg_ctl start -l logfile -D /usr/local/pgsql/data"
  ```

然后，在/etc/rc3.d中为它创建一个名为S99postgresql的符号链接。

当服务器运行时，其PID会保存在数据目录中的postmaster.pid文件里。
这可以防止多个服务器实例在同一个数据目录上运行，也可以用来关闭服务器。

17.3.1. 服务器启动失败

服务器启动失败有几种常见原因。请检查服务器日志文件，
或者手工启动服务器（不要重定向标准输出和标准错误），看看出现了什么错误消息。
下面将更详细地解释一些最常见的错误消息。

```
LOG:  could not bind IPv4 socket: Address already in use
HINT:  Is another postmaster already running on port 5432? If not,
 wait a few seconds and retry.
FATAL:  could not create TCP/IP listen socket
```

这通常就是字面上的意思：你试图在一个已有服务器运行的端口上再启动另一个服务器。
不过，如果内核错误消息不是Address already in use或类似变体，也可能是别的问题。
例如，试图在一个保留端口上启动服务器，可能会得到类似下面的消息：

```
$ postgres -p 666
LOG:  could not bind IPv4 socket: Permission denied
HINT:  Is another postmaster already running on port 666? If not,
 wait a few seconds and retry.
FATAL:  could not create TCP/IP listen socket
```

像下面这样的消息：

```
FATAL:  could not create shared memory segment: Invalid argument
DETAIL:  Failed system call was shmget(key=5440001, size=
4011376640, 03600).
```

可能表示内核对共享内存大小的限制小于 PostgreSQL 尝试创建的工作区（本例中为4011376640 字节）。也可能表示内核中根本没有配置 System V 风格的共享内存支持。作为临时解决办法，可以尝试以少于通常数量的缓冲区启动服务器（shared_buffers）。
最终还是需要重新配置内核，增大允许的共享内存大小。
如果尝试在同一台机器上启动多个服务器，而它们请求的总空间超出内核限制，
也可能看到这条消息。

像下面这样的错误：

```
FATAL:  could not create semaphores: No space left on device
DETAIL:  Failed system call was semget(5440126, 17, 03600).
```

并不意味着你已经用光了磁盘空间。它表示你的内核对System V信号量数量的限制，小于PostgreSQL想要创建的数量。和上面一样，你也许可以通过减少允许的连接数（max_connections）来暂时绕过这个问题，但最终还是应该提高内核限制。

如果收到“illegal system call”错误，很可能是内核根本不支持共享内存或信号量。在这种情况下，唯一的办法就是重新配置内核以启用这些功能。

关于配置System V IPC功能的细节请见第 17.4.1 节。

17.3.2. 客户端连接问题

客户端一侧可能出现的错误种类很多，并且依赖于具体应用，
但其中有一些可能直接与服务器的启动方式有关。除下面列出的几种情况外，
其他问题应查阅相应客户端应用的文档。

```
psql: could not connect to server: Connection refused
        Is the server running on host "server.joe.com" and
 accepting
        TCP/IP connections on port 5432?
```

这种常见的失败表示“我找不到可通信的服务器”。尝试 TCP/IP 通信时，它就会显示为上述形式。一个常见错误是忘记配置服务器以允许 TCP/IP 连接。

另一种情况是，尝试通过 Unix 域套接字与本地服务器通信时，会得到以下消息：

```
psql: could not connect to server: No such file or directory
        Is the server running locally and accepting
        connections on Unix domain socket "/tmp/.s.PGSQL.5432"?
```

最后一行有助于确认客户端是否正尝试连接到正确的位置。
如果那里实际上没有运行服务器，内核错误消息通常会像示例那样是 `Connection refused` 或 `No such file or directory`。（要注意，在这里 `Connection refused` 并不表示服务器收到了你的连接请求并拒绝了它；那种情况会产生另一条消息，如第 19.4 节所示。）
其他错误消息，例如 `Connection timed out`，可能表示更底层的问题，例如网络不通。

17.4. 管理内核资源

一个大型PostgreSQL安装会很快耗尽各种操作系统资源限制。（在某些系统上，
出厂默认值非常低，你甚至不需要真正“大型”的安装就会遇到问题。）
如果你遇到过这类问题，请继续阅读。

17.4.1. 共享内存和信号量

共享内存和信号量统称为“System V IPC”（连同消息队列，但消息队列与PostgreSQL无关）。几乎所有现代操作系统都提供这些特性，
但其中许多系统默认没有开启它们或尺寸不足，尤其是随着可用 RAM 和数据库应用的需求增长。（在 Windows 上，PostgreSQL 为这些设施提供了自己的替代实现，因此本节的大部分内容可以忽略。）

完全缺少这些功能时，通常会在服务器启动时出现 Illegal system call 错误。在这种情况下，除了重新配置内核别无选择。没有它们，PostgreSQL 就无法工作。不过，在现代操作系统中，这种情况很少见。

当PostgreSQL超出各种硬性IPC限制之一时，服务器会拒绝启动，
并应留下带有指导性的错误消息，说明问题所在以及应如何处理（另见第 17.3.1 节）。相关内核参数在不同系统上的命名基本一致，表 17.1给出了概览；不过，设置它们的方法却各不相同。下面给出一些平台上的建议。

表 17.1. System V IPC参数

名称	描述	合理的值
`SHMMAX`	共享内存段的最大尺寸（字节）	至少数 MB（见正文）
`SHMMIN`	共享内存段的最小尺寸（字节）	1
`SHMALL`	可用共享内存的总量（字节或页面）	如果是字节，同`SHMMAX`；如果是页面，为`ceil(SHMMAX/PAGE_SIZE)`
`SHMSEG`	每个进程的最大共享内存段数目	只需要 1 段，但是默认值高很多
`SHMMNI`	系统范围内的最大共享内存段数目	像`SHMSEG`外加其他应用的空间
`SEMMNI`	信号量标识符（即，集合）的最大数目	至少 `ceil((max_connections + autovacuum_max_workers + 5) / 16)`

名称	描述	合理的值
SEMMNS	系统范围内的最大信号量数目	ceil((max_connections + autovacuum_max_workers + 5) / 16) * 17，并为其他应用保留空间
SEMMSL	每个集合中信号量的最大数目	至少 17
SEMMAP	信号量映射中的项数	见文本
SEMVMX	信号量的最大值	至少 1000（默认值常常是 32767，如非必要不要更改）

最重要的共享内存参数是 SHMMAX，即共享内存段的最大字节数。如果你从 shmget 得到类似 “Invalid argument” 的错误消息，很可能是超出了这个限制。所需共享内存段的大小取决于若干 PostgreSQL 配置参数，如表 17.2所示。（你可能得到的任何错误消息都会包含失败分配请求的确切大小。）作为一种临时解决方案，你可以降低其中某些设置来避免失败。虽然 SHMMAX 小到 2 MB 时也能让 PostgreSQL 运行，但要获得可接受的性能需要大得多的值。理想的设置在几百 MB 到几 GB 之间。

有些系统还对系统中的共享内存总量有限制（SHMALL）。请确保它足以容纳 PostgreSQL 以及其他使用共享内存段的应用程序。注意，在许多系统上，SHMALL 以页而非字节为单位。

不太可能出问题的是共享内存段的最小尺寸（SHMMIN），对PostgreSQL来说应该最多大约是 500 kB。而系统范围（SHMMNI）或每个进程（SHMSEG）的最大共享内存段数目不太可能会导致问题，除非你的系统把它们设成零。

PostgreSQL 为每个允许的连接（max_connections）和自动清理工作进程（autovacuum_max_workers）各使用一个信号量，每 16 个组成一组。每组还包含第 17 个信号量，其中存放一个“魔数”，用来检测与其他应用程序使用的信号量集的冲突。
系统中的信号量最大数量由 SEMMNS 设置，因此它必须至少等于 max_connections 加上 autovacuum_max_workers，再为每 16 个允许的连接和工作进程额外增加一个信号量（参见表 17.1中的公式）。参数 SEMMNI 限制系统中同时存在的信号量集的数量。因此，此参数必须至少为 ceil((max_connections + autovacuum_max_workers + 5) / 16)。降低允许的连接数，可以临时规避 semget 函数的失败；此类失败通常会报告令人困惑的 “No space left on device” 。

在某些情况下，可能还需要增大SEMMAP，使其至少与SEMMNS处于同一数量级。
此参数定义信号量资源映射的大小；映射中每个连续的可用信号量块都需要占用一项。
每当一个信号量集合被释放时，它要么会并入与该释放块相邻的现有项，
要么会登记为一个新的映射项。如果映射已满，被释放的信号量就会丢失（直到重启）。
因此，随着时间推移，信号量空间碎片化可能会导致可用信号量少于应有数量。

SEMMSL 参数决定一个信号量集中可以包含多少个信号量，对于PostgreSQL，它必须至少为17。

与“信号量撤销”有关的其他各种设置，如SEMMNU和SEMUME，不会影响PostgreSQL。

AIX

至少从 5.1 版起，应该不需要对 SHMMAX 等参数进行任何特殊配置，因为它似乎已配置为允许将全部内存用作共享内存。这也是 DB/2 等其他数据库常用的配置。

不过，可能需要修改 /etc/security/limits 中的全局 ulimit 信息，因为文件大小（fsize）和文件数量（nofiles）的默认硬限制可能过低。

FreeBSD

默认设置只适合小型安装（例如，默认 SHMMAX 为 32 MB）。可以通过 sysctl 或 loader 接口进行修改。下列参数可用sysctl设置：

```
# sysctl kern.ipc.shmall=32768
```

```
# sysctl kern.ipc.shmmax=134217728
```

要让这些设置在重启之后也保持，请修改/etc/sysctl.conf。

这些信号量相关设置对于 sysctl 是只读的；设置它们时，可修改 /boot/loader.conf：

```
kern.ipc.semmni=256
kern.ipc.semmns=512
kern.ipc.semmnu=256
```

修改这些值后，需要重启才能使新设置生效。（注意：FreeBSD 不使用 SEMMAP。旧版本会接受但忽略 kern.ipc.semmap 设置；新版本则干脆拒绝它。）

你可能还希望配置内核，把共享内存锁定在 RAM 中，防止其被换出到交换区。这可以通过 sysctl 设置 kern.ipc.shm_use_phys 来实现。

如果通过启用 sysctl 的 security.jail.sysvipc_allowed 在 FreeBSD jail 中运行，应让不同 jail 中的 postmaster 使用不同的操作系统用户。这能提高安全性，因为它可以防止非 root 用户干扰不同 jail 中的共享内存或信号量，也能让 PostgreSQL 的 IPC 清理代码正常工作。（在 FreeBSD 6.0 及更高版本中，IPC 清理代码无法正确检测其他 jail 中的进程，导致无法在不同 jail 中使用同一端口运行 postmaster。）

4.0 之前的 FreeBSD 版本与 OpenBSD 的行为相同（见下文）。

NetBSD

在 NetBSD 5.0 及更高版本中，可以通过以下命令调整 IPC 参数：sysctl。例如：

```
# sysctl -w kern.ipc.semmni=100
```

要让这些设置在重启后仍然生效，请修改 /etc/sysctl.conf。

你可能还希望配置内核，把共享内存锁定在 RAM 中，防止其被换出到交换区。这可以通过 sysctl 设置 kern.ipc.shm_use_phys 来实现。

5.0 之前的 NetBSD 版本与 OpenBSD 的行为相同（见下文），但设置参数时应使用关键字 options，而非 option。

OpenBSD

编译内核时需要启用 SYSVSHM 和 SYSVSEM 选项（它们默认启用）。共享内存的最大大小由选项 SHMMAXPGS（以页为单位）决定。以下示例展示了如何设置各项参数：

```
option        SYSVSHM
option        SHMMAXPGS=4096
option        SHMSEG=256

option        SYSVSEM
option        SEMMNI=256
option        SEMMNS=512
option        SEMMNU=256
option        SEMMAP=256
```

你可能还希望配置内核，把共享内存锁定在 RAM 中，防止其被换出到交换区。这可以通过 sysctl 设置 kern.ipc.shm_use_phys 来实现。

HP-UX

默认设置通常足以满足普通安装的需求。在 HP-UX 10 上，SEMMNS 的出厂默认值为 128，对较大的数据库站点来说可能过低。

可以在 System Administration Manager（SAM）的 Kernel Configuration → Configurable Parameters 中设置 IPC 参数。完成后选择 Create A New Kernel。

Linux

默认最大段大小为 32 MB，这只够非常小型的 PostgreSQL 安装使用。默认最大总大小为 2097152 页。除使用“大页”的特殊内核配置外，页大小几乎总是 4096 字节（使用 `getconf PAGE_SIZE` 验证）。这意味着默认上限为 8 GB，通常够用，但并非总是如此。

可以通过以下接口更改共享内存大小设置：`sysctl`。例如，要允许 16 GB：

```
$ sysctl -w kernel.shmmax=17179869184
$ sysctl -w kernel.shmall=4194304
```

此外，可以将这些设置保存在以下文件中，使其在重启后仍然生效：`/etc/sysctl.conf`。强烈建议这样做。

很旧的发行版可能没有 `sysctl` 程序，但可以通过操作 `/proc` 文件系统进行等效更改：

```
$ echo 17179869184 >/proc/sys/kernel/shmmax
$ echo 4194304 >/proc/sys/kernel/shmall
```

其他默认值都相当充裕，通常不需要更改。

Mac OS X

在 Mac OS X 中配置共享内存的推荐方法是创建一个名为`/etc/sysctl.conf`的文件，其中包含这样的变量赋值：

```
kern.sysv.shmmax=4194304
kern.sysv.shmmin=1
kern.sysv.shmmni=32
kern.sysv.shmseg=8
kern.sysv.shmall=1024
```

注意，在某些 OS X 版本中，全部五个共享内存参数都必须在`/etc/sysctl.conf`中设置，否则这些值会被忽略。

请注意，近期 OS X 版本会忽略将 `SHMMAX` 设为非 4096 整数倍的值的尝试。

在这个平台上，`SHMALL` 以 4 kB 的页为单位。

在较旧的 OS X 版本中，必须重启才能使共享内存参数的变更生效。从 10.5 起，除了 `SHMMNI`，都可以使用 sysctl 在线更改。但最好仍通过 `/etc/sysctl.conf` 设置所需的值，以便重启后保留这些值。

文件 `/etc/sysctl.conf` 只在 OS X 10.3.9 及更高版本中生效。如果运行的是更早的 10.3.x 版本，必须编辑文件 `/etc/rc` 并更改以下命令中的值：

```
sysctl -w kern.sysv.shmmax
sysctl -w kern.sysv.shmmin
sysctl -w kern.sysv.shmmni
sysctl -w kern.sysv.shmseg
sysctl -w kern.sysv.shmall
```

注意，`/etc/rc` 通常会被 OS X 系统更新覆盖，因此每次更新后可能都需要重新进行这些编辑。

在 OS X 10.2 及更早版本中，请改为编辑 `/System/Library/StartupItems/SystemTuning/SystemTuning` 文件中的这些命令。

SCO OpenServer

在默认配置中，每个共享内存段只允许 512 kB。要提高此设置，首先切换到目录`/etc/conf/cf.d`。要显示`SHMMAX`的当前值，运行：

```
./configure -y SHMMAX
```

要为`SHMMAX`设置新值，运行：

```
./configure SHMMAX=value
```

其中*`value`*是你要使用的新值（以字节为单位）。设置`SHMMAX`后，重新构建内核：

```
./link_unix
```

然后重启。

Solaris 2.6 至 2.9（Solaris 6 至 Solaris 9）

可以在以下文件中更改相关设置：`/etc/system`。例如：

```
set shmsys:shminfo_shmmax=0x2000000
set shmsys:shminfo_shmmin=1
set shmsys:shminfo_shmmni=256
set shmsys:shminfo_shmseg=256

set semsys:seminfo_semmap=256
set semsys:seminfo_semmni=512
set semsys:seminfo_semmns=512
set semsys:seminfo_semmsl=32
```

需要重启才能使变更生效。另请参见http://sunsite.uakom.sk/sunworldonline/swol-09-1997/swol-09-insidesolaris.html，了解旧版 Solaris 中共享内存的信息。

Solaris 2.10（Solaris 10）及更高版本
OpenSolaris

在 Solaris 10 及更高版本和 OpenSolaris 中，默认共享内存和信号量设置足以满足大多数 PostgreSQL 应用的需求。Solaris 现在默认将 `SHMMAX` 设为系统 RAM 的四分之一。要进一步调整此设置，请使用与以下用户关联的项目设置：`postgres`。例如，执行以下命令时使用的用户为 `root`：

```
projadd -c "PostgreSQL DB User" -K "project.max-shm-memory=(
privileged,8GB,deny)" -U postgres -G postgres user.postgres
```

该命令会新增`user.postgres`项目，并把`postgres`用户的共享内存上限设为 8GB；它会在该用户下次登录时或重启PostgreSQL时（不是重新加载）生效。
上述示例假定PostgreSQL由`postgres`组中的`postgres`用户运行。
不需要重启操作系统。

对于会承载大量连接的数据库服务器，我们还建议修改以下内核设置：

```
project.max-shm-ids=(priv,32768,deny)
project.max-sem-ids=(priv,4096,deny)
project.max-msg-ids=(priv,4096,deny)
```

此外，如果你正在某个区（zone）中运行PostgreSQL，
可能也需要提高该区的资源使用限制。关于`projects`和`prctl`的更多信息，请参见System Administrator's Guide中的 "Chapter 2: Projects and Tasks"。

UnixWare

在 UnixWare 7 上，默认配置中共享内存段的最大尺寸为 512 kB。要显示SHMMAX的当前值，运行：

```
/etc/conf/bin/idtune -g SHMMAX
```

这会显示当前值、默认值、最小值和最大值。要为SHMMAX设置新值，运行：

```
/etc/conf/bin/idtune SHMMAX value
```

其中*value*是你要使用的新值（以字节为单位）。设置SHMMAX后，重新构建内核：

```
/etc/conf/bin/idbuild -B
```

然后重启。

表 17.2. PostgreSQL 共享内存用量

用途	所需共享内存字节数的近似值（8.3 版起）
连接	(1800 + 270 * max_locks_per_transaction) * max_connections
自动清理工作进程	(1800 + 270 * max_locks_per_transaction) * autovacuum_max_workers
预备事务	(770 + 270 * max_locks_per_transaction) * max_prepared_transactions
共享磁盘缓冲	(block_size + 208) * shared_buffers
WAL 缓冲	(wal_block_size + 8) * wal_buffers
固定空间需求	770 kB

17.4.2. 资源限制

类 Unix 操作系统会施加多种资源限制，这些限制可能干扰 PostgreSQL 服务器的运行。其中尤其重要的是：每个用户可用的进程数限制、每个进程可打开文件数限制，
以及每个进程可用内存量限制。每一种限制都有“硬”限制和“软”限制。
实际生效的是软限制，但用户可以自行把它调高到不超过硬限制；硬限制则只能由 root 用户修改。系统调用setrlimit负责设置这些参数。shell 的内置命令ulimit（Bourne shell）或limit（csh）可用于在命令行控制资源限制。在 BSD 派生系统上，/etc/login.conf文件控制登录时设置的各种资源限制。详细信息请参阅操作系统文档。
相关参数有maxproc、openfiles和datasize。例如：

```
default:\
...
        :datasize-cur=256M:\
        :maxproc-cur=256:\
        :openfiles-cur=256:\
...
```

（-cur表示软限制；把它改为-max则表示设置硬限制。）

内核还可能对某些资源施加系统范围的限制。

- 在Linux上，内核参数 /proc/sys/fs/file-max确定内核支持的最大打开文件数。可以向该文件写入一个不同的数字来修改它，也可以在/etc/sysctl.conf中添加相应赋值。每个进程的文件数上限是在内核编译时固定的；请参阅 /usr/src/linux/Documentation/proc.txt获取更多信息。

PostgreSQL服务器为每个连接使用一个进程，
因此你至少应提供与允许连接数相同数量的进程，再加上系统其他部分所需的进程数。
通常这不是问题，但如果你在一台机器上运行多个服务器，资源可能就会变得紧张。

打开文件数的出厂默认限制通常被设为“对共享环境友好”的值，
也就是允许许多用户在一台机器上共存，而不会占用不成比例的系统资源。
如果你在一台机器上运行很多服务器，这也许正合适；但在专用服务器上，
你可能会希望提高这个限制。

另一方面，有些系统允许单个进程打开非常多的文件；如果不止少数几个进程都这么做，
就很容易超过系统范围的限制。如果你遇到这种情况，又不想修改系统范围的限制，
可以设置PostgreSQL的max_files_per_process配置参数来限制其打开文件的消耗。

17.4.3. Linux 内存过量分配

在 Linux 2.4 及更高版本中，默认的虚拟内存行为对 PostgreSQL 并非最佳。由于内核实现内存“overcommit”（过量分配）的方式，如果 PostgreSQL 或其他进程的内存需求导致系统耗尽虚拟内存，内核可能会终止 PostgreSQL 的 postmaster（主管服务器进程）。

如果发生这种情况，你会看到类似下面的内核消息（至于应到哪里查看此类消息，
请参阅你的系统文档和配置）：

```
Out of Memory: Killed process 12345 (postgres).
```

这表示 `postgres` 进程因内存压力而被终止。尽管现有数据库连接仍会继续正常运行，但新连接将不再被接受。要恢复服务，必须重启PostgreSQL。

避免该问题的一种办法，是让
PostgreSQL运行在一台你能确定不会被其他进程耗尽内存的机器上。如果内存紧张，
增加操作系统交换空间也有助于避免这个问题，因为内存不足（OOM）
杀手只有在物理内存和交换空间都耗尽时才会被触发。

如果导致系统耗尽内存的正是 PostgreSQL 自身，那么你可以通过调整配置来避免该问题。在某些情况下，调低与内存相关的配置参数会有帮助，尤其是 `shared_buffers` 和 `work_mem`。在其他情况下，允许数据库服务器接受过多连接也可能导致这一问题。很多时候，更好的做法可能是减小 `max_connections`，并转而使用外部连接池软件。

在 Linux 2.6 及更高版本中，可以修改内核的行为，使其不会“过量分配”内存。虽然此设置无法完全阻止 OOM 杀手[1]被调用，但会显著降低发生概率，从而使系统行为更健壮。方法是选择严格的过量分配模式，使用以下命令：`sysctl`：

```
sysctl -w vm.overcommit_memory=2
```

或者在以下文件中加入等效条目：`/etc/sysctl.conf`。你可能还希望修改相关设置 `vm.overcommit_ratio`。详细信息参见内核文档文件 `Documentation/vm/overcommit-accounting`。

另一种方法可在修改或不修改 `vm.overcommit_memory` 的情况下使用：把 postmaster 进程专属的`oom_score_adj`值设为 `-1000`，从而保证它不会成为 OOM 杀手的目标。最简单的做法是在 postmaster 启动脚本中、调用 postmaster 之前执行：

```
echo -1000 > /proc/self/oom_score_adj
```

请注意，这个操作必须以 root 身份完成，否则不会生效；因此，由 root 拥有的启动脚本是最容易执行该操作的位置。如果这样做，还可以在构建 PostgreSQL 时向 `CPPFLAGS` 添加 `-DLINUX_OOM_SCORE_ADJ=0`。这会使 postmaster 子进程以常规的 `oom_score_adj`值零运行，以便 OOM 杀手在需要时仍可将它们作为目标。

[1] http://lwn.net/Articles/104179/

较旧的 Linux 内核不提供 /proc/self/oom_score_adj，但可能提供名为 /proc/self/oom_adj 的同类功能的早期版本。其工作方式相同，只是禁用值为 -17 而非 -1000。PostgreSQL相应的构建标志是 -DLINUX_OOM_ADJ=0。

> **注意**
>
> 据报告，某些厂商的 Linux 2.4 内核包含 2.6 版过量分配 sysctl 参数的早期实现。但是，在没有相关代码的 2.4 内核上将 vm.overcommit_memory 设为 2，会使情况更糟，而非更好。建议在 2.4 安装上尝试之前，检查实际内核源码（参见文件 mm/mmap.c 中的函数 vm_enough_memory），确认内核支持的功能。存在 overcommit-accounting 文档文件，不能作为该功能已存在的证据。如有任何疑问，请咨询内核专家或内核供应商。

17.5. 关闭服务器

有几种方法可以关闭数据库服务器。通过向主管进程发送不同的信号，可以控制关闭类型，该进程为 postgres。

SIGTERM

这是智能关闭模式。收到SIGTERM后，服务器禁止新连接，
但允许现有会话正常结束工作。仅在所有会话终止后才会关闭。
如果服务器处于在线备份模式，还会等待该模式结束。在线备份模式生效期间，
仍允许新的超级用户连接（这一例外允许超级用户连接以结束在线备份模式），
但不允许其他新连接。如果服务器在请求智能关闭时处于恢复状态，
则只有在所有常规会话终止后，恢复和流复制才会停止。

SIGINT

这是快速关闭模式。服务器禁止新连接并向所有现有服务器进程发送SIGTERM，
这将导致它们中止当前事务并迅速退出。然后等待所有服务器进程退出，最后关闭。
如果服务器处于在线备份模式，该模式将被终止，导致备份不可用。

SIGQUIT

这是立即关闭模式。主 postgres 进程将向所有子进程发送 SIGQUIT 并立即退出，而不对自身进行正常的关闭。子进程在收到 SIGQUIT 时同样会立即退出。这会导致下次启动时通过重放 WAL 日志执行恢复。仅建议在紧急情况下使用。

pg_ctl程序提供了一个便于发送这些信号以关闭服务器的接口。另外，在非 Windows 系统上，你也可以用kill直接发送这些信号。可以用ps程序，或者从数据目录中的postmaster.pid文件找到postgres进程的PID。例如，要执行一次快速关闭：

```
$ kill -INT `head -1 /usr/local/pgsql/data/postmaster.pid`
```

> **重要**
>
> 最好不要使用 SIGKILL 关闭服务器。这样做会阻止服务器释放共享内存和信号量，因此在启动新服务器之前，可能必须手动释放它们。此外，SIGKILL 会终止 postgres 进程，使其没有机会将信号转发给子进程，因此还必须手动逐个终止子进程。

要终止单个会话并允许其他会话继续运行，可使用pg_terminate_backend()（参阅表 9.59），或者向与该会话相关的子进程发送SIGTERM信号。

17.6. 升级 PostgreSQL 集簇

本节讨论如何把数据库数据从一个PostgreSQL发行版升级到较新的发行版。

PostgreSQL的主版本由版本号的前两组数字表示，例如 8.4。PostgreSQL的次版本由版本号的第三组数字表示，例如 8.4.2 就是 8.4 的第二个次版本。次版本从不改变内部存储格式，并且始终与同一主版本号中的更早和更晚次版本兼容，例如 8.4.2 与 8.4、8.4.1 和 8.4.6 兼容。要在兼容版本之间更新，只需在服务器关闭时替换可执行文件，然后重启服务器即可。数据目录保持不变 – 次版本升级就是这么简单。

对于 PostgreSQL 的major版本，内部数据存储格式可能会发生变化，因此升级会复杂得多。将数据迁移到新的主版本的传统方法是转储并重新载入数据库。此外还有其他方法可用，如下所述。

新的主版本通常还会引入一些用户可见的不兼容变化，因此应用程序可能也需要做出修改。所有用户可见的变化都列在发行说明（附录 E）中；请特别留意标为"迁移（Migration）"的小节。如果你跨越多个主版本进行升级，请务必阅读每个中间版本的发行说明。

谨慎的用户会希望在完全切换之前，先在新版本上测试客户端应用。因此，同时安装旧版本和新版本通常是个好主意。测试PostgreSQL主版本升级时，可以考虑以下几类可能的变化：

管理

管理员可用于监控和控制服务器的功能，在每个主版本中常常都会变化并有所增强。

SQL

通常这意味着 SQL 命令能力会增加，而行为不会变化，除非发行说明中特别提到。

库 API

通常像libpq这样的库只会增加新功能，除非发行说明中特别说明。

系统目录

系统目录的变化通常只影响数据库管理工具。

服务器端 C 语言 API

这涉及使用 C 语言编写的后端函数 API 的变更。这类变更会影响那些深入引用服务器内部后端函数的代码。

17.6.1. 通过pg_dump升级数据

一种升级方法是从某个主版本的 PostgreSQL 转储数据，再在另一个版本中重新载入。必须使用 pg_dump；文件系统级备份方法不起作用。（系统中有相应检查，会阻止你在不兼容版本的 PostgreSQL 上使用数据目录，因此尝试用错误版本的服务器启动某个数据目录，不会造成严重损害。）

建议使用来自新版 PostgreSQL 的 pg_dump 和 pg_dumpall 程序，以利用这些程序可能包含的改进。当前版本的转储程序可以读取从 7.0 起任意服务器版本的数据。

以下说明假定你现有的安装位于`/usr/local/pgsql`目录下，数据区域位于`/usr/local/pgsql/data`。请根据实际情况替换成你的路径。

1. 如果是在制作备份，请确认数据库此时没有正在进行更新。
   更新操作不会影响备份的完整性，但那些变更当然不会包含在备份中。必要时，可修改`/usr/local/pgsql/data/pg_hba.conf`（或等效文件）中的访问权限，禁止除你之外的其他人访问数据库。有关访问控制的更多信息见第 19 章。

   要备份整个数据库安装，请输入：

```
pg_dumpall > outputfile
```

制作备份时，你可以使用当前正在运行版本中的 pg_dumpall 命令；如果需要保留OID（例如把它们用作外键），可以在运行 pg_dumpall 时使用 -o 选项。不过，为了获得最佳结果，尽量使用PostgreSQL 9.2.24 自带的pg_dumpall命令，因为这一版本相较旧版本包含了缺陷修复和改进。这个建议看起来可能有些反常，
因为你这时还没有安装新版本；但如果你计划让新旧版本并行安装，
遵循这一建议是明智的。在那种情况下，你可以先正常完成新版本安装，
稍后再迁移数据，这样也能减少停机时间。

2. 关闭旧服务器：

```
pg_ctl stop
```

在那些会在开机时自动启动PostgreSQL的系统上，
可能已经有某个启动脚本能完成同样的工作。例如，在Red Hat Linux系统上，也许下面这条命令就可以：

```
/etc/rc.d/init.d/postgresql stop
```

有关服务器启动和停止的细节见第 17 章。

3. 如果是从备份恢复，请重命名或删除旧的安装目录，前提是它不是按版本区分的目录。
与其删除，不如重命名，这样如果你遇到问题需要回退，它仍然还在。请记住，
该目录可能会占用相当可观的磁盘空间。要重命名该目录，可以使用类似下面的命令：

```
mv /usr/local/pgsql /usr/local/pgsql.old
```

（请确保把该目录作为一个整体移动，这样相对路径才会保持不变。）

4. 安装新版本的 PostgreSQL，步骤见第 15.4 节。

5. 如有需要，创建一个新的数据库集簇。请记住，
执行这些命令时必须登录到专用数据库用户账户（如果你正在升级，
就已经拥有这个账户）。

```
/usr/local/pgsql/bin/initdb -D /usr/local/pgsql/data
```

6. 恢复先前的`pg_hba.conf`，以及你对`postgresql.conf`所做的任何修改。

7. 启动数据库服务器，同样要使用该专门的数据库用户账户：

```
/usr/local/pgsql/bin/postgres -D /usr/local/pgsql/data
```

8. 最后，使用以下命令从备份恢复数据：

```
/usr/local/pgsql/bin/psql -d postgres -f outputfile
```

这里应使用 psql。

若要把停机时间降到最低，可以把新服务器安装到不同目录中，
并让新旧两个服务器在不同端口上并行运行。这样就可以使用类似下面的命令：

```
pg_dumpall -p 5432 | psql -d postgres -p 5433
```

来传输数据。

17.6.2. 非转储升级方法

pg_upgrade模块允许把一次安装从一个PostgreSQL主版本就地迁移到下一个主版本。
升级可在几分钟内完成。

也可以使用某些复制方法，例如 Slony，创建一个运行较新版本 PostgreSQL 的备库。之所以可行，是因为 Slony 支持不同主版本 PostgreSQL 之间进行复制。该备库可以与旧服务器部署在同一台机器上，也可以部署在不同机器上。一旦它与主库（运行旧版本 PostgreSQL）同步，就可以切换主备角色，让备库成为主库，并关闭旧的数据库实例。
这种计划内切换只会为升级带来几秒钟的停机时间。

17.7. 防止服务器欺骗

当服务器正在运行时，恶意用户不可能取代正常的数据库服务器。然而，当服务器关闭时，本地用户却可能通过启动自己的服务器来冒充正常服务器。
这个伪造服务器可以读取客户端发出的密码和查询语句，但由于`PGDATA`目录仍然受目录权限保护，它无法返回任何真实数据。之所以会出现这种冒充，是因为任何用户都可以启动一个数据库服务器；而客户端除非经过专门配置，否则无法识别服务器是否为伪造。

防止`local`连接被欺骗的最简单办法，是使用一个 Unix 域套接字目录（unix_socket_directory），并且只允许受信任的本地用户对该目录拥有写权限。
这样可以防止恶意用户在其中创建自己的套接字文件。如果你担心某些应用程序仍会引用`/tmp`中的套接字文件，因此仍然容易受到欺骗，那么可以在操作系统启动时创建一个符号链接`/tmp/.s.PGSQL.5432`，使其指向迁移后的套接字文件。你可能还需要修改`/tmp`清理脚本，防止该符号链接被删除。

要防止 TCP 连接上的服务器欺骗，最佳办法是使用 SSL 证书，并确保客户端检查服务器证书。为此，服务器必须配置为仅接受 `hostssl` 连接（第 19.1 节），并且具备 SSL `server.key`（密钥）和 `server.crt`（证书）文件（第 17.9 节）。TCP 客户端必须使用 `sslmode=verify-ca` 或 `verify-full` 连接，并安装适当的根证书文件（第 31.1 节）。

17.8. 加密选项

PostgreSQL提供了多个层次的加密，并在防止数据因数据库服务器被盗、不诚实的管理员或不安全网络等原因而泄露方面提供了很高的灵活性。为了保护医疗记录、金融交易等敏感数据，也可能需要使用加密。

密码存储加密

默认情况下，数据库用户密码以 MD5 哈希形式存储，因此管理员无法得知分配给用户的实际密码。

指定列加密

pgcrypto模块允许以加密形式存储特定字段。这在只有部分数据敏感时很有用。
客户端提供解密密钥，数据在服务器端解密后再发送给客户端。

当数据被解密并在服务器与客户端之间传输时，
解密后的数据和解密密钥会在服务器端短暂存在。
这就给那些可以完全访问数据库服务器的人（例如系统管理员）
提供了一个短暂的机会来截获密钥和数据。

在 Linux 上，可以使用“loopback device”（环回设备）在文件系统之上叠加加密。这允许把整个文件系统分区在磁盘上加密，并由操作系统解密。在 FreeBSD 上，等价的设施称为 GEOM Based Disk Encryption（gbde），许多其他操作系统也支持这类功能，包括 Windows。

数据分区加密

这种机制可以防止在整台计算机或磁盘被盗时，从驱动器中直接读取未加密数据。
但它无法防御文件系统已挂载时的攻击，因为在挂载之后，

操作系统会提供数据的解密视图。不过，要挂载该文件系统，
就必须以某种方式把加密密钥提供给操作系统，
而有时这个密钥就存放在挂载该磁盘的主机上的某处。

跨网络安全密码

`MD5` 认证方法在把密码发送给服务器之前先在客户端对其进行双重加密：先基于用户名做一次 MD5 加密，再基于建立数据库连接时服务器发送的随机盐做一次加密。经过双重加密的值才是实际通过网络发送到服务器的内容。双重加密不仅能防止密码被窃取，
还能防止他人日后用同一个加密后的密码连接到数据库服务器。

跨网络加密数据

SSL 连接会加密通过网络发送的全部数据：密码、查询和返回的数据。`pg_hba.conf` 文件允许管理员指定哪些主机可以使用非加密连接（`host`），哪些主机必须使用 SSL 加密连接（`hostssl`）。此外，客户端可以指定仅通过 SSL 连接到服务器。也可以使用 Stunnel 或 SSH 加密传输。

SSL 主机认证

客户端和服务器都可以向对方提供 SSL 证书。这需要双方做一些额外配置，但它提供的认证强度高于单纯依赖密码。它可以防止某台计算机短暂冒充服务器，
只为读取客户端发送的密码；同时也有助于防御“man in the middle”攻击，即某台位于客户端与服务器之间的计算机冒充服务器，读取并转发双方之间的全部数据。

客户端加密

如果服务器所在机器的系统管理员不可信，那么就有必要由客户端自行加密数据。
这样一来，未加密的数据就永远不会出现在数据库服务器上。
数据在发送给服务器之前就在客户端加密，而查询结果也必须在客户端解密后才能使用。

17.9. 使用 SSL 的安全 TCP/IP 连接

PostgreSQL原生支持使用SSL连接来加密客户端与服务器之间的通信，以提高安全性。
这要求客户端和服务器系统上都安装了OpenSSL，并且在构建
PostgreSQL时启用了该支持（见第 15 章）。

如果构建时启用了 SSL 支持，那么在 `postgresql.conf` 中将参数ssl设为 `on`，就可以在启动 PostgreSQL 服务器时启用 SSL。服务器会在同一个 TCP 端口上同时监听普通连接和 SSL 连接，并与每个连接进来的客户端协商是否使用 SSL。默认情况下，这由客户端决定；关于如何把服务器配置为要求某些或全部连接必须使用 SSL，请参阅第 19.1 节。

PostgreSQL会读取系统范围的OpenSSL配置文件。默认情况下，该文件名为`openssl.cnf`，位于`openssl version -d`报告的目录中。可以通过把环境变量`OPENSSL_CONF`设为所需配置文件的名称，覆盖这一默认值。

OpenSSL支持种类繁多、强度不一的密码套件和认证算法。虽然可以在
OpenSSL配置文件中指定密码套件列表，但你也可以通过修改`postgresql.conf`中的ssl_ciphers，专门为数据库服务器指定要使用的密码套件。

> **注意**
>
> 使用`NULL-SHA`或`NULL-MD5`密码套件，可以在没有加密开销的情况下完成认证。
> 不过，中间人仍然能够读取并转发客户端与服务器之间的通信。此外，
> 与认证的开销相比，加密开销本身很小。基于这些原因，不建议使用 NULL 密码套件。

要以SSL模式启动服务器，必须存在包含服务器证书和私钥的文件。默认情况下，这些文件应分别命名为`server.crt`和`server.key`，并放在服务器的数据目录中；不过也可以通过配置参数ssl_cert_file和ssl_key_file指定其他名称和位置。在 Unix 系统上，`server.key`的权限必须禁止组和其他用户访问；可以通过`chmod 0600 server.key`实现。如果私钥受口令保护，服务器会提示输入口令，并在输入之前一直等待，不会启动。

在某些情况下，服务器证书可能不是由客户端直接信任的证书颁发机构签名，而是由一个 “intermediate” 证书颁发机构签名。要使用这样的证书，把签名机构的证书追加到 `server.crt` 文件，然后是其上级机构的证书，如此逐级直到一个受客户端信任的 “root” 机构。只要 `server.crt` 包含多于一个证书，就应当把根证书包含在内。

17.9.1. 使用客户端证书

要要求客户端提供受信任的证书，请把你信任的根证书颁发机构（CA）的证书放入数据目录中的 `root.crt` 文件，在 `postgresql.conf` 中将参数ssl_ca_file设为 `root.crt`，并在 `pg_hba.conf` 中相应的 `hostssl` 行上添加认证选项 `clientcert=1`。这样，SSL 连接启动时就会向客户端请求证书。（关于如何在客户端设置证书，请参见第 31.18 节。）服务器会验证客户端证书是否由某个受信任的证书颁发机构签名。如果设置了参数ssl_crl_file，还会检查证书吊销列表（CRL）条目。（展示 SSL 证书用法的示意图可参见http://h71000.www7.hp.com/doc/83final/ba554_90007/ch04s02.html。）

注意，`root.crt` 列出的是被认为可信任为客户端证书签名的顶层 CA。原则上它不必列出为服务器证书签名的 CA，不过在大多数情况下，那个 CA 对于客户端证书也会是受信任的。

`pg_hba.conf` 中的 `clientcert` 选项适用于所有认证方法，但只适用于指定为 `hostssl` 的行。未指定 `clientcert` 或将其设为 0 时，如果存在 `root.crt` 文件，服务器仍会依据它验证客户端提供的证书 – 但不会强制要求客户端提供证书。

如果正在设置客户端证书，可以考虑使用 `cert` 认证方法，让证书在提供连接安全性的同时也控制用户认证。详情参见第 19.3.10 节。

17.9.2. SSL 服务器文件用法

表 17.3概括了服务器端 SSL 设置相关的文件。

表 17.3. SSL 服务器文件用法

文件	内容	效果
ssl_cert_file (`$PGDATA/server.crt`)	服务器证书	发送给客户端，用于表明服务器身份
ssl_key_file (`$PGDATA/server.key`)	服务器私钥	证明服务器证书是其所有者发送的，并不说明证书所有者是值得信任的
ssl_ca_file (`$PGDATA/root.crt`)	受信任的证书颁发机构	检查客户端证书是由一个受信任的证书颁发机构签名的
ssl_crl_file (`$PGDATA/root.crl`)	被证书颁发机构吊销的证书	客户端证书不能出现在这个列表上

文件`server.key`、`server.crt`、`root.crt`和`root.crl`只在服务器启动期间检查；因此要让这些文件中的变更生效，必须重启服务器。

17.9.3. 创建自签名证书

要为服务器快速创建一个自签名证书，可以使用下面的OpenSSL命令：

```
openssl req -new -text -out server.req
```

按提示填写 openssl 询问的信息。确保把本地主机名作为“Common Name”输入；挑战口令可以留空。程序会生成一个受口令保护的密钥；它不接受少于四个字符的口令。
要去掉口令（如果希望服务器自动启动，就必须这样做），可以运行命令：

```
openssl rsa -in privkey.pem -out server.key
rm privkey.pem
```

输入旧口令以解锁现有密钥。然后执行：

```
openssl req -x509 -in server.req -text -key server.key -out server.
crt
```

把证书转换为自签名证书，并把密钥和证书复制到服务器查找它们的位置。最后执行：

```
chmod og-rwx server.key
```

如果该文件权限比这更宽松，服务器会拒绝使用它。
关于如何创建服务器私钥和证书的更多细节，请参阅OpenSSL文档。

虽然自签名证书可用于测试，但在生产环境中应使用由证书颁发机构（CA）（既可以是全球性 CA 之一，也可以是本地机构）签名的证书，以便客户端能够验证服务器的身份。如果客户端全部位于组织内部，推荐使用本地 CA。

17.10. 通过 SSH 隧道建立安全 TCP/IP 连接

可以使用SSH来加密客户端与PostgreSQL服务器之间的网络连接。如果配置得当，
即使客户端本身不支持 SSL，这种方法也能提供足够安全的网络连接。

首先确认在PostgreSQL服务器所在的同一台机器上有一个正常运行的SSH服务器，
并且你可以使用`ssh`以某个用户身份登录；然后就可以建立通向远程服务器的安全隧道。
安全隧道会监听本地端口，并把所有流量转发到远程机器上的某个端口。
发送到该远程端口的流量可以到达其localhost地址，或者在需要时到达其他绑定地址；
对远端来说，这些流量看起来并不是来自你的本地机器。
下面这个命令会创建一条从客户端机器到远程机器`foo.com`的安全隧道：

```
ssh -L 63333:localhost:5432 joe@foo.com
```

`-L`参数中的第一个数字 63333 是隧道在本地的端口号；它可以是任何未使用的端口。（IANA 保留 49152 到 65535 端口供私用。）其后的名称或 IP 地址是你要连接的远程绑定地址，这里是默认值localhost。第二个数字 5432 是隧道远端的端口，例如数据库服务器所使用的端口。要通过这条隧道连接数据库服务器，只需连接本地机器上的 63333 端口：

```
psql -h localhost -p 63333 postgres
```

对数据库服务器来说，它看到的是来自主机`foo.com`上用户`joe`对`localhost`
绑定地址的连接，并将应用为该用户到该绑定地址所配置的认证方式。请注意，
服务器不会认为这是 SSL 加密连接，因为实际上SSH服务器与PostgreSQL服务器之间并没有加密。不过这通常不会带来额外安全风险，因为两者位于同一台机器上。

要让这条隧道建立成功，你必须有权通过 `ssh` 以 `joe@foo.com` 身份连接，就像尝试使用 `ssh` 创建终端会话一样。

你也可以把端口转发设成这样：

```
ssh -L 63333:foo.com:5432 joe@foo.com
```

但这样一来，数据库服务器会把该连接看作是到达其`foo.com`绑定地址的连接，而默认设置 `listen_addresses = 'localhost'`并不会监听该地址。这通常不是你想要的结果。

如果你必须经由某个登录主机“跳转”到数据库服务器，那么一种可能的设置如下：

```
ssh -L 63333:db.foo.com:5432 joe@shell.foo.com
```

请注意，这种从shell.foo.com到db.foo.com的连接不会受到 SSH 隧道的加密保护。当网络受到各种限制时，SSH 还提供了许多其他配置方式。详情请参阅 SSH 文档。

> **提示**
>
> 还有一些其他应用也能提供安全隧道，其思路与刚才描述的 SSH 方法类似。

17.11. 在Windows上注册事件日志

要向操作系统注册一个Windows事件日志库，请执行以下命令：

regsvr32 *pgsql_library_directory*/pgevent.dll

这会创建被事件查看器使用的注册表项，默认事件源命名为PostgreSQL。

若要指定不同的事件源名称（见event_source），请使用/n和/i选项：

regsvr32 /n /i:*event_source_name* *pgsql_library_directory*/pgevent.dll

要从操作系统中注销该事件日志库，请执行以下命令：

regsvr32 /u [/i:*event_source_name*] *pgsql_library_directory*/pgevent.dll

> **注意**
>
> 要在数据库服务器中启用事件日志，请在postgresql.conf中修改log_destination，使其包含eventlog。

第 18 章 服务器配置

有许多配置参数会影响数据库系统的行为。本章第一节将介绍如何与配置参数交互。
后续各节将详细讨论每个参数。

18.1. 设置参数

18.1.1. 参数名称和值

所有参数名都是大小写不敏感的。每个参数都接受五种类型之一的值：布尔、整数、
浮点数、字符串或枚举。布尔值可以被写成 on、off、true、false、yes、no、1、0（都是大小写不敏感的）或者这些值的任何无歧义前缀。

有些设置指定内存或时间值。其中每一种都有一个隐含单位，即千字节、
块（通常是八千字节）、毫秒、秒或分钟之一。默认单位可以通过 pg_settings.unit 查到。为方便起见，也可以显式指定一个不同的单位。有效的内存单位有 kB（千字节）、MB（兆字节）和 GB（吉字节）；有效的时间单位有 ms（毫秒）、s（秒）、min（分钟）、h（小时）和 d（天）。注意内存单位的乘数是 1024 而不是 1000。

“枚举”类型参数的指定方式与字符串参数相同，但被限制为一组有限的值。允许的值可以从 pg_settings.enumvals 查到。枚举参数值是大小写不敏感的。

18.1.2. 通过配置文件影响参数

设置这些参数最基本的方法是编辑文件 postgresql.conf，它通常位于数据目录中。在数据库集簇目录初始化时，会安装该文件的一个默认副本。其内容示例如下：

```
# This is a comment
log_connections = yes
log_destination = 'syslog'
search_path = '"$user", public'
shared_buffers = 128MB
```

每行指定一个参数。名称和值之间的等号是可选的。
空白不重要（引号括起的参数值内部除外），空行会被忽略。井号（#）
表示该行余下部分是注释。不是简单标识符或数字的参数值必须用单引号括起。
要在参数值中嵌入单引号，可以写两个单引号（推荐）或使用反斜线转义单引号。
如果文件包含相同参数的多个条目，则忽略除最后一个之外的所有条目。

除了单个参数设置之外，postgresql.conf 文件还可以包含 include 指令，用来指定另一个要读取和处理的文件，就像把该文件插入到配置文件的这个位置一样。
这一特性允许把一个配置文件拆分成多个物理上独立的部分。include 指令的形式如下：

```
include 'filename'
```

如果文件名不是绝对路径，则会被解释为相对于引用它的配置文件所在目录的路径。include 可以嵌套。

还有一个 include_if_exists 指令，其行为与 include 相同，但在被引用文件不存在或无法读取时有所不同。普通的 include 会将其视为错误，而 include_if_exists 只会记录一条消息并继续处理引用它的配置文件。

主服务器进程每次收到SIGHUP信号（最简单的方法是用pg_ctl reload发送该信号）后都会重新读取这个配置文件。
主服务器进程还会把这个信号传播给所有正在运行的服务器进程，

这样现有的会话也能采用新值（要等待它们完成当前正在执行的客户端命令之后才会发生）。另外，你可以直接向一个单一服务器进程发送该信号。
有些参数只能在服务器启动时设置，在配置文件中对这些条目的修改将被忽略，直到下次服务器重启。配置文件中的非法参数设置也会在SIGHUP处理过程中被忽略（但是会记录日志）。

18.1.3. 设置参数的其他方法

设置这些配置参数的第二种方法是把它们作为 `postgres` 命令的命令行选项给出，例如：

```
postgres -c log_connections=yes -c log_destination='syslog'
```

命令行选项覆盖 `postgresql.conf` 中任何冲突的设置。注意这意味着你将无法通过编辑 `postgresql.conf` 即时更改该值，因此虽然命令行方法可能很方便，但日后可能会让你失去灵活性。

有时只给某个特定会话指定命令行选项很有用。为此可以在客户端使用环境变量 `PGOPTIONS`：

```
env PGOPTIONS='-c geqo=off' psql
```

（这适用于任何基于 libpq 的客户端应用，而不仅仅是 psql。）注意，对于在服务器启动时就固定、或者必须在 `postgresql.conf` 中指定的参数，这种方式无效。

此外，还可以把一组参数设置指派给某个用户或某个数据库。每当会话启动时，都会加载相关用户和数据库的默认设置。分别使用ALTER ROLE和ALTER DATABASE命令来配置这些设置。每个数据库的设置覆盖从 `postgres` 命令行或配置文件收到的任何内容，而后者又被每用户的设置覆盖；两者都会被每会话的设置覆盖。

有些参数可以在各个 SQL 会话中用SET命令更改，例如：

```
SET ENABLE_SEQSCAN TO OFF;
```

如果允许 `SET`，它将覆盖该参数所有其他来源的值。有些参数不能通过 `SET` 更改：例如，控制在不重启整个 PostgreSQL 服务器的情况下无法更改的行为的参数。此外，有些参数需要超级用户权限才能通过 `SET` 或 `ALTER` 更改。

18.1.4. 查看参数设置

SHOW命令允许查看所有参数的当前值。

虚拟表 `pg_settings` 也允许显示和更新会话运行时参数；详见第 45.64 节，其中有对不同变量类型及何时可以更改它们的描述。`pg_settings` 等价于 `SHOW` 和 `SET`，但用起来可能更方便，因为它可以与其他表连接，或者用任何想要的选择条件来查询。它还包含比 `SHOW` 可得的每个参数更多的信息。

18.2. 文件位置

除了已经提到过的`postgresql.conf`文件之外，PostgreSQL还使用另外两个手工编辑的配置文件，它们控制客户端认证（其使用在第 19 章中讨论）。默认情况下，所有三个配置文件都存放在数据库集簇的数据目录中。本节描述的参数允许配置文件放在别的地方（这么做可以简化管理，特别是如果配置文件被独立放置，可以很容易保证它得到恰当的备份）。

`data_directory`(`string`)

指定用于数据存储的目录。这个参数只能在服务器启动时设置。

config_file (string)

指定主服务器配置文件（通常叫postgresql.conf）。这个参数只能在postgres命令行上设置。

hba_file (string)

指定基于主机认证配置文件（通常叫pg_hba.conf）。这个参数只能在服务器启动的时候设置。

ident_file (string)

指定用于用户名称映射的配置文件（通常叫pg_ident.conf）。这个参数只能在服务器启动的时候设置。另见第 19.2 节。

external_pid_file (string)

指定服务器应创建的额外进程 ID（PID）文件的名称，供服务器管理程序使用。这个参数只能在服务器启动的时候设置。

在默认安装中不会显式设置以上参数。相反，命令行参数-D或者环境变量PGDATA指定数据目录，并且上述配置文件都能在数据目录中找到。

如果你想把配置文件放在别的地方而不是数据目录中，那么postgres -D命令行选项或者环境变量PGDATA必须指向包含配置文件的目录，并且postgresql.conf中（或者命令行上）的data_directory参数必须设置为数据目录的实际位置。请注意，data_directory将覆盖-D和PGDATA指定的数据目录位置，但是不覆盖配置文件的位置。

如果你愿意，可以使用选项config_file、hba_file和/或ident_file单独指定配置文件名称和位置。config_file只能在postgres命令行上指定，但是其他参数可以在主配置文件中设置。如果所有三个参数外加data_directory被显式地设置，则不必指定-D或PGDATA。

在设置任何这些参数时，相对路径将被解释为相对于启动 postgres 时所在目录的路径。

18.3. 连接和认证

18.3.1. 连接设置

listen_addresses (string)

指定服务器用于监听客户端应用连接的 TCP/IP 地址。此值采用以逗号分隔的主机名和/或数字 IP 地址列表的形式。特殊项*对应所有可用的 IP 接口。项0.0.0.0允许监听所有 IPv4 地址，而::允许监听所有 IPv6 地址。如果列表为空，服务器不会监听任何 IP 接口，此时只能通过 Unix 域套接字连接。默认值为localhost，只允许建立本地 TCP/IP “回环”连接。在客户端认证（第 19 章）允许对谁可以访问服务器进行细粒度控制的同时，listen_addresses 控制哪些接口接受连接尝试，这可以帮助防止在不安全的网络接口上重复恶意连接请求。此参数只能在服务器启动时设置。

port (integer)

服务器监听的 TCP 端口，默认是 5432。请注意，服务器监听的所有 IP 地址都使用同一个端口号。此参数只能在服务器启动时设置。

max_connections (integer)

决定数据库服务器允许的最大并发连接数。默认值通常是 100 个连接，但如果内核设置不支持（在 initdb 期间确定），则可能更少。这个参数只能在服务器启动时设置。

增大这个参数可能会导致PostgreSQL请求比操作系统默认配置所允许的更多的System V共享内存或信号量。如有必要，关于如何调整这些参数的信息请参见第 17.4.1 节。

当运行一个备库时，你必须设置这个参数等于或大于主库上的参数。否则，备库上将不允许查询。

`superuser_reserved_connections` (`integer`)

决定为 PostgreSQL 超级用户连接保留多少个连接“槽”。同时活跃的连接数最多始终只能达到max_connections。当活跃并发连接数至少达到 `max_connections` 减去 `superuser_reserved_connections` 时，新连接将只接受超级用户，并且不再接受新的复制连接。

默认值是 3 个连接。该值必须小于 `max_connections`。这个参数只能在服务器启动时设置。

`unix_socket_directory` (`string`)

指定服务器用于监听来自客户端应用连接的 Unix 域套接字的目录。默认值通常是`/tmp`，但可以在构建时更改。这个参数只能在服务器启动时设置。

除了套接字文件本身（名为`.s.PGSQL.`*`nnnn`*，其中*`nnnn`*是服务器的端口号），还会在`unix_socket_directory`目录中创建一个名为`.s.PGSQL.`*`nnnn`*`.lock`的普通文件。这两个文件都不应该被手工移除。

Windows 没有 Unix 域套接字，因此此参数在 Windows 上没有意义。

`unix_socket_group` (`string`)

设置 Unix 域套接字的所属组（套接字的所属用户总是启动服务器的用户）。可以与选项`unix_socket_permissions`一起用于对 Unix域连接进行访问控制。默认是一个空字符串，表示服务器用户的默认组。这个参数只能在服务器启动时设置。

Windows 没有 Unix 域套接字，因此此参数在 Windows 上没有意义。

`unix_socket_permissions` (`integer`)

设置 Unix 域套接字的访问权限。Unix 域套接字使用通常的 Unix 文件系统权限集。参数值应是以 `chmod` 和 `umask` 系统调用所接受格式指定的数字权限模式。（要使用惯用的八进制格式，数字必须以 `0`（零）开头。）

默认权限是 `0777`，表示任何人都可以连接。合理的其他取值包括 `0770`（仅属主和所属组，另见 `unix_socket_group`）和 `0700`（仅属主）。（注意，对 Unix 域套接字而言，只有写权限起作用，因此设置或撤销读权限和执行权限没有意义。）

此访问控制机制独立于第 19 章中描述的机制。

此参数只能在服务器启动时设置。

此参数对完全忽略套接字权限的系统无效，尤其是 Solaris（截至 Solaris 10）。在这些系统上，可以将 `unix_socket_directory` 指向一个仅向目标用户授予搜索权限的目录，以达到类似效果。Windows 没有 Unix 域套接字，因此此参数在 Windows 上也没有意义。

`bonjour` (`boolean`)

启用通过 Bonjour 通告服务器存在的功能。默认值为关闭。此参数只能在服务器启动时设置。

`bonjour_name` (`string`)

指定 Bonjour 服务名。空字符串 `''`（默认值）表示使用计算机名。如果编译时未启用 Bonjour 支持，则此参数会被忽略。此参数只能在服务器启动时设置。

`tcp_keepalives_idle` (`integer`)

指定在没有活动多少秒之后，TCP 应向客户端发送 keepalive 消息。值 0 表示使用系统默认值。此参数仅在支持`TCP_KEEPIDLE`或等效套接字选项的系统以及 Windows 上可用；在其他系统上，它必须为零。在通过 Unix 域套接字连接的会话中，此参数会被忽略，并始终读作零。

> **注意**
>
> 在 Windows 上，值 0 会将此参数设置为 2 小时，因为 Windows 不提供读取系统默认值的方法。

`tcp_keepalives_interval` (`integer`)

指定未被客户端确认收到的 TCP keepalive 消息在多少秒后应被重传。值 0 表示使用系统默认值。此参数仅在支持`TCP_KEEPINTVL`或等效套接字选项的系统以及 Windows 上可用；在其他系统上，它必须为零。在通过 Unix 域套接字连接的会话中，此参数会被忽略，并始终读作零。

> **注意**
>
> 在 Windows 上，值 0 会将此参数设置为 1 秒，因为 Windows 不提供读取系统默认值的方法。

`tcp_keepalives_count` (`integer`)

指定在服务器与客户端之间的连接被视为中断之前，可以丢失多少个 TCP keepalive 消息。值 0 表示使用系统默认值。这个参数只有在支持 `TCP_KEEPCNT` 或等效套接字选项的系统上才可用；在其他系统上，它必须为零。在通过 Unix 域套接字连接的会话中，这个参数会被忽略，并始终读作零。

> **注意**
>
> Windows 不支持此参数，它必须为零。

18.3.2. 安全和认证

`authentication_timeout` (`integer`)

允许完成客户端认证的最长时间，以秒为单位。
如果一个客户端没有在这段时间里完成认证协议，服务器将关闭连接。
这样就避免了出问题的客户端无限制地占有一个连接。默认值是 1分钟（`1m`）。这个参数只能在服务器命令行上或者在`postgresql.conf`文件中设置。

`ssl` (`boolean`)

启用SSL连接。使用前请阅读第 17.9 节。默认值是`off`。这个参数只能在服务器启动时设置。SSL通信只能在 TCP/IP 连接上使用。

`ssl_ca_file` (`string`)

指定包含 SSL 服务器证书颁发机构（CA）的文件名。默认值为空，表示没有载入 CA 文件，并且客户端证书验证没有被执行。（在以前的 PostgreSQL 版本中，此文件名被硬编码为`root.crt`。）相对路径是相对于数据目录的。这个参数只能在服务器启动时设置。

`ssl_cert_file` (`string`)

指定包含 SSL 服务器证书的文件名。默认值是`server.crt`。相对路径是相对于数据目录的。这个参数只能在服务器启动时设置。

`ssl_crl_file` (`string`)

指定包含 SSL 服务器证书吊销列表（CRL）的文件名。默认为空，表示不加载 CRL 文件。（在以前的 PostgreSQL 版本中，此文件名被硬编码为`root.crl`。）相对路径是相对于数据目录的。这个参数只能在服务器启动时设置。

`ssl_key_file` (`string`)

指定包含 SSL 服务器私钥的文件名。默认值是`server.key`。相对路径是相对于数据目录的。这个参数只能在服务器启动时设置。

`ssl_ciphers` (`string`)

指定允许在安全连接上使用的 SSL 密码列表。支持的书目列表见 openssl 手册页。

`ssl_renegotiation_limit` (`integer`)

指定在一个SSL加密连接上可以流过多少数据之后将进行会话密钥的重新协商。
当可以检查大量流量时，重新协商能降低攻击者进行密码分析的机会，
但它也会带来很大的性能损失。该限制以发送和接收流量的总和来检查。
如果这个参数被设置为 0，则禁用重新协商。默认值为`0`。

注意

2009 年 11 月之前的 SSL 库在使用 SSL 重新协商时不安全，原因是 SSL 协议中的一个漏洞。作为针对该漏洞的权宜修复，一些厂商发布的 SSL 库不具备重新协商能力。如果在客户端或服务器上正在使用这类库，应当禁用 SSL 重新协商。

警告

由于 OpenSSL 中的 bug，启用 SSL 重新协商（即把 `ssl_renegotiation_limit`配置为非零）很可能导致诸如长连接中断之类的问题。

`password_encryption` (`boolean`)

在CREATE USER或ALTER ROLE中指定密码，且既没有写 `ENCRYPTED` 也没有写 `UNENCRYPTED` 时，此参数决定密码是否被加密。默认值为 `on`（加密密码）。

`krb_server_keyfile` (`string`)

设置服务器的Kerberos密钥文件的位置。详情请参考第 19.3.5 节或第 19.3.3 节。这个参数只能在`postgresql.conf`文件中或者服务器命令行上设置。

`krb_srvname` (`string`)

设置 Kerberos 服务名。详情见第 19.3.5 节。这个参数只能在`postgresql.conf`文件中或者服务器命令行上设置。

`krb_caseins_users` (`boolean`)

设置是否应该以大小写不敏感的方式对待GSSAPI用户名。默认值是`off`（大小写敏感）。这个参数只能在`postgresql.conf`文件中或者服务器命令行上设置。

db_user_namespace (boolean)

此参数启用各数据库独立的用户名。默认关闭。此参数只能在postgresql.conf文件中或在服务器命令行上设置。

如果启用此参数，你应以username@dbname的形式创建用户。连接的客户端传入username时，会在用户名后附加@和数据库名，然后由服务器查找此数据库专属的用户名。请注意，在 SQL 环境中创建名称包含@的用户时，需要用引号括起用户名。

启用此参数后，仍然可以创建普通的全局用户。只需在客户端指定用户名时附加@，例如joe@。服务器查找用户名之前会去掉@。

db_user_namespace会使客户端和服务器的用户名表示形式不同。认证检查始终使用服务器端的用户名，因此认证方法必须针对服务器端的用户名配置，而不是客户端的用户名。由于md5在客户端和服务器上都使用用户名作为盐值，md5不能与db_user_namespace一起使用。

> **注意**
>
> 此特性旨在作为找到完整解决方案之前的临时措施。届时将移除此选项。

18.4. 资源消耗

18.4.1. 内存

shared_buffers (integer)

设置数据库服务器用于共享内存缓冲区的内存量。默认值通常为 32 兆字节（32MB），但如果内核设置不支持，则可能更小（在 initdb 期间确定）。此设置必须至少为 128 千字节。（BLCKSZ 的非默认值会改变该最小值。）不过，要获得良好性能，通常需要远高于该最小值的设置。此参数只能在服务器启动时设置。

如果专用数据库服务器具有 1GB 或更多内存，shared_buffers 的合理初始值是系统内存的 25%。对于某些工作负载，将 shared_buffers 设得很大也有效，但由于 PostgreSQL 同时依赖操作系统缓存，将超过 40% 的内存分配给 shared_buffers 不太可能比更小的值效果更好。将 shared_buffers 设得更大时，通常还需要相应增加 checkpoint_segments，以便将大量新数据或已修改数据的写入过程分散到更长的时间内。

对于内存少于 1GB 的系统，适合使用更小的内存比例，以便为操作系统留出足够空间。另外，在 Windows 上，很大的 shared_buffers 设置并不那么有效。让该设置保持相对较低、更多地使用操作系统缓存，可能得到更好的结果。Windows 系统上 shared_buffers 的有效范围通常是从 64MB 到 512MB。

增大这个参数可能会导致PostgreSQL请求比操作系统默认配置所允许的更多的System V共享内存。如有必要，关于如何调整这些参数的信息请参见第 17.4.1 节。

temp_buffers (integer)

设置每个数据库会话使用的临时缓冲区的最大数量。这些是会话本地的缓冲区，仅用于访问临时表。默认值为 8 兆字节（8MB）。可以在单个会话内更改此设置，但必须在该会话首次使用临时表之前更改；此后尝试更改该值，对该会话不会产生影响。

会话会按需分配临时缓冲区，上限为 temp_buffers。对于实际不需要很多临时缓冲区的会话，将此参数设得较大时，开销仅为 temp_buffers 每增加一就多分配一个缓冲区

描述符，约为 64 字节。不过，如果实际使用了某个缓冲区，还会为它额外消耗 8192 字节（一般而言为 BLCKSZ 字节）。

max_prepared_transactions (integer)

设置可同时处于“预备”状态的事务的最大数量（见PREPARE TRANSACTION）。将此参数设为零（默认值）会禁用预备事务功能。此参数只能在服务器启动时设置。

如果不打算使用预备事务，应将此参数设为零，以防意外创建预备事务。
如果使用预备事务，通常应将 max_prepared_transactions 设为不小于max_connections的值，以便每个会话都能有一个待处理的预备事务。

运行备库时，必须将此参数设为与主库相同或更大的值。否则，
备库上将不允许执行查询。

增大这个参数可能会导致PostgreSQL请求比操作系统默认配置所允许的更多的System V共享内存。如有必要，关于如何调整这些参数的信息请参见第 17.4.1 节。

work_mem (integer)

指定内部排序操作和哈希表在写入临时磁盘文件之前可使用的内存量。默认值为 1 兆字节（1MB）。请注意，复杂查询可能并行执行多个排序或哈希操作，
每个操作在开始向临时文件写入数据之前，都可以使用此值指定的内存量。此外，
多个正在运行的会话也可能并发执行此类操作。因此，使用的总内存量可能是 work_mem 值的数倍；选择此值时必须考虑这一点。排序操作用于 ORDER BY、DISTINCT 和归并连接。哈希表用于哈希连接、基于哈希的聚合、以及基于哈希的 IN 子查询处理。

maintenance_work_mem (integer)

指定维护操作（如 VACUUM、CREATE INDEX 和 ALTER TABLE ADD FOREIGN KEY）可使用的最大内存量。默认值为 16 兆字节（16MB）。由于一个数据库会话一次只能执行一个此类操作，而一个数据库系统通常也不会并发运行很多此类操作，
因此可以安全地将该值设得远大于 work_mem。更大的设置可能改善清理和恢复数据库转储的性能。

注意，自动清理运行时，最多可能分配此内存量的autovacuum_max_workers倍，
因此不要将默认值设得过高。

max_stack_depth (integer)

指定服务器执行栈的最大安全深度。
此参数的理想设置是由内核强制执行的实际栈大小限制（如由ulimit -s或本地等效设置），减去大约一兆字节的安全余量。
需要安全余量是因为服务器中并非每个例程都检查栈深度，
而只在可能递归的关键例程（如表达式求值）中检查。默认设置为两兆字节（2MB），
这是保守且不太可能引起崩溃的小值。但是，这可能太小，无法执行复杂函数。
只有超级用户才能更改此设置。

把max_stack_depth
参数设置得高于实际的内核限制将意味着一个失控的递归函数可能会导致一个独立的后端进程崩溃。在PostgreSQL能够检测内核限制的平台上，
服务器将不允许把这个参数设置为一个不安全的值。不过，
并非所有平台都能提供该信息，所以我们还是建议你在选择值时要小心。

18.4.2. 磁盘

temp_file_limit (integer)

指定一个会话可用于临时文件的最大磁盘空间，例如排序和哈希临时文件，
或保留游标的存储文件。尝试超过此限制的事务将被取消。该值以千字节为单位。-1（默认值）表示没有限制。只有超级用户才能更改此设置。

此设置限制单个 PostgreSQL 会话在任意时刻使用的所有临时文件的总空间。需要注意，显式临时表所用的磁盘空间不计入该上限；
计入的是查询执行过程中内部使用的临时文件。

18.4.3. 内核资源使用

`max_files_per_process` (`integer`)

设置每个服务器子进程允许同时打开的最大文件数量。默认值为一千个文件。
如果内核强制实施了安全的每进程上限，就不必担心此设置。
但在某些平台上（尤其是大多数 BSD 系统），内核允许单个进程打开的文件数量很大，
如果很多进程都尝试打开这么多文件，就会远超系统实际能够支持的总量。
如果遇到“Too many open files”（打开的文件过多）错误，可尝试减小此设置。此参数只能在服务器启动时设置。

`shared_preload_libraries` (`string`)

这个变量指定一个或者多个要在服务器启动时预载入的共享库。例如，`'$libdir/mylib'`会使 `mylib.so`（或者在某些平台上的 `mylib.sl`）从安装的标准库目录中被预载入。除非用双引号括起，所有库名都会被转换为小写。如果要载入多个库，
库名之间用逗号分隔。这个参数只能在服务器启动时设置。

PostgreSQL的过程语言库可以用这种方式预载入，通常使用 `'$libdir/plXXX'`语法，其中`XXX`是 `pgsql`、`perl`、`tcl` 或`python`。

通过预载入一个共享库，当该库被第一次使用时就可以避免库的启动时间。不过，
启动每个新服务器进程的时间可能会略有增加，即使该进程从不使用该库。因此，
推荐只把这个参数用于那些要在大多数会话中使用的库上。

> **注意**
>
> 在 Windows 主机上，在服务器启动时预载入一个库并不会减少启动每个新服务器进程所需的时间；每一个服务器进程将会重新载入所有预载入的库。不过，`shared_preload_libraries` 在 Windows 主机上仍然有用，因为某些共享库可能需要执行一些只在 postmaster 启动时发生的操作，例如分配共享内存、保留轻量级锁或者启动后台工作者。

如果指定的库没有找到，服务器将无法启动。

每一个 PostgreSQL 支持的库都有一个“魔法块”，它会被检查以保证兼容性。由于这个原因，非 PostgreSQL 的库不能以这种方式载入。

18.4.4. 基于代价的清理延迟

执行VACUUM和ANALYZE命令期间，系统维护一个内部计数器，记录已执行的各种 I/O 操作的估算代价。当累计代价达到上限（由 `vacuum_cost_limit` 指定）时，执行该操作的进程会休眠一小段时间，时长由 `vacuum_cost_delay` 指定。随后重置计数器并继续执行。

此功能让管理员能够降低这些命令对并发数据库活动的 I/O 影响。在许多情况下，`VACUUM` 和 `ANALYZE` 等维护命令是否快速完成并不重要，但避免它们显著干扰系统执行其他数据库操作的能力通常很重要。基于代价的清理延迟为管理员提供了实现这一点的方法。

对于手动执行的 `VACUUM` 命令，此功能默认禁用。要启用它，将 `vacuum_cost_delay` 变量设为非零值。

`vacuum_cost_delay` (`integer`)

超过代价上限后，进程将休眠的时长，单位为毫秒。默认值为零，
表示禁用基于代价的清理延迟功能。正值会启用基于代价的清理。注意，在许多系统上，

休眠延迟的有效分辨率为 10 毫秒；将 vacuum_cost_delay 设为不是 10 的倍数的值，可能与将它设为下一个更大的 10 的倍数效果相同。

使用基于代价的清理时，vacuum_cost_delay 的合适值通常较小，例如 10 或 20 毫秒。调整清理的资源消耗时，最好更改其他清理代价参数。

vacuum_cost_page_hit (integer)

清理在共享缓冲区缓存中找到的缓冲区时所计入的估算代价。它表示锁定缓冲池、查找共享哈希表和扫描页内容的代价。默认值为 1。

vacuum_cost_page_miss (integer)

清理必须从磁盘读取的缓冲区时所计入的估算代价。它表示锁定缓冲池、查找共享哈希表、从磁盘读取所需数据块并扫描其内容所需的工作量。默认值为 10。

vacuum_cost_page_dirty (integer)

清理操作修改原本干净的数据块时所计入的估算代价。它表示再次将脏块刷盘所需的额外 I/O。默认值为 20。

vacuum_cost_limit (integer)

会使清理进程休眠的累计代价。默认值为 200。

注意

某些操作持有关键的锁，因此应尽快完成。
这些操作期间不会发生基于代价的清理延迟，所以累计代价可能远超指定上限。
为避免此时出现无益的长时间延迟，实际延迟按 vacuum_cost_delay * accumulated_balance / vacuum_cost_limit 计算，但最大不超过 vacuum_cost_delay * 4。

18.4.5. 后台写入器

有一个独立的服务器进程，称为后台写入器，负责写出“脏”（新的或修改过的）共享缓冲区。当干净的共享缓冲区数量似乎不足时，后台写入器会将一些脏缓冲区写入文件系统，并将其标记为干净。这可以降低处理用户查询的服务器进程找不到干净缓冲区、因而不得不自行写出脏缓冲区的可能性。不过，后台写入器确实会使总体 I/O 负载有所增加：反复变脏的页面原本可能在每个检查点间隔中只写出一次，而后台写入器可能在同一间隔内随着它变脏而多次写出。本节参数可用于根据实际需求调整此行为。

bgwriter_delay (integer)

指定后台写入器各轮活动之间的延迟。每一轮中，写入器会对一定数量的脏缓冲区发出写操作（由下面的参数控制），然后休眠 bgwriter_delay 毫秒，再重复此过程。不过，当缓冲池中没有脏缓冲区时，它会进入更长的休眠，而不受 bgwriter_delay 限制。默认值为 200 毫秒（200ms）。注意，在许多系统上，休眠延迟的有效分辨率为 10 毫秒；将 bgwriter_delay 设为不是 10 的倍数的值，可能与将它设为下一个更大的 10 的倍数效果相同。此参数只能在 postgresql.conf 文件中或服务器命令行上设置。

bgwriter_lru_maxpages (integer)

后台写入器每轮写出的缓冲区数量不会超过此值。设为零会禁用后台写入。（由另一个独立的专用辅助进程管理的检查点不受影响。）默认值为 100 个缓冲区。此参数只能在 postgresql.conf 文件中或服务器命令行上设置。

bgwriter_lru_multiplier (floating point)

每轮写出的脏缓冲区数量取决于最近几轮服务器进程所需的新缓冲区数量。将近期平均需求乘以 bgwriter_lru_multiplier，即可估算下一轮所需的缓冲区数量。写入器会写出脏缓冲区，直到可用的干净且可重用缓冲区达到这一数量。（不过，每轮写出的缓冲区数量不会超过 bgwriter_lru_maxpages。）因此，设为 1.0 表示采用“恰好及时”策略，写出的缓冲区数量恰好等于预测需求量。更大的值可为需求突增留出余量，而更小的值则有意将部分写操作留给服务器进程执行。默认值为 2.0。此参数只能在 postgresql.conf 文件中或服务器命令行上设置。

较小的 bgwriter_lru_maxpages 和 bgwriter_lru_multiplier 可以降低后台写入器造成的额外 I/O 负载，但也会增加服务器进程必须自行发出写操作的可能性，从而延迟交互式查询。

18.4.6. 异步行为

effective_io_concurrency (integer)

设置 PostgreSQL 预期可以同时执行的并发磁盘 I/O 操作数量。提高此值会增加单个 PostgreSQL 会话尝试并行发起的 I/O 操作数量。允许的范围为 1 至 1000，或设为零以禁用异步 I/O 请求。目前，此设置仅影响位图堆扫描。

对于磁盘，可以将为数据库提供存储的 RAID 0 条带或 RAID 1 镜像中的独立磁盘数量作为合理初始值。（对于 RAID 5，不应计入校验盘。）不过，如果数据库经常忙于执行并发会话发出的多个查询，较小的值可能就足以使磁盘阵列保持繁忙。超过使磁盘保持繁忙所需的值只会增加 CPU 开销。

对于更特殊的系统，例如基于内存的存储或受总线带宽限制的 RAID 阵列，正确的值可能是可用 I/O 通路的数量。可能需要一些实验才能找到最佳值。

异步 I/O 依赖于有效的 posix_fadvise 函数，而某些操作系统缺少此函数。如果该函数不存在，将此参数设为任何非零值都会报错。在某些操作系统（如 Solaris）上，该函数虽然存在，却实际上不做任何事情。

18.5. 预写式日志

另请参阅第 29.4 节了解有关 WAL 和检查点调优的详细信息。

18.5.1. 设置

wal_level (enum)

wal_level决定多少信息写入到 WAL 中。默认值是minimal，它只写入从崩溃或立即关机中进行恢复所需的信息。archive会增加 WAL 归档所需的日志；hot_standby会进一步增加在备库上运行只读查询所需的信息。这个参数只能在服务器启动时设置。

在minimal级别，可以安全地跳过一些批量操作的 WAL 日志记录，使这些操作快得多（参见第 14.4.7 节）。可以应用此优化的操作包括：

CREATE TABLE AS
CREATE INDEX
CLUSTER
向同一事务中创建或截断的表执行COPY

但 minimal 级别的 WAL 不包含足够的信息来从基础备份和 WAL 日志重建数据，因此必须使用archive或hot_standby级别来启用 WAL 归档（archive_mode）和流复制。

在hot_standby级别上，记录与archive相同的信息，以及从 WAL 重构正在运行事务的状态所需的信息。要在备库上启用只读查询，必须把主库上的wal_level设置为hot_

standby，并且在备库上启用hot_standby。据认为，使用hot_standby和archive级别在性能上几乎没有可测量的差别，因此如果注意到任何生产环境影响，欢迎反馈。

fsync (boolean)

如果打开这个参数，PostgreSQL服务器将尝试确保更新被物理地写入到磁盘，做法是发出fsync()系统调用或者使用多种等价的方法（见wal_sync_method）。这保证了数据库集簇在一次操作系统或者硬件崩溃后能恢复到一个一致的状态。

虽然关闭fsync常常可以得到性能上的收益，但当发生断电或系统崩溃时可能造成不可恢复的数据损坏。因此，只有在能很容易地从外部数据中重建整个数据库时才建议关闭fsync。

可以安全关闭fsync的情形包括：从备份文件初始装载一个新数据库集簇；用数据库集簇处理一批数据，处理后就丢弃并重建该数据库；或者使用经常重建且不用于故障切换的只读数据库克隆。仅有高质量硬件不足以成为关闭fsync的理由。

在很多情况下，为非关键事务关闭synchronous_commit，可以获得关闭fsync所带来的大部分潜在性能收益，同时避免伴随的数据损坏风险。

fsync只能在postgresql.conf文件中或在服务器命令行上设置。如果你关闭这个参数，请也考虑关闭full_page_writes。

synchronous_commit (enum)

指定事务提交是否等待 WAL 记录写入磁盘后，命令才向客户端返回“成功”指示。有效值为on、remote_write、local和off。默认且安全的设置是on。当为off时，向客户端报告成功后，可能还要经过一段时间，事务才能真正保证不会因服务器崩溃而丢失。（最大延迟为wal_writer_delay的三倍。）与fsync不同，将此参数设为off不会带来数据库不一致的风险：操作系统或数据库崩溃可能会使一些最近报告已提交的事务丢失，但数据库状态会与这些事务已正常中止时完全相同。因此，当性能比完全确保事务持久性更重要时，关闭synchronous_commit可以是一种有用的替代方案。更多讨论见第 29.3 节。

如果设置了synchronous_standby_names，此参数还控制事务提交是否等待该事务的 WAL 记录复制到备库服务器。设为on时，提交会等待当前同步备库回复，确认已收到事务的提交记录，并已将其刷写到磁盘。这能保证事务不会丢失，除非主库和备库的数据库存储都损坏。设为remote_write时，提交会等待当前同步备库回复，确认已收到事务的提交记录，并已将其写入备库的操作系统，但数据未必已写入备库的持久存储。即使备库上的PostgreSQL实例崩溃，此设置也足以保证数据不丢失，但不能保证备库发生操作系统级别崩溃时数据不丢失。

使用同步复制时，通常要么等待本地刷盘和 WAL 记录复制两者都完成，要么允许事务异步提交，这样才是合理的。不过，对于希望等待本地刷盘但不等待同步复制的事务，可以使用local设置。如果未设置synchronous_standby_names，则设置on、remote_write和local提供的同步级别相同：事务提交只等待本地刷盘。

此参数可以随时更改；每个事务的行为由提交时生效的设置决定。因此，让一些事务同步提交、另一些事务异步提交是可行且有用的。例如，当默认设置要求同步提交时，可以在一个包含多条语句的事务中执行SET LOCAL synchronous_commit TO OFF，使该事务异步提交。

wal_sync_method (enum)

用于将 WAL 更新强制写入磁盘的方法。如果fsync关闭，此设置就没有作用，因为 WAL 文件更新根本不会被强制写入磁盘。可选值为：

- open_datasync（用open()选项O_DSYNC写 WAL 文件）

- fdatasync（在每次提交时调用fdatasync()）
- fsync（在每次提交时调用fsync()）
- fsync_writethrough（在每次提交时调用fsync()，强制穿透任何磁盘写缓存）
- open_sync（用open()选项O_SYNC写 WAL 文件）

open_* 选项还会使用O_DIRECT（如果可用）。不是在所有平台上都能使用所有这些选择。默认值是列表中第一个被平台支持的那个，不过fdatasync是 Linux 中的默认值。默认值不一定最合适；可能需要更改此设置或系统配置的其他方面，以确保崩溃时的数据安全或达到最佳性能。这些方面在第 29.1 节中讨论。这个参数只能在postgresql.conf文件中或在服务器命令行上设置。

full_page_writes (boolean)

启用此参数时，PostgreSQL服务器会在检查点之后首次修改每个磁盘页面时，将该页面的全部内容写入 WAL。这样做是因为，操作系统崩溃时正在进行的页面写入可能只完成了一部分，导致磁盘页面混有新旧数据。通常存储在 WAL 中的行级变更数据不足以在崩溃恢复时完整还原这样的页面。保存整页镜像能保证正确恢复页面，但会增加必须写入 WAL 的数据量。（由于 WAL 重放总是从检查点开始，只需在检查点之后首次修改每个页面时这样做。因此，减少整页写入开销的一种方法是增大检查点间隔参数。）

关闭此参数可以加快正常操作，但系统故障后可能出现不可恢复的数据损坏或静默数据损坏。风险与关闭fsync类似，虽然较小，但也只有在建议关闭fsync的相同情形下，才应关闭此参数。

关闭这个选项并不影响用于时间点恢复（PITR）的 WAL 归档使用（见第 24.3 节）。

这个参数只能在postgresql.conf文件中或在服务器命令行上设置。默认值是on。

wal_buffers (integer)

用于还未写入磁盘的 WAL 数据的共享内存量。默认值 -1 选择等于shared_buffers的 1/32 的尺寸（大约3%），但是不小于64kB也不大于 WAL 段的尺寸（通常为16MB）。如果自动的选择太大或太小可以手工设置该值，但是任何小于32kB的正值都将被当作32kB。这个参数只能在服务器启动时设置。

在每次事务提交时，WAL 缓冲区的内容被写出到磁盘，因此极大的值不太可能带来显著收益。不过，把这个值设置为至少几个兆字节可以在一个繁忙的服务器（其中很多客户端会在同一时间提交）上提高写性能。由默认设置 -1 选择的自动调节将在大部分情况下得到合理的结果。

增大这个参数可能会导致PostgreSQL请求比操作系统默认配置所允许的更多的System V 共享内存。如有必要，关于如何调整这些参数的信息请参见第 17.4.1 节。

wal_writer_delay (integer)

指定 WAL 写入器两轮活动之间的延迟。在每一轮中，写入器会把 WAL 刷写到磁盘，然后休眠wal_writer_delay毫秒，再重复这一过程。默认值为 200 毫秒（200ms）。请注意，许多系统的休眠延迟有效分辨率为 10 毫秒；将wal_writer_delay设置为非 10 的倍数的值，可能与设置为下一个更大的 10 的倍数具有相同效果。此参数只能在postgresql.conf文件中或在服务器命令行上设置。

commit_delay (integer)

当一个事务的提交数据被刷盘到磁盘时，当时已准备好的任何其他提交也会一起被刷出。commit_delay在事务尝试把 WAL 缓冲区刷出到磁盘之前增加一个以微秒为单位的时间延迟。如果系统负载足够高，使得在给定的时间间隔内有额外的事务准备好提交，那么一个非零的延迟可以让更多的事务只用一次刷盘操作提交。

但如果没有其他事务准备好提交，这个延迟就纯属浪费。因此，
只有当一个服务器进程写入其提交记录的那一刻，至少有commit_siblings
个其他事务处于活动状态时，才会执行延迟。默认的commit_delay为零（无延迟）。
因为无论此设置如何，每次刷盘都会写出所有挂起的提交数据，
所以增大此参数而增加延迟很少能真正提高性能。

commit_siblings (integer)

执行commit_delay延迟前要求的并发活动事务的最小数目。
大一些的值会导致在延迟间隔期间更可能有至少另外一个事务准备好提交。
默认值是五个事务。

18.5.2. 检查点

checkpoint_segments (integer)

自动 WAL 检查点之间日志文件段的最大数量（每个段通常为 16 兆字节）。默认值为三个段。增加这个参数会增加崩溃恢复所需的时间。此参数只能在postgresql.conf
文件中或服务器命令行上设置。

checkpoint_timeout (integer)

自动 WAL 检查点之间的最长时间，以秒为单位。合理的范围在 30 秒到 1 小时之间。默认是 5 分钟（5min）。增加这个参数的值可能会增加崩溃恢复所需的时间。这个参数只能在postgresql.conf文件中或在服务器命令行上设置。

checkpoint_completion_target (floating point)

指定检查点完成的目标，作为检查点之间总时间的一部分。默认是 0.5。这个参数只能在postgresql.conf文件中或在服务器命令行上设置。

checkpoint_warning (integer)

如果由于填充检查点段文件导致的检查点之间的间隔低于这个参数表示的秒数，
那么就向服务器日志写一个消息（它建议增加checkpoint_segments的值）。
默认值是 30 秒（30s）。零则关闭警告。这个参数只能在postgresql.conf文件中或在服务器命令行上设置。

18.5.3. 归档

archive_mode (enum)

当启用archive_mode时，完成的WAL段会通过设置archive_
command发送到归档存储。archive_mode和archive_command是单独的变量，
这样即使不离开归档模式也可以修改archive_command。
这个参数只能在服务器启动时设置。当wal_level被设置为minimal时，不能启用
archive_mode。

archive_command (string)

本地 shell 命令被执行来归档一个完成的 WAL 文件段。字符串中的任何%p被替换成要被归档的文件的路径名，而%f只被文件名替换（路径名是相对于服务器的工作目录，
即集簇的数据目录）。如果要在命令里嵌入一个真正的%字符，可以使用%%。
有一点很重要，该命令只在成功时返回一个零作为退出状态。更多信息请见第 24.3.1 节。

这个参数只能在postgresql.conf文件或服务器命令行中设置。
除非在服务器启动时启用了archive_mode，否则将被忽略。如果archive_command
是空字符串（默认值），而archive_mode已启用，WAL归档将暂时被禁用，
但服务器将继续积累WAL段文件，期望很快会提供命令。将archive_command

设置为一个什么都不做但返回true的命令，例如/bin/true（Windows上为REM），实际上禁用了归档，但也破坏了用于归档恢复所需的WAL文件链，因此只应在不寻常的情况下使用。

archive_timeout (integer)

archive_command只针对已完成的 WAL 段调用。因此，如果您的服务器生成的WAL流量较少（或者在这样做时有间歇期），在事务完成和安全记录到归档存储之间可能会有很长的延迟。为了限制未归档数据的年龄，您可以将archive_timeout设置为强制服务器定期切换到新的WAL段文件。当此参数大于零时，只要自上次段文件切换以来经过了指定的秒数，并且存在任何数据库活动（包括单个检查点），服务器将切换到新的段文件。（增大checkpoint_timeout 可以减少空闲系统上不必要的检查点。）请注意，由于强制切换而提前关闭的归档文件仍然与完全填满的文件长度相同。因此，使用非常短的archive_timeout是不明智的，它会使您的归档存储膨胀。通常，将archive_timeout设置为一分钟左右是合理的。如果您希望数据比这更快地从主库复制出来，您应该考虑使用流复制而不是归档。此参数只能在postgresql.conf文件或服务器命令行中设置。

18.6. 复制

这些设置控制内置流复制特性（见第 25.2.5 节）的行为。服务器将可以是主库或备库。主库能发送数据，而备库总是被复制数据的接收者。当使用级联复制（见第 25.2.6 节）时，备库也可以是发送者，同时也是接收者。这些参数主要用于发送服务器和备库，尽管某些只在主库上有意义。如果有必要，设置可以在集簇中变化而不出问题。

18.6.1. 发送服务器

这些参数可以在任何发送复制数据给一个或多个备库的服务器上设置。主库总是一个发送服务器，因此这些参数总是要在主库上设置。这些参数的角色和含义不会在一个备库变成主库后改变。

max_wal_senders (integer)

指定来自备库或流式基础备份客户端的并发连接的最大数量（即同时运行 WAL 发送进程的最大数）。默认值为 0，即禁用复制。WAL 发送进程计入总连接数，因此此参数的值不能高于max_connections。此参数只能在服务器启动时设置。wal_level必须设置为archive或 hot_standby，才允许来自备库的连接。

wal_keep_segments (integer)

指定在pg_xlog目录中保留的过去日志文件段的最小数量，以便备库可能需要获取它们来进行流复制。每个段通常为 16 兆字节。如果连接到发送服务器的备库落后超过wal_keep_segments个段，发送服务器可能会移除备库仍需要的 WAL 段，在这种情况下，复制连接将被终止。下游连接最终也会因此失败。（但是，如果使用了 WAL 归档，备库可以通过从归档中获取该段来恢复。）

此设置只指定在pg_xlog中保留的最小段数；系统可能需要为 WAL 归档或从检查点恢复而保留更多段。如果wal_keep_segments为零（默认值），系统不会为备库额外保留任何段，因此备库可用的旧 WAL 段数取决于前一个检查点的位置和 WAL 归档的状态。此参数只能在postgresql.conf文件中或在服务器命令行上设置。

replication_timeout (integer)

终止不活动时间超过指定毫秒数的复制连接。这有助于主库检测备库崩溃或网络中断。值为零将禁用超时机制。这个参数只能在postgresql.conf文件中或在服务器命令行上设置。默认值为 60 秒。

为防止连接被过早终止，必须在备库上启用wal_receiver_status_interval，并且它的值必须小于replication_timeout的值。

18.6.2. 主库

这些参数可以在向一个或多个备库发送复制数据的主库上设置。除这些参数外，还必须在主库上适当设置wal_level，也可以选择启用 WAL 归档（见第 18.5.3 节）。这些参数在备库上的取值不影响备库运行，不过也可以预先设置，以备将来提升为主库。

synchronous_standby_names (string)

如第 25.2.7 节所述，这个参数指定一个支持同步复制的备库名称的逗号分隔列表。在任一时刻，最多只有一个活动的同步备库；在这个备库确认收到它们的数据之后，等待提交的事务将被允许继续下去。同步备库是列表中第一个被命名、并且当前已连接并且正在实时流式传输数据（如pg_stat_replication视图中streaming的状态所示）的服务器。在这个列表中出现得较晚的其他备库表示潜在的同步备库。如果当前的同步备库因为某种原因断开连接，它将立刻被下一个最高优先级的备库替代。指定多于一台备库名称可以得到非常高的可用性。

用于这一目的的备库名称是其 WAL 接收器的primary_conninfo中设置的application_name。没有机制强制唯一性。在出现重复的情况下，匹配的备库之一将被选为同步备库，不过无法弄清到底是哪一个。特殊项*匹配任意application_name，包括默认的应用名walreceiver。

如果这里没有指定同步备库名称，那么不启用同步复制并且事务提交将不会等待复制。这是默认的配置。即便当同步复制被启用时，个体事务也可以被配置为不等待复制，做法是将synchronous_commit参数设置为local或off。

这个参数只能在postgresql.conf文件中或通过服务器命令行进行设置。

vacuum_defer_cleanup_age (integer)

指定VACUUM和HOT更新延迟清理死行版本的事务数。默认值为零个事务，意味着可以尽快移除死行版本，也就是在它们不再对任何打开的事务可见时立即移除。如第 25.5 节所述，在为热备服务器提供支持的主库上，你可能希望将此参数设置为非零值。这让备库上的查询有更多时间完成，而不会因过早清理行而发生冲突。但是，由于此值以主库上发生的写事务数计量，很难预测会给备库查询带来多少额外的宽限时间。此参数只能在postgresql.conf文件中或在服务器命令行上设置。

你也应考虑在备库上设置hot_standby_feedback，作为使用此参数的替代方案。

18.6.3. 备库

这些设置控制备库接收复制数据时的行为。它们在主库的值是无关的。

hot_standby (boolean)

指定在恢复期间，你是否能够连接并运行查询，如第 25.5 节中所述。默认值是off。这个参数只能在服务器启动时设置。它只在归档恢复期间或备库模式下才有效。

max_standby_archive_delay (integer)

当热备处于活动状态时，此参数确定备库在取消与即将应用的WAL条目冲突的备库查询之前应等待多长时间，如第 25.5.2 节中所述。max_standby_archive_delay在从WAL归档中读取WAL数据时适用（因此不是当前的）。如果未指定单位，则将其视为毫秒。默认值为30秒。值为-1允许备库永远等待冲突查询完成。此参数只能在postgresql.conf文件或服务器命令行中设置。

注意，max_standby_archive_delay
并不等同于查询在被取消前可以运行的最长时间；它表示应用任意一个 WAL 段的数据所允许的最长总时间。因此，如果某个查询先前在处理该 WAL 段时已造成显著延迟，后续冲突查询的宽限时间就会短得多。

max_standby_streaming_delay (integer)

当热备处于活动状态时，
此参数确定备库在取消与即将应用的WAL条目冲突的备库查询之前应等待多长时间，如第 25.5.2 节中所述。max_standby_streaming_delay在通过流复制接收WAL数据时应用。如果未指定单位，则将其视为毫秒。默认值为30秒。值为
-1允许备库永远等待冲突查询完成。此参数只能在postgresql.conf
文件或服务器命令行中设置。

注意，max_standby_streaming_delay
并不等同于查询在被取消前可以运行的最长时间；它表示从主库接收到 WAL 数据后，允许用于应用这些数据的最长总时间。因此，如果某个查询已造成显著延迟，
后续冲突查询的宽限时间就会短得多，直到备库再次赶上进度。

wal_receiver_status_interval (integer)

指定备库上的 WAL 接收进程向主库或上游备库发送有关复制进度的信息的最小频度，它可以使用pg_stat_replication视图看到。备库将报告最后写入的事务日志位置、
最后刷盘的位置以及最后应用的位置。这个参数的值是报告之间的最大间隔，
以秒为单位。每次写入或刷盘位置改变时会发送状态更新，
或者至少按这个参数指定的频度发送。因此，应用位置可能比真实位置略微滞后。
将此参数设置为零会完全禁用状态更新。这个参数只能在postgresql.conf
文件中或在服务器命令行上设置。默认值是 10 秒。

当在主库上启用了replication_timeout时，必须启用wal_receiver_status_interval，并且它的值必须小于replication_timeout的值。

hot_standby_feedback (boolean)

指定一个热备机是否将会向主库或上游备库发送有关于备库上当前正被执行的查询的反馈。这个参数可以被用来消除由清理记录引起的查询取消，
但是可能导致在主库上用于某些负载的数据库膨胀。反馈消息的发送频度不会高于每个wal_receiver_status_interval周期发送一次。默认值是off。
这个参数只能在postgresql.conf文件中或在服务器命令行上设置。

如果使用级联复制，反馈将被向上游传递直到它最后到达主库。
备库在接收到反馈之后除了传递给上游不会做任何其他操作。

18.7. 查询规划

18.7.1. 规划器方法配置

这些配置参数提供了一种较为粗糙的方法，用于影响查询优化器所选择的查询计划。
如果优化器为某个特定查询选择的默认计划并不理想，一种
临时解决方案是使用这些配置参数，强制优化器选择另一种计划。
改善优化器所选计划质量的更好办法包括调整规划器代价常数（见第 18.7.2 节）、手工运行ANALYZE、增加default_statistics_target配置参数的值，以及使用 ALTER TABLE SET STATISTICS 增加为特定列收集的统计信息量。

enable_bitmapscan (boolean)

启用或禁用查询规划器对位图扫描计划类型的使用。默认值为 on。

enable_hashagg (boolean)

启用或禁用查询规划器对哈希聚合计划类型的使用。默认值为 on。

enable_hashjoin (boolean)

启用或禁用查询规划器对哈希连接计划类型的使用。默认值为 on。

enable_indexscan (boolean)

启用或禁用查询规划器对索引扫描计划类型的使用。默认值为 on。

enable_indexonlyscan (boolean)

启用或禁用查询规划器对仅索引扫描计划类型的使用。默认值为 on。

enable_material (boolean)

允许或者禁止查询规划器使用物化。它不可能完全禁用物化，
但是关闭这个变量将阻止规划器插入物化节点，除非为了保证正确性。默认值是on。

enable_mergejoin (boolean)

启用或禁用查询规划器对归并连接计划类型的使用。默认值为 on。

enable_nestloop (boolean)

允许或禁止查询规划器使用嵌套循环连接计划。它不可能完全禁止嵌套循环连接，
但是关闭这个变量将使得规划器尽可能优先使用其他方法。默认值是on。

enable_seqscan (boolean)

允许或禁止查询规划器使用顺序扫描计划类型。它不可能完全禁止顺序扫描，
但是关闭这个变量将使得规划器尽可能优先使用其他方法。默认值是on。

enable_sort (boolean)

允许或禁止查询规划器使用显式排序步骤。它不可能完全禁止显式排序，
但是关闭这个变量将使得规划器尽可能优先使用其他方法。默认值是on。

enable_tidscan (boolean)

允许或禁止查询规划器使用TID扫描计划类型。默认值是on。

18.7.2. 规划器代价常量

这一节中描述的代价变量可以按照任意尺度衡量。我们只关心它们的相对值，
将它们以相同的因子缩放不会影响规划器的选择。默认情况下，
这些代价变量是基于顺序页面获取的代价的，即seq_page_cost被设置为1.0
并且其他代价变量都参考它来设置。不过你可以使用你喜欢的不同尺度，
例如在一个特定机器上以毫秒为单位的实际执行时间。

> **注意**
>
> 没有明确的方法可以确定代价变量的理想值。
> 最好将它们视为某个数据库安装实例所接收的全部查询组合的平均值。因此，
> 仅凭少数几次试验就修改它们是非常冒险的。

`seq_page_cost`(`floating point`)

设置规划器对一系列顺序磁盘页面读取中单次读取的代价估计。默认值是 1.0。对于某个表空间内的表和索引，可以通过设置该表空间的同名参数来覆盖此值（见ALTER TABLESPACE）。

`random_page_cost`(`floating point`)

设置规划器对一次非顺序磁盘页面读取的代价估计。默认值是 4.0。对于某个表空间内的表和索引，可以通过设置该表空间的同名参数来覆盖此值（见ALTER TABLESPACE）。

减少这个值（相对于`seq_page_cost`）将导致系统更倾向于索引扫描；
提高它将让索引扫描看起来相对更昂贵。你可以一起提高或降低两个值来改变磁盘 I/O 代价相对于 CPU 代价的重要性，后者由下列参数描述。

对机械磁盘存储的随机访问通常远不止比顺序访问贵四倍。不过，
仍使用较低的默认值（4.0），因为假定对磁盘的大多数随机访问（例如索引读取）
都将在缓存中命中。可以将默认值理解为：假设随机访问比顺序访问慢 40 倍，同时预期 90% 的随机读取能够命中缓存。

如果你认为 90% 的缓存命中率不符合你的工作负载，可以增大 random_page_cost，以更好地反映随机存储读取的真实代价。对应地，如果你的数据很可能完全缓存在内存中，例如数据库小于服务器总内存，则降低 random_page_cost 可能更合适。对于随机读取代价相对于顺序读取较低的存储，例如固态硬盘，也可以用较低的 random_page_cost 值来更好地建模。

> **提示**
>
> 尽管系统允许将`random_page_cost`设置得小于`seq_page_cost`，
> 但这不符合实际物理情况。不过，如果数据库完全缓存在 RAM 中，将它们设置为相等是合理的，因为此时非顺序访问页面不会产生额外代价。同样，
> 对于大部分数据已缓存的数据库，应相对于 CPU 参数降低这两个值，因为读取一个已在 RAM 中的页面的代价远小于通常的页面读取代价。

`cpu_tuple_cost`(`floating point`)

设置规划器对一次查询中处理每一行的代价估计。默认值是 0.01。

`cpu_index_tuple_cost`(`floating point`)

设置规划器对一次索引扫描中处理每一个索引项的代价估计。默认值是 0.005。

`cpu_operator_cost`(`floating point`)

设置规划器对于一次查询中处理每个操作符或函数的代价估计。默认值是 0.0025。

`effective_cache_size`(`integer`)

设置规划器对一个单一查询可用的有效磁盘缓存尺寸的假设。
这个参数会被考虑在使用一个索引的代价估计中，
更高的数值会使得索引扫描更可能被使用，更低的数值会使得顺序扫描更可能被使用。
在设置这个参数时，你还应该考虑
PostgreSQL的共享缓冲区以及将被用于PostgreSQL数据文件的内核磁盘缓存，
尽管有些数据可能在两个地方都存在。另外，还要考虑预计在不同表上的并发查询数目，
因为它们必须共享可用的空间。这个参数对PostgreSQL分配的共享内存尺寸没有影响，
它也不会预留内核磁盘缓存，它只用于估计的目的。
系统也不会假设在查询之间数据会保留在磁盘缓存中。默认值是 128 兆字节（`128MB`）。

18.7.3. 遗传查询优化器

遗传查询优化器（GEQO）是一种使用启发式搜索进行查询规划的算法。
它可以缩短复杂查询（连接很多关系的查询）的规划时间，
代价是生成的计划有时不如常规穷举搜索算法找到的计划。更多信息见第 51 章。

`geqo` (`boolean`)

允许或禁止遗传查询优化。默认是启用。在生产环境中通常最好不要关闭它。`geqo_threshold`变量提供了对 GEQO 更细粒度的控制。

`geqo_threshold` (`integer`)

只有当涉及的`FROM`项数量至少有这么多个的时候，才使用遗传查询优化（注意一个`FULL OUTER JOIN`只被计为一个`FROM`项）。默认值是 12。对于更简单的查询，通常会使用普通的穷举搜索规划器，但是对于有很多表的查询穷举搜索会花很长时间，
通常比执行一个次优的计划带来的惩罚值还要长。因此，在查询尺寸上的一个阈值是管理 GEQO 使用的一种方便的方法。

`geqo_effort` (`integer`)

控制 GEQO 中规划时间和查询计划质量之间的权衡。此变量必须是 1 到 10 之间的整数。默认值是 5。较大的值会增加查询规划所用的时间，但也会提高选择高效查询计划的可能性。

`geqo_effort`实际并不直接做任何事情；它只是被用来计算其他影响 GEQO 行为的变量（如下所述）的默认值。如果你愿意，你可以手工设置其他参数。

`geqo_pool_size` (`integer`)

控制 GEQO 使用的池尺寸，它就是遗传种群中的个体数目。它必须至少为 2，且有用的值通常在 100 到 1000 之间。如果它被设置为零（默认设置）则会基于`geqo_effort`和查询中表的数量选择一个合适的值。

`geqo_generations` (`integer`)

控制 GEQO 使用的代数，即算法的迭代次数。它必须至少为 1，通常有用的值与池大小处于相同范围。如果设置为零（默认设置），则根据`geqo_pool_size`选择合适的值。

`geqo_selection_bias` (`floating point`)

控制 GEQO 使用的选择偏好。选择偏好是种群中的选择压力。值可以是 1.5 到 2.0 之间，后者是默认值。

`geqo_seed` (`floating point`)

控制 GEQO 使用的随机数生成器的初始值，随机数生成器用于在连接顺序搜索空间中选择随机路径。该值可以从 0（默认值）到 1。变化该值会改变被探索的连接路径集合，并且可能使找到的最优路径变得更好或更差。

18.7.4. 其他规划器选项

`default_statistics_target` (`integer`)

为没有通过`ALTER TABLE SET STATISTICS`设置列相关目标的表列设置默认统计目标。更大的值增加了需要做`ANALYZE`的时间，但是可能会改善规划器的估计质量。
默认值是 100。有关PostgreSQL查询规划器使用的统计信息的更多内容，请参考第 14.2 节。

constraint_exclusion (enum)

控制查询规划器对表约束的使用，以优化查询。constraint_exclusion 的允许值是on（对所有表检查约束）、off（从不检查约束）和partition（只对继承的子表和UNION ALL子查询检查约束）。partition是默认设置。它通常与继承和分区表一起使用来提高性能。

当此参数允许对某个表使用约束排除时，规划器会比较查询条件与该表的CHECK约束，并且忽略那些条件违反约束的表扫描。例如：

```
CREATE TABLE parent(key integer, ...);
CREATE TABLE child1000(check (key between 1000 and 1999))
 INHERITS(parent);
CREATE TABLE child2000(check (key between 2000 and 2999))
 INHERITS(parent);
...
SELECT * FROM parent WHERE key = 2400;
```

在启用约束排除时，这个SELECT将完全不会扫描child1000，从而提高性能。

目前，约束排除仅在通常用于实现表分区的情况下默认启用。
为所有表启用它会增加额外的规划开销，这在简单查询上相当明显，
而且通常不会为简单查询带来好处。如果没有分区表，你可能希望完全关闭它。

更多关于使用约束排除实现分区的信息请参阅第 5.9.4 节。

cursor_tuple_fraction (floating point)

设置规划器对将被检索的一个游标的行的比例的估计。默认值是 0.1。更小的值使得规划器偏向为游标使用“快速开始”计划，
它将很快地检索前几行但是可能需要很长时间来获取所有行。
更大的值强调总的估计时间。最大设置为 1.0，游标将和普通查询完全一样地被规划，只考虑总估计时间并且不考虑前几行会被多快地返回。

from_collapse_limit (integer)

如果生成的FROM列表不超过这么多项，规划器将把子查询融合到上层查询。
较小的值可以减少规划时间，但是可能会生成较差的查询计划。默认值是 8。详见第 14.3 节。

将这个值设置为geqo_threshold或更大，可能触发使用 GEQO 规划器，从而产生非最优计划。见第 18.7.3 节。

join_collapse_limit (integer)

如果得出的列表中不超过这么多项，那么规划器将把显式JOIN（除了FULL JOIN）结构重写到 FROM项列表中。较小的值可减少规划时间，但是可能会生成差些的查询计划。

默认情况下，这个变量被设置成和from_collapse_limit相同，
这样适合大多数使用。把它设置为 1 可避免任何显式JOIN的重排序。因此查询中指定的显式连接顺序就是关系被连接的实际顺序。
因为查询规划器并不是总能选取最优的连接顺序，
高级用户可以选择暂时把这个变量设置为 1，然后显式地指定他们想要的连接顺序。更多信息请见第 14.3 节。

将这个值设置为geqo_threshold或更大，可能触发使用 GEQO 规划器，从而产生非最优计划。见第 18.7.3 节。

18.8. 错误报告和日志

18.8.1. 日志记录到哪里

`log_destination`(`string`)

PostgreSQL支持多种记录服务器消息的方法，包括stderr，csvlog和syslog。在Windows 上，还支持eventlog。将此参数设为所需日志目的地的逗号分隔列表。默认只将日志记录到stderr。此参数只能在`postgresql.conf`文件中或在服务器命令行上设置。

如果csvlog被包括在`log_destination`中，日志项会以“逗号分隔值”（CSV）格式被输出，这样可以很方便地把日志载入到程序中。详见第 18.8.4 节。要产生 CSV 格式的日志输出，必须启用logging_collector。

> **注意**
>
> 在大多数 Unix 系统上，你将需要修改系统的syslog守护进程的配置来使用`log_destination`的syslog选项。PostgreSQL可以在syslog设施`LOCAL0`到`LOCAL7`中记录（见syslog_facility），但是大部分平台上的默认syslog配置会丢弃所有这种消息。你将需要增加这样的内容：
>
> ```
> local0.*    /var/log/postgresql
> ```
>
> 到syslog守护进程的配置文件来让它工作。
>
> 在 Windows 上，当你使用`log_destination`的`eventlog`选项时，你应该在操作系统中注册一个事件源及其库，这样 Windows 事件查看器能够清楚地显示事件日志消息。详见第 17.11 节。

`logging_collector`(`boolean`)

这个参数启用日志收集器，它是一个捕捉被发送到stderr的日志消息的后台进程，并且它会将这些消息重定向到日志文件中。这种方法比记录到syslog通常更有用，因为某些类型的消息可能不会在syslog输出中出现（一个常见的示例是动态链接器错误消息；另一个示例是由`archive_command`等脚本产生的错误消息）。这个参数只能在服务器启动时设置。

> **注意**
>
> 也可以不使用日志收集器而把日志记录到stderr，日志消息将只会去到服务器的stderr被定向到的位置。不过，那种方法只适合于低日志量，因为它没有提供便捷的方法来轮转日志文件。还有，在某些平台上，不使用日志收集器可能会导致日志输出丢失或混杂，因为多个进程并发写入同一个日志文件时会覆盖彼此的输出。

> **注意**
>
> 日志收集器被设计成从来不会丢失消息。这意味着在极高的负载下，如果服务器进程试图在收集器已经落后时发送更多的日志消息，那么它可能会被阻塞。相反，syslog倾向于在无法写入消息时丢掉消息，这意味着在这样的情况下它可能会无法记录某些消息，但是它不会阻塞系统的其他部分。

log_directory (string)

当logging_collector被启用时，这个参数决定日志文件将被在哪个目录下创建。它可以被指定为一个绝对路径，也可以被指定为一个相对于集簇数据目录的相对路径。这个参数只能在postgresql.conf文件中或在服务器命令行上设置。默认是pg_log。

log_filename (string)

当logging_collector被启用时，这个参数设置被创建的日志文件的文件名。该值被视为一种strftime模式，因此%转义可以被用来指定根据时间变化的文件名（注意如果有任何依赖时区的%转义，计算将在由log_timezone指定的时区中完成）。被支持的%转义和开放组织的strftime[1]说明中列举的类似。注意系统的strftime不会被直接使用，因此平台相关（非标准）的扩展无法工作。默认是postgresql-%Y-%m-%d_%H%M%S.log。

如果你不使用转义来指定一个文件名，你应该计划使用一个日志轮转工具来避免最终填满整个磁盘。在 8.4 发行之前，如果不存在%转义，PostgreSQL将追加新日志文件创建时间的纪元，但是现在已经不再这样做了。

如果在log_destination中启用了 CSV 格式输出，.csv将会被追加到时间戳日志文件名中来创建 CSV 格式输出（如果log_filename以.log结尾，该后缀会被替换）。

这个参数只能在postgresql.conf文件中或通过服务器命令行进行设置。

log_file_mode (integer)

在 Unix 系统上，当logging_collector被启用时，这个参数设置日志文件的权限（在微软 Windows 上这个参数将被忽略）。这个参数值应当是一个数字形式的模式，它可以被chmod和umask系统调用接受（要使用通常的八进制格式，该数字必须以一个0（零）开始）。

默认的权限是0600，表示只有服务器拥有者才能读取或写入日志文件。其他常用的设置是0640，它允许拥有者的组成员读取文件。不过要注意你需要修改log_directory为将文件存储在集簇数据目录之外的某个位置，才能利用这个设置。在任何情况下，让日志文件变成任何人都可读是不明智的，因为日志文件中可能包含敏感数据。

这个参数只能在postgresql.conf文件中或通过服务器命令行进行设置。

log_rotation_age (integer)

当logging_collector被启用时，这个参数决定单个日志文件的最长使用时间。经过指定的分钟数之后，将创建一个新的日志文件。将这个参数设置为零将禁用基于时间的新日志文件创建。这个参数只能在postgresql.conf文件中或在服务器命令行上设置。

log_rotation_size (integer)

当logging_collector被启用时，这个参数决定一个个体日志文件的最大尺寸。当指定千字节数的数据被写入一个日志文件后，将创建一个新的日志文件。设置为零时将禁用基于大小创建新的日志文件。这个参数只能在postgresql.conf文件中或在服务器命令行上设置。

log_truncate_on_rotation (boolean)

当logging_collector被启用时，这个参数将导致PostgreSQL截断（覆盖而不是追加）任何已有的同名日志文件。不过，

[1] http://pubs.opengroup.org/onlinepubs/009695399/functions/strftime.html

截断只在一个新文件由于基于时间的轮转被打开时发生，
在服务器启动或基于尺寸的轮转时不会发生。如果被关闭，
在所有情况下以前存在的文件将被追加。例如，使用这个设置和一个类似postgresql-%H.log的log_filename将导致产生 24 个每小时的日志文件，并且循环地覆盖它们。
这个参数只能在postgresql.conf文件中或在服务器命令行上设置。

示例：保留7天的日志，每天一个日志文件，命名为server_log.Mon，server_log.Tue，等等，并自动用本周的日志覆盖上周的日志，将log_filename设置为server_log.%a，将log_truncate_on_rotation设置为on，将log_rotation_age设置为1440。

示例：要保留 24 小时的日志，每个小时一个日志文件，如果日志文件尺寸超过 1GB，也会提前轮转。可以这样做：将log_filename设置为server_log.%H%M、将log_truncate_on_rotation设置为on、将log_rotation_age设置为60并且将log_rotation_size设置为1000000。在log_filename中包含%M，
可让按文件大小触发的轮转选用与整点初始文件名不同的新文件名。

syslog_facility (enum)

当启用了向syslog记录时，这个参数决定要使用的syslog“设施”。你可以在LOCAL0、LOCAL1、LOCAL2、LOCAL3、LOCAL4、LOCAL5、LOCAL6、LOCAL7中选择，
默认值是LOCAL0。还请参阅系统的syslog守护进程的文档。
这个参数只能在postgresql.conf文件中或在服务器命令行上设置。

syslog_ident (string)

当启用了向syslog记录时，这个参数决定用来标识
syslog中的PostgreSQL消息的程序名。默认值是postgres。
这个参数只能在postgresql.conf文件中或在服务器命令行上设置。

event_source (string)

当启用了向事件日志记录时，这个参数决定用来标识日志中PostgreSQL消息的程序名。
默认值是PostgreSQL。这个参数只能在postgresql.conf
文件中或在服务器命令行上设置。

18.8.2. 什么时候记录日志

client_min_messages (enum)

控制将哪些消息级别发送给客户端。有效值是DEBUG5、DEBUG4、DEBUG3、DEBUG2、DEBUG1、LOG、NOTICE、WARNING、ERROR、FATAL、和PANIC。
每个级别都包括其后的所有级别。级别越靠后，被发送的消息越少。默认值是NOTICE。
注意，LOG在这里的排名与在 log_min_messages中不同。

log_min_messages (enum)

控制将哪些消息级别写入服务器日志。有效值是DEBUG5、DEBUG4、DEBUG3、DEBUG2、DEBUG1、INFO、NOTICE、WARNING、ERROR、LOG、FATAL 和 PANIC。每个级别都包含它后面的所有级别。级别越靠后，发送到日志的消息越少。默认值是WARNING。注意，LOG在这里的排名与在 client_min_messages中不同。只有超级用户可以更改此设置。

log_min_error_statement (enum)

控制在服务器日志中记录哪些导致错误条件的SQL语句。
对于达到指定严重级别或更高级别的消息，其日志条目中会包含当前 SQL语句。有效值是DEBUG5、DEBUG4、DEBUG3、DEBUG2、DEBUG1、INFO、NOTICE、WARNING、

ERROR、LOG、FATAL 和 PANIC。默认值是ERROR，表示导致错误、日志消息、致命错误或恐慌的语句将被记录。要有效地关闭失败语句的记录，可将此参数设置为PANIC。只有超级用户可以更改此设置。

log_min_duration_statement (integer)

如果一条已完成语句的运行时间至少达到指定毫秒数，就记录其持续时间。将此值设置为零会打印所有语句的持续时间。负一（默认值）禁用语句持续时间记录。例如，如果设置为250ms，则会记录所有运行 250ms 或更长时间的 SQL 语句。启用此参数有助于发现应用中未优化的查询。只有超级用户能更改这个设置。

对于使用扩展查询协议的客户端，Parse、Bind 和 Execute 步骤的持续时间将被独立记录。

> **注意**
>
> 当把这个选项和log_statement一起使用时，已经被log_statement记录的语句文本不会在持续时间日志消息中重复。如果你没有使用syslog，我们推荐你使用log_line_prefix记录 PID 或会话 ID，这样你可以使用进程 ID 或会话 ID 把语句消息链接到后来的持续时间消息。

表 18.1解释了PostgreSQL所使用的消息严重级别。如果日志输出被发送到syslog或Windows 的eventlog，严重级别会按照表中所示进行转换。

表 18.1. 消息严重级别

严重性	用法	syslog	eventlog
DEBUG1..DEBUG5	为开发者提供逐级更加详细的信息。	DEBUG	INFORMATION
INFO	提供用户隐式要求的信息，例如来自VACUUM VERBOSE的输出。	INFO	INFORMATION
NOTICE	提供可能对用户有用的信息，例如长标识符截断提示。	NOTICE	INFORMATION
WARNING	提供可能出现的问题的警告，例如在一个事务块外COMMIT。	NOTICE	WARNING
ERROR	报告一个导致当前命令中断的错误。	WARNING	ERROR
LOG	报告管理员可能感兴趣的信息，例如检查点活动。	INFO	INFORMATION
FATAL	报告一个导致当前会话中断的错误。	ERR	ERROR
PANIC	报告一个导致所有数据库会话中断的错误。	CRIT	ERROR

18.8.3. 记录哪些内容

application_name (string)

application_name可以是任意小于NAMEDATALEN个字符（标准编译中是 64 个字符）的字符串。应用通常在连接服务器时设置此值。该名称将被显示在pg_stat_activity视图中并被包括在 CSV 日志项中。也可以通过log_line_prefix将其包括在普通日志项

中。只有可打印 ASCII 字符能被使用在application_name之中。其他字符将被替换为问号（?）。

debug_print_parse (boolean)
debug_print_rewritten (boolean)
debug_print_plan (boolean)

这些参数将会让多种调试输出被发出。当被设置时，
它们为每一个被执行的查询打印结果分析树、查询重写器输出或执行计划。这些消息在LOG消息级别上被发出，
因此默认情况下它们将出现在服务器日志中但不会被发送到客户端。你可以通过调整client_min_messages和/或log_min_messages来改变这种情况。
这些参数默认是关闭的。

debug_pretty_print (boolean)

当被设置时，debug_pretty_print会缩进由debug_print_parse、debug_print_rewritten或 debug_print_plan产生的输出。这将导致比关闭参数时使用的“紧凑”模式可读性更强但是更长的输出。它默认是打开的。

log_checkpoints (boolean)

导致检查点和重启点在服务器日志中记录。日志消息中包括一些统计信息，
包括写入的缓冲区数量和写入它们所花费的时间。此参数只能在 postgresql.conf文件或服务器命令行中设置。默认值为关闭。

log_connections (boolean)

记录每一次到服务器的连接尝试，以及客户端认证的成功完成。
这个参数在会话开始后不能更改。默认为关闭。

> **注意**
>
> 某些客户端程序（例如psql）在判断是否需要密码时会尝试连接两次，
> 因此重复的“收到连接”消息并不一定表示一个错误。

log_disconnections (boolean)

在会话终止时向服务器日志输出一行与log_connections类似的内容，
并且包含会话的持续时间。默认情况下这是关闭的。这个参数在会话开始后不能更改。

log_duration (boolean)

记录每个已完成语句的持续时间。默认值为off。只有超级用户能更改这个设置。

对于使用扩展查询协议的客户端，Parse、Bind 和 Execute 步骤的持续时间将被独立记录。

> **注意**
>
> 启用这个选项和设置log_min_duration_statement为零之间的区别是，超过log_min_duration_statement指定的时长会强制记录查询文本，
> 而这个选项不会。因此，如果log_duration为on并且log_min_duration_statement为正值，所有持续时间都将被记录，
> 但是只有超过阈值的语句才会被记录查询文本。
> 这种行为有助于在高负载安装中收集统计信息。

log_error_verbosity (enum)

控制在服务器日志中记录的每条消息的详细程度。有效值为TERSE，DEFAULT和VERBOSE，它们依次在显示的消息中增加更多字段。TERSE不包括DETAIL，HINT，QUERY和CONTEXT错误信息的记录。VERBOSE输出包括SQLSTATE错误代码（另请参见附录 A）以及生成错误的源代码文件名、函数名和行号。只有超级用户能更改这个设置。

log_hostname (boolean)

默认情况下，连接日志消息只显示连接主机的 IP 地址。打开这个参数将导致也记录主机名。注意根据你的主机名解析设置，这可能会导致不可忽视的性能开销。这个参数只能在postgresql.conf文件中或在服务器命令行上设置。

log_line_prefix (string)

这是一个printf风格的字符串，会输出在每个日志行的开头。%字符用于引入“转义序列”，它们会被替换为下表所述的状态信息。
无法识别的转义会被忽略。其他字符会直接复制到日志行中。
有些转义只被会话进程识别，后台进程（例如主服务器进程）会忽略它们。此参数只能在postgresql.conf文件中或在服务器命令行上设置。默认值是一个空字符串。

转义	效果	只限会话
%a	应用名	是
%u	用户名	是
%d	数据库名	是
%r	远程主机名或 IP 地址，以及远程端口	是
%h	远程主机名或 IP 地址	是
%p	进程 ID	否
%t	无毫秒的时间戳	否
%m	带毫秒的时间戳	否
%i	命令标签：会话当前命令的类型	是
%e	SQLSTATE 错误代码	否
%c	会话 ID：见下文	否
%l	对每个会话或进程的日志行号，从 1 开始	否
%s	进程开始的时间戳	否
%v	虚拟事务 ID (backendID/local XID)	否
%x	事务 ID（如果未分配则为 0）	否
%q	不产生输出，但是告诉非会话进程在字符串的这一点停止；会话进程忽略	否
%%	字面字符 %	否

该%c转义会打印一个近乎唯一的会话标识符，由两个以点分隔的 4 字节十六进制数（不含前导零）组成。这两个数分别是进程启动时间和进程 ID，因此%c也可以用作节省空间的方式来打印这些信息。例如，要从pg_stat_activity生成会话标识符，
可以使用以下查询：

```
SELECT to_hex(EXTRACT(EPOCH FROM backend_start)::integer) || '.'
 ||
```

```
        to_hex(pid)
FROM pg_stat_activity;
```

> **提示**
>
> 如果你为log_line_prefix设置了非空值，
> 你通常应该让它的最后一个字符为空格，
> 这样用以提供和日志行的剩余部分的视觉区别。也可以使用标点符号。

> **提示**
>
> Syslog产生自己的时间戳和进程 ID 信息，因此如果你记录到syslog你可能不希望包括那些转义。

log_lock_waits (boolean)

控制会话为获取锁而等待的时间超过deadlock_timeout时是否生成日志消息。
这对于确定锁等待是否导致性能不佳很有用。默认值为off。
只有超级用户能更改这个设置。

log_statement (enum)

控制哪些 SQL 语句被记录。有效值是 none (off)、ddl、mod和 all（所有语句）。ddl记录所有数据定义语句，例如CREATE、ALTER和 DROP语句。mod记录所有ddl语句，外加INSERT、UPDATE、DELETE、TRUNCATE和COPY FROM等数据修改语句。如果PREPARE、EXECUTE和 EXPLAIN ANALYZE包含合适类型的命令，它们也会被记录。对于使用扩展查询协议的客户端，当收到一个 Execute 消息时会产生日志并且会包括 Bind 参数的值（任何内嵌的单引号会被双写）。

默认值为none。只有超级用户能更改这个设置。

> **注意**
>
> 即使使用log_statement = all设置，包含简单语法错误的语句也不会被记录。
> 这是因为只有在完成基本语法解析并确定了语句类型之后才会发出日志消息。
> 在扩展查询协议的情况下，在 Execute 阶段之前（即在解析分析或规划期间）出错的语句也不会被记录。将log_min_error_statement设置为ERROR（或更低）来记录这种语句。

log_temp_files (integer)

控制临时文件名和大小的日志记录。临时文件可以用于排序、hash 和临时查询结果。每当删除临时文件时都会发出日志记录。值为零时记录所有临时文件信息，
而正值仅记录大小大于或等于指定千字节数的文件。默认设置为-1，禁用此类日志记录。
只有超级用户能更改这个设置。

log_timezone (string)

设置在服务器日志中写入的时间戳的时区。和TimeZone不同，这个值是集簇范围的，
因此所有会话将报告一致的时间戳。内置默认值是GMT，但是通常会被在postgresql.conf中覆盖。initdb将安装一个对应于其系统环境的设置。详见第 8.5.3 节。这个参数只能在postgresql.conf文件中或在服务器命令行上设置。

18.8.4. 使用 CSV 格式的日志输出

将csvlog加入log_destination列表中，可以方便地将日志文件导入数据库表。此选项以逗号分隔值（CSV）格式输出日志行，包含以下列：带毫秒的时间戳、用户名、数据库名、进程 ID、客户端主机:端口号、会话 ID、会话内行号、命令标签、会话开始时间、虚拟事务 ID、常规事务 ID、错误严重性、SQLSTATE 代码、错误消息、错误消息详情、提示、引发错误的内部查询（如果有）、该内部查询中错误位置的字符数、错误上下文、引发错误的用户查询（如果有且由log_min_error_statement启用）、该用户查询中错误位置的字符数、错误在 PostgreSQL 源代码中的位置（如果log_error_verbosity设置为verbose）和应用名称。以下是用于存储 CSV 格式日志输出的示例表定义：

```
CREATE TABLE postgres_log
(
  log_time timestamp(3) with time zone,
  user_name text,
  database_name text,
  process_id integer,
  connection_from text,
  session_id text,
  session_line_num bigint,
  command_tag text,
  session_start_time timestamp with time zone,
  virtual_transaction_id text,
  transaction_id bigint,
  error_severity text,
  sql_state_code text,
  message text,
  detail text,
  hint text,
  internal_query text,
  internal_query_pos integer,
  context text,
  query text,
  query_pos integer,
  location text,
  application_name text,
  PRIMARY KEY (session_id, session_line_num)
);
```

使用COPY FROM命令将一个日志文件导入到这个表中：

```
COPY postgres_log FROM '/full/path/to/logfile.csv' WITH csv;
```

你可以做一些事情来简化导入 CSV 日志文件：

1. 设置log_filename和log_rotation_age，为日志文件提供一致且可预测的命名方案。这样就能预测文件名，并知道单个日志文件何时已完成写入、可以导入。

2. 将log_rotation_size设置为 0 来禁用基于尺寸的日志轮转，因为它使得日志文件名难以预测。

3. 将log_truncate_on_rotation设置为on，这样在同一个文件中旧日志数据不会与新数据混杂。

4. 上述表定义包括一个主键声明。这有助于避免意外地两次导入相同的信息。COPY 命令一次提交所有它导入的数据，因此任何错误将导致整个导入失败。如果你导入一个部分完成的日志文件并且稍后当它完全完成后再次导入，

主键冲突将导致导入失败。请等到日志完成且被关闭之后再导入。
这个过程也可以避免意外地导入部分完成的行，这种行也将导致COPY失败。

18.9. 运行时统计数据

18.9.1. 查询和索引统计信息收集器

这些参数控制服务器范围内的统计信息收集功能。启用统计信息收集后，
产生的数据可以通过pg_stat和 pg_statio系列系统视图进行访问。更多信息请参阅第 27 章。

track_activities (boolean)

启用对每个会话当前执行命令的信息收集，包括命令开始执行的时间。
此参数默认为开启状态。请注意，即使启用了此参数，
该信息也只有超级用户和拥有被报告会话的用户才能看到，因此不应构成安全风险。
只有超级用户能更改这个设置。

track_activity_query_size (integer)

为每个活动会话指定存储当前执行命令的文本所预留的字节数，它们被用于pg_stat_activity.query字段。默认值是 1024字节。这个参数只能在服务器启动时被设置。

track_counts (boolean)

启用对数据库活动的统计信息收集。此参数默认为开启，
因为自动清理守护进程需要这些收集到的信息。只有超级用户能更改这个设置。

track_io_timing (boolean)

启用数据库I/O调用的计时。默认情况下，此参数处于关闭状态，
因为它将重复查询操作系统的当前时间，这可能会在某些平台上造成显著的开销。
你可以使用pg_test_timing工具来测量系统上计时的开销。I/O计时信息显示在pg_stat_database中，也显示在使用BUFFERS选项的EXPLAIN输出中，并由pg_stat_statements提供。只有超级用户能更改这个设置。

track_functions (enum)

启用函数调用次数和耗时的跟踪。指定pl以仅跟踪过程语言函数，all
以同时跟踪SQL和C语言函数。默认值为none，即禁用函数统计跟踪。
只有超级用户能更改这个设置。

> **注意**
>
> 简单到足以被“内联”到调用查询中的 SQL 语言函数不会被跟踪，而不管这个设置。

update_process_title (boolean)

启用后，每次服务器接收到新的 SQL 命令时都会更新进程标题。进程标题通常通过ps命令查看，在 Windows 上则可以使用Process Explorer。只有超级用户能更改这个设置。

stats_temp_directory (string)

设置存放临时统计信息数据的目录。它既可以是相对于数据目录的路径，
也可以是绝对路径。默认值是 pg_stat_tmp。将其指向基于 RAM 的文件系统可以减少物理 I/O 需求，并可能提升性能。这个参数只能在postgresql.conf文件中或服务器命令行上设置。

18.9.2. 统计监控

log_statement_stats (boolean)
log_parser_stats (boolean)
log_planner_stats (boolean)
log_executor_stats (boolean)

对于每个查询，将各自模块的性能统计输出到服务器日志中。
这是一个简单的性能分析工具，类似于Unix getrusage()操作系统功能。log_statement_stats报告整个语句的统计信息，而其他选项报告每个模块的统计信息。
log_statement_stats不能与任何单独模块选项一起启用。
所有这些选项默认情况下都是禁用的。只有超级用户才能更改这些设置。

18.10. 自动清理

这些设置控制autovacuum特性的行为。更多信息见第 23.1.6 节。

autovacuum (boolean)

控制服务器是否运行自动清理启动器后台进程。默认为开启，
不过要使自动清理正常工作，还需要启用track_counts。该参数只能在postgresql.conf文件或服务器命令行中设置，不过，通过更改表存储参数可以为表禁用自动清理。

注意即使该参数被禁用，系统也会在需要防止事务ID回卷时发起自动清理进程。
详情请见第 23.1.5 节。

log_autovacuum_min_duration (integer)

当自动清理执行的操作运行时间至少达到指定毫秒数时，就会记录该操作。
将此设置为零会记录所有自动清理操作。负一（默认值）会禁用记录自动清理操作。
例如，如果将其设置为250ms，
则所有运行时间为250ms或更长的自动清理和分析都将被记录。此外，
当此参数设置为任何非-1值时，如果由于冲突锁而跳过自动清理操作，则会记录消息。
启用此参数可帮助跟踪自动清理活动。此参数只能在postgresql.conf
文件或服务器命令行中设置；但可以通过更改表存储参数来覆盖对单个表的设置。

autovacuum_max_workers (integer)

指定任一时刻可能运行的自动清理进程（自动清理启动器除外）的最大数量。
默认值为三个。这个参数只能在服务器启动时设置。

autovacuum_naptime (integer)

指定自动清理在任意给定数据库上各次运行之间的最小间隔。
在每一轮中后台进程检查数据库并根据需要为数据库中的表发出VACUUM和ANALYZE
命令。延迟以秒为单位。默认值为1分钟（1min）。该参数只能在postgresql.conf
文件或在服务器命令行上设置。

autovacuum_vacuum_threshold (integer)

指定能在一个表上触发VACUUM的被更新或被删除元组的最小数量。默认值为50个元组。
该参数只能在postgresql.conf文件或在服务器命令行上设置。
但是对个别表可以通过修改表存储参数来覆盖该设置。

autovacuum_analyze_threshold (integer)

指定能在一个表上触发ANALYZE的被插入、被更新或被删除元组的最小数量。
默认值为50个元组。该参数只能在postgresql.conf文件或在服务器命令行上设置。
但是对个别表可以通过修改表存储参数来覆盖该设置。

autovacuum_vacuum_scale_factor (floating point)

指定一个表尺寸的分数，在决定是否触发VACUUM时将它加到autovacuum_vacuum_threshold上。默认值为0.2（表尺寸的20%）。该参数只能在postgresql.conf文件或在服务器命令行上设置。但是对个别表可以通过修改表存储参数来覆盖该设置。

autovacuum_analyze_scale_factor (floating point)

指定一个表尺寸的分数，在决定是否触发ANALYZE时将它加到autovacuum_analyze_threshold上。默认值为0.1（表尺寸的10%）。该参数只能在postgresql.conf文件或在服务器命令行上设置。但是对个别表可以通过修改表存储参数来覆盖该设置。

autovacuum_freeze_max_age (integer)

指定在一个VACUUM操作被强制执行来防止表中事务ID回卷之前，一个表的pg_class.relfrozenxid域能保持的最大年龄（事务的）。注意即便自动清理被禁用，系统也将发起自动清理进程来阻止回卷。

清理也允许从pg_clog子目录中移除旧文件，这也是为什么默认值被设置为较低的2亿事务。该参数只能在服务器启动时设置，但是对于个别表可以通过修改表存储参数来降低该设置。详见第 23.1.5 节。

autovacuum_vacuum_cost_delay (integer)

指定用于自动VACUUM操作中的代价延迟值。如果指定-1，则使用vacuum_cost_delay值。默认值为20毫秒。该参数只能在postgresql.conf文件或在服务器命令行上设置。但是对个别表可以通过修改表存储参数来覆盖该设置。

autovacuum_vacuum_cost_limit (integer)

指定用于自动VACUUM操作中的代价限制值。如果指定-1（默认值），则使用vacuum_cost_limit值。
注意该值被按比例地分配到运行中的自动清理工作进程上（如果有多个），因此每一个工作进程的限制值之和不会超过这个变量中的值。该参数只能在postgresql.conf文件或在服务器命令行上设置。
但是对个别表可以通过修改表存储参数来覆盖该设置。

18.11. 客户端连接默认值

18.11.1. 语句行为

search_path (string)

这个变量指定当一个对象（表、数据类型、函数等）被用一个无模式限定的简单名称引用时，搜索该对象时的模式顺序。当在不同模式中有同名对象时，将使用第一个在搜索路径中被找到的对象。一个不属于搜索路径中任何一个模式的对象只能通过用限定名（带点号）指定包含它的模式来引用。

search_path的值必须是一个逗号分隔的模式名列表。任何不是一个已有模式的名称，或者是一个用户不具有USAGE权限的模式，将被静默忽略。

如果列表项之一是特殊名$user，则具有SESSION_USER返回的名字的模式将取代它（如果有这样一个模式并且该用户有该模式的USAGE权限；如果没有，$user会被忽略）。

系统目录模式pg_catalog总是被搜索，不管它是否在搜索路径中被提及。如果它在路径中被提及，那么它将被按照路径指定的顺序搜索。如果pg_catalog不在路径中，则它将在任何路径项之前被搜索。

同样，如果存在，当前会话的临时表模式pg_temp_*nnn*总是被搜索。可以通过使用别名pg_temp在路径中显式列出它。如果它没有列在路径中，则首先搜索它（甚至在pg_catalog之前）。然而，临时模式仅用于搜索关系（表、视图、序列等）和数据类型名称。它永远不会用于搜索函数或操作符名称。

当对象创建时没有指定一个特定目标模式，它们将被放置在search_path中第一个合法模式中。如果搜索路径为空将报告一个错误。

这个参数的默认值是"$user"，public。这种设置支持共享使用数据库（用户没有私有模式，所有人共享使用public）、每个用户拥有私有模式，以及二者的组合。还可以通过全局或针对每个用户修改默认搜索路径设置来获得其他效果。

更多有关模式处理的信息，请参考第 5.7 节。

搜索路径的当前有效值可以通过SQL函数current_schemas检查（见第 9.25 节）。它和检查search_path的值不太一样，因为current_schemas显示出现在search_path中的项如何被解析。

default_tablespace (string)

这个变量指定当一个CREATE命令没有显式指定一个表空间时，创建对象（表和索引）的默认表空间。

该值要么是一个表空间的名字，要么是一个指定使用当前数据库默认表空间的空字符串。如果该值和任何现有表空间的名字都不匹配，PostgreSQL将自动使用当前数据库的默认表空间。如果指定了一个非默认的表空间，用户必须对它有CREATE权限，否则创建尝试将失败。

这个变量不被用于临时表，对临时表会使用temp_tablespaces。

创建数据库时也不会使用这个变量。默认情况下，一个新数据库会从它的模板数据库继承其表空间设置。

有关表空间的更多的信息，请见第 21.6 节。

temp_tablespaces (string)

这个变量指定当一个CREATE命令没有显式指定一个表空间时，创建临时对象（临时表和临时表上的索引）的默认表空间。用于排序大型数据集的临时文件也被创建在这些表空间中。

该值是一个表空间名字的列表。当列表中有多于一个名称时，每次一个临时对象被创建时PostgreSQL随机选择列表中的一个成员。例外是在一个事务中，连续创建的临时对象被依次放置在列表中的连续表空间中。如果列表中被选中的元素是一个空字符串，PostgreSQL将自动使用当前数据库的默认表空间。

当temp_tablespaces被交互式地设置时，指定一个不存在的表空间是一种错误，指定一个用户不具有CREATE权限的表空间也同样是错误。不过，当使用一个之前设置的值时，不存在的表空间会被忽略，就像用户缺少CREATE权限的表空间一样。特别是，使用一个在postgresql.conf中设置的值时，这条规则起效。

默认值是一个空字符串，它使得所有临时对象被创建在当前数据库的默认表空间中。

参阅default_tablespace。

check_function_bodies (boolean)

这个参数通常为打开。当设置为off时，它禁用CREATE FUNCTION期间对函数体字符串的验证。禁用验证避免了验证处理的副作用，也避免前向引用等问题导致的误报。在代表其他用户载入函数之前设置这个参数为off；pg_dump会自动这样做。

default_transaction_isolation (enum)

每个 SQL 事务都有一个隔离级别，可以是“读未提交”、“读已提交”、“可重复读”或者“可串行化”。这个参数控制每个新事务的默认隔离级别。默认是“读已提交”。

更多信息请参阅第 13 章和SET TRANSACTION。

default_transaction_read_only (boolean)

一个只读的 SQL 事务不能修改非临时表。这个参数控制每个新事务的默认只读状态。默认是off（读/写）。

更多信息请参考SET TRANSACTION。

default_transaction_deferrable (boolean)

当运行在serializable隔离级别时，一个可延迟只读 SQL 事务可能在获准继续之前被延迟一段时间。但是，一旦它开始执行就不会产生任何用来保证可串行性的开销；
因此串行化代码将没有任何理由因为并发更新而强制它中止，
使得这个选项适合于长时间运行的只读事务。

这个参数控制每个新事务的默认可延迟状态。
目前它对读写事务或者那些运行在低于serializable隔离级别上的事务无效。
默认值是off。

更多信息请参考SET TRANSACTION。

session_replication_role (enum)

控制当前会话中与复制相关的触发器和规则的触发。设置此变量需要超级用户权限，
并会丢弃任何先前缓存的查询计划。可用值为origin（默认值）、replica和local。
更多信息参见ALTER TABLE。

statement_timeout (integer)

中止任何使用了超过指定毫秒数的语句，从客户端命令到达服务器时开始计时。如果
log_min_error_statement被设置为ERROR或更低，语句如果超时也会被记录。
一个零值（默认）将禁用超时。

不建议在 postgresql.conf中设置statement_timeout，因为它会影响所有会话。

vacuum_freeze_table_age (integer)

如果表的pg_class.relfrozenxid字段达到此设置指定的年龄，VACUUM
就会执行一次全表扫描。默认值为 1.5 亿个事务。尽管用户可以将该值设置在 0 到 20 亿之间，VACUUM仍会悄悄将其有效值限制为不超过autovacuum_freeze_max_age的 95%，以便在针对该表启动防回卷自动清理之前，周期性手工VACUUM仍有机会运行。详见第 23.1.5 节。

vacuum_freeze_min_age (integer)

指定VACUUM在扫描表时用来决定是否用FrozenXID替换事务 ID 的截止年龄（以事务计）。默认值是 5000 万个事务。尽管用户可以将该值设置为 0 到 1 亿之间的任意值，VACUUM会悄悄将有效值限制为不超过autovacuum_freeze_max_age的一半，这样强制 autovacuum 之间就不会间隔过短。更多信息请参见第 23.1.5 节。

bytea_output (enum)

设置bytea类型值的输出格式。有效值是hex（默认）和 escape（传统的 PostgreSQL 格式）。详见第 8.4 节。不管这个设置的值如何，bytea类型总是接受这两种格式的输入。

xmlbinary (enum)

设置二进制值如何被编码为 XML。例如，这适用于通过xmlelement函数或xmlforest函数将bytea值转换到 XML 值。可能的值有base64和hex，它们都是用 XML 模式标准定义的。默认值是base64。更多关于 XML 相关函数的信息可参阅第 9.14 节。

这里的实际选择主要取决于偏好，只受客户端应用中可能存在的限制的约束。两种方法都支持所有可能的值，尽管十六进制编码会比 base64 编码略大。

xmloption (enum)

设置在 XML 与字符串值之间进行转换时，隐含采用DOCUMENT还是CONTENT。有关说明参见第 8.13 节。有效值是DOCUMENT和CONTENT。默认值是CONTENT。

根据 SQL 标准，设置这个选项的命令是：

```
SET XML OPTION { DOCUMENT | CONTENT };
```

这种语法在 PostgreSQL 也可用。

18.11.2. 区域设置和格式化

DateStyle (string)

设置日期和时间值的显示格式，以及解释有歧义的日期输入值的规则。由于历史原因，这个变量包含两个独立的部分：输出格式声明（ISO、Postgres、SQL或German）、输入/输出的年/月/日顺序（DMY、MDY或YMD）。这些可以被独立设置或者一起设置。关键字Euro和European是DMY的同义词；关键字US、NonEuro和NonEuropean是MDY的同义词。详见第 8.5 节。内置默认值是ISO， MDY，但是initdb将用对应于选中的lc_time区域设置行为的设置初始化配置文件。

IntervalStyle (enum)

设置时间间隔值的显示格式。值sql_standard会生成符合SQL标准时间间隔字面量的输出。值postgres（默认值）的输出与PostgreSQL 8.4 之前版本中DateStyle设为ISO时的输出一致。值postgres_verbose的输出与PostgreSQL 8.4 之前版本中DateStyle设为非ISO输出时的输出一致。值iso_8601会生成符合 ISO 8601 第 4.4.3.2 节定义的时间间隔“带标志符格式”的输出。

IntervalStyle参数也会影响对有歧义的时间间隔输入的解释。详见第 8.5.4 节。

TimeZone (string)

设置用于显示和解释时间戳的时区。内置默认值是GMT，但是它通常会在postgresql.conf中被覆盖；initdb将安装一个对应于其系统环境的设置。详见第 8.5.3 节。

timezone_abbreviations (string)

设置服务器接受的日期时间输入中使用的时区缩写集合。默认值为'Default'，这个集合在全世界大多数地方都能工作。也还有'Australia'和'India'，以及可能为一种特定安装定义的其他集合。详见第 B.3 节。

extra_float_digits (integer)

此参数调整浮点值显示的位数，包括float4、float8和几何数据类型。参数值会加到标准位数（按情况使用FLT_DIG或DBL_DIG）上。此值最高可以设置为3，以包含部分有效的数位；这对于转储需要精确还原的浮点数据特别有用。也可以设置为负数，以抑制不需要的数位。另见第 8.1.3 节。

client_encoding (string)

设置客户端编码（字符集）。默认使用数据库编码。
PostgreSQL服务器所支持的字符集在第 22.3.1 节中描述。

lc_messages (string)

设置消息显示的语言。可接受的值是系统相关的；详见第 22.1 节。如果这个变量被设置为空字符串（默认），那么该值将以一种系统相关的方式从服务器的执行环境中继承。

在一些系统上，这个区域设置类别并不存在。仍然可以设置这个变量，
只是不会有任何效果。同样，所期望语言的翻译消息也可能不存在。在这种情况下，
你将仍然继续看到英文消息。

只有超级用户能更改这个设置，因为它会同时影响发送到服务器日志和客户端的消息，
设置不当可能降低服务器日志的可读性。

lc_monetary (string)

设置用于格式化货币量的区域设置，例如用to_char函数族。
可接受的值是系统相关的；详见第 22.1 节。如果这个变量被设置为空字符串（默认），
那么该值将以一种系统相关的方式从服务器的执行环境中继承。

lc_numeric (string)

设置用于格式化数字的区域设置，例如用to_char函数族。可接受的值是系统相关的；
详见第 22.1 节。如果这个变量被设置为空字符串（默认），那么该值将以一种系统相关的方式从服务器的执行环境中继承。

lc_time (string)

设置用于格式化日期和时间的区域设置，例如用to_char函数族。
可接受的值是系统相关的；详见第 22.1 节。如果这个变量被设置为空字符串（默认），
那么该值将以一种系统相关的方式从服务器的执行环境中继承。

default_text_search_config (string)

选择被那些没有显式参数指定配置的文本检索函数变体使用的文本检索配置。
详见第 12 章。内置默认值是pg_catalog.simple，但是如果能够标识一个匹配该区域设置的配置，initdb将用对应于选中的lc_ctype区域设置的值初始化配置文件。

18.11.3. 其他默认值

local_preload_libraries (string)

这个变量指定一个或者多个要在连接开始时预载入的共享库。如果要载入多个库，
库名之间用逗号分隔。除非用双引号括起，所有库名都会被转换为小写。
这个参数不能在特定会话启动之后更改。

任何用户都能设置这个选项。正因为如此，
能被这样载入的库被严格限制为出现于安装的标准库目录中plugins
子目录下的共享库（保证只有“安全的”库被安装到这里是数据库管理员的责任）。
local_preload_libraries中的项可以显式指定这个目录，例如$libdir/plugins/mylib，或者只是指定库的名称 – mylib 和 $libdir/plugins/mylib的效果是相同的。

与shared_preload_libraries不同，
在会话开始时载入库而不是在第一次使用时载入并没有性能上的优势。相反，
此功能的意图是允许把调试或性能测量库载入特定会话，而无需给出显式的 LOAD 命令。
例如，可以通过 PGOPTIONS 环境变量为此会话设置此参数来启用调试。

如果指定的库没有找到，连接尝试将会失败。

每一个 PostgreSQL 支持的库都有一个“魔法块”，它会被检查以保证兼容性。由于这个原因，非 PostgreSQL 的库不能以这种方式载入。

`dynamic_library_path` (`string`)

如果需要打开一个可以动态装载的模块并且在`CREATE FUNCTION`或`LOAD`命令中指定的文件名没有目录部分（即名字中不包含斜线），
那么系统将搜索这个路径以查找所需的文件。

参数`dynamic_library_path`的值必须是由冒号（Windows上为分号）
分隔的绝对目录路径列表。如果某个列表元素以特殊字符串`$libdir`开头，
则会使用编译时确定的PostgreSQL软件包的库目录来替换`$libdir`；
该目录是标准PostgreSQL发行版所提供模块的安装位置。（使用`pg_config --pkglibdir`可以找出此目录的名称。）例如：

```
dynamic_library_path = '/usr/local/lib/postgresql:/home/my_project/lib:$libdir'
```

或者，在 Windows 环境中：

```
dynamic_library_path = 'C:\tools\postgresql;H:\my_project\lib;$libdir'
```

这个参数的默认值是`'$libdir'`。如果该值被设置为一个空字符串，
则关闭自动路径搜索。

这个参数可以由超级用户在运行时更改，
但以这种方式进行的设置只会持续到客户端连接结束，
因此这种方法应该保留用于开发目的。推荐设置此参数的方法是在`postgresql.conf`配置文件中。

`gin_fuzzy_search_limit` (`integer`)

GIN 索引扫描返回的集合尺寸的软上限。详见第 55.4 节。

18.12. 锁管理

`deadlock_timeout` (`integer`)

指定在检查是否发生死锁之前等待锁的毫秒数。检查死锁相对昂贵，
因此服务器不会每次等待锁时都运行它。
我们乐观地假设在生产应用程序中死锁并不常见，
所以在检查死锁之前只是等待一段时间。
增加此值会减少在不必要的死锁检查中浪费的时间，但会减慢实际死锁错误的报告速度。
默认值为一秒（`1s`），这可能是你在实践中想要的最小值。在负载较重的服务器上，
你可能希望提高它。理想情况下，设置应超过你的典型事务时间，
以提高在等待者决定检查死锁之前释放锁的几率。只有超级用户能更改这个设置。

当log_lock_waits被设置时，这个参数也决定发出关于锁等待的日志之前等待的时间量。
如果你想调查锁延迟，你可能希望设置一个比正常的`deadlock_timeout`小的值。

`max_locks_per_transaction` (`integer`)

共享锁表跟踪在`max_locks_per_transaction` * (max_connections + max_prepared_transactions) 个对象（如表）上的锁。因此，在任何一个时刻，只有不超过这么多个可区分对象能够被锁住。这个参数控制为每个事务分配的对象锁的平均数量。
只要所有事务的锁都能放入锁表中，单个事务就可以锁住更多对象。这
不是能被锁住的行数，那个值是没有限制的。默认值 64 已经被历史证明是足够的，但是

如果你有在一个事务中涉及多个不同表的客户端，可能需要提高这个值。
这个参数只能在服务器启动时设置。

增大这个参数可能会导致PostgreSQL请求比操作系统默认配置所允许的更多的System V共享内存。如有必要，关于如何调整这些参数的信息请参见第 17.4.1 节。

运行备库时，必须将此参数设置为大于或等于主库上的值。否则，备库上将不允许查询。

`max_pred_locks_per_transaction` (`integer`)

共享谓词锁表跟踪在`max_pred_locks_per_transaction` * (max_connections + max_prepared_transactions) 个对象（如表）上的锁。因此，在任何一个时刻，只有不超过这么多个可区分对象能够被锁住。
这个参数控制为每个事务分配的对象锁的平均数量。只要所有事务的锁都能放入锁表中，单个事务就可以锁住更多对象。这不是能被锁住的行数，那个值是没有限制的。默认值 64 已经在测试中被证明通常是足够的，但是如果你有会在单个可串行化事务中访问许多不同表的客户端，则可能需要提高这个值。这个参数只能在服务器启动时设置。

增大这个参数可能会导致PostgreSQL请求比操作系统默认配置所允许的更多的System V共享内存。如有必要，关于如何调整这些参数的信息请参见第 17.4.1 节。

18.13. 版本和平台兼容性

18.13.1. 以前的 PostgreSQL 版本

`array_nulls` (`boolean`)

这个参数控制数组输入解析器是否把未用引号的`NULL`识别为一个值为空值的数组元素。默认为`on`，允许输入包含空值的数组值。但是PostgreSQL 8.2 之前的版本不支持数组中的空值，并且因此将把`NULL`当作指定一个值为字符串 “NULL” 的正常数组元素。
为了向后兼容那些要求旧行为的应用，这个变量可以被设置为`off`。

注意即使这个变量为`off`也能够创建包含空值的数组值。

`backslash_quote` (`enum`)

这个参数控制字符串字面量中的单引号是否能够用`\'`来表示。首选的 SQL 标准的方法是将其双写（`''`），但是PostgreSQL在历史上也接受`\'`。不过使用`\'`
容易导致安全风险，因为在某些客户端字符集编码中，
有多字节字符的最后一个字节在数值上等价于 ASCII 的`\`。如果客户端代码没有做到正确转义，那么就可能遭到 SQL 注入攻击。如果服务器拒绝看起来带有被反斜线转义的单引号的查询，那么就可以避免这种风险。`backslash_quote`的可用值是`on`
（总是允许`\'`）、`off`（总是拒绝）以及`safe_encoding`
（只有客户端编码不允许在多字节字符中存在 ASCII `\`时允许）。`safe_encoding`是默认设置。

注意在符合标准的字符串字面量中，`\`就表示`\`。
这个参数只影响不符合标准的字面量的处理，包括转义字符串语法（`E'...'`）。

`default_with_oids` (`boolean`)

此参数控制在既未指定`WITH OIDS`也未指定`WITHOUT OIDS`时，`CREATE TABLE`和`CREATE TABLE AS`是否在新建表中包含 OID 列。它也决定`SELECT INTO`创建的表是否包含 OID。此参数默认为`off`；在PostgreSQL 8.0 及更早版本中，默认为打开。

在用户表中使用 OID 已被弃用，因此大多数安装应让此变量保持禁用状态。需要为特定表使用 OID 的应用应在创建表时指定`WITH OIDS`。可以启用此变量，以兼容不遵循这种做法的旧应用。

escape_string_warning (boolean)

打开时，如果在普通字符串字面量中（'...'语法）出现了一个反斜线（\）并且standard_conforming_strings为关闭，那么就会发出一个警告。默认值是on。

希望使用反斜线作为转义符的应用应该被修改来使用转义字符串语法（E'...'），因为按照 SQL 标准，普通字符串现在默认将反斜线视作一个普通字符。这个变量可以被启用来帮助定位需要被更改的代码。

lo_compat_privileges (boolean)

在PostgreSQL 9.0 之前的版本中，大对象没有访问权限，实际上所有用户都可以读取和写入它们。将此变量设置为on会禁用新的权限检查，以保持与之前版本的兼容性。默认值为off。

设置此变量不会禁用所有与大对象相关的安全检查 – 只禁用那些在PostgreSQL 9.0 中默认行为发生变化的检查。例如，lo_import()和lo_export()需要超级用户权限，这与此设置无关。

quote_all_identifiers (boolean)

当数据库产生 SQL 时，强制所有标识符被引号包围，即使它们（当前）不是关键字。这将影响EXPLAIN的输出以及pg_get_viewdef等函数的结果。另请参阅pg_dump和pg_dumpall的--quote-all-identifiers选项。

sql_inheritance (boolean)

这个设置控制不带修饰的表引用是否被认为包括继承子表。默认值为on，即子表被包括在内（因此，默认假定有*后缀）。如果设为off，子表不被包括（因此，假定有ONLY前缀）。SQL 标准要求包括子表，因此off设置不符合标准，但提供它是为了与 7.1 之前版本的PostgreSQL兼容。更多信息见第 5.8 节。

将sql_inheritance关闭已被弃用，因为这种行为已被发现既容易出错又违背 SQL 标准。本手册其他地方对继承行为的讨论通常都假定它为on。

standard_conforming_strings (boolean)

控制普通字符串字面量（'...'）是否按照 SQL 标准把反斜线当普通文本。从PostgreSQL 9.1 开始，默认值为on（之前的发行中默认值为off）。应用可以检查这个参数来判断字符串字面量如何被处理。这个参数的存在也可以被当做转义字符串语法（E'...'）被支持的标志。如果一个应用希望反斜线被当做转义字符，应该使用转义字符串语法（第 4.1.2.2 节）。

synchronize_seqscans (boolean)

它允许对大型表的顺序扫描与其他扫描同步，因此并发扫描可以在几乎相同的时刻读取相同的块，这样可以分担 I/O 负载。当启用这个参数时，一个扫描可能会从表的中间开始并且之后“绕回”到开头以覆盖所有的行，这样可以与已在进行中的扫描活动同步。对于没有ORDER BY子句的查询，这样的扫描可能会在返回行的顺序中造成不可预料的改变。将这个参数设置为off以保证 8.3 之前的行为（顺序扫描总是从表的起始处开始）。默认值是on。

18.13.2. 平台和客户端兼容性

transform_null_equals (boolean)

当打开时，形如*expr* = NULL（或NULL = *expr*）的表达式将被当做*expr* IS NULL，也就是说，如果*expr*计算结果为空值则返回真，否则返回假。正确的 SQL 标准兼容的*expr* = NULL行为总是返回空值（未知）。因此这个参数默认为off。

不过，在Microsoft Access里的过滤表单生成的查询似乎使用*expr* = NULL来测试空值，因此，如果你使用这个接口访问数据库，你可能想把这个选项打开。因为*expr* = NULL形式的表达式总是返回空值（使用SQL 标准解释），它们不是非常有用并且在普通应用中也不常见，因此这个选项实际上没有什么危害。但是新用户常常对涉及空值的表达式语义感到困惑，
因此这个选项默认为关闭。

请注意这个选项只影响= NULL形式，而不影响其它比较操作符或者其它与一些涉及等值操作符的表达式在计算上等效的其他表达式（例如IN）。因此，
这个选项不能普遍修复错误的程序写法。

相关信息请见第 9.2 节。

18.14. 错误处理

exit_on_error (boolean)

如果为true，任何错误将中止当前会话。默认情况下，这个值被设置为false，这样只有FATAL 错误（致命）将中止会话。

restart_after_crash (boolean)

当被设置为true（默认值），PostgreSQL将在一次后端崩溃后自动重新初始化。
让这个值设置为true通常是将数据库可用性最大化的最佳方法。但是在某些环境中，例如PostgreSQL被集群软件调用时，禁用重启可能很有用，
这样集群软件可以得到控制并且采取它认为适当的行动。

18.15. 预置选项

下列“参数”是只读的，在编译或安装 PostgreSQL 时确定。因此，它们未列入示例 postgresql.conf 文件。这些选项报告 PostgreSQL 行为的各个方面，某些应用（特别是管理前端）可能对此感兴趣。

block_size (integer)

报告一个磁盘块的大小。它由编译服务器时BLCKSZ的值确定。默认值是 8192 字节。有些配置变量的含义（例如shared_buffers）会被block_size影响。详见第 18.4 节。

integer_datetimes (boolean)

报告PostgreSQL是否在编译时启用了对 64 位整数日期和时间的支持。构建PostgreSQL时配置--disable-integer-datetimes可以禁用它。默认值是on。

lc_collate (string)

报告对文本数据进行排序所使用的区域设置。更多信息参见第 22.1 节。此值在创建数据库时确定。

lc_ctype (string)

报告决定字符分类的区域设置。更多信息参见第 22.1 节。此值在创建数据库时确定。通常它与lc_collate相同，但特殊应用可能将它设置为不同的值。

max_function_args (integer)

报告函数参数的最大数量。它由编译服务器时的FUNC_MAX_ARGS值决定。默认值是 100 个参数。

max_identifier_length (integer)

报告标识符的最大长度。它由编译服务器时的NAMEDATALEN值减一决定。NAMEDATALEN的默认值是 64；因此max_identifier_length的默认值是 63 字节，在使用多字节编码时，这可能不足 63 个字符。

max_index_keys (integer)

报告索引键的最大数目。它由编译服务器时的INDEX_MAX_KEYS值决定。默认值是 32 个键。

segment_size (integer)

报告一个文件段中可以存储的块（页）的数量。由编译服务器时的RELSEG_SIZE值决定。一个段文件的最大尺寸（以字节计）等于segment_size乘以block_size，默认是 1GB。

server_encoding (string)

报告数据库的编码（字符集）。这是在数据库被创建时决定的。通常，客户端只需要关心client_encoding的值。

server_version (string)

报告服务器的版本号。它是由编译服务器时的PG_VERSION值决定的。

server_version_num (integer)

以整数形式报告服务器的版本号。它是由编译服务器时的PG_VERSION_NUM值决定的。

wal_block_size (integer)

报告一个 WAL 磁盘块的尺寸。由编译服务器时的XLOG_BLCKSZ值决定。默认是 8192 字节。

wal_segment_size (integer)

报告 WAL 段文件中的块（页）数。WAL 段文件以字节计的总大小等于wal_segment_size乘以wal_block_size；默认总大小为 16MB。更多信息参见第 29.4 节。

18.16. 自定义选项

这个特性允许附加模块（例如过程语言）向PostgreSQL添加系统通常不认识的参数。这样便能以标准方式配置扩展模块。

自定义选项的名称由两部分组成：扩展名称和参数名本身，中间用句点分隔，类似于 SQL 中的限定名。例如plpgsql.variable_conflict。

因为自定义选项可能需要在尚未加载相关扩展模块的进程中设置，PostgreSQL将接受任何两部分参数名称的设置。这些变量被视为占位符，在定义它们的模块加载之前没有任何功能。当加载扩展模块时，它将添加其变量定义并根据这些定义转换任何占位符值。如果存在以其扩展名称开头的任何未识别的占位符，将发出警告。

18.17. 开发者选项

下列参数用于PostgreSQL源代码的开发工作，在某些情况下也用于辅助恢复严重损坏的数据库。没有理由在生产数据库中使用它们。

因此，它们被排除在示例postgresql.conf文件之外。请注意，
许多参数需要特殊的源代码编译标志才能起作用。

allow_system_table_mods (boolean)

允许修改系统表的结构。此参数由initdb使用。此参数只能在服务器启动时设置。

ignore_system_indexes (boolean)

读取系统表时忽略系统索引（但是修改系统表时依然同时更新索引）。
这在从被破坏的系统索引中恢复数据时有用。这个参数在会话开始之后不能被更改。

post_auth_delay (integer)

如果非零，则在新服务器进程启动并执行认证过程之后，延迟指定的秒数。
这旨在给开发者一个机会，用调试器附加到服务器进程上。
此参数在会话开始之后不能更改。

pre_auth_delay (integer)

如果非零，则在新服务器进程刚刚派生之后、执行认证过程之前，延迟指定的秒数。
这旨在给开发者一个机会，用调试器附加到服务器进程上，跟踪认证过程中的异常行为。
此参数只能在postgresql.conf文件中或在服务器命令行上设置。

trace_notify (boolean)

为LISTEN和NOTIFY命令生成大量调试输出。client_min_messages或log_min_messages必须是DEBUG1或者更低才能把这种输出分别发送到客户端或者服务器日志。

trace_recovery_messages (enum)

启用原本不会记录的恢复相关调试输出。此参数允许用户覆盖log_min_messages的常规设置，但仅针对特定消息。它旨在用于调试热备。有效值为DEBUG5、DEBUG4、DEBUG3、DEBUG2、DEBUG1和LOG。默认值LOG
完全不影响是否记录消息的决定。
其他值会让该优先级或更高优先级的恢复相关调试消息按LOG优先级记录；对于log_min_messages的常见设置，这会无条件地将它们发送到服务器日志。此参数只能在postgresql.conf文件中或在服务器命令行上设置。

trace_sort (boolean)

如果开启，输出排序操作中的资源使用信息。只有在编译PostgreSQL时定义了TRACE_SORT宏，这个参数才可用（不过，当前在默认情况下就定义了TRACE_SORT）。

trace_locks (boolean)

如果开启，发出锁使用情况的信息。被转储信息中包括锁操作的类型、
锁的类型和被锁或被解锁对象的唯一标识符。
同样包括的还有已经授予这个对象的锁类型的位掩码和等待这个对象的锁类型的位掩码。
还会转储每种锁类型已授予的锁数、等待的锁数，以及它们的总数。
一个日志文件输出的示例如下：

```
LOG:  LockAcquire: new: lock(0xb7acd844) id(24688,24696,0,0,0,1)
      grantMask(0) req(0,0,0,0,0,0,0)=0 grant(0,0,0,0,0,0,0)=0
      wait(0) type(AccessShareLock)
LOG:  GrantLock: lock(0xb7acd844) id(24688,24696,0,0,0,1)
      grantMask(2) req(1,0,0,0,0,0,0)=1 grant(1,0,0,0,0,0,0)=1
      wait(0) type(AccessShareLock)
LOG:  UnGrantLock: updated: lock(0xb7acd844) id(24688,24696,0,0,
0,1)
```

```
        grantMask(0) req(0,0,0,0,0,0,0)=0 grant(0,0,0,0,0,0,0)=0
        wait(0) type(AccessShareLock)
LOG:  CleanUpLock: deleting: lock(0xb7acd844) id(24688,24696,0,
0,0,1)
        grantMask(0) req(0,0,0,0,0,0,0)=0 grant(0,0,0,0,0,0,0)=0
        wait(0) type(INVALID)
```

被转储结构体的详细信息可以在`src/include/storage/lock.h`中找到。

只有在编译PostgreSQL时定义了`LOCK_DEBUG`宏，这个参数才可用。

`trace_lwlocks` (`boolean`)

如果开启，发出轻量级锁的使用信息。
轻量级锁主要是为了提供对共享内存数据结构的互斥访问。

只有在编译PostgreSQL时定义了`LOCK_DEBUG`宏，这个参数才可用。

`trace_userlocks` (`boolean`)

如果开启，发出关于用户锁使用的信息。与`trace_locks`的输出一样，
但只用于咨询锁。

只有在编译PostgreSQL时定义了`LOCK_DEBUG`宏，这个参数才可用。

`trace_lock_oidmin` (`integer`)

如果设置，不会跟踪 OID 小于此值的表上的锁（用于避免在系统表上的输出）。

只有在编译PostgreSQL时定义了`LOCK_DEBUG`宏，这个参数才可用。

`trace_lock_table` (`integer`)

无条件地跟踪此表（OID）上的锁。

只有在编译PostgreSQL时定义了`LOCK_DEBUG`宏，这个参数才可用。

`debug_deadlocks` (`boolean`)

如果设置，当死锁超时发生时，转储所有当前锁的信息。

只有在编译PostgreSQL时定义了`LOCK_DEBUG`宏，这个参数才可用。

`log_btree_build_stats` (`boolean`)

如果设置，会记录 B-树操作上的系统资源使用情况统计（内存和 CPU）。

只有在编译PostgreSQL时定义了`BTREE_BUILD_STATS`宏，这个参数才可用。

`wal_debug` (`boolean`)

如果被打开，WAL 相关的调试输出将被发出。只有在编译PostgreSQL时定义了`WAL_DEBUG`宏的情况下，这个参数才可用。

`zero_damaged_pages` (`boolean`)

检测到损坏的页面头通常会导致PostgreSQL报告错误，中止当前事务。将`zero_damaged_pages`设置为on会导致系统报告警告，将内存中的损坏页面清零，
并继续处理。这种行为会破坏数据，即损坏页面上的所有行。但是，
它确实允许你跳过错误，并从表中可能存在的未损坏页面中检索行。
如果由于硬件或软件错误而发生损坏，这对于恢复数据很有用。

通常在放弃从表的损坏页面恢复数据的希望之前，不应将其设置为on。清零的页面不会强制写入磁盘，因此建议在再次关闭此参数之前重新创建表或索引。默认设置为off。只有超级用户能更改这个设置。

debug_assertions (boolean)

打开各种断言检查。这是一种调试辅助手段。如果你遇到了奇怪的问题或崩溃，可以打开它，因为它可能暴露出编程错误。要使用这个参数，在构建PostgreSQL时必须定义宏USE_ASSERT_CHECKING（通过configure选项--enable-cassert完成）。注意，如果PostgreSQL构建时启用了断言，debug_assertions默认为on。

18.18. 短选项

为了方便起见，系统中还为一些参数提供了单字母的命令行选项开关。它们在表 18.2中描述。其中一些选项是由于历史原因而存在，它们以单字母选项的形式存在，并不一定表示鼓励频繁使用这些选项。

表 18.2. 短选项对照

短选项	等效于
-A *x*	debug_assertions = *x*
-B *x*	shared_buffers = *x*
-d *x*	log_min_messages = DEBUG*x*
-e	datestyle = euro
-fb, -fh, -fi, -fm, -fn, -fo, -fs, -ft	enable_bitmapscan = off, enable_hashjoin = off, enable_indexscan = off, enable_mergejoin = off, enable_nestloop = off, enable_indexonlyscan = off, enable_seqscan = off, enable_tidscan = off
-F	fsync = off
-h *x*	listen_addresses = *x*
-i	listen_addresses = '*'
-k *x*	unix_socket_directory = *x*
-l	ssl = on
-N *x*	max_connections = *x*
-O	allow_system_table_mods = on
-p *x*	port = *x*
-P	ignore_system_indexes = on
-s	log_statement_stats = on
-S *x*	work_mem = *x*
-tpa, -tpl, -te	log_parser_stats = on, log_planner_stats = on, log_executor_stats = on
-W *x*	post_auth_delay = *x*

第 19 章 客户端认证

当客户端应用连接到数据库服务器时，它会指定要以哪个 PostgreSQL 数据库用户名连接，这很像以某个特定用户身份登录 Unix 计算机。在 SQL 环境中，当前活动的数据库用户名决定了对数据库对象的访问权限 – 详见第 20 章。因此，必须限制哪些数据库用户能够连接。

> **注意**
>
> 如第 20 章中所述，PostgreSQL 实际上是以“角色”为单位进行权限管理的。在本章中，我们统一使用数据库用户来表示“拥有 `LOGIN` 权限的角色”。

认证是数据库服务器确认客户端身份的过程，
并据此决定是否允许该客户端应用（或者运行该客户端应用的用户）
以请求的数据库用户名进行连接。

PostgreSQL 提供了多种不同的客户端认证方法。用于认证特定客户端连接的方法可以根据（客户端）主机地址、数据库和用户来选择。

PostgreSQL 数据库用户名在逻辑上独立于服务器所在操作系统中的用户名。如果某台服务器的所有用户在该机器上也都有账号，
那么为他们分配与操作系统用户名一致的数据库用户名是有意义的。不过，
接受远程连接的服务器可能有许多数据库用户并没有本地操作系统账号，在这种情况下，
数据库用户名与操作系统用户名之间就不必存在任何对应关系。

19.1. pg_hba.conf 文件

客户端认证由一个配置文件控制，该文件按惯例命名为 `pg_hba.conf`，并存放在数据库集簇的数据目录中。（HBA 代表 host-based authentication，即基于主机的认证。）当数据目录由 `initdb` 初始化时，会安装一个默认的 `pg_hba.conf` 文件。不过，也可以把认证配置文件放在别处；请参见配置参数hba_file。

`pg_hba.conf` 文件的一般格式是一组记录，每行一条。空行被忽略，`#` 注释字符之后的任何文本也被忽略。记录不能跨行。一条记录由若干字段组成，字段之间用空格和/
或制表符分隔。如果字段值被引号括起，字段中可以包含空白。把数据库、
用户或地址字段中的某个关键字（例如 `all` 或 `replication`）用引号括起会使该词失去特殊含义，只是匹配同名的数据库、用户或主机。

每条认证记录都指定一种连接类型、一个客户端 IP 地址范围（如果该连接类型需要）、一个数据库名、一个用户名，以及对匹配这些参数的连接要使用的认证方法。
第一条同时匹配连接类型、客户端地址、请求数据库和用户名的记录会被用来执行认证。
这里不存在“继续向后匹配”或“后备”机制：如果选中某条记录而认证失败，
就不会再考虑后续记录。如果没有任何记录匹配，则拒绝访问。

记录可以采用以下七种格式之一

```
local      database  user  auth-method  [auth-options]
host       database  user  address  auth-method  [auth-options]
hostssl    database  user  address  auth-method  [auth-options]
hostnossl  database  user  address  auth-method  [auth-options]
host       database  user  IP-address  IP-mask  auth-method  [auth-
options]
hostssl    database  user  IP-address  IP-mask  auth-method  [auth-
options]
hostnossl  database  user  IP-address  IP-mask  auth-method  [auth-
options]
```

各字段的含义如下：

`local`

该记录匹配使用 Unix 域套接字的连接尝试。没有这种类型的记录时，Unix 域套接字连接将被禁止。

`host`

此记录匹配通过 TCP/IP 发起的连接尝试。`host` 记录既匹配 SSL 连接，也匹配非 SSL 连接。

> **注意**
>
> 除非服务器以适当的listen_addresses配置参数值启动，否则远程 TCP/IP 连接将不可用，因为默认行为是只在本地回环地址 `localhost` 上监听 TCP/IP 连接。

`hostssl`

该记录匹配使用 TCP/IP 发起的连接尝试，但仅当连接使用 SSL 加密时才匹配。

要使用此选项，服务器必须在构建时启用 SSL 支持。此外，还必须在服务器启动时通过设置ssl配置参数来启用 SSL（有关更多信息，请参见第 17.9 节）。

`hostnossl`

该记录类型与 `hostssl` 的行为相反；它只匹配通过 TCP/IP 发起且不使用 SSL 的连接尝试。

`database`

指定这条记录匹配哪些数据库名。`all` 表示匹配所有数据库。`sameuser` 表示如果请求的数据库名和请求的用户名相同，则匹配。`samerole` 表示请求的用户必须是与会话请求的数据库名相同的角色的成员。（`samegroup` 是 `samerole` 的一个已经废弃但仍然被接受的拼写。）对于 `samerole`，只有显式地直接或间接属于该角色，超级用户才被视为其成员，仅凭超级用户身份并不算。The value `replication` specifies that the record matches if a replication connection is requested (note that replication connections do not specify any particular database). Otherwise, this is the name of a specific PostgreSQL database. Multiple database names can be supplied by separating them with commas. A separate file containing database names can be specified by preceding the file name with `@`.

`user`

指定这条记录匹配哪些数据库用户名。`all` 表示匹配所有用户。否则，它要么是一个具体数据库用户的名字，要么是一个前面带有 `+` 的组名。（回想一下，PostgreSQL 中用户和组之间没有真正的区别；`+` 标记的实际含义是“匹配直接或间接属于该角色成员的任何角色”，而不带 `+` 标记的名字只匹配那个具体的角色。）为此，只有显式地直接或间接属于该角色，超级用户才被视为其成员，仅凭超级用户身份并不算。
可以用逗号分隔多个用户名。也可以在文件名前加 `@`，指定一个单独存放用户名的文件。

`address`

指定此记录匹配的客户端机器地址。此字段可以包含主机名、
IP地址范围或下面提到的特殊关键字之一。

IP地址范围使用标准的数字表示法来指定起始地址，然后是斜杠（`/`）
和一个CIDR掩码长度。掩码长度表示客户端IP地址必须匹配的高位比特数。
给定IP地址中右侧的比特应为零。IP地址、`/`和CIDR掩码长度之间不得有任何空白。

以这种方式指定的IPv4地址范围的典型示例包括`172.20.143.89/32`用于单个主机，或`172.20.143.0/24`用于小型网络，或`10.6.0.0/16`用于较大的网络。

IPv6地址范围可能看起来像`::1/128`用于单个主机（在这种情况下是IPv6环回地址）或`fe80::7a31:c1ff:0000:0000/96`用于小型网络。`0.0.0.0/0`代表所有IPv4地址，`::0/0`代表所有IPv6地址。要指定单个主机，请对IPv4使用32的掩码长度，对IPv6使用128。在网络地址中，不要省略尾部的零。

以IPv4格式给出的条目将仅匹配IPv4连接，以IPv6格式给出的条目将仅匹配IPv6连接，即使所代表的地址在IPv4-in-IPv6范围内。请注意，如果系统的C库不支持IPv6地址，以IPv6格式给出的条目将被拒绝。

你也可以写`all`来匹配任何IP地址，`samehost`来匹配服务器自己的任何IP地址，或`samenet`来匹配服务器直接连接到的任何子网中的任何地址。

如果指定了主机名（任何不是IP地址范围或特殊关键字的内容都被视为主机名），则将该名称与客户端IP地址的反向名称解析结果进行比较（例如，如果使用DNS，则进行反向DNS查找）。主机名比较不区分大小写。如果匹配成功，则对主机名执行正向名称解析（例如，进行正向DNS查找），以检查其解析为的任何地址是否等于客户端IP地址。如果两个方向都匹配，则将条目视为匹配。（在`pg_hba.conf`中使用的主机名应该是客户端IP地址的地址到名称解析返回的名称，否则该行将不会匹配。一些主机名数据库允许将IP地址与多个主机名关联，但操作系统在要求解析IP地址时只会返回一个主机名。）

以点（`.`）开头的主机名规范匹配实际主机名的后缀。因此，`.example.com`将匹配`foo.example.com`（但不匹配单独的`example.com`）。

当在`pg_hba.conf`中指定主机名时，应确保名称解析相对快速。设置一个本地名称解析缓存可能会有帮助，如`nscd`。此外，还可能希望启用配置参数`log_hostname`，以便在日志中看到客户端的主机名而不是 IP 地址。

> 偶尔会有用户想知道，为什么要用这种看起来复杂的方式处理主机名——做两次名字解析并要求对 IP 地址做反向解析，而反向解析有时没有设置或者指向某个不合适的主机名。这主要是为了效率：一次连接尝试需要对当前客户端的地址做两次解析器查找。如果该地址存在解析器问题，那就只是那个客户端自己的问题。而假想的只做正向查找的替代实现，将不得不在每次连接尝试时解析 `pg_hba.conf` 中提到的每一个主机名。这本身就已经很慢了。而且如果其中某个主机名存在解析器问题，它就会成为所有人的问题。
>
> 此外，要实现后缀匹配特性也必须做反向解析，因为必须知道客户端的实际主机名才能将它与模式匹配。
>
> 请注意，这种行为与其他流行的基于主机名的访问控制实现一致，例如 Apache HTTP 服务器和 TCP Wrappers。

这个字段只适用于 `host`、`hostssl` 和 `hostnossl` 记录。

IP-address
IP-mask

这两个字段可以用作*IP-address/mask-length* 表示法的替代方案。不指定掩码长度，而是在单独一列中指定实际掩码。例如，`255.0.0.0`表示IPv4的CIDR掩码长度为8，而`255.255.255.255`表示CIDR掩码长度为32。

这些字段仅适用于 `host`、`hostssl` 和 `hostnossl` 记录。

auth-method

指定连接匹配此记录时要使用的认证方法。可选值在此处做了概述；详细说明见第 19.3 节。

`trust`

无条件允许连接。
这种方法允许任何能够连接到PostgreSQL数据库服务器的人以任意他们希望的PostgreSQL用户身份登录，无需密码或任何其他认证。详见第 19.3.1 节。

`reject`

无条件拒绝连接。这对于“过滤掉”某些主机很有用，例如一个`reject`行可以阻止特定主机连接，而后面的行允许特定网络中的其余主机连接。

`md5`

要求客户端提供一个经过双重 MD5 哈希的密码以完成认证。详见第 19.3.2 节。

`password`

要求客户端提供未加密的密码以完成认证。由于密码会以明文形式通过网络发送，因此不应在不受信任的网络上使用。详见第 19.3.2 节。

`gss`

使用 GSSAPI 认证用户。这仅适用于 TCP/IP 连接。详情参见第 19.3.3 节。

`sspi`

使用 SSPI 对用户进行认证。这仅适用于 Windows。详见第 19.3.4 节。

`krb5`

使用 Kerberos V5 对用户进行认证。这只适用于 TCP/IP 连接。详见第 19.3.5 节。

`ident`

通过联系客户端上的 ident 服务器获取客户端的操作系统用户名，并检查它是否与请求的数据库用户名匹配。Ident 认证只能用于 TCP/IP 连接。当为本地连接指定时，将改为使用 peer 认证。详见第 19.3.6 节。

`peer`

从操作系统获取客户端的操作系统用户名，并检查是否与请求的数据库用户名匹配。这仅适用于本地连接。有关详细信息，请参见第 19.3.7 节。

`ldap`

使用LDAP服务器进行认证。详见第 19.3.8 节。

`radius`

使用 RADIUS 服务器进行认证。详见第 19.3.9 节。

`cert`

使用 SSL 客户端证书进行认证。详见第 19.3.10 节。

`pam`

使用操作系统提供的可插拔认证模块（PAM）服务进行认证。详见第 19.3.11 节。

`auth-options`

在 *`auth-method`* 字段之后，可以有一个或多个形如 *`name=value`* 的字段，用于指定认证方法的选项。下文会详细说明各认证方法有哪些可用选项。

由 @ 构造引用的文件会被读取为名称列表，其中的名称可以用空白或逗号分隔。注释仍然用 # 引入，与 pg_hba.conf 中相同，并且允许嵌套的 @ 构造。除非 @ 后面的文件名是绝对路径，否则它会被视为相对于引用它的文件所在目录。

由于每次连接尝试都会按顺序检查 pg_hba.conf 记录，因此记录的顺序很重要。通常，靠前的记录使用较严格的连接匹配条件和较弱的认证方法，
靠后的记录使用较宽松的匹配条件和较强的认证方法。例如，可能希望对本地 TCP/IP 连接使用 trust 认证，而要求远程 TCP/IP 连接提供密码。此时，为来自 127.0.0.1 的连接指定 trust 认证的记录，应放在为更大范围的允许客户端 IP 地址指定密码认证的记录之前。

在启动时以及主服务器进程收到 SIGHUP 信号时，pg_hba.conf 文件会被读取。如果你在运行中的系统上编辑了该文件，就需要通知 postmaster（使用 pg_ctl reload 或 kill -HUP）重新读取该文件。

> **提示**
>
> 要连接到一个特定数据库，一个用户必须不仅要通过pg_hba.conf检查，
> 还必须要有该数据库上的CONNECT权限。
> 如果你希望限制哪些用户能够连接到哪些数据库，授予/撤销CONNECT
> 权限通常比在pg_hba.conf项中设置规则简单。

例 19.1展示了一些 pg_hba.conf 条目示例。有关不同认证方法的详细信息，参见下一节。

例 19.1. 示例 pg_hba.conf 项

```
# Allow any user on the local system to connect to any database
 with
# any database user name using Unix-domain sockets (the default for
 local
# connections).
#
# TYPE  DATABASE        USER            ADDRESS
 METHOD
local   all             all
 trust

# The same using local loopback TCP/IP connections.
#
# TYPE  DATABASE        USER            ADDRESS
 METHOD
host    all             all             127.0.0.1/32
 trust

# The same as the previous line, but using a separate netmask
 column
#
# TYPE  DATABASE        USER            IP-ADDRESS      IP-MASK
         METHOD
host    all             all             127.0.0.1       255.255.
255.255     trust

# The same over IPv6.
#
# TYPE  DATABASE        USER            ADDRESS
 METHOD
host    all             all             ::1/128
 trust
```

```
# The same using a host name (would typically cover both IPv4 and
 IPv6).
#
# TYPE  DATABASE        USER            ADDRESS
 METHOD
host    all             all             localhost
 trust

# Allow any user from any host with IP address 192.168.93.x to
 connect
# to database "postgres" as the same user name that ident reports
 for
# the connection (typically the operating system user name).
#
# TYPE  DATABASE        USER            ADDRESS
 METHOD
host    postgres        all             192.168.93.0/24
 ident

# Allow any user from host 192.168.12.10 to connect to database
# "postgres" if the user's password is correctly supplied.
#
# TYPE  DATABASE        USER            ADDRESS
 METHOD
host    postgres        all             192.168.12.10/32        md5

# Allow any user from hosts in the example.com domain to connect to
# any database if the user's password is correctly supplied.
#
# TYPE  DATABASE        USER            ADDRESS
 METHOD
host    all             all             .example.com            md5

# In the absence of preceding "host" lines, these two lines will
# reject all connections from 192.168.54.1 (since that entry will
 be
# matched first), but allow Kerberos 5 connections from anywhere
 else
# on the Internet.  The zero mask causes no bits of the host IP
# address to be considered, so it matches any host.
#
# TYPE  DATABASE        USER            ADDRESS
 METHOD
host    all             all             192.168.54.1/32
 reject
host    all             all             0.0.0.0/0
 krb5

# Allow users from 192.168.x.x hosts to connect to any database, if
# they pass the ident check.  If, for example, ident says the user
 is
# "bryanh" and he requests to connect as PostgreSQL user "guest1",
 the
# connection is allowed if there is an entry in pg_ident.conf for
 map
# "omicron" that says "bryanh" is allowed to connect as "guest1".
#
```

```
# TYPE  DATABASE        USER            ADDRESS
 METHOD
host    all             all             192.168.0.0/16
 ident map=omicron

# If these are the only three lines for local connections, they
 will
# allow local users to connect only to their own databases (
databases
# with the same name as their database user name) except for
 administrators
# and members of role "support", who can connect to all databases.
 The file
# $PGDATA/admins contains a list of names of administrators.
 Passwords
# are required in all cases.
#
# TYPE  DATABASE        USER            ADDRESS
 METHOD
local   sameuser        all                                     md5
local   all             @admins                                 md5
local   all             +support                                md5

# The last two lines above can be combined into a single line:
local   all             @admins,+support                        md5

# The database column can also use lists and file names:
local   db1,db2,@demodbs  all                                   md5
```

19.2. 用户名映射

当使用 Ident 或 GSSAPI 之类的外部认证系统时，发起连接的操作系统用户名可能不同于要使用的数据库用户。在这种情况下，
可以通过用户名映射把操作系统用户名映射为数据库用户。要使用用户名映射，需要在 `pg_hba.conf` 的选项字段中指定 `map=`*`map-name`*。此选项适用于所有会接收外部用户名的认证方法。由于不同连接可能需要不同映射，在 `pg_hba.conf` 中通过 *`map-name`* 参数指定要使用的映射，以表明每个连接应使用哪个映射。

用户名映射在 ident 映射文件中定义。该文件默认名为 `pg_ident.conf`，存储在集簇的数据目录中。（也可以将映射文件放在其他位置；参见ident_file配置参数。）ident 映射文件中各行的基本格式如下：

`map-name system-username database-username`

注释和空白的处理方式与 `pg_hba.conf` 相同。*`map-name`* 是任意名称，用于在 `pg_hba.conf` 中引用此映射。另两个字段指定操作系统用户名及与之匹配的数据库用户名。同一个 *`map-name`* 可以重复使用，以便在一项映射中指定多个用户对应关系。

一个操作系统用户可以对应多少个数据库用户，没有限制，反之亦然。因此，
映射中的条目应理解为“此操作系统用户可以作为此数据库用户连接”，而不是说二者等同。
只要存在一个映射条目，
将从外部认证系统取得的用户名与用户请求连接时使用的数据库用户名配对，
就会允许该连接。

如果 *`system-username`* 字段以斜杠（`/`）开头，该字段的其余部分就会被视为正则表达式。（参见第 9.7.3.1 节，其中介绍了 PostgreSQL 的正则表达式语法。）正则表达式可以包含一个捕获，即用圆括号括起来的子表达式，随后可以在 *`database-username`* 字段中使

用 \1（反斜杠加数字一）来引用。这样就可以在一行中映射多个用户名，对简单的语法替换尤其有用。例如，以下条目

```
mymap   /^(.*)@mydomain\.com$      \1
mymap   /^(.*)@otherdomain\.com$   guest
```

会去掉系统用户名中的域名部分（如果用户名以 @mydomain.com 结尾），并允许系统用户名以 @otherdomain.com 结尾的任何用户登录为 guest。

> **提示**
>
> 记住在默认情况下，一个正则表达式可以只匹配字符串的一部分。如上例所示，使用^和$来强制匹配整个系统用户名通常是明智的。

在启动时以及主服务器进程收到 SIGHUP 信号时，pg_ident.conf 文件会被读取。如果你在运行中的系统上编辑了该文件，就需要通知 postmaster（使用 pg_ctl reload 或 kill -HUP）重新读取该文件。

例 19.2展示了一个可与例 19.1中的 pg_hba.conf 文件配合使用的 pg_ident.conf 文件。在这个示例中，任何登录到 192.168 网络中某台机器上的用户，如果其操作系统用户名不是 bryanh、ann 或 robert，都不会被授予访问权限。Unix 用户 robert 只有在尝试以 PostgreSQL 用户 bob 身份连接时才被允许访问，而不能以 robert 或其他身份连接。ann 只能以 ann 身份连接。用户 bryanh 则可以以 bryanh 或 guest1 身份连接。

例 19.2. 示例 pg_ident.conf 文件

```
# MAPNAME       SYSTEM-USERNAME         PG-USERNAME

omicron         bryanh                  bryanh
omicron         ann                     ann
# bob has user name robert on these machines
omicron         robert                  bob
# bryanh can also connect as guest1
omicron         bryanh                  guest1
```

19.3. 认证方法

下面几节会更详细地介绍这些认证方法。

19.3.1. 信任认证

当trust认证被指定时，
PostgreSQL假设任何可以连接到服务器的人都被授权使用他们指定的任何数据库用户名（即使是超级用户）访问数据库。当然，在database和 user列中设置的限制仍然适用。只有当在操作系统层对进入服务器的连接有足够保护时，才应该使用这种方法。

trust 认证对单用户工作站上的本地连接来说是合适且非常方便的。在多用户机器上，通常单用它并不合适。不过，如果用文件系统权限限制对服务器 Unix 域套接字文件的访问，即使在多用户机器上也可以使用 trust。为此，可按第 18.3 节中的说明设置 unix_socket_permissions（可能还有 unix_socket_group）配置参数。或者，也可以设置 unix_socket_directory 配置参数，把套接字文件放在一个经过适当限制的目录中。

设置文件系统权限只能有助于 Unix 套接字连接。本地 TCP/IP 连接不会被文件系统权限限制。因此，如果你想利用文件系统权限来控制本地安全，那么从pg_hba.conf中移除host ... 127.0.0.1 ...行，或者把它改为一个非trust认证方法。

只有当你信任由 pg_hba.conf 中指定 trust 的行所允许连接的每台机器上的每个用户时，trust 认证才适合用于 TCP/IP 连接。对来自 localhost（127.0.0.1）以外的任何 TCP/IP 连接使用 trust，通常都不合理。

19.3.2. 密码认证

基于密码的认证方法有 md5 和 password。除了密码通过连接发送的方式——前者以 MD5 哈希发送、后者以明文发送——之外，这两种方法的操作是相似的。

如果你担心密码“嗅探”攻击，那么应优先使用 md5。如果可能，应始终避免使用明文 password。不过，md5 不能与db_user_namespace功能一起使用。如果连接被 SSL 加密保护着，那么可以安全地使用 password（不过如果依靠 SSL，SSL 证书认证可能是更好的选择）。

PostgreSQL 数据库密码独立于操作系统用户密码。每个数据库用户的密码存储在 pg_authid 系统目录中。可以使用 SQL 命令CREATE USER和ALTER ROLE管理密码，例如 **CREATE USER foo WITH PASSWORD 'secret'**，如果未为某个用户设置密码，存储的密码就是空值，该用户的密码认证始终会失败。

19.3.3. GSSAPI 认证

GSSAPI 是 RFC 2743 定义的安全认证行业标准协议。PostgreSQL 按照 RFC 1964 支持使用 Kerberos 认证的 GSSAPI。GSSAPI 为支持它的系统提供自动认证（单点登录）。认证过程本身是安全的，但除非使用 SSL，否则数据库连接上传输的数据不会加密。

当 GSSAPI 使用 Kerberos 时，它使用标准格式 *servicename/hostname@realm* 的 principal。有关 principal 的组成部分以及如何设置所需密钥的信息，见第 19.3.5 节。

当编译PostgreSQL时，GSSAPI 支持必须被启用，详见第 15 章。

以下配置选项适用于 GSSAPI：

include_realm

如果设为 1，已认证用户主体中的 realm 名称将包含在传递给用户名映射（第 19.2 节）的系统用户名中。这是推荐的配置，因为否则将无法区分来自不同 realm 的同名用户。该参数的默认值为 0（即系统用户名中不包含 realm），但在未来版本的 PostgreSQL 中可能会改为 1。用户可以显式设置它，以避免升级时出现任何问题。

map

允许在系统用户名与数据库用户名之间建立映射。详见第 19.2 节。对于 username@EXAMPLE.COM（或较少见的 username/hostbased@EXAMPLE.COM）这样的 GSSAPI/Kerberos 主体，映射默认使用的用户名是 username（或相应的 username/hostbased）；但如果按上文建议将 include_realm 设为了 1，那么映射时视为系统用户名的就是 username@EXAMPLE.COM（或 username/hostbased@EXAMPLE.COM）。

krb_realm

设置用于匹配用户主体名的 realm。如果设置了该参数，则只接受来自该 realm 的用户；如果未设置，则允许来自任意 realm 的用户连接，但仍受已执行的用户名映射约束。

19.3.4. SSPI 认证

SSPI 是一种提供安全认证和单点登录的 Windows 技术。PostgreSQL 会以 negotiate 模式使用 SSPI，尽可能使用 Kerberos，否则自动回退到 NTLM。只有服务器和客户端都运行 Windows，或者在非 Windows 平台上可用 GSSAPI 时，SSPI 认证才能工作。

当使用Kerberos认证时，SSPI和GSSAPI的工作方式相同，详见第 19.3.3 节。

SSPI 支持下列配置选项：

`include_realm`

如果设为 1，已认证用户主体中的 realm 名称将包含在传递给用户名映射（第 19.2 节）的系统用户名中。这是推荐的配置，因为否则将无法区分来自不同 realm 的同名用户。该参数的默认值为 0（即系统用户名中不包含 realm），但在未来版本的 PostgreSQL 中可能会改为 1。用户可以显式设置它，以避免升级时出现任何问题。

`map`

允许在系统用户名与数据库用户名之间建立映射。详见第 19.2 节。对于 SSPI/Kerberos 主体，例如 `username@EXAMPLE.COM`（或者较少见的 `username/hostbased@EXAMPLE.COM`），映射默认使用的用户名是 `username`（或相应的 `username/hostbased`）；但如果按上文建议将 `include_realm` 设为了 1，那么映射时视为系统用户名的就是 `username@EXAMPLE.COM`（或 `username/hostbased@EXAMPLE.COM`）。

`krb_realm`

设置用于匹配用户主体名的 realm。如果设置了该参数，则只接受来自该 realm 的用户；如果未设置，则允许来自任意 realm 的用户连接，但仍受已执行的用户名映射约束。

19.3.5. Kerberos 认证

注意

原生 Kerberos 认证已被弃用，只应用于向后兼容。鼓励新建和升级的安装改用行业标准 GSSAPI 认证方法（见第 19.3.3 节）。

Kerberos 是一种适合在公共网络上进行分布式计算的行业标准安全认证系统。对 Kerberos 系统的描述超出了本文档的范围；在最一般的意义上它可能相当复杂（但也很强大）。Kerberos FAQ[1] 或 MIT Kerberos 页面[2]可以作为很好的探索起点。存在若干 Kerberos 发行版来源。Kerberos 提供安全认证，但不对网络上传递的查询或数据加密；如需加密请使用 SSL。

PostgreSQL 支持 Kerberos 版本 5。构建 PostgreSQL 时必须启用 Kerberos 支持；更多信息见第 15 章。

PostgreSQL 像一个普通的 Kerberos 服务那样运作。服务主体的名称为 *servicename*/*hostname*@*realm*。

servicename can be set on the server side using the krb_srvname configuration parameter, and on the client side using the `krbsrvname` connection parameter. (See also 第 31.1 节.) 安装时的默认值可以在构建时用 `./configure --with-krb-srvnam=`*whatever* 从默认的 `postgres` 改变。在大多数环境中，这个参数永远不需要更改。但是，在同一台主机上支持多个 PostgreSQL 安装时需要更改它。Some Kerberos implementations might also require a different service name, such as Microsoft Active Directory which requires the service name to be in upper case (`POSTGRES`).

hostname 是服务器机器的完全限定主机名。服务主体的 realm 是服务器机器的首选 realm。

[1] http://www.cmf.nrl.navy.mil/CCS/people/kenh/kerberos-faq.html
[2] http://web.mit.edu/kerberos/www/

客户端主体的第一个组成部分必须是 PostgreSQL 数据库用户名，例如 `pgusername@realm`。或者，也可以使用用户名映射，将主体名的第一个组成部分映射为数据库用户名。默认情况下，PostgreSQL 不检查客户端的 realm。如果启用了跨 realm 认证并且需要验证 realm，请使用 `krb_realm` 参数，或者启用 `include_realm` 并使用用户名映射来检查 realm。

确保你的服务器密钥表文件可被 PostgreSQL 服务器账户读取（最好只有该账户可读）。（另见第 17.1 节。）密钥文件的位置由krb_server_keyfile配置参数指定。默认为 `/usr/local/pgsql/etc/krb5.keytab`（或构建时指定为 `sysconfdir` 的目录）。

keytab 文件由 Kerberos 软件生成；详情参见 Kerberos 文档。以下示例适用于兼容 MIT 的 Kerberos 5 实现：

```
kadmin% ank -randkey postgres/server.my.domain.org
kadmin% ktadd -k krb5.keytab postgres/server.my.domain.org
```

连接数据库时，请确保持有与请求的数据库用户名相匹配的主体票据。例如，数据库用户名为 `fred` 时，主体 `fred@EXAMPLE.COM` 可以连接。如果还要允许主体 `fred/users.example.com@EXAMPLE.COM`，请按第 19.2 节所述使用用户名映射。

如果你在 Apache Web 服务器上使用 mod_auth_kerb[3]和 mod_perl，可以在 mod_perl 脚本中使用 `AuthType KerberosV5SaveCredentials`。这可提供经过 Web 的安全数据库访问，而无需额外的口令。

Kerberos 支持以下配置选项：

`map`

允许在系统用户名和数据库用户名之间进行映射。详见第 19.2 节。

`include_realm`

如果设置为 1，已认证用户主体中的领域名将包含在传递给用户名映射（第 19.2 节）的系统用户名中。这有助于处理来自多个领域的用户。

`krb_realm`

设置用于匹配用户主体名的 realm。如果设置了该参数，则只接受来自该 realm 的用户；如果未设置，则允许来自任意 realm 的用户连接，但仍受已执行的用户名映射约束。

`krb_server_hostname`

设置服务主体的主机名部分。它与 `krb_srvname` 组合生成完整的服务主体，即 `krb_srvname/krb_server_hostname@`REALM。如果未设置，默认为服务器主机名。

19.3.6. Ident 认证

ident 认证方法通过从一个 ident 服务器获得客户端的操作系统用户名并且用它作为被允许的数据库用户名（和可选的用户名映射）来工作。它只在 TCP/IP 连接上支持。

> **注意**
>
> 当为一个本地（非 TCP/IP）连接指定 ident 时，将实际使用 peer 认证（见第 19.3.7 节）。

以下配置选项适用于 ident：

[3] http://modauthkerb.sf.net

`map`

允许在系统用户名和数据库用户名之间进行映射。详见第 19.2 节。

“标识协议”在 RFC 1413 中定义。几乎所有类 Unix 操作系统都自带 ident 服务器，默认监听 TCP 端口 113。ident 服务器的基本功能是回答这样的问题：“从你的端口 *X* 连到我的端口 *Y* 的连接，是哪个用户发起的？”由于建立物理连接时，PostgreSQL 已知 *X* 和 *Y*，因此可以询问连接客户端所在主机上的 ident 服务器，理论上能够确定任意给定连接的操作系统用户。

这个过程的缺点在于它依赖客户端本身的可信性：如果客户端机器不可信或者已被攻破，攻击者几乎可以在 113 端口上运行任何程序，并返回任意他选择的用户名。因此，这种认证方法只适用于封闭网络，在这类网络中每台客户端机器都受到严格控制，而且数据库管理员与系统管理员之间保持密切协作。换句话说，你必须信任运行 ident 服务器的那台机器。请注意下面的警告：

> 标识协议的本意不是作为一种授权或访问控制协议。
>
> —RFC 1413

有些 ident 服务器提供了一个非标准选项，会让返回的用户名被加密，而解密所需密钥只有发起连接机器的管理员才知道。将 ident 服务器与 PostgreSQL 配合使用时，绝不能启用这个选项，因为 PostgreSQL 无法解密返回的字符串，也就无法确定实际用户名。

19.3.7. Peer 认证

Peer 认证通过从内核获取客户端的操作系统用户名，并把它用作被允许的数据库用户名（可结合可选的用户名映射）来工作。这种方法只支持本地连接。

以下配置选项适用于 peer：

`map`

允许在系统用户名和数据库用户名之间进行映射。详见第 19.2 节。

Peer 认证只在提供 `getpeereid()` 函数、`SO_PEERCRED` 套接字参数或类似机制的操作系统上可用。目前这包括 Linux、大多数 BSD 变种（包括 OS X）以及 Solaris。

19.3.8. LDAP 认证

这种认证方法的工作方式与 `password` 类似，只不过它使用 LDAP 作为密码验证方法。LDAP 只用于验证用户名/密码对。因此，在使用 LDAP 进行认证之前，用户必须已经存在于数据库中。

LDAP 认证可以以两种模式运作。在第一种模式中，服务器将绑定到由 *`prefix username suffix`* 构造的可分辨名称上。通常，*`prefix`* 参数用于指定 `cn=`，或者在 Active Directory 环境中指定 *`DOMAIN`*`\`。*`suffix`* 用于在非 Active Directory 环境中指定 DN 的剩余部分。

在第二种模式中，服务器首先用一个固定的用户名和密码绑定到 LDAP 目录（由 *`ldapbinddn`* 和 *`ldapbindpasswd`* 指定），然后为尝试登录数据库的用户执行搜索。如果没有配置用户和密码，将尝试对目录做匿名绑定。搜索将在 *`ldapbasedn`* 的子树上进行，并尝试对 *`ldapsearchattribute`* 中指定的属性做精确匹配。如果没有指定属性，将使用 `uid` 属性。Once the user has been found in this search, the server disconnects and re-binds to the directory as this user, using the password specified by the client, to verify that the login is correct. This method allows for significantly more flexibility in where the user objects are located in the directory, but will cause two separate connections to the LDAP server to be made.

The following configuration options are supported for LDAP:

`ldapserver`

要连接的 LDAP 服务器的名字或 IP 地址。可以指定多个服务器，用空格分隔。

`ldapport`

要连接的LDAP服务器的端口号。如果未指定端口，则将使用LDAP库的默认端口设置。

`ldaptls`

设为 1 时，PostgreSQL 与 LDAP 服务器之间的连接会使用 TLS 加密。注意，这只加密与 LDAP 服务器之间的流量 – 除非使用 SSL，否则与客户端之间的连接仍不加密。

`ldapprefix`

在进行简单绑定认证时，附加到用户名前面以形成绑定 DN 的字符串。

`ldapsuffix`

在进行简单绑定认证时，附加到用户名后面以形成绑定 DN 的字符串。

`ldapbasedn`

在进行搜索+绑定认证时，用作用户搜索起点的根 DN。

`ldapbinddn`

在进行搜索+绑定认证时，用于绑定到目录并执行搜索的用户 DN。

`ldapbindpasswd`

在进行搜索+绑定认证时，用于绑定到目录并执行搜索的用户密码。

`ldapsearchattribute`

做 search+bind 认证时，在搜索中与用户名匹配的属性。

注意

由于 LDAP 通常使用逗号和空格来分隔 DN 的不同部分，配置 LDAP 选项时常常需要使用双引号括起的参数值，例如：

```
ldapserver=ldap.example.net ldapprefix="cn=" ldapsuffix=", dc=
example, dc=net"
```

19.3.9. RADIUS 认证

这种认证方法的工作方式与 `password` 类似，只不过它使用 RADIUS 作为密码验证方式。RADIUS 只用于验证用户名/密码对。因此，在使用 RADIUS 进行认证之前，用户必须已经存在于数据库中。

使用 RADIUS 认证时，会向配置好的 RADIUS 服务器发送一条 Access Request 消息。该请求的类型为 `Authenticate Only`，并包含 `user name`、`password`（加密的）以及 `NAS Identifier` 参数。该请求会使用与服务器共享的密钥进行加密。这台服务器会返回 `Access Accept` 或 `Access Reject` 作为响应。PostgreSQL 不支持 RADIUS 记账。

RADIUS 支持下列配置选项：

`radiusserver`

要连接的 RADIUS 服务器的名称或 IP 地址。此参数为必需项。

`radiussecret`

与 RADIUS 服务器进行安全通信时使用的共享密钥。它在 PostgreSQL 服务器和 RADIUS 服务器上必须完全相同。建议它至少是一个 16 个字符长的字符串。此参数为必需项。

> **注意**
>
> 只有当 PostgreSQL 在构建时启用了 OpenSSL 支持，所使用的加密向量才具有足够的密码学强度。在其他情况下，到 RADIUS 服务器的传输只能被视为经过混淆，而非受到安全保护；如有必要，应额外采取外部安全措施。

`radiusport`

要连接的 RADIUS 服务器端口号。如果未指定端口，则会使用默认端口 `1812`。

`radiusidentifier`

在 RADIUS 请求中用作 `NAS Identifier` 的字符串。这个参数可用作第二个参数，例如标识用户正尝试以哪个数据库用户进行认证，从而便于在 RADIUS 服务器上进行策略匹配。如果未指定标识符，则默认使用 `postgresql`。

19.3.10. 证书认证

这种认证方法使用 SSL 客户端证书进行认证，因此仅适用于 SSL 连接。使用此方法时，服务器要求客户端提供有效的证书，不会向客户端发送密码提示。服务器会将证书的 `cn`（通用名称）属性与请求的数据库用户名比较，匹配时才允许登录。可以使用用户名映射，允许 `cn` 与数据库用户名不同。

SSL 证书认证支持下列配置选项：

`map`

允许在系统用户名和数据库用户名之间进行映射。详见第 19.2 节。

19.3.11. PAM 认证

这种认证方法的工作方式与 `password` 类似，只是它使用 PAM（可插拔认证模块）作为认证机制。默认的 PAM 服务名是 `postgresql`。PAM 只用于验证用户名/密码对。因此，在能够使用 PAM 认证之前，用户必须已经存在于数据库中。For more information about PAM, please read the Linux-PAM Page[4] and the Solaris PAM Page[5].

PAM 支持下列配置选项：

`pamservice`

PAM 服务名称。

> **注意**
>
> 如果 PAM 被设置为读取 `/etc/shadow`，认证将会失败，因为 PostgreSQL 服务器是由非 root 用户启动的。不过，当 PAM 被配置为使用 LDAP 或其他认证方法时，这就不是问题。

[4] http://www.kernel.org/pub/linux/libs/pam/
[5] http://www.sun.com/software/solaris/pam/

19.4. 认证问题

认证失败及相关问题通常会表现为类似下面这样的错误消息：

```
FATAL:  no pg_hba.conf entry for host "123.123.123.123", user
 "andym", database "testdb"
```

这种情况通常意味着你已经成功联系到了服务器，但服务器不愿意接受你的连接。
正如消息所示，服务器拒绝了该连接请求，因为它没有在自己的 `pg_hba.conf` 配置文件中找到匹配项。

```
FATAL:  password authentication failed for user "andym"
```

这样的消息表示你已经联系到了服务器，而且服务器也愿意继续处理连接，
但前提是你必须先通过 `pg_hba.conf` 文件中指定的认证方法。请检查你提供的密码；如果错误消息提到了 Kerberos 或 ident 等认证类型，也请检查对应的软件配置。

```
FATAL:  user "andym" does not exist
```

指定的数据库用户不存在。

```
FATAL:  database "testdb" does not exist
```

你试图连接的数据库不存在。注意，如果你没有指定数据库名，
默认会使用数据库用户名作为数据库名，但这不一定是所需的数据库名。

> **提示**
>
> 服务器日志中可能包含比返回给客户端的更多认证失败信息。
> 如果你不清楚失败原因，请检查服务器日志。

第 20 章 数据库角色

PostgreSQL使用角色这一概念来管理数据库访问权限。角色可以被看作数据库用户，也可以被看作一组数据库用户，这取决于该角色的设置方式。
角色可以拥有数据库对象（例如表和函数），并且可以把这些对象上的权限赋予其他角色，以控制谁能访问哪些对象。此外，还可以把一个角色中的成员资格授予另一个角色，从而允许成员角色使用授予该角色的权限。

角色这一概念涵盖了“用户”和“组”这两个概念。在 PostgreSQL 8.1 之前的版本中，用户和组是两类不同的实体，但现在只有角色。任意角色都可以充当用户、组，或者同时充当两者。

本章描述如何创建和管理角色。有关角色权限对各种数据库对象影响的更多信息，见第 5.6 节。

20.1. 数据库角色

数据库角色在概念上与操作系统用户完全分离。在实践中，维持两者之间的对应关系可能比较方便，但这并非必需。
数据库角色在整个数据库集簇安装中是全局的（而非每个数据库各自独立）。要创建角色，请使用CREATE ROLE SQL 命令：

```
CREATE ROLE name;
```

name 遵循 SQL 标识符规则：要么不加修饰且不含特殊字符，要么用双引号括起。（在实践中，通常还需要为该命令添加其他选项，例如 `LOGIN`。下文会给出更多细节。）要移除现有角色，请使用类似的DROP ROLE命令：

```
DROP ROLE name;
```

为了方便，提供了createuser和dropuser程序，作为这些 SQL 命令的包装器，可以从 shell 命令行调用：

```
createuser name
dropuser name
```

要确定现有角色的集合，请查看 `pg_roles` 系统目录，例如

```
SELECT rolname FROM pg_roles;
```

使用psql程序的 `\du` 元命令也可以列出现有角色。

为了完成数据库系统的初始引导，一个刚初始化的系统总会包含一个预定义角色。该角色总是一个“超级用户”，并且除非另行指定了不同名称，否则它的名称与用`initdb`初始化数据库集簇的操作系统用户相同。这个角色通常命名为`postgres`。要创建更多角色，必须先以这个初始角色建立连接。

每个到数据库服务器的连接都是以某个特定角色名建立的，而这个角色决定在该连接中发出命令时的初始访问权限。某个数据库连接要使用的角色名，由发起连接请求的客户端以应用特定的方式指明。例如，`psql`程序使用`-U`命令行选项来指明要以哪个角色连接。
很多应用默认假定使用当前操作系统用户的名字（包括`createuser`和`psql`）。因此，在角色名和操作系统用户名之间维护对应关系通常很方便。

某个客户端连接能够以哪些数据库角色进行连接，由客户端认证设置决定，如第 19 章中所述。（因此，客户端并不限于只能以与其操作系统用户匹配的角色连接，

正如一个人的登录名不必与其真实姓名一致。）
由于角色身份决定了已连接客户端可用的权限集合，所以在设置多用户环境时，
仔细配置权限非常重要。

20.2. 角色属性

数据库角色可以具有多个属性，这些属性定义其权限，并与客户端认证系统交互。

登录权限

只有具有 LOGIN 属性的角色才能用作数据库连接的初始角色名。具有 LOGIN 属性的角色可以视为“数据库用户”。要创建具有登录权限的角色，可以使用以下任一命令：

```
CREATE ROLE name LOGIN;
CREATE USER name;
```

（CREATE USER 等价于 CREATE ROLE，区别在于 CREATE USER 默认假定具有 LOGIN，而 CREATE ROLE 不会如此。）

超级用户状态

数据库超级用户会绕过除登录权之外的所有权限检查。这是一种危险的权限，
不应草率使用；最好以非超级用户角色完成大部分工作。要创建新的数据库超级用户，
使用CREATE ROLE *name* SUPERUSER。这必须由已经是超级用户的角色来执行。

创建数据库

角色必须被明确授予创建数据库的权限（超级用户除外，
因为后者会绕过所有权限检查）。要创建这样的角色，使用CREATE ROLE *name* CREATEDB。

创建角色

必须显式授予角色创建更多角色的权限（超级用户除外，
因为它们会绕过所有权限检查）。要创建这样的角色，请使用CREATE ROLE *name* CREATEROLE。具有CREATEROLE权限的角色还可以修改和删除其他角色，以及授予或撤销这些角色的成员资格。不过，要创建、修改、删除超级用户角色或更改其成员资格，
必须具有超级用户身份；仅有CREATEROLE不足以执行这些操作。

发起复制

角色必须被明确授予发起流复制的权限（超级用户除外，
因为后者会绕过所有权限检查）。用于流复制的角色还必须具有LOGIN权限。
要创建这样的角色，使用CREATE ROLE *name* REPLICATION LOGIN。

密码

只有在客户端认证方法要求用户连接数据库时提供密码时，密码才有意义。
password和md5认证方法都会使用密码。数据库密码与操作系统密码是分离的。
可在创建角色时通过CREATE ROLE *name* PASSWORD '*string*'指定密码。

角色创建后，可以使用以下命令修改其属性：ALTER ROLE。详情请参见CREATE ROLE和ALTER ROLE命令的参考页。

提示

一种良好做法是创建具有CREATEDB和CREATEROLE权限但不是超级用户的角色，
然后使用此角色执行数据库和角色的所有日常管理工作。

> 这种方法可以避免在实际上并不需要超级用户权限的任务中以超级用户身份操作所带来的风险。

角色还可以为第 18 章中描述的许多运行时配置设置指定角色特定默认值。例如，如果出于某种原因你希望每次连接时都禁用索引扫描（提示：这不是个好主意），你可以使用：

```
ALTER ROLE myname SET enable_indexscan TO off;
```

这会保存该设置（但不会立即生效）。在该角色后续建立的连接中，它看起来就像在会话开始之前执行了`SET enable_indexscan TO off`一样。你仍然可以在会话期间更改该设置；它只会作为默认值。要移除角色特定默认设置，使用`ALTER ROLE rolename RESET varname`。注意，附加到没有`LOGIN`权限的角色上的角色特定默认值几乎没有用，因为它们永远不会生效。

20.3. 角色成员资格

为了简化权限管理，把用户分组通常很方便：这样，权限就可以整体授予给一个组，或者从整个组中撤销。在PostgreSQL中，这通过创建一个表示该组的角色，然后把该组角色中的成员资格授予各个用户角色来实现。

要建立一个组角色，首先创建该角色：

```
CREATE ROLE name;
```

通常，作为组使用的角色不会有`LOGIN`属性，不过如果你愿意，也可以设置它。

组角色存在后，就可以使用GRANT和REVOKE命令添加和移除成员：

```
GRANT group_role TO role1, ... ;
REVOKE group_role FROM role1, ... ;
```

也可以把成员资格授予其他组角色（因为组角色与非组角色实际上没有任何区别）。数据库不允许建立循环成员关系。此外，也不允许将角色的成员资格授予 `PUBLIC`。

组角色的成员可以通过两种方式使用该角色的权限。首先，组的每个成员都可以显式执行SET ROLE，以暂时“成为”该组角色。在这种状态下，数据库会话可以使用组角色的权限，而不是原始登录角色的权限；创建的任何数据库对象也被视为由组角色而非登录角色拥有。其次，具有 `INHERIT` 属性的成员角色可以自动使用其所属角色的权限，包括这些角色继承的所有权限。例如，假定我们已经执行了：

```
CREATE ROLE joe LOGIN INHERIT;
CREATE ROLE admin NOINHERIT;
CREATE ROLE wheel NOINHERIT;
GRANT admin TO joe;
GRANT wheel TO admin;
```

以角色 `joe` 连接后，数据库会话立即可以使用直接授予 `joe` 的权限，以及授予 `admin` 的所有权限，因为 `joe` “继承” `admin` 的权限。不过，授予 `wheel` 的权限不可用，因为即使 `joe` 间接属于 `wheel`，该成员资格也是通过 `admin` 获得的，而后者具有 `NOINHERIT` 属性。执行以下命令后：

```
SET ROLE admin;
```

会话将只能使用授予 `admin` 的权限，而不能使用授予 `joe` 的权限。执行以下命令后：

```
SET ROLE wheel;
```

会话将只能使用授予 `wheel` 的权限，而不能使用授予 `joe` 或 `admin` 的权限。使用以下任一命令可以恢复原始的权限状态：

```
SET ROLE joe;
SET ROLE NONE;
RESET ROLE;
```

> **注意**
>
> SET ROLE命令总是允许切换到原始登录角色直接或间接所属的任何角色。因此，在上例中，没有必要先成为admin再成为wheel。

> **注意**
>
> 在 SQL 标准中，用户与角色之间有明确区别，并且用户不会自动继承权限，而角色会。这种行为可以在PostgreSQL中通过让用作 SQL 角色的角色具有INHERIT属性，而让用作 SQL 用户的角色具有NOINHERIT属性来获得。不过，出于与 8.1 之前版本向后兼容的考虑，PostgreSQL默认给所有角色都赋予INHERIT属性；在那些早期版本中，用户总能使用授予其所属组的权限。

角色属性LOGIN、SUPERUSER、CREATEDB和CREATEROLE可以被视为特殊权限，但它们从不会像数据库对象上的普通权限那样被继承。要使用其中之一，必须实际执行SET ROLE切换到拥有该属性的特定角色。继续上面的例子，我们可以选择把CREATEDB和CREATEROLE赋予admin角色。这样，一个以joe角色连接的会话并不会立即拥有这些权限，只有在执行SET ROLE admin之后才会拥有。

要删除组角色，请使用DROP ROLE：

```
DROP ROLE name;
```

该组角色中的所有成员资格都会自动被撤销（但成员角色不会受到其他影响）。

20.4. 删除角色

由于角色可以拥有数据库对象，并且可以拥有访问其他对象的权限，所以删除一个角色通常不只是简单执行一次DROP ROLE。必须先删除或重新分配该角色拥有的任何对象；并且必须撤销授予该角色的任何权限。

可以使用 ALTER 命令逐个转移对象的所有权，例如：

```
ALTER TABLE bobs_table OWNER TO alice;
```

另一种办法是使用REASSIGN OWNED命令，把待删除角色拥有的所有对象的所有权重新分配给另一个角色。由于 REASSIGN OWNED 无法访问其他数据库中的对象，因此必须在包含该角色所拥有对象的每个数据库中运行它。（请注意，首次执行这样的 REASSIGN OWNED 时，会更改待删除角色所拥有的所有跨数据库共享对象（即数据库或表空间）的所有权。）

一旦有价值的对象都已转移给新拥有者，待删除角色所拥有的其余对象即可使用DROP OWNED命令删除。同样，由于该命令不能访问其他数据库中的对象，因此必须在包含该角色所拥有对象的每个数据库中运行它。另外，DROP OWNED不会删除整个数据库或表空间，因此如果该角色拥有任何尚未转移给新拥有者的数据库或表空间，就必须手工删除它们。

DROP OWNED还会负责移除目标角色在其他角色所拥有的对象上获授的所有权限。由于REASSIGN OWNED不会处理这类对象，所以通常需要同时运行REASSIGN OWNED和DROP OWNED（按这个顺序！），才能完整移除待删除角色的依赖关系。

总之，移除曾用来拥有对象的角色，最通用的做法是：

```
REASSIGN OWNED BY doomed_role TO successor_role;
DROP OWNED BY doomed_role;
-- repeat the above commands in each database of the cluster
DROP ROLE doomed_role;
```

如果并非所有拥有的对象都要转移给同一个后继拥有者，最好先手工处理这些例外情况，然后再执行上述步骤作最后清理。

如果在仍有依赖对象存在时尝试执行`DROP ROLE`，它会发出消息指出哪些对象需要被重新分配或删除。

20.5. 函数和触发器安全性

函数和触发器允许用户把代码插入后端服务器，而其他用户可能会无意中执行这些代码。因此，这两种机制使用户能够相对容易地对其他用户实施“特洛伊木马”攻击。唯一真正的保护方式是严格控制谁能定义函数。

函数在后端服务器进程中运行，并具有数据库服务器守护进程的操作系统权限。如果函数所用的编程语言允许不受检查的内存访问，就有可能更改服务器的内部数据结构。因此，这类函数可以做很多别的事情，其中就包括绕过任何系统访问控制。允许这种访问的函数语言被认为是“不受信任的”，而PostgreSQL只允许超级用户创建用这类语言编写的函数。

第 21 章 管理数据库

每个正在运行的PostgreSQL服务器实例都管理着一个或多个数据库。因此，数据库是组织SQL对象（“数据库对象”）的最高层级。本章描述数据库的属性，以及如何创建、管理和删除数据库。

21.1. 概述

数据库是一个命名的SQL对象（“数据库对象”）集合。通常，每个数据库对象（表、函数等）都属于且只属于一个数据库。（不过也有少量系统目录，例如`pg_database`，属于整个集簇，可以从集簇中的每个数据库访问。）更准确地说，数据库是模式的集合，而模式中包含表、函数等。因此完整的层次结构是：服务器、数据库、模式、表（或其他类型的对象，如函数）。

当连接到数据库服务器时，客户端必须在连接请求中指定要连接的数据库名。一个连接无法访问多个数据库。不过，一个应用可以不受限制地打开到同一数据库或其他数据库的多个连接。数据库在物理上是相互隔离的，访问控制在连接级别管理。因此，如果一个PostgreSQL服务器实例要容纳彼此应当分离、并且在大多数情况下互不知晓的项目或用户，建议把它们放到不同的数据库中。如果项目或用户是相互关联的，并且应该能够使用彼此的资源，则应当把它们放在同一个数据库中，但可能使用不同的模式。模式是纯逻辑结构，谁能访问什么由权限系统管理。更多关于管理模式的信息见第 5.7 节。

数据库使用`CREATE DATABASE`命令创建（见第 21.2 节），使用`DROP DATABASE`命令删除（见第 21.5 节）。要确定现有数据库的集合，可以检查系统目录`pg_database`，例如：

```
SELECT datname FROM pg_database;
```

psql程序的`\l`元命令和`-l`命令行选项也可用于列出现有数据库。

> **注意**
>
> SQL标准将数据库称为“目录”，但在实践中两者没有区别。

21.2. 创建数据库

要创建数据库，PostgreSQL服务器必须已经启动并正在运行（见第 17.3 节）。

数据库用 SQL 命令CREATE DATABASE创建：

```
CREATE DATABASE name;
```

其中*name*遵循SQL标识符的一般规则。当前角色会自动成为新数据库的拥有者。数据库拥有者有权在日后删除该数据库（这也会删除其中的全部对象，即使那些对象具有不同的拥有者）。

创建数据库是一项受限操作。关于如何授予该权限，见第 20.2 节。

由于执行`CREATE DATABASE`命令时必须先连接到数据库服务器，因此还会有一个问题：某个站点上的第一个数据库是如何创建的？第一个数据库总是由`initdb`命令在初始化数据存储区时创建的（见第 17.2 节）。这个数据库叫作`postgres`。因此，要创建第一个“普通”数据库，可以连接到`postgres`。

在初始化数据库集簇时，还会创建第二个数据库`template1`。每当在集簇中创建一个新数据库时，新数据库本质上都是通过克隆`template1`得到的。

这意味着你在template1中所做的任何更改，都会传播到后来创建的所有数据库中。因此，除非你希望这些对象出现在每一个新建数据库中，否则应避免在template1中创建对象。更多细节见第 21.3 节。

为了方便，还可以从 shell 中执行一个程序来创建新数据库，即createdb。

```
createdb dbname
```

createdb并没有什么特殊之处。它连接到postgres数据库并发出CREATE DATABASE命令，和上面描述的完全一样。关于调用细节见createdb参考页。注意，不带任何参数的createdb会创建一个与当前用户名同名的数据库。

> **注意**
>
> 第 19 章包含关于如何限制谁能连接到给定数据库的信息。

有时你会想为其他人创建数据库，并让他们成为新数据库的拥有者，以便他们自行配置和管理它。要实现这一点，可使用下列命令之一：

```
CREATE DATABASE dbname OWNER rolename;
```

这是在 SQL 环境中使用；或者：

```
createdb -O rolename dbname
```

这是在 shell 中使用。只有超级用户才允许为其他人（即为一个你不是其成员的角色）创建数据库。

21.3. 模板数据库

CREATE DATABASE实际上是通过复制一个现有数据库来创建新数据库。默认情况下，它复制名为template1的标准系统数据库。因此，该数据库就是创建新数据库时使用的“模板”。如果你向template1中添加对象，这些对象也会被复制到以后创建的用户数据库中。这种行为允许对数据库中的标准对象集合进行站点本地修改。例如，如果你将过程语言PL/Perl安装到template1中，那么以后创建用户数据库时无需额外操作，它就会自动可用。

还有第二个标准系统数据库，名为template0。该数据库包含与template1初始内容相同的数据，也就是说，只包含你的PostgreSQL版本预定义的标准对象。在数据库集簇初始化之后，template0就不应再被修改。通过指示CREATE DATABASE复制template0而不是template1，你可以创建一个“纯净的”用户数据库，它不包含template1中的任何站点本地附加对象。这在恢复pg_dump转储时特别方便：转储脚本应当恢复到一个纯净的数据库中，以确保重建出被转储数据库的正确内容，而不会与后来可能加入template1的对象发生冲突。

复制template0而不是template1的另一个常见原因是：复制template0时可以指定新的编码和区域设置，而template1的副本则必须使用与它相同的设置。这是因为template1可能包含依赖特定编码或区域设置的数据，而已知template0不会包含这类数据。

要通过复制template0创建数据库，可在 SQL 环境中使用：

```
CREATE DATABASE dbname TEMPLATE template0;
```

或在 shell 中使用：

```
createdb -T template0 dbname
```

可以创建额外的模板数据库，实际上也可以通过将集簇中任意数据库的名称指定为CREATE DATABASE的模板来复制它。不过，必须理解的是，这（至少目前）并不是一个通用的“COPY DATABASE”功能。最主要的限制是：复制源数据库时，不能有其他会话连接到该数据库。如果CREATE DATABASE开始时存在任何其他连接，该命令就会失败；在复制操作期间，也会阻止到源数据库的新连接。

对每个数据库，pg_database中都有两个有用的标志列：datistemplate和datallowconn。datistemplate可设置为表明某个数据库将作为CREATE DATABASE的模板。如果设置了该标志，任何拥有CREATEDB权限的用户都可以克隆该数据库；如果未设置，则只有超级用户和数据库拥有者才能克隆它。如果datallowconn为假，则不允许建立到该数据库的新连接（但仅仅把该标志设为假并不会终止已有会话）。为了防止被修改，template0数据库通常被标记为datallowconn = false。template0和template1始终都应标记为datistemplate = true。

注意

template1和template0除了template1这个名字是CREATE DATABASE默认源数据库名这一点之外，并无任何特殊地位。例如，可以删除template1，然后再从template0重新创建它，而不会有任何不良后果。如果有人不小心在template1里加入了许多杂项对象，这样做也许是可取的。（要删除template1，必须先令其具有pg_database.datistemplate = false。）

初始化数据库集簇时也会创建postgres数据库。这个数据库被用作用户和应用连接时的默认数据库。它只是template1的一个副本，因此在必要时也可以删除并重建。

21.4. 数据库配置

正如第 18 章所述，PostgreSQL服务器提供了大量运行时配置变量。你可以为其中许多设置指定数据库特定的默认值。

例如，如果你因为某种原因想在某个数据库中禁用GEQO优化器，通常要么必须对所有数据库都禁用它，要么必须确保每个连入的客户端都会执行SET geqo TO off。若要让这个设置在某个特定数据库中成为默认值，可以执行如下命令：

```
ALTER DATABASE mydb SET geqo TO off;
```

这会保存该设置（但不会立即生效）。以后连接到该数据库时，它看起来就像是在会话开始前刚执行过SET geqo TO off;一样。注意，用户仍然可以在自己的会话中更改该设置；它只会作为默认值。要撤销这样的设置，可使用ALTER DATABASE *dbname* RESET *varname*。

21.5. 删除数据库

数据库使用DROP DATABASE命令删除：

```
DROP DATABASE name;
```

只有数据库拥有者或超级用户可以删除数据库。删除数据库会移除其中包含的全部对象。数据库的删除无法撤销。

当你连接到目标数据库时，不能执行DROP DATABASE命令。不过，你可以连接到任何其他数据库，包括template1数据库。若要删除某个集簇中的最后一个用户数据库，template1将是唯一的选择。

为了方便，还有一个 shell 程序可用于删除数据库，即dropdb。

```
dropdb dbname
```

（与`createdb`不同，默认操作并不是删除与当前用户名同名的数据库。）

21.6. 表空间

PostgreSQL中的表空间允许数据库管理员在文件系统中定义存放表示数据库对象的文件的位置。表空间一旦创建，在创建数据库对象时就可以通过名称引用它。

通过使用表空间，管理员可以控制PostgreSQL安装的磁盘布局。这至少有两方面的用途。首先，如果初始化集簇所在的分区或卷空间耗尽且无法扩展，可以在另一个分区上创建表空间并使用它，直到系统能够重新配置。

其次，表空间允许管理员利用对数据库对象使用模式的了解来优化性能。例如，一个使用非常频繁的索引可以放在速度非常快且高可用的磁盘上，例如昂贵的固态设备。同时，一个存放归档数据且很少使用或对性能并不关键的表，则可以放在成本更低、速度更慢的磁盘系统上。

要定义一个表空间，使用CREATE TABLESPACE命令，例如：

```
CREATE TABLESPACE fastspace LOCATION '/mnt/sda1/postgresql/data';
```

该位置必须是一个已经存在、为空并且由PostgreSQL操作系统用户拥有的目录。随后在这个表空间中创建的所有对象都会存储在该目录下的文件中。

> **注意**
>
> 通常来说，在每个逻辑文件系统上创建多个表空间意义不大，因为你无法控制单个文件在一个逻辑文件系统中的具体位置。不过，PostgreSQL并不强制这种限制，而且它实际上并不能直接感知你的系统中文件系统的边界。它只是在你指定的目录中存储文件。

表空间本身必须由数据库超级用户创建，但之后你可以允许普通数据库用户使用它。为此，需要向他们授予其上的`CREATE`权限。

表、索引以及整个数据库都可以被分配到特定的表空间。为此，对给定表空间拥有`CREATE`权限的用户必须把表空间名作为参数传给相应命令。例如，下列命令会在表空间`space1`中创建一个表：

```
CREATE TABLE foo(i int) TABLESPACE space1;
```

另一种方式是使用default_tablespace参数：

```
SET default_tablespace = space1;
CREATE TABLE foo(i int);
```

当`default_tablespace`被设置为非空字符串时，它会为未显式指定`TABLESPACE`子句的`CREATE TABLE`和`CREATE INDEX`命令隐含地补上该子句。

还有一个temp_tablespaces参数，用来决定临时表和索引以及为大数据集排序等用途所使用的临时文件应放置在哪里。它可以是一个表空间名列表，而不只是单个表空间，这样与临时对象相关的负载就可以分散到多个表空间上。每次创建临时对象时，都会从该列表中随机挑选一个成员。

与数据库关联的表空间用于存储该数据库的系统目录。此外，如果没有给出`TABLESPACE`子句，并且`default_tablespace`或`temp_tablespaces`（视情况而定）

也没有指定其他选择，那么它还是在该数据库中创建的表、
索引和临时文件所使用的默认表空间。如果创建数据库时没有为其指定表空间，
它将使用其所复制模板数据库的同一个表空间。

初始化数据库集簇时会自动创建两个表空间。pg_global表空间只用于共享系统目录。pg_default表空间是template1和template0数据库的默认表空间（因此，除非被TABLESPACE子句覆盖，而该子句可在CREATE DATABASE中指定，否则它也会是其他数据库的默认表空间）。

表空间一旦创建，只要请求的用户具有足够权限，就可以从任何数据库中使用它。
这意味着，只有当使用该表空间的所有数据库中的所有对象都已被移除之后，
才能删除该表空间。

要移除一个空表空间，使用DROP TABLESPACE命令。

要确定现有表空间的集合，可以检查pg_tablespace 系统目录，例如：

```
SELECT spcname FROM pg_tablespace;
```

psql程序的\db元命令对列出现有表空间也很有用。

PostgreSQL使用符号链接来简化表空间的实现。
这意味着表空间只能在支持符号链接的系统上使用。

$PGDATA/pg_tblspc目录包含符号链接，它们指向集簇中定义的每一个非内置表空间。
虽然不推荐，但也可以通过重新定义这些链接来手工调整表空间布局。无论如何，
都不要在服务器运行时执行此操作。注意，在 PostgreSQL 9.1 及更早版本中，你还需要用新位置来更新pg_tablespace目录。（如果不这样做，pg_dump
将继续输出旧的表空间位置。）

第 22 章 本地化

本章从管理员的角度介绍可用的本地化特性。PostgreSQL支持两类本地化机制：

- 利用操作系统的区域设置（locale）特性，提供与区域设置相关的排序顺序、数字格式、翻译后的消息以及其他方面的支持。详见第 22.1 节和第 22.2 节。
- 提供多种字符集，以支持存储各种语言的文本，并提供客户端与服务器之间的字符集转换。详见第 22.3 节。

22.1. 区域设置支持

区域设置支持是指应用程序在字母表、排序、数字格式等方面尊重文化偏好。PostgreSQL使用服务器操作系统提供的标准 ISO C 和POSIX区域设置机制。更多信息请参考系统文档。

22.1.1. 概述

在使用initdb创建数据库集簇时，区域设置支持会自动初始化。默认情况下，initdb会使用其执行环境中的区域设置来初始化数据库集簇，因此如果你的系统已经设置为数据库集簇想要使用的区域设置，就无需额外操作。如果想使用不同的区域设置（或者不确定系统当前设置了哪种区域设置），可以通过指定--locale选项，精确告诉initdb使用哪种区域设置。例如：

```
initdb --locale=sv_SE
```

这个 Unix 系统上的示例把区域设置为在瑞典（SE）使用的瑞典语（sv）。其他可能的值包括en_US（美国英语）和fr_CA（加拿大法语）。如果一个区域设置可以使用多于一种字符集，则说明可以采用 *language_territory.codeset*这样的形式。例如，fr_BE.UTF-8表示在比利时（BE）使用、采用 UTF-8字符集编码的法语（fr）。

系统上有哪些区域设置可用，以及它们以什么名称出现，取决于操作系统供应商提供了什么以及实际安装了什么。在大多数 Unix 系统上，命令locale -a会给出所有可用区域设置的列表。Windows 使用更冗长的区域设置名称，例如German_Germany或 Swedish_Sweden.1252，但原则是相同的。

有时混合多个区域设置的规则会很有用，例如使用英语排序规则但使用西班牙语消息。为此，存在一组只控制本地化规则某些方面的区域设置子类别：

LC_COLLATE	字符串排序顺序
LC_CTYPE	字符分类（什么算字母？它的大写等价形式是什么？）
LC_MESSAGES	消息的语言
LC_MONETARY	货币数量使用的格式
LC_NUMERIC	数字的格式
LC_TIME	日期和时间的格式

这些类别名会转换成initdb选项名，用于覆盖特定类别的区域设置选择。例如，要将区域设置设为加拿大法语，但对货币格式采用美国规则，可以使用initdb --locale=fr_CA --lc-monetary=en_US。

如果希望系统表现得像没有区域设置支持一样，可使用特殊区域设置名 C，或等效的POSIX。

某些区域设置类别的值必须在创建数据库时固定下来。你可以为不同数据库使用不同设置，但数据库一旦创建，这些类别的值就不能再更改。LC_COLLATE和LC_CTYPE就属于这类类别。它们会影响索引的排序顺序，因此必须保持固定，

否则文本列上的索引会损坏。（不过，可以通过排序规则缓解这一限制，参见第 22.2 节。）这些类别的默认值在运行initdb 时确定，创建新数据库时会使用这些值，除非在CREATE DATABASE 命令中另行指定。

其他区域设置类别可以在任何时候通过设置与其同名的服务器配置参数来更改（详见第 18.11.2 节）。initdb 选择的值实际上只是写入配置文件postgresql.conf中，作为服务器启动时的默认值。如果从postgresql.conf中删除这些赋值，
服务器就会从其执行环境继承相应设置。

请注意，服务器的区域设置行为由服务器看到的环境变量决定，
而不是由任何客户端的环境决定。因此，务必在启动服务器前配置好正确的区域设置。
由此带来的一个结果是，如果客户端和服务器使用不同的区域设置，
消息可能会因其来源不同而显示为不同语言。

> **注意**
>
> 这里所说的从执行环境继承区域设置，
> 在大多数操作系统上是指：对于某个给定的区域设置类别，例如排序规则，
> 会按以下顺序检查环境变量，直到找到一个已设置的变量为止：LC_ALL、LC_COLLATE（或相应类别对应的变量）、LANG。如果这些环境变量都没有设置，
> 则区域设置默认为C。
>
> 某些消息本地化库还会查看环境变量LANGUAGE，
> 它会覆盖所有其他用于设置消息的语言的区域设置。如果有疑问，
> 请参考操作系统文档，尤其是关于gettext的文档。

若要让消息可以翻译为用户偏好的语言，构建时必须启用NLS（configure --enable-nls）。所有其他区域设置支持都会自动内置。

22.1.2. 行为

区域设置会影响下列 SQL 特性：

- 在对文本数据使用ORDER BY或标准比较操作符的查询中的排序顺序
- upper、lower和 initcap函数
- 模式匹配操作符（LIKE、SIMILAR TO 以及 POSIX 风格正则表达式）；区域设置既会影响大小写不敏感匹配，也会影响字符类正则表达式中的字符分类
- to_char函数族
- 对LIKE子句使用索引的能力

在PostgreSQL中使用C或 POSIX之外的区域设置，缺点是会影响性能。它会减慢字符处理速度，并阻止普通索引用于LIKE。因此，只有在确有需要时才应使用区域设置。

为了让PostgreSQL在非 C 区域设置下也能对 LIKE子句使用索引，存在若干自定义操作符类可供使用。它们允许创建执行严格逐字符比较、忽略区域设置比较规则的索引。详见第 11.9 节。另一种方法是创建使用C 排序规则的索引，如第 22.2 节所述。

22.1.3. 问题

如果区域设置支持没有像上面说明的那样工作，
请检查操作系统中的区域设置支持是否已经正确配置。要检查系统中安装了哪些区域设置，
可以使用 locale -a命令（如果操作系统提供该命令）。

请确认PostgreSQL实际使用的区域设置就是你预期的区域设置。LC_COLLATE和LC_CTYPE是在创建数据库时确定的，除非重新创建数据库，否则不能更改。其他区域设置，包括 LC_

MESSAGES和LC_MONETARY，最初由启动服务器时的环境决定，但可以在运行中动态修改。你可以使用SHOW 命令检查当前生效的区域设置。

源码发布包中的src/test/locale目录包含 PostgreSQL区域设置支持的测试套件。

那些通过解析错误消息文本来处理服务器端错误的客户端应用，显然会遇到问题，因为服务器消息可能使用不同语言。这类应用的作者应改用错误代码机制。

维护消息翻译目录需要许多志愿者持续投入，他们希望 PostgreSQL能够良好地支持他们偏好的语言。如果你所用语言的消息目前还不可用，或尚未完全翻译，我们将非常感谢你的协助。如果你愿意帮忙，请参考第 48 章，或向开发者邮件列表写信。

22.2. 排序规则支持

排序规则特性允许为每一列甚至每一次操作指定数据的排序顺序和字符分类行为。这缓解了数据库的LC_COLLATE和LC_CTYPE 设置在创建之后无法更改这一限制。

22.2.1. 概念

从概念上讲，每个支持排序规则的数据类型的表达式都有一个排序规则。（内置的支持排序规则的数据类型包括text、varchar和char。用户定义的基础类型也可以标记为支持排序规则，当然，建立在支持排序规则的数据类型之上的域也支持排序规则。）如果表达式是列引用，则该表达式的排序规则就是该列定义的排序规则。如果表达式是常量，则其排序规则就是该常量数据类型的默认排序规则。更复杂表达式的排序规则则按下文所述，从其输入表达式的排序规则推导出来。

表达式的排序规则可以是“默认”排序规则，这表示为数据库定义的区域设置。表达式的排序规则也可能是不确定的。在这种情况下，排序操作以及其他需要知道排序规则的操作都会失败。

当数据库系统必须执行排序或字符分类时，它会使用输入表达式的排序规则。这例如发生在ORDER BY子句以及函数或操作符调用（如 <）中。应用于ORDER BY子句的排序规则就是排序键的排序规则。应用于函数或操作符调用的排序规则则按下文所述，从参数中推导出来。除了比较操作符之外，在大小写之间进行转换的函数（例如lower、upper和 initcap）、模式匹配操作符，以及to_char 及相关函数，也都会考虑排序规则。

对于函数或操作符调用，通过检查参数排序规则推导出的排序规则，会在运行时用于执行指定操作。如果该函数或操作符调用的结果属于支持排序规则的数据类型，那么在解析时它也会被用作该函数或操作符表达式的已定义排序规则，以便在外围表达式需要知道其排序规则时使用。

表达式的排序规则派生可以是显式的，也可以是隐式的。这一区别会影响当一个表达式中出现多个不同排序规则时，系统如何把它们组合起来。使用COLLATE子句时，会发生显式排序规则派生；其他所有排序规则派生都是隐式的。当需要组合多个排序规则时，例如在函数调用中，将使用以下规则：

1. 如果任一输入表达式具有显式排序规则派生，那么输入表达式中所有显式派生的排序规则都必须相同，否则会报错。如果存在显式派生的排序规则，那么排序规则组合的结果就是该排序规则。

2. 否则，所有输入表达式都必须具有相同的隐式排序规则派生，或者为默认排序规则。如果存在任何非默认排序规则，那么排序规则组合的结果就是该排序规则；否则，结果就是默认排序规则。

3. 如果输入表达式之间存在相互冲突的非默认隐式排序规则，则该组合会被视为具有不确定排序规则。除非被调用的特定函数确实需要知道它应当使用哪个排序规则，否则这并不是错误。如果它确实需要，就会在运行时抛出错误。

例如，考虑如下表定义：

```
CREATE TABLE test1 (
    a text COLLATE "de_DE",
    b text COLLATE "es_ES",
    ...
);
```

那么在

```
SELECT a < 'foo' FROM test1;
```

中，<比较会按de_DE规则进行，
因为该表达式组合了一个隐式派生的排序规则和默认排序规则。但在

```
SELECT a < ('foo' COLLATE "fr_FR") FROM test1;
```

中，比较会按fr_FR规则进行，因为显式排序规则派生覆盖了隐式派生。进一步，给定

```
SELECT a < b FROM test1;
```

解析器无法确定应当应用哪个排序规则，因为a列和b列具有冲突的隐式排序规则。由于 <操作符确实需要知道要使用哪个排序规则，因此这会导致一个错误。
该错误可以通过给任一输入表达式附加显式排序规则说明符来解决，例如：

```
SELECT a < b COLLATE "de_DE" FROM test1;
```

或者等价地：

```
SELECT a COLLATE "de_DE" < b FROM test1;
```

另一方面，结构相似的情况

```
SELECT a || b FROM test1;
```

不会导致错误，因为||操作符不关心排序规则：无论使用什么排序规则，其结果都相同。

如果函数或操作符返回的是支持排序规则的数据类型，
那么分配给该函数或操作符组合输入表达式的排序规则，也被认为适用于其结果。因此，在

```
SELECT * FROM test1 ORDER BY a || 'foo';
```

中，排序将按de_DE规则进行。但这个查询：

```
SELECT * FROM test1 ORDER BY a || b;
```

会报错，因为即使||操作符本身不需要知道排序规则，ORDER BY子句仍然需要。与之前一样，可以通过显式排序规则说明符来解决冲突：

```
SELECT * FROM test1 ORDER BY a || b COLLATE "fr_FR";
```

22.2.2. 管理排序规则

排序规则是一个 SQL 模式对象，它把某个 SQL 名称映射到操作系统的区域设置。具体来说，它对应于LC_COLLATE和LC_CTYPE 的一种组合。（顾名思义，排序规则的主要用途是设置控制排序顺序的 LC_COLLATE。但在实践中，很少有必要让 LC_CTYPE和LC_COLLATE不同，因此把二者归为一个概念，比再建立一套为每个表达式设置LC_CTYPE 的机制更方便。）此外，排序规则还与某种字符集编码绑定（见第 22.3 节）。同一个排序规则名称可能会在不同编码中出现。

所有平台都提供名为 default、C 和 POSIX 的排序规则。根据操作系统支持情况，还可能提供其他排序规则。default 排序规则选择创建数据库时指定的 LC_COLLATE 和 LC_CTYPE 值。C 和 POSIX 排序规则都采用“传统 C”行为，仅将 ASCII 字母“A”到“Z”视为字母，并严格按字符编码的字节值排序。

如果操作系统支持在单个程序中使用多个区域设置（newlocale 及相关函数），那么在初始化数据库集簇时，initdb会根据当时在操作系统中发现的所有区域设置，用排序规则填充系统目录pg_collation。例如，操作系统可能提供一个名为de_DE.utf8的区域设置。initdb随后会为UTF8编码创建一个名为de_DE.utf8的排序规则，其 LC_COLLATE和LC_CTYPE都设置为 de_DE.utf8。它还会再创建一个从名称中去掉 .utf8标签的排序规则。因此，你也可以使用 de_DE这个名称来使用该排序规则，这样写起来更方便，且名称与编码的耦合更小。不过请注意，初始排序规则名称集合仍然取决于平台。

如果需要一个LC_COLLATE和LC_CTYPE取值不同的排序规则，可以使用CREATE COLLATION命令创建新的排序规则。该命令也可以用来从一个已有排序规则创建新的排序规则，这有助于在应用中使用与操作系统无关的排序规则名称。

在任何特定数据库中，只有使用该数据库编码的排序规则才有意义。pg_collation 中的其他条目会被忽略。因此，像 de_DE这样去掉编码后缀的排序规则名称，在某个给定数据库内可以视为唯一，即使它在全局范围内并不唯一。推荐使用这种去掉后缀的排序规则名称，因为如果你决定改用另一种数据库编码，需要改动的地方会更少。不过要注意，default、C和POSIX 排序规则不受数据库编码影响，始终都可以使用。

PostgreSQL即使面对具有相同属性的不同排序规则对象，也会把它们视为不兼容。例如：

```
SELECT a COLLATE "C" < b COLLATE "POSIX" FROM test1;
```

即使C和POSIX排序规则的行为完全相同，这仍然会报错。因此，不建议混用去掉后缀和保留后缀的排序规则名称。

22.3. 字符集支持

PostgreSQL中的字符集支持允许你以多种字符集（也称为编码）存储文本，包括 ISO 8859 系列之类的单字节字符集，以及 EUC（扩展 Unix 编码）、UTF-8 和 Mule 内部编码等多字节字符集。所有受支持的字符集都可以被客户端透明地使用，但其中少数不支持在服务器内部使用（即不能作为服务器端编码）。默认字符集是在使用 initdb初始化PostgreSQL 数据库集簇时选定的。创建数据库时可以覆盖该设置，因此你可以拥有多个数据库，并让每个数据库使用不同的字符集。

不过，一项重要限制是，每个数据库的字符集必须与数据库的 LC_CTYPE（字符分类）和 LC_COLLATE（字符串排序顺序）区域设置兼容。对于 C 或 POSIX 区域设置，允许使用任何字符集；对于其他区域设置，只有一种字符集能够正确工作。（但在 Windows 上，UTF-8 编码可以与任何区域设置一起使用。）

22.3.1. 支持的字符集

表 22.1显示了PostgreSQL中可用的字符集。

表 22.1. PostgreSQL 字符集

名称	描述	语言	是否服务器端?	字节数/字符	别名
BIG5	Big Five	繁体中文	否	1-2	WIN950, Windows950
EUC_CN	扩展 Unix 编码-CN	简体中文	是	1-3	
EUC_JP	扩展 Unix 编码-JP	日语	是	1-3	
EUC_JIS_2004	扩展 Unix 编码-JP，JIS X 0213	日语	是	1-3	

名称	描述	语言	是否服务器端?	字节数/字符	别名
EUC_KR	扩展 Unix 编码-KR	韩语	是	1-3	
EUC_TW	扩展 Unix 编码-TW	繁体中文、台湾语	是	1-3	
GB18030	国家标准	中文	否	1-4	
GBK	扩展国家标准	简体中文	否	1-2	WIN936, Windows936
ISO_8859_5	ISO 8859-5, ECMA 113	拉丁语/西里尔语	是	1	
ISO_8859_6	ISO 8859-6, ECMA 114	拉丁语/阿拉伯语	是	1	
ISO_8859_7	ISO 8859-7, ECMA 118	拉丁语/希腊语	是	1	
ISO_8859_8	ISO 8859-8, ECMA 121	拉丁语/希伯来语	是	1	
JOHAB	JOHAB	韩语	否	1-3	
KOI8R	KOI8-R	西里尔语（俄语）	是	1	KOI8
KOI8U	KOI8-U	西里尔语（乌克兰语）	是	1	
LATIN1	ISO 8859-1, ECMA 94	西欧	是	1	ISO88591
LATIN2	ISO 8859-2, ECMA 94	中欧	是	1	ISO88592
LATIN3	ISO 8859-3, ECMA 94	南欧	是	1	ISO88593
LATIN4	ISO 8859-4, ECMA 94	北欧	是	1	ISO88594
LATIN5	ISO 8859-9, ECMA 128	土耳其语	是	1	ISO88599
LATIN6	ISO 8859-10, ECMA 144	北欧	是	1	ISO885910
LATIN7	ISO 8859-13	波罗的海	是	1	ISO885913
LATIN8	ISO 8859-14	凯尔特语	是	1	ISO885914
LATIN9	ISO 8859-15	带欧元符号和重音字符的LATIN1	是	1	ISO885915
LATIN10	ISO 8859-16, ASRO SR 14111	罗马尼亚语	是	1	ISO885916
MULE_INTERNAL	Mule 内部编码	多语种 Emacs	是	1-4	
SJIS	Shift JIS	日语	否	1-2	Mskanji, ShiftJIS, WIN932, Windows932
SHIFT_JIS_2004	Shift JIS, JIS X 0213	日语	否	1-2	
SQL_ASCII	未指定（见正文）	任意	是	1	

名称	描述	语言	是否服务器端?	字节数/字符	别名
UHC	统一韩语编码	韩语	否	1-2	WIN949, Windows949
UTF8	Unicode，8 位	所有	是	1-4	Unicode
WIN866	Windows CP866	西里尔语	是	1	ALT
WIN874	Windows CP874	泰语	是	1	
WIN1250	Windows CP1250	中欧	是	1	
WIN1251	Windows CP1251	西里尔语	是	1	WIN
WIN1252	Windows CP1252	西欧	是	1	
WIN1253	Windows CP1253	希腊语	是	1	
WIN1254	Windows CP1254	土耳其语	是	1	
WIN1255	Windows CP1255	希伯来语	是	1	
WIN1256	Windows CP1256	阿拉伯语	是	1	
WIN1257	Windows CP1257	波罗的海	是	1	
WIN1258	Windows CP1258	越南语	是	1	ABC, TCVN, TCVN5712, VSCII

并非所有客户端API都支持上表中的全部字符集。例如，PostgreSQL JDBC 驱动就不支持 MULE_INTERNAL、LATIN6、LATIN8和LATIN10。

SQL_ASCII 设置的行为与其他设置有很大不同。当服务器字符集为 SQL_ASCII 时，服务器按 ASCII 标准解释字节值 0-127，而将字节值 128-255 视为不作解释的字符。设置为 SQL_ASCII 时，不会进行任何编码转换。因此，这一设置与其说是声明使用某种特定编码，不如说是声明不关心编码。在大多数情况下，如果需要处理任何非 ASCII 数据，使用 SQL_ASCII 都是不明智的，因为 PostgreSQL 将无法帮助转换或验证非 ASCII 字符。

22.3.2. 设置字符集

initdb为PostgreSQL集簇定义默认字符集（编码）。例如：

```
initdb -E EUC_JP
```

这会把默认字符集设为EUC_JP（日语的扩展 Unix 编码）。如果你喜欢更长的选项形式，也可以用--encoding代替-E。如果既没有给出-E也没有给出 --encoding，initdb会根据指定的或默认的区域设置，尝试确定应使用的合适编码。

你可以在创建数据库时指定非默认编码，只要该编码与所选区域设置兼容：

```
createdb -E EUC_KR -T template0 --lc-collate=ko_KR.euckr --lc-
ctype=ko_KR.euckr korean
```

这将创建一个名为korean的数据库，它使用 EUC_KR字符集和ko_KR区域设置。另一种实现方式是使用以下 SQL 命令：

```
CREATE DATABASE korean WITH ENCODING 'EUC_KR' LC_COLLATE='ko_KR.
euckr' LC_CTYPE='ko_KR.euckr' TEMPLATE=template0;
```

请注意，上述命令指定复制template0数据库。如果复制任何其他数据库，
就不能更改源数据库的编码和区域设置，因为这可能导致数据损坏。详见第 21.3 节。

数据库编码存储在系统目录pg_database中。可以通过 psql的-l选项或\l 命令查看它。

```
$ psql -l
                                      List of databases
   Name    |  Owner   | Encoding  |  Collation  |    Ctype    |
      Access Privileges
-----------+----------+-----------+-------------+-------------+----
--------------------------------
 clocaledb | hlinnaka | SQL_ASCII | C           | C           |
 englishdb | hlinnaka | UTF8      | en_GB.UTF8  | en_GB.UTF8  |
 japanese  | hlinnaka | UTF8      | ja_JP.UTF8  | ja_JP.UTF8  |
 korean    | hlinnaka | EUC_KR    | ko_KR.euckr | ko_KR.euckr |
 postgres  | hlinnaka | UTF8      | fi_FI.UTF8  | fi_FI.UTF8  |
 template0 | hlinnaka | UTF8      | fi_FI.UTF8  | fi_FI.UTF8  | {=
c/hlinnaka,hlinnaka=CTc/hlinnaka}
 template1 | hlinnaka | UTF8      | fi_FI.UTF8  | fi_FI.UTF8  | {=
c/hlinnaka,hlinnaka=CTc/hlinnaka}
(7 rows)
```

重要

在大多数现代操作系统上，PostgreSQL能够判断LC_CTYPE设置所隐含的字符集，
并强制只允许使用与之匹配的数据库编码。在较老的系统上，
你需要自行确保所用编码正是所选区域设置期望的编码。
这里的错误很可能会导致区域设置相关操作（例如排序）出现奇怪的行为。

即使LC_CTYPE不是C或 POSIX，PostgreSQL 仍允许超级用户创建使用SQL_ASCII
编码的数据库。正如前文所述，SQL_ASCII
并不强制数据库中存储的数据具有任何特定编码，
因此这种选择存在区域设置相关误行为的风险。这种设置组合已经被弃用，
未来某一天可能会被完全禁止。

22.3.3. 服务器和客户端之间的自动字符集转换

PostgreSQL 支持在服务器和客户端之间，对某些字符集组合进行自动字符集转换。转换信息保存在系统目录 pg_conversion 中。PostgreSQL 提供了一些预定义转换，如表 22.2所示。可以使用 SQL 命令 CREATE CONVERSION 创建新的转换。

表 22.2. 客户端/服务器字符集转换

服务器字符集	可用的客户端字符集
BIG5	不支持作为服务器编码
EUC_CN	EUC_CN, MULE_INTERNAL, UTF8
EUC_JP	EUC_JP, MULE_INTERNAL, SJIS, UTF8
EUC_KR	EUC_KR, MULE_INTERNAL, UTF8
EUC_TW	EUC_TW, BIG5, MULE_INTERNAL, UTF8
GB18030	不支持作为服务器编码
GBK	不支持作为服务器编码
ISO_8859_5	ISO_8859_5, KOI8R, MULE_INTERNAL, UTF8, WIN866, WIN1251

服务器字符集	可用的客户端字符集
ISO_8859_6	ISO_8859_6, UTF8
ISO_8859_7	ISO_8859_7, UTF8
ISO_8859_8	ISO_8859_8, UTF8
JOHAB	JOHAB, UTF8
KOI8R	KOI8R, ISO_8859_5, MULE_INTERNAL, UTF8, WIN866, WIN1251
KOI8U	KOI8U, UTF8
LATIN1	LATIN1, MULE_INTERNAL, UTF8
LATIN2	LATIN2, MULE_INTERNAL, UTF8, WIN1250
LATIN3	LATIN3, MULE_INTERNAL, UTF8
LATIN4	LATIN4, MULE_INTERNAL, UTF8
LATIN5	LATIN5, UTF8
LATIN6	LATIN6, UTF8
LATIN7	LATIN7, UTF8
LATIN8	LATIN8, UTF8
LATIN9	LATIN9, UTF8
LATIN10	LATIN10, UTF8
MULE_INTERNAL	MULE_INTERNAL, BIG5, EUC_CN, EUC_JP , EUC_KR, EUC_TW, ISO_8859_5, KOI8R, LATIN1 至 LATIN4, SJIS, WIN866, WIN1250, WIN1251
SJIS	不支持作为服务器编码
SQL_ASCII	任何（不会执行转换）
UHC	不支持作为服务器编码
UTF8	所有受支持的编码
WIN866	WIN866, ISO_8859_5, KOI8R, MULE_INTERNAL, UTF8, WIN1251
WIN874	WIN874, UTF8
WIN1250	WIN1250, LATIN2, MULE_INTERNAL, UTF8
WIN1251	WIN1251, ISO_8859_5, KOI8R, MULE_INTERNAL, UTF8, WIN866
WIN1252	WIN1252, UTF8
WIN1253	WIN1253, UTF8
WIN1254	WIN1254, UTF8
WIN1255	WIN1255, UTF8
WIN1256	WIN1256, UTF8
WIN1257	WIN1257, UTF8
WIN1258	WIN1258, UTF8

若要启用自动字符集转换，必须告诉PostgreSQL 你希望客户端使用哪种字符集（编码）。有几种方法可以做到这一点：

- 在psql中使用\encoding 命令。\encoding允许动态更改客户端编码。例如，要把编码改成SJIS，可以输入：

  ```
  \encoding SJIS
  ```

- libpq（第 31.10 节）提供了控制客户端编码的函数。

- 使用`SET client_encoding TO`。可以使用以下 SQL 命令设置客户端编码：

  ```
  SET CLIENT_ENCODING TO 'value';
  ```

  也可以为此使用标准 SQL 语法`SET NAMES`：

  ```
  SET NAMES 'value';
  ```

  要查询当前客户端编码：

  ```
  SHOW client_encoding;
  ```

  要返回到默认编码：

  ```
  RESET client_encoding;
  ```

- 使用`PGCLIENTENCODING`。如果在客户端环境中定义了 `PGCLIENTENCODING`环境变量，那么连接服务器时会自动选择该客户端编码。
  （之后仍可以用上面提到的任何其他方法覆盖它。）

- 使用配置变量client_encoding。如果设置了 `client_encoding`变量，那么连接服务器时会自动选择该客户端编码。（之后仍可以用上面提到的任何其他方法覆盖它。）

如果某个特定字符无法完成转换，就会报错。例如，假设服务器使用 `EUC_JP`而客户端使用`LATIN1`，此时若返回了一些在`LATIN1`中没有表示形式的日语字符，就会出现错误。

如果客户端字符集定义为 `SQL_ASCII`，则无论服务器字符集是什么，都会禁用编码转换。与服务器一样，除非处理的全部是 ASCII 数据，否则使用 `SQL_ASCII` 是不明智的。

22.3.4. 进一步阅读

这些都是开始了解各种编码系统的良好资料。

CJKV Information Processing: Chinese, Japanese, Korean & Vietnamese Computing

其中对`EUC_JP`、`EUC_CN`、`EUC_KR`和`EUC_TW`做了详细说明。

http://www.unicode.org/

Unicode Consortium 网站。

RFC 3629

这里定义了UTF-8（8 位 UCS/Unicode 转换格式）。

第 23 章 日常数据库维护任务

和任何数据库软件一样，PostgreSQL为了获得最佳性能，需要定期执行某些任务。这里讨论的任务是必需的，但它们本质上是重复性的，因此可以很容易地用标准工具实现自动化，例如 cron 脚本或 Windows 的 Task Scheduler。建立合适的脚本并检查其是否成功执行，是数据库管理员的职责。

一个显而易见的维护任务，是按固定计划创建数据的备份副本。没有最近的备份，在灾难（磁盘故障、火灾、误删关键表等）发生后就没有恢复的可能。PostgreSQL提供的备份和恢复机制在第 24 章中有详细讨论。

另一大类维护任务是定期对数据库进行“清理”。这一活动在第 23.1 节中讨论。与之密切相关的是更新查询规划器将会使用的统计信息，这在第 23.1.3 节中讨论。

另一项可能需要定期关注的任务是日志文件管理。这在第 23.3 节中讨论。

check_postgres[1] 可用于监控数据库健康状况并报告异常情况。check_postgres能与 Nagios 和 MRTG 集成，但也可以单独运行。

和某些其他数据库管理系统相比，PostgreSQL的维护工作量较小。不过，适当地关注这些任务，将大大有助于确保你愉快而高效地使用该系统。

23.1. 日常清理

PostgreSQL数据库需要一种称为清理的周期性维护。对于许多安装，让第 23.1.6 节所述的自动清理守护进程执行清理就足够了。为了在你的场景中获得最佳效果，你可能需要调整其中描述的自动清理参数。有些数据库管理员希望用手工管理的 `VACUUM`命令来补充甚至取代该守护进程的工作，这类命令通常由 cron 或 Task Scheduler 脚本按计划执行。要正确设置手工管理的清理，理解下面几小节讨论的问题至关重要。依赖自动清理的管理员也不妨略读这一材料，以帮助理解和调整自动清理。

23.1.1. 清理基础

PostgreSQL 的VACUUM命令必须定期处理每个表，原因如下：

1. 回收或再利用被更新或删除的行所占用的磁盘空间。
2. 更新 PostgreSQL 查询规划器使用的数据统计信息。
3. 更新可见性映射，以加快仅索引扫描。
4. 防止由于事务 ID 回卷而丢失非常旧的数据。

针对上述不同目的，需要以不同的频率和范围执行 `VACUUM` 操作，下面几节会详细说明。

`VACUUM`有两种变体：标准的 `VACUUM` 和 `VACUUM FULL`。`VACUUM FULL`可以回收更多磁盘空间，但执行速度要慢得多。此外，标准形式的 `VACUUM` 可以与生产数据库操作并行运行。（`SELECT`、`INSERT`、`UPDATE` 以及 `DELETE` 等命令会继续正常工作，不过在表被清理时，你将不能使用 `ALTER TABLE` 等命令修改该表的定义。）`VACUUM FULL` 需要对正在处理的表持有独占锁，因此不能与对该表的其他使用并行进行。一般来说，管理员应尽量使用标准 `VACUUM` 并避免 `VACUUM FULL`。

`VACUUM`会产生大量 I/O 流量，这可能导致其他活动会话性能变差。可以调整一些配置参数来降低后台清理对性能的影响，参见第 18.4.4 节。

[1] https://bucardo.org/check_postgres/

23.1.2. 回收磁盘空间

在 PostgreSQL 中，对某一行执行 UPDATE 或 DELETE 时，不会立即移除该行的旧版本。这种方法对于获得多版本并发控制（MVCC，见第 13 章）的好处是必需的：当行版本仍可能对其他事务可见时，就不能删除它。但最终，
过时或已删除的行版本将不再是任何事务关心的对象。必须回收它占用的空间，
以供新行重用，从而避免磁盘空间需求无限增长。这是通过运行 VACUUM 完成的。

标准形式的 VACUUM 会移除表和索引中的死行版本，并将空间标记为可供将来重用。不过，它不会把空间归还给操作系统，除非出现一种特殊情况：表尾的一个或多个页面完全空闲，
并且能够轻松获得一个排他表锁。相比之下，VACUUM FULL 会主动压实表，它会写出一个不含死空间的全新表文件版本。这会将表的大小降到最小，但可能耗时很长。
在操作完成之前，它还需要额外的磁盘空间来存放表的新副本。

例行清理的通常目标，是足够频繁地执行标准 VACUUM，从而避免需要 VACUUM FULL。自动清理守护进程就是按照这种方式工作的，事实上它永远不会发出 VACUUM FULL。这种方法的思路不是让表始终保持在最小尺寸，
而是让磁盘空间的使用维持在稳态：每个表占用的空间相当于其最小尺寸，
再加上两次清理之间又被用掉的空间。虽然 VACUUM FULL 可以把表重新缩小到最小尺寸，并把磁盘空间归还给操作系统，但如果该表之后还会再次增长，这样做意义并不大。因此，
对于维护被频繁更新的表，与其不常执行 VACUUM FULL，不如适度频繁地执行标准 VACUUM。

有些管理员喜欢自己安排清理，例如在夜间负载较低时完成全部工作。
按固定时间表执行清理的难点在于，如果某个表的更新活动出现意外高峰，
它可能膨胀到确实需要 VACUUM FULL 才能回收空间的程度。使用自动清理守护进程可以缓解这个问题，因为守护进程会根据更新活动动态安排清理。除非工作负载极其可预测，
否则完全禁用守护进程是不明智的。一种可能的折中办法是设置守护进程参数，
使其只对异常繁重的更新活动作出反应，从而防止情况失控，而在负载正常时，
则期望定期调度的 VACUUM 完成大部分工作。

对于不使用自动清理的人，一个典型的做法是在低使用时段每天安排一次面向整个数据库的 VACUUM，并在必要时更频繁地清理那些更新特别频繁的表。（某些更新率极高的安装，
甚至会每隔几分钟就清理一次最繁忙的表。）如果你在一个集簇中有多个数据库，
不要忘记对每一个数据库都执行 VACUUM；程序vacuumdb可能会有帮助。

> **提示**
>
> 当一个表由于大规模更新或删除活动而包含大量死行版本时，普通的 VACUUM 可能并不能令人满意。如果你有这样一个表，并且需要回收它所占用的多余磁盘空间，
> 就需要使用 VACUUM FULL，或者改用CLUSTER，又或者使用ALTER TABLE的某一种重写表变体。这些命令会重写整个表的新副本，并为其构建新的索引。
> 所有这些选项都需要独占锁。还要注意，
> 它们会临时额外占用大约等于该表大小的磁盘空间，因为在新表和新索引完成之前，
> 旧的表和索引副本都不能被释放。

> **提示**
>
> 如果你有一个表，其全部内容会被定期删除，考虑使用TRUNCATE，而不是先 DELETE 再 VACUUM。TRUNCATE 会立即移除表的全部内容，而不需要后续再执行 VACUUM 或 VACUUM FULL 来回收此时未使用的磁盘空间。缺点是它会破坏严格的 MVCC 语义。

23.1.3. 更新规划器统计信息

PostgreSQL 查询规划器依赖于有关表内容的统计信息，以便为查询生成良好的计划。这些统计信息由ANALYZE命令收集，它既可以单独调用，也可以作为 VACUUM 的一个可选步骤执行。拥有足够准确的统计信息很重要，否则糟糕的计划选择可能会降低数据库性能。

如果启用了自动清理守护进程，它会在表内容发生足够大变化时自动发出 ANALYZE 命令。不过，管理员也可能更愿意依靠手工调度的 ANALYZE 操作，尤其是在已知某个表上的更新活动不会影响“重要”列统计信息的情况下。守护进程严格依据插入或更新的行数来安排 ANALYZE；它并不知道这些变化是否会导致有意义的统计变化。

就像为了回收空间而进行的清理一样，
频繁更新统计信息对于更新频繁的表比对于很少更新的表更有用。但即便是更新频繁的表，
如果数据的统计分布变化不大，也未必需要更新统计信息。
一个简单的经验法则是考虑表中各列的最小值和最大值变化了多少。例如，
一个包含行更新时间的 timestamp 列，会随着行的插入和更新而持续增大其最大值；这样的列可能比例如存放网站页面 URL 的列更需要频繁更新统计信息。URL 列收到更改的频率可能一样高，但其值的统计分布变化大概相对缓慢。

可以对特定表，甚至仅对表中的特定列运行 ANALYZE，因此如果你的应用需要，确实可以比其他统计更频繁地更新某些统计信息。然而在实践中，通常最好直接分析整个数据库，
因为这是一项很快的操作。ANALYZE 使用对表行的统计随机抽样，而不是读取每一行。

提示

虽然按列微调 ANALYZE 的频率未必很有成效，但按列调整 ANALYZE 收集的统计信息详细程度可能是值得的。在 WHERE 子句中被频繁使用且数据分布高度不规则的列，可能需要比其他列更细粒度的数据直方图。参见 ALTER TABLE SET STATISTICS，或者使用配置参数default_statistics_target更改数据库范围的默认值。

此外，默认情况下，关于函数选择率的信息很有限。不过，
如果创建了使用函数调用的表达式索引，系统就会收集关于该函数的有用统计信息，
这可以显著改善使用该表达式索引的查询计划。

提示

自动清理守护进程不会为外部表发出 ANALYZE 命令，因为它无法判断多高的频率才有用。如果你的查询需要依赖外部表的统计信息来正确规划，
一个好办法是按照合适的时间表，在这些表上手工运行 ANALYZE 命令。

23.1.4. 更新可见性映射

清理会为每个表维护一个可见性映射，
用来跟踪哪些页面只包含已知对所有活动事务都可见的元组（并且在该页面再次被修改之前，对所有未来事务也都可见）。这样做有两个目的。第一，
下一次清理时可以跳过这样的页面，因为其中没有需要清理的内容。

第二，它使 PostgreSQL 能够在不访问底层表的情况下，仅使用索引回答某些查询。由于 PostgreSQL 的索引不包含元组可见性信息，普通索引扫描会为每个匹配的索引条目取回堆元组，以检查当前事务是否应该看到它。相反，仅索引扫描会先检查可见性映射。
如果已知该页面上的所有元组都可见，就可以跳过对堆的访问。
这在大型数据集中尤其明显，因为可见性映射能够避免磁盘访问。可见性映射比堆小得多，
因此即便堆非常大，也很容易缓存。

23.1.5. 防止事务 ID 回卷失败

PostgreSQL 的 MVCC 事务语义依赖于比较事务 ID（XID）的大小：如果某个行版本的插入 XID 大于当前事务的 XID，它就处于“未来”，不应对当前事务可见。但是，事务 ID 的大小有限（32 位），长期运行的集簇（超过 40 亿个事务）会遇到事务 ID 回卷：XID 计数器回到零，原本处于过去的事务突然看起来处于未来 – 这意味着它们的结果变得不可见。也就是灾难性的数据丢失。（数据其实还在，但如果无法访问，也无济于事。）为避免这种情况，必须至少每经过 20 亿个事务，就清理一次每个数据库中的每个表。

定期清理之所以能解决这一问题，是因为 PostgreSQL 保留了一个称为 `FrozenXID` 的特殊 XID。该 XID 不遵循普通 XID 的比较规则，始终被视为比每个普通 XID 都老。普通 XID 使用模 2^{32} 算术进行比较。这意味着，对每个普通 XID，都有 20 亿个 XID 比它“更老”，另有 20 亿个 XID 比它“更新”；换句话说，普通 XID 空间是一个没有端点的环。因此，用某个普通 XID 创建行版本后，无论该 XID 是多少，在接下来的 20 亿个事务中，该行版本都会被视为处于“过去”。如果经过超过 20 亿个事务后它仍然存在，就会突然被视为处于未来。为避免这种情况，旧的行版本在到达 20 亿事务这一年限之前，必须被重新赋予 XID `FrozenXID`。一旦被赋予这个特殊 XID，无论是否发生回卷，对所有普通事务而言，它们都处于“过去”。所以，这类行版本在被删除之前始终有效，无论存在多久。这种旧 XID 的重新赋值由 `VACUUM` 完成。

vacuum_freeze_min_age控制某个 XID 值需要达到多老才会被替换为 `FrozenXID`。该设置取较大值可以更长时间地保留事务信息，
而取较小值则会增加在表必须再次被清理之前可以流逝的事务数量。

`VACUUM` 通常会跳过没有死行版本的页面，但这些页面中可能仍有 XID 值很旧的行版本。要确保所有旧 XID 都已被替换为 `FrozenXID`，就需要对整个表进行一次扫描。vacuum_freeze_table_age控制 `VACUUM` 何时执行这种操作：如果该表在 `vacuum_freeze_table_age` 减去 `vacuum_freeze_min_age` 个事务内未曾被完整扫描，就会强制进行一次全表扫描。将其设为 0 会强制 `VACUUM` 始终扫描所有页面，实际上是忽略可见性映射。

一个表可以不清理的最长时间，是 20 亿个事务减去 `VACUUM` 上次扫描全表时的 `vacuum_freeze_min_age` 值。超过这个期限仍不清理，就可能丢失数据。为确保不会发生这种情况，对于可能包含 XID 年龄大于配置参数autovacuum_freeze_max_age所指定年龄的任何表，都会调用自动清理。（即使禁用了自动清理，也会如此。）

这意味着，如果一个表本来不会因为其他原因被清理，大约每经过 `autovacuum_freeze_max_age` 减去 `vacuum_freeze_min_age` 个事务，就会在该表上触发一次自动清理。对于那些为了回收空间而经常清理的表，这一点并不重要。然而，
对于静态表（包括只接收插入、但没有更新或删除的表），并不需要为了回收空间而清理，
因此设法尽可能拉大这些非常大的静态表上强制自动清理之间的间隔，可能会很有用。
显然，这可以通过增大 `autovacuum_freeze_max_age` 或减小 `vacuum_freeze_min_age` 来做到。

`vacuum_freeze_table_age` 的实际最大值是 0.95 * `autovacuum_freeze_max_age`；高于该值的设置会被截断到该最大值。设置一个高于 `autovacuum_freeze_max_age` 的值没有意义，因为无论如何，防回卷自动清理都会在那时被触发，而 0.95 的乘数是为了在那之前留出一些余地以手工运行一次 `VACUUM`。经验上，`vacuum_freeze_table_age` 应设置为略低于 `autovacuum_freeze_max_age` 的值，并留出足够空档，使一次常规调度的 `VACUUM` 或由正常删除、更新活动触发的自动清理能在这个窗口内运行。若设置得过于接近，即使该表最近已经为了回收空间而清理过，也可能仍会触发防回卷自动清理；
而较低的值则会导致更频繁的全表扫描。

增大 `autovacuum_freeze_max_age`（以及相应的 `vacuum_freeze_table_age`）唯一的缺点，是数据库集簇的 `pg_clog` 子目录会占用更多空间，因为必须保存追溯到 `autovacuum_freeze_max_age` 视界的所有事务的提交状态。每个事务的提交状态占用两位，因此，如果将 `autovacuum_freeze_max_age` 设为允许的最大值 20 亿，`pg_clog` 预计会增长到约 0.5GB。如果这与数据库总大小相比微不足道，建议将 `autovacuum_freeze_max_age` 设为允许的最大值。否则，应根据愿意为 `pg_clog` 分配的存储空间来设置。（默认值为 2 亿个事务，对应约 50MB 的 `pg_clog` 存储。）

减小 vacuum_freeze_min_age 的一个缺点是，它可能导致 VACUUM 做无用功：如果某个行版本随后不久就被修改（因而获得新的 XID），把表行的 XID 改为 FrozenXID 就是浪费时间。另一个缺点是，关于究竟是哪个事务插入或修改了某一行的细节会更早丢失。这些信息有时很有用，特别是在数据库故障后试图分析哪里出了问题时。
由于这两个原因，除了完全静态的表之外，不建议减小该设置。

为了跟踪数据库中最老 XID 的年龄，VACUUM 会在系统表 pg_class 和 pg_database 中存储 XID 统计信息。具体来说，某个表在 pg_class 中对应行的 relfrozenxid 列，记录了 VACUUM 上次扫描全表时所使用的冻结截止 XID。所有比该截止 XID 更老的普通 XID 都保证已在表内被替换为 FrozenXID。类似地，某个数据库在 pg_database 中对应行的 datfrozenxid 列，是出现在该数据库中的普通 XID 的下界，它就是该数据库内各表 relfrozenxid 值的最小值。查看这些信息的一种方便方式是执行如下查询：

```
SELECT c.oid::regclass as table_name,
       greatest(age(c.relfrozenxid),age(t.relfrozenxid)) as age
FROM pg_class c
LEFT JOIN pg_class t ON c.reltoastrelid = t.oid
WHERE c.relkind = 'r';

SELECT datname, age(datfrozenxid) FROM pg_database;
```

age 列度量的是从该截止 XID 到当前事务 XID 之间的事务数。

VACUUM通常只扫描自上次清理以来被修改过的页面，但只有当扫描整个表时，relfrozenxid 才会被推进。当 relfrozenxid的年龄超过vacuum_freeze_table_age个事务、使用了 VACUUM 的 FREEZE 选项，或者所有页面碰巧都需要清理以移除死行版本时，就会扫描整个表。当 VACUUM 扫描整个表时，它应把 age(relfrozenxid) 设为略高于所用 vacuum_freeze_min_age 设置的值（更高的部分等于自 VACUUM 开始以来已启动的事务数）。如果直到达到 autovacuum_freeze_max_age 之前，该表都没有执行一次全表扫描的 VACUUM，那么很快就会被强制执行一次自动清理。

如果由于某种原因，自动清理未能清除表中的旧 XID，当数据库中最老的 XID 距离回卷点只剩 1000 万个事务时，系统就会开始发出类似以下的警告：

```
WARNING:  database "mydb" must be vacuumed within 177009986
 transactions
HINT:  To avoid a database shutdown, execute a database-wide VACUUM
 in "mydb".
```

（按照提示，手动执行 VACUUM 应能解决问题；但请注意，VACUUM 必须由超级用户执行，否则无法处理系统目录，也就无法推进数据库的 datfrozenxid。）如果忽略这些警告，一旦距离回卷只剩不到 100 万个事务，系统就会关闭并拒绝启动任何新事务：

```
ERROR:  database is not accepting commands to avoid wraparound data
 loss in database "mydb"
HINT:  Stop the postmaster and use a standalone backend to VACUUM
 in "mydb".
```

预留这 100 万个事务的安全余量，是为了让管理员可以手动执行必需的 VACUUM 命令，在不丢失数据的情况下恢复。不过，系统一旦进入安全关闭模式，就不会再执行命令，因此唯一的办法是停止服务器，并使用一个单用户后端执行 VACUUM。单用户后端不会强制进入关闭模式。有关使用单用户后端的详细信息，参见postgres参考页面。

23.1.6. 自动清理守护进程

PostgreSQL 具有一个可选但强烈推荐的特性，称为自动清理，其目的是自动执行 VACUUM 和 ANALYZE 命令。启用后，自动清理会检查那些已经积累了大量插入、更新或删除元组的表。这些检查依赖于统计收集功能；因此，除非track_counts被设置为 true，否则无法使用自动清理。在默认配置下，自动清理是启用的，并且相关配置参数也设置得比较合适。

所谓“自动清理守护进程”实际上由多个进程组成。其中有一个持久运行的守护进程，称为自动清理启动器，负责为所有数据库启动自动清理工作进程。启动器会把工作分散在时间上，力图在每个数据库中每隔autovacuum_naptime秒启动一个工作进程。（因此，如果安装中有 *N* 个数据库，就会每隔 `autovacuum_naptime/N` 秒启动一个新的工作进程。）同时最多允许运行autovacuum_max_workers个工作进程。如果需要处理的数据库数量多于 `autovacuum_max_workers`，那么一旦第一个工作进程结束，就会处理下一个数据库。每个工作进程会检查其数据库中的每个表，并根据需要执行 `VACUUM` 和/或 `ANALYZE`。

如果多个大型表在很短时间内都变得需要清理，那么所有自动清理工作进程都可能会长期忙于清理这些表。这会导致其他表和数据库在某个工作进程空闲下来之前都得不到清理。单个数据库中的工作进程数量没有上限，但工作进程会尽量避免重复其他工作进程已经完成的工作。注意，正在运行的工作进程数量不计入max_connections或superuser_reserved_connections的限制。

凡是 `relfrozenxid` 值的年龄超过autovacuum_freeze_max_age个事务的表，始终都会被清理（这也适用于通过存储参数修改了最大冻结年龄的表，见下文）。否则，如果自上次 `VACUUM` 以来失效的元组数超过“清理阈值”，就会清理该表。清理阈值定义为：

```
vacuum threshold = vacuum base threshold + vacuum scale factor *
 number of tuples
```

其中，清理基础阈值为autovacuum_vacuum_threshold，清理比例因子为autovacuum_vacuum_scale_factor，元组数为 `pg_class.reltuples`。失效元组数来自统计收集器；这是一个近似计数，由每次 `UPDATE` 和 `DELETE` 操作更新。（之所以只是近似值，是因为高负载下可能丢失部分信息。）如果表的 `relfrozenxid` 值的年龄超过 `vacuum_freeze_table_age` 个事务，就会扫描整个表以冻结旧元组并推进 `relfrozenxid`；否则，只扫描自上次清理以来被修改过的页面。

对于分析操作，也使用了一个类似的条件：其阈值定义如下：

```
analyze threshold = analyze base threshold + analyze scale factor *
 number of tuples
```

该阈值会与自上次 `ANALYZE` 以来插入、更新或删除的元组总数进行比较。

自动清理无法访问临时表。因此，应通过会话 SQL 命令执行适当的清理和分析操作。

默认阈值和尺度因子取自 `postgresql.conf`，但也可以按表覆盖它们；详情见Storage Parameters。如果某个设置已经通过存储参数修改，则使用该值；否则使用全局设置。关于全局设置的更多细节，见第 18.10 节。

除了基础阈值和比例因子之外，还有六个可以通过存储参数为每张表设置的自动清理参数。第一个参数 `autovacuum_enabled` 可以设为 `false`，以指示自动清理后台进程完全跳过该表。在这种情况下，只有为防止事务 ID 回卷所必需时，自动清理才会处理该表。另外两个参数 `autovacuum_vacuum_cost_delay` 和 `autovacuum_vacuum_cost_limit`，用于为基于代价的清理延迟特性设置表级值（见第 18.4.4 节）。`autovacuum_freeze_min_age`、`autovacuum_freeze_max_age` 和 `autovacuum_freeze_table_age` 分别用于设置vacuum_freeze_min_age、autovacuum_freeze_max_age和vacuum_freeze_table_age的值。

当多个工作进程同时运行时，代价延迟参数会在所有运行中的工作进程之间“平衡”，这样无论实际运行了多少个工作进程，系统承受的总 I/O 影响都相同。不过，任何正在处理那些为每表存储参数 `autovacuum_vacuum_cost_delay` 或 `autovacuum_vacuum_cost_limit` 显式设置过值的表的工作进程，都不会被纳入这个均衡算法。

23.2. 日常重建索引

在某些情况下，值得周期性地使用REINDEX命令或一系列单独的重建步骤，来重建索引。

B-树索引中已经完全变空的索引页会被回收以供重用。不过，
仍然可能出现空间利用低效：如果某个页面上的索引键除了少数几个之外都被删除，
该页面仍会保留分配。因此，在一种使用模式下，
如果每个键范围中的大多数但不是全部键最终都被删除，就会出现糟糕的空间利用。
对于这种使用模式，建议定期重建索引。

对于非 B-树索引中的膨胀潜力，人们还没有做过充分研究。在使用任何非 B-树索引类型时，定期监控索引的物理大小是个好主意。

此外，对 B-树索引而言，新近构建的索引比已经多次更新过的索引在访问时稍快一些，因为在新建的索引中，逻辑上相邻的页面通常在物理上也彼此相邻。（这一考虑不适用于非 B-树索引。）即便只是为了提高访问速度，周期性重建索引也可能是值得的。

23.3. 日志文件维护

最好把数据库服务器的日志输出保存到某个地方，而不是简单地把它丢弃到 `/dev/null`。诊断问题时，日志输出极其宝贵。但是，
日志输出往往十分庞大（尤其是在较高调试级别下），因此你不会希望无限期保存它。
你需要对日志文件进行轮转，以便周期性地启用新的日志文件，
并在合理时间后移除旧文件。

如果你只是把 `postgres` 的 stderr 重定向到一个文件，那么虽然也能得到日志输出，但截断该日志文件的唯一办法是停止并重新启动服务器。如果你是在开发环境中使用 PostgreSQL，这也许可以接受，但几乎没有生产服务器会认为这种行为可以接受。

更好的办法是把服务器的 stderr 输出发送给某种日志轮转程序。系统内置了日志轮转功能，你可以在 `postgresql.conf` 中将配置参数 `logging_collector` 设为 `true` 来使用它。该程序的控制参数见第 18.8.1 节。你还可以用这种方式以机器可读的 CSV（逗号分隔值）格式捕获日志数据。

也可以使用外部日志轮转程序，特别是已经在其他服务器软件上使用了某个此类程序时。
例如，rotatelogs 工具包含在 Apache 发行版中，也可用于 PostgreSQL。为此，只需通过管道将服务器的 stderr 输出传递给所需程序。如果使用 `pg_ctl` 启动服务器，那么 stderr 已经被重定向到 stdout，因此只需一个管道命令，例如：

```
pg_ctl start | rotatelogs /var/log/pgsql_log 86400
```

管理日志输出的另一种生产级方法，是把它发送给 syslog，并让 syslog 负责文件轮转。为此，在 `postgresql.conf` 中把配置参数 `log_destination` 设为 `syslog`（如果只想记录到 syslog）。之后，每当你想强制它开始写入新的日志文件时，都可以向 syslog 守护进程发送一个 `SIGHUP` 信号。如果你想自动化日志轮转，也可以把 logrotate 配置为处理来自 syslog 的日志文件。

不过，在很多系统上，syslog 并不十分可靠，尤其是在日志消息很大时；它可能恰恰在你最需要消息的时候截断或丢弃它们。此外，在 Linux 上，syslog 会把每条消息都刷盘，导致性能较差。（你可以在 syslog 配置文件的文件名开头使用一个“-”来禁用同步。）

请注意，上面描述的所有方案都处理了按可配置间隔启动新日志文件的问题，
但它们并不处理删除那些已经没有用处的旧日志文件。
你很可能还需要设置一个批处理任务，定期删除旧日志文件。
另一种可能是把轮转程序配置为循环覆盖旧日志文件。

pgFouine[2] 是一个进行复杂日志文件分析的外部项目。check_postgres[3] 可以在日志文件中出现重要消息时向 Nagios 发出警报，也能检测许多其他异常情况。

[2] http://pgfouine.projects.postgresql.org/
[3] https://bucardo.org/check_postgres/

第 24 章 备份和恢复

与任何保存重要数据的系统一样，PostgreSQL数据库也应定期备份。虽然其过程基本简单，但清楚理解其底层技术和前提假设非常重要。

备份PostgreSQL数据主要有三种根本不同的方法：

- SQL转储
- 文件系统级备份
- 持续归档

每种方法都有各自的优缺点，下面各节将依次讨论。

24.1. SQL转储

这种转储方法的思路是生成一个包含 SQL 命令的文件；将其重新送回服务器后，服务器就会把数据库重建为转储时的状态。PostgreSQL为此提供了工具程序pg_dump。
该命令的基本用法是：

```
pg_dump dbname > outfile
```

如前所示，pg_dump把结果写到标准输出。稍后我们会看到这一点有何用处。

pg_dump是一个普通的PostgreSQL客户端应用程序（尽管它实现得相当巧妙）。这意味着你可以从任何能够访问该数据库的远程主机执行这种备份过程。但请记住，pg_dump并不会以特殊权限运行。特别是，它必须对你想要备份的所有表具有读权限，因此若要备份整个数据库，几乎总是必须以数据库超级用户身份运行它。

要指定pg_dump应连接哪个数据库服务器，请使用命令行选项`-h host`和`-p port`。默认主机是本地主机，或者由环境变量`PGHOST`指定的主机。类似地，默认端口由环境变量`PGPORT`指定；如果未设置，则采用编译时内置的默认值。（方便的是，服务器通常也会使用同样的内置默认值。）

和其他PostgreSQL客户端应用程序一样，pg_dump默认会以与当前操作系统用户名相同的数据库用户名进行连接。若要覆盖这一点，可以指定`-U`选项，或者设置环境变量`PGUSER`。请记住，pg_dump连接同样受常规客户端认证机制约束（见第 19 章）。

与后文介绍的其他备份方法相比，pg_dump的一个重要优点在于，pg_dump的输出通常可以重新装入较新版本的PostgreSQL，而文件级备份和持续归档都高度依赖具体的服务器版本。pg_dump也是在不同机器体系结构之间迁移数据库时唯一可行的方法，例如从 32 位服务器迁移到 64 位服务器。

由pg_dump创建的转储在内部是一致的，也就是说，该转储表示的是pg_dump开始运行时的数据库快照。pg_dump在工作期间不会阻塞数据库上的其他操作。（例外是那些需要独占锁的操作，例如大多数形式的`ALTER TABLE`。）

> **重要**
>
> 如果你的数据库模式依赖 OID（例如作为外键），必须让pg_dump把 OID 一并转储出来。为此，请使用命令行选项`-o`。

24.1.1. 恢复转储

由 pg_dump 创建的文本文件供 psql 程序读取。恢复转储的通用命令格式是

```
psql dbname < infile
```

其中，*infile* 是由 pg_dump 命令输出的文件。数据库 *dbname* 不会由此命令创建，因此必须自行基于 `template0` 创建它，再执行 psql（例如使用 `createdb -T template0 dbname`）。psql 支持与 pg_dump 类似的选项，用于指定要连接的数据库服务器及所用的用户名。更多信息参见psql参考页面。

在恢复 SQL 转储之前，所有拥有对象或者在被转储数据库中曾被授予对象权限的用户都必须已经存在。否则，恢复将无法以原有所有权和/或权限重新创建这些对象。
（有时这正是你想要的，但通常不是。）

默认情况下，psql脚本在遇到 SQL 错误后仍会继续执行。你可能希望让psql在设置了`ON_ERROR_STOP`变量的情况下运行，以改变这一行为，并让psql在发生 SQL 错误时以退出状态码 3 退出：

```
psql --set ON_ERROR_STOP=on dbname < infile
```

无论采用上述哪种方式，你最终只会得到一个部分恢复的数据库。
另一种做法是指定将整个转储作为单个事务恢复，这样恢复要么全部完成，要么全部回滚。
可以通过把`-1`或`--single-transaction`命令行选项传给psql来启用这种模式。
使用这种模式时要注意，即便是一个很小的错误，
也可能回滚一个已经运行了许多小时的恢复过程。不过，
这仍可能比在部分恢复后手工清理一个复杂数据库更可取。

pg_dump和psql能够写入或读取管道，
这使得可以把数据库直接从一台服务器转储到另一台服务器，例如：

```
pg_dump -h host1 dbname | psql -h host2 dbname
```

> **重要**
>
> pg_dump生成的转储是以`template0`为基准的。这意味着通过`template1`增加的任何语言、过程等也都会被pg_dump转储。因此，
> 恢复时如果你使用的是定制过的`template1`，就必须像上面的例子那样，从`template0`创建空数据库。

恢复完备份后，最好对每个数据库运行ANALYZE，
这样查询优化器就能拥有有用的统计信息；更多信息见第 23.1.3 节和第 23.1.6 节。关于如何高效地将大量数据装入PostgreSQL，请参见第 14.4 节。

24.1.2. 使用pg_dumpall

pg_dump一次只转储一个数据库，
而且不会转储角色或表空间的信息（因为这些是整个集簇级别的，
而不是每个数据库各自的）。为了方便地转储整个数据库集簇的全部内容，提供了pg_dumpall程序。pg_dumpall会备份给定集簇中的每个数据库，
并保留角色定义和表空间定义等集簇范围的数据。该命令的基本用法是：

```
pg_dumpall > outfile
```

生成的转储可以用psql恢复：

```
psql -f infile postgres
```

（实际上，可以指定任意一个现有数据库名作为起点，但如果要装载到一个空集簇中，通常应使用postgres。）恢复pg_dumpall转储时始终需要数据库超级用户权限，因为恢复角色和表空间信息必须使用该权限。如果使用了表空间，请确保转储中的表空间路径适合新的安装。

pg_dumpall的工作方式是先发出重新创建角色、表空间和空数据库的命令，然后针对每个数据库调用pg_dump。这意味着每个数据库本身都是内部一致的，但不同数据库的快照并不同步。

24.1.3. 处理大型数据库

某些操作系统对文件大小有上限，这会在创建大型pg_dump输出文件时带来问题。幸运的是，pg_dump可以把输出写到标准输出，因此你可以利用标准 Unix 工具绕过这个潜在问题。有几种可行方法：

使用压缩转储。. 你可以使用自己喜欢的压缩程序，例如gzip：

```
pg_dump dbname | gzip > filename.gz
```

恢复时：

```
gunzip -c filename.gz | psql dbname
```

或者：

```
cat filename.gz | gunzip | psql dbname
```

使用split。. split命令允许你把输出分割成文件系统可接受大小的较小文件。例如，要切分为 1 MB 的块：

```
pg_dump dbname | split -b 1m - filename
```

恢复时：

```
cat filename* | psql dbname
```

使用pg_dump的自定义转储格式。. 如果构建PostgreSQL所用的系统安装了zlib压缩库，自定义转储格式会在向输出文件写入数据时对其进行压缩。这会产生与使用gzip大致相同大小的转储文件，但额外的优点是其中的表可以有选择地恢复。下面的命令使用自定义转储格式转储数据库：

```
pg_dump -Fc dbname > filename
```

自定义格式的转储不是供psql执行的脚本，而必须通过pg_restore恢复，例如：

```
pg_restore -d dbname filename
```

详细信息见pg_dump和pg_restore参考页。

对于非常大的数据库，你可能需要把split与另外两种方法之一结合使用。

24.2. 文件系统级备份

另一种备份策略是直接复制PostgreSQL用来存储数据库数据的文件；第 17.2 节解释了这些文件的位置。你可以使用自己喜欢的任何方法进行文件系统备份，例如：

```
tar -cf backup.tar /usr/local/pgsql/data
```

不过，这种方法有两个限制，使得它并不实用，至少也不如pg_dump方法：

1. 为了得到可用的备份，数据库服务器必须关闭。
   像禁止所有连接这样的折中办法不起作用（部分原因是tar

和类似工具不会对文件系统状态执行原子快照，
另一部分原因是服务器内部也存在缓冲）。有关停止服务器的信息见第 17.5 节。不言而喻，恢复数据前也必须关闭服务器。

2. 如果你已经深入了解数据库文件系统布局的细节，
可能会想尝试仅通过某些表或数据库各自的文件或目录来备份或恢复它们。这是
行不通的，因为这些文件中的信息离开记录所有事务提交状态的提交日志文件`pg_clog/*`就无法使用。一个表文件只有和这些信息一起才可用。当然，
也不可能只恢复一个表以及相关的`pg_clog`数据，
因为那会让数据库集簇中的其他所有表都无法使用。因此，
文件系统备份只适用于整个数据库集簇的完整备份与恢复。

另一种文件系统备份方法是对数据目录制作一个“一致快照”，
前提是文件系统支持该功能（并且你愿意相信其实现是正确的）。
典型流程是：对包含数据库的卷制作一个“冻结快照”，
然后从该快照把整个数据目录（不是其中一部分，见上文）复制到备份设备，
再释放这个冻结快照。即使数据库服务器正在运行，这种方法也能工作。不过，
以这种方式创建的备份会把数据库文件保存为一种仿佛服务器没有被正确关闭的状态；
因此，当你基于备份数据启动数据库服务器时，它会认为前一个服务器实例发生了崩溃，
并重放 WAL 日志。这不是问题；只要意识到这一点即可（并且务必把 WAL 文件也包含在备份中）。你可以在获取快照之前执行一次`CHECKPOINT`，以缩短恢复时间。

如果数据库分布在多个文件系统上，就可能根本无法对所有卷获得完全同步的冻结快照。
例如，如果数据文件和 WAL 日志位于不同磁盘，或者表空间位于不同文件系统上，就可能无法使用快照备份，因为这些快照必须同时获取。在这种情况下，在信任一致快照技术之前，
一定要非常仔细地阅读文件系统文档。

如果无法获得同时的快照，一种选择是将数据库服务器关闭足够长的时间，
以建立所有冻结快照。另一种选择是执行持续归档基础备份（第 24.3.2 节），因为这种备份不受备份期间文件系统变化的影响。这种方法只需在备份期间启用持续归档；
恢复则使用持续归档恢复（第 24.3.4 节）。

另一种选择是使用rsync进行文件系统备份。
做法是先在数据库服务器运行时执行一次rsync，然后将数据库服务器只关闭足够再做一次rsync的时间。第二次rsync会比第一次快得多，因为它需要传输的数据相对较少，
而最终结果会因为服务器已经关闭而保持一致。
这种方法可以在最小停机时间下完成文件系统备份。

注意，文件系统备份通常会比 SQL 转储更大。（例如，pg_dump不需要转储索引内容，只需转储重建索引的命令。）不过，进行文件系统备份可能更快。

24.3. 持续归档和时间点恢复（PITR）

在任何时候，PostgreSQL都会维护一个预写式日志（WAL），它位于集簇数据目录的`pg_xlog/`子目录中。该日志记录对数据库数据文件所做的每一次修改。
这个日志首先是为崩溃安全而存在：如果系统崩溃，可以通过“重放”
自上次检查点以来的日志记录，将数据库恢复到一致状态。不过，
日志的存在也使第三种数据库备份策略成为可能：我们可以把文件系统级备份与 WAL 文件备份结合起来。如果需要恢复，就先恢复文件系统备份，再重放已备份的 WAL 文件，使系统回到当前状态。与前两种方法相比，这种方法管理起来更复杂，但它有一些显著优点：

- 我们不需要一个完全一致的文件系统备份作为起点。
  备份中的任何内部不一致性都会通过日志重放纠正（这与崩溃恢复期间发生的事情并没有本质不同）。因此，我们不需要文件系统快照能力，只需要tar或类似的归档工具。

- 由于我们可以把任意长的一串 WAL 文件串联起来进行重放，只要持续归档 WAL 文件，就可以实现连续备份。这对于大型数据库尤其有价值，
  因为此时频繁进行完整备份可能并不方便。

- 不必把 WAL 记录一直重放到最后。我们可以在任意位置停止重放，并得到数据库在当时的一致快照。因此，这项技术支持
  时间点恢复：可以把数据库恢复到自基础备份之后任意时刻的状态。

- 如果我们持续把这一串 WAL 文件输送给另一台已经装载了同一基础备份文件的机器，就得到了一个温备系统：我们随时都可以启用第二台机器，
  而它将拥有一个几乎是最新的数据库副本。

> **注意**
>
> pg_dump和pg_dumpall不会生成文件系统级备份，
> 因此不能作为持续归档方案的一部分使用。这类转储是逻辑备份，不包含 WAL 重放所需的足够信息。

和普通文件系统备份技术一样，这种方法只能支持整个数据库集簇的恢复，
而不支持其中某个子集的恢复。此外，它需要大量归档存储：基础备份可能很庞大，
繁忙系统也会产生许多兆字节的、必须归档的 WAL 流量。尽管如此，在很多需要高可靠性的场景中，它仍是首选的备份技术。

要想利用持续归档（许多数据库厂商也称之为“在线备份”）成功恢复，
你需要一串连续的已归档 WAL 文件，其时间至少要追溯到备份的开始时刻。因此，入门时应在进行第一次基础备份之前就建立并测试好归档 WAL 文件的流程。下面先讨论归档 WAL 文件的机制。

24.3.1. 设置 WAL 归档

从抽象角度看，运行中的 PostgreSQL 系统会产生一个无限延伸的 WAL 记录序列。在物理存储上，系统将该序列分为 WAL 段文件，通常每个为 16MB（但可以在构建 PostgreSQL 时修改段大小）。段文件使用数字名称，反映其在抽象 WAL 序列中的位置。不使用 WAL 归档时，系统通常只创建少量段文件，然后通过将不再需要的段文件重命名为更大的段号来“回收”它们。系统假定，内容早于倒数第二个检查点的段文件已不再需要，可以回收。

在归档 WAL 数据时，我们需要在每个段文件写满后捕获其内容，并在该段文件被回收重用之前把数据保存到某处。根据应用场景和可用硬件的不同，“把数据保存到某处”
可以有很多不同的方法：可以把段文件复制到另一台机器上的 NFS 挂载目录，把它们写到磁带机中（确保你有办法识别每个文件的原始文件名），把它们批量打包后刻录到 CD 上，或者采用完全不同的方式。为了给数据库管理员提供灵活性，
PostgreSQL尽量不对归档方式作任何假设。相反，PostgreSQL允许管理员指定一个 shell 命令，以便把已完成的段文件复制到它应去的地方。该命令可以只是一次简单的`cp`调用，
也可以调用一个复杂的 shell 脚本，全由你决定。

要启用 WAL 归档，请将wal_level配置参数设为 `archive`（或 `hot_standby`），将archive_mode设为 `on`，并在archive_command配置参数中指定要执行的 shell 命令。实际使用中，这些设置始终放在 `postgresql.conf` 文件中。对于 `archive_command`，`%p` 会被替换为要归档文件的路径名，而 `%f` 只会被替换为文件名。（路径名相对于当前工作目录，即集簇的数据目录。）使用 `%%` 可以在命令中嵌入实际的 `%` 字符。最简单的可用命令类似于：

```
archive_command = 'test ! -f /mnt/server/archivedir/%f && cp %p /
mnt/server/archivedir/%f'  # Unix
archive_command = 'copy "%p" "C:\\server\\archivedir\\%f"'  #
 Windows
```

这会将可归档的 WAL 段复制到目录 `/mnt/server/archivedir`。（这只是示例，并非推荐做法，而且不一定适用于所有平台。）在替换 `%p` 和 `%f` 参数后，实际执行的命令可能如下：

```
test ! -f /mnt/server/archivedir/00000001000000A900000065 &&
 cp pg_xlog/00000001000000A900000065 /mnt/server/archivedir/
00000001000000A900000065
```

每个需要归档的新文件都会生成一条类似的命令。

归档命令将以运行 PostgreSQL 服务器的同一用户身份执行。由于归档的一系列 WAL 文件实际上包含数据库中的全部内容，应确保归档数据不会被他人窥视；例如，将其归档到不允许所属组或其他用户读取的目录。

归档命令必须当且仅当成功时才返回退出状态零。收到零状态后，PostgreSQL 会认为该文件已成功归档，并将其删除或回收。非零状态则告诉 PostgreSQL 该文件尚未归档；它会定期重试，直到成功。

通常应将归档命令设计为拒绝覆盖任何已存在的归档文件。这是一项重要的安全保护措施，能在管理员操作失误时（例如将两台不同服务器的输出发送到同一归档目录）维护归档的完整性。

建议测试所拟定的归档命令，确保它确实不会覆盖已有文件，并且在这种情况下返回非零状态。上面给出的 Unix 示例命令通过单独加入一个`test`步骤，确保这两点。在某些 Unix 平台上，`cp`提供了诸如`-i`之类的开关，也可以更简洁地实现同样目的，但在你确认它会返回正确的退出状态之前，不应依赖这些开关。（尤其是 GNU `cp`在使用`-i`且目标文件已存在时会返回状态零，这不是我们想要的行为。）

在设计归档方案时，请考虑如果归档命令因为某些环节需要操作员干预，或者归档空间耗尽而反复失败，会发生什么。例如，如果你在没有自动换带器的情况下向磁带写入数据，那么磁带满了以后，在更换磁带之前将无法继续归档。你应确保任何错误情况或对人工操作员的请求都能得到适当报告，以便问题能够比较快地解决。在问题解决之前，`pg_xlog/`目录会继续堆积 WAL 段文件。（如果包含`pg_xlog/`的文件系统被写满，PostgreSQL将执行 PANIC 关闭。不会丢失已提交的事务，但在释放出一些空间之前，数据库会一直离线。）

归档命令的速度并不重要，只要它能跟上服务器生成 WAL 数据的平均速度即可。即使归档过程稍有滞后，正常操作也会继续。如果归档明显落后，灾难发生时可能丢失的数据量就会增加。这还意味着`pg_xlog/`目录会包含大量尚未归档的段文件，最终可能耗尽可用磁盘空间。建议监控归档过程，确保它按你的预期工作。

在编写归档命令时，应假定待归档文件名最长可达 64 个字符，并且可以包含 ASCII 字母、数字和点号的任意组合。无需保留原始相对路径（`%p`），但必须保留文件名（`%f`）。

注意，虽然 WAL 归档允许你恢复对PostgreSQL数据库中数据所做的任何修改，但它不会恢复对配置文件（即`postgresql.conf`、`pg_hba.conf`和`pg_ident.conf`）的修改，因为这些文件是手工编辑的，而不是通过 SQL 操作修改的。你可能希望把配置文件放在常规文件系统备份过程能够覆盖的位置。关于如何重定位配置文件，见第 18.2 节。

归档命令只会针对已完成的 WAL 段调用。因此，如果服务器产生的 WAL 流量很小（或者存在低谷期），事务完成到其被安全写入归档存储之间可能会有很长延迟。为了限制未归档数据可能有多旧，你可以设置archive_timeout，使服务器强制切换到新 WAL 段文件的间隔不超过这个值。注意，由强制切换而提前归档的文件长度仍与装满的文件相同。因此，把`archive_timeout`设得很短并不明智，这会使归档存储膨胀。`archive_timeout`设为大约 1 分钟通常是合理的。

此外，如果你希望确保一个刚刚完成的事务尽快被归档，可以使用`pg_switch_xlog`手工强制一次段切换。其他与 WAL 管理相关的实用函数列在表 9.60中。

当`wal_level`为`minimal`时，某些 SQL 命令会像第 14.4.7 节中所述那样被优化为避免 WAL 记录。如果在执行这些语句期间启用了归档或流复制，WAL 将不包含归档恢复所需的足够信息。（崩溃恢复不受影响。）因此，`wal_level`只能在服务器启动时更改。然而，

archive_command可以通过重新加载配置文件来更改。如果你希望暂时停止归档，一种办法是将archive_command设置为空字符串（''）。这会导致 WAL 文件在pg_xlog/中累积，直到重新建立可用的archive_command。

24.3.2. 进行基础备份

执行基础备份最简单的方法是使用pg_basebackup工具。它可以把基础备份创建为普通文件或 tar 归档。如果需要比pg_basebackup提供的更高灵活性，也可以使用低级 API 制作基础备份（见第 24.3.3 节）。

不必过于担心制作基础备份所需的时间。不过，如果你平时在关闭full_page_writes的情况下运行服务器，可能会注意到备份运行期间性能下降，因为在备份模式下full_page_writes实际上会被强制开启。

要让该备份可用，你需要保留在文件系统备份期间以及之后生成的所有 WAL 段文件。为帮助完成这件事，基础备份过程会创建一个备份历史文件，并立即将其存入 WAL 归档区域。该文件以文件系统备份所需的第一个 WAL 段文件命名。例如，如果起始 WAL 文件是000000010000123400005 5CD，备份历史文件的名称将类似于000000010000123400005 5CD.007C9330.backup。（文件名第二部分表示该 WAL 文件中的一个精确位置，通常可以忽略。）一旦你已经安全归档了文件系统备份以及备份期间使用到的 WAL 段文件（如备份历史文件所指定），所有名称在数值上更小的已归档 WAL 段就不再是恢复该文件系统备份所必需的，可以删除。不过，你仍应考虑保留多个备份集，以绝对确保能够恢复数据。

备份历史文件只是一个很小的文本文件。它包含你提供给pg_basebackup的标签字符串，以及备份的起止时间和起止 WAL 段。如果你用该标签标识了关联的备份文件，那么已归档的历史文件就足以告诉你应恢复哪个备份文件。

由于你必须保留自上一次基础备份以来的所有已归档 WAL 文件，基础备份之间的间隔通常应根据你愿意为已归档 WAL 文件投入多少存储空间来决定。你还应考虑在确实需要恢复时，你愿意花多长时间进行恢复，因为系统必须重放所有这些 WAL 段；如果距离上次基础备份已经过去很久，这可能会花费一些时间。

24.3.3. 使用低级 API 进行基础备份

使用低级 API 制作基础备份的过程，比pg_basebackup方法多几个步骤，但相对简单。务必按顺序执行这些步骤，并在继续下一步之前确认当前步骤成功。

1. 确保 WAL 归档已启用并正常工作。

2. 以超级用户身份连接到数据库，并执行命令：

   ```
   SELECT pg_start_backup('label');
   ```

   其中，label 是用于唯一标识此次备份操作的任意字符串。（一种好做法是使用你打算存放备份转储文件的完整路径。）pg_start_backup 会创建一个备份标签文件，名为backup_label，位于集簇目录中，包含备份信息，例如开始时间和标签字符串。

   连接到集簇中的哪个数据库来执行该命令都无关紧要。可以忽略该函数返回的结果；但如果它报告错误，应先处理后再继续。

   默认情况下，pg_start_backup 可能需要很长时间才能完成。这是因为它会执行一次检查点，而检查点所需的I/O会分散在较长时间内，默认是检查点间隔的一半（参见配置参数checkpoint_completion_target）。这通常是理想的行为，因为它尽量减少了对查询处理的影响。如果希望尽快开始备份，请使用：

   ```
   SELECT pg_start_backup('label', true);
   ```

   这会强制尽快完成检查点。

3. 使用任何方便的文件系统备份工具执行备份，例如tar或cpio（不要使用pg_dump或pg_dumpall）。在此过程中既没有必要，也不希望停止数据库的正常运行。

4. 再次以超级用户身份连接到数据库，并执行命令：

   ```
   SELECT pg_stop_backup();
   ```

   这会终止备份模式，并自动切换到下一个 WAL 段。切换的目的是让备份期间写入的最后一个 WAL 段文件准备好归档。

5. 备份期间处于活动状态的 WAL 段文件全部归档后，备份就完成了。pg_stop_backup 返回结果所标识的文件，是构成完整备份文件集所需的最后一个段。如果启用了 archive_mode，pg_stop_backup 会等到最后一个段归档后才返回。由于已经配置了 archive_command，这些文件会自动归档。大多数情况下，归档很快就能完成，但建议监控归档系统，确保没有延迟。如果归档进程因归档命令失败而落后，它会不断重试，直到归档成功、备份完成。如果希望限制 pg_stop_backup 的执行时间，可以设置合适的 statement_timeout 值。

某些文件系统备份工具在复制过程中，如果它们试图复制的文件发生变化，就会发出警告或错误。对活动数据库进行基础备份时，这种情况是正常的，并不表示出错。不过，你需要确保能够把这类提示与真正的错误区分开来。例如，某些版本的rsync会针对“vanished source files”返回单独的退出码，你可以编写一个驱动脚本，把这一退出码视为非错误情况。此外，某些版本的 GNU tar在文件被tar复制时如果发生截断，会返回与致命错误无法区分的错误码。幸运的是，GNU tar 1.16 及之后版本在备份过程中如果文件被更改会以 1 退出，而其他错误则以 2 退出。对于 GNU tar 1.23 及之后版本，可以使用警告选项--warning=no-file-changed --warning=no-file-removed隐藏相关警告信息。

务必确认你的备份包含数据库集簇目录（例如/usr/local/pgsql/data）下的全部文件。如果你使用的表空间不位于该目录之下，也要记得把它们包括进来（并确保备份把符号链接作为链接归档，否则恢复时会破坏表空间）。

不过，备份中应省略集簇pg_xlog/子目录内的文件。这个小调整很值得，因为它能降低恢复时出错的风险。如果pg_xlog/是一个指向集簇目录外某处的符号链接，就很容易做到这一点，而出于性能原因，这本来也是一种常见配置。你也可能想排除postmaster.pid和postmaster.opts，它们记录的是正在运行的postmaster的信息，而不是最终使用此备份的postmaster的信息。（这些文件可能会让pg_ctl感到困惑。）

另外值得一提的是，pg_start_backup函数会在数据库集簇目录中创建一个名为backup_label的文件，该文件会被pg_stop_backup删除。该文件当然也会作为备份转储文件的一部分被归档。备份标签文件包含你提供给pg_start_backup的标签字符串、执行pg_start_backup的时间以及起始 WAL 文件的名称。因此，在发生混淆时，可以查看备份文件内容，精确确定该备份文件来自哪一次备份会话。

服务器停止时也可以进行备份。在这种情况下，你显然无法使用pg_start_backup或pg_stop_backup，因此只能自己追踪各个备份的身份以及相关 WAL 文件最早需要追溯到哪里。通常最好还是遵循上面的持续归档过程。

24.3.4. 使用持续归档备份进行恢复

现在，最坏的情况发生了，你需要通过备份进行恢复。步骤如下：

1. 如果服务器仍在运行，就先停止它。

2. 如果你有足够空间，请把整个集簇数据目录以及所有表空间复制到临时位置，以备后续需要。注意，这一预防措施要求系统有足够空闲空间来保存现有数据库的两份副本。如果空间不够，

至少也应保存集簇pg_xlog子目录中的内容，因为其中可能包含系统停机前尚未归档的 WAL 文件。

3. 删除集簇数据目录下以及所有正在使用的表空间根目录下的现有文件和子目录。

4. 从文件系统备份中恢复数据库文件。务必确保它们以正确的所有者（数据库系统用户，而不是root！）和正确的权限恢复。如果使用了表空间，还应验证pg_tblspc/中的符号链接是否已正确恢复。

5. 删除pg_xlog/中现有的所有文件；这些文件来自文件系统备份，因此很可能已经过时而不是最新的。如果你根本没有归档pg_xlog/，那么就以正确权限重新创建它，并注意如果它原先是符号链接，就要重新把它设置成符号链接。

6. 如果你手头还有第 2 步中保存下来的未归档 WAL 段文件，请把它们复制到pg_xlog/中。（最好复制，而不是移动，这样一旦出问题需要重来时，你手里仍然保留着未修改的原始文件。）

7. 在集簇数据目录中创建恢复命令文件 recovery.conf（参见第 26 章）。也可以临时修改 pg_hba.conf，在确认恢复成功之前阻止普通用户连接。

8. 启动服务器。服务器会进入恢复模式，依次读取所需的已归档 WAL 文件。如果恢复因外部错误而终止，只需重启服务器，它就会继续恢复。恢复过程完成后，服务器会将 recovery.conf 重命名为 recovery.done（防止以后意外重新进入恢复模式），然后开始正常的数据库操作。

9. 检查数据库内容，确认你已经恢复到期望状态。如果不是，就回到第 1 步。如果一切正常，就把pg_hba.conf恢复为正常设置，让用户重新连接。

整个过程的关键，是设置一个恢复配置文件，说明希望如何恢复，以及恢复到什么位置。可以使用 recovery.conf.sample（通常位于安装目录的 share/ 目录中）作为模板。必须在 recovery.conf 中指定 restore_command，它告诉 PostgreSQL 如何获取已归档的 WAL 段文件。与 archive_command 一样，它是一个 shell 命令字符串，可以包含 %f，该标记会被替换为所需日志文件的名称；还可以包含 %p，该标记会被替换为日志文件要复制到的路径名。（路径名相对于当前工作目录，即集簇的数据目录。）使用 %% 可以在命令中嵌入实际的 % 字符。最简单的可用命令类似于：

```
restore_command = 'cp /mnt/server/archivedir/%f %p'
```

这会复制先前归档的 WAL 段，来源目录为 /mnt/server/archivedir。当然，也可以使用复杂得多的命令，甚至可以使用要求操作人员挂载相应磁带的 shell 脚本。

重要的是，该命令在失败时必须返回非零退出状态。系统会调用该命令来请求归档中不存在的文件；遇到这种情况时，它必须返回非零值。这不是一种错误情况。被请求的文件并不全都是 WAL 段文件；你还应预期会收到对带有.backup或.history后缀文件的请求。另外请注意，%p路径的文件名部分会与%f不同；不要指望它们可以互换使用。

在归档中找不到的 WAL 段会转而在pg_xlog/中查找；这使得可以使用最近尚未归档的段。不过，凡是归档中可用的段，都会优先于pg_xlog/中的文件使用。

通常，恢复会处理完所有可用的 WAL 段，从而把数据库恢复到当前时间点（或者在可用 WAL 段所允许的情况下尽可能接近当前时间点）。因此，一次正常恢复会以一条“file not found”消息结束，具体错误文本取决于你选择的restore_command。在恢复开始时，你也可能看到一条针对类似00000001.history文件的错误消息。这同样是正常的，在简单恢复场景中并不表示有问题；相关讨论见第 24.3.5 节。

如果希望恢复到过去的某个时间点（例如初级 DBA 删除主要事务表之前），只需在 recovery.conf 中指定所需的停止点。这个停止点称为“恢复目标”，可以通过日期/时间、命名恢复点，或某个特定事务 ID 的完成来指定。撰写本文时，只有日期/时间和命名恢复点这两种方式比较实用，因为还没有工具能帮助你准确确定应使用哪个事务 ID。

> **注意**
>
> 停止点必须晚于基础备份的结束时间，也就是`pg_stop_backup`的结束时间。
> 你不能用某次基础备份恢复到该备份仍在进行中的时间点。
> （若要恢复到这样的时间点，必须回到更早的一次基础备份，
> 再从那里向前重放日志。）

如果恢复过程中发现了损坏的 WAL 数据，恢复会在该点停止，服务器也不会启动。在这种情况下，可以从头重新执行恢复，并指定一个位于损坏点之前的“恢复目标”，
使恢复能够正常完成。如果恢复因外部原因失败，例如系统崩溃或 WAL 归档变得不可访问，
那么只需重新启动恢复，它几乎会从上次失败的位置继续。
恢复重启的工作方式很像正常运行时的检查点：服务器会周期性地把自身状态强制写盘，
然后更新`pg_control`文件，表明已处理过的 WAL 数据无需再次扫描。

24.3.5. 时间线

能够将数据库恢复到过去某个时间点，
也会带来一些类似科幻故事中时间旅行和平行宇宙的复杂情况。例如，
假设在数据库原来的历史中，你在周二下午 5:15 删除了一张重要的表，直到周三中午才发现错误。你从容地取出备份，将数据库恢复到周二下午 5:14，然后重新投入运行。在数据库宇宙的这段历史中，你从未删除过那张表。但假设你后来发现这样做不太合适，
希望回到原来历史中的周三上午某个时刻。
如果数据库恢复运行后覆盖了通往该时刻所需的某些 WAL 段文件，就无法回去了。因此，为了避免这种情况，需要区分时间点恢复后产生的一系列 WAL 记录与数据库原来历史中产生的记录。

为解决这一问题，PostgreSQL 引入了时间线的概念。每当归档恢复完成，系统都会创建一条新时间线，用于标识此次恢复之后产生的一系列 WAL 记录。时间线 ID 是 WAL 段文件名的一部分，因此新时间线不会覆盖旧时间线产生的 WAL 数据。实际上，可以归档许多不同的时间线。这个功能看似没什么用，却常常能救急。比如，你不能确定应该恢复到哪个时间点，
需要反复尝试时间点恢复，直到找到脱离旧历史的最佳分支点。没有时间线，
这个过程很快就会乱得无法管理。有了时间线，就可以恢复到任何先前的状态，
包括早先已放弃的时间线分支中的状态。

每当创建一条新的时间线时，PostgreSQL都会创建一个“时间线历史”文件，
记录它是从哪条时间线、在何时分叉出来的。当从包含多条时间线的归档中恢复时，
这些历史文件对于系统选取正确的 WAL 段文件是必需的。因此，它们会像 WAL 段文件一样被归档到 WAL 归档区域。历史文件只是很小的文本文件，因此长期保存它们既便宜也合适（而段文件通常很大）。如果你愿意，还可以在历史文件中加入注释，
记录创建这条时间线的方式和原因。当你因实验而积累出一批错综复杂的时间线时，
这类注释会特别有价值。

默认情况下，恢复会沿着制作基础备份时的当前时间线进行。
如果希望恢复到某条子时间线（即返回到一次恢复尝试之后产生的某个状态），需要在`recovery.conf` 中指定目标时间线 ID。不能恢复到在基础备份之前就已分支出去的时间线。

24.3.6. 建议和示例

这里给出一些配置持续归档的建议。

24.3.6.1. 独立热备份

可以利用PostgreSQL的备份设施生成独立热备份。这些备份不能用于时间点恢复，
但它们的制作和恢复通常都比pg_dump转储快得多。（它们也比pg_dump转储大得多，
因此在某些情况下速度优势可能会被抵消。）

和基础备份一样，生成独立热备份最简单的方法是使用pg_basebackup工具。如果在调用它时包含-x参数，使用该备份所需的全部事务日志都会自动包含在备份中，恢复该备份时也不需要额外动作。

如果复制备份文件时需要更大的灵活性，也可以使用更低级的流程制作独立热备份。要准备低级独立热备份，将 wal_level 设为 archive（或 hot_standby），将 archive_mode 设为 on，并配置一个 archive_command，使其仅在开关文件存在时执行归档。例如：

```
archive_command = 'test ! -f /var/lib/pgsql/backup_in_progress |
| (test ! -f /var/lib/pgsql/archive/%f && cp %p /var/lib/pgsql/
archive/%f)'
```

当 /var/lib/pgsql/backup_in_progress 存在时，这条命令会执行归档；否则会静默地返回零退出状态（允许 PostgreSQL 回收不需要的 WAL 文件）。

做好上述准备后，就可以使用类似下面的脚本进行备份：

```
touch /var/lib/pgsql/backup_in_progress
psql -c "select pg_start_backup('hot_backup');"
tar -cf /var/lib/pgsql/backup.tar /var/lib/pgsql/data/
psql -c "select pg_stop_backup();"
rm /var/lib/pgsql/backup_in_progress
tar -rf /var/lib/pgsql/backup.tar /var/lib/pgsql/archive/
```

首先创建开关文件 /var/lib/pgsql/backup_in_progress，以启用已完成 WAL 文件的归档。备份完成后删除该开关文件。随后将已归档的 WAL 文件加入备份，使基础备份与所有必需的 WAL 文件都包含在同一个 tar 文件中。请记得在备份脚本中加入错误处理。

24.3.6.2. 压缩的归档日志

如果担心归档存储空间，可以使用 gzip 压缩归档文件：

```
archive_command = 'gzip < %p > /var/lib/pgsql/archive/%f'
```

恢复时则需要使用 gunzip：

```
restore_command = 'gunzip < /mnt/server/archivedir/%f > %p'
```

24.3.6.3. archive_command脚本

很多人选择使用脚本来定义自己的archive_command，这样postgresql.conf中的配置项就会显得非常简单：

```
archive_command = 'local_backup_script.sh "%p" "%f"'
```

只要你希望在归档过程中使用不止一条命令，就建议使用单独的脚本文件。这样一来，所有复杂性都可以在脚本内部管理，而脚本可以使用诸如bash或perl之类的常见脚本语言编写。

可以在脚本中处理的需求示例包括：

- 将数据复制到安全的异地数据存储
- 把 WAL 文件成批处理，使其每三个小时传输一次，而不是每次只传输一个
- 与其他备份和恢复软件对接
- 与监控软件对接以报告错误

> **提示**
>
> 使用`archive_command`脚本时，最好启用logging_collector。
> 脚本写到stderr的任何消息都会出现在数据库服务器日志中，这样一来，
> 当复杂配置失败时就更容易诊断。

24.3.7. 注意事项

在撰写本文时，持续归档技术还存在若干局限。这些问题很可能会在未来版本中得到修复：

- 哈希索引上的操作目前不会写入 WAL 日志，因此重放不会更新这些索引。这意味着任何新插入的行都会被索引忽略，被更新的行会看起来消失了，而被删除的行仍会保留指针。
  换句话说，如果你修改了一张带有哈希索引的表，
  那么在备库服务器上会得到不正确的查询结果。恢复完成后，
  建议在恢复操作结束后对每个这样的索引手工执行REINDEX。

- 如果在进行基础备份时执行了CREATE DATABASE命令，而该`CREATE DATABASE`所复制的模板数据库又在基础备份尚未结束时被修改，
  那么恢复时可能会把这些修改也传播到新建数据库中。这当然并不理想。
  为避免这种风险，最好在进行基础备份时不要修改任何模板数据库。

- CREATE TABLESPACE命令会以字面绝对路径写入 WAL，因此重放时会按相同的绝对路径创建表空间。如果 WAL 在另一台机器上重放，这可能并不理想。即使 WAL 在同一台机器上、但重放到新的数据目录中，也可能有危险：重放仍会覆盖原表空间的内容。
  为避免此类潜在陷阱，最佳做法是在创建或删除表空间后重新执行一次基础备份。

还应注意，默认的WAL格式相当臃肿，因为它包含许多磁盘页面快照。
这些页面快照是为支持崩溃恢复而设计的，因为我们可能需要修复部分写入的磁盘页。
根据你的系统硬件和软件情况，部分写入的风险可能小到可以忽略；在这种情况下，
可以通过full_page_writes参数关闭页面快照，从而显著减少已归档 WAL 文件的总量。（在这样做之前，请先阅读第 29 章中的说明和警告。）关闭页面快照并不妨碍把 WAL 用于PITR 操作。未来一个可能的开发方向，是在`full_page_writes`开启的情况下，通过去除不必要的页面副本来压缩归档 WAL 数据。在此之前，管理员可以考虑尽可能增大检查点间隔相关参数，以减少 WAL 中包含的页面快照数量。

第 25 章 高可用、负载均衡和复制

数据库服务器可以协同工作，
从而在主库失效时让第二台服务器快速接管其任务（高可用性），
或者让多台计算机提供同一份数据（负载均衡）。理想情况下，
数据库服务器应当能够无缝协同工作。提供静态网页服务的 Web 服务器只需把 Web 请求分发到多台机器，就能很容易地组合起来。事实上，只读数据库服务器也相对容易组合起来。
不幸的是，大多数数据库服务器同时处理读写请求，而读/写服务器要更难组合。这是因为，
只读数据只需要在每台服务器上放置一次，
而对任意一台服务器的写入都必须传播到所有服务器，
以保证后续对这些服务器的读取请求返回一致的结果。

这种同步问题是服务器协同工作的根本难点。
由于不存在一种能够对所有使用场景都消除同步问题影响的单一方案，
因此出现了多种不同的解决方案。每一种方案都以不同方式处理这个问题，
并且针对特定负载将其影响降到最低。

某些方案通过只允许一台服务器修改数据来处理同步。能够修改数据的服务器称为读/写、主库（master）或主库（primary）服务器。跟踪主库变更的服务器称为备库（standby）或备库（slave）。只能在被提升为主库之后才能连接的备库称为温备，
而能够接受连接并提供只读查询的备库称为热备。

某些方案是同步的，
即一个修改数据的事务只有在所有服务器都提交该事务之后才被视为已提交。
这保证一次故障切换不会丢失任何数据，
并且所有负载均衡的服务器无论查询哪一台都将返回一致的结果。相反，
异步方案允许一次提交与其传播到其他服务器之间存在一定延迟，
这就带来了切换到备库时丢失某些事务的可能性，
也意味着负载均衡的服务器可能返回略微陈旧的结果。当同步通信过慢时，
就会使用异步通信。

这些方案还可以按粒度分类。有些方案只能处理整个数据库服务器，
而另一些则允许在每个表或者每个数据库级别进行控制。

无论做出哪种选择，都必须考虑性能。功能与性能之间通常存在权衡。例如，
在低速网络上采用完全同步的方案，性能可能下降一半以上，
而异步方案对性能的影响则可能很小。

本节其余部分将概述多种故障切换、复制和负载均衡方案。

25.1. 不同方案的比较

共享磁盘故障切换

共享磁盘故障切换通过只保留一份数据库拷贝来避免同步开销。
它使用一个由多个服务器共享的单一磁盘阵列。如果主库失效，
备库就可以挂载并启动数据库，就好像它正在从一次数据库崩溃中恢复一样。
这允许快速故障切换而不会丢失数据。

共享硬件功能常见于网络存储设备。使用网络文件系统也是可行的，
但必须注意文件系统是否具备完整的 POSIX 行为（见第 17.2.2 节）。这种方法的一大局限是，如果共享磁盘阵列失效或损坏，主库和备库都会无法工作。另一个问题是，
在主库运行期间，备库绝不应访问共享存储。

文件系统（块设备）复制

共享硬件功能的一种变体是文件系统复制，
其中对一个文件系统的所有更改都会镜像到位于另一台计算机上的文件系统。

唯一的限制是，这种镜像必须以能够保证备库拥有该文件系统一致拷贝的方式完成 – 特别是，对备库的写入必须与主库上的写入保持相同顺序。DRBD 是 Linux 上一种流行的文件系统复制方案。

事务日志传送

温备和热备服务器能够通过读取预写式日志（WAL）记录流来保持最新状态。
如果主库失效，备库拥有主库的几乎全部数据，并且能够迅速成为新的主库。
这可以是同步的，也可以是异步的，并且只能用于整个数据库服务器。

可以使用基于文件的日志传送（第 25.2 节）、流复制（见第 25.2.5 节）或两者结合的方式来实现备库。有关热备的信息，见第 25.5 节。

基于触发器的主备复制

主备复制配置会将所有修改数据的查询发送到主库。主库将数据更改异步发送到备库。
主库运行期间，备库可以响应只读查询。备库很适合处理数据仓库查询。

Slony-I 是这种复制类型的一个示例。它按表粒度工作，并且支持多个备库。由于它会以批处理方式异步更新备库，因此在故障切换期间可能会发生数据丢失。

基于语句的复制中间件

使用基于语句的复制中间件时，一个程序会截获每条 SQL 查询，并将其发送到一台或所有服务器。每台服务器独立运行。读写查询必须发送到所有服务器，
确保每台服务器都收到所有更改。但只读查询可以只发送到一台服务器，
从而将读取负载分散到这些服务器上。

如果只是原样广播查询，`random()`、`CURRENT_TIMESTAMP` 等函数以及序列在不同服务器上可能产生不同的值。这是因为每台服务器独立运行，且广播的是 SQL 查询，而不是实际修改后的行。如果无法接受这种情况，
中间件或应用就必须从同一台服务器查询这些值，然后在写入查询中使用这些值。
另一种选择是将这种复制方式与传统主备配置结合使用，
即只向主库发送修改数据的查询，再通过主备复制将其传播到备库，
而不是通过复制中间件传播。还必须确保所有事务在所有服务器上要么全部提交，
要么全部中止，为此可以使用两阶段提交（PREPARE TRANSACTION和COMMIT PREPARED）。Pgpool-II 和 Continuent Tungsten 就是这类复制的例子。

异步多主复制

对于那些并不经常连接或通信链路较慢的服务器，例如笔记本电脑或远程服务器，
保持服务器间的数据一致是一个挑战。使用异步多主复制时，每台服务器都独立工作，
并定期与其他服务器通信，以识别冲突事务。
这些冲突可以由用户或冲突解决规则来解决。Bucardo 是这种复制类型的一个示例。

同步多主复制

在同步多主复制中，每台服务器都能接受写请求，并且在每个事务提交之前，
修改过的数据都会从原始服务器传送给其他每台服务器。
繁重的写入活动可能导致过多锁定和提交延迟，进而带来较差的性能。
读请求可以发送给任意服务器。某些实现使用共享磁盘来减少通信开销。
同步多主复制尤其适合以读为主的负载，尽管它的一大优势是任意服务器都能接受写请求 – 无需在主库和备库之间划分负载，并且由于数据更改是从一台服务器传送到另一台服务器，因此不会出现非确定函数（如 `random()`）的问题。

PostgreSQL 本身不提供这种复制，不过可以在应用代码或中间件中利用 PostgreSQL 的两阶段提交（PREPARE TRANSACTION和COMMIT PREPARED）来实现这一功能。

商业解决方案

由于 PostgreSQL 是开源的，而且易于扩展，许多公司以 PostgreSQL 为基础，创建了具有独特故障切换、复制和负载均衡功能的商业闭源解决方案。

表 25.1总结了上述多种方案的能力。

表 25.1. 高可用、负载均衡和复制特性矩阵

特性	共享磁盘故障切换	文件系统复制	事务日志传送	基于触发器的主备复制	基于语句的复制中间件	异步多主复制	同步多主复制
最常见的实现	NAS	DRBD	流复制	Slony	pgpool-II	Bucardo	
通信方法	共享磁盘	磁盘块	WAL	表行	SQL	表行	表行和行锁
不要求特殊硬件		•	•	•	•	•	•
允许多个主库					•	•	•
主库无额外开销	•		•		•		
不等待多个服务器	•		关闭同步时	•		•	
主库失效时绝不丢失数据	•	•	启用同步时		•		•
备库接受只读查询			启用热备时	•	•	•	•
每个表粒度				•		•	•
不需要冲突解决	•	•	•	•	•		•

有一些方案不适合上述的类别：

数据分区

数据分区会把表拆分成多个数据集。每个数据集只能由一台服务器修改。例如，可以按办公室划分数据，如伦敦和巴黎，每个办公室各有一台服务器。如果有必要执行组合伦敦和巴黎数据的查询，应用可以同时查询两台服务器，或者可以使用主库/备库复制，在每台服务器上保留其他办公室数据的只读副本。

多服务器并行查询执行

上述许多方案都允许多台服务器处理多个查询，但没有一种允许单个查询同时使用多台服务器来更快完成。这种方案允许多台服务器在同一个查询上并发工作。通常的做法是把数据分散到各台服务器上，让每台服务器执行该查询中属于自己的部分，然后把结果返回给一台中心服务器，由它汇总结果并返回给用户。Pgpool-II就具备这种能力。此外，这种方案也可以使用PL/Proxy工具集来实现。

25.2. 日志传送备库

持续归档可用于创建一种高可用性（HA）集簇配置，其中有一个或多个备库随时准备在主库失效时接管操作。这种能力通常称为温备或日志传送。

主库和备库协同工作以提供这种能力，不过两者之间只是松耦合。主库运行在持续归档模式下，而每台备库都运行在持续恢复模式下，从主库读取 WAL 文件。启用这种能力不需要修改数据库表，因此与其他一些复制方案相比，它的管理开销较低。这种配置对主库的性能影响也相对较小。

将 WAL 记录直接从一台数据库服务器移动到另一台数据库服务器，通常称为日志传送。PostgreSQL 通过一次传送一个文件（WAL 段）中的 WAL 记录来实现基于文件的日志传送。WAL 文件（16MB）可以轻松且低成本地传送到任意距离，无论是邻近系统、同一站点的另一台系统，还是地球另一端的系统。该技术所需的带宽取决于主库的事务速率。

基于记录的日志传送粒度更细，会通过网络连接增量地流式传输 WAL 变更（见第 25.2.5 节）。

需要注意的是，日志传送是异步的，也就是说 WAL 记录是在事务提交之后才被传送的。因此，如果主库发生灾难性故障，就会存在一个数据丢失窗口；尚未传送的事务将会丢失。对于基于文件的日志传送，可以通过使用 `archive_timeout` 参数来限制这个数据丢失窗口，它可以设置为低至数秒。不过，如此低的设置会显著增加文件传送所需的带宽。流复制（见第 25.2.5 节）允许把数据丢失窗口缩得更小。

恢复性能已经足够好，因此一旦备库被激活，通常只需片刻就能达到完全可用状态。因此，这种配置被称为温备配置，它提供了高可用性。从归档的基础备份中恢复服务器并向前重放则需要更长时间，因此这种技术只能用于灾难恢复，而不是高可用性。备库还可以用于只读查询，这种情况下它被称为热备服务器。更多信息见第 25.5 节。

25.2.1. 规划

通常，最好让主库和备库尽可能相似，至少从数据库服务器的角度看应如此。尤其是，与表空间相关的路径名会原样传递，因此如果使用该特性，主库和备库必须为表空间配置完全相同的挂载路径。请记住，如果在主库上执行CREATE TABLESPACE，则它所需的任何新挂载点都必须在执行该命令之前先在主库和所有备库上创建好。硬件不必完全相同，但经验表明，在应用和系统的整个生命周期内，维护两个相同的系统要比维护两个不同的系统更容易。无论如何，硬件架构必须相同 – 例如，从 32 位系统向 64 位系统传送日志是不可行的。

一般来说，不能在运行不同主版本 PostgreSQL 的服务器之间传送日志。PostgreSQL 全球开发组的策略是在次版本升级期间不改变磁盘格式，因此主库和备库运行不同次版本通常也能正常工作。不过，这方面并没有正式支持，因此仍建议主库和备库尽量保持在相同的发行级别。当升级到新的次版本时，最安全的策略是先升级备库 – 新的次版本更有可能兼容读取前一个次版本生成的 WAL 文件，反过来则未必。

25.2.2. 备库操作

在备库模式下，服务器会持续应用从主库收到的 WAL。备库可以从 WAL 归档读取 WAL（参见restore_command），也可以通过 TCP 连接直接从主库读取（流复制）。备库还会尝试恢复其集簇 `pg_xlog` 目录中找到的所有 WAL。这通常发生在服务器重启后，此时备库会重新重放重启前从主库接收的 WAL；不过，也可以随时手动将文件复制到 `pg_xlog` 中，让备库重放它们。

启动时，备库首先调用 `restore_command`，恢复归档位置中所有可用的 WAL。当读到归档中可用 WAL 的末尾，且 `restore_command` 失败后，它会尝试恢复 `pg_xlog` 目录中的所有可用 WAL。如果这也失败，且已配置流复制，备库就会尝试连接主库，从归档或 `pg_xlog` 中找到的最后一条有效记录开始接收 WAL 流。如果连接失败、未配置流复制，或连接后来断开，备库就会返回第一步，再次尝试从归档恢复文件。这种依次从归档、`pg_xlog` 和流复制重试的循环，会一直持续到服务器停止，或触发文件触发故障切换。

运行 `pg_ctl promote` 或发现触发文件（`trigger_file`）时，服务器会退出备库模式，切换到正常运行。在故障切换前，会恢复归档或 `pg_xlog` 中能立即获取的所有 WAL，但不会尝试连接主库。

25.2.3. 为备库准备主库

如第 24.3 节所述，在主库上设置持续归档，将日志归档到备库可访问的归档目录。即使主库宕机，该归档位置也应该对备库可访问；也就是说，它应位于备库本身或另一台可信服务器上，而不是位于主库上。

如果想使用流复制，就需要在主库上配置认证，以允许来自备库的复制连接；也就是说，创建一个角色，并在 pg_hba.conf 中添加一个或多个把数据库字段设置为 replication 的合适项。还要确保主库配置文件中的 max_wal_senders 被设置为足够大的值。

如第 24.3.2 节所述，获取一个基础备份来引导备库。

25.2.4. 设置备库

要配置备库，请恢复从主库取得的基础备份（参见第 24.3.4 节）。在备库的集簇数据目录中创建恢复命令文件 recovery.conf，并启用 standby_mode。将 restore_command 设为一个从 WAL 归档复制文件的简单命令。如果计划使用多个备库提供高可用，请将 recovery_target_timeline 设为 latest，使备库能够跟随故障切换到另一个备库时发生的时间线变化。

> **注意**
>
> 不要将 pg_standby 或类似工具与这里介绍的内置备库模式一起使用。如果文件不存在，restore_command 应立即返回；服务器会在必要时重试该命令。有关 pg_standby 等工具的使用，参见第 25.4 节。

如果希望使用流复制，请在 primary_conninfo 中填写 libpq 连接字符串，包含连接主库所需的主机名（或 IP 地址）及其他信息。如果主库使用密码认证，还需要在 primary_conninfo 中指定密码。

如果你是为了高可用目的设置备库，那么也应像主库一样设置 WAL 归档、连接和认证，因为故障切换后该备库将作为主库工作。

如果使用 WAL 归档，可以借助archive_cleanup_command参数删除备库不再需要的文件，以尽量缩小归档大小。pg_archivecleanup 工具专门设计用于在典型的单备库配置中与 archive_cleanup_command 配合使用，见pg_archivecleanup。不过请注意，如果你还把归档用于备份目的，即使这些文件对备库已经不再需要，
也必须至少保留从最新基础备份恢复所需的那些文件。

一个简单的 recovery.conf 示例如下：

```
standby_mode = 'on'
primary_conninfo = 'host=192.168.1.50 port=5432 user=foo password=
foopass'
restore_command = 'cp /path/to/archive/%f %p'
archive_cleanup_command = 'pg_archivecleanup /path/to/archive %r'
```

备库的数量可以任意多，但如果使用流复制，请确保主库上的 max_wal_senders 设置得足够高，以允许它们同时连接。

25.2.5. 流复制

与基于文件的日志传送相比，流复制可以让备库保持得更接近最新状态。备库连接到主库，主库在生成 WAL 记录时就会把它们流式发送给备库，而不必等待 WAL 文件被写满。

默认情况下流复制是异步的（见第 25.2.7 节），在这种情况下主库上提交一个事务与该变化在备库上变得可见之间存在短暂的延迟。
不过这种延迟比基于文件的日志传送方式中要小得多，
在备库的能力足以跟得上负载的前提下，延迟通常低于一秒。在流复制中，不需要 archive_timeout 来缩减数据丢失窗口。

如果你使用流复制，但没有启用基于文件的持续归档，就必须把主库上的 wal_keep_segments 设置为一个足够大的值，以确保旧的 WAL 段不会在备库还需要它们追赶时就被过

早回收。如果备库落后得太多，就需要通过新的基础备份重新初始化。
如果配置了一个备库可访问的 WAL 归档，则不需要 wal_keep_segments，因为备库始终可以利用归档追赶上来。

要使用流复制，请先按第 25.2 节所述配置基于文件的日志传送备库。将这种备库转为流复制备库的关键步骤，是在 recovery.conf 文件中设置 primary_conninfo，使其指向主库。在主库上设置listen_addresses和认证选项（参见 pg_hba.conf），使备库能够连接主库的 replication 伪数据库（参见第 25.2.5.1 节）。

在支持 keepalive 套接字选项的系统上，设置tcp_keepalives_idle、tcp_keepalives_interval和tcp_keepalives_count有助于主库迅速注意到断开的连接。

设置来自备库的最大并发连接数（详见max_wal_senders）。

备库启动后，如果正确设置了 primary_conninfo，备库会在重放归档中所有可用的 WAL 文件后连接主库。如果连接成功，就会在备库上看到 walreceiver 进程，在主库上看到对应的 walsender 进程。

25.2.5.1. 认证

务必正确设置复制访问权限，使只有受信任的用户才能读取 WAL 流，因为很容易从中提取需要权限才能访问的信息。备库必须以超级用户或具有 REPLICATION 权限的账户向主库认证。建议为复制创建专用用户账户，并授予 REPLICATION 和 LOGIN 权限。虽然 REPLICATION 权限赋予的权限很高，但它不允许用户修改主库系统上的任何数据，而 SUPERUSER 权限允许这样做。

复制的客户端认证由 pg_hba.conf 中的一条记录控制，该记录需要把 replication 指定在 *database* 字段中。例如，如果备库运行在主机 IP 192.168.1.100 上，并且用于复制的账户名为 foo，管理员可以在主库上的 pg_hba.conf 文件中加入下列行：

```
# Allow the user "foo" from host 192.168.1.100 to connect to the
 primary
# as a replication standby if the user's password is correctly
 supplied.
#
# TYPE  DATABASE        USER            ADDRESS
 METHOD
host    replication     foo             192.168.1.100/32        md5
```

主库的主机名、端口号、连接用户名和密码在 recovery.conf 文件中指定。也可以将密码设置在备库的 ~/.pgpass 文件中（将 replication 指定为 *database* 字段的值）。例如，如果主库所在主机的 IP 地址为 192.168.1.50，端口为 5432，用于复制的账户名为 foo，密码为 foopass，管理员可以将以下行添加到 recovery.conf 文件（位于备库）中：

```
# The standby connects to the primary that is running on host 192.
168.1.50
# and port 5432 as the user "foo" whose password is "foopass".
primary_conninfo = 'host=192.168.1.50 port=5432 user=foo password=
foopass'
```

25.2.5.2. 监控

流复制的一个重要健康指标，是主库上已经生成但尚未在备库上应用的 WAL 记录量。你可以通过比较主库上的当前 WAL 写入位置和备库收到的最后一个 WAL 位置来计算这种滞后。这些位置分别可以用主库上的 pg_current_xlog_location 和备库上的 pg_last_xlog_receive_location 取得（详见表 9.60和表 9.61）。备库上的最后一个 WAL 接收位置也会显示在 WAL 接收进程的进程状态中，即通过 ps 命令显示的状态（详见第 27.1 节）。

可以通过 pg_stat_replication 视图获取 WAL 发送进程列表。pg_current_xlog_location 与 sent_location 字段相差较大，可能表示主库负载很高；而 sent_location 与备库上 pg_last_xlog_receive_location 之间存在差异，则可能表示网络延迟或备库负载很高。

25.2.6. 级联复制

级联复制特性允许一台备库接受复制连接，并像中继器一样把 WAL 记录流式发送给其他备库。这可以用来减少直接连接到主库的连接数，并使站点间的带宽开销最小化。

一台同时扮演接收者和发送者角色的备库称为级联备库。
与主库连接更直接（经过更少级联备库）的备库称为上游服务器，
而距离更远的备库称为下游服务器。级联复制并不限制下游服务器的数量和拓扑，
不过每台备库只连接到一台上游服务器，而这条链路最终都会通向同一台主库。

级联备库不仅发送从主库接收到的 WAL 记录，也会发送那些从归档中恢复的记录。因此，即使某条上游复制连接被中断，只要仍有新的 WAL 记录可用，下游的流复制就会继续。

级联复制目前是异步的。同步复制（见第 25.2.7 节）设置当前对级联复制无影响。

热备反馈会向上传播，无论级联拓扑如何。

提升级联备库会终止它所服务的直接下游复制连接。
这是因为各备库之间的时间线变得不同，它们无法继续复制。
受影响的备库可以重新连接以重建流复制。

要使用级联复制，需要把级联备库设置为能够接受复制连接（也就是设置max_wal_senders和hot_standby，并配置基于主机的认证）。你还需要把下游备库中的 primary_conninfo 设置为指向级联备库。

25.2.7. 同步复制

PostgreSQL 的流复制默认是异步的。如果主库崩溃，则某些已提交的事务可能尚未复制到备库，从而导致数据丢失。数据丢失量与故障切换时的复制延迟成正比。

同步复制能够确认一个事务所做的全部修改已经被传送到一台同步备库。
这扩展了事务提交所提供的标准持久性级别。在计算机科学理论中，这种保护级别被称为 2-safe 复制。

请求同步复制时，每个写事务的提交都会等待，直到收到确认，
表明该提交已被写入主库和备库磁盘上的事务日志。数据唯一可能丢失的情况，
是主库和备库同时崩溃。这可以提供更高的持久性级别，
不过前提是系统管理员必须谨慎地部署和管理这两台服务器。
等待确认会增强用户对服务器崩溃时更改不会丢失的信心，
但也必然会增加请求事务的响应时间。最短等待时间是主库与备库之间的往返时间。

只读事务和事务回滚不需要等待备库的回应。子事务提交也不需要等待备库响应，
只有顶层提交才需要等待。数据装载或索引构建等长时间运行的动作，
直到最终提交时才会等待。所有两阶段提交操作都需要等待提交，
包括准备和提交两个阶段。

25.2.7.1. 基本配置

一旦流复制已经配置好，配置同步复制只需要额外一步：必须把synchronous_standby_names设置为非空值。synchronous_commit也必须设置为on，但由于这是默认值，
通常无需更改（见第 18.5.1 节和第 18.6.2 节）。这样的配置会导致每次提交都等待确认，以保证备库已经把提交记录写入持久存储。synchronous_commit可以由单个用户设置，
因此既可以在配置文件中配置，也可以针对特定用户或数据库配置，或者由应用动态配置，
从而在每事务级别控制持久性保证。

当提交记录已经在主库上写入磁盘之后，WAL 记录就会被发送到备库。每当新的一批 WAL 数据被写入磁盘时，备库就会发送回复消息，除非备库上的wal_receiver_status_interval被设置为零。如果根据主库上的synchronous_standby_names，该备库是第一个匹配的备库，那么它发出的回复将被用于唤醒那些正在等待确认提交记录已被收到的事务。这些参数允许管理员指定哪些备库应作为同步备库。注意，同步复制的配置主要在主库上进行。被命名的备库必须直接连接到主库；主库并不知道使用级联复制的下游备库。

把synchronous_commit设置为remote_write，会使每次提交都等待，直到备库确认已经收到提交记录并把它写入自己的操作系统，但不会等待数据被刷到备库磁盘上。与on相比，这种设置提供较弱一些的持久性保证：在操作系统崩溃时，备库可能丢失数据，尽管在PostgreSQL崩溃时不会。不过在实践中，这是一种有用的设置，因为它可以降低事务响应时间。只有当主库和备库都崩溃，并且主库数据库同时发生损坏时，才可能发生数据丢失。

如果请求快速关闭，用户将停止等待。不过，与使用异步复制时一样，在所有尚未传送的 WAL 记录传输到当前已连接的备库之前，服务器不会完全关闭。

25.2.7.2. 性能规划

同步复制通常要求对备库进行仔细规划和部署，才能保证应用具有可接受的性能。等待本身不会占用系统资源，但事务锁会一直保持到传输得到确认。因此，若不谨慎使用同步复制，数据库应用的性能会因为响应时间增加和争用加剧而下降。

PostgreSQL允许应用开发者通过复制来指定所需的持久性级别。这可以在整个系统级别指定，也可以针对特定用户、特定连接，甚至单个事务指定。

例如，一个应用负载可能由如下部分组成：10% 的变更是重要的客户资料，而 90% 的变更是不太重要、即使丢失业务也较容易承受的数据，例如用户之间的聊天消息。

通过在应用级别（在主库上）指定同步复制选项，我们可以只对最重要的变更提供同步复制，而不会拖慢大部分工作负载。应用级别选项是让高性能应用获得同步复制收益的一种重要而实用的工具。

你应当确保网络带宽高于 WAL 数据的生成速率。

25.2.7.3. 高可用规划

当synchronous_commit设置为on或remote_write时，事务提交将等待，直到同步备库做出响应。如果最后一台（或唯一一台）同步备库崩溃，这种响应可能永远不会到来。

对于避免数据丢失而言，最好的办法是确保你不会失去最后一台剩余的同步备库。这可以通过在synchronous_standby_names中命名多个潜在同步备库来实现。第一台被命名的备库将被用作同步备库。列在它之后的备库，会在第一台失效时接替同步备库的角色。

当一台备库第一次连接到主库时，它还没有正确同步。这种状态称为catchup模式。一旦备库与主库之间的滞后第一次变为零，它就会进入实时的streaming状态。备库刚创建之后，追赶阶段可能会持续较长时间。如果备库被关闭，则追赶阶段会随着它停机时间的延长而变长。只有在到达streaming状态后，备库才能成为同步备库。

如果主库在提交正等待确认时重启，这些等待中的事务会在主库恢复后被标记为已完全提交。无法确定在主库崩溃时，所有备库是否已经收到全部待传送的 WAL 数据。因此，某些事务可能不会在备库上显示为已提交，即使它们在主库上显示为已提交。我们所提供的保证是：只有在确认 WAL 数据已经被该备库安全接收之后，应用才会收到事务成功提交的显式确认。

如果你确实失去了最后一台备库服务器，那么就应当禁用synchronous_standby_names，然后在主库上重新加载配置文件。

如果主库与剩余的备库隔离开了，你应当故障切换到那些剩余备库中最佳的候选者。

如果需要在事务等待期间重新创建备库，请确保在 synchronous_commit = off 的会话中运行 pg_start_backup() 和 pg_stop_backup() 命令，否则这些请求会一直等待备库出现。

25.3. 故障切换

如果主库失效，备库就应该开始执行故障切换过程。

如果备库失效，则不需要发生故障切换。如果备库能够重新启动，
即使是在稍后某个时间点，恢复过程也可以立即重新开始，从而利用可重启恢复的优势。
如果备库无法重新启动，则应创建一个全新的备库实例。

如果主库失效，而备库成为新的主库，那么旧主库之后如果重新启动，
你必须有一种机制通知它，它已经不再是主库。这有时被称为STONITH（Shoot The Other Node In The Head），它对于避免两个系统都认为自己是主库的情况至关重要，因为那种情况会导致混乱，并最终造成数据丢失。

许多故障切换系统只使用两个系统，即主库和备库，并通过某种心跳机制连接它们，
以持续验证两者之间的连通性以及主库的可用性。
也可以使用第三个系统（称为见证服务器）来防止某些不恰当的故障切换，
但除非设置得足够谨慎并经过严格测试，否则额外增加的复杂性可能并不值得。

PostgreSQL并不提供用于识别主库故障并通知备库的系统软件。
现在已经存在许多这样的工具，
并且它们通常能很好地与成功故障切换所需的操作系统设施整合在一起，例如 IP 地址迁移。

一旦故障切换到备库，系统中就只剩下一台服务器在运行。这被称为退化状态。
原来的备库现在成为主库，而原来的主库已经停机，并且可能持续停机。
要恢复到正常运行状态，就必须重新创建一台备库：要么在原主库恢复后在其上重建，
要么在第三台可能是全新的服务器上重建。一旦完成，
就可以认为主库和备库已经交换了角色。有些人会选择使用第三台服务器，
在新的备库重建完成之前为新的主库提供后备支持，
但显然这会让系统配置和操作流程更加复杂。

因此，从主库切换到备库可以很快，但重新准备故障切换集簇仍然需要时间。
定期在主库与备库之间进行切换是有益的，因为它允许每个系统定期停机维护。
这也相当于对故障切换机制进行测试，以确保真正需要它时它能够正常工作。
建议编写书面的管理操作规程。

要触发日志传送备库的故障切换，可以运行 pg_ctl promote，或按照 recovery.conf 中 trigger_file 设置指定的文件名和路径创建触发文件。如果计划使用 pg_ctl promote 进行故障切换，就不需要设置 trigger_file。如果所配置的报表服务器只是用来分担主库的只读查询，而不用于高可用，则不需要提升它。

25.4. 日志传送的另一种方法

除了前几节介绍的内置备库模式，也可以使用一个轮询归档位置的 restore_command。在 8.4 及更早版本中，这是唯一可用的方法。采用这种配置时，应关闭 standby_mode，因为备库运行所需的轮询由你自己实现。其参考实现参见pg_standby模块。

注意，在这种模式下，服务器每次应用一整个 WAL 文件。因此，如果使用备库处理查询（参见热备），主库上的操作与备库上能看到该操作的结果之间，会有一段延迟，
其长度相当于写满一个 WAL 文件所需的时间。可以用 archive_timeout 缩短这一延迟。还要注意，这种方法不能与流复制结合使用。

主库和备库上执行的操作都是普通的持续归档和恢复任务。
两台数据库服务器之间唯一的联系，就是它们共享的 WAL 文件归档：主库写入归档，备库从归档读取。必须确保不同主库的 WAL 归档不会混在一起或弄错。如果归档仅用于备库运行，则无需保留很大的归档。

让这两台松散耦合的服务器协同工作的关键，只是备库上的 `restore_command`：当请求下一个 WAL 文件时，它会等待主库提供该文件。`restore_command` 在备库的 `recovery.conf` 文件中指定。正常恢复处理会从 WAL 归档请求文件，如果文件不可用，就报告失败。对备库处理而言，下一个 WAL 文件尚不可用是正常情况，因此备库必须等待它出现。对于以 `.backup` 或 `.history` 结尾的文件，则无需等待，必须返回非零返回码。可以编写一个自定义脚本，循环检查下一个 WAL 文件是否存在，从而实现会等待的 `restore_command`。还必须提供触发故障切换的方法，用来中断 `restore_command`、跳出循环，并向备库返回文件未找到错误。这会结束恢复，随后备库就会作为普通服务器启动。

一个合适的 `restore_command` 的伪代码如下：

```
triggered = false;
while (!NextWALFileReady() && !triggered)
{
    sleep(100000L);         /* wait for ~0.1 sec */
    if (CheckForExternalTrigger())
        triggered = true;
}
if (!triggered)
        CopyWALFileForRecovery();
```

pg_standby模块提供了会等待的 `restore_command` 的可用示例。应参考它来正确实现上述逻辑，也可以按需扩展，以支持特定配置和环境。

触发故障切换的方法是规划和设计的重要部分。一个可能的选择是 `restore_command` 命令。每个 WAL 文件都会执行一次该命令，但运行 `restore_command` 的进程针对每个文件单独创建和结束，因此不存在守护进程或服务器进程，也无法使用信号或信号处理程序。所以，`restore_command` 不适合触发故障切换。可以使用简单的超时机制，尤其是在已知主库 `archive_timeout` 设置的情况下配合使用。不过，这容易出错，因为网络问题或主库繁忙就可能导致故障切换。如果能够安排，
显式创建触发文件之类的通知机制是理想的方式。

25.4.1. 实现

使用这种替代方法配置备库的简要流程如下。各步骤的完整细节，参见所注明的前面章节。

1. 尽可能将主库和备库系统配置得相同，包括安装两个完全相同、发行版本一致的 PostgreSQL 副本。

2. 配置持续归档，将主库的 WAL 归档到备库上的一个目录。确保在主库上正确设置archive_mode、archive_command和archive_timeout（参见第 24.3.1 节）。

3. 制作主库的基础备份（参见第 24.3.2 节），并将这些数据装载到备库上。

4. 在备库上从本地 WAL 归档开始恢复，在 `recovery.conf` 中指定前面所述的会等待的 `restore_command`（参见第 24.3.4 节）。

恢复过程将 WAL 归档视为只读，因此，WAL 文件一旦复制到备库系统，就可以在备库读取它的同时，将其复制到磁带上。这样，既能运行备库提供高可用，
又能保存文件用于更长期的灾难恢复。

为了测试，可以在同一个系统上同时运行主库和备库。
这不会对服务器的稳健性带来有意义的提升，也不能称为高可用。

25.4.2. 基于记录的日志传送

也可以使用这种替代方法实现基于记录的日志传送，但需要自行开发，而且只有整个 WAL 文件传送完成后，更改才会对热备上的查询可见。

外部程序可以调用 `pg_xlogfile_name_offset()` 函数（参见第 9.26 节），取得当前 WAL 末尾所在的文件名及其在文件中的精确字节偏移。随后就可以直接访问 WAL 文件，将上次已知的 WAL 末尾到当前末尾之间的数据复制到备库。采用这种方法，可能丢失数据的时间窗口就是复制程序的轮询周期，可以很短，
而且不会因为强制归档未写满的段文件而浪费带宽。注意，备库的 `restore_command` 脚本只能处理完整的 WAL 文件，因此这些增量复制的数据通常不会提供给备库使用。它们只有在主库失效时才有用 – 此时，在允许备库启动前，将最后一个不完整的 WAL 文件交给备库。要正确实现这一过程，`restore_command` 脚本必须与数据复制程序配合。

从 PostgreSQL 9.0 开始，可以使用流复制（参见第 25.2.5 节），更省力地获得同样的收益。

25.5. 热备

热备是用来描述服务器在归档恢复或备库模式下仍然可以接受连接并运行只读查询的能力的术语。这对于复制用途以及把备份以极高精度恢复到目标状态都很有用。
术语热备还指服务器能够在用户持续运行查询和/或保持连接不断开的同时，
从恢复状态切换到正常运行状态的能力。

在热备模式下运行查询与正常查询操作类似，不过正如下文所述，
在使用和管理上存在一些差异。

25.5.1. 用户概览

当备库上的hot_standby参数被设置为真时，一旦恢复把系统带到一致状态，
它就会开始接受连接。所有这类连接都严格是只读的，甚至不能写入临时表。

备库上的数据需要一些时间才能从主库到达，因此主库和备库之间会有可测量的延迟。
因此，在主库和备库上几乎同时运行同一查询，可能会返回不同的结果。
我们说备库上的数据与主库是最终一致的。一旦某个事务的提交记录在备库上被重放，
该事务所做的修改就会对备库上之后取得的所有新快照可见。
快照可以在每个查询开始时取得，也可以在每个事务开始时取得，
这取决于当前的事务隔离级别。详见第 13.2 节。

在热备期间启动的事务可以发出下列命令：

- 查询访问：`SELECT`、`COPY TO`
- 游标命令：`DECLARE`、`FETCH`、`CLOSE`
- 参数：`SHOW`、`SET`、`RESET`
- 事务管理命令：
  - `BEGIN`、`END`、`ABORT`、`START TRANSACTION`
  - `SAVEPOINT`、`RELEASE`、`ROLLBACK TO SAVEPOINT`
  - `EXCEPTION`块和其他内部子事务
- `LOCK TABLE`，不过只限于显式指定下列模式之一时：`ACCESS SHARE`、`ROW SHARE` 或 `ROW EXCLUSIVE`
- 计划和资源：`PREPARE`、`EXECUTE`、`DEALLOCATE`、`DISCARD`

- 插件和扩展：LOAD

在热备期间启动的事务永远不会被分配事务 ID，也不能写入系统预写式日志。因此，下列动作都会产生错误消息：

- 数据操纵语言（DML）：INSERT、UPDATE、DELETE、COPY FROM、TRUNCATE。请注意，恢复期间不存在任何允许执行触发器的动作。这个限制甚至适用于临时表，因为不分配事务 ID 就无法读取或写入表行，而目前热备环境中无法分配事务 ID。
- 数据定义语言（DDL）：CREATE、DROP、ALTER、COMMENT。这个限制甚至适用于临时表，因为执行这些操作需要更新系统目录表。
- SELECT ... FOR SHARE | UPDATE，因为不更新底层数据文件就无法获取行锁。
- 作用在SELECT语句上、会生成 DML 命令的规则。
- LOCK，如果它显式请求高于 ROW EXCLUSIVE MODE 的模式。
- 简写默认形式的LOCK，因为它请求的是 ACCESS EXCLUSIVE MODE。
- 显式设置为非只读状态的事务管理命令：
  - BEGIN READ WRITE、START TRANSACTION READ WRITE
  - SET TRANSACTION READ WRITE、SET SESSION CHARACTERISTICS AS TRANSACTION READ WRITE
  - SET transaction_read_only = off
- 两阶段提交命令：PREPARE TRANSACTION、COMMIT PREPARED、ROLLBACK PREPARED，因为即使是只读事务，在准备阶段（两阶段提交的第一阶段）也需要写入 WAL。
- 序列更新：nextval()、setval()
- LISTEN、UNLISTEN、NOTIFY

在正常运行中，“只读”事务允许使用LISTEN、UNLISTEN和NOTIFY，因此热备会话受到的限制要比普通只读会话稍微更严格一些。将来的版本中，这些限制中的某些可能会被放宽。

在热备期间，参数transaction_read_only始终为真，并且不能更改。不过，只要不尝试修改数据库，热备期间的连接行为与其他数据库连接大致相同。如果发生故障切换或计划内切换，数据库将切换到正常处理模式。服务器切换模式时，会话仍会保持连接。一旦热备结束，就可以发起读写事务（即使该会话是在热备期间开始的）。

用户可以执行 SHOW transaction_read_only，判断自己的会话是否只读。此外，还有一组函数（表 9.61）可用于获取备库信息。借助它们，可以编写感知数据库当前状态的程序，用来监控恢复进度，或编写将数据库恢复到特定状态的复杂程序。

25.5.2. 处理查询冲突

主库和备库在许多方面都是松耦合的。主库上的动作会对备库产生影响，因此它们之间可能出现负面交互或冲突。最容易理解的冲突是性能：如果主库上正在执行一次大规模数据装载，那么备库上也会产生类似的 WAL 记录流，因此备库查询可能会争用系统资源，例如 I/O。

热备还可能发生其他类型的冲突。这些冲突属于硬冲突，因为查询可能需要被取消，并且在某些情况下还需要断开会话来解决冲突。系统为用户提供了若干种处理这些冲突的方法。冲突场景包括：

- 主库上获取的 Access Exclusive 锁，包括显式的`LOCK`命令和各种DDL操作，会与备库查询中的表访问发生冲突。
- 主库上删除表空间，会与备库查询把该表空间用于临时工作文件发生冲突。
- 主库上删除数据库，会与备库上连接到该数据库的会话发生冲突。
- 来自 WAL 的清理记录在应用时，会与那些快照仍然能够“看到”将被删除的行的备库事务发生冲突。
- 来自 WAL 的清理记录在应用时，会与备库上访问目标页面的查询发生冲突，而不管待删除的数据是否可见。

在主库上，这些情况只会导致等待；用户可以选择取消冲突动作中的任意一方。
但是在备库上没有这种选择：已经被写入 WAL 的动作早已在主库上发生，因此备库在应用它时不能失败。此外，让 WAL 应用无限期等待通常是很不可取的，因为备库的状态会越来越落后于主库。因此，系统提供了一种机制，用来强制取消那些与即将应用的 WAL 记录发生冲突的备库查询。

一个典型例子是，主库上的管理员在某个表上执行`DROP TABLE`，而备库上正有查询访问该表。显然，如果`DROP TABLE`在备库上被应用，该备库查询就无法继续。如果这种情况发生在主库上，`DROP TABLE`会等待直到其他查询结束。但在主库上执行`DROP TABLE`时，主库并不知道备库上正在运行哪些查询，因此它不会等待这些备库查询。于是，WAL 变更记录会在备库查询仍在运行时到达备库，从而引发冲突。备库要么延迟应用这条 WAL 记录（以及之后的所有记录），要么取消冲突查询，以便应用`DROP TABLE`。

当冲突查询很短时，通常最好通过稍微延迟 WAL 应用来让它完成；但 WAL 应用长时间延迟通常并不可取。因此，取消机制提供了max_standby_archive_delay和max_standby_streaming_delay这两个参数，用于定义 WAL 应用允许的最大延迟。一旦应用新收到的 WAL 数据所花费的时间超过相应延迟设置，冲突查询就会被取消。之所以有两个参数，是为了分别针对从归档读取 WAL 数据的场景（即从基础备份进行初始恢复，或者“追赶”一台已经远远落后的备库）以及通过流复制读取 WAL 数据的场景指定不同的延迟值。

如果一台备库主要用于高可用性，那么最好把这些延迟参数设置得较短，
这样服务器就不会因为备库查询导致的延迟而远远落后于主库。但是，
如果该备库的用途是执行长时间运行的查询，那么较高甚至无限的延迟值可能更合适。
不过要记住，如果某个长时间运行的查询延迟了 WAL 记录的应用，它也可能使备库上的其他会话看不到主库上的最新变更。

一旦`max_standby_archive_delay`或`max_standby_streaming_delay`
指定的延迟被超越，冲突查询将被取消。这通常仅导致一个取消错误，尽管在重放一个`DROP DATABASE`的情况下整个冲突会话都将被终止。另外，
如果冲突发生在一个被空闲事务持有的锁上，
该冲突会话会被终止（这种行为可能在未来被改变）。

被取消的查询可以立即重试（当然，要先开始一个新事务）。
由于查询取消取决于正在重放的 WAL 记录的性质，被取消的查询再次执行时完全可能成功。

请记住，延迟参数要与备库收到 WAL 数据之后经过的时间进行比较。因此，留给备库上任何一个查询的宽限期从不会超过延迟参数，
并且如果备库已经由于等待之前的查询完成而落后或者因为过重的更新负载而无法跟上主库，宽限期可能会更少。

备库查询与 WAL 重放发生冲突的最常见原因是“过早清理”。正常情况下，当不再有事务需要看到旧行版本来保证符合 MVCC 规则的数据可见性时，PostgreSQL允许清理这些旧行版本。不过，这条规则只能应用于在主库上执行的事务。因此，
主库上的清理有可能移除某个备库事务仍然可见的行版本。

有经验的用户应注意，行版本清理和行版本冻结都有可能与备库查询冲突。手动运行 `VACUUM FREEZE` 很可能导致冲突，即使表中没有更新过或删除过的行也是如此。

用户应当清楚，主库上经常并且大量更新的表，会很快导致备库上的长时间运行查询被取消。在这种情况下，把max_standby_archive_delay或max_standby_streaming_delay设置为有限值，可以视作类似于设置statement_timeout。

如果备库查询被取消的次数多得令人无法接受，也存在补救办法。第一种选择是设置hot_standby_feedback参数，它会阻止VACUUM移除最近死亡的行，因此不会发生清理冲突。如果这样做，你应当注意这会延迟主库上死行的清理，从而可能导致不希望出现的表膨胀。不过，清理情况不会比直接在主库上运行这些备库查询更糟，而且你仍然可以获得把执行卸载到备库上的好处。在这种情况下必须保持较大的 max_standby_archive_delay，因为延迟的 WAL 文件中可能已经包含与所需备库查询相冲突的条目。

另一种选择是增大主库上的vacuum_defer_cleanup_age，使死行不会像通常那样很快被清理。这样，无需将 max_standby_streaming_delay 设得很大，就能让备库查询在被取消前有更多执行时间。不过，这种方法难以保证具体的执行时间窗口，因为 vacuum_defer_cleanup_age 是按主库上执行的事务数来衡量的。

查询取消的数量及其原因可以通过备库上的pg_stat_database_conflicts系统视图查看。pg_stat_database系统视图也包含汇总信息。

25.5.3. 管理员概览

如果 hot_standby 为 on（默认值，设置在 postgresql.conf 中），且存在 recovery.conf 文件，服务器就会以热备模式运行。不过，可能还要过一段时间才允许热备连接，因为服务器只有完成足够的恢复，达到可供查询运行的一致状态后，才会接受连接。在此期间，尝试连接的客户端会被拒绝，并收到错误消息。要确认服务器已启动，可以在应用中循环尝试连接，或在服务器日志中查找以下消息：

```
LOG:  entering standby mode

... then some time later ...

LOG:  consistent recovery state reached
LOG:  database system is ready to accept read only connections
```

主库每次检查点都会记录一致性信息。如果主库的 wal_level 未设为 hot_standby，则读取这一期间写入的 WAL 时不能启用热备。如果同时满足以下两个条件，达到一致状态也可能延迟：

- 某个写事务包含超过 64 个子事务
- 非常长生命周期的写事务

如果运行基于文件的日志传送（“温备”），可能需要等到下一个 WAL 文件到达，最长等待时间可达到主库上的 archive_timeout 设置值。

如果在主库上更改了某些参数，就需要重新配置备库上的相应设置。对于这些参数，备库上的值必须大于或等于主库上的值。如果这些参数设置得不够大，备库会拒绝启动。此时可以提高参数值，然后重启服务器，再次开始恢复。这些参数是：

- max_connections
- max_prepared_transactions
- max_locks_per_transaction

管理员为max_standby_archive_delay和max_standby_streaming_delay选择合适的设置非常重要。最佳选择取决于业务优先级。例如，如果服务器的主要任务是充当高可用服务器，那么你会希望延迟设置较低，

甚至可能设为零，尽管这是一个非常激进的设置。
如果备库承担的是决策支持查询的附加服务器角色，那么把最大延迟设置为数小时甚至 -1（意味着永远等待查询完成）也可能是可以接受的。

主库上写出的事务状态“提示位”不会被 WAL 记录，因此备库上的数据很可能会再次写出这些提示位。这样一来，即使所有用户都是只读的，备库仍然会执行磁盘写操作；
不过数据值本身并不会发生变化。用户仍然会写出大型排序临时文件，并重新生成 relcache 信息文件，因此在热备模式下，数据库没有任何部分是真正只读的。还要注意，使用 dblink模块写入远程数据库，以及借助 PL 函数执行其他数据库外部操作，依然是可能的，即使该事务在本地是只读的。

在恢复模式下，不接受下列类型的管理命令：

- 数据定义语言（DDL）：例如 `CREATE INDEX`
- 权限和所有权：`GRANT`、`REVOKE`、`REASSIGN`
- 维护命令：`ANALYZE`、`VACUUM`、`CLUSTER`、`REINDEX`

再次注意，这些命令中的某些在主库上的“只读”事务中实际上是被允许的。

因此，你不能创建只存在于备库上的额外索引，也不能创建只存在于备库上的统计信息。
如果需要这些管理命令，应在主库上执行，最终这些变更会传播到备库。

`pg_cancel_backend()`和`pg_terminate_backend()`可以作用于用户后端，
但不能作用于执行恢复操作的启动进程。`pg_stat_activity`
不会把正在恢复的事务显示为活动状态。因此，恢复期间`pg_prepared_xacts`始终为空。
如果你希望解决处于不确定状态的预备事务，请查看主库上的`pg_prepared_xacts`，
并在那里发出命令解决这些事务，或者在恢复结束后再解决它们。

与正常情况一样，`pg_locks`会显示后端持有的锁。`pg_locks`
还会显示一个由启动进程管理的虚拟事务，它拥有所有正在被恢复重放的事务所持有的 `AccessExclusiveLocks`。请注意，启动进程不会为了修改数据库而获取锁，因此除了 `AccessExclusiveLocks`之外，其他锁不会在启动进程的`pg_locks`中显示；
它们只是被假定存在。

Nagios的check_pgsql插件可以工作，因为它检查的简单信息是存在的。check_postgres监控脚本也可以工作，
尽管其中某些被报告的值可能会给出不同或令人困惑的结果。例如，
最近一次清理时间不会被维护，因为备库上不会发生清理。不过，
主库上执行的清理仍然会把其变更发送到备库。

恢复期间，WAL 文件控制命令不可用，例如`pg_start_backup`、`pg_switch_xlog`等。

可动态载入的模块可以工作，包括`pg_stat_statements`。

咨询锁在恢复期间可以正常工作，包括死锁检测。注意，咨询锁从不会被 WAL 记录，因此主库或备库上的咨询锁都不可能与 WAL 重放发生冲突。同样，也不可能在主库上获取一个咨询锁，却在备库上触发一个类似的咨询锁。咨询锁只与获取它们的那台服务器相关。

基于触发器的复制系统，如Slony、Londiste和Bucardo，根本无法在备库上运行；
不过只要它们的变更不会被发送到备库并在那里应用，它们在主库上就可以正常工作。WAL 重放并不是基于触发器的，
因此你不能把备库作为任何需要额外数据库写操作或依赖触发器的系统的中继节点。

不能分配新的 OID，不过某些UUID生成器仍可工作，只要它们不依赖于向数据库写入新的状态。

目前，在只读事务期间不允许创建临时表，
因此某些现有脚本在这种情况下将无法正常运行。这个限制可能会在未来版本中放宽。
这既涉及 SQL 标准兼容性问题，也涉及技术问题。

只有在表空间为空时DROP TABLESPACE才能成功。某些备库用户可能正在通过他们的temp_tablespaces参数使用该表空间。如果该表空间中存在临时文件，所有活动查询都将被取消，以确保临时文件被移除，这样该表空间才能被移除并且 WAL 重放可以继续。

在主库上执行DROP DATABASE或ALTER DATABASE ... SET TABLESPACE会生成一条 WAL 记录，从而强制断开备库上所有连接到该数据库的用户。这个动作会立即发生，而不受max_standby_streaming_delay设置的影响。注意，ALTER DATABASE ... RENAME不会断开用户，这在大多数情况下不会被注意到，但如果程序依赖某种基于数据库名的机制，在某些情况下可能会导致混乱。

在普通（非恢复）模式下，如果你对一个具有登录能力的角色执行DROP USER或DROP ROLE，而该用户仍然处于连接状态，那么已连接用户不会发生任何变化 – 他们会继续保持连接，不过之后不能重新连接。这种行为在恢复期间同样适用，因此在主库上执行一次DROP USER并不会断开备库上的该用户连接。

统计收集器在恢复期间保持活动状态。所有扫描、读取、块、索引使用情况等，都会在备库上正常记录。重放操作不会重复记录其在主库上产生的统计影响，因此重放一次插入不会增加 pg_stat_user_tables 的 Inserts 列。统计文件会在恢复开始时被删除，因此主库和备库的统计信息不同；这是特性，不是缺陷。

恢复期间自动清理不会运行。它会在恢复结束时正常启动。

检查点进程和后台写入进程在恢复期间是活动的。检查点进程会执行重启点（类似于主库上的检查点），后台写入进程会执行正常的块清理活动。这可能包括更新存储在备库上的提示位信息。恢复期间接受CHECKPOINT命令，不过它执行的是重启点，而不是新的检查点。

25.5.4. 热备参数参考

多个参数已经在第 25.5.2 节和第 25.5.3 节中提到过。

在主库上，可以使用wal_level和vacuum_defer_cleanup_age参数。max_standby_archive_delay和max_standby_streaming_delay在主库上设置时没有效果。

在备库上，可以使用hot_standby、max_standby_archive_delay和max_standby_streaming_delay参数。只要服务器仍处于备库模式，vacuum_defer_cleanup_age就没有效果，不过备库成为主库后，它就会起作用。

25.5.5. 注意事项

热备有若干限制。这些限制在未来的版本中可以、也很可能会被修复：

- 哈希索引上的操作目前不会写入 WAL 日志，因此重放不会更新这些索引。
- 在能够取得快照之前，必须完整了解正在运行的事务。使用大量子事务（目前超过 64 个）的事务，会把只读连接的启动推迟到持续时间最长的写事务完成之后。如果发生这种情况，会向服务器日志发送解释性消息。
- 备库查询的有效起始点是在主库的每个检查点上生成的。如果主库处于关闭状态时备库被关闭，那么在主库再次启动并在 WAL 日志中生成更多起始点之前，备库可能无法重新进入热备状态。不过在最常见的场景中，这通常不是问题。一般来说，如果主库关闭且不再可用，很可能是出现了严重故障，此时无论如何都需要把备库提升为新的主库。而在主库被有意关闭的情况下，协调好备库顺利成为新主库，本来也就是标准流程。
- 在恢复结束时，由预备事务持有的AccessExclusiveLocks将需要正常数量两倍的锁表项。如果你计划运行大量并发的预备事务，而这些事务通常会持有AccessExclusiveLocks，或者你计划运行一个会持有许多

AccessExclusiveLocks的大事务，那么建议选择更大的max_locks_per_transaction值，可能要达到主库上该参数值的两倍。如果你的max_prepared_transactions设置为 0，则完全不需要考虑这一点。

- 热备中尚不支持可串行化事务隔离级别。（详见第 13.2.3 节和第 13.4.1 节。）在热备模式下尝试把事务设置为可串行化隔离级别会产生错误。

第 26 章 恢复配置

本章介绍 recovery.conf 文件中可用的设置。它们只在恢复期间生效。如果之后还要进行恢复，必须重新设置这些参数。恢复开始后，就不能再更改它们。

recovery.conf 中的设置使用 name = 'value' 格式，每行指定一个参数。井号（#）表示该行余下内容为注释。要在参数值中嵌入单引号，请写两个单引号（''）。

安装目录的 share/ 目录中提供了示例文件 share/recovery.conf.sample。

26.1. 归档恢复设置

restore_command (string)

用于获取 WAL 文件系列的一个已归档段的本地 shell 命令。这个参数是归档恢复所必需的，但是对于流复制是可选的。在该字符串中的任何%f
会被替换为从归档中获得的文件的名字，并且任何%p
会被在服务器上的复制目标路径名替换（该路径名是相对于当前工作目录的，
即集簇的数据目录）。任何%r会被包含上一个可用重启点的文件的名字所替换。
在那些必须被保留用于使得一次恢复变成可重启的文件中，这个文件是其中最早的一个，
因此这个信息可以被用来把归档截断为支持从当前恢复重启所需的最小值。%r
通常只被温备配置（见第 25.2 节）所使用。要嵌入一个真正的%字符，需要写成%%。

很重要的一点是，该命令只有在成功时才返回一个为零的退出状态。该命令
将会被要求获取归档中不存在的文件；遇到这种情况时，它必须返回非零。示例：

```
restore_command = 'cp /mnt/server/archivedir/%f "%p"'
restore_command = 'copy "C:\\server\\archivedir\\%f" "%p"'  #
 Windows
```

archive_cleanup_command (string)

这个可选参数指定了一个 shell 命令，它将在每一个重启点被执行。archive_cleanup_command的目的是提供一种清除不再被备库需要的旧的已归档 WAL 文件的机制。任何%r会被替换为包含最后一个可用重启点的文件的名称。
那是使一次恢复变成可重启的所必须被保留的最早的文件，并且因此比%r
更早的所有文件可以被安全地移除。
这个信息可以被用来把归档截断为支持从当前恢复重启所需的最小值。
对于单一备库配置，pg_archivecleanup模块常常被用在archive_cleanup_command中，例如：

```
archive_cleanup_command = 'pg_archivecleanup /mnt/server/
archivedir %r'
```

但是注意，如果多个备库正在从同一个归档目录中恢复，
你将需要保证只有当所有服务器都不再需要这些 WAL 文件时才会删除它们。archive_cleanup_command通常被用于一种温备配置（见第 25.2 节）中。要在该命令中嵌入一个真正的%字符，需要写成%%。

如果该命令返回一个非零退出状态，则将会写出一个警告日志消息。
一个例外是如果该命令被一个信号或者一个 shell 错误（例如命令未找到）终止，则会抛出一个致命错误。

recovery_end_command (string)

此参数指定一个 shell 命令，在恢复结束时仅执行一次。此参数可选。recovery_end_command 用于提供复制或恢复后的清理机制。与archive_cleanup_command一样，所有 %r 都会被替换为包含最后一个有效重启点的文件的名称。

如果该命令返回一个非零退出状态，
则一个警告日志消息将被写出并且不管怎样该数据库将继续启动。
一个例外是如果该命令被一个信号或者 shell 错误（例如命令未找到）中止，该数据库将不会继续启动。

26.2. 恢复目标设置

`recovery_target_name` (`string`)

这个参数指定（`pg_create_restore_point()`所创建）的已命名的恢复点，
恢复将进行到该恢复点。在 `recovery_target_name`、recovery_target_time或recovery_target_xid中，最多只能指定一个。默认行为是恢复到 WAL 日志的末尾。

`recovery_target_time` (`timestamp`)

此参数指定恢复要进行到的时间戳。在 `recovery_target_time`、recovery_target_name或recovery_target_xid中，最多只能指定一个。默认行为是恢复到 WAL 日志的末尾。精确的停止点还受recovery_target_inclusive影响。

`recovery_target_xid` (`string`)

此参数指定恢复要进行到的事务 ID。请注意，虽然事务 ID 在事务开始时按顺序分配，但事务完成的顺序可能与 ID 的数值顺序不同。恢复的事务是那些在指定事务之前提交的事务，并可选择包括指定事务本身。在 `recovery_target_xid`、recovery_target_name或recovery_target_time中，最多只能指定一个。默认行为是恢复到 WAL 日志的末尾。精确的停止点还受recovery_target_inclusive影响。

`recovery_target_inclusive` (`boolean`)

指定是在所设恢复目标之后立即停止（`true`），还是在其之前立即停止（`false`）。
在指定recovery_target_time或recovery_target_xid时适用。
此设置控制恢复是否包含提交时间或事务 ID 分别恰好等于目标值的事务。默认值为 `true`。

`recovery_target_timeline` (`string`)

指定恢复到某条特定时间线。默认沿着制作基础备份时的当前时间线恢复。设为 `latest` 时，会恢复到归档中找到的最新时间线，这对备库很有用。除此之外，只有在复杂的再次恢复场景中才需要设置此参数：需要返回的状态本身就是一次时间点恢复之后达到的状态。相关讨论参见第 24.3.5 节。

`pause_at_recovery_target` (`boolean`)

指定在达到恢复目标时恢复是否应暂停。默认值为 true。这是为了允许对数据库执行查询，以检查这个恢复目标是否为最合适的恢复位置。暂停的状态可以使用`pg_xlog_replay_resume()`（见表 9.62）继续，这会让恢复终结。如果这个恢复目标不是想要的停止点，那么关闭服务器，将恢复目标设置改为一个稍后的目标并且重启以继续恢复。

如果未启用hot_standby，或者未设置恢复目标，此设置不起作用。

26.3. 备库设置

`standby_mode` (`boolean`)

指定是否将 PostgreSQL 服务器作为备库启动。如果此参数为 `on`，服务器到达已归档 WAL 的末尾时不会停止恢复，而会继续尝试使用 `restore_command` 获取新的 WAL 段，和/或按照 `primary_conninfo` 设置连接主库，以继续恢复。

primary_conninfo (string)

指定备库连接主库所用的连接字符串。该字符串采用 libpq 的 PQconnectdb 函数所接受的格式，见第 31.1 节。如果某个选项未在字符串中指定，就会检查相应的环境变量（参见第 31.14 节）。如果环境变量也未设置，则使用默认值。

连接字符串应指定主库的主机名（或地址）；如果端口号与备库的默认端口不同，也应指定端口号。还应指定一个用户名，对应主库上具有适当权限的角色（参见第 25.2.5.1 节）。如果主库要求密码认证，还需要提供密码。密码可以放在 primary_conninfo 字符串中，也可以放在备库上单独的 ~/.pgpass 文件中（使用 replication 作为数据库名）。不要在 primary_conninfo 字符串中指定数据库名。

如果 standby_mode 为 off，此设置不起作用。

trigger_file (string)

指定一个触发文件，其出现会使备库结束恢复。即使未设置此值，也仍可以使用 pg_ctl promote 提升备库。如果 standby_mode 为 off，此设置不起作用。

第 27 章 监控数据库活动

数据库管理员常常会问："系统现在正在做什么？"本章将讨论如何回答这个问题。

有几种工具可用于监控数据库活动和分析性能。
本章的大部分内容将用于描述PostgreSQL的统计收集器，但也不应忽略常见的 Unix 监控程序，例如ps、top、iostat和vmstat。此外，一旦确定了某个查询性能不佳，
可能需要使用PostgreSQL的EXPLAIN命令进行进一步调查。第 14.1 节讨论了EXPLAIN以及理解单个查询行为的其他方法。

27.1. 标准 Unix 工具

在大多数 Unix 平台上，PostgreSQL会修改由ps报告的命令标题，以便轻松识别各个服务器进程。以下是一个显示示例：

```
$ ps auxww | grep ^postgres
postgres  15551  0.0  0.1  57536  7132 pts/0    S    18:02   0:00
 postgres -i
postgres  15554  0.0  0.0  57536  1184 ?        Ss   18:02   0:00
 postgres: writer process
postgres  15555  0.0  0.0  57536   916 ?        Ss   18:02   0:00
 postgres: checkpointer process
postgres  15556  0.0  0.0  57536   916 ?        Ss   18:02   0:00
 postgres: wal writer process
postgres  15557  0.0  0.0  58504  2244 ?        Ss   18:02   0:00
 postgres: autovacuum launcher process
postgres  15558  0.0  0.0  17512  1068 ?        Ss   18:02   0:00
 postgres: stats collector process
postgres  15582  0.0  0.0  58772  3080 ?        Ss   18:04   0:00
 postgres: joe runbug 127.0.0.1 idle
postgres  15606  0.0  0.0  58772  3052 ?        Ss   18:07   0:00
 postgres: tgl regression [local] SELECT waiting
postgres  15610  0.0  0.0  58772  3056 ?        Ss   18:07   0:00
 postgres: tgl regression [local] idle in transaction
```

（ps的适当调用方式随平台而异，显示内容的细节也是如此。此示例来自较新的 Linux 系统。）这里列出的第一个进程是主服务器进程。
显示的命令参数与启动该进程时使用的参数相同。
接下来的五个进程是主进程自动启动的后台工作进程。
（如果系统被设置为不启动统计收集器，则不会出现"stats collector"进程；同样，也可以禁用"autovacuum launcher"进程。）其余每个进程都是处理一个客户端连接的服务器进程。每个这样的进程都将其命令行显示设置为以下形式：

```
postgres: user database host activity
```

在客户端连接的整个生命周期内，用户、数据库和（客户端）主机这几项保持不变，
但活动指示符会改变。活动可以是idle（即等待客户端命令）、idle in transaction（在BEGIN块内等待客户端），或者命令类型名称，例如SELECT。此外，
服务器进程当前若正在等待另一会话持有的锁，还会附加waiting。在上面的示例中，
可以推断进程 15606 正在等待进程 15610 完成事务，从而释放某个锁。（进程15610必定是阻塞者，因为不存在其他活动会话。在更复杂的情况下，必须查看pg_locks系统视图，才能确定谁在阻塞谁。）

如果您关闭了update_process_title，则活动指示器不会更新；
进程标题仅在新进程启动时设置一次。在某些平台上，这可以节省可观的每命令开销；
在其他平台上，则微不足道。

> **提示**
>
> Solaris需要特殊处理。您必须使用/usr/ucb/ps，而不是/bin/ps。
> 您还必须使用两个w标志，而不仅仅是一个。此外，您对postgres
> 命令的原始调用必须具有比每个服务器进程提供的更短的ps状态显示。
> 如果未满足这三个条件，每个服务器进程的ps输出都会显示原始的postgres
> 命令行。

27.2. 统计收集器

PostgreSQL的统计收集器是一个支持收集和报告服务器活动信息的子系统。目前，
对表和索引的访问会按磁盘块和行两种口径计数。每个表中的总行数，
以及每个表的清理和分析操作的信息也被计数。
它还可以统计对用户定义函数的调用和每个函数中花费的总时间。

PostgreSQL还支持报告其他服务器进程当前正在执行的确切命令。
此功能独立于收集器进程。

27.2.1. 统计信息收集配置

由于收集统计信息会增加查询执行的开销，因此可以配置系统是否收集信息。这由通常在
postgresql.conf中设置的配置参数控制（有关设置配置参数的详细信息，
请参阅第 18 章）。

参数track_activities启用对任意服务器进程当前执行命令的监控。

参数track_counts控制是否收集关于表和索引访问的统计信息。

参数track_functions启用对用户定义函数使用的跟踪。

参数track_io_timing启用对块读取和写入时间的监控。

通常，这些参数会设置在postgresql.conf中，以便它们适用于所有服务器进程，
但也可以在单个会话中使用SET命令打开或关闭它们。
（为防止普通用户隐藏其活动不被管理员发现，只有超级用户才能使用SET更改这些参数。）

统计收集器通过临时文件把收集到的信息传递给其他PostgreSQL进程。
这些文件默认存放在由stats_temp_directory参数指定的目录中，即pg_stat_tmp。
为了获得更好的性能，可以把stats_temp_directory指向基于 RAM 的文件系统，以减少
物理 I/O 需求。当服务器正常关闭时，统计信息数据的永久副本会存储在global子目录中，
以便在服务器重启后保留这些统计信息。当服务器启动时执行恢复时（例如立即关闭后、
服务器崩溃后以及进行时间点恢复时），所有统计计数器都会被重置。

27.2.2. 查看收集到的统计信息

有几个预定义视图列在表 27.1中，可用于显示统计信息收集的结果。也可以像第 27.2.3 节中
所述那样，使用底层的统计函数构建自定义视图。

当使用统计信息来监视当前活动时，重要的是要意识到信息不会立即更新。
每个单独的服务器进程会在转为空闲之前把新的统计计数发送给收集器，
因此仍在进行中的查询或事务不会影响显示的总计。并且，收集器本身最多每隔PGSTAT_
STAT_INTERVAL毫秒才会发出一份新报告一次（除非在构建服务器时做过修改，否则为 500
毫秒）。因此，显示的信息会滞后于实际活动。然而，由track_activities
收集的当前查询信息始终是最新的。

另一个重要点是，当要求服务器进程显示这些统计信息中的任何一种时，
它会先取回收集器进程最近发出的报告，然后在当前事务结束之前，
对所有统计视图和函数一直使用这一快照。因此，只要你继续当前事务，

统计信息就会显示静态信息。类似地，在事务中第一次请求这类当前查询信息时，就会收集所有会话当前查询的信息，并且在整个事务期间显示的都是同一份信息。这是一个特性，而不是一个错误，因为它允许你对统计信息执行多个查询并对结果做关联，而不必担心数字在你眼前变化。但如果希望每个查询都看到新的结果，请确保在任何事务块之外执行这些查询。或者，也可以调用pg_stat_clear_snapshot()，它会丢弃当前事务的统计快照（如果有）。下一次使用统计信息时，就会获取一个新的快照。

事务还可以在视图pg_stat_xact_all_tables、pg_stat_xact_sys_tables、pg_stat_xact_user_tables和 pg_stat_xact_user_functions中看到自己的统计信息（这些统计信息尚未传送给收集器）。这些数字不像上面所述那样起作用；相反，它们在事务期间持续更新。

表 27.1. 标准统计视图

视图名称	描述
pg_stat_activity	每个服务器进程一行，显示与该进程当前活动有关的信息，例如状态和当前查询。详见pg_stat_activity。
pg_stat_bgwriter	只有一行，显示后台写入器活动的统计信息。详见pg_stat_bgwriter。
pg_stat_database	每个数据库一行，显示整个数据库的统计信息。详见pg_stat_database。
pg_stat_all_tables	当前数据库中的每个表一行，显示访问该表的统计信息。详见pg_stat_all_tables。
pg_stat_sys_tables	和pg_stat_all_tables一样，但只显示系统表。
pg_stat_user_tables	和pg_stat_all_tables一样，但只显示用户表。
pg_stat_xact_all_tables	与 pg_stat_all_tables 相似，但只统计当前事务中截至目前执行的操作（这些操作尚未计入 pg_stat_all_tables 及相关视图）。此视图不包含存活行和死行数量以及清理、分析操作的列。
pg_stat_xact_sys_tables	和pg_stat_xact_all_tables一样，但只显示系统表。
pg_stat_xact_user_tables	和pg_stat_xact_all_tables一样，但只显示用户表。
pg_stat_all_indexes	当前数据库中的每个索引一行，显示访问该索引的统计信息。详见pg_stat_all_indexes。
pg_stat_sys_indexes	和pg_stat_all_indexes一样，但只显示系统表上的索引。
pg_stat_user_indexes	和pg_stat_all_indexes一样，但只显示用户表上的索引。
pg_statio_all_tables	当前数据库中的每个表一行，显示该表的 I/O 统计信息。详见pg_statio_all_tables。
pg_statio_sys_tables	和pg_statio_all_tables一样，但只显示系统表。
pg_statio_user_tables	和pg_statio_all_tables一样，但只显示用户表。
pg_statio_all_indexes	当前数据库中的每个索引一行，显示该索引的 I/O 统计信息。详见pg_statio_all_indexes。
pg_statio_sys_indexes	和pg_statio_all_indexes一样，但只显示系统表上的索引。
pg_statio_user_indexes	和pg_statio_all_indexes一样，但只显示用户表上的索引。

视图名称	描述
pg_statio_all_sequences	当前数据库中的每个序列一行，显示该序列的 I/O 统计信息。详见pg_statio_all_sequences。
pg_statio_sys_sequences	和pg_statio_all_sequences一样，但只显示系统序列（目前没有定义系统序列，因此这个视图总是为空）。
pg_statio_user_sequences	和pg_statio_all_sequences一样，但只显示用户序列。
pg_stat_user_functions	每个被跟踪的函数一行，显示该函数执行的统计信息。详见pg_stat_user_functions。
pg_stat_xact_user_functions	和pg_stat_user_functions相似，但是只统计在当前事务期间的调用（还没有被包括在pg_stat_user_functions中）。
pg_stat_replication	每个 WAL 发送进程一行，显示向其所连接的备库进行复制的统计信息。详见pg_stat_replication。
pg_stat_database_conflicts	每个数据库一行，显示备库上因与恢复冲突而取消查询的数据库范围统计信息。详见pg_stat_database_conflicts。

每个索引的统计信息对于判断哪些索引正在被使用以及它们有多有效尤其有用。

pg_statio_视图主要用于判断缓冲区缓存的有效性。
如果实际磁盘读取次数远少于缓冲区命中次数，缓存就能满足大多数读取请求，
而无需调用内核。不过，这些统计信息并不能反映全部情况：由于PostgreSQL处理磁盘 I/O 的方式，不在PostgreSQL缓冲区缓存中的数据仍可能驻留在内核的 I/O 缓存中，因而仍可在不执行物理读取的情况下取得。对于希望更详细地了解PostgreSQL I/O 行为的用户，建议将PostgreSQL统计收集器与能够深入查看内核 I/O 处理情况的操作系统工具结合使用。

表 27.2. pg_stat_activity 视图

列	类型	描述
datid	oid	这个后端连接到的数据库的OID
datname	name	这个后端连接到的数据库的名称
pid	integer	这个后端的进程 ID
usesysid	oid	登录到这个后端的用户的 OID
usename	name	登录到此后端的用户名称
application_name	text	连接到此后端的应用名称
client_addr	inet	连接到这个后端的客户端的 IP 地址。如果这个字段为空值，它表示客户端通过服务器机器上的一个 Unix 套接字连接或者这是一个内部进程，如自动清理。
client_hostname	text	已连接的客户端的主机名，由client_addr的反向 DNS 查找报告。这个字段将只对 IP 连接非空，并且只有log_hostname被启用时才会非空。
client_port	integer	客户端用于与此后端通信的 TCP 端口号；如果使用 Unix 套接字，则为-1
backend_start	timestamp with time zone	这个进程被启动的时间，也就是客户端连接到服务器的时间
xact_start	timestamp with time zone	这个进程的当前事务被启动的时间，如果没有活动事务则为空值。如果当前查询是其事务中的

列	类型	描述
		第一个查询，这一列等于query_start列。
query_start	timestamp with time zone	当前活动查询开始的时间；如果state不是active，则为上一个查询的开始时间
state_change	timestamp with time zone	state上一次发生改变的时间
waiting	boolean	如果此后端当前正在等待一个锁，则为真
state	text	该后端当前的总体状态。可能的值为： • active：后端正在执行一个查询。 • idle：后端正在等待新的客户端命令。 • idle in transaction：后端处于事务中，但当前没有执行查询。 • idle in transaction (aborted)：该状态与idle in transaction类似，但事务中的某个语句导致了错误。 • fastpath function call：后端正在执行一个快速路径函数。 • disabled：如果在此后端中禁用了track_activities，就会报告此状态。
query	text	这个后端最近查询的文本。如果state为active，这个字段显示当前正在执行的查询。在所有其他状态下，它显示上一个被执行的查询。

pg_stat_activity视图每个服务器进程将有一行，显示与该进程当前活动相关的信息。

> **注意**
>
> waiting和state列彼此独立。如果某个后端处于active状态，它可能正在waiting（等待），也可能没有。如果状态为active且waiting为真，就意味着某个查询正在执行，但在系统中的某处被锁阻塞。

表 27.3. pg_stat_bgwriter 视图

列	类型	描述
checkpoints_timed	bigint	已执行的计划检查点数
checkpoints_req	bigint	已执行的请求检查点数
checkpoint_write_time	double precision	检查点处理中将文件写入磁盘部分所花费的总时间，单位为毫秒
checkpoint_sync_time	double precision	检查点处理中将文件同步到磁盘部分所花费的总时间，单位为毫秒

列	类型	描述
buffers_checkpoint	bigint	检查点期间写入的缓冲区数
buffers_clean	bigint	后台写入器写入的缓冲区数量
maxwritten_clean	bigint	后台写入器因写入了过多缓冲区而停止清理扫描的次数
buffers_backend	bigint	后端直接写入的缓冲区数
buffers_backend_fsync	bigint	后端必须自行执行fsync调用的次数（通常即使后端自行写入，也由后台写入器处理这些调用）
buffers_alloc	bigint	分配的缓冲区数量
stats_reset	timestamp with time zone	这些统计信息上次被重置的时间

pg_stat_bgwriter 视图始终只有一行，包含整个集簇的全局数据。

表 27.4. pg_stat_database 视图

列	类型	描述
datid	oid	数据库的OID
datname	name	数据库的名称
numbackends	integer	当前连接到此数据库的后端数。这是该视图中唯一返回反映当前状态的值的列；所有其他列返回的都是自上次重置以来累积的值。
xact_commit	bigint	此数据库中已提交的事务数
xact_rollback	bigint	该数据库中已回滚的事务数
blks_read	bigint	在该数据库中读取的磁盘块数
blks_hit	bigint	在缓冲区缓存中发现磁盘块、因而无需读取的次数（这里只统计PostgreSQL 缓冲区缓存中的命中，不包括操作系统文件系统缓存中的命中）
tup_returned	bigint	此数据库中的查询返回的行数
tup_fetched	bigint	此数据库中的查询获取的行数
tup_inserted	bigint	查询在该数据库中插入的行数
tup_updated	bigint	这个数据库中查询更新的行数
tup_deleted	bigint	这个数据库中被查询删除的行数
conflicts	bigint	由于与此数据库中的恢复冲突而被取消的查询数。（冲突只会发生在备库上；请参阅pg_stat_database_conflicts。）
temp_files	bigint	这个数据库中查询创建的临时文件的数量。所有临时文件都将被计数，而不顾及临时文件为什么被创建（例如，排序或散列），也不考虑log_temp_files设置。
temp_bytes	bigint	这个数据库中的查询写入临时文件的数据总量。所有临时文件都将被计数，而不考虑临时文件为什么被创建，也不考虑log_temp_files设置。
deadlocks	bigint	在此数据库中检测到的死锁数

列	类型	描述
blk_read_time	double precision	此数据库的后端读取数据文件块所花费的时间，单位为毫秒
blk_write_time	double precision	此数据库的后端写入数据文件块所花费的时间，单位为毫秒
stats_reset	timestamp with time zone	这些统计数据最后一次重置的时间

pg_stat_database视图为集簇中的每个数据库包含一行，显示整个数据库的统计信息。

表 27.5. pg_stat_all_tables 视图

列	类型	描述
relid	oid	表的OID
schemaname	name	该表所在的模式的名称
relname	name	这个表的名称
seq_scan	bigint	在此表上启动的顺序扫描数
seq_tup_read	bigint	顺序扫描获取的存活行数
idx_scan	bigint	对这个表发起的索引扫描数
idx_tup_fetch	bigint	索引扫描获取的存活行数
n_tup_ins	bigint	插入的行数
n_tup_upd	bigint	更新的行数
n_tup_del	bigint	删除的行数
n_tup_hot_upd	bigint	HOT 更新的行数（即不需要单独更新索引）
n_live_tup	bigint	活的行的估计数量
n_dead_tup	bigint	死行的估计数量
last_vacuum	timestamp with time zone	最后一次手动清理这个表的时间（不包括VACUUM FULL）
last_autovacuum	timestamp with time zone	这个表最后一次被自动清理守护进程清理的时间
last_analyze	timestamp with time zone	上一次手动分析这个表的时间
last_autoanalyze	timestamp with time zone	自动清理守护进程最后一次分析这个表的时间
vacuum_count	bigint	这个表被手动清理的次数（VACUUM FULL不计数）
autovacuum_count	bigint	这个表被自动清理守护进程清理的次数
analyze_count	bigint	手动分析这个表的次数
autoanalyze_count	bigint	这个表被自动清理守护进程分析的次数

pg_stat_all_tables视图将为当前数据库中的每一个表（包括 TOAST 表）包含一行，该行显示与对该表的访问相关的统计信息。pg_stat_user_tables和pg_stat_sys_tables视图包含相同的信息，但是被过滤得分别只显示用户和系统表。

表 27.6. pg_stat_all_indexes 视图

列	类型	描述
relid	oid	对于此索引的表的OID
indexrelid	oid	这个索引的OID

列	类型	描述
schemaname	name	这个索引所在的模式名称
relname	name	这个索引的表的名称
indexrelname	name	这个索引的名称
idx_scan	bigint	在这个索引上开启的索引扫描的数量
idx_tup_read	bigint	扫描此索引返回的索引项数
idx_tup_fetch	bigint	使用此索引进行简单索引扫描获取的存活表行数

pg_stat_all_indexes视图将为当前数据库中的每个索引包含一行，该行显示关于对该索引访问的统计信息。pg_stat_user_indexes和pg_stat_sys_indexes视图包含相同的信息，但是被过滤得只分别显示用户和系统索引。

索引可以被简单索引扫描或者“位图”索引扫描使用。在一次位图扫描中，多个索引的输出可以被通过 AND 或 OR 规则组合，因此当使用一次位图扫描时难以将取得的个体堆行与特定的索引关联起来。因此，一次位图扫描会增加它使用的索引的pg_stat_all_indexes.idx_tup_read计数，并且为该表增加pg_stat_all_tables.idx_tup_fetch计数，但是它不影响pg_stat_all_indexes.idx_tup_fetch。

> **注意**
>
> 即使不用位图扫描，idx_tup_read和idx_tup_fetch计数也可能不同，因为idx_tup_read统计从该索引取得的索引项而idx_tup_fetch统计从表取得的存活行。如果使用该索引取得了任何死亡行或还未提交的行，或者如果通过一次仅索引扫描的方式避免了任何堆获取，后者将较小。

表 27.7. pg_statio_all_tables 视图

列	类型	描述
relid	oid	表的OID
schemaname	name	该表所在的模式的名称
relname	name	这个表的名称
heap_blks_read	bigint	从该表中读取的磁盘块的数量
heap_blks_hit	bigint	该表中的缓冲区命中数
idx_blks_read	bigint	从这个表上所有索引读取的磁盘块数
idx_blks_hit	bigint	这个表上所有索引中的缓冲区命中数
toast_blks_read	bigint	从这个表的TOAST表中读取的磁盘块的数量（如果有的话）
toast_blks_hit	bigint	这个表的TOAST表中的缓冲区命中数（如果有的话）
tidx_blks_read	bigint	从这个表的TOAST表索引中读取的磁盘块的数量（如果有的话）
tidx_blks_hit	bigint	这个表的TOAST表索引中的缓冲区命中数（如果有的话）

pg_statio_all_tables视图将为当前数据库中的每个表（包括 TOAST 表）包含一行，该行显示指定表上有关 I/O 的统计信息。pg_statio_user_tables和pg_statio_sys_tables视图包含相同的信息，但是被过滤得分别只显示用户表和系统表。

表 27.8. pg_statio_all_indexes 视图

列	类型	描述
relid	oid	对于此索引的表的OID
indexrelid	oid	这个索引的OID
schemaname	name	这个索引所在的模式名称
relname	name	这个索引的表的名称
indexrelname	name	这个索引的名称
idx_blks_read	bigint	从此索引中读取的磁盘块的数量
idx_blks_hit	bigint	此索引中的缓冲区命中数

pg_statio_all_indexes视图将为当前数据库中的每个索引包含一行，该行显示指定索引上有关 I/O 的统计信息。pg_statio_user_indexes和pg_statio_sys_indexes视图包含相同的信息，但是被过滤得分别只显示用户索引和系统索引。

表 27.9. pg_statio_all_sequences 视图

列	类型	描述
relid	oid	序列的OID
schemaname	name	此序列所在的模式的名称
relname	name	此序列的名称
blks_read	bigint	从这个序列中读取的磁盘块的数量
blks_hit	bigint	在此序列中的缓冲区命中数

pg_statio_all_sequences视图将为当前数据库中的每个序列包含一行，该行显示在指定序列上有关 I/O 的统计信息。

表 27.10. pg_stat_user_functions 视图

列	类型	描述
funcid	oid	函数的OID
schemaname	name	这个函数所在的模式的名称
funcname	name	这个函数的名称
calls	bigint	这个函数已经被调用的次数
total_time	double precision	在这个函数以及它所调用的其他函数中花费的总时间，以毫秒计
self_time	double precision	在这个函数本身花费的总时间，不包括被它调用的其他函数，以毫秒计

pg_stat_user_functions视图将为每一个被追踪的函数包含一行，该行显示有关该函数执行的统计信息。track_functions参数控制到底哪些函数被跟踪。

表 27.11. pg_stat_replication 视图

列	类型	描述
pid	integer	一个 WAL 发送进程的进程 ID
usesysid	oid	登录到这个 WAL 发送进程的用户的 OID
usename	name	登录到这个 WAL 发送进程的用户的名称

列	类型	描述
application_name	text	连接到这个 WAL 发送进程的应用的名称
client_addr	inet	连接到这个 WAL 发送进程的客户端的 IP 地址。如果这个字段为空值，它表示该客户端通过服务器机器上的一个Unix 套接字连接。
client_hostname	text	已连接的客户端的主机名，由client_addr的反向 DNS 查找报告。这个字段将只对 IP 连接非空，并且只有log_hostname被启用时才会非空。
client_port	integer	客户端用来与这个 WAL 发送进程通讯的 TCP 端口号，如果使用Unix 套接字则为-1
backend_start	timestamp with time zone	这个进程开始的时间，即客户端是何时连接到这个WAL 发送进程的。
state	text	当前 WAL 发送进程的状态
sent_location	text	在这个连接上发送的最后一个事务日志的位置
write_location	text	该备库已写入磁盘的最后一个事务日志位置
flush_location	text	该备库已刷入磁盘的最后一个事务日志位置
replay_location	text	已在该备库数据库中重放的最后一个事务日志位置
sync_priority	integer	该备库被选为同步备库的优先级。
sync_state	text	此备库的同步状态

pg_stat_replication视图为每个 WAL 发送进程包含一行，显示有关复制到该发送进程所连接备库的统计信息。这里只列出直接连接的备库；不包含下游备库的信息。

表 27.12. pg_stat_database_conflicts 视图

列	类型	描述
datid	oid	数据库的OID
datname	name	数据库的名称
confl_tablespace	bigint	这个数据库中由于删除表空间而取消的查询的数量
confl_lock	bigint	此数据库中由于锁定超时而被取消的查询数
confl_snapshot	bigint	此数据库中由于旧快照而取消的查询数
confl_bufferpin	bigint	此数据库中由于缓冲区被钉住而被取消的查询数
confl_deadlock	bigint	此数据库中由于死锁而被取消的查询数

pg_stat_database_conflicts视图为每个数据库包含一行，
显示备库上因与恢复冲突而取消查询的数据库范围统计信息。
由于主库上不会发生这些冲突，该视图只会在备库上包含信息。

27.2.3. 统计函数

其他查看统计信息的方法是直接使用查询，这些查询使用上述标准视图用到的底层统计信息访问函数。如要了解如函数名等细节，可参考标准视图的定义（例如，在psql中你可以发出`\d+ pg_stat_activity`）。针对每一个数据库统计信息的访问函数把一个数据库 OID 作为参数来标识要报告哪个数据库。而针对每个表和每个索引的函数要求表或索引 OID。针对每个函数统计信息的函数用一个函数 OID。注意只有在当前数据库中的表、索引和函数才能被这些函数看到。

与统计收集器相关的其他功能在表 27.13中列出。

表 27.13. 附加统计函数

函数	返回类型	描述
`pg_backend_pid()`	`integer`	处理当前会话的服务器进程的进程 ID
`pg_stat_get_activity(integer)`	`setof record`	返回一条记录，包含指定 PID 的后端的信息；如果指定`NULL`，则为系统中的每个活动后端返回一条记录。返回字段是`pg_stat_activity`视图字段的子集。
`pg_stat_clear_snapshot()`	`void`	丢弃当前统计信息快照
`pg_stat_reset()`	`void`	将当前数据库的所有统计计数器重置为零（需要超级用户权限）
`pg_stat_reset_shared(text)`	`void`	根据参数将某些集簇范围的统计计数器重置为零（需要超级用户权限）。调用`pg_stat_reset_shared('bgwriter')`会将`pg_stat_bgwriter`视图中显示的所有计数器清零。
`pg_stat_reset_single_table_counters(oid)`	`void`	将当前数据库中单个表或索引的统计信息重置为零（需要超级用户权限）
`pg_stat_reset_single_function_counters(oid)`	`void`	将当前数据库中单个函数的统计信息重置为零（需要超级用户权限）

`pg_stat_get_activity`是`pg_stat_activity`视图的底层函数，它返回一个行集合，其中包含有关每个后端进程所有可用的信息。有时只获得该信息的一个子集可能会更方便。在那些情况中，可以使用一组更老的针对每个后端的统计访问函数，这些显示在表 27.14中。这些访问函数使用一个后端 ID 号，范围从 1 到当前活动后端数目。函数`pg_stat_get_backend_idset`提供了一种方便的方法为每个活动后端产生一行来调用这些函数。例如，要显示PID以及所有后端当前的查询：

```
SELECT pg_stat_get_backend_pid(s.backendid) AS pid,
       pg_stat_get_backend_activity(s.backendid) AS query
    FROM (SELECT pg_stat_get_backend_idset() AS backendid) AS s;
```

表 27.14. 按后端统计函数

函数	返回类型	描述
`pg_stat_get_backend_idset()`	`setof integer`	当前活动后端的 ID 编号集合（从 1 到活动后端数）
`pg_stat_get_backend_activity(integer)`	`text`	此后端最近查询的文本
`pg_stat_get_backend_activity_start(integer)`	`timestamp with time zone`	最近查询的开始时间

函数	返回类型	描述
pg_stat_get_backend_client_addr(integer)	inet	连接到此后端的客户端的 IP 地址
pg_stat_get_backend_client_port(integer)	integer	客户端用于通信的 TCP 端口号
pg_stat_get_backend_dbid(integer)	oid	这个后端连接到的数据库的OID
pg_stat_get_backend_pid(integer)	integer	这个后端的进程 ID
pg_stat_get_backend_start(integer)	timestamp with time zone	此进程的启动时间
pg_stat_get_backend_userid(integer)	oid	登录到这个后端的用户的 OID
pg_stat_get_backend_waiting(integer)	boolean	如果此后端当前正在等待一个锁，则为真
pg_stat_get_backend_xact_start(integer)	timestamp with time zone	当前事务的开始时间

27.3. 查看锁

监控数据库活动的另外一个有用的工具是pg_locks系统表。
这样就允许数据库管理员查看锁管理器中当前存在的锁的信息。例如，
这个功能可以被用于：

- 查看当前存在的所有锁、在一个特定数据库中的关系上所有的锁、
  在一个特定关系上所有的锁，或者由一个特定PostgreSQL会话持有的所有的锁。
- 判断当前数据库中带有最多未授予锁的关系（它可能是数据库客户端之间的竞争来源）。
- 判断锁竞争给数据库总体性能带来的影响，以及锁竞争随着整个数据库流量的变化范围。

pg_locks视图的细节在第 45.58 节中。更多有关PostgreSQL的锁和管理并发性的信息，请参考第 13 章。

27.4. 动态追踪

PostgreSQL提供了功能来支持数据库服务器的动态追踪。
这样就允许在代码中的特定点上调用外部工具来追踪执行过程。

一些探针或追踪点已经被插入在源代码中。
这些探针的目的是被数据库开发者和管理员使用。默认情况下，探针不被编译到
PostgreSQL中；用户需要显式地告诉配置脚本使得探针可用。

目前，只有DTrace[1]工具被支持，它可用于 OpenSolaris、Solaris 10 和 Mac OS X Leopard。预计 DTrace 将来也会在 FreeBSD 以及可能的其他操作系统上可用。Linux 的 SystemTap[2]项目也提供了一种 DTrace 等价物。支持其他动态追踪工具在理论上可以通过改变src/include/utils/probes.h中的宏定义实现。

27.4.1. 为动态追踪编译

默认情况下，探针是不可用的，因此你将需要显式地告诉配置脚本让探针在
PostgreSQL中可用。要包括 DTrace 支持，在配置时指定--enable-dtrace。更多信息请见第 15.4 节。

[1] http://opensolaris.org/os/community/dtrace/
[2] http://sourceware.org/systemtap/

27.4.2. 内置探针

如表 27.15所示，源代码中提供了一些标准探针。表 27.16显示了在探针中使用的类型。当然，可以增加更多探针来增强PostgreSQL的可观测性。

表 27.15. 内置 DTrace 探针

名称	参数	描述
transaction-start	(LocalTransactionId)	在一个新事务开始时触发的探针。arg0 是事务 ID。
transaction-commit	(LocalTransactionId)	在一个事务成功完成时触发的探针。arg0 是事务 ID。
transaction-abort	(LocalTransactionId)	当一个事务失败结束时触发的探针。arg0 是事务 ID。
query-start	(const char *)	当一个查询的处理开始时触发的探针。arg0 是查询字符串。
query-done	(const char *)	当一个查询的处理完成时触发的探针。arg0 是查询字符串。
query-parse-start	(const char *)	当一个查询的解析开始时触发的探针。arg0 是查询字符串。
query-parse-done	(const char *)	当一个查询的解析完成时触发的探针。arg0 是查询字符串。
query-rewrite-start	(const char *)	当一个查询的重写开始时触发的探针。arg0 是查询字符串。
query-rewrite-done	(const char *)	当一个查询的重写完成时触发的探针。arg0 是查询字符串。
query-plan-start	()	当一个查询的规划开始时触发的探针。
query-plan-done	()	当一个查询的规划完成时触发的探针。
query-execute-start	()	当一个查询的执行开始时触发的探针。
query-execute-done	()	当一个查询的执行完成时触发的探针。
statement-status	(const char *)	任何时候当服务器进程更新它的 `pg_stat_activity.status` 时触发的探针。arg0 是新的状态字符串。
checkpoint-start	(int)	当一个检查点开始时触发的探针。arg0 传递位标志来区分不同的检查点类型，例如关闭（shutdown）、立即（immediate）或强制（force）。
checkpoint-done	(int, int, int, int, int)	当一个检查点完成时触发的探针（下面列出的探针会在检查点处理过程中依次触发）。arg0 是已写入的缓冲区数量。arg1 是缓冲区的总数。arg2、arg3 和 arg4 分别包含了增加、删除和循环回收的 WAL 文件的数量。
clog-checkpoint-start	(bool)	当一个检查点的 CLOG 部分开始时触发的探针。arg0 为真表示正常检查点，为假表示关闭检查点。

名称	参数	描述
clog-checkpoint-done	(bool)	当一个检查点的 CLOG 部分完成时触发的探针。arg0 的含义与clog-checkpoint-start中相同。
subtrans-checkpoint-start	(bool)	当一个检查点的 SUBTRANS 部分开始时触发的探针。arg0 为真表示正常检查点，为假表示关闭检查点。
subtrans-checkpoint-done	(bool)	当一个检查点的 SUBTRANS 部分完成时触发的探针。arg0 的含义与subtrans-checkpoint-start中相同。
multixact-checkpoint-start	(bool)	当一个检查点的 MultiXact 部分开始时触发的探针。arg0 为真表示正常检查点，为假表示关闭检查点。
multixact-checkpoint-done	(bool)	当一个检查点的 MultiXact 部分完成时触发的探针。arg0 的含义与multixact-checkpoint-start中相同。
buffer-checkpoint-start	(int)	当一个检查点的写缓冲区部分开始时触发的探针。arg0 传递位标志来区分不同的检查点类型，例如关闭（shutdown）、立即（immediate）或强制（force）。
buffer-sync-start	(int, int)	当我们在检查点期间开始写脏缓冲区时（在标识哪些缓冲区必须被写之后）触发的探针。arg0 是缓冲区总数，arg1 是当前为脏并且需要被写的缓冲区数量。
buffer-sync-written	(int)	在检查点期间当每个缓冲区被写完之后触发的探针。arg0 是缓冲区的 ID。
buffer-sync-done	(int, int, int)	当所有脏缓冲区被写之后触发的探针。arg0 是缓冲区总数。arg1 是检查点进程实际写的缓冲区数量。arg2 是期望写的数目（buffer-sync-start的 arg1）；arg1 和 arg2 的任何的不同反映在该检查点期间有其他进程刷写了缓冲区。
buffer-checkpoint-sync-start	()	在脏缓冲区被写入到内核之后并且在开始发出 fsync 请求之前触发的探针。
buffer-checkpoint-done	()	当同步缓冲区到磁盘完成时触发的探针。
twophase-checkpoint-start	()	当一个检查点的两阶段部分开始时触发的探针。
twophase-checkpoint-done	()	当一个检查点的两阶段部分完成时触发的探针。
buffer-read-start	(ForkNumber, BlockNumber, Oid, Oid, Oid, int, bool)	当一次缓冲区读开始时触发的探针。arg0 和 arg1 包含该页的分支号和块号（如果这是一次关系扩展请求，arg1 为 -1）。arg2、arg3 和 arg4 包含表空间、数据库和关系 OID 用以识别该关系。对一个本地缓冲区，

名称	参数	描述
		arg5 是创建临时关系的后端的 ID；对于一个共享缓冲区，arg5 是 InvalidBackendId（-1）。arg6 为真表示一次关系扩展请求，为假表示正常读。
buffer-read-done	(ForkNumber, BlockNumber, Oid, Oid, Oid, int, bool, bool)	当一次缓冲区读完成时触发的探针。arg0 和 arg1 包含该页的分支号和块号（如果这是一次关系扩展请求，arg1 现在包含新增加块的块号）。arg2、arg3 和 arg4 包含表空间、数据库和关系 OID 用以识别该关系。对一个本地缓冲区，arg5 是创建临时关系的后端的 ID；对于一个共享缓冲区，arg5 是 InvalidBackendId（-1）。arg6 为真表示一次关系扩展请求，为假表示正常读。arg7 为真表示在池中找到该缓冲区，为假表示没有找到。
buffer-flush-start	(ForkNumber, BlockNumber, Oid, Oid, Oid)	在发出对一个共享缓冲区的任意写请求之前触发的探针。arg0 和 arg1 包含该页的分支号和块号。arg2、arg3 和 arg4 包含表空间、数据库和关系 OID 用以识别该关系。
buffer-flush-done	(ForkNumber, BlockNumber, Oid, Oid, Oid)	当一个写请求完成时触发的探针（注意这只反映传递数据给内核的时间，它通常并没有实际地被写入到磁盘）。参数和buffer-flush-start的相同。
buffer-write-dirty-start	(ForkNumber, BlockNumber, Oid, Oid, Oid)	当一个服务器进程开始写一个脏缓冲区时触发的探针（如果这经常发生，表示shared_buffers太小，或需要调整后台写入器的控制参数）。arg0 和 arg1 包含该页的分支号和块号。arg2、arg3 和 arg4 包含表空间、数据库和关系 OID 用以识别该关系。
buffer-write-dirty-done	(ForkNumber, BlockNumber, Oid, Oid, Oid)	当一次脏缓冲区写完成时触发的探针。参数与buffer-write-dirty-start相同。
wal-buffer-write-dirty-start	()	当一个服务器进程因为没有可用 WAL 缓冲区空间开始写一个脏 WAL 缓冲区时触发的探针（如果这经常发生，表示wal_buffers太小）。
wal-buffer-write-dirty-done	()	当一次脏 WAL 缓冲区写入完成时触发的探针。
xlog-insert	(unsigned char, unsigned char)	当一个 WAL 记录被插入时触发的探针。arg0 是该记录的资源管理器（rmid）。arg1 包含 info 标志。
xlog-switch	()	当请求一次 WAL 段切换时触发的探针。
smgr-md-read-start	(ForkNumber, BlockNumber, Oid, Oid, Oid, int)	当开始从一个关系读取一块时触发的探针。arg0 和 arg1 包含该页的分支号和块号。arg2、arg3 和 arg4 包含表空间、数据库和关系 OID 用以识别该关系。对一

名称	参数	描述
		个本地缓冲区，arg5 是创建临时关系的后端的 ID；对于一个共享缓冲区，arg5 是InvalidBackendId（-1）。
smgr-md-read-done	(ForkNumber, BlockNumber, Oid, Oid, Oid, int, int, int)	当一次块读取完成时触发的探针。arg0 和 arg1 包含该页的分支号和块号。arg2、arg3 和 arg4 包含表空间、数据库和关系 OID 用以识别该关系。对一个本地缓冲区，arg5 是创建临时关系的后端的 ID；对于一个共享缓冲区，arg5 是InvalidBackendId（-1）。arg6 是实际读取的字节数，而 arg7 是请求读取的字节数（如果两者不同就意味着麻烦）。
smgr-md-write-start	(ForkNumber, BlockNumber, Oid, Oid, Oid, int)	当开始向一个关系中写入一个块时触发的探针。arg0 和 arg1 包含该页的分支号和块号。arg2、arg3 和 arg4 包含表空间、数据库和关系 OID 用以识别该关系。对一个本地缓冲区，arg5 是创建临时关系的后端的 ID；对于一个共享缓冲区，arg5 是InvalidBackendId（-1）。
smgr-md-write-done	(ForkNumber, BlockNumber, Oid, Oid, Oid, int, int, int)	当一个块写操作完成时触发的探针。arg0 和 arg1 包含该页的分支号和块号。arg2、arg3和 arg4 包含表空间、数据库和关系 OID来标识该关系。对于一个本地缓冲区，arg5 是创建临时关系的后端 ID；对于一个共享缓冲区，arg5 是InvalidBackendId（-1）。arg6 是实际写的字节数，而 arg7 是要求写的字节数（如果这两者不同，则意味着麻烦）。
sort-start	(int, bool, int, int, bool)	当一次排序操作开始时触发的探针。arg0 指示是堆排序、索引排序或 datum 排序。arg1 为真表示强制唯一值。arg2 是键列数。arg3 是允许使用的工作内存大小（以千字节计）。如果要求随机访问排序结果，则 arg4 为真。
sort-done	(bool, long)	当一次排序完成时触发的探针。arg0 为真表示外部排序，为假表示内排序。arg1 是用于一次外部排序的磁盘块的数目，或用于一次内排序的以千字节计的内存。
lwlock-acquire	(char *, int, LWLockMode)	当成功获得一个 LWLock 时触发的探针。arg0 是该 LWLock 所在的切片（Tranche）。arg1 是该 LWLock 在其切片内的偏移量。arg2 是所请求的锁模式，是排他或共享。
lwlock-release	(char *, int)	当一个 LWLock 被释放时（但是注意还没有唤醒任何一个被释放的等待者）触发的探针。arg0 是该 LWLock 所在的切片（Tranche）。

名称	参数	描述
lwlock-wait-start	(char *, int, LWLockMode)	当一个 LWLock不是当即可用并且一个服务器进程因此开始等待该锁变为可用时触发的探针。arg0 是该 LWLock 所在的切片（Tranche）。arg1 是该 LWLock 在其切片内的偏移量。arg2 是所请求的锁模式，是排他或共享。
lwlock-wait-done	(char *, int, LWLockMode)	当一个进程从对一个 LWLock 的等待中被释放（它实际还没有得到该锁）时触发的探针。arg0 是该 LWLock 所在的切片（Tranche）。arg1 是该 LWLock 在其切片内的偏移量。arg2 是所请求的锁模式，是排他或共享。
lwlock-condacquire	(char *, int, LWLockMode)	当调用者指定无需等待而成功获得一个 LWLock 时触发的探针。arg0 是该 LWLock 所在的切片（Tranche）。arg1 是该 LWLock 在其切片内的偏移量。arg2 是所请求的锁模式，是排他或共享。
lwlock-condacquire-fail	(char *, int, LWLockMode)	当调用者指定无需等待而没有成功获得一个 LWLock 时触发的探针。arg0 是该 LWLock 所在的切片（Tranche）。arg1 是该 LWLock 在其切片内的偏移量。arg2 是所请求的锁模式，是排他或共享。
lock-wait-start	(unsigned int, unsigned int, unsigned int, unsigned int, unsigned int, LOCKMODE)	当一个重量级锁（lmgr锁）的请求由于锁不可用开始等待时触发的探针。arg0 到 arg3 是标识被锁定对象的标签字段。arg4 指示被锁对象的类型。arg5 表示被请求的锁类型。
lock-wait-done	(unsigned int, unsigned int, unsigned int, unsigned int, unsigned int, LOCKMODE)	当一个重量级锁（lmgr 锁）的请求结束等待时（即已经得到锁）触发的探针。参数与lock-wait-start一样。
deadlock-found	()	当死锁检测器发现死锁时触发的探针。

表 27.16. 探针参数中使用的已定义类型

类型	定义
LWLockId	int
LocalTransactionId	unsigned int
LWLockMode	int
LOCKMODE	int
BlockNumber	unsigned int
Oid	unsigned int
ForkNumber	int
bool	char

27.4.3. 使用探针

下面的示例展示了一个分析系统中事务计数的 DTrace 脚本，可以用来代替一次性能测试之前和之后的pg_stat_database快照：

```
#!/usr/sbin/dtrace -qs

postgresql$1:::transaction-start
{
      @start["Start"] = count();
      self->ts  = timestamp;
}

postgresql$1:::transaction-abort
{
      @abort["Abort"] = count();
}

postgresql$1:::transaction-commit
/self->ts/
{
      @commit["Commit"] = count();
      @time["Total time (ns)"] = sum(timestamp - self->ts);
      self->ts=0;
}
```

当被执行时，该示例 D 脚本给出这样的输出：

```
# ./txn_count.d `pgrep -n postgres` or ./txn_count.d <PID>
^C

Start                                          71
Commit                                         70
Total time (ns)                        2312105013
```

> **注意**
>
> SystemTap 的追踪脚本记法与 DTrace 不同，但底层探针是兼容的。需要注意的一点是，截至本文编写时，SystemTap 脚本必须使用双下划线来代替连字符引用探针名。预计未来的 SystemTap 版本会修复这一点。

你应该记住，DTrace 脚本需要细心地编写和调试，否则被收集的追踪信息可能会毫无意义。在大多数发现问题的情况下，出错的是插桩，而不是底层系统。
当讨论使用动态追踪发现的信息时，一定要附上使用的脚本以便其也被检查和讨论。

更多示例脚本可以在 PgFoundry 的dtrace 项目[3]中找到。

27.4.4. 定义新探针

开发者可以在代码中任意位置定义新的探针，当然这要重新编译之后才能生效。
下面是插入新探针的步骤：

1. 决定探针名称以及要通过探针提供的数据
2. 把该探针定义加入到src/backend/utils/probes.d

[3] http://pgfoundry.org/projects/dtrace/

3. 如果pg_trace.h尚未被包含该探针点的模块引用，则将它包含进来，
   并且在源代码中期望的位置插入TRACE_POSTGRESQL探针宏

4. 重新编译并验证新探针是可用的

示例：. 这里是一个如何增加一个探针来用事务 ID 追踪所有新事务的示例。

1. 决定探针将被命名为transaction-start
   并且需要一个LocalTransactionId类型的参数

2. 将该探针定义加入到src/backend/utils/probes.d：

   ```
   probe transaction__start(LocalTransactionId);
   ```

   注意探针名字中双下划线的使用。在一个使用探针的 DTrace 脚本中，双下划线需要被替换为一个连字符，因此，文档中应向用户说明的名称是transaction-start。

3. 在编译时，transaction__start被转换成一个名为TRACE_POSTGRESQL_TRANSACTION_START的宏（注意这里是单下划线），可以通过包含头文件pg_trace.h获得。将宏调用加入到源代码中的合适位置。在这种情况下，看起来类似：

   ```
   TRACE_POSTGRESQL_TRANSACTION_START(vxid.localTransactionId);
   ```

4. 在重新编译和运行新的二进制文件之后，通过运行下面的 DTrace 命令来检查新增的探针是否可用。你应该看到类似下面的输出：

   ```
   # dtrace -ln transaction-start
      ID    PROVIDER          MODULE           FUNCTION NAME
   18705 postgresql49878     postgres     StartTransactionCommand
    transaction-start
   18755 postgresql49877     postgres     StartTransactionCommand
    transaction-start
   18805 postgresql49876     postgres     StartTransactionCommand
    transaction-start
   18855 postgresql49875     postgres     StartTransactionCommand
    transaction-start
   18986 postgresql49873     postgres     StartTransactionCommand
    transaction-start
   ```

向C代码中添加追踪宏时，有一些事情需要注意：

- 需要小心的是，为探针参数指定的数据类型要匹配宏中使用的变量的数据类型，
  否则会发生编译错误。

- 在大多数平台上，如果用--enable-dtrace编译了PostgreSQL，
  无论何时当控制经过一个追踪宏时，都会对该宏的参数求值，
  即使没有进行追踪也会这样做。如果只是报告少数局部变量的值，通常无需担心这一点。
  但是要注意不要将开销大的函数调用放入参数中。如果你需要这样做，
  考虑通过检查追踪是否真的被启用来保护该宏：

  ```
  if (TRACE_POSTGRESQL_TRANSACTION_START_ENABLED())
      TRACE_POSTGRESQL_TRANSACTION_START(some_function(...));
  ```

  每个追踪宏有一个对应的ENABLED宏。

第 28 章 监控磁盘使用情况

本章讨论如何监控PostgreSQL数据库系统的磁盘使用情况。

28.1. 确定磁盘使用情况

每个表都有一个主堆磁盘文件，其中存储了大部分数据。如果该表有任何值可能较宽的列，还可能会有一个与该表关联的TOAST文件，
用于存储宽到无法方便地放入主表中的值（见第 56.2 节）。如果存在，还会在TOAST表上有一个有效索引。也可能会有与基表关联的索引。每个表和索引都存储在单独的磁盘文件中 – 如果文件会超过 1 GB，则可能分成多个文件。这些文件的命名约定见第 56.1 节。

可以通过三种方式监控磁盘空间：使用表 9.64中列出的 SQL 函数，使用oid2name模块，或者手工检查系统目录。SQL 函数最容易使用，通常也是推荐的方法。本节其余部分展示如何通过检查系统目录来完成此事。

在最近进行过清理或分析的数据库上使用psql时，
可以发出查询以查看任意表的磁盘使用情况：

```
SELECT pg_relation_filepath(oid), relpages FROM pg_class WHERE
 relname = 'customer';

 pg_relation_filepath | relpages
----------------------+----------
 base/16384/16806     |       60
(1 row)
```

每个页通常为 8 KB。（请记住，`relpages`只会由`VACUUM`、`ANALYZE`以及少数 DDL 命令（如`CREATE INDEX`）更新。）如果你想直接检查该表的磁盘文件，文件路径名就很有用。

要显示TOAST表使用的空间，可使用如下查询：

```
SELECT relname, relpages
FROM pg_class,
     (SELECT reltoastrelid
      FROM pg_class
      WHERE relname = 'customer') AS ss
WHERE oid = ss.reltoastrelid OR
      oid = (SELECT reltoastidxid
             FROM pg_class
             WHERE oid = ss.reltoastrelid)
ORDER BY relname;

       relname        | relpages
----------------------+----------
 pg_toast_16806       |        0
 pg_toast_16806_index |        1
```

也可以轻松显示索引大小：

```
SELECT c2.relname, c2.relpages
FROM pg_class c, pg_class c2, pg_index i
WHERE c.relname = 'customer' AND
      c.oid = i.indrelid AND
      c2.oid = i.indexrelid
ORDER BY c2.relname;
```

```
       relname        | relpages
----------------------+----------
 customer_id_indexdex |       26
```

利用这些信息，你可以很容易地找出最大的表和索引：

```
SELECT relname, relpages
FROM pg_class
ORDER BY relpages DESC;
```

```
       relname        | relpages
----------------------+----------
 bigtable             |     3290
 customer             |     3144
```

28.2. 磁盘已满故障

数据库管理员最重要的磁盘监控任务，是确保磁盘不会被写满。
数据磁盘写满不会导致数据损坏，但可能会阻止有用的活动继续进行。如果保存 WAL 文件的磁盘被写满，则数据库服务器可能会 panic 并随后关闭。

如果无法通过删除其他内容来释放磁盘上的额外空间，
可以利用表空间将部分数据库文件移到其他文件系统上。有关更多信息，请参见第 21.6 节。

> **提示**
>
> 有些文件系统在接近写满时性能会明显变差，
> 因此不要等到磁盘完全写满才采取措施。

如果你的系统支持按用户设置磁盘配额，
那么数据库自然也会受到运行服务器的那个用户所拥有配额的限制。
超出配额会带来与磁盘空间完全耗尽相同的不良影响。

第 29 章 可靠性与预写式日志

本章说明如何使用预写式日志，实现高效、可靠的运行。

29.1. 可靠性

可靠性是任何严肃数据库系统的重要属性，而 PostgreSQL 会尽一切可能保证运行可靠。可靠运行的一个方面是，已提交事务所记录的所有数据都应存储在不受断电、操作系统故障和硬件故障影响的非易失性区域中（当然不包括该非易失性区域本身的故障）。通常，只要能成功将数据写入计算机的永久存储设备（磁盘驱动器或等效设备），就满足这一要求。实际上，即使一台计算机遭受致命损坏，只要磁盘驱动器得以幸存，就可以将它们移到另一台硬件相近的计算机上，所有已提交事务仍将完好无损。

虽然周期性地强制将数据写到磁盘盘片似乎是个简单操作，但事实并非如此。由于磁盘驱动器远慢于主存和 CPU，在计算机主存与磁盘盘片之间存在多层缓存。首先是操作系统的缓冲区缓存，它会缓存常用磁盘块并合并磁盘写入。幸运的是，所有操作系统都为应用提供了强制将数据从缓冲区缓存写入磁盘的方法，而 PostgreSQL 会使用这些机制。（关于如何调整这一过程，见参数wal_sync_method。）

其次，磁盘驱动器控制器中可能还有缓存；这在 RAID 控制卡上尤为常见。其中一些缓存是直写式，也就是说写请求一到达就立刻发送到驱动器。另一些是回写式，也就是说数据会在稍后的某个时间发送到驱动器。这类缓存可能构成可靠性隐患，因为磁盘控制器缓存中的内存是易失性的，断电时会丢失内容。较好的控制卡配有后备电池单元（BBU），也就是卡上带有电池，可在系统断电时维持缓存供电。电力恢复后，数据便会写入磁盘驱动器。

最后，大多数磁盘驱动器自身也带有缓存。有些是直写式，有些是回写式；对于回写式驱动器缓存，同样存在与磁盘控制器缓存相同的数据丢失隐患。消费级 IDE 和 SATA 驱动器尤其可能带有在断电后无法保留内容的回写式缓存。很多固态驱动器（SSD）也带有易失性的回写式缓存。

这些缓存通常都可以禁用；不过，具体方法会因操作系统和驱动器类型而异：

- 在 Linux 上，可以使用 `hdparm -I` 查询 IDE 驱动器；如果 `Write cache` 旁边有一个 `*`，就表示启用了写缓存。可以使用 `hdparm -W 0` 关闭写缓存。SCSI 驱动器可以使用sdparm[1]查询。使用 `sdparm --get=WCE` 检查是否启用了写缓存，使用 `sdparm --clear=WCE` 将其禁用。
- 在 FreeBSD 上，可以使用 `atacontrol` 查询 IDE 驱动器，并通过在 `/boot/loader.conf` 中设置 `hw.ata.wc=0` 来关闭写缓存；SCSI 驱动器也可以使用 `camcontrol identify` 查询，并在可用时借助 `sdparm` 查询和修改写缓存。
- 在 Solaris 上，磁盘写缓存由 `format -e`[2] 控制。（Solaris 的 ZFS 文件系统在启用磁盘写缓存时仍然是安全的，因为它会发出自己的磁盘缓存刷新命令。）
- 在 Windows 上，如果 `wal_sync_method` 是 `open_datasync`（默认值），可以通过取消勾选 `My Computer\Open\disk drive\Properties\Hardware\Properties\Policies\Enable write caching on the disk` 来禁用写缓存。另一种办法是将 `wal_sync_method` 设为 `fsync` 或 `fsync_writethrough`，这两者都会阻止写缓存。
- 在 Mac OS X 上，可以通过将 `wal_sync_method` 设为 `fsync_writethrough` 来防止写缓存。

较新的 SATA 驱动器（遵循 ATAPI-6 或更高版本标准的那些）提供了驱动器缓存刷新命令（`FLUSH CACHE EXT`），而 SCSI 驱动器则早已支持类似的 `SYNCHRONIZE CACHE` 命令。这

[1] http://sg.danny.cz/sg/sdparm.html
[2] http://www.sun.com/bigadmin/content/submitted/format_utility.jsp

些命令对 PostgreSQL 并不直接可见，但某些文件系统（如 ZFS、ext4）可以利用它们将启用了回写缓存的驱动器上的数据刷新到盘片。遗憾的是，这类文件系统与带后备电池单元（BBU）的磁盘控制器配合时表现并不理想。在这种配置中，同步命令会强制把控制器缓存中的全部数据写到磁盘，削弱 BBU 的大部分好处。可以运行 pg_test_fsync模块查看自己是否受此影响。如果确实受影响，而又能这样做，则可通过关闭文件系统中的写屏障或重新配置磁盘控制器，恢复 BBU 的性能优势。如果关闭了写屏障，一定要确保电池保持正常工作；电池故障有可能导致数据丢失。希望文件系统和磁盘控制器的设计者最终能解决这种次优行为。

当操作系统向存储硬件发出写请求后，它几乎做不了什么来确保数据已经到达真正的非易失性存储区域。相反，管理员有责任确保所有存储组件都能保证数据完整性。应避免使用写缓存没有电池后备的磁盘控制器。在驱动器层面，如果驱动器无法保证在关闭前把数据写出，就应禁用回写缓存。如果使用 SSD，要注意其中很多默认并不遵守缓存刷新命令。可以借助 `diskchecker.pl`[3] 测试 I/O 子系统行为是否可靠。

数据丢失的另一种风险来自磁盘盘片写操作本身。磁盘盘片被分为若干扇区，通常每个 512 字节。每一次物理读写都处理整个扇区。当写请求到达驱动器时，其数据量可能是 512 字节的某个倍数（PostgreSQL 通常一次写入 8192 字节，也就是 16 个扇区），而写入过程可能在任何时刻因断电失败，这就意味着某些 512 字节扇区写成功了，另一些却没有。为防范这类故障，PostgreSQL 会在修改磁盘上的实际页面之前，定期将整页镜像写入持久 WAL 存储。这样一来，在崩溃恢复时 PostgreSQL 就能从 WAL 恢复部分写入的页面。如果你使用了能阻止部分页面写入的文件系统软件（如 ZFS），可以通过关闭参数full_page_writes来禁用这种页面镜像。带后备电池单元（BBU）的磁盘控制器并不能阻止部分页面写入，除非它们保证写入 BBU 的数据总是完整的整页（8kB）。

29.2. 预写式日志（WAL）

预写式日志（WAL）是确保数据完整性的标准方法。大多数（即使不是全部）事务处理书籍中都能找到详细描述。简而言之，WAL 的核心思想是，对数据文件（即表和索引所在之处）的更改，必须在这些更改被记入日志之后才能写出；也就是说，必须在描述这些更改的 WAL 记录被刷入持久存储之后，再去写数据文件。如果遵循这个过程，就无需在每次事务提交时都把数据页刷盘，因为我们知道，一旦发生崩溃，就可以利用日志恢复数据库：任何尚未应用到数据页上的更改，都可以根据 WAL 记录重做。（这就是前滚恢复，也称 REDO。）

> **提示**
>
> 由于 WAL 会在崩溃后恢复数据库文件内容，因此要可靠地存储数据文件或 WAL 文件，并不需要日志化文件系统。实际上，日志开销反而可能降低性能，特别是在日志会导致文件系统数据刷盘时。幸运的是，通常可以通过文件系统挂载选项关闭日志期间的数据刷写，例如 Linux ext3 文件系统上的 `data=writeback`。不过，日志化文件系统确实能够提高崩溃后的启动速度。

使用 WAL 会显著减少磁盘写入次数，因为要保证事务已提交，通常只需把 WAL 文件刷入磁盘，而不必把该事务修改过的每个数据文件都刷盘。WAL 文件按顺序写入，因此同步 WAL 的代价远低于刷写数据页。对于处理大量小事务、且这些事务触及数据存储不同部分的服务器，这一点尤其明显。此外，当服务器正在处理大量并发的小事务时，对 WAL 文件执行一次 `fsync` 就可能足以提交多个事务。

WAL 还使在线备份和时间点恢复成为可能，如第 24.3 节所述。通过归档 WAL 数据，我们就能支持回退到可用 WAL 数据覆盖范围内的任意时刻：只需先安装数据库的一个较早物理备

[3] http://brad.livejournal.com/2116715.html

份，然后把 WAL 重放到目标时刻即可。更进一步说，这个物理备份不必是数据库状态的瞬时快照 – 即使备份是在一段时间内完成的，重放这段时间内的 WAL 也能修复任何内部不一致。

29.3. 异步提交

异步提交是一项可选功能，它允许事务更快完成，但代价是如果数据库发生崩溃，
最近的事务可能会丢失。在许多应用中，这是可以接受的权衡。

如前一节所述，事务提交通常是同步的：服务器要等到该事务的 WAL 记录被刷入持久存储后，才向客户端返回成功指示。因此，
客户端可以确信那些已被报告为提交成功的事务一定会被保留，
即使服务器紧接着立刻崩溃也是如此。然而，对于短事务而言，
这种等待是总事务时间的主要组成部分。
选择异步提交模式意味着服务器会在事务从逻辑上完成后就立刻返回成功，
而不必等到它生成的 WAL 记录真正到达磁盘。这会显著提升小事务的吞吐量。

异步提交引入了数据丢失风险。在向客户端报告事务完成与事务真正被提交（也就是说，
即使服务器崩溃也保证不会丢失）之间，存在一个很短的时间窗口。因此，
如果客户端会基于“系统一定会记住这个事务”这一假设采取外部动作，
就不应使用异步提交。例如，银行显然不会对记录 ATM 吐钞的事务使用异步提交。但在很多场景中，例如事件日志记录，并不需要这种强保证。

使用异步提交所承担的风险是数据丢失，而不是数据损坏。如果数据库发生崩溃，
它会通过重放 WAL 直到最后一条已刷写记录来恢复。因此，数据库会被恢复到自洽状态，但任何尚未刷到磁盘的事务都不会反映在该状态中。其最终效果就是丢失最后几个事务。
由于事务按提交顺序重放，所以不会引入任何不一致性 – 例如，如果事务 B 做出的更改依赖于更早的事务 A 的效果，就不可能出现 A 的效果丢失而 B 的效果却被保留的情况。

用户可以为每个事务选择提交模式，因此同步提交事务和异步提交事务可以并发运行。
这使得在性能与事务持久性的确定性之间能够灵活权衡。提交模式由用户可设置的参数synchronous_commit控制，而该参数可以用任何设置配置参数的方式来更改。
任一事务实际使用的模式取决于事务提交开始时 `synchronous_commit` 的值。

某些实用命令，例如 `DROP TABLE`，无论 `synchronous_commit` 如何设置，都会强制同步提交。这是为了保证服务器文件系统与数据库逻辑状态之间的一致性。
支持两阶段提交的命令，例如 `PREPARE TRANSACTION`，也始终是同步的。

如果数据库在异步提交与写出该事务 WAL 记录之间的风险窗口内崩溃，那么该事务所做的更改将丢失。该风险窗口的时长是受限的，因为一个后台进程（“WAL 写入器”）会将尚未写出的 WAL 记录每隔wal_writer_delay毫秒刷到磁盘。风险窗口的实际最大时长是 `wal_writer_delay` 的三倍，因为 WAL 写入器在系统繁忙时被设计为倾向于一次写出整页。

> **小心**
>
> 立即模式关闭等同于服务器崩溃，因此也会造成任何未刷写的异步提交丢失。

异步提交的行为不同于将fsync设为 off。`fsync` 是影响所有事务的全服务器设置。它会禁用 PostgreSQL 内部所有试图同步数据库不同部分写入的逻辑，因此系统崩溃（即硬件或操作系统崩溃，而不是 PostgreSQL 本身故障）可能导致数据库状态遭受任意严重的损坏。在许多场景中，异步提交能带来关闭 `fsync` 所能获得的大部分性能提升，却不承担数据损坏的风险。

commit_delay听起来也很像异步提交，但它实际上是一种同步提交方法（事实上，
在异步提交期间 `commit_delay` 会被忽略）。`commit_delay` 会在事务把 WAL 刷到磁盘之前引入一段延迟，希望某个事务执行的一次刷盘也能服务于其他差不多同时提交的事务。可以把这个设置看成是扩大一个时间窗口，
让事务得以加入即将参与同一次刷盘的组，从而在多个事务之间摊销这次刷盘的代价。
只有在存在许多并发事务时，设置 `commit_delay` 才可能有帮助。

29.4. WAL 配置

有若干与 WAL 相关的配置参数会影响数据库性能。本节解释它们的用法。有关设置服务器配置参数的一般信息，请参见第 18 章。

检查点是事务序列中的一些点，
在这些点上可以保证堆和索引数据文件已经用检查点之前写入的全部信息进行了更新。
执行检查点时，所有脏数据页都会刷盘，并向 WAL 文件写入一条特殊的检查点记录。（更改记录此前已写入 WAL 文件并刷盘。）如果发生崩溃，崩溃恢复过程会查看最新的检查点记录，以确定应该从 WAL 的哪个位置（称为重做记录）开始执行 REDO。该点之前对数据文件所做的任何更改都保证已经在磁盘上。因此，检查点之后，位于包含重做记录的段之前的 WAL 段不再需要，可以被回收或删除。（若正在执行 WAL 归档，则这些 WAL 段必须先归档，随后才能回收或删除。）

检查点要求将所有脏数据页刷盘，这可能造成可观的 I/O 负载。因此，检查点活动会被节流，使 I/O 从检查点开始时就启动，并在下一个检查点预计开始前完成；这样可将检查点期间的性能下降降到最低。

服务器的检查点进程会定期自动执行检查点。每写入checkpoint_segments个日志段，
或者每经过checkpoint_timeout秒，就会开始一个检查点，以先到者为准。默认设置分别是 3 个段和 300 秒（5 分钟）。如果自上一个检查点以来没有写入任何 WAL，那么即使 checkpoint_timeout 已经过去，也会跳过新的检查点。如果正在使用 WAL 归档，而你希望对文件归档频率设定下限，从而限制潜在数据丢失，那么应当调整 archive_timeout 参数，而不是检查点参数。也可以使用 SQL 命令 `CHECKPOINT` 强制执行检查点。

减小 `checkpoint_segments` 和/或 `checkpoint_timeout` 会让检查点更频繁地发生。这样可以加快崩溃后恢复，因为需要重做的工作更少。不过，
这必须与更频繁地将脏数据页刷盘所增加的成本权衡。如果设置了full_page_
writes（默认就是如此），还要考虑另一个因素。为了保证数据页一致性，
每个检查点之后对某个数据页的首次修改，都会导致把整页内容写入日志。在这种情况下，
更短的检查点间隔会增加输出到 WAL 的数据量，部分抵消缩短间隔的目的，并且无论如何都会带来更多磁盘 I/O。

检查点的代价相当高，首先是因为它要求写出当前所有脏缓冲区，
其次是因为它还会像上文所述那样带来额外的后续 WAL 流量。因此，明智的做法是将检查点参数设得足够高，避免检查点发生得过于频繁。作为对检查点参数的一种简单合理性检查，
你可以设置参数checkpoint_warning。如果检查点之间的间隔小于 `checkpoint_warning` 秒，服务器日志就会输出一条消息，建议增大 `checkpoint_segments`。偶尔出现这样的消息无需惊慌，但如果它频繁出现，就应当增大检查点控制参数。大型 `COPY` 传输之类的批量操作，如果 `checkpoint_segments` 设得不够高，也可能导致出现多次此类警告。

为了避免大量页面写出在短时间内淹没 I/O 系统，检查点期间对脏缓冲区的写出会被分散到一段时间内。这段时间由checkpoint_completion_target控制，
它表示为检查点间隔的一个分数。系统会调节 I/O 速率，使检查点在自检查点开始以来已消耗给定比例的 `checkpoint_segments` 个 WAL 段、或者已过去给定比例的 `checkpoint_timeout` 秒时完成，以较早者为准。使用默认值 0.5 时，可以预期 PostgreSQL 会在下一次检查点开始前的这段时间约过一半时完成每个检查点。在正常运行时已非常接近最大 I/O 吞吐量的系统上，你可能希望增大 `checkpoint_completion_target`，以降低检查点带来的 I/O 负载。这样做的缺点是，延长检查点会影响恢复时间，因为需要保留更多 WAL 段以备恢复时使用。虽然 `checkpoint_completion_target` 可以设到 1.0，但最好保持在该值以下（或许最多为 0.9），因为检查点除了写出脏缓冲区之外还包含其他活动。设置为 1.0 很可能导致检查点不能按时完成，从而因为所需 WAL 段数量的意外波动而损失性能。

WAL 段文件至少会有一个，通常不会多于 (2 + `checkpoint_completion_target`) * `checkpoint_segments` + 1 或 `checkpoint_segments` + wal_keep_segments + 1 个文件。每个段文件通常是 16 MB（不过在构建服务器时可以更改这个大小）。可以据此估算 WAL 的空间需求。通常，当旧的日志段文件不再需要时，它们会被回收（也就是重命名为编号序列中的未来段文件）。如果由于日志输出速率的短期突增，段文件数量超过了 3 *

checkpoint_segments + 1，那么不再需要的段文件就会被删除而不是回收，直到系统重新回到该限制之下。

在归档恢复或备库模式下，服务器会周期性地执行重启点，其行为类似于正常运行时的检查点：服务器会强制将其所有状态刷盘，更新 pg_control 文件，以表明已经处理过的 WAL 数据无需再次扫描，然后回收 pg_xlog 目录中的旧 WAL 段文件。重启点的执行频率不会高于主库上的检查点，因为重启点只能在检查点记录处执行。重启点总是会等待至少 checkpoint_segments 个日志段或 checkpoint_timeout 秒（以较长者为准）。

有两个常用的内部 WAL 函数：LogInsert 和 XLogFlush。LogInsert 用于把新记录放入共享内存中的 WAL 缓冲区。如果新记录没有可用空间，LogInsert 就不得不把若干已满的 WAL 缓冲区写出（移入内核缓存）。这并不理想，因为 LogInsert 会在每一次数据库底层修改（例如插入行）时被调用，而在调用它时，受影响数据页上正持有排他锁，因此这一操作必须尽可能快。更糟的是，写 WAL 缓冲区还可能迫使系统创建新的 WAL 段，这会花费更多时间。正常情况下，WAL 缓冲区应由 XLogFlush 请求负责写出并刷盘；该请求大多发生在事务提交时，以确保事务记录已刷盘至持久存储。在 WAL 输出量很高的系统上，XLogFlush 请求可能不够频繁，无法避免 LogInsert 自己去执行写出。在这种系统上，应增加 WAL 缓冲区的数量，具体做法是修改参数wal_buffers。当设置了full_page_writes且系统非常繁忙时，把该值设置得更高有助于平滑每次检查点后紧接着那段时期的响应时间。

commit_delay参数定义了服务器进程在用 LogInsert 把一条提交记录写入日志之后、执行 XLogFlush 之前会睡眠多少微秒。这一延迟允许其他服务器进程把它们的提交记录追加到日志中，从而让所有这些记录通过一次日志同步一起刷盘。如果未启用fsync，或者当前处于活动事务中的其他会话少于commit_siblings个，则不会发生睡眠；这避免了在不太可能有其他会话很快提交时进行睡眠。注意，在大多数平台上，睡眠请求的分辨率是十毫秒，因此 1 到 10000 微秒之间的任何非零 commit_delay 设置效果都相同。这些参数的理想取值目前尚不明确；鼓励进行实验。

参数wal_sync_method决定 PostgreSQL 将如何请求内核把 WAL 更新强制刷盘。就可靠性而言，所有选项应当都是相同的；例外是 fsync_writethrough，它有时能够在其他选项做不到时强制将磁盘缓存中的数据刷盘。不过，具体哪个选项速度最快则高度依赖平台。可以使用pg_test_fsync模块测试不同选项的速度。请注意，如果 fsync 已被关闭，那么这个参数就没有意义。

启用配置参数wal_debug（前提是 PostgreSQL 编译时支持它）会导致每一次 LogInsert 和 XLogFlush 的 WAL 调用都被记录到服务器日志中。将来，这个选项可能会被更通用的机制所取代。

29.5. WAL 内部机制

WAL 会自动启用；管理员除了确保满足 WAL 文件的磁盘空间需求，并完成任何必要调优之外，不需要再做其他操作（见第 29.4 节）。

WAL 文件保存在数据目录下的 pg_xlog 目录中，由一组段文件组成，每个段通常为 16 MB（不过可在构建服务器时通过修改 --with-wal-segsize configure 选项来改变大小）。每个段又分为若干页面，通常每页 8 kB（该大小可通过 configure 选项 --with-wal-blocksize 更改）。WAL 记录头定义在 access/xlogrecord.h 中；记录内容则取决于被记录的事件类型。段文件以不断递增的数字命名，从 000000010000000000000000 开始。这些编号不会回绕，但要把可用的编号耗尽还需要非常、非常长的时间。

如果能将 WAL 放在与主数据库文件不同的磁盘上，会更有利。这可以通过把 pg_xlog 目录移动到其他位置（当然要在服务器关闭时进行），然后在主数据目录中的原位置创建一个指向新位置的符号链接来实现。

WAL 的目标是确保数据库记录在被修改之前，日志已经先被写出；但如果磁盘驱动器实际上只是缓存了数据、尚未将其存储到磁盘上，却向内核谎报写入成功，这一目标就会被破坏。

在这种情况下，断电可能导致不可恢复的数据损坏。管理员应尽量确保保存 PostgreSQL WAL 文件的磁盘不会做出这种虚假报告。（见第 29.1 节。）

在完成检查点并刷写 WAL 之后，该检查点的位置会保存在文件 `pg_control` 中。因此，在恢复开始时，服务器会先读取 `pg_control`，再读取检查点记录；然后从检查点记录给出的 WAL 位置向前扫描并执行 REDO。由于检查点之后对数据页的第一次修改，会把整页内容保存在 WAL 中（假设没有禁用full_page_writes），因而自该检查点以来被修改的所有页面都会恢复到一致状态。

为处理 `pg_control` 损坏的情况，我们本应支持按相反顺序扫描现有 WAL 段 – 也就是从新到旧 –，以便找到最新检查点。这一功能尚未实现。`pg_control` 足够小（不足一个磁盘页），因此不会受部分写问题影响；到目前为止，也没有仅仅因为无法读取 `pg_control` 本身而导致数据库故障的报告。所以，尽管理论上它是个薄弱环节，`pg_control` 在实践中似乎不是问题。

第 30 章 回归测试

回归测试是针对PostgreSQL中 SQL 实现的一套全面测试。它们既测试标准 SQL 操作，也测试 PostgreSQL的扩展能力。

30.1. 运行测试

回归测试既可以针对已经安装并正在运行的服务器执行，
也可以在构建树中使用临时安装来执行。此外，运行测试时还有“parallel”和“sequential”两种模式。顺序方式一次只运行一个测试脚本，而并行方式会启动多个服务器进程，
以并行方式运行一组测试。并行测试能进一步确认进程间通信和锁定机制是否正常工作。

30.1.1. 针对临时安装运行测试

要在构建完成后、安装之前运行并行回归测试，请输入：

```
gmake check
```

运行位置为顶层目录。（也可以切换到`src/test/regress`并在那里运行此命令。）
结束时，应该会看到类似下面的结果：

```
=======================
 All 115 tests passed.
=======================
```

否则会显示哪些测试失败的说明。请参阅第 30.2 节中的下文说明，不要过早认定一次“失败”就意味着存在严重问题。

由于这种测试方法会运行一个临时服务器，如果你以 root 用户身份进行构建，它将无法工作，因为服务器不会以 root 身份启动。推荐做法是不要以 root 身份构建；否则应在安装完成后再执行测试。

如果你把PostgreSQL配置为安装到某个已经存在旧版 PostgreSQL安装的位置，并且在安装新版本之前执行 `gmake check`，可能会发现测试失败，因为新程序会尝试使用已安装的共享库。（典型症状是报出未定义符号。）如果你希望在覆盖旧安装之前运行测试，就需要使用 `configure --disable-rpath` 进行构建。不过，不建议在最终安装时使用这个选项。

并行回归测试会在你的用户 ID 下启动相当多的进程。目前最大并发度为二十个并行测试脚本，也就是四十个进程：每个测试脚本都有一个服务器进程和一个 psql进程。因此，如果你的系统对每个用户可创建的进程数施加限制，请确保该限制至少有五十个左右，
否则并行测试中可能会出现看似随机的失败。如果无法提高该限制，可以通过设置`MAX_CONNECTIONS` 参数来降低并行度。例如：

```
gmake MAX_CONNECTIONS=10 check
```

这样一次最多只会并发运行十个测试。

30.1.2. 针对现有安装运行测试

要在安装之后运行测试（见第 15 章），请初始化一个数据区域并启动服务器，具体说明见第 17 章，然后输入：

```
gmake installcheck
```

或者，要并行测试：

```
gmake installcheck-parallel
```

测试默认通过本地主机和默认端口号连接服务器，除非通过PGHOST和PGPORT
环境变量另行指定。测试将在名为regression的数据库中运行；
任何现有的同名数据库都会被删除。这些测试还会临时创建名为 regressuser*N* 的用户身份。

30.1.3. 附加测试套件

gmake check 和 gmake installcheck 命令只运行“核心”回归测试，这些测试用于检验 PostgreSQL 服务器的内置功能。源码发行版还包含其他测试套件，其中大部分与附加功能有关，例如可选的过程语言。

要运行适用于已选定构建模块的全部测试套件（包括核心测试），
请在构建树顶层输入下列命令之一：

```
gmake check-world
gmake installcheck-world
```

这些命令分别使用临时服务器或已安装的服务器运行测试，与前面对gmake check和gmake installcheck的说明相同。其他注意事项也与前面对各个方法的说明相同。注意，gmake check-world会为每个被测试的模块构建一个独立的临时安装树，因此需要多得多的时间和磁盘空间，相比之下耗用较少的是gmake installcheck-world。

或者，也可以在构建树相应的子目录中执行gmake check 或gmake installcheck来运行单个测试套件。请记住，gmake installcheck假定你已经安装了相关模块，而不仅仅是核心服务器。

可以按这种方式调用的附加测试包括：

- 可选过程语言的回归测试（PL/pgSQL 除外，它由核心测试负责）。这些测试位于 src/pl 下。
- contrib模块的回归测试，位于 contrib下。并非所有contrib 模块都有测试。
- ECPG 接口库的回归测试，位于 src/interfaces/ecpg/test。
- 针对并发会话行为的压力测试，位于src/test/isolation。

使用 installcheck 模式时，这些测试会销毁任何名为 pl_regression、contrib_regression、isolationtest、regress1 或 connectdb 的现有数据库，以及名为 regression 的现有数据库。

30.1.4. 区域设置和编码

默认情况下，使用临时安装的测试会采用当前环境中定义的区域设置，以及由 initdb决定的相应数据库编码。通过设置适当的环境变量来测试不同的区域设置可能很有用，例如：

```
gmake check LANG=C
gmake check LC_COLLATE=en_US.utf8 LC_CTYPE=fr_CA.utf8
```

出于实现原因，设置LC_ALL不能用于此目的；其他所有与区域设置相关的环境变量都可以。

在针对现有安装测试时，区域设置由现有数据库集簇决定，不能为测试单独设置。

也可以通过设置变量ENCODING来显式选择数据库编码，例如：

```
gmake check LANG=C ENCODING=EUC_JP
```

只有在区域设置为 C 时，这样设置数据库编码通常才有意义；否则编码会从区域设置自动选择，而指定与区域设置不匹配的编码会导致错误。

无论是针对临时安装还是现有安装测试，都可以设置数据库编码；不过在后一种情况下，它必须与该安装的区域设置兼容。

30.1.5. 额外测试

核心回归测试套件包含几个默认不运行的测试文件，因为它们可能依赖于平台，或者运行时间很长。可以通过设置以下变量来运行这些或其他额外的测试文件：`EXTRA_TESTS`。例如，要运行`numeric_big`测试：

```
gmake check EXTRA_TESTS=numeric_big
```

要运行排序规则测试：

```
gmake check EXTRA_TESTS=collate.linux.utf8 LANG=en_US.utf8
```

其中，`collate.linux.utf8`测试仅适用于 Linux/glibc 平台，且只有在使用 UTF-8 编码的数据库中运行时才会成功。

30.1.6. 测试热备

源代码分发包中还包含针对热备静态行为的回归测试。这些测试需要一个正在运行的主库和一个正在运行的备库，备库正在从主库接收新的 WAL 更改（使用基于文件的日志传送或流复制均可）。这些服务器不会自动创建，复制的设置也不在此处文档说明。请查阅文档中有关所需命令和相关问题的各个章节。

要运行热备测试，首先在主库上创建一个名为`regression` 的数据库：

```
psql -h primary -c "CREATE DATABASE regression"
```

接下来，在主库上的 regression 数据库中运行准备脚本 `src/test/regress/sql/hs_primary_setup.sql`，例如：

```
psql -h primary -f src/test/regress/sql/hs_primary_setup.sql
 regression
```

等待这些更改传播到备库。

现在将默认数据库连接指向要测试的备库（例如，设置 `PGHOST`和`PGPORT` 环境变量）。最后，在回归测试目录中运行 `gmake standbycheck`：

```
cd src/test/regress
gmake standbycheck
```

还可以使用脚本 `src/test/regress/sql/hs_primary_extremes.sql` 在主库上生成一些极端行为，以测试备库的行为。

30.2. 测试结果评估

某些安装正确且功能完备的PostgreSQL实例，可能会因为平台特有因素而在部分回归测试中“失败”，例如浮点表示或消息措辞不同。目前这些测试是通过把输出与参考系统生成的输出做简单的`diff`比较来评估的，因此结果会对细微的系统差异很敏感。报告某项测试“失败”时，请务必检查预期结果与实际结果之间的差异；你可能会发现这些差异并不重要。尽管如此，我们仍努力在所有受支持平台上维护准确的参考文件，因此原则上应期待所有测试都能通过。

回归测试的实际输出位于`src/test/regress/results` 目录中的文件里。测试脚本使用`diff`将每个输出文件与存放在`src/test/regress/expected`

目录中的参考输出进行比较。所有差异都会保存在 src/test/regress/regression.diffs中供你检查。（运行核心测试以外的测试套件时，
这些文件当然会出现在相应的子目录中，而不是src/test/regress。）

如果由于某种原因，某个特定平台在某个测试上产生了“失败”，
但对输出的检查令你相信结果是有效的，那么可以新增一个比较文件，
让后续测试运行不再报告该失败。详情见第 30.3 节。

30.2.1. 错误消息差异

某些回归测试包含故意构造的非法输入值。错误消息可能来自 PostgreSQL代码，也可能来自宿主平台的系统例程。在后一种情况下，消息会因平台而异，但应反映相近的信息。
这类消息差异会导致回归测试显示为“失败”，但可以通过检查确认其有效性。

30.2.2. 区域设置差异

如果你针对一个使用 C 以外排序规则顺序的区域设置初始化的服务器运行测试，就可能因为排序顺序不同而出现差异，并导致后续失败。
回归测试套件通过提供备用结果文件来处理这个问题；
这些文件已知可以覆盖大量区域设置。

使用临时安装方法时，要在不同区域设置下运行测试，可在 gmake命令行上传递合适的区域设置相关环境变量，例如：

```
gmake check LANG=de_DE.utf8
```

（回归测试驱动器会取消设置LC_ALL，因此不能用该变量选择区域设置。）
如果不使用区域设置，要么取消所有与区域设置相关的环境变量（或将它们设为C），
要么使用下面这个特殊调用：

```
gmake check NO_LOCALE=1
```

针对现有安装运行测试时，区域设置由现有安装决定。要改变它，
需要在初始化数据库集簇时向initdb传入合适选项，使用另一种区域设置。

一般来说，建议在计划用于生产环境的区域设置下运行回归测试，
因为这样可以覆盖实际将在生产中使用的区域设置和编码相关代码部分。
根据操作系统环境不同，可能会出现失败，
但至少你会知道在真实应用运行时应当预期哪些与区域设置相关的行为。

30.2.3. 日期和时间差异

大多数日期和时间结果依赖于时区环境。参考文件是在时区 PST8PDT（加利福尼亚州伯克利）下生成的，如果测试未在该时区设置下运行，就会出现表面上的失败。
回归测试驱动器会将环境变量 PGTZ设为PST8PDT，这通常能够确保获得正确结果。

30.2.4. 浮点差异

某些测试涉及根据表列计算 64 位浮点数（double precision）。
我们已经观察到涉及double precision列数学函数的结果存在差异。float8和 geometry测试尤其容易在不同平台之间，甚至在不同编译器优化设置下出现细微差异。
要判断这些通常出现在小数点右侧第 10 位附近的差异究竟是否重要，需要人工目测比较。

某些系统把负零显示为-0，而另一些只显示 0。

某些系统对pow()和exp() 发出错误的方式，与当前PostgreSQL代码所预期的机制不同。

30.2.5. 行顺序差异

你可能会看到这样的差异：同样的行在输出中的顺序与预期文件中的顺序不同。
在大多数情况下，严格说来这并不是缺陷。大多数回归测试脚本并没有细致到为每一个SELECT都使用ORDER BY，因此按 SQL 规范，它们的结果行顺序并没有良好定义。实际上，由于我们看到的是同一软件在同一数据上执行相同查询，
通常在所有平台上都会得到相同的结果顺序，所以缺少ORDER BY并不是问题。不过，有些查询确实会表现出跨平台的顺序差异。针对已安装服务器测试时，非 C 区域设置或非默认参数设置，例如自定义的work_mem值或规划器代价参数，也可能导致顺序差异。

因此，如果你看到顺序差异，一般无需担心，除非查询确实包含 ORDER BY而你的结果违反了它。不过，仍请报告该问题，这样我们可以为那个特定查询加上ORDER BY，以在后续版本中消除这种虚假的“失败”。

你可能会好奇，为什么我们不显式地为所有回归测试查询排序，
从而一劳永逸地解决这个问题。原因在于，那样反而会降低回归测试的价值，
因为测试会倾向于覆盖能产生有序结果的查询计划类型，
而排除那些不能产生有序结果的计划类型。

30.2.6. 栈深度不足

如果errors测试在执行 select infinite_recurse()命令时导致服务器崩溃，这意味着平台对进程栈大小的限制比max_stack_depth参数所表明的更小。
可通过在更高的栈大小限制下运行服务器来修复（对于max_stack_depth的默认值，建议4MB）。如果做不到，另一种办法是减小max_stack_depth 的值。

在支持getrlimit()的平台上，服务器应自动选择 max_stack_depth的安全值；因此，除非你手工覆盖了该设置，否则这类失败就是一个应报告的缺陷。

30.2.7. “random” 测试

random测试脚本本来就是要产生随机结果的。在极少数情况下，这会导致该回归测试失败。输入：

```
diff results/random.out expected/random.out
```

通常只应产生一两行差异。除非 random 测试反复失败，否则不必担心。

30.2.8. 配置参数

在针对现有安装运行测试时，某些非默认参数设置可能导致测试失败。例如，改变enable_seqscan或 enable_indexscan等参数，可能导致计划发生变化，进而影响使用EXPLAIN的测试结果。

30.3. 变体比较文件

由于某些测试天生会产生依赖环境的结果，我们提供了指定备用“预期”结果文件的方法。
每个回归测试都可以有多个比较文件，用来展示不同平台上的可能结果。
对于每个测试应使用哪个比较文件，有两种相互独立的判定机制。

第一种机制允许针对特定平台选择比较文件。有一个映射文件 src/test/regress/resultmap，用于定义每个平台应使用哪个比较文件。
要为某个特定平台消除虚假的测试“失败”，你需要先选择或创建一个变体结果文件，然后在resultmap 文件中增加一行。

映射文件中的每一行都采用如下形式：

```
testname:output:platformpattern=comparisonfilename
```

测试名就是相应回归测试模块的名称。输出值指明要检查哪个输出文件。
对标准回归测试而言，这里始终是out。该值对应于输出文件的扩展名。平台模式是 Unix 工具expr风格的模式（也就是在开头隐含^锚点的正则表达式）。它会与config.guess打印出的平台名进行匹配。比较文件名则是替代结果比较文件的基名。

例如：某些系统会把极小的浮点数值解释为零，而不是报告下溢错误。这会导致float8回归测试出现一些差异。因此，我们提供了一个变体比较文件float8-small-is-zero.out，其中包含这些系统上的预期结果。为了消除虚假的“失败”消息，在OpenBSD平台上，resultmap包含以下内容：

```
float8:out:i.86-.*-openbsd=float8-small-is-zero.out
```

如果一台机器上的config.guess输出匹配i.86-.*-openbsd，就会触发该规则。resultmap中的其他行会为其他适用的平台选择变体比较文件。

第二种用于选择变体比较文件的机制更加自动化：它只是在多个提供的比较文件中选取“最佳匹配”。回归测试驱动脚本既考虑某个测试的标准比较文件*testname*.out，也考虑名为*testname_digit*.out 的变体文件（其中*digit*可以是任一单个数字 0-9）。如果其中任何一个文件完全匹配，就认为测试通过；否则，会使用生成最短 diff 的那个文件来创建失败报告。（如果resultmap为该特定测试包含一项，那么基础*testname*就是 resultmap中给出的替代名称。）

例如，对于char测试，比较文件 char.out包含在C和 POSIX区域设置中应当出现的结果，而文件 char_1.out则包含在许多其他区域设置中出现的排序结果。

最佳匹配机制原本是为了解决依赖区域设置的结果问题，
但它也可以用于任何仅凭平台名无法轻易预测测试结果的场景。该机制的一个局限是，
测试驱动器无法判断哪一个变体对当前环境才真正“正确”；
它只会选择看起来最合适的那个。因此，
最安全的做法是只对那些你愿意在所有上下文中都视为同样有效的变体结果使用这种机制。

30.4. 测试覆盖率检查

可以用覆盖率测试插装编译 PostgreSQL 源代码，从而检查回归测试或任何其他与该代码一起运行的测试套件覆盖了代码的哪些部分。目前，这只在使用 GCC 编译时受支持，并且需要gcov和 lcov程序。

一个典型的工作流程如下：

```
./configure --enable-coverage ... OTHER OPTIONS ...
gmake
gmake check # or other test suite
gmake coverage-html
```

然后将 HTML 浏览器指向coverage/index.html。这些gmake命令在子目录中也可以使用。

要在各次测试运行之间重置执行计数，请运行：

```
gmake coverage-clean
```

部分 IV. 客户端接口

本部分描述随PostgreSQL一起发布的客户端编程接口。这些章节彼此独立，可以分别阅读。
还有许多面向客户端程序的外部编程接口是单独发布的，它们各自也附带文档（附录 H列出了一些较常用的接口）。本部分的读者应熟悉如何使用SQL操作和查询数据库（见第 II 部分），当然还应熟悉自己所选的编程语言。

目录

31. libpq - C 库 549
31.1. 数据库连接控制函数 549
31.1.1. 连接字符串 555
31.1.2. 参数关键词 556
31.2. 连接状态函数 559
31.3. 命令执行函数 563
31.3.1. 主要函数 563
31.3.2. 检索查询结果信息 569
31.3.3. 检索其他结果信息 572
31.3.4. 用于在 SQL 命令中嵌入字符串的转义 573
31.4. 异步命令处理 576
31.5. 逐行检索查询结果 579
31.6. 取消进行中的查询 580
31.7. 快速路径接口 581
31.8. 异步通知 582
31.9. COPY命令相关的函数 582
31.9.1. 用于发送COPY数据的函数 583
31.9.2. 用于接收COPY数据的函数 584
31.9.3. 用于COPY的废弃函数 584
31.10. 控制函数 586
31.11. 杂项函数 587
31.12. 通知处理 589
31.13. 事件系统 590
31.13.1. 事件类型 591
31.13.2. 事件回调函数 592
31.13.3. 事件支持函数 593
31.13.4. 事件实例 594
31.14. 环境变量 596
31.15. 密码文件 598
31.16. 连接服务文件 598
31.17. 连接参数的 LDAP 查找 598
31.18. SSL 支持 599
31.18.1. 服务器证书的客户端验证 599
31.18.2. 客户端证书 600
31.18.3. 不同模式中提供的保护 600
31.18.4. SSL 客户端文件使用 601
31.18.5. SSL 库初始化 602
31.19. 在线程化程序中的行为 602
31.20. 编译 libpq 程序 603
31.21. 示例程序 604
32. 大对象 614
32.1. 简介 614
32.2. 实现特性 614
32.3. 客户端接口 614
32.3.1. 创建一个大对象 614
32.3.2. 导入一个大对象 615
32.3.3. 导出一个大对象 615
32.3.4. 打开一个现有的大对象 615
32.3.5. 向大对象写入数据 616
32.3.6. 从大对象读取数据 616
32.3.7. 在大对象中定位 616
32.3.8. 获取大对象的当前位置 616
32.3.9. 截断一个大对象 616
32.3.10. 关闭一个大对象描述符 616
32.3.11. 移除一个大对象 617

32.4. 服务器端函数 617
32.5. 示例程序 617
33. ECPG - C 中的嵌入式 SQL 623
33.1. 概念 623
33.2. 管理数据库连接 623
33.2.1. 连接到数据库服务器 623
33.2.2. 选择一个连接 624
33.2.3. 关闭一个连接 625
33.3. 运行 SQL 命令 626
33.3.1. 执行 SQL 语句 626
33.3.2. 使用游标 626
33.3.3. 管理事务 627
33.3.4. 预备语句 627
33.4. 使用主变量 628
33.4.1. 概述 628
33.4.2. 声明节 628
33.4.3. 检索查询结果 629
33.4.4. 类型映射 629
33.4.5. 处理非原始 SQL 数据类型 636
33.4.6. 指示符 640
33.5. 动态 SQL 640
33.5.1. 执行没有结果集的语句 640
33.5.2. 执行一个有输入参数的语句 641
33.5.3. 执行一个有结果集的语句 641
33.6. pgtypes 库 642
33.6.1. numeric类型 642
33.6.2. 日期类型 645
33.6.3. 时间戳类型 648
33.6.4. interval 类型 652
33.6.5. decimal类型 652
33.6.6. pgtypeslib 的 errno 值 653
33.6.7. pgtypeslib 的特殊常量 654
33.7. 使用描述符区域 654
33.7.1. 命名 SQL 描述符区域 654
33.7.2. SQLDA 描述符区域 656
33.8. 错误处理 666
33.8.1. 设置回调 666
33.8.2. sqlca 667
33.8.3. SQLSTATE 与 SQLCODE 669
33.9. 预处理器指令 672
33.9.1. 包括文件 672
33.9.2. define 和 undef 指令 673
33.9.3. ifdef、ifndef、else、elif 和 endif 指令 673
33.10. 处理嵌入式 SQL 程序 674
33.11. 库函数 675
33.12. 大对象 675
33.13. C++ 应用 677
33.13.1. 主变量的可见范围 677
33.13.2. 使用外部 C 模块的 C++ 应用开发 678
33.14. 嵌入式 SQL 命令 680
33.15. Informix兼容模式 701
33.15.1. 附加类型 701
33.15.2. 附加的/缺少的嵌入式 SQL 语句 701
33.15.3. Informix-兼容的 SQLDA 描述符区域 702
33.15.4. 附加函数 705
33.15.5. 额外的常量 713
33.16. 内部 715
34. 信息模式 717

34.1. 模式 717
34.2. 数据类型 717
34.3. information_schema_catalog_name 718
34.4. administrable_role_authorizations 718
34.5. applicable_roles 718
34.6. attributes 719
34.7. character_sets 721
34.8. check_constraint_routine_usage 722
34.9. check_constraints 722
34.10. collations 722
34.11. collation_character_set_applicability 723
34.12. column_domain_usage 723
34.13. column_options 723
34.14. column_privileges 724
34.15. column_udt_usage 724
34.16. columns 725
34.17. constraint_column_usage 727
34.18. constraint_table_usage 728
34.19. data_type_privileges 728
34.20. domain_constraints 729
34.21. domain_udt_usage 729
34.22. domains 730
34.23. element_types 731
34.24. enabled_roles 733
34.25. foreign_data_wrapper_options 734
34.26. foreign_data_wrappers 734
34.27. foreign_server_options 734
34.28. foreign_servers 734
34.29. foreign_table_options 735
34.30. foreign_tables 735
34.31. key_column_usage 735
34.32. parameters 736
34.33. referential_constraints 738
34.34. role_column_grants 738
34.35. role_routine_grants 739
34.36. role_table_grants 739
34.37. role_udt_grants 740
34.38. role_usage_grants 740
34.39. routine_privileges 741
34.40. routines 741
34.41. schemata 745
34.42. sequences 745
34.43. sql_features 746
34.44. sql_implementation_info 746
34.45. sql_languages 747
34.46. sql_packages 747
34.47. sql_parts 748
34.48. sql_sizing 748
34.49. sql_sizing_profiles 748
34.50. table_constraints 749
34.51. table_privileges 749
34.52. tables 750
34.53. triggered_update_columns 750
34.54. triggers 751
34.55. udt_privileges 752
34.56. usage_privileges 752

34.57. user_defined_types 753
34.58. user_mapping_options 754
34.59. user_mappings 754
34.60. view_column_usage 755
34.61. view_routine_usage 755
34.62. view_table_usage 756
34.63. views 756

第 31 章 libpq - C 库

libpq是PostgreSQL的C应用程序编程接口。libpq是一组库函数，客户端程序可用它们向PostgreSQL后端服务器发送查询并接收查询结果。

libpq也是其他几个PostgreSQL应用程序接口的底层引擎，包括为 C++、Perl、Python、Tcl 和ECPG编写的接口。因此，对于这些包的用户，libpq行为的某些方面也很重要。特别是，第 31.14 节、第 31.15 节和第 31.18 节描述了任何使用libpq的应用程序的用户都能观察到的行为。

本章末尾（第 31.21 节）包含一些简短程序，展示如何编写使用libpq的程序。源代码发行包的`src/test/examples`目录中还提供了几个完整的libpq应用程序示例。

使用libpq的客户端程序必须包含头文件`libpq-fe.h`，并且必须与libpq库链接。

31.1. 数据库连接控制函数

以下函数用于建立到PostgreSQL后端服务器的连接。应用程序可以同时保持多个后端连接。（这样做的原因之一是访问多个数据库。）每个连接由一个`PGconn`对象表示，该对象可以通过以下函数获取：`PQconnectdb`，`PQconnectdbParams`，或`PQsetdbLogin`。注意，这些函数总是返回非空的对象指针，除非内存不足，甚至无法分配`PGconn`对象。应调用`PQstatus`函数检查返回值，确认连接成功后，再通过连接对象发送查询。

> **警告**
>
> 在 Unix 上，对持有已打开 libpq 连接的进程执行 fork 操作可能导致不可预料的结果，因为父进程和子进程会共享相同的套接字和操作系统资源。出于这个原因，我们不推荐这样的用法，尽管从子进程执行一个`exec`来载入新的可执行程序是安全的。

> **注意**
>
> 在 Windows 上，如果单个数据库连接被反复地启动和关闭，有一种办法可以改进性能。在内部，libpq 分别调用 `WSAStartup()` 和 `WSACleanup()` 来启动和关闭连接。`WSAStartup()` 会增加一个内部 Windows 库引用计数，而 `WSACleanup()` 会递减该计数。当引用计数恰好为 1 时，调用 `WSACleanup()` 会释放所有资源并卸载所有 DLL。这是一项代价很高的操作。为避免这种情况，应用程序可以手工调用 `WSAStartup()`，这样在最后一个数据库连接关闭时资源就不会被释放。

`PQconnectdbParams`

开启一个到数据库服务器的新连接。

```
PGconn *PQconnectdbParams(const char * const *keywords,
                          const char * const *values,
                          int expand_dbname);
```

这个函数使用从两个以`NULL`结尾的数组中取得的参数打开一个新的数据库连接。第一个数组`keywords`是一个字符串数组，其中每个元素都是一个关键词。第二个数组`values`给出每个关键词的值。和下面的`PQsetdbLogin`不同，参数集合可以在不改变函数签名的情况下扩展，因此对于新应用，

最好使用这个函数（或者相应的非阻塞函数PQconnectStartParams和PQconnectPoll）。

当前能被识别的参数关键词被列举在第 31.1.2 节中。

当 expand_dbname 非零时，允许把 *dbname* 关键词的值识别为连接串。只有第一次出现的 *dbname* 会按这种方式展开，后续任何 *dbname* 的值都作为普通数据库名处理。连接串可能格式的更多细节见第 31.1.1 节。

传入的数组可以为空以使用全部默认参数，也可以包含一个或多个参数设置。
两个数组的长度应当相同。处理会在 keywords 数组的最后一个非 NULL 元素处停止。

如果有参数未指定，则会检查相应的环境变量（见第 31.14 节）。如果环境变量也未设置，则使用所示的内置默认值。

通常，关键词按索引顺序从这些数组的开头开始处理。其效果是，当关键词重复出现时，保留最后处理的值。因此，通过仔细放置 *dbname* 关键词，可以决定哪些值可能被 *conninfo* 串覆盖，哪些不能。

PQconnectdb

开启一个到数据库服务器的新连接。

```
PGconn *PQconnectdb(const char *conninfo);
```

这个函数使用从字符串conninfo中得到的参数开启一个新的数据库连接。

被传递的字符串可以为空，这样将会使用所有的默认参数。
也可以包含由空白分隔的一个或多个参数设置，还可以包含一个URI。详见第 31.1.1 节。

PQsetdbLogin

开启一个到数据库服务器的新连接。

```
PGconn *PQsetdbLogin(const char *pghost,
                     const char *pgport,
                     const char *pgoptions,
                     const char *pgtty,
                     const char *dbName,
                     const char *login,
                     const char *pwd);
```

这是 PQconnectdb 的前身，使用固定的一组参数。除缺失参数始终采用默认值之外，功能相同。对于要使用默认值的任意固定参数，请传入 NULL 或空字符串。

如果 *dbName* 包含 = 符号，或具有有效的连接 URI 前缀，就会将其当作 *conninfo* 字符串处理，方式与将其传给 PQconnectdb 完全相同，然后按照 PQconnectdbParams 的规则应用其余参数。

PQsetdb

开启一个到数据库服务器的新连接。

```
PGconn *PQsetdb(char *pghost,
                char *pgport,
                char *pgoptions,
                char *pgtty,
                char *dbName);
```

这是一个调用PQsetdbLogin的宏，其中为*login*和*pwd*参数使用空指针。
提供它是为了向后兼容非常老的程序。

PQconnectStartParams
PQconnectStart
PQconnectPoll

以非阻塞的方式建立一个到数据库服务器的连接。

```
PGconn *PQconnectStartParams(const char * const *keywords,
                             const char * const *values,
                             int expand_dbname);

PGconn *PQconnectStart(const char *conninfo);

PostgresPollingStatusType PQconnectPoll(PGconn *conn);
```

这三个函数被用来开启一个到数据库服务器的连接，
这样你的应用的执行线程不会因为远程的I/O而被阻塞。这种方法的要点在于等待 I/O 完成可能在应用的主循环中发生，而不是在PQconnectdbParams 或 PQconnectdb中，并且因此应用能够把这种操作和其他动作并行处理。

在PQconnectStartParams中，数据库连接使用从keywords和values
数组中取得的参数创建，并且被expand_dbname控制，这和之前描述的PQconnectdb Params相同。

在PQconnectStart中，数据库连接使用从字符串conninfo中取得的参数创建，
这和之前描述的PQconnectdb相同。

无论是PQconnectStartParams还是PQconnectStart还是PQconnectPoll
都不会阻塞，只要满足以下限制：

- 必须恰当地使用 hostaddr 和 host 参数，以确保不会进行名称和反向名称查询。详细信息请参见第 31.1.2 节中这些参数的说明。
- 如果你调用PQtrace，确保接收追踪输出的流对象不会阻塞。
- 如后文所述，你要确保在调用PQconnectPoll之前，套接字处于合适的状态。

注意：PQconnectStartParams的用法与下文展示的PQconnectStart类似。

要开始无阻塞的连接请求，可调用conn = PQconnectStart("*connection_info_string*")。如果conn为空，则libpq无法分配一个新的PGconn结构体。否则，
一个有效的PGconn指针会被返回（不过还没有表示一个到数据库的有效连接）。
从PQconnectStart返回后，调用status = PQstatus(conn)。如果status等于CONNECTION_BAD，则PQconnectStart失败。

如果PQconnectStart成功，下一个阶段是轮询libpq，这样它能够继续进行连接序列。
使用PQsocket(conn)来获得该数据库连接底层的套接字描述符。
这样循环：如果PQconnectPoll(conn)上一次返回PGRES_POLLING_READING，
等到该套接字准备好读取（按照select()、poll()或类似的系统函数所指示的）。
则再次调用PQconnectPoll(conn)。反之，如果PQconnectPoll(conn)
上一次返回PGRES_POLLING_WRITING，等到该套接字准备好写入，则再次调用
PQconnectPoll(conn)。如果你还没有调用过PQconnectPoll，即刚刚调用过
PQconnectStart之后，行为就像是它上次返回了PGRES_POLLING_WRITING。
持续这个循环直到PQconnectPoll(conn)返回PGRES_POLLING_FAILED
指示连接过程已经失败，或者返回PGRES_POLLING_OK指示连接已经被成功地建立。

在连接过程中的任何时刻，都可以通过调用PQstatus来检查连接状态。
如果该调用返回CONNECTION_BAD，则连接过程已经失败；如果返回CONNECTION_OK，则连接已就绪。通过以下函数的返回值也同样可以检测这两种状态：PQconnect Poll，该函数已在上文介绍。在异步连接过程中（也仅在此过程中）

还可能出现其他状态。它们指明连接过程的当前阶段，例如可以用来向用户提供反馈。这些状态如下：

`CONNECTION_STARTED`

等待连接被建立。

`CONNECTION_MADE`

连接 OK，等待发送。

`CONNECTION_AWAITING_RESPONSE`

等待来自服务器的一个回应。

`CONNECTION_AUTH_OK`

收到认证，等待后端启动结束。

`CONNECTION_SSL_STARTUP`

协商 SSL 加密。

`CONNECTION_SETENV`

协商环境驱动的参数设置。

注意，虽然这些常量会保留下来以维持兼容性，
但应用程序绝不能依赖它们按某种特定顺序出现、必定出现，
或状态始终是这些已记录的值之一。应用程序可以采用类似以下的做法：

```
switch(PQstatus(conn))
{
        case CONNECTION_STARTED:
            feedback = "Connecting...";
            break;

        case CONNECTION_MADE:
            feedback = "Connected to server...";
            break;
.
.
.
        default:
            feedback = "Connecting...";
}
```

在使用`PQconnectPoll`时，连接参数`connect_timeout`
会被忽略：判断是否超时是应用的责任。除此之外，`PQconnectStart`
后面跟着`PQconnectPoll`循环等效于`PQconnectdb`。

注意如果`PQconnectStart`返回一个非空的指针，你必须在用完它之后调用`PQfinish`
来处理该结构体和任何相关的内存块。
即使连接尝试失败或被放弃时也必须完成这些工作。

`PQconndefaults`

返回默认连接选项。

```
PQconninfoOption *PQconndefaults(void);
```

```
typedef struct
{
    char   *keyword;   /* 该选项的关键词 */
    char   *envvar;    /* 后备环境变量名 */
    char   *compiled;  /* 编译时设置的后备默认值 */
    char   *val;       /* 选项的当前值，或者 NULL */
    char   *label;     /* 连接对话框中字段的标签 */
    char   *dispchar;  /* 指示如何在连接对话框中显示此字段。可取值：
                          ""          显示输入的值
                          "*"         密码字段 – 隐藏值
                          "D"         调试选项 – 默认不显示 */
    int     dispsize;  /* 对话框中的字段宽度，以字符计 */
} PQconninfoOption;
```

返回一个连接选项数组。这可以用来确定用于连接服务器的所有可能的PQconnectdb选项和它们的当前默认值。返回值指向一个PQconninfoOption结构体的数组，该数组以一个包含空keyword指针的条目结束。如果无法分配内存，则返回空指针。注意当前默认值（val字段）将依赖于环境变量和其他上下文。一个缺失或者无效的服务文件将会被无声地忽略掉。调用者必须把连接选项当作只读对待。

在处理完选项数组后，把它交给PQconninfoFree释放。如果没有这么做，每次调用PQconndefaults都会导致一小部分内存泄漏。

PQconninfoParse

返回从提供的连接字符串中解析到的连接选项。

```
PQconninfoOption *PQconninfoParse(const char *conninfo, char
 **errmsg);
```

解析一个连接字符串并且将结果选项作为一个数组返回，或者在连接字符串有问题时返回NULL。这个函数可以用来抽取所提供的连接字符串中的PQconnectdb选项。返回值指向一个PQconninfoOption结构体的数组，该数组以一个包含空keyword指针的条目结束。

所有合法选项将出现在结果数组中，但是任何在连接字符串中没有出现的选项的PQconninfoOption的val会被设置为NULL，默认值不会被插入。

如果 errmsg 不是 NULL，则成功时将 *errmsg 设为 NULL；失败时将其设为由 malloc 分配的、用于说明问题的错误字符串。（也可能出现 *errmsg 被设为 NULL，同时函数返回 NULL 的情况；这表示内存不足。）

在处理完选项数组后，把它交给PQconninfoFree释放。如果没有这么做，每次调用PQconninfoParse都会导致一小部分内存泄漏。反过来，如果发生一个错误并且errmsg不是NULL，确保使用PQfreemem释放错误字符串。

PQfinish

关闭与服务器的连接。同时释放PGconn对象使用的内存。

```
void PQfinish(PGconn *conn);
```

注意，即使与服务器的连接尝试失败（由PQstatus指示），应用也应当调用PQfinish来释放PGconn对象使用的内存。不能在调用PQfinish之后再使用PGconn指针。

PQreset

重置与服务器的通信通道。

```
void PQreset(PGconn *conn);
```

此函数将关闭与服务器的连接，并尝试重新建立到同一服务器的新连接，使用之前使用过的所有参数。这可能有助于在工作连接丢失后的错误恢复。

PQresetStart
PQresetPoll

以非阻塞方式重置与服务器的通信通道。

```
int PQresetStart(PGconn *conn);

PostgresPollingStatusType PQresetPoll(PGconn *conn);
```

这些函数会关闭与服务器的连接，并尝试重新建立到同一服务器的新连接，使用之前使用过的所有参数。如果原本可用的连接丢失，这可以用于错误恢复。它们与上文的 PQreset 不同之处在于采用非阻塞方式。它们受到与 PQconnectStartParams、PQconnectStart 和 PQconnectPoll 相同的限制。

要开始重置连接，请调用 PQresetStart。如果返回 0，表示重置失败。如果返回 1，则使用 PQresetPoll 轮询重置过程，方式与使用 PQconnectPoll 建立连接完全相同。

PQpingParams

PQpingParams报告服务器状态。它接受的连接参数与上文介绍的PQconnectdbParams相同。但取得服务器状态并不需要提供正确的用户名、密码或数据库名值。

```
PGPing PQpingParams(const char **keywords, const char **values,
 int expand_dbname);
```

该函数返回以下值之一：

PQPING_OK

The server is running and appears to be accepting connections.

PQPING_REJECT

服务器正在运行，但是处于一种不允许连接的状态（启动、关闭或崩溃恢复）。

PQPING_NO_RESPONSE

无法联系到服务器。这可能表示服务器没有运行，或者给定的连接参数中有些错误（例如，错误的端口号），或者有一个网络连接问题（例如，一个防火墙阻断了连接请求）。

PQPING_NO_ATTEMPT

没有尝试联系服务器，因为提供的参数显然不正确，或者有一些客户端问题（例如，内存用完）。

PQping

PQping报告服务器状态。它接受的连接参数与上文介绍的PQconnectdb相同。但取得服务器状态并不需要提供正确的用户名、密码或数据库名值。

```
PGPing PQping(const char *conninfo);
```

返回值与 PQpingParams 相同。

31.1.1. 连接字符串

几个libpq函数解析用户指定的字符串以获取连接参数。
这些字符串有两种被接受的格式：普通的`keyword = value`字符串和 RFC 3986[1] URI。

31.1.1.1. 关键词/值连接字符串

在第一种格式中，每一个参数设置的形式都是`keyword = value`。设置的等号周围的空白是可选的。要写一个空值或一个包含空白的值，将它用单引号包围，例如`keyword = 'a value'`。值中的单引号和反斜线必须用一个反斜线转义，即`\'`和`\\`。

示例：

```
host=localhost port=5432 dbname=mydb connect_timeout=10
```

能被识别的参数关键词在第 31.1.2 节中列出。

31.1.1.2. 连接 URI

一个连接URI的一般形式是：

```
postgresql://[user[:password]@][netloc][:port][/dbname][?param1=
value1&...]
```

URI模式标志符可以是`postgresql://`或`postgres://`。每个URI部分都是可选的。
下列示例展示了合法的URI语法：

```
postgresql://
postgresql://localhost
postgresql://localhost:5433
postgresql://localhost/mydb
postgresql://user@localhost
postgresql://user:secret@localhost
postgresql://other@localhost/otherdb?connect_timeout=
10&application_name=myapp
```

URI的层次部分的组件也可以作为参数给出。例如：

```
postgresql:///mydb?host=localhost&port=5433
```

任何不对应第 31.1.2 节中所列关键词的连接参数都会被忽略，并且会向`stderr`发送一条关于它们的警告消息。

为了改进与 JDBC 连接URI的兼容性，参数`ssl=true`的实例会被转换为`sslmode=require`。

可以在 URI 的任意部分使用百分号编码来包含具有特殊含义的符号。

主机部分可以是主机名或 IP 地址。要指定一个 IPv6 主机地址，将它封闭在方括号中：

```
postgresql://[2001:db8::1234]/database
```

主机组件会被按照参数host对应的描述来解释。特别地，如果主机部分为空或以斜线开头，将使用一个 Unix 域套接字连接，否则将启动一个 TCP/IP 连接。不过要注意，斜线是 URI 层次部分中的一个保留字符。因此，要指定一个非标准的 Unix 域套接字目录，要么省略 URI 中的主机指定并且把主机指定为一个参数，要么在 URI 的主机组件中用百分号编码路径：

[1] http://www.ietf.org/rfc/rfc3986.txt

```
postgresql:///dbname?host=/var/lib/postgresql
postgresql://%2Fvar%2Flib%2Fpostgresql/dbname
```

31.1.2. 参数关键词

当前识别的参数关键字如下：

`host`

要连接的主机名。如果主机名以斜杠开头，则指定的是 Unix 域通信，而非 TCP/IP 通信；此值是存放套接字文件的目录名。当未指定 `host` 或其值为空时，默认连接到 `/tmp`（或构建 PostgreSQL 时指定的套接字目录）中的 Unix 域套接字。在没有 Unix 域套接字的机器上，默认连接到 `localhost`。

`hostaddr`

要连接的主机的数字 IP 地址。应采用标准的 IPv4 地址格式，例如 `172.28.40.9`。如果机器支持 IPv6，也可以使用 IPv6 地址。只要此参数指定了非空字符串，就始终使用 TCP/IP 通信。

使用`hostaddr`代替`host`可以避免主机名查找，
这对于有时间限制的应用程序可能很重要。但是，采用 GSSAPI 或 SSPI 认证方式，以及进行`verify-full`SSL 证书验证时，都需要主机名。遵循以下规则：

- 如果指定了`host`而没有指定`hostaddr`，则会发生主机名查找。
- 如果指定了`hostaddr`而没有指定`host`，则`hostaddr`的值给出服务器的网络地址。如果认证方法需要主机名，则连接尝试将失败。
- 如果同时指定了 `host` 和 `hostaddr`，则 `hostaddr` 的值给出服务器的网络地址。只有认证方法需要主机名时，才会将 `host` 的值用作主机名；否则忽略该值。

注意，以下情况很可能导致认证失败：`host`不是位于网络地址`hostaddr`
的服务器名称。另外，注意在`~/.pgpass`中是用`host`而不是`hostaddr`
来标识连接的（参见第 31.15 节）。

如果既没有主机名也没有主机地址，libpq 会使用本地 Unix 域套接字连接；在没有 Unix 域套接字的机器上，则会尝试连接到 `localhost`。

`port`

连接到服务器主机的端口号，或者Unix域连接的套接字文件名扩展。

`dbname`

数据库名称。默认为与用户名相同。在某些情况下，该值会被检查是否为扩展格式；
有关更多详细信息，请参阅第 31.1.1 节。

`user`

建立连接所用的 PostgreSQL 用户名。默认与运行应用程序的操作系统用户名相同。

`password`

服务器要求密码认证时所使用的密码。

`connect_timeout`

连接时的最长等待时间，以秒为单位（写成十进制整数字符串）。
零或未指定表示无限等待。不建议使用小于2秒的超时时间。

client_encoding

这将为此连接设置client_encoding配置参数。除了对应服务器选项接受的值外，您还可以使用auto来从客户端的当前区域设置（Unix系统上的LC_CTYPE环境变量）确定正确的编码。

options

添加在运行时发送给服务器的命令行选项。例如，将其设置为-c geqo=off会把会话的geqo参数值设为off。有关可用选项的详细讨论，请参阅第 18 章。

application_name

指定application_name配置参数的值。

fallback_application_name

指定application_name配置参数的后备值。如果没有通过连接参数或PGAPPNAME环境变量为application_name指定值，则将使用此值。
在通用实用程序中指定后备名称很有用，该程序希望设置默认应用程序名称，但允许用户覆盖它。

keepalives

控制是否使用客户端 TCP keepalive。默认值为 1，表示开启；如果不需要 keepalive，可以将其设为 0，表示关闭。对于通过 Unix 域套接字建立的连接，此参数会被忽略。

keepalives_idle

控制在多久没有活动后，TCP 应向服务器发送 keepalive 消息，以秒为单位。值为零时使用系统默认值。对于通过 Unix 域套接字建立的连接，或禁用 keepalive 时，此参数会被忽略。此参数仅在支持 TCP_KEEPIDLE 或等效套接字选项的系统以及 Windows 上受支持；在其他系统上无效。

keepalives_interval

控制未被服务器确认收到的 TCP keepalive 消息在多少秒后应被重传。值为零时使用系统默认值。对于通过 Unix 域套接字建立的连接，或禁用 keepalive 时，此参数会被忽略。此参数仅在支持 TCP_KEEPINTVL 或等效套接字选项的系统以及 Windows 上受支持；在其他系统上无效。

keepalives_count

控制在客户端与服务器之间的连接被视为中断之前，可以丢失多少个 TCP keepalive 消息。值为零时使用系统默认值。对于通过 Unix 域套接字建立的连接，或禁用 keepalive 时，此参数会被忽略。此参数仅在支持 TCP_KEEPCNT 或等效套接字选项的系统上受支持；在其他系统上无效。

tty

忽略此参数（以前用于指定服务器调试输出的发送位置）。

sslmode

这个选项确定是否以及以何种优先级与服务器协商安全的SSL TCP/IP连接。有六种模式：

disable

仅尝试非SSL连接

allow

首先尝试非SSL连接；如果失败，则尝试SSL连接

`prefer` (默认)

首先尝试SSL连接；如果失败，则尝试非SSL连接

`require`

仅尝试SSL连接。如果存在根CA文件，则验证证书的方式与指定了`verify-ca`时相同

`verify-ca`

仅尝试SSL连接，并验证服务器证书是否由受信任的证书颁发机构（CA）颁发

`verify-full`

仅尝试SSL连接，验证服务器证书是否由受信任的CA颁发，并且请求的服务器主机名与证书中的匹配

详细了解这些选项如何工作，请参阅第 31.18 节。

在 Unix 域套接字通信中，`sslmode` 会被忽略。如果PostgreSQL编译时未启用 SSL 支持，使用选项`require`、`verify-ca`或 `verify-full`会导致错误，而选项`allow`和`prefer` 将被接受，但libpq实际上不会尝试建立SSL 连接。

`requiressl`

此选项已弃用，请改用 `sslmode` 设置。

如果设置为1，则需要与服务器建立SSL连接（这相当于`sslmode require`）。libpq将拒绝连接，如果服务器不接受 SSL连接。如果设置为0（默认值），libpq将与服务器协商连接类型（相当于`sslmode prefer`）。此选项仅在 PostgreSQL 编译时启用了 SSL 支持的情况下可用。

`sslcompression`

如果设为 1（默认值），则会压缩通过 SSL 连接发送的数据（这要求 OpenSSL 0.9.8 或更高版本）。如果设为 0，则禁用压缩（这要求 OpenSSL 1.0.0 或更高版本）。如果建立的是非 SSL 连接，或者所用 OpenSSL 版本不支持此功能，则忽略此参数。

压缩会消耗 CPU 时间，但在网络成为瓶颈时能够提高吞吐量。如果 CPU 性能是限制因素，禁用压缩能够改善响应时间和吞吐量。

`sslcert`

这个参数指定客户端SSL证书的文件名，替换默认的 `~/.postgresql/postgresql.crt`。如果没有建立SSL连接，则此参数将被忽略。

`sslkey`

这个参数指定了用于客户端证书的密钥的位置。它可以指定一个文件名，该文件名将被用来替代默认的 `~/.postgresql/postgresql.key`，或者它可以指定一个从外部“引擎”（引擎是OpenSSL可加载模块）获取的密钥。外部引擎的指定形式应包含一个由冒号分隔的引擎名称和一个引擎特定的密钥标识符。如果没有进行SSL连接，则此参数将被忽略。

`sslrootcert`

这个参数指定一个包含SSL证书颁发机构（CA）证书的文件名。如果文件存在，服务器的证书将被验证是否由这些机构之一签名。默认值是`~/.postgresql/root.crt`。

`sslcrl`

此参数指定 SSL 证书吊销列表（CRL）的文件名。如果该文件存在，在验证服务器证书时，会拒绝其中列出的证书。默认值为 `~/.postgresql/root.crl`。

`requirepeer`

这个参数指定了服务器的操作系统用户名，例如`requirepeer=postgres`。在建立Unix域套接字连接时，如果设置了这个参数，客户端会在连接开始时检查服务器进程是否在指定的用户下运行；如果不是，则连接会因错误而中止。这个参数可用于提供类似于在TCP/IP连接上使用SSL证书的服务器认证。（请注意，如果Unix域套接字位于`/tmp`或其他公共可写位置，任何用户都可以在那里启动一个服务器监听。使用这个参数来确保您连接到由受信任用户运行的服务器。）此选项仅在实现了`peer`认证方法的平台上受支持；请参见第 19.3.7 节。

`krbsrvname`

使用 Kerberos 5 或 GSSAPI 认证时所用的 Kerberos 服务名。这必须与服务器配置中指定的 Kerberos 认证服务名称匹配，才能成功进行认证。（另请参见第 19.3.5 节和第 19.3.3 节。）

`gsslib`

用于GSSAPI认证的GSS库。目前，除了包含GSSAPI和SSPI支持的Windows构建之外，这将被忽略。在这种情况下，将其设置为`gssapi`，以使libpq使用GSSAPI库进行认证，而不是默认的SSPI。

`service`

用于额外参数的服务名称。它指定了`pg_service.conf`中保存额外连接参数的服务名称。这允许应用程序只指定一个服务名称，以便可以集中维护连接参数。参见第 31.16 节。

31.2. 连接状态函数

这些函数可用于查询现有数据库连接对象的状态。

> **提示**
>
> 编写 libpq 应用程序时，应注意维护 `PGconn` 的抽象。请使用下述访问函数获取 `PGconn` 的内容。不建议通过 `libpq-int.h` 引用 `PGconn` 的内部字段，因为这些字段将来可能改变。

以下函数返回建立连接时确定的参数值。这些值在`PGconn`对象的生命周期内保持不变。

`PQdb`

返回该连接的数据库名。

```
char *PQdb(const PGconn *conn);
```

`PQuser`

返回该连接的用户名。

```
char *PQuser(const PGconn *conn);
```

PQpass

返回该连接的密码。

```
char *PQpass(const PGconn *conn);
```

PQhost

返回连接的服务器主机名。可能是主机名、IP 地址或者一个目录路径（如果通过 Unix 套接字连接）。

```
char *PQhost(const PGconn *conn);
```

PQport

返回连接的端口。

```
char *PQport(const PGconn *conn);
```

PQtty

返回该连接用于调试的TTY。（此功能已过时，因为服务器不再使用TTY设置，但为保持向后兼容仍保留此函数。）

```
char *PQtty(const PGconn *conn);
```

PQoptions

返回连接请求中传递的命令行选项。

```
char *PQoptions(const PGconn *conn);
```

以下函数返回的状态数据，可能随着对 PGconn 对象执行操作而改变。

PQstatus

返回该连接的状态。

```
ConnStatusType PQstatus(const PGconn *conn);
```

该状态可以是一系列值之一。不过，其中只有两个在一个异步连接过程之外可见：CONNECTION_OK和CONNECTION_BAD。
一个到数据库的完好连接的状态为CONNECTION_OK。
一个失败的连接尝试则由状态CONNECTION_BAD表示。通常，一个 OK 状态将一直保持到PQfinish，但是一次通信失败可能导致该状态过早地改变为CONNECTION_BAD。在那种情况下，该应用可以通过调用PQreset尝试恢复。

关于其他可能会被返回的状态代码，请见PQconnectStartParams、PQconnectStart和PQconnectPoll的条目。

PQtransactionStatus

返回服务器的当前事务内状态。

```
PGTransactionStatusType PQtransactionStatus(const PGconn *conn);
```

该状态可能是PQTRANS_IDLE（当前空闲）、PQTRANS_ACTIVE（一个命令运行中）、PQTRANS_INTRANS（空闲，处于一个有效的事务块中）或者PQTRANS_INERROR

（空闲，处于一个失败的事务块中）。如果该连接异常，将会报告PQTRANS_UNKNOWN。只有当一个查询已经被发送给服务器并且还没有完成时，才会报告PQTRANS_ACTIVE。

> **小心**
>
> 当使用 PostgreSQL 7.3 服务器且其 autocommit 参数被设为 off 时，PQtransactionStatus 会给出不正确的结果。服务器端的 autocommit 特性已被废弃，在之后的服务器版本中已不存在。

PQparameterStatus

查找服务器某个参数的当前设置。

```
const char *PQparameterStatus(const PGconn *conn, const char
 *paramName);
```

服务器会在连接启动时，以及某些参数值发生变化时，自动报告这些参数值。PQparameterStatus可用于查询这些设置。如果已知该参数，则返回其当前值；如果未知，则返回NULL。

当前版本报告的参数包括：server_version、server_encoding、client_encoding、application_name、is_superuser、session_authorization、DateStyle、IntervalStyle、TimeZone、integer_datetimes、standard_conforming_strings。（8.0 之前的版本不报告 server_encoding、TimeZone 和 integer_datetimes；8.1 之前的版本不报告 standard_conforming_strings；8.4 之前的版本不报告 IntervalStyle；9.0 之前的版本不报告 application_name。）注意，server_version、server_encoding 和 integer_datetimes 在启动后不能改变。

使用 3.0 之前协议的服务器不报告参数设置，但 libpq 仍包含获取 server_version 和 client_encoding 值的逻辑。建议应用程序使用 PQparameterStatus，而不是专门编写代码来确定这些值。（但要注意，在使用 3.0 之前协议的连接上，连接启动后通过 SET 改变 client_encoding，不会反映在 PQparameterStatus 的结果中。）对于 server_version，另请参见 PQserverVersion，它以数值形式返回此信息，更易于比较。

如果服务器未报告standard_conforming_strings的值，应用程序可以假定其为off，即反斜杠在字符串字面量中被视为转义字符。此外，服务器报告此参数也表明它接受转义字符串语法（E'...'）。

返回的指针虽然被声明为const，但实际上指向与PGconn结构体关联的可变存储。不能假定该指针在执行其他查询后仍然有效。

PQprotocolVersion

查询正在使用的前端/后端协议。

```
int PQprotocolVersion(const PGconn *conn);
```

应用程序可以使用此函数判断是否支持某些特性。目前，可能的值为 2（协议 2.0）、3（协议 3.0）或零（连接异常）。连接启动完成后，协议版本不会改变，但理论上可能在连接重置期间改变。通常，与 PostgreSQL 7.4 或更高版本的服务器通信时使用协议 3.0；7.4 之前的服务器仅支持协议 2.0。（协议 1.0 已过时，libpq 不支持它。）

PQserverVersion

返回一个表示后端版本的整数。

```
int PQserverVersion(const PGconn *conn);
```

应用可能会使用这个函数来判断它们连接到的数据库服务器的版本。
这个数字是这样形成的：将主版本、次版本和修订版本号分别转换成两位十进制数，
然后把它们拼接在一起。例如，版本8.1.5将被返回为80105，而版本8.
2将被返回为80200（不显示前导零）。如果连接无效则返回零。

PQerrorMessage

返回连接上的一个操作最近产生的错误消息。

```
char *PQerrorMessage(const PGconn *conn);
```

几乎所有 libpq 函数在失败时都会设置一条供 PQerrorMessage 返回的消息。注意，按照 libpq 的约定，非空的 PQerrorMessage 结果可能包含多行，并以换行符结尾。调用者不应直接释放该结果；当关联的 PGconn 句柄被传给 PQfinish 时，结果会被释放。不能假定在对 PGconn 结构体执行其他操作后，结果字符串仍保持不变。

PQsocket

获取与服务器相连的套接字的文件描述符编号。有效描述符大于或等于 0。结果为 -1 表示当前没有打开服务器连接（在普通操作期间这将不会改变，但是在连接设置或重置期间可能改变）。

```
int PQsocket(const PGconn *conn);
```

PQbackendPID

返回处理这个连接的后端进程的进程ID（PID）。

```
int PQbackendPID(const PGconn *conn);
```

后端 PID 可用于调试，也可与 NOTIFY 消息进行比较（消息包含发出通知的后端进程的 PID）。注意，该 PID 属于在数据库服务器主机上运行的进程，而非本地主机上的进程！

PQconnectionNeedsPassword

如果连接认证方法要求一个密码但没有可用的密码，返回真（1）。否则返回假（0）。

```
int PQconnectionNeedsPassword(const PGconn *conn);
```

这个函数可以在连接尝试失败后被应用于决定是否向用户提示要求一个密码。

PQconnectionUsedPassword

如果连接认证方法使用一个密码，返回真（1）。否则返回假（0）。

```
int PQconnectionUsedPassword(const PGconn *conn);
```

这个函数能在一次连接尝试失败或成功后用于检测该服务器是否要求一个密码。

PQgetssl

返回在连接中使用的SSL结构体，如果未使用SSL，则返回NULL。

```
SSL *PQgetssl(const PGconn *conn);
```

该结构体可以用来验证加密级别、检查服务器证书等。有关此结构体的信息，请参阅 OpenSSL文档。

必须定义 USE_SSL 才能得到这个函数正确的原型。这样做还会自动包含来自OpenSSL的ssl.h。

31.3. 命令执行函数

与数据库服务器的连接成功建立后，此处描述的函数用于执行 SQL 查询和命令。

31.3.1. 主要函数

PQexec

提交一个命令给服务器并且等待结果。

```
PGresult *PQexec(PGconn *conn, const char *command);
```

返回一个 PGresult 指针，也可能返回空指针。除内存不足或无法向服务器发送命令等严重错误外，通常都会返回非空指针。应调用 PQresultStatus 函数检查返回值是否表示错误（也应检查空指针，此时该函数返回 PGRES_FATAL_ERROR）。使用 PQerrorMessage 获取此类错误的更多信息。

命令字符串可以包含多个 SQL 命令（以分号分隔）。在一次PQexec调用中发送的多个查询会在同一个事务中处理，除非查询字符串中显式包含BEGIN/COMMIT命令将其划分为多个事务。不过要注意，返回的PGresult结构体只描述该字符串中最后执行的命令的结果。如果其中一条命令失败，就会在此处停止处理该字符串，返回的PGresult则描述该错误。

PQexecParams

提交一个命令给服务器并且等待结果，它可以在 SQL 命令文本之外独立地传递参数。

```
PGresult *PQexecParams(PGconn *conn,
                       const char *command,
                       int nParams,
                       const Oid *paramTypes,
                       const char * const *paramValues,
                       const int *paramLengths,
                       const int *paramFormats,
                       int resultFormat);
```

PQexecParams与PQexec相似，但是提供了额外的功能：参数值可以与命令字符串分开指定，并且可以以文本或二进制格式请求查询结果。PQexecParams 仅支持使用协议 3.0 及更高版本的连接；使用协议 2.0 时会失败。

该函数的参数是：

conn

要在其中发送命令的连接对象。

command

要执行的 SQL 命令字符串。如果使用了参数，它们在该命令字符串中被引用为$1、$2等。

nParams

提供的参数数量。它是数组*paramTypes[]*、*paramValues[]*、*paramLengths[]*和*paramFormats[]*的长度（当*nParams*为零时，数组指针可以是NULL）。

paramTypes[]

通过 OID 指定要赋予给参数符号的数据类型。如果*paramTypes*为NULL或者该数组中任何特定元素为零，
服务器会用对待未指定类型的字符串字面量的方式为参数符号推测一种数据类型。

paramValues[]

指定参数的实际值。数组中的空指针表示对应参数为 null；否则，指针指向以零结尾的文本字符串（文本格式），
或采用服务器所要求格式的二进制数据（二进制格式）。

paramLengths[]

指定二进制格式参数的实际数据长度。对于 null 参数和文本格式参数，该值会被忽略。如果没有二进制参数，数组指针可以为空指针。

paramFormats[]

指定参数采用文本格式（在对应数组元素中填入零）还是二进制格式（填入一）。
如果数组指针为空指针，则将所有参数视为文本字符串。

以二进制格式传递值时，需要了解后端所要求的内部表示形式。例如，
整数必须以网络字节序传递。传递 numeric 值时，需要了解服务器的存储格式，其实现见 src/backend/utils/adt/numeric.c::numeric_send() 和 src/backend/utils/adt/numeric.c::numeric_recv()。

resultFormat

指定零以获取文本格式的结果，指定一以获取二进制格式的结果。
（目前无法让不同结果列使用不同格式，尽管底层协议支持这样做。）

PQexecParams 相对于 PQexec 的主要优点是可以将参数值与命令字符串分开，从而避免繁琐且容易出错的加引号和转义工作。

和PQexec不同，PQexecParams至多允许在给定串中出现一个 SQL 命令（其中可以有分号，但是不能有超过一个非空命令）。这是底层协议的一个限制，但是有助于抵抗 SQL 注入攻击。

提示

通过 OID 指定参数类型较为繁琐，尤其是在不希望将具体 OID 值写死在程序中时。不过，即使服务器无法自行确定参数类型，或者推断出的类型与你所需的不同，
也可以避免直接指定 OID。在 SQL 命令文本中，为参数符号添加显式类型转换，即可指定要发送的数据类型。例如：

```
SELECT * FROM mytable WHERE x = $1::bigint;
```

这会强制将参数 $1 当作 bigint，而默认情况下会为它分配与 x 相同的类型。以二进制格式发送参数值时，强烈建议采用这种方式，或直接指定类型的数值 OID，来明确决定参数类型。因为二进制格式的冗余比文本格式更少，服务器发现类型不匹配错误的机会也更少。

PQprepare

提交一个请求用给定参数创建一个预备语句并且等待完成。

```
PGresult *PQprepare(PGconn *conn,
```

```
                        const char *stmtName,
                        const char *query,
                        int nParams,
                        const Oid *paramTypes);
```

PQprepare 创建一个预备语句，供随后使用 PQexecPrepared 执行。此功能允许将被反复使用的命令只解析和规划一次，而不是每次执行时都重复进行。PQprepare 仅在使用协议 3.0 及更高版本的连接中受支持，使用协议 2.0 时会失败。

该函数从*query*串创建一个名为*stmtName*的预备语句，该串必须包含一个单一 SQL 命令。*stmtName*可以是""来创建一个未命名语句，
在这种情况下任何已存在的未命名语句将被自动替换。否则，
如果语句名称已经在当前会话中被定义，则是一种错误。如果使用了任何参数，
它们在查询中以$1、$2等引用。*nParams* 是在数组 *paramTypes[]* 中预先指定了类型的参数数量（当*nParams*为零时，该数组指针可以是NULL）。*paramTypes[]*通过 OID 指定要赋予给参数符号的数据类型。如果*paramTypes*是NULL
或者该数组中任何特定元素为零，
服务器会用对待未指定类型的字符串字面量的方式为参数符号推测一种数据类型。还有，查询能够使用编号高于*nParams*的参数符号，
它们的数据类型也会被自动推测（找出推测出的数据类型的方法见PQdescribePrepared）。

正如PQexec一样，结果通常是一个PGresult对象，其内容代表服务器端成功或失败。返回空指针表示内存不足，或者根本无法发送命令。关于错误的更多信息请见PQerrorMessage。

用于PQexecPrepared的预备语句也可以通过执行 SQLPREPARE语句来创建。此外，虽然没有libpq函数可删除预备语句，但可以使用 SQLDEALLOCATE语句来完成。

PQexecPrepared

发送一个请求来执行一个带有给定参数的预备语句，并等待结果。

```
PGresult *PQexecPrepared(PGconn *conn,
                         const char *stmtName,
                         int nParams,
                         const char * const *paramValues,
                         const int *paramLengths,
                         const int *paramFormats,
                         int resultFormat);
```

PQexecPrepared类似于PQexecParams，
但它通过已创建的预备语句的名称指定要执行的命令，而非提供查询字符串。
此功能使重复使用的命令只需解析和规划一次，而不必在每次执行时都进行这些工作。
该语句必须事先在当前会话中创建为预备语句。PQexecPrepared 仅支持使用协议 3.0 及更高版本的连接；使用协议 2.0 时会失败。

参数与PQexecParams相同，只是给出了预备语句的名称而不是查询字符串，并且*paramTypes[]*参数不存在（因为在创建预备语句时已确定了参数类型）。

PQdescribePrepared

提交请求以获取有关指定预备语句的信息，并等待完成。

```
PGresult *PQdescribePrepared(PGconn *conn, const char *stmtName)
;
```

PQdescribePrepared允许应用程序获取关于先前创建的预备语句的信息。
PQdescribePrepared 仅支持使用协议 3.0 及更高版本的连接；使用协议 2.0 时会失败。

*stmtName*可以是""或NULL来引用未命名的语句，否则必须是现有预备语句的名称。成功时，返回一个状态为PGRES_COMMAND_OK的PGresult。函数PQnparams和PQparamtype可以应用于此 PGresult以获取有关预备语句参数的信息，函数PQnfields、PQfname、PQftype等提供有关语句的结果列（如果有）的信息。

PQdescribePortal

提交请求以获取有关指定 portal 的信息，并等待完成。

```
PGresult *PQdescribePortal(PGconn *conn, const char *portalName)
;
```

PQdescribePortal 允许应用程序获取先前创建的 portal 的信息。（libpq 不提供对 portal 的直接访问，但可以用此函数检查通过 DECLARE CURSOR SQL 命令创建的游标的属性。）PQdescribePortal 仅支持使用协议 3.0 及更高版本的连接；使用协议 2.0 时会失败。

*portalName*可以是""或NULL来引用未命名的 portal，否则必须是现有 portal 的名称。成功时，将返回一个带有状态PGRES_COMMAND_OK的PGresult。函数PQnfields、PQfname、PQftype等可应用于 PGresult，以获取有关 portal 的结果列（如果有）的信息。

PGresult 结构体封装服务器返回的结果。编写 libpq 应用程序时，应注意维护 PGresult 的抽象。请使用下述访问函数获取 PGresult 的内容。避免直接引用 PGresult 结构体的字段，因为这些字段将来可能改变。

PQresultStatus

返回该命令的结果状态。

```
ExecStatusType PQresultStatus(const PGresult *res);
```

PQresultStatus可以返回以下值之一：

PGRES_EMPTY_QUERY

发送给服务器的字符串为空。

PGRES_COMMAND_OK

一个不返回数据的命令成功完成。

PGRES_TUPLES_OK

一个返回数据的命令（例如SELECT或者SHOW）成功完成。

PGRES_COPY_OUT

从服务器复制出数据的传输开始。

PGRES_COPY_IN

复制数据到服务器的传输开始。

PGRES_BAD_RESPONSE

无法理解服务器的响应。

PGRES_NONFATAL_ERROR

发生了一次非致命错误（一个提示或警告）。

PGRES_FATAL_ERROR

发生了一次致命错误。

PGRES_COPY_BOTH

向服务器复制数据/从服务器复制数据的传输开始。这个特性当前只被用于流复制，因此这个状态应该不会在普通应用中出现。

PGRES_SINGLE_TUPLE

PGresult包含来自于当前命令的一个单一结果元组。这个状态只在查询选择了单行模式时发生（见第 31.5 节）。

如果结果状态为PGRES_TUPLES_OK或PGRES_SINGLE_TUPLE，可以使用下述函数取得查询返回的行。注意，即使SELECT命令恰好返回零行，状态仍为PGRES_TUPLES_OK。PGRES_COMMAND_OK用于不可能返回行的命令（例如INSERT、UPDATE等）。如果响应为PGRES_EMPTY_QUERY，可能表示客户端软件中存在缺陷。

一个状态为PGRES_NONFATAL_ERROR的结果将不会被PQexec或者其他查询执行函数直接返回，这类结果将被传递给通知处理器（见第 31.12 节）。

PQresStatus

将 PQresultStatus 返回的枚举值转换为描述该状态码的字符串常量。调用者不应释放此结果。

```
char *PQresStatus(ExecStatusType status);
```

PQresultErrorMessage

返回与命令关联的错误消息；如果没有错误，则返回空字符串。

```
char *PQresultErrorMessage(const PGresult *res);
```

如果发生了错误，返回的字符串会以换行符结尾。调用者不应直接释放结果；当关联的 PGresult 句柄被传给 PQclear 时，结果会被释放。

紧跟着一个PQexec 或 PQgetResult调用，PQerrorMessage（在连接上）将返回与PQresultErrorMessage相同的字符串（在结果上）。不过，一个PGresult将保持它的错误消息直到被销毁，而连接的错误消息将在后续操作被执行时被更改。当你想要知道与一个特定PGresult相关的状态，使用PQresultErrorMessage。而当你想要知道连接上最后一个操作的状态，使用PQerrorMessage。

PQresultErrorField

返回错误报告中的单个字段。

```
char *PQresultErrorField(const PGresult *res, int fieldcode);
```

fieldcode 是错误字段标识符，参见下文列出的符号。如果 PGresult 不是错误或警告结果，或者不包含指定字段，则返回 NULL。字段值通常不含末尾换行符。调用者不应直接释放结果；当关联的 PGresult 句柄被传给 PQclear 时，结果会被释放。

可以使用以下字段代码：

PG_DIAG_SEVERITY

严重性。字段的内容是ERROR、FATAL或PANIC（在一个错误消息中）。或者是WARNING、NOTICE、DEBUG、INFO或LOG（在一个提示消息中）。或者是其中之一的一个本地化翻译。总是存在。

PG_DIAG_SQLSTATE

用于错误的 SQLSTATE 代码。SQLSTATE 代码标识了已经发生的错误的类型，它可以被前端应用用来执行特定操作（例如错误处理）来响应一个特定数据库错误。
一个可能的 SQLSTATE 代码列表可见附录 A。这个字段无法被本地化，并且总是存在。

PG_DIAG_MESSAGE_PRIMARY

主要的人类可读的错误消息（通常是一行）。总是存在。

PG_DIAG_MESSAGE_DETAIL

细节：一个可选的次级错误消息，它携带了关于问题的更多细节。可能有多行。

PG_DIAG_MESSAGE_HINT

提示：一个关于如何处理该问题的可选建议。
它与细节的区别在于它提供了建议（可能不合适）而不是确切事实。可能有多行。

PG_DIAG_STATEMENT_POSITION

包含一个十进制整数的字符串，它表示一个错误游标位置，
该位置是原始语句字符串的索引。第一个字符的索引是 1，位置以字符计算而不是以字节计算。

PG_DIAG_INTERNAL_POSITION

这被定义为与PG_DIAG_STATEMENT_POSITION字段相同，
但是它被用在游标位置引用一个内部产生的命令而不是客户端提交的命令时。
当这个字段出现时，PG_DIAG_INTERNAL_QUERY字段将总是出现。

PG_DIAG_INTERNAL_QUERY

一个失败的内部产生的命令的文本。例如，这可能是由一个 PL/pgSQL 函数发出的 SQL 查询。

PG_DIAG_CONTEXT

指示错误发生的上下文。
当前这包括活动过程语言函数的调用栈追踪以及内部生成的查询。追踪是每行一项，最近的排在最前面。

PG_DIAG_SOURCE_FILE

报告错误的源代码所在的文件名。

PG_DIAG_SOURCE_LINE

报告错误的源代码行号。

PG_DIAG_SOURCE_FUNCTION

报告错误的源代码函数的名字。

客户端负责按自身需要格式化所显示的信息，尤其应在需要时将长行折行。
错误消息字段中的换行符应当视为段落分隔，而非行分隔。

libpq 内部产生的错误包含严重性和主要消息，但通常没有其他字段。使用 3.0 之前协议的服务器返回的错误包含严重性和主要消息，有时还包含详细消息，但没有其他字段。

注意，错误字段只对PGresult对象有效，对PGconn对象无效。没有PQerrorField函数。

PQclear

释放与 PGresult 关联的存储空间。每个命令结果在不再需要时都应通过 PQclear 释放。

```
void PQclear(PGresult *res);
```

你可以在需要时一直保留PGresult对象；它不会在你发出新命令时消失，
甚至在关闭连接后也不会消失。要销毁它，你必须调用PQclear。
否则应用程序会发生内存泄漏。

31.3.2. 检索查询结果信息

这些函数用于从表示成功查询结果的 PGresult 对象中提取信息，即状态为 PGRES_TUPLES_OK 或 PGRES_SINGLE_TUPLE 的对象。它们也可用于提取成功 Describe 操作的结果信息：Describe 结果包含的列信息与实际执行查询时相同，但行数为零。对于其他状态值的对象，这些函数会将结果视为零行、零列。

PQntuples

返回查询结果中的行（元组）数（注意，PGresult对象被限制为不超过INT_MAX行，因此一个int结果就足够了）。

```
int PQntuples(const PGresult *res);
```

PQnfields

返回查询结果中每一行的列（字段）数。

```
int PQnfields(const PGresult *res);
```

PQfname

返回给定列号对应的列名。列号从 0 开始。调用者不应直接释放结果；当关联的 PGresult 句柄被传给 PQclear 时，结果会被释放。

```
char *PQfname(const PGresult *res,
              int column_number);
```

如果列号超出范围，将返回NULL。

PQfnumber

返回与给定列名相关联的列号。

```
int PQfnumber(const PGresult *res,
              const char *column_name);
```

如果给定的名字不匹配任何列，将返回 -1。

给定名称按 SQL 命令中的标识符处理，即除非用双引号引用，否则会转换为小写。例如，对于以下 SQL 命令生成的查询结果：

```
SELECT 1 AS FOO, 2 AS "BAR";
```

会得到以下结果：

```
PQfname(res, 0)              foo
PQfname(res, 1)              BAR
PQfnumber(res, "FOO")        0
PQfnumber(res, "foo")        0
PQfnumber(res, "BAR")        -1
PQfnumber(res, "\"BAR\"")    1
```

PQftable

返回给定列所取自的表的 OID。列号从 0 开始。

```
Oid PQftable(const PGresult *res,
             int column_number);
```

如果列号超出范围、指定的列不是对表列的简单引用，或者使用 3.0 之前的协议，则返回 `InvalidOid`。可以查询系统表 `pg_class`，确定所引用的具体表。

包含 libpq 头文件后，将定义类型 `Oid` 和常量 `InvalidOid`。它们都属于某种整数类型。

PQftablecol

返回指定查询结果列所对应的表列在表中的列号。查询结果的列号从 0 开始，而表列的编号非零。

```
int PQftablecol(const PGresult *res,
                int column_number);
```

如果列号超出范围、指定的列不是对表列的简单引用，或者使用 3.0 之前的协议，则返回零。

PQfformat

返回表示给定列格式的格式代码。列号从 0 开始。

```
int PQfformat(const PGresult *res,
              int column_number);
```

格式代码零表示文本数据，格式代码一表示二进制数据。（其他代码保留供将来定义。）

PQftype

返回与给定列号相关联的数据类型。被返回的整数是该类型的内部 OID 号。列号从 0 开始。

```
Oid PQftype(const PGresult *res,
            int column_number);
```

可以查询系统表 `pg_type` 来获取各种数据类型的名称和属性。内置数据类型的 OID 定义在源码树的 `src/include/catalog/pg_type.h` 文件中。

PQfmod

返回与给定列号相关联的列的类型修饰符。列号从 0 开始。

```
int PQfmod(const PGresult *res,
           int column_number);
```

修饰符值的含义由数据类型决定，通常表示精度或大小限制。值 -1 表示“没有可用信息”。大多数数据类型不使用修饰符，此时该值始终为 -1。

PQfsize

返回给定列号对应列的大小，以字节计。列号从 0 开始。

```
int PQfsize(const PGresult *res,
            int column_number);
```

PQfsize 返回数据库行中为该列分配的空间，即服务器内部表示该数据类型所需的大小。（因此，它对客户端用处不大。）负值表示该数据类型是变长类型。

PQbinaryTuples

如果PGresult包含二进制数据，返回 1。如果包含的是文本数据，返回 0。

```
int PQbinaryTuples(const PGresult *res);
```

此函数已弃用（与 COPY 配合使用的情况除外），因为单个 PGresult 可能在部分列中包含文本数据，而在其他列中包含二进制数据。推荐使用 PQfformat。只有结果中的所有列都采用二进制格式（格式 1）时，PQbinaryTuples 才返回 1。

PQgetvalue

返回 PGresult 中某一行的单个字段值。行号和列号从 0 开始。调用者不应直接释放结果；当关联的 PGresult 句柄被传给 PQclear 时，结果会被释放。

```
char *PQgetvalue(const PGresult *res,
                 int row_number,
                 int column_number);
```

对于文本格式的数据，PQgetvalue 返回字段值的字符串表示，以零字节结尾。对于二进制格式的数据，返回值采用该数据类型的 typsend 和 typreceive 函数所决定的二进制表示。（这种情况下，值后面实际上也有一个零字节，但通常没有用处，因为值本身很可能包含零字节。）

如果字段值为 null，则返回空字符串。关于如何区分 null 值与空字符串值，参见 PQgetisnull。

PQgetvalue 返回的指针指向属于 PGresult 结构体的存储空间。不应修改它所指向的数据；如果需要在 PGresult 结构体的生命周期结束后继续使用这些数据，就必须显式地将数据复制到其他存储空间。

PQgetisnull

检查字段是否为 null 值。行号和列号从 0 开始。

```
int PQgetisnull(const PGresult *res,
                int row_number,
                int column_number);
```

如果字段为 null，此函数返回 1；如果包含非 null 值，则返回 0。（注意，对于 null 字段，PQgetvalue 返回空字符串，而非空指针。）

PQgetlength

返回字段值的实际长度，以字节计。行号和列号从 0 开始。

```
int PQgetlength(const PGresult *res,
                int row_number,
                int column_number);
```

这是该数据值的实际数据长度，即 `PQgetvalue` 所指对象的大小。对于文本格式的数据，它与 `strlen()` 的结果相同。对于二进制格式，这是必不可少的信息。注意，不应依赖 `PQfsize` 来获取实际数据长度。

`PQnparams`

返回一个预备语句的参数数量。

```
int PQnparams(const PGresult *res);
```

只有在查看`PQdescribePrepared`的结果时，这个函数才有用。对于其他类型的查询，它将返回零。

`PQparamtype`

返回所指示的语句参数的数据类型。参数号从 0 开始。

```
Oid PQparamtype(const PGresult *res, int param_number);
```

只有在查看`PQdescribePrepared`的结果时，这个函数才有用。对于其他类型的查询，它将返回零。

`PQprint`

将所有行输出到指定的输出流，并可选择输出列名。

```
void PQprint(FILE *fout,      /* 输出流 */
             const PGresult *res,
             const PQprintOpt *po);
typedef struct
{
    pqbool  header;      /* 打印输出字段标题和行数 */
    pqbool  align;       /* 填充并对齐字段 */
    pqbool  standard;    /* 旧的简陋格式 */
    pqbool  html3;       /* 输出 HTML 表格 */
    pqbool  expanded;    /* 展开表格 */
    pqbool  pager;       /* 需要时使用分页器输出 */
    char    *fieldSep;   /* 字段分隔符 */
    char    *tableOpt;   /* 用于 HTML 表格元素的属性 */
    char    *caption;    /* HTML 表格标题 */
    char    **fieldName; /* 以空指针结尾的替换字段名数组 */
} PQprintOpt;
```

psql 以前用此函数打印查询结果，现在已不再使用。注意，此函数假定所有数据都是文本格式。

31.3.3. 检索其他结果信息

这些函数被用来从`PGresult`对象中抽取其他信息。

`PQcmdStatus`

返回来自于产生`PGresult`的 SQL 命令的命令状态标签。

```
char *PQcmdStatus(PGresult *res);
```

通常这就是该命令的名称，但是它可能包括额外数据，例如已被处理的行数。
调用者不应该直接释放该结果。它将在相关的`PGresult`句柄被传递给`PQclear`之后被释放。

PQcmdTuples

返回受该 SQL 命令影响的行数。

```
char *PQcmdTuples(PGresult *res);
```

此函数返回一个字符串，其中包含产生该 PGresult 的 SQL 语句所影响的行数。此函数只能在执行 SELECT、CREATE TABLE AS、INSERT、UPDATE、DELETE、MOVE、FETCH 或 COPY 语句之后使用，也可以在对包含 INSERT、UPDATE 或 DELETE 语句的预备查询执行 EXECUTE 之后使用。如果产生 PGresult 的是其他命令，PQcmdTuples 将返回空字符串。调用者不应直接释放返回值；当关联的 PGresult 句柄被传给 PQclear 时，返回值会被释放。

PQoidValue

如果该SQL命令是一个正好将一行插入到具有 OID 的表的INSERT，或者是一个包含合适INSERT语句的预备查询的EXECUTE，这个函数返回被插入行的 OID。否则，这个函数返回InvalidOid。如果被INSERT语句影响的表不包含 OID，这个函数也将返回InvalidOid。

```
Oid PQoidValue(const PGresult *res);
```

PQoidStatus

此函数不是线程安全的，现已弃用，推荐改用 PQoidValue。它返回包含插入行 OID 的字符串，而 PQoidValue 返回 OID 值。

```
char *PQoidStatus(const PGresult *res);
```

31.3.4. 用于在 SQL 命令中嵌入字符串的转义

PQescapeLiteral

```
char *PQescapeLiteral(PGconn *conn, const char *str, size_t
 length);
```

为了让一个字符串可用于 SQL 命令，PQescapeLiteral会对它进行转义。当在 SQL 命令中把数据值作为字符串字面量插入时，这个函数很有用。一些字符（例如引号和反斜杠）必须经过转义，才不会被 SQL 解析器解释成特殊含义。PQescapeLiteral执行这种操作。

PQescapeLiteral 返回 *str* 参数的转义版本，存放在通过 malloc() 分配的内存中。结果不再需要时，应使用 PQfreemem() 释放该内存。输入不必以零字节结尾，末尾零字节也不应计入 *length*。（如果在处理完 *length* 个字节之前遇到末尾零字节，PQescapeLiteral 会在该字节处停止；这一行为类似于 strncpy。）返回字符串中的所有特殊字符均已替换，以便 PostgreSQL 字符串字面量解析器正确处理。还会添加一个末尾零字节。包围 PostgreSQL 字符串字面量所需的单引号包含在结果字符串中。

发生错误时，PQescapeLiteral返回NULL并且一个合适的消息会被存储在*conn*对象中。

> **提示**
>
> 在处理来自不可信来源的字符串时，正确转义尤其重要。
> 否则就会有安全风险：你很容易受到“SQL 注入”攻击，不期望的 SQL 命令可能会被送入数据库。

注意，当一个数据值被作为PQexecParams或相关例程中的一个独立参数传递时，没有必要做转义而且做转义也不正确。

PQescapeIdentifier

```
char *PQescapeIdentifier(PGconn *conn, const char *str, size_t
 length);
```

PQescapeIdentifier 对字符串进行转义，使其可用作 SQL 标识符，例如表名、列名或函数名。当用户提供的标识符可能包含未经转义就不会被 SQL 解析器视为标识符一部分的特殊字符，或包含需要保留大小写的大写字符时，此函数很有用。

PQescapeIdentifier 返回 *str* 参数作为 SQL 标识符的转义版本，存放在通过 malloc() 分配的内存中。结果不再需要时，必须使用 PQfreemem() 释放该内存。输入不必以零字节结尾，末尾零字节也不应计入 *length*。（如果在处理完 *length* 个字节之前遇到末尾零字节，PQescapeIdentifier 会在该字节处停止；这一行为类似于 strncpy。）返回字符串中的所有特殊字符均已替换，以便正确地作为 SQL 标识符处理。还会添加一个末尾零字节，并用双引号包围返回的字符串。

发生错误时，PQescapeIdentifier返回NULL并且一个合适的消息会被存储在*conn*对象中。

> **提示**
>
> 与字符串字面量一样，为防止 SQL 注入攻击，从不可信来源接收到的 SQL 标识符必须经过转义。

PQescapeStringConn

```
size_t PQescapeStringConn(PGconn *conn,
                          char *to, const char *from, size_t
 length,
                          int *error);
```

PQescapeStringConn 对字符串字面量进行转义，与 PQescapeLiteral 类似。与 PQescapeLiteral 不同，调用者需要提供大小合适的缓冲区。此外，PQescapeStringConn 不会生成包围 PostgreSQL 字符串字面量所需的单引号；应在包含转义结果的 SQL 命令中提供这些引号。*from* 参数指向待转义字符串的首字符，*length* 参数给出该字符串的字节数。输入不必以零字节结尾，末尾零字节也不应计入 *length*。（如果在处理完 *length* 个字节之前遇到末尾零字节，PQescapeStringConn 会在该字节处停止；这一行为类似于 strncpy。）*to* 必须指向一个缓冲区，其容量至少为 *length* 的两倍加一个字节，否则行为未定义。如果 *to* 与 *from* 字符串重叠，行为同样未定义。

如果 *error* 参数不是 NULL，则成功时将 *error 设为零，出错时设为非零。目前唯一可能的错误是源字符串中存在无效的多字节编码。出错时仍会生成输出字符串，但预计服务器会因其格式错误而拒绝它。发生错误时，无论 *error* 是否为 NULL，都会在 *conn* 对象中保存相应消息。

PQescapeStringConn返回写到*to*的字节数，不包括终止的零字节。

PQescapeString

PQescapeString 是 PQescapeStringConn 的旧版本，现已弃用。

```
size_t PQescapeString (char *to, const char *from, size_t
 length);
```

与 PQescapeStringConn 唯一的区别是，PQescapeString 不接受 PGconn 或 *error* 参数。因此，它无法根据连接属性（例如字符编码）调整行为，可能给出错误的结果。此外，它也无法报告错误情况。

PQescapeString 可以在同一时刻仅使用一个 PostgreSQL 连接的客户端程序中安全使用（在这种情况下，它能够“在内部”取得所需信息）。在其他情形下，它存在安全隐患，应改用 PQescapeStringConn。

PQescapeByteaConn

对二进制数据进行转义，使其能够在 SQL 命令中用作 bytea 类型的值。与 PQescapeStringConn 一样，这仅用于将数据直接插入 SQL 命令字符串的情况。

```
unsigned char *PQescapeByteaConn(PGconn *conn,
                                 const unsigned char *from,
                                 size_t from_length,
                                 size_t *to_length);
```

当某些字节值被用作一个SQL语句中的bytea字面量的一部分时，它们必须被转义。PQescapeByteaConn使用十六进制编码或反斜杠转义来转义这些字节。详见第 8.4 节。

from 参数指向待转义字符串的首字节，*from_length* 参数给出该二进制字符串的字节数。（末尾零字节既不需要，也不计入长度。）*to_length* 参数指向用于保存转义后字符串长度的变量。该结果字符串长度包含结果末尾的零字节。

PQescapeByteaConn 返回 *from* 参数所指二进制字符串的转义版本，存放在通过 malloc() 分配的内存中。结果不再需要时，应使用 PQfreemem() 释放该内存。返回字符串中的所有特殊字符都已替换，以便 PostgreSQL 字符串字面量解析器和 bytea 输入函数正确处理。还会添加一个末尾零字节。包围 PostgreSQL 字符串字面量所需的单引号不包含在结果字符串中。

在发生错误时，将返回一个空指针，并且一个合适的错误消息被存储在*conn*对象中。当前，唯一可能的错误是没有足够的内存用于结果串。

PQescapeBytea

PQescapeBytea 是 PQescapeByteaConn 的旧版本，现已弃用。

```
unsigned char *PQescapeBytea(const unsigned char *from,
                             size_t from_length,
                             size_t *to_length);
```

与 PQescapeByteaConn 唯一的区别是，PQescapeBytea 不接受 PGconn 参数。因此，PQescapeBytea 只能在同一时刻仅使用一个 PostgreSQL 连接的客户端程序中安全使用（在这种情况下，它能够“在内部”取得所需信息）。如果在使用多个数据库连接的程序中使用，它可能给出错误的结果（此时应使用 PQescapeByteaConn）。

PQunescapeBytea

将二进制数据的字符串表示转换为二进制数据，这是PQescapeBytea的逆操作。以文本格式取得bytea数据时需要此操作；以二进制格式取得时则不需要。

```
unsigned char *PQunescapeBytea(const unsigned char *from, size_t
 *to_length);
```

*from*参数指向一个字符串，例如对bytea列调用PQgetvalue时返回的字符串。PQunescapeBytea将这个字符串表示转换为二进制表示。它返回指向通过malloc()

分配的缓冲区的指针，出错时返回NULL，并将缓冲区大小存入*to_length*。
不再需要结果时，必须使用PQfreemem释放它。

此转换并不完全是PQescapeBytea的逆操作，因为从PQgetvalue
收到的字符串并非经过“转义”的形式。具体而言，这意味着无需考虑字符串引号，
因此也不需要PGconn参数。

31.4. 异步命令处理

该PQexec函数足以满足普通同步应用程序提交命令的需要。不过，它有一些缺点，
对某些用户可能很重要：

- PQexec会等待命令完成。该应用可能有其他的工作要做（例如维护用户界面），
  这时它将不希望阻塞等待回应。
- 因为客户端应用的执行在它等待结果时会被挂起，
  对于应用来说很难决定要不要尝试取消正在进行的命令（这可以在一个信号处理器中完成，但别无他法）。
- PQexec只能返回一个PGresult结构体。如果提交的命令串包含多个SQL命令，
  除了最后一个PGresult之外都会被PQexec丢弃。
- PQexec 总是收集命令的完整结果，并将其缓存在单个 PGresult 中。这简化了应用程序的错误处理逻辑，但对于包含大量行的结果，可能不切实际。

如果应用程序不希望受到这些限制，可以改用构成 PQexec 的底层函数：PQsendQuery 和 PQgetResult。此外，PQsendQueryParams、PQsendPrepare、PQsendQuery Prepared、PQsendDescribePrepared、PQsendDescribePortal 可以与 PQget Result 配合使用，分别实现 PQexecParams、PQprepare、PQexecPrepared、PQdescribePrepared、PQdescribePortal 的功能。

PQsendQuery

向服务器提交命令，不等待结果。命令发送成功时返回 1，否则返回 0（此时可使用 PQerrorMessage 获取更多失败信息）。

```
int PQsendQuery(PGconn *conn, const char *command);
```

成功调用 PQsendQuery 后，应调用 PQgetResult 一次或多次来获取结果。在 PQgetResult 返回空指针、表明命令已完成之前，不得在同一连接上再次调用 PQsend Query。

PQsendQueryParams

向服务器提交命令及独立指定的参数，不等待结果。

```
int PQsendQueryParams(PGconn *conn,
                      const char *command,
                      int nParams,
                      const Oid *paramTypes,
                      const char * const *paramValues,
                      const int *paramLengths,
                      const int *paramFormats,
                      int resultFormat);
```

该函数等价于 PQsendQuery，但查询参数可以与查询字符串分开指定。函数参数的处理方式与 PQexecParams 相同。与 PQexecParams 一样，它不能用于协议 2.0 的连接，并且查询字符串中只允许包含一条命令。

PQsendPrepare

发送按给定参数创建预备语句的请求，不等待完成。

```
int PQsendPrepare(PGconn *conn,
                  const char *stmtName,
                  const char *query,
                  int nParams,
                  const Oid *paramTypes);
```

这是 PQprepare 的异步版本：请求发送成功时返回 1，否则返回 0。调用成功后，再调用 PQgetResult，确定服务器是否成功创建了预备语句。函数参数的处理方式与 PQprepare 相同。与 PQprepare 一样，它不能用于协议 2.0 的连接。

PQsendQueryPrepared

发送使用给定参数执行预备语句的请求，不等待结果。

```
int PQsendQueryPrepared(PGconn *conn,
                        const char *stmtName,
                        int nParams,
                        const char * const *paramValues,
                        const int *paramLengths,
                        const int *paramFormats,
                        int resultFormat);
```

该函数类似于 PQsendQueryParams，但通过已创建的预备语句的名称指定要执行的命令，而非提供查询字符串。函数参数的处理方式与 PQexecPrepared 相同。与 PQexecPrepared 一样，它不能用于协议 2.0 的连接。

PQsendDescribePrepared

提交获取指定预备语句信息的请求，不等待完成。

```
int PQsendDescribePrepared(PGconn *conn, const char *stmtName);
```

这是PQdescribePrepared的异步版本：请求发送成功时返回 1，否则返回 0。调用成功后，再调用PQgetResult获取结果。函数参数的处理方式与PQdescribePrepared相同。与PQdescribePrepared一样，它不能用于协议 2.0 的连接。

PQsendDescribePortal

提交获取指定 portal 信息的请求，不等待完成。

```
int PQsendDescribePortal(PGconn *conn, const char *portalName);
```

这是PQdescribePortal的异步版本：请求发送成功时返回 1，否则返回 0。调用成功后，再调用PQgetResult获取结果。函数参数的处理方式与PQdescribePortal相同。与PQdescribePortal一样，它不能用于协议 2.0 的连接。

PQgetResult

等待先前的PQsendQuery，PQsendQueryParams，PQsendPrepare，PQsendQueryPrepared，PQsendDescribePrepared，或PQsendDescribePortal调用产生的下一个结果，并返回该结果。当命令执行完毕且不再有其他结果时，返回空指针。

```
PGresult *PQgetResult(PGconn *conn);
```

必须反复调用 PQgetResult，直到它返回空指针，表明命令已经完成。（如果当前没有正在执行的命令，调用 PQgetResult 会立即返回空指针。）对于 PQgetResult 返回的

非空指针，应使用前文介绍的 PGresult 访问函数处理相应结果。使用完毕后，不要忘记调用 PQclear 释放每个结果对象。注意，只有存在正在执行的命令，且所需响应数据尚未被 PQconsumeInput 读取时，PQgetResult 才会阻塞。

> **注意**
>
> 即使PQresultStatus指示发生了致命错误，也应该调用PQgetResult直到它返回一个空指针，以便libpq完全处理错误信息。

使用 PQsendQuery 和 PQgetResult 可以解决 PQexec 的一个问题：如果命令字符串包含多个 SQL 命令，就能分别获取这些命令的结果。（这也支持一种简单的重叠处理方式：客户端可以处理某条命令的结果，同时服务器继续处理同一命令字符串中后面的查询。）

使用 PQsendQuery 和 PQgetResult 还可以实现另一项常见需求：从大型查询结果中一次读取一行。详见第 31.5 节。

仅仅调用 PQgetResult 仍会使客户端阻塞，直到服务器完成下一条 SQL 命令。可以通过正确使用另外两个函数来避免这种情况：

PQconsumeInput

如果服务器有可读取的输入，则读取这些输入。

```
int PQconsumeInput(PGconn *conn);
```

PQconsumeInput 通常返回 1，表示“没有错误”；发生问题时则返回 0（此时可查看 PQerrorMessage）。注意，返回值并不说明是否实际读取了输入数据。调用 PQconsumeInput 后，应用程序可以检查 PQisBusy 和/或 PQnotifies，以确定其状态是否发生变化。

即使应用程序尚未准备好处理结果或通知，也可以调用 PQconsumeInput。此函数会读取可用数据并将其保存在缓冲区中，从而清除 select() 的可读就绪指示。因此，应用程序可以用 PQconsumeInput 立即清除 select() 的就绪条件，随后在合适的时候检查结果。

PQisBusy

如果命令仍在忙碌，则返回 1，意味着PQgetResult会阻塞等待输入。返回 0 则表示可以调用PQgetResult，并保证不会阻塞。

```
int PQisBusy(PGconn *conn);
```

PQisBusy本身将不会尝试从服务器读取数据，因此必须先调用PQconsumeInput，否则繁忙状态将永远不会结束。

使用这些函数的典型应用程序会在主循环中通过 select() 或 poll() 等待需要响应的各种条件。其中一个条件是服务器有可读取的输入；对于 select()，这意味着 PQsocket 标识的文件描述符上有可读数据。主循环检测到输入就绪时，应调用 PQconsumeInput 读取输入，然后调用 PQisBusy。如果 PQisBusy 返回假（0），就可以接着调用 PQgetResult。还可以调用 PQnotifies 检测 NOTIFY 消息（见第 31.8 节）。

一个使用PQsendQuery/PQgetResult的客户端也可以尝试取消一个正在被服务器处理的命令，见第 31.6 节。但是，不管PQcancel的返回值是什么，应用都必须继续使用PQgetResult进行正常的结果读取序列。一次成功的取消只会导致命令比不取消时更快终止。

使用上述函数可以避免在等待数据库服务器输入时阻塞。不过，应用程序仍可能在等待向服务器发送输出时阻塞。这种情况较少见，但发送很长的 SQL 命令

或数据值时可能发生。（如果应用程序通过 COPY IN 发送数据，发生的可能性则大得多。）为了防止这种情况，实现完全非阻塞的数据库操作，可以使用以下附加函数。

PQsetnonblocking

设置连接的非阻塞状态。

```
int PQsetnonblocking(PGconn *conn, int arg);
```

如果*arg*为1，则将连接状态设置为非阻塞，如果 *arg*为0，则设置为阻塞。如果成功返回0，出错返回-1。

在非阻塞状态下，对 PQsendQuery、PQputline、PQputnbytes 和 PQendcopy 的调用不会阻塞；如果需要再次调用，它们会返回错误。

请注意，PQexec不遵守非阻塞模式；如果调用它，它将以阻塞方式执行。

PQisnonblocking

返回数据库连接的阻塞状态。

```
int PQisnonblocking(const PGconn *conn);
```

如果连接设置为非阻塞模式，则返回1，如果为阻塞，则返回0。

PQflush

尝试将发送队列中的输出数据发送到服务器。成功（或发送队列为空）时返回 0；因某种原因失败时返回 -1；如果尚未能发送队列中的全部数据，则返回 1（这种情况只可能发生在非阻塞连接上）。

```
int PQflush(PGconn *conn);
```

在非阻塞连接上发送命令或数据后，应调用 PQflush。如果返回 1，就等待套接字变为可读或可写。套接字可写时，再次调用 PQflush；可读时，先调用 PQconsumeInput，再调用 PQflush。重复上述步骤，直到 PQflush 返回 0。（必须检查套接字是否可读，并用 PQconsumeInput 读完输入，因为服务器可能在尝试向客户端发送数据时阻塞，例如发送 NOTICE 消息；在客户端读取这些数据之前，服务器不会读取客户端发送的数据。）当 PQflush 返回 0 后，等待套接字变为可读，再按前述方法读取响应。

31.5. 逐行检索查询结果

通常，libpq 会收集一条 SQL 命令的完整结果，并将其作为单个 PGresult 返回给应用程序。对于返回大量行的命令，这种方式可能不可行。在这种情况下，应用程序可以在单行模式下使用 PQsendQuery 和 PQgetResult。此模式在从服务器收到结果行时，每次将一行返回给应用程序。

要进入单行模式，请在成功调用 PQsendQuery（或其同类函数）后，立即调用 PQsetSingleRowMode。此模式选择仅对当前正在执行的查询生效。然后按照第 31.4 节所述，反复调用 PQgetResult，直到返回空指针。如果查询返回了行，每一行都作为独立的 PGresult 对象返回。这些对象看起来与普通查询结果相同，只是状态码为 PGRES_SINGLE_TUPLE，而不是 PGRES_TUPLES_OK。最后一行之后，或者查询返回零行时立即返回一个不含行、状态为 PGRES_TUPLES_OK 的对象，表示不会再有更多行到达。（但注意，仍然必须继续调用 PQgetResult，直到返回空指针。）所有这些 PGresult 对象都包含与该查询普通 PGresult 对象相同的行描述数据（列名、类型等）。每个对象都应像往常一样使用 PQclear 释放。

PQsetSingleRowMode

为当前正在执行的查询选择单行模式。

```
int PQsetSingleRowMode(PGconn *conn);
```

此函数只能在调用 PQsendQuery 或其某个同类函数后立即调用，并且必须在对该连接执行任何其他操作之前，例如 PQconsumeInput 或 PQgetResult。在正确时机调用时，此函数会为当前查询启用单行模式并返回 1；否则，模式保持不变，函数返回 0。无论哪种情况，当前查询完成后都会恢复普通模式。

> **小心**
>
> 处理查询时，服务器可能先返回一些行，然后遇到错误，导致查询中止。通常，libpq 会丢弃这些行，只报告错误。但在单行模式下，这些行已经返回给了应用程序。
> 因此，应用程序会先看到一些 PGRES_SINGLE_TUPLE PGresult 对象，随后看到一个 PGRES_FATAL_ERROR 对象。为了保证正确的事务行为，如果查询最终失败，应用程序必须能够丢弃或撤销此前对这些行所做的全部操作。

31.6. 取消进行中的查询

客户端应用程序可以使用本节介绍的函数，请求取消服务器仍在处理的命令。

PQgetCancel

创建一个数据结构，其中包含取消通过特定数据库连接发出的命令所需的信息。

```
PGcancel *PQgetCancel(PGconn *conn);
```

PQgetCancel基于一个PGconn连接对象创建PGcancel对象。如果给定的*conn*为NULL或无效连接，则返回NULL。PGcancel是不透明结构体，不应由应用程序直接访问；它只能传给PQcancel或PQfreeCancel。

PQfreeCancel

释放由以下函数创建的数据结构：PQgetCancel。

```
void PQfreeCancel(PGcancel *cancel);
```

PQfreeCancel 释放之前由 PQgetCancel 创建的数据对象。

PQcancel

请求服务器停止处理当前命令。

```
int PQcancel(PGcancel *cancel, char *errbuf, int errbufsize);
```

返回值为 1 表示取消请求成功发送，为 0 表示未成功发送。如果未成功发送，*errbuf*将填充解释性错误消息。*errbuf*必须是大小为*errbufsize*的字符数组（推荐大小为 256 字节）。

不过，成功发送请求并不保证请求会产生效果。如果取消生效，当前命令会提前终止并返回错误结果。如果取消失败（例如服务器已经处理完该命令），则不会产生任何可见结果。

如果*errbuf*是信号处理程序中的局部变量，则PQcancel可以安全地从信号处理程序中调用。对于PQcancel而言，PGcancel对象是只读的，因此也可以从与操作PGconn对象的线程不同的线程中调用。

PQrequestCancel

PQrequestCancel是以下函数的已弃用变体：PQcancel。

```
int PQrequestCancel(PGconn *conn);
```

请求服务器放弃当前命令的处理。它直接作用于PGconn对象，
失败时会把错误消息存储到PGconn对象中（可通过PQerrorMessage获取）。
虽然功能相同，但这种方法在多线程程序或信号处理程序中并不安全，因为它可能覆盖PGconn中的错误消息，从而破坏当前连接上正在进行的操作。

31.7. 快速路径接口

PostgreSQL提供一种快速路径接口来向服务器发送简单的函数调用。

> **提示**
>
> 此接口已显得有些过时，因为可以通过建立定义函数调用的预备语句，
> 获得相近的性能和更多功能。随后使用二进制格式传输参数和结果来执行该语句，
> 就可以替代快速路径函数调用。

函数PQfn请求通过快速路径接口执行服务器函数。

```
PGresult *PQfn(PGconn *conn,
               int fnid,
               int *result_buf,
               int *result_len,
               int result_is_int,
               const PQArgBlock *args,
               int nargs);

typedef struct
{
    int len;
    int isint;
    union
    {
        int *ptr;
        int integer;
    } u;
} PQArgBlock;
```

fnid 参数是要执行函数的 OID。*args* 和 *nargs* 指定传给函数的参数，必须与函数声明中的参数列表匹配。参数结构体的 *isint* 字段为真时，*u.integer* 值会以指定长度的整数发送到服务器，该长度必须是 2 或 4 字节，并会进行适当的字节序转换。*isint* 为假时，位于 **u.ptr* 的指定数量字节会原样发送；数据必须符合服务器对该函数参数数据类型的二进制传输格式要求。（将 *u.ptr* 声明为 `int *` 是历史原因；将其视为 `void *` 更合适。）*result_buf* 指向用于存放函数返回值的缓冲区。调用者必须事先分配足够空间来保存返回值，这里不会检查！实际结果长度以字节为单位，返回到 *result_len* 指向的整数中。如果预期结果是 2 或 4 字节整数，将 *result_is_int* 设为 1，否则设为 0。将 *result_is_int* 设为 1 后，libpq 会按需转换字节序，使结果成为适合客户端机器的 `int` 值；注意，无论是哪种允许的结果大小，传入 **result_buf* 的都是 4 字节整数。*result_is_int* 为 0 时，服务器发送的二进制格式字节串会原样返回。（此时，将 *result_buf* 视为 `void *` 更合适。）

`PQfn` 总是返回有效的 `PGresult` 指针。使用结果前应检查其状态。不再需要结果时，调用者负责使用 `PQclear` 释放 `PGresult`。

注意，使用此接口时无法处理 NULL 参数、NULL 结果以及集合值结果。

31.8. 异步通知

PostgreSQL通过LISTEN和NOTIFY命令提供异步通知。客户端会话可使用LISTEN命令注册其感兴趣的特定通知通道（也可以用UNLISTEN命令停止监听）。当任何会话执行带有该通道名的NOTIFY命令时，所有监听该通道的会话都会异步收到通知。还可以传递一个“载荷”字符串，向监听者传递附加数据。

libpq应用把LISTEN、UNLISTEN和NOTIFY命令作为普通 SQL 命令提交。随后通过调用PQnotifies来检测NOTIFY消息的到达。

函数 PQnotifies 从已收到但尚未处理的服务器通知消息列表中返回下一条通知。没有待处理通知时返回空指针。一旦 PQnotifies 返回一条通知，该通知就被视为已处理，并从通知列表中移除。

```
PGnotify *PQnotifies(PGconn *conn);

typedef struct pgNotify
{
    char *relname;              /* 通知通道名称 */
    int  be_pid;                /* 发送通知的服务器进程 ID */
    char *extra;                /* 通知载荷字符串 */
} PGnotify;
```

处理完 PQnotifies 返回的 PGnotify 对象后，务必使用 PQfreemem 释放它。只需释放 PGnotify 指针；relname 和 extra 字段并非独立分配。（这些字段名称是历史遗留的；尤其是，通道名称与关系名称不必有任何关联。）

例 31.2给出了一个示例程序展示异步通知的使用。

PQnotifies实际上并不从服务器读取数据；它只是返回之前已被其他libpq函数读取的消息。在较早版本的libpq中，及时收到NOTIFY消息的唯一方法是不断提交命令，哪怕是空命令，然后在每次PQexec后检查PQnotifies。虽然这种方法仍然有效，但由于浪费处理能力，现已弃用。

当没有需要执行的实际命令时，一种更好的检查NOTIFY消息的方法是调用PQconsumeInput，然后检查PQnotifies。你可以使用select()等待服务器数据到达，这样在无事可做时就不会浪费CPU资源（参见PQsocket以获得可传给select()的文件描述符）。注意不管是用PQsendQuery/PQgetResult提交命令还是简单地使用PQexec，这种方法都能正常工作。不过，你应该记住在每次PQgetResult 或 PQexec之后检查PQnotifies，看看在命令的处理过程中是否有通知到达。

31.9. COPY命令相关的函数

PostgreSQL 的 COPY 命令提供了选项，可以通过 libpq 使用的网络连接读取或写入数据。本节介绍的函数允许应用程序通过提供或接收复制数据来使用这一能力。

整体流程如下：应用程序先通过 PQexec 或等效函数发出 SQL COPY 命令。如果命令没有错误，响应就是一个 PGresult 对象，其状态码为 PGRES_COPY_OUT 或 PGRES_COPY_IN，取决于指定的复制方向。应用程序随后应使用本节函数接收或发送数据行。数据传输完成后，会返回另一个 PGresult 对象，表示传输成功或失败：成功时状态为 PGRES_COMMAND_OK，出现问题时为 PGRES_FATAL_ERROR。此时可以通过 PQexec 继续发出 SQL 命令。（COPY 操作进行期间，不能在同一连接上执行其他 SQL 命令。）

如果一个COPY命令是通过PQexec在一个可能包含额外命令的字符串中发出的，那么应用在完成COPY序列之后必须继续用PQgetResult取得结果。只有在PQgetResult返回NULL时，我们才能确信PQexec的命令字符串已经处理完毕，并且可以安全地发出更多命令。

只有从 PQexec 或 PQgetResult 获得 PGRES_COPY_OUT 或 PGRES_COPY_IN 结果状态后，才应调用本节函数。

带有上述某个状态值的 PGresult 对象，还会携带关于即将开始的 COPY 操作的附加数据。这些数据可以通过下列函数获取，这些函数也用于查询结果：

PQnfields

返回要复制的列（字段）的数量。

PQbinaryTuples

0表示整体复制格式为文本（行由换行符分隔，列由分隔符分隔等）。
1表示整体复制格式为二进制。有关更多信息，请参见COPY。

PQfformat

返回与复制操作的每列关联的格式代码（0表示文本，1表示二进制）。
当整体复制格式为文本时，每列的格式代码始终为零，
但二进制格式可以支持文本和二进制列。（但是，在当前的 COPY 实现中，只有二进制列出现在二进制复制中；因此，每列格式目前始终与整体格式匹配。）

> **注意**
>
> 这些附加数据值仅在使用协议 3.0 时可用。使用协议 2.0 时，这些函数都返回 0。

31.9.1. 用于发送COPY数据的函数

这些函数用于在 COPY FROM STDIN 期间发送数据。如果连接不处于 COPY_IN 状态，调用它们会失败。

PQputCopyData

在COPY_IN状态中向服务器发送数据。

```
int PQputCopyData(PGconn *conn,
                  const char *buffer,
                  int nbytes);
```

将指定 *buffer* 中长度为 *nbytes* 的 COPY 数据传输到服务器。数据发送成功时返回 1；因尝试会阻塞而未发送时返回零（仅可能发生在连接处于非阻塞模式时）；发生错误时返回 -1。（返回 -1 时，可用 PQerrorMessage 获取详细信息。返回零时，应等待可写就绪后重试。）

应用程序可以将 COPY 数据流分成任意方便大小的数据块，逐块装入缓冲区。发送时，这些数据块的边界没有语义含义。数据流内容必须符合 COPY 命令预期的数据格式；详见COPY。

PQputCopyEnd

在COPY_IN状态中向服务器发送数据结束的指示。

```
int PQputCopyEnd(PGconn *conn,
                 const char *errormsg);
```

如果 *errormsg* 为 NULL，则成功结束 COPY_IN 操作。如果 *errormsg* 不为 NULL，则强制 COPY 失败，并将 *errormsg* 指向的字符串用作错误消息。（但不应假定服务器一定会返回这条完全相同的错误消息，因为服务器可能已经因自身原因使 COPY 失败。还要注意，在使用 3.0 之前协议的连接上，强制失败选项不起作用。）

终止数据发送成功时返回 1；因尝试会阻塞而未发送时返回零（仅可能发生在连接处于非阻塞模式时）；发生错误时返回 -1。（返回 -1 时，可用 PQerrorMessage 获取详细信息。返回零时，应等待可写就绪后重试。）

成功调用 PQputCopyEnd 后，调用 PQgetResult 获取 COPY 命令的最终结果状态。可以按通常方式等待该结果就绪，然后恢复正常操作。

31.9.2. 用于接收COPY数据的函数

这些函数用于在COPY TO STDOUT的过程中接收数据。如果连接不在COPY_OUT状态，那么调用它们将会失败。

PQgetCopyData

在COPY_OUT状态下从服务器接收数据。

```
int PQgetCopyData(PGconn *conn,
                  char **buffer,
                  int async);
```

在 COPY 期间尝试从服务器获取下一行数据。每次总是返回一个完整数据行；如果只有部分行可用，则不返回。成功返回数据行时，会分配一块内存保存数据。*buffer* 参数必须为非 NULL。**buffer* 会被设置为指向所分配的内存；如果没有返回缓冲区，则设为 NULL。非 NULL 的结果缓冲区在不再需要时应使用 PQfreemem 释放。

成功返回一行时，返回值是该行的数据字节数，始终大于零。
返回的字符串总是以零字节结尾，不过这可能仅对文本 COPY 有用。返回零表示 COPY 仍在进行，但尚无可用行（仅在 *async* 为真时可能发生）。返回 -1 表示 COPY 已完成；返回 -2 表示发生了错误（可用 PQerrorMessage 查看原因）。

当 *async* 为真（非零）时，PQgetCopyData 不会阻塞等待输入；如果 COPY 仍在进行，但没有完整行可用，则返回零。（此时，应等待读就绪，先调用 PQconsumeInput，再调用 PQgetCopyData。）当 *async* 为假（零）时，PQgetCopyData 会阻塞，直到数据可用或操作完成。

在 PQgetCopyData 返回 -1 后，调用 PQgetResult 获取 COPY 命令的最终结果状态。可以按通常方式等待该结果就绪，然后恢复正常操作。

31.9.3. 用于COPY的废弃函数

这些函数使用较旧的方式处理 COPY。虽然仍然可用，但由于错误处理欠佳、检测数据结束的方式不便，而且缺少对二进制或非阻塞传输的支持，已被弃用。

PQgetline

将服务器传来的、以换行符结尾的一行字符读入大小为 *length* 的字符串缓冲区。

```
int PQgetline(PGconn *conn,
              char *buffer,
              int length);
```

此函数最多将 *length*-1 个字符复制到缓冲区，并将末尾的换行符转换为零字节。PQgetline 在输入结束时返回 EOF，读完一整行时返回 0，缓冲区已满但尚未读到末尾换行符时返回 1。

注意，应用程序必须检查新读入的一行是否仅由 \. 两个字符组成，这表示服务器已发送完 COPY 命令的结果。如果可能收到长度超过 *length*-1 个字符的行，必须确保正确识别 \. 行，例如不能把长数据行的末尾误当作终止行。

PQgetlineAsync

以非阻塞方式将服务器传来的一行 COPY 数据读入缓冲区。

```
int PQgetlineAsync(PGconn *conn,
                   char *buffer,
                   int bufsize);
```

此函数类似于 PQgetline，但可用于必须异步读取 COPY 数据的应用程序，即读取时不阻塞。发出 COPY 命令并收到 PGRES_COPY_OUT 响应后，应用程序应调用 PQconsumeInput 和 PQgetlineAsync，直到检测到数据结束信号。

与 PQgetline 不同，此函数会负责检测数据结束。

每次调用时，如果 libpq 的输入缓冲区中有完整数据行，PQgetlineAsync 就会返回数据；否则，要等该行剩余部分到达后才返回数据。识别到复制数据结束标记时返回 -1，没有可用数据时返回 0，否则返回正数，表示返回的数据字节数。返回 -1 后，调用者必须接着调用 PQendcopy，然后恢复正常处理。

返回的数据不会跨越数据行边界。只要可能，每次就返回一整行；
但如果调用者提供的缓冲区太小，容不下服务器发送的一行，则只返回部分行。
对于文本数据，可检查最后返回的字节是否为 \n，以判断是否返回了完整行。（对于二进制 COPY，则必须实际解析 COPY 数据格式才能作出相同判断。）返回的字符串不以零字节结尾。（如果要自行添加末尾的零字节，务必将传入的 *bufsize* 设置为比实际可用空间少一字节。）

PQputline

向服务器发送以零字节结尾的字符串。成功时返回 0，无法发送字符串时返回 EOF。

```
int PQputline(PGconn *conn,
              const char *string);
```

连续调用 PQputline 发送的 COPY 数据流，与 PQgetlineAsync 返回的数据格式相同。不过，应用程序不必在每次 PQputline 调用中恰好发送一个数据行；每次发送部分行或多行也可以。

> **注意**
>
> 在 PostgreSQL 协议 3.0 之前，应用程序必须显式发送由 \. 两个字符组成的最后一行，告知服务器应用程序已发送完 COPY 数据。虽然这种方式仍然有效，但已被弃用，\. 的特殊含义预计会在未来版本中移除。发送完实际数据后，调用 PQendcopy 即可。

PQputnbytes

向服务器发送不以零字节结尾的字符串。成功时返回 0，无法发送字符串时返回 EOF。

```
int PQputnbytes(PGconn *conn,
                const char *buffer,
                int nbytes);
```

此函数与 PQputline 完全相同，只是直接指定了要发送的字节数，因此数据缓冲区不必以零字节结尾。发送二进制数据时可使用此函数。

PQendcopy

与服务器同步。

```
int PQendcopy(PGconn *conn);
```

此函数会等待服务器完成复制。应在使用 PQputline 向服务器发送最后一个字符串后，或使用 PQgetline 从服务器接收最后一个字符串后调用它。必须调用此函数，否则服务器与客户端会“失去同步”。函数返回后，服务器便准备好接收下一条 SQL 命令。成功完成时返回 0，否则返回非零值。（返回非零值时，可用 PQerrorMessage 获取详细信息。）

使用 PQgetResult 时，收到 PGRES_COPY_OUT 结果后，应用程序应反复调用 PQgetline，并在看到终止行后调用 PQendcopy。随后应回到 PQgetResult 循环，直到 PQgetResult 返回空指针。类似地，收到 PGRES_COPY_IN 结果后，应连续调用 PQputline，再调用 PQendcopy，然后回到 PQgetResult 循环。这样可以保证嵌在一系列 SQL 命令中的 COPY 命令正确执行。

旧的应用很可能会通过PQexec提交一个COPY命令并且假定事务在PQendcopy之后完成。只有在COPY是命令字符串中唯一的SQL命令时才能正确工作。

31.10. 控制函数

这些函数控制 libpq 行为的各种细节。

PQclientEncoding

返回客户端编码。

```
int PQclientEncoding(const PGconn *conn);
```

注意，返回的是编码 ID，而不是 EUC_JP 这样的符号字符串。失败时返回 -1。要将编码 ID 转换为编码名称，可以使用：

```
char *pg_encoding_to_char(int encoding_id);
```

PQsetClientEncoding

设置客户端编码。

```
int PQsetClientEncoding(PGconn *conn, const char *encoding);
```

*conn*是到服务器的连接，而*encoding*是要使用的编码。如果成功设置编码，函数返回 0，否则返回 -1。此连接的当前编码可以通过以下函数确定：PQclientEncoding。

PQsetErrorVerbosity

设置以下函数所返回消息的详细程度：PQerrorMessage和PQresultErrorMessage。

```
typedef enum
{
    PQERRORS_TERSE,
    PQERRORS_DEFAULT,
    PQERRORS_VERBOSE
} PGVerbosity;

PGVerbosity PQsetErrorVerbosity(PGconn *conn, PGVerbosity
 verbosity);
```

PQsetErrorVerbosity设置详细程度模式，并返回该连接先前的设置。在TERSE模式下，返回的消息只包括严重性、主要文本和位置，通常一行就能容纳。默认模式生成的消息除上述字段外，还包括详细信息、

提示或上下文字段（这些内容可能跨越多行）。VERBOSE模式包括所有可用字段。
改变详细程度不会影响已有PGresult对象中的消息，只影响随后创建的对象。

PQtrace

启用客户端/服务器通信跟踪，将跟踪信息写入调试文件流。

```
void PQtrace(PGconn *conn, FILE *stream);
```

> **注意**
>
> 在 Windows 上，如果 libpq 库与应用程序使用不同的标志编译，调用此函数会导致应用程序崩溃，因为 FILE 指针的内部表示不同。具体而言，库与所有使用它的应用程序都应使用相同的多线程/单线程、发布/调试和静态/动态标志。

PQuntrace

禁用由 PQtrace 启动的跟踪。

```
void PQuntrace(PGconn *conn);
```

31.11. 杂项函数

一如往常，总有一些函数不适合放在任何其他地方。

PQfreemem

释放libpq分配的内存。

```
void PQfreemem(void *ptr);
```

释放 libpq 分配的内存，特别是 PQescapeByteaConn、PQescapeBytea、PQunescapeBytea 和 PQnotifies 分配的内存。在 Microsoft Windows 上，务必使用此函数，而不是 free()。这是因为只有 DLL 与应用程序使用相同的多线程/单线程、发布/调试和静态/动态标志，才能在 DLL 中分配内存并在应用程序中释放它。在 Microsoft Windows 以外的平台上，此函数与标准库函数 free() 相同。

PQconninfoFree

释放PQconndefaults 或 PQconninfoParse分配的数据结构。

```
void PQconninfoFree(PQconninfoOption *connOptions);
```

仅调用 PQfreemem 不足以完成此项释放，因为数组还包含指向附属字符串的引用。

PQencryptPassword

准备一个PostgreSQL密码的加密形式。

```
char * PQencryptPassword(const char *passwd, const char *user);
```

这个函数旨在用于那些希望发送类似于ALTER USER joe PASSWORD 'pwd'命令的客户端应用。不在这样一个命令中发送原始的明文密码是一个好习惯，
因为它可能被暴露在命令日志、活动显示等等中。相反，
在发送之前使用这个函数可以将密码转换为加密的形式。
参数是明文密码和该密码所属用户的 SQL 名称。返回值是由malloc分配的字符串，如果

内存不足则为NULL。调用者可以假定该字符串不包含任何需要转义的特殊字符。
用完之后用PQfreemem释放结果。

PQmakeEmptyPGresult

用给定的状态，构造一个空PGresult对象。

```
PGresult *PQmakeEmptyPGresult(PGconn *conn, ExecStatusType
 status);
```

这是libpq内部用于分配并初始化一个空PGresult对象的函数。如果无法分配内存，此函数返回NULL。将它导出供外部调用，是因为一些应用需要自行生成结果对象，特别是带有错误状态的对象。如果*conn* 不为 null，并且*status*表示错误，指定连接的当前错误消息会被复制到PGresult中。此外，如果*conn* 不为 null，连接中注册的所有事件过程也会被复制到PGresult中。（这些过程不会收到PGEVT_RESULTCREATE调用，但可参见PQfireResultCreateEvents。）注意，最终应对该对象调用PQclear，就像处理libpq自身返回的PGresult一样。

PQfireResultCreateEvents

为每一个在PGresult对象中注册的事件过程触发一个PGEVT_RESULTCREATE事件（见第 31.13 节）。成功时返回非 0，如果任何事件过程失败则返回 0。

```
int PQfireResultCreateEvents(PGconn *conn, PGresult *res);
```

conn 参数会传递给事件过程，但此函数不会直接使用它。如果事件过程不使用此参数，则可以传入 NULL。

已经接收到这个对象的PGEVT_RESULTCREATE或PGEVT_RESULTCOPY事件的事件过程不会被再次触发。

此函数与 PQmakeEmptyPGresult 分开的主要原因是，通常适合先创建 PGresult 并填充数据，然后再调用事件过程。

PQcopyResult

创建 PGresult 对象的副本。副本与源结果没有任何关联，不再需要副本时必须调用PQclear。函数失败时返回 NULL。

```
PGresult *PQcopyResult(const PGresult *src, int flags);
```

这不是为了制作一个精确的副本。返回的结果总是放在PGRES_TUPLES_OK状态中，并且不复制源中的任何错误消息。（但是会复制命令状态字符串。）*flags*参数确定要复制的其他内容。它是几个标志的按位或。PG_COPYRES_ATTRS指定复制源结果的属性（列定义）。PG_COPYRES_TUPLES指定复制源结果的元组。（这也意味着复制属性。）PG_COPYRES_NOTICEHOOKS指定复制源结果的通知钩子。PG_COPYRES_EVENTS指定复制源结果的事件。
（但不复制与源相关的任何实例数据。）

PQsetResultAttrs

设置PGresult对象的属性。

```
int PQsetResultAttrs(PGresult *res, int numAttributes, PGresAtt
Desc *attDescs);
```

提供的*attDescs*被复制到结果中。如果*attDescs*指针为NULL或*numAttributes*小于1，那么请求将被忽略并且函数成功。如果*res*已经包含属性，那么函数会失败。如果函数失败，返回值是 0。如果函数成功，返回值是非 0。

PQsetvalue

设置 PGresult 对象中某个元组的字段值。

```
int PQsetvalue(PGresult *res, int tup_num, int field_num, char
 *value, int len);
```

此函数会根据需要自动扩展结果内部的元组数组。不过，*tup_num* 参数必须小于或等于 PQntuples，也就是说，每次只能向元组数组增加一个元组。已有元组的任何字段都可以按任意顺序修改。如果 *field_num* 指定的位置已有值，该值会被覆盖。如果 *len* 为 -1 或 *value* 为 NULL，则将该字段设置为 SQL null 值。*value* 会被复制到结果的私有存储中，因此函数返回后就不再需要它。函数失败时返回零，成功时返回非零值。

PQresultAlloc

为一个PGresult对象分配附属存储。

```
void *PQresultAlloc(PGresult *res, size_t nBytes);
```

使用此函数分配的所有内存都会在清除 *res* 时释放。函数失败时返回 NULL。与 malloc 一样，返回的内存保证满足任意数据类型的对齐要求。

PQlibVersion

返回所使用的libpq版本。

```
int PQlibVersion(void);
```

在运行时，这个函数的结果可以被用来判断在当前已载入的 libpq 版本中特定的功能是否可用。例如，这个函数可以被用来判断PQconnectdb可以使用哪些连接选项，或者是否支持PostgreSQL 9.0 中加入的bytea的hex输出格式。

这个数字是这样形成的：将主版本、次版本和修订版本号分别转换成两位十进制数，然后把它们拼接在一起。例如，版本9.1将被返回为90100，而版本9.1.2将被返回为90102（不显示前导零）。

注意

这个函数出现于PostgreSQL版本 9.1，因此它不能被用来在早期的版本中检测所需的功能，因为链接到它将会创建一个对版本9.1的链接依赖。

31.12. 通知处理

服务器生成的通知和警告消息不由查询执行函数返回，因为它们并不表示查询失败。这些消息会传递给通知处理函数，处理函数返回后，执行正常继续。默认通知处理函数将消息打印到 stderr，应用程序也可以提供自己的处理函数来替换这一行为。

由于历史原因，通知处理有两个级别，称为通知接收器和通知处理器。通知接收器的默认行为是格式化通知并且将一个字符串传递给通知处理器来打印。不过，如果一个应用选择提供自己的通知接收器，它通常会忽略通知处理器层并且在通知接收器中完成所有工作。

函数PQsetNoticeReceiver 为一个连接对象设置或者检查当前的通知接收器。相似地，PQsetNoticeProcessor 设置或检查当前的通知处理器。

```
typedef void (*PQnoticeReceiver) (void *arg, const PGresult *res);

PQnoticeReceiver
PQsetNoticeReceiver(PGconn *conn,
                    PQnoticeReceiver proc,
                    void *arg);

typedef void (*PQnoticeProcessor) (void *arg, const char *message);

PQnoticeProcessor
PQsetNoticeProcessor(PGconn *conn,
                     PQnoticeProcessor proc,
                     void *arg);
```

这些函数中的每一个会返回之前的通知接收器或处理器函数指针，并且设置新值。
如果你提供了一个空函数指针，将不会采取任何动作，只会返回当前指针。

从服务器收到通知或警告消息，或者 libpq 内部生成这类消息时，便会调用通知接收器函数。消息以状态为 `PGRES_NONFATAL_ERROR` 的 `PGresult` 对象传给该函数。（接收器可使用 `PQresultErrorField` 提取单个字段，或使用 `PQresultErrorMessage` 获取完整的预格式化消息。）传给 `PQsetNoticeReceiver` 的同一个 void 指针也会传入。（必要时，可通过该指针访问应用程序特有的状态。）

默认的通知接收器会简单地抽取消息（使用`PQresultErrorMessage`）
并且将它传递给通知处理器。

通知处理器负责处理以文本形式提供的通知或警告消息。
传给它的参数包括消息字符串（含末尾的换行符），以及先前传给 `PQsetNoticeProcessor` 的同一个 void 指针。（必要时，可通过该指针访问应用程序特有的状态。）

默认的通知处理器很简单：

```
static void
defaultNoticeProcessor(void *arg, const char *message)
{
    fprintf(stderr, "%s", message);
}
```

设置通知接收器或处理器后，应当预期：只要 `PGconn` 对象或由它生成的 `PGresult` 对象仍存在，就可能调用该函数。创建 `PGresult` 时，会将 `PGconn` 当前的通知处理指针复制到 `PGresult` 中，供 `PQgetvalue` 等函数在需要时使用。

31.13. 事件系统

libpq的事件系统被设计为通知已注册的事件处理器它感兴趣的libpq事件，例如`PGconn`以及`PGresult`对象的创建和销毁。一个主要用途是允许应用将自己的数据与一个`PGconn`或者`PGresult`关联在一起，并且确保那些数据在适当的时候被释放。

每个注册的事件处理程序都与两项数据相关联，libpq仅将其视为不透明的`void *`指针。有一个透传指针，由应用程序在向 `PGconn` 注册事件处理程序时提供。透传指针在`PGconn`及其生成的所有`PGresult`的生命周期内永远不会更改；因此，如果使用，
它必须指向长期存在的数据。此外，还有一个实例数据指针，在每个`PGconn`和`PGresult`中一开始都是`NULL`。可以使用`PQinstanceData`、`PQsetInstanceData`、`PQresultInstanceData`和`PQresultSetInstanceData`函数来操作此指针。请注意，
与透传指针不同，`PGconn`的实例数据不会自动继承到从中创建的`PGresult`。
libpq不知道透传和实例数据指针指向的内容（如果有的话），也永远不会尝试释放它们 – 这是事件处理程序的责任。

31.13.1. 事件类型

枚举PGEventId命名了事件系统处理的事件类型。它的所有值的名称都以PGEVT开始。对于每一种事件类型，都有一个相应的事件信息结构体用来承载传递给事件处理器的参数。事件类型是：

PGEVT_REGISTER

注册事件在PQregisterEventProc被调用时触发。
此时最适合初始化事件过程可能需要的instanceData。
每个连接中的每个事件过程只会触发一次注册事件。如果事件过程失败，则中止注册。

```
typedef struct
{
    PGconn *conn;
} PGEventRegister;
```

收到PGEVT_REGISTER事件时，应将*evtInfo*指针强制转换为PGEventRegister *。此结构体包含一个PGconn，它应处于CONNECTION_OK状态；如果调用PQregisterEventProc紧接在取得正常的PGconn之后，就能保证这一点。返回失败代码时，必须完成全部清理工作，因为不会发送PGEVT_CONNDESTROY事件。

PGEVT_CONNRESET

连接重置事件会在完成以下调用时触发：PQreset或PQresetPoll。在这两种情况下，只有重置成功才会触发该事件。如果事件过程失败，整个连接重置就会失败；PGconn会被置于CONNECTION_BAD状态，并且PQresetPoll将返回PGRES_POLLING_FAILED。

```
typedef struct
{
    PGconn *conn;
} PGEventConnReset;
```

收到PGEVT_CONNRESET事件时，应将*evtInfo*指针强制转换为PGEventConnReset *。虽然其中的PGconn刚刚被重置，但所有事件数据都保持不变。应利用此事件重置、重新加载或重新查询相关联的instanceData。注意，即使事件过程未能处理PGEVT_CONNRESET，它仍会在连接关闭时收到PGEVT_CONNDESTROY事件。

PGEVT_CONNDESTROY

连接销毁事件由以下调用触发：PQfinish。事件过程负责正确清理其事件数据，因为libpq 无法管理这部分内存。如果不清理，就会造成内存泄漏。

```
typedef struct
{
    PGconn *conn;
} PGEventConnDestroy;
```

收到PGEVT_CONNDESTROY事件时，应将*evtInfo*指针强制转换为PGEventConnDestroy *。该事件触发于以下函数执行任何其他清理工作之前：PQfinish。事件过程的返回值会被忽略，因为无法通过以下函数报告失败：PQfinish。此外，事件过程失败不应中止清理不再使用的内存的过程。

PGEVT_RESULTCREATE

任何生成结果的查询执行函数都会触发结果创建事件，其中包括PQgetResult。只有成功创建结果后才会触发该事件。

```
typedef struct
```

```
{
    PGconn *conn;
    PGresult *result;
} PGEventResultCreate;
```

收到PGEVT_RESULTCREATE事件时，应将*evtInfo*指针转换为PGEventResultCreate *。其中，*conn*是用于生成结果的连接。这是初始化需要与结果关联的instanceData的理想位置。如果事件过程失败，结果会被清除，失败也会向外传递。事件过程不得自行调用PQclear来清除结果对象。返回失败代码时，必须完成所有清理工作，因为不会发送PGEVT_RESULTDESTROY事件。

PGEVT_RESULTCOPY

结果复制事件会在调用PQcopyResult时触发。只有复制完成后才会触发该事件。只有为源结果成功处理过PGEVT_RESULTCREATE或PGEVT_RESULTCOPY事件的事件过程，才会收到PGEVT_RESULTCOPY事件。

```
typedef struct
{
    const PGresult *src;
    PGresult *dest;
} PGEventResultCopy;
```

收到PGEVT_RESULTCOPY事件时，应将*evtInfo*指针强制转换为PGEventResultCopy *。其中，*src*结果是复制源，而*dest*结果是复制目标。可以利用此事件对instanceData进行深复制，因为PQcopyResult无法完成这项工作。如果事件过程失败，整个复制操作就会失败，并且*dest*结果将被清除。返回失败代码时，必须完成所有清理工作，因为不会为目标结果发送PGEVT_RESULTDESTROY事件。

PGEVT_RESULTDESTROY

结果销毁事件由以下调用触发：PQclear。事件过程负责正确清理其事件数据，因为libpq 无法管理这部分内存。如果不清理，就会造成内存泄漏。

```
typedef struct
{
    PGresult *result;
} PGEventResultDestroy;
```

收到PGEVT_RESULTDESTROY事件时，应将*evtInfo*指针强制转换为PGEventResultDestroy *。该事件触发于以下函数执行任何其他清理工作之前：PQclear。事件过程的返回值会被忽略，因为无法通过以下函数报告失败：PQclear。此外，事件过程失败不应中止清理不再使用的内存的过程。

31.13.2. 事件回调函数

PGEventProc

PGEventProc 是通过 typedef 定义的事件过程指针类型，也就是接收 libpq 事件的用户回调函数的指针类型。事件过程的签名必须为：

```
int eventproc(PGEventId evtId, void *evtInfo, void *passThrough)
```

evtId 参数指示发生了哪一种 PGEVT 事件。必须将 *evtInfo* 指针强制转换为适当的结构体类型，以获取关于该事件的更多信息。*passThrough* 参数是在注册事件过程时传给 PQregisterEventProc 的指针。函数应在成功时返回非零值，在失败时返回零。

在任何一个PGconn中，一个特定事件过程只能被注册一次。
这是因为该过程的地址被用作查找键来标识相关的实例数据。

> **小心**
>
> 在 Windows 上，函数可能有两个不同的地址：一个在 DLL 外部可见，另一个在 DLL 内部可见。使用 libpq 的事件过程函数时，务必始终使用其中同一个地址，否则会产生混淆。确保代码正常工作的最简单做法，是将事件过程声明为 static。如果需要在过程所在的源文件之外取得其地址，
> 应提供一个单独的函数来返回该地址。

31.13.3. 事件支持函数

PQregisterEventProc

为 libpq 注册一个事件回调过程。

```
int PQregisterEventProc(PGconn *conn, PGEventProc proc,
                        const char *name, void *passThrough);
```

对于希望接收其事件的每个 PGconn，都必须注册一次事件过程。一个连接可注册的事件过程数量只受内存限制。函数成功时返回非零值，失败时返回零。

当一个 libpq 事件被触发时，*proc*参数将被调用。它的内存地址也被用来查找instanceData。*name*参数被用来在错误消息中引用该事件过程。
这个值不能是NULL或一个零长度串。名字串被复制到PGconn中，
因此传递进来的东西不需要长期存在。当一个事件发生时，*passThrough*
指针被传递给*proc*。这个参数可以是NULL。

PQsetInstanceData

设置连接*conn*的用于过程*proc*的instanceData为*data*。它在成功时返回非零值，失败时返回零（只有*proc*没有被正确地注册在*conn*中，才可能会失败）。

```
int PQsetInstanceData(PGconn *conn, PGEventProc proc, void
 *data);
```

PQinstanceData

返回连接*conn*的与过程*proc*相关的instanceData，如果没有则返回NULL。

```
void *PQinstanceData(const PGconn *conn, PGEventProc proc);
```

PQresultSetInstanceData

将结果中针对 *proc* 的 instanceData 设置为 *data*。成功时返回非零值，失败时返回零。（只有当 *proc* 未在结果中正确注册时，才可能失败。）

```
int PQresultSetInstanceData(PGresult *res, PGEventProc proc,
 void *data);
```

PQresultInstanceData

返回结果的与过程*proc*相关的instanceData，如果没有则返回NULL。

```
void *PQresultInstanceData(const PGresult *res, PGEventProc
 proc);
```

31.13.4. 事件实例

下面给出一个示例框架，用于管理与 libpq 连接和结果关联的私有数据。

```
/* required header for libpq events (note: includes libpq-fe.h) */
#include <libpq-events.h>

/* The instanceData */
typedef struct
{
    int n;
    char *str;
} mydata;

/* PGEventProc */
static int myEventProc(PGEventId evtId, void *evtInfo, void *pass
Through);

int
main(void)
{
    mydata *data;
    PGresult *res;
    PGconn *conn =
        PQconnectdb("dbname = postgres");

    if (PQstatus(conn) != CONNECTION_OK)
    {
        fprintf(stderr, "Connection to database failed: %s",
                PQerrorMessage(conn));
        PQfinish(conn);
        return 1;
    }

    /* called once on any connection that should receive events.
     * Sends a PGEVT_REGISTER to myEventProc.
     */
    if (!PQregisterEventProc(conn, myEventProc, "mydata_proc",
 NULL))
    {
        fprintf(stderr, "Cannot register PGEventProc\n");
        PQfinish(conn);
        return 1;
    }

    /* conn instanceData is available */
    data = PQinstanceData(conn, myEventProc);

    /* Sends a PGEVT_RESULTCREATE to myEventProc */
    res = PQexec(conn, "SELECT 1 + 1");

    /* result instanceData is available */
    data = PQresultInstanceData(res, myEventProc);

    /* If PG_COPYRES_EVENTS is used, sends a PGEVT_RESULTCOPY to my
EventProc */
    res_copy = PQcopyResult(res, PG_COPYRES_TUPLES | PG_COPYRES_
EVENTS);
```

```
    /* result instanceData is available if PG_COPYRES_EVENTS was
     * used during the PQcopyResult call.
     */
    data = PQresultInstanceData(res_copy, myEventProc);

    /* Both clears send a PGEVT_RESULTDESTROY to myEventProc */
    PQclear(res);
    PQclear(res_copy);

    /* Sends a PGEVT_CONNDESTROY to myEventProc */
    PQfinish(conn);

    return 0;
}

static int
myEventProc(PGEventId evtId, void *evtInfo, void *passThrough)
{
    switch (evtId)
    {
        case PGEVT_REGISTER:
        {
            PGEventRegister *e = (PGEventRegister *)evtInfo;
            mydata *data = get_mydata(e->conn);

            /* associate app specific data with connection */
            PQsetInstanceData(e->conn, myEventProc, data);
            break;
        }

        case PGEVT_CONNRESET:
        {
            PGEventConnReset *e = (PGEventConnReset *)evtInfo;
            mydata *data = PQinstanceData(e->conn, myEventProc);

            if (data)
              memset(data, 0, sizeof(mydata));
            break;
        }

        case PGEVT_CONNDESTROY:
        {
            PGEventConnDestroy *e = (PGEventConnDestroy *)evtInfo;
            mydata *data = PQinstanceData(e->conn, myEventProc);

            /* free instance data because the conn is being
 destroyed */
            if (data)
              free_mydata(data);
            break;
        }

        case PGEVT_RESULTCREATE:
        {
            PGEventResultCreate *e = (PGEventResultCreate *)evt
Info;
```

```
            mydata *conn_data = PQinstanceData(e->conn, myEvent
Proc);
            mydata *res_data = dup_mydata(conn_data);

            /* associate app specific data with result (copy it
 from conn) */
            PQsetResultInstanceData(e->result, myEventProc, res_
data);
            break;
        }

        case PGEVT_RESULTCOPY:
        {
            PGEventResultCopy *e = (PGEventResultCopy *)evtInfo;
            mydata *src_data = PQresultInstanceData(e->src, myEvent
Proc);
            mydata *dest_data = dup_mydata(src_data);

            /* associate app specific data with result (copy it
 from a result) */
            PQsetResultInstanceData(e->dest, myEventProc, dest_
data);
            break;
        }

        case PGEVT_RESULTDESTROY:
        {
            PGEventResultDestroy *e = (PGEventResultDestroy *)evt
Info;
            mydata *data = PQresultInstanceData(e->result, myEvent
Proc);

            /* free instance data because the result is being
 destroyed */
            if (data)
              free_mydata(data);
            break;
        }

        /* unknown event ID, just return TRUE. */
        default:
            break;
    }

    return TRUE; /* event processing succeeded */
}
```

31.14. 环境变量

以下环境变量可用于选择连接参数的默认值，供 `PQconnectdb`、`PQsetdbLogin` 和 `PQsetdb` 在调用代码未直接指定参数值时使用。例如，这样可以避免在简单的客户端应用程序中硬编码数据库连接信息。

- `PGHOST`的行为与host连接参数相同。
- `PGHOSTADDR`的行为与hostaddr连接参数相同。这可以替代或者与`PGHOST`一起设置，以避免DNS查找开销。

- PGPORT的行为与port连接参数相同。
- PGDATABASE的行为与dbname连接参数相同。
- PGUSER的行为与user连接参数相同。
- PGPASSWORD的行为与password连接参数相同。出于安全原因，不建议使用此环境变量，因为一些操作系统允许非root用户通过ps查看进程环境变量；而应考虑使用~/.pgpass文件（参见第 31.15 节）。
- PGPASSFILE指定用于查找的密码文件名。如果未设置，默认为~/.pgpass（参见第 31.15 节）。
- PGSERVICE的行为与service连接参数相同。
- PGSERVICEFILE指定每个用户的连接服务文件的名称。如果未设置，默认为~/.pg_service.conf（参见第 31.16 节）。
- PGOPTIONS的行为与options连接参数相同。
- PGAPPNAME的行为与application_name连接参数相同。
- PGSSLMODE的行为与sslmode连接参数相同。
- PGREQUIRESSL的行为与requiressl连接参数相同。这个环境变量已被弃用，推荐使用PGSSLMODE变量，但同时设置两个变量会抑制PGSSLMODE的效果。
- PGSSLCOMPRESSION的行为与sslcompression连接参数相同。
- PGSSLCERT的行为与sslcert连接参数相同。
- PGSSLKEY的行为与sslkey连接参数相同。
- PGSSLROOTCERT的行为与sslrootcert连接参数相同。
- PGSSLCRL的行为与sslcrl连接参数相同。
- PGREQUIREPEER的行为与requirepeer连接参数相同。
- PGKRBSRVNAME的行为与krbsrvname连接参数相同。
- PGGSSLIB的行为与gsslib连接参数相同。
- PGCONNECT_TIMEOUT的行为与connect_timeout连接参数相同。
- PGCLIENTENCODING的行为与client_encoding连接参数相同。

以下环境变量可用于指定每个 PostgreSQL 会话的默认行为。（也可参见ALTER ROLE和ALTER DATABASE命令，了解按用户或按数据库设置默认行为的方法。）

- PGDATESTYLE设置日期/时间表示的默认风格（等同于SET datestyle TO ...）。
- PGTZ设置默认的时区（等同于SET timezone TO ...）。
- PGGEQO为遗传查询优化器设置默认模式（等同于SET geqo TO ...）。

有关这些环境变量的正确取值，请参见SQL命令SET。

下面的环境变量决定libpq的内部行为，它们会覆盖编译在程序中的默认值。

- PGSYSCONFDIR设置包含pg_service.conf文件以及未来版本中可能出现的其他系统范围配置文件的目录。
- PGLOCALEDIR设置包含用于消息国际化的locale文件的目录。

31.15. 密码文件

用户主目录中的.pgpass文件或PGPASSFILE引用的文件可保存密码，
供连接需要密码且尚未通过其他方式指定密码时使用。在 Microsoft Windows 上，文件名为 %APPDATA%\postgresql\pgpass.conf（其中%APPDATA%
指用户配置文件中的应用数据子目录）。

该文件中的行应采用以下格式：

hostname:*port*:*database*:*username*:*password*

（可以复制上面这一行，并在行首加上#，在文件中加入提示注释。）
前四个字段中的每一个都可以是字面值，或者是*，后者可以匹配任何内容。
将使用与当前连接参数匹配的第一行中的密码字段。（因此，使用通配符时，
应将更具体的条目放在前面。）如果条目需要包含:或\，请使用\转义该字符。
主机名localhost同时匹配来自本地机器的 TCP 连接（主机名localhost）和 Unix 域套接字连接（pghost为空或为默认套接字目录）。在备库中，数据库字段为replication时，
匹配连接到主库的流复制连接。除此之外，database字段的用途有限，
因为同一用户在同一数据库集簇的所有数据库中使用相同的密码。

在 Unix 系统上，.pgpass文件上的权限必须不允许所有人或组内访问，可以用chmod 0600 ~/.pgpass这样的命令实现。如果权限没有这么严格，该文件将被忽略。在微软 Windows 上，该文件被假定存储在一个安全的目录中，因此不会进行特别的权限检查。

31.16. 连接服务文件

连接服务文件允许 libpq 连接参数与一个单一服务名称关联。之后在一个 libpq 连接中就可以指定那个服务名称，与其相关的设置将被使用。这允许在不重新编译 libpq 应用的前提下修改连接参数。服务名称也可以使用PGSERVICE环境变量来指定。

连接服务文件可以是位于~/.pg_service.conf或由环境变量PGSERVICEFILE
指定的位置处的每用户服务文件，也可以是位于`pg_config --sysconfdir`/pg_service.conf或由环境变量 PGSYSCONFDIR指定的目录中的系统范围文件。如果在用户文件和系统文件中存在同名的服务定义，则用户文件优先。

该文件使用一种“INI 文件”格式，其中小节名是服务名并且参数是连接参数。列表见第 31.1.2 节。例如：

```
# comment
[mydb]
host=somehost
port=5433
user=admin
```

在share/pg_service.conf.sample中提供了一个示例文件。

31.17. 连接参数的 LDAP 查找

如果libpq已经在编译时打开了 LDAP 支持（configure的选项--with-ldap），就可以通过 LDAP 从一个中央服务器检索host或dbname之类的连接参数。这样做的好处是如果一个数据库的连接参数改变，不需要在所有的客户端机器上更新连接信息。

LDAP 连接参数查询使用连接服务文件pg_service.conf（参见第 31.16 节）。在pg_service.conf的配置段中，以ldap://开头的行会被识别为 LDAP URL，并执行 LDAP 查询。结果必须是一个keyword = value键值对列表，用于设置连接选项。URL 必须符合 RFC 1959，格式如下：

ldap://[*hostname*[:*port*]]/*search_base*?*attribute*?*search_scope*?*filter*

其中，*hostname*默认为localhost而*port*默认为 389。

LDAP 查找成功后就会停止处理 pg_service.conf；如果无法联系 LDAP 服务器，则会继续处理。这使后续指向其他 LDAP 服务器的 LDAP URL 行、常规的 keyword = value 对或默认连接选项能够作为后备。如果希望在这种情况下得到错误消息，可以在 LDAP URL 后添加一个语法不正确的行。

例如，使用以下 LDIF 文件创建的 LDAP 条目：

```
version:1
dn:cn=mydatabase,dc=mycompany,dc=com
changetype:add
objectclass:top
objectclass:device
cn:mydatabase
description:host=dbserver.mycompany.com
description:port=5439
description:dbname=mydb
description:user=mydb_user
description:sslmode=require
```

可以通过以下 LDAP URL 查询：

```
ldap://ldap.mycompany.com/dc=mycompany,dc=com?description?one?(cn=
mydatabase)
```

你也可以将常规的服务文件条目和 LDAP 查找混合。pg_service.conf中一节的完整示例：

```
# 仅 host 和 port 存储在 LDAP 中；显式指定 dbname 和 user
[customerdb]
dbname=customer
user=appuser
ldap://ldap.acme.com/cn=dbserver,cn=hosts?pgconnectinfo?base?(
objectclass=*)
```

31.18. SSL 支持

PostgreSQL 原生支持使用 SSL 连接来加密客户端与服务器之间的通信，以提高安全性。有关服务器端 SSL 功能的详细信息，请参见第 17.9 节。

libpq读取系统范围的OpenSSL配置文件。默认情况下，这个文件被命名为openssl.cnf并且位于openssl version -d所报告的目录中。可以通过设置环境变量OPENSSL_CONF把这个默认值覆盖为想要的配置文件的名称。

31.18.1. 服务器证书的客户端验证

默认情况下，PostgreSQL不会对服务器证书执行任何验证。
这意味着可以在客户端不知情的情况下伪造服务器身份，例如修改 DNS 记录或接管服务器的 IP 地址。为了防止伪造，必须使用SSL证书验证。

如果参数sslmode设为verify-ca，libpq 会沿证书链检查到一个受信任的证书机构（CA），以验证服务器是否可信。如果sslmode设为verify-full，libpq 还会验证服务器主机名是否与其证书匹配。如果无法验证服务器证书，SSL 连接将失败。在大多数对安全敏感的环境中，建议使用verify-full。

在 verify-full 模式下，会将主机名与证书的 cn（通用名称）属性匹配。如果 cn 属性以星号（*）开头，该星号会被视为通配符，匹配除点（.）以外的所有字符。这意味着该证书不会匹配子域。如果使用 IP 地址而不是主机名建立连接，则会匹配该 IP 地址（不执行任何 DNS 查询）。

要允许服务器证书验证，必须将一个或多个受信任的CA的证书放置在用户主目录下的~/.postgresql/root.crt文件中（在 Microsoft Windows 上该文件名为%APPDATA%\postgresql\root.crt）。

如果文件~/.postgresql/root.crl存在（微软 Windows 上的%APPDATA%\postgresql\root.crl），证书撤销列表（CRL）项也会被检查。

可以通过设置连接参数 sslrootcert 和 sslcrl，或环境变量 PGSSLROOTCERT 和 PGSSLCRL，更改根证书文件与 CRL 的位置。

> **注意**
>
> 为与 PostgreSQL 的早期版本向后兼容，如果存在根 CA 文件，sslmode=require 的行为将与 verify-ca 相同，即根据 CA 验证服务器证书。不建议依赖这种行为；需要证书验证的应用程序应始终使用 verify-ca 或 verify-full。

31.18.2. 客户端证书

如果服务器请求一个受信任的客户端证书，libpq将发送存储在用户主目录下的文件~/.postgresql/postgresql.crt中的证书。该证书必须由服务器信任的证书机构（CA）之一签发。匹配的私钥文件~/.postgresql/postgresql.key也必须存在。私钥文件的权限必须不允许任何对世界或组的访问；可以通过命令chmod 0600 ~/.postgresql/postgresql.key实现。在 Microsoft Windows 上，这些文件的名称分别为%APPDATA%\postgresql\postgresql.crt和%APPDATA%\postgresql\postgresql.key，并且不进行特别的权限检查，因为该目录被假定为安全的。证书和密钥文件的位置可以通过连接参数sslcert和sslkey或环境变量PGSSLCERT和PGSSLKEY来覆盖。

在某些情况下，客户端证书可能由一个“中间”证书机构签发，而不是由服务器直接信任的机构签发。要使用这样的证书，把签发机构的证书追加到postgresql.crt文件中，然后是其父机构的证书，依此类推直到一个服务器信任的“根”机构。在postgresql.crt包含多个证书的所有情况中，都应包含根证书。

注意，root.crt列出的是被认为可以信任其为服务器证书签名的顶层 CA。原则上它不必列出为客户端证书签名的 CA，尽管在大多数情况下该 CA 也会被信任用于服务器证书。

31.18.3. 不同模式中提供的保护

sslmode 参数的不同值提供不同级别的保护。SSL 可以防范三类攻击：

窃听

如果一个第三方能够检查客户端和服务器之间的网络流量，它能读取连接信息（包括用户名和密码）以及被传递的数据。SSL使用加密来阻止这种攻击。

中间人（MITM）

如果第三方能修改客户端与服务器之间传输的数据，就可以冒充服务器，进而查看和修改数据，即使数据已经加密。随后，第三方可以将连接信息和数据转发给原来的服务器，使攻击无法被察觉。常见的手段包括 DNS 污染和地址劫持，从而将客户端引向预期之外的服务器。还有其他几种攻击手段可以达到同样的目的。SSL 使用证书验证，让客户端认证服务器身份，以防范这种攻击。

模仿

如果第三方能冒充获授权的客户端，就能直接访问其无权访问的数据。这通常可能由不安全的密码管理导致。SSL 使用客户端证书，确保只有持有有效证书的客户端才能访问服务器，以防范这种攻击。

要确保连接安全，必须在建立连接之前，在客户端和服务器两端配置 SSL。如果仅在服务器上配置，客户端可能在得知服务器要求高安全性之前就已发送敏感信息（例如密码）。在 libpq 中，可以将 `sslmode` 参数设为 `verify-full` 或 `verify-ca`，并向系统提供用于验证的根证书，以确保连接安全。这类似于使用 `https` URL 进行加密的网页浏览。

服务器通过身份认证后，客户端便可以传送敏感数据。这意味着，在此之前，
客户端无需知道是否会使用证书进行认证，因此可以安全地仅在服务器配置中指定这一点。

所有 SSL 选项都会产生加密和密钥交换的开销，因此必须在性能与安全性之间作出权衡。表 31.1说明了不同 `sslmode` 值所能防范的风险，以及它们所表达的对安全性和开销的取舍。

表 31.1. SSL 模式描述

sslmode	**窃听保护**	**MITM protection**	**Statement**
`disable`	No	No	我不关心安全性，并且我不想为加密增加负荷。
`allow`	Maybe	No	我不关心安全性，但如果服务器坚持，我将承担加密带来的负荷。
`prefer`	Maybe	No	我不关心安全性，但如果服务器支持，我希望承担加密带来的负荷。
`require`	Yes	No	我想要对数据加密，并且我接受因此带来的负荷。我信任该网络会保证我总是连接到想要连接的服务器。
`verify-ca`	Yes	`Depends on CA-`policy	我想要对数据加密，并且我接受因此带来的负荷。我想要确保我连接到的是我信任的服务器。
`verify-full`	Yes	Yes	我想要对数据加密，并且我接受因此带来的负荷。我想要确保我连接到的是我信任的服务器，并且就是我指定的那一个。

`verify-ca`和`verify-full`之间的区别取决于根CA的策略。如果使用了一个公共CA，`verify-ca`允许连接到那些可能已经被其他人注册到该CA的服务器。在这种情况下，总是应该使用`verify-full`。如果使用了一个本地CA或者甚至是一个自签名的证书，使用`verify-ca`常常就可以提供足够的保护。

`sslmode` 的默认值是 `prefer`。如表所示，从安全角度看，这一设置没有意义；它只会在可能时带来性能开销。将其作为默认值仅出于向后兼容的考虑，
不建议在有安全要求的部署中使用。

31.18.4. SSL 客户端文件使用

表 31.2总结了与客户端 SSL 设置相关的文件。

表 31.2. libpq/客户端 SSL 文件用法

文件	**内容**	**作用**
`~/.postgresql/postgresql.crt`	客户端证书	被服务器请求

文件	内容	作用
~/.postgresql/postgresql.key	客户端私钥	证明客户端证书由其所有者发送；并不表明证书所有者可信
~/.postgresql/root.crt	受信任的证书机构	检查服务器证书是否由受信任的证书机构签发
~/.postgresql/root.crl	被证书机构撤销的证书	服务器证书不能出现在这个列表中

31.18.5. SSL 库初始化

如果你的应用程序初始化libssl和/或libcrypto库，并且libpq 构建时带有SSL支持，你应该调用PQinitOpenSSL告诉libpq libssl和/或libcrypto库已被你的应用程序初始化，以便 libpq不会再初始化这些库。SSL API 的详细信息见http://h71000.www7.hp.com/doc/83final/ba554_90007/ch04.html。

PQinitOpenSSL

允许应用程序选择要初始化的安全库。

```
void PQinitOpenSSL(int do_ssl, int do_crypto);
```

当*do_ssl*是非零时，libpq将在第一次打开数据库连接前初始化OpenSSL库。当*do_crypto*是非零时，libcrypto库将被初始化。默认情况下（如果没有调用PQinitOpenSSL），两个库都会被初始化。当 SSL 支持没有被编译时，这个函数也存在但是什么也不做。

如果你的应用使用并且初始化OpenSSL或者它的底层libcrypto库，你必须在第一次打开数据库连接前调用这个函数，并把相应参数设为零。同时要确保在打开一个数据库连接前已经完成了初始化。

PQinitSSL

允许应用程序选择要初始化的安全库。

```
void PQinitSSL(int do_ssl);
```

这个函数等效于PQinitOpenSSL(do_ssl, do_ssl)。这对于要么初始化OpenSSL以及libcrypto要么都不初始化的应用足够用了。

PQinitSSL从PostgreSQL 8.0 就存在了，而PQinitOpenSSL直到PostgreSQL 8.4 才被加入，因此PQinitSSL可能对那些需要与旧版本libpq一起工作的应用来说更合适。

31.19. 在线程化程序中的行为

libpq 默认是可重入且线程安全的。编译应用程序代码时，可能需要使用特殊的编译器命令行选项。有关如何构建支持线程的应用程序，请参阅系统文档，或查看 src/Makefile.global 中的 PTHREAD_CFLAGS 和 PTHREAD_LIBS。以下函数可用于查询 libpq 的线程安全状态：

PQisthreadsafe

返回libpq库的线程安全状态。

```
int PQisthreadsafe();
```

如果libpq是线程安全的则返回 1，否则返回 0。

线程使用的一项限制是：两个线程不能同时操作同一个 PGconn 对象。尤其不能通过同一个连接对象从不同线程并发发出命令。（如果需要并发执行命令，请使用多个连接。）

PGresult对象在创建后通常是只读的，因此可以在线程之间自由传递。不过，如果你使用第 31.11 节或第 31.13 节中描述的任何会修改PGresult的函数，则需要自行避免对同一个PGresult执行并发操作。

已弃用的函数 PQrequestCancel 和 PQoidStatus 不是线程安全的，不应在多线程程序中使用。PQrequestCancel 可以用 PQcancel 替代；PQoidStatus 可以用 PQoidValue 替代。

如果你在应用程序中使用 Kerberos（除了libpq内部之外），则需要在 Kerberos 调用周围加锁，因为 Kerberos 函数不是线程安全的。可参考libpq源代码中的PQregisterThreadLock函数，它提供了一种在libpq与应用程序之间协作加锁的方法。

如果你在多线程应用程序中遇到问题，请运行 src/tools/thread 目录中的程序，检查你的平台是否存在线程不安全的函数。configure 会运行这个程序，但对于二进制发行版，你自己的库可能与编译这些二进制文件时所用的库不一致。

31.20. 编译 libpq 程序

要构建（即编译和链接）一个使用 libpq 的程序，需要完成以下所有步骤：

- 包含 libpq-fe.h 头文件：

  ```
  #include <libpq-fe.h>
  ```

  如果没有这样做，编译器通常会给出类似以下内容的错误消息：

  ```
  foo.c: In function `main':
  foo.c:34: `PGconn' undeclared (first use in this function)
  foo.c:35: `PGresult' undeclared (first use in this function)
  foo.c:54: `CONNECTION_BAD' undeclared (first use in this
   function)
  foo.c:68: `PGRES_COMMAND_OK' undeclared (first use in this
   function)
  foo.c:95: `PGRES_TUPLES_OK' undeclared (first use in this
   function)
  ```

- 向编译器提供 -I*directory* 选项，指定 PostgreSQL 头文件的安装目录。（某些情况下，编译器默认会搜索该目录，此时可以省略这个选项。）例如，编译命令行可以是：

  ```
  cc -c -I/usr/local/pgsql/include testprog.c
  ```

  如果使用 makefile，请将该选项添加到 CPPFLAGS 变量中：

  ```
  CPPFLAGS += -I/usr/local/pgsql/include
  ```

  如果你的程序可能由其他用户编译，那么你不应该像那样硬编码目录位置。
  你可以运行工具pg_config在本地系统上找出头文件在哪里：

  ```
  $ pg_config --includedir
  /usr/local/include
  ```

  未向编译器指定正确的选项将导致类似以下内容的错误消息：

  ```
  testlibpq.c:8:22: libpq-fe.h: No such file or directory
  ```

- 链接最终程序时，指定 -lpq 选项以链接 libpq 库，同时指定 -L*directory* 选项，告知编译器 libpq 库所在的目录。（同样，编译器默认会搜索某些目录。）为获得最大的可移植性，请将 -L 选项放在 -lpq 选项之前。例如：

  ```
  cc -o testprog testprog1.o testprog2.o -L/usr/local/pgsql/lib -
  lpq
  ```

你也可以使用pg_config找出库目录：

```
$ pg_config --libdir
/usr/local/pgsql/lib
```

指出这一部分问题的错误消息可能看起来像：

```
testlibpq.o: In function `main':
testlibpq.o(.text+0x60): undefined reference to `PQsetdbLogin'
testlibpq.o(.text+0x71): undefined reference to `PQstatus'
testlibpq.o(.text+0xa4): undefined reference to `PQerrorMessage'
```

这意味着你忘记了 -lpq。

```
/usr/bin/ld: cannot find -lpq
```

这意味着你忘记了-L选项或者没有指定正确的目录。

31.21. 示例程序

这些示例和其他示例可在源代码发行包的src/test/examples目录中找到。

例 31.1. libpq 示例程序 1

```
/*
 * testlibpq.c
 *
 *      Test the C version of libpq, the PostgreSQL frontend
 library.
 */
#include <stdio.h>
#include <stdlib.h>
#include <libpq-fe.h>

static void
exit_nicely(PGconn *conn)
{
    PQfinish(conn);
    exit(1);
}

int
main(int argc, char **argv)
{
    const char *conninfo;
    PGconn     *conn;
    PGresult   *res;
    int         nFields;
    int         i,
                j;

    /*
     * If the user supplies a parameter on the command line, use it
 as the
     * conninfo string; otherwise default to setting dbname=
postgres and using
     * environment variables or defaults for all other connection
 parameters.
```

```
     */
    if (argc > 1)
        conninfo = argv[1];
    else
        conninfo = "dbname = postgres";

    /* Make a connection to the database */
    conn = PQconnectdb(conninfo);

    /* Check to see that the backend connection was successfully
 made */
    if (PQstatus(conn) != CONNECTION_OK)
    {
        fprintf(stderr, "Connection to database failed: %s",
                PQerrorMessage(conn));
        exit_nicely(conn);
    }

    /*
     * Our test case here involves using a cursor, for which we
 must be inside
     * a transaction block.  We could do the whole thing with a
 single
     * PQexec() of "select * from pg_database", but that's too
 trivial to make
     * a good example.
     */

    /* Start a transaction block */
    res = PQexec(conn, "BEGIN");
    if (PQresultStatus(res) != PGRES_COMMAND_OK)
    {
        fprintf(stderr, "BEGIN command failed: %s", PQerrorMessage(
conn));
        PQclear(res);
        exit_nicely(conn);
    }

    /*
     * Should PQclear PGresult whenever it is no longer needed to
 avoid memory
     * leaks
     */
    PQclear(res);

    /*
     * Fetch rows from pg_database, the system catalog of databases
     */
    res = PQexec(conn, "DECLARE myportal CURSOR FOR select * from
 pg_database");
    if (PQresultStatus(res) != PGRES_COMMAND_OK)
    {
        fprintf(stderr, "DECLARE CURSOR failed: %s", PQerror
Message(conn));
        PQclear(res);
        exit_nicely(conn);
    }
    PQclear(res);
```

```
    res = PQexec(conn, "FETCH ALL in myportal");
    if (PQresultStatus(res) != PGRES_TUPLES_OK)
    {
        fprintf(stderr, "FETCH ALL failed: %s", PQerrorMessage(
conn));
        PQclear(res);
        exit_nicely(conn);
    }

    /* first, print out the attribute names */
    nFields = PQnfields(res);
    for (i = 0; i < nFields; i++)
        printf("%-15s", PQfname(res, i));
    printf("\n\n");

    /* next, print out the rows */
    for (i = 0; i < PQntuples(res); i++)
    {
        for (j = 0; j < nFields; j++)
            printf("%-15s", PQgetvalue(res, i, j));
        printf("\n");
    }

    PQclear(res);

    /* close the portal ... we don't bother to check for errors ...
 */
    res = PQexec(conn, "CLOSE myportal");
    PQclear(res);

    /* end the transaction */
    res = PQexec(conn, "END");
    PQclear(res);

    /* close the connection to the database and cleanup */
    PQfinish(conn);

    return 0;
}
```

例 31.2. libpq示例程序 2

```
/*
 * testlibpq2.c
 *      Test of the asynchronous notification interface
 *
 * Start this program, then from psql in another window do
 *   NOTIFY TBL2;
 * Repeat four times to get this program to exit.
 *
 * Or, if you want to get fancy, try this:
 * populate a database with the following commands
 * (provided in src/test/examples/testlibpq2.sql):
 *
 *   CREATE TABLE TBL1 (i int4);
 *
 *   CREATE TABLE TBL2 (i int4);
```

```
 *
 *   CREATE RULE r1 AS ON INSERT TO TBL1 DO
 *     (INSERT INTO TBL2 VALUES (new.i); NOTIFY TBL2);
 *
 * and do this four times:
 *
 *   INSERT INTO TBL1 VALUES (10);
 */
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <errno.h>
#include <sys/time.h>
#include <libpq-fe.h>

static void
exit_nicely(PGconn *conn)
{
    PQfinish(conn);
    exit(1);
}

int
main(int argc, char **argv)
{
    const char *conninfo;
    PGconn     *conn;
    PGresult   *res;
    PGnotify   *notify;
    int         nnotifies;

    /*
     * If the user supplies a parameter on the command line, use it
 as the
     * conninfo string; otherwise default to setting dbname=
postgres and using
     * environment variables or defaults for all other connection
 parameters.
     */
    if (argc > 1)
        conninfo = argv[1];
    else
        conninfo = "dbname = postgres";

    /* Make a connection to the database */
    conn = PQconnectdb(conninfo);

    /* Check to see that the backend connection was successfully
 made */
    if (PQstatus(conn) != CONNECTION_OK)
    {
        fprintf(stderr, "Connection to database failed: %s",
                PQerrorMessage(conn));
        exit_nicely(conn);
    }

    /*
```

```
     * Issue LISTEN command to enable notifications from the rule's
 NOTIFY.
     */
    res = PQexec(conn, "LISTEN TBL2");
    if (PQresultStatus(res) != PGRES_COMMAND_OK)
    {
        fprintf(stderr, "LISTEN command failed: %s", PQerror
Message(conn));
        PQclear(res);
        exit_nicely(conn);
    }

    /*
     * should PQclear PGresult whenever it is no longer needed to
 avoid memory
     * leaks
     */
    PQclear(res);

    /* Quit after four notifies are received. */
    nnotifies = 0;
    while (nnotifies < 4)
    {
        /*
         * Sleep until something happens on the connection.  We use
 select(2)
         * to wait for input, but you could also use poll() or
 similar
         * facilities.
         */
        int         sock;
        fd_set      input_mask;

        sock = PQsocket(conn);

        if (sock < 0)
            break;              /* shouldn't happen */

        FD_ZERO(&input_mask);
        FD_SET(sock, &input_mask);

        if (select(sock + 1, &input_mask, NULL, NULL, NULL) < 0)
        {
            fprintf(stderr, "select() failed: %s\n", strerror(
errno));
            exit_nicely(conn);
        }

        /* Now check for input */
        PQconsumeInput(conn);
        while ((notify = PQnotifies(conn)) != NULL)
        {
            fprintf(stderr,
                    "ASYNC NOTIFY of '%s' received from backend PID
 %d\n",
                    notify->relname, notify->be_pid);
            PQfreemem(notify);
            nnotifies++;
```

```
        }
    }

    fprintf(stderr, "Done.\n");

    /* close the connection to the database and cleanup */
    PQfinish(conn);

    return 0;
}
```

例 31.3. libpq示例程序 3

```
/*
 * testlibpq3.c
 *      Test out-of-line parameters and binary I/O.
 *
 * Before running this, populate a database with the following
 commands
 * (provided in src/test/examples/testlibpq3.sql):
 *
 * CREATE TABLE test1 (i int4, t text, b bytea);
 *
 * INSERT INTO test1 values (1, 'joe''s place', '\\000\\001\\002\
\003\\004');
 * INSERT INTO test1 values (2, 'ho there', '\\004\\003\\002\\001\
\000');
 *
 * The expected output is:
 *
 * tuple 0: got
 *  i = (4 bytes) 1
 *  t = (11 bytes) 'joe's place'
 *  b = (5 bytes) \000\001\002\003\004
 *
 * tuple 0: got
 *  i = (4 bytes) 2
 *  t = (8 bytes) 'ho there'
 *  b = (5 bytes) \004\003\002\001\000
 */
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/types.h>
#include <libpq-fe.h>

/* for ntohl/htonl */
#include <netinet/in.h>
#include <arpa/inet.h>


static void
exit_nicely(PGconn *conn)
{
    PQfinish(conn);
    exit(1);
}
```

```
/*
 * This function prints a query result that is a binary-format
 fetch from
 * a table defined as in the comment above.  We split it out
 because the
 * main() function uses it twice.
 */
static void
show_binary_results(PGresult *res)
{
    int         i,
                j;
    int         i_fnum,
                t_fnum,
                b_fnum;

    /* Use PQfnumber to avoid assumptions about field order in
 result */
    i_fnum = PQfnumber(res, "i");
    t_fnum = PQfnumber(res, "t");
    b_fnum = PQfnumber(res, "b");

    for (i = 0; i < PQntuples(res); i++)
    {
        char       *iptr;
        char       *tptr;
        char       *bptr;
        int         blen;
        int         ival;

        /* Get the field values (we ignore possibility they are
 null!) */
        iptr = PQgetvalue(res, i, i_fnum);
        tptr = PQgetvalue(res, i, t_fnum);
        bptr = PQgetvalue(res, i, b_fnum);

        /*
         * The binary representation of INT4 is in network byte
 order, which
         * we'd better coerce to the local byte order.
         */
        ival = ntohl(*((uint32_t *) iptr));

        /*
         * The binary representation of TEXT is, well, text, and
 since libpq
         * was nice enough to append a zero byte to it, it'll work
 just fine
         * as a C string.
         *
         * The binary representation of BYTEA is a bunch of bytes,
 which could
         * include embedded nulls so we have to pay attention to
 field length.
         */
        blen = PQgetlength(res, i, b_fnum);

        printf("tuple %d: got\n", i);
```

```
        printf(" i = (%d bytes) %d\n",
               PQgetlength(res, i, i_fnum), ival);
        printf(" t = (%d bytes) '%s'\n",
               PQgetlength(res, i, t_fnum), tptr);
        printf(" b = (%d bytes) ", blen);
        for (j = 0; j < blen; j++)
            printf("\\%03o", bptr[j]);
        printf("\n\n");
    }
}

int
main(int argc, char **argv)
{
    const char *conninfo;
    PGconn     *conn;
    PGresult   *res;
    const char *paramValues[1];
    int         paramLengths[1];
    int         paramFormats[1];
    uint32_t    binaryIntVal;

    /*
     * If the user supplies a parameter on the command line, use it
 as the
     * conninfo string; otherwise default to setting dbname=
postgres and using
     * environment variables or defaults for all other connection
 parameters.
     */
    if (argc > 1)
        conninfo = argv[1];
    else
        conninfo = "dbname = postgres";

    /* Make a connection to the database */
    conn = PQconnectdb(conninfo);

    /* Check to see that the backend connection was successfully
 made */
    if (PQstatus(conn) != CONNECTION_OK)
    {
        fprintf(stderr, "Connection to database failed: %s",
                PQerrorMessage(conn));
        exit_nicely(conn);
    }

    /*
     * The point of this program is to illustrate use of PQexec
Params() with
     * out-of-line parameters, as well as binary transmission of
 data.
     *
     * This first example transmits the parameters as text, but
 receives the
     * results in binary format.  By using out-of-line parameters
 we can
```

```
    * avoid a lot of tedious mucking about with quoting and
escaping, even
    * though the data is text.  Notice how we don't have to do
anything
    * special with the quote mark in the parameter value.
    */

   /* Here is our out-of-line parameter value */
   paramValues[0] = "joe's place";

   res = PQexecParams(conn,
                      "SELECT * FROM test1 WHERE t = $1",
                      1,       /* one param */
                      NULL,    /* let the backend deduce param
type */
                      paramValues,
                      NULL,    /* don't need param lengths since
text */
                      NULL,    /* default to all text params */
                      1);      /* ask for binary results */

   if (PQresultStatus(res) != PGRES_TUPLES_OK)
   {
       fprintf(stderr, "SELECT failed: %s", PQerrorMessage(conn));
       PQclear(res);
       exit_nicely(conn);
   }

   show_binary_results(res);

   PQclear(res);

   /*
    * In this second example we transmit an integer parameter in
binary
    * form, and again retrieve the results in binary form.
    *
    * Although we tell PQexecParams we are letting the backend
deduce
    * parameter type, we really force the decision by casting the
parameter
    * symbol in the query text.  This is a good safety measure
when sending
    * binary parameters.
    */

   /* Convert integer value "2" to network byte order */
   binaryIntVal = htonl((uint32_t) 2);

   /* Set up parameter arrays for PQexecParams */
   paramValues[0] = (char *) &binaryIntVal;
   paramLengths[0] = sizeof(binaryIntVal);
   paramFormats[0] = 1;        /* binary */

   res = PQexecParams(conn,
                      "SELECT * FROM test1 WHERE i = $1::int4",
                      1,       /* one param */
```

```
                        NULL,    /* let the backend deduce param
 type */
                        paramValues,
                        paramLengths,
                        paramFormats,
                        1);      /* ask for binary results */

    if (PQresultStatus(res) != PGRES_TUPLES_OK)
    {
        fprintf(stderr, "SELECT failed: %s", PQerrorMessage(conn));
        PQclear(res);
        exit_nicely(conn);
    }

    show_binary_results(res);

    PQclear(res);

    /* close the connection to the database and cleanup */
    PQfinish(conn);

    return 0;
}
```

第 32 章 大对象

PostgreSQL提供一种大对象机制，
允许以流式方式访问存储在专用大对象结构中的用户数据。
在处理大到无法方便地整体操作的数据值时，这种流式访问非常有用。

本章介绍PostgreSQL大对象数据的实现，以及相应的编程接口和查询语言接口。
本章示例使用libpq C 库，但大多数PostgreSQL原生编程接口都支持等效的功能。其他接口也可能在内部使用大对象接口来为大值提供通用支持，这里不作说明。

32.1. 简介

所有大对象都存放在一个名为`pg_largeobject`的系统表中。
PostgreSQL还支持一种名为“TOAST”的存储系统，
它会自动将大于单个数据库页的值存储到每个表各自的二级存储区域中。
这使得大对象机制在一定程度上已经过时。大对象机制仍然保留的一个优势是，
它允许的值大小可达 2 GB，而经过TOAST处理的字段最大只能到 1 GB。此外，大对象可以通过一种读写 API 进行随机修改，这比使用TOAST执行此类操作更高效。

32.2. 实现特性

大对象实现会把大对象拆分成若干“块”，并将这些块存储在数据库行中。一个 B-树索引保证在执行随机访问读写时，能够快速找到正确的块号。

为与以前的版本兼容，请参见lo_compat_privileges。

自PostgreSQL 9.0 起，大对象具有拥有者和一组访问权限，可用GRANT和REVOKE管理。读取大对象需要`SELECT`权限，写入或截断大对象需要`UPDATE`权限。
只有大对象的拥有者（或者数据库超级用户）才能删除（unlink）、
注释或更改大对象的拥有者。

32.3. 客户端接口

PostgreSQL客户端接口库为访问大对象提供了支持。
使用这些函数对大对象进行的所有操作都必须发生在一个 SQL 事务块内。PostgreSQL的大对象接口是仿照Unix文件系统接口设计的，提供了与`open`、`read`、`write`、`lseek`
等相对应的操作。

在libpq中使用大对象接口的客户端应用应包含头文件`libpq/libpq-fs.h`
并与libpq库链接。

32.3.1. 创建一个大对象

函数

```
Oid lo_creat(PGconn *conn, int mode);
```

创建一个新的大对象。返回值是分配给新大对象的 OID，失败时为`InvalidOid`（零）。自PostgreSQL 8.1 起，*mode*未被使用且被忽略；不过，为了与更早版本的向后兼容，最好将其设置为`INV_READ`、`INV_WRITE`或`INV_READ | INV_WRITE`。（这些符号常量定义在头文件`libpq/libpq-fs.h`中。）

例如：

```
inv_oid = lo_creat(conn, INV_READ|INV_WRITE);
```

函数

```
Oid lo_create(PGconn *conn, Oid lobjId);
```

也会创建一个新的大对象。要分配的 OID 可以由*lobjId*指定；如果指定了它，而该 OID 已经被某个大对象使用，则会失败。如果*lobjId*是InvalidOid（零），则lo_create会分配一个未使用的 OID（其行为与lo_creat相同）。返回值是分配给新大对象的OID，失败时为InvalidOid（零）。

lo_create是PostgreSQL 8.1 新增的；如果将该函数用于更早版本的服务器，它会失败并返回InvalidOid。

例如：

```
inv_oid = lo_create(conn, desired_oid);
```

32.3.2. 导入一个大对象

要把一个操作系统文件导入为大对象，调用

```
Oid lo_import(PGconn *conn, const char *filename);
```

*filename*指定要作为大对象导入的操作系统文件名。返回值是分配给新大对象的 OID，失败时为InvalidOid（零）。注意，该文件是由客户端接口库读取的，
而不是由服务器读取的；因此它必须存在于客户端文件系统中，并且对客户端应用可读。

函数

```
Oid lo_import_with_oid(PGconn *conn, const char *filename, Oid lobj
Id);
```

也会导入一个新的大对象。要分配的 OID 可以由*lobjId*指定；如果指定了它，而该 OID 已经被某个大对象使用，则会失败。如果*lobjId*是InvalidOid（零），则lo_import_with_oid会分配一个未使用的 OID（其行为与lo_import相同）。返回值是分配给新大对象的 OID，失败时为InvalidOid（零）。

lo_import_with_oid是PostgreSQL 8.4 新增的，它在内部使用了 8.1 新增的lo_create；如果将该函数用于 8.0 或更早版本的服务器，它会失败并返回InvalidOid。

32.3.3. 导出一个大对象

要把一个大对象导出到操作系统文件中，调用

```
int lo_export(PGconn *conn, Oid lobjId, const char *filename);
```

*lobjId*参数指定要导出的大对象的 OID，*filename*参数指定操作系统文件名。注意，该文件是由客户端接口库写入的，而不是由服务器写入的。成功时返回 1，失败时返回 -1。

32.3.4. 打开一个现有的大对象

要打开一个现有的大对象以供读取或写入，调用

```
int lo_open(PGconn *conn, Oid lobjId, int mode);
```

*lobjId*参数指定要打开的大对象的 OID。*mode*位控制该对象是以读取（INV_READ）、写入（INV_WRITE）还是两者兼有的方式打开。（这些符号常量定义在头文件libpq/libpq-fs.h中。）lo_open返回一个（非负的）大对象描述符，供后续在lo_read、lo_write、lo_lseek、lo_tell和lo_close中使用。大对象在被创建之前不能被打开。
该描述符只在当前事务持续期间有效。失败时返回 -1。

服务器当前不区分INV_WRITE和INV_READ | INV_WRITE这两种模式：在这两种情况下都允许通过该描述符读取。不过，这两种模式与单独使用INV_READ
存在一个重要区别：使用INV_READ时，不能通过该描述符写入，
而且从该描述符读取到的数据会反映执行lo_open时活动事务快照中的大对象内容，

而不受本事务或其他事务之后写入的影响。对于以INV_WRITE打开的描述符，读取返回的数据会反映其他已提交事务的所有写入以及当前事务的写入。这类似于普通 SQL SELECT命令在REPEATABLE READ与READ COMMITTED事务模式下的行为差异。

例如：

```
inv_fd = lo_open(conn, inv_oid, INV_READ|INV_WRITE);
```

32.3.5. 向大对象写入数据

函数

```
int lo_write(PGconn *conn, int fd, const char *buf, size_t len);
```

将*buf*中的*len*字节写入大对象描述符*fd*。*fd*参数必须是先前由lo_open返回的大对象描述符。返回值是实际写入的字节数。发生错误时，返回值为负。

32.3.6. 从大对象读取数据

函数

```
int lo_read(PGconn *conn, int fd, char *buf, size_t len);
```

从大对象描述符*fd*中读取*len*字节到*buf*中。*fd*参数必须是先前由lo_open返回的大对象描述符。返回实际读取的字节数。发生错误时，返回值为负。

32.3.7. 在大对象中定位

要改变与大对象描述符关联的当前读或写位置，调用

```
int lo_lseek(PGconn *conn, int fd, int offset, int whence);
```

该函数将由*fd*标识的大对象描述符的当前位置指针移动到由*offset*指定的新位置。*whence*的有效值是SEEK_SET（从对象起始处定位）、SEEK_CUR（从当前位置定位）以及SEEK_END（从对象末尾定位）。返回值是新的位置指针，出错时为 -1。

32.3.8. 获取大对象的当前位置

要取得大对象描述符当前的读或写位置，调用

```
int lo_tell(PGconn *conn, int fd);
```

如果发生错误，返回值是 -1。

32.3.9. 截断一个大对象

要把一个大对象截断为给定长度，调用

```
int lo_truncate(PGcon *conn, int fd, size_t len);
```

该函数把大对象描述符*fd*对应的大对象截断为长度*len*。*fd*参数必须是先前由lo_open返回的大对象描述符。如果*len*大于大对象当前的长度，则会用零字节（'\0'）把该大对象扩展。成功时，lo_truncate返回零；出错时返回值为 -1。

文件偏移量不会改变。

lo_truncate是PostgreSQL 8.3 新增的；如果将该函数用于更早版本的服务器，它会失败并返回一个负值。

32.3.10. 关闭一个大对象描述符

可以通过调用

```
int lo_close(PGconn *conn, int fd);
```

来关闭一个大对象描述符，其中*fd*是由lo_open返回的大对象描述符。成功时，lo_close返回零；出错时返回值为 -1。

任何在事务结束时仍保持打开的大对象描述符都会被自动关闭。

32.3.11. 移除一个大对象

要从数据库中移除一个大对象，调用

```
int lo_unlink(PGconn *conn, Oid lobjId);
```

*lobjId*参数指定要移除的大对象的 OID。成功时返回 1，失败时返回 -1。

32.4. 服务器端函数

前面介绍的每个客户端函数都有对应的服务器端函数可以通过 SQL 调用；实际上，大多数客户端函数都只是等价服务器端函数的接口。通过 SQL 命令调用真正有用的是 lo_creat、lo_create、lo_unlink、lo_import 和 lo_export。下面是它们的使用示例：

```
CREATE TABLE image (
    name            text,
    raster          oid
);

SELECT lo_creat(-1);       -- returns OID of new, empty large
 object

SELECT lo_create(43213);   -- attempts to create large object with
 OID 43213

SELECT lo_unlink(173454);  -- deletes large object with OID 173454

INSERT INTO image (name, raster)
    VALUES ('beautiful image', lo_import('/etc/motd'));

INSERT INTO image (name, raster)  -- same as above, but specify OID
 to use
    VALUES ('beautiful image', lo_import('/etc/motd', 68583));

SELECT lo_export(image.raster, '/tmp/motd') FROM image
    WHERE name = 'beautiful image';
```

服务器端 lo_import 和 lo_export 函数的行为与客户端对应函数有很大不同。这两个函数使用运行数据库服务器的用户的权限，读写服务器文件系统中的文件，
因此它们的用途受限制。相比之下，客户端导入和导出函数读写客户端文件系统中的文件，
使用客户端程序的权限。客户端函数不要求任何特殊的数据库权限。

lo_read 和 lo_write 的功能也可以通过服务器端调用获得，但服务器端函数名与客户端接口不同：不包含下划线。必须将这些函数调用为loread 和 lowrite。

32.5. 示例程序

例 32.1是一个示例程序，它展示了如何使用libpq中的大对象接口。程序中有些部分已经被注释掉，但仍保留在源代码中供读者参考。该程序也可以在源代码发行包的src/test/examples/testlo.c中找到。

例 32.1. 用libpq操作大对象的示例程序

```
/*-------------------------------------------------------------
 *
 * testlo.c--
 *    test using large objects with libpq
 *
 * Copyright (c) 1994, Regents of the University of California
 *
 *-------------------------------------------------------------
 */
#include <stdio.h>
#include "libpq-fe.h"
#include "libpq/libpq-fs.h"

#define BUFSIZE          1024

/*
 * importFile
 *    import file "in_filename" into database as large object "lobj
Oid"
 *
 */
Oid
importFile(PGconn *conn, char *filename)
{
    Oid         lobjId;
    int         lobj_fd;
    char        buf[BUFSIZE];
    int         nbytes,
                tmp;
    int         fd;

    /*
     * open the file to be read in
     */
    fd = open(filename, O_RDONLY, 0666);
    if (fd < 0)
    {                           /* error */
        fprintf(stderr, "cannot open unix file %s\n", filename);
    }

    /*
     * create the large object
     */
    lobjId = lo_creat(conn, INV_READ | INV_WRITE);
    if (lobjId == 0)
        fprintf(stderr, "cannot create large object\n");

    lobj_fd = lo_open(conn, lobjId, INV_WRITE);

    /*
     * read in from the Unix file and write to the inversion file
     */
    while ((nbytes = read(fd, buf, BUFSIZE)) > 0)
    {
        tmp = lo_write(conn, lobj_fd, buf, nbytes);
        if (tmp < nbytes)
```

```
                fprintf(stderr, "error while reading large object\n");
        }

        (void) close(fd);
        (void) lo_close(conn, lobj_fd);

        return lobjId;
    }

    void
    pickout(PGconn *conn, Oid lobjId, int start, int len)
    {
        int         lobj_fd;
        char       *buf;
        int         nbytes;
        int         nread;

        lobj_fd = lo_open(conn, lobjId, INV_READ);
        if (lobj_fd < 0)
        {
            fprintf(stderr, "cannot open large object %d\n",
                    lobjId);
        }

        lo_lseek(conn, lobj_fd, start, SEEK_SET);
        buf = malloc(len + 1);

        nread = 0;
        while (len - nread > 0)
        {
            nbytes = lo_read(conn, lobj_fd, buf, len - nread);
            buf[nbytes] = ' ';
            fprintf(stderr, ">>> %s", buf);
            nread += nbytes;
        }
        free(buf);
        fprintf(stderr, "\n");
        lo_close(conn, lobj_fd);
    }

    void
    overwrite(PGconn *conn, Oid lobjId, int start, int len)
    {
        int         lobj_fd;
        char       *buf;
        int         nbytes;
        int         nwritten;
        int         i;

        lobj_fd = lo_open(conn, lobjId, INV_WRITE);
        if (lobj_fd < 0)
        {
            fprintf(stderr, "cannot open large object %d\n",
                    lobjId);
        }

        lo_lseek(conn, lobj_fd, start, SEEK_SET);
        buf = malloc(len + 1);
```

```
    for (i = 0; i < len; i++)
        buf[i] = 'X';
    buf[i] = ' ';

    nwritten = 0;
    while (len - nwritten > 0)
    {
        nbytes = lo_write(conn, lobj_fd, buf + nwritten, len -
 nwritten);
        nwritten += nbytes;
    }
    free(buf);
    fprintf(stderr, "\n");
    lo_close(conn, lobj_fd);
}

/*
 * exportFile
 *    export large object "lobjOid" to file "out_filename"
 *
 */
void
exportFile(PGconn *conn, Oid lobjId, char *filename)
{
    int         lobj_fd;
    char        buf[BUFSIZE];
    int         nbytes,
                tmp;
    int         fd;

    /*
     * open the large object
     */
    lobj_fd = lo_open(conn, lobjId, INV_READ);
    if (lobj_fd < 0)
    {
        fprintf(stderr, "cannot open large object %d\n",
                lobjId);
    }

    /*
     * open the file to be written to
     */
    fd = open(filename, O_CREAT | O_WRONLY, 0666);
    if (fd < 0)
    {                           /* error */
        fprintf(stderr, "cannot open unix file %s\n",
                filename);
    }

    /*
     * read in from the inversion file and write to the Unix file
     */
    while ((nbytes = lo_read(conn, lobj_fd, buf, BUFSIZE)) > 0)
    {
        tmp = write(fd, buf, nbytes);
        if (tmp < nbytes)
```

```
        {
            fprintf(stderr, "error while writing %s\n",
                    filename);
        }
    }

    (void) lo_close(conn, lobj_fd);
    (void) close(fd);

    return;
}

void
exit_nicely(PGconn *conn)
{
    PQfinish(conn);
    exit(1);
}

int
main(int argc, char **argv)
{
    char       *in_filename,
               *out_filename;
    char       *database;
    Oid         lobjOid;
    PGconn     *conn;
    PGresult   *res;

    if (argc != 4)
    {
        fprintf(stderr, "Usage: %s database_name in_filename out_
filename\n",
                argv[0]);
        exit(1);
    }

    database = argv[1];
    in_filename = argv[2];
    out_filename = argv[3];

    /*
     * set up the connection
     */
    conn = PQsetdb(NULL, NULL, NULL, NULL, database);

    /* check to see that the backend connection was successfully
 made */
    if (PQstatus(conn) == CONNECTION_BAD)
    {
        fprintf(stderr, "Connection to database '%s' failed.\n",
 database);
        fprintf(stderr, "%s", PQerrorMessage(conn));
        exit_nicely(conn);
    }

    res = PQexec(conn, "begin");
    PQclear(res);
```

```
    printf("importing file %s\n", in_filename);
/*  lobjOid = importFile(conn, in_filename); */
    lobjOid = lo_import(conn, in_filename);
/*
    printf("as large object %d.\n", lobjOid);

    printf("picking out bytes 1000-2000 of the large object\n");
    pickout(conn, lobjOid, 1000, 1000);

    printf("overwriting bytes 1000-2000 of the large object with
 X's\n");
    overwrite(conn, lobjOid, 1000, 1000);
*/

    printf("exporting large object to file %s\n", out_filename);
/*    exportFile(conn, lobjOid, out_filename); */
    lo_export(conn, lobjOid, out_filename);

    res = PQexec(conn, "end");
    PQclear(res);
    PQfinish(conn);
    exit(0);
}
```

第 33 章 ECPG - C 中的嵌入式 SQL

这一章描述了用于PostgreSQL的嵌入式SQL包。它由 Linus Tolke（<linus@epact.se>）和 Michael Meskes（<meskes@postgresql.org>）编写。最初它是为了与C一起工作而编写的。它也能与C++配合，但是它还不识别所有的C++结构。

这份文档还很不完整。不过，由于这一接口是标准化的，因此可以在许多关于 SQL 的资料中找到补充信息。

33.1. 概念

嵌入式 SQL 程序由普通编程语言编写的代码（这里是 C）与位于特殊标记区段中的 SQL 命令混合组成。要构建该程序，首先要将源代码（*.pgc）交给嵌入式 SQL 预处理器，它会把源代码转换成普通 C 程序（*.c），之后就可以由 C 编译器处理。（关于编译和链接的细节见第 33.10 节。）转换后的 ECPG 应用会通过嵌入式 SQL 库（ecpglib）调用 libpq 库中的函数，并使用普通的前端/后端协议与 PostgreSQL 服务器通信。

与其他在 C 代码中处理SQL命令的方法相比，嵌入式SQL有其优势。首先，它能处理与你的C程序变量之间来回传递信息的繁琐工作。其次，程序中的 SQL 代码会在构建时接受语法正确性检查。第三，C 中的嵌入式SQL由SQL标准规定，并且得到许多其他SQL数据库系统的支持。PostgreSQL 的实现被设计为尽可能贴合该标准，因此通常可以比较容易地把为其他 SQL 数据库编写的嵌入式SQL程序移植到PostgreSQL。

如前所述，为嵌入式 SQL 接口编写的程序，就是插入了特殊代码的普通 C 程序，这些特殊代码用于执行数据库相关操作，其形式始终如下：

```
EXEC SQL ...;
```

这些语句在语法上占据一条 C 语句的位置。根据具体语句的不同，它们可以出现在全局层级，也可以出现在函数内部。嵌入式 SQL 语句遵循普通 SQL 代码的大小写规则，而不是 C 的规则。

下列小节解释了所有嵌入式 SQL 语句。

33.2. 管理数据库连接

这一节描述如何打开、关闭以及切换数据库连接。

33.2.1. 连接到数据库服务器

使用以下语句连接数据库：

```
EXEC SQL CONNECT TO target [AS connection-name] [USER user-name];
```

其中，*target* 可以按以下方式指定：

- *dbname*[@*hostname*][:*port*]
- tcp:postgresql://*hostname*[:*port*][/*dbname*][?*options*]
- unix:postgresql://*hostname*[:*port*][/*dbname*][?*options*]
- 包含上述某种形式的 SQL 字符串常量
- 对包含上述某种形式的字符变量的引用（见示例）
- DEFAULT

如果直接写出连接目标（即不通过变量引用），且未用引号括起该值，就会应用普通 SQL 的大小写不敏感规则。此时，也可以按需分别用双引号括起各个参数。实际使用中，采用单引号括起的字符串字面值或变量引用，可能更不容易出错。连接目标 `DEFAULT` 会以默认用户名发起到默认数据库的连接，此时不能单独指定用户名或连接名。

也有不同的方法来指定用户名：

- *username*
- *username*/*password*
- *username* IDENTIFIED BY *password*
- *username* USING *password*

如上所述，参数*username*以及*password*可以是一个 SQL 标识符、一个 SQL 字符串常量或者一个对字符变量的引用。

*connection-name*被用来在一个程序中处理多个连接。如果一个程序只使用一个连接，则可以省略它。最近被打开的连接将成为当前连接，当一个 SQL 语句要被执行时，将默认使用它（见这一章稍后的部分）。

以下是一些 `CONNECT` 语句的示例：

```
EXEC SQL CONNECT TO mydb@sql.mydomain.com;

EXEC SQL CONNECT TO unix:postgresql://sql.mydomain.com/mydb AS
 myconnection USER john;

EXEC SQL BEGIN DECLARE SECTION;
const char *target = "mydb@sql.mydomain.com";
const char *user = "john";
const char *passwd = "secret";
EXEC SQL END DECLARE SECTION;
 ...
EXEC SQL CONNECT TO :target USER :user USING :passwd;
/* or EXEC SQL CONNECT TO :target USER :user/:passwd; */
```

最后一种形式使用了上面提到的字符变量引用。后续章节会说明，如何在 C 变量前加冒号，以便在 SQL 语句中使用它们。

请注意，连接目标的格式并未在 SQL 标准中规定。因此，如果你想开发可移植的应用，最好采用类似上面最后一个示例的方法，把连接目标字符串封装在某个地方。

33.2.2. 选择一个连接

嵌入式 SQL 程序中的 SQL 语句默认在当前连接上执行，即最近打开的连接。如果应用需要管理多个连接，可以采用两种方式。

第一种方式是显式地为每一个 SQL 语句选择一个连接，例如：

```
EXEC SQL AT connection-name SELECT ...;
```

如果应用需要以混合的顺序使用多个连接，这种方式特别合适。

如果你的应用使用多个线程执行，它们不能并发地共享一个连接。
你必须显式地控制对连接的访问（使用互斥量）或者为每个线程使用一个连接。

第二种方式是执行一个语句来切换当前的连接。该语句是：

```
EXEC SQL SET CONNECTION connection-name;
```

如果有很多语句要在同一个连接上执行，这种方式尤其方便。

这里有一个管理多个数据库连接的例子程序：

```
#include <stdio.h>

EXEC SQL BEGIN DECLARE SECTION;
    char dbname[1024];
EXEC SQL END DECLARE SECTION;

int
main()
{
    EXEC SQL CONNECT TO testdb1 AS con1 USER testuser;
    EXEC SQL CONNECT TO testdb2 AS con2 USER testuser;
    EXEC SQL CONNECT TO testdb3 AS con3 USER testuser;

    /* 这个查询将在最近打开的数据库 "testdb3" 中执行。 */
    EXEC SQL SELECT current_database() INTO :dbname;
    printf("current=%s (should be testdb3)\n", dbname);

    /* 使用 "AT" 在 "testdb2" 中运行查询 */
    EXEC SQL AT con2 SELECT current_database() INTO :dbname;
    printf("current=%s (should be testdb2)\n", dbname);

    /* 将当前连接切换到 "testdb1"。 */
    EXEC SQL SET CONNECTION con1;

    EXEC SQL SELECT current_database() INTO :dbname;
    printf("current=%s (should be testdb1)\n", dbname);

    EXEC SQL DISCONNECT ALL;
    return 0;
}
```

这个例子将产生这样的输出：

```
current=testdb3 (should be testdb3)
current=testdb2 (should be testdb2)
current=testdb1 (should be testdb1)
```

33.2.3. 关闭一个连接

要关闭连接，请使用以下语句：

```
EXEC SQL DISCONNECT [connection];
```

其中，*connection* 可以按以下方式指定：

- *connection-name*
- DEFAULT
- CURRENT
- ALL

如果未指定连接名，则关闭当前连接。

应用程序始终显式断开其打开的每个连接，是良好的编程习惯。

33.3. 运行 SQL 命令

任何 SQL 命令都可以在一个嵌入式 SQL 应用中被运行。下面是一些在嵌入式 SQL 应用中运行 SQL 命令的例子。

33.3.1. 执行 SQL 语句

创建一个表：

```
EXEC SQL CREATE TABLE foo (number integer, ascii char(16));
EXEC SQL CREATE UNIQUE INDEX num1 ON foo(number);
EXEC SQL COMMIT;
```

插入行：

```
EXEC SQL INSERT INTO foo (number, ascii) VALUES (9999, 'doodad');
EXEC SQL COMMIT;
```

删除行：

```
EXEC SQL DELETE FROM foo WHERE number = 9999;
EXEC SQL COMMIT;
```

更新：

```
EXEC SQL UPDATE foo
    SET ascii = 'foobar'
    WHERE number = 9999;
EXEC SQL COMMIT;
```

返回一个单一结果行的`SELECT`语句也可以直接使用`EXEC SQL`执行。要处理有多行的结果集，一个应用必须使用一个游标，可参考下面的第 33.3.2 节（作为一种特殊情况，一个应用可以一次取出多行到一个数组主变量中，参考第 33.4.4.3.1 节）。

单行选择：

```
EXEC SQL SELECT foo INTO :FooBar FROM table1 WHERE ascii =
 'doodad';
```

还有，一个配置参数可以用`SHOW`命令检索：

```
EXEC SQL SHOW search_path INTO :var;
```

`:`*`something`*形式的词元是主变量，即它们指向 C 程序中的变量。其说明见第 33.4 节。

33.3.2. 使用游标

要读取包含多行的结果集，应用程序必须声明一个游标，并从中逐行取出数据。
使用游标的步骤如下：声明游标、打开游标、从中取出一行、重复取行，最后关闭游标。

使用游标选择：

```
EXEC SQL DECLARE foo_bar CURSOR FOR
    SELECT number, ascii FROM foo
    ORDER BY ascii;
EXEC SQL OPEN foo_bar;
EXEC SQL FETCH foo_bar INTO :FooBar, DooDad;
...
EXEC SQL CLOSE foo_bar;
EXEC SQL COMMIT;
```

有关游标声明的更多细节，参见DECLARE；有关 FETCH 命令的详细信息，参见FETCH。

> **注意**
>
> ECPG DECLARE命令实际上不会向 PostgreSQL 后端发送语句。在执行OPEN命令时，游标才会在后端被打开（使用后端的DECLARE命令）。

33.3.3. 管理事务

在默认模式下，只有发出EXEC SQL COMMIT时命令才会提交。嵌入式 SQL 接口也支持事务自动提交（类似于psql的默认行为），可以通过ecpg的-t命令行选项（见ecpg）或者EXEC SQL SET AUTOCOMMIT TO ON语句启用。在自动提交模式中，除非位于显式事务块内，每条命令都会自动提交。这种模式也可以通过EXEC SQL SET AUTOCOMMIT TO OFF显式关闭。

可以使用下列事务管理命令：

EXEC SQL COMMIT

提交一个进行中的事务。

EXEC SQL ROLLBACK

回滚一个进行中的事务。

EXEC SQL SET AUTOCOMMIT TO ON

启用自动提交模式。

SET AUTOCOMMIT TO OFF

禁用自动提交模式。这是默认值。

33.3.4. 预备语句

当要传给 SQL 语句的值在编译时未知，或者同一语句将使用多次时，预备语句就很有用。

语句使用命令PREPARE进行预备。对于还未知的值，使用占位符“?”：

```
EXEC SQL PREPARE stmt1 FROM "SELECT oid, datname FROM pg_database
 WHERE oid = ?";
```

如果一个语句返回一个单一行，应用可以在PREPARE之后调用EXECUTE来执行该语句，同时要用一个USING子句为占位符提供真实的值：

```
EXEC SQL EXECUTE stmt1 INTO :dboid, :dbname USING 1;
```

如果一个语句返回多行，应用可以使用一个基于该预备语句声明的游标。要绑定输入参数，该游标必须用一个USING子句打开：

```
EXEC SQL PREPARE stmt1 FROM "SELECT oid,datname FROM pg_database
 WHERE oid > ?";
EXEC SQL DECLARE foo_bar CURSOR FOR stmt1;

/* 到达结果集末尾时，跳出 while 循环 */
EXEC SQL WHENEVER NOT FOUND DO BREAK;

EXEC SQL OPEN foo_bar USING 100;
...
```

```
while (1)
{
    EXEC SQL FETCH NEXT FROM foo_bar INTO :dboid, :dbname;
    ...
}
EXEC SQL CLOSE foo_bar;
```

当你不再需要该预备语句时，你应该释放它：

```
EXEC SQL DEALLOCATE PREPARE name;
```

更多有关`PREPARE`的细节，可参考PREPARE。关于使用占位符和输入参数的细节，可参考第 33.5 节。

33.4. 使用主变量

在第 33.3 节中，你已经看到如何从嵌入式 SQL 程序中执行 SQL 语句。其中一些语句只使用固定值，既无法把用户提供的值插入语句中，也无法让程序处理查询返回的值。这类语句在实际应用中并不真正有用。本节将详细解释如何使用一种称为主变量的简单机制，在 C 程序与嵌入式 SQL 语句之间传递数据。在嵌入式 SQL 程序中，我们把 SQL 语句视为 C 程序代码中的客体，而 C 是主语言。因此，C 程序中的变量被称为主变量。

另一种在 PostgreSQL 后端和 ECPG 应用之间交换值的方式是使用 SQL 描述符，它在第 33.7 节中介绍。

33.4.1. 概述

在嵌入式 SQL 中，在 C 程序和 SQL 语句之间传递数据特别简单。我们不必让程序把数据拼接进语句中去，那样会带来诸如正确为值加引号等各种复杂问题；只需在 SQL 语句中写出 C 变量的名称，并在前面加上一个冒号即可。例如：

```
EXEC SQL INSERT INTO sometable VALUES (:v1, 'foo', :v2);
```

该语句引用了两个名为`v1`和`v2`的 C 变量，同时还使用了一个普通的 SQL 字符串常量，以说明你并不局限于只使用某一种数据。

这种在 SQL 语句中插入 C 变量的方式可以用在 SQL 语句中每一个应该出现值表达式的地方。

33.4.2. 声明节

要把数据从程序传给数据库（例如作为查询参数），或者把数据从数据库传回程序，用来承载这些数据的 C 变量必须声明在经过特殊标记的区段中，这样嵌入式 SQL 预处理器才能识别它们。

这个区段始于：

```
EXEC SQL BEGIN DECLARE SECTION;
```

并结束于：

```
EXEC SQL END DECLARE SECTION;
```

在这两行之间，必须是正常的 C 变量声明，例如：

```
int   x = 4;
char  foo[16], bar[16];
```

如你所见，可以选择为变量指定初始值。变量的作用域由其声明节在程序中的位置决定。你也可以使用下面的语法声明变量，这种写法会隐式创建一个声明节：

```
EXEC SQL int i = 4;
```

一个程序中可以根据需要放置任意多个声明节。

这些声明也会作为普通 C 变量声明写入输出文件，因此无需再次声明它们。不准备在 SQL 命令中使用的变量可以照常在这些特殊声明节之外声明。

结构体或联合的定义也必须写在DECLARE声明节中。否则预处理器无法处理这些类型，因为它不知道它们的定义。

33.4.3. 检索查询结果

现在你应该能够把程序产生的数据传递到一个 SQL 命令中了。但是怎么检索一个查询的结果呢？为此，嵌入式 SQL 提供了常规命令SELECT和FETCH的特殊变体。这些命令有一个特殊的INTO子句，它指定被检索到的值要被存储在哪些主变量中。SELECT被用于只返回单一行的查询，而FETCH被用于使用一个游标返回多行的查询。

这里是一个例子：

```
/*
 * 假定有下列表:
 * CREATE TABLE test1 (a int, b varchar(50));
 */

EXEC SQL BEGIN DECLARE SECTION;
int v1;
VARCHAR v2;
EXEC SQL END DECLARE SECTION;

 ...

EXEC SQL SELECT a, b INTO :v1, :v2 FROM test;
```

那么INTO子句出现在选择列表和FROM子句之间。选择列表中的元素数量必须和INTO后面列表（也被称为目标列表）的元素数量相等。

这里有一个使用命令FETCH的例子：

```
EXEC SQL BEGIN DECLARE SECTION;
int v1;
VARCHAR v2;
EXEC SQL END DECLARE SECTION;

 ...

EXEC SQL DECLARE foo CURSOR FOR SELECT a, b FROM test;

 ...

do
{
    ...
    EXEC SQL FETCH NEXT FROM foo INTO :v1, :v2;
    ...
} while (...);
```

这里INTO子句出现在所有正常子句的后面。

33.4.4. 类型映射

当 ECPG 应用在 PostgreSQL 服务器和 C 应用之间交换值时（例如从服务器检索查询结果时或者用输入参数执行 SQL 语句时），值需要在 PostgreSQL 数据类型和主语言变量类型（具体

来说是 C 语言数据类型）之间转换。ECPG 的要点之一就是它会在大多数情况下自动完成这种转换。

在这方面有两类数据类型：一些简单 PostgreSQL 数据类型（例如`integer`和`text`）可以被应用直接读取和写入。其他 PostgreSQL 数据类型（例如`timestamp`和`numeric`）只能通过特殊库函数访问，见第 33.4.4.2 节。

表 33.1展示了哪种 PostgreSQL 数据类型对应于哪一种 C 数据类型。当你希望发送或接收一种给定 PostgreSQL 数据类型的值时，你应该在声明节中声明一个具有相应 C 数据类型的 C 变量。

表 33.1. 在 PostgreSQL 数据类型和 C 变量类型之间映射

PostgreSQL 数据类型	主变量类型
`smallint`	`short`
`integer`	`int`
`bigint`	`long long int`
`decimal`	`decimal`[a]
`numeric`	`numeric`[a]
`real`	`float`
`double precision`	`double`
`serial`	`int`
`smallserial`	`short`
`bigserial`	`long long int`
`oid`	`unsigned int`
`character(n)`,`varchar(n)`,`text`	`char[n+1]`,`VARCHAR[n+1]`[b]
`name`	`char[NAMEDATALEN]`
`timestamp`	`timestamp`[a]
`interval`	`interval`[a]
`date`	`date`[a]
`boolean`	`bool`[c]

[a]这种类型只能通过特殊的库函数访问，见第 33.4.4.2 节。
[b]在 `ecpglib.h` 中声明
[c]如果不是原生类型，则在`ecpglib.h`中声明

33.4.4.1. 处理字符串

要处理 SQL 字符串数据类型（例如`varchar`以及`text`），有两种可能的方式来声明主变量。

一种方式是使用`char[]`（一个`char`数组），这是在 C 中处理字符数据最常见的方式。

```
EXEC SQL BEGIN DECLARE SECTION;
    char str[50];
EXEC SQL END DECLARE SECTION;
```

注意你必须自行管理长度。
如果你把这个主变量用作一个查询的目标变量并且该查询返回超过 49 个字符的字符串，那么将会发生缓冲区溢出。

另一种方式是使用`VARCHAR`类型，它是 ECPG 提供的一种特殊类型。`VARCHAR`数组的定义会为每个变量分别转换成一个具名`struct`。这样一个声明：

```
VARCHAR var[180];
```

会被转变成：

```
struct varchar_var { int len; char arr[180]; } var;
```

成员arr容纳包含一个终止零字节的字符串。因此，要在一个VARCHAR主变量中存储一个字符串，该主变量必须被声明为具有包括零字节终止符的长度。成员len保存存储在arr中的字符串的长度，不包括终止零字节。当一个主变量被用作一个查询的输入时，如果strlen(arr)和len不同，将使用短的那一个。

VARCHAR可以被写成大写或小写形式，但是不能大小写混合。

char和VARCHAR主变量也可以保存其他 SQL 类型的值，它们将被存储为字符串形式。

33.4.4.2. 访问特殊数据类型

ECPG 包含一些特殊类型帮助你容易地与来自 PostgreSQL 服务器的一些特殊数据类型交互。特别地，它已经实现了对于numeric、decimal、date、timestamp以及interval类型的支持。这些数据类型无法有效地被映射到简单的主变量类型（例如int、long long int或者char[]），因为它们有一种复杂的内部结构。应用通过声明特殊类型的主变量以及使用 pgtypes 库中的函数来处理这些类型。pgtypes 库（在第 33.6 节中详细描述）包含了处理这些类型的基本函数，这样你不需要仅仅为了给一个时间戳加上一个时间间隔而发送一个查询给 SQL 服务器。

下面的小节描述了这些特殊数据类型。关于 pgtypes 库函数的更多细节，请参考第 33.6 节。

33.4.4.2.1. timestamp, date

这里有一种在 ECPG 主应用中处理timestamp变量的模式。

首先，程序必须包括用于timestamp类型的头文件：

```
#include <pgtypes_timestamp.h>
```

接着，在声明节中声明一个主变量为类型timestamp：

```
EXEC SQL BEGIN DECLARE SECTION;
timestamp ts;
EXEC SQL END DECLARE SECTION;
```

并且在读入一个值到该主变量中之后，使用 pgtypes 库函数处理它。在下面的例子中，timestamp值被PGTYPEStimestamp_to_asc()函数转变成文本（ASCII）形式：

```
EXEC SQL SELECT now()::timestamp INTO :ts;

printf("ts = %s\n", PGTYPEStimestamp_to_asc(ts));
```

这个例子将展示像下面形式的一些结果：

```
ts = 2010-06-27 18:03:56.949343
```

另外，DATE 类型可以用相同的方式处理。程序必须包括pgtypes_date.h，声明一个主变量为日期类型并且将一个 DATE 值使用PGTYPESdate_to_asc()函数转变成一种文本形式。关于 pgtypes 库函数的更多细节，请参考第 33.6 节。

33.4.4.2.2. interval

对interval类型的处理也类似于timestamp和date类型。不过，必须显式为一个interval类型分配内存。换句话说，该变量的内存空间必须在堆内存中分配，而不是在栈内存中分配。

这里是一个例子程序：

```
#include <stdio.h>
#include <stdlib.h>
#include <pgtypes_interval.h>

int
main(void)
{
EXEC SQL BEGIN DECLARE SECTION;
    interval *in;
EXEC SQL END DECLARE SECTION;

    EXEC SQL CONNECT TO testdb;

    in = PGTYPESinterval_new();
    EXEC SQL SELECT '1 min'::interval INTO :in;
    printf("interval = %s\n", PGTYPESinterval_to_asc(in));
    PGTYPESinterval_free(in);

    EXEC SQL COMMIT;
    EXEC SQL DISCONNECT ALL;
    return 0;
}
```

33.4.4.2.3. numeric, decimal

`numeric`和`decimal`类型的处理类似于`interval`类型：需要定义一个指针、在堆上分配一些内存空间并且使用 pgtypes 库函数访问该变量。关于 pgtypes 库函数的更多细节，请参考第 33.6 节。

pgtypes 库没有特别为`decimal`类型提供函数。一个应用必须使用一个 pgtypes 库函数把它转变成一个`numeric`变量以便进一步处理。

这里是一个处理`numeric`和`decimal`类型变量的例子程序。

```
#include <stdio.h>
#include <stdlib.h>
#include <pgtypes_numeric.h>

EXEC SQL WHENEVER SQLERROR STOP;

int
main(void)
{
EXEC SQL BEGIN DECLARE SECTION;
    numeric *num;
    numeric *num2;
    decimal *dec;
EXEC SQL END DECLARE SECTION;

    EXEC SQL CONNECT TO testdb;

    num = PGTYPESnumeric_new();
    dec = PGTYPESdecimal_new();

    EXEC SQL SELECT 12.345::numeric(4,2), 23.456::decimal(4,2)
 INTO :num, :dec;
```

```
    printf("numeric = %s\n", PGTYPESnumeric_to_asc(num, 0));
    printf("numeric = %s\n", PGTYPESnumeric_to_asc(num, 1));
    printf("numeric = %s\n", PGTYPESnumeric_to_asc(num, 2));

    /* 将 decimal 转换为 numeric，以显示 decimal 值。 */
    num2 = PGTYPESnumeric_new();
    PGTYPESnumeric_from_decimal(dec, num2);

    printf("decimal = %s\n", PGTYPESnumeric_to_asc(num2, 0));
    printf("decimal = %s\n", PGTYPESnumeric_to_asc(num2, 1));
    printf("decimal = %s\n", PGTYPESnumeric_to_asc(num2, 2));

    PGTYPESnumeric_free(num2);
    PGTYPESdecimal_free(dec);
    PGTYPESnumeric_free(num);

    EXEC SQL COMMIT;
    EXEC SQL DISCONNECT ALL;
    return 0;
}
```

33.4.4.3. 非简单类型的主变量

你也可以把数组、typedef、结构体和指针用作主变量。

33.4.4.3.1. 数组

将数组用作主变量有两种情况。第一种如第 33.4.4.1 节所述，是一种将一些文本字符串存储在`char[]`或`VARCHAR[]`中的方法。第二种是不用一个游标从一个查询结果中检索多行。
如果没有一个数组，要处理由多个行组成的查询结果，我们需要使用一个游标以及`FETCH`命令。但是使用数组主变量，多个行可以被一次收取。
该数组的长度必须被定义成足以容纳所有的行，否则很可能会发生一次缓冲区溢出。

下面的例子扫描`pg_database`系统表并且显示所有可用数据库的 OID 和名称：

```
int
main(void)
{
EXEC SQL BEGIN DECLARE SECTION;
    int dbid[8];
    char dbname[8][16];
    int i;
EXEC SQL END DECLARE SECTION;

    memset(dbname, 0, sizeof(char)* 16 * 8);
    memset(dbid, 0, sizeof(int) * 8);

    EXEC SQL CONNECT TO testdb;

    /* 一次将多行取入数组。 */
    EXEC SQL SELECT oid,datname INTO :dbid, :dbname FROM pg_
database;

    for (i = 0; i < 8; i++)
        printf("oid=%d, dbname=%s\n", dbid[i], dbname[i]);

    EXEC SQL COMMIT;
    EXEC SQL DISCONNECT ALL;
```

```
    return 0;
}
```

这个例子显示下面的结果（确切的值取决于本地环境）。

```
oid=1, dbname=template1
oid=11510, dbname=template0
oid=11511, dbname=postgres
oid=313780, dbname=testdb
oid=0, dbname=
oid=0, dbname=
oid=0, dbname=
```

33.4.4.3.2. 结构体

一个成员名称匹配查询结果列名的结构体可以被用来一次检索多列。
该结构体使得我们能够在一个单一主变量中处理多列值。

下面的例子通过 `pg_database` 系统表和 `pg_database_size()` 函数检索可用数据库的 OID、名称和尺寸。在这个例子中，一个成员名匹配`SELECT`结果的每一列的结构体变量`dbinfo_t`被用来检索结果行，而不需要把多个主变量放在`FETCH`语句中。

```
EXEC SQL BEGIN DECLARE SECTION;
    typedef struct
    {
       int oid;
       char datname[65];
       long long int size;
    } dbinfo_t;

    dbinfo_t dbval;
EXEC SQL END DECLARE SECTION;

    memset(&dbval, 0, sizeof(dbinfo_t));

    EXEC SQL DECLARE cur1 CURSOR FOR SELECT oid, datname, pg_
database_size(oid) AS size FROM pg_database;
    EXEC SQL OPEN cur1;

    /* 到达结果集末尾时，跳出 while 循环 */
    EXEC SQL WHENEVER NOT FOUND DO BREAK;

    while (1)
    {
        /* 将多列取入一个结构体。 */
        EXEC SQL FETCH FROM cur1 INTO :dbval;

        /* 打印结构体的成员。 */
        printf("oid=%d, datname=%s, size=%lld\n", dbval.oid, dbval.
datname, dbval.size);
    }

    EXEC SQL CLOSE cur1;
```

这个例子会显示下列结果（确切的值取决于本地环境）。

```
oid=1, datname=template1, size=4324580
oid=11510, datname=template0, size=4243460
oid=11511, datname=postgres, size=4324580
```

```
oid=313780, datname=testdb, size=8183012
```

结构体主变量“吸收”的列数与结构体的字段数相同。额外的列可以被分配给其他主变量。例如，上面的程序也可以使用结构体外部的`size`变量改写：

```
EXEC SQL BEGIN DECLARE SECTION;
    typedef struct
    {
       int oid;
       char datname[65];
    } dbinfo_t;

    dbinfo_t dbval;
    long long int size;
EXEC SQL END DECLARE SECTION;

    memset(&dbval, 0, sizeof(dbinfo_t));

    EXEC SQL DECLARE cur1 CURSOR FOR SELECT oid, datname, pg_
database_size(oid) AS size FROM pg_database;
    EXEC SQL OPEN cur1;

    /* 到达结果集末尾时，跳出 while 循环 */
    EXEC SQL WHENEVER NOT FOUND DO BREAK;

    while (1)
    {
        /* 将多列取入一个结构体。 */
        EXEC SQL FETCH FROM cur1 INTO :dbval, :size;

        /* 打印结构体的成员。 */
        printf("oid=%d, datname=%s, size=%lld\n", dbval.oid, dbval.
datname, size);
    }

    EXEC SQL CLOSE cur1;
```

33.4.4.3.3. typedef

使用 `typedef` 关键字将新类型映射到已有类型。

```
EXEC SQL BEGIN DECLARE SECTION;
    typedef char mychartype[40];
    typedef long serial_t;
EXEC SQL END DECLARE SECTION;
```

注意，也可以使用：

```
EXEC SQL TYPE serial_t IS long;
```

此声明不必放在声明区段内。

33.4.4.3.4. 指针

你可以声明指向最常见类型的指针。但要注意，只有使用自动内存分配时，才能将指针用作查询的目标变量。关于自动内存分配的更多信息，参见第 33.7 节。

```
EXEC SQL BEGIN DECLARE SECTION;
    int   *intp;
    char **charp;
```

```
EXEC SQL END DECLARE SECTION;
```

33.4.5. 处理非原始 SQL 数据类型

本节介绍如何在 ECPG 应用中处理非标量以及用户定义的 SQL 层数据类型。注意，这与上一节介绍的非简单类型主变量的处理方式是不同的。

33.4.5.1. 数组

ECPG 不直接支持 SQL 层的多维数组。一维 SQL 数组可以映射为 C 数组主变量，反之亦然。不过，在创建语句时，ecpg 并不知道列的类型，因此无法检查某个 C 数组是否会作为对应 SQL 数组的输入。在处理 SQL 语句输出时，ecpg 则掌握所需信息，因此会检查双方是否都是数组。

如果查询是分别访问数组的元素，那就可以避开在 ECPG 中直接使用数组。此时应使用一种能映射到该元素类型的主变量。例如，如果某列的类型是`integer`数组，就可以使用`int`类型的主变量；如果元素类型是`varchar`或`text`，则可以使用`char[]`或`VARCHAR[]`类型的主变量。

下面是一个示例。假定有下列表：

```
CREATE TABLE t3 (
    ii integer[]
);

testdb=> SELECT * FROM t3;
     ii
-------------
 {1,2,3,4,5}
(1 row)
```

下面的示例程序检索该数组的第 4 个元素，并将其存入一个类型为`int`的主变量中：

```
EXEC SQL BEGIN DECLARE SECTION;
int ii;
EXEC SQL END DECLARE SECTION;

EXEC SQL DECLARE cur1 CURSOR FOR SELECT ii[4] FROM t3;
EXEC SQL OPEN cur1;

EXEC SQL WHENEVER NOT FOUND DO BREAK;

while (1)
{
    EXEC SQL FETCH FROM cur1 INTO :ii ;
    printf("ii=%d\n", ii);
}

EXEC SQL CLOSE cur1;
```

该示例会显示下列结果：

```
ii=4
```

要把多个数组元素映射到一个数组类型主变量中的多个元素，
数组列的每一个元素以及主变量数组的每一个元素都必须被单独管理，例如：

```
EXEC SQL BEGIN DECLARE SECTION;
int ii_a[8];
EXEC SQL END DECLARE SECTION;
```

```
EXEC SQL DECLARE cur1 CURSOR FOR SELECT ii[1], ii[2], ii[3], ii[4]
 FROM t3;
EXEC SQL OPEN cur1;

EXEC SQL WHENEVER NOT FOUND DO BREAK;

while (1)
{
    EXEC SQL FETCH FROM cur1 INTO :ii_a[0], :ii_a[1], :ii_a[2], :
ii_a[3];
    ...
}
```

注意

```
EXEC SQL BEGIN DECLARE SECTION;
int ii_a[8];
EXEC SQL END DECLARE SECTION;

EXEC SQL DECLARE cur1 CURSOR FOR SELECT ii FROM t3;
EXEC SQL OPEN cur1;

EXEC SQL WHENEVER NOT FOUND DO BREAK;

while (1)
{
    /* 错误 */
    EXEC SQL FETCH FROM cur1 INTO :ii_a;
    ...
}
```

在这种情况中不会正确工作，因为你无法把一个数组类型列直接映射到一个数组主变量。

另一种变通办法是把数组的外部字符串表示存入`char[]`或`VARCHAR[]`类型的主变量中。关于这种表示的更多细节见第 8.14.2 节。注意，这意味着在宿主程序中无法自然地把它当作数组访问，除非再做一步解析文本表示的处理。

33.4.5.2. 复合类型

ECPG 不直接支持复合类型，但有一种简单的变通办法。可用方案与上文处理数组时类似：要么分别访问每个属性，要么使用外部字符串表示。

对于下列例子，假定有下面的类型和表：

```
CREATE TYPE comp_t AS (intval integer, textval varchar(32));
CREATE TABLE t4 (compval comp_t);
INSERT INTO t4 VALUES ( (256, 'PostgreSQL') );
```

最显而易见的解决方案是单独访问每一个属性。下面的程序通过单独选择类型`comp_t`的每一个属性从例子表中检索数据：

```
EXEC SQL BEGIN DECLARE SECTION;
int intval;
varchar textval[33];
EXEC SQL END DECLARE SECTION;

/* 将复合类型列的每个元素放入 SELECT 列表。 */
EXEC SQL DECLARE cur1 CURSOR FOR SELECT (compval).intval, (compval)
.textval FROM t4;
```

```
EXEC SQL OPEN cur1;

EXEC SQL WHENEVER NOT FOUND DO BREAK;

while (1)
{
    /* 将复合类型列的每个元素取入主变量。 */
    EXEC SQL FETCH FROM cur1 INTO :intval, :textval;

    printf("intval=%d, textval=%s\n", intval, textval.arr);
}

EXEC SQL CLOSE cur1;
```

这个示例还可以进一步改进：把在FETCH命令中接收值的主变量集中到一个结构体里。关于结构体形式主变量的更多细节，见第 33.4.4.3.2 节。改写成结构体形式后，两个主变量intval和textval就成为comp_t结构体的成员，并在FETCH命令中指定该结构体。

```
EXEC SQL BEGIN DECLARE SECTION;
typedef struct
{
    int intval;
    varchar textval[33];
} comp_t;

comp_t compval;
EXEC SQL END DECLARE SECTION;

/* 将复合类型列的每个元素放入 SELECT 列表。 */
EXEC SQL DECLARE cur1 CURSOR FOR SELECT (compval).intval, (compval)
.textval FROM t4;
EXEC SQL OPEN cur1;

EXEC SQL WHENEVER NOT FOUND DO BREAK;

while (1)
{
    /* 将 SELECT 列表中的所有值放入一个结构体。 */
    EXEC SQL FETCH FROM cur1 INTO :compval;

    printf("intval=%d, textval=%s\n", compval.intval, compval.
textval.arr);
}

EXEC SQL CLOSE cur1;
```

尽管在FETCH命令中使用了结构体，SELECT子句中的属性名仍要逐一列出。可以通过使用*来请求该复合类型值的所有属性，从而进一步改进：

```
...
EXEC SQL DECLARE cur1 CURSOR FOR SELECT (compval).* FROM t4;
EXEC SQL OPEN cur1;

EXEC SQL WHENEVER NOT FOUND DO BREAK;

while (1)
{
    /* 将 SELECT 列表中的所有值放入一个结构体。 */
    EXEC SQL FETCH FROM cur1 INTO :compval;
```

```
    printf("intval=%d, textval=%s\n", compval.intval, compval.
textval.arr);
}
...
```

采用这种方法后，尽管 ECPG 本身并不理解复合类型，复合类型仍然几乎可以无缝映射到结构体上。

最后，也可以把复合类型值的外部字符串表示存入`char[]`或`VARCHAR[]`类型的主变量中。不过，如果采用这种方法，就很难在宿主程序中访问该值的各个字段。

33.4.5.3. 用户定义的基础类型

ECPG 并不直接支持新的用户定义的基础类型。你可以使用外部字符串表示以及类型`char[]`或`VARCHAR[]`的主变量，并且这种方案事实上对很多类型都是合适和足够的。

下面的示例使用数据类型 `complex`，它来自第 35.11 节中的示例。该类型的外部字符串表示为 `(%f,%f)`，由函数 `complex_in()` 和 `complex_out()` 定义，见第 35.11 节。以下示例将 complex 类型的值 `(1,1)` 和 `(3,3)` 分别插入列 `a` 和 `b`，然后从表中查询它们。

```
EXEC SQL BEGIN DECLARE SECTION;
    varchar a[64];
    varchar b[64];
EXEC SQL END DECLARE SECTION;

    EXEC SQL INSERT INTO test_complex VALUES ('(1,1)', '(3,3)');

    EXEC SQL DECLARE cur1 CURSOR FOR SELECT a, b FROM test_complex;
    EXEC SQL OPEN cur1;

    EXEC SQL WHENEVER NOT FOUND DO BREAK;

    while (1)
    {
        EXEC SQL FETCH FROM cur1 INTO :a, :b;
        printf("a=%s, b=%s\n", a.arr, b.arr);
    }

    EXEC SQL CLOSE cur1;
```

该示例的结果如下：

```
a=(1,1), b=(3,3)
```

另一种变通方案是避免在 ECPG 中直接使用用户定义的类型，而是创建一个在用户定义的类型和 ECPG 能处理的简单类型之间转换的函数或者类型转换。不过要注意，在类型系统中引入类型转换（特别是隐式类型转换）要非常小心。

例如，

```
CREATE FUNCTION create_complex(r double, i double) RETURNS complex
LANGUAGE SQL
IMMUTABLE
AS $$ SELECT $1 * complex '(1,0')' + $2 * complex '(0,1)' $$;
```

在这个定义之后，下面的语句

```
EXEC SQL BEGIN DECLARE SECTION;
double a, b, c, d;
```

```
EXEC SQL END DECLARE SECTION;

a = 1;
b = 2;
c = 3;
d = 4;

EXEC SQL INSERT INTO test_complex VALUES (create_complex(:a, :b),
 create_complex(:c, :d));
```

具有和

```
EXEC SQL INSERT INTO test_complex VALUES ('(1,2)', '(3,4)');
```

相同的效果。

33.4.6. 指示符

上面的示例没有处理空值。实际上，如果读取示例从数据库取得空值，就会抛出错误。
要向数据库传入空值或从数据库取得空值，需要在每个包含数据的主变量后，
再指定第二个主变量。第二个主变量称为指示符，其中包含一个标志，
表示该数据是否为空值；如果为空值，就会忽略实际主变量的值。
下面的示例能正确处理空值的读取：

```
EXEC SQL BEGIN DECLARE SECTION;
VARCHAR val;
int val_ind;
EXEC SQL END DECLARE SECTION:

 ...

EXEC SQL SELECT b INTO :val :val_ind FROM test1;
```

指示符变量 `val_ind` 在数据非空时为零，在数据为空值时为负数。

指示符有另一种功能：如果指示符值为正，它表示值不为空，
但是当它被存储在主变量中时已被截断。

如果向预处理器`ecpg`传入参数`-r no_indicator`，它就会工作在“无指示符”模式下。在这种模式中，如果没有指定指示符变量，那么对于字符串类型，
空值会在输入和输出时表现为空串；对于整数类型，
空值会表现为该类型可能的最小值（例如对于`int`就是`INT_MIN`）。

33.5. 动态 SQL

在很多情况中，一个应用必须要执行的特定 SQL 语句在编写该应用时就已知。不过在某些情况中，SQL 语句在运行时构造或者由一个外部来源提供。这样你就不能直接把 SQL 语句嵌入到 C 源代码，不过有一种功能允许你调用在一个字符串变量中提供的任意 SQL 语句。

33.5.1. 执行没有结果集的语句

执行一个任意 SQL 语句的最简单方法是使用命令`EXECUTE IMMEDIATE`。例如：

```
EXEC SQL BEGIN DECLARE SECTION;
const char *stmt = "CREATE TABLE test1 (...);";
EXEC SQL END DECLARE SECTION;

EXEC SQL EXECUTE IMMEDIATE :stmt;
```

EXECUTE IMMEDIATE可以被用于不返回结果集的 SQL 语句（例如 DDL、INSERT、UPDATE、DELETE）。你不能用这种方法执行检索数据的语句（例如SELECT）。
下一节将描述如何执行这一种语句。

33.5.2. 执行一个有输入参数的语句

执行任意 SQL 语句的一种更强大的方法是将语句预备一次并且在每次需要时执行该预备语句。也可以预备一个一般化的语句，然后通过替换参数执行它的特定版本。在预备语句时，在你想要稍后替换参数的地方写上问号。例如：

```
EXEC SQL BEGIN DECLARE SECTION;
const char *stmt = "INSERT INTO test1 VALUES(?, ?);";
EXEC SQL END DECLARE SECTION;

EXEC SQL PREPARE mystmt FROM :stmt;
 ...
EXEC SQL EXECUTE mystmt USING 42, 'foobar';
```

当你不再需要该预备语句时，你应该释放它：

```
EXEC SQL DEALLOCATE PREPARE name;
```

33.5.3. 执行一个有结果集的语句

要执行一个只有单一结果行的 SQL 语句，可以使用EXECUTE。要保存结果，在其中增加一个INTO子句。

```
EXEC SQL BEGIN DECLARE SECTION;
const char *stmt = "SELECT a, b, c FROM test1 WHERE a > ?";
int v1, v2;
VARCHAR v3[50];
EXEC SQL END DECLARE SECTION;

EXEC SQL PREPARE mystmt FROM :stmt;
 ...
EXEC SQL EXECUTE mystmt INTO :v1, :v2, :v3 USING 37;
```

一个EXECUTE命令可以有一个INTO子句、一个USING子句，可以同时有这两个子句，也可以不带这两个子句。

如果一个查询被期望返回多于一个结果行，
应该如下列例子所示使用一个游标（关于游标详见第 33.3.2 节）。

```
EXEC SQL BEGIN DECLARE SECTION;
char dbaname[128];
char datname[128];
char *stmt = "SELECT u.usename as dbaname, d.datname "
             "  FROM pg_database d, pg_user u "
             "  WHERE d.datdba = u.usesysid";
EXEC SQL END DECLARE SECTION;

EXEC SQL CONNECT TO testdb AS con1 USER testuser;

EXEC SQL PREPARE stmt1 FROM :stmt;

EXEC SQL DECLARE cursor1 CURSOR FOR stmt1;
EXEC SQL OPEN cursor1;

EXEC SQL WHENEVER NOT FOUND DO BREAK;
```

```
while (1)
{
    EXEC SQL FETCH cursor1 INTO :dbaname,:datname;
    printf("dbaname=%s, datname=%s\n", dbaname, datname);
}

EXEC SQL CLOSE cursor1;

EXEC SQL COMMIT;
EXEC SQL DISCONNECT ALL;
```

33.6. pgtypes 库

pgtypes 库将PostgreSQL数据库类型映射到 C 中等价的类型以便在 C 程序中使用。它还提供在 C 中对这些类型进行基本计算的函数，即不依赖PostgreSQL服务器进行计算。请看下面的例子：

```
EXEC SQL BEGIN DECLARE SECTION;
   date date1;
   timestamp ts1, tsout;
   interval iv1;
   char *out;
EXEC SQL END DECLARE SECTION;

PGTYPESdate_today(&date1);
EXEC SQL SELECT started, duration INTO :ts1, :iv1 FROM datetbl
 WHERE d=:date1;
PGTYPEStimestamp_add_interval(&ts1, &iv1, &tsout);
out = PGTYPEStimestamp_to_asc(&tsout);
printf("Started + duration: %s\n", out);
free(out);
```

33.6.1. numeric类型

numeric 类型支持任意精度计算。PostgreSQL服务器中的等效类型见第 8.1 节。为了支持任意精度，这类变量需要能够动态扩展和收缩。因此，numeric 变量只能在堆上创建，并通过`PGTYPESnumeric_new`和`PGTYPESnumeric_free`函数管理。decimal 类型与之相似，但精度有限，既可以在栈上创建，也可以在堆上创建。

下列函数可用于处理 numeric 类型：

`PGTYPESnumeric_new`

请求一个指向新分配的numeric变量的指针。

```
numeric *PGTYPESnumeric_new(void);
```

`PGTYPESnumeric_free`

释放一个 numeric 变量及其占用的全部内存。

```
void PGTYPESnumeric_free(numeric *var);
```

`PGTYPESnumeric_from_asc`

从字符串表示中解析一个 numeric 值。

```
numeric *PGTYPESnumeric_from_asc(char *str, char **endptr);
```

例如，可用的格式是：-2、.794、+3.44、592.49E07或者 -32.84e-4。如果值能被成功地解析，将返回一个有效的指针，否则返回 NULL 指针。目前 ECPG 总是解析整个字符串并且因此当前不支持把第一个非法字符的地址存储在*endptr中。
你可以安全地把endptr设置为 NULL。

PGTYPESnumeric_to_asc

返回由malloc分配的字符串的指针，它包含numeric类型num的字符串表示。

```
char *PGTYPESnumeric_to_asc(numeric *num, int dscale);
```

numeric 值将以dscale位小数输出，必要时进行舍入。

PGTYPESnumeric_add

把两个numeric变量相加放到第三个numeric变量中。

```
int PGTYPESnumeric_add(numeric *var1, numeric *var2, numeric
 *result);
```

该函数把变量var1和var2相加放到结果变量result中。成功时该函数返回 0，出错时返回 -1。

PGTYPESnumeric_sub

把两个numeric变量相减并且把结果返回到第三个numeric变量。

```
int PGTYPESnumeric_sub(numeric *var1, numeric *var2, numeric
 *result);
```

该函数把变量var2从变量var1中减除。该操作的结果被存储在变量result中。
成功时该函数返回 0，出错时返回 -1。

PGTYPESnumeric_mul

把两个numeric变量相乘并且把结果返回到第三个numeric变量。

```
int PGTYPESnumeric_mul(numeric *var1, numeric *var2, numeric
 *result);
```

该函数把变量var1和var2相乘。该操作的结果被存储在变量result中。
成功时该函数返回 0，出错时返回 -1。

PGTYPESnumeric_div

把两个numeric变量相除并且把结果返回到第三个numeric变量。

```
int PGTYPESnumeric_div(numeric *var1, numeric *var2, numeric
 *result);
```

该函数用变量var2除变量var1。该操作的结果被存储在变量result中。
成功时该函数返回 0，出错时返回 -1。

PGTYPESnumeric_cmp

比较两个numeric变量。

```
int PGTYPESnumeric_cmp(numeric *var1, numeric *var2)
```

这个函数比较两个numeric变量。错误时会返回INT_MAX。成功时，
该函数返回三种可能结果之一：

- var1大于var2则返回 1

- 如果var1小于var2则返回 -1
- 如果var1和var2相等则返回 0

PGTYPESnumeric_from_int

将一个 int 变量转换为 numeric 变量。

```
int PGTYPESnumeric_from_int(signed int int_val, numeric *var);
```

该函数接收一个 signed int 类型的变量，并将其值存入 numeric 变量var中。成功时返回 0，失败时返回 -1。

PGTYPESnumeric_from_long

将一个 long int 变量转换为 numeric 变量。

```
int PGTYPESnumeric_from_long(signed long int long_val, numeric
 *var);
```

该函数接收一个 signed long int 类型的变量，并将其值存入 numeric 变量var中。成功时返回 0，失败时返回 -1。

PGTYPESnumeric_copy

把一个numeric变量复制到另一个中。

```
int PGTYPESnumeric_copy(numeric *src, numeric *dst);
```

这个函数把src指向的变量的值复制到dst指向的变量中。成功时该函数返回 0，出错时返回 -1。

PGTYPESnumeric_from_double

将一个 double 类型的变量转换为 numeric 变量。

```
int  PGTYPESnumeric_from_double(double d, numeric *dst);
```

该函数接收一个 double 类型的变量，并将结果存入dst指向的变量中。成功时返回 0，出错时返回 -1。

PGTYPESnumeric_to_double

将一个 numeric 类型的变量转换为 double。

```
int PGTYPESnumeric_to_double(numeric *nv, double *dp)
```

该函数将nv指向的变量中的 numeric 值转换为 double，并存入dp指向的变量中。成功时返回 0，发生错误（包括溢出）时返回 -1。发生溢出时，还会将全局变量errno设为PGTYPES_NUM_OVERFLOW。

PGTYPESnumeric_to_int

将一个 numeric 类型的变量转换为 int。

```
int PGTYPESnumeric_to_int(numeric *nv, int *ip);
```

该函数将nv指向的变量中的 numeric 值转换为 int，并存入ip指向的变量中。成功时返回 0，发生错误（包括溢出）时返回 -1。发生溢出时，还会将全局变量errno设为PGTYPES_NUM_OVERFLOW。

PGTYPESnumeric_to_long

将一个 numeric 类型的变量转换为 long。

```
int PGTYPESnumeric_to_long(numeric *nv, long *lp);
```

该函数将 nv 所指变量中的 numeric 值转换为 long，并存入 lp 所指的变量。成功时返回 0，发生错误（包括溢出）时返回 -1。发生溢出时，还会将全局变量 errno 设为 PGTYPES_NUM_OVERFLOW。

PGTYPESnumeric_to_decimal

将一个 numeric 类型的变量转换为 decimal。

```
int PGTYPESnumeric_to_decimal(numeric *src, decimal *dst);
```

该函数将src指向的变量中的 numeric 值转换成dst指向的 decimal 变量。成功时该函数返回 0，出错时返回 -1，包括溢出。发生溢出时，全局变量errno会被设置为PGTYPES_NUM_OVERFLOW。

PGTYPESnumeric_from_decimal

将一个 decimal 类型的变量转换为 numeric。

```
int PGTYPESnumeric_from_decimal(decimal *src, numeric *dst);
```

该函数将src指向的变量中的 decimal 值转换成dst指向的 numeric 变量。成功时该函数返回 0，出错时返回 -1。由于 decimal 类型被实现为 numeric 类型的一个受限版本，这种转换不会发生溢出。

33.6.2. 日期类型

C 中的 date 类型让程序能够处理 SQL date 类型的数据。PostgreSQL服务器中的等效类型见第 8.5 节。

下列函数可用于处理 date 类型：

PGTYPESdate_from_timestamp

从一个时间戳中抽取日期部分。

```
date PGTYPESdate_from_timestamp(timestamp dt);
```

该函数接收一个时间戳作为它的唯一参数并且从这个时间戳返回抽取的日期部分。

PGTYPESdate_from_asc

从日期的文本表示解析一个日期。

```
date PGTYPESdate_from_asc(char *str, char **endptr);
```

该函数接收一个 C char* 字符串str以及一个指向 C char* 字符串的指针endptr。目前 ECPG 始终解析整个字符串，因此尚不支持将第一个无效字符的地址存入*endptr中。可以放心地将endptr设为 NULL。

注意，该函数始终假定日期采用 MDY 格式，目前 ECPG 中没有变量可以改变这一点。

表 33.2展示了所有允许的输入格式。

表 33.2. PGTYPESdate_from_asc的合法输入格式

Input	Result
January 8, 1999	January 8, 1999
1999-01-08	January 8, 1999
1/8/1999	January 8, 1999

Input	Result
1/18/1999	January 18, 1999
01/02/03	February 1, 2003
1999-Jan-08	January 8, 1999
Jan-08-1999	January 8, 1999
08-Jan-1999	January 8, 1999
99-Jan-08	January 8, 1999
08-Jan-99	January 8, 1999
08-Jan-06	January 8, 2006
Jan-08-99	January 8, 1999
19990108	ISO 8601; January 8, 1999
990108	ISO 8601; January 8, 1999
1999.008	year and day of year
J2451187	Julian day
January 8, 99 BC	year 99 before the Common Era

`PGTYPESdate_to_asc`

返回一个日期变量的文本表示。

```
char *PGTYPESdate_to_asc(date dDate);
```

该函数接收日期`dDate`作为它的唯一参数。它将以形式`1999-01-18`输出该日期，即以`YYYY-MM-DD`格式输出。

`PGTYPESdate_julmdy`

从一个 date 类型的变量中提取日、月和年的值。

```
void PGTYPESdate_julmdy(date d, int *mdy);
```

该函数接收日期`d`以及一个指向有 3 个整数值的数组`mdy`的指针。变量名就表明了顺序：`mdy[0]`将被设置为包含月份，`mdy[1]`将被设置为日的值，而`mdy[2]`将包含年。

`PGTYPESdate_mdyjul`

根据一个包含 3 个整数的数组创建日期值，这些整数指定日期的日、月和年。

```
void PGTYPESdate_mdyjul(int *mdy, date *jdate);
```

该函数的第一个参数是包含 3 个整数的数组（`mdy`），第二个参数是一个指向 date 类型变量的指针，用来保存操作结果。

`PGTYPESdate_dayofweek`

为一个日期值返回表示它是星期几的数字。

```
int PGTYPESdate_dayofweek(date d);
```

这个函数接收日期变量`d`作为它唯一的参数并且返回一个整数说明这个日期是星期几。

- 0 - 星期日
- 1 - 星期一
- 2 - 星期二
- 3 - 星期三

- 4 - 星期四
- 5 - 星期五
- 6 - 星期六

`PGTYPESdate_today`

得到当前日期。

```
void PGTYPESdate_today(date *d);
```

该函数接收一个指向 date 变量的指针（`d`），并将其指向的变量设为当前日期。

`PGTYPESdate_fmt_asc`

使用格式掩码将一个 date 类型的变量转换为文本表示。

```
int PGTYPESdate_fmt_asc(date dDate, char *fmtstring, char
 *outbuf);
```

该函数接收要转换的日期（`dDate`）、格式掩码（`fmtstring`）
以及将要保存日期的文本表示的字符串（`outbuf`）。

成功时，返回 0；如果发生错误，则返回一个负值。

下列字面量是可用的字段说明符：

- `dd` - 一个月中的第几天。
- `mm` - 一年中的第几个月。
- `yy` - 两位数的年份。
- `yyyy` - 四位数的年份。
- `ddd` - 星期几的名称（简写）。
- `mmm` - 月份的名称（简写）。

All other characters are copied 1:1 to the output string.

表 33.3给出了一些可用格式，有助于理解此函数的用法。所有输出行都基于同一个日期：1959 年 11 月 23 日。

表 33.3. PGTYPESdate_fmt_asc的合法输入格式

Format	Result
mmddyy	112359
ddmmyy	231159
yymmdd	591123
yy/mm/dd	59/11/23
yy mm dd	59 11 23
yy.mm.dd	59.11.23
.mm.yyyy.dd.	.11.1959.23.
mmm. dd, yyyy	Nov. 23, 1959
mmm dd yyyy	Nov 23 1959
yyyy dd mm	1959 23 11
ddd, mmm. dd, yyyy	Mon, Nov. 23, 1959
(ddd) mmm. dd, yyyy	(Mon) Nov. 23, 1959

`PGTYPESdate_defmt_asc`

使用格式掩码把一个 C `char*` 字符串转换成 date 类型的值。

```
int PGTYPESdate_defmt_asc(date *d, char *fmt, char *str);
```

该函数接收一个指向用于保存操作结果的 date 值的指针（`d`）、用于解析日期的格式掩码（`fmt`），以及包含日期文本表示的 C char* 字符串（`str`）。文本表示应与格式掩码匹配，但字符串与格式掩码无需一一对应。该函数只分析先后顺序，
并查找表示年份位置的`yy`或`yyyy`、表示月份位置的`mm`以及表示日位置的`dd`。

表 33.4给出了一些可用格式，有助于理解此函数的用法。

表 33.4. rdefmtdate的合法输入格式

Format	String	Result
ddmmyy	21-2-54	1954-02-21
ddmmyy	2-12-54	1954-12-02
ddmmyy	20111954	1954-11-20
ddmmyy	130464	1964-04-13
mmm.dd.yyyy	MAR-12-1967	1967-03-12
yy/mm/dd	1954, February 3rd	1954-02-03
mmm.dd.yyyy	041269	1969-04-12
yy/mm/dd	In the year 2525, in the month of July, mankind will be alive on the 28th day	2525-07-28
dd-mm-yy	I said on the 28th of July in the year 2525	2525-07-28
mmm.dd.yyyy	9/14/58	1958-09-14
yy/mm/dd	47/03/29	1947-03-29
mmm.dd.yyyy	oct 28 1975	1975-10-28
mmddyy	Nov 14th, 1985	1985-11-14

33.6.3. 时间戳类型

C 中的 timestamp 类型让程序能够处理 SQL timestamp 类型的数据。PostgreSQL服务器中的等效类型见第 8.5 节。

下列函数可用于处理 timestamp 类型：

`PGTYPEStimestamp_from_asc`

将时间戳的文本表示解析为 timestamp 变量。

```
timestamp PGTYPEStimestamp_from_asc(char *str, char **endptr);
```

该函数接收要解析的字符串（`str`）和一个指向 C char* 的指针（`endptr`）。目前 ECPG 始终解析整个字符串，因此尚不支持将第一个无效字符的地址存入`*endptr`中。可以放心地将`endptr`设为 NULL。

该函数在成功时返回解析后的时间戳。出错时返回`PGTYPESInvalidTimestamp`，并将`errno`设置为`PGTYPES_TS_BAD_TIMESTAMP`。关于该值的重要说明，
见`PGTYPESInvalidTimestamp`。

通常，输入字符串可以包含允许的日期规范、空白字符和允许的时间规范的任意组合。请注意，ECPG 不支持时区。它可以解析时区，但不像PostgreSQL服务器那样应用任何计算。时区标识符会被静默丢弃。

表 33.5包含了一些输入字符串的示例。

表 33.5. PGTYPEStimestamp_from_asc的合法输入格式

Input	Result
1999-01-08 04:05:06	1999-01-08 04:05:06
January 8 04:05:06 1999 PST	1999-01-08 04:05:06
1999-Jan-08 04:05:06.789-8	1999-01-08 04:05:06.789 (time zone specifier ignored)
J2451187 04:05-08:00	1999-01-08 04:05:00 (time zone specifier ignored)

PGTYPEStimestamp_to_asc

将时间戳转换为 C char* 字符串。

```
char *PGTYPEStimestamp_to_asc(timestamp tstamp);
```

该函数接收时间戳tstamp作为其唯一参数，
并返回一个包含时间戳文本表示的已分配字符串。

PGTYPEStimestamp_current

检索当前时间戳。

```
void PGTYPEStimestamp_current(timestamp *ts);
```

该函数检索当前时间戳，并将其保存到ts指向的时间戳变量中。

PGTYPEStimestamp_fmt_asc

使用格式掩码将时间戳变量转换为 C char*。

```
int PGTYPEStimestamp_fmt_asc(timestamp *ts, char *output, int
 str_len, char *fmtstr);
```

该函数接收一个指向要转换的时间戳的指针作为第一个参数（ts），
一个指向输出缓冲区的指针（output），已为输出缓冲区分配的最大长度（str_len），以及用于转换的格式掩码（fmtstr）。

成功时返回 0，发生错误时返回负值。

格式掩码可以使用以下格式说明符。它们与libc中strftime
函数使用的格式说明符相同。不是格式说明符的内容都会被复制到输出缓冲区中。

- %A - 替换为完整星期名称的本地化表示。
- %a - 替换为缩写星期名称的本地化表示。
- %B - 替换为完整月份名称的本地化表示。
- %b - 替换为缩写月份名称的本地化表示。
- %C - 替换为年份除以 100 的结果，以十进制数表示；一位数前补零。
- %c - 替换为时间和日期的本地化表示。
- %D - 等同于 %m/%d/%y。

- `%d` - 替换为一个月中的日，以十进制数表示（01-31）。
- `%E* %O*` - POSIX 区域设置扩展。序列 `%Ec %EC %Ex %EX %Ey %EY %Od %Oe %OH %OI %Om %OM %OS %Ou %OU %OV %Ow %OW %Oy` 应该提供替代表示。

  此外，`%OB`提供月份名称的替代形式，用于单独显示月份、不提及具体日的场合。
- `%e` - 替换为一个月中的日，以十进制数表示（1-31）；一位数前补空格。
- `%F` - 等同于`%Y-%m-%d`。
- `%G` - 替换为包含世纪的年份，以十进制数表示。该年份是这一周中多数天所在的年份（以星期一为每周第一天）。
- `%g` - 替换为与 `%G` 相同的年份，但以不带世纪的十进制数表示（00-99）。
- `%H` - 替换为小时（24 小时制），以十进制数表示（00-23）。
- `%h` - 与`%b`相同。
- `%I` - 替换为小时（12 小时制），以十进制数表示（01-12）。
- `%j` - 替换为一年中的第几天，以十进制数表示（001-366）。
- `%k` - 替换为小时（24 小时制），以十进制数表示（0-23）；一位数前补空格。
- `%l` - 替换为小时（12 小时制），以十进制数表示（1-12）；一位数前补空格。
- `%M` - 替换为分钟，以十进制数表示（00-59）。
- `%m` - 替换为月份，以十进制数表示（01-12）。
- `%n` - 被换行符替换。
- `%O*` - 与`%E*`相同。
- `%p` - 按需要替换为“上午”或“下午”的本地化表示。
- `%R` - 等同于`%H:%M`。
- `%r` - 等同于`%I:%M:%S %p`。
- `%S` - 替换为秒，以十进制数表示（00-60）。
- `%s` - 替换为从 UTC 纪元起经过的秒数。
- `%T` - 等同于`%H:%M:%S`
- `%t` - 被替换为一个制表符。
- `%U` - 替换为一年中的周数，以星期日为每周第一天，以十进制数表示（00-53）。
- `%u` - 替换为星期几，以星期一为每周第一天，以十进制数表示（1-7）。
- `%V` - 替换为一年中的周数，以星期一为每周第一天，以十进制数表示（01-53）。如果包含 1 月 1 日的那一周有四天或更多天属于新年，该周就是第 1 周；否则，该周是上一年的最后一周，下一周才是第 1 周。
- `%v` - 等同于 `%e-%b-%Y`。
- `%W` - 替换为一年中的周数，以星期一为每周第一天，以十进制数表示（00-53）。
- `%w` - 替换为星期几，以星期日为每周第一天，以十进制数表示（0-6）。

- `%X` - 被时间的本地化表示替换。
- `%x` - 替换为日期的本地化表示。
- `%Y` - 替换为包含世纪的年份，以十进制数表示。
- `%y` - 替换为不带世纪的年份，以十进制数表示（00-99）。
- `%Z` - 被时区名称替换。
- `%z` - 替换为时区相对于 UTC 的偏移量；前导加号表示 UTC 以东，减号表示 UTC 以西，后接小时和分钟，各占两位数字，中间没有分隔符（RFC 822日期头的常见形式）。
- `%+` - 被日期和时间的本地化表示所替换。
- `%-*` - GNU libc扩展。在执行数字输出时不进行任何填充。
- $_* - GNU libc扩展。明确指定填充空间。
- `%0*` - GNU libc 扩展。显式指定用零填充。
- `%%` - 被`%`替换。

`PGTYPEStimestamp_sub`

从一个时间戳减去另一个时间戳，并将结果保存在 interval 类型变量中。

```
int PGTYPEStimestamp_sub(timestamp *ts1, timestamp *ts2,
 interval *iv);
```

该函数从`ts1`指向的时间戳变量中减去`ts2`指向的时间戳变量，并将结果存入`iv`指向的 interval 变量中。

成功时返回 0，发生错误时返回负值。

`PGTYPEStimestamp_defmt_asc`

使用格式掩码从文本表示中解析时间戳值。

```
int PGTYPEStimestamp_defmt_asc(char *str, char *fmt, timestamp
 *d);
```

该函数接收变量`str`中的时间戳文本表示，以及变量`fmt`中的格式掩码。结果存入`d`指向的变量中。

如果格式掩码`fmt`为 NULL，函数将退回到默认格式掩码，即`%Y-%m-%d %H:%M:%S`。

这是`PGTYPEStimestamp_fmt_asc`的逆函数。可用的格式掩码项见该函数的说明。

`PGTYPEStimestamp_add_interval`

向时间戳变量加上一个 interval 变量。

```
int PGTYPEStimestamp_add_interval(timestamp *tin, interval
 *span, timestamp *tout);
```

该函数接收一个指向 timestamp 变量`tin`的指针，以及一个指向 interval 变量`span`的指针。它将时间间隔加到时间戳上，并将得到的时间戳存入`tout`指向的变量中。

成功时返回 0，发生错误时返回负值。

`PGTYPEStimestamp_sub_interval`

从时间戳变量中减去一个 interval 变量。

```
int PGTYPEStimestamp_sub_interval(timestamp *tin, interval
 *span, timestamp *tout);
```

该函数从tin指向的时间戳变量中减去span指向的时间间隔变量，并将结果存入tout指向的变量中。

成功时返回 0，发生错误时返回负值。

33.6.4. interval 类型

C 中的 interval 类型允许程序处理 SQL interval 类型的数据。PostgreSQL服务器中的等效类型见第 8.5 节。

下列函数可用于处理 interval 类型：

PGTYPESinterval_new

返回一个指向新分配 interval 变量的指针。

```
interval *PGTYPESinterval_new(void);
```

PGTYPESinterval_free

释放先前分配的 interval 变量所占用的内存。

```
void PGTYPESinterval_new(interval *intvl);
```

PGTYPESinterval_from_asc

从文本表示中解析一个 interval 值。

```
interval *PGTYPESinterval_from_asc(char *str, char **endptr);
```

该函数解析输入字符串str，并返回一个已分配 interval 变量的指针。目前 ECPG 总是解析整个字符串，因此暂不支持把第一个非法字符的地址存储在*endptr中。
可以安全地把endptr设为 NULL。

PGTYPESinterval_to_asc

将 interval 类型变量转换成文本表示。

```
char *PGTYPESinterval_to_asc(interval *span);
```

该函数把span指向的 interval 变量转换成 C char*。输出形式类似于：`@ 1 day 12 hours 59 mins 10 secs`。

PGTYPESinterval_copy

复制一个 interval 类型变量。

```
int PGTYPESinterval_copy(interval *intvlsrc, interval
 *intvldest);
```

该函数把intvlsrc指向的 interval 变量复制到intvldest指向的 interval 变量中。注意，需要事先为目标变量分配好内存。

33.6.5. decimal类型

decimal类型和numeric类型相似。不过，它被限制为最大精度是 30 个有效位。与numeric类型只能在堆上创建相反，decimal类型既可以在栈上也可以在堆上创建（使用函数PGTYPESdecimal_new 和PGTYPESdecimal_free）。在第 33.15 节中描述的Informix兼容模式中有很多其它函数可以处理decimal类型。

下列函数可用于处理 decimal 类型，并非仅在`libcompat`库中提供。

`PGTYPESdecimal_new`

要求一个指向新分配的decimal变量的指针。

```
decimal *PGTYPESdecimal_new(void);
```

`PGTYPESdecimal_free`

释放一个 decimal 变量及其占用的全部内存。

```
void PGTYPESdecimal_free(decimal *var);
```

33.6.6. pgtypeslib 的 errno 值

`PGTYPES_NUM_BAD_NUMERIC`

一个参数应该包含一个numeric变量（或者指向一个numeric变量），但是实际上它的内存表示非法。

`PGTYPES_NUM_OVERFLOW`

发生一次溢出。由于numeric类型可以处理几乎任何精度，将一个numeric变量转换成其他类型可能导致溢出。

`PGTYPES_NUM_UNDERFLOW`

发生一次下溢。由于numeric类型可以处理几乎任何精度，将一个numeric变量转换成其他类型可能导致下溢。

`PGTYPES_NUM_DIVIDE_ZERO`

尝试了一次除零。

`PGTYPES_DATE_BAD_DATE`

一个非法的日期字符串被传给了`PGTYPESdate_from_asc`函数。

`PGTYPES_DATE_ERR_EARGS`

非法参数被传给了`PGTYPESdate_defmt_asc`函数。

`PGTYPES_DATE_ERR_ENOSHORTDATE`

`PGTYPESdate_defmt_asc`函数在输入字符串中发现了一个非法词元。

`PGTYPES_INTVL_BAD_INTERVAL`

向`PGTYPESinterval_from_asc`函数传入了无效的 interval 字符串，或者向`PGTYPESinterval_to_asc`函数传入了无效的 interval 值。

`PGTYPES_DATE_ERR_ENOTDMY`

`PGTYPESdate_defmt_asc`函数中的日、月、年赋值不匹配。

`PGTYPES_DATE_BAD_DAY`

`PGTYPESdate_defmt_asc`函数发现了月中的一个非法日值。

`PGTYPES_DATE_BAD_MONTH`

`PGTYPESdate_defmt_asc`函数发现了一个非法的月值。

`PGTYPES_TS_BAD_TIMESTAMP`

一个非法的时间戳字符串被传给了`PGTYPEStimestamp_from_asc`函数，或者一个非法的时间戳值被传给了`PGTYPEStimestamp_to_asc`函数。

`PGTYPES_TS_ERR_EINFTIME`

在一个无法处理无限时间戳值的环境中遇到了这样一个值。

33.6.7. pgtypeslib 的特殊常量

`PGTYPESInvalidTimestamp`

一个表示无效时间戳的 timestamp 类型值。`PGTYPEStimestamp_from_asc`函数在解析出错时返回此值。注意，由于`timestamp`数据类型的内部表示，`PGTYPESInvalidTimestamp`同时也是一个有效的时间戳，其值为`1899-12-31 23:59:59`。为了检测错误，应用程序每次调用`PGTYPEStimestamp_from_asc`后，不仅要检查`PGTYPESInvalidTimestamp`，还要检查`errno != 0`。

33.7. 使用描述符区域

SQL 描述符区域是一种处理`SELECT`、`FETCH`或`DESCRIBE`语句结果的高级方法。SQL 描述符区域将一行数据及其元数据项组合到一个数据结构中。在执行动态 SQL 语句时，结果列的性质可能无法提前获知，此时元数据特别有用。PostgreSQL 提供两种使用描述符区域的方式：命名 SQL 描述符区域和 C 结构体 SQLDA。

33.7.1. 命名 SQL 描述符区域

一个命名 SQL 描述符区域由一个头部以及一个或多个条目描述符区域构成，头部包含与整个描述符相关的信息，而条目描述符区域则描述结果行中的每一列。

在使用 SQL 描述符区域之前，需要先分配一个：

```
EXEC SQL ALLOCATE DESCRIPTOR identifier;
```

identifier 会作为该描述符区域的“变量名”。当不再需要该描述符时，应当释放它：

```
EXEC SQL DEALLOCATE DESCRIPTOR identifier;
```

要使用一个描述符区域，把它指定为`INTO`子句的存储目标（而不是列出主变量）：

```
EXEC SQL FETCH NEXT FROM mycursor INTO SQL DESCRIPTOR mydesc;
```

如果结果集为空，该描述符区域仍然会包含查询的元数据，即字段的名称。

对于还没有执行的预备查询，`DESCRIBE`可以被用来得到其结果集的元数据：

```
EXEC SQL BEGIN DECLARE SECTION;
char *sql_stmt = "SELECT * FROM table1";
EXEC SQL END DECLARE SECTION;

EXEC SQL PREPARE stmt1 FROM :sql_stmt;
EXEC SQL DESCRIBE stmt1 INTO SQL DESCRIPTOR mydesc;
```

在 PostgreSQL 9.0 之前，`SQL`关键词是可选的，因此使用`DESCRIPTOR`和`SQL DESCRIPTOR`都会产生命名 SQL 描述符区域。现在该关键词是强制性的，省略`SQL`关键词会产生 SQLDA 描述符区域（见第 33.7.2 节）。

在`DESCRIBE`和`FETCH`语句中，`INTO`和`USING`关键词的用法相似：它们在描述符区域中生成结果集和元数据。

要从描述符区域取得数据，可以将它视为一个包含命名字段的结构。使用以下命令，从头部取得某个字段的值，并存入主变量：

```
EXEC SQL GET DESCRIPTOR name :hostvar = field;
```

目前只定义了一个头部字段：*COUNT*，它表示有多少个条目描述符区域（即结果包含多少列）。主变量必须是整数类型。要从条目描述符区域取得字段，请使用以下命令：

```
EXEC SQL GET DESCRIPTOR name VALUE num :hostvar = field;
```

*num*可以是整数字面值，也可以是包含整数的主变量。可用字段如下：

CARDINALITY（整数）

结果集中的行数

DATA

实际的数据项（因此，这个字段的数据类型取决于查询）

DATETIME_INTERVAL_CODE（整数）

当TYPE是9时，DATETIME_INTERVAL_CODE将具有以下值之一：1 表示 DATE，2 表示 TIME，3 表示 TIMESTAMP，4 表示 TIME WITH TIME ZONE，5 表示 TIMESTAMP WITH TIME ZONE。

DATETIME_INTERVAL_PRECISION（整数）

没有实现

INDICATOR（整数）

指示符（表示一个空值或者一个值截断）

KEY_MEMBER（整数）

没有实现

LENGTH（整数）

以字符计的数据长度

NAME（字符串）

列名

NULLABLE（整数）

没有实现

OCTET_LENGTH（整数）

以字节计的数据字符表示的长度

PRECISION（整数）

精度（用于类型numeric）

RETURNED_LENGTH（整数）

以字符计的数据长度

RETURNED_OCTET_LENGTH（整数）

以字节计的数据字符表示的长度

SCALE（整数）

小数位数（用于类型 numeric）

TYPE（整数）

列的数据类型的数字编码

在EXECUTE、DECLARE以及OPEN语句中，INTO和USING关键词的效果不同。
也可以手工建立一个描述符区域来为一个查询或者游标提供输入参数，并且USING SQL DESCRIPTOR *name*是用来传递输入参数给参数化查询的方法。建立一个命名 SQL 描述符区域的语句如下：

```
EXEC SQL SET DESCRIPTOR name VALUE num field = :hostvar;
```

PostgreSQL 支持在一条FETCH语句中检索多条记录。在这种情况下，要将数据存入主变量，该变量必须是数组。例如：

```
EXEC SQL BEGIN DECLARE SECTION;
int id[5];
EXEC SQL END DECLARE SECTION;

EXEC SQL FETCH 5 FROM mycursor INTO SQL DESCRIPTOR mydesc;

EXEC SQL GET DESCRIPTOR mydesc VALUE 1 :id = DATA;
```

33.7.2. SQLDA 描述符区域

SQLDA 描述符区域是一个 C 语言结构体，它也能被用来得到一个查询的结果集和元数据。一个结构体存储一个来自结果集的记录。

```
EXEC SQL include sqlda.h;
sqlda_t         *mysqlda;

EXEC SQL FETCH 3 FROM mycursor INTO DESCRIPTOR mysqlda;
```

注意，这里省略了SQL关键词。第 33.7.1 节中关于INTO和USING关键词用法的说明也适用于这里，并有一项补充规则：在DESCRIBE语句中，如果使用了INTO关键词，就可以完全省略DESCRIPTOR关键词：

```
EXEC SQL DESCRIBE prepared_statement INTO mysqlda;
```

使用 SQLDA 的程序的一般流程是：

1. 准备一个查询，并且为它声明一个游标。
2. 为结果行声明一个 SQLDA。
3. 为输入参数声明一个 SQLDA，并且初始化它们（内存分配、参数设置）。
4. 用输入 SQLDA 打开一个游标。
5. 从游标中取得行，并且把它们存储到一个输出 SQLDA。
6. 从输出 SQLDA 读取值到主变量中（必要时使用转换）。
7. 关闭游标。

8. 关闭为输入 SQLDA 分配的内存区域。

33.7.2.1. SQLDA 数据结构

SQLDA 使用三种数据结构类型：`sqlda_t`、`sqlvar_t`以及`struct sqlname`。

> **提示**
>
> PostgreSQL 的 SQLDA 数据结构与 IBM DB2 Universal Database 中的 SQLDA 相似，因此一些有关 DB2 SQLDA 的技术资料有助于更好地理解 PostgreSQL 的 SQLDA。

33.7.2.1.1. sqlda_t 结构体

结构体类型`sqlda_t`是实际 SQLDA 的类型。它保存一个记录。并且两个或者更多个`sqlda_t`结构体能够以`desc_next`字段中的指针连接成一个链表，
这样可以表示一个有序的行集合。因此，当两个或多个行被取得时，
应用可以通过沿着每一个`sqlda_t`节点中的`desc_next`指针读取它们。

`sqlda_t`的定义是：

```
struct sqlda_struct
{
    char            sqldaid[8];
    long            sqldabc;
    short           sqln;
    short           sqld;
    struct sqlda_struct *desc_next;
    struct sqlvar_struct sqlvar[1];
};

typedef struct sqlda_struct sqlda_t;
```

各字段的含义如下：

`sqldaid`

它包含一个字符串"`SQLDA `"。

`sqldabc`

它包含已分配空间的尺寸（以字节计）。

`sqln`

当它通过`USING`关键词传递给`OPEN`、`DECLARE`或`EXECUTE`语句时，
它包含参数化查询的输入参数个数。如果它用作`SELECT`、`EXECUTE`或`FETCH`语句的输出，其值与`sqld`相同。

`sqld`

它包含一个结果集中的域的数量。

`desc_next`

如果查询返回不止一个记录，会返回多个链接在一起的 SQLDA 结构体，并且`desc_next`保存一个指向下一个项的指针。

`sqlvar`

这是结果集中列的数组。

33.7.2.1.2. sqlvar_t 结构体

结构体类型sqlvar_t保存一个列值和元数据（例如类型和长度）。该类型的定义是：

```
struct sqlvar_struct
{
    short          sqltype;
    short          sqllen;
    char          *sqldata;
    short         *sqlind;
    struct sqlname sqlname;
};

typedef struct sqlvar_struct sqlvar_t;
```

各字段的含义如下：

sqltype

包含该字段的类型标识符。值可以参考ecpgtype.h中的enum ECPGttype。

sqllen

包含字段的二进制长度，例如ECPGt_int是 4 字节。

sqldata

指向数据。数据的格式在第 33.4.4 节中描述。

sqlind

指向空值指示符。0 表示非空，-1 表示空。

sqlname

域的名称。

33.7.2.1.3. struct sqlname 结构体

一个struct sqlname结构体保存一个列名。它用作sqlvar_t结构体的成员。该结构体的定义如下：

```
#define NAMEDATALEN 64

struct sqlname
{
        short           length;
        char            data[NAMEDATALEN];
};
```

各字段的含义如下：

length

包含字段名称的长度。

data

包含实际的域名称。

33.7.2.2. 使用一个 SQLDA 检索一个结果集

通过一个 SQLDA 检索一个查询结果集的一般步骤是：

1. 声明一个sqlda_t结构体来接收结果集。
2. 执行 FETCH/EXECUTE/DESCRIBE 命令来处理一个指定已声明 SQLDA 的查询。
3. 通过查看sqlda_t结构体的成员sqln来检查结果集中记录的数量。
4. 从sqlda_t结构体的成员sqlvar[0]、sqlvar[1]等中得到每一列的值。
5. 沿着sqlda_t结构体的成员desc_next指针到达下一行（sqlda_t）。
6. 根据你的需要重复上述步骤。

这里是一个通过 SQLDA 检索结果集的例子。

首先，声明一个sqlda_t结构体来接收结果集。

```
sqlda_t *sqlda1;
```

接下来，在命令中指定 SQLDA。下面是一个FETCH命令的例子。

```
EXEC SQL FETCH NEXT FROM cur1 INTO DESCRIPTOR sqlda1;
```

运行一个循环顺着链表来检索行。

```
sqlda_t *cur_sqlda;

for (cur_sqlda = sqlda1;
     cur_sqlda != NULL;
     cur_sqlda = cur_sqlda->desc_next)
{
    ...
}
```

在循环内部，运行另一个循环来检索行中每一列的数据（sqlvar_t结构体）。

```
for (i = 0; i < cur_sqlda->sqld; i++)
{
    sqlvar_t v = cur_sqlda->sqlvar[i];
    char *sqldata = v.sqldata;
    short sqllen  = v.sqllen;
    ...
}
```

要得到一列的值，应检查sqlvar_t结构体的成员sqltype的值。然后，
根据列类型切换到一种合适的方法从sqlvar字段中复制数据到一个主变量。

```
char var_buf[1024];

switch (v.sqltype)
{
    case ECPGt_char:
        memset(&var_buf, 0, sizeof(var_buf));
        memcpy(&var_buf, sqldata, (sizeof(var_buf) <= sqllen ?
 sizeof(var_buf) - 1 : sqllen));
        break;

    case ECPGt_int: /* integer */
        memcpy(&intval, sqldata, sqllen);
        snprintf(var_buf, sizeof(var_buf), "%d", intval);
        break;
```

```
    ...
}
```

33.7.2.3. 使用一个 SQLDA 传递查询参数

使用一个 SQLDA 传递输入参数给一个预备查询的一般步骤是：

1. 创建一个预备查询（预备语句）。
2. 声明一个 sqlda_t 结构，作为输入 SQLDA。
3. 为输入 SQLDA 分配内存区域（作为 sqlda_t 结构体）。
4. 在分配好的内存中设置（复制）输入值。
5. 打开一个说明了输入 SQLDA 的游标。

这里是一个例子。

首先，创建一个预备语句。

```
EXEC SQL BEGIN DECLARE SECTION;
char query[1024] = "SELECT d.oid, * FROM pg_database d, pg_stat_
database s WHERE d.oid = s.datid AND (d.datname = ? OR d.oid = ?)";
EXEC SQL END DECLARE SECTION;

EXEC SQL PREPARE stmt1 FROM :query;
```

接下来为一个 SQLDA 分配内存，并且在`sqlda_t`结构体的`sqln`成员变量中设置输入参数的数量。当预备查询要求两个或多个输入参数时，应用必须分配额外的内存空间，空间的大小为 (参数数目 - 1) * sizeof(sqlvar_t)。这里的例子展示了为两个输入参数分配内存空间。

```
sqlda_t *sqlda2;

sqlda2 = (sqlda_t *) malloc(sizeof(sqlda_t) + sizeof(sqlvar_t));
memset(sqlda2, 0, sizeof(sqlda_t) + sizeof(sqlvar_t));

sqlda2->sqln = 2; /* 输入变量的数目 */
```

内存分配之后，把参数值存储到`sqlvar[]`数组（当 SQLDA 在接收结果集时，这也是用来检索列值的数组）。在这个例子中，输入参数是`"postgres"`（字符串类型）和`1`（整数类型）。

```
sqlda2->sqlvar[0].sqltype = ECPGt_char;
sqlda2->sqlvar[0].sqldata = "postgres";
sqlda2->sqlvar[0].sqllen  = 8;

int intval = 1;
sqlda2->sqlvar[1].sqltype = ECPGt_int;
sqlda2->sqlvar[1].sqldata = (char *) &intval;
sqlda2->sqlvar[1].sqllen  = sizeof(intval);
```

打开游标并指定事先设置好的 SQLDA，即可将输入参数传递给预备语句。

```
EXEC SQL OPEN cur1 USING DESCRIPTOR sqlda2;
```

最后，用完输入 SQLDA 后必须显式地释放已分配的内存空间，这与用于接收查询结果的 SQLDA 不同。

```
free(sqlda2);
```

33.7.2.4. 一个使用 SQLDA 的应用例子

这里是一个例子程序，
它描述了如何按照输入参数的指定从系统目录中取得数据库的访问统计。

这个应用在数据库 OID 上连接两个系统表（pg_database 和 pg_stat_database），并且还取得和显示通过两个输入参数（一个数据库postgres和 OID 1）检索到的数据库统计。

首先，为输入和输出分别声明一个 SQLDA。

```
EXEC SQL include sqlda.h;

sqlda_t *sqlda1; /* 一个输出描述符 */
sqlda_t *sqlda2; /* 一个输入描述符  */
```

接下来，连接到数据库，预备一个语句，并为该预备语句声明游标。

```
int
main(void)
{
    EXEC SQL BEGIN DECLARE SECTION;
    char query[1024] = "SELECT d.oid,* FROM pg_database d, pg_stat_
database s WHERE d.oid=s.datid AND ( d.datname=? OR d.oid=? )";
    EXEC SQL END DECLARE SECTION;

    EXEC SQL CONNECT TO testdb AS con1 USER testuser;

    EXEC SQL PREPARE stmt1 FROM :query;
    EXEC SQL DECLARE cur1 CURSOR FOR stmt1;
```

然后，为输入参数在输入 SQLDA 中放入一些值。为输入 SQLDA 分配内存，并且在sqln中设置输入参数的数目。在sqlvar结构体的sqltype、sqldata和sqllen中存入类型、
值和值长度。

```
    /* 为输入参数创建 SQLDA 结构。 */
    sqlda2 = (sqlda_t *) malloc(sizeof(sqlda_t) + sizeof(sqlvar_t))
;
    memset(sqlda2, 0, sizeof(sqlda_t) + sizeof(sqlvar_t));
    sqlda2->sqln = 2; /* 输入变量的数量 */

    sqlda2->sqlvar[0].sqltype = ECPGt_char;
    sqlda2->sqlvar[0].sqldata = "postgres";
    sqlda2->sqlvar[0].sqllen  = 8;

    intval = 1;
    sqlda2->sqlvar[1].sqltype = ECPGt_int;
    sqlda2->sqlvar[1].sqldata = (char *)&intval;
    sqlda2->sqlvar[1].sqllen  = sizeof(intval);
```

设置完输入 SQLDA 之后，用输入 SQLDA 打开一个游标。

```
    /* 用输入参数打开一个游标。 */
    EXEC SQL OPEN cur1 USING DESCRIPTOR sqlda2;
```

从打开的游标中取得行，并存入输出 SQLDA（通常，需要在循环中反复调用FETCH，才能取出结果集中的所有行）。

```
    while (1)
    {
        sqlda_t *cur_sqlda;
```

```
        /* 分配描述符给游标  */
        EXEC SQL FETCH NEXT FROM cur1 INTO DESCRIPTOR sqlda1;
```

接下来，沿着sqlda_t结构体的链表，从 SQLDA 中读取已取得的记录。

```
    for (cur_sqlda = sqlda1 ;
         cur_sqlda != NULL ;
         cur_sqlda = cur_sqlda->desc_next)
    {
        ...
```

读取第一个记录中的每一列。列的数量被存储在sqld中，第一列的实际数据被存储在sqlvar[0]中，两者都是sqlda_t结构体的成员。

```
        /* 打印一行中的每一列。 */
        for (i = 0; i < sqlda1->sqld; i++)
        {
            sqlvar_t v = sqlda1->sqlvar[i];
            char *sqldata = v.sqldata;
            short sqllen  = v.sqllen;

            strncpy(name_buf, v.sqlname.data, v.sqlname.length);
            name_buf[v.sqlname.length] = '\0';
```

现在，列数据已存储在变量v中。将每个数据项复制到主变量，并查看v.sqltype来确定列的类型。

```
            switch (v.sqltype) {
                int intval;
                double doubleval;
                unsigned long long int longlongval;

                case ECPGt_char:
                    memset(&var_buf, 0, sizeof(var_buf));
                    memcpy(&var_buf, sqldata, (sizeof(var_buf) <=
 sqllen ? sizeof(var_buf)-1 : sqllen));
                    break;

                case ECPGt_int: /* 整数 */
                    memcpy(&intval, sqldata, sqllen);
                    snprintf(var_buf, sizeof(var_buf), "%d",
 intval);
                    break;

                ...

                default:
                    ...
            }

            printf("%s = %s (type: %d)\n", name_buf, var_buf, v.
sqltype);
        }
```

处理所有记录后关闭游标，并且从数据库断开连接。

```
    EXEC SQL CLOSE cur1;
    EXEC SQL COMMIT;

    EXEC SQL DISCONNECT ALL;
```

整个程序显示在例 33.1中。

例 33.1. 示例 SQLDA 程序

```
#include <stdlib.h>
#include <string.h>
#include <stdlib.h>
#include <stdio.h>
#include <unistd.h>

EXEC SQL include sqlda.h;

sqlda_t *sqlda1; /* descriptor for output */
sqlda_t *sqlda2; /* descriptor for input */

EXEC SQL WHENEVER NOT FOUND DO BREAK;
EXEC SQL WHENEVER SQLERROR STOP;

int
main(void)
{
    EXEC SQL BEGIN DECLARE SECTION;
    char query[1024] = "SELECT d.oid,* FROM pg_database d, pg_stat_
database s WHERE d.oid=s.datid AND ( d.datname=? OR d.oid=? )";

    int intval;
    unsigned long long int longlongval;
    EXEC SQL END DECLARE SECTION;

    EXEC SQL CONNECT TO uptimedb AS con1 USER uptime;

    EXEC SQL PREPARE stmt1 FROM :query;
    EXEC SQL DECLARE cur1 CURSOR FOR stmt1;

    /* Create a SQLDA structure for an input parameter */
    sqlda2 = (sqlda_t *)malloc(sizeof(sqlda_t) + sizeof(sqlvar_t));
    memset(sqlda2, 0, sizeof(sqlda_t) + sizeof(sqlvar_t));
    sqlda2->sqln = 2; /* a number of input variables */

    sqlda2->sqlvar[0].sqltype = ECPGt_char;
    sqlda2->sqlvar[0].sqldata = "postgres";
    sqlda2->sqlvar[0].sqllen  = 8;

    intval = 1;
    sqlda2->sqlvar[1].sqltype = ECPGt_int;
    sqlda2->sqlvar[1].sqldata = (char *) &intval;
    sqlda2->sqlvar[1].sqllen  = sizeof(intval);

    /* Open a cursor with input parameters. */
    EXEC SQL OPEN cur1 USING DESCRIPTOR sqlda2;

    while (1)
    {
        sqlda_t *cur_sqlda;

        /* Assign descriptor to the cursor  */
        EXEC SQL FETCH NEXT FROM cur1 INTO DESCRIPTOR sqlda1;
```

```
        for (cur_sqlda = sqlda1 ;
             cur_sqlda != NULL ;
             cur_sqlda = cur_sqlda->desc_next)
        {
            int i;
            char name_buf[1024];
            char var_buf[1024];

            /* Print every column in a row. */
            for (i=0 ; i<cur_sqlda->sqld ; i++)
            {
                sqlvar_t v = cur_sqlda->sqlvar[i];
                char *sqldata = v.sqldata;
                short sqllen  = v.sqllen;

                strncpy(name_buf, v.sqlname.data, v.sqlname.length)
;
                name_buf[v.sqlname.length] = '\0';

                switch (v.sqltype)
                {
                    case ECPGt_char:
                        memset(&var_buf, 0, sizeof(var_buf));
                        memcpy(&var_buf, sqldata, (sizeof(var_buf)
<=sqllen ? sizeof(var_buf)-1 : sqllen) );
                        break;

                    case ECPGt_int: /* integer */
                        memcpy(&intval, sqldata, sqllen);
                        snprintf(var_buf, sizeof(var_buf), "%d",
 intval);
                        break;

                    case ECPGt_long_long: /* bigint */
                        memcpy(&longlongval, sqldata, sqllen);
                        snprintf(var_buf, sizeof(var_buf), "%lld",
 longlongval);
                        break;

                    default:
                    {
                        int i;
                        memset(var_buf, 0, sizeof(var_buf));
                        for (i = 0; i < sqllen; i++)
                        {
                            char tmpbuf[16];
                            snprintf(tmpbuf, sizeof(tmpbuf), "%02x
", (unsigned char) sqldata[i]);
                            strncat(var_buf, tmpbuf, sizeof(var_
buf));
                        }
                    }
                        break;
                }

                printf("%s = %s (type: %d)\n", name_buf, var_buf,
 v.sqltype);
            }
```

```
            printf("\n");
        }
    }

    EXEC SQL CLOSE cur1;
    EXEC SQL COMMIT;

    EXEC SQL DISCONNECT ALL;

    return 0;
}
```

这个例子的输出应该看起来类似下面的结果（一些数字会变化）。

```
oid = 1 (type: 1)
datname = template1 (type: 1)
datdba = 10 (type: 1)
encoding = 0 (type: 5)
datistemplate = t (type: 1)
datallowconn = t (type: 1)
datconnlimit = -1 (type: 5)
datlastsysoid = 11510 (type: 1)
datfrozenxid = 379 (type: 1)
dattablespace = 1663 (type: 1)
datconfig =  (type: 1)
datacl = {=c/uptime,uptime=CTc/uptime} (type: 1)
datid = 1 (type: 1)
datname = template1 (type: 1)
numbackends = 0 (type: 5)
xact_commit = 113606 (type: 9)
xact_rollback = 0 (type: 9)
blks_read = 130 (type: 9)
blks_hit = 7341714 (type: 9)
tup_returned = 38262679 (type: 9)
tup_fetched = 1836281 (type: 9)
tup_inserted = 0 (type: 9)
tup_updated = 0 (type: 9)
tup_deleted = 0 (type: 9)

oid = 11511 (type: 1)
datname = postgres (type: 1)
datdba = 10 (type: 1)
encoding = 0 (type: 5)
datistemplate = f (type: 1)
datallowconn = t (type: 1)
datconnlimit = -1 (type: 5)
datlastsysoid = 11510 (type: 1)
datfrozenxid = 379 (type: 1)
dattablespace = 1663 (type: 1)
datconfig =  (type: 1)
datacl =  (type: 1)
datid = 11511 (type: 1)
datname = postgres (type: 1)
numbackends = 0 (type: 5)
xact_commit = 221069 (type: 9)
xact_rollback = 18 (type: 9)
blks_read = 1176 (type: 9)
```

```
blks_hit = 13943750 (type: 9)
tup_returned = 77410091 (type: 9)
tup_fetched = 3253694 (type: 9)
tup_inserted = 0 (type: 9)
tup_updated = 0 (type: 9)
tup_deleted = 0 (type: 9)
```

33.8. 错误处理

这一节描述如何在嵌入式 SQL 程序中处理异常情况和警告。为此提供了两种可以同时使用的机制。

- 可以使用`WHENEVER`命令配置回调来处理警告和错误情况。
- 可以从`sqlca`变量中获得错误或警告的详细信息。

33.8.1. 设置回调

捕获错误和警告的一种简单方法是指定一个动作，在特定条件发生时执行。一般形式如下：

EXEC SQL WHENEVER *condition action*;

*condition*可以是以下值之一：

`SQLERROR`

只要在 SQL 语句执行期间发生一个错误就调用指定的动作。

`SQLWARNING`

只要在 SQL 语句执行期间发生一个警告就调用指定的动作。

`NOT FOUND`

只要一个 SQL 语句检索或者影响零行就调用指定的动作（这种情况不是一个错误，但是你可能需要特别地处理它）。

*action*可以是以下值之一：

`CONTINUE`

这实际上表示该情况被忽略。这是默认值。

GOTO *label*
GO TO *label*

跳到指定的标签（使用一个 C `goto`语句）。

`SQLPRINT`

把一个消息打印到标准错误。这对简单程序或原型开发很有用。消息的细节无法配置。

`STOP`

调用`exit(1)`终止程序。

`DO BREAK`

执行 C 语句`break`。只应被用在循环或`switch`语句中。

```
CALL name (args)
DO name (args)
```

使用指定参数调用指定的 C 函数。

SQL 标准只规定了以下动作：`CONTINUE` 和 `GOTO`（以及 `GO TO`）。

这里有一个可能会用在简单程序中的例子。当一个警告发生时它打印一个简单消息，而发生一个错误时它会中止程序：

```
EXEC SQL WHENEVER SQLWARNING SQLPRINT;
EXEC SQL WHENEVER SQLERROR STOP;
```

语句`EXEC SQL WHENEVER`是 SQL 预处理器的指令，而不是 C 语句。它设置的错误或警告动作适用于源代码中出现在处理器设置位置之后的所有嵌入式 SQL 语句，除非在第一个`EXEC SQL WHENEVER`与触发该条件的 SQL 语句之间，为同一条件设置了其他动作。这与 C 程序的控制流无关。因此，下面两个 C 程序片段都不会产生预期效果：

```
/*
 * 错误
 */
int main(int argc, char *argv[])
{
    ...
    if (verbose) {
        EXEC SQL WHENEVER SQLWARNING SQLPRINT;
    }
    ...
    EXEC SQL SELECT ...;
    ...
}

/*
 * 错误
 */
int main(int argc, char *argv[])
{
    ...
    set_error_handler();
    ...
    EXEC SQL SELECT ...;
    ...
}

static void set_error_handler(void)
{
    EXEC SQL WHENEVER SQLERROR STOP;
}
```

33.8.2. sqlca

为了支持更强大的错误处理，嵌入式 SQL 接口提供了一个名为`sqlca`（SQL 通信区域）的全局变量，其结构如下：

```
struct
{
    char sqlcaid[8];
    long sqlabc;
```

```
    long sqlcode;
    struct
    {
        int sqlerrml;
        char sqlerrmc[SQLERRMC_LEN];
    } sqlerrm;
    char sqlerrp[8];
    long sqlerrd[6];
    char sqlwarn[8];
    char sqlstate[5];
} sqlca;
```

（在一个多线程程序中，每一个线程会自动得到它自己的sqlca副本。这和对于标准 C 全局变量errno的处理相似。）

sqlca覆盖了警告和错误。如果执行一个语句时发生了多个警告或错误，那么sqlca将只包含关于最后一个的信息。

如果在上一个SQL语句中没有产生错误，sqlca.sqlcode将为 0 并且sqlca.sqlstate将为"00000"。如果发生一个警告或错误，则sqlca.sqlcode将为负并且sqlca.sqlstate将不为"00000"。一个正的sqlca.sqlcode表示一种无害的情况，例如上一个查询返回零行。sqlcode和sqlstate是两种不同的错误编码方案，详见下文。

如果上一个 SQL 语句成功，则sqlca.sqlerrd[1]包含所处理行的 OID（如果适用），sqlca.sqlerrd[2]包含处理或返回的行数（如果适用于该命令）。

发生错误或警告时，sqlca.sqlerrm.sqlerrmc包含描述该错误的字符串。字段sqlca.sqlerrm.sqlerrml包含存储在sqlca.sqlerrm.sqlerrmc中的错误消息的长度（即strlen()的结果，对于 C 程序员通常没有多少用处）。注意，有些消息太长，无法存入定长的sqlerrmc数组，因此会被截断。

发生警告时，sqlca.sqlwarn[2]被设为W（其他情况下，它被设为非W的值）。如果sqlca.sqlwarn[1]被设为W，说明某个值在存入主变量时被截断了。如果其他任一元素被设置为指示警告，sqlca.sqlwarn[0]就会被设为W。

字段sqlcaid、sqlabc、sqlerrp，以及sqlerrd和sqlwarn的其余元素，目前不包含有用信息。

SQL 标准中没有定义sqlca结构体，但是在一些其他的 SQL 数据库系统中都有实现。在核心上这些定义都相似，但是如果你想要编写可移植的应用，那么你应该仔细研究不同的实现。

这里有一个整合使用WHENEVER和sqlca的例子，当一个错误发生时打印出sqlca的内容。在安装一个更“用户友好”的错误处理器之前，这可能对调试或开发原型应用有用。

```
EXEC SQL WHENEVER SQLERROR CALL print_sqlca();

void
print_sqlca()
{
    fprintf(stderr, "==== sqlca ====\n");
    fprintf(stderr, "sqlcode: %ld\n", sqlca.sqlcode);
    fprintf(stderr, "sqlerrm.sqlerrml: %d\n", sqlca.sqlerrm.
sqlerrml);
    fprintf(stderr, "sqlerrm.sqlerrmc: %s\n", sqlca.sqlerrm.
sqlerrmc);
    fprintf(stderr, "sqlerrd: %ld %ld %ld %ld %ld %ld\n", sqlca.
sqlerrd[0],sqlca.sqlerrd[1],sqlca.sqlerrd[2],
                                                    sqlca.
sqlerrd[3],sqlca.sqlerrd[4],sqlca.sqlerrd[5]);
```

```
    fprintf(stderr, "sqlwarn: %d %d %d %d %d %d %d %d\n", sqlca.
sqlwarn[0], sqlca.sqlwarn[1], sqlca.sqlwarn[2],
                                                          sqlca.
sqlwarn[3], sqlca.sqlwarn[4], sqlca.sqlwarn[5],
                                                          sqlca.
sqlwarn[6], sqlca.sqlwarn[7]);
    fprintf(stderr, "sqlstate: %5s\n", sqlca.sqlstate);
    fprintf(stderr, "===============\n");
}
```

结果看起来像（这里的错误是一个拼写错误的表名）：

```
==== sqlca ====
sqlcode: -400
sqlerrm.sqlerrml: 49
sqlerrm.sqlerrmc: relation "pg_databasep" does not exist on line 38
sqlerrd: 0 0 0 0 0 0
sqlwarn: 0 0 0 0 0 0 0 0
sqlstate: 42P01
===============
```

33.8.3. SQLSTATE 与 SQLCODE

字段`sqlca.sqlstate`和`sqlca.sqlcode`采用两种不同的错误编码方案。两者都源自 SQL 标准，但`SQLCODE`在 SQL-92 标准中已被标记为弃用，并在后续版本中被删除。因此，强烈建议新应用使用`SQLSTATE`。

`SQLSTATE`是一个包含五个字符的数组。这五个字符由数字或大写字母组成，表示各种错误和警告条件的代码。`SQLSTATE`采用分层编码方案：前两个字符表示条件所属的大类，后三个字符表示该大类中的子类。代码`00000`表示成功状态。大部分`SQLSTATE`代码由 SQL 标准定义。PostgreSQL服务器原生支持`SQLSTATE`错误码，因此在所有应用中统一使用这种错误编码方案，可以实现较高的一致性。更多信息见附录 A。

已弃用的错误编码方案`SQLCODE`使用一个简单的整数。0 表示成功，正值表示成功并有附加信息，负值表示错误。SQL 标准只定义了正值 +100，表示上一个命令返回或影响了零行，没有定义具体的负值。因此，这种方案的可移植性较差，也不采用分层编码。历史上，PostgreSQL的嵌入式 SQL 处理器为自身使用分配了一些特定的`SQLCODE`值，下文列出了它们的数值和符号名称。注意，这些值不能移植到其他 SQL 实现。为了便于将应用迁移到`SQLSTATE`方案，还列出了对应的`SQLSTATE`。不过，两种方案之间不存在一对一或一对多的映射关系（实际上是多对多），因此每次都应查阅附录 A中的完整`SQLSTATE`列表。

以下是已分配的`SQLCODE`值：

0 (`ECPG_NO_ERROR`)

表示没有错误（SQLSTATE 00000）。

100 (`ECPG_NOT_FOUND`)

这是一种无害情况，它表示上一个命令检索或者处理了零行，或者你到达了游标的末尾（SQLSTATE 02000）。

在一个循环中处理一个游标时，你可以使用这个代码作为一种方法来检测何时中止该循环，像这样：

```
while (1)
{
```

```
        EXEC SQL FETCH ... ;
        if (sqlca.sqlcode == ECPG_NOT_FOUND)
            break;
    }
```

但是WHENEVER NOT FOUND DO BREAK实际上会在内部这样做，因此显式地把它写出来通常没有什么好处。

-12 (ECPG_OUT_OF_MEMORY)

表示你的虚拟内存已被耗尽。数字值被定义为-ENOMEM（SQLSTATE YE001）。

-200 (ECPG_UNSUPPORTED)

表示预处理器生成了该库无法识别的内容。
可能是所用的预处理器与库的版本不兼容（SQLSTATE YE002）。

-201 (ECPG_TOO_MANY_ARGUMENTS)

这表示命令指定了超过该命令预期数量的主变量（SQLSTATE 07001 或 07002）。

-202 (ECPG_TOO_FEW_ARGUMENTS)

这表示命令指定的主变量数量低于该命令的预期（SQLSTATE 07001 或 07002）。

-203 (ECPG_TOO_MANY_MATCHES)

这意味着一个查询已经返回了多个行，但是该语句只准备存储一个结果行（例如，因为指定的变量不是数组）（SQLSTATE 21000）。

-204 (ECPG_INT_FORMAT)

主变量是类型int而数据库中的数据是一种不同的类型并且含有一个不能被解释为int的值。该库使用strtol()进行这种转换（SQLSTATE 42804）。

-205 (ECPG_UINT_FORMAT)

主变量是类型unsigned int而数据库中的数据是一种不同的类型并且含有一个不能被解释为unsigned int的值。该库使用strtoul()进行这种转换（SQLSTATE 42804）。

-206 (ECPG_FLOAT_FORMAT)

主变量是类型float而数据库中的数据是另一种类型并且含有一个不能被解释为float的值。该库使用strtod()进行这种转换（SQLSTATE 42804）。

-207 (ECPG_NUMERIC_FORMAT)

主变量是类型numeric
而数据库中的数据是另一种类型并且含有一个不能被解释为numeric的值（SQLSTATE 42804）。

-208 (ECPG_INTERVAL_FORMAT)

主变量是类型interval
而数据库中的数据是另一种类型并且含有一个不能被解释为interval的值（SQLSTATE 42804）。

-209 (ECPG_DATE_FORMAT)

主变量是类型date而数据库中的数据是另一种类型并且含有一个不能被解释为date的值（SQLSTATE 42804）。

-210 (`ECPG_TIMESTAMP_FORMAT`)

主变量是类型`timestamp`
而数据库中的数据是另一种类型并且含有一个不能被解释为`timestamp`的值（SQLSTATE 42804）。

-211 (`ECPG_CONVERT_BOOL`)

这表示主变量是类型`bool`而数据库中的数据既不是`'t'`也不是`'f'`（SQLSTATE 42804）。

-212 (`ECPG_EMPTY`)

发送给PostgreSQL服务器的语句是空的（通常在一个嵌入式 SQL 程序中不会发生，因此它可能指向一个内部错误）（SQLSTATE YE002）。

-213 (`ECPG_MISSING_INDICATOR`)

返回了一个空值，但未提供空值指示符变量（SQLSTATE 22002）。

-214 (`ECPG_NO_ARRAY`)

在要求一个数组的地方使用了一个普通变量（SQLSTATE 42804）。

-215 (`ECPG_DATA_NOT_ARRAY`)

在一个要求数组值的地方数据库返回了一个普通变量（SQLSTATE 42804）。

-220 (`ECPG_NO_CONN`)

程序尝试访问一个不存在的连接（SQLSTATE 08003）。

-221 (`ECPG_NOT_CONN`)

程序尝试访问一个存在的连接但是它没有打开（这是一个内部错误）（SQLSTATE YE002）。

-230 (`ECPG_INVALID_STMT`)

尝试使用的语句尚未预备（SQLSTATE 26000）。

-239 (`ECPG_INFORMIX_DUPLICATE_KEY`)

重复键错误，违背唯一约束（Informix 兼容模式）（SQLSTATE 23505）。

-240 (`ECPG_UNKNOWN_DESCRIPTOR`)

未找到指定的描述符。尝试使用的语句尚未预备（SQLSTATE 33000）。

-241 (`ECPG_INVALID_DESCRIPTOR_INDEX`)

指定的描述符超出范围（SQLSTATE 07009）。

-242 (`ECPG_UNKNOWN_DESCRIPTOR_ITEM`)

请求了一个无效的描述符条目（这是一个内部错误）（SQLSTATE YE002）。

-243 (`ECPG_VAR_NOT_NUMERIC`)

执行动态语句时，数据库返回了数值，而主变量不是数值类型（SQLSTATE 07006）。

-244 (`ECPG_VAR_NOT_CHAR`)

执行动态语句时，数据库返回了非数值的值，而主变量是数值类型（SQLSTATE 07006）。

-284 (`ECPG_INFORMIX_SUBSELECT_NOT_ONE`)

子查询的结果不是单一行（Informix 兼容模式）（SQLSTATE 21000）。

-400 (`ECPG_PGSQL`)

PostgreSQL服务器导致了某个错误。该消息包含来自PostgreSQL服务器的错误消息。

-401 (`ECPG_TRANS`)

PostgreSQL服务器通知我们不能启动、提交或回滚事务（SQLSTATE 08007）。

-402 (`ECPG_CONNECT`)

到数据库的连接尝试没有成功（SQLSTATE 08001）。

-403 (`ECPG_DUPLICATE_KEY`)

重复键错误，违背唯一约束（SQLSTATE 23505）。

-404 (`ECPG_SUBSELECT_NOT_ONE`)

子查询的结果不是单一行（SQLSTATE 21000）。

-602 (`ECPG_WARNING_UNKNOWN_PORTAL`)

指定了一个非法的游标名（SQLSTATE 34000）。

-603 (`ECPG_WARNING_IN_TRANSACTION`)

事务正在进行（SQLSTATE 25001）。

-604 (`ECPG_WARNING_NO_TRANSACTION`)

没有活动（正在进行）的事务（SQLSTATE 25P01）。

-605 (`ECPG_WARNING_PORTAL_EXISTS`)

指定了一个现有的游标名（SQLSTATE 42P03）。

33.9. 预处理器指令

一些预处理器指令可以用来改变`ecpg`预处理器解析和处理一个文件的方式。

33.9.1. 包括文件

要在嵌入式 SQL 程序中包含外部文件，可使用：

```
EXEC SQL INCLUDE filename;
EXEC SQL INCLUDE <filename>;
EXEC SQL INCLUDE "filename";
```

嵌入式 SQL 预处理器会查找名为*filename*`.h`的文件，对它进行预处理，并将其包含在生成的 C 输出中。这样，被包含文件中的嵌入式 SQL 语句就能得到正确处理。

`ecpg`预处理器将以下列顺序在几个目录中搜索一个文件：

- 当前目录
- `/usr/local/include`
- PostgreSQL 的头文件目录，在编译时定义（例如`/usr/local/pgsql/include`）

- /usr/include

但是当使用EXEC SQL INCLUDE "*filename*"时，只有当前目录会被搜索。

在每一个目录中，预处理器将首先按给定的文件名搜索，如果没有找到将会追加.h到文件名并且重试（除非指定的文件名已经具有该后缀）。

注意EXEC SQL INCLUDE不同于：

```
#include <filename.h>
```

因为这样包含的文件不会经过 SQL 命令预处理。当然，仍可使用 C 的#include指令包含其他头文件。

> **注意**
>
> 被包含文件的名称区分大小写，尽管EXEC SQL INCLUDE命令的其他部分遵循通常的 SQL 大小写规则。

33.9.2. define 和 undef 指令

与 C 中我们熟知的指令#define相似，嵌入式 SQL 具有类似的概念：

```
EXEC SQL DEFINE name;
EXEC SQL DEFINE name value;
```

因此你可以定义一个名称：

```
EXEC SQL DEFINE HAVE_FEATURE;
```

并且你也可以定义常量：

```
EXEC SQL DEFINE MYNUMBER 12;
EXEC SQL DEFINE MYSTRING 'abc';
```

使用undef来移除一个之前的定义：

```
EXEC SQL UNDEF MYNUMBER;
```

当然在你的嵌入式 SQL 程序中你可以继续使用 C 版本的#define和#undef。区别在于你定义的值会在哪里被计算。如果你使用EXEC SQL DEFINE，那么ecpg预处理器会计算这些定义并且替换值。例如，如果你写：

```
EXEC SQL DEFINE MYNUMBER 12;
...
EXEC SQL UPDATE Tbl SET col = MYNUMBER;
```

那么ecpg将已经做过替换并且你的 C 编译器将永远不会看见名为MYNUMBER的任何名称或标识符。注意你不能把#define用于一个将要在一个嵌入式 SQL 查询中使用的常量，因为在这种情况下嵌入式 SQL 预处理器不能看到这个声明。

33.9.3. ifdef、ifndef、else、elif 和 endif 指令

可以使用以下指令对代码区段进行条件编译：

```
EXEC SQL ifdef name;
```

检查 *name*，如果已通过 EXEC SQL define *name* 创建了 *name*，就处理后续各行。

`EXEC SQL ifndef name;`

检查 *name*，如果尚未通过 `EXEC SQL define name` 创建 *name*，就处理后续各行。

`EXEC SQL else;`

开始处理一个备选区段，对应由 `EXEC SQL ifdef name` 或 `EXEC SQL ifndef name` 引入的区段。

`EXEC SQL elif name;`

检查 *name*，如果已通过 `EXEC SQL define name` 创建了 *name*，就开始一个备选区段。

`EXEC SQL endif;`

结束备选区段。

示例：

```
EXEC SQL ifndef TZVAR;
EXEC SQL SET TIMEZONE TO 'GMT';
EXEC SQL elif TZNAME;
EXEC SQL SET TIMEZONE TO TZNAME;
EXEC SQL else;
EXEC SQL SET TIMEZONE TO TZVAR;
EXEC SQL endif;
```

33.10. 处理嵌入式 SQL 程序

现在你已经对如何构造嵌入式 SQL C 程序有所了解了，你可能希望知道如何编译它们。在编译之前，你需要让该文件通过嵌入式SQL C预处理器，它会把你用到的SQL转换成特殊的函数调用。在编译之后，你必须链接一个包含所需函数的特殊库。这些函数从参数中取得信息、使用libpq执行SQL命令并且把结果放在指定的参数中用来输出。

该预处理器程序被称作`ecpg`并且被包括在一个正常的PostgreSQL安装中。嵌入式 SQL 程序通常带有扩展名`.pgc`。如果你有一个程序文件`prog1.pgc`，你可以调用下面的命令对它进行预处理：

```
ecpg prog1.pgc
```

这将创建一个文件`prog1.c`。如果你的输入文件不遵循建议的命名模式，你可以用`-o`选项显式地指定输出文件。

预处理过的文件可以被正常地编译，例如：

```
cc -c prog1.c
```

生成的 C 源文件会包含PostgreSQL安装中的头文件。因此，如果PostgreSQL安装在默认搜索路径之外，就必须在编译命令行中添加类似`-I/usr/local/pgsql/include`的选项。

要链接嵌入式 SQL 程序，需要链接`libecpg`库，例如：

```
cc -o myprog prog1.o prog2.o ... -lecpg
```

同样，可能需要在该命令行中添加类似`-L/usr/local/pgsql/lib`的选项。

如果你使用make来管理一个大工程的构建过程，把下面的隐式规则包括在你的 makefile 中将会很方便：

```
ECPG = ecpg
```

```
%.c: %.pgc
        $(ECPG) $<
```

ecpg命令的完整语法可见ecpg。

ecpg库默认是线程安全的。不过，
你可能需要使用一些线程命令行选项来编译你的客户端代码。

33.11. 库函数

libecpg库主要包含用于实现嵌入式 SQL 命令功能的“隐藏”函数，也有一些适合直接调用的函数。注意，直接调用这些函数会使代码失去可移植性。

- 调用ECPGdebug(int *on*, FILE **stream*)时，如果第一个参数非零，就会启用调试日志。日志写入*stream*，包含代入全部输入变量后的所有SQL语句，以及PostgreSQL服务器返回的结果。这对于查找SQL语句中的错误很有用。

> **注意**
>
> 在 Windows 上，如果ecpg库和应用是使用不同标志编译的，这个函数调用将会使应用崩溃，因为FILE指针的内部表示不同。特别地，
> 库和使用库的应用应该使用相同的多线程/单线程、发行/调试以及静态/动态标志。

- ECPGget_PGconn(const char **connection_name*) 返回由给定名称标识的库数据库连接句柄。如果*connection_name*被设置为NULL，当前连接句柄将被返回。
  如果无法定位到连接句柄，该函数返回NULL。如果需要，
  返回的连接句柄可以被用来调用任何其他来自于libpq的函数。

> **注意**
>
> 直接使用libpq例程来操纵ecpg中建立的数据库连接句柄是一种糟糕的做法。

- ECPGtransactionStatus(const char **connection_name*) 返回由*connection_name* 标识的连接的当前事务状态。返回状态码的详细信息，参见第 31.2 节和 libpq 的 PQtransactionStatus()。

- 如果你连接到了一个数据库，ECPGstatus(int *lineno*, const char* *connection_name*)会返回真；否则返回假。如果使用的是一个单一连接，*connection_name*可以为NULL。

33.12. 大对象

ECPG 并不直接支持大对象，在调用ECPGget_PGconn()函数获得所需的PGconn对象后，ECPG 应用能通过 libpq 大对象函数操纵大对象（不过，对ECPGget_PGconn()函数的使用以及直接接触PGconn对象都必须非常小心，并且最好不要与其他 ECPG 数据库访问调用混合在一起）。

更多关于ECPGget_PGconn()的细节可见第 33.11 节。大对象函数接口的相关信息可见第 32 章。

大对象函数必须在一个事务块中被调用，因此当自动提交关闭时，必须显式地发出BEGIN命令。

例 33.2给出了一个例子程序，它展示了在一个 ECPG 应用中如何创建、写入和读取一个大对象。

例 33.2. 访问大对象的 ECPG 程序

```
#include <stdio.h>
#include <stdlib.h>
#include <libpq-fe.h>
#include <libpq/libpq-fs.h>

EXEC SQL WHENEVER SQLERROR STOP;

int
main(void)
{
    PGconn     *conn;
    Oid         loid;
    int         fd;
    char        buf[256];
    int         buflen = 256;
    char        buf2[256];
    int         rc;

    memset(buf, 1, buflen);

    EXEC SQL CONNECT TO testdb AS con1;

    conn = ECPGget_PGconn("con1");
    printf("conn = %p\n", conn);

    /* create */
    loid = lo_create(conn, 0);
    if (loid &lt; 0)
        printf("lo_create() failed: %s", PQerrorMessage(conn));

    printf("loid = %d\n", loid);

    /* write test */
    fd = lo_open(conn, loid, INV_READ|INV_WRITE);
    if (fd &lt; 0)
        printf("lo_open() failed: %s", PQerrorMessage(conn));

    printf("fd = %d\n", fd);

    rc = lo_write(conn, fd, buf, buflen);
    if (rc &lt; 0)
        printf("lo_write() failed\n");

    rc = lo_close(conn, fd);
    if (rc &lt; 0)
        printf("lo_close() failed: %s", PQerrorMessage(conn));

    /* read test */
    fd = lo_open(conn, loid, INV_READ);
    if (fd &lt; 0)
        printf("lo_open() failed: %s", PQerrorMessage(conn));

    printf("fd = %d\n", fd);
```

```
    rc = lo_read(conn, fd, buf2, buflen);
    if (rc &lt; 0)
        printf("lo_read() failed\n");

    rc = lo_close(conn, fd);
    if (rc &lt; 0)
        printf("lo_close() failed: %s", PQerrorMessage(conn));

    /* check */
    rc = memcmp(buf, buf2, buflen);
    printf("memcmp() = %d\n", rc);

    /* cleanup */
    rc = lo_unlink(conn, loid);
    if (rc &lt; 0)
        printf("lo_unlink() failed: %s", PQerrorMessage(conn));

    EXEC SQL COMMIT;
    EXEC SQL DISCONNECT ALL;
    return 0;
}
```

33.13. C++ 应用

ECPG 对 C++ 应用提供有限支持。本节介绍一些注意事项。

`ecpg` 预处理器接受一个用 C（或类似 C 的语言）和嵌入式 SQL 命令编写的输入文件，将嵌入式 SQL 命令转换为 C 代码片段，最终生成 `.c` 文件。在 C++ 中使用时，`ecpg` 生成的 C 代码片段所调用的库函数，其头文件声明会包在 `extern "C" { ... }` 块中，因此应该能在 C++ 中无缝工作。

不过，通常`ecpg`预处理器只理解 C，它无法处理 C++ 语言的特殊语法和保留词。因此，一些写在 C++ 应用代码中的使用了 C++ 特定复杂特性的嵌入式 SQL 代码可能无法被正确地预处理或者无法按预期工作。

使用 C++ 应用中嵌入式 SQL 代码的安全方法是把 ECPG 调用隐藏在一个 C 模块中，C++ 应用代码会调用它来访问数据库，还要把它和剩余的 C++ 代码链接起来。详见第 33.13.2 节。

33.13.1. 主变量的可见范围

`ecpg`预处理器理解 C 中变量的作用域。在 C 语言中，这比较简单，因为变量的作用域由所在代码块决定。但在 C++ 中，类成员变量可能在与声明位置不同的代码块中被引用，因此`ecpg`预处理器无法理解类成员变量的作用域。

例如，在下面的情况中，`ecpg`预处理器无法为`test`方法中的变量`dbname`找到任何声明，因此将发生一个错误。

```
class TestCpp
{
    EXEC SQL BEGIN DECLARE SECTION;
    char dbname[1024];
    EXEC SQL END DECLARE SECTION;

  public:
    TestCpp();
    void test();
    ~TestCpp();
```

```
};

TestCpp::TestCpp()
{
    EXEC SQL CONNECT TO testdb1;
}

void Test::test()
{
    EXEC SQL SELECT current_database() INTO :dbname;
    printf("current_database = %s\n", dbname);
}

TestCpp::~TestCpp()
{
    EXEC SQL DISCONNECT ALL;
}
```

这段代码将导致一个这样的错误：

```
ecpg test_cpp.pgc
test_cpp.pgc:28: ERROR: variable "dbname" is not declared
```

为避免这个作用域问题，可以修改`test`方法，使用局部变量作为中间存储。不过，这只是一个较差的变通办法，因为它使代码变得难看，还会降低性能。

```
void TestCpp::test()
{
    EXEC SQL BEGIN DECLARE SECTION;
    char tmp[1024];
    EXEC SQL END DECLARE SECTION;

    EXEC SQL SELECT current_database() INTO :tmp;
    strlcpy(dbname, tmp, sizeof(tmp));

    printf("current_database = %s\n", dbname);
}
```

33.13.2. 使用外部 C 模块的 C++ 应用开发

了解`ecpg`预处理器在 C++ 中的这些技术限制后，可能会认为：在链接阶段将 C 和 C++ 的目标文件链接起来，使 C++ 应用能够使用 ECPG 功能，会比直接在 C++ 代码中编写嵌入式 SQL 命令更好。本节用一个简单示例介绍如何将嵌入式 SQL 命令与 C++ 应用代码分离。在这个示例中，应用用 C++ 实现，连接 PostgreSQL 服务器的部分则使用 C 和 ECPG。

需要创建三种文件：C 文件（`*.pgc`）、头文件和 C++ 文件：

`test_mod.pgc`

一个执行嵌入在 C 中的 SQL 命令的子例程模块。它将被预处理器转换成`test_mod.c`。

```
#include "test_mod.h"
#include <stdio.h>

void
db_connect()
{
    EXEC SQL CONNECT TO testdb1;
}
```

```
void
db_test()
{
    EXEC SQL BEGIN DECLARE SECTION;
    char dbname[1024];
    EXEC SQL END DECLARE SECTION;

    EXEC SQL SELECT current_database() INTO :dbname;
    printf("current_database = %s\n", dbname);
}

void
db_disconnect()
{
    EXEC SQL DISCONNECT ALL;
}
```

test_mod.h

包含 C 模块（test_mod.pgc）中函数声明的头文件，由test_cpp.cpp包含。由于需要从 C++ 模块链接这些函数，声明必须包在extern "C"块中。

```
#ifdef __cplusplus
extern "C" {
#endif

void db_connect();
void db_test();
void db_disconnect();

#ifdef __cplusplus
}
#endif
```

test_cpp.cpp

应用的主要代码，包括main例程，在本示例中还包括一个 C++ 类。

```
#include "test_mod.h"

class TestCpp
{
  public:
    TestCpp();
    void test();
    ~TestCpp();
};

TestCpp::TestCpp()
{
    db_connect();
}

void
TestCpp::test()
{
    db_test();
}
```

```
TestCpp::~TestCpp()
{
    db_disconnect();
}

int
main(void)
{
    TestCpp *t = new TestCpp();

    t->test();
    return 0;
}
```

要构建该应用，按以下步骤处理。通过运行ecpg将test_mod.pgc转换为test_mod.c，并且用 C 编译器将test_mod.c编译成test_mod.o：

```
ecpg -o test_mod.c test_mod.pgc
cc -c test_mod.c -o test_mod.o
```

接着，用 C++ 编译器把test_cpp.cpp编译成test_cpp.o：

```
c++ -c test_cpp.cpp -o test_cpp.o
```

最后，通过 C++ 编译器驱动程序，将test_cpp.o和test_mod.o这两个目标文件链接成一个可执行文件：

```
c++ test_cpp.o test_mod.o -lecpg -o test_cpp
```

33.14. 嵌入式 SQL 命令

本节介绍专用于嵌入式 SQL 的全部 SQL 命令。另请参见SQL Commands中列出的 SQL 命令，除非另有说明，它们也可用于嵌入式 SQL。

ALLOCATE DESCRIPTOR

ALLOCATE DESCRIPTOR — allocate an SQL descriptor area

大纲

```
ALLOCATE DESCRIPTOR name
```

描述

`ALLOCATE DESCRIPTOR`分配一个新的命名 SQL 描述符区域，可用于在 PostgreSQL 服务器与主程序之间交换数据。

描述符区域使用后应通过`DEALLOCATE DESCRIPTOR`命令释放。

参数

name

SQL 描述符的名称，大小写敏感。这可以是一个 SQL 标识符或者一个主变量。

例子

```
EXEC SQL ALLOCATE DESCRIPTOR mydesc;
```

兼容性

SQL 标准中说明了`ALLOCATE DESCRIPTOR`。

参见

DEALLOCATE DESCRIPTOR, GET DESCRIPTOR, SET DESCRIPTOR

CONNECT

CONNECT — establish a database connection

大纲

```
CONNECT TO connection_target [ AS connection_name ] [
 USER connection_user ]
CONNECT TO DEFAULT
CONNECT connection_user
DATABASE connection_target
```

描述

`CONNECT`命令在客户端和 PostgreSQL 服务器之间建立一个连接。

参数

connection_target

*connection_target*以下列形式之一指定连接的目标服务器。

[*database_name*][@*host*][:*port*]

通过 TCP/IP 连接

unix:postgresql://*host*[:*port*]/[*database_name*][?*connection_option*]

通过 Unix 域套接字

tcp:postgresql://*host*[:*port*]/[*database_name*][?*connection_option*]

通过 TCP/IP 连接

SQL 字符串常量

包含上述形式之一的一个值

主变量

类型`char[]`或`VARCHAR[]`的主变量，它包含上述形式之一的一个值

connection_object

用于该连接的一个可选标识符，这样可以在其他命令中引用它。这可以是一个 SQL 标识符或者一个主变量。

connection_user

用于数据库连接的用户名。

使用 *user_name*/*password*、*user_name* IDENTIFIED BY *password*或者 *user_name* USING *password*之一，这个参数也能指定用户名和密码。

用户名和密码可以是 SQL 标识符、字符串常量或者主变量。

DEFAULT

按 libpq 的定义使用所有默认连接参数。

例子

这里是一些指定连接参数的变体:

```
EXEC SQL CONNECT TO "connectdb" AS main;
EXEC SQL CONNECT TO "connectdb" AS second;
EXEC SQL CONNECT TO "unix:postgresql://200.46.204.71/connectdb" AS
 main USER connectuser;
EXEC SQL CONNECT TO "unix:postgresql://localhost/connectdb" AS main
 USER connectuser;
EXEC SQL CONNECT TO 'connectdb' AS main;
EXEC SQL CONNECT TO 'unix:postgresql://localhost/connectdb' AS main
 USER :user;
EXEC SQL CONNECT TO :db AS :id;
EXEC SQL CONNECT TO :db USER connectuser USING :pw;
EXEC SQL CONNECT TO @localhost AS main USER connectdb;
EXEC SQL CONNECT TO REGRESSDB1 as main;
EXEC SQL CONNECT TO AS main USER connectdb;
EXEC SQL CONNECT TO connectdb AS :id;
EXEC SQL CONNECT TO connectdb AS main USER connectuser/connectdb;
EXEC SQL CONNECT TO connectdb AS main;
EXEC SQL CONNECT TO connectdb@localhost AS main;
EXEC SQL CONNECT TO tcp:postgresql://localhost/ USER connectdb;
EXEC SQL CONNECT TO tcp:postgresql://localhost/connectdb USER
 connectuser IDENTIFIED BY connectpw;
EXEC SQL CONNECT TO tcp:postgresql://localhost:20/connectdb USER
 connectuser IDENTIFIED BY connectpw;
EXEC SQL CONNECT TO unix:postgresql://localhost/ AS main USER
 connectdb;
EXEC SQL CONNECT TO unix:postgresql://localhost/connectdb AS main
 USER connectuser;
EXEC SQL CONNECT TO unix:postgresql://localhost/connectdb USER
 connectuser IDENTIFIED BY "connectpw";
EXEC SQL CONNECT TO unix:postgresql://localhost/connectdb USER
 connectuser USING "connectpw";
EXEC SQL CONNECT TO unix:postgresql://localhost/connectdb?connect_
timeout=14 USER connectuser;
```

这里是一个展示使用主变量指定连接参数的例子程序:

```
int
main(void)
{
EXEC SQL BEGIN DECLARE SECTION;
    char *dbname     = "testdb";    /* 数据库名 */
    char *user       = "testuser";  /* 连接用户名 */
    char *connection = "tcp:postgresql://localhost:5432/testdb";
                                    /* 连接字符串 */
    char ver[256];                  /* 存储版本字符串的缓冲区 */
EXEC SQL END DECLARE SECTION;

    ECPGdebug(1, stderr);

    EXEC SQL CONNECT TO :dbname USER :user;
    EXEC SQL SELECT version() INTO :ver;
    EXEC SQL DISCONNECT;

    printf("version: %s\n", ver);
```

```
    EXEC SQL CONNECT TO :connection USER :user;
    EXEC SQL SELECT version() INTO :ver;
    EXEC SQL DISCONNECT;

    printf("version: %s\n", ver);

    return 0;
}
```

兼容性

SQL 标准中说明了`CONNECT`，但是连接参数的格式是与实现相关的。

参见

DISCONNECT, SET CONNECTION

DEALLOCATE DESCRIPTOR

DEALLOCATE DESCRIPTOR — deallocate an SQL descriptor area

大纲

```
DEALLOCATE DESCRIPTOR name
```

描述

`DEALLOCATE DESCRIPTOR`释放一个命名的 SQL 描述符区域。

参数

name

要被释放的描述符的名称。它是大小写敏感的。这可以是一个 SQL 标识符或者一个主变量。

例子

```
EXEC SQL DEALLOCATE DESCRIPTOR mydesc;
```

兼容性

SQL 标准说明了`DEALLOCATE DESCRIPTOR`。

参见

ALLOCATE DESCRIPTOR, GET DESCRIPTOR, SET DESCRIPTOR

DECLARE

DECLARE — define a cursor

大纲

```
DECLARE cursor_name [ BINARY ] [ INSENSITIVE ] [ [ NO ] SCROLL ]
 CURSOR [ { WITH | WITHOUT } HOLD ] FOR prepared_name
DECLARE cursor_name [ BINARY ] [ INSENSITIVE ] [ [ NO ] SCROLL ]
 CURSOR [ { WITH | WITHOUT } HOLD ] FOR query
```

描述

`DECLARE`声明一个游标用来在一个预备语句的结果集上迭代。这个命令与直接的 SQL 命令`DECLARE`在语义上有一点点区别：后者会执行一个查询并且准备结果集用于检索，而这个嵌入式 SQL 命令仅仅声明一个名称作为 “循环变量” 用于在一个查询的结果集上迭代，实际的执行在游标被`OPEN`命令打开时才发生。

参数

cursor_name

一个游标名称，大小写敏感。这可以是一个 SQL 标识符或者一个主变量。

prepared_name

一个预备查询的名称，可以是一个 SQL 标识符或者一个主变量。

query

一个提供游标要返回的行的SELECT或者VALUES命令。

游标选项的含义请见DECLARE。

例子

为一个查询声明一个游标的例子：

```
EXEC SQL DECLARE C CURSOR FOR SELECT * FROM My_Table;
EXEC SQL DECLARE C CURSOR FOR SELECT Item1 FROM T;
EXEC SQL DECLARE cur1 CURSOR FOR SELECT version();
```

为一个预备语句声明一个游标的例子：

```
EXEC SQL PREPARE stmt1 AS SELECT version();
EXEC SQL DECLARE cur1 CURSOR FOR stmt1;
```

兼容性

SQL 标准中说明了`DECLARE`。

参见

OPEN, CLOSE, DECLARE

DESCRIBE

DESCRIBE — obtain information about a prepared statement or result set

大纲

```
DESCRIBE [ OUTPUT ] prepared_name USING [ SQL ]
 DESCRIPTOR descriptor_name
DESCRIBE [ OUTPUT ] prepared_name INTO [ SQL ]
 DESCRIPTOR descriptor_name
DESCRIBE [ OUTPUT ] prepared_name INTO sqlda_name
```

描述

`DESCRIBE`获取预备语句所含结果列的元数据，而不实际读取任何行。

参数

prepared_name

一个预备语句的名称。这可以是一个 SQL 标识符或者一个主变量。

descriptor_name

一个描述符名称。它是大小写敏感的。它可以是一个 SQL 标识符或者一个主变量。

sqlda_name

一个 SQLDA 变量的名称。

例子

```
EXEC SQL ALLOCATE DESCRIPTOR mydesc;
EXEC SQL PREPARE stmt1 FROM :sql_stmt;
EXEC SQL DESCRIBE stmt1 INTO SQL DESCRIPTOR mydesc;
EXEC SQL GET DESCRIPTOR mydesc VALUE 1 :charvar = NAME;
EXEC SQL DEALLOCATE DESCRIPTOR mydesc;
```

兼容性

SQL 标准中说明了`DESCRIBE`。

参见

ALLOCATE DESCRIPTOR, GET DESCRIPTOR

DISCONNECT

DISCONNECT — terminate a database connection

大纲

```
DISCONNECT connection_name
DISCONNECT [ CURRENT ]
DISCONNECT DEFAULT
DISCONNECT ALL
```

描述

`DISCONNECT`关闭一个（或者所有）到数据库的连接。

参数

connection_name

一个由`CONNECT`命令建立的数据库连接名称。

`CURRENT`

关闭“当前的”连接，它可以是最近打开的连接或者是由`SET CONNECTION`命令设置的连接。如果没有参数被传给`DISCONNECT`命令，这将是默认值。

`DEFAULT`

关闭默认连接。

`ALL`

关闭所有打开的连接。

例子

```
int
main(void)
{
    EXEC SQL CONNECT TO testdb AS DEFAULT USER testuser;
    EXEC SQL CONNECT TO testdb AS con1 USER testuser;
    EXEC SQL CONNECT TO testdb AS con2 USER testuser;
    EXEC SQL CONNECT TO testdb AS con3 USER testuser;

    EXEC SQL DISCONNECT CURRENT;  /* close con3          */
    EXEC SQL DISCONNECT DEFAULT;  /* close DEFAULT       */
    EXEC SQL DISCONNECT ALL;      /* close con2 and con1 */

    return 0;
}
```

兼容性

SQL 标准中说明了`DISCONNECT`。

参见

CONNECT, SET CONNECTION

EXECUTE IMMEDIATE

EXECUTE IMMEDIATE — dynamically prepare and execute a statement

大纲

`EXECUTE IMMEDIATE` *`string`*

描述

`EXECUTE IMMEDIATE`立刻预备并且执行一个动态指定的 SQL 语句，不检索结果行。

参数

`string`

C 字符串字面值，或包含待执行 SQL 语句的主变量。

例子

这里是一个用`EXECUTE IMMEDIATE`和一个名为`command`的主变量执行`INSERT`语句的例子：

```
sprintf(command, "INSERT INTO test (name, amount, letter) VALUES (
'db: ''r1''', 1, 'f')");
EXEC SQL EXECUTE IMMEDIATE :command;
```

兼容性

SQL 标准中说明了`EXECUTE IMMEDIATE`。

GET DESCRIPTOR

GET DESCRIPTOR — get information from an SQL descriptor area

大纲

```
GET DESCRIPTOR descriptor_name :cvariable = descriptor_header_item
 [, ... ]
GET DESCRIPTOR descriptor_name VALUE column_number :cvariable =
 descriptor_item [, ... ]
```

描述

`GET DESCRIPTOR`从一个 SQL 描述符区域检索关于一个查询结果集的信息并且把它存储在主变量中。在使用这个命令把信息传输到主语言变量之前，一个描述符区域通常是用`FETCH`或`SELECT`填充的。

此命令有两种形式。第一种取得描述符的“头部”项目，它们适用于整个结果集，例如行数。第二种需要额外提供列号参数，取得特定列的信息，例如列名和实际列值。

参数

descriptor_name

一个描述符名称。

descriptor_header_item

一个标识要检索哪一个头部信息项的词元。当前只支持用于得到结果集中列数的`COUNT`。

column_number

要检索其信息的列号。计数从 1 开始。

descriptor_item

一个标识要检索哪一个有关一列信息的项的词元。被支持的项可见第 33.7.1 节。

cvariable

接收从描述符区域检索到的数据的主变量。

例子

检索一个结果集中列数的例子：

```
EXEC SQL GET DESCRIPTOR d :d_count = COUNT;
```

检索第一列中数据长度的例子：

```
EXEC SQL GET DESCRIPTOR d VALUE 1 :d_returned_octet_length =
 RETURNED_OCTET_LENGTH;
```

把第二列的数据体检索成一个字符串的例子：

```
EXEC SQL GET DESCRIPTOR d VALUE 2 :d_data = DATA;
```

这里是执行`SELECT current_database();`并且显示列数、列数据长度和列数据的完整过程的例子：

```
int
main(void)
{
EXEC SQL BEGIN DECLARE SECTION;
    int  d_count;
    char d_data[1024];
    int  d_returned_octet_length;
EXEC SQL END DECLARE SECTION;

    EXEC SQL CONNECT TO testdb AS con1 USER testuser;
    EXEC SQL ALLOCATE DESCRIPTOR d;

    /* 声明、打开游标，并为游标指定描述符 */
    EXEC SQL DECLARE cur CURSOR FOR SELECT current_database();
    EXEC SQL OPEN cur;
    EXEC SQL FETCH NEXT FROM cur INTO SQL DESCRIPTOR d;

    /* 得到全部列的数量 */
    EXEC SQL GET DESCRIPTOR d :d_count = COUNT;
    printf("d_count                 = %d\n", d_count);

    /* 得到一个返回列的长度 */
    EXEC SQL GET DESCRIPTOR d VALUE 1 :d_returned_octet_length =
 RETURNED_OCTET_LENGTH;
    printf("d_returned_octet_length = %d\n", d_returned_octet_
length);

    /* 将返回的列取出成一个字符串 */
    EXEC SQL GET DESCRIPTOR d VALUE 1 :d_data = DATA;
    printf("d_data                  = %s\n", d_data);

    /* 关闭 */
    EXEC SQL CLOSE cur;
    EXEC SQL COMMIT;

    EXEC SQL DEALLOCATE DESCRIPTOR d;
    EXEC SQL DISCONNECT ALL;

    return 0;
}
```

当该例子被执行时，结果看起来是：

```
d_count                 = 1
d_returned_octet_length = 6
d_data                  = testdb
```

兼容性

SQL 标准中说明了`GET DESCRIPTOR`。

参见

ALLOCATE DESCRIPTOR, SET DESCRIPTOR

OPEN

OPEN — open a dynamic cursor

大纲

```
OPEN cursor_name
OPEN cursor_name USING value [, ... ]
OPEN cursor_name USING SQL DESCRIPTOR descriptor_name
```

描述

`OPEN`打开一个游标并且可选地绑定实际值到游标声明中的占位符。
该游标必须之前用`DECLARE`命令声明。`OPEN`的执行会导致查询开始在服务器上执行。

参数

cursor_name

要被打开的游标的名称。这可以是一个 SQL 标识符或者一个主变量。

value

要被绑定到游标中一个占位符的值。这可以是一个 SQL 常量、一个主变量或者一个带有指示符的主变量。

descriptor_name

包含要绑定到游标中占位符的值的描述符的名称。这可以是一个 SQL 标识符或者一个主变量。

例子

```
EXEC SQL OPEN a;
EXEC SQL OPEN d USING 1, 'test';
EXEC SQL OPEN c1 USING SQL DESCRIPTOR mydesc;
EXEC SQL OPEN :curname1;
```

兼容性

SQL 标准中说明了`OPEN`。

参见

DECLARE, CLOSE

PREPARE

PREPARE — prepare a statement for execution

大纲

```
PREPARE name FROM string
```

描述

`PREPARE`预备一个以字符串动态指定的语句，以供执行。这不同于直接的 SQL 语句PREPARE，后者也可用于嵌入式程序。EXECUTE命令可用于执行任一种预备语句。

参数

prepared_name

预备查询的一个标识符。

string

C 字符串字面值，或包含可预备语句的主变量，语句可以是 SELECT、INSERT、UPDATE 或 DELETE。

例子

```
char *stmt = "SELECT * FROM test1 WHERE a = ? AND b = ?";

EXEC SQL ALLOCATE DESCRIPTOR outdesc;
EXEC SQL PREPARE foo FROM :stmt;

EXEC SQL EXECUTE foo USING SQL DESCRIPTOR indesc INTO SQL
 DESCRIPTOR outdesc;
```

兼容性

SQL 标准中说明了`PREPARE`。

参见

EXECUTE

SET AUTOCOMMIT

SET AUTOCOMMIT — set the autocommit behavior of the current session

大纲

```
SET AUTOCOMMIT { = | TO } { ON | OFF }
```

描述

`SET AUTOCOMMIT`设置当前数据库会话的自动提交行为。默认情况下，嵌入式 SQL 程序不处于自动提交模式，需要提交时必须显式发出`COMMIT`。此命令可以将会话改为自动提交模式，使每条语句都被隐式提交。

兼容性

`SET AUTOCOMMIT`是 PostgreSQL ECPG 的扩展。

SET CONNECTION

SET CONNECTION — select a database connection

大纲

```
SET CONNECTION [ TO | = ] connection_name
```

描述

`SET CONNECTION`设置“当前的”数据库连接，除非被覆盖，所有命令都会使用这个连接。

参数

connection_name

一个由`CONNECT`命令建立的数据库连接名称。

`DEFAULT`

将连接设为默认连接。

例子

```
EXEC SQL SET CONNECTION TO con2;
EXEC SQL SET CONNECTION = con1;
```

兼容性

SQL 标准中说明了`SET CONNECTION`。

参见

CONNECT, DISCONNECT

SET DESCRIPTOR

SET DESCRIPTOR — set information in an SQL descriptor area

大纲

```
SET DESCRIPTOR descriptor_name descriptor_header_item = value [, ..
. ]
SET DESCRIPTOR descriptor_name VALUE number descriptor_item = value
 [, ...]
```

描述

`SET DESCRIPTOR`用值填充一个 SQL 描述符区域。然后该描述符区域通常会被用来在一个预备查询执行中绑定参数。

此命令有两种形式：第一种适用于描述符的“头部”，与某个特定数据项无关。
第二种为以编号标识的特定数据项赋值。

参数

descriptor_name

一个描述符名称。

descriptor_header_item

一个标识要设置哪个头部信息项的词元。当前只有设置描述符项数量的`COUNT`被支持。

number

要设置的描述符项的编号。计数从 1 开始。

descriptor_item

一个标识在描述符中要设置哪个信息项的词元。受支持的项的列表可见第 33.7.1 节。

value

要存入描述符项的值，可以是 SQL 常量或主变量。

例子

```
EXEC SQL SET DESCRIPTOR indesc COUNT = 1;
EXEC SQL SET DESCRIPTOR indesc VALUE 1 DATA = 2;
EXEC SQL SET DESCRIPTOR indesc VALUE 1 DATA = :val1;
EXEC SQL SET DESCRIPTOR indesc VALUE 2 INDICATOR = :val1, DATA =
 'some string';
EXEC SQL SET DESCRIPTOR indesc VALUE 2 INDICATOR = :val2null, DATA
 = :val2;
```

兼容性

SQL 标准中说明了`SET DESCRIPTOR`。

参见

ALLOCATE DESCRIPTOR, GET DESCRIPTOR

TYPE

TYPE — define a new data type

大纲

```
TYPE type_name IS ctype
```

描述

`TYPE`命令定义一个新的 C 类型。它等效于把一个`typedef`放在声明节中。

只有使用选项`-c`运行`ecpg`时才能识别这个命令。

参数

type_name

新类型的名称。这必须是一个合法的 C 类型名。

ctype

一个 C 类型说明。

例子

```
EXEC SQL TYPE customer IS
    struct
    {
        varchar name[50];
        int     phone;
    };

EXEC SQL TYPE cust_ind IS
    struct ind
    {
        short   name_ind;
        short   phone_ind;
    };

EXEC SQL TYPE c IS char reference;
EXEC SQL TYPE ind IS union { int integer; short smallint; };
EXEC SQL TYPE intarray IS int[AMOUNT];
EXEC SQL TYPE str IS varchar[BUFFERSIZ];
EXEC SQL TYPE string IS char[11];
```

这里是一个使用`EXEC SQL TYPE`的例子程序:

```
EXEC SQL WHENEVER SQLERROR SQLPRINT;

EXEC SQL TYPE tt IS
    struct
    {
        varchar v[256];
        int     i;
    };

EXEC SQL TYPE tt_ind IS
```

```
    struct ind {
        short   v_ind;
        short   i_ind;
    };

int
main(void)
{
EXEC SQL BEGIN DECLARE SECTION;
    tt t;
    tt_ind t_ind;
EXEC SQL END DECLARE SECTION;

    EXEC SQL CONNECT TO testdb AS con1;

    EXEC SQL SELECT current_database(), 256 INTO :t:t_ind LIMIT 1;

    printf("t.v = %s\n", t.v.arr);
    printf("t.i = %d\n", t.i);

    printf("t_ind.v_ind = %d\n", t_ind.v_ind);
    printf("t_ind.i_ind = %d\n", t_ind.i_ind);

    EXEC SQL DISCONNECT con1;

    return 0;
}
```

这个程序的输出看起来像:

```
t.v = testdb
t.i = 256
t_ind.v_ind = 0
t_ind.i_ind = 0
```

兼容性

`TYPE`命令是一种 PostgreSQL 扩展。

VAR

VAR — define a variable

大纲

```
VAR varname IS ctype
```

描述

VAR命令为主变量指定一个新的 C 数据类型。该主变量必须已在声明节中声明。

参数

varname

一个 C 变量名。

ctype

一个 C 类型说明。

例子

```
Exec sql begin declare section;
short a;
exec sql end declare section;
EXEC SQL VAR a IS int;
```

兼容性

VAR命令是一个 PostgreSQL 扩展。

WHENEVER

WHENEVER — specify the action to be taken when an SQL statement causes a specific class condition to be raised

大纲

```
WHENEVER { NOT FOUND | SQLERROR | SQLWARNING } action
```

描述

定义 SQL 执行结果出现特殊情况（未找到行、SQL 警告或错误）时采取的行为。

参数

参数描述见第 33.8.1 节。

例子

```
EXEC SQL WHENEVER NOT FOUND CONTINUE;
EXEC SQL WHENEVER NOT FOUND DO BREAK;
EXEC SQL WHENEVER SQLWARNING SQLPRINT;
EXEC SQL WHENEVER SQLWARNING DO warn();
EXEC SQL WHENEVER SQLERROR sqlprint;
EXEC SQL WHENEVER SQLERROR CALL print2();
EXEC SQL WHENEVER SQLERROR DO handle_error("select");
EXEC SQL WHENEVER SQLERROR DO sqlnotice(NULL, NONO);
EXEC SQL WHENEVER SQLERROR DO sqlprint();
EXEC SQL WHENEVER SQLERROR GOTO error_label;
EXEC SQL WHENEVER SQLERROR STOP;
```

典型用法是通过`WHENEVER NOT FOUND BREAK`处理遍历结果集的循环：

```
int
main(void)
{
    EXEC SQL CONNECT TO testdb AS con1;
    EXEC SQL ALLOCATE DESCRIPTOR d;
    EXEC SQL DECLARE cur CURSOR FOR SELECT current_database(),
 'hoge', 256;
    EXEC SQL OPEN cur;

    /* 到达结果集末尾时，跳出 while 循环 */
    EXEC SQL WHENEVER NOT FOUND DO BREAK;

    while (1)
    {
        EXEC SQL FETCH NEXT FROM cur INTO SQL DESCRIPTOR d;
        ...
    }

    EXEC SQL CLOSE cur;
    EXEC SQL COMMIT;

    EXEC SQL DEALLOCATE DESCRIPTOR d;
    EXEC SQL DISCONNECT ALL;

    return 0;
```

```
}
```

兼容性

SQL 标准中说明了`WHENEVER`，但是大部分动作是 PostgreSQL 扩展。

33.15. Informix兼容模式

`ecpg`可以运行在一种所谓的Informix 兼容模式中。如果这种模式被激活，它的行为就好像它是一个用于Informix E/SQL 的Informix预处理器。一般而言，这将允许你使用美元符号替代`EXEC SQL`来引入嵌入式 SQL 命令：

```
$int j = 3;
$CONNECT TO :dbname;
$CREATE TABLE test(i INT PRIMARY KEY, j INT);
$INSERT INTO test(i, j) VALUES (7, :j);
$COMMIT;
```

> **注意**
>
> `$`与后面的预处理器指令（如`include`、`define`、`ifdef`等）之间不能有任何空白。否则，预处理器会将该词元解析为主变量。

有两种兼容性模式：`INFORMIX`、`INFORMIX_SE`

在链接使用这种兼容性模式的程序时，要记得链接上和 ECPG 一起发布的`libcompat`。

除了之前解释过的语法糖，Informix兼容模式从 E/SQL 中移植了一些用于输入、输出和数据转换的函数以及嵌入式 SQL 语句到 ECPG 中。

Informix兼容模式与 ECPG 的 pgtypeslib 库密切相关。pgtypeslib 将 SQL 数据类型映射到 C 主程序中的数据类型，Informix兼容模式中的大多数附加函数用于操作这些 C 主程序类型。但兼容范围有限。它并不试图复制Informix的行为，而是允许执行大致相同的操作，并提供名称和基本行为相同的函数；如果目前使用的是Informix，则不能用它直接替代。此外，有些数据类型也有差异。例如，PostgreSQL的日期时间和 interval 类型不支持`YEAR TO MINUTE`这样的范围，ECPG 也不支持。

33.15.1. 附加类型

Informix 模式现在支持 Informix 特有的“string”伪类型，用于存储去除尾随空格的字符串数据，无需使用`typedef`。事实上，在 Informix 模式中，ECPG 拒绝处理包含`typedef sometype string;`的源文件。

```
EXEC SQL BEGIN DECLARE SECTION;
string userid; /* 此变量将包含去除尾随空格的数据 */
EXEC SQL END DECLARE SECTION;

EXEC SQL FETCH MYCUR INTO :userid;
```

33.15.2. 附加的/缺少的嵌入式 SQL 语句

`CLOSE DATABASE`

这个语句关闭当前连接。实际上，这是ECPG的`DISCONNECT CURRENT`的同义词：

```
$CLOSE DATABASE;                /* close the current connection
 */
```

```
   EXEC SQL CLOSE DATABASE;
```

`FREE cursor_name`

由于ECPG的工作方式与Informix的ESQL/C有所不同（即，哪些步骤纯粹是语法转换，哪些步骤依赖于底层运行时库），在ECPG中没有`FREE cursor_name`语句。这是因为在ECPG中，`DECLARE CURSOR`不会转换为使用游标名称的运行时库的函数调用。这意味着在ECPG运行时库中没有SQL游标的运行时记录，只在PostgreSQL服务器中有。

`FREE statement_name`

`FREE statement_name`是`DEALLOCATE PREPARE statement_name`的同义词。

33.15.3. Informix-兼容的 SQLDA 描述符区域

Informix 兼容模式支持的结构体与第 33.7.2 节中所述的不同，如下所示：

```
struct sqlvar_compat
{
    short   sqltype;
    int     sqllen;
    char   *sqldata;
    short  *sqlind;
    char   *sqlname;
    char   *sqlformat;
    short   sqlitype;
    short   sqlilen;
    char   *sqlidata;
    int     sqlxid;
    char   *sqltypename;
    short   sqltypelen;
    short   sqlownerlen;
    short   sqlsourcetype;
    char   *sqlowner name;
    int     sqlsourceid;
    char   *sqlilongdata;
    int     sqlflags;
    void   *sqlreserved;
};

struct sqlda_compat
{
    short  sqld;
    struct sqlvar_compat *sqlvar;
    char   desc_name[19];
    short  desc_occ;
    struct sqlda_compat *desc_next;
    void  *reserved;
};

typedef struct sqlvar_compat    sqlvar_t;
typedef struct sqlda_compat     sqlda_t;
```

全局属性是：

`sqld`

`SQLDA`描述符中的字段数量。

`sqlvar`

每一个域属性的指针。

`desc_name`

未使用，用零字节填充。

`desc_occ`

已分配结构体的尺寸。

`desc_next`

如果结果集包含多于一个记录，这个字段是下一个 SQLDA 结构体的指针。

`reserved`

未使用的指针，包含 NULL。为 Informix-兼容性而保留。

每个字段的属性如下，它们存储在 `sqlvar` 数组中：

`sqltype`

域的类型。可以使用的常量定义在`sqltypes.h`中。

`sqllen`

域数据的长度。

`sqldata`

指向字段数据的指针，类型为`char *`，所指的数据采用二进制格式。例如：

```
int intval;

switch (sqldata->sqlvar[i].sqltype)
{
    case SQLINTEGER:
        intval = *(int *)sqldata->sqlvar[i].sqldata;
        break;
  ...
}
```

`sqlind`

指向 NULL 指示符的指针。如果由 DESCRIBE 或 FETCH 返回，它始终是有效指针。如果作为`EXECUTE ... USING sqlda;`的输入，则 NULL 指针表示此字段的值不是 NULL。否则，必须正确设置有效指针和`sqlitype`。例如：

```
if (*(int2 *)sqldata->sqlvar[i].sqlind != 0)
    printf("value is NULL\n");
```

`sqlname`

域的名称。以 0 终止的字符串。

`sqlformat`

在 Informix 中保留；这里是对该字段调用 `PQfformat()` 所得的值。

`sqlitype`

NULL 指示符数据的类型。从服务器返回数据时，它始终是 SQLSMINT。将`SQLDA`用于参数化查询时，按所设置的类型处理数据。

`sqlilen`

NULL 指示符数据的长度。

`sqlxid`

字段的扩展类型，即 `PQftype()` 的结果。

`sqltypename`
`sqltypelen`
`sqlownerlen`
`sqlsourcetype`
`sqlownername`
`sqlsourceid`
`sqlflags`
`sqlreserved`

未使用。

`sqlilongdata`

如果`sqllen`大于 32KB，它等于`sqldata`。

示例：

```
EXEC SQL INCLUDE sqlda.h;

    sqlda_t        *sqlda; /* 这不需要在嵌入式 DECLARE SECTION 下 */

    EXEC SQL BEGIN DECLARE SECTION;
    char *prep_stmt = "select * from table1";
    int i;
    EXEC SQL END DECLARE SECTION;

    ...

    EXEC SQL PREPARE mystmt FROM :prep_stmt;

    EXEC SQL DESCRIBE mystmt INTO sqlda;

    printf("# of fields: %d\n", sqlda->sqld);
    for (i = 0; i < sqlda->sqld; i++)
      printf("field %d: \"%s\"\n", sqlda->sqlvar[i]->sqlname);

    EXEC SQL DECLARE mycursor CURSOR FOR mystmt;
    EXEC SQL OPEN mycursor;
    EXEC SQL WHENEVER NOT FOUND GOTO out;

    while (1)
    {
      EXEC SQL FETCH mycursor USING sqlda;
    }

    EXEC SQL CLOSE mycursor;
```

free(sqlda); /* 主结构完全被 free(), sqlda 和 sqlda->sqlvar 在一个已分配区域中 */

更多信息参见 sqlda.h 头文件和 src/interfaces/ecpg/test/compat_informix/sqlda.pgc 回归测试。

33.15.4. 附加函数

decadd

将两个 decimal 类型的值相加。

```
int decadd(decimal *arg1, decimal *arg2, decimal *sum);
```

该函数接收指向第一个 decimal 类型操作数的指针（arg1）、指向第二个 decimal 类型操作数的指针（arg2），以及指向用于保存和的 decimal 类型值的指针（sum）。成功时返回 0，发生溢出时返回ECPG_INFORMIX_NUM_OVERFLOW，发生下溢时返回ECPG_INFORMIX_NUM_UNDERFLOW。其他失败情况下返回 -1，并将errno设为 pgtypeslib 中对应的errno值。

deccmp

比较两个 decimal 类型的变量。

```
int deccmp(decimal *arg1, decimal *arg2);
```

该函数接收指向第一个 decimal 值的指针（arg1）和指向第二个 decimal 值的指针（arg2），并返回一个整数值，表示哪个值更大。

- 1，如果arg1指向的值大于var2指向的值
- -1，如果arg1指向的值小于arg2指向的值
- 0，如果arg1指向的值和arg2指向的值相等

deccopy

复制一个 decimal 值。

```
void deccopy(decimal *src, decimal *target);
```

该函数的第一个参数是指向待复制 decimal 值的指针（src），第二个参数是指向目标 decimal 类型结构体的指针（target）。

deccvasc

将一个值的 ASCII 表示转换为 decimal 类型。

```
int deccvasc(char *cp, int len, decimal *np);
```

该函数接收指向待转换数值的字符串表示的指针（cp）及其长度len。np 是指向用于保存操作结果的 decimal 值的指针。

例如，有效的格式包括：-2、.794、+3.44、592.49E07或 -32.84e-4。

该函数成功返回0。如果发生溢出或下溢，则返回ECPG_INFORMIX_NUM_OVERFLOW或ECPG_INFORMIX_NUM_UNDERFLOW。如果ASCII表示无法解析，则返回ECPG_INFORMIX_BAD_NUMERIC，或者如果在解析指数时出现问题，则返回 ECPG_INFORMIX_BAD_EXPONENT。

`deccvdbl`

将 double 类型的值转换为 decimal 类型的值。

```
int deccvdbl(double dbl, decimal *np);
```

该函数接收应该被转换的 double 类型变量作为其第一个参数（`dbl`）。作为第二个参数（`np`），该函数接收一个指向应该保存操作结果的 decimal 变量的指针。

该函数在成功时返回0，在转换失败时返回负值。

`deccvint`

将 int 类型的值转换为 decimal 类型的值。

```
int deccvint(int in, decimal *np);
```

该函数接收应该被转换的 int 类型变量作为其第一个参数（`in`）。作为第二个参数（`np`），该函数接收一个指向应该保存操作结果的 decimal 变量的指针。

该函数在成功时返回0，在转换失败时返回负值。

`deccvlong`

将类型为 long 的值转换为类型为 decimal 的值。

```
int deccvlong(long lng, decimal *np);
```

该函数接收应该被转换的类型为 long 的变量作为其第一个参数（`lng`）。作为第二个参数（`np`），该函数接收一个指向应该保存操作结果的 decimal 变量的指针。

该函数在成功时返回0，在转换失败时返回负值。

`decdiv`

将两个 decimal 类型的变量相除。

```
int decdiv(decimal *n1, decimal *n2, decimal *result);
```

该函数接收指向第一个（`n1`）和第二个（`n2`）操作数的变量的指针，并计算`n1/n2`。`result`是应该保存操作结果的变量的指针。

在成功时返回0，如果除法失败则返回负值。如果发生溢出或下溢，则函数分别返回`ECPG_INFORMIX_NUM_OVERFLOW`或 `ECPG_INFORMIX_NUM_UNDERFLOW`。如果尝试除以零，则函数返回`ECPG_INFORMIX_DIVIDE_ZERO`。

`decmul`

将两个 decimal 值相乘。

```
int decmul(decimal *n1, decimal *n2, decimal *result);
```

该函数接收指向第一个（`n1`）和第二个（`n2`）操作数的变量的指针，并计算`n1*n2`。`result`是一个指向应该保存操作结果的变量的指针。

在成功时返回0，如果乘法失败则返回负值。如果发生溢出或下溢，函数分别返回 `ECPG_INFORMIX_NUM_OVERFLOW`或 `ECPG_INFORMIX_NUM_UNDERFLOW`。

`decsub`

从一个 decimal 值中减去另一个 decimal 值。

```
int decsub(decimal *n1, decimal *n2, decimal *result);
```

该函数接收指向第一个（n1）和第二个（n2）操作数的变量的指针，并计算n1-n2。result是应该保存操作结果的变量的指针。

在成功时返回0，如果减法失败则返回负值。如果发生溢出或下溢，函数分别返回 ECPG_INFORMIX_NUM_OVERFLOW或 ECPG_INFORMIX_NUM_UNDERFLOW。

dectoasc

将 decimal 类型变量转换为 C char* 字符串中的 ASCII 表示。

```
int dectoasc(decimal *np, char *cp, int len, int right)
```

该函数接收指向 decimal 类型变量的指针（np），并将该变量转换为文本表示。cp是用于保存操作结果的缓冲区。参数right指定输出中小数点右侧的位数，结果将舍入到这个小数位数。将right设为 -1 表示输出所有可用的小数位。如果len指定的输出缓冲区长度不足以容纳文本表示及末尾的零字节，则只在结果中存入一个*字符，并返回 -1。

如果缓冲区cp太小，函数返回 -1；如果内存耗尽，则返回ECPG_INFORMIX_OUT_OF_MEMORY。

dectodbl

将 decimal 类型的变量转换为 double。

```
int dectodbl(decimal *np, double *dblp);
```

该函数接受指向待转换 decimal 值的指针（np），以及指向用于保存操作结果的 double 变量的指针（dblp）。

当成功时，返回0；如果转换失败，则返回负值。

dectoint

将 decimal 类型的变量转换为整数。

```
int dectoint(decimal *np, int *ip);
```

该函数接受指向待转换 decimal 值的指针（np），以及指向用于保存操作结果的整数变量的指针（ip）。

当成功时，返回0；如果转换失败，则返回负值。如果发生溢出，将返回ECPG_INFORMIX_NUM_OVERFLOW。

请注意，ECPG实现与Informix实现不同。Informix将整数限制在-32767到32767的范围内，而ECPG实现中的限制取决于架构（-INT_MAX .. INT_MAX）。

dectolong

将 decimal 类型的变量转换为长整数。

```
int dectolong(decimal *np, long *lngp);
```

该函数接受指向待转换 decimal 值的指针（np），以及指向用于保存操作结果的 long 变量的指针（lngp）。

当成功时，返回0；如果转换失败，则返回负值。如果发生溢出，将返回ECPG_INFORMIX_NUM_OVERFLOW。

请注意，ECPG实现与Informix实现不同。Informix将长整型限制在-2,147,483,647到2,147,483,647的范围内，而ECPG实现中的限制取决于架构（-LONG_MAX .. LONG_MAX）。

rdatestr

将日期转换成 C char* 字符串。

```
int rdatestr(date d, char *str);
```

该函数接受两个参数，第一个是待转换的日期（d），第二个是指向目标字符串的指针。输出格式始终为 yyyy-mm-dd，因此必须为字符串分配至少 11 字节（包括值为零的终止字节）。

该函数在成功时返回0，在错误时返回负值。

请注意，ECPG的实现与Informix的实现不同。在Informix中，格式可以通过设置环境变量来影响。然而，在ECPG中，你无法更改输出格式。

rstrdate

解析日期的文本表示。

```
int rstrdate(char *str, date *d);
```

该函数接收要转换的日期的文本表示（str）和指向类型为 date 的变量的指针（d）。此函数不允许你指定格式掩码。它使用Informix的默认格式掩码，即mm/dd/yyyy。在内部，此函数通过rdefmtdate实现。因此，rstrdate不会更快，如果可以选择，应选择允许你显式指定格式掩码的 rdefmtdate。

这个函数返回与rdefmtdate相同的值。

rtoday

得到当前日期。

```
void rtoday(date *d);
```

该函数接收一个指向 date 变量的指针（d），并将其指向的变量设为当前日期。

在内部，此函数使用PGTYPESdate_today函数。

rjulmdy

从一个 date 类型的变量中提取日、月和年的值。

```
int rjulmdy(date d, short mdy[3]);
```

该函数接收日期d和一个指向包含 3 个 short 整数值的数组mdy的指针。变量名表明了顺序：mdy[0]设为月份数，mdy[1]设为日，mdy[2]包含年份。

这个函数目前总是返回0。

在内部，该函数使用PGTYPESdate_julmdy函数。

rdefmtdate

使用格式掩码将字符串转换为日期类型的值。

```
int rdefmtdate(date *d, char *fmt, char *str);
```

该函数接收一个指向用于保存操作结果的 date 值的指针（d）、用于解析日期的格式掩码（fmt），以及包含日期文本表示的 C char* 字符串（str）。文本表示应与格式掩码

匹配，但字符串与格式掩码无需一一对应。该函数只分析先后顺序，
并查找表示年份位置的yy或yyyy、表示月份位置的mm以及表示日位置的dd。

该函数返回以下值：

- 0 - 函数成功终止。
- ECPG_INFORMIX_ENOSHORTDATE - 日期不包含日、月和年之间的分隔符。在这种情况下，输入字符串必须恰好为6或8个字节长，但实际不是。
- ECPG_INFORMIX_ENOTDMY - 格式字符串未正确指示年、月和日的顺序。
- ECPG_INFORMIX_BAD_DAY - 输入字符串不包含有效的日。
- ECPG_INFORMIX_BAD_MONTH - 输入字符串不包含有效的月。
- ECPG_INFORMIX_BAD_YEAR - 输入字符串不包含有效的年。

在内部，此函数实现为使用PGTYPESdate_defmt_asc函数。请参阅那里的参考资料，
了解示例输入表。

rfmtdate

使用格式掩码将 date 类型的变量转换为文本表示。

```
int rfmtdate(date d, char *fmt, char *str);
```

该函数接收要转换的日期（d）、格式掩码（fmt）
和将保存日期文本表示的字符串（str）。

成功时，返回 0；如果发生错误，则返回一个负值。

在内部，此函数使用PGTYPESdate_fmt_asc函数，有关示例，
请参阅那里的参考资料。

rmdyjul

根据一个包含 3 个 short 整数的数组创建日期值，这些整数指定日期的日、月和年。

```
int rmdyjul(short mdy[3], date *d);
```

该函数接收包含 3 个 short 整数的数组（mdy）和一个指向 date 类型变量的指针，用来保存操作结果。

目前该函数始终返回0。

在内部，该函数实现为使用函数PGTYPESdate_mdyjul。

rdayofweek

为一个日期值返回表示它是星期几的数字。

```
int rdayofweek(date d);
```

这个函数接收日期变量d作为它唯一的参数并且返回一个整数说明这个日期是星期几。

- 0 - 星期日
- 1 - 星期一
- 2 - 星期二

- 3 - 星期三
- 4 - 星期四
- 5 - 星期五
- 6 - 星期六

在内部，该函数被实现为使用函数`PGTYPESdate_dayofweek`。

`dtcurrent`

检索当前时间戳。

```
void dtcurrent(timestamp *ts);
```

该函数检索当前时间戳，并将其保存到`ts`指向的时间戳变量中。

`dtcvasc`

将时间戳的文本表示解析为 timestamp 变量。

```
int dtcvasc(char *str, timestamp *ts);
```

该函数接收要解析的字符串（`str`），以及指向用于保存操作结果的 timestamp 变量的指针（`ts`）。

该函数在成功时返回0，在错误时返回负值。

在内部，此函数使用`PGTYPEStimestamp_from_asc`函数。请参阅那里的参考资料，了解包含示例输入的表格。

`dtcvfmtasc`

使用格式掩码将时间戳的文本表示解析为 timestamp 变量。

```
dtcvfmtasc(char *inbuf, char *fmtstr, timestamp *dtvalue)
```

该函数接收要解析的字符串（`inbuf`）、使用的格式掩码（`fmtstr`），以及指向用于保存操作结果的 timestamp 变量的指针（`dtvalue`）。

这个函数是通过`PGTYPEStimestamp_defmt_asc`函数实现的。请参阅那里的文档，了解可用的格式说明符列表。

该函数在成功时返回0，在错误时返回负值。

`dtsub`

从一个时间戳中减去另一个时间戳，并返回一个 interval 类型的变量。

```
int dtsub(timestamp *ts1, timestamp *ts2, interval *iv);
```

该函数从`ts1`指向的时间戳变量中减去`ts2`指向的时间戳变量，并将结果存入`iv`指向的 interval 变量中。

成功时返回 0，发生错误时返回负值。

`dttoasc`

将 timestamp 变量转换为 C char* 字符串。

```
int dttoasc(timestamp *ts, char *output);
```

该函数接收指向待转换 timestamp 变量的指针（ts），以及用于保存操作结果的字符串（output）。它将ts转换为符合 SQL 标准的文本表示，格式为YYYY-MM-DD HH:MM:SS。

成功时返回 0，发生错误时返回负值。

dttofmtasc

使用格式掩码将 timestamp 变量转换为 C char*。

```
int dttofmtasc(timestamp *ts, char *output, int str_len, char
 *fmtstr);
```

该函数的第一个参数是指向待转换时间戳的指针（ts），
随后是指向输出缓冲区的指针（output）、为输出缓冲区分配的最大长度（str_len），以及转换所用的格式掩码（fmtstr）。

成功时返回 0，发生错误时返回负值。

在内部，此函数使用PGTYPEStimestamp_fmt_asc函数。请参阅那里的参考资料，了解可以使用哪些格式掩码说明符。

intoasc

将 interval 变量转换为 C char* 字符串。

```
int intoasc(interval *i, char *str);
```

该函数接受指向待转换 interval 变量的指针（i），以及用于保存操作结果的字符串（str）。它将 i 转换为符合 SQL 标准的文本表示，格式为 YYYY-MM-DD HH:MM:SS。

成功时返回 0，发生错误时返回负值。

rfmtlong

将长整型值使用格式掩码转换为其文本表示形式。

```
int rfmtlong(long lng_val, char *fmt, char *outbuf);
```

该函数接收长整型值lng_val，格式掩码fmt和指向输出缓冲区outbuf的指针。
它根据格式掩码将长整型值转换为其文本表示形式。

格式掩码可以由以下格式指定字符组成：

- *（星号）- 如果此位置为空白，用星号填充。
- &（和号）- 如果此位置为空白，用零填充。
- # - 将前导零转换为空格。
- < - 将数字左对齐在字符串中。
- ,（逗号）- 将四位或更多位的数字每三位分为一组，用逗号分隔。
- .（句点）- 此字符将整数部分与小数部分分隔开。
- –（减号）- 如果数字是负值，则显示减号。
- +（加号）- 如果数字是正值，则显示加号。
- （ - 这个字符替换负数前面的减号。减号不会显示。
- ）- 这个字符替换减号，并打印在负值后面。

- `$` - 货币符号。

`rupshift`

将字符串转换为大写。

```
void rupshift(char *str);
```

该函数接收一个指向字符串的指针，并将每个小写字符转换为大写。

`byleng`

返回字符串中字符的数量，不包括末尾的空格。

```
int byleng(char *str, int len);
```

该函数期望一个固定长度的字符串作为第一个参数（`str`），并将其长度作为第二个参数（`len`）。它返回有效字符的数量，即不包括末尾空格的字符串长度。

`ldchar`

将固定长度的字符串复制到以空字符结尾的字符串中。

```
void ldchar(char *src, int len, char *dest);
```

该函数接收要复制的固定长度字符串（`src`）、其长度（`len`）
和指向目标内存的指针（`dest`）。请注意，你需要为`dest`指向的字符串保留至少`len+1`字节。该函数最多复制`len`字节到新位置（如果源字符串具有尾随空格，则会少一些），并添加空字符终止符。

`rgetmsg`

```
int rgetmsg(int msgnum, char *s, int maxsize);
```

这个函数存在，但目前尚未实现！

`rtypalign`

```
int rtypalign(int offset, int type);
```

这个函数存在，但目前尚未实现！

`rtypmsize`

```
int rtypmsize(int type, int len);
```

这个函数存在，但目前尚未实现！

`rtypwidth`

```
int rtypwidth(int sqltype, int sqllen);
```

这个函数存在，但目前尚未实现！

`rsetnull`

将一个变量设置为NULL。

```
int rsetnull(int t, char *ptr);
```

该函数接收一个整数，表示变量的类型，以及一个指向变量本身的指针，
该指针被转换为C中的char*指针。

下列类型存在：

- CCHARTYPE - 用于char或char*类型的变量
- CSHORTTYPE - 用于short int类型的变量
- CINTTYPE - 用于int类型的变量
- CBOOLTYPE - 用于boolean类型的变量
- CFLOATTYPE - 用于float类型的变量
- CLONGTYPE - 用于long类型的变量
- CDOUBLETYPE - 用于double类型的变量
- CDECIMALTYPE - 用于decimal类型的变量
- CDATETYPE - 用于date类型的变量
- CDTIMETYPE - 用于timestamp类型的变量

这是调用此函数的示例：

```
$char c[] = "abc       ";
$short s = 17;
$int i = -74874;

rsetnull(CCHARTYPE, (char *) c);
rsetnull(CSHORTTYPE, (char *) &s);
rsetnull(CINTTYPE, (char *) &i);
```

risnull

测试变量是否为NULL。

```
int risnull(int t, char *ptr);
```

该函数接收要测试的变量类型（t）以及指向该变量的指针（ptr）。请注意，后者需要转换为char*类型。查看函数rsetnull以获取可能的变量类型列表。

这是如何使用这个函数的示例：

```
$char c[] = "abc       ";
$short s = 17;
$int i = -74874;

risnull(CCHARTYPE, (char *) c);
risnull(CSHORTTYPE, (char *) &s);
risnull(CINTTYPE, (char *) &i);
```

33.15.5. 额外的常量

注意这里所有的常量都描述错误并且它们都被定义为表示负值。
在每一种不同常量的描述中，你还可以找到在当前实现中该常量表示的值。
不过你不应该依赖于这个数字。但是你可以相信所有的这些常量都是被定义为表示负值。

ECPG_INFORMIX_NUM_OVERFLOW

如果在一次计算中发生了溢出，函数会返回这个值。在内部它被定义为 -1200（Informix 定义）。

`ECPG_INFORMIX_NUM_UNDERFLOW`

如果在一次计算中发生了下溢，函数会返回这个值。在内部它被定义为 -1201（Informix定义）。

`ECPG_INFORMIX_DIVIDE_ZERO`

如果发现尝试除零，函数会返回这个值。在内部它被定义为 -1202（Informix定义）。

`ECPG_INFORMIX_BAD_YEAR`

如果解析日期时发现无效的年份值，函数返回此值。其内部定义为 -1204（Informix的定义）。

`ECPG_INFORMIX_BAD_MONTH`

如果解析日期时发现无效的月份值，函数返回此值。其内部定义为 -1205（Informix的定义）。

`ECPG_INFORMIX_BAD_DAY`

如果解析日期时发现无效的日值，函数返回此值。其内部定义为 -1206（Informix的定义）。

`ECPG_INFORMIX_ENOSHORTDATE`

如果解析例程需要短日期表示，但未得到长度正确的日期字符串，函数返回此值。
其内部定义为 -1209（Informix的定义）。

`ECPG_INFORMIX_DATE_CONVERT`

如果在日期格式化时产生了一个错误，函数会返回这个值。在内部它被定义为 -1210（Informix定义）。

`ECPG_INFORMIX_OUT_OF_MEMORY`

如果在操作时内存被耗尽，函数会返回这个值。在内部它被定义为 -1211（Informix定义）。

`ECPG_INFORMIX_ENOTDMY`

如果一个解析例程被假定为得到一个格式掩码（如`mmddyy`）
但是列出的域并不是全部正确，函数会返回这个值。在内部它被定义为 -1212（Informix定义）。

`ECPG_INFORMIX_BAD_NUMERIC`

如果解析例程因 numeric 值的文本表示有误而无法解析，或者例程因至少一个 numeric 变量无效而无法完成涉及 numeric 变量的计算，函数返回此值。其内部定义为 -1213（Informix的定义）。

`ECPG_INFORMIX_BAD_EXPONENT`

如果一个解析例程不能解析一个指数，函数会返回这个值。在内部它被定义为 -1216（Informix定义）。

`ECPG_INFORMIX_BAD_DATE`

如果一个解析例程不能解析一个日期，函数会返回这个值。在内部它被定义为 -1218（Informix定义）。

`ECPG_INFORMIX_EXTRA_CHARS`

如果一个解析例程被传递了它不能解析的额外字符，函数会返回这个值。
在内部它被定义为 -1264（Informix定义）。

33.16. 内部

这一节解释ECPG在内部如何工作。这些信息有时有助于用户理解如何使用ECPG。

`ecpg`写入输出的前四行是固定的：其中两行是注释，另外两行是与库接口所必需的包含行。随后预处理器会通读整个文件并写出输出，通常它只是把所有内容原样回显到输出中。

当它看到`EXEC SQL`语句时，就会介入并对其进行改写。命令以`EXEC SQL`开始，以`;`结束。其间的所有内容都会被视为一条SQL语句，并进行变量替换解析。

当某个符号以冒号（`:`）开头时，就会发生变量替换。系统会在此前于`EXEC SQL DECLARE`声明节中声明的变量里查找同名变量。

库中最重要的函数是`ECPGdo`，负责执行大多数命令。它接收可变数量的参数，
参数数量很容易达到 50 个左右；我们希望这在任何平台上都不会造成问题。

参数是：

一个行号

这是原始行的行号，只用于错误消息。

一个字符串

这是将要发出的SQL命令。输入变量会对它进行修改，也就是那些在编译时尚未知晓、但要填入命令中的变量。字符串中的`?`表示变量应当放置的位置。

输入变量

每个输入变量都会生成十个参数（见下文）。

`ECPGt_EOIT`

一个表明后面没有更多输入变量的`enum`值。

输出变量

每个输出变量都会生成十个参数（见下文）。这些变量将由该函数填充。

`ECPGt_EORT`

一个表明后面没有更多变量的`enum`值。

对于每一个作为SQL命令一部分的变量，该函数得到十个参数：

1. 用特殊符号表示的类型。
2. 一个值的指针或者一个指针的指针。
3. 如果变量是一个`char`或者`varchar`，这是它的尺寸。
4. 数组中元素的数量（用于数组获取）。
5. 数组中下一个元素的偏移量（用于数组获取）。
6. 作为一个特殊符号的指示符变量的类型。

7. 一个指示符变量的指针。
8. 0
9. 指示符数组中的元素数量（用于数组取数）。
10. 到指示符数组中下一个元素的偏移量（用于数组取数）。

注意并非所有 SQL 命令都被以这种方式对待。例如，一个打开游标语句：

```
EXEC SQL OPEN cursor;
```

它不会被复制到输出中。相反，游标的`DECLARE`命令会被放在`OPEN`命令的位置上，因为真正打开游标的正是它。

这里有一个完整的例子，它描述了一个文件`foo.pgc`的预处理器输出（对预处理器的每一个特定版本细节可能不同）：

```
EXEC SQL BEGIN DECLARE SECTION;
int index;
int result;
EXEC SQL END DECLARE SECTION;
...
EXEC SQL SELECT res INTO :result FROM mytable WHERE index = :index;
```

会被转换成：

```
/* Processed by ecpg (2.6.0) */
/* These two include files are added by the preprocessor */
#include <ecpgtype.h>;
#include <ecpglib.h>;

/* 声明节开始 */

#line 1 "foo.pgc"

 int index;
 int result;
/* 声明节结束 */
...
ECPGdo(__LINE__, NULL, "SELECT res FROM mytable WHERE index = ?
 ",
        ECPGt_int,&(index),1L,1L,sizeof(int),
        ECPGt_NO_INDICATOR, NULL , 0L, 0L, 0L, ECPGt_EOIT,
        ECPGt_int,&(result),1L,1L,sizeof(int),
        ECPGt_NO_INDICATOR, NULL , 0L, 0L, 0L, ECPGt_EORT);
#line 147 "foo.pgc"
```

（这里的缩进是为了可读性而添加的，并非是预处理器做的处理）。

第 34 章 信息模式

信息模式由一组视图组成，其中包含当前数据库中定义的对象的信息。信息模式由 SQL 标准定义，因此可以预期它具有可移植性并保持稳定；而系统目录则不同，它们是 PostgreSQL 特有的，并且是围绕实现方面的考虑建模的。不过，信息模式视图不包含 PostgreSQL 特有功能的信息；要查询这些信息，需要查询系统目录或其他 PostgreSQL 特有视图。

注意

当在数据库中查询约束信息时，一个期望返回一行的标准兼容的查询可能返回多行。这是因为 SQL 标准要求约束名在一个模式中唯一，但是PostgreSQL并不强制这种限制。PostgreSQL自动产生的约束名避免在相同的模式中重复，
但是用户能够指定这种重复的名称。

这个问题可能在查询信息模式视图时出现，例如`check_constraint_routine_usage`、`check_constraints`、`domain_constraints`和 `referential_constraints`。一些其他视图也有相似的问题，
但是它们包含了表名来帮助区分重复行，例如`constraint_column_usage`、`constraint_table_usage`、`table_constraints`。

34.1. 模式

信息模式本身是一个名为`information_schema`的模式。
这个模式自动存在于所有数据库中。这个模式的拥有者是集簇中的初始数据库用户，
并且该用户自然地拥有这个模式上的所有权限，
包括删除它的能力（但是这样节省的空间是很小的）。

默认情况下，信息模式不在模式搜索路径中，因此你需要使用限定名访问其中的所有对象。
因为信息模式中的某些对象的名称是可能出现在用户应用中的一般名称，
如果你想把该信息模式放在路径中，你应该小心。

34.2. 数据类型

信息模式视图的列使用在信息模式中定义的特殊数据类型。
这些类型被定义为普通内置类型之上的简单域。不应在信息模式之外使用这些类型，
但如果应用要从信息模式中查询数据，就必须能够处理它们。

这些类型为：

`cardinal_number`

一种非负整数。

`character_data`

一种字符串（没有指定最大长度）。

`sql_identifier`

一种字符串。该类型用于 SQL 标识符，其他任何文本数据则使用类型`character_data`。

`time_stamp`

在类型`timestamp with time zone`之上的一个域。

yes_or_no

一种字符串域，只包含YES或NO。它用于在信息模式中表示布尔（真/假）数据。（信息模式是在类型boolean被纳入 SQL 标准之前设计出来的，因此需要采用这种约定以保持向后兼容。）

信息模式中的每一列都具有这五种类型之一。

34.3. information_schema_catalog_name

information_schema_catalog_name 是一个表，它始终只包含一行一列，用于存放当前数据库的名称（按 SQL 术语称为当前目录）。

表 34.1. information_schema_catalog_name 列

名称	数据类型	描述
catalog_name	sql_identifier	包含这个信息模式的数据库名称

34.4. administrable_role_authorizations

视图administrable_role_authorizations
标识当前用户对其有管理选项的所有角色。

表 34.2. administrable_role_authorizations 列

名称	数据类型	描述
grantee	sql_identifier	被授予该角色成员资格的角色名称（可以是当前用户，也可以是在嵌套角色成员资格情况下的另一个角色）
role_name	sql_identifier	一个角色的名称
is_grantable	yes_or_no	总是 YES

34.5. applicable_roles

视图applicable_roles标识当前用户可以使用其权限的所有角色。
这意味着从当前用户到相关角色之间存在某条角色授予链。
当前用户本身也是一个适用角色。适用角色的集合通常用于权限检查。

表 34.3. applicable_roles 列

名称	数据类型	描述
grantee	sql_identifier	被授予该角色成员资格的角色名称（可以是当前用户，也可以是在嵌套角色成员资格情况下的另一个角色）
role_name	sql_identifier	一个角色的名称
is_grantable	yes_or_no	如果被授权人在角色上有管理选项则为YES，如果没有则为NO

34.6. attributes

视图attributes包含数据库中定义的组合数据类型的属性信息。（注意，该视图不提供表列的信息，而表列在 PostgreSQL 的语境中有时也被称为属性。）只有当前用户能够访问的那些属性才会显示出来（通过拥有该类型或拥有其上的某种权限）。

表 34.4. attributes 列

名称	数据类型	描述
udt_catalog	sql_identifier	包含该数据类型的数据库名称（总是当前数据库）
udt_schema	sql_identifier	包含该数据类型的模式名称
udt_name	sql_identifier	数据类型名称
attribute_name	sql_identifier	属性名称
ordinal_position	cardinal_number	属性在该数据类型内部的顺序位置（从 1 开始计算）
attribute_default	character_data	该属性的默认表达式
is_nullable	yes_or_no	如果该属性是可能为空的，值为YES，否则为NO。
data_type	character_data	如果该属性是一个内置类型，此列值为该属性的数据类型；如果该属性是某种数组，此列值为ARRAY（在这种情况下，见视图element_types）；其他情况，此列值为USER-DEFINED（在这种情况下，该类型在attribute_udt_name和相关列中标识）。
character_maximum_length	cardinal_number	如果data_type标识字符类型或位串类型，这里是声明的最大长度；如果未声明最大长度，或者对于所有其他数据类型，则为空。
character_octet_length	cardinal_number	如果data_type标识字符类型，这里是一个数据值可能的最大字节长度；对于所有其他数据类型则为空。最大字节长度取决于声明的字符最大长度（见上文）和服务器编码。
character_set_catalog	sql_identifier	对应 PostgreSQL 不支持的特性
character_set_schema	sql_identifier	对应 PostgreSQL 不支持的特性
character_set_name	sql_identifier	对应 PostgreSQL 不支持的特性
collation_catalog	sql_identifier	包含该属性排序规则的数据库名称（总是当前数据库）；如果使用默认排序规则，或该属性的数据类型不支持排序规则，则为null。
collation_schema	sql_identifier	包含该属性排序规则的模式名称；如果使用默认排序规则，或该属性的数据类型不支持排序规则，则为 null。
collation_name	sql_identifier	该属性排序规则的名称；如果使用默认排序规则，或该属性的数据类型不支持排序规则，则为null。

名称	数据类型	描述
numeric_precision	cardinal_number	如果data_type标识一种数字类型，这列包含这个属性类型的（声明的或隐式的）精度。精度指示了有效位数。它可以按照列numeric_precision_radix中指定的被表示为十进制（基于10）或二进制（基于2）。对于所有其他数据类型，这一列为空。
numeric_precision_radix	cardinal_number	如果data_type标识一种数字类型，这一列指示numeric_precision和numeric_scale列中的值是基于什么来表示。该值为2或10。对于所有其他数据类型，这一列为空。
numeric_scale	cardinal_number	如果data_type标识精确数字类型，这一列包含该属性类型的标度（声明的或隐式的）。标度表示小数点右侧的有效数字位数。它可以按列numeric_precision_radix的指定表示为十进制（基数10）或二进制（基数2）。对于所有其他数据类型，这一列为空。
datetime_precision	cardinal_number	如果data_type标识一种日期、时间、时间戳或时间间隔类型，这一列包含这个属性类型的（声明的或隐式的）分数秒的精度，也就是秒值的小数点后的十进制位数。对于所有其他数据类型，这一列为空。
interval_type	character_data	如果data_type标识一种时间间隔类型，这一列包含时间间隔为这个属性包括哪些域的声明，例如YEAR TO MONTH、DAY TO SECOND等等。如果没有指定域限制（也就是该时间间隔接受所有域），并且对于所有其他数据类型，这个域为空。
interval_precision	cardinal_number	对应PostgreSQL不支持的特性（关于interval类型属性的分数秒精度，见datetime_precision）
attribute_udt_catalog	sql_identifier	属性数据类型被定义的数据库名（总是当前数据库）
attribute_udt_schema	sql_identifier	属性数据类型被定义的模式名
attribute_udt_name	sql_identifier	属性数据类型的名称
scope_catalog	sql_identifier	对应PostgreSQL不支持的特性
scope_schema	sql_identifier	对应PostgreSQL不支持的特性
scope_name	sql_identifier	对应PostgreSQL不支持的特性
maximum_cardinality	cardinal_number	总是为空，因为PostgreSQL中数组的最大基数始终不受限制
dtd_identifier	sql_identifier	该列的数据类型描述符的一个标识符，在从属于该表的数据类型标识符之中唯一。这主要用于与这类标识符的其他实例进行连接

名称	数据类型	描述
		（该标识符的指定格式没有被定义并且不保证在未来的版本中保持相同）。
is_derived_reference_attribute	yes_or_no	对应 PostgreSQL 不支持的特性

关于某些列的详情，参见第 34.16 节之下的一个相似结构的视图。

34.7. character_sets

视图character_sets标识当前数据库中可用的字符集。因为 PostgreSQL 不支持在同一个数据库中有多个字符集，这个视图只显示一个字符集，它就是数据库编码。

请注意 SQL 标准中以下术语的用法：

character repertoire

一个抽象的字符集合，例如UNICODE，UCS或 LATIN1。不作为SQL对象公开，但在此视图中可见。

character encoding form

一种字符集的编码形式。大多数旧的字符集只使用一种编码形式，
因此它们没有单独的名称（例如，LATIN1是适用于LATIN1字符集的编码形式）。
但是例如Unicode有编码形式UTF8，UTF16等（并非所有都受PostgreSQL支持）。
编码形式不作为SQL对象公开，但在此视图中可见。

character set

一个命名的SQL对象，用于标识字符集、字符编码和默认排序规则。
预定义的字符集通常与编码形式同名，但用户可以定义其他名称。例如，字符集UTF8
通常会标识字符集 UCS，编码形式UTF8和一些默认排序规则。

在 PostgreSQL 中，可以把“编码”视为字符集，也可以视为字符编码形式。它们具有相同的名称，并且一个数据库中只能有一个。

表 34.5. character_sets 列

名称	数据类型	描述
character_set_catalog	sql_identifier	当前字符集并未被实现为模式对象，因此这一列为空。
character_set_schema	sql_identifier	当前字符集并未被实现为模式对象，因此这一列为空。
character_set_name	sql_identifier	该字符集的名字，当前实现为显示该数据库编码的名字
character_repertoire	sql_identifier	字符集合；如果编码为UTF8则显示UCS，否则仅显示编码名称
form_of_use	sql_identifier	字符编码形式，与数据库编码相同
default_collate_catalog	sql_identifier	包含该默认排序规则的数据库名（如果任意排序规则被标识，总是当前数据库）
default_collate_schema	sql_identifier	包含该默认排序规则的模式名
default_collate_name	sql_identifier	默认排序规则的名字。该默认排序规则被标识为匹配当前数据库的COLLATE和CTYPE设置的排

名称	数据类型	描述
		序规则。如果没有这样的排序规则，则这一列以及相关的模式列和目录列都为空。

34.8. check_constraint_routine_usage

视图check_constraint_routine_usage标识检查约束所使用的例程（函数和过程）。只有当前启用角色拥有的那些例程才会显示出来。

表 34.6. check_constraint_routine_usage 列

名称	数据类型	描述
constraint_catalog	sql_identifier	包含约束的数据库名称(总是当前数据库)
constraint_schema	sql_identifier	包含约束的模式的名称
constraint_name	sql_identifier	约束的名称
specific_catalog	sql_identifier	包含该函数的数据库名称(总是当前数据库)
specific_schema	sql_identifier	包含函数的模式的名称
specific_name	sql_identifier	函数的“特定名称”。详见第 34.40 节。

34.9. check_constraints

视图check_constraints包含所有检查约束，无论它们是定义在表上还是定义在域上，只要它们归当前启用角色所有即可显示（表或域的拥有者就是约束的拥有者）。

表 34.7. check_constraints 列

名称	数据类型	描述
constraint_catalog	sql_identifier	包含约束的数据库名称(总是当前数据库)
constraint_schema	sql_identifier	包含约束的模式的名称
constraint_name	sql_identifier	约束的名称
check_clause	character_data	该检查约束的检查表达式

34.10. collations

视图collations包含在当前数据库中可用的排序规则。

表 34.8. collations 列

名称	数据类型	描述
collation_catalog	sql_identifier	包含该排序规则的数据库名称（总是当前数据库）
collation_schema	sql_identifier	包含该排序规则的模式名称
collation_name	sql_identifier	默认排序规则的名称
pad_attribute	character_data	总是NO PAD（另一种选择PAD SPACE没有被 PostgreSQL 支持）

34.11. collation_character_set_applicability

视图collation_character_set_applicability
标识可用的排序规则适用于哪些字符集。在 PostgreSQL 中，每个数据库中只有一种字符集（解释见第 34.7 节），因此这个视图没有提供很有用的信息。

表 34.9. collation_character_set_applicability 列

名称	数据类型	描述
collation_catalog	sql_identifier	包含该排序规则的数据库名称（总是当前数据库）
collation_schema	sql_identifier	包含该排序规则的模式名称
collation_name	sql_identifier	默认排序规则的名称
character_set_catalog	sql_identifier	字符集当前还未被实现为模式对象，所以这一列为空
character_set_schema	sql_identifier	字符集当前还未被实现为模式对象，所以这一列为空
character_set_name	sql_identifier	字符集名称

34.12. column_domain_usage

视图column_domain_usage
标识所有使用定义在当前数据库中并且被一个当前启用的角色拥有的域的列（表列或视图列）。

表 34.10. column_domain_usage 列

名称	数据类型	描述
domain_catalog	sql_identifier	包含该域的数据库名称（总是当前数据库）
domain_schema	sql_identifier	包含该域的模式名称
domain_name	sql_identifier	域名称
table_catalog	sql_identifier	包含表的数据库的名称(总是当前数据库)
table_schema	sql_identifier	包含表的模式的名称
table_name	sql_identifier	表的名称
column_name	sql_identifier	列名称

34.13. column_options

视图column_options包含为当前数据库中外部表列定义的所有选项。
只有当前用户能够访问（作为拥有者或具有某些权限）的那些外部表列才被显示。

表 34.11. column_options 列

名称	数据类型	描述
table_catalog	sql_identifier	包含该外部表的数据库名称（总是当前数据库）
table_schema	sql_identifier	包含该外部表的模式名称

名称	数据类型	描述
table_name	sql_identifier	外部表名称
column_name	sql_identifier	列名称
option_name	sql_identifier	一个选项名称
option_value	character_data	该选项的值

34.14. column_privileges

视图column_privileges
标识所有授予给一个当前启用的角色或者被一个当前启用的角色授予的权限。对每一个列、授予者、被授予者的组合只有一行。

如果一个权限被授予在一整个表上，它在这个视图中被显示为在每一列上授予，但是只有可用于列粒度的权限类型才会这样：SELECT、INSERT、UPDATE、REFERENCES。

表 34.12. column_privileges 列

名称	数据类型	描述
grantor	sql_identifier	授予该权限的角色名称
grantee	sql_identifier	被授予该权限的角色名称
table_catalog	sql_identifier	包含该列的表所在的数据库名称（总是当前数据库）
table_schema	sql_identifier	包含该列的表所在的模式名称
table_name	sql_identifier	包含该列的表名称
column_name	sql_identifier	列名称
privilege_type	character_data	权限类型：SELECT、INSERT、UPDATE或 REFERENCES
is_grantable	yes_or_no	如果该权限是可授予的，则为YES，否则为NO

34.15. column_udt_usage

视图column_udt_usage标识所有使用被一个当前启用的角色拥有的数据类型的列。
注意在PostgreSQL中，内置数据类型的行为和用户定义的类型相似，因此它们也被包括在这里。详见第 34.16 节。

表 34.13. column_udt_usage 列

名称	数据类型	描述
udt_catalog	sql_identifier	该列数据类型（如果适用，底层的域类型）被定义的数据库名称（总是当前数据库）
udt_schema	sql_identifier	该列数据类型（如果适用，底层的域类型）被定义的模式名称
udt_name	sql_identifier	该列数据类型（如果适用，底层的域类型）的名称
table_catalog	sql_identifier	包含表的数据库的名称(总是当前数据库)
table_schema	sql_identifier	包含表的模式的名称
table_name	sql_identifier	表的名称
column_name	sql_identifier	列名称

34.16. columns

视图columns包含数据库中有关所有表列（或视图列）的信息。系统列（oid等）不被包括在内。只有那些当前用户能够访问（作为拥有者或具有某些权限）的列才被显示。

表 34.14. columns 列

名称	数据类型	描述
table_catalog	sql_identifier	包含表的数据库的名称(总是当前数据库)
table_schema	sql_identifier	包含表的模式的名称
table_name	sql_identifier	表的名称
column_name	sql_identifier	列名称
ordinal_position	cardinal_number	该列在表内的顺序位置（从 1 开始计）
column_default	character_data	该列的默认表达式
is_nullable	yes_or_no	如果该列可以为空，则为YES，否则为NO。一个非空约束是让一列成为不能为空的方法，但还有其他方法。
data_type	character_data	如果该列的数据类型是一种内置类型，则为该列的数据类型；如果是某种数组（此种情况见视图element_types），则为ARRAY；否则为USER-DEFINED（此种情况下该类型被标识在udt_name和相关列中）。如果该列基于一个域，这一列引用该域底层的类型（该列被标识在domain_name和相关列中）。
character_maximum_length	cardinal_number	如果data_type标识一个字符或位串类型，这里是声明的最大长度；如果没有声明最大长度，则对于所有其他数据类型为空。
character_octet_length	cardinal_number	如果data_type标识一个字符类型，这里是一个数据的最大可能长度（以字节计）；对其他所有数据类型为空。最大字节长度取决于声明的字符最大长度（见上文）和服务器编码。
numeric_precision	cardinal_number	如果data_type标识一种数字类型，这列包含这个属性类型的（声明的或隐式的）精度。精度指示了有效位数。它可以按照列numeric_precision_radix中指定的被表示为十进制（基于10）或二进制（基于 2）。对于所有其他数据类型，这一列为空。
numeric_precision_radix	cardinal_number	如果data_type标识一种数字类型，这一列指示numeric_precision和numeric_scale列中的值是基于什么来表示。该值为 2 或 10。对于所有其他数据类型，这一列为空。
numeric_scale	cardinal_number	如果data_type标识一种准确数字类型，这列包含这个属性类

名称	数据类型	描述
		型的（声明的或隐式的）比例。标度表示小数点右侧的有效数字位数。它可以按照列numeric_precision_radix中指定的被表示为十进制（基于 10）或二进制（基于 2）。对于所有其他数据类型，这一列为空。
datetime_precision	cardinal_number	如果data_type标识一种日期、时间、时间戳或时间间隔类型，这一列包含这个属性类型的（声明的或隐式的）分数秒的精度，也就是秒值的小数点后的十进制位数。对于所有其他数据类型，这一列为空。
interval_type	character_data	如果data_type标识一种时间间隔类型，这一列包含时间间隔为这个属性包括哪些域的声明，例如YEAR TO MONTH、DAY TO SECOND等等。如果没有指定域限制（也就是该时间间隔接受所有域），并且对于所有其他数据类型，这个域为空。
interval_precision	cardinal_number	对应 PostgreSQL 不支持的特性（关于时间间隔类型属性的分数秒精度可见datetime_precision）
character_set_catalog	sql_identifier	对应 PostgreSQL 不支持的特性
character_set_schema	sql_identifier	对应 PostgreSQL 不支持的特性
character_set_name	sql_identifier	对应 PostgreSQL 不支持的特性
collation_catalog	sql_identifier	包含该列排序规则的数据库名称（总是当前数据库）；如果使用默认排序规则，或该列的数据类型不支持排序规则，则为 null。
collation_schema	sql_identifier	包含该列排序规则的模式名称；如果使用默认排序规则，或该列的数据类型不支持排序规则，则为 null。
collation_name	sql_identifier	该列排序规则的名称；如果使用默认排序规则，或该列的数据类型不支持排序规则，则为 null。
domain_catalog	sql_identifier	如果该列有一个域类型，这里是该域所在的数据库名（总是当前数据库），否则为空。
domain_schema	sql_identifier	如果该列有一个域类型，这里是该域所在的模式名，否则为空。
domain_name	sql_identifier	如果该列有一个域类型，这里是该域的名称，否则为空。
udt_catalog	sql_identifier	该列数据类型（如果适用，底层的域类型）被定义的数据库名称（总是当前数据库）
udt_schema	sql_identifier	该列数据类型（如果适用，底层的域类型）被定义的模式名称
udt_name	sql_identifier	该列数据类型（如果适用，底层的域类型）的名称
scope_catalog	sql_identifier	对应 PostgreSQL 不支持的特性

名称	数据类型	描述
scope_schema	sql_identifier	对应 PostgreSQL 不支持的特性
scope_name	sql_identifier	对应 PostgreSQL 不支持的特性
maximum_cardinality	cardinal_number	总是为空，因为 PostgreSQL 中数组的最大基数始终不受限制
dtd_identifier	sql_identifier	该列的数据类型描述符的一个标识符，在从属于该表的数据类型标识符之中唯一。这主要用于与这类标识符的其他实例进行连接（该标识符的指定格式没有被定义并且不保证在未来的版本中保持相同）。
is_self_referencing	yes_or_no	对应 PostgreSQL 不支持的特性
is_identity	yes_or_no	Applies to a feature not supported by PostgreSQL
identity_generation	character_data	Applies to a feature not supported by PostgreSQL
identity_start	character_data	Applies to a feature not supported by PostgreSQL
identity_increment	character_data	Applies to a feature not supported by PostgreSQL
identity_maximum	character_data	Applies to a feature not supported by PostgreSQL
identity_minimum	character_data	Applies to a feature not supported by PostgreSQL
identity_cycle	yes_or_no	Applies to a feature not supported by PostgreSQL
is_generated	character_data	对应 PostgreSQL 不支持的特性
generation_expression	character_data	对应 PostgreSQL 不支持的特性
is_updatable	yes_or_no	如果该列是可更新的，则为YES，否则为NO（基表中的列总是可更新的，视图中的列则不一定）

因为在 SQL 中有多种方式定义数据类型，而PostgreSQL还包含额外的方式来定义数据类型，它们在信息模式中的表示可能有点困难。列data_type应该标识列的底层内置类型。在PostgreSQL中，这表示定义在系统目录模式pg_catalog中的类型。如果应用能够特别地（例如以不同方式格式化数字类型或使用精度列中的数据）处理众所周知的内置类型，这列可能会有用。列udt_name、udt_schema和udt_catalog总是标识列的底层数据类型，即使该列是基于一个域的（因为PostgreSQL对待内置类型和用户定义类型的方式是一样的，内置类型也出现在这里。这是SQL 标准的一种扩展）。如果一个应用想要根据该类型以不同的方式处理数据，就应该使用这些列，因为在那种情况下即使该列真地基于一个域也没有关系。如果该列是基于一个域，该域的标识被存储在列domain_name、domain_schema和domain_catalog。如果你想要把列和它们相关的数据类型配对并且把域视作单独的类型，你可以写coalesce(domain_name, udt_name)等等。

34.17. constraint_column_usage

视图constraint_column_usage标识当前数据库中被某种约束使用的所有列。只显示那些位于当前启用角色所拥有表中的列。对于检查约束，此视图标识检查表达式中使用的列。对于非空约束，此视图标识定义该约束的列。对于外键约束，此视图标识外键所引用的列。对于唯一约束或主键约束，此视图标识受该约束限制的列。

表 34.15. constraint_column_usage 列

名称	数据类型	描述
table_catalog	sql_identifier	包含被某个约束使用的列的表所在的数据库名称（总是当前数据库）
table_schema	sql_identifier	包含被某个约束使用的列的表所在的模式名称
table_name	sql_identifier	包含被某个约束使用的列的表名称
column_name	sql_identifier	包含被某个约束使用的列名称
constraint_catalog	sql_identifier	包含该约束的数据库名称（总是当前数据库）
constraint_schema	sql_identifier	包含该约束的模式名称
constraint_name	sql_identifier	约束的名称

34.18. constraint_table_usage

视图constraint_table_usage
标识在当前数据库中被某个约束使用的所有表（这与视图table_constraints不同，
它标识哪些表约束定义在哪些表上）。对于一个外键约束，这个视图标识该外键引用的表。
对于一个唯一或主键约束，这个视图仅标识该约束属于的表。
检查约束和非空约束不被包括在这个视图中。

表 34.16. constraint_table_usage 列

名称	数据类型	描述
table_catalog	sql_identifier	包含被某个约束使用的表的数据库名称（总是当前数据库）
table_schema	sql_identifier	包含被某个约束使用的表的模式名称
table_name	sql_identifier	包含被某个约束使用的表名称
constraint_catalog	sql_identifier	包含该约束的数据库名称（总是当前数据库）
constraint_schema	sql_identifier	包含该约束的模式名称
constraint_name	sql_identifier	约束的名称

34.19. data_type_privileges

视图data_type_privileges
标识当前用户能够访问（作为被描述对象的拥有者或者具有其上的某种权限）
的所有数据类型描述符。只要一个数据类型被用在一个表列、
一个域或一个函数（作为参数或返回类型）
就会生成一个数据类型描述符并且在那个实例中存储一些有关该数据类型如何被使用的信息
（例如，声明的最大长度，如果适用）。每一个数据类型描述符被赋予一个任意的标识符，
它在被赋予给一个对象（表、域、函数）的数据类型描述符中唯一。
这个视图对于应用可能没什么用，但是它被用于定义信息模式中的一些其他视图。

表 34.17. data_type_privileges 列

名称	数据类型	描述
object_catalog	sql_identifier	包含该被描述对象的数据库名称（总是当前数据库）

名称	数据类型	描述
object_schema	sql_identifier	包含该被描述对象的模式名称
object_name	sql_identifier	被描述对象的名称
object_type	character_data	被描述对象的类型：TABLE（从属于表的一列的数据类型描述符）、DOMAIN（从属于域的数据类型描述符）、ROUTINE（从属于函数的一个参数或返回数据类型的数据类型描述符）。
dtd_identifier	sql_identifier	数据类型描述符的标识符，它在同一对象的数据类型描述符之间唯一。

34.20. domain_constraints

视图domain_constraints包含所有属于当前数据库中定义的域的约束。
只有当前用户能访问的那些域才被显示（作为拥有者或具有某些权限）。

表 34.18. domain_constraints 列

名称	数据类型	描述
constraint_catalog	sql_identifier	包含该约束的数据库名称（总是当前数据库）
constraint_schema	sql_identifier	包含该约束的模式名称
constraint_name	sql_identifier	约束的名称
domain_catalog	sql_identifier	包含该域的数据库名称（总是当前数据库）
domain_schema	sql_identifier	包含该域的模式名称
domain_name	sql_identifier	域名称
is_deferrable	yes_or_no	如果该约束是可延迟的，则为YES，否则为NO
initially_deferred	yes_or_no	如果该约束是可延迟的且初始就被延迟，则为YES，否则为NO

34.21. domain_udt_usage

视图domain_udt_usage标识所有基于被一个当前启用的角色拥有的数据类型的域。
注意在PostgreSQL中，内置数据类型的行为相似于用户定义的类型，
因此它们也被包括在这里。

表 34.19. domain_udt_usage 列

名称	数据类型	描述
udt_catalog	sql_identifier	该域数据类型被定义的数据库名称（总是当前数据库）
udt_schema	sql_identifier	该域数据类型被定义的模式名称
udt_name	sql_identifier	该域数据类型的名称
domain_catalog	sql_identifier	包含该域的数据库名称（总是当前数据库）
domain_schema	sql_identifier	包含该域的模式名称
domain_name	sql_identifier	域名称

34.22. domains

视图domains包含当前数据库中定义的所有域。
仅显示当前用户有权限访问的域（通过拥有者或某些权限）。

表 34.20. domains 列

名称	数据类型	描述
domain_catalog	sql_identifier	包含该域的数据库名称（总是当前数据库）
domain_schema	sql_identifier	包含该域的模式名称
domain_name	sql_identifier	域名称
data_type	character_data	该域的数据类型如果是一种内置类型，这里是该域的数据类型；如果是某种数组（此种情况见视图element_types），则为ARRAY；否则为USER-DEFINED（此种情况中，该类型被标识在udt_name和相关列中）。
character_maximum_length	cardinal_number	如果该域有一个字符或位串类型，这里是声明的最大长度；如果没有声明最大长度，则对于所有其他数据类型为空。
character_octet_length	cardinal_number	如果该域有一个字符类型，这里是一个数据的最大可能长度（以字节计）；对其他所有数据类型为空。最大字节长度取决于声明的字符最大长度（见上文）和服务器编码。
character_set_catalog	sql_identifier	对应 PostgreSQL 不支持的特性
character_set_schema	sql_identifier	对应 PostgreSQL 不支持的特性
character_set_name	sql_identifier	对应 PostgreSQL 不支持的特性
collation_catalog	sql_identifier	包含该域排序规则的数据库名称（总是当前数据库）；如果使用默认排序规则，或该域的数据类型不支持排序规则，则为 null。
collation_schema	sql_identifier	包含该域排序规则的模式名称；如果使用默认排序规则，或该域的数据类型不支持排序规则，则为 null。
collation_name	sql_identifier	该域排序规则的名称；如果使用默认排序规则，或该域的数据类型不支持排序规则，则为 null。
numeric_precision	cardinal_number	如果该域有一种数字类型，这列包含这个域类型的（声明的或隐式的）精度。精度指示了有效位数。它可以按照列numeric_precision_radix中指定的被表示为十进制（基于 10）或二进制（基于 2）。对于所有其他数据类型，这一列为空。
numeric_precision_radix	cardinal_number	如果该域有一种数字类型，这一列指示numeric_precision和numeric_scale列中的值是基于什么来表示。该值为 2 或 10。

名称	数据类型	描述
		对于所有其他数据类型，这一列为空。
numeric_scale	cardinal_number	如果该域有一种准确数字类型，这列包含这个域类型的（声明的或隐式的）比例。标度表示小数点右侧的有效数字位数。它可以按照列numeric_precision_radix中指定的被表示为十进制（基于 10）或二进制（基于 2）。对于所有其他数据类型，这一列为空。
datetime_precision	cardinal_number	如果data_type标识一种日期、时间、时间戳或时间间隔类型，这一列包含这个域类型的（声明的或隐式的）分数秒的精度，也就是秒值的小数点后的十进制位数。对于所有其他数据类型，这一列为空。
interval_type	character_data	如果data_type标识一种时间间隔类型，这一列包含时间间隔为这个域包括哪些域的声明，例如YEAR TO MONTH、DAY TO SECOND等等。如果没有指定域限制（也就是该时间间隔接受所有域），并且对于所有其他数据类型，这个域为空。
interval_precision	cardinal_number	对应 PostgreSQL 不支持的特性（关于时间间隔类型域的分数秒精度可见datetime_precision）
domain_default	character_data	该域的默认表达式
udt_catalog	sql_identifier	该域数据类型被定义的数据库名称（总是当前数据库）
udt_schema	sql_identifier	该域数据类型被定义的模式名称
udt_name	sql_identifier	该域数据类型的名称
scope_catalog	sql_identifier	对应 PostgreSQL 不支持的特性
scope_schema	sql_identifier	对应 PostgreSQL 不支持的特性
scope_name	sql_identifier	对应 PostgreSQL 不支持的特性
maximum_cardinality	cardinal_number	总是为空，因为 PostgreSQL 中数组的最大基数始终不受限制
dtd_identifier	sql_identifier	该域的数据类型描述符的一个标识符，在从属于该域的数据类型标识符之中唯一（这不重要，因为一个域只包含一个数据类型描述符）。这主要用于与这类标识符的其他实例进行连接（该标识符的指定格式没有被定义并且不保证在未来的版本中保持相同）。

34.23. element_types

视图element_types包含数组元素的数据类型描述符。当一个表列、复合类型属性、域、函数参数或函数返回值被定义为一种数组类型时，相应的信息模式视图只在列data_type

中包含ARRAY。要获得该数组元素类型的信息，你可以连接该相应的视图和这个视图。例如，要显示一个表的列及其数据类型和数组元素类型（如果适用），你可以：

```
SELECT c.column_name, c.data_type, e.data_type AS element_type
FROM information_schema.columns c LEFT JOIN information_schema.
element_types e
     ON ((c.table_catalog, c.table_schema, c.table_name, 'TABLE',
 c.dtd_identifier)
       = (e.object_catalog, e.object_schema, e.object_name, e.
object_type, e.collection_type_identifier))
WHERE c.table_schema = '...' AND c.table_name = '...'
ORDER BY c.ordinal_position;
```

这个视图只包括当前用户能够访问（作为拥有者或具有某些权限）的对象。

表 34.21. element_types 列

名称	数据类型	描述
object_catalog	sql_identifier	包含使用被描述的数组的对象的数据库名称（总是当前数据库）
object_schema	sql_identifier	包含使用被描述的数组的对象的模式名称
object_name	sql_identifier	使用被描述的模式的对象名称
object_type	character_data	使用被描述的数组的对象的类型：TABLE（被一个表列使用的数组）、USER-DEFINED TYPE（被复合类型的一个属性使用的数组）、DOMAIN（被域使用的数组）、ROUTINE（被函数的一个参数或返回数据类型使用的数组）。
collection_type_identifier	sql_identifier	被描述的数组的数据类型描述符的标识符。使用这个去与其他信息模式视图的dtd_identifier列连接。
data_type	character_data	如果数组元素的数据类型是内置类型，这里是数组元素的数据类型，否则为USER-DEFINED（在那种情况下，该类型被标识在udt_name和相关列中）。
character_maximum_length	cardinal_number	总是为空，因为这种信息不适用于PostgreSQL中的数组元素数据类型
character_octet_length	cardinal_number	总是为空，因为这种信息不适用于PostgreSQL中的数组元素数据类型
character_set_catalog	sql_identifier	对应 PostgreSQL 不支持的特性
character_set_schema	sql_identifier	对应 PostgreSQL 不支持的特性
character_set_name	sql_identifier	对应 PostgreSQL 不支持的特性
collation_catalog	sql_identifier	包含该元素类型排序规则的数据库名称（总是当前数据库）；如果使用默认排序规则，或该元素类型不支持排序规则，则为null。
collation_schema	sql_identifier	包含该元素类型排序规则的模式名称；如果使用默认排序规则，

名称	数据类型	描述
		或该元素类型不支持排序规则，则为 null。
collation_name	sql_identifier	该元素类型排序规则的名称；如果使用默认排序规则，或该元素类型不支持排序规则，则为 null。
numeric_precision	cardinal_number	总是为空，因为这种信息不适用于PostgreSQL中的数组元素数据类型
numeric_precision_radix	cardinal_number	总是为空，因为这种信息不适用于PostgreSQL中的数组元素数据类型
numeric_scale	cardinal_number	总是为空，因为这种信息不适用于PostgreSQL中的数组元素数据类型
datetime_precision	cardinal_number	总是为空，因为这种信息不适用于PostgreSQL中的数组元素数据类型
interval_type	character_data	总是为空，因为这种信息不适用于PostgreSQL中的数组元素数据类型
interval_precision	cardinal_number	总是为空，因为这种信息不适用于PostgreSQL中的数组元素数据类型
domain_default	character_data	尚未实现
udt_catalog	sql_identifier	元素的数据类型所在的数据库名称（总是当前数据库）
udt_schema	sql_identifier	定义元素的数据类型的模式的名称
udt_name	sql_identifier	元素的数据类型名
scope_catalog	sql_identifier	对应 PostgreSQL 不支持的特性
scope_schema	sql_identifier	对应 PostgreSQL 不支持的特性
scope_name	sql_identifier	对应 PostgreSQL 不支持的特性
maximum_cardinality	cardinal_number	总是为空，因为 PostgreSQL 中数组的最大基数始终不受限制
dtd_identifier	sql_identifier	该元素的数据类型描述符的标识符。当前没有用。

34.24. enabled_roles

视图enabled_roles标识当前“已启用的角色”。已启用的角色递归地定义为：当前用户，以及所有通过自动继承方式授予给已启用角色的角色。换句话说，这些角色都是当前用户直接或间接自动继承其成员资格的角色。

对于权限检查，使用的是一组“适用角色”，其范围可能比已启用角色的集合更广。因此，一般来说最好使用视图applicable_roles而不是这个视图；另见该处。

表 34.22. enabled_roles 列

名称	数据类型	描述
role_name	sql_identifier	一个角色的名称

34.25. foreign_data_wrapper_options

视图foreign_data_wrapper_options
包含为当前数据库中外部数据包装器定义的所有选项。
只有那些当前用户能够访问（作为拥有者或具有某些权限）的外部数据包装器被显示。

表 34.23. foreign_data_wrapper_options 列

名称	数据类型	描述
foreign_data_wrapper_catalog	sql_identifier	该外部数据包装器所在的数据库名称（总是当前数据库）
foreign_data_wrapper_name	sql_identifier	该外部数据包装器的名称
option_name	sql_identifier	一个选项名称
option_value	character_data	该选项的值

34.26. foreign_data_wrappers

视图foreign_data_wrappers包含定义在当前数据库中的所有外部数据包装器。
只有那些当前用户能够访问（作为拥有者或具有某些权限）的外部数据包装器才会被显示。

表 34.24. foreign_data_wrappers 列

名称	数据类型	描述
foreign_data_wrapper_catalog	sql_identifier	包含该外部数据包装器的数据库名称（总是当前数据库）
foreign_data_wrapper_name	sql_identifier	该外部数据包装器的名称
authorization_identifier	sql_identifier	外部服务器拥有者的名称
library_name	character_data	实现这个外部数据包装器的库文件名称
foreign_data_wrapper_language	character_data	用于实现这个外部数据包装器的语言

34.27. foreign_server_options

视图foreign_server_options包含为当前数据库中外部服务器定义的所有选项。
只有那些当前用户能够访问（作为拥有者或具有某些权限）的外部服务器才会被显示。

表 34.25. foreign_server_options 列

名称	数据类型	描述
foreign_server_catalog	sql_identifier	该外部服务器所在的数据库名称（总是当前数据库）
foreign_server_name	sql_identifier	该外部服务器的名称
option_name	sql_identifier	一个选项名称
option_value	character_data	该选项的值

34.28. foreign_servers

视图foreign_servers包含当前数据库中定义的所有外部服务器。
只有那些当前用户能够访问（作为拥有者或具有某些权限）的外部服务器才会被显示。

表 34.26. foreign_servers 列

名称	数据类型	描述
foreign_server_catalog	sql_identifier	该外部服务器所在的数据库名称（总是当前数据库）
foreign_server_name	sql_identifier	该外部服务器的名称
foreign_data_wrapper_catalog	sql_identifier	包含被该外部服务器使用的外部数据包装器的数据库名称（总是当前数据库）
foreign_data_wrapper_name	sql_identifier	被该外部服务器所使用的外部数据包装器的名称
foreign_server_type	character_data	外部服务器类型信息，如果在创建时指定过
foreign_server_version	character_data	外部服务器版本信息，如果在创建时指定过
authorization_identifier	sql_identifier	外部服务器拥有者的名称

34.29. foreign_table_options

视图foreign_table_options包含为当前数据库中外部表定义的所有选项。
只有那些当前用户能够访问（作为拥有者或具有某些权限）的外部表才会被显示。

表 34.27. foreign_table_options 列

名称	数据类型	描述
foreign_table_catalog	sql_identifier	包含该外部表的数据库名称（总是当前数据库）
foreign_table_schema	sql_identifier	包含该外部表的模式名称
foreign_table_name	sql_identifier	外部表的名称
option_name	sql_identifier	一个选项名称
option_value	character_data	该选项的值

34.30. foreign_tables

视图foreign_tables包含定义在当前数据库中的所有外部表。
只有那些当前用户能够访问（作为拥有者或具有某些权限）的外部表才会被显示。

表 34.28. foreign_tables 列

名称	数据类型	描述
foreign_table_catalog	sql_identifier	该外部表所在的数据库名称（总是当前数据库）
foreign_table_schema	sql_identifier	包含该外部表的模式名称
foreign_table_name	sql_identifier	外部表的名称
foreign_server_catalog	sql_identifier	该外部服务器所在的数据库名称（总是当前数据库）
foreign_server_name	sql_identifier	该外部服务器的名称

34.31. key_column_usage

视图key_column_usage标识当前数据库中所有被某种唯一、主键或外键约束限制的列。
检查约束不被包括在这个视图中。
只有那些当前用户能够访问的列才会被显示（作为拥有者或具有某些权限）。

表 34.29. key_column_usage 列

名称	数据类型	描述
constraint_catalog	sql_identifier	包含该约束的数据库名称（总是当前数据库）
constraint_schema	sql_identifier	包含该约束的模式名称
constraint_name	sql_identifier	约束的名称
table_catalog	sql_identifier	包含被这个约束限制的列的表所在的数据库名称（总是当前数据库）
table_schema	sql_identifier	包含被这个约束限制的列的表所在的模式名称
table_name	sql_identifier	包含被这个约束限制的列的表的名称
column_name	sql_identifier	被这个约束限制的列的名称
ordinal_position	cardinal_number	该列在约束键中的顺序位置（从 1 开始计数）
position_in_unique_constraint	cardinal_number	对于一个外键约束，被引用行在其唯一约束中的顺序位置（从 1 开始计数）；对于其他约束为空

34.32. parameters

视图parameters包含当前数据库中所有函数的参数的有关信息。
只有那些当前用户能够访问（作为拥有者或具有某些权限）的函数才会被显示。

表 34.30. parameters 列

名称	数据类型	描述
specific_catalog	sql_identifier	包含该函数的数据库名称(总是当前数据库)
specific_schema	sql_identifier	包含函数的模式的名称
specific_name	sql_identifier	函数的“特定名称”。详见第 34.40 节。
ordinal_position	cardinal_number	该参数在函数参数列表中的顺序位置（从 1 开始计数）
parameter_mode	character_data	IN表示输入参数，OUT表示输出参数，INOUT表示输入/输出参数。
is_result	yes_or_no	对应 PostgreSQL 不支持的特性
as_locator	yes_or_no	对应 PostgreSQL 不支持的特性
parameter_name	sql_identifier	参数名称，如果参数没有名称则为空
data_type	character_data	该参数的数据类型如果是一种内置类型，这里是该参数的数据类型；如果是某种数组（此种情况见视图element_types），则为ARRAY；否则为USER-DEFINED（此种情况中，该类型被标识在udt_name和相关列中）。
character_maximum_length	cardinal_number	总是为空，因为这种信息不适用于PostgreSQL中的返回数据类型

名称	数据类型	描述
character_octet_length	cardinal_number	总是为空，因为这种信息不适用于PostgreSQL中的返回数据类型
character_set_catalog	sql_identifier	对应 PostgreSQL 不支持的特性
character_set_schema	sql_identifier	对应 PostgreSQL 不支持的特性
character_set_name	sql_identifier	对应 PostgreSQL 不支持的特性
collation_catalog	sql_identifier	总是为空，因为这种信息不适用于PostgreSQL中的返回数据类型
collation_schema	sql_identifier	总是为空，因为这种信息不适用于PostgreSQL中的返回数据类型
collation_name	sql_identifier	总是为空，因为这种信息不适用于PostgreSQL中的返回数据类型
numeric_precision	cardinal_number	总是为空，因为这种信息不适用于PostgreSQL中的返回数据类型
numeric_precision_radix	cardinal_number	总是为空，因为这种信息不适用于PostgreSQL中的返回数据类型
numeric_scale	cardinal_number	总是为空，因为这种信息不适用于PostgreSQL中的返回数据类型
datetime_precision	cardinal_number	总是为空，因为这种信息不适用于PostgreSQL中的返回数据类型
interval_type	character_data	总是为空，因为这种信息不适用于PostgreSQL中的返回数据类型
interval_precision	cardinal_number	总是为空，因为这种信息不适用于PostgreSQL中的返回数据类型
udt_catalog	sql_identifier	该参数的数据类型所在的数据库名称（总是当前数据库）
udt_schema	sql_identifier	该参数的数据类型所在的模式名称
udt_name	sql_identifier	该参数的数据类型的名称
scope_catalog	sql_identifier	对应 PostgreSQL 不支持的特性
scope_schema	sql_identifier	对应 PostgreSQL 不支持的特性
scope_name	sql_identifier	对应 PostgreSQL 不支持的特性
maximum_cardinality	cardinal_number	总是为空，因为 PostgreSQL 中数组的最大基数始终不受限制
dtd_identifier	sql_identifier	该参数的数据类型描述符的一个标识符，在从属于该函数的数据类型标识符之中唯一（这不重要，因为一个域只包含一个数据类型描述符）。这主要用于与这类标识符的其他实例进行连接（该标识符的指定格式没有被定义并且不保证在未来的版本中保持相同）。

34.33. referential_constraints

视图referential_constraints包含当前数据库中的所有引用（外键）约束。只有那些当前用户具有其引用表上写权限（作为拥有者或具有某些除SELECT之外的权限）的约束才会被显示。

表 34.31. referential_constraints 列

名称	数据类型	描述
constraint_catalog	sql_identifier	包含约束的数据库名称(总是当前数据库)
constraint_schema	sql_identifier	包含约束的模式的名称
constraint_name	sql_identifier	约束的名称
unique_constraint_catalog	sql_identifier	包含该外键约束所引用的唯一或主键约束的数据库名称（总是当前数据库）
unique_constraint_schema	sql_identifier	包含该外键约束所引用的唯一或主键约束的模式名称
unique_constraint_name	sql_identifier	包含该外键约束所引用的唯一或主键约束的名称
match_option	character_data	外键约束的匹配选项：FULL、PARTIAL或NONE。
update_rule	character_data	外键约束的更新规则：CASCADE、SET NULL、SET DEFAULT、RESTRICT或 NO ACTION。
delete_rule	character_data	外键约束的删除规则：CASCADE、SET NULL、SET DEFAULT、RESTRICT或 NO ACTION。

34.34. role_column_grants

视图role_column_grants标识所有在列上授予的权限，这些权限的授予者或者被授予者是当前已启用的角色。更多信息可以在column_privileges中找到。这个视图和column_privileges之间的唯一实质性区别是：这个视图忽略那些以授予给PUBLIC的方式使当前用户获得其访问权限的列。

表 34.32. role_column_grants 列

名称	数据类型	描述
grantor	sql_identifier	授予该权限的角色名称
grantee	sql_identifier	被授予该权限的角色名称
table_catalog	sql_identifier	包含该列的表所在的数据库名称（总是当前数据库）
table_schema	sql_identifier	包含该列的表所在的模式名称
table_name	sql_identifier	包含该列的表名称
column_name	sql_identifier	列名称
privilege_type	character_data	权限类型：SELECT、INSERT、UPDATE或 REFERENCES
is_grantable	yes_or_no	如果该权限是可授予的，则为YES，否则为NO

34.35. role_routine_grants

视图role_routine_grants标识所有在函数上授予的权限，
这些权限的授予者或者被授予者是当前已启用的角色。更多信息可以在routine_privileges中找到。这个视图和routine_privileges
之间的唯一实质性区别是：这个视图忽略那些以授予给PUBLIC
的方式使当前用户获得其访问权限的函数。

表 34.33. role_routine_grants 列

名称	数据类型	描述
grantor	sql_identifier	授予该权限的角色名称
grantee	sql_identifier	被授予该权限的角色名称
specific_catalog	sql_identifier	包含该函数的数据库名称(总是当前数据库)
specific_schema	sql_identifier	包含函数的模式的名称
specific_name	sql_identifier	函数的“特定名称”。详见第 34.40 节。
routine_catalog	sql_identifier	包含该函数的数据库名称(总是当前数据库)
routine_schema	sql_identifier	包含函数的模式的名称
routine_name	sql_identifier	该函数的名称（在重载的情况下可能会重复）
privilege_type	character_data	总是为EXECUTE（函数唯一的权限类型）
is_grantable	yes_or_no	如果该权限是可授予的，则为YES，否则为NO

34.36. role_table_grants

视图role_table_grants标识所有在表或视图上授予的权限，
这些权限的授予者或者被授予者是当前已启用的角色。更多信息可以在table_privileges中找到。这个视图和table_privileges
之间的唯一实质性区别是：这个视图忽略那些以授予给PUBLIC
的方式使当前用户获得其访问权限的表。

表 34.34. role_table_grants 列

名称	数据类型	描述
grantor	sql_identifier	授予该权限的角色名称
grantee	sql_identifier	被授予该权限的角色名称
table_catalog	sql_identifier	包含该表的数据库名称（总是当前数据库）
table_schema	sql_identifier	包含该表的模式名称
table_name	sql_identifier	表的名称
privilege_type	character_data	该权限的类型：SELECT、INSERT、UPDATE、DELETE、TRUNCATE、REFERENCES或TRIGGER
is_grantable	yes_or_no	如果该权限是可授予的，则为YES，否则为NO

名称	数据类型	描述
with_hierarchy	yes_or_no	在 SQL 标准中，WITH HIERARCHY OPTION是一个独立的（子）权限，它允许在表继承层级上的特定操作。在 PostgreSQL 中，这被包括在SELECT权限中，因此这一列在权限为SELECT时显示YES，其他时候显示NO。

34.37. role_udt_grants

视图role_udt_grants标识所有在用户定义类型上授予的USAGE权限，这些权限的授予者或者被授予者是当前已启用的角色。更多信息可以在udt_privileges中找到。这个视图和udt_privileges之间的唯一实质性区别是：这个视图忽略那些以授予给PUBLIC的方式使当前用户获得其访问权限的对象。因为数据类型在 PostgreSQL 中并没有真正的权限，而是只有一个给PUBLIC的隐式授予，这个视图为空。

表 34.35. role_udt_grants 列

名称	数据类型	描述
grantor	sql_identifier	授予该权限的角色名称
grantee	sql_identifier	被授予该权限的角色名称
udt_catalog	sql_identifier	包含该类型的数据库名称（总是当前数据库）
udt_schema	sql_identifier	包含该类型的模式名称
udt_name	sql_identifier	类型的名称
privilege_type	character_data	总是TYPE USAGE
is_grantable	yes_or_no	如果该权限是可授予的，则为YES，否则为NO

34.38. role_usage_grants

视图role_usage_grants标识所有在多种对象上授予的USAGE权限，这些权限的授予者或者被授予者是当前已启用的角色。更多信息可以在usage_privileges中找到。这个视图和usage_privileges之间的唯一实质性区别是：这个视图忽略那些以授予给PUBLIC的方式使当前用户获得其访问权限的对象。

表 34.36. role_usage_grants 列

名称	数据类型	描述
grantor	sql_identifier	授予该权限的角色名称
grantee	sql_identifier	被授予该权限的角色名称
object_catalog	sql_identifier	包含该对象的数据库名称（总是当前数据库）
object_schema	sql_identifier	如果适用，则为包含该对象的模式名称，否则为一个空字符串
object_name	sql_identifier	对象的名称
object_type	character_data	COLLATION或DOMAIN或FOREIGN DATA WRAPPER或FOREIGN SERVER或SEQUENCE

名称	数据类型	描述
privilege_type	character_data	总是USAGE
is_grantable	yes_or_no	如果该权限是可授予的，则为YES，否则为NO

34.39. routine_privileges

视图routine_privileges标识所有在函数上授予的权限，
其授予者或被授予者是当前已启用的角色。对于每一种函数、授予者和被授予者的组合，
这里都有一行。

表 34.37. routine_privileges 列

名称	数据类型	描述
grantor	sql_identifier	授予该权限的角色名称
grantee	sql_identifier	被授予该权限的角色名称
specific_catalog	sql_identifier	包含该函数的数据库名称(总是当前数据库)
specific_schema	sql_identifier	包含函数的模式的名称
specific_name	sql_identifier	函数的“特定名称”。详见第 34.40 节。
routine_catalog	sql_identifier	包含该函数的数据库名称(总是当前数据库)
routine_schema	sql_identifier	包含函数的模式的名称
routine_name	sql_identifier	该函数的名称（在重载的情况下可能会重复）
privilege_type	character_data	总是为EXECUTE（函数唯一的权限类型）
is_grantable	yes_or_no	如果该权限是可授予的，则为YES，否则为NO

34.40. routines

视图routines包含当前数据库中所有的函数。
只有那些当前用户能够访问（作为拥有者或具有某些权限）的函数才会被显示。

表 34.38. routines 列

名称	数据类型	描述
specific_catalog	sql_identifier	包含该函数的数据库名称(总是当前数据库)
specific_schema	sql_identifier	包含函数的模式的名称
specific_name	sql_identifier	该函数的“专用名”。这是一个在模式中唯一标识该函数的名称，即使该函数真正的名称已经被重载。专用名的格式尚未被定义，它应当仅被用来与指定例程名称的其他实例进行比较。
routine_catalog	sql_identifier	包含该函数的数据库名称(总是当前数据库)
routine_schema	sql_identifier	包含函数的模式的名称
routine_name	sql_identifier	该函数的名称（在重载的情况下可能会重复）

名称	数据类型	描述
routine_type	character_data	总是FUNCTION（将来可能会有其他类型的例程。）
module_catalog	sql_identifier	对应 PostgreSQL 不支持的特性
module_schema	sql_identifier	对应 PostgreSQL 不支持的特性
module_name	sql_identifier	对应 PostgreSQL 不支持的特性
udt_catalog	sql_identifier	对应 PostgreSQL 不支持的特性
udt_schema	sql_identifier	对应 PostgreSQL 不支持的特性
udt_name	sql_identifier	对应 PostgreSQL 不支持的特性
data_type	character_data	该函数的返回数据类型如果是一种内置类型，这里是该数据类型；如果是某种数组（此种情况见视图element_types），则为ARRAY；否则为USER-DEFINED（此种情况中，该类型被标识在type_udt_name和相关列中）。
character_maximum_length	cardinal_number	总是为空，因为这种信息不适用于PostgreSQL中的返回数据类型
character_octet_length	cardinal_number	总是为空，因为这种信息不适用于PostgreSQL中的返回数据类型
character_set_catalog	sql_identifier	对应 PostgreSQL 不支持的特性
character_set_schema	sql_identifier	对应 PostgreSQL 不支持的特性
character_set_name	sql_identifier	对应 PostgreSQL 不支持的特性
collation_catalog	sql_identifier	总是为空，因为这种信息不适用于PostgreSQL中的返回数据类型
collation_schema	sql_identifier	总是为空，因为这种信息不适用于PostgreSQL中的返回数据类型
collation_name	sql_identifier	总是为空，因为这种信息不适用于PostgreSQL中的返回数据类型
numeric_precision	cardinal_number	总是为空，因为这种信息不适用于PostgreSQL中的返回数据类型
numeric_precision_radix	cardinal_number	总是为空，因为这种信息不适用于PostgreSQL中的返回数据类型
numeric_scale	cardinal_number	总是为空，因为这种信息不适用于PostgreSQL中的返回数据类型
datetime_precision	cardinal_number	总是为空，因为这种信息不适用于PostgreSQL中的返回数据类型
interval_type	character_data	总是为空，因为这种信息不适用于PostgreSQL中的返回数据类型
interval_precision	cardinal_number	总是为空，因为这种信息不适用于PostgreSQL中的返回数据类型

名称	数据类型	描述
type_udt_catalog	sql_identifier	该函数的返回数据类型所在的数据库名（总是当前数据库）。
type_udt_schema	sql_identifier	该函数的返回数据类型所在的模式名。
type_udt_name	sql_identifier	该函数的返回数据类型的名字。
scope_catalog	sql_identifier	对应 PostgreSQL 不支持的特性
scope_schema	sql_identifier	对应 PostgreSQL 不支持的特性
scope_name	sql_identifier	对应 PostgreSQL 不支持的特性
maximum_cardinality	cardinal_number	总是为空，因为 PostgreSQL 中数组的最大基数始终不受限制
dtd_identifier	sql_identifier	该函数返回数据类型的数据类型描述符的一个标识符，在从属于该函数的数据类型标识符之中唯一。这主要用于与这类标识符的其他实例进行连接（该标识符的指定格式没有被定义并且不保证在未来的版本中保持相同）。
routine_body	character_data	如果该函数是一个 SQL 函数，则为SQL，否则为EXTERNAL。
routine_definition	character_data	该函数的源文本（如果该函数不归当前已启用的角色所有，则为空）（根据 SQL 标准，只有routine_body为SQL时这一列才适用。但是在PostgreSQL中，它将会包含该函数被创建时所指定的任何源文本。）
external_name	character_data	如果这个函数是一个 C 函数，则为该函数的外部名称（链接符号），否则为空（这会产生和显示在routine_definition中相同的值）。
external_language	character_data	该函数所用的语言
parameter_style	character_data	总是GENERAL（SQL 标准定义了其他参数风格，但在PostgreSQL中不可用）
is_deterministic	yes_or_no	如果该函数被声明为不变（在 SQL 标准中被称为确定性的），则为YES，否则为NO（你不能通过该信息模式查询在PostgreSQL中可用的其他易变性级别）。
sql_data_access	character_data	总是MODIFIES，表示该函数可能修改 SQL 数据。这种信息对PostgreSQL没有用处。
is_null_call	yes_or_no	如果该函数在任一参数为空时自动返回空值，则为YES，否则为NO。
sql_path	character_data	对应 PostgreSQL 不支持的特性
schema_level_routine	yes_or_no	总是YES（反例是一个用户定义类型的方法，这是在PostgreSQL不可用的一种特性）。
max_dynamic_result_sets	cardinal_number	对应 PostgreSQL 不支持的特性
is_user_defined_cast	yes_or_no	对应 PostgreSQL 不支持的特性
is_implicitly_invocable	yes_or_no	对应 PostgreSQL 不支持的特性

名称	数据类型	描述
security_type	character_data	如果该函数以当前用户的权限运行，则为INVOKER；如果该函数以定义它的用户的权限运行，则为DEFINER。
to_sql_specific_catalog	sql_identifier	对应 PostgreSQL 不支持的特性
to_sql_specific_schema	sql_identifier	对应 PostgreSQL 不支持的特性
to_sql_specific_name	sql_identifier	对应 PostgreSQL 不支持的特性
as_locator	yes_or_no	对应 PostgreSQL 不支持的特性
created	time_stamp	对应 PostgreSQL 不支持的特性
last_altered	time_stamp	对应 PostgreSQL 不支持的特性
new_savepoint_level	yes_or_no	对应 PostgreSQL 不支持的特性
is_udt_dependent	yes_or_no	当前总是NO。另一个选项YES对应 PostgreSQL 不支持的特性。
result_cast_from_data_type	character_data	对应 PostgreSQL 不支持的特性
result_cast_as_locator	yes_or_no	对应 PostgreSQL 不支持的特性
result_cast_char_max_length	cardinal_number	对应 PostgreSQL 不支持的特性
result_cast_char_octet_length	character_data	对应 PostgreSQL 不支持的特性
result_cast_char_set_catalog	sql_identifier	对应 PostgreSQL 不支持的特性
result_cast_char_set_schema	sql_identifier	对应 PostgreSQL 不支持的特性
result_cast_char_set_name	sql_identifier	对应 PostgreSQL 不支持的特性
result_cast_collation_catalog	sql_identifier	对应 PostgreSQL 不支持的特性
result_cast_collation_schema	sql_identifier	对应 PostgreSQL 不支持的特性
result_cast_collation_name	sql_identifier	对应 PostgreSQL 不支持的特性
result_cast_numeric_precision	cardinal_number	对应 PostgreSQL 不支持的特性
result_cast_numeric_precision_radix	cardinal_number	对应 PostgreSQL 不支持的特性
result_cast_numeric_scale	cardinal_number	对应 PostgreSQL 不支持的特性
result_cast_datetime_precision	character_data	对应 PostgreSQL 不支持的特性
result_cast_interval_type	character_data	对应 PostgreSQL 不支持的特性
result_cast_interval_precision	cardinal_number	对应 PostgreSQL 不支持的特性
result_cast_type_udt_catalog	sql_identifier	对应 PostgreSQL 不支持的特性
result_cast_type_udt_schema	sql_identifier	对应 PostgreSQL 不支持的特性
result_cast_type_udt_name	sql_identifier	对应 PostgreSQL 不支持的特性

名称	数据类型	描述
result_cast_scope_catalog	sql_identifier	对应 PostgreSQL 不支持的特性
result_cast_scope_schema	sql_identifier	对应 PostgreSQL 不支持的特性
result_cast_scope_name	sql_identifier	对应 PostgreSQL 不支持的特性
result_cast_maximum_cardinality	cardinal_number	对应 PostgreSQL 不支持的特性
result_cast_dtd_identifier	sql_identifier	对应 PostgreSQL 不支持的特性

34.41. schemata

视图schemata包含当前数据库中由当前启用角色所拥有的所有模式。

表 34.39. schemata 列

名称	数据类型	描述
catalog_name	sql_identifier	该模式所在的数据库名称（总是当前数据库）
schema_name	sql_identifier	模式的名称
schema_owner	sql_identifier	模式拥有者的名称
default_character_set_catalog	sql_identifier	对应 PostgreSQL 不支持的特性
default_character_set_schema	sql_identifier	对应 PostgreSQL 不支持的特性
default_character_set_name	sql_identifier	对应 PostgreSQL 不支持的特性
sql_path	character_data	对应 PostgreSQL 不支持的特性

34.42. sequences

视图sequences包含所有定义在当前数据库中的序列。
只有那些当前用户能够访问（作为拥有者或具有某些权限）的序列才会被显示。

表 34.40. sequences 列

名称	数据类型	描述
sequence_catalog	sql_identifier	包含该序列的数据库名称（总是当前数据库）
sequence_schema	sql_identifier	包含该序列的模式名称
sequence_name	sql_identifier	序列的名称
data_type	character_data	序列的数据类型。在 PostgreSQL 中，它目前总是 bigint。
numeric_precision	cardinal_number	这列包含这个序列数据类型（见上文）的（声明的或隐式的）精度。精度指示了有效位数。它可以按照列numeric_precision_radix中指定的被表示为十进制（基于 10）或二进制（基于 2）。
numeric_precision_radix	cardinal_number	这一列指示numeric_precision和numeric_scale列中的

名称	数据类型	描述
		值是基于什么来表示。该值为 2 或 10。
numeric_scale	cardinal_number	这一列包含该序列数据类型（见上文）的标度（声明的或隐式的）。标度表示小数点右侧的有效数字位数。它可以按照列numeric_precision_radix中指定的被表示为十进制（基于10）或二进制（基于 2）。
start_value	character_data	该序列的开始值
minimum_value	character_data	该序列的最小值
maximum_value	character_data	该序列的最大值
increment	character_data	该序列的增量
cycle_option	yes_or_no	如果该序列会循环，则为YES，否则为NO

注意依照 SQL 标准，开始值、最小值、最大值和增量值被作为字符串返回。

34.43. sql_features

表sql_features包含的信息指示了哪些 SQL 标准中定义的正式特性被PostgreSQL所支持。这和附录 D中的信息一样。这里你也能找到一些额外的背景信息。

表 34.41. sql_features 列

名称	数据类型	描述
feature_id	character_data	特性的标识符字符串
feature_name	character_data	特性的描述性名称
sub_feature_id	character_data	该子特性的标识符字符串，或者如果不是一个子特性则为一个长度为零的字符串
sub_feature_name	character_data	该子特性的描述性名称，或者如果不是一个子特性则为一个长度为零的字符串
is_supported	yes_or_no	如果当前版本的 PostgreSQL 完全支持该特性，则为YES，否则为NO
is_verified_by	character_data	总是为空，因为PostgreSQL开发组没有对特性的一致性执行正式的测试
comments	character_data	可能会是关于该特性被支持状态的一段注释

34.44. sql_implementation_info

表sql_implementation_info包含的信息指示剩下的由 SQL 标准实现定义的多个方面。这类信息主要用来在 ODBC 接口的情境中使用；其它接口的用户可能将发现这类信息用处不大。由于这个原因，个体实现信息项没有在这里描述，你将会在 ODBC 接口的描述中找到它们。

表 34.42. sql_implementation_info 列

名称	数据类型	描述
implementation_info_id	character_data	实现信息项的标识符字符串

名称	数据类型	描述
implementation_info_name	character_data	实现信息项的描述性名称
integer_value	cardinal_number	实现信息项的整数值；如果该值在列character_value中给出，则本列为空
character_value	character_data	实现信息项的字符值；如果该值在列integer_value中给出，则本列为空
comments	character_data	可能是从属于该实现信息项的一段注释

34.45. sql_languages

表sql_languages为PostgreSQL支持的每一种 SQL 语言绑定包含一行。PostgreSQL支持直接 SQL 和 C 中的嵌入式 SQL；从此表中只能了解到这些信息。

此表已在 SQL:2008 中从 SQL 标准移除，因此没有引用 SQL:2003 之后标准的条目。

表 34.43. sql_languages 列

名称	数据类型	描述
sql_language_source	character_data	语言定义来源的名称；总是ISO 9075，即 SQL 标准
sql_language_year	character_data	sql_language_source引用的标准获批的年份。
sql_language_conformance	character_data	该语言绑定的标准符合性级别。对于 ISO 9075:2003，此值总是CORE。
sql_language_integrity	character_data	总是为空（此值与 SQL 标准的较早版本有关。）
sql_language_implementation	character_data	总是为空
sql_language_binding_style	character_data	语言绑定方式，为DIRECT或EMBEDDED
sql_language_programming_language	character_data	如果绑定方式为EMBEDDED，则为所用编程语言，否则为空。PostgreSQL只支持 C 语言。

34.46. sql_packages

表sql_packages包含PostgreSQL支持 SQL 标准所定义的哪些功能包的信息。功能包的背景信息请参阅附录 D。

表 34.44. sql_packages 列

名称	数据类型	描述
feature_id	character_data	功能包的标识符字符串
feature_name	character_data	功能包的描述性名称
is_supported	yes_or_no	如果当前版本的PostgreSQL完全支持此功能包，则为YES，否则为NO
is_verified_by	character_data	总是为空，因为PostgreSQL开发组没有对特性的一致性执行正式的测试

名称	数据类型	描述
comments	character_data	关于此功能包支持状态的可选注释

34.47. sql_parts

表sql_parts包含的信息指示哪些定义在 SQL 标准中的部分被PostgreSQL支持。

表 34.45. sql_parts 列

名称	数据类型	描述
feature_id	character_data	包含该部分编号的一个标识符字符串
feature_name	character_data	该部分的描述性名称
is_supported	yes_or_no	如果当前版本的 PostgreSQL 完全支持该部分，则为YES，否则为NO
is_verified_by	character_data	总是为空，因为PostgreSQL开发组没有对特性的一致性执行正式的测试
comments	character_data	可能会是关于该部分被支持状态的一段注释

34.48. sql_sizing

表sql_sizing包含有关PostgreSQL中多种尺寸限制和最大值的信息。这类信息主要用来在 ODBC 接口的情境中使用；其它接口的用户可能将发现这类信息用处不大。由于这个原因，个体实现信息项没有在这里描述，你将会在 ODBC 接口的描述中找到它们。

表 34.46. sql_sizing 列

名称	数据类型	描述
sizing_id	cardinal_number	尺寸项的标识符
sizing_name	character_data	尺寸项的描述性名称
supported_value	cardinal_number	尺寸项的值，如果尺寸是不受限制或不能确定的则为 0，如果尺寸项适用的特性不受支持则为空
comments	character_data	可能是从属于尺寸项的一段注释

34.49. sql_sizing_profiles

表sql_sizing_profiles包含 SQL 标准的各类配置规范所要求的sql_sizing值的信息。PostgreSQL不跟踪任何 SQL 配置规范，因此此表为空。

表 34.47. sql_sizing_profiles 列

名称	数据类型	描述
sizing_id	cardinal_number	尺寸项的标识符
sizing_name	character_data	尺寸项的描述性名称
profile_id	character_data	配置规范的标识符字符串
required_value	cardinal_number	SQL 配置规范对该尺寸限制项要求的值；如果配置规范对此项不

名称	数据类型	描述
		设限制，则为 0；如果配置规范不要求此项所适用的任何功能，则为空
comments	character_data	关于此配置规范中该尺寸限制项的可选注释

34.50. table_constraints

视图table_constraints包含属于特定表的所有约束，
这些表要满足的条件是：当前用户拥有表或者是当前用户在表上具有某种除SELECT
之外的权限。

表 34.48. table_constraints 列

名称	数据类型	描述
constraint_catalog	sql_identifier	包含该约束的数据库名称（总是当前数据库）
constraint_schema	sql_identifier	包含该约束的模式名称
constraint_name	sql_identifier	约束的名称
table_catalog	sql_identifier	包含该表的数据库名称（总是当前数据库）
table_schema	sql_identifier	包含该表的模式名称
table_name	sql_identifier	表的名称
constraint_type	character_data	约束类型：CHECK（包括非空约束）、FOREIGN KEY、PRIMARY KEY 或 UNIQUE
is_deferrable	yes_or_no	如果该约束是可延迟的，则为YES，否则为NO
initially_deferred	yes_or_no	如果该约束是可延迟的且初始就被延迟，则为YES，否则为NO

34.51. table_privileges

视图table_privileges标识在表或视图上所有被授予的权限，
这些权限必须是被当前已启用角色授出或者被授予给当前已启用角色。对每一个表、
授予者和被授予者的组合都有一行。

表 34.49. table_privileges 列

名称	数据类型	描述
grantor	sql_identifier	授予该权限的角色名称
grantee	sql_identifier	被授予该权限的角色名称
table_catalog	sql_identifier	包含该表的数据库名称（总是当前数据库）
table_schema	sql_identifier	包含该表的模式名称
table_name	sql_identifier	表的名称
privilege_type	character_data	该权限的类型：SELECT、INSERT、UPDATE、DELETE、TRUNCATE、REFERENCES或TRIGGER

名称	数据类型	描述
is_grantable	yes_or_no	如果该权限是可授予的，则为YES，否则为NO
with_hierarchy	yes_or_no	在 SQL 标准中，WITH HIERARCHY OPTION是一个独立的（子）权限，它允许在表继承层级上的特定操作。在 PostgreSQL 中，这被包括在SELECT权限中，因此这一列在权限为SELECT时显示YES，其他时候显示NO。

34.52. tables

视图tables包含定义在当前数据库中的所有表和视图。
只有那些当前用户能够访问（作为拥有者或具有某些权限）的表和视图才会被显示。

表 34.50. tables 列

名称	数据类型	描述
table_catalog	sql_identifier	包含该表的数据库名称（总是当前数据库）
table_schema	sql_identifier	包含该表的模式名称
table_name	sql_identifier	表的名称
table_type	character_data	该表的类型：BASE TABLE表示一个持久的基本表（常见表类型），VIEW表示一个视图，FOREIGN TABLE表示一个外部表，LOCAL TEMPORARY表示一个临时表
self_referencing_column_name	sql_identifier	对应 PostgreSQL 不支持的特性
reference_generation	character_data	对应 PostgreSQL 不支持的特性
user_defined_type_catalog	sql_identifier	如果该表是一个有类型的表，则是包含其底层数据类型的数据库名称（总是当前数据库），否则为空。
user_defined_type_schema	sql_identifier	如果该表是一个有类型的表，则是包含其底层数据类型的模式名，否则为空。
user_defined_type_name	sql_identifier	如果该表是一个有类型的表，则是其底层数据类型的名称，否则为空。
is_insertable_into	yes_or_no	如果该表能够被插入，则为YES，否则为NO（基本表总是能被插入，而视图则不一定）。
is_typed	yes_or_no	如果该表是一个有类型的表，则为YES，否则为NO
commit_action	character_data	还未被实现

34.53. triggered_update_columns

对于当前数据库中指定一个列列表（如UPDATE OF column1, column2）的触发器，视图triggered_update_columns标识这些列。

没有指定一个列列表的触发器不被包括在这个视图中。
只有那些当前用户拥有或具有某种除SELECT之外权限的列才会被显示。

表 34.51. triggered_update_columns 列

名称	数据类型	描述
trigger_catalog	sql_identifier	包含该触发器的数据库名称（总是当前数据库）
trigger_schema	sql_identifier	包含该触发器的模式名称
trigger_name	sql_identifier	触发器的名称
event_object_catalog	sql_identifier	包含触发器所在的表的数据库名称（总是当前数据库）
event_object_schema	sql_identifier	包含触发器所在的表的模式名称
event_object_table	sql_identifier	触发器所在的表的名称
event_object_column	sql_identifier	触发器所在的列的名称

34.54. triggers

视图triggers包含所有定义在当前数据库中表和视图上的触发器，
并且只显示当前用户拥有的触发器或者是当前用户在其上具有某种除SELECT
之外权限的触发器。

表 34.52. triggers 列

名称	数据类型	描述
trigger_catalog	sql_identifier	包含该触发器的数据库名称（总是当前数据库）
trigger_schema	sql_identifier	包含该触发器的模式名称
trigger_name	sql_identifier	触发器的名称
event_manipulation	character_data	触发该触发器的事件（INSERT、UPDATE或DELETE）
event_object_catalog	sql_identifier	包含触发器所在的表的数据库名称（总是当前数据库）
event_object_schema	sql_identifier	包含触发器所在的表的模式名称
event_object_table	sql_identifier	触发器所在的表的名称
action_order	cardinal_number	尚未实现
action_condition	character_data	触发器的WHEN条件，如果没有则为空（如果该表不被当前已启用角色拥有也是为空）
action_statement	character_data	触发器执行的语句（当前总是EXECUTE PROCEDURE *function*(...)）
action_orientation	character_data	标识触发器是对每个被处理的行触发一次还是为每个语句触发一次（ROW或STATEMENT）
action_timing	character_data	触发器在什么时候触发（BEFORE、AFTER或INSTEAD OF）
action_reference_old_table	sql_identifier	对应 PostgreSQL 不支持的特性
action_reference_new_table	sql_identifier	对应 PostgreSQL 不支持的特性

名称	数据类型	描述
action_reference_old_row	sql_identifier	对应 PostgreSQL 不支持的特性
action_reference_new_row	sql_identifier	对应 PostgreSQL 不支持的特性
created	time_stamp	对应 PostgreSQL 不支持的特性

PostgreSQL中的触发器有两点与 SQL 标准不兼容，这会影响在该信息模式中的表示。第一，在PostgreSQL中触发器的名字是局限于每个表的，而不是独立于模式对象。因此可能在一个模式中会有重复的触发器名称，只要它们属于不同的表（trigger_catalog和trigger_schema才真正标识了触发器被定义在哪个表上）。第二，在PostgreSQL中触发器可以被定义为在多个事件上触发（例如ON INSERT OR UPDATE），而在 SQL 标准中只允许一个。如果一个触发器被定义为在多个事件上触发，它在信息模式中被表示为多行，每一行对应于一类事件。作为这两个问题的结果，视图triggers的主键实际上是(trigger_catalog, trigger_schema, event_object_table, trigger_name, event_manipulation)，而不是(trigger_catalog, trigger_schema, trigger_name)（这是 SQL 标准指定的）。尽管如此，如果你以符合 SQL 标准（在模式中触发器名称唯一并且每个触发器只能有一种事件类型）的方式定义你的触发器，这将不会影响你。

注意

在PostgreSQL 9.1 之前，这个视图的列 action_timing、action_reference_old_table、action_reference_new_table、action_reference_old_row和 action_reference_new_row 分别被命名为 condition_timing、condition_reference_old_table、condition_reference_new_table、condition_reference_old_row和 condition_reference_new_row。那也是它们在 SQL:1999 标准中的命名。新的命名遵循 SQL:2003 及其后的版本。

34.55. udt_privileges

视图udt_privileges标识所有在用户定义类型上授予的USAGE权限，这些权限的授予者或者被授予者是当前已启用的角色。对每一个类型、授予者和被授予者的组合都有一行。这个视图只显示复合类型（原因见下面的第 34.57 节）。域权限见第 34.56 节。

表 34.53. udt_privileges 列

名称	数据类型	描述
grantor	sql_identifier	授予该权限的角色名称
grantee	sql_identifier	被授予该权限的角色名称
udt_catalog	sql_identifier	包含该类型的数据库名称（总是当前数据库）
udt_schema	sql_identifier	包含该类型的模式名称
udt_name	sql_identifier	类型的名称
privilege_type	character_data	总是TYPE USAGE
is_grantable	yes_or_no	如果该权限是可授予的，则为YES，否则为NO

34.56. usage_privileges

视图usage_privileges标识在多种对象上授予的所有USAGE权限，这些权限的授予者或被授予者是当前已启用的角色。在PostgreSQL中，

这目前适用于排序规则、域、外部数据包装器、外部服务器和序列。对于每个对象、授予者和被授予者的组合，这里都有一行。

由于在PostgreSQL中排序规则并没有真正的权限，这个视图对所有排序规则显示由拥有者授予给PUBLIC的隐式非可授予的USAGE权限。但是对其他对象类型则显示真实的权限。

在 PostgreSQL 中，序列也支持除USAGE之外的SELECT和UPDATE权限。这些是非标准的并且因此在该信息模式中不可见。

表 34.54. usage_privileges 列

名称	数据类型	描述
grantor	sql_identifier	授予该权限的角色名称
grantee	sql_identifier	被授予该权限的角色名称
object_catalog	sql_identifier	包含该对象的数据库名称（总是当前数据库）
object_schema	sql_identifier	如果适用，则为包含该对象的模式名称，否则为一个空字符串
object_name	sql_identifier	对象的名称
object_type	character_data	COLLATION或DOMAIN或FOREIGN DATA WRAPPER或FOREIGN SERVER或SEQUENCE
privilege_type	character_data	总是USAGE
is_grantable	yes_or_no	如果该权限是可授予的，则为YES，否则为NO

34.57. user_defined_types

视图user_defined_types目前包含定义在当前数据库中的所有复合类型。只有那些当前用户能够访问（作为拥有者或具有某些权限）的类型才会被显示。

SQL 中有两种用户定义类型：结构类型（在PostgreSQL中也称为复合类型）以及独立类型（distinct type，PostgreSQL 尚未实现）。为兼顾将来，请使用列user_defined_type_category来区分它们。其他用户定义类型，如基础类型和枚举（两者都是PostgreSQL的扩展），不会在这里显示。域的相关信息见第 34.22 节。

表 34.55. user_defined_types 列

名称	数据类型	描述
user_defined_type_catalog	sql_identifier	包含该类型的数据库名称（总是当前数据库）
user_defined_type_schema	sql_identifier	包含该类型的模式名称
user_defined_type_name	sql_identifier	类型的名称
user_defined_type_category	character_data	当前总是STRUCTURED
is_instantiable	yes_or_no	对应 PostgreSQL 不支持的特性
is_final	yes_or_no	对应 PostgreSQL 不支持的特性
ordering_form	character_data	对应 PostgreSQL 不支持的特性
ordering_category	character_data	对应 PostgreSQL 不支持的特性
ordering_routine_catalog	sql_identifier	对应 PostgreSQL 不支持的特性
ordering_routine_schema	sql_identifier	对应 PostgreSQL 不支持的特性

名称	数据类型	描述
ordering_routine_name	sql_identifier	对应 PostgreSQL 不支持的特性
reference_type	character_data	对应 PostgreSQL 不支持的特性
data_type	character_data	对应 PostgreSQL 不支持的特性
character_maximum_length	cardinal_number	对应 PostgreSQL 不支持的特性
character_octet_length	cardinal_number	对应 PostgreSQL 不支持的特性
character_set_catalog	sql_identifier	对应 PostgreSQL 不支持的特性
character_set_schema	sql_identifier	对应 PostgreSQL 不支持的特性
character_set_name	sql_identifier	对应 PostgreSQL 不支持的特性
collation_catalog	sql_identifier	对应 PostgreSQL 不支持的特性
collation_schema	sql_identifier	对应 PostgreSQL 不支持的特性
collation_name	sql_identifier	对应 PostgreSQL 不支持的特性
numeric_precision	cardinal_number	对应 PostgreSQL 不支持的特性
numeric_precision_radix	cardinal_number	对应 PostgreSQL 不支持的特性
numeric_scale	cardinal_number	对应 PostgreSQL 不支持的特性
datetime_precision	cardinal_number	对应 PostgreSQL 不支持的特性
interval_type	character_data	对应 PostgreSQL 不支持的特性
interval_precision	cardinal_number	对应 PostgreSQL 不支持的特性
source_dtd_identifier	sql_identifier	对应 PostgreSQL 不支持的特性
ref_dtd_identifier	sql_identifier	对应 PostgreSQL 不支持的特性

34.58. user_mapping_options

视图user_mapping_options包含在当前数据库中为用户映射定义的所有选项。只有那些当前用户能够访问其相应外部服务器（作为拥有者或具有某些权限）的用户映射才会被显示。

表 34.56. user_mapping_options 列

名称	数据类型	描述
authorization_identifier	sql_identifier	被映射的用户名称，如果映射是公共的则为PUBLIC
foreign_server_catalog	sql_identifier	这个映射所使用的外部服务器所在的数据库名称（总是当前数据库）
foreign_server_name	sql_identifier	这个映射所使用的外部服务器的名称
option_name	sql_identifier	一个选项名称
option_value	character_data	选项的值。除非当前用户是被映射的用户或者映射是PUBLIC的并且当前用户是服务器拥有者或者超级用户，这一列将显示为空。这样做的目的是保护作为用户映射选项存储的密码信息。

34.59. user_mappings

视图user_mappings包含定义在当前数据库中的所有用户映射。只有当前用户能够访问其对应外部服务器（作为拥有者或具有某些权限）的用户映射才会被显示。

表 34.57. user_mappings 列

名称	数据类型	描述
authorization_identifier	sql_identifier	被映射的用户名称，如果映射是公共的则为PUBLIC
foreign_server_catalog	sql_identifier	这个映射所使用的外部服务器所在的数据库名称（总是当前数据库）
foreign_server_name	sql_identifier	这个映射所使用的外部服务器的名称

34.60. view_column_usage

视图view_column_usage标识视图查询表达式（即定义该视图的 SELECT 语句）中使用的所有列。只有当包含该列的表归当前已启用角色所有时，该列才会出现在这个视图中。

> **注意**
>
> 系统表列不被包括。在某个时候这应该会被修复。

表 34.58. view_column_usage 列

名称	数据类型	描述
view_catalog	sql_identifier	包含该视图的数据库名称（总是当前数据库）
view_schema	sql_identifier	包含该视图的模式名称
view_name	sql_identifier	视图的名称
table_catalog	sql_identifier	被视图所使用的列所属表的数据库名称（总是当前数据库）
table_schema	sql_identifier	被视图所使用的列所属表的模式名称
table_name	sql_identifier	被视图所使用的列所属表的名称
column_name	sql_identifier	被该视图所使用的列名称

34.61. view_routine_usage

视图view_routine_usage标识视图查询表达式（即定义该视图的 SELECT 语句）中使用的所有例程（函数和过程）。只有归当前已启用角色所有的例程才会出现在这个视图中。

表 34.59. view_routine_usage 列

名称	数据类型	描述
table_catalog	sql_identifier	包含该视图的数据库名称（总是当前数据库）
table_schema	sql_identifier	包含该视图的模式名称
table_name	sql_identifier	视图的名称
specific_catalog	sql_identifier	包含该函数的数据库名称(总是当前数据库)
specific_schema	sql_identifier	包含函数的模式的名称
specific_name	sql_identifier	函数的“特定名称”。详见第 34.40 节。

34.62. view_table_usage

视图view_table_usage标识视图查询表达式（即定义该视图的 SELECT 语句）中使用的所有表。只有归当前已启用角色所有的表才会出现在这个视图中。

> **注意**
>
> 系统表没有被包括。这应当会在某个时候被修复。

表 34.60. view_table_usage 列

名称	数据类型	描述
view_catalog	sql_identifier	包含该视图的数据库名称（总是当前数据库）
view_schema	sql_identifier	包含该视图的模式名称
view_name	sql_identifier	视图的名称
table_catalog	sql_identifier	包含被该视图所使用的表的数据库名称（总是当前数据库）
table_schema	sql_identifier	包含被该视图所使用的表的模式名称
table_name	sql_identifier	包含被该视图所使用的表的名称

34.63. views

视图views包含定义在当前数据库中的所有视图。
只有当前用户能够访问（作为拥有者或具有某些权限）的视图才会被显示。

表 34.61. views 列

名称	数据类型	描述
table_catalog	sql_identifier	包含该视图的数据库名称（总是当前数据库）
table_schema	sql_identifier	包含该视图的模式名称
table_name	sql_identifier	视图的名称
view_definition	character_data	定义视图的查询表达式（如果该视图不被当前已启用角色拥有则为空）
check_option	character_data	适用于 PostgreSQL 中不可用的特性
is_updatable	yes_or_no	如果视图可更新（允许UPDATE和DELETE），则为YES，否则为NO
is_insertable_into	yes_or_no	如果视图可插入（允许INSERT），则为YES，否则为NO
is_trigger_updatable	yes_or_no	如果该视图上定义了 INSTEAD OF UPDATE 触发器，则为YES，否则为NO
is_trigger_deletable	yes_or_no	如果该视图上定义了 INSTEAD OF DELETE 触发器，则为YES，否则为NO

名称	数据类型	描述
is_trigger_insertable_into	yes_or_no	如果该视图上定义了 INSTEAD OF INSERT 触发器，则为YES，否则为NO

部分 V. 服务器编程

本部分介绍如何使用用户定义函数、数据类型、触发器等扩展服务器功能。这些属于高级主题，
通常应在读者已经理解其他有关PostgreSQL的用户文档之后再来学习。
本部分后面的章节介绍PostgreSQL发行版中提供的服务器端编程语言，
以及与服务器端编程有关的一般问题。在深入学习服务器端编程的内容之前，至少应先阅读第 35 章的前几节（涵盖函数）。

目录

35. 扩展 SQL 763
35.1. 可扩展性如何运作 763
35.2. PostgreSQL 类型系统 763
35.2.1. 基础类型 763
35.2.2. 复合类型 763
35.2.3. 域 763
35.2.4. 伪类型 764
35.2.5. 多态类型 764
35.3. User-defined Functions 764
35.4. Query Language (SQL) Functions 765
35.4.1. Arguments for SQL Functions 766
35.4.2. SQL Functions on Base Types 766
35.4.3. SQL Functions on Composite Types 768
35.4.4. SQL Functions with Output Parameters 770
35.4.5. SQL Functions with Variable Numbers of Arguments 771
35.4.6. SQL Functions with Default Values for Arguments 772
35.4.7. SQL Functions as Table Sources 773
35.4.8. SQL Functions Returning Sets 773
35.4.9. SQL Functions Returning `TABLE` 775
35.4.10. Polymorphic SQL Functions 775
35.4.11. SQL Functions with Collations 777
35.5. Function Overloading 777
35.6. Function Volatility Categories 778
35.7. Procedural Language Functions 780
35.8. Internal Functions 780
35.9. C-Language Functions 780
35.9.1. Dynamic Loading 781
35.9.2. Base Types in C-Language Functions 782
35.9.3. Version 0 Calling Conventions 785
35.9.4. Version 1 Calling Conventions 787
35.9.5. Writing Code 789
35.9.6. 编译和链接动态装载函数 790
35.9.7. Composite-type Arguments 792
35.9.8. Returning Rows (Composite Types) 794
35.9.9. Returning Sets 796
35.9.10. Polymorphic Arguments and Return Types 801
35.9.11. Transform Functions 802
35.9.12. Shared Memory and LWLocks 802
35.9.13. Using C++ for Extensibility 803
35.10. User-defined Aggregates 803
35.11. 用户定义的类型 806
35.12. 用户定义的操作符 809
35.13. 操作符优化信息 809
35.13.1. `COMMUTATOR` 810
35.13.2. `NEGATOR` 810
35.13.3. `RESTRICT` 810
35.13.4. `JOIN` 811
35.13.5. `HASHES` 812
35.13.6. `MERGES` 812
35.14. 索引扩展接口 813
35.14.1. 索引方法和操作符类 813
35.14.2. 索引方法策略 814
35.14.3. 索引方法支持例程 815
35.14.4. 示例 817
35.14.5. 操作符类和操作符族 818

35.14.6. 系统对操作符类的依赖 ... 821
35.14.7. 排序操作符 ... 822
35.14.8. 操作符类的特殊特性 ... 822
35.15. 将相关对象打包成扩展 ... 823
35.15.1. 扩展文件 ... 823
35.15.2. 扩展的可重定位性 ... 825
35.15.3. 扩展配置表 ... 825
35.15.4. 扩展更新 ... 826
35.15.5. 扩展示例 ... 827
35.16. 扩展构建基础设施 ... 828
36. 触发器 ... 831
36.1. 触发器行为概述 ... 831
36.2. 数据更改的可见性 ... 832
36.3. 用 C 编写触发器函数 ... 833
36.4. 一个完整的触发器示例 ... 835
37. 规则系统 ... 839
37.1. 查询树 ... 839
37.2. 视图和规则系统 ... 840
37.2.1. SELECT 规则如何工作 ... 841
37.2.2. 非 SELECT 语句中的视图规则 ... 845
37.2.3. PostgreSQL 中视图的威力 ... 846
37.2.4. 更新视图 ... 846
37.3. INSERT、UPDATE和DELETE规则 ... 847
37.3.1. 更新规则如何工作 ... 847
37.3.2. 与视图的协作 ... 850
37.4. 规则和权限 ... 856
37.5. 规则和命令状态 ... 858
37.6. 规则与触发器 ... 858
38. 过程语言 ... 861
38.1. 安装过程语言 ... 861
39. PL/pgSQL – SQL 过程语言 ... 863
39.1. 概述 ... 863
39.1.1. 使用PL/pgSQL的优点 ... 863
39.1.2. 支持的参数和结果数据类型 ... 863
39.2. PL/pgSQL的结构 ... 864
39.3. 声明 ... 865
39.3.1. 声明函数参数 ... 866
39.3.2. ALIAS ... 868
39.3.3. 复制类型 ... 868
39.3.4. 行类型 ... 868
39.3.5. 记录类型 ... 869
39.3.6. PL/pgSQL变量的排序规则 ... 869
39.4. 表达式 ... 870
39.5. 基本语句 ... 871
39.5.1. 赋值 ... 871
39.5.2. 执行没有结果的命令 ... 871
39.5.3. 执行返回单行结果的查询 ... 872
39.5.4. 执行动态命令 ... 873
39.5.5. 获取结果状态 ... 876
39.5.6. 什么也不做 ... 876
39.6. 控制结构 ... 877
39.6.1. 从函数返回 ... 877
39.6.2. 条件语句 ... 878
39.6.3. 简单循环 ... 881
39.6.4. 遍历查询结果 ... 884
39.6.5. 遍历数组 ... 885
39.6.6. 捕获错误 ... 886
39.7. 游标 ... 888

39.7.1. 声明游标变量 ... 888
39.7.2. 打开游标 ... 889
39.7.3. 使用游标 ... 890
39.7.4. 遍历游标结果 ... 893
39.8. 错误和消息 ... 893
39.9. 触发器函数 ... 894
39.10. PL/pgSQL 内部机制 ... 901
39.10.1. 变量替换 ... 901
39.10.2. 计划缓存 ... 903
39.11. PL/pgSQL开发提示 ... 904
39.11.1. 引号的处理 ... 905
39.12. 从Oracle PL/SQL 移植 ... 906
39.12.1. 移植示例 ... 907
39.12.2. 其他要关注的事项 ... 912
39.12.3. 附录 ... 913
40. PL/Tcl - Tcl 过程语言 ... 916
40.1. 概述 ... 916
40.2. PL/Tcl 函数和参数 ... 916
40.3. PL/Tcl 中的数据值 ... 917
40.4. PL/Tcl 中的全局数据 ... 917
40.5. 从 PL/Tcl 访问数据库 ... 918
40.6. PL/Tcl 中的触发器函数 ... 920
40.7. 模块与unknown命令 ... 921
40.8. Tcl 过程名 ... 922
41. PL/Perl - Perl 过程语言 ... 923
41.1. PL/Perl 函数和参数 ... 923
41.2. PL/Perl 中的数据值 ... 926
41.3. 内置函数 ... 927
41.3.1. 从 PL/Perl 访问数据库 ... 927
41.3.2. PL/Perl 中的辅助函数 ... 930
41.4. PL/Perl 中的全局值 ... 931
41.5. 受信任与不受信任的 PL/Perl ... 932
41.6. PL/Perl 触发器 ... 933
41.7. PL/Perl 内部机制 ... 934
41.7.1. 配置 ... 934
41.7.2. 限制与缺失特性 ... 935
42. PL/Python - Python 过程语言 ... 936
42.1. Python 2 与 Python 3 ... 936
42.2. PL/Python 函数 ... 937
42.3. 数据值 ... 938
42.3.1. 数据类型映射 ... 938
42.3.2. 空值、None ... 939
42.3.3. 数组、列表 ... 939
42.3.4. 复合类型 ... 940
42.3.5. 集合返回函数 ... 941
42.4. 共享数据 ... 942
42.5. 匿名代码块 ... 942
42.6. 触发器函数 ... 943
42.7. 数据库访问 ... 943
42.7.1. 数据库访问函数 ... 944
42.7.2. 捕获错误 ... 946
42.8. 显式子事务 ... 947
42.8.1. 子事务上下文管理器 ... 947
42.8.2. 较早的 Python 版本 ... 948
42.9. 辅助函数 ... 948
42.10. 环境变量 ... 949
43. 服务器编程接口 ... 950
43.1. 接口函数 ... 950

43.2. 接口支持函数 982
43.3. 内存管理 990
43.4. 数据更改的可见性 999
43.5. 示例 999

第 35 章 扩展 SQL

在接下来的各节中，我们将讨论如何通过增加以下内容来扩展 PostgreSQL SQL 查询语言：

- 函数（从第 35.3 节开始）
- 聚合（从第 35.10 节开始）
- 数据类型（从第 35.11 节开始）
- 操作符（从第 35.12 节开始）
- 索引的操作符类（从第 35.14 节开始）
- 相关对象的包（从第 35.15 节开始）

35.1. 可扩展性如何运作

PostgreSQL 之所以具有可扩展性，是因为其运作由系统目录驱动。如果你熟悉标准的关系数据库系统，就会知道它们把有关数据库、表、
列等的信息存储在通常所说的系统目录中（有些系统把这称为数据字典）。
这些目录对用户而言看起来就像普通表一样，但 DBMS 会在其中保存自己的内部管理信息。PostgreSQL 与标准关系数据库系统的一个关键差别是，PostgreSQL 在目录中存储的信息要多得多：不仅有关于表和列的信息，还有关于数据类型、函数、访问方法等的信息。
这些表可以由用户修改，而 PostgreSQL 又是基于这些表来运行的，这意味着 PostgreSQL 可以由用户扩展。相比之下，传统数据库系统只能通过修改源代码中的硬编码过程，或加载由 DBMS 供应商专门编写的模块来扩展。

此外，PostgreSQL 服务器还能通过动态加载把用户编写的代码纳入自身。也就是说，用户可以指定一个实现了新类型或新函数的目标代码文件（例如共享库），而 PostgreSQL 会在需要时加载它。把用 SQL 编写的代码加入服务器就更为简单了。这种能够“即时”修改自身行为的能力，使 PostgreSQL 特别适合用于新应用和新存储结构的快速原型设计。

35.2. PostgreSQL 类型系统

PostgreSQL 数据类型可以分为基础类型、复合类型、域和伪类型。

35.2.1. 基础类型

基础类型是在 SQL 语言层之下实现的类型（通常使用 C 等低级语言），例如 `int4`。它们通常对应于所谓的抽象数据类型。PostgreSQL 只能通过用户提供的函数操作这些类型，对其行为的理解也仅限于用户所描述的内容。基础类型进一步分为标量类型和数组类型。
对于每个标量类型，都会自动创建相应的数组类型，
以存储由该标量类型组成的可变大小数组。

35.2.2. 复合类型

用户每次创建表时，都会创建复合类型（也称行类型）。还可以使用CREATE TYPE定义不关联任何表的“独立”复合类型。复合类型就是一个带有对应字段名的类型列表。
复合类型的值是由字段值组成的一行或一条记录。用户可以在 SQL 查询中访问各个组成字段。关于复合类型的更多信息，参见第 8.15 节。

35.2.3. 域

域基于某一种特定的基础类型，并且在很多场合下都可以与其基础类型互换。不过，
域可以带有约束，把其合法值限制在底层基础类型所允许值的一个子集内。

可以使用 SQL 命令CREATE DOMAIN创建域。本章不讨论域的创建和使用。

35.2.4. 伪类型

有少数几种“伪类型”用于特殊目的。伪类型不能作为表的列或复合类型的属性出现，
但它们可以用来声明函数的参数类型和结果类型。
这为在类型系统内部识别特殊类别的函数提供了一种机制。表 8.24列出了现有的伪类型。

35.2.5. 多态类型

五种特别值得关注的伪类型是 `anyelement`、`anyarray`、`anynonarray`、`anyenum` 和 `anyrange`，它们统称为多态类型。使用这些类型声明的函数称为多态函数。
多态函数可以操作多种不同的数据类型，具体类型由某次调用时实际传入的数据类型决定。

多态参数和结果相互关联，在解析调用多态函数的查询时，
会被解析为某种具体的数据类型。每个声明为 `anyelement` 的位置（参数或返回值）都可以具有任意具体的实际数据类型，但在同一次调用中，它们必须具有相同的实际类型。
每个声明为 `anyarray` 的位置可以具有任意数组数据类型，但同样必须全部具有相同的类型。类似地，声明为 `anyrange` 的位置必须全部具有相同的范围类型。此外，如果某些位置声明为 `anyarray`，另一些位置声明为 `anyelement`，那么 `anyarray` 位置上的实际数组类型，其元素类型必须与 `anyelement` 位置上的类型相同。类似地，如果某些位置声明为 `anyrange`，另一些位置声明为 `anyelement`，那么 `anyrange` 位置上的实际范围类型，其子类型必须与 `anyelement` 位置上的类型相同。`anynonarray` 的处理与 `anyelement` 完全相同，但增加了实际类型不能为数组类型的约束。`anyenum` 的处理与 `anyelement` 完全相同，但增加了实际类型必须为枚举类型的约束。

因此，当多个参数位置被声明为多态类型时，
其总体效果就是只允许某些实际参数类型组合。例如，声明为 `equal(anyelement, anyelement)` 的函数可以接受任意两个输入值，只要它们属于同一数据类型。

如果函数的返回值声明为多态类型，那么至少有一个参数位置也必须是多态的，
并且实际传入的参数数据类型决定该次调用的实际结果类型。例如，
如果尚不存在数组下标机制，可以将实现下标访问的函数定义为 `subscript(anyarray, integer) returns anyelement`。这一声明将实际的第一个参数约束为数组类型，并允许解析器根据第一个参数的实际类型推导正确的结果类型。另一个例子是，声明为 `f(anyarray) returns anyenum` 的函数只接受枚举类型的数组。

注意，`anynonarray` 和 `anyenum` 并不表示独立的类型变量；它们与 `anyelement` 是同一个类型，只是附加了额外约束。例如，把函数声明为 `f(anyelement, anyenum)`，等价于把它声明为 `f(anyenum, anyenum)`：两个实际参数都必须是同一种枚举类型。

可变参数函数（接受可变数量参数的函数，见第 35.4.5 节）可以是多态的：将其最后一个参数声明为 `VARIADIC anyarray` 即可。就参数匹配和确定实际结果类型而言，这样的函数与写出相应数量的 `anynonarray` 参数时行为相同。

35.3. User-defined Functions

PostgreSQL provides four kinds of functions:

- query language functions (functions written in SQL) (第 35.4 节)
- procedural language functions (functions written in, for example, PL/pgSQL or PL/Tcl) (第 35.7 节)
- internal functions (第 35.8 节)
- C-language functions (第 35.9 节)

Every kind of function can take base types, composite types, or combinations of these as arguments (parameters). In addition, every kind of function can return a base type

or a composite type. Functions can also be defined to return sets of base or composite values.

Many kinds of functions can take or return certain pseudo-types (such as polymorphic types), but the available facilities vary. Consult the description of each kind of function for more details.

It's easiest to define SQL functions, so we'll start by discussing those. Most of the concepts presented for SQL functions will carry over to the other types of functions.

Throughout this chapter, it can be useful to look at the reference page of the CREATE FUNCTION command to understand the examples better. Some examples from this chapter can be found in `funcs.sql` and `funcs.c` in the `src/tutorial` directory in the PostgreSQL source distribution.

35.4. Query Language (SQL) Functions

SQL functions execute an arbitrary list of SQL statements, returning the result of the last query in the list. In the simple (non-set) case, the first row of the last query's result will be returned. (Bear in mind that "the first row" of a multirow result is not well-defined unless you use `ORDER BY`.) If the last query happens to return no rows at all, the null value will be returned.

Alternatively, an SQL function can be declared to return a set, by specifying the function's return type as `SETOF` *`sometype`*, or equivalently by declaring it as `RETURNS TABLE(`*`columns`*`)`. In this case all rows of the last query's result are returned. Further details appear below.

The body of an SQL function must be a list of SQL statements separated by semicolons. A semicolon after the last statement is optional. Unless the function is declared to return `void`, the last statement must be a `SELECT`, or an `INSERT`, `UPDATE`, or `DELETE` that has a `RETURNING` clause.

Any collection of commands in the SQL language can be packaged together and defined as a function. Besides `SELECT` queries, the commands can include data modification queries (`INSERT`, `UPDATE`, and `DELETE`), as well as other SQL commands. (You cannot use transaction control commands, e.g. `COMMIT`, `SAVEPOINT`, and some utility commands, e.g. `VACUUM`, in SQL functions.) However, the final command must be a `SELECT` or have a `RETURNING` clause that returns whatever is specified as the function's return type. Alternatively, if you want to define a SQL function that performs actions but has no useful value to return, you can define it as returning `void`. For example, this function removes rows with negative salaries from the `emp` table:

```
CREATE FUNCTION clean_emp() RETURNS void AS '
    DELETE FROM emp
        WHERE salary < 0;
' LANGUAGE SQL;

SELECT clean_emp();

 clean_emp
-----------

(1 row)
```

The syntax of the `CREATE FUNCTION` command requires the function body to be written as a string constant. It is usually most convenient to use dollar quoting (see 第 4.1.2. 4 节) for the string constant. If you choose to use regular single-quoted string constant

syntax, you must double single quote marks (`'`) and backslashes (`\`) (assuming escape string syntax) in the body of the function (see 第 4.1.2.1 节).

35.4.1. Arguments for SQL Functions

Arguments of a SQL function can be referenced in the function body using either names or numbers. Examples of both methods appear below.

To use a name, declare the function argument as having a name, and then just write that name in the function body. If the argument name is the same as any column name in the current SQL command within the function, the column name will take precedence. To override this, qualify the argument name with the name of the function itself, that is *`function_name`*`.`*`argument_name`*. (If this would conflict with a qualified column name, again the column name wins. You can avoid the ambiguity by choosing a different alias for the table within the SQL command.)

In the older numeric approach, arguments are referenced using the syntax `$`*`n`*: `$1` refers to the first input argument, `$2` to the second, and so on. This will work whether or not the particular argument was declared with a name.

If an argument is of a composite type, then the dot notation, e.g., `argname.fieldname` or `$1.fieldname`, can be used to access attributes of the argument. Again, you might need to qualify the argument's name with the function name to make the form with an argument name unambiguous.

SQL function arguments can only be used as data values, not as identifiers. Thus for example this is reasonable:

```
INSERT INTO mytable VALUES ($1);
```

but this will not work:

```
INSERT INTO $1 VALUES (42);
```

> **注意**
>
> The ability to use names to reference SQL function arguments was added in PostgreSQL 9.2. Functions to be used in older servers must use the `$`*`n`* notation.

35.4.2. SQL Functions on Base Types

The simplest possible SQL function has no arguments and simply returns a base type, such as `integer`:

```
CREATE FUNCTION one() RETURNS integer AS $$
    SELECT 1 AS result;
$$ LANGUAGE SQL;

-- Alternative syntax for string literal:
CREATE FUNCTION one() RETURNS integer AS '
    SELECT 1 AS result;
' LANGUAGE SQL;

SELECT one();

 one
-----
   1
```

Notice that we defined a column alias within the function body for the result of the function (with the name `result`), but this column alias is not visible outside the function. Hence, the result is labeled `one` instead of `result`.

It is almost as easy to define SQL functions that take base types as arguments:

```
CREATE FUNCTION add_em(x integer, y integer) RETURNS integer AS $$
    SELECT x + y;
$$ LANGUAGE SQL;

SELECT add_em(1, 2) AS answer;

 answer
--------
      3
```

Alternatively, we could dispense with names for the arguments and use numbers:

```
CREATE FUNCTION add_em(integer, integer) RETURNS integer AS $$
    SELECT $1 + $2;
$$ LANGUAGE SQL;

SELECT add_em(1, 2) AS answer;

 answer
--------
      3
```

Here is a more useful function, which might be used to debit a bank account:

```
CREATE FUNCTION tf1 (accountno integer, debit numeric) RETURNS
 integer AS $$
    UPDATE bank
        SET balance = balance - debit
        WHERE accountno = tf1.accountno;
    SELECT 1;
$$ LANGUAGE SQL;
```

A user could execute this function to debit account 17 by $100.00 as follows:

```
SELECT tf1(17, 100.0);
```

In this example, we chose the name `accountno` for the first argument, but this is the same as the name of a column in the `bank` table. Within the `UPDATE` command, `accountno` refers to the column `bank.accountno`, so `tf1.accountno` must be used to refer to the argument. We could of course avoid this by using a different name for the argument.

In practice one would probably like a more useful result from the function than a constant 1, so a more likely definition is:

```
CREATE FUNCTION tf1 (accountno integer, debit numeric) RETURNS
 integer AS $$
    UPDATE bank
        SET balance = balance - debit
        WHERE accountno = tf1.accountno;
    SELECT balance FROM bank WHERE accountno = tf1.accountno;
$$ LANGUAGE SQL;
```

which adjusts the balance and returns the new balance. The same thing could be done in one command using `RETURNING`:

```
CREATE FUNCTION tf1 (accountno integer, debit numeric) RETURNS
 integer AS $$
    UPDATE bank
        SET balance = balance - debit
        WHERE accountno = tf1.accountno
    RETURNING balance;
$$ LANGUAGE SQL;
```

35.4.3. SQL Functions on Composite Types

When writing functions with arguments of composite types, we must not only specify which argument we want but also the desired attribute (field) of that argument. For example, suppose that `emp` is a table containing employee data, and therefore also the name of the composite type of each row of the table. Here is a function `double_salary` that computes what someone's salary would be if it were doubled:

```
CREATE TABLE emp (
    name        text,
    salary      numeric,
    age         integer,
    cubicle     point
);

INSERT INTO emp VALUES ('Bill', 4200, 45, '(2,1)');

CREATE FUNCTION double_salary(emp) RETURNS numeric AS $$
    SELECT $1.salary * 2 AS salary;
$$ LANGUAGE SQL;

SELECT name, double_salary(emp.*) AS dream
    FROM emp
    WHERE emp.cubicle ~= point '(2,1)';

 name | dream
------+-------
 Bill |  8400
```

Notice the use of the syntax `$1.salary` to select one field of the argument row value. Also notice how the calling `SELECT` command uses *`table_name`*`.*` to select the entire current row of a table as a composite value. The table row can alternatively be referenced using just the table name, like this:

```
SELECT name, double_salary(emp) AS dream
    FROM emp
    WHERE emp.cubicle ~= point '(2,1)';
```

but this usage is deprecated since it's easy to get confused. (See 第 8.15.5 节 for details about these two notations for the composite value of a table row.)

Sometimes it is handy to construct a composite argument value on-the-fly. This can be done with the `ROW` construct. For example, we could adjust the data being passed to the function:

```
SELECT name, double_salary(ROW(name, salary*1.1, age, cubicle)) AS
 dream
    FROM emp;
```

It is also possible to build a function that returns a composite type. This is an example of a function that returns a single `emp` row:

```
CREATE FUNCTION new_emp() RETURNS emp AS $$
    SELECT text 'None' AS name,
        1000.0 AS salary,
        25 AS age,
        point '(2,2)' AS cubicle;
$$ LANGUAGE SQL;
```

In this example we have specified each of the attributes with a constant value, but any computation could have been substituted for these constants.

Note two important things about defining the function:

- The select list order in the query must be exactly the same as that in which the columns appear in the table associated with the composite type. (Naming the columns, as we did above, is irrelevant to the system.)
- You must typecast the expressions to match the definition of the composite type, or you will get errors like this:

  ```
  ERROR:  function declared to return emp returns varchar instead
   of text at column 1
  ```

A different way to define the same function is:

```
CREATE FUNCTION new_emp() RETURNS emp AS $$
    SELECT ROW('None', 1000.0, 25, '(2,2)')::emp;
$$ LANGUAGE SQL;
```

Here we wrote a `SELECT` that returns just a single column of the correct composite type. This isn't really better in this situation, but it is a handy alternative in some cases – for example, if we need to compute the result by calling another function that returns the desired composite value.

We could call this function directly either by using it in a value expression:

```
SELECT new_emp();

         new_emp
--------------------------
 (None,1000.0,25,"(2,2)")
```

or by calling it as a table function:

```
SELECT * FROM new_emp();

 name | salary | age | cubicle
------+--------+-----+---------
 None | 1000.0 |  25 | (2,2)
```

The second way is described more fully in 第 35.4.7 节.

When you use a function that returns a composite type, you might want only one field (attribute) from its result. You can do that with syntax like this:

```
SELECT (new_emp()).name;

 name
------
 None
```

The extra parentheses are needed to keep the parser from getting confused. If you try to do it without them, you get something like this:

```
SELECT new_emp().name;
ERROR:  syntax error at or near "."
LINE 1: SELECT new_emp().name;
                        ^
```

Another option is to use functional notation for extracting an attribute:

```
SELECT name(new_emp());

 name
------
 None
```

As explained in 第 8.15.5 节, the field notation and functional notation are equivalent.

Another way to use a function returning a composite type is to pass the result to another function that accepts the correct row type as input:

```
CREATE FUNCTION getname(emp) RETURNS text AS $$
    SELECT $1.name;
$$ LANGUAGE SQL;

SELECT getname(new_emp());
 getname
---------
 None
(1 row)
```

35.4.4. SQL Functions with Output Parameters

An alternative way of describing a function's results is to define it with output parameters, as in this example:

```
CREATE FUNCTION add_em (IN x int, IN y int, OUT sum int)
AS 'SELECT x + y'
LANGUAGE SQL;

SELECT add_em(3,7);
 add_em
--------
     10
(1 row)
```

This is not essentially different from the version of `add_em` shown in 第 35.4.2 节. The real value of output parameters is that they provide a convenient way of defining functions that return several columns. For example,

```
CREATE FUNCTION sum_n_product (x int, y int, OUT sum int, OUT
 product int)
AS 'SELECT x + y, x * y'
LANGUAGE SQL;

 SELECT * FROM sum_n_product(11,42);
 sum | product
-----+---------
  53 |     462
(1 row)
```

What has essentially happened here is that we have created an anonymous composite type for the result of the function. The above example has the same end result as

```
CREATE TYPE sum_prod AS (sum int, product int);

CREATE FUNCTION sum_n_product (int, int) RETURNS sum_prod
AS 'SELECT $1 + $2, $1 * $2'
LANGUAGE SQL;
```

but not having to bother with the separate composite type definition is often handy. Notice that the names attached to the output parameters are not just decoration, but determine the column names of the anonymous composite type. (If you omit a name for an output parameter, the system will choose a name on its own.)

Notice that output parameters are not included in the calling argument list when invoking such a function from SQL. This is because PostgreSQL considers only the input parameters to define the function's calling signature. That means also that only the input parameters matter when referencing the function for purposes such as dropping it. We could drop the above function with either of

```
DROP FUNCTION sum_n_product (x int, y int, OUT sum int, OUT product
 int);
DROP FUNCTION sum_n_product (int, int);
```

Parameters can be marked as `IN` (the default), `OUT`, `INOUT`, or `VARIADIC`. An `INOUT` parameter serves as both an input parameter (part of the calling argument list) and an output parameter (part of the result record type). `VARIADIC` parameters are input parameters, but are treated specially as described next.

35.4.5. SQL Functions with Variable Numbers of Arguments

SQL functions can be declared to accept variable numbers of arguments, so long as all the "optional" arguments are of the same data type. The optional arguments will be passed to the function as an array. The function is declared by marking the last parameter as `VARIADIC`; this parameter must be declared as being of an array type. For example:

```
CREATE FUNCTION mleast(VARIADIC arr numeric[]) RETURNS numeric AS $
$
    SELECT min($1[i]) FROM generate_subscripts($1, 1) g(i);
$$ LANGUAGE SQL;

SELECT mleast(10, -1, 5, 4.4);
 mleast
--------
     -1
(1 row)
```

Effectively, all the actual arguments at or beyond the `VARIADIC` position are gathered up into a one-dimensional array, as if you had written

```
SELECT mleast(ARRAY[10, -1, 5, 4.4]);    -- doesn't work
```

You can't actually write that, though – or at least, it will not match this function definition. A parameter marked `VARIADIC` matches one or more occurrences of its element type, not of its own type.

Sometimes it is useful to be able to pass an already-constructed array to a variadic function; this is particularly handy when one variadic function wants to pass on its array parameter to another one. You can do that by specifying `VARIADIC` in the call:

```
SELECT mleast(VARIADIC ARRAY[10, -1, 5, 4.4]);
```

This prevents expansion of the function's variadic parameter into its element type, thereby allowing the array argument value to match normally. `VARIADIC` can only be attached to the last actual argument of a function call.

Specifying `VARIADIC` in the call is also the only way to pass an empty array to a variadic function, for example:

```
SELECT mleast(VARIADIC ARRAY[]::numeric[]);
```

Simply writing `SELECT mleast()` does not work because a variadic parameter must match at least one actual argument. (You could define a second function also named `mleast`, with no parameters, if you wanted to allow such calls.)

The array element parameters generated from a variadic parameter are treated as not having any names of their own. This means it is not possible to call a variadic function using named arguments (第 4.3 节), except when you specify `VARIADIC`. For example, this will work:

```
SELECT mleast(VARIADIC arr := ARRAY[10, -1, 5, 4.4]);
```

but not these:

```
SELECT mleast(arr := 10);
SELECT mleast(arr := ARRAY[10, -1, 5, 4.4]);
```

35.4.6. SQL Functions with Default Values for Arguments

Functions can be declared with default values for some or all input arguments. The default values are inserted whenever the function is called with insufficiently many actual arguments. Since arguments can only be omitted from the end of the actual argument list, all parameters after a parameter with a default value have to have default values as well. (Although the use of named argument notation could allow this restriction to be relaxed, it's still enforced so that positional argument notation works sensibly.)

For example:

```
CREATE FUNCTION foo(a int, b int DEFAULT 2, c int DEFAULT 3)
RETURNS int
LANGUAGE SQL
AS $$
    SELECT $1 + $2 + $3;
$$;

SELECT foo(10, 20, 30);
 foo
-----
  60
(1 row)

SELECT foo(10, 20);
 foo
-----
  33
(1 row)

SELECT foo(10);
 foo
```

```
-----
  15
(1 row)

SELECT foo();  -- fails since there is no default for the first
 argument
ERROR:  function foo() does not exist
```

The = sign can also be used in place of the key word `DEFAULT`.

35.4.7. SQL Functions as Table Sources

All SQL functions can be used in the `FROM` clause of a query, but it is particularly useful for functions returning composite types. If the function is defined to return a base type, the table function produces a one-column table. If the function is defined to return a composite type, the table function produces a column for each attribute of the composite type.

Here is an example:

```
CREATE TABLE foo (fooid int, foosubid int, fooname text);
INSERT INTO foo VALUES (1, 1, 'Joe');
INSERT INTO foo VALUES (1, 2, 'Ed');
INSERT INTO foo VALUES (2, 1, 'Mary');

CREATE FUNCTION getfoo(int) RETURNS foo AS $$
    SELECT * FROM foo WHERE fooid = $1;
$$ LANGUAGE SQL;

SELECT *, upper(fooname) FROM getfoo(1) AS t1;

 fooid | foosubid | fooname | upper
-------+----------+---------+-------
     1 |        1 | Joe     | JOE
(1 row)
```

As the example shows, we can work with the columns of the function's result just the same as if they were columns of a regular table.

Note that we only got one row out of the function. This is because we did not use `SETOF` . That is described in the next section.

35.4.8. SQL Functions Returning Sets

When an SQL function is declared as returning `SETOF` *`sometype`*, the function's final query is executed to completion, and each row it outputs is returned as an element of the result set.

This feature is normally used when calling the function in the `FROM` clause. In this case each row returned by the function becomes a row of the table seen by the query. For example, assume that table `foo` has the same contents as above, and we say:

```
CREATE FUNCTION getfoo(int) RETURNS SETOF foo AS $$
    SELECT * FROM foo WHERE fooid = $1;
$$ LANGUAGE SQL;

SELECT * FROM getfoo(1) AS t1;
```

Then we would get:

```
 fooid | foosubid | fooname
-------+----------+---------
     1 |        1 | Joe
     1 |        2 | Ed
(2 rows)
```

It is also possible to return multiple rows with the columns defined by output parameters, like this:

```
CREATE TABLE tab (y int, z int);
INSERT INTO tab VALUES (1, 2), (3, 4), (5, 6), (7, 8);

CREATE FUNCTION sum_n_product_with_tab (x int, OUT sum int, OUT
 product int)
RETURNS SETOF record
AS $$
    SELECT $1 + tab.y, $1 * tab.y FROM tab;
$$ LANGUAGE SQL;

SELECT * FROM sum_n_product_with_tab(10);
 sum | product
-----+---------
  11 |      10
  13 |      30
  15 |      50
  17 |      70
(4 rows)
```

The key point here is that you must write `RETURNS SETOF record` to indicate that the function returns multiple rows instead of just one. If there is only one output parameter, write that parameter's type instead of `record`.

Currently, functions returning sets can also be called in the select list of a query. For each row that the query generates by itself, the function returning set is invoked, and an output row is generated for each element of the function's result set. Note, however, that this capability is deprecated and might be removed in future releases. The following is an example function returning a set from the select list:

```
CREATE FUNCTION listchildren(text) RETURNS SETOF text AS $$
    SELECT name FROM nodes WHERE parent = $1
$$ LANGUAGE SQL;

SELECT * FROM nodes;
   name    | parent
-----------+--------
 Top       |
 Child1    | Top
 Child2    | Top
 Child3    | Top
 SubChild1 | Child1
 SubChild2 | Child1
(6 rows)

SELECT listchildren('Top');
 listchildren
--------------
 Child1
 Child2
 Child3
```

```
(3 rows)

SELECT name, listchildren(name) FROM nodes;
  name   | listchildren
--------+--------------
 Top    | Child1
 Top    | Child2
 Top    | Child3
 Child1 | SubChild1
 Child1 | SubChild2
(5 rows)
```

In the last `SELECT`, notice that no output row appears for `Child2`, `Child3`, etc. This happens because `listchildren` returns an empty set for those arguments, so no result rows are generated.

> **注意**
>
> If a function's last command is `INSERT`, `UPDATE`, or `DELETE` with `RETURNING`, that command will always be executed to completion, even if the function is not declared with `SETOF` or the calling query does not fetch all the result rows. Any extra rows produced by the `RETURNING` clause are silently dropped, but the commanded table modifications still happen (and are all completed before returning from the function).

35.4.9. SQL Functions Returning `TABLE`

There is another way to declare a function as returning a set, which is to use the syntax `RETURNS TABLE(columns)`. This is equivalent to using one or more `OUT` parameters plus marking the function as returning `SETOF record` (or `SETOF` a single output parameter's type, as appropriate). This notation is specified in recent versions of the SQL standard, and thus may be more portable than using `SETOF`.

For example, the preceding sum-and-product example could also be done this way:

```
CREATE FUNCTION sum_n_product_with_tab (x int)
RETURNS TABLE(sum int, product int) AS $$
    SELECT $1 + tab.y, $1 * tab.y FROM tab;
$$ LANGUAGE SQL;
```

It is not allowed to use explicit `OUT` or `INOUT` parameters with the `RETURNS TABLE` notation – you must put all the output columns in the `TABLE` list.

35.4.10. Polymorphic SQL Functions

SQL functions can be declared to accept and return the polymorphic types `anyelement`, `anyarray`, `anynonarray`, `anyenum`, and `anyrange`. See 第 35.2.5 节 for a more detailed explanation of polymorphic functions. Here is a polymorphic function `make_array` that builds up an array from two arbitrary data type elements:

```
CREATE FUNCTION make_array(anyelement, anyelement) RETURNS anyarray
 AS $$
    SELECT ARRAY[$1, $2];
$$ LANGUAGE SQL;

SELECT make_array(1, 2) AS intarray, make_array('a'::text, 'b') AS
 textarray;
```

```
 intarray | textarray
----------+-----------
 {1,2}    | {a,b}
(1 row)
```

Notice the use of the typecast `'a'::text` to specify that the argument is of type `text`. This is required if the argument is just a string literal, since otherwise it would be treated as type `unknown`, and array of `unknown` is not a valid type. Without the typecast, you will get errors like this:

```
ERROR:  could not determine polymorphic type because input has type
 "unknown"
```

It is permitted to have polymorphic arguments with a fixed return type, but the converse is not. For example:

```
CREATE FUNCTION is_greater(anyelement, anyelement) RETURNS boolean
 AS $$
    SELECT $1 > $2;
$$ LANGUAGE SQL;

SELECT is_greater(1, 2);
 is_greater
------------
 f
(1 row)

CREATE FUNCTION invalid_func() RETURNS anyelement AS $$
    SELECT 1;
$$ LANGUAGE SQL;
ERROR:  cannot determine result data type
DETAIL:  A function returning a polymorphic type must have at least
 one polymorphic argument.
```

Polymorphism can be used with functions that have output arguments. For example:

```
CREATE FUNCTION dup (f1 anyelement, OUT f2 anyelement, OUT f3
 anyarray)
AS 'select $1, array[$1,$1]' LANGUAGE SQL;

SELECT * FROM dup(22);
 f2 |   f3
----+---------
 22 | {22,22}
(1 row)
```

Polymorphism can also be used with variadic functions. For example:

```
CREATE FUNCTION anyleast (VARIADIC anyarray) RETURNS anyelement AS
 $$
    SELECT min($1[i]) FROM generate_subscripts($1, 1) g(i);
$$ LANGUAGE SQL;

SELECT anyleast(10, -1, 5, 4);
 anyleast
----------
       -1
(1 row)

SELECT anyleast('abc'::text, 'def');
```

```
 anyleast
----------
 abc
(1 row)

CREATE FUNCTION concat_values(text, VARIADIC anyarray) RETURNS text
 AS $$
    SELECT array_to_string($2, $1);
$$ LANGUAGE SQL;

SELECT concat_values('|', 1, 4, 2);
 concat_values
---------------
 1|4|2
(1 row)
```

35.4.11. SQL Functions with Collations

When a SQL function has one or more parameters of collatable data types, a collation is identified for each function call depending on the collations assigned to the actual arguments, as described in 第 22.2 节. If a collation is successfully identified (i.e., there are no conflicts of implicit collations among the arguments) then all the collatable parameters are treated as having that collation implicitly. This will affect the behavior of collation-sensitive operations within the function. For example, using the `anyleast` function described above, the result of

```
SELECT anyleast('abc'::text, 'ABC');
```

will depend on the database's default collation. In `C` locale the result will be `ABC`, but in many other locales it will be `abc`. The collation to use can be forced by adding a `COLLATE` clause to any of the arguments, for example

```
SELECT anyleast('abc'::text, 'ABC' COLLATE "C");
```

Alternatively, if you wish a function to operate with a particular collation regardless of what it is called with, insert `COLLATE` clauses as needed in the function definition. This version of `anyleast` would always use `en_US` locale to compare strings:

```
CREATE FUNCTION anyleast (VARIADIC anyarray) RETURNS anyelement AS
 $$
    SELECT min($1[i] COLLATE "en_US") FROM generate_subscripts($1,
 1) g(i);
$$ LANGUAGE SQL;
```

But note that this will throw an error if applied to a non-collatable data type.

If no common collation can be identified among the actual arguments, then a SQL function treats its parameters as having their data types' default collation (which is usually the database's default collation, but could be different for parameters of domain types).

The behavior of collatable parameters can be thought of as a limited form of polymorphism, applicable only to textual data types.

35.5. Function Overloading

More than one function can be defined with the same SQL name, so long as the arguments they take are different. In other words, function names can be overloaded. When a query is executed, the server will determine which function to call from the

data types and the number of the provided arguments. Overloading can also be used to simulate functions with a variable number of arguments, up to a finite maximum number.

When creating a family of overloaded functions, one should be careful not to create ambiguities. For instance, given the functions:

```
CREATE FUNCTION test(int, real) RETURNS ...
CREATE FUNCTION test(smallint, double precision) RETURNS ...
```

it is not immediately clear which function would be called with some trivial input like `test(1, 1.5)`. The currently implemented resolution rules are described in 第 10 章, but it is unwise to design a system that subtly relies on this behavior.

A function that takes a single argument of a composite type should generally not have the same name as any attribute (field) of that type. Recall that `attribute(table)` is considered equivalent to `table.attribute`. In the case that there is an ambiguity between a function on a composite type and an attribute of the composite type, the attribute will always be used. It is possible to override that choice by schema-qualifying the function name (that is, `schema.func(table)`) but it's better to avoid the problem by not choosing conflicting names.

Another possible conflict is between variadic and non-variadic functions. For instance, it is possible to create both `foo(numeric)` and `foo(VARIADIC numeric[])`. In this case it is unclear which one should be matched to a call providing a single numeric argument, such as `foo(10.1)`. The rule is that the function appearing earlier in the search path is used, or if the two functions are in the same schema, the non-variadic one is preferred.

When overloading C-language functions, there is an additional constraint: The C name of each function in the family of overloaded functions must be different from the C names of all other functions, either internal or dynamically loaded. If this rule is violated, the behavior is not portable. You might get a run-time linker error, or one of the functions will get called (usually the internal one). The alternative form of the `AS` clause for the SQL `CREATE FUNCTION` command decouples the SQL function name from the function name in the C source code. For instance:

```
CREATE FUNCTION test(int) RETURNS int
    AS 'filename', 'test_1arg'
    LANGUAGE C;
CREATE FUNCTION test(int, int) RETURNS int
    AS 'filename', 'test_2arg'
    LANGUAGE C;
```

The names of the C functions here reflect one of many possible conventions.

35.6. Function Volatility Categories

Every function has a volatility classification, with the possibilities being `VOLATILE`, `STABLE`, or `IMMUTABLE`. `VOLATILE` is the default if the CREATE FUNCTION command does not specify a category. The volatility category is a promise to the optimizer about the behavior of the function:

- A `VOLATILE` function can do anything, including modifying the database. It can return different results on successive calls with the same arguments. The optimizer makes no assumptions about the behavior of such functions. A query using a volatile function will re-evaluate the function at every row where its value is needed.
- A `STABLE` function cannot modify the database and is guaranteed to return the same results given the same arguments for all rows within a single statement. This

category allows the optimizer to optimize multiple calls of the function to a single call. In particular, it is safe to use an expression containing such a function in an index scan condition. (Since an index scan will evaluate the comparison value only once, not once at each row, it is not valid to use a `VOLATILE` function in an index scan condition.)

- An `IMMUTABLE` function cannot modify the database and is guaranteed to return the same results given the same arguments forever. This category allows the optimizer to pre-evaluate the function when a query calls it with constant arguments. For example, a query like `SELECT ... WHERE x = 2 + 2` can be simplified on sight to `SELECT ... WHERE x = 4`, because the function underlying the integer addition operator is marked `IMMUTABLE`.

For best optimization results, you should label your functions with the strictest volatility category that is valid for them.

Any function with side-effects must be labeled `VOLATILE`, so that calls to it cannot be optimized away. Even a function with no side-effects needs to be labeled `VOLATILE` if its value can change within a single query; some examples are `random()`, `currval()`, `timeofday()`.

Another important example is that the `current_timestamp` family of functions qualify as `STABLE`, since their values do not change within a transaction.

There is relatively little difference between `STABLE` and `IMMUTABLE` categories when considering simple interactive queries that are planned and immediately executed: it doesn't matter a lot whether a function is executed once during planning or once during query execution startup. But there is a big difference if the plan is saved and reused later. Labeling a function `IMMUTABLE` when it really isn't might allow it to be prematurely folded to a constant during planning, resulting in a stale value being re-used during subsequent uses of the plan. This is a hazard when using prepared statements or when using function languages that cache plans (such as PL/pgSQL).

For functions written in SQL or in any of the standard procedural languages, there is a second important property determined by the volatility category, namely the visibility of any data changes that have been made by the SQL command that is calling the function. A `VOLATILE` function will see such changes, a `STABLE` or `IMMUTABLE` function will not. This behavior is implemented using the snapshotting behavior of MVCC (see 第 13 章): `STABLE` and `IMMUTABLE` functions use a snapshot established as of the start of the calling query, whereas `VOLATILE` functions obtain a fresh snapshot at the start of each query they execute.

> **注意**
>
> Functions written in C can manage snapshots however they want, but it's usually a good idea to make C functions work this way too.

Because of this snapshotting behavior, a function containing only `SELECT` commands can safely be marked `STABLE`, even if it selects from tables that might be undergoing modifications by concurrent queries. PostgreSQL will execute all commands of a `STABLE` function using the snapshot established for the calling query, and so it will see a fixed view of the database throughout that query.

The same snapshotting behavior is used for `SELECT` commands within `IMMUTABLE` functions. It is generally unwise to select from database tables within an `IMMUTABLE` function at all, since the immutability will be broken if the table contents ever change. However, PostgreSQL does not enforce that you do not do that.

A common error is to label a function `IMMUTABLE` when its results depend on a configuration parameter. For example, a function that manipulates timestamps might well have results that depend on the TimeZone setting. For safety, such functions should be labeled `STABLE` instead.

> **注意**
>
> Before PostgreSQL release 8.0, the requirement that `STABLE` and `IMMUTABLE` functions cannot modify the database was not enforced by the system. Releases 8.0 and later enforce it by requiring SQL functions and procedural language functions of these categories to contain no SQL commands other than `SELECT`. (This is not a completely bulletproof test, since such functions could still call `VOLATILE` functions that modify the database. If you do that, you will find that the `STABLE` or `IMMUTABLE` function does not notice the database changes applied by the called function, since they are hidden from its snapshot.)

35.7. Procedural Language Functions

PostgreSQL allows user-defined functions to be written in other languages besides SQL and C. These other languages are generically called procedural languages (PLs). Procedural languages aren't built into the PostgreSQL server; they are offered by loadable modules. See 第 38 章 and following chapters for more information.

35.8. Internal Functions

Internal functions are functions written in C that have been statically linked into the PostgreSQL server. The “body” of the function definition specifies the C-language name of the function, which need not be the same as the name being declared for SQL use. (For reasons of backward compatibility, an empty body is accepted as meaning that the C-language function name is the same as the SQL name.)

Normally, all internal functions present in the server are declared during the initialization of the database cluster (see 第 17.2 节), but a user could use `CREATE FUNCTION` to create additional alias names for an internal function. Internal functions are declared in `CREATE FUNCTION` with language name `internal`. For instance, to create an alias for the `sqrt` function:

```
CREATE FUNCTION square_root(double precision) RETURNS double
 precision
    AS 'dsqrt'
    LANGUAGE internal
    STRICT;
```

(Most internal functions expect to be declared “strict” .)

> **注意**
>
> Not all “predefined” functions are “internal” in the above sense. Some predefined functions are written in SQL.

35.9. C-Language Functions

User-defined functions can be written in C (or a language that can be made compatible with C, such as C++). Such functions are compiled into dynamically

loadable objects (also called shared libraries) and are loaded by the server on demand. The dynamic loading feature is what distinguishes "C language" functions from "internal" functions – the actual coding conventions are essentially the same for both. (Hence, the standard internal function library is a rich source of coding examples for user-defined C functions.)

Two different calling conventions are currently used for C functions. The newer "version 1" calling convention is indicated by writing a `PG_FUNCTION_INFO_V1()` macro call for the function, as illustrated below. Lack of such a macro indicates an old-style ("version 0") function. The language name specified in `CREATE FUNCTION` is `C` in either case. Old-style functions are now deprecated because of portability problems and lack of functionality, but they are still supported for compatibility reasons.

35.9.1. Dynamic Loading

The first time a user-defined function in a particular loadable object file is called in a session, the dynamic loader loads that object file into memory so that the function can be called. The `CREATE FUNCTION` for a user-defined C function must therefore specify two pieces of information for the function: the name of the loadable object file, and the C name (link symbol) of the specific function to call within that object file. If the C name is not explicitly specified then it is assumed to be the same as the SQL function name.

The following algorithm is used to locate the shared object file based on the name given in the `CREATE FUNCTION` command:

1. If the name is an absolute path, the given file is loaded.
2. If the name starts with the string `$libdir`, that part is replaced by the PostgreSQL package library directory name, which is determined at build time.
3. If the name does not contain a directory part, the file is searched for in the path specified by the configuration variable dynamic_library_path.
4. Otherwise (the file was not found in the path, or it contains a non-absolute directory part), the dynamic loader will try to take the name as given, which will most likely fail. (It is unreliable to depend on the current working directory.)

If this sequence does not work, the platform-specific shared library file name extension (often `.so`) is appended to the given name and this sequence is tried again. If that fails as well, the load will fail.

It is recommended to locate shared libraries either relative to `$libdir` or through the dynamic library path. This simplifies version upgrades if the new installation is at a different location. The actual directory that `$libdir` stands for can be found out with the command `pg_config --pkglibdir`.

The user ID the PostgreSQL server runs as must be able to traverse the path to the file you intend to load. Making the file or a higher-level directory not readable and/or not executable by the postgres user is a common mistake.

In any case, the file name that is given in the `CREATE FUNCTION` command is recorded literally in the system catalogs, so if the file needs to be loaded again the same procedure is applied.

> **注意**
>
> PostgreSQL will not compile a C function automatically. The object file must be compiled before it is referenced in a `CREATE FUNCTION` command. See 第 35.9.6 节 for additional information.

To ensure that a dynamically loaded object file is not loaded into an incompatible server, PostgreSQL checks that the file contains a "magic block" with the appropriate contents. This allows the server to detect obvious incompatibilities, such as code compiled for a different major version of PostgreSQL. A magic block is required as of PostgreSQL 8.2. To include a magic block, write this in one (and only one) of the module source files, after having included the header `fmgr.h`:

```
#ifdef PG_MODULE_MAGIC
PG_MODULE_MAGIC;
#endif
```

The `#ifdef` test can be omitted if the code doesn't need to compile against pre-8.2 PostgreSQL releases.

After it is used for the first time, a dynamically loaded object file is retained in memory. Future calls in the same session to the function(s) in that file will only incur the small overhead of a symbol table lookup. If you need to force a reload of an object file, for example after recompiling it, begin a fresh session.

Optionally, a dynamically loaded file can contain initialization and finalization functions. If the file includes a function named `_PG_init`, that function will be called immediately after loading the file. The function receives no parameters and should return void. If the file includes a function named `_PG_fini`, that function will be called immediately before unloading the file. Likewise, the function receives no parameters and should return void. Note that `_PG_fini` will only be called during an unload of the file, not during process termination. (Presently, unloads are disabled and will never occur, but this may change in the future.)

35.9.2. Base Types in C-Language Functions

To know how to write C-language functions, you need to know how PostgreSQL internally represents base data types and how they can be passed to and from functions. Internally, PostgreSQL regards a base type as a "blob of memory". The user-defined functions that you define over a type in turn define the way that PostgreSQL can operate on it. That is, PostgreSQL will only store and retrieve the data from disk and use your user-defined functions to input, process, and output the data.

Base types can have one of three internal formats:

- pass by value, fixed-length
- pass by reference, fixed-length
- pass by reference, variable-length

By-value types can only be 1, 2, or 4 bytes in length (also 8 bytes, if `sizeof(Datum)` is 8 on your machine). You should be careful to define your types such that they will be the same size (in bytes) on all architectures. For example, the `long` type is dangerous because it is 4 bytes on some machines and 8 bytes on others, whereas `int` type is 4 bytes on most Unix machines. A reasonable implementation of the `int4` type on Unix machines might be:

```
/* 4-byte integer, passed by value */
typedef int int4;
```

On the other hand, fixed-length types of any size can be passed by-reference. For example, here is a sample implementation of a PostgreSQL type:

```
/* 16-byte structure, passed by reference */
typedef struct
{
    double  x, y;
} Point;
```

Only pointers to such types can be used when passing them in and out of PostgreSQL functions. To return a value of such a type, allocate the right amount of memory with `palloc`, fill in the allocated memory, and return a pointer to it. (Also, if you just want to return the same value as one of your input arguments that's of the same data type, you can skip the extra `palloc` and just return the pointer to the input value.)

Finally, all variable-length types must also be passed by reference. All variable-length types must begin with an opaque length field of exactly 4 bytes, which will be set by `SET_VARSIZE`; never set this field directly! All data to be stored within that type must be located in the memory immediately following that length field. The length field contains the total length of the structure, that is, it includes the size of the length field itself.

Another important point is to avoid leaving any uninitialized bits within data type values; for example, take care to zero out any alignment padding bytes that might be present in structs. Without this, logically-equivalent constants of your data type might be seen as unequal by the planner, leading to inefficient (though not incorrect) plans.

> **警告**
>
> Never modify the contents of a pass-by-reference input value. If you do so you are likely to corrupt on-disk data, since the pointer you are given might point directly into a disk buffer. The sole exception to this rule is explained in 第 35. 10 节.

As an example, we can define the type `text` as follows:

```
typedef struct {
    int4 length;
    char data[1];
} text;
```

Obviously, the data field declared here is not long enough to hold all possible strings. Since it's impossible to declare a variable-size structure in C, we rely on the knowledge that the C compiler won't range-check array subscripts. We just allocate the necessary amount of space and then access the array as if it were declared the right length. (This is a common trick, which you can read about in many textbooks about C.)

When manipulating variable-length types, we must be careful to allocate the correct amount of memory and set the length field correctly. For example, if we wanted to store 40 bytes in a `text` structure, we might use a code fragment like this:

```
#include "postgres.h"
...
char buffer[40]; /* our source data */
...
text *destination = (text *) palloc(VARHDRSZ + 40);
SET_VARSIZE(destination, VARHDRSZ + 40);
memcpy(destination->data, buffer, 40);
...
```

VARHDRSZ is the same as sizeof(int32), but it's considered good style to use the macro VARHDRSZ to refer to the size of the overhead for a variable-length type. Also, the length field must be set using the SET_VARSIZE macro, not by simple assignment.

表 35.1 specifies which C type corresponds to which SQL type when writing a C-language function that uses a built-in type of PostgreSQL. The "Defined In" column gives the header file that needs to be included to get the type definition. (The actual definition might be in a different file that is included by the listed file. It is recommended that users stick to the defined interface.) Note that you should always include postgres.h first in any source file, because it declares a number of things that you will need anyway.

表 35.1. Equivalent C Types for Built-in SQL Types

SQL Type	C Type	Defined In
abstime	AbsoluteTime	utils/nabstime.h
boolean	bool	postgres.h (maybe compiler built-in)
box	BOX*	utils/geo_decls.h
bytea	bytea*	postgres.h
"char"	char	(compiler built-in)
character	BpChar*	postgres.h
cid	CommandId	postgres.h
date	DateADT	utils/date.h
smallint (int2)	int2 or int16	postgres.h
int2vector	int2vector*	postgres.h
integer (int4)	int4 or int32	postgres.h
real (float4)	float4*	postgres.h
double precision (float8)	float8*	postgres.h
interval	Interval*	datatype/timestamp.h
lseg	LSEG*	utils/geo_decls.h
name	Name	postgres.h
oid	Oid	postgres.h
oidvector	oidvector*	postgres.h
path	PATH*	utils/geo_decls.h
point	POINT*	utils/geo_decls.h
regproc	regproc	postgres.h
reltime	RelativeTime	utils/nabstime.h
text	text*	postgres.h
tid	ItemPointer	storage/itemptr.h
time	TimeADT	utils/date.h
time with time zone	TimeTzADT	utils/date.h
timestamp	Timestamp*	datatype/timestamp.h
tinterval	TimeInterval	utils/nabstime.h
varchar	VarChar*	postgres.h
xid	TransactionId	postgres.h

Now that we've gone over all of the possible structures for base types, we can show some examples of real functions.

35.9.3. Version 0 Calling Conventions

We present the “old style” calling convention first – although this approach is now deprecated, it's easier to get a handle on initially. In the version-0 method, the arguments and result of the C function are just declared in normal C style, but being careful to use the C representation of each SQL data type as shown above.

Here are some examples:

```
#include "postgres.h"
#include <string.h>
#include "utils/geo_decls.h"

#ifdef PG_MODULE_MAGIC
PG_MODULE_MAGIC;
#endif

/* by value */

int
add_one(int arg)
{
    return arg + 1;
}

/* by reference, fixed length */

float8 *
add_one_float8(float8 *arg)
{
    float8    *result = (float8 *) palloc(sizeof(float8));

    *result = *arg + 1.0;

    return result;
}

Point *
makepoint(Point *pointx, Point *pointy)
{
    Point     *new_point = (Point *) palloc(sizeof(Point));

    new_point->x = pointx->x;
    new_point->y = pointy->y;

    return new_point;
}

/* by reference, variable length */

text *
copytext(text *t)
{
    /*
     * VARSIZE is the total size of the struct in bytes.
     */
    text *new_t = (text *) palloc(VARSIZE(t));
    SET_VARSIZE(new_t, VARSIZE(t));
```

```
    /*
     * VARDATA is a pointer to the data region of the struct.
     */
    memcpy((void *) VARDATA(new_t), /* destination */
           (void *) VARDATA(t),     /* source */
           VARSIZE(t) - VARHDRSZ);  /* how many bytes */
    return new_t;
}

text *
concat_text(text *arg1, text *arg2)
{
    int32 new_text_size = VARSIZE(arg1) + VARSIZE(arg2) - VARHDRSZ;
    text *new_text = (text *) palloc(new_text_size);

    SET_VARSIZE(new_text, new_text_size);
    memcpy(VARDATA(new_text), VARDATA(arg1), VARSIZE(arg1) -
 VARHDRSZ);
    memcpy(VARDATA(new_text) + (VARSIZE(arg1) - VARHDRSZ),
           VARDATA(arg2), VARSIZE(arg2) - VARHDRSZ);
    return new_text;
}
```

Supposing that the above code has been prepared in file `funcs.c` and compiled into a shared object, we could define the functions to PostgreSQL with commands like this:

```
CREATE FUNCTION add_one(integer) RETURNS integer
     AS 'DIRECTORY/funcs', 'add_one'
     LANGUAGE C STRICT;

-- note overloading of SQL function name "add_one"
CREATE FUNCTION add_one(double precision) RETURNS double precision
     AS 'DIRECTORY/funcs', 'add_one_float8'
     LANGUAGE C STRICT;

CREATE FUNCTION makepoint(point, point) RETURNS point
     AS 'DIRECTORY/funcs', 'makepoint'
     LANGUAGE C STRICT;

CREATE FUNCTION copytext(text) RETURNS text
     AS 'DIRECTORY/funcs', 'copytext'
     LANGUAGE C STRICT;

CREATE FUNCTION concat_text(text, text) RETURNS text
     AS 'DIRECTORY/funcs', 'concat_text'
     LANGUAGE C STRICT;
```

Here, *`DIRECTORY`* stands for the directory of the shared library file (for instance the PostgreSQL tutorial directory, which contains the code for the examples used in this section). (Better style would be to use just `'funcs'` in the `AS` clause, after having added *`DIRECTORY`* to the search path. In any case, we can omit the system-specific extension for a shared library, commonly `.so` or `.sl`.)

Notice that we have specified the functions as "strict", meaning that the system should automatically assume a null result if any input value is null. By doing this, we avoid having to check for null inputs in the function code. Without this, we'd have to check for null values explicitly, by checking for a null pointer for each pass-by-reference argument. (For pass-by-value arguments, we don't even have a way to check!)

Although this calling convention is simple to use, it is not very portable; on some architectures there are problems with passing data types that are smaller than `int` this way. Also, there is no simple way to return a null result, nor to cope with null arguments in any way other than making the function strict. The version-1 convention, presented next, overcomes these objections.

35.9.4. Version 1 Calling Conventions

The version-1 calling convention relies on macros to suppress most of the complexity of passing arguments and results. The C declaration of a version-1 function is always:

```
Datum funcname(PG_FUNCTION_ARGS)
```

In addition, the macro call:

```
PG_FUNCTION_INFO_V1(funcname);
```

must appear in the same source file. (Conventionally, it's written just before the function itself.) This macro call is not needed for `internal`-language functions, since PostgreSQL assumes that all internal functions use the version-1 convention. It is, however, required for dynamically-loaded functions.

In a version-1 function, each actual argument is fetched using a `PG_GETARG_xxx()` macro that corresponds to the argument's data type, and the result is returned using a `PG_RETURN_xxx()` macro for the return type. `PG_GETARG_xxx()` takes as its argument the number of the function argument to fetch, where the count starts at 0. `PG_RETURN_xxx()` takes as its argument the actual value to return.

Here we show the same functions as above, coded in version-1 style:

```
#include "postgres.h"
#include <string.h>
#include "fmgr.h"
#include "utils/geo_decls.h"

#ifdef PG_MODULE_MAGIC
PG_MODULE_MAGIC;
#endif

/* by value */

PG_FUNCTION_INFO_V1(add_one);

Datum
add_one(PG_FUNCTION_ARGS)
{
    int32   arg = PG_GETARG_INT32(0);

    PG_RETURN_INT32(arg + 1);
}

/* by reference, fixed length */

PG_FUNCTION_INFO_V1(add_one_float8);

Datum
add_one_float8(PG_FUNCTION_ARGS)
{
    /* The macros for FLOAT8 hide its pass-by-reference nature. */
    float8   arg = PG_GETARG_FLOAT8(0);
```

```
    PG_RETURN_FLOAT8(arg + 1.0);
}

PG_FUNCTION_INFO_V1(makepoint);

Datum
makepoint(PG_FUNCTION_ARGS)
{
    /* Here, the pass-by-reference nature of Point is not hidden.
 */
    Point     *pointx = PG_GETARG_POINT_P(0);
    Point     *pointy = PG_GETARG_POINT_P(1);
    Point     *new_point = (Point *) palloc(sizeof(Point));

    new_point->x = pointx->x;
    new_point->y = pointy->y;

    PG_RETURN_POINT_P(new_point);
}

/* by reference, variable length */

PG_FUNCTION_INFO_V1(copytext);

Datum
copytext(PG_FUNCTION_ARGS)
{
    text     *t = PG_GETARG_TEXT_P(0);
    /*
     * VARSIZE is the total size of the struct in bytes.
     */
    text     *new_t = (text *) palloc(VARSIZE(t));
    SET_VARSIZE(new_t, VARSIZE(t));
    /*
     * VARDATA is a pointer to the data region of the struct.
     */
    memcpy((void *) VARDATA(new_t), /* destination */
           (void *) VARDATA(t),     /* source */
           VARSIZE(t) - VARHDRSZ);  /* how many bytes */
    PG_RETURN_TEXT_P(new_t);
}

PG_FUNCTION_INFO_V1(concat_text);

Datum
concat_text(PG_FUNCTION_ARGS)
{
    text  *arg1 = PG_GETARG_TEXT_P(0);
    text  *arg2 = PG_GETARG_TEXT_P(1);
    int32 new_text_size = VARSIZE(arg1) + VARSIZE(arg2) - VARHDRSZ;
    text *new_text = (text *) palloc(new_text_size);

    SET_VARSIZE(new_text, new_text_size);
    memcpy(VARDATA(new_text), VARDATA(arg1), VARSIZE(arg1) -
 VARHDRSZ);
    memcpy(VARDATA(new_text) + (VARSIZE(arg1) - VARHDRSZ),
           VARDATA(arg2), VARSIZE(arg2) - VARHDRSZ);
```

```
    PG_RETURN_TEXT_P(new_text);
}
```

The `CREATE FUNCTION` commands are the same as for the version-0 equivalents.

At first glance, the version-1 coding conventions might appear to be just pointless obscurantism. They do, however, offer a number of improvements, because the macros can hide unnecessary detail. An example is that in coding `add_one_float8`, we no longer need to be aware that `float8` is a pass-by-reference type. Another example is that the `GETARG` macros for variable-length types allow for more efficient fetching of "toasted" (compressed or out-of-line) values.

One big improvement in version-1 functions is better handling of null inputs and results. The macro `PG_ARGISNULL(n)` allows a function to test whether each input is null. (Of course, doing this is only necessary in functions not declared "strict".) As with the `PG_GETARG_xxx()` macros, the input arguments are counted beginning at zero. Note that one should refrain from executing `PG_GETARG_xxx()` until one has verified that the argument isn't null. To return a null result, execute `PG_RETURN_NULL()`; this works in both strict and nonstrict functions.

Other options provided in the new-style interface are two variants of the `PG_GETARG_xxx()` macros. The first of these, `PG_GETARG_xxx_COPY()`, guarantees to return a copy of the specified argument that is safe for writing into. (The normal macros will sometimes return a pointer to a value that is physically stored in a table, which must not be written to. Using the `PG_GETARG_xxx_COPY()` macros guarantees a writable result.) The second variant consists of the `PG_GETARG_xxx_SLICE()` macros which take three arguments. The first is the number of the function argument (as above). The second and third are the offset and length of the segment to be returned. Offsets are counted from zero, and a negative length requests that the remainder of the value be returned. These macros provide more efficient access to parts of large values in the case where they have storage type "external". (The storage type of a column can be specified using `ALTER TABLE tablename ALTER COLUMN colname SET STORAGE storagetype`. `storagetype` is one of `plain`, `external`, `extended`, or `main`.)

Finally, the version-1 function call conventions make it possible to return set results (第 35.9.9 节) and implement trigger functions (第 36 章) and procedural-language call handlers (第 49 章). Version-1 code is also more portable than version-0, because it does not break restrictions on function call protocol in the C standard. For more details see `src/backend/utils/fmgr/README` in the source distribution.

35.9.5. Writing Code

Before we turn to the more advanced topics, we should discuss some coding rules for PostgreSQL C-language functions. While it might be possible to load functions written in languages other than C into PostgreSQL, this is usually difficult (when it is possible at all) because other languages, such as C++, FORTRAN, or Pascal often do not follow the same calling convention as C. That is, other languages do not pass argument and return values between functions in the same way. For this reason, we will assume that your C-language functions are actually written in C.

The basic rules for writing and building C functions are as follows:

- Use `pg_config --includedir-server` to find out where the PostgreSQL server header files are installed on your system (or the system that your users will be running on).
- Compiling and linking your code so that it can be dynamically loaded into PostgreSQL always requires special flags. See 第 35.9.6 节 for a detailed explanation of how to do it for your particular operating system.

- Remember to define a “magic block” for your shared library, as described in 第 35.9.1 节.
- When allocating memory, use the PostgreSQL functions `palloc` and `pfree` instead of the corresponding C library functions `malloc` and `free`. The memory allocated by `palloc` will be freed automatically at the end of each transaction, preventing memory leaks.
- Always zero the bytes of your structures using `memset` (or allocate them with `palloc0` in the first place). Even if you assign to each field of your structure, there might be alignment padding (holes in the structure) that contain garbage values. Without this, it's difficult to support hash indexes or hash joins, as you must pick out only the significant bits of your data structure to compute a hash. The planner also sometimes relies on comparing constants via bitwise equality, so you can get undesirable planning results if logically-equivalent values aren't bitwise equal.
- Most of the internal PostgreSQL types are declared in `postgres.h`, while the function manager interfaces (`PG_FUNCTION_ARGS`, etc.) are in `fmgr.h`, so you will need to include at least these two files. For portability reasons it's best to include `postgres.h` first, before any other system or user header files. Including `postgres.h` will also include `elog.h` and `palloc.h` for you.
- Symbol names defined within object files must not conflict with each other or with symbols defined in the PostgreSQL server executable. You will have to rename your functions or variables if you get error messages to this effect.

35.9.6. 编译和链接动态装载函数

在你能够使用以 C 编写的 PostgreSQL 扩展函数之前，必须以特殊方式对它们进行编译和链接，以生成一个可由服务器动态装载的文件。更准确地说，需要创建一个共享库。

若想了解本节未涵盖的信息，你应阅读操作系统的文档，特别是 C 编译器 `cc` 和链接编辑器 `ld` 的手册页。此外，PostgreSQL 源代码在 `contrib` 目录中包含若干可用的示例。不过，如果你依赖这些示例，就会使你的模块依赖于 PostgreSQL 源代码是否可用。

创建共享库通常与链接可执行文件类似：先把源文件编译为目标文件，
再把目标文件链接在一起。目标文件需要以位置无关代码（PIC）形式生成。从概念上讲，
这意味着当它们被可执行文件装载时，可以放在内存中的任意位置。
（面向可执行文件的目标文件通常不会这样编译。）
链接共享库的命令中也包含一些特殊标志，
用来把它与链接可执行文件的命令区分开来（至少理论上如此，
某些系统上的实际做法要丑陋得多）。

在下面的示例中，我们假定你的源代码位于文件 `foo.c` 中，并将创建共享库 `foo.so`。除非另有说明，中间目标文件名为 `foo.o`。共享库可以包含多个目标文件，但这里我们只使用一个。

FreeBSD

生成 PIC 的编译器选项是 `-fPIC`。创建共享库时使用的编译器选项是 `-shared`。

```
gcc -fPIC -c foo.c
gcc -shared -o foo.so foo.o
```

这从 FreeBSD 3.0 版起适用。

HP-UX

生成 PIC 的系统编译器选项是 `+z`。使用 GCC 时则是 `-fPIC`。用于共享库的链接器选项是 `-b`。因此：

```
cc +z -c foo.c
```

或者：

```
gcc -fPIC -c foo.c
```

然后：

```
ld -b -o foo.sl foo.o
```

HP-UX 使用 `.sl` 作为共享库扩展名，这与大多数其他系统不同。

IRIX

PIC是默认的，不需要特殊的编译器选项。生成共享库的链接器选项是`-shared`。

```
cc -c foo.c
ld -shared -o foo.so foo.o
```

Linux

生成 PIC 的编译器选项是 `-fPIC`。创建共享库的编译器选项是 `-shared`。完整示例如下：

```
cc -fPIC -c foo.c
cc -shared -o foo.so foo.o
```

OS X

下面是一个示例。它假定开发者工具已经安装。

```
cc -c foo.c
cc -bundle -flat_namespace -undefined suppress -o foo.so foo.o
```

NetBSD

生成 PIC 的编译器选项是 `-fPIC`。对于 ELF 系统，使用带有 `-shared` 选项的编译器来链接共享库。对于较早的非 ELF 系统，则使用 `ld -Bshareable`。

```
gcc -fPIC -c foo.c
gcc -shared -o foo.so foo.o
```

OpenBSD

生成 PIC 的编译器选项是 `-fPIC`。使用 `ld -Bshareable` 来链接共享库。

```
gcc -fPIC -c foo.c
ld -Bshareable -o foo.so foo.o
```

Solaris

生成 PIC 的编译器选项，对于 Sun 编译器是 `-KPIC`，而 `-fPIC` 则用于 GCC。链接共享库时，两种编译器都可以使用 `-G`，或者也可以改用 `-shared`，但这只适用于 GCC。

```
cc -KPIC -c foo.c
cc -G -o foo.so foo.o
```

或者

```
gcc -fPIC -c foo.c
```

```
gcc -G -o foo.so foo.o
```

Tru64 UNIX

PIC 是默认值，因此编译命令就是常规的那条。链接时需要使用带特殊选项的`ld`。

```
cc -c foo.c
ld -shared -expect_unresolved '*' -o foo.so foo.o
```

使用 GCC 代替系统编译器时过程相同；不需要特殊选项。

UnixWare

生成 PIC 的编译器选项，对于 SCO 编译器是 `-K PIC`，而 `-fpic` 则用于 GCC。链接共享库时，编译器选项对于 SCO 编译器是 `-G`，对于 GCC 则是 `-shared`。

```
cc -K PIC -c foo.c
cc -G -o foo.so foo.o
```

或者

```
gcc -fpic -c foo.c
gcc -shared -o foo.so foo.o
```

提示

如果这些内容对你来说过于复杂，可以考虑使用 GNU Libtool[1]，它通过统一接口隐藏了平台差异。

生成的共享库文件随后就可以装载到 PostgreSQL 中。在向 `CREATE FUNCTION` 命令指定文件名时，必须给出共享库文件名，而不是中间目标文件名。请注意，系统标准的共享库扩展名（通常是 `.so` 或 `.sl`）可以在 `CREATE FUNCTION` 命令中省略，并且通常也应省略，以获得最佳可移植性。

关于服务器期望在何处找到共享库文件，请回头参见第 35.9.1 节。

35.9.7. Composite-type Arguments

Composite types do not have a fixed layout like C structures. Instances of a composite type can contain null fields. In addition, composite types that are part of an inheritance hierarchy can have different fields than other members of the same inheritance hierarchy. Therefore, PostgreSQL provides a function interface for accessing fields of composite types from C.

Suppose we want to write a function to answer the query:

```
SELECT name, c_overpaid(emp, 1500) AS overpaid
    FROM emp
    WHERE name = 'Bill' OR name = 'Sam';
```

Using call conventions version 0, we can define `c_overpaid` as:

```
#include "postgres.h"
```

[1] http://www.gnu.org/software/libtool/

```
#include "executor/executor.h"  /* for GetAttributeByName() */

#ifdef PG_MODULE_MAGIC
PG_MODULE_MAGIC;
#endif

bool
c_overpaid(HeapTupleHeader t, /* the current row of emp */
           int32 limit)
{
    bool isnull;
    int32 salary;

    salary = DatumGetInt32(GetAttributeByName(t, "salary", &isnull)
);
    if (isnull)
        return false;
    return salary > limit;
}
```

In version-1 coding, the above would look like this:

```
#include "postgres.h"
#include "executor/executor.h"  /* for GetAttributeByName() */

#ifdef PG_MODULE_MAGIC
PG_MODULE_MAGIC;
#endif

PG_FUNCTION_INFO_V1(c_overpaid);

Datum
c_overpaid(PG_FUNCTION_ARGS)
{
    HeapTupleHeader  t = PG_GETARG_HEAPTUPLEHEADER(0);
    int32            limit = PG_GETARG_INT32(1);
    bool isnull;
    Datum salary;

    salary = GetAttributeByName(t, "salary", &isnull);
    if (isnull)
        PG_RETURN_BOOL(false);
    /* Alternatively, we might prefer to do PG_RETURN_NULL() for
 null salary. */

    PG_RETURN_BOOL(DatumGetInt32(salary) > limit);
}
```

`GetAttributeByName` is the PostgreSQL system function that returns attributes out of the specified row. It has three arguments: the argument of type `HeapTupleHeader` passed into the function, the name of the desired attribute, and a return parameter that tells whether the attribute is null. `GetAttributeByName` returns a `Datum` value that you can convert to the proper data type by using the appropriate `DatumGet`*`XXX`*`()` macro. Note that the return value is meaningless if the null flag is set; always check the null flag before trying to do anything with the result.

There is also `GetAttributeByNum`, which selects the target attribute by column number instead of name.

The following command declares the function `c_overpaid` in SQL:

```
CREATE FUNCTION c_overpaid(emp, integer) RETURNS boolean
    AS 'DIRECTORY/funcs', 'c_overpaid'
    LANGUAGE C STRICT;
```

Notice we have used `STRICT` so that we did not have to check whether the input arguments were NULL.

35.9.8. Returning Rows (Composite Types)

To return a row or composite-type value from a C-language function, you can use a special API that provides macros and functions to hide most of the complexity of building composite data types. To use this API, the source file must include:

```
#include "funcapi.h"
```

There are two ways you can build a composite data value (henceforth a "tuple"): you can build it from an array of Datum values, or from an array of C strings that can be passed to the input conversion functions of the tuple's column data types. In either case, you first need to obtain or construct a `TupleDesc` descriptor for the tuple structure. When working with Datums, you pass the `TupleDesc` to `BlessTupleDesc`, and then call `heap_form_tuple` for each row. When working with C strings, you pass the `TupleDesc` to `TupleDescGetAttInMetadata`, and then call `BuildTupleFromCStrings` for each row. In the case of a function returning a set of tuples, the setup steps can all be done once during the first call of the function.

Several helper functions are available for setting up the needed `TupleDesc`. The recommended way to do this in most functions returning composite values is to call:

```
TypeFuncClass get_call_result_type(FunctionCallInfo fcinfo,
                                   Oid *resultTypeId,
                                   TupleDesc *resultTupleDesc)
```

passing the same `fcinfo` struct passed to the calling function itself. (This of course requires that you use the version-1 calling conventions.) `resultTypeId` can be specified as `NULL` or as the address of a local variable to receive the function's result type OID. `resultTupleDesc` should be the address of a local `TupleDesc` variable. Check that the result is `TYPEFUNC_COMPOSITE`; if so, `resultTupleDesc` has been filled with the needed `TupleDesc`. (If it is not, you can report an error along the lines of "function returning record called in context that cannot accept type record".)

> **提示**
>
> `get_call_result_type` can resolve the actual type of a polymorphic function result; so it is useful in functions that return scalar polymorphic results, not only functions that return composites. The `resultTypeId` output is primarily useful for functions returning polymorphic scalars.

> **注意**
>
> `get_call_result_type` has a sibling `get_expr_result_type`, which can be used to resolve the expected output type for a function call represented by an expression tree. This can be used when trying to determine the result type from

> outside the function itself. There is also `get_func_result_type`, which can be used when only the function's OID is available. However these functions are not able to deal with functions declared to return `record`, and `get_func_result_type` cannot resolve polymorphic types, so you should preferentially use `get_call_result_type`.

Older, now-deprecated functions for obtaining `TupleDesc`s are:

```
TupleDesc RelationNameGetTupleDesc(const char *relname)
```

to get a `TupleDesc` for the row type of a named relation, and:

```
TupleDesc TypeGetTupleDesc(Oid typeoid, List *colaliases)
```

to get a `TupleDesc` based on a type OID. This can be used to get a `TupleDesc` for a base or composite type. It will not work for a function that returns `record`, however, and it cannot resolve polymorphic types.

Once you have a `TupleDesc`, call:

```
TupleDesc BlessTupleDesc(TupleDesc tupdesc)
```

if you plan to work with Datums, or:

```
AttInMetadata *TupleDescGetAttInMetadata(TupleDesc tupdesc)
```

if you plan to work with C strings. If you are writing a function returning set, you can save the results of these functions in the `FuncCallContext` structure – use the `tuple_desc` or `attinmeta` field respectively.

When working with Datums, use:

```
HeapTuple heap_form_tuple(TupleDesc tupdesc, Datum *values, bool
 *isnull)
```

to build a `HeapTuple` given user data in Datum form.

When working with C strings, use:

```
HeapTuple BuildTupleFromCStrings(AttInMetadata *attinmeta, char
 **values)
```

to build a `HeapTuple` given user data in C string form. `values` is an array of C strings, one for each attribute of the return row. Each C string should be in the form expected by the input function of the attribute data type. In order to return a null value for one of the attributes, the corresponding pointer in the *`values`* array should be set to `NULL`. This function will need to be called again for each row you return.

Once you have built a tuple to return from your function, it must be converted into a `Datum`. Use:

```
HeapTupleGetDatum(HeapTuple tuple)
```

to convert a `HeapTuple` into a valid Datum. This `Datum` can be returned directly if you intend to return just a single row, or it can be used as the current return value in a set-returning function.

An example appears in the next section.

35.9.9. Returning Sets

There is also a special API that provides support for returning sets (multiple rows) from a C-language function. A set-returning function must follow the version-1 calling conventions. Also, source files must include `funcapi.h`, as above.

A set-returning function (SRF) is called once for each item it returns. The SRF must therefore save enough state to remember what it was doing and return the next item on each call. The structure `FuncCallContext` is provided to help control this process. Within a function, `fcinfo->flinfo->fn_extra` is used to hold a pointer to `FuncCallContext` across calls.

```
typedef struct
{
    /*
     * Number of times we've been called before
     *
     * call_cntr is initialized to 0 for you by SRF_FIRSTCALL_INIT(
), and
     * incremented for you every time SRF_RETURN_NEXT() is called.
     */
    uint32 call_cntr;

    /*
     * OPTIONAL maximum number of calls
     *
     * max_calls is here for convenience only and setting it is
 optional.
     * If not set, you must provide alternative means to know when
 the
     * function is done.
     */
    uint32 max_calls;

    /*
     * OPTIONAL pointer to result slot
     *
     * This is obsolete and only present for backward
 compatibility, viz,
     * user-defined SRFs that use the deprecated TupleDescGetSlot()
.
     */
    TupleTableSlot *slot;

    /*
     * OPTIONAL pointer to miscellaneous user-provided context
 information
     *
     * user_fctx is for use as a pointer to your own data to retain
     * arbitrary context information between calls of your
 function.
     */
    void *user_fctx;

    /*
     * OPTIONAL pointer to struct containing attribute type input
 metadata
     *
```

```
     * attinmeta is for use when returning tuples (i.e., composite
 data types)
     * and is not used when returning base data types. It is only
 needed
     * if you intend to use BuildTupleFromCStrings() to create the
 return
     * tuple.
     */
    AttInMetadata *attinmeta;

    /*
     * memory context used for structures that must live for
 multiple calls
     *
     * multi_call_memory_ctx is set by SRF_FIRSTCALL_INIT() for
 you, and used
     * by SRF_RETURN_DONE() for cleanup. It is the most appropriate
 memory
     * context for any memory that is to be reused across multiple
 calls
     * of the SRF.
     */
    MemoryContext multi_call_memory_ctx;

    /*
     * OPTIONAL pointer to struct containing tuple description
     *
     * tuple_desc is for use when returning tuples (i.e., composite
 data types)
     * and is only needed if you are going to build the tuples with
     * heap_form_tuple() rather than with BuildTupleFromCStrings().
  Note that
     * the TupleDesc pointer stored here should usually have been
 run through
     * BlessTupleDesc() first.
     */
    TupleDesc tuple_desc;

} FuncCallContext;
```

An SRF uses several functions and macros that automatically manipulate the `FuncCallContext` structure (and expect to find it via `fn_extra`). Use:

```
SRF_IS_FIRSTCALL()
```

to determine if your function is being called for the first or a subsequent time. On the first call (only) use:

```
SRF_FIRSTCALL_INIT()
```

to initialize the `FuncCallContext`. On every function call, including the first, use:

```
SRF_PERCALL_SETUP()
```

to properly set up for using the `FuncCallContext` and clearing any previously returned data left over from the previous pass.

If your function has data to return, use:

```
SRF_RETURN_NEXT(funcctx, result)
```

to return it to the caller. (`result` must be of type `Datum`, either a single value or a tuple prepared as described above.) Finally, when your function is finished returning data, use:

```
SRF_RETURN_DONE(funcctx)
```

to clean up and end the SRF.

The memory context that is current when the SRF is called is a transient context that will be cleared between calls. This means that you do not need to call `pfree` on everything you allocated using `palloc`; it will go away anyway. However, if you want to allocate any data structures to live across calls, you need to put them somewhere else. The memory context referenced by `multi_call_memory_ctx` is a suitable location for any data that needs to survive until the SRF is finished running. In most cases, this means that you should switch into `multi_call_memory_ctx` while doing the first-call setup.

A complete pseudo-code example looks like the following:

```
Datum
my_set_returning_function(PG_FUNCTION_ARGS)
{
    FuncCallContext  *funcctx;
    Datum            result;
    further declarations as needed

    if (SRF_IS_FIRSTCALL())
    {
        MemoryContext oldcontext;

        funcctx = SRF_FIRSTCALL_INIT();
        oldcontext = MemoryContextSwitchTo(funcctx->multi_call_
memory_ctx);
        /* One-time setup code appears here: */
        user code
        if returning composite
            build TupleDesc, and perhaps AttInMetadata
        endif returning composite
        user code
        MemoryContextSwitchTo(oldcontext);
    }

    /* Each-time setup code appears here: */
    user code
    funcctx = SRF_PERCALL_SETUP();
    user code

    /* this is just one way we might test whether we are done: */
    if (funcctx->call_cntr < funcctx->max_calls)
    {
        /* Here we want to return another item: */
        user code
        obtain result Datum
        SRF_RETURN_NEXT(funcctx, result);
    }
    else
    {
        /* Here we are done returning items and just need to clean
 up: */
        user code
```

```
        SRF_RETURN_DONE(funcctx);
    }
}
```

A complete example of a simple SRF returning a composite type looks like:

```
PG_FUNCTION_INFO_V1(retcomposite);

Datum
retcomposite(PG_FUNCTION_ARGS)
{
    FuncCallContext     *funcctx;
    int                  call_cntr;
    int                  max_calls;
    TupleDesc            tupdesc;
    AttInMetadata       *attinmeta;

    /* stuff done only on the first call of the function */
    if (SRF_IS_FIRSTCALL())
    {
        MemoryContext   oldcontext;

        /* create a function context for cross-call persistence */
        funcctx = SRF_FIRSTCALL_INIT();

        /* switch to memory context appropriate for multiple
 function calls */
        oldcontext = MemoryContextSwitchTo(funcctx->multi_call_
memory_ctx);

        /* total number of tuples to be returned */
        funcctx->max_calls = PG_GETARG_UINT32(0);

        /* Build a tuple descriptor for our result type */
        if (get_call_result_type(fcinfo, NULL, &tupdesc) !=
 TYPEFUNC_COMPOSITE)
            ereport(ERROR,
                    (errcode(ERRCODE_FEATURE_NOT_SUPPORTED),
                     errmsg("function returning record called in
 context "
                            "that cannot accept type record")));

        /*
         * generate attribute metadata needed later to produce
 tuples from raw
         * C strings
         */
        attinmeta = TupleDescGetAttInMetadata(tupdesc);
        funcctx->attinmeta = attinmeta;

        MemoryContextSwitchTo(oldcontext);
    }

    /* stuff done on every call of the function */
    funcctx = SRF_PERCALL_SETUP();

    call_cntr = funcctx->call_cntr;
    max_calls = funcctx->max_calls;
```

```
    attinmeta = funcctx->attinmeta;

    if (call_cntr < max_calls)    /* do when there is more left to
 send */
    {
        char       **values;
        HeapTuple    tuple;
        Datum        result;

        /*
         * Prepare a values array for building the returned tuple.
         * This should be an array of C strings which will
         * be processed later by the type input functions.
         */
        values = (char **) palloc(3 * sizeof(char *));
        values[0] = (char *) palloc(16 * sizeof(char));
        values[1] = (char *) palloc(16 * sizeof(char));
        values[2] = (char *) palloc(16 * sizeof(char));

        snprintf(values[0], 16, "%d", 1 * PG_GETARG_INT32(1));
        snprintf(values[1], 16, "%d", 2 * PG_GETARG_INT32(1));
        snprintf(values[2], 16, "%d", 3 * PG_GETARG_INT32(1));

        /* build a tuple */
        tuple = BuildTupleFromCStrings(attinmeta, values);

        /* make the tuple into a datum */
        result = HeapTupleGetDatum(tuple);

        /* clean up (this is not really necessary) */
        pfree(values[0]);
        pfree(values[1]);
        pfree(values[2]);
        pfree(values);

        SRF_RETURN_NEXT(funcctx, result);
    }
    else    /* do when there is no more left */
    {
        SRF_RETURN_DONE(funcctx);
    }
}
```

One way to declare this function in SQL is:

```
CREATE TYPE __retcomposite AS (f1 integer, f2 integer, f3 integer);

CREATE OR REPLACE FUNCTION retcomposite(integer, integer)
    RETURNS SETOF __retcomposite
    AS 'filename', 'retcomposite'
    LANGUAGE C IMMUTABLE STRICT;
```

A different way is to use OUT parameters:

```
CREATE OR REPLACE FUNCTION retcomposite(IN integer, IN integer,
    OUT f1 integer, OUT f2 integer, OUT f3 integer)
    RETURNS SETOF record
    AS 'filename', 'retcomposite'
    LANGUAGE C IMMUTABLE STRICT;
```

Notice that in this method the output type of the function is formally an anonymous `record` type.

The directory contrib/tablefunc module in the source distribution contains more examples of set-returning functions.

35.9.10. Polymorphic Arguments and Return Types

C-language functions can be declared to accept and return the polymorphic types `anyelement`, `anyarray`, `anynonarray`, `anyenum`, and `anyrange`. See 第 35.2.5 节 for a more detailed explanation of polymorphic functions. When function arguments or return types are defined as polymorphic types, the function author cannot know in advance what data type it will be called with, or need to return. There are two routines provided in `fmgr.h` to allow a version-1 C function to discover the actual data types of its arguments and the type it is expected to return. The routines are called `get_fn_expr_rettype(FmgrInfo *flinfo)` and `get_fn_expr_argtype(FmgrInfo *flinfo, int argnum)`. They return the result or argument type OID, or `InvalidOid` if the information is not available. The structure `flinfo` is normally accessed as `fcinfo->flinfo`. The parameter `argnum` is zero based. `get_call_result_type` can also be used as an alternative to `get_fn_expr_rettype`.

For example, suppose we want to write a function to accept a single element of any type, and return a one-dimensional array of that type:

```
PG_FUNCTION_INFO_V1(make_array);
Datum
make_array(PG_FUNCTION_ARGS)
{
    ArrayType  *result;
    Oid         element_type = get_fn_expr_argtype(fcinfo->flinfo,
 0);
    Datum       element;
    bool        isnull;
    int16       typlen;
    bool        typbyval;
    char        typalign;
    int         ndims;
    int         dims[MAXDIM];
    int         lbs[MAXDIM];

    if (!OidIsValid(element_type))
        elog(ERROR, "could not determine data type of input");

    /* get the provided element, being careful in case it's NULL */
    isnull = PG_ARGISNULL(0);
    if (isnull)
        element = (Datum) 0;
    else
        element = PG_GETARG_DATUM(0);

    /* we have one dimension */
    ndims = 1;
    /* and one element */
    dims[0] = 1;
    /* and lower bound is 1 */
    lbs[0] = 1;

    /* get required info about the element type */
```

```
    get_typlenbyvalalign(element_type, &typlen, &typbyval,
 &typalign);

    /* now build the array */
    result = construct_md_array(&element, &isnull, ndims, dims,
 lbs,
                                element_type, typlen, typbyval,
 typalign);

    PG_RETURN_ARRAYTYPE_P(result);
}
```

The following command declares the function `make_array` in SQL:

```
CREATE FUNCTION make_array(anyelement) RETURNS anyarray
    AS 'DIRECTORY/funcs', 'make_array'
    LANGUAGE C IMMUTABLE;
```

There is a variant of polymorphism that is only available to C-language functions: they can be declared to take parameters of type `"any"`. (Note that this type name must be double-quoted, since it's also a SQL reserved word.) This works like `anyelement` except that it does not constrain different `"any"` arguments to be the same type, nor do they help determine the function's result type. A C-language function can also declare its final parameter to be `VARIADIC "any"`. This will match one or more actual arguments of any type (not necessarily the same type). These arguments will not be gathered into an array as happens with normal variadic functions; they will just be passed to the function separately. The `PG_NARGS()` macro and the methods described above must be used to determine the number of actual arguments and their types when using this feature.

35.9.11. Transform Functions

Some function calls can be simplified during planning based on properties specific to the function. For example, `int4mul(n, 1)` could be simplified to just `n`. To define such function-specific optimizations, write a transform function and place its OID in the `protransform` field of the primary function's `pg_proc` entry. The transform function must have the SQL signature `protransform(internal) RETURNS internal`. The argument, actually `FuncExpr *`, is a dummy node representing a call to the primary function. If the transform function's study of the expression tree proves that a simplified expression tree can substitute for all possible concrete calls represented thereby, build and return that simplified expression. Otherwise, return a `NULL` pointer (not a SQL null).

We make no guarantee that PostgreSQL will never call the primary function in cases that the transform function could simplify. Ensure rigorous equivalence between the simplified expression and an actual call to the primary function.

Currently, this facility is not exposed to users at the SQL level because of security concerns, so it is only practical to use for optimizing built-in functions.

35.9.12. Shared Memory and LWLocks

Add-ins can reserve LWLocks and an allocation of shared memory on server startup. The add-in's shared library must be preloaded by specifying it in shared_preload_libraries. Shared memory is reserved by calling:

```
void RequestAddinShmemSpace(int size)
```

from your `_PG_init` function.

LWLocks are reserved by calling:

```
void RequestAddinLWLocks(int n)
```

from `_PG_init`.

To avoid possible race-conditions, each backend should use the LWLock `AddinShmemInitLock` when connecting to and initializing its allocation of shared memory, as shown here:

```
static mystruct *ptr = NULL;

if (!ptr)
{
        bool    found;

        LWLockAcquire(AddinShmemInitLock, LW_EXCLUSIVE);
        ptr = ShmemInitStruct("my struct name", size, &found);
        if (!found)
        {
                initialize contents of shmem area;
                acquire any requested LWLocks using:
                ptr->mylockid = LWLockAssign();
        }
        LWLockRelease(AddinShmemInitLock);
}
```

35.9.13. Using C++ for Extensibility

Although the PostgreSQL backend is written in C, it is possible to write extensions in C++ if these guidelines are followed:

- All functions accessed by the backend must present a C interface to the backend; these C functions can then call C++ functions. For example, `extern C` linkage is required for backend-accessed functions. This is also necessary for any functions that are passed as pointers between the backend and C++ code.
- Free memory using the appropriate deallocation method. For example, most backend memory is allocated using `palloc()`, so use `pfree()` to free it. Using C++ `delete` in such cases will fail.
- Prevent exceptions from propagating into the C code (use a catch-all block at the top level of all `extern C` functions). This is necessary even if the C++ code does not explicitly throw any exceptions, because events like out-of-memory can still throw exceptions. Any exceptions must be caught and appropriate errors passed back to the C interface. If possible, compile C++ with `-fno-exceptions` to eliminate exceptions entirely; in such cases, you must check for failures in your C++ code, e.g. check for NULL returned by `new()`.
- If calling backend functions from C++ code, be sure that the C++ call stack contains only plain old data structures (POD). This is necessary because backend errors generate a distant `longjmp()` that does not properly unroll a C++ call stack with non-POD objects.

In summary, it is best to place C++ code behind a wall of `extern C` functions that interface to the backend, and avoid exception, memory, and call stack leakage.

35.10. User-defined Aggregates

Aggregate functions in PostgreSQL are expressed in terms of state values and state transition functions. That is, an aggregate operates using a state value that is updated

as each successive input row is processed. To define a new aggregate function, one selects a data type for the state value, an initial value for the state, and a state transition function. The state transition function is just an ordinary function that could also be used outside the context of the aggregate. A final function can also be specified, in case the desired result of the aggregate is different from the data that needs to be kept in the running state value.

Thus, in addition to the argument and result data types seen by a user of the aggregate, there is an internal state-value data type that might be different from both the argument and result types.

If we define an aggregate that does not use a final function, we have an aggregate that computes a running function of the column values from each row. `sum` is an example of this kind of aggregate. `sum` starts at zero and always adds the current row's value to its running total. For example, if we want to make a `sum` aggregate to work on a data type for complex numbers, we only need the addition function for that data type. The aggregate definition would be:

```
CREATE AGGREGATE sum (complex)
(
    sfunc = complex_add,
    stype = complex,
    initcond = '(0,0)'
);

SELECT sum(a) FROM test_complex;

   sum
-----------
 (34,53.9)
```

(Notice that we are relying on function overloading: there is more than one aggregate named `sum`, but PostgreSQL can figure out which kind of sum applies to a column of type `complex`.)

The above definition of `sum` will return zero (the initial state condition) if there are no nonnull input values. Perhaps we want to return null in that case instead – the SQL standard expects `sum` to behave that way. We can do this simply by omitting the `initcond` phrase, so that the initial state condition is null. Ordinarily this would mean that the `sfunc` would need to check for a null state-condition input. But for `sum` and some other simple aggregates like `max` and `min`, it is sufficient to insert the first nonnull input value into the state variable and then start applying the transition function at the second nonnull input value. PostgreSQL will do that automatically if the initial condition is null and the transition function is marked "strict" (i.e., not to be called for null inputs).

Another bit of default behavior for a "strict" transition function is that the previous state value is retained unchanged whenever a null input value is encountered. Thus, null values are ignored. If you need some other behavior for null inputs, do not declare your transition function as strict; instead code it to test for null inputs and do whatever is needed.

`avg` (average) is a more complex example of an aggregate. It requires two pieces of running state: the sum of the inputs and the count of the number of inputs. The final result is obtained by dividing these quantities. Average is typically implemented by using an array as the state value. For example, the built-in implementation of `avg(float8)` looks like:

```
CREATE AGGREGATE avg (float8)
(
```

```
    sfunc = float8_accum,
    stype = float8[],
    finalfunc = float8_avg,
    initcond = '{0,0,0}'
);
```

(`float8_accum` requires a three-element array, not just two elements, because it accumulates the sum of squares as well as the sum and count of the inputs. This is so that it can be used for some other aggregates besides `avg`.)

Aggregate functions can use polymorphic state transition functions or final functions, so that the same functions can be used to implement multiple aggregates. See 第 35.2.5 节 for an explanation of polymorphic functions. Going a step further, the aggregate function itself can be specified with polymorphic input type(s) and state type, allowing a single aggregate definition to serve for multiple input data types. Here is an example of a polymorphic aggregate:

```
CREATE AGGREGATE array_accum (anyelement)
(
    sfunc = array_append,
    stype = anyarray,
    initcond = '{}'
);
```

Here, the actual state type for any aggregate call is the array type having the actual input type as elements. The behavior of the aggregate is to concatenate all the inputs into an array of that type. (Note: the built-in aggregate `array_agg` provides similar functionality, with better performance than this definition would have.)

Here's the output using two different actual data types as arguments:

```
SELECT attrelid::regclass, array_accum(attname)
    FROM pg_attribute
    WHERE attnum > 0 AND attrelid = 'pg_tablespace'::regclass
    GROUP BY attrelid;

   attrelid    |              array_accum
---------------+---------------------------------------
 pg_tablespace | {spcname,spcowner,spcacl,spcoptions}
(1 row)

SELECT attrelid::regclass, array_accum(atttypid::regtype)
    FROM pg_attribute
    WHERE attnum > 0 AND attrelid = 'pg_tablespace'::regclass
    GROUP BY attrelid;

   attrelid    |        array_accum
---------------+---------------------------
 pg_tablespace | {name,oid,aclitem[],text[]}
(1 row)
```

A function written in C can detect that it is being called as an aggregate transition or final function by calling `AggCheckCallContext`, for example:

```
if (AggCheckCallContext(fcinfo, NULL))
```

One reason for checking this is that when it is true for a transition function, the first input must be a temporary transition value and can therefore safely be modified in-place rather than allocating a new copy. See `int8inc()` for an example. (This is the only case where it is safe for a function to modify a pass-by-reference input.

In particular, aggregate final functions should not modify their inputs in any case, because in some cases they will be re-executed on the same final transition value.)

For further details see the CREATE AGGREGATE command.

35.11. 用户定义的类型

如第 35.2 节中所述，PostgreSQL可以扩展以支持新的数据类型。本节描述如何定义新的基础类型，也就是在SQL语言层之下定义的数据类型。创建新的基础类型需要用底层语言（通常是C）实现操作该类型的函数。

本节中的示例位于源码分发包的`src/tutorial`目录中的 `complex.sql`和`complex.c`。关于如何运行这些示例，请参见该目录中的`README`文件。

用户定义类型必须始终具有输入函数和输出函数。这些函数决定该类型如何以字符串形式出现（供用户输入和向用户输出），以及该类型在内存中如何组织。
输入函数接受一个以空字符结尾的字符串作为参数，并返回该类型的内部（内存中）表示。
输出函数接受该类型的内部表示作为参数，并返回一个以空字符结尾的字符串。
如果我们希望该类型除了存储之外还能做别的事情，
就必须提供额外的函数来实现我们希望该类型支持的各种操作。

假设我们要定义一种表示复数的类型`complex`。
在内存中表示复数的一种自然方式是下面这个 C 结构体：

```
typedef struct Complex {
    double      x;
    double      y;
} Complex;
```

我们需要把它做成按引用传递的类型，因为它太大了，无法放进单个 `Datum`值中。

作为该类型的外部字符串表示，我们选择形如`(x,y)`的字符串。

输入函数和输出函数通常都不难编写，尤其是输出函数。
但是在定义该类型的外部字符串表示时，要记住，
最终你必须为这种表示编写一个完整而健壮的解析器，作为输入函数。例如：

```
PG_FUNCTION_INFO_V1(complex_in);

Datum
complex_in(PG_FUNCTION_ARGS)
{
    char       *str = PG_GETARG_CSTRING(0);
    double      x,
                y;
    Complex    *result;

    if (sscanf(str, " ( %lf , %lf )", &x, &y) != 2)
        ereport(ERROR,
                (errcode(ERRCODE_INVALID_TEXT_REPRESENTATION),
                 errmsg("invalid input syntax for complex: \"%s\"",
                        str)));

    result = (Complex *) palloc(sizeof(Complex));
    result->x = x;
    result->y = y;
    PG_RETURN_POINTER(result);
}
```

输出函数可以简单地写成：

```
PG_FUNCTION_INFO_V1(complex_out);

Datum
complex_out(PG_FUNCTION_ARGS)
{
    Complex    *complex = (Complex *) PG_GETARG_POINTER(0);
    char       *result;

    result = (char *) palloc(100);
    snprintf(result, 100, "(%g,%g)", complex->x, complex->y);
    PG_RETURN_CSTRING(result);
}
```

应当注意让输入函数和输出函数互为逆运算。如果不是这样，
当你需要把数据转储到文件中再读回时，就会遇到严重问题。这在涉及浮点数时尤为常见。

可选地，用户定义类型还可以提供二进制输入和输出例程。二进制 I/O 通常比文本 I/O 更快，但可移植性较差。与文本 I/O 一样，外部二进制表示的精确定义完全由你决定。大多数内置数据类型都尽量提供与机器无关的二进制表示。对于`complex`，我们将借助类型`float8`
的二进制 I/O 转换器：

```
PG_FUNCTION_INFO_V1(complex_recv);

Datum
complex_recv(PG_FUNCTION_ARGS)
{
    StringInfo  buf = (StringInfo) PG_GETARG_POINTER(0);
    Complex    *result;

    result = (Complex *) palloc(sizeof(Complex));
    result->x = pq_getmsgfloat8(buf);
    result->y = pq_getmsgfloat8(buf);
    PG_RETURN_POINTER(result);
}

PG_FUNCTION_INFO_V1(complex_send);

Datum
complex_send(PG_FUNCTION_ARGS)
{
    Complex    *complex = (Complex *) PG_GETARG_POINTER(0);
    StringInfoData buf;

    pq_begintypsend(&buf);
    pq_sendfloat8(&buf, complex->x);
    pq_sendfloat8(&buf, complex->y);
    PG_RETURN_BYTEA_P(pq_endtypsend(&buf));
}
```

一旦我们写好了 I/O 函数并将它们编译进共享库，就可以在 SQL 中定义 `complex`类型。首先将其声明为一种 shell 类型：

```
CREATE TYPE complex;
```

这会建立一个占位符，使我们可以在定义其 I/O 函数时引用该类型。现在我们可以定义这些 I/O 函数：

```
CREATE FUNCTION complex_in(cstring)
    RETURNS complex
```

```
    AS 'filename'
    LANGUAGE C IMMUTABLE STRICT;

CREATE FUNCTION complex_out(complex)
    RETURNS cstring
    AS 'filename'
    LANGUAGE C IMMUTABLE STRICT;

CREATE FUNCTION complex_recv(internal)
   RETURNS complex
   AS 'filename'
   LANGUAGE C IMMUTABLE STRICT;

CREATE FUNCTION complex_send(complex)
   RETURNS bytea
   AS 'filename'
   LANGUAGE C IMMUTABLE STRICT;
```

最后，我们可以给出该数据类型的完整定义：

```
CREATE TYPE complex (
   internallength = 16,
   input = complex_in,
   output = complex_out,
   receive = complex_recv,
   send = complex_send,
   alignment = double
);
```

当你定义一种新的基础类型时，PostgreSQL会自动提供对该类型数组的支持。该数组类型通常与基础类型同名，只是在前面加一个下划线字符（_）。

一旦该数据类型存在，我们就可以声明额外的函数，为该数据类型提供有用的操作。
随后可以在这些函数之上定义操作符；如果需要，
还可以创建操作符类以支持该数据类型的索引。这些附加层会在后续各节中讨论。

如果你的数据类型的值在内部形式上的大小可变，通常最好让该数据类型支持
TOAST（见第 56.2 节）。即使这些值总是小到不需要压缩或外部存储，也应该这样做，因为TOAST还可以通过减少头部开销来为小数据节省空间。

为此，内部表示必须遵循可变长度数据的标准布局：前四个字节必须是一个从不直接访问的`char[4]`字段（惯例上命名为 `vl_len_`）。必须使用 `SET_VARSIZE()`在该字段中存储该 datum 的大小，并使用 `VARSIZE()`取回它。操作该数据类型的 C 级函数必须始终使用`PG_DETOAST_DATUM`对传给它们的任何 TOAST 化值执行去 TOAST 化处理。（这一细节通常通过定义特定于该类型的 `GETARG_DATATYPE_P`宏来隐藏。）然后，在执行 `CREATE TYPE`命令时，把内部长度指定为 `variable`并选择合适的存储选项。

如果数据对齐并不重要（无论只是对某个特定函数而言，
还是因为该数据类型本来就指定了字节对齐），那么就有可能避免 `PG_DETOAST_DATUM`的一部分开销。你可以改用 `PG_DETOAST_DATUM_PACKED`（通常通过定义 `GETARG_DATATYPE_PP`宏来隐藏），并使用 `VARSIZE_ANY_EXHDR`和`VARDATA_ANY` 宏访问一个可能采用打包形式的 datum。再次注意，即使数据类型定义指定了对齐方式，这些宏返回的数据也不是对齐的。如果对齐很重要，就必须使用常规的`PG_DETOAST_DATUM`接口。

> **注意**
>
> 较旧的代码常把`vl_len_`声明为 `int32`字段，而不是`char[4]`字段。只要结构体定义中还有其他至少按`int32`对齐的字段，这样做是可以的。但在处理可能未对齐的

> datum 时使用这种结构体定义就很危险；编译器可能据此假定该 datum 实际上是对齐的，从而在对齐要求严格的体系结构上导致核心转储。

更多细节见CREATE TYPE命令的说明。

35.12. 用户定义的操作符

每个操作符都是对执行实际工作的底层函数调用的一种“语法糖”；因此，
你必须先创建底层函数，才能创建操作符。不过，操作符并不仅仅是语法糖，
因为它还携带一些额外信息，可帮助查询规划器优化使用该操作符的查询。
下一节将专门解释这些附加信息。

PostgreSQL支持左一元、右一元和二元操作符。操作符可以被重载；也就是说，
同一个操作符名可以用于不同的操作符，而这些操作符具有不同数量和类型的操作数。
执行查询时，系统会根据提供的操作数数量和类型确定应调用哪个操作符。

下面是创建一个用于将两个复数相加的操作符的示例。假定我们已经创建了类型`complex`的定义（见第 35.11 节）。首先需要一个完成实际工作的函数，然后就可以定义操作符：

```
CREATE FUNCTION complex_add(complex, complex)
    RETURNS complex
    AS 'filename', 'complex_add'
    LANGUAGE C IMMUTABLE STRICT;

CREATE OPERATOR + (
    leftarg = complex,
    rightarg = complex,
    procedure = complex_add,
    commutator = +
);
```

现在我们就可以执行下面这样的查询：

```
SELECT (a + b) AS c FROM test_complex;

        c
-----------------
 (5.2,6.05)
 (133.42,144.95)
```

上面展示了如何创建一个二元操作符。要创建一元操作符，只需省略 `leftarg`（左一元操作符）或 `rightarg`（右一元操作符）之一。`procedure` 子句和参数子句是 `CREATE OPERATOR` 中唯一必需的项。示例中出现的 `commutator` 子句，则是给查询优化器的一个可选提示。关于 `commutator` 以及其他优化器提示的更多细节，会在下一节说明。

35.13. 操作符优化信息

PostgreSQL 的操作符定义可以包含若干可选子句，用来告诉系统该操作符行为的一些有用信息。只要适用，就应当提供这些子句，
因为它们可以显著加快使用该操作符的查询执行速度。但如果你提供了这些信息，
就必须确保它们是正确的！错误地使用优化子句可能导致查询变慢、输出出现隐蔽错误，
或者引发其他糟糕后果。如果你拿不准，完全可以省略某个优化子句；唯一的后果，
只是查询可能会比本来需要的更慢。

未来版本的 PostgreSQL 可能会加入更多优化子句。这里描述的是 9.2.24 版本所理解的全部子句。

35.13.1. COMMUTATOR

如果给出 COMMUTATOR 子句，它指定一个与正在定义的操作符互为交换子的操作符。若对所有可能的输入值 x、y，都有 (x A y) 等于 (y B x)，则称操作符 A 是操作符 B 的交换子。注意，B 也同样是 A 的交换子。例如，对某种特定数据类型而言，< 和 > 通常互为交换子，而操作符 + 通常与其自身可交换。但操作符 - 通常并不与任何操作符可交换。

一个可交换操作符的左操作数类型，与其交换子的右操作数类型相同，反之亦然。因此，PostgreSQL 只需知道交换子操作符的名称，就可以查找出该交换子，而这也正是 COMMUTATOR 子句中所需提供的全部内容。

对于将用于索引和连接子句的操作符，提供交换子信息至关重要，因为这样查询优化器才能把这类子句“翻转”成不同计划类型所需的形式。例如，考虑这样一个查询，其 WHERE 子句形如 tab1.x = tab2.y，其中 tab1.x 和 tab2.y 都是某种用户定义类型，并假设 tab2.y 上建有索引。除非优化器能够确定如何把该子句翻转成 tab2.y = tab1.x，否则它就无法生成索引扫描，因为索引扫描机制要求传给它的操作符左侧必须是已建立索引的列。PostgreSQL 不会仅凭假设就认为这种转换有效；= 操作符的创建者必须通过为该操作符标记交换子信息，明确声明这种转换是有效的。

定义一个与自身可交换的操作符时，直接定义即可。但定义一对可交换操作符时，情况就稍微复杂一些：第一个要定义的操作符如何引用另一个尚未定义的操作符呢？这个问题有两种解决办法：

- 一种办法是在定义第一个操作符时省略 COMMUTATOR 子句，然后在第二个操作符的定义中提供该子句。由于 PostgreSQL 知道可交换操作符是成对出现的，因此它在看到第二个定义时，会自动回头补全第一个定义中缺少的 COMMUTATOR 子句。

- 另一种更直接的办法是在两个定义中都包含 COMMUTATOR 子句。当 PostgreSQL 处理第一个定义并发现 COMMUTATOR 引用了一个不存在的操作符时，系统会在系统目录中为该操作符建立一个占位项。这个占位项只有操作符名称、左右操作数类型和结果类型包含有效数据，因为这就是 PostgreSQL 此时能够推断出的全部信息。第一个操作符的目录项将链接到这个占位项。之后，当你定义第二个操作符时，系统会用第二个定义中的附加信息更新该占位项。如果在占位操作符补全之前尝试使用它，就只会得到一条错误消息。

35.13.2. NEGATOR

如果给出 NEGATOR 子句，它指定一个与正在定义的操作符互为求反器的操作符。若操作符 A 和 B 都返回布尔结果，并且对所有可能的输入 x、y，都有 (x A y) 等于 NOT (x B y)，则称 A 是 B 的求反器。注意，B 也同样是 A 的求反器。例如，对大多数数据类型而言，< 和 >= 构成一对求反器。一个操作符永远不可能合法地成为其自身的求反器。

与交换子不同，一对一元操作符完全可能合法地被标记为彼此的求反器；这意味着对所有 x，都有 (A x) 等于 NOT (B x)，或者对右一元操作符有等价的关系。

一个操作符的求反器必须与待定义操作符具有相同的左操作数类型和/或右操作数类型，因此与 COMMUTATOR 一样，在 NEGATOR 子句中只需给出操作符名称即可。

提供求反器对查询优化器很有帮助，因为它允许把 NOT (x = y) 这样的表达式简化为 x <> y。这种情况比你想象得更常见，因为 NOT 操作可能会作为其他重排的结果被插入进来。

可以使用上面解释的定义交换子对的相同方法，来定义成对的求反器操作符。

35.13.3. RESTRICT

如果给出 RESTRICT 子句，它指定该操作符的限制选择率估算函数。（注意，这里是函数名，而不是操作符名。）RESTRICT 子句只对返回 boolean 的二元操作符有意义。限制选择

率估算器的作用，是针对当前操作符和某个特定常量值，
猜测一张表中有多少比例的行会满足如下形式的 WHERE 子句条件：

```
column OP constant
```

这会帮助优化器大致了解具有这种形式的 WHERE 子句将淘汰多少行。（你可能会问：如果常量在左边会怎样？嗯，这正是 COMMUTATOR 的作用之一.....）

编写新的限制选择率估算函数远远超出了本章的范围，不过幸运的是，
对于你自己的很多操作符，通常都可以直接使用系统提供的某个标准估算器。
标准的限制选择率估算器如下：

eqsel 用于 =
neqsel 用于 <>
scalarltsel 用于 < 或 <=
scalargtsel 用于 > 或 >=

这些分类可能看起来有点奇怪，但仔细想想就会发现它们是有道理的。= 通常只会接受表中很小一部分行；<> 通常只会拒绝很小一部分行。< 接受的比例取决于给定常量落在该表列的值范围中的什么位置（恰好，这正是 ANALYZE 收集并提供给选择率估算器的信息）。对于相同的比较常量，<= 接受的比例会比 < 略大，但二者足够接近，不值得区分，尤其是无论如何我们很可能也只能做出粗略猜测。类似的说法也适用于 > 和 >=。

对于选择率非常高或非常低的操作符，即使它们实际上并不是真正的相等或不等比较，
你也常常可以勉强使用 eqsel 或 neqsel。例如，几何类型中的近似相等操作符就使用 eqsel，其依据是它们通常只会匹配表中很小一部分项。

对于那些能够以某种合理方式转换为数值标量、从而可进行范围比较的数据类型，
你可以使用 scalarltsel 和 scalargtsel。如果可能的话，请把该数据类型加入函数 convert_to_scalar()（位于 src/backend/utils/adt/selfuncs.c）所理解的范围中。（最终，这个函数应被通过 pg_type 系统目录某一列标识的、按数据类型划分的函数所取代；但这件事目前还没有发生。）如果不这样做，系统仍然可以工作，
但优化器的估算效果就不会像本可达到的那样好。

在 src/backend/utils/adt/geo_selfuncs.c 中，还为几何操作符提供了其他选择率估算函数：areasel、positionsel 和 contsel。截至目前，这些函数都还只是桩实现，但你也许仍会想使用它们（或者更好的是，改进它们）。

35.13.4. JOIN

如果给出 JOIN 子句，它指定该操作符的连接选择率估算函数。（注意，这里是函数名，而不是操作符名。）JOIN 子句只对返回 boolean 的二元操作符有意义。连接选择率估算器的作用，是针对当前操作符，猜测两张表中有多少比例的行对会满足如下形式的 WHERE 子句条件：

```
table1.column1 OP table2.column2
```

与 RESTRICT 子句一样，这会极大地帮助优化器判断在若干可能的连接顺序中，哪一种预计工作量最小。

与前面一样，本章不会尝试解释如何编写连接选择率估算函数，
而只是建议你在适用时使用某个标准估算器：

eqjoinsel 用于 =
neqjoinsel 用于 <>
scalarltjoinsel 用于 < 或 <=
scalargtjoinsel 用于 > 或 >=
areajoinsel 用于基于二维面积的比较
positionjoinsel 用于基于二维位置的比较
contjoinsel 用于基于二维包含关系的比较

35.13.5. HASHES

如果存在 HASHES 子句，它会告知系统：在基于该操作符的连接中，可以使用哈希连接方法。HASHES 只对返回 boolean 的二元操作符有意义；在实践中，该操作符还必须表示某种数据类型或某对数据类型上的相等关系。

哈希连接背后的假设是：只有当左值和右值被哈希到同一个哈希码时，
连接操作符才有可能返回 true。如果两个值被放进不同的哈希桶，连接过程就根本不会去比较它们，这实际上隐含假定连接操作符的结果必定为 false。因此，对于那些并不表示某种相等关系的操作符，指定 HASHES 永远没有意义。在大多数情况下，只有对两侧都接受同一数据类型的操作符支持哈希才是现实可行的。不过，
有时也可以为两种或更多数据类型设计兼容的哈希函数；也就是说，
尽管这些值的表示不同，但对于“相等”的值，函数仍会生成相同的哈希码。例如，
对不同位宽的整数做到这一点就相当简单。

若要被标记为 HASHES，该连接操作符必须出现在某个哈希索引操作符族中。创建操作符时并不会强制这一点，因为那时引用它的操作符族当然还不存在。
但如果没有这样的操作符族存在，运行时尝试在哈希连接中使用该操作符就会失败。
系统需要通过该操作符族来查找与该操作符输入数据类型相对应的哈希函数。当然，
在创建该操作符族之前，你还必须先创建合适的哈希函数。

准备哈希函数时应格外小心，因为它有一些依赖机器的方式可能会导致结果不正确。例如，
如果你的数据类型是某种结构体，其中可能包含无意义的填充位，
那就不能简单地把整个结构体传给 hash_any。（除非你编写了其他操作符和函数，以确保这些未使用位始终为零，而这正是推荐的策略。）再比如，在符合 IEEE 浮点标准的机器上，
负零和正零是不同的值（位模式不同），但它们被定义为比较相等。
如果某个浮点值可能包含负零，就需要采取额外步骤，确保它生成与正零相同的哈希值。

一个可参与哈希连接的操作符，必须拥有一个交换子（如果两侧操作数数据类型相同，
则为它自身；如果不同，则应是相关的相等操作符），
并且该交换子也必须出现在同一个操作符族中。否则，
在使用该操作符时可能会发生规划器错误。另外，对于支持多种数据类型的哈希操作符族，
最好（虽然并非严格必需）为每一种数据类型组合都提供相等操作符；
这样可以获得更好的优化效果。

> **注意**
>
> 作为可哈希连接操作符底层实现的函数，必须标记为 immutable 或 stable。如果它是 volatile，系统就永远不会尝试把该操作符用于哈希连接。

> **注意**
>
> 如果某个可哈希连接操作符的底层函数被标记为 strict，那么该函数还必须是完备的：也就是说，对于任意两个非空输入，它都应返回 true 或 false，而绝不能返回 null。如果不遵循这条规则，对 IN 操作进行哈希优化时可能会产生错误结果。（具体来说，IN 可能会在按标准本应返回 null 的地方返回 false；或者抛出一条错误，抱怨它没有为 null 结果做好准备。）

35.13.6. MERGES

如果存在 MERGES 子句，它会告知系统：在基于该操作符的连接中，可以使用归并连接方法。MERGES 只对返回 boolean 的二元操作符有意义；在实践中，该操作符还必须表示某种数据类型或某对数据类型上的相等关系。

归并连接的基本思想，是先把左表和右表分别排序，然后同步扫描它们。因此，两种数据类型都必须能够被完全排序，而连接操作符必须只能在那对值位于排序次序中“同一位置”时才成功。实际效果上，这意味着连接操作符必须表现得像相等比较一样。不过，只要两种不同的数据类型在逻辑上兼容，也完全可以对它们进行归并连接。例如，`smallint` 与 `integer` 之间的相等操作符就是可归并连接的。我们只需要能把两种数据类型都带入逻辑兼容排序序列的排序操作符即可。

若要被标记为 `MERGES`，该连接操作符必须作为相等成员出现在某个 `btree` 索引操作符族中。创建操作符时并不会强制这一点，因为那时引用它的操作符族当然还不存在。但除非能找到匹配的操作符族，否则该操作符实际上不会被用于归并连接。因此，`MERGES` 标记的作用，是向规划器提供一个提示，告诉它值得去查找匹配的操作符族。

一个可参与归并连接的操作符，必须拥有一个交换子（如果两侧操作数数据类型相同，则为它自身；如果不同，则应是相关的相等操作符），并且该交换子也必须出现在同一个操作符族中。否则，在使用该操作符时可能会发生规划器错误。另外，对于支持多种数据类型的 `btree` 操作符族，最好（虽然并非严格必需）为每一种数据类型组合都提供相等操作符；这样可以获得更好的优化效果。

> **注意**
>
> 作为可归并连接操作符底层实现的函数，必须标记为 immutable 或 stable。如果它是 volatile，系统就永远不会尝试把该操作符用于归并连接。

35.14. 索引扩展接口

到目前为止所描述的过程使我们能够定义新类型、新函数以及新操作符。然而，我们还不能在一种新数据类型的列上定义索引。为此，必须为该新数据类型定义一个操作符类。本节稍后将用一个示例说明这一概念：为 B-树索引方法定义一个新的操作符类，以便按绝对值升序存储和排序复数。

操作符类可以分组成操作符族，以展示语义兼容的类之间的关系。只涉及单一数据类型时，一个操作符类就已足够，因此我们先关注这种情况，然后再回到操作符族。

35.14.1. 索引方法和操作符类

`pg_am` 表为每个索引方法（内部称为访问方法）保存一行。对表进行常规访问的支持内置于 PostgreSQL 中，但所有索引方法都在 `pg_am` 中描述。可以定义所需的接口例程，然后在 `pg_am` 中创建一行，从而添加新的索引方法 – 但这超出了本章的范围（参见第 52 章）。

索引方法的例程并不直接知道它将处理哪些数据类型。相反，一个操作符类标识了索引方法在处理特定数据类型时需要使用的那组操作。之所以称为操作符类，是因为它指定的一项内容就是可与索引一起使用的 `WHERE` 子句操作符集合（也就是能被转换成索引扫描条件的操作符）。操作符类还可以指定索引方法内部操作所需的一些支持函数，但这些函数并不直接对应任何可与索引一起使用的 `WHERE` 子句操作符。

可以为同一种数据类型和索引方法定义多个操作符类。这样就能为一种数据类型定义多套索引语义。例如，一个 B-树索引要求为其处理的每一种数据类型定义一种排序顺序。对于复数数据类型，也许既需要一个按复数绝对值排序的 B-树操作符类，也需要另一个按实部排序的操作符类，等等。通常，其中一个操作符类会被视为最常用，并标记为该数据类型在该索引方法上的默认操作符类。

同一个操作符类名可以用于多个不同的索引方法（例如，B-树和哈希索引方法都有名为 `int4_ops` 的操作符类），但每一个这样的类都是独立实体，必须分别定义。

35.14.2. 索引方法策略

与操作符类关联的操作符通过“策略号”来标识，用以表示每个操作符在其操作符类上下文中的语义。例如，B-树对键施加了严格的从小到大的顺序，因此像“小于”和“大于等于”这样的操作符，对 B-树来说就很重要。由于 PostgreSQL 允许用户定义操作符，PostgreSQL 不能仅凭操作符名称（例如 < 或 >=）就判断它是哪一类比较。取而代之的是，索引方法定义了一组“策略”，可以把它们看成是广义的操作符。每个操作符类都会说明，对于某种特定数据类型和某种索引语义解释，每一种策略分别对应哪个实际操作符。

B-树索引方法定义了五种策略，如表 35.2所示。

表 35.2. B-树策略

操作	策略号
小于	1
小于等于	2
等于	3
大于等于	4
大于	5

哈希索引只支持等值比较，因此它们只使用一种策略，如表 35.3所示。

表 35.3. 哈希策略

操作	策略号
等于	1

GiST 索引更加灵活：它们根本没有固定的策略集合。相反，每个特定 GiST 操作符类中负责“一致性”检查的支持例程会按自己的方式解释策略号。举例来说，一些内置的 GiST 索引操作符类会为二维几何对象建立索引，并提供“R 树”策略，如表 35.4所示。其中四个是真正的二维测试（重叠、相同、包含、被包含），四个只考虑 X 方向，另外四个则在 Y 方向上提供相同测试。

表 35.4. GiST 二维“R 树”策略

操作	策略号
严格位于左侧	1
不延伸到右侧	2
重叠	3
不延伸到左侧	4
严格位于右侧	5
相同	6
包含	7
被包含	8
不延伸到上方	9
严格位于下方	10
严格位于上方	11
不延伸到下方	12

SP-GiST 索引在灵活性方面与 GiST 索引类似：它们也没有固定的策略集合。相反，每个操作符类的支持例程会根据该操作符类的定义解释策略号。举例来说，内置点操作符类所使用的策略号如表 35.5所示。

表 35.5. SP-GiST 点策略

操作	策略号
严格位于左侧	1
严格位于右侧	5
相同	6
被包含	8
严格位于下方	10
严格位于上方	11

GIN 索引与 GiST 和 SP-GiST 索引类似，也没有固定的策略集合。相反，每个操作符类的支持例程会根据该操作符类的定义解释策略号。举例来说，内置数组操作符类所使用的策略号如表 35.6所示。

表 35.6. GIN 数组策略

操作	策略号
重叠	1
包含	2
被包含	3
等于	4

注意，上面列出的所有操作符都返回布尔值。实际上，所有被定义为索引方法搜索操作符的操作符都必须返回 `boolean`，因为要与索引配合使用，它们必须出现在 `WHERE` 子句的顶层。（某些索引访问方法还支持排序操作符，这类操作符通常不返回布尔值；该特性见第 35.14.7 节。）

35.14.3. 索引方法支持例程

仅靠策略信息通常不足以让系统知道如何使用索引。实际上，索引方法还需要额外的支持例程才能工作。例如，B-树索引方法必须能够比较两个键，并判断其中一个是大于、等于还是小于另一个。类似地，哈希索引方法必须能够为键值计算哈希码。这些操作并不对应 SQL 命令条件中使用的操作符；它们是索引方法内部使用的管理例程。

与策略一样，操作符类会标识对于给定的数据类型和语义解释，应由哪些具体函数承担这些角色。索引方法定义它需要的函数集合，而操作符类则会通过为函数分配由索引方法规定的“支持函数号”来标识正确的函数。

如表 35.7所示，B-树要求一个支持函数，并允许操作符类作者按需再提供一个支持函数。

表 35.7. B-树支持函数

函数	支持号
比较两个键，并返回一个小于零、等于零或大于零的整数，用以表示第一个键是小于、等于还是大于第二个键	1
返回可从 C 调用的排序支持函数的地址，详见 `utils/sortsupport.h`（可选）	2

哈希索引要求一个支持函数，如表 35.8所示。

表 35.8. 哈希支持函数

函数	支持号
计算一个键的哈希值	1

GiST 索引需要七个支持函数，外加一个可选的第八个，如表 35.9所示。（详见第 53 章。）

表 35.9. GiST 支持函数

函数	描述	支持号
consistent	确定键是否满足查询条件	1
union	计算一组键的并集	2
compress	计算将被索引的键或值的压缩表示	3
decompress	计算压缩键的解压表示	4
penalty	计算把新键插入具有给定子树键的子树时的罚值	5
picksplit	确定页面中的哪些项要移到新页面，并计算结果页面的并集键	6
equal	比较两个键，并在它们相等时返回真	7
distance	确定键到查询值的距离（可选）	8

如表 35.10所示，SP-GiST 索引要求五个支持函数。（详见第 54 章。）

表 35.10. SP-GiST 支持函数

函数	描述	支持号
config	提供有关该操作符类的基本信息	1
choose	确定如何把一个新值插入内部元组	2
picksplit	确定如何划分一组值	3
inner_consistent	确定对某个查询需要搜索哪些子分区	4
leaf_consistent	确定键是否满足查询条件	5

如表 35.11所示，GIN 索引需要四个支持函数，外加一个可选的第五个（详见第 55 章）。

表 35.11. GIN 支持函数

函数	描述	支持号
compare	比较两个键，并返回一个小于零、等于零或大于零的整数，用以表示第一个键是小于、等于还是大于第二个键	1
extractValue	从待索引值中提取键	2
extractQuery	从查询条件中提取键	3
consistent	确定值是否匹配查询条件	4
comparePartial	比较查询中的部分键与索引中的键，并返回一个小于零、等于零或大于零的整数，用以指示 GIN 应忽略该索引项、将该项视为匹配，还是停止索引扫描（可选）	5

与搜索操作符不同，支持函数返回的是特定索引方法所期望的数据类型；例如，对 B-树的比较函数来说，就是一个有符号整数。
每个支持函数的参数个数和类型也同样取决于索引方法。对于 B-树和哈希，比较支持函数和哈希支持函数接受的输入数据类型，与操作符类中包含的操作符相同；但对大多数 GiST、SP-GiST 和 GIN 支持函数来说并非如此。

35.14.4. 示例

现在我们已经了解了这些基本思想，下面给出先前承诺的创建新操作符类示例。（这个可运行示例位于源码发布包中的src/tutorial/complex.c和src/tutorial/complex.sql。）该操作符类封装了一组按绝对值顺序对复数排序的操作符，因此我们把它命名为complex_abs_ops。首先，我们需要一组操作符。定义操作符的过程已经在第 35.12 节中讨论过。对于 B-树上的操作符类，我们需要如下操作符：

- 绝对值小于（策略 1）
- 绝对值小于等于（策略 2）
- 绝对值等于（策略 3）
- 绝对值大于等于（策略 4）
- 绝对值大于（策略 5）

定义一组相关比较操作符时，最不容易出错的方式是先编写 B-树比较支持函数，再把其他函数写成围绕该支持函数的一行包装器函数。这样可以降低在边界情况下得到不一致结果的概率。按照这种方法，我们首先编写：

```
#define Mag(c)  ((c)->x*(c)->x + (c)->y*(c)->y)

static int
complex_abs_cmp_internal(Complex *a, Complex *b)
{
    double      amag = Mag(a),
                bmag = Mag(b);

    if (amag < bmag)
        return -1;
    if (amag > bmag)
        return 1;
    return 0;
}
```

现在，小于函数如下所示：

```
PG_FUNCTION_INFO_V1(complex_abs_lt);

Datum
complex_abs_lt(PG_FUNCTION_ARGS)
{
    Complex    *a = (Complex *) PG_GETARG_POINTER(0);
    Complex    *b = (Complex *) PG_GETARG_POINTER(1);

    PG_RETURN_BOOL(complex_abs_cmp_internal(a, b) < 0);
}
```

其他四个函数的区别只在于它们如何比较内部函数的结果与 0。

接下来，在 SQL 中声明这些函数，以及基于这些函数的操作符：

```
CREATE FUNCTION complex_abs_lt(complex, complex) RETURNS bool
    AS 'filename', 'complex_abs_lt'
    LANGUAGE C IMMUTABLE STRICT;

CREATE OPERATOR < (
   leftarg = complex, rightarg = complex, procedure = complex_abs_
lt,
   commutator = > , negator = >= ,
```

```
   restrict = scalarltsel, join = scalarltjoinsel
);
```

必须指定正确的交换子和求反器操作符，以及合适的限制选择率与连接选择率函数，
否则优化器无法有效使用索引。注意，小于、
等于和大于这几种情况应使用不同的选择率函数。

这里还有几点值得注意：

- 只能有一个名为 `=` 且两个操作数都为 `complex` 类型的操作符。在这个例子里，我们并没有任何其他 `=` 操作符可用于 `complex`；但如果我们是在构造一种实际使用的数据类型，可能会希望 `=` 表示复数的普通相等，而不是绝对值相等。在那种情况下，我们就需要为 `complex_abs_eq` 选用其他操作符名。
- 尽管 PostgreSQL 能处理 SQL 名称相同但参数数据类型不同的函数，C 却只能处理给定名称的一个全局函数。因此，我们不应该把 C 函数简单命名成 `abs_eq` 之类。通常，在 C 函数名中包含数据类型名称是个好习惯，这样就不会与其他数据类型的函数发生冲突。
- 我们原本也可以把该函数的 SQL 名称取为 `abs_eq`，并依靠 PostgreSQL 通过参数数据类型把它与其他同名 SQL 函数区分开。为了让示例保持简单，这里我们让 C 层和 SQL 层的函数使用相同的名称。

下一步是注册 B-树要求的支持例程。实现该例程的 C 示例代码与操作符函数位于同一个文件中。该函数的声明如下：

```
CREATE FUNCTION complex_abs_cmp(complex, complex)
    RETURNS integer
    AS 'filename'
    LANGUAGE C IMMUTABLE STRICT;
```

现在我们已经有了所需的操作符和支持例程，就可以最终创建操作符类：

```
CREATE OPERATOR CLASS complex_abs_ops
    DEFAULT FOR TYPE complex USING btree AS
        OPERATOR        1       < ,
        OPERATOR        2       <= ,
        OPERATOR        3       = ,
        OPERATOR        4       >= ,
        OPERATOR        5       > ,
        FUNCTION        1       complex_abs_cmp(complex, complex);
```

这样就完成了！现在应该可以在 `complex` 列上创建并使用 B-树索引。

我们本来也可以把操作符项写得更详细一些，例如：

```
        OPERATOR        1       < (complex, complex) ,
```

但是当操作符接受的数据类型与该操作符类所服务的数据类型相同时，就没有必要这样写。

上述示例假定你希望把这个新操作符类设为 `complex` 数据类型的默认 B-树操作符类。如果不是这样，只需省去 `DEFAULT` 这个词。

35.14.5. 操作符类和操作符族

到目前为止，我们一直隐含地假定一个操作符类只处理一种数据类型。
虽然某个特定的索引列当然只能有一种数据类型，
但对把被索引列与另一种数据类型的值进行比较的操作建立索引往往也很有用。此外，
如果某个与操作符类相关的跨数据类型操作符有用，
通常另一种数据类型本身也会有一个相关的操作符类。
把相关类之间的联系显式表示出来会很有帮助，因为这有助于规划器优化 SQL 查询（尤其是对 B-树操作符类而言，因为规划器中包含大量有关如何使用它们的知识）。

为了满足这些需求，PostgreSQL使用操作符族这一概念。
一个操作符族包含一个或多个操作符类，还可以包含属于整个族、
但不属于族中任何单一类的可索引操作符及其相应的支持函数。
我们称这样的操作符和函数在该族中是“松散”的，而不是绑定在某个特定类中。通常，
每个操作符类只包含单一数据类型的操作符，
而跨数据类型操作符则作为操作符族中的松散成员存在。

一个操作符族中的所有操作符和函数都必须具有兼容的语义，
而兼容性的要求由索引方法设定。因此，你也许会疑惑，
为什么还要把该族的某些子集单独划成操作符类；事实上，对很多用途而言，
类的划分并不重要，真正有意义的分组只有操作符族。之所以定义操作符类，
是因为它们规定了支持特定索引所需的操作符族内容。如果某个索引使用了某个操作符类，
那么在不删除该索引的情况下就不能删除该操作符类 – 但操作符族中的其他部分，也就是其他操作符类和松散操作符，则可以被删除。因此，
一个操作符类应当只包含在特定数据类型上支持索引所合理需要的最小操作符和函数集合，
而那些相关但非必需的操作符，则可以作为操作符族的松散成员加入。

例如，PostgreSQL内置了 B-树操作符族`integer_ops`，其中包含操作符类`int8_ops`，`int4_ops`和`int2_ops`，它们分别用于以下列类型上的索引：`bigint`（`int8`），`integer`（`int4`）和`smallint`（`int2`）。该操作符族还包含跨数据类型的比较操作符，
允许对这些类型中的任意两个进行比较，
因此可以使用一种类型的比较值搜索另一种类型上的索引。
可以用以下定义复制这个操作符族：

```
CREATE OPERATOR FAMILY integer_ops USING btree;

CREATE OPERATOR CLASS int8_ops
DEFAULT FOR TYPE int8 USING btree FAMILY integer_ops AS
  -- standard int8 comparisons
  OPERATOR 1 < ,
  OPERATOR 2 <= ,
  OPERATOR 3 = ,
  OPERATOR 4 >= ,
  OPERATOR 5 > ,
  FUNCTION 1 btint8cmp(int8, int8) ,
  FUNCTION 2 btint8sortsupport(internal) ;

CREATE OPERATOR CLASS int4_ops
DEFAULT FOR TYPE int4 USING btree FAMILY integer_ops AS
  -- standard int4 comparisons
  OPERATOR 1 < ,
  OPERATOR 2 <= ,
  OPERATOR 3 = ,
  OPERATOR 4 >= ,
  OPERATOR 5 > ,
  FUNCTION 1 btint4cmp(int4, int4) ,
  FUNCTION 2 btint4sortsupport(internal) ;

CREATE OPERATOR CLASS int2_ops
DEFAULT FOR TYPE int2 USING btree FAMILY integer_ops AS
  -- standard int2 comparisons
  OPERATOR 1 < ,
  OPERATOR 2 <= ,
  OPERATOR 3 = ,
  OPERATOR 4 >= ,
  OPERATOR 5 > ,
  FUNCTION 1 btint2cmp(int2, int2) ,
  FUNCTION 2 btint2sortsupport(internal) ;
```

```
ALTER OPERATOR FAMILY integer_ops USING btree ADD
  -- cross-type comparisons int8 vs int2
  OPERATOR 1 < (int8, int2) ,
  OPERATOR 2 <= (int8, int2) ,
  OPERATOR 3 = (int8, int2) ,
  OPERATOR 4 >= (int8, int2) ,
  OPERATOR 5 > (int8, int2) ,
  FUNCTION 1 btint82cmp(int8, int2) ,

  -- cross-type comparisons int8 vs int4
  OPERATOR 1 < (int8, int4) ,
  OPERATOR 2 <= (int8, int4) ,
  OPERATOR 3 = (int8, int4) ,
  OPERATOR 4 >= (int8, int4) ,
  OPERATOR 5 > (int8, int4) ,
  FUNCTION 1 btint84cmp(int8, int4) ,

  -- cross-type comparisons int4 vs int2
  OPERATOR 1 < (int4, int2) ,
  OPERATOR 2 <= (int4, int2) ,
  OPERATOR 3 = (int4, int2) ,
  OPERATOR 4 >= (int4, int2) ,
  OPERATOR 5 > (int4, int2) ,
  FUNCTION 1 btint42cmp(int4, int2) ,

  -- cross-type comparisons int4 vs int8
  OPERATOR 1 < (int4, int8) ,
  OPERATOR 2 <= (int4, int8) ,
  OPERATOR 3 = (int4, int8) ,
  OPERATOR 4 >= (int4, int8) ,
  OPERATOR 5 > (int4, int8) ,
  FUNCTION 1 btint48cmp(int4, int8) ,

  -- cross-type comparisons int2 vs int8
  OPERATOR 1 < (int2, int8) ,
  OPERATOR 2 <= (int2, int8) ,
  OPERATOR 3 = (int2, int8) ,
  OPERATOR 4 >= (int2, int8) ,
  OPERATOR 5 > (int2, int8) ,
  FUNCTION 1 btint28cmp(int2, int8) ,

  -- cross-type comparisons int2 vs int4
  OPERATOR 1 < (int2, int4) ,
  OPERATOR 2 <= (int2, int4) ,
  OPERATOR 3 = (int2, int4) ,
  OPERATOR 4 >= (int2, int4) ,
  OPERATOR 5 > (int2, int4) ,
  FUNCTION 1 btint24cmp(int2, int4) ;
```

注意，此定义“重载”了操作符策略编号和支持函数编号：每个编号在操作符族中出现多次。只要同一编号的每个实例具有不同的输入数据类型，这就是允许的。
两个输入类型都等于某个操作符类输入类型的实例，
是该操作符类的主要操作符和支持函数，通常应声明为操作符类的一部分，
而不是操作符族的松散成员。

在一个 B-树操作符族中，该族中的所有操作符都必须以兼容的方式排序，也就是说，传递律必须适用于该族支持的所有数据类型：“如果 A = B 且 B = C，则 A = C”，以及“如果 A < B 且 B < C，则 A

< C”。此外，操作符族中所表示的类型之间的隐式或二进制强制类型转换不得改变相关的排序顺序。对族中的每一个操作符，都必须有一个具有相同两个输入数据类型的支持函数。
建议让操作符族保持完整，也就是说，对每一种数据类型组合都应包含全部操作符。
每个操作符类只应包含其数据类型对应的非跨数据类型操作符和支持函数。

要构建一个多数据类型的哈希操作符族，
必须为该族支持的每一种数据类型创建相互兼容的哈希支持函数。
这里的兼容性是指：对任意两个被该族中的等值操作符视为相等的值，
这些函数都保证返回相同的哈希码，即使这两个值属于不同类型也是如此。
当这些类型具有不同的物理表示时，这通常难以实现，但在某些情况下可以做到。此外，
将该操作符族中一种数据类型的值通过隐式或二进制强制转换转为该族中另一种数据类型
时，不得改变所计算出的哈希值。注意，每种数据类型只有一个支持函数，
而不是每个等值操作符一个。建议让操作符族保持完整，也就是说，
对每一种数据类型组合都提供一个等值操作符。
每个操作符类只应包含其数据类型对应的非跨数据类型等值操作符和支持函数。

GiST、SP-GiST 和 GIN 索引没有任何显式的跨数据类型操作概念。它们所支持的操作符集合，就是给定操作符类的主要支持函数所能处理的那些操作符。

> **注意**
>
> 在 PostgreSQL 8.3 之前，并没有操作符族这一概念，因此任何打算与索引一起使用的跨数据类型操作符都必须直接绑定到该索引的操作符类中。虽然这种做法仍然有效，
> 但已被弃用，因为它会使索引的依赖关系过于宽泛，
> 而且当两种数据类型都在同一操作符族中拥有操作符时，
> 规划器能更有效地处理跨数据类型比较。

35.14.6. 系统对操作符类的依赖

PostgreSQL利用操作符类来从多方面推断操作符的属性，
而不仅仅是判断它们能否用于索引。因此，即便你并不打算为自己的数据类型列建立索引，
也可能会想创建操作符类。

特别地，`ORDER BY`和`DISTINCT`等 SQL 特性要求对值的比较和排序。为了在用户定义的数据类型上实现这些特性，PostgreSQL会为数据类型查找默认 B-树操作符类。这个操作符类的“相等”成员定义了用于`GROUP BY`和`DISTINCT`的值的等值概念，而该操作符类施加的排序顺序定义了默认的`ORDER BY`顺序。

用户定义类型的数组比较也依赖于该类型默认 B-树操作符类所定义的语义。

如果一种数据类型没有默认的 B-树操作符类，系统就会查找默认的哈希操作符类。但由于这类操作符类只提供等值语义，因此在实践中它只足以支持数组相等比较。

如果某种数据类型没有默认操作符类，而你又试图将这些 SQL 特性用于该数据类型，就会得到类似“无法识别排序操作符”这样的错误。

> **注意**
>
> 在版本 7.4 以前的PostgreSQL中，排序和分组操作将隐式地使用名为=、<以及>的操作符。
> 新的依赖于默认操作符类的行为避免了对具有特定名字的操作符行为作出任何假设。

另一个重要点在于，出现在哈希操作符族中的操作符，都是哈希连接、
哈希聚合以及相关优化的候选对象。这里哈希操作符族至关重要，
因为它标识了应当使用的哈希函数。

35.14.7. 排序操作符

有些索引访问方法（目前只有 GiST）支持排序操作符的概念。我们到目前为止讨论的是搜索操作符。对于搜索操作符，可以搜索索引以找出满足以下条件的所有行：`WHERE indexed_column operator constant`。注意，不保证匹配行的返回顺序。排序操作符则不限制可以返回的行集合，而是确定这些行的顺序。对于排序操作符，可以扫描索引，按以下表达式表示的顺序返回行：`ORDER BY indexed_column operator constant`。这样定义排序操作符，是因为当操作符用于度量距离时，它可以支持最近邻搜索。例如，如下查询：

```
SELECT * FROM places ORDER BY location <-> point '(101,456)' LIMIT
 10;
```

可以找到距离指定目标点最近的十个地点。location 列上的 GiST 索引能够高效完成此操作，因为`<->`是排序操作符。

搜索操作符必须返回布尔结果，而排序操作符通常返回其他类型的结果，例如用于表示距离的 float 或 numeric。这种类型通常不同于被索引的数据类型。为了避免对不同数据类型行为作硬编码假设，在定义排序操作符时，必须指定一个 B-树操作符族，用来说明结果数据类型的排序顺序。正如上一节所述，B-树操作符族定义了 PostgreSQL 的顺序概念，因此这是一种自然的表示方式。由于点的 `<->` 操作符返回 `float8`，因此可以在创建操作符类时这样指定它：

```
OPERATOR 15    <-> (point, point) FOR ORDER BY float_ops
```

其中 `float_ops` 是包含针对 `float8` 的操作的内置操作符族。这个声明表明，该索引能够按 `<->` 操作符值递增的顺序返回行。

35.14.8. 操作符类的特殊特性

还有两个操作符类的特殊特性我们尚未讨论，主要是因为它们对最常用的索引方法没有用处。

通常，把一个操作符声明为某个操作符类（或操作符族）的成员，意味着该索引方法能够利用该操作符准确检索出满足 `WHERE` 条件的那组行。例如：

```
SELECT * FROM table WHERE integer_column < 4;
```

这个查询可以由整数列上的 B-树索引精确满足。但也有一些情况下，索引只能作为匹配行的不精确指引。例如，如果某个 GiST 索引只存储几何对象的边界框，那么它就无法精确满足测试非矩形对象（如多边形）之间重叠的 `WHERE` 条件。不过，我们仍然可以利用该索引找出边界框与目标对象边界框重叠的对象，然后只对索引找到的那些对象执行精确的重叠测试。如果适用于这种场景，就称该索引对这个操作符是“有损的”。有损索引搜索的实现方式是：当某一行可能满足、也可能不满足查询条件时，由索引方法返回一个recheck标志。随后，核心系统会在取回的行上重新测试原始查询条件，以判断它是否应当作为合法匹配返回。只要索引能保证返回所有必需的行，外加可能存在的一些额外行，这种方法就是有效的，因为这些额外行可以通过执行原始操作符调用来剔除。支持有损搜索的索引方法（目前有 GiST、SP-GiST 和 GIN）允许个别操作符类的支持函数设置 recheck 标志，因此这本质上也是一种操作符类特性。

再次考虑只在索引中存储多边形等复杂对象的包围盒的情况。此时，在索引条目中存储整个多边形没有多少价值，不如只存储一个更简单的对象，其类型为`box`。这种情况由`STORAGE`选项表达，该选项位于`CREATE OPERATOR CLASS`：可以写成如下形式：

```
CREATE OPERATOR CLASS polygon_ops
    DEFAULT FOR TYPE polygon USING gist AS
        ...
```

```
        STORAGE box;
```

目前，只有 GiST 和 GIN 索引方法支持与列数据类型不同的STORAGE类型。GiST 的compress和decompress支持函数在使用STORAGE时必须处理数据类型转换。在 GIN 中，STORAGE类型标识“键”值的类型，通常与被索引列的类型不同。例如，整数数组列的操作符类可以只使用整数作为键。GIN 的extractValue和extractQuery支持函数负责从被索引值中提取键。

35.15. 将相关对象打包成扩展

一个有用的 PostgreSQL 扩展通常包含多个 SQL 对象；例如，一种新的数据类型将需要新的函数、新的操作符，并且很可能还需要新的索引操作符类。
把所有这些对象收集到一个单独的包中，有助于简化数据库管理。PostgreSQL 将这样的包称为扩展。要定义一个扩展，至少需要一个脚本文件，其中包含用于创建扩展对象的 SQL 命令，以及一个控制文件，用来指定扩展本身的一些基本属性。如果扩展包含 C 代码，通常还会有一个共享库文件，C 代码已被构建到其中。一旦准备好这些文件，只需执行一个简单的CREATE EXTENSION命令，就能把这些对象装入数据库。

使用扩展而不是仅仅运行 SQL 脚本把一堆“松散”对象装入数据库，最主要的优点在于 PostgreSQL 能够理解这些对象是属于同一个扩展的。你可以用一条DROP EXTENSION命令删除全部对象（无需维护单独的“卸载”脚本）。更有用的是，pg_dump 知道自己不应转储扩展的各个成员对象 – 它只会在转储中包含一条 CREATE EXTENSION 命令。这极大地简化了迁移到扩展新版本的过程，即使新版本包含比旧版本更多或不同的对象也是如此。不过要注意，在把这样的转储装载到新数据库时，
必须能够访问该扩展的控制文件、脚本文件以及其他相关文件。

PostgreSQL 不允许你删除扩展中包含的单个对象，除非删除整个扩展。另外，虽然你可以修改扩展成员对象的定义（例如对函数使用 CREATE OR REPLACE FUNCTION），但要记住，被修改后的定义不会被 pg_dump 转储。这样的修改通常只有在你同时在扩展脚本文件中做出同样修改时才有意义。（不过，对于包含配置数据的表有特殊规定；见见下文。）

扩展机制还提供了用于打包修改脚本的支持，以便调整扩展中所含 SQL 对象的定义。例如，如果扩展 1.1 版相比 1.0 版增加了一个函数，并修改了另一个函数的函数体，那么扩展作者可以提供一个更新脚本来只完成这两项修改。随后就可以使用 ALTER EXTENSION UPDATE 命令应用这些修改，并跟踪在某个数据库中实际安装的是该扩展的哪个版本。

哪些 SQL 对象种类可以成为扩展成员，见ALTER EXTENSION的说明。特别是，数据库集簇范围内的对象，如数据库、角色和表空间，不能成为扩展成员，
因为扩展只在单个数据库内可见。（尽管并不禁止扩展脚本创建这类对象，但如果这样做，
它们不会作为扩展的一部分受到跟踪。）还要注意，虽然表可以成为扩展成员，
但其附属对象（如索引）并不直接被视为扩展成员。另一个重要点是，模式可以属于扩展，
但反过来不成立：扩展本身只有一个非限定名，并不“位于”任何模式中。不过，
扩展的成员对象会在其对象类型适用的情况下属于某个模式。
扩展是否应当拥有其成员对象所在的模式，则要视具体情况而定。

35.15.1. 扩展文件

CREATE EXTENSION命令依赖于每个扩展都有一个控制文件，其名称必须是扩展名加上 .control 后缀，并放在安装目录的 SHAREDIR/extension 目录中。还必须至少有一个 SQL 脚本文件，其名称遵循 *extension*--*version*.sql 的模式（例如，扩展 foo 的 1.0 版本使用 foo--1.0.sql）。默认情况下，脚本文件也放在 SHAREDIR/extension 目录中，但控制文件可以为脚本文件指定其他目录。

扩展控制文件的格式与 postgresql.conf 文件相同，即由一组 *parameter_name* = *value* 赋值组成，每行一条。允许空行和以 # 引入的注释。任何不是单个单词或数字的值都要记得加引号。

控制文件可以设置下列参数：

directory (string)

包含扩展 SQL 脚本文件的目录。除非给出的是绝对路径，否则该名称相对于安装的 SHAREDIR 目录。默认行为等价于指定 directory = 'extension'。

default_version (string)

扩展的默认版本（即在 CREATE EXTENSION 中未指定版本时将安装的版本）。虽然这个参数可以省略，但那样一来，如果没有给出 VERSION 选项，CREATE EXTENSION 就会失败，因此一般不希望这样做。

comment (string)

关于扩展的注释（任意字符串）。另外，也可以通过脚本文件中的 COMMENT命令来设置该注释。

encoding (string)

脚本文件所使用的字符集编码。如果脚本文件包含任何非 ASCII 字符，就应指定这个参数。否则将假定这些文件使用数据库编码。

module_pathname (string)

该参数的值会替换脚本文件中每次出现的 MODULE_PATHNAME。如果未设置该参数，则不会进行替换。通常会把它设置为 $libdir/*shared_library_name*，然后在 C 语言函数的 CREATE FUNCTION 命令中使用 MODULE_PATHNAME，这样脚本文件就无需把共享库的名字硬编码进去。

requires (string)

本扩展所依赖的其他扩展名称列表，例如 requires = 'foo, bar'。这些被依赖的扩展必须先安装好，本扩展才能安装。

superuser (boolean)

如果此参数为 true（默认值），则只有超级用户可以创建扩展或将其更新到新版本。如果设为 false，则只需具有执行安装或更新脚本中命令所需的权限。

relocatable (boolean)

如果一个扩展在初次创建之后仍然可以把其包含的对象移动到不同的模式中，那么它就是可重定位的。默认值为 false，也就是该扩展不可重定位。详见下文。

schema (string)

该参数只能为不可重定位扩展设置。它强制扩展装载到指定名称的模式中，而不能装载到其他模式。详见下文。

除主控制文件 *extension*.control 外，扩展还可以有按如下样式命名的次级控制文件：*extension*--*version*.control。如果提供了这些文件，它们必须位于脚本文件目录中。次级控制文件遵循与主控制文件相同的格式。在安装或更新到该扩展的相应版本时，次级控制文件中设置的任何参数都会覆盖主控制文件中的设置。不过，directory 和 default_version 这两个参数不能在次级控制文件中设置。

扩展的 SQL 脚本文件可以包含任何 SQL 命令，但事务控制命令（BEGIN、COMMIT 等）和无法在事务块内执行的命令（例如 VACUUM）除外。这是因为脚本文件会被隐式地放在事务块中执行。

扩展的 SQL 脚本文件也可以包含以 \echo 开头的行，扩展机制会忽略这些行（将其视为注释）。这一约定通常用于在脚本文件被直接交给 psql 而不是通过 CREATE EXTENSION 装载时抛出错误（见下文中的示例脚本）。如果没有这种机制，用户可能会意外地把扩展内容作为“松散”对象装载，而不是作为一个扩展来装载，这种状态恢复起来会有些麻烦。

虽然脚本文件可以包含指定编码允许的任意字符，但控制文件应只包含纯 ASCII 字符，因为 PostgreSQL 无法知道控制文件采用的是什么编码。在实践中，只有当你想在扩展注释中使用非 ASCII 字符时这才会成为问题。对此推荐的做法是不要使用控制文件中的 comment 参数，而是在脚本文件中使用 COMMENT ON EXTENSION 来设置注释。

35.15.2. 扩展的可重定位性

用户经常希望把扩展中的对象装载到与扩展作者原先设想不同的模式中。
对于这种可重定位性，支持三个级别：

- 完全可重定位的扩展可以在任何时候移动到另一个模式中，
  即使它已经被装载到数据库之后也是如此。这通过 ALTER EXTENSION SET SCHEMA 命令完成，该命令会自动把所有成员对象重命名到新模式中。通常，只有当扩展对其任何对象所在模式都没有内部假设时，才有可能做到这一点。此外，
  扩展的对象一开始必须全部位于同一个模式中（不属于任何模式的对象，如过程语言，
  不算在内）。要把一个扩展标记为完全可重定位，只需在其控制文件中设置 relocatable = true。

- 扩展可能在安装期间可重定位，但安装之后不可重定位。
  如果扩展脚本文件需要显式引用目标模式，例如为 SQL 函数设置 search_path 属性时，通常就是这种情况。对于这样的扩展，应在控制文件中设置 relocatable = false，并在脚本文件中使用 @extschema@ 来引用目标模式。在脚本执行前，该字符串的每次出现都会被替换为实际目标模式的名称。用户可以通过 CREATE EXTENSION 的 SCHEMA 选项设置目标模式。

- 如果扩展完全不支持重定位，应在控制文件中设置 relocatable = false，并把 schema 设置为预定目标模式的名称。这样将阻止使用 CREATE EXTENSION 的 SCHEMA 选项，除非它指定的正是控制文件中命名的那个模式。如果扩展对模式名称有无法通过 @extschema@ 替换解决的内部假设，通常就需要采用这种方式。在这种情况下，@extschema@ 替换机制仍然可用，只是由于模式名由控制文件决定，其用途比较有限。

在所有情况下，脚本文件执行时，其初始search_path都会指向目标模式；也就是说，
CREATE EXTENSION 做的事情等价于：

```
SET LOCAL search_path TO @extschema@;
```

这使得脚本文件创建的对象能够进入目标模式。脚本文件当然也可以修改 search_path，但通常不建议这样做。CREATE EXTENSION 完成后，search_path 会恢复为先前的设置。

目标模式由控制文件中的 schema 参数决定（如果给出了该参数）；否则由 CREATE EXTENSION 的 SCHEMA 选项决定（如果给出了该选项）；否则由当前默认的对象创建模式决定（也就是调用者的 search_path 中第一个模式）。当使用控制文件中的 schema 参数时，如果目标模式尚不存在，则会自动创建；而在另外两种情况下，目标模式必须已经存在。

如果控制文件的 requires 中列出了任何前置扩展，它们的目标模式会追加到 search_path 的初始设置中。这样新扩展的脚本文件就可以看到这些扩展的对象。

虽然不可重定位扩展可以包含分布在多个模式中的对象，
但通常仍然希望把所有供外部使用的对象放在单个模式中，
这个模式会被视为该扩展的目标模式。这种安排与创建依赖扩展时 search_path 的默认设置配合起来会比较方便。

35.15.3. 扩展配置表

有些扩展包含配置表，其中存放的数据可能会在扩展安装后被用户新增或修改。通常，
如果表属于某个扩展，那么 pg_dump 既不会转储该表的定义，也不会转储其内容。但这种

行为对于配置表来说并不理想；用户所做的数据更改必须包含在转储中，
否则在转储并恢复之后，扩展的行为就会发生变化。

为了解决这个问题，扩展脚本文件可以把它创建的某个表标记为配置表，这样 pg_dump 就会把该表的内容（而不是定义）包含到转储中。做法是在创建表之后调用函数 `pg_extension_config_dump(regclass, text)`，例如：

```
CREATE TABLE my_config (key text, value text);
SELECT pg_catalog.pg_extension_config_dump('my_config', '');
```

可以用这种方式标记任意数量的表。

当 `pg_extension_config_dump` 的第二个参数是空字符串时，pg_dump 会转储该表的全部内容。通常只有当该表在扩展脚本创建时最初为空时，这样做才是正确的。
如果表中混有初始数据和用户提供的数据，那么 `pg_extension_config_dump` 的第二个参数就提供了一个用于选择要转储数据的 `WHERE` 条件。例如，你可以这样做：

```
CREATE TABLE my_config (key text, value text, standard_entry
 boolean);

SELECT pg_catalog.pg_extension_config_dump('my_config', 'WHERE NOT
 standard_entry');
```

然后确保只有扩展脚本创建的那些行，其 `standard_entry` 才为真。

更复杂的情况，例如用户可能会修改最初提供的数据行，
可以通过在配置表上创建触发器来处理，以确保被修改的行被正确标记。

你可以再次调用 `pg_extension_config_dump` 来修改与配置表关联的过滤条件。（这通常在扩展更新脚本中很有用。）要把某个表标记为不再是配置表，唯一的方法是通过 `ALTER EXTENSION ... DROP TABLE` 将它与扩展解除关联。

注意，这些表之间的外键关系会决定 pg_dump 转储它们的顺序。具体来说，pg_dump 会尝试先转储被引用表，再转储引用表。由于外键关系是在 CREATE EXTENSION 时建立的（那时数据尚未装入这些表中），因此不支持环状依赖。若存在环状依赖，数据仍会被转储出来，
但该转储将无法直接恢复，需要用户介入处理。

35.15.4. 扩展更新

扩展机制的一个优点是，它为管理定义扩展对象的 SQL 命令的更新提供了便利方式。这是通过为每个已发布版本的扩展安装脚本关联一个版本名或版本号来实现的。此外，
如果你希望用户能够把数据库从一个版本动态更新到下一个版本，就应提供更新脚本，
以完成从一个版本切换到下一版本所需的修改。更新脚本的名称遵循如下模式：*extension--oldversion--newversion*`.sql`（例如，`foo--1.0--1.1.sql` 包含把扩展 `foo` 的 `1.0` 版修改为 `1.1` 版所需的命令）。

在有合适更新脚本可用的前提下，`ALTER EXTENSION UPDATE` 命令会把已安装的扩展更新到指定的新版本。更新脚本运行在 `CREATE EXTENSION` 为安装脚本提供的同一环境中：尤其是，`search_path` 的设置方式完全相同，而且脚本创建的任何新对象都会自动加入扩展中。

如果扩展有次级控制文件，那么用于更新脚本的控制参数就是与该脚本目标（新）
版本相关联的那些参数。

更新机制可以用于解决一个重要的特殊情况：将“松散”对象集合转换为扩展。在
PostgreSQL 于 9.1 加入扩展机制之前，许多人编写的扩展模块只是创建各种未打包的对象。
对于包含这些对象的现有数据库，
如何将它们转换为正确打包的扩展？删除它们再执行普通的 `CREATE EXTENSION` 是一种办法，但如果对象具有依赖关系（例如，某些表列使用扩展创建的数据类型），就不适合这样做。解决方法是先创建一个空扩展，再使用 `ALTER EXTENSION ADD`

将每个现有对象附加到扩展中，最后创建当前扩展版本中存在、而未打包版本中没有的新对象。CREATE EXTENSION 通过 FROM *old_version* 选项支持这种情况：它不运行目标版本的普通安装脚本，而是运行名为 *extension--oldversion--newversion*.sql 的更新脚本。用作 *old_version* 的虚拟版本名由扩展作者选择，不过通常约定使用 unpackaged。如果需要将多个先前版本更新为扩展形式，应使用不同的虚拟版本名来标识它们。

ALTER EXTENSION 能够执行一系列更新脚本文件来完成请求的更新。例如，如果只有 foo--1.0--1.1.sql 和 foo--1.1--2.0.sql 可用，那么当当前安装的是 1.0，而请求更新到 2.0 时，ALTER EXTENSION 就会按顺序应用这两个脚本。

PostgreSQL 并不对版本名称的属性作任何假设：例如，它并不知道 1.1 是否跟在 1.0 之后。它只是匹配可用的版本名，并选择需要应用更新脚本最少的那条路径。（实际上，版本名可以是任何不包含 --，且不以 - 开头或结尾的字符串。）

有时提供“降级”脚本也是有用的，例如 foo--1.1--1.0.sql 可用于回退与 1.1 版相关的修改。如果你这样做，要注意某个降级脚本可能会因为产生更短的路径而意外被采用。危险情况是，存在一个跨越多个版本的“快速路径”更新脚本，
同时又有一个可降级到该快速路径起点的脚本。这时，先降级再走快速路径，
可能比按版本逐步前进所需的步数更少。如果降级脚本删除了任何不可替代的对象，
就会产生不希望看到的结果。

要检查是否存在意外的更新路径，可使用以下命令：

```
SELECT * FROM pg_extension_update_paths('extension_name');
```

它会显示指定扩展的每一对不同已知版本名，
以及从源版本到目标版本将采用的更新路径序列；如果没有可用的更新路径，则显示 NULL。
路径会以文本形式显示，并使用 -- 作为分隔符。如果你更喜欢数组形式，可以使用 regexp_split_to_array(path,'--')。

35.15.5. 扩展示例

下面给出一个纯 SQL 扩展的完整示例：一个双元素复合类型，它可以在两个槽位中存储任意类型的值，这两个槽位名为“k”和“v”。非文本值会自动强制转换为文本后再存储。

脚本文件pair--1.0.sql如下所示：

```
-- complain if script is sourced in psql, rather than via CREATE
 EXTENSION
\echo Use "CREATE EXTENSION pair" to load this file. \quit

CREATE TYPE pair AS ( k text, v text );

CREATE OR REPLACE FUNCTION pair(anyelement, text)
RETURNS pair LANGUAGE SQL AS 'SELECT ROW($1, $2)::pair';

CREATE OR REPLACE FUNCTION pair(text, anyelement)
RETURNS pair LANGUAGE SQL AS 'SELECT ROW($1, $2)::pair';

CREATE OR REPLACE FUNCTION pair(anyelement, anyelement)
RETURNS pair LANGUAGE SQL AS 'SELECT ROW($1, $2)::pair';

CREATE OR REPLACE FUNCTION pair(text, text)
RETURNS pair LANGUAGE SQL AS 'SELECT ROW($1, $2)::pair;';

CREATE OPERATOR ~> (LEFTARG = text, RIGHTARG = anyelement,
 PROCEDURE = pair);
CREATE OPERATOR ~> (LEFTARG = anyelement, RIGHTARG = text,
 PROCEDURE = pair);
```

```
CREATE OPERATOR ~> (LEFTARG = anyelement, RIGHTARG = anyelement,
 PROCEDURE = pair);
CREATE OPERATOR ~> (LEFTARG = text, RIGHTARG = text, PROCEDURE =
 pair);
```

控制文件pair.control如下所示：

```
# pair extension
comment = 'A key/value pair data type'
default_version = '1.0'
relocatable = false
```

虽然你几乎不需要专门写一个 makefile 来把这两个文件安装到正确的目录中，但你也可以使用如下内容的 Makefile：

```
EXTENSION = pair
DATA = pair--1.0.sql

PG_CONFIG = pg_config
PGXS := $(shell $(PG_CONFIG) --pgxs)
include $(PGXS)
```

这个 makefile 依赖于 PGXS，其说明见第 35.16 节。执行 make install 命令会把控制文件和脚本文件安装到 pg_config 报告的正确目录中。

安装文件后，使用CREATE EXTENSION命令将这些对象装入某个具体数据库。

35.16. 扩展构建基础设施

如果你打算分发自己的 PostgreSQL 扩展模块，那么为它们搭建一个可移植的构建系统可能相当困难。因此，PostgreSQL 安装提供了一套称为 PGXS 的扩展构建基础设施，使简单的扩展模块可以针对已安装好的服务器直接构建。PGXS 主要面向包含 C 代码的扩展，不过它也能用于纯 SQL 扩展。注意，PGXS 并不打算成为一个可以用来构建任意与 PostgreSQL 交互软件的通用构建系统框架；它只是把简单服务器扩展模块的常见构建规则自动化。对于更复杂的软件包，你可能还是需要自己编写构建系统。

要使用PGXS基础设施构建扩展，必须编写一个简单的 makefile。在其中需要设置一些变量，并包含全局PGXSmakefile。下面的示例构建一个扩展模块，名为isbn_issn，由包含一些 C 代码的共享库、扩展控制文件、SQL 脚本以及文档文本文件组成：

```
MODULES = isbn_issn
EXTENSION = isbn_issn
DATA = isbn_issn--1.0.sql
DOCS = README.isbn_issn

PG_CONFIG = pg_config
PGXS := $(shell $(PG_CONFIG) --pgxs)
include $(PGXS)
```

最后三行应始终相同。在文件的前面部分，可以给变量赋值或添加自定义的make规则。

设定下列三个变量中的一个，以指定要构建的内容：

MODULES

要从同名源文件构建的共享库对象列表（列表中不要包含库后缀）

MODULE_big

要从多个源文件构建的共享库（在 OBJS 中列出目标文件）

PROGRAM

要构建的可执行程序（在 OBJS 中列出目标文件）

还可以设置以下变量：

EXTENSION

扩展名称；对于每个名称，你都必须提供一个 *extension*.control 文件，它将安装到 *prefix*/share/extension

MODULEDIR

prefix/share 下的子目录，用于安装 DATA 和 DOCS 文件（若未设置，则在设置了 EXTENSION 时默认为 extension，否则默认为 contrib）

DATA

要安装到 *prefix*/share/$MODULEDIR 的任意文件

DATA_built

要安装到 *prefix*/share/$MODULEDIR 的任意文件，但它们需要先被构建

DATA_TSEARCH

要安装到 *prefix*/share/tsearch_data 下的任意文件

DOCS

要安装到 *prefix*/doc/$MODULEDIR 下的任意文件

SCRIPTS

要安装到 *prefix*/bin 的脚本文件（不是二进制文件）

SCRIPTS_built

要安装到 *prefix*/bin 的脚本文件（不是二进制文件），但它们需要先被构建

REGRESS

回归测试用例列表（不带后缀），详见下文

REGRESS_OPTS

传递给 pg_regress 的额外开关

EXTRA_CLEAN

在 make clean 中要额外删除的文件

PG_CPPFLAGS

将被添加到 CPPFLAGS 前面

PG_LIBS

将被加入 PROGRAM 的链接命令行

SHLIB_LINK

将被加入 MODULE_big 的链接命令行

PG_CONFIG

要针对其进行构建的 PostgreSQL 安装所对应的 pg_config 程序路径（通常只写 pg_config，表示使用你 PATH 中找到的第一个）

> **小心**
>
> 更改 PG_CONFIG 只在针对 PostgreSQL 8.3 或更高版本构建时才有效。对于更早的版本，把它设置为 pg_config 以外的任何值都不起作用；你必须修改 PATH 来选择要针对其构建的安装。

把这个 makefile 命名为 Makefile，并放在保存扩展的目录中。然后你就可以执行 make 进行编译，再执行 make install 安装你的模块。默认情况下，该扩展会针对你 PATH 中找到的第一个 pg_config 所对应的 PostgreSQL 安装进行编译和安装。你也可以使用不同的安装，只需让 PG_CONFIG 指向它的 pg_config 程序，无论是在 makefile 中设置，还是在 make 命令行上设置都可以。

在 REGRESS 变量中列出的脚本用于对模块做回归测试，可在执行完 make install 之后，通过 make installcheck 来运行。要让它工作，你必须有一个正在运行的 PostgreSQL 服务器。列在 REGRESS 中的脚本文件必须位于扩展目录下名为 sql/ 的子目录中。这些文件必须具有 .sql 扩展名，而该扩展名不能出现在 makefile 的 REGRESS 列表中。对于每个测试，还应在名为 expected/ 的子目录中有一个包含期望输出的文件，其主干名相同，扩展名为 .out。make installcheck 会用 psql 执行每个测试脚本，并把得到的输出与对应的期望文件比较。任何差异都会以 diff -c 格式写入 regression.diffs 文件。注意，如果尝试运行一个缺少期望文件的测试，将被报告为“trouble”，所以请确保所有期望文件都已准备好。

> **提示**
>
> 创建期望文件最简单的方法是先建立空文件，
> 然后运行一次测试（当然这会报告差异）。检查 results/ 目录中的实际结果文件；如果它们与你对测试的预期一致，就把它们复制到 expected/ 中。

第 36 章 触发器

本章提供有关编写触发器函数的一般信息。触发器函数可以用大多数可用的过程语言编写，包括 PL/pgSQL（第 39 章）、PL/Tcl（第 40 章）、PL/Perl（第 41 章）和 PL/Python（第 42 章）。读完本章后，你还应查阅所用过程语言对应的章节，了解用该语言编写触发器的语言相关细节。

也可以用 C 编写触发器函数，不过大多数人会觉得使用某种过程语言更容易。目前还不能用普通 SQL 函数语言编写触发器函数。

36.1. 触发器行为概述

触发器是一种规定：每当执行某种类型的操作时，数据库就自动执行某个特定函数。
触发器可以附加到表和视图上。

在表上，触发器可以定义为在任何 INSERT、UPDATE 或 DELETE 操作之前或之后执行；既可以对每个被修改的行执行一次，也可以对每个 SQL 语句执行一次。UPDATE 触发器还可以设置成只有在 UPDATE 语句的 SET 子句中提到特定列时才触发。触发器也可以针对 TRUNCATE 语句触发。一旦发生触发器事件，就会在适当的时机调用该触发器的函数来处理该事件。

在视图上，触发器可以定义为代替 INSERT、UPDATE 或 DELETE 操作执行。这种 INSTEAD OF 触发器会对视图中每个需要修改的行触发一次。触发器函数负责对视图底层的基表执行必要的修改，并在适当时返回该修改后的行，使其在视图中呈现为应有的样子。
视图上的触发器也可以定义为在 INSERT、UPDATE 或 DELETE 操作之前或之后，对每个 SQL 语句执行一次。

必须先定义触发器函数，之后才能创建触发器本身。
触发器函数必须声明为不接受参数并返回类型 trigger 的函数。（触发器函数通过特殊传入的 TriggerData 结构体接收输入，而不是通过普通函数参数。）

创建出合适的触发器函数后，就可以使用CREATE TRIGGER创建触发器。同一个触发器函数可以供多个触发器使用。

PostgreSQL 同时提供每行触发器和每语句触发器。对于每行触发器，触发它的语句所影响的每一行都会调用一次触发器函数。相反，每语句触发器只会在执行相应语句时调用一次，
不管该语句影响了多少行。特别是，即使某个语句一行也没有影响，
仍会执行所有适用的每语句触发器。这两类触发器有时也分别称为
行级触发器和语句级触发器。TRUNCATE 的触发器只能定义为语句级。在视图上，之前或之后触发的触发器只能定义为语句级，而代替 INSERT、UPDATE 或 DELETE 触发的触发器只能定义为行级。

触发器还可以按照它们是在操作之前、之后，还是取代该操作而触发来分类。这些分别称为 BEFORE 触发器、AFTER 触发器和 INSTEAD OF 触发器。语句级 BEFORE 触发器自然会在语句开始执行任何操作之前触发，而语句级 AFTER 触发器则在语句末尾触发。这些类型的触发器可以定义在表或视图上。行级 BEFORE 触发器会在对某一行执行操作之前立即触发，而行级 AFTER 触发器会在语句结束时触发（但早于任何语句级 AFTER 触发器）。这些类型的触发器只能定义在表上。行级 INSTEAD OF 触发器只能定义在视图上；当视图中的每一行被识别为需要执行操作时，它们就会立即触发。

由每语句触发器调用的触发器函数应始终返回 NULL。由每行触发器调用的触发器函数则可以根据需要向调用它的执行器返回一个表行（即类型 HeapTuple 的值）。在操作之前触发的行级触发器可以作出如下选择：

- 它可以返回 NULL，以跳过对当前行的操作。这会指示执行器不要执行引发该触发器的行级操作（即对某个具体表行的插入、修改或删除）。
- 仅对于行级 INSERT 和 UPDATE 触发器，返回的行会成为将要插入的行，或替换正在更新的行。这使得触发器函数可以修改即将插入或更新的行。

不打算产生上述任何一种行为的行级 BEFORE 触发器，必须小心地返回传入的同一行作为结果（也就是说，INSERT 和 UPDATE 触发器返回 NEW 行，DELETE 触发器返回 OLD 行）。

行级 INSTEAD OF 触发器应当要么返回 NULL，表示它没有修改视图底层基表中的任何数据；要么返回传入的视图行（INSERT 和 UPDATE 操作返回 NEW 行，DELETE 操作返回 OLD 行）。非空返回值用来表明触发器已经在视图中完成了必要的数据修改。这会使该命令所影响的行数计数增加。仅对于 INSERT 和 UPDATE 操作，触发器在返回 NEW 行之前还可以修改它。这会改变 INSERT RETURNING 或 UPDATE RETURNING 返回的数据；当视图显示的数据与提供给它的数据不完全相同时，这很有用。

对于在操作之后触发的行级触发器，其返回值会被忽略，因此它们可以返回 NULL。

如果为同一关系上的同一事件定义了多个触发器，它们会按触发器名称的字母顺序触发。
对于 BEFORE 和 INSTEAD OF 触发器，每个触发器返回的、可能已经被修改过的行都会成为下一个触发器的输入。如果任何一个 BEFORE 或 INSTEAD OF 触发器返回 NULL，则会放弃对该行执行该操作，并且后续触发器（针对该行）都不会再触发。

触发器定义还可以指定一个布尔 WHEN 条件，用来测试是否应当触发该触发器。对于行级触发器，WHEN 条件可以检查该行各列的旧值和/或新值。（语句级触发器也可以有 WHEN 条件，不过这个特性对它们用处不大。）在 BEFORE 触发器中，WHEN 条件就在函数即将执行或本会执行之前求值，因此使用 WHEN 与在触发器函数开头测试同样的条件并无实质差别。不过，在 AFTER 触发器中，WHEN 条件会在行更改发生后立即求值，并决定是否要将一个事件排入队列，以便在语句末尾触发该触发器。因此，当 AFTER 触发器的 WHEN 条件不返回真时，就不必排队该事件，也不必在语句末尾重新取出该行。如果触发器只需要针对少数几行触发，这会让修改大量行的语句显著加快。INSTEAD OF 触发器不支持 WHEN 条件。

通常，行级 BEFORE 触发器用于检查或修改将要插入或更新的数据。例如，BEFORE 触发器可以用来把当前时间写入某个 timestamp 列，或者检查该行的两个元素是否一致。行级 AFTER 触发器最适合用于把更新传播到其他表，或对其他表执行一致性检查。之所以这样分工，是因为 AFTER 触发器可以确定自己看到的是该行的最终值，而 BEFORE 触发器不能；它之后可能还会有其他 BEFORE 触发器触发。如果没有特定理由把触发器做成 BEFORE 或 AFTER，那么 BEFORE 形式更高效，因为关于该操作的信息无需一直保存到语句末尾。

如果触发器函数执行 SQL 命令，那么这些命令可能会再次触发触发器。这就是所谓的级联触发器。对级联层数没有直接限制。级联还有可能导致同一个触发器被递归调用；例如，一个 INSERT 触发器可能执行一条向同一个表再插入一行的命令，从而导致该 INSERT 触发器再次触发。因此，在这种场景下，避免无限递归是触发器编写者自己的责任。

在定义触发器时，可以为它指定参数。在触发器定义中包含参数，
是为了让需求相似的不同触发器能够调用同一个函数。举例来说，
可以有一个通用触发器函数，它接受两个列名作为参数，把当前用户写入其中一个列，
把当前时间戳写入另一个列。只要编写得当，
这个触发器函数就应独立于它所作用的具体表。因此，
同一个函数可用于任何具有适当列的表上的 INSERT 事件，例如自动跟踪某个事务表中记录的创建。如果把它定义成 UPDATE 触发器，还可以用来跟踪最近更新事件。

每种支持触发器的编程语言都有自己的方法，使触发器输入数据可供触发器函数使用。
这些输入数据包括触发器事件的类型（例如 INSERT 或 UPDATE），以及 CREATE TRIGGER 中列出的任何参数。对于行级触发器，输入数据还包括 INSERT 和 UPDATE 触发器的 NEW 行，以及/或者 UPDATE 和 DELETE 触发器的 OLD 行。语句级触发器目前无法检查该语句修改的各个行。

36.2. 数据更改的可见性

如果你在触发器函数中执行 SQL 命令，而这些命令又访问该触发器所属的表，就需要了解数据可见性规则，因为这些规则决定了这些 SQL 命令是否能看到引发触发器的数据更改。简而言之：

- 语句级触发器遵循简单的可见性规则：语句所做的任何更改对在语句之前调用的语句级触发器都不可见，而所有修改对语句级 AFTER 触发器都可见。
- 导致触发器触发的数据更改（插入、更新或删除）对于在行级 BEFORE 触发器中执行的 SQL 命令自然是不可见的，因为它尚未发生。
- 然而，在行级 BEFORE 触发器中执行的 SQL 命令会看到同一外层命令中先前已处理各行上的数据更改结果。这里必须小心，因为这些更改事件的顺序通常不可预测；一条影响多行的 SQL 命令可能按任意顺序访问这些行。
- 类似地，行级 INSTEAD OF 触发器会看到同一外层命令中此前触发的 INSTEAD OF 触发器所造成的数据更改结果。
- 当行级 AFTER 触发器触发时，外层命令所做的所有数据更改都已经完成，并且对被调用的触发器函数可见。

如果你的触发器函数是用任何一种标准过程语言编写的，那么只有在该函数被声明为 VOLATILE 时，上述说法才成立。被声明为 STABLE 或 IMMUTABLE 的函数在任何情况下都看不到调用命令所做的更改。

有关数据可见性规则的更多信息可见第 43.4 节。第 36.4 节中的示例展示了这些规则。

36.3. 用 C 编写触发器函数

本节说明触发器函数接口的底层细节。这些信息仅在用 C 编写触发器函数时才需要。如果你使用更高层语言，这些细节会由系统代为处理。在多数情况下，在用 C 编写触发器之前，应先考虑使用过程语言。每种过程语言的文档都会说明如何用该语言编写触发器。

触发器函数必须使用“版本 1”函数管理器接口。

当函数由触发器管理器调用时，不会向它传递任何普通参数，但会传递一个指向 TriggerData 结构体的“上下文”指针。C 函数可以通过执行下列宏来检查自己是否由触发器管理器调用：

```
CALLED_AS_TRIGGER(fcinfo)
```

它会展开为：

```
((fcinfo)->context != NULL && IsA((fcinfo)->context, TriggerData))
```

如果该宏返回真，就可以安全地把 fcinfo->context 转换为 TriggerData * 类型，并使用其所指向的 TriggerData 结构体。函数绝不能修改 TriggerData 结构体本身，也不能修改它所指向的任何数据。

struct TriggerData定义在commands/trigger.h：

```
typedef struct TriggerData
{
    NodeTag       type;
    TriggerEvent  tg_event;
    Relation      tg_relation;
    HeapTuple     tg_trigtuple;
    HeapTuple     tg_newtuple;
    Trigger      *tg_trigger;
    Buffer        tg_trigtuplebuf;
    Buffer        tg_newtuplebuf;
} TriggerData;
```

其中各成员的定义如下：

type

始终是 T_TriggerData。

tg_event

描述调用函数的事件。可以使用以下宏检查 tg_event：

TRIGGER_FIRED_BEFORE(tg_event)

如果触发器在操作之前触发，则返回真。

TRIGGER_FIRED_AFTER(tg_event)

如果触发器在操作之后触发，则返回真。

TRIGGER_FIRED_INSTEAD(tg_event)

如果触发器是取代该操作而触发，则返回真。

TRIGGER_FIRED_FOR_ROW(tg_event)

如果触发器是因行级事件触发，则返回真。

TRIGGER_FIRED_FOR_STATEMENT(tg_event)

如果触发器是因语句级事件触发，则返回真。

TRIGGER_FIRED_BY_INSERT(tg_event)

如果触发器是由 INSERT 命令触发，则返回真。

TRIGGER_FIRED_BY_UPDATE(tg_event)

如果触发器是由 UPDATE 命令触发，则返回真。

TRIGGER_FIRED_BY_DELETE(tg_event)

如果触发器是由 DELETE 命令触发，则返回真。

TRIGGER_FIRED_BY_TRUNCATE(tg_event)

如果触发器是由 TRUNCATE 命令触发，则返回真。

tg_relation

指向一个描述该触发器所针对关系的结构体。有关此结构体的细节见 utils/rel.h。其中最值得关注的是 tg_relation->rd_att（关系元组的描述符）和 tg_relation->rd_rel->relname（关系名；其类型不是 char*，而是 NameData；如果需要名称的副本，请使用 SPI_getrelname(tg_relation) 取得一个 char*）。

tg_trigtuple

指向触发该触发器的那一行。这是正在被插入、更新或删除的行。如果该触发器因 INSERT 或 DELETE 而触发，那么如果你不想用另一行替换该行（在 INSERT 的情形下）或跳过该操作，就应从函数中返回它。

tg_newtuple

如果触发器因 UPDATE 而触发，则指向该行的新版本；如果因 INSERT 或 DELETE 而触发，则为 NULL。如果事件是 UPDATE，而你不想用另一行替换该行或跳过该操作，就必须从函数中返回它。

tg_trigger

一个指向Trigger类型结构体的指针，该结构体定义在utils/reltrigger.h：

```
typedef struct Trigger
{
    Oid         tgoid;
    char       *tgname;
    Oid         tgfoid;
    int16       tgtype;
    char        tgenabled;
    bool        tgisinternal;
    Oid         tgconstrrelid;
    Oid         tgconstrindid;
    Oid         tgconstraint;
    bool        tgdeferrable;
    bool        tginitdeferred;
    int16       tgnargs;
    int16       tgnattr;
    int16      *tgattr;
    char      **tgargs;
    char       *tgqual;
} Trigger;
```

其中tgname是触发器名称，tgnargs是tgargs中参数的数量，而tgargs是一个指针数组，指向CREATE TRIGGER语句中指定的参数。其他成员仅供内部使用。

tg_trigtuplebuf

包含 tg_trigtuple 的缓冲区；如果没有这样的元组，或它没有存储在磁盘缓冲区中，则为 InvalidBuffer。

tg_newtuplebuf

包含 tg_newtuple 的缓冲区；如果没有这样的元组，或它没有存储在磁盘缓冲区中，则为 InvalidBuffer。

触发器函数必须返回一个 HeapTuple 指针或一个 NULL 指针（不是 SQL 空值，也就是说不要把 *isNull* 设为真）。如果你不想修改正在处理的行，就要小心地根据情况返回适当的 tg_trigtuple 或 tg_newtuple。

36.4. 一个完整的触发器示例

这里给出一个非常简单的、用 C 编写的触发器函数示例。（用过程语言编写的触发器示例可见各过程语言的文档。）

函数 trigf 会报告表 ttest 中的行数，并在命令试图向列 x 中插入空值时跳过实际操作。（这样，这个触发器就起到了非空约束的作用，但不会中止事务。）

首先，表定义：

```
CREATE TABLE ttest (
    x integer
```

```
);
```

下面是触发器函数的源代码：

```
#include "postgres.h"
#include "executor/spi.h"       /* this is what you need to work
 with SPI */
#include "commands/trigger.h"   /* ... triggers ... */
#include "utils/rel.h"          /* ... and relations */
#ifdef PG_MODULE_MAGIC
PG_MODULE_MAGIC;
#endif

extern Datum trigf(PG_FUNCTION_ARGS);

PG_FUNCTION_INFO_V1(trigf);

Datum
trigf(PG_FUNCTION_ARGS)
{
    TriggerData *trigdata = (TriggerData *) fcinfo->context;
    TupleDesc   tupdesc;
    HeapTuple   rettuple;
    char       *when;
    bool        checknull = false;
    bool        isnull;
    int         ret, i;

    /* make sure it's called as a trigger at all */
    if (!CALLED_AS_TRIGGER(fcinfo))
        elog(ERROR, "trigf: not called by trigger manager");

    /* tuple to return to executor */
    if (TRIGGER_FIRED_BY_UPDATE(trigdata->tg_event))
        rettuple = trigdata->tg_newtuple;
    else
        rettuple = trigdata->tg_trigtuple;

    /* check for null values */
    if (!TRIGGER_FIRED_BY_DELETE(trigdata->tg_event)
        && TRIGGER_FIRED_BEFORE(trigdata->tg_event))
        checknull = true;

    if (TRIGGER_FIRED_BEFORE(trigdata->tg_event))
        when = "before";
    else
        when = "after ";

    tupdesc = trigdata->tg_relation->rd_att;

    /* connect to SPI manager */
    if ((ret = SPI_connect()) < 0)
        elog(ERROR, "trigf (fired %s): SPI_connect returned %d",
 when, ret);

    /* get number of rows in table */
    ret = SPI_exec("SELECT count(*) FROM ttest", 0);
```

```
    if (ret < 0)
        elog(ERROR, "trigf (fired %s): SPI_exec returned %d", when,
 ret);

    /* count(*) returns int8, so be careful to convert */
    i = DatumGetInt64(SPI_getbinval(SPI_tuptable->vals[0],
                                    SPI_tuptable->tupdesc,
                                    1,
                                    &isnull));

    elog (INFO, "trigf (fired %s): there are %d rows in ttest",
 when, i);

    SPI_finish();

    if (checknull)
    {
        SPI_getbinval(rettuple, tupdesc, 1, &isnull);
        if (isnull)
            rettuple = NULL;
    }

    return PointerGetDatum(rettuple);
}
```

编译完源代码后（参见第 35.9.6 节），声明函数和触发器：

```
CREATE FUNCTION trigf() RETURNS trigger
    AS 'filename'
    LANGUAGE C;

CREATE TRIGGER tbefore BEFORE INSERT OR UPDATE OR DELETE ON ttest
    FOR EACH ROW EXECUTE PROCEDURE trigf();

CREATE TRIGGER tafter AFTER INSERT OR UPDATE OR DELETE ON ttest
    FOR EACH ROW EXECUTE PROCEDURE trigf();
```

现在可以测试该触发器的行为：

```
=> INSERT INTO ttest VALUES (NULL);
INFO:  trigf (fired before): there are 0 rows in ttest
INSERT 0 0

-- Insertion skipped and AFTER trigger is not fired

=> SELECT * FROM ttest;
 x
---
(0 rows)

=> INSERT INTO ttest VALUES (1);
INFO:  trigf (fired before): there are 0 rows in ttest
INFO:  trigf (fired after ): there are 1 rows in ttest
                                        ^^^^^^^^
                              remember what we said about
 visibility.
INSERT 167793 1
vac=> SELECT * FROM ttest;
 x
```

```
---
 1
(1 row)

=> INSERT INTO ttest SELECT x * 2 FROM ttest;
INFO:  trigf (fired before): there are 1 rows in ttest
INFO:  trigf (fired after ): there are 2 rows in ttest
                                         ^^^^^^
                             remember what we said about
 visibility.
INSERT 167794 1
=> SELECT * FROM ttest;
 x
---
 1
 2
(2 rows)

=> UPDATE ttest SET x = NULL WHERE x = 2;
INFO:  trigf (fired before): there are 2 rows in ttest
UPDATE 0
=> UPDATE ttest SET x = 4 WHERE x = 2;
INFO:  trigf (fired before): there are 2 rows in ttest
INFO:  trigf (fired after ): there are 2 rows in ttest
UPDATE 1
vac=> SELECT * FROM ttest;
 x
---
 1
 4
(2 rows)

=> DELETE FROM ttest;
INFO:  trigf (fired before): there are 2 rows in ttest
INFO:  trigf (fired before): there are 1 rows in ttest
INFO:  trigf (fired after ): there are 0 rows in ttest
INFO:  trigf (fired after ): there are 0 rows in ttest
                                         ^^^^^^
                             remember what we said about
 visibility.
DELETE 2
=> SELECT * FROM ttest;
 x
---
(0 rows)
```

更复杂的示例可见 `src/test/regress/regress.c` 和spi。

第 37 章 规则系统

本章讨论PostgreSQL中的规则系统。产生式规则系统在概念上很简单，
但在实际使用时涉及许多微妙之处。

有些其他数据库系统定义了主动型数据库规则，它们通常表现为存储过程和触发器。在PostgreSQL中，这类功能同样可以用函数和触发器实现。

规则系统（更准确地说，查询重写规则系统）与存储过程和触发器完全不同。
它会修改查询，使其将规则纳入考虑，
然后把修改后的查询交给查询规划器进行规划和执行。它非常强大，可用于查询语言过程、视图和版本等许多用途。关于这一规则系统的理论基础及其能力，另见
[ston90b]和[ong90]。

37.1. 查询树

要理解规则系统如何工作，必须先知道它在何时被调用，以及它的输入和输出是什么。

规则系统位于解析器和规划器之间。它接收解析器的输出，即一棵查询树，
以及用户定义的重写规则；这些规则本身也是带有一些附加信息的查询树。
它会生成零棵或多棵查询树作为结果。因此，
它的输入和输出始终都是解析器本身也可能产生的东西，
所以它看到的任何内容基本上都可以表示为一个SQL语句。

那么什么是查询树？它是SQL语句的一种内部表示，其中构成语句的各个部分被分别存储。
如果你设置了配置参数`debug_print_parse`、`debug_print_rewritten`或`debug_print_plan`，这些查询树就可以显示在服务器日志中。
规则动作也以查询树的形式存储在系统目录`pg_rewrite`中。它们的格式不像日志输出，
但包含的却是完全相同的信息。

阅读原始查询树需要一些经验。但由于查询树的SQL表示已足以理解规则系统，
本章不会讲解如何阅读它们。

阅读本章查询树的 SQL 表示时，必须能够识别语句以查询树结构表示时被拆分成的各个部分。查询树的组成部分如下：

命令类型

这是一个简单的值，用来说明是哪一种命令（`SELECT`、`INSERT`、`UPDATE`、`DELETE`）
产生了该查询树。

范围表

范围表是查询中使用的关系列表。在`SELECT`语句中，它们就是关键字`FROM`
后面给出的关系。

每个范围表项标识一个表或视图，并说明在查询的其他部分以哪个名称来称呼它。
在查询树中，范围表项是按编号而不是按名称引用的，因此即使像
SQL语句中那样出现重名，这里也无关紧要。
这种情况可能出现在规则的范围表被合并之后。本章中的示例不会涉及这种情形。

结果关系

这是范围表中的一个索引，用来标识查询结果应写入哪个关系。

`SELECT`查询没有结果关系。（`SELECT INTO`这一特殊情况与`CREATE TABLE`后接`INSERT ... SELECT`几乎相同，这里不再单独讨论。）

对于INSERT、UPDATE和DELETE命令，结果关系就是要让修改生效的表（或视图！）。

目标列表

目标列表是定义查询结果的表达式列表。对于SELECT，这些表达式构成查询的最终输出。它们对应于关键字SELECT和FROM之间的表达式。（* 只是某个关系全部列名的缩写。解析器会把它展开为各个独立列，因此规则系统永远不会看到它。）

DELETE命令不需要普通的目标列表，因为它们不产生任何结果。相反，规划器会向空目标列表加入一个特殊的CTID项，以便执行器找到要删除的行。（当结果关系是普通表时，会加入CTID；如果结果关系是视图，则规则系统会改为加入一个整行变量，如第 37.2.4 节所述。）

对于INSERT命令，目标列表描述的是将要进入结果关系的新行。它由VALUES子句中的表达式，或INSERT ... SELECT里SELECT子句中的表达式组成。重写过程的第一步会为原始命令未赋值但带有默认值的列补上目标列表项。其余列（既没有给定值也没有默认值）将由规划器填入常量空值表达式。

对于UPDATE命令，目标列表描述要替换旧行的新行。在规则系统中，它只包含命令中SET column = expression部分的表达式。规划器会通过插入把旧行值复制到新行的表达式来处理缺失列。与DELETE一样，还会加入一个CTID或整行变量，以便执行器标识要更新的旧行。

目标列表中的每一项都包含一个表达式，它可以是常量值、指向范围表中某个关系列的变量、一个参数，或者由函数调用、常量、变量、操作符等构成的表达式树。

条件

查询的条件是一个表达式，与目标列表项中的表达式很相似。该表达式的结果值是布尔值，用来说明是否应对最终结果行执行相应操作（INSERT、UPDATE、DELETE或SELECT）。它对应于SQL语句的WHERE子句。

连接树

查询的连接树显示了FROM子句的结构。对于SELECT ... FROM a, b, c这样的简单查询，连接树只是FROM项的一个列表，因为允许按任意顺序连接它们。但当使用JOIN表达式，特别是外连接时，就必须按连接所显示的顺序进行连接。在这种情况下，连接树展示的是JOIN表达式的结构。与特定JOIN子句相关的限制（来自ON或USING表达式）会作为条件表达式附着在相应的连接树节点上。把顶层WHERE表达式也存储为附着在顶层连接树项上的一个条件，会很方便。因此，连接树实际上同时表示SELECT的FROM和WHERE子句。

其他

查询树的其他部分，如ORDER BY子句，这里不作关注。规则系统在应用规则时会替换其中某些项，但这与规则系统的基本原理关系不大。

37.2. 视图和规则系统

在 PostgreSQL 中，视图通过规则系统实现。实际上，以下命令：

```
CREATE VIEW myview AS SELECT * FROM mytab;
```

与下面这两条命令基本没有区别：

```
CREATE TABLE myview (same column list as mytab);
```

```
CREATE RULE "_RETURN" AS ON SELECT TO myview DO INSTEAD
    SELECT * FROM mytab;
```

因为这正是 CREATE VIEW 命令在内部所做的事情。这会带来一些副作用。其中之一是，在 PostgreSQL 系统目录中，视图的信息与表的信息完全相同。因此，对解析器而言，表和视图完全没有区别。它们是同一种东西：关系。

37.2.1. SELECT 规则如何工作

ON SELECT规则会在所有查询上作为最后一步应用，即使给出的命令是INSERT、UPDATE或DELETE也一样。它们与其他命令类型上的规则在语义上不同，
因为它们是就地修改查询树，而不是创建新的查询树。因此我们先讨论SELECT规则。

目前，一个ON SELECT规则中只能有一个动作，而且它必须是一个无条件、带有INSTEAD的SELECT动作。之所以有这个限制，是为了让规则足够安全，从而能够向普通用户开放；
它也把ON SELECT规则限制为像视图那样工作。

本章的示例是两个进行一些计算的连接视图，以及另外一些依次使用它们的视图。
最初两个视图中的一个会在后面通过为INSERT、UPDATE和DELETE操作添加规则来定制，
从而最终得到一个在行为上像真正的表、但又带有某些特殊功能的视图。作为入门示例，
这并不算简单，因此会让理解变得更难一些。但与其使用许多可能令人混淆的不同示例，
不如用一个示例逐步覆盖这里讨论的全部要点。

这个例子需要一个返回两个整数中较小者的 min 函数。我们这样创建它：

```
CREATE FUNCTION min(integer, integer) RETURNS integer AS $$
    SELECT CASE WHEN $1 < $2 THEN $1 ELSE $2 END
$$ LANGUAGE SQL STRICT;
```

在前两节对规则系统的描述中，我们需要用到如下真实表：

```
CREATE TABLE shoe_data (
    shoename   text,          -- primary key
    sh_avail   integer,       -- available number of pairs
    slcolor    text,          -- preferred shoelace color
    slminlen   real,          -- minimum shoelace length
    slmaxlen   real,          -- maximum shoelace length
    slunit     text           -- length unit
);

CREATE TABLE shoelace_data (
    sl_name    text,          -- primary key
    sl_avail   integer,       -- available number of pairs
    sl_color   text,          -- shoelace color
    sl_len     real,          -- shoelace length
    sl_unit    text           -- length unit
);

CREATE TABLE unit (
    un_name    text,          -- primary key
    un_fact    real           -- factor to transform to cm
);
```

如你所见，它们表示的是鞋店数据。

这些视图是这样创建的：

```
CREATE VIEW shoe AS
    SELECT sh.shoename,
```

```
           sh.sh_avail,
           sh.slcolor,
           sh.slminlen,
           sh.slminlen * un.un_fact AS slminlen_cm,
           sh.slmaxlen,
           sh.slmaxlen * un.un_fact AS slmaxlen_cm,
           sh.slunit
      FROM shoe_data sh, unit un
     WHERE sh.slunit = un.un_name;

CREATE VIEW shoelace AS
    SELECT s.sl_name,
           s.sl_avail,
           s.sl_color,
           s.sl_len,
           s.sl_unit,
           s.sl_len * u.un_fact AS sl_len_cm
      FROM shoelace_data s, unit u
     WHERE s.sl_unit = u.un_name;

CREATE VIEW shoe_ready AS
    SELECT rsh.shoename,
           rsh.sh_avail,
           rsl.sl_name,
           rsl.sl_avail,
           least(rsh.sh_avail, rsl.sl_avail) AS total_avail
      FROM shoe rsh, shoelace rsl
     WHERE rsl.sl_color = rsh.slcolor
       AND rsl.sl_len_cm >= rsh.slminlen_cm
       AND rsl.sl_len_cm <= rsh.slmaxlen_cm;
```

创建shoelace视图的CREATE VIEW命令（这是我们这里最简单的一个例子）会创建一个关系shoelace，并在pg_rewrite中创建一项，说明只要查询的范围表中引用了关系shoelace，就必须应用一条重写规则。该规则没有规则条件（稍后在讨论非SELECT规则时再谈，因为目前SELECT规则不能有规则条件），并且它是INSTEAD规则。注意，规则条件与查询条件不是一回事。这里我们的规则动作带有一个查询条件。规则动作本身是一棵查询树，它是视图创建命令中SELECT语句的一个副本。

> **注意**
>
> 你在pg_rewrite项中看到的、用于NEW和OLD的两个额外范围表项，与SELECT规则无关。

现在我们填充 unit，shoe_data 和 shoelace_data，然后对视图运行一个简单查询：

```
INSERT INTO unit VALUES ('cm', 1.0);
INSERT INTO unit VALUES ('m', 100.0);
INSERT INTO unit VALUES ('inch', 2.54);

INSERT INTO shoe_data VALUES ('sh1', 2, 'black', 70.0, 90.0, 'cm');
INSERT INTO shoe_data VALUES ('sh2', 0, 'black', 30.0, 40.0,
 'inch');
INSERT INTO shoe_data VALUES ('sh3', 4, 'brown', 50.0, 65.0, 'cm');
INSERT INTO shoe_data VALUES ('sh4', 3, 'brown', 40.0, 50.0,
 'inch');
```

```
INSERT INTO shoelace_data VALUES ('sl1', 5, 'black', 80.0, 'cm');
INSERT INTO shoelace_data VALUES ('sl2', 6, 'black', 100.0, 'cm');
INSERT INTO shoelace_data VALUES ('sl3', 0, 'black', 35.0 , 'inch')
;
INSERT INTO shoelace_data VALUES ('sl4', 8, 'black', 40.0 , 'inch')
;
INSERT INTO shoelace_data VALUES ('sl5', 4, 'brown', 1.0 , 'm');
INSERT INTO shoelace_data VALUES ('sl6', 0, 'brown', 0.9 , 'm');
INSERT INTO shoelace_data VALUES ('sl7', 7, 'brown', 60 , 'cm');
INSERT INTO shoelace_data VALUES ('sl8', 1, 'brown', 40 , 'inch');

SELECT * FROM shoelace;

 sl_name   | sl_avail | sl_color | sl_len | sl_unit | sl_len_cm
-----------+----------+----------+--------+---------+-----------
 sl1       |        5 | black    |     80 | cm      |        80
 sl2       |        6 | black    |    100 | cm      |       100
 sl7       |        7 | brown    |     60 | cm      |        60
 sl3       |        0 | black    |     35 | inch    |      88.9
 sl4       |        8 | black    |     40 | inch    |     101.6
 sl8       |        1 | brown    |     40 | inch    |     101.6
 sl5       |        4 | brown    |      1 | m       |       100
 sl6       |        0 | brown    |    0.9 | m       |        90
(8 rows)
```

这是你能在这些视图上执行的最简单的SELECT，因此我们借此机会说明视图规则的基础。SELECT * FROM shoelace由解析器解释后，会生成如下查询树：

```
SELECT shoelace.sl_name, shoelace.sl_avail,
       shoelace.sl_color, shoelace.sl_len,
       shoelace.sl_unit, shoelace.sl_len_cm
  FROM shoelace shoelace;
```

然后它会被交给规则系统。规则系统遍历范围表，检查其中的关系是否有相应规则。在处理shoelace的范围表项时（到目前为止只有这一个），它会找到带有如下查询树的_RETURN规则：

```
SELECT s.sl_name, s.sl_avail,
       s.sl_color, s.sl_len, s.sl_unit,
       s.sl_len * u.un_fact AS sl_len_cm
  FROM shoelace old, shoelace new,
       shoelace_data s, unit u
 WHERE s.sl_unit = u.un_name;
```

为了展开视图，重写器只需创建一个子查询范围表条目，其中包含规则的动作查询树，然后用这个范围表条目替换原先引用视图的条目。
所得的重写后查询树几乎等同于输入以下语句的结果：

```
SELECT shoelace.sl_name, shoelace.sl_avail,
       shoelace.sl_color, shoelace.sl_len,
       shoelace.sl_unit, shoelace.sl_len_cm
  FROM (SELECT s.sl_name,
               s.sl_avail,
               s.sl_color,
               s.sl_len,
               s.sl_unit,
               s.sl_len * u.un_fact AS sl_len_cm
          FROM shoelace_data s, unit u
         WHERE s.sl_unit = u.un_name) shoelace;
```

不过有一点不同：子查询的范围表中有两个额外条目，shoelace old 和 shoelace new。这些条目不直接参与查询，因为子查询的连接树或目标列表并未引用它们。重写器用它们保存原先引用视图的范围表条目中的访问权限检查信息。这样，
即使重写后的查询没有直接使用视图，
执行器仍会检查用户是否具有访问该视图所需的权限。

这就是应用的第一条规则。
规则系统接着会检查顶层查询中剩余的范围表项（本例中已经没有了），
并递归检查新增子查询中的范围表项，看它们是否引用了视图。（但它不会展开old或new，
否则就会出现无限递归！）在这个例子中，shoelace_data和unit都没有重写规则，
因此重写到此结束，上面的结果就是交给规划器的最终结果。

现在我们想写一个查询，找出商店里目前哪些鞋子有颜色和长度都匹配的鞋带，
并且完全匹配的总双数大于等于二。

```
SELECT * FROM shoe_ready WHERE total_avail >= 2;

 shoename | sh_avail | sl_name | sl_avail | total_avail
----------+----------+---------+----------+-------------
 sh1      |        2 | sl1     |        5 |           2
 sh3      |        4 | sl7     |        7 |           4
(2 rows)
```

这一次解析器的输出是如下查询树：

```
SELECT shoe_ready.shoename, shoe_ready.sh_avail,
       shoe_ready.sl_name, shoe_ready.sl_avail,
       shoe_ready.total_avail
  FROM shoe_ready shoe_ready
 WHERE shoe_ready.total_avail >= 2;
```

首先应用的是用于 shoe_ready 视图的规则，它得到如下查询树：

```
SELECT shoe_ready.shoename, shoe_ready.sh_avail,
       shoe_ready.sl_name, shoe_ready.sl_avail,
       shoe_ready.total_avail
  FROM (SELECT rsh.shoename,
               rsh.sh_avail,
               rsl.sl_name,
               rsl.sl_avail,
               least(rsh.sh_avail, rsl.sl_avail) AS total_avail
          FROM shoe rsh, shoelace rsl
         WHERE rsl.sl_color = rsh.slcolor
           AND rsl.sl_len_cm >= rsh.slminlen_cm
           AND rsl.sl_len_cm <= rsh.slmaxlen_cm) shoe_ready
 WHERE shoe_ready.total_avail >= 2;
```

类似地，shoe 和 shoelace 的规则会被替换进子查询的范围表中，最终得到一棵三层查询树：

```
SELECT shoe_ready.shoename, shoe_ready.sh_avail,
       shoe_ready.sl_name, shoe_ready.sl_avail,
       shoe_ready.total_avail
  FROM (SELECT rsh.shoename,
               rsh.sh_avail,
               rsl.sl_name,
               rsl.sl_avail,
               least(rsh.sh_avail, rsl.sl_avail) AS total_avail
          FROM (SELECT sh.shoename,
```

```
                        sh.sh_avail,
                        sh.slcolor,
                        sh.slminlen,
                        sh.slminlen * un.un_fact AS slminlen_cm,
                        sh.slmaxlen,
                        sh.slmaxlen * un.un_fact AS slmaxlen_cm,
                        sh.slunit
                  FROM shoe_data sh, unit un
                 WHERE sh.slunit = un.un_name) rsh,
               (SELECT s.sl_name,
                        s.sl_avail,
                        s.sl_color,
                        s.sl_len,
                        s.sl_unit,
                        s.sl_len * u.un_fact AS sl_len_cm
                  FROM shoelace_data s, unit u
                 WHERE s.sl_unit = u.un_name) rsl
         WHERE rsl.sl_color = rsh.slcolor
           AND rsl.sl_len_cm >= rsh.slminlen_cm
           AND rsl.sl_len_cm <= rsh.slmaxlen_cm) shoe_ready
 WHERE shoe_ready.total_avail > 2;
```

事实证明，规划器会把这棵树折叠成一个两层的查询树：最底层的 SELECT 命令会被“上拉”到中间的 SELECT 中，因为没有必要单独处理它们。但中间的 SELECT 会保持与顶层分离，因为它包含聚合函数。如果把它们也上拉，就会改变最顶层 SELECT 的行为，而这不是我们想要的。不过，折叠查询树是一种优化，重写系统不必关心它。

37.2.2. 非 SELECT 语句中的视图规则

上文对视图规则的说明没有涉及查询树中的两个细节：命令类型和结果关系。实际上，视图规则并不需要命令类型，但结果关系可能会影响查询重写器的工作方式，因为当结果关系是视图时，必须进行特殊处理。

SELECT的查询树与其他任何命令的查询树之间只有少数差别。显然，它们的命令类型不同；对于SELECT之外的命令，结果关系会指向结果应写入的那个范围表项。除此之外，其余部分完全相同。因此，假设有两个表t1和t2，都具有列a和b，那么下面两条语句的查询树：

```
SELECT t2.b FROM t1, t2 WHERE t1.a = t2.a;

UPDATE t1 SET b = t2.b FROM t2 WHERE t1.a = t2.a;
```

几乎是一样的。特别是：

- 范围表中包含表t1和t2的项。
- 目标列表中都包含一个变量，该变量指向表t2的范围表项中的列b。
- 条件表达式会比较两个范围表项中的列a是否相等。
- 连接树都表示t1与t2之间的一次简单连接。

结果是，这两个查询树都会产生相似的执行计划：它们都是对这两个表的连接。对于UPDATE，规划器会把t1中缺失的列补入目标列表，最终查询树会变成：

```
UPDATE t1 SET a = t1.a, b = t2.b FROM t2 WHERE t1.a = t2.a;
```

因而，执行器在这个连接上运行时会产生与下面语句完全相同的结果集：

```
SELECT t1.a, t2.b FROM t1, t2 WHERE t1.a = t2.a;
```

但在UPDATE
中有个小问题：执行器计划中负责连接的那一部分并不关心连接结果将被用于什么。
它只是生成一个行结果集。一个是SELECT命令，另一个是UPDATE命令，
这一差别是在执行器更高层处理的；在那里，系统知道这是一个UPDATE，
也知道结果应写入表t1。但问题在于：其中哪一行应当被新行替换？

为了解决这个问题，UPDATE（以及DELETE）语句的目标列表中会额外加入一项：当前元组ID（CTID）。这是一个系统列，包含该行所在的文件块号以及在块中的位置。已知表之后，就可以利用CTID取回要更新的t1原始行。把CTID加入目标列表后，查询实际上会变成：

```
SELECT t1.a, t2.b, t1.ctid FROM t1, t2 WHERE t1.a = t2.a;
```

现在还要引入PostgreSQL的另一个细节。旧表行不会被覆盖，这也是ROLLBACK
之所以很快的原因。在UPDATE中，新结果行会被插入表中（去掉CTID之后），而
CTID所指向的旧行，其行头中的cmax和xmax项会被设置为当前命令计数器和当前事务 ID。
于是旧行被隐藏起来，事务提交后，清理器最终就可以移除这条死行。

知道了这些以后，我们就可以用完全相同的方式把视图规则应用到任何命令上。没有区别。

37.2.3. PostgreSQL 中视图的威力

上文演示了规则系统如何把视图定义整合进原始查询树。在第二个示例中，
从一个视图发出的简单SELECT最终生成了一棵四表连接的查询树（unit
以不同名称使用了两次）。

用规则系统实现视图的好处在于：规划器能够在单棵查询树中同时看到哪些表必须被扫描、
这些表之间的关系、来自视图的限制条件，以及原始查询自身的条件。
即使原始查询本身已经是对若干视图的连接，情况也仍然如此。
规划器必须决定执行查询的最佳路径，而它掌握的信息越多，这个决定通常就越好。
PostgreSQL实现的规则系统能够保证，到那个阶段为止，
关于该查询的全部可用信息都已经集中在这里。

37.2.4. 更新视图

如果把一个视图命名为 INSERT、UPDATE 或 DELETE 的目标关系会发生什么？简单地做上述替换会得到一个结果关系指向子查询范围表项的查询树，这是行不通的。相反，
重写器假定该操作将由视图上的一个 INSTEAD OF 触发器处理。（如果没有这样的触发器，执行器会在开始执行时抛出一个错误。）在这种情形下，重写的工作方式略有不同。对于 INSERT，重写器完全不处理该视图，而是让它继续作为查询的结果关系。对于 UPDATE 和 DELETE，仍然需要展开视图查询以产生 the “old” rows that the command will attempt to update or delete. So the view is expanded as normal, but another unexpanded range-table entry is added to the query to represent the view in its capacity as the result relation.

此时出现的问题是，如何标识视图中需要更新的行。回想一下，当结果关系是表时，
目标列表中会加入一个特殊的CTID项，用来标识待更新行的物理位置。
如果结果关系是视图，这就行不通了，因为视图没有CTID，它的行并没有实际物理位置。
取而代之的是，对于UPDATE或DELETE操作，目标列表中会加入一个特殊的wholerow项，
它会展开为该视图的全部列。执行器利用这个值把“旧”行传给INSTEAD OF触发器。之后由触发器根据旧行值和新行值决定应当更新什么。

如果没有更新视图的 INSTEAD OF 触发器，执行器将抛出一个错误，因为它自己无法自动更新视图。要改变这一点，我们可以定义规则来修改视图上 INSERT、UPDATE 和 DELETE 命令的行为。这些规则会重写命令，通常是把它改写成更新一个或多个表而不是视图的命令。这正是下一节的主题。

注意，规则总是先被处理，也就是在原始查询被规划和执行之前先完成重写。因此，
如果一个视图上同时有INSTEAD OF触发器以及INSERT、UPDATE或DELETE规则，那么规则会先执行；根据其结果，触发器甚至可能完全不会被用到。

37.3. INSERT、UPDATE和DELETE规则

定义在INSERT、UPDATE和DELETE上的规则与前几节描述的视图规则有明显的不同。首先，它们的CREATE RULE命令允许更多：

- 它们可以没有动作。
- 它们可以有多个动作。
- 它们可以是INSTEAD或ALSO（缺省）。
- 伪关系NEW和OLD可以派上用场。
- 它们可以有规则条件。

第二，它们不是就地修改查询树，而是创建零棵或多棵新查询树，并且可能丢弃原始查询树。

37.3.1. 更新规则如何工作

记住以下语法：

```
CREATE [ OR REPLACE ] RULE name AS ON event
    TO table [ WHERE condition ]
    DO [ ALSO | INSTEAD ] { NOTHING | command | ( command ; command
 ... ) }
```

在后文中，更新规则是指定义在INSERT、UPDATE或DELETE上的规则。

当查询树的结果关系和命令类型分别等于CREATE RULE命令中给出的对象和事件时，规则系统就会应用更新规则。对于更新规则，规则系统会创建一个查询树列表，初始时该列表为空。动作可以有零个（关键字NOTHING）、一个或多个。为简化说明，我们先看只有一个动作的规则。这个规则可以有条件，也可以没有条件；它还可以是INSTEAD或ALSO（默认值）。

什么是规则条件？它是一种限制，用来说明何时执行规则动作、何时不执行。这个条件只能引用伪关系NEW和/或OLD，它们基本上表示作为对象给出的那个关系，只是带有特殊含义。

因此，对于只有一个动作的规则，有以下三种情况会产生相应的查询树。

没有条件，有ALSO或INSTEAD

规则动作的查询树，再附加原始查询树的条件

给出了条件，有ALSO

规则动作的查询树，再附加规则条件和原始查询树的条件

给出了条件，有INSTEAD

规则动作的查询树，再附加规则条件和原始查询树的条件；以及附加了规则条件取反后的原始查询树

最后，如果规则是 ALSO，就把未经更改的原始查询树加入列表。由于只有带条件的INSTEAD 规则已经加入了原始查询树，因此，对于只有一个动作的规则，最终会得到一棵或两棵输出查询树。

对于ON INSERT规则，原始查询（如果未被INSTEAD抑制）会先于规则添加的任何动作执行。这样一来，这些动作就能看到被插入的行。但对ON UPDATE和ON DELETE规则，原始查

询会在规则添加的动作之后执行。这就保证了这些动作能够看到将被更新或删除的行；否则，动作可能什么也做不了，因为它们找不到符合条件的行。

由规则动作生成的查询树会再次送入重写系统，随后可能还会有更多规则被应用，从而产生更多或更少的查询树。因此，一个规则的动作必须具有不同的命令类型，或者具有与该规则自身不同的结果关系。否则，这种递归过程就会陷入无限循环。（规则的递归展开会被检测出来，并作为错误报告。）

pg_rewrite系统目录中动作里的查询树只是模板。由于它们可以引用NEW和OLD的范围表项，因此在使用前必须进行一些替换。对任何NEW的引用，都会先在原始查询的目标列表中查找相应项。如果找到了，就用该项的表达式替换该引用。否则，NEW就与OLD含义相同（对于UPDATE），或者被替换为一个空值（对于INSERT）。任何对OLD的引用，都会被替换为对结果关系对应范围表项的引用。

在系统完成应用更新规则后，它再应用视图规则到生成的查询树上。视图无法插入新的更新动作，所以没有必要向视图重写的输出应用更新规则。

37.3.1.1. 第一个规则：逐步分析

假设我们想跟踪shoelace_data关系中sl_avail列的变化。因此，我们建立一个日志表和一条规则，使其在shoelace_data上执行UPDATE时，有条件地写入一条日志记录。

```
CREATE TABLE shoelace_log (
    sl_name    text,          -- shoelace changed
    sl_avail   integer,       -- new available value
    log_who    text,          -- who did it
    log_when   timestamp      -- when
);

CREATE RULE log_shoelace AS ON UPDATE TO shoelace_data
    WHERE NEW.sl_avail <> OLD.sl_avail
    DO INSERT INTO shoelace_log VALUES (
                                        NEW.sl_name,
                                        NEW.sl_avail,
                                        current_user,
                                        current_timestamp
                                    );
```

现在有人执行了：

```
UPDATE shoelace_data SET sl_avail = 6 WHERE sl_name = 'sl7';
```

然后我们查看日志表：

```
SELECT * FROM shoelace_log;

 sl_name | sl_avail | log_who | log_when
---------+----------+---------+----------------------------------
 sl7     |        6 | Al      | Tue Oct 20 16:14:45 1998 MET DST
(1 row)
```

这正是我们预期的结果。后台发生的事情如下。解析器创建了查询树：

```
UPDATE shoelace_data SET sl_avail = 6
  FROM shoelace_data shoelace_data
 WHERE shoelace_data.sl_name = 'sl7';
```

此时存在一条ON UPDATE规则log_shoelace，它带有如下规则条件表达式：

```
NEW.sl_avail <> OLD.sl_avail
```

其动作是：

```
INSERT INTO shoelace_log VALUES (
       new.sl_name, new.sl_avail,
       current_user, current_timestamp )
  FROM shoelace_data new, shoelace_data old;
```

（这看起来有点奇怪，因为通常你不能写INSERT ... VALUES ... FROM。这里的FROM子句只是为了表明查询树中存在用于new和old的范围表项。之所以需要这些项，是为了让INSERT命令查询树中的变量能够引用它们。）

该规则是一条带条件的ALSO规则，因此规则系统必须返回两棵查询树：修改后的规则动作，以及原始查询树。第 1 步中，原始查询的范围表会并入规则动作的查询树，得到：

```
INSERT INTO shoelace_log VALUES (
       new.sl_name, new.sl_avail,
       current_user, current_timestamp )
  FROM shoelace_data new, shoelace_data old,
       shoelace_data shoelace_data;
```

第 2 步将规则条件加进去，因此结果集被限制为sl_avail发生变化的行：

```
INSERT INTO shoelace_log VALUES (
       new.sl_name, new.sl_avail,
       current_user, current_timestamp )
  FROM shoelace_data new, shoelace_data old,
       shoelace_data shoelace_data
 WHERE new.sl_avail <> old.sl_avail;
```

（这看起来更奇怪，因为INSERT ... VALUES同样没有WHERE子句，但规划器和执行器处理它并无困难。反正它们本来也需要为INSERT ... SELECT支持相同功能。）

第 3 步把原始查询树的条件加进去，把结果集进一步限制为只有那些原始查询会触及的行：

```
INSERT INTO shoelace_log VALUES (
       new.sl_name, new.sl_avail,
       current_user, current_timestamp )
  FROM shoelace_data new, shoelace_data old,
       shoelace_data shoelace_data
 WHERE new.sl_avail <> old.sl_avail
   AND shoelace_data.sl_name = 'sl7';
```

第 4 步把NEW引用替换为原始查询树中的目标列表项，或者替换为结果关系中相应的变量引用：

```
INSERT INTO shoelace_log VALUES (
       shoelace_data.sl_name, 6,
       current_user, current_timestamp )
  FROM shoelace_data new, shoelace_data old,
       shoelace_data shoelace_data
 WHERE 6 <> old.sl_avail
   AND shoelace_data.sl_name = 'sl7';
```

第 5 步把OLD引用替换为结果关系引用：

```
INSERT INTO shoelace_log VALUES (
       shoelace_data.sl_name, 6,
       current_user, current_timestamp )
```

```
  FROM shoelace_data new, shoelace_data old,
       shoelace_data shoelace_data
 WHERE 6 <> shoelace_data.sl_avail
   AND shoelace_data.sl_name = 'sl7';
```

至此就完成了。由于规则是ALSO，我们还要输出原始查询树。简而言之，规则系统输出的是一个包含两棵查询树的列表，它们对应于以下语句：

```
INSERT INTO shoelace_log VALUES (
       shoelace_data.sl_name, 6,
       current_user, current_timestamp )
  FROM shoelace_data
 WHERE 6 <> shoelace_data.sl_avail
   AND shoelace_data.sl_name = 'sl7';

UPDATE shoelace_data SET sl_avail = 6
 WHERE sl_name = 'sl7';
```

它们会按这个顺序执行，而这正是该规则想要达到的效果。

上述替换以及附加的条件能够保证：如果原始查询是下面这样，就不会写入任何日志记录：

```
UPDATE shoelace_data SET sl_color = 'green'
 WHERE sl_name = 'sl7';
```

在这种情况下，原始查询树不包含sl_avail的目标列表项，因此NEW.sl_avail会被shoelace_data.sl_avail替换。于是，规则生成的额外命令是：

```
INSERT INTO shoelace_log VALUES (
       shoelace_data.sl_name, shoelace_data.sl_avail,
       current_user, current_timestamp )
  FROM shoelace_data
 WHERE shoelace_data.sl_avail <> shoelace_data.sl_avail
   AND shoelace_data.sl_name = 'sl7';
```

而该条件永远不可能为真。

如果原始查询修改多行，这种机制同样能够正常工作。例如，假设有人发出如下命令：

```
UPDATE shoelace_data SET sl_avail = 0
 WHERE sl_color = 'black';
```

实际上会更新四行（sl1、sl2、sl3和sl4）。但sl3本来就已经是sl_avail = 0。在这种情况下，原始查询树的条件不同，因此规则会产生额外的查询树：

```
INSERT INTO shoelace_log
SELECT shoelace_data.sl_name, 0,
       current_user, current_timestamp
  FROM shoelace_data
 WHERE 0 <> shoelace_data.sl_avail
   AND shoelace_data.sl_color = 'black';
```

这棵查询树必然会插入三条新的日志记录。这完全正确。

到这里就能看出，为什么原始查询树最后执行至关重要。如果先执行UPDATE，那么所有行都已经被设为零，记日志的INSERT就找不到任何满足0 <> shoelace_data.sl_avail的行了。

37.3.2. 与视图的协作

为了防止有人像前面提到的那样对视图关系执行INSERT、UPDATE或DELETE，
一种简单的办法是让那些查询树直接被丢弃。因此我们可以创建如下规则：

```
CREATE RULE shoe_ins_protect AS ON INSERT TO shoe
    DO INSTEAD NOTHING;
CREATE RULE shoe_upd_protect AS ON UPDATE TO shoe
    DO INSTEAD NOTHING;
CREATE RULE shoe_del_protect AS ON DELETE TO shoe
    DO INSTEAD NOTHING;
```

如果现在某人尝试对视图关系shoe执行这些操作，规则系统就会应用这些规则。
由于这些规则没有动作而且是INSTEAD，生成的查询树列表将为空；
整个查询也就什么都不会做，因为规则系统处理完后已经没有任何内容可供优化或执行。

另一种更完善的做法，是创建一些规则，
把查询树重写成在真实表上执行正确操作的查询树。要在视图shoelace上做到这一点，
我们创建下列规则：

```
CREATE RULE shoelace_ins AS ON INSERT TO shoelace
    DO INSTEAD
    INSERT INTO shoelace_data VALUES (
           NEW.sl_name,
           NEW.sl_avail,
           NEW.sl_color,
           NEW.sl_len,
           NEW.sl_unit
    );

CREATE RULE shoelace_upd AS ON UPDATE TO shoelace
    DO INSTEAD
    UPDATE shoelace_data
       SET sl_name = NEW.sl_name,
           sl_avail = NEW.sl_avail,
           sl_color = NEW.sl_color,
           sl_len = NEW.sl_len,
           sl_unit = NEW.sl_unit
     WHERE sl_name = OLD.sl_name;

CREATE RULE shoelace_del AS ON DELETE TO shoelace
    DO INSTEAD
    DELETE FROM shoelace_data
     WHERE sl_name = OLD.sl_name;
```

如果你想在视图上支持RETURNING查询，就需要让规则包含计算视图行的RETURNING子句。
对于单表视图，这通常很简单；但对于像shoelace这样的连接视图，就会有点烦琐。
插入场景的一个示例如下：

```
CREATE RULE shoelace_ins AS ON INSERT TO shoelace
    DO INSTEAD
    INSERT INTO shoelace_data VALUES (
           NEW.sl_name,
           NEW.sl_avail,
           NEW.sl_color,
           NEW.sl_len,
           NEW.sl_unit
    )
    RETURNING
           shoelace_data.*,
           (SELECT shoelace_data.sl_len * u.un_fact
```

```
            FROM unit u WHERE shoelace_data.sl_unit = u.un_name);
```

请注意，这一条规则同时支持该视图上的INSERT和INSERT RETURNING查询；对于普通INSERT，RETURNING子句会被简单忽略。

现在假定商店每隔一段时间就会收到一包鞋带，随附一份很长的部件清单。
但你不想每次都手工更新 shoelace 视图。因此，我们建立两个小表：一个用于插入部件清单中的项目，另一个则采用一个特殊技巧。创建命令如下：

```
CREATE TABLE shoelace_arrive (
    arr_name    text,
    arr_quant   integer
);

CREATE TABLE shoelace_ok (
    ok_name     text,
    ok_quant    integer
);

CREATE RULE shoelace_ok_ins AS ON INSERT TO shoelace_ok
    DO INSTEAD
    UPDATE shoelace
       SET sl_avail = sl_avail + NEW.ok_quant
     WHERE sl_name = NEW.ok_name;
```

现在可以将部件清单中的数据填入 shoelace_arrive 表：

```
SELECT * FROM shoelace_arrive;

 arr_name | arr_quant
----------+-----------
 sl3      |        10
 sl6      |        20
 sl8      |        20
(3 rows)
```

快速查看一下当前数据：

```
SELECT * FROM shoelace;

 sl_name  | sl_avail | sl_color | sl_len | sl_unit | sl_len_cm
----------+----------+----------+--------+---------+-----------
 sl1      |        5 | black    |     80 | cm      |        80
 sl2      |        6 | black    |    100 | cm      |       100
 sl7      |        6 | brown    |     60 | cm      |        60
 sl3      |        0 | black    |     35 | inch    |      88.9
 sl4      |        8 | black    |     40 | inch    |     101.6
 sl8      |        1 | brown    |     40 | inch    |     101.6
 sl5      |        4 | brown    |      1 | m       |       100
 sl6      |        0 | brown    |    0.9 | m       |        90
(8 rows)
```

现在把到货的鞋带入库：

```
INSERT INTO shoelace_ok SELECT * FROM shoelace_arrive;
```

然后检查结果：

```
SELECT * FROM shoelace ORDER BY sl_name;
```

```
 sl_name  | sl_avail | sl_color | sl_len | sl_unit | sl_len_cm
----------+----------+----------+--------+---------+-----------
 sl1      |        5 | black    |     80 | cm      |        80
 sl2      |        6 | black    |    100 | cm      |       100
 sl7      |        6 | brown    |     60 | cm      |        60
 sl4      |        8 | black    |     40 | inch    |     101.6
 sl3      |       10 | black    |     35 | inch    |      88.9
 sl8      |       21 | brown    |     40 | inch    |     101.6
 sl5      |        4 | brown    |      1 | m       |       100
 sl6      |       20 | brown    |    0.9 | m       |        90
(8 rows)

SELECT * FROM shoelace_log;

 sl_name | sl_avail | log_who| log_when
---------+----------+--------+----------------------------------
 sl7     |        6 | Al     | Tue Oct 20 19:14:45 1998 MET DST
 sl3     |       10 | Al     | Tue Oct 20 19:25:16 1998 MET DST
 sl6     |       20 | Al     | Tue Oct 20 19:25:16 1998 MET DST
 sl8     |       21 | Al     | Tue Oct 20 19:25:16 1998 MET DST
(4 rows)
```

从一个INSERT ... SELECT到这些结果，中间要经历相当长的一段过程。本章最后再来描述这一查询树转换的细节。首先，看一下解析器的输出：

```
INSERT INTO shoelace_ok
SELECT shoelace_arrive.arr_name, shoelace_arrive.arr_quant
  FROM shoelace_arrive shoelace_arrive, shoelace_ok shoelace_ok;
```

现在应用第一条规则shoelace_ok_ins，它会把这一输出转换成：

```
UPDATE shoelace
   SET sl_avail = shoelace.sl_avail + shoelace_arrive.arr_quant
  FROM shoelace_arrive shoelace_arrive, shoelace_ok shoelace_ok,
       shoelace_ok old, shoelace_ok new,
       shoelace shoelace
 WHERE shoelace.sl_name = shoelace_arrive.arr_name;
```

同时会丢弃针对shoelace_ok的原始INSERT。这个重写后的查询会再次交给规则系统，而接下来应用的规则shoelace_upd会生成：

```
UPDATE shoelace_data
   SET sl_name = shoelace.sl_name,
       sl_avail = shoelace.sl_avail + shoelace_arrive.arr_quant,
       sl_color = shoelace.sl_color,
       sl_len = shoelace.sl_len,
       sl_unit = shoelace.sl_unit
  FROM shoelace_arrive shoelace_arrive, shoelace_ok shoelace_ok,
       shoelace_ok old, shoelace_ok new,
       shoelace shoelace, shoelace old,
       shoelace new, shoelace_data shoelace_data
 WHERE shoelace.sl_name = shoelace_arrive.arr_name
   AND shoelace_data.sl_name = shoelace.sl_name;
```

这同样是一条INSTEAD规则，因此前一个查询树会被丢弃。注意，这个查询仍然使用视图shoelace。但规则系统尚未完成，因此会继续在其上应用_RETURN规则，于是得到：

```
UPDATE shoelace_data
   SET sl_name = s.sl_name,
```

```
       sl_avail = s.sl_avail + shoelace_arrive.arr_quant,
       sl_color = s.sl_color,
       sl_len = s.sl_len,
       sl_unit = s.sl_unit
  FROM shoelace_arrive shoelace_arrive, shoelace_ok shoelace_ok,
       shoelace_ok old, shoelace_ok new,
       shoelace shoelace, shoelace old,
       shoelace new, shoelace_data shoelace_data,
       shoelace old, shoelace new,
       shoelace_data s, unit u
 WHERE s.sl_name = shoelace_arrive.arr_name
   AND shoelace_data.sl_name = s.sl_name;
```

最后，规则log_shoelace被应用，生成额外的查询树：

```
INSERT INTO shoelace_log
SELECT s.sl_name,
       s.sl_avail + shoelace_arrive.arr_quant,
       current_user,
       current_timestamp
  FROM shoelace_arrive shoelace_arrive, shoelace_ok shoelace_ok,
       shoelace_ok old, shoelace_ok new,
       shoelace shoelace, shoelace old,
       shoelace new, shoelace_data shoelace_data,
       shoelace old, shoelace new,
       shoelace_data s, unit u,
       shoelace_data old, shoelace_data new
       shoelace_log shoelace_log
 WHERE s.sl_name = shoelace_arrive.arr_name
   AND shoelace_data.sl_name = s.sl_name
   AND (s.sl_avail + shoelace_arrive.arr_quant) <> s.sl_avail;
```

到这里，规则系统已经没有更多规则可用，并返回生成的查询树。

因此，我们最终得到两棵查询树，它们等效于以下SQL语句：

```
INSERT INTO shoelace_log
SELECT s.sl_name,
       s.sl_avail + shoelace_arrive.arr_quant,
       current_user,
       current_timestamp
  FROM shoelace_arrive shoelace_arrive, shoelace_data shoelace_
data,
       shoelace_data s
 WHERE s.sl_name = shoelace_arrive.arr_name
   AND shoelace_data.sl_name = s.sl_name
   AND s.sl_avail + shoelace_arrive.arr_quant <> s.sl_avail;

UPDATE shoelace_data
   SET sl_avail = shoelace_data.sl_avail + shoelace_arrive.arr_
quant
  FROM shoelace_arrive shoelace_arrive,
       shoelace_data shoelace_data,
       shoelace_data s
 WHERE s.sl_name = shoelace_arrive.sl_name
   AND shoelace_data.sl_name = s.sl_name;
```

结果是：来自一个关系的数据被插入到另一个关系中，
这个插入被改写为对第三个关系的更新，

再被改写为对第四个关系的更新外加在第五个关系中记录该更新，
最后整个过程被化简为两个查询。

这里有个稍显难看的小细节。观察这两个查询会发现，shoelace_data
关系在范围表中出现了两次，而实际上完全可以缩减成一次。规划器不会处理这一点，
因此规则系统为INSERT输出的执行计划会是

```
Nested Loop
  ->  Merge Join
        ->  Seq Scan
              ->  Sort
                    ->  Seq Scan on s
        ->  Seq Scan
              ->  Sort
                    ->  Seq Scan on shoelace_arrive
  ->  Seq Scan on shoelace_data
```

而省略那个额外的范围表项则会得到

```
Merge Join
  ->  Seq Scan
        ->  Sort
              ->  Seq Scan on s
  ->  Seq Scan
        ->  Sort
              ->  Seq Scan on shoelace_arrive
```

这会在日志表中生成完全相同的项。因此，规则系统导致了对shoelace_data
表的一次完全不必要的额外扫描。而在UPDATE中，同样的冗余扫描还会再发生一次。不过，
能让这一切总体上工作起来，已经是一项相当艰巨的工作。

现在我们来做一个关于PostgreSQL规则系统及其威力的最后演示。
假设你向数据库中加入了一些颜色特别的鞋带：

```
INSERT INTO shoelace VALUES ('sl9', 0, 'pink', 35.0, 'inch', 0.0);
INSERT INTO shoelace VALUES ('sl10', 1000, 'magenta', 40.0, 'inch',
 0.0);
```

我们想建立一个视图，用来检查哪些shoelace项在颜色上与任何鞋子都不匹配。
这个视图是：

```
CREATE VIEW shoelace_mismatch AS
    SELECT * FROM shoelace WHERE NOT EXISTS
        (SELECT shoename FROM shoe WHERE slcolor = sl_color);
```

它的输出是：

```
SELECT * FROM shoelace_mismatch;

 sl_name | sl_avail | sl_color | sl_len | sl_unit | sl_len_cm
---------+----------+----------+--------+---------+-----------
 sl9     |        0 | pink     |     35 | inch    |      88.9
 sl10    |     1000 | magenta  |     40 | inch    |     101.6
```

现在希望将没有库存的不匹配鞋带从数据库中删除。为了给 PostgreSQL 增加一点难度，我们
不直接删除，而是再创建一个视图：

```
CREATE VIEW shoelace_can_delete AS
    SELECT * FROM shoelace_mismatch WHERE sl_avail = 0;
```

然后这样执行：

```
DELETE FROM shoelace WHERE EXISTS
    (SELECT * FROM shoelace_can_delete
             WHERE sl_name = shoelace.sl_name);
```

Voilà:

```
SELECT * FROM shoelace;

 sl_name | sl_avail | sl_color | sl_len | sl_unit | sl_len_cm
---------+----------+----------+--------+---------+-----------
 sl1     |        5 | black    |     80 | cm      |        80
 sl2     |        6 | black    |    100 | cm      |       100
 sl7     |        6 | brown    |     60 | cm      |        60
 sl4     |        8 | black    |     40 | inch    |     101.6
 sl3     |       10 | black    |     35 | inch    |      88.9
 sl8     |       21 | brown    |     40 | inch    |     101.6
 sl10    |     1000 | magenta  |     40 | inch    |     101.6
 sl5     |        4 | brown    |      1 | m       |       100
 sl6     |       20 | brown    |    0.9 | m       |        90
(9 rows)
```

作用于一个视图上的DELETE，其子查询条件总共使用了四个嵌套/连接的视图，
其中一个视图自身又带有一个包含视图的子查询条件，并且还用到了计算得到的视图列，
最终仍会被重写成单棵查询树，从真正的表中删除所请求的数据。

在现实世界中，大概只有很少的场景会需要这样的构造。但知道它确实能工作，
总归让人安心。

37.4. 规则和权限

由于PostgreSQL规则系统会重写查询，因此可能访问原始查询中并未使用的其他表或视图。
使用更新规则时，这甚至可能包括对表的写访问。

重写规则没有单独的所有者。关系（表或视图）
的所有者会自动成为为其定义的重写规则的所有者。
PostgreSQL规则系统改变了默认访问控制系统的行为。
由于规则而使用的关系都会根据规则所有者的权限进行检查，而不是调用规则的用户。
这意味着用户只需要对其查询中明确命名的表/视图具有所需的权限。

例如，某个用户有一份电话号码列表，其中一部分是私人的，另一部分对办公室秘书有用。
该用户可以这样构造：

```
CREATE TABLE phone_data (person text, phone text, private boolean);
CREATE VIEW phone_number AS
    SELECT person, CASE WHEN NOT private THEN phone END AS phone
    FROM phone_data;
GRANT SELECT ON phone_number TO assistant;
```

除了该用户本人（以及数据库超级用户）之外，没有人可以访问phone_data表。
但由于GRANT的存在，秘书可以对phone_number视图执行SELECT。
规则系统会把对phone_number的SELECT重写成对phone_data的SELECT。
由于该用户是phone_number的所有者，因此也是规则的所有者，对phone_data
的读访问会按照该用户的权限进行检查，于是查询被允许。同时，对phone_number
本身的访问检查仍会执行，但这是针对调用用户进行的，因此除了用户本人和秘书之外，
没有其他人能使用它。

权限是逐条规则检查的。因此，目前秘书是唯一能看到公开电话号码的人。
但秘书还可以再建立一个视图，并把该视图授予公众访问。这样，
任何人都可以通过秘书的视图看到phone_number中的数据。

秘书做不到的是创建一个直接访问phone_data的视图。（其实秘书可以创建，
但它不会起作用，因为每次访问都会在权限检查时被拒绝。）而且，
一旦用户发现秘书开放了其phone_number视图，用户就可以撤销秘书的访问权限。
那样一来，对秘书视图的任何访问都会立即失败。

这种逐条规则的检查看起来似乎像是一个安全漏洞，但实际上并不是。
因为即使不这样工作，秘书也可以建立一个与phone_number具有相同列的表，
每天把数据复制进去。那样一来，这些就是秘书自己的数据，
秘书同样可以把访问权限授予任何人。GRANT的含义本来就是“我信任你”。
如果某个你信任的人做了上面的事，那么该重新考虑这份信任，并使用REVOKE了。

需要注意的是，虽然视图可以用前文展示的技术来隐藏某些列的内容，但除非设置了
security_barrier标志，否则它们不能被用来可靠地隐藏不可见行中的数据。例如，
下面这个视图就是不安全的：

```
CREATE VIEW phone_number AS
    SELECT person, phone FROM phone_data WHERE phone NOT LIKE
 '412%';
```

这个视图看起来似乎很安全，因为规则系统会把对phone_number的任何SELECT
重写为对phone_data的SELECT，并附加条件，要求只处理phone不以 412 开头的项。但
是，如果用户可以创建自己的函数，就不难让规划器在NOT LIKE表达式之前先执行用户定义
函数。例如：

```
CREATE FUNCTION tricky(text, text) RETURNS bool AS $$
BEGIN
    RAISE NOTICE '% => %', $1, $2;
    RETURN true;
END;
$$ LANGUAGE plpgsql COST 0.0000000000000000000001;

SELECT * FROM phone_number WHERE tricky(person, phone);
```

phone_data表中的每个人和电话号码都会被打印为一条NOTICE，
因为规划器会选择先执行廉价的tricky函数，再执行更昂贵的NOT LIKE。即使禁止用户定
义新函数，内置函数也可以用于类似攻击。（例如，
大多数类型转换函数都会在其错误消息中包含输入值。）

类似的考虑也适用于更新规则。在上一节的示例中，示例数据库中那些表的所有者可以把
shoelace视图上的SELECT、INSERT、UPDATE和DELETE权限授予其他用户，但对
shoelace_log只授予SELECT权限。写日志记录的规则动作仍会成功执行，
因此其他用户可以看到日志记录。但他们不能伪造记录，也不能操纵或删除已有记录。
在这个例子里，不存在通过说服规划器改变操作顺序来破坏规则的可能性，因为唯一引用
shoelace_log的规则是一条无条件的INSERT。在更复杂的场景中，情况未必如此。

当视图需要提供行级安全时，应为该视图应用 security_barrier 属性。这可以防止在视
图完成其工作之前，将行中的值传给恶意选择的函数和操作符。例如，
如果上面展示的视图按以下方式创建，就会是安全的：

```
CREATE VIEW phone_number WITH (security_barrier) AS
    SELECT person, phone FROM phone_data WHERE phone NOT LIKE
 '412%';
```

使用 security_barrier 创建的视图，其性能可能远低于未使用此选项的视图。通常无法
避免这一点：如果最快的执行计划可能危及安全，就必须拒绝它。因此，
此选项默认不启用。

在处理没有副作用的函数时，查询规划器有更大的灵活性。这类函数称为LEAKPROOF，
其中包括许多简单且常用的操作符，例如大量等值操作符。

规划器可以安全地允许这类函数在查询执行过程中的任意位置求值，
因为即便把它们应用到用户不可见的行上，也不会泄露这些不可见行的信息。相反，
那些可能根据参数值抛出错误的函数（例如在发生溢出或除零时抛错的函数）
就不具备防泄漏性；如果在安全视图的行过滤之前对其求值，
就可能泄露关于不可见行的重要信息。

需要理解的一点是，即使一个视图是用`security_barrier`选项创建的，
它的安全性也只限于这样一种意义：不可见元组的内容不会被传递给可能不安全的函数。
用户仍然可能通过其他方式推断不可见数据；例如，他们可以使用`EXPLAIN`查看查询计划，
或者测量针对该视图执行查询所需的时间。
恶意攻击者或许能够推断出不可见数据的大致数量，
甚至获得有关数据分布或高频值的一些信息（因为这些因素可能影响计划运行时间，
甚至影响计划的选择，因为它们同样反映在优化器统计信息中）。如果这类“隐通道”
攻击值得担心，那么向任何人授予这类数据的访问权限本身就可能是不明智的。

37.5. 规则和命令状态

PostgreSQL服务器会为收到的每条命令返回一个命令状态字符串，例如`INSERT 149592 1`。在没有规则参与时，这很简单；但当查询被规则重写时会发生什么呢？

规则对命令状态的影响如下：

- 如果该查询没有无条件的`INSTEAD`规则，那么原始给出的查询就会照常执行，
  并返回它自身的命令状态。（但请注意，如果存在任何带条件的`INSTEAD`规则，
  那么这些规则条件的取反会被加到原始查询上。这可能会减少它处理的行数，
  从而影响报告的状态。）

- 如果该查询存在任意无条件的`INSTEAD`规则，那么原始查询将完全不会执行。
  在这种情况下，服务器会返回由`INSTEAD`规则（带条件或不带条件）插入的、
  最后一条且与原始查询具有相同命令类型（`INSERT`、`UPDATE`或`DELETE`）
  的查询的命令状态。如果没有任何规则添加出满足这些条件的查询，
  那么返回的命令状态会显示原始查询类型，并且把行计数和 OID 字段都置零。

（这套机制是在 PostgreSQL 7.3 中确立的。在那之前的版本中，存在规则时命令状态可能会显示不同的结果。）

在第二种情况下，程序员可以通过为所需的`INSTEAD`
规则指定一个在活动规则中按字母顺序排在最后的规则名，来确保由它设置命令状态，
因为它会最后被应用。

37.6. 规则与触发器

许多能够用触发器完成的事情，同样也可以用PostgreSQL规则系统实现。
不能用规则实现的内容之一是某些约束，尤其是外键。
你可以放置一条带条件的规则：当某列的值没有出现在另一张表中时，把命令重写成
`NOTHING`。但那样做会悄无声息地丢弃数据，这并不是好主意。如果需要检查值的有效性，
并在值无效时生成错误消息，就必须使用触发器。

在本章中，我们重点讨论了用规则来更新视图。本章中所有更新规则的示例，
也都可以用视图上的`INSTEAD OF`触发器来实现。编写这类触发器通常比编写规则更容易，尤其是在需要用复杂逻辑执行更新时。

对于两者都能实现的事情，哪一种更好取决于数据库的使用方式。
触发器会对每个受影响的行触发一次，而规则会修改查询，或生成额外的查询。因此，
如果一条语句会影响很多行，那么发出一条额外命令的规则往往会比触发器更快，
因为触发器必须为每一行调用一次，并反复判断该做什么。不过，
触发器方法在概念上要比规则方法简单得多，也更容易让新手写对。

下面通过一个例子说明，在一种具体情形中选择规则或触发器会有什么不同。
这里有两个表：

```
CREATE TABLE computer (
    hostname        text,    -- indexed
    manufacturer    text     -- indexed
);

CREATE TABLE software (
    software        text,    -- indexed
    hostname        text     -- indexed
);
```

两个表都有数千行，并且 `hostname` 上的索引都是唯一索引。规则或触发器应实现这样一个约束：从 `software` 中删除引用了已删除计算机的行。触发器会使用以下命令：

```
DELETE FROM software WHERE hostname = $1;
```

由于从 `computer` 中删除的每一行都会调用触发器，因此触发器可以为这条命令准备并保存执行计划，并通过参数传入 `hostname` 值。规则则会写成：

```
CREATE RULE computer_del AS ON DELETE TO computer
    DO DELETE FROM software WHERE hostname = OLD.hostname;
```

现在我们来看不同类型的删除。对于下面这种情况：

```
DELETE FROM computer WHERE hostname = 'mypc.local.net';
```

`computer`表会通过索引扫描（很快），
由触发器发出的命令也会使用索引扫描（同样很快）。规则产生的额外查询是：

```
DELETE FROM software WHERE computer.hostname = 'mypc.local.net'
                       AND software.hostname = computer.hostname;
```

由于已经建立了合适的索引，规划器将创建一个规划

```
Nestloop
  ->  Index Scan using comp_hostidx on computer
  ->  Index Scan using soft_hostidx on software
```

因此，触发器实现和规则实现之间的速度差异不会太大。

在下一个删除场景中，我们想去掉全部 2000 台`hostname`以`old`开头的计算机。有两种命令可以做到这件事。其中一种是：

```
DELETE FROM computer WHERE hostname >= 'old'
                       AND hostname <  'ole'
```

规则添加出来的命令将是：

```
DELETE FROM software WHERE computer.hostname >= 'old' AND computer.
hostname < 'ole'
                       AND software.hostname = computer.hostname;
```

其计划为：

```
Hash Join
  ->  Seq Scan on software
  ->  Hash
    ->  Index Scan using comp_hostidx on computer
```

另一个可能的命令是：

```
DELETE FROM computer WHERE hostname ~ '^old';
```

它会让规则所添加的命令得到下面这个执行计划：

```
Nestloop
  ->  Index Scan using comp_hostidx on computer
  ->  Index Scan using soft_hostidx on software
```

这说明，当多个条件表达式通过AND组合在一起时，规划器并没有意识到，对computer中hostname的限制同样也可以用于software上的索引扫描，而在该命令的正则表达式版本里它却能做到。触发器会对需要删除的 2000 台旧计算机中的每一台各调用一次，这意味着对computer做一次索引扫描，并对software做 2000 次索引扫描。规则实现则用两条使用索引的命令完成这件事。不过，在顺序扫描这种情况下，规则是否仍然更快，还取决于software表的总体大小。即使所有索引块很快都会进入缓存，通过 SPI 管理器执行来自触发器的 2000 条命令仍然要耗费一些时间。

我们要看的最后一个命令是：

```
DELETE FROM computer WHERE manufacturer = 'bim';
```

同样，这也可能导致从computer中删除很多行。因此，触发器同样会通过执行器运行很多条命令。规则生成的命令则是：

```
DELETE FROM software WHERE computer.manufacturer = 'bim'
                       AND software.hostname = computer.hostname;
```

这个命令的计划又将是在两个索引扫描上的嵌套循环，只不过使用了computer上的另一个索引：

```
Nestloop
  ->  Index Scan using comp_manufidx on computer
  ->  Index Scan using soft_hostidx on software
```

在上述任何一种情况下，规则系统产生的额外命令都或多或少与该命令影响的行数无关。

总结来说，只有当规则动作导致了规模很大且条件很差的连接，而规划器又无能为力时，规则才会明显慢于触发器。

第 38 章 过程语言

PostgreSQL允许使用 SQL 和 C 之外的其他语言编写用户定义的函数。这些语言统称为过程语言（PL）。对于用过程语言编写的函数，
数据库服务器并不内置关于如何解释函数源文本的知识。相反，
这项任务会交给一个了解该语言细节的专门的调用处理器。
该调用处理器既可以自行完成解析、语法分析、执行等全部工作，也可以在
PostgreSQL与某种现有编程语言实现之间充当“粘合剂”。与其他任何 C 函数一样，调用处理器本身也是一个被编译进共享对象并按需装载的 C 语言函数。

标准PostgreSQL发行版当前提供四种过程语言：PL/pgSQL（第 39 章）、PL/Tcl（第 40 章）、PL/Perl（第 41 章）以及 PL/Python（第 42 章）。另有一些可用的过程语言并未包含在核心发行版中。附录 H提供了查找它们的信息。此外，用户还可以自行定义其他语言；开发新过程语言的基础知识见第 49 章。

38.1. 安装过程语言

过程语言必须在每个要使用它的数据库中“安装”。不过，安装在数据库`template1`
中的过程语言会自动在随后创建的所有数据库中可用，因为它们在`template1`中的条目会由`CREATE DATABASE`复制。因此，数据库管理员可以决定哪些数据库提供哪些语言，并且如果愿意，也可以让某些语言默认可用。

对于标准发行版附带的语言，只需执行`CREATE EXTENSION` *`language_name`*，即可将该语言安装到当前数据库中。此外，也可以从 shell 命令行使用程序createlang来完成这项工作。例如，要把语言 PL/Perl 安装到数据库 `template1`，可以使用：

```
createlang plperl template1
```

下文所述的手工过程只建议用于安装那些尚未打包为扩展的语言。

手工安装过程语言

在数据库中安装过程语言需要五个步骤，且必须由数据库超级用户执行。多数情况下，
所需的 SQL 命令都应打包成某个“扩展”的安装脚本，这样就可以用`CREATE EXTENSION`来执行它们。

1. 该语言调用处理器的共享对象必须先编译好并安装到合适的库目录中。
   这与构建和安装含有普通用户定义 C 函数的模块的方法相同；见第 35.9.6 节。该语言调用处理器往往还会依赖一个提供实际编程语言引擎的外部库；如果是这样，该库也必须安装。

2. 必须用如下命令声明该调用处理器：

   ```
   CREATE FUNCTION handler_function_name()
       RETURNS language_handler
       AS 'path-to-shared-object'
       LANGUAGE C;
   ```

   特殊返回类型`language_handler`会告诉数据库系统，该函数返回的不是某种已定义的SQL数据类型，并且不能在 SQL语句中直接使用。

3. (可选的) 可选地，语言调用处理器可以提供一个“内联”处理器函数，用于执行用该语言编写的匿名代码块（DO命令）。如果该语言提供了内联处理器函数，
   可用类似下面的命令声明它：

   ```
   CREATE FUNCTION inline_function_name(internal)
       RETURNS void
       AS 'path-to-shared-object'
   ```

```
    LANGUAGE C;
```

4. (可选的) 可选地，语言调用处理器可以提供一个“验证器”函数，用于在不实际执行的情况下检查函数定义是否正确。如果存在验证器函数，CREATE FUNCTION就会调用它。如果该语言提供了验证器函数，可用类似下面的命令声明它：

```
CREATE FUNCTION validator_function_name(oid)
    RETURNS void
    AS 'path-to-shared-object'
    LANGUAGE C STRICT;
```

5. 最后，必须用如下命令声明该 PL：

```
CREATE [TRUSTED] [PROCEDURAL] LANGUAGE language-name
    HANDLER handler_function_name
    [INLINE inline_function_name]
    [VALIDATOR validator_function_name] ;
```

可选关键字TRUSTED表示，该语言不会授予用户原本不具备的数据访问能力。
受信任的语言是为普通数据库用户（即没有超级用户权限的用户）设计的，
并允许他们安全地创建函数和触发器过程。由于 PL 函数是在数据库服务器内部执行的，
因此TRUSTED标记只应赋予那些不允许访问数据库服务器内部或文件系统的语言。
语言PL/pgSQL，PL/Tcl以及PL/Perl被认为是受信任的；而语言PL/TclU，
PL/PerlU以及PL/PythonU旨在提供无限制的功能，因此不应被标记为受信任的。

例 38.1展示了以 PL/Perl为例时，手工安装过程是如何进行的。

例 38.1. 手工安装PL/Perl

下面的命令告诉数据库服务器到哪里查找 PL/Perl语言调用处理器函数的共享对象：

```
CREATE FUNCTION plperl_call_handler() RETURNS language_handler AS
    '$libdir/plperl' LANGUAGE C;
```

PL/Perl具有内联处理器函数和验证器函数，因此我们也声明它们：

```
CREATE FUNCTION plperl_inline_handler(internal) RETURNS void AS
    '$libdir/plperl' LANGUAGE C;

CREATE FUNCTION plperl_validator(oid) RETURNS void AS
    '$libdir/plperl' LANGUAGE C STRICT;
```

下面的命令：

```
CREATE TRUSTED PROCEDURAL LANGUAGE plperl
    HANDLER plperl_call_handler
    INLINE plperl_inline_handler
    VALIDATOR plperl_validator;
```

随后指定应为以下函数和触发器过程调用前面声明的函数：其语言属性为plperl。

在默认的PostgreSQL安装中，PL/pgSQL语言的调用处理器会被构建并安装到“库”目录中；此外，PL/pgSQL语言本身也安装在所有数据库中。如果在构建时配置了Tcl支持，那么 PL/Tcl和PL/TclU 的调用处理器会被构建并安装到该库目录中，但这两种语言本身默认并不会安装到任何数据库中。同样，如果在构建时配置了 Perl 支持，就会构建并安装 PL/Perl和PL/PerlU 的调用处理器；如果在构建时配置了 Python 支持，则会安装 PL/PythonU的调用处理器，但这些语言默认也不会安装到任何数据库中。

第 39 章 PL/pgSQL – SQL 过程语言

39.1. 概述

PL/pgSQL 是一种用于 PostgreSQL 数据库系统的可载入过程语言。PL/pgSQL 的设计目标，是创建一种具备以下特性的可载入过程语言：

- 可用于创建函数和触发器函数，
- 为SQL语言增加控制结构，
- 可以执行复杂计算，
- 继承所有用户定义的类型、函数和操作符，
- 可定义为受服务器信任的语言，
- 便于使用。

用 PL/pgSQL 创建的函数，可以用于任何可以使用内置函数的地方。例如，可以创建带有复杂条件逻辑的计算函数，随后用它们定义操作符，或者把它们用于索引表达式。

在 PostgreSQL 9.0 及更高版本中，PL/pgSQL 默认安装。不过它仍然是一个可载入模块，因此对安全性特别敏感的管理员也可以选择移除它。

39.1.1. 使用PL/pgSQL的优点

SQL 是 PostgreSQL 以及大多数其他关系数据库使用的查询语言。它具有可移植性，也容易学习。但每条 SQL 语句都必须由数据库服务器单独执行。

这意味着客户端应用必须把每个查询发送给数据库服务器，等待其处理完成，
接收并处理结果，做一些计算，然后再向服务器发送更多查询。
这一切都会产生进程间通信开销；如果客户端与数据库服务器不在同一台机器上，
还会额外产生网络开销。

借助 PL/pgSQL，你可以把一段计算逻辑和一系列查询放在数据库服务器内部执行。这样既具备了过程语言的能力，也保留了 SQL 的易用性，同时还能显著减少客户端/服务器通信开销。

- 消除了客户端和服务器之间额外的往返通信
- 客户端不需要的中间结果不必在服务器与客户端之间编组或传输
- 可以避免多轮查询解析

与不使用存储函数的应用相比，这可能带来相当可观的性能提升。

此外，通过 PL/pgSQL 你可以使用 SQL 的全部数据类型、操作符和函数。

39.1.2. 支持的参数和结果数据类型

用 PL/pgSQL 编写的函数，可以接受服务器支持的任意标量或数组数据类型作为参数，也可以返回这些类型中的任意一种。
它们还可以接受或返回任何按名称指定的复合类型（行类型）。也可以把 PL/pgSQL 函数声明为返回 `record`，这表示结果是一种行类型，其列由调用查询中的指定内容决定，如第 7.2.1.4 节所述。

PL/pgSQL 函数还可以通过使用 `VARIADIC` 标记声明为接受可变数量的参数。这与 SQL 函数中的行为完全相同，详见第 35.4.5 节。

PL/pgSQL 函数也可以声明为接受和返回多态类型 anyelement、anyarray、anynonarray、anyenum 和 anyrange。多态函数处理的实际数据类型可以在不同调用之间变化，如第 35.2.5 节所述。示例见第 39.3.1 节。

PL/pgSQL 函数还可以声明为返回任意可单独返回的数据类型的“集合”（或表）。这类函数可以通过对结果集中的每个目标元素执行 RETURN NEXT 生成输出，也可以通过 RETURN QUERY 输出某个查询的结果。

最后，如果某个 PL/pgSQL 函数没有有意义的返回值，可以将其声明为返回 void。

PL/pgSQL 函数也可以声明为使用输出参数，而不是显式指定返回类型。这并没有为该语言增加新的基本能力，但通常会更方便，尤其是在需要返回多个值时。RETURNS TABLE 记法也可以用来代替 RETURNS SETOF。

具体示例见第 39.3.1 节和第 39.6.1 节。

39.2. PL/pgSQL的结构

通过执行CREATE FUNCTION命令，可以把用 PL/pgSQL 编写的函数定义到服务器中。这类命令通常类似于：

```
CREATE FUNCTION somefunc(integer, text) RETURNS integer
AS 'function body text'
LANGUAGE plpgsql;
```

就 CREATE FUNCTION 而言，函数体只是一个字符串字面量。编写函数体时，通常最好使用美元引用（见第 4.1.2.4 节），而不是普通的单引号语法。如果不使用美元引用，函数体中的任何单引号或反斜线都必须通过双写进行转义。
本章几乎所有示例的函数体都使用了美元引用。

PL/pgSQL 是一种块结构语言。函数体的完整文本必须是一个块。块的定义如下：

```
[ <<label>> ]
[ DECLARE
    declarations ]
BEGIN
    statements
END [ label ];
```

块中的每个声明和每条语句都以分号结束。如上所示，嵌套在其他块中的块，必须在 END 之后写分号；但是结束整个函数体的最后一个 END 不需要分号。

> **提示**
>
> 一个常见错误是在 BEGIN 后面立刻写分号。这是不正确的，并会导致语法错误。

只有在你希望通过 EXIT 语句标识某个块，或者希望用块名限定块内声明的变量名时，才需要 *label*。如果在 END 之后写了标签，它必须与块开始处的标签一致。

所有关键字都不区分大小写。标识符除非用双引号括起，否则会像普通 SQL 命令中那样被隐式转换为小写。

PL/pgSQL 代码中的注释与普通 SQL 相同。双连字符（--）开始一段持续到行尾的注释。/* 开始一段块注释，它会持续到匹配的 */。块注释可以嵌套。

块的语句部分中的任意语句都可以是一个子块。子块可用于逻辑分组，
或者把变量局部化到一小组语句中。在子块存续期间，

子块中声明的变量会遮蔽外层块中的同名变量。不过，如果使用块标签限定外层变量名，仍然可以访问它们。例如：

```
CREATE FUNCTION somefunc() RETURNS integer AS $$
<< outerblock >>
DECLARE
    quantity integer := 30;
BEGIN
    RAISE NOTICE 'Quantity here is %', quantity;  -- Prints 30
    quantity := 50;
    --
    -- 创建一个子块
    --
    DECLARE
        quantity integer := 80;
    BEGIN
        RAISE NOTICE 'Quantity here is %', quantity;  -- Prints 80
        RAISE NOTICE 'Outer quantity here is %', outerblock.
quantity;  -- Prints 50
    END;

    RAISE NOTICE 'Quantity here is %', quantity;  -- Prints 50

    RETURN quantity;
END;
$$ LANGUAGE plpgsql;
```

> **注意**
>
> 实际上，任何 PL/pgSQL 函数体外面都包着一个隐藏的“外层块”。该块提供函数参数（如果有）的声明，以及一些诸如 FOUND 之类的特殊变量（见第 39.5.5 节）。这个外层块带有函数名标签，这意味着参数和特殊变量可以用函数名来限定。

需要特别注意，不要把BEGIN/END在PL/pgSQL中用于分组语句的用法，与用于事务控制的同名 SQL 命令混淆。PL/pgSQL中的BEGIN/END只用于分组，不会开始或结束事务。函数和触发器函数总是在外层查询建立的事务中执行 – 它们不能开始或提交该事务，否则就没有可供其执行的上下文。但是，包含EXCEPTION子句的块实际上会形成一个可回滚而不影响外层事务的子事务。详见第 39.6.6 节。

39.3. 声明

在一个块中使用的所有变量，都必须在该块的声明部分声明。（唯一的例外是：在整数范围上迭代的 FOR 循环变量会被自动声明为整数变量；同样，在游标结果上迭代的 FOR 循环变量会被自动声明为记录变量。）

PL/pgSQL 变量可以是任意 SQL 数据类型，例如 integer、varchar 和 char。

这里是变量声明的一些示例：

```
user_id integer;
quantity numeric(5);
url varchar;
myrow tablename%ROWTYPE;
myfield tablename.columnname%TYPE;
arow RECORD;
```

一个变量声明的一般语法是：

```
name [ CONSTANT ] type [ COLLATE collation_name ] [ NOT NULL ] [ {
 DEFAULT | := } expression ];
```

如果给定 DEFAULT 子句，它会指定进入该块时赋给该变量的初始值。如果没有给出 DEFAULT 子句，则变量会被初始化为 SQL 空值。CONSTANT 选项会阻止该变量在初始化之后再次被赋值，因此其值在整个块的持续期间保持不变。COLLATE 选项指定该变量使用的排序规则（见第 39.3.6 节）。如果指定了 NOT NULL，给该变量赋空值将导致运行时错误。所有声明为 NOT NULL 的变量都必须指定非空默认值。

变量的默认值会在每次进入该块时重新计算并赋给该变量，
而不是每次函数调用只计算一次。因此，例如把 now() 赋给一个 timestamp 类型变量，会使该变量得到当前函数调用时的时间，而不是函数预编译时的时间。

例如：

```
quantity integer DEFAULT 32;
url varchar := 'http://mysite.com';
user_id CONSTANT integer := 10;
```

39.3.1. 声明函数参数

传递给函数的参数使用标识符 $1、$2 等命名。也可以为这些 $*n* 形式的参数名声明别名，以提高可读性。之后既可以使用别名，也可以使用数字标识符来引用参数值。

有两种方式可以创建别名。推荐的方式是在 CREATE FUNCTION 命令中直接为参数命名。例如：

```
CREATE FUNCTION sales_tax(subtotal real) RETURNS real AS $$
BEGIN
    RETURN subtotal * 0.06;
END;
$$ LANGUAGE plpgsql;
```

另一种方式——这也是 PostgreSQL 8.0 之前唯一可用的方式——是使用声明语法显式声明别名。

```
name ALIAS FOR $n;
```

同样的示例，按这种写法如下：

```
CREATE FUNCTION sales_tax(real) RETURNS real AS $$
DECLARE
    subtotal ALIAS FOR $1;
BEGIN
    RETURN subtotal * 0.06;
END;
$$ LANGUAGE plpgsql;
```

> **注意**
>
> 这两个示例并不完全等价。在第一种情况下，subtotal 可以写成 sales_tax.subtotal 来引用；但在第二种情况下则不行。（如果我们给内层块附上一个标签，那么 subtotal 可以用那个标签来限定。）

更多一些示例：

```
CREATE FUNCTION instr(varchar, integer) RETURNS integer AS $$
DECLARE
    v_string ALIAS FOR $1;
```

```
    index ALIAS FOR $2;
BEGIN
    -- 这里是一些使用 v_string 和 index 的计算
END;
$$ LANGUAGE plpgsql;

CREATE FUNCTION concat_selected_fields(in_t sometablename) RETURNS
 text AS $$
BEGIN
    RETURN in_t.f1 || in_t.f3 || in_t.f5 || in_t.f7;
END;
$$ LANGUAGE plpgsql;
```

当 PL/pgSQL 函数声明了输出参数时，输出参数也会像普通输入参数一样获得 `$n` 名称和可选别名。输出参数本质上是一个初始值为 NULL 的变量，应在函数执行期间给它赋值。该参数的最终值就是返回值。例如，销售税的示例也可以这样写：

```
CREATE FUNCTION sales_tax(subtotal real, OUT tax real) AS $$
BEGIN
    tax := subtotal * 0.06;
END;
$$ LANGUAGE plpgsql;
```

注意这里省略了 `RETURNS real` – 当然也可以写上，但那只是冗余。

当需要返回多个值时，输出参数尤其有用。下面是一个简单示例：

```
CREATE FUNCTION sum_n_product(x int, y int, OUT sum int, OUT prod
 int) AS $$
BEGIN
    sum := x + y;
    prod := x * y;
END;
$$ LANGUAGE plpgsql;
```

如第 35.4.4 节所述，这实际上会为函数结果创建一个匿名记录类型。如果写了 `RETURNS` 子句，它必须是 `RETURNS record`。

声明 PL/pgSQL 函数的另一种方式是使用 `RETURNS TABLE`，例如：

```
CREATE FUNCTION extended_sales(p_itemno int)
RETURNS TABLE(quantity int, total numeric) AS $$
BEGIN
    RETURN QUERY SELECT s.quantity, s.quantity * s.price FROM sales
 AS s
                 WHERE s.itemno = p_itemno;
END;
$$ LANGUAGE plpgsql;
```

这与声明一个或多个 `OUT` 参数并指定 `RETURNS SETOF` *sometype* 完全等效。

当 PL/pgSQL 函数的返回类型被声明为多态类型（`anyelement`、`anyarray`、`anynonarray`、`anyenum` 或 `anyrange`）时，会创建一个特殊参数 `$0`。它的数据类型就是该函数的实际返回类型，由实际输入类型推导而来（见第 35.2.5 节）。这样，函数就能像第 39.3.3 节所示的那样访问自己的实际返回类型。`$0` 被初始化为空值，并且可以由函数修改，因此如果需要，可以用它来保存返回值，不过这并不是必须的。`$0` 也可以被赋予别名。例如，下面的函数适用于任何带有 + 操作符的数据类型：

```
CREATE FUNCTION add_three_values(v1 anyelement, v2 anyelement, v3
 anyelement)
```

```
RETURNS anyelement AS $$
DECLARE
    result ALIAS FOR $0;
BEGIN
    result := v1 + v2 + v3;
    RETURN result;
END;
$$ LANGUAGE plpgsql;
```

把一个或多个输出参数声明为多态类型，也可以达到同样的效果。在这种情况下，不使用特殊参数 `$0`，输出参数本身就承担相同作用。例如：

```
CREATE FUNCTION add_three_values(v1 anyelement, v2 anyelement, v3
 anyelement,
                                 OUT sum anyelement)
AS $$
BEGIN
    sum := v1 + v2 + v3;
END;
$$ LANGUAGE plpgsql;
```

39.3.2. ALIAS

newname ALIAS FOR *oldname*;

`ALIAS` 语法比上一节所展示的更一般化：你可以为任何变量声明别名，而不只是函数参数。它最主要的实际用途，是为那些名称预先固定的变量指定另一个名字，例如触发器函数中的 `NEW` 或 `OLD`。

示例：

```
DECLARE
  prior ALIAS FOR old;
  updated ALIAS FOR new;
```

由于 `ALIAS` 为同一个对象提供了两种命名方式，滥用它会让代码变得混乱。最好只把它用于改写那些预先固定的名称。

39.3.3. 复制类型

variable%TYPE

`%TYPE`提供变量或表列的数据类型。可以用它声明将保存数据库值的变量。例如，假设有一个名为`user_id`的列，位于`users`表中。要声明一个数据类型与`users.user_id`相同的变量，可以写：

```
user_id users.user_id%TYPE;
```

使用 `%TYPE` 的好处是，你不必知道所引用结构的实际数据类型；更重要的是，如果被引用项的数据类型将来发生变化（例如把 `user_id` 的类型从 `integer` 改成 `real`），你可能就不需要修改函数定义。

`%TYPE` 在多态函数中特别有价值，因为内部变量所需的数据类型可能在不同调用之间变化。可以把 `%TYPE` 应用到函数参数或结果占位符上，以创建合适的变量。

39.3.4. 行类型

name table_name%ROWTYPE;

name composite_type_name;

复合类型的变量称为行变量（或行类型变量）。
只要查询的列集合与该变量声明的类型相匹配，这种变量就可以保存 SELECT 或 FOR 查询结果中的整行。行值的各个字段可以使用通常的点号记法访问，例如 rowvar.field。

行变量既可以通过 *table_name*%ROWTYPE 记法声明为与现有表或视图的行具有相同类型，也可以通过给出某个复合类型的名称来声明。（由于每个表都有一个同名的关联复合类型，所以在 PostgreSQL 中实际上写不写 %ROWTYPE 并无区别；不过带 %ROWTYPE 的形式可移植性更好。）

函数参数也可以是复合类型（完整的表行）。在这种情况下，相应的标识符 $*n* 就是一个行变量，并且可以从中选取字段，例如 $1.user_id。

在行类型变量中，只能访问表行中用户定义的列，不能访问 OID 或其他系统列（因为该行可能来自视图）。对于char(*n*)这样的数据类型，
行类型中的字段会继承表字段的大小或精度。

下面是一个使用复合类型的示例。table1 和 table2 是已经存在的表，它们至少包含下面提到的字段：

```
CREATE FUNCTION merge_fields(t_row table1) RETURNS text AS $$
DECLARE
    t2_row table2%ROWTYPE;
BEGIN
    SELECT * INTO t2_row FROM table2 WHERE ... ;
    RETURN t_row.f1 || t2_row.f3 || t_row.f5 || t2_row.f7;
END;
$$ LANGUAGE plpgsql;

SELECT merge_fields(t.*) FROM table1 t WHERE ... ;
```

39.3.5. 记录类型

name RECORD;

记录变量与行类型变量类似，但没有预定义结构。它会在 SELECT 或 FOR 命令为其赋值时采用相应行的实际结构。记录变量的内部结构在每次被赋值时都可能变化。
其后果是：在记录变量第一次被赋值之前，它没有任何子结构，
任何试图访问其中字段的行为都会引发运行时错误。

注意，RECORD 并不是真正的数据类型，它只是一个占位符。还需要认识到，PL/pgSQL 函数被声明为返回 record，与记录变量并不是完全相同的概念，尽管这样的函数可能会用记录变量保存结果。这两种情况下，在编写函数时都不知道实际的行结构；但对于返回 record 的函数，实际结构会在解析调用查询时确定，而记录变量的行结构则可以在运行过程中随时变化。

39.3.6. PL/pgSQL变量的排序规则

当 PL/pgSQL 函数具有一个或多个支持排序规则的数据类型参数时，每次函数调用都会根据分配给实际参数的排序规则确定出一个排序规则，如第 22.2 节所述。如果该排序规则能成功确定出来（即参数之间的隐式排序规则没有冲突），
那么所有支持排序规则的参数都会被视为隐式带有该排序规则。
这会影响函数中那些受排序规则影响的操作。例如，考虑

```
CREATE FUNCTION less_than(a text, b text) RETURNS boolean AS $$
BEGIN
    RETURN a < b;
```

```
END;
$$ LANGUAGE plpgsql;

SELECT less_than(text_field_1, text_field_2) FROM table1;
SELECT less_than(text_field_1, text_field_2 COLLATE "C") FROM
 table1;
```

第一次调用 `less_than` 时，比较会使用 `text_field_1` 与 `text_field_2` 的共同排序规则；第二次则会使用 `C` 排序规则。

此外，确定出的排序规则也会被视为任何支持排序规则的数据类型局部变量的排序规则。因此，即使把这个函数写成下面这样，其行为也不会有任何不同：

```
CREATE FUNCTION less_than(a text, b text) RETURNS boolean AS $$
DECLARE
    local_a text := a;
    local_b text := b;
BEGIN
    RETURN local_a < local_b;
END;
$$ LANGUAGE plpgsql;
```

如果函数没有支持排序规则的数据类型的参数，或者无法为它们确定共同排序规则，那么参数和局部变量将使用其数据类型的默认排序规则（通常是数据库默认排序规则，但对于域类型变量也可能不同）。

通过在支持排序规则的数据类型局部变量的声明中加入 `COLLATE` 选项，可以为其指定不同的排序规则，例如

```
DECLARE
    local_a text COLLATE "en_US";
```

这个选项会覆盖按上述规则原本应赋给该变量的排序规则。

当然，如果某个函数希望在特定操作中强制使用特定排序规则，也可以在函数内部显式写出 `COLLATE` 子句。例如：

```
CREATE FUNCTION less_than_c(a text, b text) RETURNS boolean AS $$
BEGIN
    RETURN a < b COLLATE "C";
END;
$$ LANGUAGE plpgsql;
```

这会覆盖表达式中表列、参数或局部变量所关联的排序规则，就像在普通 SQL 命令中一样。

39.4. 表达式

PL/pgSQL 语句中使用的所有表达式都由服务器的主 SQL 执行器处理。例如，当你写出这样一条 PL/pgSQL 语句

```
IF expression THEN ...
```

PL/pgSQL 会通过把如下查询交给主 SQL 引擎来对表达式求值：

```
SELECT expression
```

在形成 `SELECT` 命令时，所有出现的 PL/pgSQL 变量名都会被替换为参数，详见第 39.10.1 节。这样，`SELECT` 的查询计划只需准备一次，随后就可以在变量值不同时重复使用。因此，首次使用表达式时，实际发生的事情本质上是执行一个 `PREPARE` 命令。例如，如果已经声明了两个整数变量 `x` 和 `y`，并写下

```
IF x < y THEN ...
```

背后发生的事情就等同于：

```
PREPARE statement_name(integer, integer) AS SELECT $1 < $2;
```

随后，对该预备语句调用 EXECUTE，以完成每次 IF 语句的执行，并把 PL/pgSQL 变量的当前值作为参数值提供。通常，这些细节对 PL/pgSQL 用户并不重要，但在诊断问题时了解它们很有帮助。更多信息见第 39.10.2 节。

39.5. 基本语句

在本节和后续各节中，我们描述PL/pgSQL能够明确理解的所有语句类型。
凡是不被识别为这些类型之一的内容，都被视为 SQL 命令并发送给主数据库引擎执行，详见第 39.5.2 节和第 39.5.3 节。

39.5.1. 赋值

向PL/pgSQL变量赋值的写法为：

```
variable := expression;
```

如前所述，这种语句中的表达式通过发送给主数据库引擎的 SQLSELECT命令求值。表达式必须产生单个值（如果变量是行变量或记录变量，也可以是行值）。
目标变量可以是简单变量（可以用块名限定）、行变量或记录变量的字段，
或者由简单变量或字段表示的数组中的元素。

如果该表达式的结果数据类型不匹配变量的数据类型，或者变量具有特定的大小/精度（例如 char(20)），结果值将由 PL/pgSQL解释器使用结果类型的输出函数和变量类型的输入函数隐式转换。注意如果结果值的字符串形式无法被输入函数所接受，
这可能会导致由输入函数产生的运行时错误。

例如：

```
tax := subtotal * 0.06;
my_record.user_id := 20;
```

39.5.2. 执行没有结果的命令

对于不返回行的 SQL 命令，例如没有RETURNING子句的INSERT，只需写出命令，就可以在PL/pgSQL函数中执行它。

命令文本中出现的任何PL/pgSQL变量名都被视为参数，
然后在运行时将变量的当前值作为参数值提供。这与前面描述的表达式处理完全相同；
详情见第 39.10.1 节。

以这种方式执行 SQL 命令时，PL/pgSQL 可能会缓存并重用该命令的执行计划，如第 39.10.2 节所述。

有时，对表达式或SELECT查询求值但丢弃结果会很有用，
例如调用有副作用但没有有用结果值的函数时。要在PL/pgSQL中这样做，可以使用PERFORM语句：

```
PERFORM query;
```

这会执行*query*并丢弃结果。编写*query*的方式与编写 SQLSELECT命令相同，只需把开头的关键字SELECT替换为PERFORM。对于WITH查询，使用PERFORM，然后用圆括号括住查询。

（此时查询只能返回一行。）PL/pgSQL变量会被替换到查询中，
方式与不返回结果的命令相同，计划也会以相同方式缓存。另外，特殊变量FOUND
在查询至少产生一行时设为真，没有产生行时设为假（见第 39.5.5 节）。

> **注意**
>
> 我们可能期望直接写SELECT能实现这个结果，但是当前唯一被接受的方式是PERFORM。一个能返回行的 SQL 命令（例如SELECT）将被当成一个错误拒绝，除非它像下一节中讨论的有一个INTO子句。

一个示例：

```
PERFORM create_mv('cs_session_page_requests_mv', my_query);
```

39.5.3. 执行返回单行结果的查询

产生单行（可能有多列）结果的 SQL 命令，其结果可以赋给记录变量、行类型变量或标量变量列表。方法是在基本 SQL 命令中添加INTO子句。例如：

```
SELECT select_expressions INTO [STRICT] target FROM ...;
INSERT ... RETURNING expressions INTO [STRICT] target;
UPDATE ... RETURNING expressions INTO [STRICT] target;
DELETE ... RETURNING expressions INTO [STRICT] target;
```

其中，*target*可以是记录变量、行变量，或逗号分隔的简单变量及记录/行字段列表。
PL/pgSQL变量会被替换到查询的其余部分，计划也会被缓存，
正如上面描述的不返回行的命令一样。这适用于SELECT，INSERT/UPDATE/DELETE
带有RETURNING的情况，以及返回行集结果的工具命令（例如EXPLAIN）。除了INTO
子句以外，SQL 命令的写法与在PL/pgSQL之外的写法相同。

> **提示**
>
> 注意带INTO的SELECT的这种解释和PostgreSQL常规的SELECT INTO命令有很大的不同，后者的INTO目标是一个新创建的表。
> 如果你想要在一个PL/pgSQL函数中从一个SELECT的结果创建一个表，请使用语法CREATE TABLE ... AS SELECT。

如果使用行变量或变量列表作为目标，
查询结果列的数量和数据类型必须与目标结构完全匹配，否则会发生运行时错误。
如果目标是记录变量，它会自动将自身配置为查询结果列的行类型。

INTO子句几乎可以出现在 SQL 命令中的任何位置。通常它被写成刚好在SELECT命令中的*select_expressions*列表之前或之后，或者在其他命令类型的命令最后。
我们推荐你遵循这种惯例，以防PL/pgSQL的解析器在未来的版本中变得更严格。

如果STRICT没有在INTO子句中指定，那么*target*会被设为查询返回的第一行；
如果查询没有返回行，则设为空值。（注意，“第一行”并没有明确定义，除非使用了ORDER BY。）第一行之后的所有结果行都会被丢弃。可以检查特殊的FOUND变量（见第 39.5.5 节），以确定是否返回了行：

```
SELECT * INTO myrec FROM emp WHERE empname = myname;
IF NOT FOUND THEN
    RAISE EXCEPTION 'employee % not found', myname;
END IF;
```

如果指定STRICT选项，查询必须恰好返回一行，否则会报告运行时错误：NO_DATA_FOUND（没有行）或TOO_MANY_ROWS（多于一行）。如果希望捕获错误，可以使用异常块，例如：

```
BEGIN
    SELECT * INTO STRICT myrec FROM emp WHERE empname = myname;
    EXCEPTION
        WHEN NO_DATA_FOUND THEN
            RAISE EXCEPTION 'employee % not found', myname;
        WHEN TOO_MANY_ROWS THEN
            RAISE EXCEPTION 'employee % not unique', myname;
END;
```

成功执行带有STRICT的命令总会将FOUND设为真。

对于带有RETURNING的INSERT/UPDATE/DELETE，即使没有指定STRICT，PL/pgSQL也会针对多于一个返回行的情况报告一个错误。这是因为没有类似于ORDER BY的选项可以用来决定应该返回哪个被影响的行。

> **注意**
>
> STRICT选项匹配 Oracle PL/SQL 的SELECT INTO和相关语句的行为。

对于需要处理 SQL 查询的多个结果行的情况，请参见第 39.6.4 节。

39.5.4. 执行动态命令

很多时候你将想要在PL/pgSQL函数中产生动态命令，
也就是每次执行中会涉及到不同表或不同数据类型的命令。
PL/pgSQL通常对于命令所做的缓存计划尝试（如第 39.10.2 节中讨论）在这种情境下无法工作。要处理这一类问题，提供了EXECUTE语句：

```
EXECUTE command-string [ INTO [STRICT] target ] [ USING expression
 [, ... ] ];
```

其中*command-string*是一个表达式，其求值结果（类型为 text）是包含待执行命令的字符串。可选的*target*是一个记录变量、一个行变量或者一个逗号分隔的简单变量以及记录/行字段的列表，该命令的结果将存储在其中。可选的USING
表达式提供要被插入到该命令中的值。

在计算得到的命令字符串中，不会做PL/pgSQL变量的替换。
任何所需的变量值必须在命令字符串被构造时被插入其中，
或者你可以使用下面描述的参数。

另外，通过 EXECUTE 执行的命令不会缓存计划。相反，每次运行该语句时都会重新准备命令。因此，可以在函数内动态创建命令字符串，以对不同的表和列执行操作。

INTO子句指定一个返回行的 SQL 命令的结果应该被赋值到哪里。如果提供了一个行变量或变量列表，它必须完全匹配查询命令结果的结构（当一个记录变量被提供时，
它会自动把它自己配置为匹配结果结构）。如果返回多个行，只有第一个行会被赋值给INTO变量。如果没有返回行，NULL 会被赋值给INTO变量。如果没有指定INTO子句，该查询结果会被抛弃。

如果给出了STRICT选项，除非该查询刚好产生一行，否则将会报告一个错误。

命令字符串可以使用参数值，它们在命令中用$1、$2等引用。这些符号引用在USING
子句中提供的值。

这种方法通常比把数据值作为文本插入命令字符串更可取：它避免了将该值转换为文本以及转换回来的运行时负荷，并且它更不容易被 SQL 注入攻击，因为不需要引用或转义。一个示例是：

```
EXECUTE 'SELECT count(*) FROM mytable WHERE inserted_by = $1 AND
 inserted <= $2'
   INTO c
   USING checked_user, checked_date;
```

注意，参数符号只能用于数据值 – 如果要使用动态确定的表名或列名，必须将其作为文本插入命令字符串。例如，如果前面的查询需要针对动态选择的表执行，可以这样写：

```
EXECUTE 'SELECT count(*) FROM '
    || tabname::regclass
    || ' WHERE inserted_by = $1 AND inserted <= $2'
   INTO c
   USING checked_user, checked_date;
```

参数符号的另一个限制是，它们只能用于SELECT，INSERT，UPDATE以及DELETE命令。在其他语句类型（统称为工具语句）中，即使只是数据值，也必须以文本形式插入。

带简单常量命令字符串和一些 USING 参数的 EXECUTE（如上面第一个示例那样），在功能上等价于直接在 PL/pgSQL 中写出该命令，并让 PL/pgSQL 变量的替换自动发生。重要的区别在于：EXECUTE 会在每次执行时重新规划命令，生成针对当前参数值的特定计划；而 PL/pgSQL 通常创建的是通用计划并将其缓存以供重用。在最佳计划强烈依赖参数值的情况下，EXECUTE 可能明显更快；而当计划对参数值不敏感时，重新规划就是浪费。

EXECUTE目前不支持SELECT INTO。但是可以执行一个纯的SELECT命令并且指定INTO作为EXECUTE本身的一部分。

注意

PL/pgSQL中的EXECUTE语句与EXECUTE这一由PostgreSQL服务器支持的 SQL 语句无关。服务器的EXECUTE
语句不能直接在PL/pgSQL函数中使用（并且也没有必要）。

例 39.1. 在动态查询中为值加引号

在使用动态命令时，经常需要处理单引号的转义。
我们推荐在函数体中使用美元引用来引用固定文本。（如果你有未使用美元引用的旧代码，请参阅第 39.11.1 节中的概述；在把这类代码转换成更合理的写法时，它会帮你省下一些工夫。）

将被插入到构造出的查询中的动态值需要被小心地处理，因为它们自身可能包含引号字符。
一个示例（这假设你对函数整体使用了美元引用，因此引号标记不需要被双写）：

```
EXECUTE 'UPDATE tbl SET '
        || quote_ident(colname)
        || ' = '
        || quote_literal(newvalue)
        || ' WHERE key = '
        || quote_literal(keyvalue);
```

这个示例展示了quote_ident和quote_literal函数的用法（见第 9.4 节）。为了安全起见，在插入动态查询之前，包含列名或表名标识符的表达式应先传给quote_ident。
在构造出的命令中应作为字符串字面量出现的值，则应传给quote_literal。

这两个函数都会采取适当措施，分别返回用双引号或单引号括起来的输入文本，
并正确转义其中嵌入的特殊字符。

由于 quote_literal 被标记为 STRICT，因此用 null 参数调用时它总会返回 null。在上面的示例中，如果 newvalue 或 keyvalue 为 null，整个动态查询字符串都会变成 null，进而导致 EXECUTE 报错。可以通过使用 quote_nullable 函数来避免这个问题；它与 quote_literal 的工作方式相同，只是在用 null 参数调用时会返回字符串 NULL。例如：

```
EXECUTE 'UPDATE tbl SET '
        || quote_ident(colname)
        || ' = '
        || quote_nullable(newvalue)
        || ' WHERE key = '
        || quote_nullable(keyvalue);
```

如果正在处理的参数值可能为空值，那么通常应该用 quote_nullable 来代替 quote_literal。

通常，必须小心地确保查询中的空值不会产生意料之外的结果。例如如果keyvalue为空值，下面的WHERE子句

```
'WHERE key = ' || quote_nullable(keyvalue)
```

永远不会成功，因为在=操作符中使用空值操作数得到的结果总是空值。
如果想让空值像普通键值一样工作，你应该将上面的命令重写成

```
'WHERE key IS NOT DISTINCT FROM ' || quote_nullable(keyvalue)
```

（目前，IS NOT DISTINCT FROM的处理效率不如=，因此只有在非常必要时才这样做。关于空值和IS DISTINCT的详细信息请见第 9.2 节）。

请注意美元符号引用只对引用固定文本有用。
尝试写出下面这个示例是一个非常糟糕的主意：

```
EXECUTE 'UPDATE tbl SET '
        || quote_ident(colname)
        || ' = $$'
        || newvalue
        || '$$ WHERE key = '
        || quote_literal(keyvalue);
```

因为如果newvalue的内容碰巧含有$$，那么这段代码就会出问题。
同样的问题也适用于你选择的任何其他美元符号引用定界符。因此，
要想安全地引用事先不知道的文本，必须恰当地使用quote_literal、quote_nullable或quote_ident。

动态 SQL 语句也可以使用format（见第 9.4 节）函数来安全地构造。例如：

```
EXECUTE format('UPDATE tbl SET %I = %L WHERE key = %L', colname,
 newvalue, keyvalue);
```

format函数可以和 USING子句一起使用：

```
EXECUTE format('UPDATE tbl SET %I = $1 WHERE key = $2', colname)
   USING newvalue, keyvalue;
```

这种形式更高效，因为参数newvalue和keyvalue 不会被转换为文本。

动态命令和EXECUTE的一个更大的示例可以在例 39.8中找到，它会构建并且执行一个CREATE FUNCTION命令来定义一个新的函数。

39.5.5. 获取结果状态

有好几种方法可以判断一条命令的效果。第一种方法是使用 GET DIAGNOSTICS 命令，其形式如下：

```
GET [ CURRENT ] DIAGNOSTICS variable = item [ , ... ];
```

这条命令允许检索系统状态指示符。每个 *item* 是一个关键字，它标识一个要赋给指定变量（该变量应具有正确的数据类型来接收它）的状态值。当前可用的状态项是 ROW_COUNT（发送到 SQL 引擎的最后一条 SQL 命令所处理的行数）和 RESULT_OID（最近一条 SQL 命令插入的最后一行的 OID）。注意，RESULT_OID 只有在向包含 OID 的表执行 INSERT 命令之后才有用。

例如：

```
GET DIAGNOSTICS integer_var = ROW_COUNT;
```

第二种确定命令效果的方法是检查名为FOUND的特殊变量，类型为boolean。
在每次PL/pgSQL函数调用中，FOUND的初始值都是 false。它由以下类型的语句设置：

- SELECT INTO语句在为目标赋上一行值时将FOUND设置为true，如果没有返回行则设置为false。
- PERFORM语句在生成（和丢弃）一个或多个行时将FOUND设置为true，如果没有生成行则设置为false。
- UPDATE、INSERT和DELETE语句在至少影响一行时将FOUND设置为真，如果没有影响行则设置为假。
- FETCH语句在返回行时将FOUND设置为true，如果没有返回行则设置为false。
- MOVE语句在成功重新定位游标时将FOUND设置为true，否则设置为false。
- FOR或FOREACH语句在迭代一次或多次时将 FOUND设置为true，否则设置为false。当循环退出时，FOUND会按上述方式设置；在循环执行过程中，FOUND不会被循环语句修改，尽管它可能会被循环体内的其他语句执行修改。
- RETURN QUERY和RETURN QUERY EXECUTE语句在查询返回至少一行时将FOUND设置为true，如果没有返回行则设置为false。

Other PL/pgSQL statements do not change the state of FOUND. Note in particular that EXECUTE changes the output of GET DIAGNOSTICS, but does not change FOUND.

FOUND是每个PL/pgSQL函数的局部变量；任何对它的修改只影响当前的函数。

39.5.6. 什么也不做

有时一个什么也不做的占位语句也很有用。例如，它能够指示 if/then/else 链中故意留出的空分支。可以使用NULL语句达到这个目的：

```
NULL;
```

例如，下面的两段代码是等价的：

```
BEGIN
    y := x / 0;
EXCEPTION
    WHEN division_by_zero THEN
        NULL;  -- 忽略错误
```

```
END;

BEGIN
    y := x / 0;
EXCEPTION
    WHEN division_by_zero THEN  -- 忽略错误
END;
```

究竟使用哪一种取决于各人的喜好。

> **注意**
>
> 在 Oracle 的 PL/SQL 中，不允许出现空语句列表，并且因此在这种情况下必须使用NULL语句。而PL/pgSQL允许你什么也不写。

39.6. 控制结构

控制结构可能是PL/pgSQL中最有用的（以及最重要）的部分了。
利用PL/pgSQL的控制结构，你可以以非常灵活而且强大的方法操纵PostgreSQL的数据。

39.6.1. 从函数返回

有两个命令让我们能够从函数中返回数据：RETURN和RETURN NEXT。

39.6.1.1. RETURN

```
RETURN expression;
```

带有一个表达式的RETURN用于终止函数并把*expression*的值返回给调用者。
这种形式被用于不返回集合的PL/pgSQL函数。

如果函数返回的是标量类型，任何表达式都可以使用。
表达式结果会按照赋值部分的说明自动转换为函数的返回类型。但要返回一个复合（行）值，你必须把记录变量或行变量写为 *expression*。

如果你声明带输出参数的函数，那么就只需要写不带表达式的RETURN。
输出参数变量的当前值将被返回。

如果你声明函数返回void，一个RETURN语句可以被用来提前退出函数；但是不要在RETURN后面写一个表达式。

一个函数的返回值不能是未定义。
如果控制到达了函数最顶层块的末尾而没有碰到一个RETURN语句，
那么会发生一个运行时错误。不过，这个限制不适用于带输出参数的函数以及返回void的函数。在这些情况中，如果顶层的块结束，将自动执行一个RETURN语句。

39.6.1.2. RETURN NEXT 和 RETURN QUERY

```
RETURN NEXT expression;
RETURN QUERY query;
RETURN QUERY EXECUTE command-string [ USING expression [, ... ] ];
```

当 PL/pgSQL 函数被声明为返回 SETOF *sometype* 时，返回过程会略有不同。在这种情况下，要返回的各个项通过一系列 RETURN NEXT 或 RETURN QUERY 命令指定，最后再用一个不带参数的 RETURN 命令表明函数已经执行完毕。RETURN NEXT 可用于标量和复合数据类型；对于复合结果类型，会返回完整的结果“表”。RETURN QUERY 会把查询执行结果追加到

函数的结果集中。在同一个集合返回函数中，RETURN NEXT 和 RETURN QUERY 可以自由混用，此时它们的结果会被串接起来。

RETURN NEXT和RETURN QUERY实际上不会从函数中返回 – 它们简单地向函数的结果集中追加零或多行。然后会继续执行PL/pgSQL函数中的下一条语句。随着后继的RETURN NEXT和RETURN QUERY命令的执行，结果集就建立起来了。最后一个RETURN（应该没有参数）会导致控制退出该函数（或者你可以让控制到达函数的结尾）。

RETURN QUERY有一种变体RETURN QUERY EXECUTE，它可以动态指定要被执行的查询。可以通过USING向计算出的查询字符串插入参数表达式，这和在EXECUTE命令中的方式相同。

如果你声明函数带有输出参数，只需要写不带表达式的RETURN NEXT。在每一次执行时，输出参数变量的当前值将被保存下来用于最终返回为结果的一行。
注意为了创建一个带有输出参数的集合返回函数，在有多个输出参数时，
你必须声明函数为返回SETOF record；或者如果只有一个类型为*sometype*的输出参数时，声明函数为SETOF *sometype*。

下面是一个使用 RETURN NEXT 的函数示例：

```
CREATE TABLE foo (fooid INT, foosubid INT, fooname TEXT);
INSERT INTO foo VALUES (1, 2, 'three');
INSERT INTO foo VALUES (4, 5, 'six');

CREATE OR REPLACE FUNCTION getAllFoo() RETURNS SETOF foo AS
$BODY$
DECLARE
    r foo%rowtype;
BEGIN
    FOR r IN SELECT * FROM foo
    WHERE fooid > 0
    LOOP
        -- can do some processing here
        RETURN NEXT r; -- return current row of SELECT
    END LOOP;
    RETURN;
END
$BODY$
LANGUAGE plpgsql;

SELECT * FROM getallfoo();
```

注意

如上所述，目前RETURN NEXT和RETURN QUERY的实现在从函数返回之前会把整个结果集都保存起来。这意味着如果一个PL/pgSQL函数生成一个非常大的结果集，
性能可能会很差：数据将被写到磁盘上以避免内存耗尽，
但是函数本身在整个结果集都生成之前不会退出。将来的
PL/pgSQL版本可能会允许用户定义没有这种限制的集合返回函数。目前，
数据开始被写入到磁盘的时机由配置变量work_mem控制。
拥有足够内存来存储大型结果集的管理员可以考虑增大这个参数。

39.6.2. 条件语句

IF 和 CASE 语句让你可以根据条件执行不同的命令。PL/pgSQL 有三种形式的 IF：

- IF ... THEN

- IF ... THEN ... ELSE
- IF ... THEN ... ELSIF ... THEN ... ELSE

and two forms of CASE:

- CASE ... WHEN ... THEN ... ELSE ... END CASE
- CASE WHEN ... THEN ... ELSE ... END CASE

39.6.2.1. IF-THEN

```
IF boolean-expression THEN
    statements
END IF;
```

IF-THEN 语句是 IF 的最简单形式。如果条件为真，在 THEN 和 END IF 之间的语句将被执行。否则，将忽略它们。

示例：

```
IF v_user_id <> 0 THEN
    UPDATE users SET email = v_email WHERE user_id = v_user_id;
END IF;
```

39.6.2.2. IF-THEN-ELSE

```
IF boolean-expression THEN
    statements
ELSE
    statements
END IF;
```

IF-THEN-ELSE 语句对 IF-THEN 进行了增加，它让你能够指定一组在条件不为真时应该被执行的语句（注意这也包括条件求值为 NULL 的情况）。

示例：

```
IF parentid IS NULL OR parentid = ''
THEN
    RETURN fullname;
ELSE
    RETURN hp_true_filename(parentid) || '/' || fullname;
END IF;

IF v_count > 0 THEN
    INSERT INTO users_count (count) VALUES (v_count);
    RETURN 't';
ELSE
    RETURN 'f';
END IF;
```

39.6.2.3. IF-THEN-ELSIF

```
IF boolean-expression THEN
    statements
[ ELSIF boolean-expression THEN
    statements
[ ELSIF boolean-expression THEN
```

```
    statements
    ...]]
[ ELSE
    statements ]
END IF;
```

有时会有多于两种选择。IF-THEN-ELSIF 则提供了一个简便的方法来检查多个条件。IF 条件会被一个接一个测试，直到找到第一个为真的。然后执行相关语句，之后控制会被交给 END IF 之后的下一个语句（后续的任何 IF 条件不会被测试）。如果没有一个 IF 条件为真，那么 ELSE 块（如果有）将被执行。

示例：

```
IF v_user_id <> 0 THEN
    UPDATE users SET email = v_email WHERE user_id = v_user_id;
END IF;
```

IF-THEN-ELSE语句对IF-THEN进行了增加，
它让你能够指定一组在条件不为真时应该被执行的语句（注意这也包括条件为 NULL 的情况）。

另一个可以完成相同任务的方法是嵌套 IF-THEN-ELSE 语句，如下例：

```
IF demo_row.sex = 'm' THEN
    pretty_sex := 'man';
ELSE
    IF demo_row.sex = 'f' THEN
        pretty_sex := 'woman';
    END IF;
END IF;
```

However, 这种方法要求为每个 IF 编写匹配的 END IF，因此当有许多备选方案时，它比使用 ELSIF 笨拙得多。

39.6.2.4. 简单 CASE

```
CASE search-expression
    WHEN expression [, expression [ ... ]] THEN
      statements
  [ WHEN expression [, expression [ ... ]] THEN
      statements
    ... ]
  [ ELSE
      statements ]
END CASE;
```

CASE 的简单形式提供基于操作数相等的条件执行。*search-expression* 被求值（一次），并依次与 WHEN 子句中的每个 *expression* 比较。如果找到匹配，则执行相应的 *statements*，然后控制传递到 END CASE 之后的下一个语句。（后续的 WHEN 表达式不会被求值。）如果没有找到匹配，则执行 ELSE 的 *statements*；但如果没有 ELSE，则会引发 CASE_NOT_FOUND 异常。

这里有一个简单示例：

```
CASE x
    WHEN 1, 2 THEN
        msg := 'one or two';
    ELSE
        msg := 'other value than one or two';
```

```
END CASE;
```

39.6.2.5. 搜索式 CASE

```
CASE
    WHEN boolean-expression THEN
      statements
  [ WHEN boolean-expression THEN
      statements
    ... ]
  [ ELSE
      statements ]
END CASE;
```

CASE 的搜索形式提供基于布尔表达式真实性的条件执行。每个 WHEN 子句的 *boolean-expression* 依次被求值，直到找到产生 true 的那一个。然后执行相应的 *statements*，之后控制传递到 END CASE 之后的下一个语句。（后续的 WHEN 表达式不会被求值。）如果没有找到为真的结果，则执行 ELSE 的 *statements*；但如果没有 ELSE，则会引发 CASE_NOT_FOUND 异常。

下面是一个例子：

```
CASE
    WHEN x BETWEEN 0 AND 10 THEN
        msg := 'value is between zero and ten';
    WHEN x BETWEEN 11 AND 20 THEN
        msg := 'value is between eleven and twenty';
END CASE;
```

这种形式的 CASE 完全等价于 IF-THEN-ELSIF，唯一的规则差异是：到达被省略的 ELSE 子句时会报错，而不是什么也不做。

39.6.3. 简单循环

使用LOOP、EXIT、CONTINUE、WHILE、FOR和FOREACH语句，可以让PL/pgSQL函数重复执行一系列命令。

39.6.3.1. LOOP

```
[ <<label>> ]
LOOP
    statements
END LOOP [ label ];
```

LOOP 定义了一个无条件循环，它会无限重复，直到被 EXIT 或 RETURN 语句终止。可选的 *label* 可以供嵌套循环中的 EXIT 和 CONTINUE 语句使用，以指定这些语句作用的是哪一个循环。

39.6.3.2. EXIT

```
EXIT [ label ] [ WHEN boolean-expression ];
```

如果没有给出 *label*，那么最内层的循环会被终止，然后跟在 END LOOP 后面的语句会被执行。如果给出了 *label*，那么它必须是当前或者更高层的嵌套循环或者语句块的标签。然后该命名循环或块就会被终止，并且控制会转移到该循环/块相应的 END 之后的语句上。

如果指定了 WHEN，则只有在 *boolean-expression* 为真时才会退出循环。否则，控制传递到 EXIT 之后的语句。

EXIT 可以与所有类型的循环一起使用；它并不限于在无条件循环中使用。

当与 BEGIN 块一起使用时，EXIT 会把控制转交给该块结束后的下一条语句。需要注意的是，为此必须使用标签；未加标签的 EXIT 永远不会被视为匹配某个 BEGIN 块。这与 PostgreSQL 8.4 之前的版本不同，旧版本允许未加标签的 EXIT 匹配 BEGIN 块。）

例子：

```
LOOP
    -- some computations
    IF count > 0 THEN
        EXIT;  -- exit loop
    END IF;
END LOOP;

LOOP
    -- some computations
    EXIT WHEN count > 0;  -- same result as previous example
END LOOP;

<<ablock>>
BEGIN
    -- some computations
    IF stocks > 100000 THEN
        EXIT ablock;  -- causes exit from the BEGIN block
    END IF;
    -- computations here will be skipped when stocks > 100000
END;
```

39.6.3.3. CONTINUE

CONTINUE [*label*] [WHEN *boolean-expression*];

如果没有给出 *label*，最内层循环的下一次迭代会开始。也就是，循环体中剩余的所有语句将被跳过，并且控制会返回到循环控制表达式（如果有）来决定是否需要另一次循环迭代。如果 *label* 存在，它指定应该继续执行的循环的标签。

如果指定了 WHEN，则只有在 *boolean-expression* 为真时才会开始循环的下一次迭代。否则，控制传递到 CONTINUE 之后的语句。

CONTINUE 可以与所有类型的循环一起使用；它并不限于在无条件循环中使用。

示例：

```
LOOP
    -- 一些计算
    IF count > 0 THEN
        EXIT;  -- 退出循环
    END IF;
END LOOP;

LOOP
    -- 一些计算
    EXIT WHEN count > 0;  -- 和前一个示例相同的结果
END LOOP;

<<ablock>>
BEGIN
    -- 一些计算
```

```
    IF stocks > 100000 THEN
        EXIT ablock;  -- 导致从 BEGIN 块中退出
    END IF;
    -- 当stocks > 100000时，这里的计算将被跳过
END;
```

39.6.3.4. WHILE

```
[ <<label>> ]
WHILE boolean-expression LOOP
    statements
END LOOP [ label ];
```

只要 *boolean-expression* 被计算为真，WHILE 语句就会重复一个语句序列。在每次进入到循环体之前都会检查该表达式。

例如：

```
WHILE amount_owed > 0 AND gift_certificate_balance > 0 LOOP
    -- some computations here
END LOOP;

WHILE NOT done LOOP
    -- some computations here
END LOOP;
```

39.6.3.5. FOR（整型变体）

```
[ <<label>> ]
FOR name IN [ REVERSE ] expression .. expression [ BY expression ]
 LOOP
    statements
END LOOP [ label ];
```

这种形式的 FOR 会创建一个在一个整数范围上迭代的循环。变量 *name* 会自动定义为类型 integer 并且只在循环内存在（任何该变量名的现有定义在此循环内都将被忽略）。给出范围上下界的两个表达式在进入循环的时候计算一次。如果没有指定 BY 子句，迭代步长为 1，否则步长是 BY 中指定的值，该值也只在循环进入时计算一次。如果指定了 REVERSE，那么在每次迭代后会减去步长，而不是加上步长。

整数 FOR 循环的一些示例：

```
FOR i IN 1..10 LOOP
    -- i will take on the values 1,2,3,4,5,6,7,8,9,10 within the
 loop
END LOOP;

FOR i IN REVERSE 10..1 LOOP
    -- i will take on the values 10,9,8,7,6,5,4,3,2,1 within the
 loop
END LOOP;

FOR i IN REVERSE 10..1 BY 2 LOOP
    -- i will take on the values 10,8,6,4,2 within the loop
END LOOP;
```

如果下界大于上界（或者在 REVERSE 情况下是小于），循环体根本不会被执行。而且不会抛出任何错误。

如果 FOR 循环附加了 *label*，则整数循环变量可以用该 *label* 限定的名称来引用。

39.6.4. 遍历查询结果

使用另一种形式的FOR循环，可以迭代一个查询的结果并相应地处理这些数据。语法是：

```
[ <<label>> ]
FOR target IN query LOOP
    statements
END LOOP [ label ];
```

The *target* is a record variable, row variable, or comma-separated list of scalar variables. The *target* is successively assigned each row resulting from the *query* and the loop body is executed for each row. Here is an example:

```
CREATE FUNCTION cs_refresh_mviews() RETURNS integer AS $$
DECLARE
    mviews RECORD;
BEGIN
    RAISE NOTICE 'Refreshing materialized views...';

    FOR mviews IN SELECT * FROM cs_materialized_views ORDER BY
 sort_key LOOP

        -- Now "mviews" has one record from cs_materialized_views

        RAISE NOTICE 'Refreshing materialized view %s ...', quote_
ident(mviews.mv_name);
        EXECUTE 'TRUNCATE TABLE ' || quote_ident(mviews.mv_name);
        EXECUTE 'INSERT INTO '
                   || quote_ident(mviews.mv_name) || ' '
                   || mviews.mv_query;
    END LOOP;

    RAISE NOTICE 'Done refreshing materialized views.';
    RETURN 1;
END;
$$ LANGUAGE plpgsql;
```

If the loop is terminated by an EXIT statement, the last assigned row value is still accessible after the loop.

这种形式的 FOR 语句所使用的 *query* 可以是任何向调用者返回行的 SQL 命令：SELECT 是最常见的情况，但也可以使用带 RETURNING 子句的 INSERT、UPDATE 或 DELETE。某些工具命令（如 EXPLAIN）也可以。

PL/pgSQL 变量会被替换到查询文本中，并且查询计划会被缓存以备可能的重用，详见第 39.10.1 节和第 39.10.2 节。

FOR-IN-EXECUTE 语句是另一种遍历行的方式：

```
[ <<label>> ]
FOR target IN EXECUTE text_expression [ USING expression [, ... ] ]
 LOOP
    statements
END LOOP [ label ];
```

它与前一种形式类似，只是源查询被指定为一个字符串表达式，在每次进入 FOR 循环时都会重新求值和重新规划。

这让程序员可以在预先规划的查询的速度与动态查询的灵活性之间做出选择，
就像使用普通的 EXECUTE 语句一样。与 EXECUTE 一样，可以通过 USING 向动态命令中插入参数值。

指定要遍历其结果的查询的另一种方式，是把它声明为游标。这在第 39.7.4 节中描述。

39.6.5. 遍历数组

FOREACH循环与FOR循环很相似，但它不是迭代一个 SQL 查询返回的行，而是迭代一个数组值的元素。（一般而言，FOREACH旨在循环遍历组合值表达式的组成部分；
将来可能会增加遍历数组之外的其他组合的变体。）用于遍历数组的FOREACH语句是：

```
[ <<label>> ]
FOREACH target [ SLICE number ] IN ARRAY expression LOOP
    statements
END LOOP [ label ];
```

未指定SLICE或指定了SLICE 0时，循环会遍历计算*expression*得到的数组的各个元素。
*target*变量被依次赋予每个元素的值，循环体为每个元素执行一次。
下面是遍历一个整数数组元素的例子：

```
CREATE FUNCTION sum(int[]) RETURNS int8 AS $$
DECLARE
  s int8 := 0;
  x int;
BEGIN
  FOREACH x IN ARRAY $1
  LOOP
    s := s + x;
  END LOOP;
  RETURN s;
END;
$$ LANGUAGE plpgsql;
```

The elements are visited in storage order, regardless of the number of array dimensions. Although the *target* is usually just a single variable, it can be a list of variables when looping through an array of composite values (records). In that case, for each array element, the variables are assigned from successive columns of the composite value.

当SLICE值为正数时，FOREACH迭代的是数组的切片而非单个元素。SLICE
值必须是一个不大于数组维数的整数常量。*target*变量必须是一个数组，
它会依次收到数组值的各个切片，每个切片的维数由SLICE指定。
下面是遍历一维切片的例子：

```
CREATE FUNCTION scan_rows(int[]) RETURNS void AS $$
DECLARE
  x int[];
BEGIN
  FOREACH x SLICE 1 IN ARRAY $1
  LOOP
    RAISE NOTICE 'row = %', x;
  END LOOP;
END;
$$ LANGUAGE plpgsql;

SELECT scan_rows(ARRAY[[1,2,3],[4,5,6],[7,8,9],[10,11,12]]);

NOTICE:  row = {1,2,3}
```

```
NOTICE:  row = {4,5,6}
NOTICE:  row = {7,8,9}
NOTICE:  row = {10,11,12}
```

39.6.6. 捕获错误

默认情况下，PL/pgSQL 函数中发生的任何错误都会中止函数及其外围事务的执行。你可以使用带有 EXCEPTION 子句的 BEGIN 块来捕获错误并从中恢复。其语法是在普通 BEGIN 块语法上的扩展：

```
[ <<label>> ]
[ DECLARE
    declarations ]
BEGIN
    statements
EXCEPTION
    WHEN condition [ OR condition ... ] THEN
        handler_statements
    [ WHEN condition [ OR condition ... ] THEN
          handler_statements
      ... ]
END;
```

如果没有发生错误，这种形式的块只是简单地执行所有 *statements*，并且接着控制转到 END 之后的下一个语句。但是如果在该 *statements* 内发生了一个错误，则会放弃对 *statements* 的进一步处理，然后控制会转到 EXCEPTION 列表。系统会在列表中寻找匹配所发生错误的第一个 *condition*。如果找到一个匹配，则执行对应的 *handler_statements*，并且接着把控制转到 END 之后的下一个语句。如果没有找到匹配，该错误就会传播出去，就好像根本没有 EXCEPTION 一样：错误可以被一个带有 EXCEPTION 的外围块捕捉，如果没有这样的块则中止该函数的处理。

condition 的名字可以是附录 A中显示的任何名字。一个分类名匹配其中所有的错误。特殊的条件名 OTHERS 匹配除了 QUERY_CANCELED 之外的所有错误类型（虽然可能但通常并不明智，还是可以用名字捕获 QUERY_CANCELED）。条件名是大小写无关的。一个错误条件也可以通过 SQLSTATE 代码指定，例如以下是等价的：

```
WHEN division_by_zero THEN ...
WHEN SQLSTATE '22012' THEN ...
```

如果在选中的 *handler_statements* 内发生了新的错误，那么它不能被这个 EXCEPTION 子句捕获，而是被传播出去。一个外层的 EXCEPTION 子句可以捕获它。

当一个错误被 EXCEPTION 子句捕获时，PL/pgSQL 函数的局部变量会保持错误发生时的值，但是该块中所有对持久数据库状态的改变都会被回滚。例如，考虑这个片段：

```
INSERT INTO mytab(firstname, lastname) VALUES('Tom', 'Jones');
BEGIN
    UPDATE mytab SET firstname = 'Joe' WHERE lastname = 'Jones';
    x := x + 1;
    y := x / 0;
EXCEPTION
    WHEN division_by_zero THEN
        RAISE NOTICE 'caught division_by_zero';
        RETURN x;
END;
```

当控制到达对 y 的赋值时，它会以 division_by_zero 错误失败。该错误会被 EXCEPTION 子句捕获。RETURN 语句返回的值将是 x 递增后的值，但 UPDATE 命令的效果已被回滚。不

过，块之前的 INSERT 命令没有被回滚，因此最终结果是数据库中包含的是 Tom Jones 而不是 Joe Jones。

> **提示**
>
> 包含 EXCEPTION 子句的块，其进入和退出的开销明显高于没有该子句的块。因此，不要在没有必要的情况下使用 EXCEPTION。

例 39.2. UPDATE/INSERT的异常

这个例子使用异常处理来恰当地执行UPDATE或INSERT：

```
CREATE TABLE db (a INT PRIMARY KEY, b TEXT);

CREATE FUNCTION merge_db(key INT, data TEXT) RETURNS VOID AS
$$
BEGIN
    LOOP
        -- first try to update the key
        UPDATE db SET b = data WHERE a = key;
        IF found THEN
            RETURN;
        END IF;
        -- not there, so try to insert the key
        -- if someone else inserts the same key concurrently,
        -- we could get a unique-key failure
        BEGIN
            INSERT INTO db(a,b) VALUES (key, data);
            RETURN;
        EXCEPTION WHEN unique_violation THEN
            -- Do nothing, and loop to try the UPDATE again.
        END;
    END LOOP;
END;
$$
LANGUAGE plpgsql;

SELECT merge_db(1, 'david');
SELECT merge_db(1, 'dennis');
```

这段代码假定 unique_violation 错误是由 INSERT 引起的，而不是由表上的某个触发器函数中的 INSERT 之类引起。如果表上有多个唯一索引，它也可能表现不当，因为无论错误由哪个索引引起，它都会重试该操作。
使用下一节讨论的特性来检查被捕获的错误是否就是预期的错误，可以获得更多的安全性。

39.6.6.1. 获取错误信息

异常处理器经常需要识别所发生的具体错误。
有两种方法可以获取PL/pgSQL中当前异常的信息：特殊变量和GET STACKED DIAGNOSTICS命令。

在一个异常处理器内，特殊变量SQLSTATE
包含了对应于被抛出异常的错误代码（可能的错误代码列表见表 A.1）。特殊变量SQLERRM包含与该异常相关的错误消息。这些变量在异常处理器外是未定义的。

在一个异常处理器内，我们也可以用GET STACKED DIAGNOSTICS命令检索有关当前异常的信息，该命令的形式为：

```
GET STACKED DIAGNOSTICS variable = item [ , ... ];
```

每个*item*是一个关键词，它标识一个被赋给指定变量（应该具有接收该值的正确数据类型）的状态值。表 39.1中显示了当前可用的状态项。

表 39.1. 错误诊断项

名称	类型	描述
RETURNED_SQLSTATE	text	该异常的 SQLSTATE 错误代码
MESSAGE_TEXT	text	该异常的主要消息的文本
PG_EXCEPTION_DETAIL	text	该异常的详细消息文本（如果有）
PG_EXCEPTION_HINT	text	该异常的提示消息文本（如果有）
PG_EXCEPTION_CONTEXT	text	描述产生异常时调用栈的文本行

如果异常没有为一个项设置值，将返回一个空字符串。

这里是一个示例：

```
DECLARE
  text_var1 text;
  text_var2 text;
  text_var3 text;
BEGIN
  -- 某些可能导致异常的处理
  ...
EXCEPTION WHEN OTHERS THEN
  GET STACKED DIAGNOSTICS text_var1 = MESSAGE_TEXT,
                          text_var2 = PG_EXCEPTION_DETAIL,
                          text_var3 = PG_EXCEPTION_HINT;
END;
```

39.7. 游标

和一次执行整个查询不同，可以建立一个游标来封装该查询，
并且接着一次读取该查询结果的一些行。
这样做的原因之一是在结果中包含大量行时避免内存不足（不过，
PL/pgSQL用户通常不需要担心这些，因为FOR
循环在内部会自动使用一个游标来避免内存问题）。
一种更有趣的用法是返回一个函数已经创建的游标的引用，允许调用者读取行。
这提供了一种有效的方法从函数中返回大型行集。

39.7.1. 声明游标变量

所有在PL/pgSQL中对游标的访问都会通过游标变量，它总是特殊的数据类型refcursor。
创建游标变量的一种方法是把它声明为一个类型为refcursor的变量。
另外一种方法是使用游标声明语法，通常是：

```
name [ [ NO ] SCROLL ] CURSOR [ ( arguments ) ] FOR query;
```

（为兼容 Oracle，可以用 IS 代替 FOR。）如果指定了 SCROLL，游标就支持向后滚动；如果指定了 NO SCROLL，向后提取会被拒绝；如果两者都未指定，是否允许向后提取则取决于查询本身。如果指定了 *arguments*，它就是一个由 *name datatype* 对组成的逗号分隔列表，这些名字会在给定查询中被参数值替换。实际替换这些名字的值会在打开游标时提供。

Some examples:

```
DECLARE
    curs1 refcursor;
    curs2 CURSOR FOR SELECT * FROM tenk1;
    curs3 CURSOR (key integer) FOR SELECT * FROM tenk1 WHERE
 unique1 = key;
```

所有这三个变量都是 `refcursor` 类型，但是第一个可以用于任何查询，而第二个已经被绑定了一个完全指定的查询，并且最后一个被绑定了一个参数化查询。（游标被打开时，`key` 将被一个整数参数值替换。）变量 `curs1` 被称为未绑定，因为它没有被绑定到任何特定查询。

39.7.2. 打开游标

在能够使用游标检索行之前，必须先将其打开（这等效于 SQL 命令`DECLARE CURSOR`）。PL/pgSQL有三种形式的`OPEN`命令，其中两种用于未绑定游标变量，另一种用于已绑定游标变量。

> **注意**
>
> 可以通过第 39.7.4 节中描述的`FOR`语句在不显式打开游标的情况下使用已绑定的游标变量。

39.7.2.1. OPEN FOR *query*

```
OPEN unbound_cursorvar [ [ NO ] SCROLL ] FOR query;
```

该游标变量会被打开，并被赋予要执行的指定查询。该游标不能已经处于打开状态，并且它必须已被声明为未绑定游标变量（即，一个简单的`refcursor`变量）。该查询必须是`SELECT`，或其他会返回行的命令（例如`EXPLAIN`）。该查询会以与PL/pgSQL中其他 SQL 命令相同的方式处理：替换PL/pgSQL变量名，并缓存查询计划以备后续重用。当一个PL/pgSQL变量被替换到游标查询中时，被替换的是它在`OPEN`时刻所具有的值；之后对该变量的更改不会影响游标的行为。`SCROLL`和`NO SCROLL`选项与已绑定游标中的含义相同。

一个示例：

```
OPEN curs1 FOR SELECT * FROM foo WHERE key = mykey;
```

39.7.2.2. OPEN FOR EXECUTE

```
OPEN unbound_cursorvar [ [ NO ] SCROLL ] FOR EXECUTE query_string
                                        [ USING expression [, ... ] ];
```

该游标变量会被打开，并被赋予要执行的指定查询。该游标不能已经处于打开状态，并且必须已被声明为未绑定游标变量（即，一个简单的`refcursor`变量）。该查询以字符串表达式的形式给出，这一点与`EXECUTE`命令相同。像往常一样，这提供了灵活性，因此查询计划可以在不同执行之间变化（见第 39.10.2 节），同时也意味着不会在该命令字符串上执行变量替换。与`EXECUTE`一样，可以通过`USING`把参数值插入动态命令中。`SCROLL`和`NO SCROLL`选项与已绑定游标中的含义相同。

一个示例：

```
OPEN curs1 FOR EXECUTE 'SELECT * FROM ' || quote_ident(tabname)
                                        || ' WHERE col1 = $1' USING
 keyvalue;
```

在这个示例中，表名以文本方式插入到查询中，因此建议使用 `quote_ident()`来防范 SQL 注入。`col1` 的比较值通过`USING`参数插入，因此无需加引号。

39.7.2.3. 打开已绑定游标

```
OPEN bound_cursorvar [ ( [ argument_name := ] argument_value [, ...
] ) ];
```

这种形式的`OPEN`用于打开一个查询在声明时已绑定到其上的游标变量。
该游标不能已被打开。当且仅当游标被声明为接受参数时，才必须出现实参值表达式列表。
这些值将被代入查询中。

绑定游标的查询计划总是被视为可缓存的；这种情况没有等价的`EXECUTE`。注意，不能在`OPEN`中指定`SCROLL`和`NO SCROLL`，因为游标的滚动行为已经确定。

使用位置或命名记法可以传递参数值。在位置记法中，所有参数都必须按照顺序指定。
在命名记法中，每一个参数的名字使用`:=`与参数表达式分隔。类似于第 4.3 节中描述的调用函数，也允许混合位置和命名记法。

因为已绑定游标的查询会进行变量替换，实际上有两种方式把值传给游标：要么向`OPEN`传入显式参数，要么在查询中隐式引用PL/pgSQL变量。不过，
只有在声明已绑定游标之前就已声明的变量才会被替换到查询中。在这两种情况下，
要传递的值都在`OPEN`时确定。例如，获得与上面`curs3`示例相同效果的另一种方式是

```
DECLARE
    key integer;
    curs4 CURSOR FOR SELECT * FROM tenk1 WHERE unique1 = key;
BEGIN
    key := 42;
    OPEN curs4;
```

示例（这些示例使用上面的游标声明示例）：

```
OPEN curs2;
OPEN curs3(42);
OPEN curs3(key := 42);
```

39.7.3. 使用游标

一旦一个游标已经被打开，那么就可以用这里描述的语句操作它。

这些操作不必发生在最初打开该游标的同一个函数中。你可以从函数中返回一个`refcursor`值，让调用者来操作该游标。（在内部，`refcursor`值只是一个所谓 portal 的字符串名称，该 portal 包含了该游标活动查询的状态。这个名称可以被传递、赋给其他`refcursor`变量等等，而不会干扰该 portal。）

所有 portal 都会在事务结束时被隐式关闭。因此，`refcursor`值只能在事务结束之前用于引用一个打开的游标。

39.7.3.1. FETCH

```
FETCH [ direction { FROM | IN } ] cursor INTO target;
```

`FETCH`从游标中检索下一行到目标中，目标可以是一个行变量、
记录变量或者逗号分隔的简单变量列表，就像`SELECT INTO`一样。如果没有下一行，目标会被设置为 NULL。与`SELECT INTO`一样，可以检查特殊变量`FOUND`来看是否获得了一行。

direction 子句可以是 SQL FETCH命令中允许的任何变体，除了那些能够取得多于一行的。即它可以是 `NEXT`、`PRIOR`、`FIRST`、`LAST`、`ABSOLUTE` *count*、`RELATIVE` *count*、

FORWARD 或者 BACKWARD。省略 *direction* 和指定 NEXT 是一样的。要求反向移动的 *direction* 值很可能会失败，除非游标被使用 SCROLL 选项声明或打开。

*cursor*必须是一个引用已打开游标 portal 的refcursor变量名。

示例：

```
FETCH curs1 INTO rowvar;
FETCH curs2 INTO foo, bar, baz;
FETCH LAST FROM curs3 INTO x, y;
FETCH RELATIVE -2 FROM curs4 INTO x;
```

39.7.3.2. MOVE

```
MOVE [ direction { FROM | IN } ] cursor;
```

MOVE重新定位一个游标而不检索任何数据。MOVE的工作方式与FETCH完全一样，只是仅重新定位游标，而不返回移动到的行。与SELECT INTO一样，可以检查特殊变量FOUND来看是否存在可移动到的下一行。

direction 子句可以是 SQL FETCH命令中允许的任何变体，即 NEXT、PRIOR、FIRST、LAST、ABSOLUTE *count*、RELATIVE *count*、ALL、FORWARD [*count* | ALL] 或 BACKWARD [*count* | ALL]。省略 *direction* 和指定 NEXT 是一样的。要求反向移动的 *direction* 值很可能会失败，除非游标被使用 SCROLL 选项声明或打开。

示例：

```
MOVE curs1;
MOVE LAST FROM curs3;
MOVE RELATIVE -2 FROM curs4;
MOVE FORWARD 2 FROM curs4;
```

39.7.3.3. UPDATE/DELETE WHERE CURRENT OF

```
UPDATE table SET ... WHERE CURRENT OF cursor;
DELETE FROM table WHERE CURRENT OF cursor;
```

当游标定位在某个表行上时，可以使用该游标来标识该行，并对其执行更新或删除。游标查询的形式有一些限制（尤其不能包含分组），并且在这类场景中最好对游标使用 FOR UPDATE。详见DECLARE参考页。

示例：

```
FETCH curs1 INTO rowvar;
FETCH curs2 INTO foo, bar, baz;
FETCH LAST FROM curs3 INTO x, y;
FETCH RELATIVE -2 FROM curs4 INTO x;
```

39.7.3.4. CLOSE

```
CLOSE cursor;
```

CLOSE关闭打开游标所基于的 portal。这样就可以在事务结束之前提前释放资源，或者释放该游标变量以便再次打开。

示例：

```
MOVE curs1;
MOVE LAST FROM curs3;
```

```
MOVE RELATIVE -2 FROM curs4;
MOVE FORWARD 2 FROM curs4;
```

39.7.3.5. 返回游标

PL/pgSQL函数可以向调用者返回游标。这对于返回多行或多列，特别是非常大的结果集时很有用。要做到这一点，函数需要打开游标，并把游标名返回给调用者（或者直接使用调用者指定或已知的 portal 名称来打开游标）。随后调用者就可以从该游标中提取行。游标既可以由调用者关闭，也会在事务结束时自动关闭。

游标使用的 portal 名称既可以由程序员指定，也可以自动生成。要指定 portal 名称，只需在打开`refcursor`变量之前给它赋一个字符串值。`OPEN`会把该`refcursor`变量的字符串值用作底层 portal 的名称。不过，如果`refcursor`变量为 null，`OPEN`就会自动生成一个与任何现有 portal 都不冲突的名称，并把它赋回给`refcursor`变量。

> **注意**
>
> 已绑定游标变量会被初始化为表示其名称的字符串值，因此除非程序员在打开游标之前通过赋值覆盖它，否则 portal 名称与游标变量名相同。而未绑定游标变量在初始时默认为空值，因此除非被覆盖，否则它会得到一个自动生成的唯一名称。

一个示例：

```
CLOSE curs1;
```

下面的示例使用了自动游标名生成：

```
CREATE FUNCTION reffunc2() RETURNS refcursor AS '
DECLARE
    ref refcursor;
BEGIN
    OPEN ref FOR SELECT col FROM test;
    RETURN ref;
END;
' LANGUAGE plpgsql;

-- 需要在一个事务中使用游标。
BEGIN;
SELECT reffunc2();

      reffunc2
--------------------
 <unnamed cursor 1>
(1 row)

FETCH ALL IN "<unnamed cursor 1>";
COMMIT;
```

下面的示例展示了从一个函数中返回多个游标的一种方法：

```
CREATE FUNCTION myfunc(refcursor, refcursor) RETURNS SETOF
 refcursor AS $$
BEGIN
    OPEN $1 FOR SELECT * FROM table_1;
    RETURN NEXT $1;
```

```
    OPEN $2 FOR SELECT * FROM table_2;
    RETURN NEXT $2;
END;
$$ LANGUAGE plpgsql;

-- 需要在一个事务中使用游标。
BEGIN;

SELECT * FROM myfunc('a', 'b');

FETCH ALL FROM a;
FETCH ALL FROM b;
COMMIT;
```

39.7.4. 遍历游标结果

FOR 语句还有一种变体，允许循环通过游标返回的行。语法是：

```
[ <<label>> ]
FOR recordvar IN bound_cursorvar [ ( [ argument_name := ] argument_
value [, ...] ) ] LOOP
    statements
END LOOP [ label ];
```

该游标变量必须在声明时已经被绑定到某个查询，并且它不能已经处于打开状态。FOR 语句会自动打开游标，并且在退出循环时自动关闭游标。当且仅当游标被声明要使用参数时，才必须出现一个实际参数值表达式的列表。这些值会被替换到查询中，采用 OPEN 期间的方式（见第 39.7.2.3 节）。

变量 *recordvar* 会被自动定义为 record 类型，并且只存在于循环内部（循环中该变量名任何已有定义都会被忽略）。
每一个由游标返回的行都会被陆续地赋值给这个记录变量并且执行循环体。

39.8. 错误和消息

使用RAISE语句报告消息和抛出错误。

```
RAISE [ level ] 'format' [, expression [, ... ]] [ USING option =
 expression [, ... ] ];
RAISE [ level ] condition_name [ USING option = expression [, ...
 ] ];
RAISE [ level ] SQLSTATE 'sqlstate' [ USING option = expression [,
 ... ] ];
RAISE [ level ] USING option = expression [, ... ];
RAISE ;
```

其中，*level*选项指定错误的严重程度。允许的级别为DEBUG，LOG，INFO，NOTICE，WARNING以及EXCEPTION，其中EXCEPTION是默认值。EXCEPTION
会抛出错误（通常会中止当前事务）；其他级别只会生成不同优先级的消息。
特定优先级的消息是报告给客户端、写入服务器日志，还是两者都做，由log_min_messages和client_min_messages配置变量控制。更多信息见第 18 章中的说明。

在*level*（如果有）之后，可以写一个*format*（必须是简单的字符串字面量，不能是表达式）。格式字符串指定要报告的错误消息文本。
格式字符串之后可以跟上可选的参数表达式，其值将被插入消息中。在格式字符串内，%会被替换为下一个可选参数值的字符串表示。写成%%可以输出一个字面的%。

在这个示例中，v_job_id的值将替换字符串中的%：

```
RAISE NOTICE 'Calling cs_create_job(%)', v_job_id;
```

可以通过写出 USING，后面跟上 *option* = *expression* 项目，为错误报告附加额外信息。允许的 *option* 关键字是 MESSAGE、DETAIL、HINT 和 ERRCODE，每个 *expression* 都可以是任意字符串值表达式。MESSAGE 设置错误消息文本（该选项不能用于在 USING 之前包含格式字符串的 RAISE 形式）。DETAIL 提供错误的详细信息，HINT 提供一个提示消息。ERRCODE 指定要报告的错误代码（SQLSTATE），可以用附录 A中所示的条件名，或者直接作为一个五字符 SQLSTATE 代码。

这个例子会中止事务，并给出指定的错误消息和提示：

```
RAISE EXCEPTION 'Nonexistent ID --> %', user_id
      USING HINT = 'Please check your user ID';
```

这两个示例展示了设置 SQLSTATE 的两种等价的方法：

```
RAISE 'Duplicate user ID: %', user_id USING ERRCODE = 'unique_
violation';
RAISE 'Duplicate user ID: %', user_id USING ERRCODE = '23505';
```

还有第二种RAISE语法，其中主参数是要报告的条件名或 SQLSTATE，例如：

```
RAISE division_by_zero;
RAISE SQLSTATE '22012';
```

在这种语法中，USING可以用来提供自定义的错误消息、细节或提示。
另一种达到前面示例同样效果的方式是：

```
RAISE unique_violation USING MESSAGE = 'Duplicate user ID: ' ||
 user_id;
```

还有另一种变体是写RAISE USING或RAISE *level* USING，并把其余内容都放在USING列表里。

RAISE的最后一种变体根本没有参数。这种形式只能被用在一个BEGIN块的EXCEPTION子句中，它导致当前正在被处理的错误被重新抛出。

> **注意**
>
> 在PostgreSQL 9.1 之前，没有参数的RAISE被解释为重新抛出来自包含活动异常处理器的块的错误。因此一个嵌套在那个处理器中的EXCEPTION子句无法捕捉它，即使RAISE位于嵌套EXCEPTION子句的块中也是这样。这种行为很奇怪，也并不兼容 Oracle 的 PL/SQL。

如果在RAISE EXCEPTION命令中没有指定条件名称或SQLSTATE，则默认使用RAISE_EXCEPTION (P0001)。如果没有指定消息文本，则默认使用条件名称或SQLSTATE作为消息文本。

> **注意**
>
> 当用 SQLSTATE 代码指定错误代码时，你并不受限于预定义错误代码，而是可以选择任何由五位数字和/或大写 ASCII 字母构成的错误代码，唯一不能使用的是 00000。我们建议尽量避免抛出以三个零结尾的错误代码，因为这些是类别代码，只能通过捕获整个类别来捕获这类错误。

39.9. 触发器函数

PL/pgSQL可以被用来定义触发器函数。触发器函数用CREATE FUNCTION命令创建，它被声明为一个没有参数并且返回类型为trigger的函数。注意，即便该函数准备接收一些在CREATE TRIGGER中指定的参数 – 这类参数通过TG_ARGV传递（如下所述），也必须把它声明为没有参数。

当一个 PL/pgSQL 函数作为触发器被调用时，会在顶层块中自动创建一些特殊变量。它们是：

NEW

数据类型为 RECORD；该变量保存行级触发器中用于 INSERT/UPDATE 操作的新数据行。在语句级触发器和 DELETE 操作中，该变量为 NULL。

OLD

数据类型为 RECORD；该变量保存行级触发器中用于 UPDATE/DELETE 操作的旧数据行。在语句级触发器和 INSERT 操作中，该变量为 NULL。

TG_NAME

数据类型为 name；包含实际触发的触发器名称的变量。

TG_WHEN

数据类型为 text；根据触发器定义，其值为字符串 BEFORE、AFTER 或 INSTEAD OF。

TG_LEVEL

数据类型为 text；根据触发器定义，其值为字符串 ROW 或 STATEMENT。

TG_OP

数据类型为 text；表示触发器对应操作的字符串：INSERT、UPDATE、DELETE 或 TRUNCATE。

TG_RELID

数据类型为 name；包含实际触发的触发器名称的变量。

TG_RELNAME

数据类型为 name；导致触发器调用的表名。该变量已弃用，未来版本可能移除；请改用 TG_TABLE_NAME。

TG_TABLE_NAME

数据类型为 name；导致触发器调用的表名。

TG_TABLE_SCHEMA

数据类型为 name；导致触发器调用的表所在模式名。

TG_NARGS

数据类型为integer；CREATE TRIGGER语句中传给触发器函数的参数个数。

TG_ARGV[]

数据类型为text数组；来自CREATE TRIGGER语句的参数。索引从 0 开始；非法索引（小于 0 或大于等于tg_nargs）返回 null。

一个触发器函数必须返回NULL或者是一个与触发器为之引发的表结构完全相同的记录/行值。

BEFORE 行级触发器可以返回 null，以通知触发器管理器跳过该行后续的操作（也就是说，不再触发后续触发器，并且不会对该行执行INSERT/UPDATE/DELETE）。如果返回非 null 值，则操作会继续进行，并使用该行值。返回一个不同于原始NEW的行值会改变即将插入或更新的行。因此，如果触发器函数希望触发动作正常成功而不修改行值，就必须返回NEW（或与之相等的值）。若要修改将被存储的行，可以直接替换NEW中的单个值并返回修改后的NEW，或者构造一个完整的新记录/行来返回。对于作用于DELETE的 before 触发器，返回值本身没有直接效果，但必须为非 null 才能让触发器动作继续。注意，在DELETE触发器中NEW为 null，因此通常没有理由返回它。在DELETE触发器中，常见写法是返回OLD。

INSTEAD OF触发器（总是行级触发器，并且只能用于视图）能够返回 null 来表示它们没有执行任何更新，并且对该行剩余的操作应被跳过（即后续的触发器不会被引发，并且该行不会被计入外围INSERT/UPDATE/DELETE的行影响状态中）。否则应该返回一个非 null 值用以表示该触发器执行了所请求的操作。对于INSERT 和UPDATE操作，返回值应该是NEW，触发器函数可能对它进行了修改来支持INSERT RETURNING和UPDATE RETURNING（这也将影响被传递给任何后续触发器的行值）。对于DELETE操作，返回值应该是OLD。

一个AFTER行级触发器或一个BEFORE或AFTER语句级触发器的返回值总是会被忽略，因此也可以返回 null。不过，任何这些类型的触发器可能仍会通过抛出一个错误来中止整个操作。

例 39.3展示了PL/pgSQL中一个触发器函数的示例。

例 39.3. 一个 PL/pgSQL 触发器函数

这个示例触发器保证：任何时候一个行在表中被插入或更新时，当前用户名和时间也会被标记在该行中。并且它会检查给出了一个雇员的姓名以及薪水是一个正值。

```
CREATE TABLE emp (
    empname text,
    salary integer,
    last_date timestamp,
    last_user text
);

CREATE FUNCTION emp_stamp() RETURNS trigger AS $emp_stamp$
    BEGIN
        -- Check that empname and salary are given
        IF NEW.empname IS NULL THEN
            RAISE EXCEPTION 'empname cannot be null';
        END IF;
        IF NEW.salary IS NULL THEN
            RAISE EXCEPTION '% cannot have null salary', NEW.
empname;
        END IF;

        -- Who works for us when she must pay for it?
        IF NEW.salary < 0 THEN
            RAISE EXCEPTION '% cannot have a negative salary', NEW.
empname;
        END IF;

        -- Remember who changed the payroll when
        NEW.last_date := current_timestamp;
        NEW.last_user := current_user;
        RETURN NEW;
    END;
```

```
$emp_stamp$ LANGUAGE plpgsql;

CREATE TRIGGER emp_stamp BEFORE INSERT OR UPDATE ON emp
    FOR EACH ROW EXECUTE PROCEDURE emp_stamp();
```

另一种记录对表的改变的方法涉及到创建一个新表来为每一个发生的插入、更新或删除保持一行。这种方法可以被认为是对一个表的改变的审计。例 39.4展示了PL/pgSQL中一个审计触发器函数的示例。

例 39.4. 一个用于审计的 PL/pgSQL 触发器函数

这个示例触发器保证了在emp表上的任何插入、更新或删除一行的动作都被记录（即审计）在emp_audit表中。当前时间和用户名会被记录到行中，还有在其上执行的操作类型。

```
CREATE TABLE emp (
    empname           text NOT NULL,
    salary            integer
);

CREATE TABLE emp_audit(
    operation         char(1)   NOT NULL,
    stamp             timestamp NOT NULL,
    userid            text      NOT NULL,
    empname           text      NOT NULL,
    salary integer
);

CREATE OR REPLACE FUNCTION process_emp_audit() RETURNS TRIGGER AS
 $emp_audit$
    BEGIN
        --
        -- Create a row in emp_audit to reflect the operation
 performed on emp,
        -- make use of the special variable TG_OP to work out the
 operation.
        --
        IF (TG_OP = 'DELETE') THEN
            INSERT INTO emp_audit SELECT 'D', now(), user, OLD.*;
            RETURN OLD;
        ELSIF (TG_OP = 'UPDATE') THEN
            INSERT INTO emp_audit SELECT 'U', now(), user, NEW.*;
            RETURN NEW;
        ELSIF (TG_OP = 'INSERT') THEN
            INSERT INTO emp_audit SELECT 'I', now(), user, NEW.*;
            RETURN NEW;
        END IF;
        RETURN NULL; -- result is ignored since this is an AFTER
 trigger
    END;
$emp_audit$ LANGUAGE plpgsql;

CREATE TRIGGER emp_audit
AFTER INSERT OR UPDATE OR DELETE ON emp
    FOR EACH ROW EXECUTE PROCEDURE process_emp_audit();
```

前一个示例的一个变体使用视图把主表和审计表连接起来，以显示每个条目最后一次被修改的时间。这种方法仍然记录了表更改的完整审计轨迹，同时也提供了一个简化视图，只显示从审计轨迹中导出的每个条目的最后修改时间戳。例 39.5展示了PL/pgSQL中一个定义在视图上的审计触发器示例。

例 39.5. 一个用于审计的 PL/pgSQL 视图触发器函数

这个示例在视图上使用了一个触发器让它变得可更新，并且确保视图中一行的任何插入、更新或删除被记录（即审计）在emp_audit表中。
当前时间和用户名会被与执行的操作类型一起记录，
并且该视图会显示每一行的最后修改时间。

```
CREATE TABLE emp (
    empname           text PRIMARY KEY,
    salary            integer
);

CREATE TABLE emp_audit(
    operation         char(1)   NOT NULL,
    userid            text      NOT NULL,
    empname           text      NOT NULL,
    salary            integer,
    stamp             timestamp NOT NULL
);

CREATE VIEW emp_view AS
    SELECT e.empname,
           e.salary,
           max(ea.stamp) AS last_updated
      FROM emp e
      LEFT JOIN emp_audit ea ON ea.empname = e.empname
     GROUP BY 1, 2;

CREATE OR REPLACE FUNCTION update_emp_view() RETURNS TRIGGER AS $$
    BEGIN
        --
        -- Perform the required operation on emp, and create a row
 in emp_audit
        -- to reflect the change made to emp.
        --
        IF (TG_OP = 'DELETE') THEN
            DELETE FROM emp WHERE empname = OLD.empname;
            IF NOT FOUND THEN RETURN NULL; END IF;

            OLD.last_updated = now();
            INSERT INTO emp_audit VALUES('D', user, OLD.*);
            RETURN OLD;
        ELSIF (TG_OP = 'UPDATE') THEN
            UPDATE emp SET salary = NEW.salary WHERE empname = OLD.
empname;
            IF NOT FOUND THEN RETURN NULL; END IF;

            NEW.last_updated = now();
            INSERT INTO emp_audit VALUES('U', user, NEW.*);
            RETURN NEW;
        ELSIF (TG_OP = 'INSERT') THEN
            INSERT INTO emp VALUES(NEW.empname, NEW.salary);

            NEW.last_updated = now();
            INSERT INTO emp_audit VALUES('I', user, NEW.*);
            RETURN NEW;
        END IF;
    END;
```

```
$$ LANGUAGE plpgsql;

CREATE TRIGGER emp_audit
INSTEAD OF INSERT OR UPDATE OR DELETE ON emp_view
    FOR EACH ROW EXECUTE PROCEDURE update_emp_view();
```

触发器的一种用途是维护另一个表的汇总表。
得到的汇总结果可以在某些查询中替代原始表使用，而且通常能大幅缩短运行时间。
这种技术在数据仓库中很常见，其中用于度量或观察数据的表（称为事实表）
可能会非常大。例　39.6展示了PL/pgSQL中一个为数据仓库事实表维护汇总表的触发器函数示例。

例 39.6. 一个用于维护汇总表的 PL/pgSQL 触发器函数

这里详述的模式有一部分是基于 Ralph Kimball 所作的数据仓库工具包中的Grocery Store示例。

```
--
-- Main tables - time dimension and sales fact.
--
CREATE TABLE time_dimension (
    time_key                    integer NOT NULL,
    day_of_week                 integer NOT NULL,
    day_of_month                integer NOT NULL,
    month                       integer NOT NULL,
    quarter                     integer NOT NULL,
    year                        integer NOT NULL
);
CREATE UNIQUE INDEX time_dimension_key ON time_dimension(time_key);

CREATE TABLE sales_fact (
    time_key                    integer NOT NULL,
    product_key                 integer NOT NULL,
    store_key                   integer NOT NULL,
    amount_sold                 numeric(12,2) NOT NULL,
    units_sold                  integer NOT NULL,
    amount_cost                 numeric(12,2) NOT NULL
);
CREATE INDEX sales_fact_time ON sales_fact(time_key);

--
-- Summary table - sales by time.
--
CREATE TABLE sales_summary_bytime (
    time_key                    integer NOT NULL,
    amount_sold                 numeric(15,2) NOT NULL,
    units_sold                  numeric(12) NOT NULL,
    amount_cost                 numeric(15,2) NOT NULL
);
CREATE UNIQUE INDEX sales_summary_bytime_key ON sales_summary_
bytime(time_key);

--
-- Function and trigger to amend summarized column(s) on UPDATE,
 INSERT, DELETE.
--
CREATE OR REPLACE FUNCTION maint_sales_summary_bytime() RETURNS
 TRIGGER
AS $maint_sales_summary_bytime$
```

```
   DECLARE
       delta_time_key          integer;
       delta_amount_sold       numeric(15,2);
       delta_units_sold        numeric(12);
       delta_amount_cost       numeric(15,2);
   BEGIN

       -- Work out the increment/decrement amount(s).
       IF (TG_OP = 'DELETE') THEN

           delta_time_key = OLD.time_key;
           delta_amount_sold = -1 * OLD.amount_sold;
           delta_units_sold = -1 * OLD.units_sold;
           delta_amount_cost = -1 * OLD.amount_cost;

       ELSIF (TG_OP = 'UPDATE') THEN

           -- forbid updates that change the time_key -
           -- (probably not too onerous, as DELETE + INSERT is how
most
           -- changes will be made).
           IF ( OLD.time_key != NEW.time_key) THEN
               RAISE EXCEPTION 'Update of time_key : % -> % not
allowed',
                                                       OLD.time_key,
NEW.time_key;
           END IF;

           delta_time_key = OLD.time_key;
           delta_amount_sold = NEW.amount_sold - OLD.amount_sold;
           delta_units_sold = NEW.units_sold - OLD.units_sold;
           delta_amount_cost = NEW.amount_cost - OLD.amount_cost;

       ELSIF (TG_OP = 'INSERT') THEN

           delta_time_key = NEW.time_key;
           delta_amount_sold = NEW.amount_sold;
           delta_units_sold = NEW.units_sold;
           delta_amount_cost = NEW.amount_cost;

       END IF;


       -- Insert or update the summary row with the new values.
       <<insert_update>>
       LOOP
           UPDATE sales_summary_bytime
               SET amount_sold = amount_sold + delta_amount_sold,
                   units_sold = units_sold + delta_units_sold,
                   amount_cost = amount_cost + delta_amount_cost
               WHERE time_key = delta_time_key;

           EXIT insert_update WHEN found;

           BEGIN
               INSERT INTO sales_summary_bytime (
                           time_key,
                           amount_sold,
```

```
                        units_sold,
                        amount_cost)
                VALUES (
                        delta_time_key,
                        delta_amount_sold,
                        delta_units_sold,
                        delta_amount_cost
                       );

            EXIT insert_update;

        EXCEPTION
            WHEN UNIQUE_VIOLATION THEN
                -- do nothing
        END;
    END LOOP insert_update;

    RETURN NULL;

    END;
$maint_sales_summary_bytime$ LANGUAGE plpgsql;

CREATE TRIGGER maint_sales_summary_bytime
AFTER INSERT OR UPDATE OR DELETE ON sales_fact
    FOR EACH ROW EXECUTE PROCEDURE maint_sales_summary_bytime();

INSERT INTO sales_fact VALUES(1,1,1,10,3,15);
INSERT INTO sales_fact VALUES(1,2,1,20,5,35);
INSERT INTO sales_fact VALUES(2,2,1,40,15,135);
INSERT INTO sales_fact VALUES(2,3,1,10,1,13);
SELECT * FROM sales_summary_bytime;
DELETE FROM sales_fact WHERE product_key = 1;
SELECT * FROM sales_summary_bytime;
UPDATE sales_fact SET units_sold = units_sold * 2;
SELECT * FROM sales_summary_bytime;
```

39.10. PL/pgSQL 内部机制

本节讨论一些实现细节，了解这些细节对 PL/pgSQL 用户通常很重要。

39.10.1. 变量替换

在PL/pgSQL函数中，SQL 语句和表达式可以引用函数的变量和参数。在内部，PL/pgSQL会用查询参数替换这些引用。只有在语法上允许参数或列引用的位置才会进行参数替换。
作为一个极端例子，来看下面这种不良编程风格：

```
INSERT INTO foo (foo) VALUES (foo);
```

第一次出现的`foo`在语法上必须是表名，所以不会被替换，即使函数有一个变量也叫`foo`。
第二次出现的位置必须是该表的列名，所以也不会被替换。
只有第三次出现的位置才可能是对函数变量的引用。

> **注意**
>
> PostgreSQL 9.0 以前的版本会尝试在这三个位置都进行变量替换，从而导致语法错误。

由于变量名在语法上与表列名没有区别，
所以在同时引用表的语句中就可能产生歧义：某个给定名称到底是指表列，
还是变量？把前面的示例改成下面这样：

```
INSERT INTO dest (col) SELECT foo + bar FROM src;
```

这里，`dest` 和 `src` 必须是表名，`col` 也必须是 `dest` 的一列，但 `foo` 和 `bar` 既可能是该函数的变量，也可能是 `src` 的列。

默认情况下，如果 SQL 语句中的名称既可能指变量，也可能指表列，PL/pgSQL就会报告错误。可以通过重命名变量或列、限定有歧义的引用，或者告诉
PL/pgSQL优先采用哪种解释来解决这类问题。

最简单的解决方案是重命名变量或列。
一种常用的编码规则是为PL/pgSQL变量使用一种不同于列名的命名习惯。例如，
如果你将函数变量统一地命名为`v_`*`something`*，而你的列名不会开始于`v_`，
就不会发生冲突。

另外你可以限定有歧义的引用让它们变清晰。在上面的示例中，`src.foo`
将是对表列的一种无歧义的引用。要创建对一个变量的无歧义引用，
在一个被标记的块中声明它并且使用块的标签（见第 39.2 节）。例如

```
<<block>>
DECLARE
    foo int;
BEGIN
    foo := ...;
    INSERT INTO dest (col) SELECT block.foo + bar FROM src;
```

这里`block.foo`表示变量，即使在`src`中有一个列`foo`。函数参数以及诸如`FOUND`
的特殊变量，都能通过函数的名称被限定，
因为它们被隐式地声明在一个带有该函数名称的外层块中。

有时候在一个大型的PL/pgSQL代码体中修复所有的有歧义引用是不现实的。在这种情况下，
你可以指定PL/pgSQL应该将有歧义的引用解析为变量（这与PL/pgSQL在 PostgreSQL 9.0
之前的行为兼容）或表列（这与某些其他系统兼容，例如Oracle）。

要在系统范围内改变这种行为，将配置参数`plpgsql.variable_conflict`
设置为`error`、`use_variable`或者`use_column`（这里`error`是出厂设置）之一。
这个参数会影响PL/pgSQL函数中语句的后续编译，
但是不会影响在当前会话中已经编译过的语句。因为改变这个设置能够导致
PL/pgSQL函数中行为的意想不到的改变，所以只能由一个超级用户来更改它。

你也可以按函数单独设置这一行为，方法是在函数文本开头插入以下特殊命令之一：

```
#variable_conflict error
#variable_conflict use_variable
#variable_conflict use_column
```

这些命令只影响它们所属的函数，并且会覆盖`plpgsql.variable_conflict`的设置。
一个示例是：

```
CREATE FUNCTION stamp_user(id int, comment text) RETURNS void AS $$
    #variable_conflict use_variable
    DECLARE
        curtime timestamp := now();
    BEGIN
        UPDATE users SET last_modified = curtime, comment = comment
          WHERE users.id = id;
```

```
    END;
$$ LANGUAGE plpgsql;
```

在UPDATE命令中，curtime、comment以及id将引用该函数的变量和参数，不管users有没有这些名称的列。注意，我们不得不在WHERE子句中对users.id的引用加以限定，以便让它引用表列。但是我们不需要对UPDATE列表中作为目标的comment引用加以限定，因为语法上那必须是users的一列。
我们可以用下面的方式写一个相同的不依赖于variable_conflict设置的函数：

```
CREATE FUNCTION stamp_user(id int, comment text) RETURNS void AS $$
    <<fn>>
    DECLARE
        curtime timestamp := now();
    BEGIN
        UPDATE users SET last_modified = fn.curtime, comment =
 stamp_user.comment
          WHERE users.id = stamp_user.id;
    END;
$$ LANGUAGE plpgsql;
```

传递给 EXECUTE 及其变体的命令字符串中，不会发生变量替换。如果你需要向这种命令中插入变化的值，应在构造字符串值时完成，或者像第 39.5.4 节所说明的那样使用 USING。

目前，变量替换只在SELECT、INSERT、UPDATE和DELETE命令中生效，因为主 SQL 引擎只允许在这些命令中使用查询参数。若要在其他语句类型（统称为工具语句）中使用非常量名称或值，就必须把该工具语句构造为字符串，再用EXECUTE执行。

39.10.2. 计划缓存

在函数第一次被调用时（每个会话中都会如此），PL/pgSQL解释器会解析函数源文本，并生成一棵内部的二进制指令树。该指令树完整表示了PL/pgSQL语句结构，但函数中使用的各个SQL表达式和SQL命令并不会立即被分析。

当函数中的某个表达式或 SQL 命令第一次执行时，PL/pgSQL 解释器会使用 SPI 管理器的 SPI_prepare 函数，对该命令进行解析和分析，以创建预备语句。之后再次执行该表达式或命令时，就会重用该预备语句。因此，对于那些很少访问的条件分支，函数不会承担分析当前会话中从未执行到的命令的开销。缺点是，某个具体表达式或命令中的错误，只有在执行到函数的那一部分时才能被发现。（简单语法错误会在最初的解析阶段发现，但更深层的问题只有在执行时才会显现。）

PL/pgSQL（更准确地说，是 SPI 管理器）还可以尝试缓存与某个预备语句相关的执行计划。如果没有使用缓存计划，那么每次执行该语句时都会生成新的执行计划，而当前参数值（也就是 PL/pgSQL 变量值）可用于优化所选计划。如果该语句没有参数，或者会被执行很多次，SPI 管理器就会考虑创建一个不依赖具体参数值的通用计划，并将其缓存起来供重复使用。通常只有在执行计划对其中引用的 PL/pgSQL 变量值不太敏感时，才会这样做。如果执行计划对参数值非常敏感，那么每次重新生成计划总体上反而更划算。关于预备语句的行为，详

由于PL/pgSQL保存预备语句并且有时候以这种方式保存执行计划，直接出现在一个PL/pgSQL函数中的 SQL 命令必须在每次执行时引用相同的表和列。也就是说，你不能在一个 SQL 命令中把一个参数用作表或列的名字。要绕过这种限制，你可以构建PL/pgSQL EXECUTE使用的动态命令，但是会付出在每次执行时需要执行新解析分析以及构建新执行计划的代价。

记录变量的可变特性在这里还会带来另一个问题。当记录变量的字段被用于表达式或语句中时，这些字段的数据类型不能在函数的不同调用之间发生变化，因为每个表达式都会按照第一次执行到它时所看到的数据类型来分析。必要时，可以用EXECUTE绕过这个问题。

如果同一个函数被用作多个表的触发器，
PL/pgSQL会针对每个这样的表独立地准备并缓存语句。也就是说，缓存是按“触发器函数 + 表”的组合建立的，而不是每个函数只有一个缓存。这缓解了数据类型变化带来的部分问题；例如，即使不同表中名为key的列类型不同，一个触发器函数也仍然能够成功使用它。

同样，
具有多态参数类型的函数也会为它们已经被调用的每一种实参类型组合都保留一个独立的缓存，这样数据类型差异不会导致意想不到的失败。

语句缓存有时可能在解释时间敏感的值时产生令人惊讶的效果。
例如这两个函数做的事情就有区别：

```
CREATE FUNCTION logfunc1(logtxt text) RETURNS void AS $$
    BEGIN
        INSERT INTO logtable VALUES (logtxt, 'now');
    END;
$$ LANGUAGE plpgsql;
```

以及：

```
CREATE FUNCTION logfunc2(logtxt text) RETURNS void AS $$
    DECLARE
        curtime timestamp;
    BEGIN
        curtime := 'now';
        INSERT INTO logtable VALUES (logtxt, curtime);
    END;
$$ LANGUAGE plpgsql;
```

在logfunc1中，PostgreSQL的主解析器在分析INSERT时就知道字符串'now'
应该被解释为timestamp，因为logtable的目标列是这种类型。因此，在INSERT
被分析时'now'将被转换为一个timestamp常量，并且在该会话的生命周期内被用于所有对logfunc1的调用。不用说，这不是程序员想要的。一个更好的主意是使用now()
或current_timestamp函数。

在logfunc2中，PostgreSQL的主解析器不知道应该把'now'解释成什么类型，
因此返回一个text类型的数据值，其中包含字符串now。在随后给局部变量curtime
赋值时，PL/pgSQL解释器通过调用用于该转换的text_out和timestamp_in函数，
把这个字符串转换为timestamp类型。因此，
计算得到的时间戳会按程序员预期在每次执行时更新。虽然这恰好能得到预期结果，
但效率并不高，因此使用now()函数仍然是更好的主意。

39.11. PL/pgSQL开发提示

在PL/pgSQL中进行开发的一种好方法，是用你喜欢的文本编辑器编写函数，
并在另一个窗口里使用psql载入和测试这些函数。如果你采用这种方式，最好使用CREATE OR REPLACE FUNCTION来编写函数。这样只需重新载入该文件，就可以更新函数定义。例如：

```
CREATE OR REPLACE FUNCTION testfunc(integer) RETURNS integer AS $$
          ....
$$ LANGUAGE plpgsql;
```

在运行psql期间，你可以用下面的命令载入或者重载这样一个函数定义文件：

```
\i filename.sql
```

并且接着立即发出 SQL 命令来测试该函数。

另一种使用PL/pgSQL进行开发的好方法，是使用有助于过程语言开发的 GUI 数据库访问工具。这类工具的一个例子是pgAdmin，当然也有其他工具。它们通常提供一些方便的特性，例如转义单引号，以及让重新创建和调试函数变得更容易。

39.11.1. 引号的处理

一个PL/pgSQL函数的代码在一个`CREATE FUNCTION`中被指定为一个字符串字面量。如果你用通常的方式把该字符串写在单引号中间，那么该函数体中的任何单引号都必须被双写；同样任何反斜线也必须被双写（假定使用了转义字符串语法）。双写引号本身就很繁琐，并且在更复杂的情况中代码会变得完全无法理解，因为你很容易发现你需要半打或者更多相邻的引号。
我们推荐你转而把函数体写成一个“美元引用”的字符串（见第 4.1.2.4 节）。在美元引用方法中，你从不需要双写任何引号。
但是要注意为你需要的每一层嵌套选择一个不同的美元引用定界符。例如，你可能把`CREATE FUNCTION`命令写成：

```
CREATE OR REPLACE FUNCTION testfunc(integer) RETURNS integer AS
 $PROC$
          ....
$PROC$ LANGUAGE plpgsql;
```

在这里面，你可以在 SQL 命令中为简单字符串使用引号并且用`$$`来界定被你组装成字符串的 SQL 命令片段。如果你需要引用包括`$$`的文本，你可以使用`$Q$`等等。

下表展示了在不使用美元引用时，编写引号必须采取的做法。
在把旧式的非美元引用代码改写成更易理解的形式时，它可能会有所帮助。

1 个引号

用来开始和结束函数体，例如：

```
CREATE FUNCTION foo() RETURNS integer AS '
          ....
' LANGUAGE plpgsql;
```

在一个单引号引用的函数体中的任何位置，引号必须成对出现。

2 个引号

用于函数体内的字符串字面量，例如：

```
a_output := ''Blah'';
SELECT * FROM users WHERE f_name=''foobar'';
```

在美元引用方法中，你只需要写：

```
a_output := 'Blah';
SELECT * FROM users WHERE f_name='foobar';
```

这恰好就是PL/pgSQL解析器在两种情况中会看到的。

4 个引号

当你在函数内的一个字符串常量中需要一个单引号时，例如：

```
a_output := a_output || '' AND name LIKE ''''foobar'''' AND
 xyz''
```

实际会被追加到`a_output`的值将是：`AND name LIKE 'foobar' AND xyz`。

在美元引用方法中，你可以写：

```
a_output := a_output || $$ AND name LIKE 'foobar' AND xyz$$
```

要小心在这周围的任何美元引用定界符都不能是`$$`。

6 个引号

当在函数体内的一个字符串中的一个单引号与该字符串常量末尾相邻，例如：

```
a_output := a_output || '' AND name LIKE ''''foobar''''''
```

被追加到`a_output`的值则将是：`AND name LIKE 'foobar'`。

在美元引用方法中，这会变成：

```
a_output := a_output || $$ AND name LIKE 'foobar'$$
```

10 个引号

当字符串常量中需要两个单引号（这需要 8 个引号），而且它们紧邻该字符串常量的末尾（还需 2 个引号）时。通常只有在编写生成其他函数的函数时（如例 39.8所示），才会需要这种写法。例如：

```
a_output := a_output || '' if v_'' ||
    referrer_keys.kind || '' like '''''''''
    || referrer_keys.key_string || '''''''''
    then return ''''''  || referrer_keys.referrer_type
    || ''''''; end if;'';
```

`a_output`的值将是：

```
if v_... like ''...'' then return ''...''; end if;
```

在美元引用方法中，这会变成：

```
a_output := a_output || $$ if v_$$ || referrer_keys.kind || $$
 like '$$
    || referrer_keys.key_string || $$'
    then return '$$  || referrer_keys.referrer_type
    || $$'; end if;$$;
```

这里假定只需在`a_output`中放入单引号，因为在使用前会对它再次加引号。

39.12. 从Oracle PL/SQL 移植

这一节解释了PostgreSQL的PL/pgSQL语言和 Oracle 的PL/SQL语言之间的差别，用以帮助那些从Oracle®向PostgreSQL移植应用的人。

PL/pgSQL在许多方面都与 PL/SQL 类似。它是一种具有块结构的命令式语言，所有变量都必须声明。赋值、循环和条件语句也都很相似。在从PL/SQL移植到PL/pgSQL时，
应当记住以下主要差异：

- 如果 SQL 命令中使用的名字既可能是表的列名，也可能是对函数中变量的引用，PL/SQL会将其当作列名。这对应于PL/pgSQL的`plpgsql.variable_conflict = use_column`行为，这不是默认行为，如第 39.10.1 节所述。通常最好一开始就避免这种歧义，但如果必须移植大量依赖这种行为的代码，设置`variable_conflict`可能是最好的解决办法。

- 在PostgreSQL中，函数体必须写成字符串字面量。
  因此你需要使用美元符引用或者转义函数体中的单引号（见第 39.11.1 节）。

- 应该用模式把函数组织成不同的分组，而不是用包。
- 因为没有包，所以也没有包级变量。这一点有时会带来不便。
  你可以改为在临时表中保存会话级状态。
- 带有REVERSE的整数FOR循环的工作方式不同：PL/SQL中是从第二个数向第一个数倒数，而PL/pgSQL是从第一个数向第二个数倒数，因此在移植时需要交换循环边界。
  不幸的是这种不兼容性是不太可能改变的（见第 39.6.3.5 节）。
- 查询上的FOR循环（不是游标）的工作方式同样不同：目标变量必须已经被声明，而PL/SQL总是会隐式地声明它们。但是这样做的优点是在退出循环后，
  变量值仍然可以访问。
- 应该用模式把函数组织成不同的分组，而不是用包。

39.12.1. 移植示例

例 39.7展示了如何从PL/SQL移植一个简单的函数到PL/pgSQL中。

例 39.7. 把一个简单函数从 PL/SQL 移植到 PL/pgSQL

这里有一个Oracle PL/SQL函数：

```
CREATE OR REPLACE FUNCTION cs_fmt_browser_version(v_name varchar,
                                                  v_version
 varchar)
RETURN varchar IS
BEGIN
    IF v_version IS NULL THEN
        RETURN v_name;
    END IF;
    RETURN v_name || '/' || v_version;
END;
/
show errors;
```

让我们过一遍这个函数并且看看与PL/pgSQL相比有什么样的不同：

- 在函数原型中（不是函数体中）的RETURN关键字在PostgreSQL中变成了RETURNS。
  还有，IS变成了AS，并且你还需要增加一个LANGUAGE子句，因为
  PL/pgSQL并非唯一可用的函数语言。
- 在PostgreSQL中，函数体被认为是一个字符串字面量，
  所以你需要使用引号或者美元引用定界符包围它。这代替了Oracle 方法中的用于终止的/。
- 在PostgreSQL中没有show errors命令，并且也不需要这个命令，因为错误是自动报告的。

这个函数被移植到PostgreSQL后看起来会是这样：

```
CREATE OR REPLACE FUNCTION cs_fmt_browser_version(v_name varchar,
                                                  v_version
 varchar)
RETURNS varchar AS $$
BEGIN
    IF v_version IS NULL THEN
        RETURN v_name;
    END IF;
    RETURN v_name || '/' || v_version;
```

```
END;
$$ LANGUAGE plpgsql;
```

例 39.8展示了如何移植一个会创建另一个函数的函数，以及如何处理引号问题。

例 39.8. 把一个创建其它函数的函数从 PL/SQL 移植到 PL/pgSQL

下面的过程从 `SELECT` 语句读取行，并将结果写入 `IF` 语句，从而构造一个大型函数，以提高效率。

这是 Oracle 版本：

```
CREATE OR REPLACE PROCEDURE cs_update_referrer_type_proc IS
    CURSOR referrer_keys IS
        SELECT * FROM cs_referrer_keys
        ORDER BY try_order;
    func_cmd VARCHAR(4000);
BEGIN
    func_cmd := 'CREATE OR REPLACE FUNCTION cs_find_referrer_type(
v_host IN VARCHAR,
                 v_domain IN VARCHAR, v_url IN VARCHAR) RETURN
 VARCHAR IS BEGIN';

    FOR referrer_key IN referrer_keys LOOP
        func_cmd := func_cmd ||
          ' IF v_' || referrer_key.kind
          || ' LIKE ''' || referrer_key.key_string
          || ''' THEN RETURN ''' || referrer_key.referrer_type
          || '''; END IF;';
    END LOOP;

    func_cmd := func_cmd || ' RETURN NULL; END;';

    EXECUTE IMMEDIATE func_cmd;
END;
/
show errors;
```

下面是这个函数的最终移植结果，目标数据库为PostgreSQL：

```
CREATE OR REPLACE FUNCTION cs_update_referrer_type_proc() RETURNS
 void AS $func$
DECLARE
    referrer_keys CURSOR IS
        SELECT * FROM cs_referrer_keys
        ORDER BY try_order;
    func_body text;
    func_cmd text;
BEGIN
    func_body := 'BEGIN';

    FOR referrer_key IN referrer_keys LOOP
        func_body := func_body ||
          ' IF v_' || referrer_key.kind
          || ' LIKE ' || quote_literal(referrer_key.key_string)
          || ' THEN RETURN ' || quote_literal(referrer_key.
referrer_type)
          || '; END IF;' ;
    END LOOP;
```

```
    func_body := func_body || ' RETURN NULL; END;';

    func_cmd :=
      'CREATE OR REPLACE FUNCTION cs_find_referrer_type(v_host
 varchar,
                                                        v_domain
 varchar,
                                                        v_url
 varchar)
        RETURNS varchar AS '
      || quote_literal(func_body)
      || ' LANGUAGE plpgsql;' ;

    EXECUTE func_cmd;
END;
$func$ LANGUAGE plpgsql;
```

注意，这里单独构造函数体，然后将其传给quote_literal，
使其中的每个引号都变成两个。这种技术是必需的，
因为不能安全地使用美元引用来定义新函数：我们无法确定会插入什么字符串，其来源是referrer_key.key_string字段。（这里假定referrer_key.kind可信，其值总是host，domain或url，但是referrer_key.key_string可能是任何内容，
尤其可能包含美元符号。）这个函数实际上改进了 Oracle 原版：当referrer_key.key_string或referrer_key.referrer_type中包含引号时，它也不会生成有问题的代码。

例 39.9展示了如何移植一个带有OUT参数和字符串处理的函数。PostgreSQL没有内置的instr函数，但是你可以用其它函数的组合来创建一个。在第 39.12.3 节中有一个instr的PL/pgSQL实现，你可以用它让你的移植变得更容易。

例 39.9. 把一个带字符串处理和 OUT 参数的过程从 PL/SQL 移植到 PL/pgSQL

下面的Oracle PL/SQL 过程被用来解析一个 URL 并且返回一些元素（主机、路径和查询）。

这是 Oracle 版本：

```
CREATE OR REPLACE PROCEDURE cs_update_referrer_type_proc IS
    CURSOR referrer_keys IS
        SELECT * FROM cs_referrer_keys
        ORDER BY try_order;
    func_cmd VARCHAR(4000);
BEGIN
    func_cmd := 'CREATE OR REPLACE FUNCTION cs_find_referrer_type(
v_host IN VARCHAR,
                 v_domain IN VARCHAR, v_url IN VARCHAR) RETURN
 VARCHAR IS BEGIN';

    FOR referrer_key IN referrer_keys LOOP
        func_cmd := func_cmd ||
          ' IF v_' || referrer_key.kind
          || ' LIKE ''' || referrer_key.key_string
          || ''' THEN RETURN ''' || referrer_key.referrer_type
          || '''; END IF;';
    END LOOP;

    func_cmd := func_cmd || ' RETURN NULL; END;';

    EXECUTE IMMEDIATE func_cmd;
END;
```

```
/
show errors;
```

这里给出一种可能的 PL/pgSQL 写法：

```
CREATE OR REPLACE FUNCTION cs_parse_url(
    v_url IN VARCHAR,
    v_host OUT VARCHAR,  -- 这个值将被返回
    v_path OUT VARCHAR,  -- 这个也是
    v_query OUT VARCHAR) -- 还有这个
AS $$
DECLARE
    a_pos1 INTEGER;
    a_pos2 INTEGER;
BEGIN
    v_host := NULL;
    v_path := NULL;
    v_query := NULL;
    a_pos1 := instr(v_url, '//');

    IF a_pos1 = 0 THEN
        RETURN;
    END IF;
    a_pos2 := instr(v_url, '/', a_pos1 + 2);
    IF a_pos2 = 0 THEN
        v_host := substr(v_url, a_pos1 + 2);
        v_path := '/';
        RETURN;
    END IF;

    v_host := substr(v_url, a_pos1 + 2, a_pos2 - a_pos1 - 2);
    a_pos1 := instr(v_url, '?', a_pos2 + 1);

    IF a_pos1 = 0 THEN
        v_path := substr(v_url, a_pos2);
        RETURN;
    END IF;

    v_path := substr(v_url, a_pos2, a_pos1 - a_pos2);
    v_query := substr(v_url, a_pos1 + 1);
END;
$$ LANGUAGE plpgsql;
```

这个函数可以这样使用：

```
SELECT * FROM cs_parse_url('http://foobar.com/query.cgi?baz');
```

例 39.10展示了如何移植一个使用了多种 Oracle 专属特性的过程。

例 39.10. 把一个过程从 PL/SQL 移植到 PL/pgSQL

Oracle 版本：

```
CREATE OR REPLACE PROCEDURE cs_create_job(v_job_id IN INTEGER) IS
    a_running_job_count INTEGER;
    PRAGMA AUTONOMOUS_TRANSACTION;1
BEGIN
    LOCK TABLE cs_jobs IN EXCLUSIVE MODE;2
```

```
    SELECT count(*) INTO a_running_job_count FROM cs_jobs WHERE
 end_stamp IS NULL;

    IF a_running_job_count > 0 THEN
        COMMIT; -- free lock❸
        raise_application_error(-20000,
                 'Unable to create a new job: a job is currently
 running.');
    END IF;

    DELETE FROM cs_active_job;
    INSERT INTO cs_active_job(job_id) VALUES (v_job_id);

    BEGIN
        INSERT INTO cs_jobs (job_id, start_stamp) VALUES (v_job_id,
 now());
    EXCEPTION
        WHEN dup_val_on_index THEN NULL; -- don't worry if it
 already exists
    END;
    COMMIT;
END;
/
show errors
```

这样的过程可以很容易地转换为PostgreSQL中返回`void`的函数。这个过程尤其值得关注，因为它能说明以下几点：

❶ PostgreSQL中没有`PRAGMA`语句。
❷ 如果在PL/pgSQL中执行`LOCK TABLE`，锁要等到调用事务结束时才会释放。
❸ 不能在PL/pgSQL函数中发出`COMMIT`。函数运行在某个外层事务之中，因此`COMMIT`意味着终止函数的执行。不过，在这个特定情形中本来也不需要提交，因为抛出错误时，`LOCK TABLE`获得的锁就会被释放。

下面展示了如何移植这个过程，目标语言为PL/pgSQL：

```
CREATE OR REPLACE FUNCTION cs_create_job(v_job_id integer) RETURNS
 void AS $$
DECLARE
    a_running_job_count integer;
BEGIN
    LOCK TABLE cs_jobs IN EXCLUSIVE MODE;

    SELECT count(*) INTO a_running_job_count FROM cs_jobs WHERE
 end_stamp IS NULL;

    IF a_running_job_count > 0 THEN
        RAISE EXCEPTION 'Unable to create a new job: a job is
 currently running';❶
    END IF;

    DELETE FROM cs_active_job;
    INSERT INTO cs_active_job(job_id) VALUES (v_job_id);

    BEGIN
        INSERT INTO cs_jobs (job_id, start_stamp) VALUES (v_job_id,
 now());
    EXCEPTION
        WHEN unique_violation THEN ❷
```

```
            -- don't worry if it already exists
    END;
END;
$$ LANGUAGE plpgsql;
```

1. RAISE的语法与 Oracle 的语句相当不同，尽管基本的形式RAISE *exception_name*工作起来是相似的。
2. PL/pgSQL所支持的异常名称不同于 Oracle。内置的异常名称集合要更大（见附录 A）。目前没有办法声明用户定义的异常名称，尽管你能够抛出用户选择的 SQLSTATE 值。

The main functional difference between this procedure and the Oracle equivalent is that the exclusive lock on the cs_jobs table will be held until the calling transaction completes. Also, if the caller later aborts (for example due to an error), the effects of this procedure will be rolled back.

39.12.2. 其他要关注的事项

这一节解释了在移植 Oracle PL/SQL函数到PostgreSQL中时要关注的一些其他问题。

39.12.2.1. 异常后隐式回滚

在PL/pgSQL中，当异常被EXCEPTION子句捕获后，自该块的BEGIN
以来所做的所有数据库更改都会被自动回滚。也就是说，这种行为等效于你在 Oracle 中使用下面的代码所得到的效果：

```
BEGIN
    SAVEPOINT s1;
    ... code here ...
EXCEPTION
    WHEN ... THEN
        ROLLBACK TO s1;
        ... code here ...
    WHEN ... THEN
        ROLLBACK TO s1;
        ... code here ...
END;
```

如果你正在移植一个以这种方式使用SAVEPOINT和ROLLBACK TO的 Oracle 过程，工作会比较简单：只要省略SAVEPOINT和ROLLBACK TO即可。如果你的 Oracle 过程以不同方式使用SAVEPOINT和ROLLBACK TO，那就需要认真思考了。

39.12.2.2. EXECUTE

PL/pgSQL中的EXECUTE与PL/SQL中的版本工作方式相似，但必须记得按照第 39.5.4 节中的说明使用quote_literal和quote_ident。像 EXECUTE 'SELECT * FROM $1'; 这样的写法如果不借助这些函数，将无法可靠地工作。

39.12.2.3. 优化 PL/pgSQL 函数

PostgreSQL提供了两种函数创建修饰符来优化执行：“volatility”（对于给定的相同参数，函数是否总是返回相同的结果）以及“strictness”（如果任何参数为空值，
函数是否返回空值）。详见CREATE FUNCTION参考页。

在利用这些优化属性时，你的CREATE FUNCTION语句可能像这样：

```
CREATE FUNCTION foo(...) RETURNS integer AS $$
...
$$ LANGUAGE plpgsql STRICT IMMUTABLE;
```

39.12.3. 附录

这一节包含了一组 Oracle 兼容的`instr`函数代码，你可以用它来简化你的移植工作。

```
--
-- instr functions that mimic Oracle's counterpart
-- Syntax: instr(string1, string2, [n], [m]) where [] denotes
 optional parameters.
--
-- Searches string1 beginning at the nth character for the mth
 occurrence
-- of string2.  If n is negative, search backwards.  If m is not
 passed,
-- assume 1 (search starts at first character).
--

CREATE FUNCTION instr(varchar, varchar) RETURNS integer AS $$
DECLARE
    pos integer;
BEGIN
    pos:= instr($1, $2, 1);
    RETURN pos;
END;
$$ LANGUAGE plpgsql STRICT IMMUTABLE;


CREATE FUNCTION instr(string varchar, string_to_search varchar,
 beg_index integer)
RETURNS integer AS $$
DECLARE
    pos integer NOT NULL DEFAULT 0;
    temp_str varchar;
    beg integer;
    length integer;
    ss_length integer;
BEGIN
    IF beg_index > 0 THEN
        temp_str := substring(string FROM beg_index);
        pos := position(string_to_search IN temp_str);

        IF pos = 0 THEN
            RETURN 0;
        ELSE
            RETURN pos + beg_index - 1;
        END IF;
    ELSE
        ss_length := char_length(string_to_search);
        length := char_length(string);
        beg := length + beg_index - ss_length + 2;

        WHILE beg > 0 LOOP
            temp_str := substring(string FROM beg FOR ss_length);
            pos := position(string_to_search IN temp_str);

            IF pos > 0 THEN
                RETURN beg;
            END IF;
```

```
            beg := beg - 1;
        END LOOP;

        RETURN 0;
    END IF;
END;
$$ LANGUAGE plpgsql STRICT IMMUTABLE;


CREATE FUNCTION instr(string varchar, string_to_search varchar,
                      beg_index integer, occur_index integer)
RETURNS integer AS $$
DECLARE
    pos integer NOT NULL DEFAULT 0;
    occur_number integer NOT NULL DEFAULT 0;
    temp_str varchar;
    beg integer;
    i integer;
    length integer;
    ss_length integer;
BEGIN
    IF beg_index > 0 THEN
        beg := beg_index;
        temp_str := substring(string FROM beg_index);

        FOR i IN 1..occur_index LOOP
            pos := position(string_to_search IN temp_str);

            IF i = 1 THEN
                beg := beg + pos - 1;
            ELSE
                beg := beg + pos;
            END IF;

            temp_str := substring(string FROM beg + 1);
        END LOOP;

        IF pos = 0 THEN
            RETURN 0;
        ELSE
            RETURN beg;
        END IF;
    ELSE
        ss_length := char_length(string_to_search);
        length := char_length(string);
        beg := length + beg_index - ss_length + 2;

        WHILE beg > 0 LOOP
            temp_str := substring(string FROM beg FOR ss_length);
            pos := position(string_to_search IN temp_str);

            IF pos > 0 THEN
                occur_number := occur_number + 1;

                IF occur_number = occur_index THEN
                    RETURN beg;
                END IF;
            END IF;
```

```
            beg := beg - 1;
        END LOOP;

        RETURN 0;
    END IF;
END;
$$ LANGUAGE plpgsql STRICT IMMUTABLE;
```

第 40 章 PL/Tcl - Tcl 过程语言

PL/Tcl 是 PostgreSQL 数据库系统的一种可加载过程语言，可以使用 Tcl 语言[1]编写函数和触发器函数。

40.1. 概述

PL/Tcl 提供了函数编写者在 C 语言中所具备的大部分能力，但会施加少量限制，并额外提供 Tcl 可用的强大字符串处理库。

其中一个颇具吸引力的良性限制是，所有内容都在 Tcl 解释器的安全上下文中执行。除了安全 Tcl 受限的命令集之外，只提供了少数通过 SPI 访问数据库以及通过`elog()`发出消息的命令。与 C 函数不同，PL/Tcl 不提供访问数据库服务器内部机制的方法，也不能在PostgreSQL服务器进程的权限下获得操作系统级访问。因此，可以信任非特权数据库用户使用这种语言；它不会授予他们无限制的权限。

另一项值得注意的实现限制是，Tcl 函数不能用来为新数据类型创建输入/输出函数。

有时需要编写不受安全 Tcl 限制的 Tcl 函数。例如，可能希望某个 Tcl 函数能够发送电子邮件。为处理这类情况，PL/Tcl 提供了一个名为`PL/TclU`的变体（其中 U 表示不受信任的 Tcl）。除了使用完整的 Tcl 解释器之外，它与前者是完全相同的语言。如果使用PL/TclU，就必须将其安装为一种不受信任的过程语言，这样就只有数据库超级用户才能在其中创建函数。PL/TclU函数的编写者必须确保该函数不会被用来做任何不希望发生的事情，因为它能够执行任何以数据库管理员身份登录的用户所能做的事情。

如果在安装过程的配置步骤中指定了 Tcl 支持，则PL/Tcl和PL/TclU调用处理器的共享对象代码会自动构建并安装到PostgreSQL的库目录中。要在某个特定数据库中安装PL/Tcl和/或PL/TclU，请使用`CREATE EXTENSION`命令或`createlang`程序，例如`createlang pltcl dbname`或`createlang pltclu dbname`。

40.2. PL/Tcl 函数和参数

要用PL/Tcl语言创建函数，可使用标准的CREATE FUNCTION语法：

```
CREATE FUNCTION funcname (argument-types) RETURNS return-type AS $$
    # PL/Tcl 函数体
$$ LANGUAGE pltcl;
```

PL/TclU的写法相同，只是语言必须指定为`pltclu`。

函数体就是一段 Tcl 脚本。调用函数时，参数值以变量 `$1` ... `$n` 的形式传入 Tcl 脚本。Tcl 代码按通常方式使用 `return` 语句返回结果。

例如，一个返回两个整数值中较大值的函数可以定义为：

```
CREATE FUNCTION tcl_max(integer, integer) RETURNS integer AS $$
    if {$1 > $2} {return $1}
    return $2
$$ LANGUAGE pltcl STRICT;
```

注意`STRICT`子句，它让我们不必考虑空值输入：如果传入的是空值，函数根本不会被调用，而是会自动返回空值结果。

在非严格函数中，如果某个参数的实际值为空值，对应的`$n`变量会被设置为空串。要检测某个特定参数是否为空值，可使用函数`argisnull`。例如，假设我们希望在`tcl_max`的两个参数中一个为空值、一个非空值时返回非空值参数，而不是返回空值：

[1] http://www.tcl.tk/

```
CREATE FUNCTION tcl_max(integer, integer) RETURNS integer AS $$
    if {[argisnull 1]} {
        if {[argisnull 2]} { return_null }
        return $2
    }
    if {[argisnull 2]} { return $1 }
    if {$1 > $2} {return $1}
    return $2
$$ LANGUAGE pltcl;
```

如上所示，要从 PL/Tcl 函数返回空值，请执行`return_null`。无论函数是否为严格函数，都可以这样做。

复合类型参数会作为 Tcl 数组传递给函数。数组元素名就是该复合类型的属性名。如果传入行中的某个属性为空值，它就不会出现在数组中。下面是一个示例：

```
CREATE TABLE employee (
    name text,
    salary integer,
    age integer
);

CREATE FUNCTION overpaid(employee) RETURNS boolean AS $$
    if {200000.0 < $1(salary)} {
        return "t"
    }
    if {$1(age) < 30 && 100000.0 < $1(salary)} {
        return "t"
    }
    return "f"
$$ LANGUAGE pltcl;
```

目前尚不支持返回复合类型的结果值，也不支持返回集合。

PL/Tcl目前对域类型还没有完整支持：它把域当作底层标量类型同样对待。
这意味着与该域关联的约束不会被强制执行。对于函数参数来说这不是问题，但如果把PL/Tcl函数声明为返回域类型，就会带来隐患。

40.3. PL/Tcl 中的数据值

提供给 PL/Tcl 函数代码的参数值，只是将输入参数转换成文本形式（就像用`SELECT`语句将它们显示出来一样）。反过来，`return`命令会接受任何字符串，
只要它是该函数声明返回类型的可接受输入格式。因此，在 PL/Tcl 函数内部，所有值都只是文本字符串。

40.4. PL/Tcl 中的全局数据

有时需要在一次函数调用结束后保留某些全局数据，供下一次调用继续使用，
或者在不同函数之间共享全局数据。在 PL/Tcl 中这很容易做到，但必须理解其中的一些限制。

出于安全原因，PL/Tcl 会针对每个 SQL 角色，在该角色各自独立的 Tcl 解释器中执行其调用的函数。这样可以防止一个用户无意或恶意地干扰另一用户的 PL/Tcl 函数行为。每个这样的解释器都为任何“global” Tcl 变量维护各自的值。因此，当且仅当两个 PL/Tcl 函数由同一个 SQL 角色执行时，它们才会共享同一组全局变量。在某些应用中，一个会话会在多个 SQL 角色下执行代码（例如通过`SECURITY DEFINER`函数、使用`SET ROLE`等），这时你可能需要显式采取一些步骤，确保 PL/Tcl 函数能够共享数据。为此，应确保需要互相通信的函数由同

一用户拥有，并将它们标记为SECURITY DEFINER。当然，你必须小心，确保这类函数不会被用来执行任何非预期操作。

一个会话中使用的所有 PL/TclU 函数都在同一个 Tcl 解释器中执行，而这个解释器当然不同于 PL/Tcl 函数所使用的解释器。因此，PL/TclU 函数之间会自动共享全局数据。这不被视为安全风险，因为所有 PL/TclU 函数都在相同的信任级别上执行，也就是数据库超级用户的级别。

为帮助防止 PL/Tcl 函数无意间彼此干扰，每个函数都可通过upvar命令访问一个全局数组。该变量的全局名称是函数的内部名称，局部名称是GD。建议将GD
用于保存函数的持久私有数据。只有当你明确希望某些值在多个函数之间共享时，
才应使用常规的 Tcl 全局变量。（注意GD数组只在某个特定解释器内部是全局的，因此它们不会绕过上文提到的安全限制。）

下面spi_execp的示例展示了GD的用法。

40.5. 从 PL/Tcl 访问数据库

下列命令可用于从 PL/Tcl 函数体中访问数据库：

spi_exec ?-count *n*? ?-array *name*? *command* ?*loop-body*?

执行以字符串形式给出的 SQL 命令。命令出错时会引发错误。否则，spi_exec的返回值是该命令处理的行数（选出、插入、更新或删除的行），如果命令是工具语句则返回零。此外，如果命令是SELECT语句，则所选列的值会按下文所述放入 Tcl 变量中。

可选的 -count 值告诉 spi_exec 此命令最多处理多少行，其效果类似于将查询设为游标后执行 FETCH *n*。

如果命令是SELECT语句，则结果列的值会放入以列名命名的 Tcl 变量中。如果给定了-array选项，则列值会存储在指定的关联数组元素中，列名用作数组索引。此外，
结果中的当前行号（从零开始计数）将存储在数组元素中，该数组元素的名称为“.tupno”，除非该名称在结果中已被用作列名。

如果命令是SELECT语句且未给出*loop-body* 脚本，则只会把结果的第一行存入 Tcl 变量或数组元素中；其余行如果存在，会被忽略。如果查询没有返回任何行，
则不会存储任何内容。（可以通过检查spi_exec的结果来检测这种情况。）例如：

```
spi_exec "SELECT count(*) AS cnt FROM pg_proc"
```

会把 Tcl 变量$cnt设置为pg_proc系统目录中的行数。

如果给出了可选的*loop-body*参数，它是一段 Tcl 脚本，会对查询结果中的每一行执行一次。（如果给定命令不是SELECT，则*loop-body*会被忽略。）在每次迭代前，
当前行各列的值都会被存入 Tcl 变量或数组元素中。例如：

```
spi_exec -array C "SELECT * FROM pg_class" {
    elog DEBUG "have table $C(relname)"
}
```

会为pg_class的每一行打印一条日志消息。这个特性与其他 Tcl 循环结构的工作方式类似；特别是continue和break在循环体内按通常方式工作。

如果查询结果中的某一列为空值，则对应的目标变量会被“unset”，而不是被设值。

spi_prepare *query typelist*

准备并保存一个查询计划以供后续执行。
保存的计划会在当前会话的整个生命周期内保留。

该查询可以使用参数，也就是在实际执行计划时提供值的占位符。在查询字符串中，通过符号$1 ... $*n*引用参数。如果查询使用了参数，则必须把参数类型名称作为 Tcl 列表给出。（如果不使用参数，则给*typelist*写一个空列表。）

spi_prepare的返回值是一个查询 ID，供后续调用spi_execp时使用。示例参见spi_execp。

spi_execp ?-count *n*? ?-array *name*? ?-nulls *string*? *queryid* ?*value-list*? ?*loop-body*?

执行先前用spi_prepare准备好的查询。*queryid*是spi_prepare返回的 ID。如果查询引用了参数，则必须提供*value-list*。这是参数实际值构成的 Tcl 列表。该列表的长度必须与先前提供给spi_prepare的参数类型列表相同。如果查询没有参数，则省略*value-list*。

可选的-nulls值是由空格和'n'字符组成的字符串，用来告诉spi_execp哪些参数是空值。如果给出，它的长度必须与*value-list*完全相同。如果未给出，则所有参数值都视为非空值。

除了指定查询及其参数的方式不同之外，spi_execp的工作方式与spi_exec完全一样。-count、-array和*loop-body*选项都相同，返回值也相同。

下面是使用已准备计划的 PL/Tcl 函数示例：

```
CREATE FUNCTION t1_count(integer, integer) RETURNS integer AS $$
    if {![ info exists GD(plan) ]} {
        # 在首次调用时准备并保存计划
        set GD(plan) [ spi_prepare \
                "SELECT count(*) AS cnt FROM t1 WHERE num >= \$1
 AND num <= \$2" \
                [ list int4 int4 ] ]
    }
    spi_execp -count 1 $GD(plan) [ list $1 $2 ]
    return $cnt
$$ LANGUAGE pltcl;
```

我们需要在传给spi_prepare的查询字符串中加入反斜杠，以确保$*n*标记会原样传递给spi_prepare，而不会被 Tcl 执行变量替换。

spi_lastoid

如果最后一次 spi_exec 或 spi_execp 执行的是单行 INSERT，且被修改的表包含 OID，则返回所插入行的 OID。（否则返回零。）

quote *string*

将给定字符串中的所有单引号和反斜杠字符都加倍。这可用于安全地为那些要插入到传给spi_exec或spi_prepare的 SQL 命令中的字符串加引号。例如，考虑如下 SQL 命令字符串：

```
"SELECT '$val' AS ret"
```

其中 Tcl 变量val的实际内容是doesn't。这会得到最终命令字符串：

```
SELECT 'doesn't' AS ret
```

这会在spi_exec或spi_prepare期间导致解析错误。要使其正常工作，提交的命令应当包含：

```
SELECT 'doesn''t' AS ret
```

在 PL/Tcl 中，可以用下面这种方式构造它：

```
"SELECT '[ quote $val ]' AS ret"
```

spi_execp的一个优点是，你不必像这样给参数值加引号，因为这些参数永远不会作为SQL 命令字符串的一部分被解析。

elog *level msg*

发出日志或错误消息。可用级别包括 DEBUG, LOG, INFO, NOTICE, WARNING, ERROR和 FATAL。ERROR 会引发错误条件；如果外围 Tcl 代码没有捕获它，该错误就会传播到调用查询，导致当前事务或子事务中止。这实际上与 Tcl 的error命令相同。FATAL 会中止事务并导致当前会话关闭。（在 PL/Tcl 函数中使用这个错误级别可能并没有什么充分理由，但为了完整性仍提供它。）其他级别只会生成不同优先级的消息。
某个特定优先级的消息是报告给客户端、写入服务器日志，还是两者都做，由log_min_messages和client_min_messages配置变量控制。参见第 18 章了解更多信息。

40.6. PL/Tcl 中的触发器函数

可以用 PL/Tcl 编写触发器函数。PostgreSQL 要求作为触发器调用的函数必须声明为无参数、返回类型为 trigger 的函数。

触发器管理器的信息通过以下变量传入函数体：

$TG_name

CREATE TRIGGER语句中触发器的名称。

$TG_relid

导致触发器函数被调用的表的对象 ID。

$TG_table_name

导致触发器函数被调用的表的名称。

$TG_table_schema

导致触发器函数被调用的表所属的模式。

$TG_relatts

表列名构成的 Tcl 列表，其前面带有一个空列表元素。因此，使用Tcl的lsearch命令在该列表中查找列名时，返回的元素编号会从 1 开始表示第一列，这与PostgreSQL中列的惯常编号方式一致。（空列表元素也会出现在已删除列的位置上，
这样其右侧列的属性编号仍然正确。）

$TG_when

其值为字符串BEFORE、AFTER或INSTEAD OF，具体取决于触发器事件的类型。

$TG_level

其值为字符串ROW或STATEMENT，取决于触发器事件的类型。

$TG_op

其值为字符串INSERT、UPDATE、DELETE或TRUNCATE，取决于触发器事件的类型。

$NEW

对于INSERT或UPDATE动作，这是一个包含新表行值的关联数组；对于DELETE则为空。该数组以列名为索引。值为空值的列不会出现在数组中。对于语句级触发器，
不会设置这个变量。

$OLD

对于UPDATE或DELETE动作，这是一个包含旧表行值的关联数组；对于INSERT则为空。该数组以列名为索引。值为空值的列不会出现在数组中。对于语句级触发器，不会设置这个变量。

$args

一个 Tcl 列表，包含 CREATE TRIGGER 语句中给出的触发器函数参数。在函数体中，也可以通过 $1 ... $*n* 访问这些参数。

触发器函数可以返回字符串 OK 或 SKIP，也可以返回列名与值的配对列表。如果返回 OK，触发它的操作（INSERT/UPDATE/DELETE）会正常继续。SKIP 告诉触发器管理器静默地取消针对该行的操作。如果返回列表，则告诉 PL/Tcl 向触发器管理器返回修改后的行；修改后行的内容由列表中的列名和值指定，未在列表中提及的列设为空值。
返回修改后的行只对以下触发器有意义：行级 BEFORE INSERT 或 UPDATE 触发器，此时将插入修改后的行，替代 $NEW 中给出的行；以及行级 INSTEAD OF INSERT 或 UPDATE 触发器，此时返回的行用作 INSERT RETURNING 或 UPDATE RETURNING 子句的源数据。对于行级 BEFORE DELETE 或 INSTEAD OF DELETE 触发器，返回修改后的行与返回 OK 的效果相同，即继续执行操作。其他所有类型的触发器都会忽略返回值。

提示

可使用 Tcl 的array get命令，根据修改后元组的数组表示构造结果列表。

下面是一个简单的触发器函数示例，强制用表中的一个整数值记录该行的更新次数。
新插入的行将此值初始化为 0，以后每次更新操作都将其递增。

```
CREATE FUNCTION trigfunc_modcount() RETURNS trigger AS $$
    switch $TG_op {
        INSERT {
            set NEW($1) 0
        }
        UPDATE {
            set NEW($1) $OLD($1)
            incr NEW($1)
        }
        default {
            return OK
        }
    }
    return [array get NEW]
$$ LANGUAGE pltcl;

CREATE TABLE mytab (num integer, description text, modcnt integer);

CREATE TRIGGER trig_mytab_modcount BEFORE INSERT OR UPDATE ON mytab
    FOR EACH ROW EXECUTE PROCEDURE trigfunc_modcount('modcnt');
```

注意，触发器函数本身不知道列名，列名通过触发器参数提供。这样，
同一个触发器函数就可以在不同表上复用。

40.7. 模块与unknown命令

PL/Tcl 支持在使用时自动装载 Tcl 代码。它会识别一张特殊的表 pltcl_modules，该表被假定包含若干 Tcl 代码模块。如果这张表存在，模块unknown会从该表取出，并在数据库会话中

第一次执行 PL/Tcl 函数之前立即装载到 Tcl 解释器中。（如果一个会话中使用了多个 Tcl 解释器，则对每个解释器分别执行这一过程；参见第 40.4 节。）

虽然unknown模块实际上可以包含你需要的任何初始化脚本，但它通常会定义一个 Tcl 的unknown过程，每当 Tcl 无法识别被调用的过程名时，就会调用该过程。PL/Tcl的这一过程的标准版本会尝试在pltcl_modules中找到一个能够定义所需过程的模块。如果找到了，就把它装载到解释器中，然后允许继续执行最初尝试的过程调用。辅助表pltcl_modfuncs提供了哪个模块定义了哪些函数的索引，使查找相当快速。

PostgreSQL发行版中包含用于维护这些表的支持脚本：pltcl_loadmod、pltcl_listmod、pltcl_delmod，以及标准unknown模块的源代码，位于share/unknown.pltcl。要支持自动装载机制，最初必须把这个模块装载到每个数据库中。

表pltcl_modules和pltcl_modfuncs 必须对所有用户可读，但最好只让数据库管理员拥有并可以写入它们。作为一种安全预防措施，除非pltcl_modules由超级用户拥有，否则 PL/Tcl 会忽略它（因而不尝试装载unknown模块）。不过，如果你足够信任其他用户，可以把这张表的 UPDATE 权限授予他们。

40.8. Tcl 过程名

在PostgreSQL中，只要参数个数或参数类型不同，就可以复用同一个函数名。不过，Tcl 要求所有过程名都必须不同。PL/Tcl 处理这一问题的方式是：在内部 Tcl 过程名中包含系统表 pg_proc 中的函数对象 ID 作为其名称的一部分。因此，名称相同但参数类型不同的PostgreSQL函数，也会对应不同的 Tcl 过程。这通常不是 PL/Tcl 程序员需要关心的事情，但在调试时可能会看见。

第 41 章 PL/Perl - Perl 过程语言

PL/Perl 是一种可加载的过程语言，可以用 Perl 编程语言[1]编写 PostgreSQL 函数。

使用 PL/Perl 的主要好处，是可以在存储函数中使用 Perl 提供的多种“字符串处理”操作符和函数。与使用 PL/pgSQL 提供的字符串函数和控制结构相比，用 Perl 解析复杂字符串可能更容易。

要在特定数据库中安装 PL/Perl，可使用 `CREATE EXTENSION plperl`，或者从 shell 命令行使用 `createlang plperl dbname`。

> **提示**
>
> 如果把某种语言安装到`template1`中，之后创建的所有数据库都会自动安装该语言。

> **注意**
>
> 使用源码包的用户必须在安装过程中专门启用 PL/Perl 的构建（更多信息见第 15 章）。使用二进制包的用户可能会在单独的子包中找到 PL/Perl。

41.1. PL/Perl 函数和参数

要用 PL/Perl 语言创建一个函数，可使用标准的CREATE FUNCTION语法：

```
CREATE FUNCTION funcname (argument-types) RETURNS return-type AS $$
    # PL/Perl 函数体
$$ LANGUAGE plperl;
```

函数的主体就是普通的 Perl 代码。事实上，PL/Perl 的粘合代码会把它包裹在一个 Perl 子程序中。PL/Perl 函数在标量上下文中被调用，因此不能返回列表。如下文所述，可以通过返回引用来返回非标量值（数组、记录和集合）。

PL/Perl 也支持用DO语句调用的匿名代码块：

```
DO $$
    # PL/Perl 代码
$$ LANGUAGE plperl;
```

匿名代码块不接收参数，它可能返回的任何值都会被丢弃。除此之外，
它的行为与函数完全相同。

> **注意**
>
> 在 Perl 中使用命名嵌套子程序是有危险的，特别是当它们引用外层作用域中的词法变量时。因为 PL/Perl 函数被包装成一个子程序，任何放在其中的命名子程序都会成为嵌套子程序。一般来说，创建通过代码引用（coderef）
> 调用的匿名子程序会安全得多。更多信息可见 perldiag手册页中的`Variable "%s" will not stay shared`以及`Variable "%s" is not available`，或者在互联网上搜索“perl nested named subroutine”。

[1] http://www.perl.org

`CREATE FUNCTION` 命令的语法要求函数体写成一个字符串常量。通常对该字符串常量使用美元引用（见第 4.1.2.4 节）最方便。如果选择使用转义字符串语法 `E''`，则必须将函数体中用到的单引号（`'`）和反斜线（`\`）都写成双份（见第 4.1.2.1 节）。

参数和结果的处理与其他 Perl 子程序一样：参数通过 `@_` 传入，结果值通过 `return` 返回，或者作为函数中最后一个被求值的表达式返回。

例如，一个返回两个整数值中较大值的函数可以定义为：

```
CREATE FUNCTION perl_max (integer, integer) RETURNS integer AS $$
    if ($_[0] > $_[1]) { return $_[0]; }
    return $_[1];
$$ LANGUAGE plperl;
```

> **注意**
>
> 参数会从数据库编码转换为 PL/Perl 中使用的 UTF-8，返回时再从 UTF-8 转回数据库编码。

如果一个 SQL 空值被传给一个函数，在 Perl 中该参数值将呈现为“未定义”。上述函数定义对于空值输入的行为并不理想（实际上，它会把它们当作零）。我们可以在函数定义中添加 `STRICT`，让 PostgreSQL 采取更合理的做法：如果传入空值，函数将根本不会被调用，而是自动返回空值结果。另一种方式是在函数体中检查未定义输入。例如，假设我们希望在 `perl_max` 的两个参数中一个为空值、另一个非空值时，返回非空值参数而不是空值：

```
CREATE FUNCTION perl_max (integer, integer) RETURNS integer AS $$
    my ($x, $y) = @_;
    if (not defined $x) {
        return undef if not defined $y;
        return $y;
    }
    return $x if not defined $y;
    return $x if $x > $y;
    return $y;
$$ LANGUAGE plperl;
```

如上所示，要从 PL/Perl 函数返回 SQL 空值，就返回一个未定义值。无论函数是否为严格函数，都可以这样做。

函数参数中任何不是引用的值都是字符串，其格式是相应数据类型的标准 PostgreSQL 外部文本表示。对于普通的数值或文本类型，Perl 通常会正确处理，程序员一般不必操心。不过在其他情况下，参数可能需要转换成 Perl 中更易用的形式。例如，可以用 `decode_bytea` 函数把 `bytea` 类型的参数转换成未转义的二进制数据。

同样，返回给 PostgreSQL 的值也必须采用外部文本表示格式。例如，可以用 `encode_bytea` 函数把二进制数据转义成适合作为 `bytea` 类型返回值的形式。

Perl 可以把 PostgreSQL 数组返回为对 Perl 数组的引用。这里有一个示例：

```
CREATE OR REPLACE function returns_array()
RETURNS text[][] AS $$
    return [['a"b','c,d'],['e\\f','g']];
$$ LANGUAGE plperl;

select returns_array();
```

Perl 把 PostgreSQL 数组作为一个 blessed 的 PostgreSQL::InServer::ARRAY 对象传递。这个对象可以被当作数组引用或字符串对待，允许为 PostgreSQL 9.1 之前版本编写的 Perl 代码继续运行。例如：

```
CREATE OR REPLACE FUNCTION concat_array_elements(text[]) RETURNS
 TEXT AS $$
    my $arg = shift;
    my $result = "";
    return undef if (!defined $arg);

    # as an array reference
    for (@$arg) {
        $result .= $_;
    }

    # also works as a string
    $result .= $arg;

    return $result;
$$ LANGUAGE plperl;

SELECT concat_array_elements(ARRAY['PL','/','Perl']);
```

> **注意**
>
> Multi-dimensional arrays are represented as references to lower-dimensional arrays of references in a way common to every Perl programmer.

复合类型参数被作为哈希的引用传递给函数。哈希的键是复合类型的属性名。
这里是一个示例：

```
CREATE TABLE employee (
    name text,
    basesalary integer,
    bonus integer
);

CREATE FUNCTION empcomp(employee) RETURNS integer AS $$
    my ($emp) = @_;
    return $emp->{basesalary} + $emp->{bonus};
$$ LANGUAGE plperl;

SELECT name, empcomp(employee.*) FROM employee;
```

PL/Perl 函数也可以用相同的方法返回复合类型：返回一个具有所需属性的哈希的引用。例如：

```
CREATE TYPE testrowperl AS (f1 integer, f2 text, f3 text);

CREATE OR REPLACE FUNCTION perl_row() RETURNS testrowperl AS $$
    return {f2 => 'hello', f1 => 1, f3 => 'world'};
$$ LANGUAGE plperl;

SELECT * FROM perl_row();
```

在声明的结果数据类型中，凡是哈希中未出现的列都将作为空值返回。

PL/Perl 函数还可以返回标量类型或复合类型的集合。通常会希望逐行返回，这既能缩短启动时间，也能避免把整个结果集排入内存。如下所示，可以用 `return_next` 实现这一点。注意，在最后一次 `return_next` 之后，必须写上 `return` 或者（更好）`return undef`。

```
CREATE OR REPLACE FUNCTION perl_set_int(int)
RETURNS SETOF INTEGER AS $$
    foreach (0..$_[0]) {
        return_next($_);
    }
    return undef;
$$ LANGUAGE plperl;

SELECT * FROM perl_set_int(5);

CREATE OR REPLACE FUNCTION perl_set()
RETURNS SETOF testrowperl AS $$
    return_next({ f1 => 1, f2 => 'Hello', f3 => 'World' });
    return_next({ f1 => 2, f2 => 'Hello', f3 => 'PostgreSQL' });
    return_next({ f1 => 3, f2 => 'Hello', f3 => 'PL/Perl' });
    return undef;
$$ LANGUAGE plperl;
```

对于较小的结果集，可以返回一个数组引用，其中对简单类型、
数组类型和复合类型分别包含标量、数组引用或哈希引用。
下面是把整个结果集作为数组引用返回的几个简单示例：

```
CREATE OR REPLACE FUNCTION perl_set_int(int) RETURNS SETOF INTEGER
 AS $$
    return [0..$_[0]];
$$ LANGUAGE plperl;

SELECT * FROM perl_set_int(5);

CREATE OR REPLACE FUNCTION perl_set() RETURNS SETOF testrowperl AS
 $$
    return [
        { f1 => 1, f2 => 'Hello', f3 => 'World' },
        { f1 => 2, f2 => 'Hello', f3 => 'PostgreSQL' },
        { f1 => 3, f2 => 'Hello', f3 => 'PL/Perl' }
    ];
$$ LANGUAGE plperl;

SELECT * FROM perl_set();
```

如果希望在代码中使用 `strict` 编译指示，可以有几种选择。对于临时的全局用法，可以 `SET plperl.use_strict` 为真。这会影响后续编译的 PL/Perl 函数，但不会影响当前会话中已经编译的函数。对于永久的全局用法，可以在 `postgresql.conf` 文件中将 `plperl.use_strict` 设为真。

如果要在特定函数中永久启用它，可以简单地把

```
use strict;
```

放在函数体顶部。

如果 Perl 版本为 5.10.0 或更高，也可以通过 `use` 使用 `feature` 编译指示。

41.2. PL/Perl 中的数据值

提供给 PL/Perl 函数代码的参数值，就是转换成文本形式的输入参数（就像它们由 `SELECT` 语句显示出来一样）。相反，`return` 和 `return_next` 命令接受任何可作为该函数声明返回类型输入格式的字符串。

41.3. 内置函数

41.3.1. 从 PL/Perl 访问数据库

可以通过下列函数在 Perl 函数中访问数据库本身：

spi_exec_query(*query* [, *max-rows*])

spi_exec_query 执行 SQL 命令，并返回整个行集；返回值是对一个数组的引用，数组元素为哈希引用。只有在确定结果集会比较小时，才应使用此命令。
下面是一个查询（SELECT 命令）的示例，指定了可选的最大行数：

```
$rv = spi_exec_query('SELECT * FROM my_table', 5);
```

这会返回最多 5 行，来自表 my_table。如果 my_table 包含列 my_column，就可以取得结果中第 $i 行的该列值，方法如下：

```
$foo = $rv->{rows}[$i]->{my_column};
```

要取得 SELECT 查询返回的总行数，可以这样做：

```
$nrows = $rv->{processed}
```

下面是使用另一种命令类型的示例：

```
$query = "INSERT INTO my_table VALUES (1, 'test')";
$rv = spi_exec_query($query);
```

可以这样访问命令状态（例如 SPI_OK_INSERT）：

```
$res = $rv->{status};
```

要获取受影响的行数，可使用：

```
$nrows = $rv->{processed};
```

这里是一个完整的示例：

```
CREATE TABLE test (
    i int,
    v varchar
);

INSERT INTO test (i, v) VALUES (1, 'first line');
INSERT INTO test (i, v) VALUES (2, 'second line');
INSERT INTO test (i, v) VALUES (3, 'third line');
INSERT INTO test (i, v) VALUES (4, 'immortal');

CREATE OR REPLACE FUNCTION test_munge() RETURNS SETOF test AS $$
    my $rv = spi_exec_query('select i, v from test;');
    my $status = $rv->{status};
    my $nrows = $rv->{processed};
    foreach my $rn (0 .. $nrows - 1) {
        my $row = $rv->{rows}[$rn];
        $row->{i} += 200 if defined($row->{i});
        $row->{v} =~ tr/A-Za-z/a-zA-Z/ if (defined($row->{v}));
        return_next($row);
    }
```

```
        return undef;
    $$ LANGUAGE plperl;

    SELECT * FROM test_munge();
```

spi_query(*command*)
spi_fetchrow(*cursor*)
spi_cursor_close(*cursor*)

spi_query 和 spi_fetchrow 需要配合使用，适用于结果集可能很大，或者希望在行到达时立即返回的情况。spi_fetchrow 只能与 spi_query 一起使用。下面的示例演示了它们的配合方式：

```
    CREATE TYPE foo_type AS (the_num INTEGER, the_text TEXT);

    CREATE OR REPLACE FUNCTION lotsa_md5 (INTEGER) RETURNS SETOF
     foo_type AS $$
        use Digest::MD5 qw(md5_hex);
        my $file = '/usr/share/dict/words';
        my $t = localtime;
        elog(NOTICE, "opening file $file at $t" );
        open my $fh, '<', $file # 注意，这里访问了文件!
            or elog(ERROR, "cannot open $file for reading: $!");
        my @words = <$fh>;
        close $fh;
        $t = localtime;
        elog(NOTICE, "closed file $file at $t");
        chomp(@words);
        my $row;
        my $sth = spi_query("SELECT * FROM generate_series(1,$_[0])
     AS b(a)");
        while (defined ($row = spi_fetchrow($sth))) {
            return_next({
                the_num => $row->{a},
                the_text => md5_hex($words[rand @words])
            });
        }
        return;
    $$ LANGUAGE plperlu;

    SELECT * from lotsa_md5(500);
```

通常，应重复调用 spi_fetchrow，直到它返回 undef，这表示已经没有更多行可读。当 spi_fetchrow 返回 undef 时，spi_query 返回的游标会被自动释放。如果不打算读取所有行，则应调用 spi_cursor_close 释放游标。否则会导致内存泄漏。

spi_prepare(*command*, *argument types*)
spi_query_prepared(*plan*, *arguments*)
spi_exec_prepared(*plan* [, *attributes*], *arguments*)
spi_freeplan(*plan*)

spi_prepare, spi_query_prepared, spi_exec_prepared, 以及 spi_freeplan 实现相同的功能，但用于预备查询。spi_prepare 接受包含编号参数占位符（$1、$2 等）的查询字符串，以及由参数类型字符串组成的列表：

```
    $plan = spi_prepare('SELECT * FROM test WHERE id > $1 AND name =
     $2',
                                                        'INTEGER',
     'TEXT');
```

调用 spi_prepare 预备查询计划后，就可以使用该计划代替查询字符串。可以将其用于 spi_exec_prepared，其结果与以下函数返回的结果相同：spi_exec_query；也可以用于 spi_query_prepared，它返回游标，行为与 spi_query 完全相同。随后可以将该游标传给 spi_fetchrow。spi_exec_prepared 可选的第二个参数是属性哈希引用；目前唯一支持的属性是 limit，用于设置查询返回的最大行数。

预备查询的优点在于，一个准备好的计划可以用于多次查询执行。当计划不再需要时，可以用 spi_freeplan 将其释放：

```
CREATE OR REPLACE FUNCTION init() RETURNS VOID AS $$
        $_SHARED{my_plan} = spi_prepare('SELECT (now() + $1)::
date AS now',
                                        'INTERVAL');
$$ LANGUAGE plperl;

CREATE OR REPLACE FUNCTION add_time( INTERVAL ) RETURNS TEXT AS
 $$
        return spi_exec_prepared(
                $_SHARED{my_plan},
                $_[0]
        )->{rows}->[0]->{now};
$$ LANGUAGE plperl;

CREATE OR REPLACE FUNCTION done() RETURNS VOID AS $$
        spi_freeplan( $_SHARED{my_plan});
        undef $_SHARED{my_plan};
$$ LANGUAGE plperl;

SELECT init();
SELECT add_time('1 day'), add_time('2 days'), add_time('3 days')
;
SELECT done();

  add_time  |  add_time  |  add_time
------------+------------+------------
 2005-12-10 | 2005-12-11 | 2005-12-12
```

请注意，spi_prepare 中的参数下标由 $1、$2、$3 等表示，因此应避免用双引号声明查询字符串，以免轻易引入难以察觉的错误。

下面的另一个示例展示了如何在 spi_exec_prepared 中使用可选参数：

```
CREATE TABLE hosts AS SELECT id, ('192.168.1.'||id)::inet AS
 address
                      FROM generate_series(1,3) AS id;

CREATE OR REPLACE FUNCTION init_hosts_query() RETURNS VOID AS $$
        $_SHARED{plan} = spi_prepare('SELECT * FROM hosts
                                      WHERE address << $1',
 'inet');
$$ LANGUAGE plperl;

CREATE OR REPLACE FUNCTION query_hosts(inet) RETURNS SETOF hosts
 AS $$
        return spi_exec_prepared(
                $_SHARED{plan},
                {limit => 2},
                $_[0]
        )->{rows};
```

```
$$ LANGUAGE plperl;

CREATE OR REPLACE FUNCTION release_hosts_query() RETURNS VOID AS
 $$
        spi_freeplan($_SHARED{plan});
        undef $_SHARED{plan};
$$ LANGUAGE plperl;

SELECT init_hosts_query();
SELECT query_hosts('192.168.1.0/30');
SELECT release_hosts_query();

    query_hosts
-----------------
 (1,192.168.1.1)
 (2,192.168.1.2)
(2 rows)
```

41.3.2. PL/Perl 中的辅助函数

`elog(level, msg)`

发出日志消息或错误信息。可用级别包括 `DEBUG`、`LOG`、`INFO`、`NOTICE`、`WARNING`以及`ERROR`。`ERROR` 会引发错误条件；如果周围的 Perl 代码没有捕获它，错误就会传播到调用查询，导致当前事务或子事务被中止。这实质上等同于 Perl 的 `die` 命令。其他级别只会生成不同优先级的消息。某一优先级的消息是报告给客户端、写入服务器日志，还是两者兼有，由配置变量log_min_messages和client_min_messages控制。详见第 18 章。

`quote_literal(string)`

返回给定字符串适当加引号后的形式，以便把它用作 SQL 语句字符串中的字符串字面量。嵌入的单引号和反斜线会被正确地双写。注意，对于 undef 输入，`quote_literal` 会返回 undef；如果参数可能为 undef，`quote_nullable` 往往更合适。

`quote_nullable(string)`

返回给定字符串适当加引号后的形式，以便把它用作 SQL 语句字符串中的字符串字面量；如果参数为 undef，则返回未加引号的字符串“NULL”。嵌入的单引号和反斜线会被正确地双写。

`quote_ident(string)`

返回给定字符串适当加引号后的形式，以便把它用作 SQL 语句字符串中的标识符。只有在必要时才会添加引号，也就是当字符串包含非标识符字符，或者会发生大小写折叠时。嵌入的引号会被正确地双写。

`decode_bytea(string)`

返回由给定字符串的内容表示的未转义二进制数据，该字符串应为 `bytea` 编码形式。

`encode_bytea(string)`

返回给定字符串中二进制数据内容的 `bytea` 编码形式。

`encode_array_literal(array)`
`encode_array_literal(array, delimiter)`

将引用数组的内容以数组字面量格式（见第 8.14.2 节）返回为字符串。若参数不是数组引用，则原样返回该参数值。如果未指定分隔符，或者分隔符为 undef，则数组字面量元素之间默认使用“, ”作为分隔符。

`encode_typed_literal(value, typename)`

将一个 Perl 变量转换为第二个参数指定的数据类型的值，并返回该值的字符串表示。它能正确处理嵌套数组和复合类型的值。

`encode_array_constructor(array)`

将引用数组的内容以数组构造器格式（第 4.2.12 节）返回为字符串。其中每个值都使用 `quote_nullable` 加引号。如果参数不是数组引用，则返回使用 `quote_nullable` 加引号后的参数值。

`looks_like_number(string)`

如果按照 Perl 的规则看，给定字符串的内容像数字，则返回真值，否则返回假值。如果参数为 undef，则返回 undef。前导和尾随空格会被忽略。`Inf` 和 `Infinity` 被视为数字。

`is_array_ref(argument)`

如果给定参数可被视为数组引用，则返回真值；也就是该参数的 ref 值为 `ARRAY` 或 `PostgreSQL::InServer::ARRAY`。否则返回假值。

41.4. PL/Perl 中的全局值

可以使用全局哈希 `%_SHARED` 在当前会话的整个生命周期内跨函数调用存储数据，包括代码引用。

这是共享数据的一个简单示例：

```
CREATE OR REPLACE FUNCTION set_var(name text, val text) RETURNS
 text AS $$
    if ($_SHARED{$_[0]} = $_[1]) {
        return 'ok';
    } else {
        return "cannot set shared variable $_[0] to $_[1]";
    }
$$ LANGUAGE plperl;

CREATE OR REPLACE FUNCTION get_var(name text) RETURNS text AS $$
    return $_SHARED{$_[0]};
$$ LANGUAGE plperl;

SELECT set_var('sample', 'Hello, PL/Perl!  How''s tricks?');
SELECT get_var('sample');
```

下面是一个稍复杂一些的、使用代码引用的示例：

```
CREATE OR REPLACE FUNCTION myfuncs() RETURNS void AS $$
    $_SHARED{myquote} = sub {
        my $arg = shift;
        $arg =~ s/(['\\])/\\$1/g;
        return "'$arg'";
    };
$$ LANGUAGE plperl;

SELECT myfuncs(); /* 初始化函数 */

/* 创建一个使用加引号函数的函数 */
```

```
CREATE OR REPLACE FUNCTION use_quote(TEXT) RETURNS text AS $$
    my $text_to_quote = shift;
    my $qfunc = $_SHARED{myquote};
    return &$qfunc($text_to_quote);
$$ LANGUAGE plperl;
```

（上面的代码也可以替换为单行 `return $_SHARED{myquote}->($_[0]);`，但代价是可读性会变差。）

出于安全原因，PL/Perl 会在每个 SQL 角色各自独立的 Perl 解释器中执行该角色调用的函数。这可以防止一个用户意外或恶意地干扰另一个用户的 PL/Perl 函数行为。每个这样的解释器都有自己的 `%_SHARED` 变量值和其他全局状态。因此，当且仅当两个 PL/Perl 函数都由同一 SQL 角色执行时，它们才会共享 `%_SHARED` 的值。在某些应用中，一个会话可能会在多个 SQL 角色下执行代码（通过 `SECURITY DEFINER` 函数、使用 `SET ROLE` 等），这时可能需要显式采取措施，确保 PL/Perl 函数能够通过 `%_SHARED` 共享数据。为此，要确保需要相互通信的函数由同一用户拥有，并将它们标记为 `SECURITY DEFINER`。当然，必须注意不要让这些函数被用于任何非预期用途。

41.5. 受信任与不受信任的 PL/Perl

通常，PL/Perl 会被安装为一种名为 `plperl` 的“受信任的”编程语言。在这种设置下，为了保持安全性，某些 Perl 操作会被禁用。一般来说，受限制的是那些与环境交互的操作，包括文件句柄操作、`require` 和 `use`（针对外部模块）。它无法像 C 函数那样访问数据库服务器进程的内部，也无法以服务器进程的权限获取操作系统级访问。因此，可以允许任何无特权的数据库用户使用这种语言。

下面示例中的函数将无法工作，因为出于安全原因不允许它做文件操作：

```
CREATE FUNCTION badfunc() RETURNS integer AS $$
    my $tmpfile = "/tmp/badfile";
    open my $fh, '>', $tmpfile
        or elog(ERROR, qq{could not open the file "$tmpfile": $!});
    print $fh "Testing writing to a file\n";
    close $fh or elog(ERROR, qq{could not close the file
 "$tmpfile": $!});
    return 1;
$$ LANGUAGE plperl;
```

这个函数的创建会失败，因为验证器会捕捉到它使用了禁用的操作。

有时需要编写不受这些限制的 Perl 函数。例如，可能需要一个能发送邮件的 Perl 函数。为处理这类情况，也可以把 PL/Perl 安装成一种“不受信任的”语言（通常称为 PL/PerlU）。在这种情况下，完整的 Perl 语言都可用。安装该语言时，使用语言名 `plperlu` 就会选择不受信任的 PL/Perl 变体。

PL/PerlU 函数的编写者必须注意，函数不能被用于任何非预期用途，因为它能够执行以数据库管理员身份登录的用户所能做的任何事情。请注意，
数据库系统只允许数据库超级用户用不受信任的语言创建函数。

如果上述函数是由超级用户使用 `plperlu` 语言创建的，执行就会成功。

同样地，如果把语言指定为 `plperlu` 而不是 `plperl`，用 Perl 编写的匿名代码块也可以使用原本受限的操作，但调用者必须是超级用户。

> **注意**
>
> 虽然 PL/Perl 函数会为每个 SQL 角色在独立的 Perl 解释器中运行，但给定会话中执行的所有 PL/PerlU 函数都运行在同一个 Perl 解释器中（而且这个解释器并不是任何一

个用于 PL/Perl 函数的解释器）。这允许 PL/PerlU 函数自由共享数据，但 PL/Perl 与 PL/PerlU 函数之间不能进行通信。

注意

Perl 无法在单个进程内支持多个解释器，除非构建时使用了适当的标志，即 `usemultiplicity` 或 `useithreads` 之一。（除非确实需要使用线程，否则更推荐 `usemultiplicity`。更多细节见 perlembed 手册页。）如果 PL/Perl 使用的是未按这种方式构建的 Perl 副本，那么每个会话中就只能有一个 Perl 解释器，因此任一会话只能执行 PL/PerlU 函数，或者执行全部由同一 SQL 角色调用的 PL/Perl 函数。

41.6. PL/Perl 触发器

PL/Perl 可用于编写触发器函数。在触发器函数中，哈希引用 `$_TD` 包含有关当前触发器事件的信息。`$_TD` 是一个全局变量，对触发器的每一次调用都会得到一个单独的局部值。`$_TD` 哈希引用包含以下字段：

`$_TD->{new}{foo}`

列 `foo` 的 `NEW` 值

`$_TD->{old}{foo}`

列 `foo` 的 `OLD` 值

`$_TD->{name}`

被调用的触发器名称

`$_TD->{event}`

触发器事件：`INSERT`、`UPDATE`、`DELETE`、`TRUNCATE`或者`UNKNOWN`

`$_TD->{when}`

触发器被调用的时机：`BEFORE`、`AFTER`、`INSTEAD OF` 或 `UNKNOWN`

`$_TD->{level}`

触发器级别：`ROW`、`STATEMENT`或者`UNKNOWN`

`$_TD->{relid}`

触发该触发器的表的 OID

`$_TD->{table_name}`

触发该触发器的表名

`$_TD->{relname}`

触发该触发器的表名。该字段已弃用，并且可能会在未来版本中移除。请改用 `$_TD->{table_name}`。

`$_TD->{table_schema}`

触发该触发器的表所在模式的名称

`$_TD->{argc}`

触发器函数的参数数目

`@{$_TD->{args}}`

触发器函数的参数。如果 `$_TD->{argc}` 为 0，则该字段不存在

行级触发器可以返回下列之一：

`return;`

执行操作

`"SKIP"`

不执行操作

`"MODIFY"`

指示触发器函数修改了 `NEW` 行

下面的触发器函数示例演示了上述部分内容：

```
CREATE TABLE test (
    i int,
    v varchar
);

CREATE OR REPLACE FUNCTION valid_id() RETURNS trigger AS $$
    if (($_TD->{new}{i} >= 100) || ($_TD->{new}{i} <= 0)) {
        return "SKIP";    # 跳过 INSERT/UPDATE 命令
    } elsif ($_TD->{new}{v} ne "immortal") {
        $_TD->{new}{v} .= "(modified by trigger)";
        return "MODIFY";  # 修改行并执行 INSERT/UPDATE 命令
    } else {
        return;           # 执行 INSERT/UPDATE 命令
    }
$$ LANGUAGE plperl;

CREATE TRIGGER test_valid_id_trig
    BEFORE INSERT OR UPDATE ON test
    FOR EACH ROW EXECUTE PROCEDURE valid_id();
```

41.7. PL/Perl 内部机制

41.7.1. 配置

本节列出影响 PL/Perl 的配置参数。

`plperl.on_init` (`string`)

指定在 Perl 解释器首次初始化时、在它被专用于 `plperl` 或 `plperlu` 之前，要执行的 Perl 代码。执行这段代码时 SPI 函数不可用。如果代码因错误而失败，就会中止解释器初始化，并把错误传播到调用查询，导致当前事务或子事务被中止。

Perl 代码仅限一个字符串。更长的代码可以放在模块中，并由 `on_init` 字符串加载。示例：

```
plperl.on_init = 'require "plperlinit.pl"'
```

```
plperl.on_init = 'use lib "/my/app"; use MyApp::PgInit;'
```

任何通过 `plperl.on_init` 直接或间接加载的模块，都可供 `plperl` 使用。这可能带来安全风险。要查看已经加载了哪些模块，可以使用：

```
DO 'elog(WARNING, join ", ", sort keys %INC)' LANGUAGE plperl;
```

如果 plperl 库被包含在shared_preload_libraries中，初始化就会在 postmaster 中发生；在这种情况下，需要额外考虑使 postmaster 不稳定的风险。使用此特性的主要原因是，通过 `plperl.on_init` 加载的 Perl 模块只需在 postmaster 启动时加载一次，之后在各个数据库会话中都可立即使用，而无需承担加载开销。不过要记住，这种开销只会对数据库会话使用的第一个 Perl 解释器被省掉，也就是 PL/PerlU，或者第一个调用 PL/Perl 函数的 SQL 角色所对应的 PL/Perl。数据库会话中后来创建的任何额外 Perl 解释器，都必须重新执行 `plperl.on_init`。此外，在 Windows 上预加载完全不会带来节省，因为在 postmaster 进程中创建的 Perl 解释器不会传播到子进程中。

这个参数只能在 `postgresql.conf` 文件中，或在服务器命令行上设置。

`plperl.on_plperl_init` (`string`)
`plperl.on_plperlu_init` (`string`)

这些参数分别指定在 Perl 解释器被专用于 `plperl` 或 `plperlu` 时要执行的 Perl 代码。这会在数据库会话中首次执行 PL/Perl 或 PL/PerlU 函数时发生；如果因为调用另一种语言，或某个新的 SQL 角色调用了 PL/Perl 函数而需要创建额外解释器时，也会发生。这是在 `plperl.on_init` 完成的任何初始化之后进行的。执行这段代码时 SPI 函数不可用。`plperl.on_plperl_init` 中的 Perl 代码是在对解释器进行“锁定”之后执行的，因此只能执行受信任的操作。

如果代码因错误而失败，就会中止初始化，并把错误传播到调用查询，导致当前事务或子事务被中止。在 Perl 中已经完成的任何动作都不会被撤销；不过，该解释器将不再被使用。如果再次使用该语言，就会在一个新的 Perl 解释器中再次尝试初始化。

只有超级用户可以更改这些设置。尽管这些设置可以在会话中修改，但这类更改不会影响已经用于执行函数的 Perl 解释器。

`plperl.use_strict` (`boolean`)

如果将其设为真，后续编译的 PL/Perl 函数都会启用 `strict` 编译指示。该参数不影响当前会话中已经编译的函数。

41.7.2. 限制与缺失特性

PL/Perl 目前仍缺少下列特性，但欢迎为此作出贡献。

- PL/Perl 函数不能直接调用彼此。
- SPI 尚未完全实现。
- 如果使用 `spi_exec_query` 取回非常大的数据集，应注意这些数据都会进入内存。可以像前面所示那样，通过使用 `spi_query`/`spi_fetchrow` 来避免这种情况。

  如果集合返回函数通过 `return` 把一个大型行集返回给 PostgreSQL，也会出现类似问题。前面已经说明过，可以改为对每一行使用 `return_next`，从而避免这个问题。
- 当会话正常结束（而不是由于致命错误结束）时，任何已经定义的 `END` 块都会被执行。目前不会执行其他操作。具体来说，文件句柄不会自动刷写，对象也不会自动销毁。

第 42 章 PL/Python - Python 过程语言

PL/Python 过程语言允许用 Python 语言[1]编写 PostgreSQL 函数。

要在特定数据库中安装 PL/Python，可使用`CREATE EXTENSION plpythonu`，或者从 shell 命令行使用`createlang plpythonu` *dbname*（另见第 42.1 节）。

> **提示**
>
> 如果把某种语言安装到`template1`中，之后创建的所有数据库都会自动安装该语言。

从 PostgreSQL 7.4 起，PL/Python 只能作为一种“不受信任的”语言使用，这意味着它不提供任何机制来限制用户能在其中做什么。因此它被重命名为`plpythonu`。
如果将来能开发出新的安全执行机制，受信任的变体`plpython`可能会重新出现。
使用不受信任的 PL/Python 编写函数时，函数编写者必须确保该函数不会被用来做任何不希望发生的事情，因为它能够做到任何以数据库管理员身份登录的用户所能做的事情。
只有超级用户才能在`plpythonu`这类不受信任的语言中创建函数。

> **注意**
>
> 源码包用户必须在安装过程中专门启用 PL/Python 的构建（更多信息请参阅安装说明）。二进制包用户则可能会在单独的子包中找到 PL/Python。

42.1. Python 2 与 Python 3

PL/Python 同时支持 Python 2 和 Python 3 语言变体。（PostgreSQL 安装说明中可能包含有关所支持 Python 精确次版本号的更多信息。）由于 Python 2 和 Python 3 语言变体在一些重要方面不兼容，PL/Python 使用以下命名和过渡方案以避免混用：

- PostgreSQL 中名为`plpython2u`的语言实现了基于 Python 2 语言变体的 PL/Python。
- PostgreSQL 中名为`plpython3u`的语言实现了基于 Python 3 语言变体的 PL/Python。
- 名为`plpythonu`的语言实现了基于默认 Python 语言变体的 PL/Python，当前默认为 Python 2。（此默认值独立于任何本地 Python 安装可能认为的其“默认”版本，例如`/usr/bin/python`可能是哪个版本。）默认值将来可能会在较远的 PostgreSQL 版本中改为 Python 3，具体取决于 Python 社区向 Python 3 迁移的进展。

该方案类似于PEP 394[2]中关于`python`命令命名和过渡的建议。

基于 Python 2 或 Python 3 的 PL/Python 是否可用（或者两者都可用），取决于构建配置或已安装的软件包。

> **提示**
>
> 所构建的变体取决于安装时找到的 Python 版本，或使用`PYTHON`环境变量显式设置的版本；请参见第 15.4 节。要在同一个安装中同时提供两种 PL/Python 变体，需要对源代码树进行两次配置和构建。

由此可以采用以下使用和迁移策略：

[1] http://www.python.org
[2] http://www.python.org/dev/peps/pep-0394/

- 现有用户和目前不打算使用 Python 3 的用户，可以使用语言名 `plpythonu`，在可预见的将来无需更改任何内容。建议通过迁移到 Python 2.6/2.7，逐步让代码“适应未来”，以简化最终迁移到 Python 3 的工作。

  实际上，许多 PL/Python 函数只需很少或不做改动即可迁移到 Python 3。

- 如果用户清楚自己的代码严重依赖 Python 2，且不打算更改，可以使用语言名 `plpython2u`。它在很久以后仍会继续工作，直到 PostgreSQL 可能完全停止支持 Python 2。

- 希望开始使用 Python 3 的用户可以使用语言名 `plpython3u`，按照目前的标准，它会一直可用。在遥远的未来，当 Python 3 可能成为默认版本时，用户也许会出于美观而去掉名称中的“3”。

- 希望构建仅包含 Python 3 的操作系统环境的大胆用户，可以修改 `pg_pltemplate` 的内容，使 `plpythonu` 等同于 `plpython3u`，但要记住，这会使该安装环境与绝大多数其他环境不兼容。

有关迁移到 Python 3 的更多信息，也可参见文档 Python 3.0 的新变化[3]。

不允许在同一个会话中同时使用基于 Python 2 和基于 Python 3 的 PL/Python，因为动态模块中的符号会冲突，可能导致 PostgreSQL 服务器进程崩溃。系统会检查同一会话是否混用了不同的 Python 大版本，如果发现不匹配，就会中止会话。不过，可以在同一数据库的不同会话中分别使用这两种 PL/Python。

42.2. PL/Python 函数

PL/Python 中的函数通过标准的CREATE FUNCTION语法声明：

```
CREATE FUNCTION funcname (argument-list)
  RETURNS return-type
AS $$
  # PL/Python 函数体
$$ LANGUAGE plpythonu;
```

函数体就是一段 Python 脚本。调用函数时，参数作为列表 `args` 的元素传入；命名参数还会作为普通变量传给 Python 脚本。使用命名参数通常更易读。Python 代码按通常方式用 `return` 或 `yield`（返回结果集时）返回结果。如果没有提供返回值，Python 默认返回 `None`。PL/Python 会将 Python 的 `None` 转换为 SQL 空值。

例如，一个返回两个整数中较大值的函数可以定义为：

```
CREATE FUNCTION pymax (a integer, b integer)
  RETURNS integer
AS $$
  if a > b:
    return a
  return b
$$ LANGUAGE plpythonu;
```

函数定义体中给出的 Python 代码会被转换成一个 Python 函数。例如，上面的定义会变成：

```
def __plpython_procedure_pymax_23456():
  if a > b:
    return a
  return b
```

这里假定 23456 是由PostgreSQL分配给该函数的 OID。

[3] http://docs.python.org/py3k/whatsnew/3.0.html

参数会被设置为全局变量。由于 Python 的作用域规则，这会带来一个微妙的结果：除非在代码块中再次将该变量声明为全局变量，
否则不能在函数内部把参数变量重新赋值为一个引用该变量名自身的表达式的值。例如，
下面这样是行不通的：

```
CREATE FUNCTION pystrip(x text)
  RETURNS text
AS $$
  x = x.strip()  # 错误
  return x
$$ LANGUAGE plpythonu;
```

因为对`x`赋值会使`x`在整个代码块中都成为局部变量，因此赋值语句右侧的`x`
指向的是尚未赋值的局部变量`x`，而不是 PL/Python 函数参数。使用`global`语句后，这样写就能工作：

```
CREATE FUNCTION pystrip(x text)
  RETURNS text
AS $$
  global x
  x = x.strip()  # 现在可以正常工作
  return x
$$ LANGUAGE plpythonu;
```

但是，建议不要依赖 PL/Python 的这一实现细节。最好将函数参数视为只读。

42.3. 数据值

一般来说，PL/Python 的目标是在 PostgreSQL 世界和 Python 世界之间提供一种 “自然” 的映射。下面描述的数据映射规则就是基于这一目标。

42.3.1. 数据类型映射

函数参数会从其 PostgreSQL 类型转换为相应的 Python 类型：

- PostgreSQL `boolean` 会转换为 Python `bool`。
- PostgreSQL 的 `smallint` 和 `int` 转换为 Python 的 `int`。PostgreSQL 的 `bigint` 在 Python 2 中转换为 `long`，在 Python 3 中转换为 `int`。
- PostgreSQL 的 `real`、`double` 和 `numeric` 会转换为 Python 的 `float`。注意，对 `numeric` 来说这会丢失信息，可能导致不正确的结果。这一点可能会在将来的版本中修复。
- PostgreSQL 的 `bytea` 在 Python 2 中转换为 `str`，在 Python 3 中转换为 `bytes`。在 Python 2 中，应将该字符串视为不带任何字符编码的字节序列。
- 所有其他数据类型，包括 PostgreSQL 字符串类型，都会转换为 Python `str`。在 Python 2 中，该字符串采用 PostgreSQL 服务器编码；在 Python 3 中，它与所有字符串一样，都是 Unicode 字符串。
- 对于非标量数据类型，请参见下文。

函数返回值会按如下方式转换为声明的 PostgreSQL 返回数据类型：

- 当 PostgreSQL 返回类型为`boolean`时，返回值会按照Python规则进行真值判定。也就是说，0 和空字符串为假，但值得注意的是，`'f'` 为真。
- 如果 PostgreSQL 返回类型是 `bytea`，会先使用相应的 Python 内置函数，将返回值转换为字符串（Python 2）或 bytes（Python 3），再将结果转换为 `bytea`。

- 对于所有其他 PostgreSQL 返回类型，返回值会使用 Python 内置函数`str`转换为字符串，然后将结果传给 PostgreSQL 数据类型的输入函数。

  Python 2 中的字符串传给 PostgreSQL 时，必须采用 PostgreSQL 服务器编码。在当前服务器编码中无效的字符串会引发错误，但并非所有编码不匹配都能被检测到，因此处理不当仍可能产生乱码数据。Unicode 字符串会自动转换为正确的编码，因而使用它们可能更安全、更方便。在 Python 3 中，所有字符串都是 Unicode 字符串。

- 对于非标量数据类型，请参见下文。

请注意，即使声明的 PostgreSQL 返回类型与实际返回对象的 Python 数据类型在逻辑上并不匹配，也不会报错；无论如何该值都会被转换。

42.3.2. 空值、None

如果将 SQL 空值传递给函数，该参数值在 Python 中会表现为 `None`。例如，函数 `pymax` 的定义（见第 42.2 节）对空值输入会返回错误结果。可以在函数定义中添加 `STRICT`，让 PostgreSQL 采取更合理的行为：如果传入空值，就完全不调用函数，而是自动返回空值结果。也可以在函数体中检查空值输入：

```
CREATE FUNCTION pymax (a integer, b integer)
  RETURNS integer
AS $$
  if (a is None) or (b is None):
    return None
  if a > b:
    return a
  return b
$$ LANGUAGE plpythonu;
```

如上所示，要从 PL/Python 函数返回 SQL 空值，只需返回 `None`。无论函数是否严格，都可以这样做。

42.3.3. 数组、列表

SQL 数组值会作为 Python 列表传入 PL/Python。要从 PL/Python 函数返回 SQL 数组值，请返回一个 Python 序列，例如列表或元组：

```
CREATE FUNCTION return_arr()
  RETURNS int[]
AS $$
return (1, 2, 3, 4, 5)
$$ LANGUAGE plpythonu;

SELECT return_arr();
 return_arr
-------------
 {1,2,3,4,5}
(1 row)
```

请注意，在 Python 中，字符串也是序列，这可能会带来一些 Python 程序员熟悉但并不理想的效果：

```
CREATE FUNCTION return_str_arr()
  RETURNS varchar[]
AS $$
return "hello"
$$ LANGUAGE plpythonu;
```

```
SELECT return_str_arr();
 return_str_arr
----------------
 {h,e,l,l,o}
(1 row)
```

42.3.4. 复合类型

复合类型参数会以 Python 映射的形式传给函数。映射中的元素名就是复合类型的属性名。如果传入行中的某个属性为空值，那么它在映射中的值就是`None`。例如：

```
CREATE TABLE employee (
  name text,
  salary integer,
  age integer
);

CREATE FUNCTION overpaid (e employee)
  RETURNS boolean
AS $$
  if e["salary"] > 200000:
    return True
  if (e["age"] < 30) and (e["salary"] > 100000):
    return True
  return False
$$ LANGUAGE plpythonu;
```

有多种方法可以从 Python 函数返回行类型或复合类型。以下示例假定我们有：

```
CREATE TYPE named_value AS (
  name   text,
  value  integer
);
```

复合结果可以按以下形式返回：

序列类型（元组或列表，但不能是集合，因为集合不可通过索引访问）

返回的序列对象，其项目数必须与复合结果类型的字段数相同。索引 0 的项目赋给复合类型的第一个字段，索引 1 的项目赋给第二个字段，依此类推。例如：

```
CREATE FUNCTION make_pair (name text, value integer)
  RETURNS named_value
AS $$
  return [ name, value ]
  # 也可以使用元组： return ( name, value )
$$ LANGUAGE plpythonu;
```

要为某一列返回 SQL 空值，请将 `None` 放在对应位置。

映射（字典）

结果类型中每一列的值，都使用列名作为键从映射中取得。例如：

```
CREATE FUNCTION make_pair (name text, value integer)
  RETURNS named_value
AS $$
  return { "name": name, "value": value }
$$ LANGUAGE plpythonu;
```

字典中多余的键值对会被忽略，缺少键则会被视为错误。要为某一列返回 SQL 空值，请插入 None，并以对应列名为键。

对象（提供方法__getattr__的任何对象）

其工作方式与映射相同。例如：

```
CREATE FUNCTION make_pair (name text, value integer)
  RETURNS named_value
AS $$
  class named_value:
    def __init__ (self, n, v):
      self.name = n
      self.value = v
  return named_value(name, value)

  # 或者简写为
  class nv: pass
  nv.name = name
  nv.value = value
  return nv
$$ LANGUAGE plpythonu;
```

也支持带OUT参数的函数。例如：

```
CREATE FUNCTION multiout_simple(OUT i integer, OUT j integer) AS $$
return (1, 2)
$$ LANGUAGE plpythonu;

SELECT * FROM multiout_simple();
```

42.3.5. 集合返回函数

PL/Python函数也可以返回标量类型或复合类型的集合。实现方式有多种，
因为返回的对象在内部会被转换成一个迭代器。以下示例假定我们有如下复合类型：

```
CREATE TYPE greeting AS (
  how text,
  who text
);
```

集合结果可以通过以下对象返回：

序列类型（元组、列表、集合）

```
CREATE FUNCTION greet (how text)
  RETURNS SETOF greeting
AS $$
  # 返回包含列表的元组，以列表表示复合类型
  # 其他组合方式也都可用
  return ( [ how, "World" ], [ how, "PostgreSQL" ], [ how, "PL/
Python" ] )
$$ LANGUAGE plpythonu;
```

迭代器（任何提供 __iter__ 和 next 方法的对象）

```
CREATE FUNCTION greet (how text)
  RETURNS SETOF greeting
AS $$
  class producer:
```

```
      def __init__ (self, how, who):
        self.how = how
        self.who = who
        self.ndx = -1

      def __iter__ (self):
        return self

      def next (self):
        self.ndx += 1
        if self.ndx == len(self.who):
          raise StopIteration
        return ( self.how, self.who[self.ndx] )

    return producer(how, [ "World", "PostgreSQL", "PL/Python" ])
  $$ LANGUAGE plpythonu;
```

生成器（`yield`）

```
  CREATE FUNCTION greet (how text)
    RETURNS SETOF greeting
  AS $$
    for who in [ "World", "PostgreSQL", "PL/Python" ]:
      yield ( how, who )
  $$ LANGUAGE plpythonu;
```

> **警告**
>
> 由于 Python 的缺陷 #1483133[4]，某些调试版本的 Python 2.4（以`--with-pydebug`选项配置并编译）在使用迭代器返回集合结果时会导致PostgreSQL服务器崩溃。未打补丁的 Fedora 4 就包含此缺陷。在 Python 的正式版本或已打补丁的 Fedora 4 中不会发生这种情况。

也支持带`OUT`参数的集合返回函数（使用`RETURNS SETOF record`）。例如：

```
CREATE FUNCTION multiout_simple_setof(n integer, OUT integer, OUT
 integer) RETURNS SETOF record AS $$
return [(1, 2)] * n
$$ LANGUAGE plpythonu;

SELECT * FROM multiout_simple_setof(3);
```

42.4. 共享数据

全局字典`SD`可用于在函数调用之间保存数据。这一变量是私有的静态数据。全局字典`GD`则是可供一个会话中的所有 Python 函数使用的公共数据；使用时要小心。

每个函数在 Python 解释器中都有自己的执行环境，因此`myfunc`中的全局数据和函数参数对`myfunc2`不可见。例外是前面提到的`GD`字典中的数据。

42.5. 匿名代码块

PL/Python 也支持通过DO语句调用的匿名代码块：

[4] http://bugs.python.org/issue1483133

```
DO $$
    # PL/Python 代码
$$ LANGUAGE plpythonu;
```

匿名代码块不接受任何参数，而且无论返回什么值都会被丢弃。除此之外，它的行为与函数完全相同。

42.6. 触发器函数

当函数被用作触发器时，字典`TD`包含与触发器相关的值：

`TD["event"]`

以字符串形式包含事件：`INSERT`、`UPDATE`、`DELETE`或者`TRUNCATE`。

`TD["when"]`

包含`BEFORE`、`AFTER`或者`INSTEAD OF`之一。

`TD["level"]`

包含`ROW`或者`STATEMENT`。

`TD["new"]`
`TD["old"]`

对于行级触发器，这两个字段中的一个或两个会根据触发器事件包含相应的触发行。

`TD["name"]`

包含触发器名称。

`TD["table_name"]`

包含触发器所在表的名称。

`TD["table_schema"]`

包含触发器所在表的模式。

`TD["relid"]`

包含触发器所在表的 OID。

`TD["args"]`

如果`CREATE TRIGGER`命令包含参数，这些参数可在`TD["args"][0]`到`TD["args"][n-1]`中取得。

如果`TD["when"]`是`BEFORE`或`INSTEAD OF`，且`TD["level"]`是`ROW`，那么可以从 Python 函数返回`None`或`"OK"`来表示该行未被修改，返回`"SKIP"`来中止该事件；如果`TD["event"]`是`INSERT`或`UPDATE`，还可以返回`"MODIFY"`来表示你已经修改了新行。否则返回值会被忽略。

42.7. 数据库访问

PL/Python 语言模块会自动导入一个名为`plpy`的 Python 模块。该模块中的函数和常量在 Python 代码中可以通过`plpy.foo`这样的形式访问。

42.7.1. 数据库访问函数

plpy模块提供若干函数来执行数据库命令：

plpy.execute(*query* [, *max-rows*])

使用查询字符串和可选的行数限制参数调用plpy.execute会执行该查询，并把结果作为结果对象返回。

结果对象的行为类似于列表或字典对象。可以通过行号和列名来访问结果对象。例如：

```
rv = plpy.execute("SELECT * FROM my_table", 5)
```

会从my_table中返回最多 5 行。如果my_table有一列名为my_column，则可以这样访问它：

```
foo = rv[i]["my_column"]
```

返回的行数可以用内置的len函数获取。

结果对象还具有以下附加方法：

nrows()

返回该命令处理的行数。注意，这不一定与返回的行数相同。例如，UPDATE 命令会设置这个值，但不会返回任何行（除非使用RETURNING）。

status()

SPI_execute()的返回值。

colnames()
coltypes()
coltypmods()

分别返回列名列表、列类型 OID 列表以及列的类型相关修饰符列表。

如果在来自不产生结果集的命令的结果对象上调用这些方法，就会引发异常，例如不带RETURNING的UPDATE或DROP TABLE。但在包含零行的结果集上使用这些方法是没有问题的。

结果对象是可修改的。

注意，调用plpy.execute会把整个结果集读入内存。只有在确信结果集相对较小时才应使用这个函数。获取大型结果时，如果不想承担过高的内存占用风险，应使用plpy.cursor而不是plpy.execute。

plpy.prepare(*query* [, *argtypes*])
plpy.execute(*plan* [, *arguments* [, *max-rows*]])

plpy.prepare为查询准备执行计划。若查询中有参数引用，则它接受查询字符串和参数类型列表作为参数。例如：

```
plan = plpy.prepare("SELECT last_name FROM my_users WHERE first_
name = $1", ["text"])
```

text是你要传给$1的变量类型。如果查询不需要任何参数，第二个参数可以省略。

准备好语句后，可以使用plpy.execute函数的另一种调用形式来执行它：

```
rv = plpy.execute(plan, ["name"], 5)
```

将计划对象作为第一个参数传递（而不是查询字符串），
并将要代入查询的值列表作为第二个参数传递。如果查询不需要任何参数，
第二个参数可以省略。和前面一样，第三个参数仍然是可选的行数限制。

查询参数和结果行字段会按照第 42.3 节中所述，在 PostgreSQL 与 Python 数据类型之间进行转换。

当你使用 PL/Python 模块准备一个计划时，它会被自动保存。关于这意味着什么，请参阅 SPI 文档（第 43 章）。为了在函数调用之间有效利用这一点，需要使用持久存储字典 `SD`或`GD`之一（见第 42.4 节）。例如：

```
CREATE FUNCTION usesavedplan() RETURNS trigger AS $$
    if "plan" in SD:
        plan = SD["plan"]
    else:
        plan = plpy.prepare("SELECT 1")
        SD["plan"] = plan
    # 函数的其余部分
$$ LANGUAGE plpythonu;
```

`plpy.cursor(`*`query`*`)`
`plpy.cursor(`*`plan`* `[, `*`arguments`*`])`

`plpy.cursor`函数接受与`plpy.execute`相同的参数（只是没有行数限制），
并返回一个游标对象，使你可以分块处理大型结果集。与`plpy.execute`一样，
可以使用查询字符串，也可以使用带参数列表的计划对象。

游标对象提供 `fetch` 方法，接受一个整数参数并返回结果对象。每次调用 `fetch`，返回对象都包含下一批行，行数不会超过参数值。所有行都取完后，`fetch` 开始返回空的结果对象。游标对象还提供迭代器接口[5]，每次产生一行，直到所有行取完。
通过这种方式取得的数据不是结果对象，而是字典，每个字典对应一行结果。

下面示例展示了处理大表中数据的两种方式：

```
CREATE FUNCTION count_odd_iterator() RETURNS integer AS $$
odd = 0
for row in plpy.cursor("select num from largetable"):
    if row['num'] % 2:
         odd += 1
return odd
$$ LANGUAGE plpythonu;

CREATE FUNCTION count_odd_fetch(batch_size integer) RETURNS
 integer AS $$
odd = 0
cursor = plpy.cursor("select num from largetable")
while True:
    rows = cursor.fetch(batch_size)
    if not rows:
        break
    for row in rows:
        if row['num'] % 2:
            odd += 1
return odd
$$ LANGUAGE plpythonu;
```

[5] http://docs.python.org/library/stdtypes.html#iterator-types

```
CREATE FUNCTION count_odd_prepared() RETURNS integer AS $$
odd = 0
plan = plpy.prepare("select num from largetable where num % $1
 <> 0", ["integer"])
rows = list(plpy.cursor(plan, [2]))

return len(rows)
$$ LANGUAGE plpythonu;
```

游标会被自动释放。但如果想显式释放游标持有的全部资源，可使用`close`方法。一旦关闭，就不能再从该游标中取数。

> **提示**
>
> 不要将`plpy.cursor`创建的对象，与 Python 数据库 API 规范[6]定义的 DB-API 游标混淆。除了名称相同，它们没有共同之处。

42.7.2. 捕获错误

访问数据库的函数可能会遇到错误，这会导致它们中止并抛出异常。`plpy.execute`和`plpy.prepare`都可能抛出`plpy.SPIError`某个子类的实例，默认情况下这会终止函数。这个错误可以像其他 Python 异常一样，通过`try/except`结构来处理。例如：

```
CREATE FUNCTION try_adding_joe() RETURNS text AS $$
    try:
        plpy.execute("INSERT INTO users(username) VALUES ('joe')")
    except plpy.SPIError:
        return "something went wrong"
    else:
        return "Joe added"
$$ LANGUAGE plpythonu;
```

所抛出异常的实际类，对应引发错误的具体条件。参见表 A.1中列出的可能条件。模块`plpy.spiexceptions`为每种 PostgreSQL 条件定义了一个异常类，类名由条件名派生。例如，`division_by_zero`变为`DivisionByZero`，`unique_violation`变为`UniqueViolation`，`fdw_error`变为`FdwError`，依此类推。所有这些异常类都继承自`SPIError`。这样区分之后，更容易处理特定错误，例如：

```
CREATE FUNCTION insert_fraction(numerator int, denominator int)
 RETURNS text AS $$
from plpy import spiexceptions
try:
    plan = plpy.prepare("INSERT INTO fractions (frac) VALUES ($1 /
 $2)", ["int", "int"])
    plpy.execute(plan, [numerator, denominator])
except spiexceptions.DivisionByZero:
    return "denominator cannot equal zero"
except spiexceptions.UniqueViolation:
    return "already have that fraction"
except plpy.SPIError, e:
    return "other error, SQLSTATE %s" % e.sqlstate
else:
    return "fraction inserted"
```

[6] http://www.python.org/dev/peps/pep-0249/

```
$$ LANGUAGE plpythonu;
```

注意，因为 `plpy.spiexceptions` 模块中的所有异常都继承自 `SPIError`，所以处理它的 `except` 子句会捕获任何数据库访问错误。

作为处理不同错误情况的另一种方法，你可以捕获`SPIError`异常，并在`except`
块中通过查看异常对象的`sqlstate`属性来判断具体的错误条件。
该属性是一个包含“SQLSTATE”错误代码的字符串值。这种方法大体上提供了相同的功能。

42.8. 显式子事务

如第 42.7.2 节中所述，从数据库访问引发的错误中恢复，可能会造成一种不理想的情况：在某个操作失败之前，其他一些操作已经成功，而在从该错误恢复后，
数据却处于不一致状态。PL/Python 以显式子事务的形式为这个问题提供了解决方案。

42.8.1. 子事务上下文管理器

考虑以下实现两个账户之间转账的函数：

```
CREATE FUNCTION transfer_funds() RETURNS void AS $$
try:
    plpy.execute("UPDATE accounts SET balance = balance - 100 WHERE
 account_name = 'joe'")
    plpy.execute("UPDATE accounts SET balance = balance + 100 WHERE
 account_name = 'mary'")
except plpy.SPIError, e:
    result = "error transferring funds: %s" % e.args
else:
    result = "funds transferred correctly"
plan = plpy.prepare("INSERT INTO operations (result) VALUES ($1)",
 ["text"])
plpy.execute(plan, [result])
$$ LANGUAGE plpythonu;
```

如果第二条 `UPDATE` 语句引发异常，此函数会报告错误，但第一条 `UPDATE` 的结果仍会提交。换句话说，资金会从 Joe 的账户中扣除，却不会转入 Mary 的账户。

为避免此类问题，可以将 `plpy.execute` 调用放在显式子事务中。`plpy` 模块提供了用于管理显式子事务的辅助对象，可通过 `plpy.subtransaction()` 函数创建。此函数创建的对象实现了上下文管理器接口[7]。使用显式子事务后，可以将函数改写为：

```
CREATE FUNCTION transfer_funds2() RETURNS void AS $$
try:
    with plpy.subtransaction():
        plpy.execute("UPDATE accounts SET balance = balance - 100
 WHERE account_name = 'joe'")
        plpy.execute("UPDATE accounts SET balance = balance + 100
 WHERE account_name = 'mary'")
except plpy.SPIError, e:
    result = "error transferring funds: %s" % e.args
else:
    result = "funds transferred correctly"
plan = plpy.prepare("INSERT INTO operations (result) VALUES ($1)",
 ["text"])
plpy.execute(plan, [result])
$$ LANGUAGE plpythonu;
```

[7] http://docs.python.org/library/stdtypes.html#context-manager-types

注意，仍需要使用 `try/catch`。否则，异常会传播到 Python 调用栈顶层，使整个函数因 PostgreSQL 错误而中止，从而不会向 `operations` 表插入任何行。子事务上下文管理器不会捕获错误，只保证在其作用域内执行的所有数据库操作以原子方式提交或回滚。
任何异常退出都会使子事务块回滚，并不限于数据库访问错误。显式子事务块中抛出的普通 Python 异常，也会导致该子事务回滚。

42.8.2. 较早的 Python 版本

使用 `with` 关键字的上下文管理器语法，从 Python 2.6 起默认可用。如果 PL/Python 使用更早的 Python 版本，仍然可以使用显式子事务，只是没那么方便。可以通过便捷别名 `enter` 和 `exit`，调用子事务管理器的 `__enter__` 和 `__exit__` 函数。转账示例函数可以写成：

```
CREATE FUNCTION transfer_funds_old() RETURNS void AS $$
try:
    subxact = plpy.subtransaction()
    subxact.enter()
    try:
        plpy.execute("UPDATE accounts SET balance = balance - 100
 WHERE account_name = 'joe'")
        plpy.execute("UPDATE accounts SET balance = balance + 100
 WHERE account_name = 'mary'")
    except:
        import sys
        subxact.exit(*sys.exc_info())
        raise
    else:
        subxact.exit(None, None, None)
except plpy.SPIError, e:
    result = "error transferring funds: %s" % e.args
else:
    result = "funds transferred correctly"

plan = plpy.prepare("INSERT INTO operations (result) VALUES ($1)",
 ["text"])
plpy.execute(plan, [result])
$$ LANGUAGE plpythonu;
```

注意

虽然 Python 2.5 已实现上下文管理器，但在该版本中使用 `with` 语法，需要使用 future 语句[8]。不过，由于实现细节的限制，PL/Python 函数中不能使用 future 语句。

42.9. 辅助函数

`plpy`模块还提供以下函数：`plpy.debug(msg)`、`plpy.log(msg)`、`plpy.info(msg)`、`plpy.notice(msg)`、`plpy.warning(msg)`、`plpy.error(msg)` 和 `plpy.fatal(msg)`。`plpy.error`和`plpy.fatal`实际上会引发 Python 异常；如果这些异常未被捕获，就会传播到调用查询，导致当前事务或子事务中止。`raise plpy.Error(msg)`和`raise plpy.Fatal(msg)`分别等价于调用`plpy.error`和`plpy.fatal`。其他函数只会生成不同优先级的消息。某一优先级的消息是报告给客户端、
写入服务器日志还是两者兼有，由配置变量log_min_messages和client_min_messages控制。更多信息见第 18 章。

[8] http://docs.python.org/release/2.5/ref/future.html

另一组辅助函数是`plpy.quote_literal(string)`、`plpy.quote_nullable(string)`以及`plpy.quote_ident(string)`。它们等价于第 9.4 节中描述的内置加引号函数。在构造临时查询时，这些函数很有用。例 39.1中动态 SQL 的一个 PL/Python 等价写法如下：

```
plpy.execute("UPDATE tbl SET %s = %s WHERE key = %s" % (
    plpy.quote_ident(colname),
    plpy.quote_nullable(newvalue),
    plpy.quote_literal(keyvalue)))
```

42.10. 环境变量

Python 解释器接受的某些环境变量也可用于影响 PL/Python 的行为。这些变量需要在 PostgreSQL 主服务器进程的环境中设置，例如在启动脚本中设置。可用的环境变量取决于 Python 版本；详情请参阅 Python 文档。在撰写本文时，假定 Python 版本足以支持这些变量，下列环境变量会影响 PL/Python：

- `PYTHONHOME`
- `PYTHONPATH`
- `PYTHONY2K`
- `PYTHONOPTIMIZE`
- `PYTHONDEBUG`
- `PYTHONVERBOSE`
- `PYTHONCASEOK`
- `PYTHONDONTWRITEBYTECODE`
- `PYTHONIOENCODING`
- `PYTHONUSERBASE`
- `PYTHONHASHSEED`

（这似乎是 Python 的实现细节，超出了 PL/Python 的控制范围：列在`python`手册页中的某些环境变量只在命令行解释器中有效，而在嵌入式 Python 解释器中无效。）

第 43 章 服务器编程接口

服务器编程接口（SPI）使用户定义 C 函数的编写者能够在其函数中运行 SQL 命令。SPI 是一组接口函数，用于简化对解析器、规划器和执行器的访问。SPI 还负责一部分内存管理工作。

> **注意**
>
> 可用的过程语言提供了多种从过程中执行SQL命令的方法。这些设施大多基于SPI，因此本文档对这些语言的用户也会有帮助。

为避免误解，本文用“函数”指SPI接口函数，而用“过程”指使用SPI的用户定义 C 函数。

注意，如果通过 SPI 调用的某条命令失败，控制不会返回到你的过程。相反，执行该过程的事务或子事务会被回滚。（考虑到 SPI 函数大多记录了错误返回约定，这一点可能看起来有些意外。不过，这些约定只适用于在 SPI 函数自身内部检测到的错误。）如果在可能失败的 SPI 调用外围建立自己的子事务，则可以在出错后重新取得控制权。

SPI 函数在成功时返回非负结果（要么直接作为整数返回值，要么如后文所述存放在全局变量 `SPI_result` 中）。发生错误时，则返回负值或 `NULL`。

使用 SPI 的源代码文件必须包含头文件 `executor/spi.h`。

43.1. 接口函数

SPI_connect

SPI_connect — 将一个过程连接到 SPI 管理器

大纲

```
int SPI_connect(void)
```

描述

SPI_connect 会为某次过程调用打开到 SPI 管理器的连接。如果要通过 SPI 执行命令，就必须调用此函数。不过，有些 SPI 辅助函数可以在未连接的过程中调用。

如果你的过程已经处于连接状态，SPI_connect将返回错误代码→ SPI_ERROR_CONNECT。当某个直接调用了SPI_connect的过程又直接调用另一个调用SPI_connect的过程时，就会发生这种情况。虽然当通过 SPI 调用的 SQL 命令调用另一个使用 SPI 的函数时，允许对 SPI 管理器的递归调用，但直接嵌套调用SPI_connect和SPI_finish是被禁止的。（但请参见SPI_push和SPI_pop。）

返回值

SPI_OK_CONNECT

成功时

SPI_ERROR_CONNECT

发生错误时

SPI_finish

SPI_finish — 将一个过程与 SPI 管理器断开

大纲

```
int SPI_finish(void)
```

描述

`SPI_finish` 关闭到 SPI 管理器的现有连接。在完成当前这次过程调用所需的 SPI 操作后，必须调用此函数。不过，如果通过 `elog(ERROR)` 中止了事务，就不必担心是否显式调用它；这种情况下 SPI 会自行清理。

如果在没有有效连接的情况下调用`SPI_finish`，它将返回`SPI_ERROR_UNCONNECTED`。这并不会带来根本性问题；它只表示 SPI 管理器无事可做。

返回值

`SPI_OK_FINISH`

如果正确地断开连接

`SPI_ERROR_UNCONNECTED`

如果从未连接的过程中调用

SPI_push

SPI_push — 压入 SPI 栈以允许递归使用 SPI

大纲

```
void SPI_push(void)
```

描述

SPI_push应当在执行另一个自身可能希望使用 SPI 的过程之前调用。在SPI_push之后，SPI 不再处于“连接”状态，除非重新执行SPI_connect，否则 SPI 函数调用将被拒绝。这确保了你的过程的 SPI 状态与你所调用的另一个过程的状态之间的清晰分离。在另一个过程返回之后，调用SPI_pop恢复对你自己 SPI 状态的访问。

注意，SPI_execute及相关函数在把控制交还给 SQL 执行引擎之前，会自动完成相当于SPI_push的操作，因此在使用这些函数时无需为此操心。只有当你直接调用可能包含SPI_connect调用的任意代码时，才需要发出SPI_push和SPI_pop。

SPI_pop

SPI_pop — 弹出 SPI 栈以从递归 SPI 使用中返回

大纲

```
void SPI_pop(void)
```

描述

`SPI_pop`从 SPI 调用栈中弹出先前的环境。参见`SPI_push`。

SPI_execute

SPI_execute — 执行一个命令

大纲

```
int SPI_execute(const char * command, bool read_only, long count)
```

描述

SPI_execute 执行指定的 SQL 命令，并最多检索 *count* 行。如果 *read_only* 为 true，该命令必须是只读的，且执行开销会略有降低。

此函数只能从已连接的过程中调用。

如果 *count* 为零，则该命令会针对其适用的所有行执行。如果 *count* 大于零，则最多检索 *count* 行；达到该计数时就会停止执行，这很像给查询增加了一个 LIMIT 子句。例如：

```
SPI_execute("SELECT * FROM foo", true, 5);
```

最多会从该表中检索 5 行。注意，这种限制只有在命令实际返回行时才有效。例如：

```
SPI_execute("INSERT INTO foo SELECT * FROM bar", false, 5);
```

会忽略 *count* 参数，把 bar 中的所有行都插入进去。不过：

```
SPI_execute("INSERT INTO foo SELECT * FROM bar RETURNING *", false,
 5);
```

最多只会插入 5 行，因为取到第 5 行 RETURNING 结果后就会停止执行。

你可以在一个字符串中传递多条命令；SPI_execute 返回最后执行的那条命令的结果。*count* 限制会分别作用于每条命令（尽管实际返回的只有最后一条命令的结果）。该限制不适用于规则生成的任何隐藏命令。

当 *read_only* 为 false 时，SPI_execute 会递增命令计数器，并在执行字符串中的每条命令前计算新的快照。如果当前事务隔离级别是 SERIALIZABLE 或 REPEATABLE READ，这个快照实际上不会变化；但在 READ COMMITTED 模式下，更新快照会让每条命令都能看到其他会话中新近提交事务的结果。这对于修改数据库的命令获得一致行为至关重要。

当 *read_only* 为 true 时，SPI_execute 不会更新快照和命令计数器，并且只允许命令字符串中出现普通的 SELECT 命令。这些命令会使用外围查询先前建立的快照来执行。由于消除了每条命令的额外开销，这种执行模式比读写模式略快。它还允许构造真正稳定的函数：由于连续执行都会使用同一个快照，结果也就不会发生变化。

在同一个使用 SPI 的函数中混合只读命令和读写命令通常并不明智，因为只读查询看不到读写查询所做的数据库更新，这可能导致非常令人困惑的行为。

（最后一条）命令实际执行所处理的行数，会通过全局变量 SPI_processed 返回。如果函数返回值是 SPI_OK_SELECT、SPI_OK_INSERT_RETURNING、SPI_OK_DELETE_RETURNING 或 SPI_OK_UPDATE_RETURNING，则可以通过全局指针 SPITupleTable *SPI_tuptable 访问结果行。有些工具命令（如 EXPLAIN）也会返回结果行集，此时 SPI_tuptable 同样会保存结果。

结构 SPITupleTable 定义如下：

```
typedef struct
{
    MemoryContext tuptabcxt;    /* 结果表的内存上下文 */
```

```
    uint32      alloced;        /* 已分配的 vals 数量 */
    uint32      free;           /* 空闲的 vals 数量 */
    TupleDesc   tupdesc;        /* 行描述符 */
    HeapTuple  *vals;           /* 行 */
} SPITupleTable;
```

vals 是一个指向行的指针数组。（有效项数为 SPI_processed。）tupdesc 是一个行描述符，可以传给处理行的 SPI 函数。tuptabcxt，alloced 和 free 是内部字段，不供 SPI 调用者使用。

SPI_finish 会释放当前过程调用期间分配的全部 SPITupleTable。如果某个结果表已经不再需要，也可以提前调用 SPI_freetuptable 释放它。

参数

const char * *command*

包含待执行命令的字符串

bool *read_only*

true 表示只读执行

long *count*

要返回的最大行数，或者用 0 表示不限制

返回值

如果命令执行成功，则返回下列（非负）值之一：

SPI_OK_SELECT

执行了 SELECT（但不是 SELECT INTO）

SPI_OK_SELINTO

执行了 SELECT INTO

SPI_OK_INSERT

执行了 INSERT

SPI_OK_DELETE

执行了 DELETE

SPI_OK_UPDATE

执行了 UPDATE

SPI_OK_INSERT_RETURNING

执行了 INSERT RETURNING

SPI_OK_DELETE_RETURNING

执行了 DELETE RETURNING

SPI_OK_UPDATE_RETURNING

执行了 UPDATE RETURNING

`SPI_OK_UTILITY`

执行了工具命令（例如 `CREATE TABLE`）

`SPI_OK_REWRITTEN`

命令被规则重写成了另一类命令（例如 `UPDATE` 变成了 `INSERT`）

出错时，返回以下负值之一：

`SPI_ERROR_ARGUMENT`

command 为 `NULL`，或者 *count* 小于 0

`SPI_ERROR_COPY`

尝试执行了 `COPY TO stdout` 或 `COPY FROM stdin`

`SPI_ERROR_TRANSACTION`

尝试执行了事务控制命令（`BEGIN`、`COMMIT`、`ROLLBACK`、`SAVEPOINT`、`PREPARE TRANSACTION`、`COMMIT PREPARED`、`ROLLBACK PREPARED` 及其各种变体）

`SPI_ERROR_OPUNKNOWN`

命令类型未知（理论上不应发生）

`SPI_ERROR_UNCONNECTED`

如果从未连接的过程中调用

注解

所有 SPI 查询执行函数都会设置 `SPI_processed` 和 `SPI_tuptable`（只设置指针，而不更改结构体内容）。如果需要在后续调用之后继续访问 `SPI_execute` 或其他查询执行函数的结果表，请把这两个全局变量保存到过程局部变量中。

SPI_exec

SPI_exec — 执行一个读/写命令

大纲

```
int SPI_exec(const char * command, long count)
```

描述

`SPI_exec`和 `SPI_execute`相同，但后者的 *`read_only`*参数的值总是取 `false`。

参数

`const char * command`

包含待执行命令的字符串

`long count`

要返回的最大行数，或者用 `0` 表示不限制

返回值

见`SPI_execute`。

SPI_execute_with_args

SPI_execute_with_args — 执行使用外部参数的命令

大纲

```
int SPI_execute_with_args(const char *command,
                          int nargs, Oid *argtypes,
                          Datum *values, const char *nulls,
                          bool read_only, long count)
```

描述

SPI_execute_with_args 执行一条可能包含外部提供参数引用的命令。命令文本中的参数引用写作 $*n*，而该调用会为每个这类符号指定数据类型和值。*read_only* 和 *count* 的含义与 SPI_execute 中相同。

相比 SPI_execute，这个例程的主要优点在于可将数据值插入命令而无需繁琐的引用和转义，因此能够显著降低 SQL 注入攻击的风险。

也可以通过先调用 SPI_prepare，再调用 SPI_execute_plan 达到类似效果。不过，使用本函数时，查询计划会针对所提供的具体参数值进行定制。对于一次性查询执行，应优先选择本函数。如果同一条命令要用许多不同参数重复执行，则两种方式孰快取决于重新规划的代价与定制计划收益之间的权衡。

参数

const char * *command*

命令字符串

int *nargs*

输入参数的数量（$1、$2 等）

Oid * *argtypes*

一个数组，包含各参数数据类型的 OID

Datum * *values*

一个实际参数值的数组

const char * *nulls*

一个描述哪些参数为空值的数组

如果 *nulls* 为 NULL，则 SPI_execute_with_args 假定没有参数为空值。

bool *read_only*

true 表示只读执行

long *count*

要返回的最大行数，或者用 0 表示不限制

返回值

返回值与 SPI_execute 相同。

成功时，SPI_processed 和 SPI_tuptable 的设置方式与 SPI_execute 相同。

SPI_prepare

SPI_prepare — 准备一个计划，但暂不执行

大纲

```
SPIPlanPtr SPI_prepare(const char * command, int nargs, Oid
 * argtypes)
```

描述

SPI_prepare 为指定命令创建并返回一个预备语句，但不执行该命令。该预备语句之后可以使用 SPI_execute_plan 反复执行。

当同一条或相似命令需要反复执行时，通常只做一次解析分析是有利的，
而重用该命令的执行计划也可能进一步带来收益。SPI_prepare 会把命令字符串转换成一个封装了解析分析结果的预备语句。如果发现为每次执行都生成定制计划并无益处，
该预备语句还可用来缓存执行计划。

预备命令可以通过在普通命令原本应写常量的位置写入参数（$1、$2 等）而得到泛化。这些参数的实际值会在调用 SPI_execute_plan 时指定。这样，预备语句就能适用于比无参形式更广泛的场景。

SPI_prepare 返回的计划只能在当前这次过程调用中使用，因为 SPI_finish 会释放为计划分配的内存。但可以使用函数 SPI_keepplan 或 SPI_saveplan 把预备语句保存得更久。

参数

const char * *command*

命令字符串

int *nargs*

输入参数的数量（$1、$2 等）

Oid * *argtypes*

一个数组指针，它指向的数组包含参数的数据类型的 OID

返回值

SPI_prepare 返回一个指向 SPIPlan（表示预备语句的不透明结构体）的非空指针。发生错误时会返回 NULL，并将 SPI_result 设为 SPI_execute 所使用的那些错误码之一；但如果 *command* 为 NULL，或者 *nargs* 小于 0，或者 *nargs* 大于 0 且 *argtypes* 为 NULL，则会将其设置为 SPI_ERROR_ARGUMENT。

注解

如果没有定义参数，则在第一次使用 SPI_execute_plan 时会创建一个通用计划，并在之后的所有执行中继续使用它。如果存在参数，SPI_execute_plan 在最初几次使用时会根据提供的参数值生成定制计划。当同一个预备语句被使用足够多次之后，SPI_execute_plan 会构建一个通用计划；如果它的代价没有比定制计划高出太多，就会开始改用通用计划，而不是每次都重新规划。如果这种默认行为不合适，可以把 CURSOR_OPT_GENERIC_PLAN 或 CURSOR_OPT_CUSTOM_PLAN 标志传给 SPI_prepare_cursor，分别强制使用通用计划或定制计划。

该函数只能从已连接的过程调用。

`SPIPlanPtr` 这个名字多少带有历史色彩，因为该数据结构已不再必然包含执行计划。

`SPIPlanPtr` 在 `spi.h` 中被声明为指向不透明结构体类型的指针。尝试直接访问其内容并不明智，因为这会让你的代码在 PostgreSQL 后续版本中更容易失效。

SPI_prepare_cursor

SPI_prepare_cursor — 准备一个计划，但暂不执行

大纲

```
SPIPlanPtr SPI_prepare_cursor(const char * command, int nargs,
                              Oid * argtypes, int cursorOptions)
```

描述

SPI_prepare_cursor 与 SPI_prepare 相同，但额外允许指定规划器的“游标选项”参数。该参数是一个位掩码，其可取值对应于 nodes/parsenodes.h 中 DeclareCursorStmt 的 options 字段。SPI_prepare 总是把游标选项设为零。

参数

const char * *command*

命令字符串

int *nargs*

输入参数的数量（$1、$2 等）

Oid * *argtypes*

一个数组指针，它指向的数组包含参数的数据类型的 OID

int *cursorOptions*

整数形式的游标选项位掩码，零会导致默认行为

返回值

SPI_prepare_cursor 的返回约定与 SPI_prepare 相同。

注解

在 *cursorOptions* 中可设置的有用位包括 CURSOR_OPT_SCROLL、CURSOR_OPT_NO_SCROLL、CURSOR_OPT_FAST_PLAN、CURSOR_OPT_GENERIC_PLAN 和 CURSOR_OPT_CUSTOM_PLAN。特别要注意，CURSOR_OPT_HOLD 会被忽略。

SPI_prepare_params

SPI_prepare_params — 准备一个计划，但暂不执行

大纲

```
SPIPlanPtr SPI_prepare_params(const char * command,
                              ParserSetupHook parserSetup,
                              void * parserSetupArg,
                              int cursorOptions)
```

描述

`SPI_prepare_params` 为指定命令创建并返回一个执行计划，但不执行该命令。该函数等价于 `SPI_prepare_cursor`，只是额外允许调用者指定解析器钩子函数，以控制外部参数引用的解析。

参数

`const char * command`

命令字符串

`ParserSetupHook parserSetup`

解析器钩子设置函数

`void * parserSetupArg`

传给 *parserSetup* 的透传参数

`int cursorOptions`

整数形式的游标选项位掩码，零会导致默认行为

返回值

`SPI_prepare_params` 的返回约定与 `SPI_prepare` 相同。

SPI_getargcount

SPI_getargcount — return the number of arguments needed by a plan prepared by `SPI_prepare`

大纲

```
int SPI_getargcount(SPIPlanPtr plan)
```

描述

`SPI_getargcount` 返回执行由 `SPI_prepare` 准备的计划所需的参数个数。

参数

`SPIPlanPtr plan`

执行计划（由 `SPI_prepare` 返回）

返回值

plan 所需的参数个数。如果 *plan* 为 `NULL` 或无效，则 `SPI_result` 会被设置为 `SPI_ERROR_ARGUMENT`，并返回 -1。

SPI_getargtypeid

SPI_getargtypeid — return the data type OID for an argument of a plan prepared by `SPI_prepare`

大纲

`Oid SPI_getargtypeid(SPIPlanPtr plan, int argIndex)`

描述

`SPI_getargtypeid` 返回由 `SPI_prepare` 准备的计划中第 *argIndex* 个参数的类型的OID。第一个参数的下标为零。

参数

`SPIPlanPtr plan`

执行计划（由 `SPI_prepare` 返回）

`int argIndex`

参数的索引，从零开始

返回值

给定索引处参数的类型 OID。如果 *plan* 为 `NULL` 或无效，或者 *argIndex* 小于 0，或者不小于为 *plan* 声明的参数个数，则 `SPI_result` 会被设置为 `SPI_ERROR_ARGUMENT`，并返回 `InvalidOid`。

SPI_is_cursor_plan

SPI_is_cursor_plan — return `true` if a plan prepared by `SPI_prepare` can be used with `SPI_cursor_open`

大纲

```
bool SPI_is_cursor_plan(SPIPlanPtr plan)
```

描述

如果由 `SPI_prepare` 准备的计划可以作为参数传给 `SPI_cursor_open`，则 `SPI_is_cursor_plan` 返回 `true`；否则返回 `false`。判定标准是 *plan* 必须表示单条命令，并且该命令会向调用者返回元组。例如，不包含 `INTO` 子句的 `SELECT` 是允许的，而 `UPDATE` 只包含 `RETURNING` 子句时才允许。

参数

`SPIPlanPtr` *plan*

执行计划（由 `SPI_prepare` 返回）

返回值

返回 `true` 或 `false`，表示 *plan* 能否产生游标，同时将 `SPI_result` 设为零。如果无法确定答案（例如 *plan* 为 `NULL` 或无效，或者在未连接到 SPI 时调用），则会将 `SPI_result` 设为合适的错误码，并返回 `false`。

SPI_execute_plan

SPI_execute_plan — execute a plan prepared by `SPI_prepare`

大纲

```
int SPI_execute_plan(SPIPlanPtr plan, Datum * values, const char
 * nulls,
                     bool read_only, long count)
```

描述

`SPI_execute_plan` 执行由 `SPI_prepare` 准备的计划。*`read_only`* 和 *`count`* 的含义与 `SPI_execute` 中相同。

参数

`SPIPlanPtr` *`plan`*

执行计划（由 `SPI_prepare` 返回）

`Datum *` *`values`*

一个实际参数值的数组。必须与该计划的参数个数等长。

`const char *` *`nulls`*

一个描述哪些参数为空值的数组。必须与该计划的参数个数等长。`n` 表示空值（*`values`* 中的对应项会被忽略）；空格表示非空值（*`values`* 中的对应项有效）。

如果 *`nulls`* 为 `NULL`，则 `SPI_execute_plan` 假定没有参数为空值。

`bool` *`read_only`*

`true` 表示只读执行

`long` *`count`*

要返回的最大行数，或者用 `0` 表示不限制

返回值

返回值和`SPI_execute`相同，还有下列额外可能的错误（负值）结果：

`SPI_ERROR_ARGUMENT`

如果*`plan`*为`NULL` 或者非法，或者*`count`*小于 0

`SPI_ERROR_PARAM`

如果*`values`*为`NULL`，但 *`plan`*在准备时带有参数

成功时，`SPI_processed` 和 `SPI_tuptable` 的设置方式与 `SPI_execute` 相同。

SPI_execute_plan_with_paramlist

SPI_execute_plan_with_paramlist — execute a plan prepared by `SPI_prepare`

大纲

```
int SPI_execute_plan_with_paramlist(SPIPlanPtr plan,
                                    ParamListInfo params,
                                    bool read_only,
                                    long count)
```

描述

`SPI_execute_plan_with_paramlist` 执行由 `SPI_prepare` 准备好的计划。该函数等价于 `SPI_execute_plan`，只是向查询传递参数值信息的形式不同。`ParamListInfo` 表示形式便于传递已经按这种格式存在的值，也支持通过 `ParamListInfo` 中指定的钩子函数使用动态参数集。

参数

`SPIPlanPtr plan`

执行计划（由 `SPI_prepare` 返回）

`ParamListInfo params`

包含参数类型和值的数据结构；没有参数时为 NULL

`bool read_only`

`true` 表示只读执行

`long count`

要返回的最大行数，或者用 `0` 表示不限制

返回值

返回值与 `SPI_execute_plan` 相同。

成功时，`SPI_processed` 和 `SPI_tuptable` 的设置方式与 `SPI_execute_plan` 相同。

SPI_execp

SPI_execp — 以读/写模式执行一个计划

大纲

```
int SPI_execp(SPIPlanPtr plan, Datum * values, const char * nulls,
 long count)
```

描述

SPI_execp 与 SPI_execute_plan 相同，只不过后者的 *read_only* 参数固定为 false。

参数

SPIPlanPtr *plan*

执行计划（由 SPI_prepare 返回）

Datum * *values*

一个实际参数值的数组。必须与该计划的参数个数等长。

const char * *nulls*

一个描述哪些参数为空值的数组。必须与该计划的参数个数等长。n 表示空值（*values* 中的对应项会被忽略）；空格表示非空值（*values* 中的对应项有效）。

如果 *nulls* 为 NULL，则 SPI_execp 假定没有参数为空值。

long *count*

要返回的最大行数，或者用 0 表示不限制

返回值

见 SPI_execute_plan。

成功时，SPI_processed 和 SPI_tuptable 的设置方式与 SPI_execute 相同。

SPI_cursor_open

SPI_cursor_open — set up a cursor using a plan created with `SPI_prepare`

大纲

```
Portal SPI_cursor_open(const char * name, SPIPlanPtr plan,
                       Datum * values, const char * nulls,
                       bool read_only)
```

描述

`SPI_cursor_open` 建立一个游标（在内部是一个 portal），该游标将执行由 `SPI_prepare` 准备好的一个计划。参数具有和 `SPI_execute_plan` 的相应参数相同的含义。

使用游标而不是直接执行该计划有两个好处。首先，可以一次只取出少量结果行，从而避免对返回大量行的查询耗尽内存。其次，一个 portal 可以比当前的过程存活更久（实际上，它可以一直存活到当前事务结束）。把 portal 的名称返回给该过程的调用者，就提供了一种把行集作为结果返回的方法。

传入的参数数据会被复制到该游标的 portal 中，因此即使该游标仍然存在，也可以释放这些参数数据。

参数

`const char * name`

portal 的名称，或为`NULL`以让系统选择名称

`SPIPlanPtr plan`

执行计划（由 `SPI_prepare` 返回）

`Datum * values`

一个实际参数值的数组。必须与该计划的参数个数等长。

`const char * nulls`

一个描述哪些参数为空值的数组。必须与该计划的参数个数等长。`n` 表示空值（*values* 中的对应项会被忽略）；空格表示非空值（*values* 中的对应项有效）。

如果 *nulls* 为 `NULL`，则 `SPI_cursor_open` 假定没有参数为空值。

`bool read_only`

`true` 表示只读执行

返回值

指向包含该游标的 portal 的指针。注意这里没有错误返回约定，任何错误都将通过`elog`报告。

SPI_cursor_open_with_args

SPI_cursor_open_with_args — 使用一个查询和参数建立一个游标

大纲

```
Portal SPI_cursor_open_with_args(const char *name,
                                 const char *command,
                                 int nargs, Oid *argtypes,
                                 Datum *values, const char *nulls,
                                 bool read_only, int cursorOptions)
```

描述

`SPI_cursor_open_with_args`建立一个将执行指定查询的游标（在内部是一个portal）。大部分参数具有和 `SPI_prepare_cursor` 和`SPI_cursor_open`中相应参数相同的含义。

对于一次性查询执行，应优先使用此函数，而不是先调用 `SPI_prepare_cursor` 再调用 `SPI_cursor_open`。如果同一条命令要用许多不同参数执行，哪种方法更快取决于重新规划的代价与定制计划收益之间的权衡。

传入的参数数据会被复制到该游标的 portal 中，因此即使该游标仍然存在，也可以释放这些参数数据。

参数

`const char * name`

portal 的名称，或为`NULL`以让系统选择名称

`const char * command`

命令字符串

`int nargs`

输入参数的数量（`$1`、`$2` 等）

`Oid * argtypes`

一个数组，包含各参数数据类型的 OID

`Datum * values`

一个实际参数值的数组

`const char * nulls`

一个描述哪些参数为空值的数组

如果 *nulls* 为 `NULL`，则 `SPI_cursor_open_with_args` 假定没有参数为空值。

`bool read_only`

`true` 表示只读执行

`int cursorOptions`

整数形式的游标选项位掩码，零会导致默认行为

返回值

指向包含该游标的 portal 的指针。注意这里没有错误返回约定，任何错误都将通过`elog`报告。

SPI_cursor_open_with_paramlist

SPI_cursor_open_with_paramlist — 使用参数建立一个游标

大纲

```
Portal SPI_cursor_open_with_paramlist(const char *name,
                                      SPIPlanPtr plan,
                                      ParamListInfo params,
                                      bool read_only)
```

描述

`SPI_cursor_open_with_paramlist` 建立一个游标（在内部是一个 portal），该游标将执行由 `SPI_prepare` 准备好的计划。该函数等价于 `SPI_cursor_open`，只是向查询传递参数值信息的形式不同。`ParamListInfo` 表示形式便于传递已经按这种格式存在的值，也支持通过 `ParamListInfo` 中指定的钩子函数使用动态参数集。

传入的参数数据会被复制到该游标的 portal 中，因此即使该游标仍然存在，也可以释放这些参数数据。

参数

`const char * name`

portal 的名称，或为`NULL`以让系统选择名称

`SPIPlanPtr plan`

执行计划（由 `SPI_prepare` 返回）

`ParamListInfo params`

包含参数类型和值的数据结构；没有参数时为 NULL

`bool read_only`

`true` 表示只读执行

返回值

指向包含该游标的 portal 的指针。注意这里没有错误返回约定，任何错误都将通过`elog`报告。

SPI_cursor_find

SPI_cursor_find — 用名称查找一个现有的游标

大纲

```
Portal SPI_cursor_find(const char * name)
```

描述

`SPI_cursor_find` 按名称查找现有的 portal。这主要用于解析由其他函数以文本形式返回的游标名。

参数

`const char * name`

该 portal 的名称

返回值

带有指定名称的 portal 的指针，如果没有找到就是 `NULL`

SPI_cursor_fetch

SPI_cursor_fetch — 从一个游标取出一些行

大纲

```
void SPI_cursor_fetch(Portal portal, bool forward, long count)
```

描述

SPI_cursor_fetch从一个游标取得一些行。这等效于 SQL 命令FETCH的一个子集（更多功能见SPI_scroll_cursor_fetch）。

参数

Portal *portal*

包含该游标的 portal

bool *forward*

为真表示向前获取，为假表示向后获取

long *count*

要取得的最大行数

返回值

成功时，SPI_processed 和 SPI_tuptable 的设置方式与 SPI_execute 相同。

注解

如果该游标的计划不是用CURSOR_OPT_SCROLL 选项创建的，向后获取可能失败。

SPI_cursor_move

SPI_cursor_move — 移动一个游标

大纲

```
void SPI_cursor_move(Portal portal, bool forward, long count)
```

描述

SPI_cursor_move跳过一个游标中的一些行。这等效于 SQL 命令MOVE的一个子集（更多的功能请见SPI_scroll_cursor_move）。

参数

Portal *portal*

包含该游标的 portal

bool *forward*

为真表示前移，为假表示后移

long *count*

要移动的最大行数

注解

如果该游标的计划不是用CURSOR_OPT_SCROLL 选项创建的，向后移动可能失败。

SPI_scroll_cursor_fetch

SPI_scroll_cursor_fetch — 从一个游标取出一些行

大纲

```
void SPI_scroll_cursor_fetch(Portal portal, Fetch
Direction direction,
                             long count)
```

描述

SPI_scroll_cursor_fetch从一个游标中取出一些行。这等效于 SQL 命令FETCH。

参数

Portal *portal*

包含该游标的 portal

FetchDirection *direction*

FETCH_FORWARD、FETCH_BACKWARD、FETCH_ABSOLUTE或者 FETCH_RELATIVE之一

long *count*

FETCH_FORWARD或 FETCH_BACKWARD方式中要取出的行数；FETCH_ABSOLUTE方式中要取出的绝对行号；FETCH_RELATIVE方式中要取出的相对行号

返回值

成功时，SPI_processed 和 SPI_tuptable 的设置方式与 SPI_execute 相同。

注解

参数*direction*和 *count*的详细解释请见 SQL FETCH命令。

如果该游标的计划不是用CURSOR_OPT_SCROLL 选项创建的，除FETCH_FORWARD之外的方向值可能失败。

SPI_scroll_cursor_move

SPI_scroll_cursor_move — 移动一个游标

大纲

```
void SPI_scroll_cursor_move(Portal portal, Fetch
Direction direction,
                            long count)
```

描述

SPI_scroll_cursor_move在一个游标中跳过一定数量的行。这等效于 SQL 命令MOVE。

参数

Portal *portal*

包含该游标的 portal

FetchDirection *direction*

FETCH_FORWARD、FETCH_BACKWARD、FETCH_ABSOLUTE或者 FETCH_RELATIVE之一

long *count*

FETCH_FORWARD或者 FETCH_BACKWARD方式中要移动的行数；FETCH_ABSOLUTE方式中要移动到的绝对行号；FETCH_RELATIVE方式中要移动到的相对行号

返回值

成功时，SPI_processed 的设置方式与 SPI_execute 相同。SPI_tuptable 会被设置为 NULL，因为本函数不会返回任何行。

注解

参数*direction*和 *count*的详细解释请见 SQL FETCH命令。

如果该游标的计划不是用CURSOR_OPT_SCROLL 选项创建的，除FETCH_FORWARD之外的方向值可能失败。

SPI_cursor_close

SPI_cursor_close — 关闭一个游标

大纲

`void SPI_cursor_close(Portal portal)`

描述

`SPI_cursor_close`关闭一个之前创建的游标并且释放它的 portal 存储。

所有打开的游标会在事务结束时自动被关闭。只有在希望尽快释放资源时，才需要调用 `SPI_cursor_close`。

参数

`Portal portal`

包含该游标的 portal

SPI_keepplan

SPI_keepplan — 保存一个预备语句

大纲

```
int SPI_keepplan(SPIPlanPtr plan)
```

描述

SPI_keepplan 保存传入的预备语句（由 SPI_prepare 准备），使其不会被 SPI_finish 或事务管理器释放。这让你能够在当前会话后续的过程调用中重用该预备语句。

参数

SPIPlanPtr *plan*

要保存的预备语句

返回值

成功返回 0；如果*plan*为NULL 或者无效则返回SPI_ERROR_ARGUMENT

注解

这个函数通过调整指针（无需复制数据）的方式，将传入的预备语句重定位到永久存储中。如果之后需要删除它，可以对其使用 SPI_freeplan。

SPI_saveplan

SPI_saveplan — 保存一个计划

大纲

`SPIPlanPtr SPI_saveplan(SPIPlanPtr plan)`

描述

`SPI_saveplan` 会把传入的计划（由 `SPI_prepare` 准备）保存到不会被 `SPI_finish` 或事务管理器释放的内存中，并返回指向所保存计划的指针。这让你能够在当前会话后续的过程调用中重用预备计划。

参数

`SPIPlanPtr plan`

要保存的计划

返回值

指向所保存计划的指针；如果不成功则为 `NULL`。出错时，`SPI_result` 会被设置为：

`SPI_ERROR_ARGUMENT`

如果 *plan* 为 `NULL` 或无效

`SPI_ERROR_UNCONNECTED`

如果从未连接的过程中调用

注解

原始传入的计划不会被释放，因此你可能需要对其调用 `SPI_freeplan`，以避免在 `SPI_finish` 之前发生内存泄露。

在大多数情况下，`SPI_keepplan` 比此函数更合适，因为它基本能达到相同效果，同时无需实际复制该预备语句的数据结构。

43.2. 接口支持函数

这里介绍的函数提供了一个接口，用于从 `SPI_execute` 及其他 SPI 函数返回的结果集中提取信息。

本节介绍的所有函数都可以在已连接和未连接的过程中使用。

SPI_fname

SPI_fname — 为指定的列号确定列名

大纲

```
char * SPI_fname(TupleDesc rowdesc, int colnumber)
```

描述

`SPI_fname` 返回指定列列名的一个副本。（不再需要时，可以用 `pfree` 释放它。）

参数

`TupleDesc` *`rowdesc`*

输入行描述

`int` *`colnumber`*

列号（从 1 开始计）

返回值

列名；如果 *`colnumber`* 超出范围则返回 `NULL`。出错时，`SPI_result` 会被设置为 `SPI_ERROR_NOATTRIBUTE`。

SPI_fnumber

SPI_fnumber — 根据指定列名确定列号

大纲

```
int SPI_fnumber(TupleDesc rowdesc, const char * colname)
```

描述

`SPI_fnumber` 返回指定列名对应的列号。

如果 *colname* 指向的是系统列（例如 `oid`），则会返回相应的负列号。调用者应当通过检查返回值是否恰好等于 `SPI_ERROR_NOATTRIBUTE` 来判断错误；除非本就要拒绝系统列，否则测试结果是否小于等于 0 并不正确。

参数

`TupleDesc` *rowdesc*

输入行描述

`const char *` *colname*

列名

返回值

列号（从 1 开始计），如果没有找到所提到的列名则返回 `SPI_ERROR_NOATTRIBUTE`。

SPI_getvalue

SPI_getvalue — 返回指定列的字符串值

大纲

```
char * SPI_getvalue(HeapTuple row, TupleDesc rowdesc,
 int colnumber)
```

描述

SPI_getvalue返回指定列的值的字符串表示。

结果保存在由 palloc 分配的内存中返回。（不再需要时，可以用 pfree 释放该内存。）

参数

HeapTuple *row*

要检查的输入行

TupleDesc *rowdesc*

输入行描述

int *colnumber*

列号（从 1 开始计）

返回值

列值；如果该列为空值、*colnumber* 超出范围（SPI_result 被设置为 SPI_ERROR_NOATTRIBUTE）或者没有输出函数可用（SPI_result 被设置为 SPI_ERROR_NOOUTFUNC）则返回 NULL。

SPI_getbinval

SPI_getbinval — 返回指定列的二进制值

大纲

```
Datum SPI_getbinval(HeapTuple row, TupleDesc rowdesc,
 int colnumber,
                    bool * isnull)
```

描述

`SPI_getbinval` 以内部形式（即 `Datum` 类型）返回指定列的值。

此函数不会为该 datum 分配新空间。对于传引用的数据类型，返回值将是指向传入行内部数据的指针。

参数

`HeapTuple` *`row`*

要检查的输入行

`TupleDesc` *`rowdesc`*

输入行描述

`int` *`colnumber`*

列号（从 1 开始计）

`bool *` *`isnull`*

列中是否为空值的标志

返回值

返回该列的二进制值。如果该列为空值，则 *`isnull`* 指向的变量会被设为真，否则设为假。

出错时，`SPI_result` 会被设置为 `SPI_ERROR_NOATTRIBUTE`。

SPI_gettype

SPI_gettype — 返回指定列的数据类型名称

大纲

```
char * SPI_gettype(TupleDesc rowdesc, int colnumber)
```

描述

`SPI_gettype`返回该指定列的数据类型名称的副本（当你不再需要该副本后，可以使用 `pfree` 释放它）。

参数

`TupleDesc` *`rowdesc`*

输入行描述

`int` *`colnumber`*

列号（从 1 开始计）

返回值

指定列的数据类型名称；出错时返回 `NULL`。出错时，`SPI_result` 会被设置为 `SPI_ERROR_NOATTRIBUTE`。

SPI_gettypeid

SPI_gettypeid — return the data type OID of the specified column

大纲

`Oid SPI_gettypeid(TupleDesc rowdesc, int colnumber)`

描述

`SPI_gettypeid`返回该指定列的数据类型的 OID。

参数

`TupleDesc rowdesc`

输入行描述

`int colnumber`

列号（从 1 开始计）

返回值

指定列数据类型的 OID；出错时返回 `InvalidOid`。出错时，`SPI_result` 会被设置为 `SPI_ERROR_NOATTRIBUTE`。

SPI_getrelname

SPI_getrelname — 返回指定关系的名称

大纲

```
char * SPI_getrelname(Relation rel)
```

描述

`SPI_getrelname`返回该指定关系的名称的副本（当你不再需要该副本后，可以使用`pfree`释放它）。

参数

`Relation rel`

输入关系

返回值

指定关系的名称。

SPI_getnspname

SPI_getnspname — 返回指定关系所属的名字空间

大纲

```
char * SPI_getnspname(Relation rel)
```

描述

`SPI_getnspname` 返回指定 `Relation` 所属名字空间名称的一个副本。这等价于该关系的模式名。用完该返回值后，应调用 `pfree` 释放它。

参数

`Relation rel`

输入关系

返回值

指定关系的名字空间的名称。

43.3. 内存管理

PostgreSQL 在内存上下文中分配内存。内存上下文为管理那些在许多不同位置创建、且生命周期各不相同的内存分配提供了便捷手段。销毁某个上下文时，
会释放其中分配的全部内存。因此，不必为了避免内存泄漏而跟踪每个单独对象；
只需管理数量相对较少的上下文即可。`palloc` 及相关函数都从“当前”上下文中分配内存。

`SPI_connect` 创建新的内存上下文并将其设为当前上下文。`SPI_finish` 则恢复先前的当前上下文，并销毁由 `SPI_connect` 创建的上下文。这些操作可以确保在过程内部所做的临时内存分配会在过程退出时被回收，从而避免内存泄漏。

不过，如果过程需要返回位于已分配内存中的对象（例如传引用数据类型的值），
就不能使用 `palloc` 来分配这块内存，至少在连接到 SPI 的期间不能这么做。否则，该对象会在 `SPI_finish` 时被释放，过程也就无法可靠工作。解决办法是用 `SPI_palloc` 为返回对象分配内存。`SPI_palloc` 在“上层执行器上下文”中分配内存，也就是调用 `SPI_connect` 时的当前内存上下文；这正是从过程返回值最合适的上下文。

如果在过程尚未连接到 SPI 时调用`SPI_palloc`，它的行为与普通的`palloc`相同。在过程连接到 SPI 管理器之前，当前内存上下文就是上层执行器上下文，因此过程通过`palloc`或 SPI 辅助函数所做的所有分配都位于这个上下文中。

调用 `SPI_connect` 时，会把该过程的私有上下文（由 `SPI_connect` 创建）设为当前上下文。所有通过 `palloc`、`repalloc` 或 SPI 辅助函数分配的内存（但`SPI_copytuple`、`SPI_returntuple`、`SPI_modifytuple` 和`SPI_palloc`除外）都位于这个上下文中。当过程通过 `SPI_finish` 与 SPI 管理器断开连接时，当前上下文会恢复为上层执行器上下文，而在该过程内存上下文中分配的所有内存都会被释放，之后就不能再使用。

本节介绍的所有函数既可由已连接的过程使用，也可由未连接的过程使用。
在未连接的过程中，它们的行为与底层普通服务器函数（`palloc`等）相同。

SPI_palloc

SPI_palloc — 在上层执行器上下文中分配内存

大纲

```
void * SPI_palloc(Size size)
```

描述

`SPI_palloc` 在上层执行器上下文中分配内存。

参数

`Size size`

要分配的存储空间大小（以字节计）

返回值

指向具有指定大小的新存储空间的指针

SPI_repalloc

SPI_repalloc — 在上层执行器上下文中重分配内存

大纲

```
void * SPI_repalloc(void * pointer, Size size)
```

描述

`SPI_repalloc`改变之前用`SPI_palloc` 分配的内存段的大小。

这个函数如今与普通的`repalloc`已无区别。保留它只是为了与现有代码保持向后兼容。

参数

`void * pointer`

指向要改变的现有存储空间的指针

`Size size`

要分配的存储空间大小（以字节计）

返回值

指向具有指定大小的新存储空间的指针，现有区域的内容会被复制到其中

SPI_pfree

SPI_pfree — 在上层执行器上下文中释放内存

大纲

```
void SPI_pfree(void * pointer)
```

描述

`SPI_pfree`释放之前使用 `SPI_palloc`或者 `SPI_repalloc`分配的内存。

这个函数如今与普通的`pfree`已无区别。保留它只是为了与现有代码保持向后兼容。

参数

`void * pointer`

指向要释放的现有存储空间的指针

SPI_copytuple

SPI_copytuple — 在上层执行器上下文中创建一行的副本

大纲

```
HeapTuple SPI_copytuple(HeapTuple row)
```

描述

`SPI_copytuple` 在上层执行器上下文中创建一行的副本。它通常用于从触发器中返回修改后的行。对于声明为返回复合类型的函数，应改用 `SPI_returntuple`。

参数

`HeapTuple` *`row`*

要复制的行

返回值

复制后的行；仅当*`tuple`*为 `NULL` 时才返回 `NULL`

SPI_returntuple

SPI_returntuple — 准备把一个元组返回为一个 Datum

大纲

`HeapTupleHeader SPI_returntuple(HeapTuple row, TupleDesc rowdesc)`

描述

`SPI_returntuple` 在上层执行器上下文中创建一行的副本，并以行类型 `Datum` 的形式返回。返回的指针只需在返回前用 `PointerGetDatum` 转换为 `Datum` 即可。

请注意，它应用于声明为返回复合类型的函数，而不用于触发器；
触发器中返回修改后的行应使用 `SPI_copytuple`。

参数

`HeapTuple row`

要复制的行

`TupleDesc rowdesc`

行描述符（若要获得最佳缓存效果，每次都传入同一个描述符）

返回值

指向复制后行的 `HeapTupleHeader`；仅当*row* 或 *rowdesc*为 `NULL` 时才返回 `NULL`

SPI_modifytuple

SPI_modifytuple — 通过替换给定行的选定字段来创建新行

大纲

```
HeapTuple SPI_modifytuple(Relation rel, HeapTuple row, int ncols,
                          int * colnum, Datum * values, const char
 * nulls)
```

描述

`SPI_modifytuple` 通过为选定列替换新值、并在其他位置复制原行的列来创建一行新记录。输入行本身不会被修改。新行会在上层执行器上下文中返回。

参数

`Relation rel`

仅用作该行的行描述符来源。（传递关系而不是行描述符是个设计缺陷。）

`HeapTuple row`

要被修改的行

`int ncols`

数组 *colnum* 中列号的数量

`int * colnum`

一个数组，包含要被修改的列的编号（列号从 1 开始）

`Datum * values`

指定列的新值

`const char * Nulls`

哪些新值为空值（如果有的话；格式参见 `SPI_execute_plan`）

返回值

修改后的新行，分配在上层执行器上下文中；仅当*row* 为 `NULL` 时才返回 `NULL`

出错时，`SPI_result`被设置如下：

`SPI_ERROR_ARGUMENT`

如果*rel*为`NULL`，或者 *row*为`NULL`，或者*ncols* 小于等于 0，或者*colnum*为`NULL`，或者*values*为`NULL`。

`SPI_ERROR_NOATTRIBUTE`

如果*colnum*包含一个无效的列号（小于等于 0 或者大于 *row*中的列数）。

SPI_freetuple

SPI_freetuple — 释放一个在上层执行器上下文中分配的行

大纲

```
void SPI_freetuple(HeapTuple row)
```

描述

`SPI_freetuple`释放之前在上层执行器上下文中分配的一行。

这个函数如今与普通的`heap_freetuple`已无区别。
保留它只是为了与现有代码保持向后兼容。

参数

`HeapTuple` *`row`*

要释放的行

SPI_freetuptable

SPI_freetuptable — free a row set created by `SPI_execute` or a similar function

大纲

`void SPI_freetuptable(SPITupleTable * tuptable)`

描述

`SPI_freetuptable` 释放由先前的 SPI 命令执行函数（例如 `SPI_execute`）创建的结果行集。因此，调用它时经常会直接把全局变量 `SPI_tupletable` 作为参数。

如果某个使用 SPI 的过程需要执行多条命令，并且不想将前面命令的结果一直保留到过程结束，那么这个函数就很有用。注意，任何尚未释放的结果行集都会在 `SPI_finish` 时被释放。

参数

`SPITupleTable * tuptable`

要释放的行集的指针

SPI_freeplan

SPI_freeplan — 释放一个之前保存的计划

大纲

`int SPI_freeplan(SPIPlanPtr plan)`

描述

`SPI_freeplan` 释放先前由 `SPI_prepare` 返回、或由 `SPI_keepplan` 或 `SPI_saveplan` 保存的预备语句。

参数

`SPIPlanPtr plan`

要释放的计划的指针

返回值

成功时返回 0；如果 *plan* 为 `NULL` 或无效，则返回 `SPI_ERROR_ARGUMENT`

43.4. 数据更改的可见性

下列规则决定了使用 SPI 的函数（以及其他任何 C 函数）中数据更改的可见性：

- 在某条 SQL 命令执行期间，该命令所做的任何数据更改对该命令本身都是不可见的。例如，在

  ```
  INSERT INTO a SELECT * FROM a;
  ```

  中，被插入的行对`SELECT`部分不可见。

- 某条命令 C 所做的更改，对所有在 C 之后启动的命令都可见，无论这些命令是在 C 内部（即 C 执行期间）启动，还是在 C 完成之后启动。

- 在由 SQL 命令调用的函数中（无论是普通函数还是触发器）通过 SPI 执行的命令，究竟遵循上述哪条规则，取决于传给 SPI 的读写标志。只读模式下执行的命令遵循第一条规则：它们看不到调用它们的命令所做的更改。
  读写模式下执行的命令遵循第二条规则：它们能够看到截至当前为止的所有更改。

- 所有标准过程语言都会根据函数的易变性属性设置 SPI 的读写模式。`STABLE` 和 `IMMUTABLE` 函数中的命令会以只读模式执行，而 `VOLATILE` 函数中的命令则以读写模式执行。虽然 C 函数作者能够违背这一约定，但通常并不是好主意。

下一节给出了一个展示这些规则如何应用的示例。

43.5. 示例

本节给出一个非常简单的 SPI 用法示例。过程 `execq` 以一条 SQL 命令作为第一个参数、以行数计数作为第二个参数，使用 `SPI_exec` 执行该命令，并返回该命令实际处理的行数。更复杂的 SPI 示例可以在源码树的 `src/test/regress/regress.c` 以及spi模块中找到。

```
#include "postgres.h"

#include "executor/spi.h"
#include "utils/builtins.h"
```

```
#ifdef PG_MODULE_MAGIC
PG_MODULE_MAGIC;
#endif

int execq(text *sql, int cnt);

int
execq(text *sql, int cnt)
{
    char *command;
    int ret;
    int proc;

    /* Convert given text object to a C string */
    command = text_to_cstring(sql);

    SPI_connect();

    ret = SPI_exec(command, cnt);

    proc = SPI_processed;
    /*
     * If some rows were fetched, print them via elog(INFO).
     */
    if (ret > 0 && SPI_tuptable != NULL)
    {
        TupleDesc tupdesc = SPI_tuptable->tupdesc;
        SPITupleTable *tuptable = SPI_tuptable;
        char buf[8192];
        int i, j;

        for (j = 0; j < proc; j++)
        {
            HeapTuple tuple = tuptable->vals[j];

            for (i = 1, buf[0] = 0; i <= tupdesc->natts; i++)
                snprintf(buf + strlen (buf), sizeof(buf) - strlen(
buf), " %s%s",
                         SPI_getvalue(tuple, tupdesc, i),
                         (i == tupdesc->natts) ? " " : " |");
            elog(INFO, "EXECQ: %s", buf);
        }
    }

    SPI_finish();
    pfree(command);

    return (proc);
}
```

（此函数使用版本 0 调用约定，以使示例更易于理解。在实际应用中，你应该使用新的版本 1 接口。）

将该函数编译成共享库之后（详见第 35.9.6 节），可以这样声明它：

```
CREATE FUNCTION execq(text, integer) RETURNS integer
    AS 'filename'
    LANGUAGE C;
```

以下是一个示例会话：

```
=> SELECT execq('CREATE TABLE a (x integer)', 0);
 execq
-------
     0
(1 row)

=> INSERT INTO a VALUES (execq('INSERT INTO a VALUES (0)', 0));
INSERT 0 1
=> SELECT execq('SELECT * FROM a', 0);
INFO:  EXECQ:  0    -- 由 execq 插入
INFO:  EXECQ:  1    -- 由 execq 返回并被外层 INSERT 插入

 execq
-------
     2
(1 row)

=> SELECT execq('INSERT INTO a SELECT x + 2 FROM a', 1);
 execq
-------
     1
(1 row)

=> SELECT execq('SELECT * FROM a', 10);
INFO:  EXECQ:  0
INFO:  EXECQ:  1
INFO:  EXECQ:  2    -- 0 + 2，按指定只插入一行

 execq
-------
     3              -- 10 只是最大值，3 才是真正的行数
(1 row)

=> DELETE FROM a;
DELETE 3
=> INSERT INTO a VALUES (execq('SELECT * FROM a', 0) + 1);
INSERT 0 1
=> SELECT * FROM a;
 x
---
 1                  -- a 中没有行 (0) + 1
(1 row)

=> INSERT INTO a VALUES (execq('SELECT * FROM a', 0) + 1);
INFO:  EXECQ:  1
INSERT 0 1
=> SELECT * FROM a;
 x
---
 1
 2                  -- a 中原有一行 + 1
(2 rows)

-- 下面演示数据更改可见性规则：

=> INSERT INTO a SELECT execq('SELECT * FROM a', 0) * x FROM a;
```

```
INFO:  EXECQ:  1
INFO:  EXECQ:  2
INFO:  EXECQ:  1
INFO:  EXECQ:  2
INFO:  EXECQ:  2
INSERT 0 2
=> SELECT * FROM a;
 x
---
 1
 2
 2                  -- 2 行 * 1 (第一行中的 x)
 6                  -- 3 行 (前面的 2 行加上刚插入的 1 行) * 2 (第二行中的
 x)
(4 rows)               ^^^^^^
                       不同调用中 execq() 可见的行
```

部分 VI. Reference

The entries in this Reference are meant to provide in reasonable length an authoritative, complete, and formal summary about their respective subjects. More information about the use of PostgreSQL, in narrative, tutorial, or example form, can be found in other parts of this book. See the cross-references listed on each reference page.

The reference entries are also available as traditional “man” pages.

目录

I. SQL Commands 1008
ABORT 1011
ALTER AGGREGATE 1012
ALTER COLLATION 1014
ALTER CONVERSION 1015
ALTER DATABASE 1016
ALTER DEFAULT PRIVILEGES 1018
ALTER DOMAIN 1021
ALTER EXTENSION 1024
ALTER FOREIGN DATA WRAPPER 1027
ALTER FOREIGN TABLE 1029
ALTER FUNCTION 1032
ALTER GROUP 1035
ALTER INDEX 1036
ALTER LANGUAGE 1038
ALTER LARGE OBJECT 1039
ALTER OPERATOR 1040
ALTER OPERATOR CLASS 1041
ALTER OPERATOR FAMILY 1042
ALTER ROLE 1046
ALTER SCHEMA 1049
ALTER SEQUENCE 1050
ALTER SERVER 1053
ALTER TABLE 1055
ALTER TABLESPACE 1066
ALTER TEXT SEARCH CONFIGURATION 1067
ALTER TEXT SEARCH DICTIONARY 1069
ALTER TEXT SEARCH PARSER 1071
ALTER TEXT SEARCH TEMPLATE 1072
ALTER TRIGGER 1073
ALTER TYPE 1074
ALTER USER 1077
ALTER USER MAPPING 1078
ALTER VIEW 1079
ANALYZE 1081
BEGIN 1083
CHECKPOINT 1085
CLOSE 1086
CLUSTER 1087
COMMENT 1089
COMMIT 1093
COMMIT PREPARED 1094
COPY 1095
CREATE AGGREGATE 1104
CREATE CAST 1107
CREATE COLLATION 1111
CREATE CONVERSION 1113
CREATE DATABASE 1115
CREATE DOMAIN 1118
CREATE EXTENSION 1120
CREATE FOREIGN DATA WRAPPER 1122
CREATE FOREIGN TABLE 1124
CREATE FUNCTION 1126
CREATE GROUP 1133
CREATE INDEX 1134

CREATE LANGUAGE 1140
CREATE OPERATOR 1143
CREATE OPERATOR CLASS 1146
CREATE OPERATOR FAMILY 1149
CREATE ROLE 1150
CREATE RULE 1154
CREATE SCHEMA 1157
CREATE SEQUENCE 1159
CREATE SERVER 1162
CREATE TABLE 1164
CREATE TABLE AS 1179
CREATE TABLESPACE 1182
CREATE TEXT SEARCH CONFIGURATION 1184
CREATE TEXT SEARCH DICTIONARY 1185
CREATE TEXT SEARCH PARSER 1187
CREATE TEXT SEARCH TEMPLATE 1189
CREATE TRIGGER 1190
CREATE TYPE 1195
CREATE USER 1203
CREATE USER MAPPING 1204
CREATE VIEW 1205
DEALLOCATE 1208
DECLARE 1209
DELETE 1212
DISCARD 1215
DO 1216
DROP AGGREGATE 1217
DROP CAST 1218
DROP COLLATION 1219
DROP CONVERSION 1220
DROP DATABASE 1221
DROP DOMAIN 1222
DROP EXTENSION 1223
DROP FOREIGN DATA WRAPPER 1224
DROP FOREIGN TABLE 1225
DROP FUNCTION 1226
DROP GROUP 1228
DROP INDEX 1229
DROP LANGUAGE 1230
DROP OPERATOR 1231
DROP OPERATOR CLASS 1233
DROP OPERATOR FAMILY 1235
DROP OWNED 1236
DROP ROLE 1237
DROP RULE 1238
DROP SCHEMA 1239
DROP SEQUENCE 1240
DROP SERVER 1241
DROP TABLE 1242
DROP TABLESPACE 1243
DROP TEXT SEARCH CONFIGURATION 1244
DROP TEXT SEARCH DICTIONARY 1245
DROP TEXT SEARCH PARSER 1246
DROP TEXT SEARCH TEMPLATE 1247
DROP TRIGGER 1248
DROP TYPE 1249
DROP USER 1250
DROP USER MAPPING 1251

DROP VIEW 1252
END 1253
EXECUTE 1254
EXPLAIN 1255
FETCH 1260
GRANT 1264
INSERT 1271
LISTEN 1274
LOAD 1275
LOCK 1276
MOVE 1278
NOTIFY 1280
PREPARE 1282
PREPARE TRANSACTION 1284
REASSIGN OWNED 1286
REINDEX 1287
RELEASE SAVEPOINT 1289
RESET 1290
REVOKE 1291
ROLLBACK 1295
ROLLBACK PREPARED 1296
ROLLBACK TO SAVEPOINT 1297
SAVEPOINT 1299
SECURITY LABEL 1301
SELECT 1304
SELECT INTO 1322
SET 1324
SET CONSTRAINTS 1327
SET ROLE 1328
SET SESSION AUTHORIZATION 1330
SET TRANSACTION 1332
SHOW 1334
START TRANSACTION 1336
TRUNCATE 1337
UNLISTEN 1339
UPDATE 1340
VACUUM 1344
VALUES 1346
II. PostgreSQL Client Applications 1349
clusterdb 1350
createdb 1353
createlang 1356
createuser 1359
dropdb 1363
droplang 1366
dropuser 1369
ecpg 1371
pg_basebackup 1373
pg_config 1378
pg_dump 1381
pg_dumpall 1392
pg_receivexlog 1397
pg_restore 1400
psql 1408
reindexdb 1441
vacuumdb 1444
III. PostgreSQL Server Applications 1447
initdb 1448

pg_controldata 1451
pg_ctl 1452
pg_resetxlog 1457
postgres 1459
postmaster 1466

SQL Commands

This part contains reference information for the SQL commands supported by PostgreSQL. By "SQL" the language in general is meant; information about the standards conformance and compatibility of each command can be found on the respective reference page.

目录

ABORT 1011
ALTER AGGREGATE 1012
ALTER COLLATION 1014
ALTER CONVERSION 1015
ALTER DATABASE 1016
ALTER DEFAULT PRIVILEGES 1018
ALTER DOMAIN 1021
ALTER EXTENSION 1024
ALTER FOREIGN DATA WRAPPER 1027
ALTER FOREIGN TABLE 1029
ALTER FUNCTION 1032
ALTER GROUP 1035
ALTER INDEX 1036
ALTER LANGUAGE 1038
ALTER LARGE OBJECT 1039
ALTER OPERATOR 1040
ALTER OPERATOR CLASS 1041
ALTER OPERATOR FAMILY 1042
ALTER ROLE 1046
ALTER SCHEMA 1049
ALTER SEQUENCE 1050
ALTER SERVER 1053
ALTER TABLE 1055
ALTER TABLESPACE 1066
ALTER TEXT SEARCH CONFIGURATION 1067
ALTER TEXT SEARCH DICTIONARY 1069
ALTER TEXT SEARCH PARSER 1071
ALTER TEXT SEARCH TEMPLATE 1072
ALTER TRIGGER 1073
ALTER TYPE 1074
ALTER USER 1077
ALTER USER MAPPING 1078
ALTER VIEW 1079
ANALYZE 1081
BEGIN 1083
CHECKPOINT 1085
CLOSE 1086
CLUSTER 1087
COMMENT 1089
COMMIT 1093
COMMIT PREPARED 1094
COPY 1095
CREATE AGGREGATE 1104
CREATE CAST 1107
CREATE COLLATION 1111
CREATE CONVERSION 1113
CREATE DATABASE 1115

CREATE DOMAIN 1118
CREATE EXTENSION 1120
CREATE FOREIGN DATA WRAPPER 1122
CREATE FOREIGN TABLE 1124
CREATE FUNCTION 1126
CREATE GROUP 1133
CREATE INDEX 1134
CREATE LANGUAGE 1140
CREATE OPERATOR 1143
CREATE OPERATOR CLASS 1146
CREATE OPERATOR FAMILY 1149
CREATE ROLE 1150
CREATE RULE 1154
CREATE SCHEMA 1157
CREATE SEQUENCE 1159
CREATE SERVER 1162
CREATE TABLE 1164
CREATE TABLE AS 1179
CREATE TABLESPACE 1182
CREATE TEXT SEARCH CONFIGURATION 1184
CREATE TEXT SEARCH DICTIONARY 1185
CREATE TEXT SEARCH PARSER 1187
CREATE TEXT SEARCH TEMPLATE 1189
CREATE TRIGGER 1190
CREATE TYPE 1195
CREATE USER 1203
CREATE USER MAPPING 1204
CREATE VIEW 1205
DEALLOCATE 1208
DECLARE 1209
DELETE 1212
DISCARD 1215
DO 1216
DROP AGGREGATE 1217
DROP CAST 1218
DROP COLLATION 1219
DROP CONVERSION 1220
DROP DATABASE 1221
DROP DOMAIN 1222
DROP EXTENSION 1223
DROP FOREIGN DATA WRAPPER 1224
DROP FOREIGN TABLE 1225
DROP FUNCTION 1226
DROP GROUP 1228
DROP INDEX 1229
DROP LANGUAGE 1230
DROP OPERATOR 1231
DROP OPERATOR CLASS 1233
DROP OPERATOR FAMILY 1235
DROP OWNED 1236
DROP ROLE 1237
DROP RULE 1238
DROP SCHEMA 1239
DROP SEQUENCE 1240
DROP SERVER 1241
DROP TABLE 1242
DROP TABLESPACE 1243
DROP TEXT SEARCH CONFIGURATION 1244

DROP TEXT SEARCH DICTIONARY 1245
DROP TEXT SEARCH PARSER 1246
DROP TEXT SEARCH TEMPLATE 1247
DROP TRIGGER 1248
DROP TYPE 1249
DROP USER 1250
DROP USER MAPPING 1251
DROP VIEW 1252
END 1253
EXECUTE 1254
EXPLAIN 1255
FETCH 1260
GRANT 1264
INSERT 1271
LISTEN 1274
LOAD 1275
LOCK 1276
MOVE 1278
NOTIFY 1280
PREPARE 1282
PREPARE TRANSACTION 1284
REASSIGN OWNED 1286
REINDEX 1287
RELEASE SAVEPOINT 1289
RESET 1290
REVOKE 1291
ROLLBACK 1295
ROLLBACK PREPARED 1296
ROLLBACK TO SAVEPOINT 1297
SAVEPOINT 1299
SECURITY LABEL 1301
SELECT 1304
SELECT INTO 1322
SET 1324
SET CONSTRAINTS 1327
SET ROLE 1328
SET SESSION AUTHORIZATION 1330
SET TRANSACTION 1332
SHOW 1334
START TRANSACTION 1336
TRUNCATE 1337
UNLISTEN 1339
UPDATE 1340
VACUUM 1344
VALUES 1346

ABORT

ABORT — 中止当前事务

大纲

```
ABORT [ WORK | TRANSACTION ]
```

描述

`ABORT`回滚当前事务，并丢弃该事务所做的全部更新。这个命令在行为上与标准 SQL 命令ROLLBACK完全相同，只是出于历史原因而保留。

参数

`WORK`
`TRANSACTION`

可选关键字，没有任何作用。

注解

使用COMMIT可成功地终止一个事务。

在事务之外发出`ABORT`并无害处，但会产生一条警告消息。

示例

要中止所有更改：

```
ABORT;
```

兼容性

这个命令是PostgreSQL扩展，只是出于历史原因而保留。`ROLLBACK`是与之等效的标准 SQL 命令。

另见

BEGIN, COMMIT, ROLLBACK

ALTER AGGREGATE

ALTER AGGREGATE — 更改一个聚合函数的定义

大纲

```
ALTER AGGREGATE name ( argtype [ , ... ] ) RENAME TO new_name
ALTER AGGREGATE name ( argtype [ , ... ] ) OWNER TO new_owner
ALTER AGGREGATE name ( argtype [ , ... ] ) SET SCHEMA new_schema
```

描述

`ALTER AGGREGATE`更改聚合函数的定义。

要使用`ALTER AGGREGATE`，必须拥有该聚合函数。要更改聚合函数的模式，还必须在新模式上拥有`CREATE`权限。要更改所有者，还必须是新所有者角色的直接或间接成员，并且该角色必须在聚合函数所在模式上拥有`CREATE`权限。（这些限制确保更改所有者不会执行任何无法通过删除并重新创建聚合函数完成的操作。不过，超级用户无论如何都可以更改任何聚合函数的所有权。）

参数

name

现有聚合函数的名称（可以用模式限定）。

argtype

聚合函数作用于其上的输入数据类型。要引用零参数聚合函数，请在参数说明列表的位置写上`*`。

new_name

聚合函数的新名称。

new_owner

聚合函数的新拥有者。

new_schema

聚合函数的新模式。

示例

要把用于类型`integer`的聚合函数 `myavg`重命名为`my_average`：

```
ALTER AGGREGATE myavg(integer) RENAME TO my_average;
```

要把用于类型`integer`的聚合函数 `myavg`的拥有者改为`joe`：

```
ALTER AGGREGATE myavg(integer) OWNER TO joe;
```

要把用于类型`integer`的聚合函数 `myavg`移动到模式`myschema`中：

```
ALTER AGGREGATE myavg(integer) SET SCHEMA myschema;
```

兼容性

在 SQL 标准中没有`ALTER AGGREGATE`语句。

另见

CREATE AGGREGATE, DROP AGGREGATE

ALTER COLLATION

ALTER COLLATION — 更改排序规则的定义

大纲

```
ALTER COLLATION name RENAME TO new_name
ALTER COLLATION name OWNER TO new_owner
ALTER COLLATION name SET SCHEMA new_schema
```

描述

`ALTER COLLATION`更改排序规则的定义。

要使用`ALTER COLLATION`，必须拥有该排序规则。要更改所有者，还必须是新所有者角色的直接或间接成员，并且该角色必须在排序规则所在模式上拥有`CREATE`权限。
（这些限制确保更改所有者不会执行任何无法通过删除并重新创建排序规则完成的操作。
不过，超级用户无论如何都可以更改任何排序规则的所有权。）

参数

name

一个现有排序规则的名称（可以是模式限定的）。

new_name

排序规则的新名称。

new_owner

排序规则的新拥有者。

new_schema

排序规则的新模式。

示例

要将排序规则`de_DE`重命名为`german`：

```
ALTER COLLATION "de_DE" RENAME TO german;
```

要将排序规则`en_US`的拥有者改为`joe`：

```
ALTER COLLATION "en_US" OWNER TO joe;
```

兼容性

SQL 标准中没有`ALTER COLLATION`语句。

参见

CREATE COLLATION, DROP COLLATION

ALTER CONVERSION

ALTER CONVERSION — 更改一个转换的定义

大纲

```
ALTER CONVERSION name RENAME TO new_name
ALTER CONVERSION name OWNER TO new_owner
ALTER CONVERSION name SET SCHEMA new_schema
```

描述

`ALTER CONVERSION`更改一个转换的定义。

要使用`ALTER CONVERSION`，必须拥有该转换。要更改所有者，还必须是新所有者角色的直接或间接成员，并且该角色必须在转换所在模式上拥有`CREATE`权限。
（这些限制确保更改所有者不会执行任何无法通过删除并重新创建转换完成的操作。不过，超级用户无论如何都可以更改任何转换的所有权。）

参数

name

一个现有转换的名称（可以是模式限定的）。

new_name

转换的新名称。

new_owner

转换的新拥有者。

new_schema

转换的新模式。

示例

要把转换`iso_8859_1_to_utf8`重命名为`latin1_to_unicode`：

```
ALTER CONVERSION iso_8859_1_to_utf8 RENAME TO latin1_to_unicode;
```

要把转换`iso_8859_1_to_utf8`的拥有者改成`joe`：

```
ALTER CONVERSION iso_8859_1_to_utf8 OWNER TO joe;
```

兼容性

在 SQL 标准中没有`ALTER CONVERSION`语句。

另见

CREATE CONVERSION, DROP CONVERSION

ALTER DATABASE

ALTER DATABASE — 更改一个数据库

大纲

```
ALTER DATABASE name [ [ WITH ] option [ ... ] ]
```

其中*option*可以是：

```
    CONNECTION LIMIT connlimit

ALTER DATABASE name RENAME TO new_name

ALTER DATABASE name OWNER TO new_owner

ALTER DATABASE name SET TABLESPACE new_tablespace

ALTER DATABASE name SET configuration_parameter { TO | = } { value
 | DEFAULT }
ALTER DATABASE name SET configuration_parameter FROM CURRENT
ALTER DATABASE name RESET configuration_parameter
ALTER DATABASE name RESET ALL
```

描述

`ALTER DATABASE`更改一个数据库的属性。

第一种形式更改某些数据库级设置（详见下文）。
只有数据库拥有者或者超级用户才能更改这些设置。

第二种形式更改数据库的名称。只有数据库拥有者或者超级用户可以重命名数据库；
非超级用户拥有者还必须具有`CREATEDB`权限。
当前连接的数据库不能被重命名（如果需要这样做，请连接到另一个数据库）。

第三种形式更改数据库的所有者。要更改所有者，必须拥有该数据库，
并且还必须是新所有者角色的直接或间接成员，并且必须拥有`CREATEDB`权限。（请注意，
超级用户会自动拥有所有这些权限。）

第四种形式更改数据库的默认表空间。只有数据库拥有者或者超级用户可以这样做；
你还必须对新表空间具有创建权限。
该命令会将数据库旧默认表空间中的所有表和索引在物理上移动到新表空间中。
对于该数据库而言，新默认表空间必须为空，并且不能有人连接到该数据库。
位于非默认表空间中的表和索引不受影响。

其余形式会更改某个PostgreSQL数据库的运行时配置变量的会话默认值。
此后每当在该数据库中启动一个新会话时，指定的值就会成为会话默认值。
数据库特定的默认值会覆盖`postgresql.conf`中的设置，或者从`postgres`
命令行接收到的设置。只有数据库拥有者或者超级用户才能更改该数据库的会话默认值。
某些变量不能以这种方式设置，或者只能由超级用户设置。

参数

name

要修改其属性的数据库名称。

connlimit

这个数据库允许多少并发连接。-1 表示没有限制。

new_name

数据库的新名称。

new_owner

数据库的新拥有者。

new_tablespace

数据库的新默认表空间。

configuration_parameter
value

将此数据库在指定配置参数上的会话默认值设为给定值。如果 *value*是DEFAULT，或者等效地使用了 RESET，则数据库特定设置会被移除，因此新会话将继承系统范围的默认设置。使用 RESET ALL可清除所有数据库特定设置。SET FROM CURRENT会把该参数在当前会话中的值保存为数据库特定值。

关于允许的参数名和值的更多信息，见SET和第 18 章。

注解

也可以把会话默认值绑定到特定角色，而不是数据库；见ALTER ROLE。如果发生冲突，角色特定设置会覆盖数据库特定设置。

示例

要在数据库test中默认禁用索引扫描：

```
ALTER DATABASE test SET enable_indexscan TO off;
```

兼容性

ALTER DATABASE语句是一个PostgreSQL扩展。

参见

CREATE DATABASE, DROP DATABASE, SET, CREATE TABLESPACE

ALTER DEFAULT PRIVILEGES

ALTER DEFAULT PRIVILEGES — 定义默认访问权限

大纲

```
ALTER DEFAULT PRIVILEGES
    [ FOR { ROLE | USER } target_role [, ...] ]
    [ IN SCHEMA schema_name [, ...] ]
    abbreviated_grant_or_revoke
```

其中*abbreviated_grant_or_revoke*为以下之一:

```
GRANT { { SELECT | INSERT | UPDATE | DELETE | TRUNCATE | REFERENCES
 | TRIGGER }
    [, ...] | ALL [ PRIVILEGES ] }
    ON TABLES
    TO { [ GROUP ] role_name | PUBLIC } [, ...] [ WITH GRANT
 OPTION ]

GRANT { { USAGE | SELECT | UPDATE }
    [, ...] | ALL [ PRIVILEGES ] }
    ON SEQUENCES
    TO { [ GROUP ] role_name | PUBLIC } [, ...] [ WITH GRANT
 OPTION ]

GRANT { EXECUTE | ALL [ PRIVILEGES ] }
    ON FUNCTIONS
    TO { [ GROUP ] role_name | PUBLIC } [, ...] [ WITH GRANT
 OPTION ]

GRANT { USAGE | ALL [ PRIVILEGES ] }
    ON TYPES
    TO { [ GROUP ] role_name | PUBLIC } [, ...] [ WITH GRANT
 OPTION ]

REVOKE [ GRANT OPTION FOR ]
    { { SELECT | INSERT | UPDATE | DELETE | TRUNCATE | REFERENCES |
 TRIGGER }
    [, ...] | ALL [ PRIVILEGES ] }
    ON TABLES
    FROM { [ GROUP ] role_name | PUBLIC } [, ...]
    [ CASCADE | RESTRICT ]

REVOKE [ GRANT OPTION FOR ]
    { { USAGE | SELECT | UPDATE }
    [, ...] | ALL [ PRIVILEGES ] }
    ON SEQUENCES
    FROM { [ GROUP ] role_name | PUBLIC } [, ...]
    [ CASCADE | RESTRICT ]

REVOKE [ GRANT OPTION FOR ]
    { EXECUTE | ALL [ PRIVILEGES ] }
    ON FUNCTIONS
    FROM { [ GROUP ] role_name | PUBLIC } [, ...]
    [ CASCADE | RESTRICT ]
```

```
REVOKE [ GRANT OPTION FOR ]
    { USAGE | ALL [ PRIVILEGES ] }
    ON TYPES
    FROM { [ GROUP ] role_name | PUBLIC } [, ...]
    [ CASCADE | RESTRICT ]
```

描述

`ALTER DEFAULT PRIVILEGES`允许设置将应用于将来创建的对象的权限。（它不影响已存在对象的权限。）目前只能更改表（包括视图和外部表）、序列、函数和类型（包括域）的权限。

虽然你可以修改自己的默认权限，也可以修改你所属角色的默认权限，但在创建对象时，新对象的权限只受当前角色的默认权限影响，不会从当前角色所属的任何角色继承。
权限可以全局设置（即针对当前数据库中创建的所有对象），
也可以只针对在指定模式中创建的对象。
按模式指定的默认权限会添加到特定对象类型的全局默认权限中。

如GRANT下所述，任何对象类型的默认权限通常会向对象所有者授予所有可授予的权限，也可能向`PUBLIC`授予一些权限。不过，可以通过使用`ALTER DEFAULT PRIVILEGES`更改全局默认权限来改变这种行为。

参数

target_role

当前角色所属的现有角色的名称。如果省略`FOR ROLE`，则假定为当前角色。

schema_name

一个现有模式的名称。如果指定，将修改以后在该模式中创建的对象的默认权限。
如果省略`IN SCHEMA`，则修改全局默认权限。

role_name

要为其授予或撤销权限的现有角色的名称。此参数以及 *abbreviated_grant_or_revoke* 中的所有其他参数，均按GRANT或REVOKE中的描述工作，只不过这里设置的是整类对象的权限，而不是特定的具名对象。

注解

使用psql的`\ddp`命令获取现有默认权限分配的信息。权限值的含义与GRANT下对`\dp`的说明相同。

如果你希望删除一个其默认权限已被修改的角色，则必须先撤销对其默认权限所做的更改，或者使用`DROP OWNED BY`去除该角色的默认权限条目。

示例

为你后续在模式`myschema`中创建的所有表（以及视图）向所有人授予 SELECT 权限，并且也允许角色`webuser`对它们执行 INSERT：

```
ALTER DEFAULT PRIVILEGES IN SCHEMA myschema GRANT SELECT ON TABLES
 TO PUBLIC;
ALTER DEFAULT PRIVILEGES IN SCHEMA myschema GRANT INSERT ON TABLES
 TO webuser;
```

撤销上述操作，使后续创建的表不再拥有任何超出正常情况的权限：

```
ALTER DEFAULT PRIVILEGES IN SCHEMA myschema REVOKE SELECT ON TABLES
 FROM PUBLIC;
ALTER DEFAULT PRIVILEGES IN SCHEMA myschema REVOKE INSERT ON TABLES
 FROM webuser;
```

移除通常会在函数上授予的公共 EXECUTE 权限，使后续由角色`admin`创建的所有函数不再自动具有该权限：

```
ALTER DEFAULT PRIVILEGES FOR ROLE admin REVOKE EXECUTE ON FUNCTIONS
 FROM PUBLIC;
```

兼容性

在 SQL 标准中没有`ALTER DEFAULT PRIVILEGES`语句。

另见

GRANT, REVOKE

ALTER DOMAIN

ALTER DOMAIN — 更改一个域的定义

大纲

```
ALTER DOMAIN name
    { SET DEFAULT expression | DROP DEFAULT }
ALTER DOMAIN name
    { SET | DROP } NOT NULL
ALTER DOMAIN name
    ADD domain_constraint [ NOT VALID ]
ALTER DOMAIN name
    DROP CONSTRAINT [ IF EXISTS ] constraint_name [ RESTRICT |
 CASCADE ]
ALTER DOMAIN name
     RENAME CONSTRAINT constraint_name TO new_constraint_name
ALTER DOMAIN name
    VALIDATE CONSTRAINT constraint_name
ALTER DOMAIN name
    OWNER TO new_owner
ALTER DOMAIN name
    RENAME TO new_name
ALTER DOMAIN name
    SET SCHEMA new_schema
```

描述

`ALTER DOMAIN`更改一个现有域的定义。有以下几种子形式：

SET/DROP DEFAULT

这些形式为域设置或移除默认值。注意，默认值只适用于后续的 `INSERT`命令；它们不会影响已存在于使用该域的表中的行。

SET/DROP NOT NULL

这些形式更改域是被标记为允许 NULL 值还是拒绝 NULL 值。只有当使用该域的列中不包含空值时，才能执行`SET NOT NULL`。

ADD *domain_constraint* [NOT VALID]

此形式使用与`CREATE DOMAIN`相同的语法向域添加新约束。向域添加新约束时，所有使用该域的列都会根据新添加的约束进行检查。可以使用`NOT VALID`选项添加新约束来抑制这些检查；之后可以使用`ALTER DOMAIN ... VALIDATE CONSTRAINT`验证该约束。新插入或更新的行始终根据所有约束进行检查，即使这些约束标记为`NOT VALID`。`NOT VALID`只能用于`CHECK`约束。

DROP CONSTRAINT [IF EXISTS]

这种形式删除域上的约束。如果指定了`IF EXISTS`而该约束不存在，则不会抛出错误，而是发出一条提示。

RENAME CONSTRAINT

这种形式更改域上某个约束的名称。

VALIDATE CONSTRAINT

这种形式验证一个先前以NOT VALID方式添加的约束，也就是说，验证使用该域的所有列中的数据都满足指定的约束。

OWNER

这种形式将域的拥有者更改为指定用户。

RENAME

这种形式更改域的名称。

SET SCHEMA

这种形式更改域所在的模式。与该域关联的任何约束也会一并移动到新模式中。

要使用ALTER DOMAIN，必须拥有该域。要更改域的模式，还必须在新模式上拥有CREATE权限。要更改所有者，还必须是新所有者角色的直接或间接成员，
并且该角色必须在域所在模式上拥有CREATE权限。
（这些限制确保更改所有者不会执行任何无法通过删除并重新创建域完成的操作。不过，
超级用户无论如何都可以更改任何域的所有权。）

参数

name

要修改的现有域的名称（可以是模式限定的）。

domain_constraint

该域的新约束。

constraint_name

要删除或重命名的现有约束名称。

NOT VALID

不验证现有列数据是否满足该约束。

CASCADE

自动删除依赖于该约束的对象。

RESTRICT

如果存在任何依赖对象，则拒绝删除该约束。这是默认行为。

new_name

域的新名称。

new_constraint_name

约束的新名称。

new_owner

域的新拥有者的用户名。

new_schema

域的新模式。

注解

目前，如果要验证的命名域或任何派生域被数据库中任意表的复合类型列使用，`ALTER DOMAIN ADD CONSTRAINT`和`ALTER DOMAIN SET NOT NULL`会失败。最终应改进这些命令，使其能够验证此类嵌套列上的新约束。

示例

向域添加一个`NOT NULL`约束：

```
ALTER DOMAIN zipcode SET NOT NULL;
```

从域中移除一个`NOT NULL`约束：

```
ALTER DOMAIN zipcode DROP NOT NULL;
```

向域添加一个检查约束：

```
ALTER DOMAIN zipcode ADD CONSTRAINT zipchk CHECK (char_length(
VALUE) = 5);
```

从域中移除一个检查约束：

```
ALTER DOMAIN zipcode DROP CONSTRAINT zipchk;
```

重命名域上的一个检查约束：

```
ALTER DOMAIN zipcode RENAME CONSTRAINT zipchk TO zip_check;
```

将域移动到另一个模式中：

```
ALTER DOMAIN zipcode SET SCHEMA customers;
```

兼容性

`ALTER DOMAIN`符合SQL标准，但 `OWNER`、`RENAME`、`SET SCHEMA`和`VALIDATE CONSTRAINT` 这些变体是PostgreSQL扩展；`ADD CONSTRAINT`变体中的`NOT VALID`子句也是PostgreSQL扩展。

另见

CREATE DOMAIN, DROP DOMAIN

ALTER EXTENSION

ALTER EXTENSION — 更改扩展的定义

大纲

```
ALTER EXTENSION name UPDATE [ TO new_version ]
ALTER EXTENSION name SET SCHEMA new_schema
ALTER EXTENSION name ADD member_object
ALTER EXTENSION name DROP member_object
```

其中*member_object*为:

```
  AGGREGATE agg_name (agg_type [, ...] ) |
  CAST (source_type AS target_type) |
  COLLATION object_name |
  CONVERSION object_name |
  DOMAIN object_name |
  FOREIGN DATA WRAPPER object_name |
  FOREIGN TABLE object_name |
  FUNCTION function_name ( [ [ argmode ] [ argname ] argtype [, ...
] ] ) |
  OPERATOR operator_name (left_type, right_type) |
  OPERATOR CLASS object_name USING index_method |
  OPERATOR FAMILY object_name USING index_method |
  [ PROCEDURAL ] LANGUAGE object_name |
  SCHEMA object_name |
  SEQUENCE object_name |
  SERVER object_name |
  TABLE object_name |
  TEXT SEARCH CONFIGURATION object_name |
  TEXT SEARCH DICTIONARY object_name |
  TEXT SEARCH PARSER object_name |
  TEXT SEARCH TEMPLATE object_name |
  TYPE object_name |
  VIEW object_name
```

描述

`ALTER EXTENSION`更改已安装扩展的定义。其子形式如下:

`UPDATE`

这种形式将扩展更新到较新的版本。该扩展必须提供适当的更新脚本（或一系列脚本），能够将当前已安装版本修改为所请求的版本。

`SET SCHEMA`

这种形式将扩展的对象移入另一个模式。要使该命令成功，扩展必须是可重定位的。

`ADD` *member_object*

这种形式将一个现有对象加入扩展。这主要在扩展更新脚本中有用。
该对象随后将被视为扩展的成员；特别是，只能通过删除该扩展来删除它。

DROP *member_object*

这种形式从扩展中移除一个成员对象。这主要在扩展更新脚本中有用。该对象不会被删除，而只是与扩展解除关联。

有关这些操作的更多信息，见第 35.15 节。

要使用ALTER EXTENSION，你必须拥有该扩展。ADD/DROP形式还要求你拥有被添加/移除对象的所有权。

参数

name

一个已安装扩展的名称。

new_version

扩展要更新到的新版本。它可以写成标识符或字符串常量。如果未指定，ALTER EXTENSION UPDATE会尝试更新到该扩展控制文件中标明的默认版本。

new_schema

扩展的新模式。

object_name
agg_name
function_name
operator_name

要添加到扩展或从扩展中移除的对象名称。表、聚合、域、外部表、函数、操作符、操作符类、操作符族、序列、文本检索对象、类型和视图的名称可以带模式限定。

agg_type

聚合函数作用于其上的输入数据类型。要引用零参数聚合函数，请在输入数据类型列表的位置写*。

source_type

类型转换的源数据类型的名称。

target_type

类型转换的目标数据类型的名称。

argmode

函数参数的模式：IN、OUT、INOUT或VARIADIC。如果省略，默认值为IN。请注意，ALTER EXTENSION实际上不会关注OUT参数，因为确定函数身份只需要输入参数。因此，列出IN、INOUT和VARIADIC参数就足够了。

argname

函数参数的名称。请注意，ALTER EXTENSION实际上不会关注参数名称，因为确定函数身份只需要参数数据类型。

argtype

函数参数的参数数据类型（可带模式限定），如果有的话。

left_type
right_type

操作符参数的数据类型（可带模式限定）。对于前缀或后缀操作符缺少的参数，写作NONE。

PROCEDURAL

这是一个噪声词。

示例

将hstore扩展更新到版本 2.0：

```
ALTER EXTENSION hstore UPDATE TO '2.0';
```

将hstore扩展的模式更改为utils：

```
ALTER EXTENSION hstore SET SCHEMA utils;
```

将现有函数添加到hstore扩展中：

```
ALTER EXTENSION hstore ADD FUNCTION populate_record(anyelement,
 hstore);
```

兼容性

ALTER EXTENSION是一个PostgreSQL 扩展。

另见

CREATE EXTENSION, DROP EXTENSION

ALTER FOREIGN DATA WRAPPER

ALTER FOREIGN DATA WRAPPER — 更改外部数据包装器的定义

大纲

```
ALTER FOREIGN DATA WRAPPER name
    [ HANDLER handler_function | NO HANDLER ]
    [ VALIDATOR validator_function | NO VALIDATOR ]
    [ OPTIONS ( [ ADD | SET | DROP ] option ['value'] [, ... ]) ]
ALTER FOREIGN DATA WRAPPER name OWNER TO new_owner
ALTER FOREIGN DATA WRAPPER name RENAME TO new_name
```

描述

`ALTER FOREIGN DATA WRAPPER`更改外部数据包装器的定义。该命令的第一种形式更改外部数据包装器的支持函数或通用选项（至少需要一个子句）。
第二种形式更改外部数据包装器的拥有者。

只有超级用户可以更改外部数据包装器。此外，
只有超级用户才能成为外部数据包装器的拥有者。

参数

name

一个现有外部数据包装器的名称。

`HANDLER` *handler_function*

为外部数据包装器指定一个新的处理器函数。

`NO HANDLER`

用于指定该外部数据包装器不再具有一个处理器函数。

注意，使用没有处理器函数的外部数据包装器的外部表将无法访问。

`VALIDATOR` *validator_function*

为外部数据包装器指定一个新的验证器函数。

注意，更改验证器之后，外部数据包装器、服务器和用户映射的选项可能已变得无效。
用户必须在使用该外部数据包装器之前自行确保这些选项正确。

`NO VALIDATOR`

用于指定该外部数据包装器不再具有验证器函数。

`OPTIONS ( [ ADD | SET | DROP ] option ['value'] [, ... ] )`

更改外部数据包装器的选项。`ADD`、`SET` 和`DROP`指定要执行的动作。如果未显式指定操作，则假定为`ADD`。选项名称必须唯一；如果存在验证器函数，
选项名称和值也会使用该外部数据包装器的验证器函数进行验证。

new_owner

该外部数据包装器的新拥有者的用户名。

new_name

该外部数据包装器的新名称。

示例

更改一个外部数据包装器dbi，添加选项foo，删除bar：

```
ALTER FOREIGN DATA WRAPPER dbi OPTIONS (ADD foo '1', DROP 'bar');
```

将外部数据包装器dbi的验证器更改为 bob.myvalidator：

```
ALTER FOREIGN DATA WRAPPER dbi VALIDATOR bob.myvalidator;
```

兼容性

ALTER FOREIGN DATA WRAPPER符合 ISO/IEC 9075-9 (SQL/MED)，但HANDLER、VALIDATOR、OWNER TO和 RENAME子句属于扩展。

另见

CREATE FOREIGN DATA WRAPPER, DROP FOREIGN DATA WRAPPER

ALTER FOREIGN TABLE

ALTER FOREIGN TABLE — 更改外部表的定义

大纲

```
ALTER FOREIGN TABLE [ IF EXISTS ] name
    action [, ... ]
ALTER FOREIGN TABLE [ IF EXISTS ] name
    RENAME [ COLUMN ] column_name TO new_column_name
ALTER FOREIGN TABLE [ IF EXISTS ] name
    RENAME TO new_name
ALTER FOREIGN TABLE [ IF EXISTS ] name
    SET SCHEMA new_schema
```

其中*action*为以下之一：

```
    ADD [ COLUMN ] column_name data_type [ NULL | NOT NULL ]
    DROP [ COLUMN ] [ IF EXISTS ] column_name [ RESTRICT |
 CASCADE ]
    ALTER [ COLUMN ] column_name [ SET DATA ] TYPE data_type
            ALTER [ COLUMN ] column_name { SET | DROP } NOT NULL
    ALTER [ COLUMN ] column_name SET STATISTICS integer
    ALTER [ COLUMN ] column_name SET ( attribute_option = value [,
 ... ] )
    ALTER [ COLUMN ] column_name RESET ( attribute_option [, ... ]
 )
    ALTER [ COLUMN ] column_name OPTIONS ( [ ADD | SET | DROP ]
 option ['value'] [, ... ])
                     OWNER TO new_owner
    OPTIONS ( [ ADD | SET | DROP ] option ['value'] [, ... ])
```

描述

`ALTER FOREIGN TABLE`更改现有外部表的定义。有几种子形式：

`ADD COLUMN`

该形式使用与CREATE FOREIGN TABLE相同的语法向外部表添加一个新列。

`DROP COLUMN [ IF EXISTS ]`

该形式从外部表中删除一列。如果该表之外的任何对象依赖于该列，例如视图，则需要指定`CASCADE`。如果指定了`IF EXISTS` 而该列不存在，则不会抛出错误，而是改为发出一条提示。

`IF EXISTS`

如果外部表不存在，则不抛出错误。这种情况下会发出一个提示。

`SET DATA TYPE`

该形式更改外部表中一列的类型。

`SET/DROP NOT NULL`

把一列标记为允许或不允许空值。

`SET STATISTICS`

该形式为后续ANALYZE操作设置每列的统计信息收集目标。详见ALTER TABLE的类似形式。

`SET ( attribute_option = value [, ... ] )`
`RESET ( attribute_option [, ... ] )`

该形式设置或重置每个属性的选项。详见ALTER TABLE的类似形式。

`OWNER`

该形式把外部表的拥有者更改为指定用户。

`OPTIONS ( [ ADD | SET | DROP ] option ['value'] [, ... ] )`

更改外部表或其某一列的选项。ADD、SET 和DROP指定要执行的动作。如果没有显式指定操作，则默认是 ADD。不允许重复的选项名（不过表选项和列选项同名是可以的）。选项名和值也会通过外部数据包装器库进行验证。

`RENAME`

RENAME形式更改外部表的名称，或者更改其中某一列的名称。

`SET SCHEMA`

该形式把外部表移动到另一个模式中。

除了RENAME和SET SCHEMA之外，所有动作都可以组合在一个包含多项修改的列表中一并应用。例如，可以在一条命令中添加多个列和/或更改多个列的类型。

要使用ALTER FOREIGN TABLE，必须拥有该表。要更改外部表的模式，还必须在新模式上拥有CREATE权限。要更改所有者，还必须是新所有者角色的直接或间接成员，并且该角色必须在表所在模式上拥有CREATE权限。（这些限制确保更改所有者不会执行任何无法通过删除并重新创建该表完成的操作。不过，超级用户无论如何都可以更改任何表的所有权。）要添加列或更改列类型，还必须在数据类型上拥有USAGE权限。

参数

name

要修改的现有外部表的名称（可以带模式限定）。

column_name

新列或现有列的名称。

new_column_name

现有列的新名称。

new_name

表的新名称。

data_type

新列的数据类型，或现有列的新数据类型。

`CASCADE`

自动删除依赖于该列的对象（例如引用该列的视图）。

`RESTRICT`

如果存在任何依赖对象，则拒绝删除该列。这是默认行为。

`new_owner`

该表的新拥有者的用户名。

`new_schema`

该表将被移动到的模式名称。

注解

关键字`COLUMN`只是噪声，可以省略。

使用`ADD COLUMN`或`DROP COLUMN`添加或移除列，添加`NOT NULL`约束，或使用`SET DATA TYPE`更改列类型时，不会检查与外部服务器的一致性。用户有责任确保表定义与远端一致。

关于有效参数的进一步说明，请参见CREATE FOREIGN TABLE。

示例

要把一列标记为非空：

```
ALTER FOREIGN TABLE distributors ALTER COLUMN street SET NOT NULL;
```

要更改外部表的选项：

```
ALTER FOREIGN TABLE myschema.distributors OPTIONS (ADD opt1
 'value', SET opt2, 'value2', DROP opt3 'value3');
```

兼容性

`ADD`、`DROP`和`SET DATA TYPE` 形式符合 SQL 标准。其他形式是 PostgreSQL对 SQL 标准的扩展。另外，在单条`ALTER FOREIGN TABLE`命令中指定多个修改也是一种扩展。

`ALTER FOREIGN TABLE DROP COLUMN`可用于删除外部表的唯一一列，从而留下一个零列的表。这是 SQL 的一种扩展，因为 SQL 不允许零列外部表。

ALTER FUNCTION

ALTER FUNCTION — 更改函数的定义

大纲

```
ALTER FUNCTION name ( [ [ argmode ] [ argname ] argtype [, ...] ] )
    action [ ... ] [ RESTRICT ]
ALTER FUNCTION name ( [ [ argmode ] [ argname ] argtype [, ...] ] )
    RENAME TO new_name
ALTER FUNCTION name ( [ [ argmode ] [ argname ] argtype [, ...] ] )
    OWNER TO new_owner
ALTER FUNCTION name ( [ [ argmode ] [ argname ] argtype [, ...] ] )
    SET SCHEMA new_schema
```

其中*action*为以下之一：

```
    CALLED ON NULL INPUT | RETURNS NULL ON NULL INPUT | STRICT
    IMMUTABLE | STABLE | VOLATILE | [ NOT ] LEAKPROOF
    [ EXTERNAL ] SECURITY INVOKER | [ EXTERNAL ] SECURITY DEFINER
    COST execution_cost
    ROWS result_rows
    SET configuration_parameter { TO | = } { value | DEFAULT }
    SET configuration_parameter FROM CURRENT
    RESET configuration_parameter
    RESET ALL
```

描述

`ALTER FUNCTION`更改函数的定义。

要使用`ALTER FUNCTION`，必须拥有该函数。要更改函数的模式，还必须在新模式上拥有`CREATE`权限。要更改所有者，还必须是新所有者角色的直接或间接成员，
并且该角色必须在函数所在模式上拥有`CREATE`权限。
（这些限制确保更改所有者不会执行任何无法通过删除并重新创建函数完成的操作。不过，超级用户无论如何都可以更改任何函数的所有权。）

参数

name

现有函数的名称（可以用模式限定）。

argmode

参数的模式：`IN`、`OUT`、`INOUT`或`VARIADIC`。如果省略，默认为`IN`。注意`ALTER FUNCTION` 实际上并不关心任何`OUT`参数，因为确定函数身份只需要输入参数。因此，只列出`IN`、`INOUT`和`VARIADIC`参数就足够了。

argname

参数的名称。注意`ALTER FUNCTION`实际上并不关心参数名，因为确定函数身份只需要参数数据类型。

argtype

函数参数的数据类型（如果有，可以用模式限定）。

new_name

该函数的新名称。

new_owner

该函数的新拥有者。注意，如果该函数被标记为 SECURITY DEFINER，随后它将以新拥有者的身份执行。

new_schema

该函数的新模式。

CALLED ON NULL INPUT
RETURNS NULL ON NULL INPUT
STRICT

CALLED ON NULL INPUT将该函数改为在其部分或全部参数为 null 时仍会被调用。RETURNS NULL ON NULL INPUT或 STRICT将该函数改为只要任一参数为 null 就不调用，而是自动假定结果为 null。详见CREATE FUNCTION。

IMMUTABLE
STABLE
VOLATILE

将该函数的易变性更改为指定的设置。详见CREATE FUNCTION。

[EXTERNAL] SECURITY INVOKER
[EXTERNAL] SECURITY DEFINER

更改该函数是否为安全性定义者。关键字EXTERNAL 为了符合 SQL 标准而被忽略。关于这一能力的更多信息，参见CREATE FUNCTION。

LEAKPROOF

更改该函数是否被视为防泄漏的。关于这一能力的更多信息，参见CREATE FUNCTION。

COST *execution_cost*

更改该函数的估计执行代价。详见CREATE FUNCTION。

ROWS *result_rows*

更改集合返回函数估计返回的行数。详见CREATE FUNCTION。

configuration_parameter
value

添加或更改在调用该函数时要赋给某个配置参数的值。如果 *value*是DEFAULT，或者等效地使用了RESET，则会移除函数局部设置，
这样该函数就会使用其环境中现有的值执行。使用RESET ALL 可以清除所有函数局部设置。SET FROM CURRENT 会把执行ALTER FUNCTION时该参数的当前值保存为进入函数时要应用的值。

关于允许的参数名和值的更多信息，参见SET和第 18 章。

RESTRICT

为符合 SQL 标准而被忽略。

示例

要将类型integer的函数sqrt 重命名为square_root：

```
ALTER FUNCTION sqrt(integer) RENAME TO square_root;
```

要将类型integer的函数sqrt 的拥有者改为joe：

```
ALTER FUNCTION sqrt(integer) OWNER TO joe;
```

要将类型integer的函数sqrt 的模式改为maths：

```
ALTER FUNCTION sqrt(integer) SET SCHEMA maths;
```

要调整自动为函数设置的搜索路径：

```
ALTER FUNCTION check_password(text) SET search_path = admin, pg_
temp;
```

要禁用自动为函数设置的search_path：

```
ALTER FUNCTION check_password(text) RESET search_path;
```

现在该函数将使用其调用者所使用的搜索路径执行。

兼容性

该语句与 SQL 标准中的ALTER FUNCTION语句部分兼容。标准允许修改函数的更多属性，但不提供重命名函数、将函数设为安全性定义者、为函数附加配置参数值，
或者更改函数的拥有者、模式或易变性的能力。标准还要求使用RESTRICT关键字，而它在PostgreSQL中是可选的。

另见

CREATE FUNCTION, DROP FUNCTION

ALTER GROUP

ALTER GROUP — 更改角色名称或成员资格

大纲

```
ALTER GROUP role_name ADD USER user_name [, ... ]
ALTER GROUP role_name DROP USER user_name [, ... ]

ALTER GROUP group_name RENAME TO new_name
```

描述

`ALTER GROUP`更改用户组的属性。这是一个已废弃的命令，但出于向后兼容的考虑仍然接受它，因为组（以及用户）已经被更一般的角色概念所取代。

前两个变体将用户加入一个组，或将其从组中移除。（为此目的，任何角色都可以充当“用户”或“组”。）这些变体实际上等效于对名为“组”的角色授予或撤销成员资格，因此更推荐使用GRANT或REVOKE。

第三个变体更改组的名称。这完全等同于使用ALTER ROLE重命名角色。

参数

group_name

要修改的组（角色）的名称。

user_name

要加入该组或从该组中移除的用户（角色）。这些用户必须已经存在；`ALTER GROUP`不会创建或删除用户。

new_name

该组的新名称。

示例

向组中添加用户：

```
ALTER GROUP staff ADD USER karl, john;
```

从组中移除用户：

```
ALTER GROUP workers DROP USER beth;
```

兼容性

SQL 标准中没有`ALTER GROUP`语句。

另见

GRANT, REVOKE, ALTER ROLE

ALTER INDEX

ALTER INDEX — 更改索引的定义

大纲

```
ALTER INDEX [ IF EXISTS ] name RENAME TO new_name
ALTER INDEX [ IF EXISTS ] name SET TABLESPACE tablespace_name
ALTER INDEX [ IF EXISTS ] name SET ( storage_parameter = value [, .
.. ] )
ALTER INDEX [ IF EXISTS ] name RESET ( storage_parameter [, ... ] )
```

描述

`ALTER INDEX`更改现有索引的定义。有几种子形式：

`RENAME`

`RENAME`形式更改索引的名称。存储的数据不受影响。

`SET TABLESPACE`

该形式将索引的表空间更改为指定的表空间，
并将与索引关联的数据文件移动到新表空间。另请参见CREATE TABLESPACE。

`SET ( storage_parameter = value [, ... ] )`

该形式更改索引的一个或多个索引方法特定存储参数。有关可用参数的详情，
请参见CREATE INDEX。请注意，该命令不会立即修改索引内容；根据参数的不同，可能需要使用REINDEX重建索引才能达到预期效果。

`RESET ( storage_parameter [, ... ] )`

这种形式将一个或多个索引方法相关的存储参数重置为默认值。与 `SET`一样，可能仍需要执行一次`REINDEX` 才能让该索引被完全更新。

参数

`IF EXISTS`

如果该索引不存在，则不要抛出错误。这种情况下会发出一条提示。

`name`

要修改的现有索引名称（可以是模式限定的）。

`new_name`

该索引的新名称。

`tablespace_name`

该索引将被移动到的表空间。

`storage_parameter`

一个索引方法相关的存储参数的名称。

value

一个索引方法相关的存储参数的新值。根据具体参数，它可能是数字或一个词。

注解

这些操作也可以使用ALTER TABLE完成。实际上，`ALTER INDEX`只是`ALTER TABLE`中适用于索引的那些形式的别名。

以前曾有一种`ALTER INDEX OWNER`变体，但现在会被忽略（同时发出警告）。索引的拥有者不能不同于其所属表的拥有者。更改表的拥有者时，索引的拥有者也会自动更改。

不允许更改系统目录索引的任何部分。

示例

要重命名现有索引：

```
ALTER INDEX distributors RENAME TO suppliers;
```

要将索引移到另一个表空间：

```
ALTER INDEX distributors SET TABLESPACE fasttablespace;
```

要更改索引的填充因子（假设该索引方法支持fillfactor参数）：

```
ALTER INDEX distributors SET (fillfactor = 75);
REINDEX INDEX distributors;
```

兼容性

`ALTER INDEX`是一种 PostgreSQL扩展。

另见

CREATE INDEX, REINDEX

ALTER LANGUAGE

ALTER LANGUAGE — 更改一种过程语言的定义

大纲

```
ALTER [ PROCEDURAL ] LANGUAGE name RENAME TO new_name
ALTER [ PROCEDURAL ] LANGUAGE name OWNER TO new_owner
```

描述

`ALTER LANGUAGE`更改一种过程语言的定义。唯一的功能是重命名该语言或为其指定新的所有者。要使用`ALTER LANGUAGE`，你必须是超级用户或者该语言的所有者。

参数

name

语言的名称

new_name

该语言的新名称

new_owner

该语言的新所有者

兼容性

SQL 标准中没有`ALTER LANGUAGE`语句。

另见

CREATE LANGUAGE, DROP LANGUAGE

ALTER LARGE OBJECT

ALTER LARGE OBJECT — 更改一个大对象的定义

大纲

```
ALTER LARGE OBJECT large_object_oid OWNER TO new_owner
```

描述

`ALTER LARGE OBJECT`更改一个大对象的定义。目前唯一的功能是指定新的所有者。必须是超级用户或该大对象的拥有者，才能使用`ALTER LARGE OBJECT`。

参数

large_object_oid

要修改的大对象的 OID

new_owner

该大对象的新拥有者

兼容性

SQL 标准中没有`ALTER LARGE OBJECT`语句。

另见

第 32 章

ALTER OPERATOR

ALTER OPERATOR — 更改一个操作符的定义

大纲

```
ALTER OPERATOR name ( { left_type | NONE } , { right_type | NONE }
 ) OWNER TO new_owner

ALTER OPERATOR name ( { left_type | NONE } , { right_type | NONE }
 )
    SET SCHEMA new_schema
```

描述

`ALTER OPERATOR`更改一个操作符的定义。目前唯一可用的功能是更改操作符的拥有者。

要使用`ALTER OPERATOR`，必须拥有该操作符。要更改所有者，还必须是新所有者角色的直接或间接成员，并且该角色必须在操作符所在模式上拥有`CREATE`权限。
（这些限制确保更改所有者不会执行任何无法通过删除并重新创建操作符完成的操作。
不过，超级用户无论如何都可以更改任何操作符的所有权。）

参数

name

现有操作符的名称（可带模式限定）。

left_type

操作符左操作数的数据类型；如果操作符没有左操作数，写作`NONE`。

right_type

操作符右操作数的数据类型；如果操作符没有右操作数，写作`NONE`。

new_owner

操作符的新所有者。

new_schema

操作符的新模式。

示例

更改自定义操作符的限制和连接选择率估计函数，操作符为`a && b`，类型为`int[]`：

```
ALTER OPERATOR && (_int4, _int4) SET (RESTRICT = _int_contsel, JOIN
 = _int_contjoinsel);
```

兼容性

在 SQL 标准中没有`ALTER OPERATOR`语句。

另见

CREATE OPERATOR, DROP OPERATOR

ALTER OPERATOR CLASS

ALTER OPERATOR CLASS — 更改一个操作符类的定义

大纲

```
ALTER OPERATOR CLASS name USING index_method RENAME TO new_name
ALTER OPERATOR CLASS name USING index_method OWNER TO new_owner
ALTER OPERATOR CLASS name USING index_method SET SCHEMA new_schema
```

描述

`ALTER OPERATOR CLASS`更改一个操作符类的定义。

要使用`ALTER OPERATOR CLASS`，必须拥有该操作符类。要更改所有者，还必须是新所有者角色的直接或间接成员，并且该角色必须在操作符类所在模式上拥有`CREATE`权限。（这些限制确保更改所有者不会执行任何无法通过删除并重新创建操作符类完成的操作。不过，超级用户无论如何都可以更改任何操作符类的所有权。）

参数

name

一个现有操作符类的名称（可以是模式限定的）。

index_method

该操作符类所对应的索引方法的名称。

new_name

该操作符类的新名称。

new_owner

该操作符类的新拥有者。

new_schema

该操作符类的新模式。

兼容性

SQL 标准中没有`ALTER OPERATOR CLASS`语句。

另见

CREATE OPERATOR CLASS, DROP OPERATOR CLASS, ALTER OPERATOR FAMILY

ALTER OPERATOR FAMILY

ALTER OPERATOR FAMILY — 更改一个操作符族的定义

大纲

```
ALTER OPERATOR FAMILY name USING index_method ADD
  {  OPERATOR strategy_number operator_name ( op_type, op_type )
             [ FOR SEARCH | FOR ORDER BY sort_family_name ]
   | FUNCTION support_number [ ( op_type [ , op_type ] ) ]
             function_name ( argument_type [, ...] )
  } [, ... ]

ALTER OPERATOR FAMILY name USING index_method DROP
  {  OPERATOR strategy_number ( op_type [ , op_type ] )
   | FUNCTION support_number ( op_type [ , op_type ] )
  } [, ... ]

ALTER OPERATOR FAMILY name USING index_method
    RENAME TO new_name

ALTER OPERATOR FAMILY name USING index_method
    OWNER TO { new_owner | CURRENT_USER | SESSION_USER }

ALTER OPERATOR FAMILY name USING index_method
    SET SCHEMA new_schema
```

描述

`ALTER OPERATOR FAMILY`更改一个操作符族的定义。你可以向该族添加操作符和支持函数、从该族中移除它们，或者更改该族的名称或拥有者。

使用`ALTER OPERATOR FAMILY`向一个族中添加操作符和支持函数时，它们并不属于该族内任何特定的操作符类，而只是作为该族中的“松散”成员存在。
这表明这些操作符和函数与该族的语义兼容，但并不是任何特定索引正确工作所必需的。
（那些确实必需的操作符和函数应当声明为某个操作符类的一部分；参见CREATE OPERATOR CLASS。）PostgreSQL允许随时从一个族中删除松散成员，
但如果不删除整个类以及依赖它的任何索引，就不能删除操作符类的成员。通常，
单一数据类型的操作符和函数属于操作符类，因为支持该特定数据类型上的索引需要它们，
而跨数据类型的操作符和函数则作为该族中的松散成员。

要使用`ALTER OPERATOR FAMILY`，你必须是超级用户。（之所以这样限制，是因为错误的操作符族定义可能会使服务器混乱，甚至导致崩溃。）

`ALTER OPERATOR FAMILY`目前不会检查操作符族定义是否包括该索引方法所要求的全部操作符和函数，也不会检查这些操作符和函数是否构成一个自洽的集合。
定义一个合法的操作符族是用户的责任。

更多信息请参见第 35.14 节。

参数

name

一个现有操作符族的名称（可以是模式限定的）。

index_method

该操作符族所对应的索引方法名称。

strategy_number

与该操作符族关联的操作符的索引方法策略号。

operator_name

与该操作符族关联的操作符名称（可以是模式限定的）。

op_type

在OPERATOR子句中，操作符的操作数数据类型，或使用NONE
表示左一元或右一元操作符。与CREATE OPERATOR CLASS中的对应语法不同，必须始终指定操作数数据类型。

在ADD FUNCTION子句中，如果函数要支持的操作数数据类型不同于函数的输入数据类型，则指定这些数据类型。对于 B-树比较函数和哈希函数，不需要指定*op_type*，因为函数的输入数据类型始终是应使用的正确类型。对于 B-树排序支持函数以及 GiST、SP-GiST 和 GIN 操作符类中的所有函数，则必须指定该函数要配合使用的操作数数据类型。

在DROP FUNCTION子句中，必须指定该函数意图支持的操作数数据类型。

sort_family_name

一个现有btree操作符族的名称（可以是模式限定的），
它描述与某个排序操作符关联的排序顺序。

如果既没有指定FOR SEARCH，也没有指定 FOR ORDER BY，则默认值是FOR SEARCH。

support_number

与操作符族关联的函数的索引方法支持函数编号。

function_name

作为操作符族索引方法支持函数的函数名称（可带模式限定）。

argument_type

该函数的参数数据类型。

new_name

该操作符族的新名称。

new_owner

该操作符族的新拥有者。

new_schema

该操作符族所在的新模式。

OPERATOR和FUNCTION子句可以以任何顺序出现。

注解

注意，DROP
语法只通过策略号或支持函数编号以及输入数据类型来指定操作符族中的“槽位”，
不会提及占用该槽位的操作符或函数名称。另外，对于DROP FUNCTION，要指定的类型是该函数意图支持的输入数据类型；对于 GiST、SP-GiST 和 GIN 索引，这些类型可能与该函数实际的输入参数类型毫无关系。

因为索引机制在使用函数之前不会检查函数的访问权限，
将一个函数或者操作符包括在一个操作符族中相当于在其上授予公共执行权限。
这对操作符族中很有用的这类函数来说通常不成问题。

不应由 SQL 函数定义这些操作符。SQL 函数很可能会被内联到调用查询中，从而阻止优化器识别该查询与索引匹配。

在PostgreSQL 8.4 之前，`OPERATOR`子句可以包含`RECHECK`选项。现在不再支持该选项，因为索引操作符是否“有损”会在运行时动态确定。
这样可以高效处理操作符可能有损也可能无损的情况。

示例

下列示例命令向一个已经包含数据类型`int4`和`int2` 的 B-树操作符类的操作符族中添加跨数据类型的操作符和支持函数。

```
ALTER OPERATOR FAMILY integer_ops USING btree ADD

  -- int4 对 int2
  OPERATOR 1 < (int4, int2) ,
  OPERATOR 2 <= (int4, int2) ,
  OPERATOR 3 = (int4, int2) ,
  OPERATOR 4 >= (int4, int2) ,
  OPERATOR 5 > (int4, int2) ,
  FUNCTION 1 btint42cmp(int4, int2) ,

  -- int2 对 int4
  OPERATOR 1 < (int2, int4) ,
  OPERATOR 2 <= (int2, int4) ,
  OPERATOR 3 = (int2, int4) ,
  OPERATOR 4 >= (int2, int4) ,
  OPERATOR 5 > (int2, int4) ,
  FUNCTION 1 btint24cmp(int2, int4) ;
```

要再次移除这些条目：

```
ALTER OPERATOR FAMILY integer_ops USING btree DROP

  -- int4 对 int2
  OPERATOR 1 (int4, int2) ,
  OPERATOR 2 (int4, int2) ,
  OPERATOR 3 (int4, int2) ,
  OPERATOR 4 (int4, int2) ,
  OPERATOR 5 (int4, int2) ,
  FUNCTION 1 (int4, int2) ,

  -- int2 对 int4
  OPERATOR 1 (int2, int4) ,
  OPERATOR 2 (int2, int4) ,
  OPERATOR 3 (int2, int4) ,
  OPERATOR 4 (int2, int4) ,
  OPERATOR 5 (int2, int4) ,
  FUNCTION 1 (int2, int4) ;
```

兼容性

在 SQL 标准中没有 `ALTER OPERATOR FAMILY`语句。

另见

CREATE OPERATOR FAMILY, DROP OPERATOR FAMILY, CREATE OPERATOR CLASS, ALTER OPERATOR CLASS, DROP OPERATOR CLASS

ALTER ROLE

ALTER ROLE — 更改数据库角色

大纲

```
ALTER ROLE name [ [ WITH ] option [ ... ] ]
```

其中*option*可以是：

```
      SUPERUSER | NOSUPERUSER
    | CREATEDB | NOCREATEDB
    | CREATEROLE | NOCREATEROLE
    | CREATEUSER | NOCREATEUSER
    | INHERIT | NOINHERIT
    | LOGIN | NOLOGIN
    | REPLICATION | NOREPLICATION
    | CONNECTION LIMIT connlimit
    | [ ENCRYPTED | UNENCRYPTED ] PASSWORD 'password'
    | VALID UNTIL 'timestamp'

ALTER ROLE name RENAME TO new_name

ALTER ROLE name [ IN DATABASE database_name ] SET configuration_
parameter { TO | = } { value | DEFAULT }
ALTER ROLE name [ IN DATABASE database_name ] SET configuration_
parameter FROM CURRENT
ALTER ROLE name [ IN DATABASE database_name ] RESET configuration_
parameter
ALTER ROLE name [ IN DATABASE database_name ] RESET ALL
```

描述

`ALTER ROLE`更改 PostgreSQL角色的属性。

概要中列出的此命令的第一种形式可以更改CREATE ROLE中可指定的许多角色属性。（涵盖了所有可能的属性，但没有用于添加或移除成员资格的选项；请使用GRANT和REVOKE完成此操作。）命令中未提及的属性保留其以前的设置。数据库超级用户可以更改任何角色的任何这些设置。拥有`CREATEROLE`权限的角色可以更改任何这些设置，但只能针对非超级用户且非复制角色。普通角色只能更改自己的密码。

第二种形式更改角色的名称。数据库超级用户可以重命名任何角色。拥有`CREATEROLE`权限的角色可以重命名非超级用户角色。当前会话用户不能被重命名。（如果需要这样做，请以其他用户连接。）由于`MD5`加密的密码使用角色名作为密码学盐，重命名角色会清除其密码（如果密码使用`MD5`加密）。

其余形式更改角色针对配置变量的会话默认值，可以针对所有数据库，也可以在指定`IN DATABASE`子句时仅针对指定数据库中的会话。当该角色随后启动新会话时，指定值将成为会话默认值，覆盖`postgresql.conf`中的设置或从`postgres`命令行接收的设置。这只发生在登录时；执行SET ROLE或SET SESSION AUTHORIZATION不会导致设置新的配置值。为所有数据库设置的设置会被附加到角色的数据库特定设置覆盖。超级用户可以更改任何人的会话默认值。拥有`CREATEROLE`权限的角色可以更改非超级用户角色的默认值。普通角色只能为自己设置默认值。某些配置变量不能以这种方式设置，或者只能由超级用户执行命令来设置。

参数

name

将要修改其属性的角色名称。

SUPERUSER
NOSUPERUSER
CREATEDB
NOCREATEDB
CREATEROLE
NOCREATEROLE
CREATEUSER
NOCREATEUSER
INHERIT
NOINHERIT
LOGIN
NOLOGIN
REPLICATION
NOREPLICATION
CONNECTION LIMIT *connlimit*
PASSWORD *password*
ENCRYPTED
UNENCRYPTED
VALID UNTIL '*timestamp*'

这些子句更改最初由CREATE ROLE设置的属性。更多信息请参见CREATE ROLE参考页。

new_name

角色的新名称。

database_name

应在其中设置该配置变量的数据库名称。

configuration_parameter
value

将该角色在指定配置参数上的会话默认值设为给定值。如果 *value*是DEFAULT，或者等效地使用了RESET，则会移除角色特定变量设置，
因此该角色在新会话中将继承系统范围的默认设置。使用 RESET ALL可清除所有角色特定设置。SET FROM CURRENT会将该参数在当前会话中的值保存为角色特定值。如果指定了IN DATABASE，则只会为给定角色和数据库设置或移除该配置参数。

角色特定的变量设置只在登录时生效；SET ROLE和SET SESSION AUTHORIZATION不会处理角色特定的变量设置。

关于允许的参数名称和值详见SET和第 18 章。

注解

使用CREATE ROLE添加新角色，使用DROP ROLE移除角色。

ALTER ROLE不能更改角色的成员资格。请使用GRANT和REVOKE完成此操作。

用这个命令指定未加密密码时必须加以注意。密码将以明文形式传输到服务器，
并且也可能被记录在客户端命令历史或服务器日志中。psql包含一个命令\password，
可用于更改角色的密码而不暴露明文密码。

也可以把会话默认值绑定到特定数据库，而不是角色；详见ALTER DATABASE。如果发生冲突，同时针对数据库和角色的设置会覆盖角色特定设置，
而角色特定设置又会覆盖数据库特定设置。

示例

更改角色的密码：

```
ALTER ROLE davide WITH PASSWORD 'hu8jmn3';
```

移除角色的密码：

```
ALTER ROLE davide WITH PASSWORD NULL;
```

更改密码的失效日期，并指定该密码应在比 UTC快 1 小时的时区中，于 2015 年 5 月 4 日中午失效：

```
ALTER ROLE chris VALID UNTIL 'May 4 12:00:00 2015 +1';
```

使密码永久有效：

```
ALTER ROLE fred VALID UNTIL 'infinity';
```

授予角色创建其他角色和新数据库的能力：

```
ALTER ROLE miriam CREATEROLE CREATEDB;
```

为角色指定maintenance_work_mem参数的非默认设置：

```
ALTER ROLE worker_bee SET maintenance_work_mem = 100000;
```

为角色指定client_min_messages参数的数据库特定的非默认设置：

```
ALTER ROLE fred IN DATABASE devel SET client_min_messages = DEBUG;
```

兼容性

`ALTER ROLE`语句是 PostgreSQL的扩展。

另见

CREATE ROLE, DROP ROLE, SET

ALTER SCHEMA

ALTER SCHEMA — 更改一个模式的定义

大纲

```
ALTER SCHEMA name RENAME TO new_name
ALTER SCHEMA name OWNER TO new_owner
```

描述

`ALTER SCHEMA`更改一个模式的定义。

要使用`ALTER SCHEMA`，必须拥有该模式。要重命名模式，还必须在数据库上拥有`CREATE`权限。要更改所有者，还必须是新所有者角色的直接或间接成员，并且必须在数据库上拥有`CREATE`权限。（请注意，超级用户会自动拥有所有这些权限。）

参数

name

现有模式的名称。

new_name

该模式的新名称。新名称不能以`pg_`开头，因为这类名称是为系统模式保留的。

new_owner

该模式的新拥有者。

兼容性

SQL 标准中没有`ALTER SCHEMA`语句。

另见

CREATE SCHEMA, DROP SCHEMA

ALTER SEQUENCE

ALTER SEQUENCE — 更改序列发生器的定义

大纲

```
ALTER SEQUENCE [ IF EXISTS ] name [ INCREMENT [ BY ] increment ]
    [ MINVALUE minvalue | NO MINVALUE ] [ MAXVALUE maxvalue | NO
 MAXVALUE ]
    [ START [ WITH ] start ]
    [ RESTART [ [ WITH ] restart ] ]
    [ CACHE cache ] [ [ NO ] CYCLE ]
    [ OWNED BY { table_name.column_name | NONE } ]
ALTER SEQUENCE [ IF EXISTS ] name OWNER TO new_owner
ALTER SEQUENCE [ IF EXISTS ] name RENAME TO new_name
ALTER SEQUENCE [ IF EXISTS ] name SET SCHEMA new_schema
```

描述

`ALTER SEQUENCE`更改现有序列发生器的参数。在`ALTER SEQUENCE`命令中未明确设置的参数，会保留其原有设置。

要使用`ALTER SEQUENCE`，必须拥有该序列。要更改序列的模式，还必须在新模式上拥有`CREATE`权限。要更改所有者，还必须是新所有者角色的直接或间接成员，
并且该角色必须在序列所在模式上拥有`CREATE`权限。
（这些限制确保更改所有者不会执行任何无法通过删除并重新创建序列完成的操作。不过，超级用户无论如何都可以更改任何序列的所有权。）

参数

name

要更改的序列名称（可以是模式限定的）。

`IF EXISTS`

如果该序列不存在，则不要抛出错误。这种情况下会发出一条提示。

increment

子句`INCREMENT BY` *increment*是可选的。正值会创建升序序列，负值会创建降序序列。如果未指定，则保留原有的增量值。

minvalue
`NO MINVALUE`

可选子句`MINVALUE` *minvalue*确定序列可生成的最小值。如果指定了`NO MINVALUE`，则升序序列和降序序列将分别使用默认值 1 和 $-2^{63}-1$。如果两种选项都未指定，则保留当前最小值。

maxvalue
`NO MAXVALUE`

可选子句`MAXVALUE` *maxvalue*确定序列的最大值。如果指定了`NO MAXVALUE`，则升序序列和降序序列将分别使用默认值 $2^{63}-1$ 和 -1。如果两种选项都未指定，则保留当前最大值。

start

可选子句START WITH *start*更改序列记录的起始值。这不会影响当前序列值；它只是设置未来ALTER SEQUENCE RESTART命令将使用的值。

restart

可选子句RESTART [WITH *restart*]更改序列的当前值。这等价于以is_called = false调用setval函数：指定的值会由对nextval的下一次调用返回。写成不带*restart*值的RESTART，等价于提供由CREATE SEQUENCE记录、或最近由ALTER SEQUENCE START WITH设置的起始值。

cache

CACHE *cache*子句启用预分配序列号并将其存储在内存中，以便更快访问。最小值为1（一次只能生成一个值，即不缓存）。如果未指定，则保留旧的缓存值。

CYCLE

可选的CYCLE关键字可用于启用序列回绕：当递增或递减序列分别达到*maxvalue*或*minvalue*时回绕。如果达到限制，下一个生成的数将分别为*minvalue*或*maxvalue*。

NO CYCLE

如果指定了可选关键字NO CYCLE，则当序列达到其最大值后，任何对 nextval的调用都会返回错误。如果既未指定CYCLE也未指定 NO CYCLE，则保留原有的循环行为。

OWNED BY *table_name.column_name*
OWNED BY NONE

OWNED BY选项使序列与特定表列关联；如果该列（或其整个表）被删除，序列也会自动删除。如果指定此选项，该关联会替换序列此前指定的任何关联。
指定的表必须与序列拥有相同的所有者，并且位于同一模式中。指定OWNED BY NONE会移除现有的关联，使序列成为“独立的”对象。

new_owner

序列的新所有者的用户名。

new_name

该序列的新拥有者的用户名。

new_schema

该序列的新名称。

注解

为了避免阻塞从同一序列获取数字的并发事务，ALTER SEQUENCE对序列生成参数的更改永远不会被回滚；这些更改会立即生效且不可逆转。不过，OWNED BY、OWNER TO、RENAME TO和SET SCHEMA子句引起的是普通的目录更新，可以回滚。

ALTER SEQUENCE不会立即影响除当前后端之外、已经预分配（缓存）了序列值的其他后端中的 nextval结果。它们会先用完所有缓存值，然后才会注意到序列生成参数已经改变。当前后端则会立即受到影响。

ALTER SEQUENCE不会影响该序列的currval状态。（在PostgreSQL 8.3 之前，它有时会这样做。）

由于历史原因，序列也可以使用`ALTER TABLE`；但允许用于序列的 `ALTER TABLE`变体仅限于与上述形式等价的那些。

示例

将一个名为`serial`的序列重启到 105：

```
ALTER SEQUENCE serial RESTART WITH 105;
```

兼容性

`ALTER SEQUENCE`符合SQL标准，但 `START WITH`、`OWNED BY`、`OWNER TO`、`RENAME TO`和 `SET SCHEMA`子句是PostgreSQL扩展。

另见

CREATE SEQUENCE, DROP SEQUENCE

ALTER SERVER

ALTER SERVER — 更改外部服务器的定义

大纲

```
ALTER SERVER name [ VERSION 'new_version' ]
    [ OPTIONS ( [ ADD | SET | DROP ] option ['value'] [, ... ] ) ]
ALTER SERVER name OWNER TO new_owner
ALTER SERVER name RENAME TO new_name
```

描述

`ALTER SERVER`更改外部服务器的定义。第一种形式更改服务器的版本字符串或其通用选项（至少需要一个子句）。第二种形式更改服务器的拥有者。

要更改服务器，必须是服务器的所有者。此外，要更改所有者，还必须拥有该服务器，是新所有者角色的直接或间接成员，并且必须在服务器的外部数据包装器上拥有`USAGE`权限。（请注意，超级用户会自动满足所有这些条件。）

参数

name

一个现有服务器的名称。

new_version

新服务器版本。

`OPTIONS ( [ ADD | SET | DROP ] option ['value'] [, ... ] )`

更改服务器的选项。`ADD`、`SET`和 `DROP`指定要执行的动作。如果未显式指定操作，则假定为`ADD`。选项名称必须唯一；
名称和值也会使用该服务器的外部数据包装器库进行验证。

new_owner

该外部服务器的新拥有者的用户名。

new_name

该外部服务器的新名称。

示例

更改服务器`foo`，添加连接选项：

```
ALTER SERVER foo OPTIONS (host 'foo', dbname 'foodb');
```

更改服务器`foo`，更改版本并修改 `host`选项：

```
ALTER SERVER foo VERSION '8.4' OPTIONS (SET host 'baz');
```

兼容性

`ALTER SERVER`符合 ISO/IEC 9075-9 (SQL/MED)。`OWNER TO`和`RENAME`形式是 PostgreSQL 扩展。

另见

CREATE SERVER, DROP SERVER

ALTER TABLE

ALTER TABLE — change the definition of a table

大纲

```
ALTER TABLE [ IF EXISTS ] [ ONLY ] name [ * ]
    action [, ... ]
ALTER TABLE [ IF EXISTS ] [ ONLY ] name [ * ]
    RENAME [ COLUMN ] column_name TO new_column_name
ALTER TABLE [ IF EXISTS ] [ ONLY ] name [ * ]
    RENAME CONSTRAINT constraint_name TO new_constraint_name
ALTER TABLE [ IF EXISTS ] name
    RENAME TO new_name
ALTER TABLE [ IF EXISTS ] name
    SET SCHEMA new_schema

where action is one of:

    ADD [ COLUMN ] column_name data_type [ COLLATE collation ] [
 column_constraint [ ... ] ]
    DROP [ COLUMN ] [ IF EXISTS ] column_name [ RESTRICT |
 CASCADE ]
    ALTER [ COLUMN ] column_name [ SET DATA ] TYPE data_type [
 COLLATE collation ] [ USING expression ]
    ALTER [ COLUMN ] column_name SET DEFAULT expression
    ALTER [ COLUMN ] column_name DROP DEFAULT
    ALTER [ COLUMN ] column_name { SET | DROP } NOT NULL
    ALTER [ COLUMN ] column_name SET STATISTICS integer
    ALTER [ COLUMN ] column_name SET ( attribute_option = value [,
 ... ] )
    ALTER [ COLUMN ] column_name RESET ( attribute_option [, ... ]
 )
    ALTER [ COLUMN ] column_name SET STORAGE { PLAIN | EXTERNAL |
 EXTENDED | MAIN }
    ADD table_constraint [ NOT VALID ]
    ADD table_constraint_using_index
    VALIDATE CONSTRAINT constraint_name
    DROP CONSTRAINT [ IF EXISTS ]  constraint_name [ RESTRICT |
 CASCADE ]
    DISABLE TRIGGER [ trigger_name | ALL | USER ]
    ENABLE TRIGGER [ trigger_name | ALL | USER ]
    ENABLE REPLICA TRIGGER trigger_name
    ENABLE ALWAYS TRIGGER trigger_name
    DISABLE RULE rewrite_rule_name
    ENABLE RULE rewrite_rule_name
    ENABLE REPLICA RULE rewrite_rule_name
    ENABLE ALWAYS RULE rewrite_rule_name
    CLUSTER ON index_name
    SET WITHOUT CLUSTER
    SET WITH OIDS
    SET WITHOUT OIDS
    SET ( storage_parameter = value [, ... ] )
    RESET ( storage_parameter [, ... ] )
    INHERIT parent_table
    NO INHERIT parent_table
```

```
    OF type_name
    NOT OF
    OWNER TO new_owner
    SET TABLESPACE new_tablespace
```

and *table_constraint_using_index* is:

```
    [ CONSTRAINT constraint_name ]
    { UNIQUE | PRIMARY KEY } USING INDEX index_name
    [ DEFERRABLE | NOT DEFERRABLE ] [ INITIALLY DEFERRED |
 INITIALLY IMMEDIATE ]
```

Description

`ALTER TABLE` changes the definition of an existing table. There are several subforms:

`ADD COLUMN`

This form adds a new column to the table, using the same syntax as CREATE TABLE.

`DROP COLUMN [ IF EXISTS ]`

This form drops a column from a table. Indexes and table constraints involving the column will be automatically dropped as well. You will need to say `CASCADE` if anything outside the table depends on the column, for example, foreign key references or views. If `IF EXISTS` is specified and the column does not exist, no error is thrown. In this case a notice is issued instead.

`IF EXISTS`

Do not throw an error if the table does not exist. A notice is issued in this case.

`SET DATA TYPE`

This form changes the type of a column of a table. Indexes and simple table constraints involving the column will be automatically converted to use the new column type by reparsing the originally supplied expression. The optional `COLLATE` clause specifies a collation for the new column; if omitted, the collation is the default for the new column type. The optional `USING` clause specifies how to compute the new column value from the old; if omitted, the default conversion is the same as an assignment cast from old data type to new. A `USING` clause must be provided if there is no implicit or assignment cast from old to new type.

`SET/DROP DEFAULT`

These forms set or remove the default value for a column. The default values only apply to subsequent `INSERT` commands; they do not cause rows already in the table to change. Defaults can also be created for views, in which case they are inserted into `INSERT` statements on the view before the view's `ON INSERT` rule is applied.

`SET/DROP NOT NULL`

These forms change whether a column is marked to allow null values or to reject null values. You can only use `SET NOT NULL` when the column contains no null values.

`SET STATISTICS`

This form sets the per-column statistics-gathering target for subsequent ANALYZE operations. The target can be set in the range 0 to 10000; alternatively, set it to -1

to revert to using the system default statistics target (default_statistics_target). For more information on the use of statistics by the PostgreSQL query planner, refer to 第 14.2 节.

`SET ( attribute_option = value [, ... ] )`
`RESET ( attribute_option [, ... ] )`

This form sets or resets per-attribute options. Currently, the only defined per-attribute options are `n_distinct` and `n_distinct_inherited`, which override the number-of-distinct-values estimates made by subsequent ANALYZE operations. `n_distinct` affects the statistics for the table itself, while `n_distinct_inherited` affects the statistics gathered for the table plus its inheritance children. When set to a positive value, `ANALYZE` will assume that the column contains exactly the specified number of distinct nonnull values. When set to a negative value, which must be greater than or equal to -1, `ANALYZE` will assume that the number of distinct nonnull values in the column is linear in the size of the table; the exact count is to be computed by multiplying the estimated table size by the absolute value of the given number. For example, a value of -1 implies that all values in the column are distinct, while a value of -0.5 implies that each value appears twice on the average. This can be useful when the size of the table changes over time, since the multiplication by the number of rows in the table is not performed until query planning time. Specify a value of 0 to revert to estimating the number of distinct values normally. For more information on the use of statistics by the PostgreSQL query planner, refer to 第 14.2 节.

`SET STORAGE`

This form sets the storage mode for a column. This controls whether this column is held inline or in a secondary TOAST table, and whether the data should be compressed or not. `PLAIN` must be used for fixed-length values such as `integer` and is inline, uncompressed. `MAIN` is for inline, compressible data. `EXTERNAL` is for external, uncompressed data, and `EXTENDED` is for external, compressed data. `EXTENDED` is the default for most data types that support non-`PLAIN` storage. Use of `EXTERNAL` will make substring operations on very large `text` and `bytea` values run faster, at the penalty of increased storage space. Note that `SET STORAGE` doesn't itself change anything in the table, it just sets the strategy to be pursued during future table updates. See 第 56.2 节 for more information.

`ADD table_constraint [ NOT VALID ]`

This form adds a new constraint to a table using the same syntax as CREATE TABLE, plus the option `NOT VALID`, which is currently only allowed for foreign key and CHECK constraints. If the constraint is marked `NOT VALID`, the potentially-lengthy initial check to verify that all rows in the table satisfy the constraint is skipped. The constraint will still be enforced against subsequent inserts or updates (that is, they'll fail unless there is a matching row in the referenced table, in the case of foreign keys; and they'll fail unless the new row matches the specified check constraints). But the database will not assume that the constraint holds for all rows in the table, until it is validated by using the `VALIDATE CONSTRAINT` option.

`ADD table_constraint_using_index`

This form adds a new `PRIMARY KEY` or `UNIQUE` constraint to a table based on an existing unique index. All the columns of the index will be included in the constraint.

The index cannot have expression columns nor be a partial index. Also, it must be a b-tree index with default sort ordering. These restrictions ensure that the index

is equivalent to one that would be built by a regular `ADD PRIMARY KEY` or `ADD UNIQUE` command.

If `PRIMARY KEY` is specified, and the index's columns are not already marked `NOT NULL`, then this command will attempt to do `ALTER COLUMN SET NOT NULL` against each such column. That requires a full table scan to verify the column(s) contain no nulls. In all other cases, this is a fast operation.

If a constraint name is provided then the index will be renamed to match the constraint name. Otherwise the constraint will be named the same as the index.

After this command is executed, the index is "owned" by the constraint, in the same way as if the index had been built by a regular `ADD PRIMARY KEY` or `ADD UNIQUE` command. In particular, dropping the constraint will make the index disappear too.

> **注意**
>
> Adding a constraint using an existing index can be helpful in situations where a new constraint needs to be added without blocking table updates for a long time. To do that, create the index using `CREATE INDEX CONCURRENTLY`, and then install it as an official constraint using this syntax. See the example below.

`VALIDATE CONSTRAINT`

This form validates a foreign key or check constraint that was previously created as `NOT VALID`, by scanning the table to ensure there are no rows for which the constraint is not satisfied. Nothing happens if the constraint is already marked valid.

Validation can be a long process on larger tables and currently requires an `ACCESS EXCLUSIVE` lock. The value of separating validation from initial creation is that you can defer validation to less busy times, or can be used to give additional time to correct pre-existing errors while preventing new errors.

`DROP CONSTRAINT [ IF EXISTS ]`

This form drops the specified constraint on a table. If `IF EXISTS` is specified and the constraint does not exist, no error is thrown. In this case a notice is issued instead.

`DISABLE/ENABLE [ REPLICA | ALWAYS ] TRIGGER`

These forms configure the firing of trigger(s) belonging to the table. A disabled trigger is still known to the system, but is not executed when its triggering event occurs. For a deferred trigger, the enable status is checked when the event occurs, not when the trigger function is actually executed. One can disable or enable a single trigger specified by name, or all triggers on the table, or only user triggers (this option excludes internally generated constraint triggers such as those that are used to implement foreign key constraints or deferrable uniqueness and exclusion constraints). Disabling or enabling internally generated constraint triggers requires superuser privileges; it should be done with caution since of course the integrity of the constraint cannot be guaranteed if the triggers are not executed. The trigger firing mechanism is also affected by the configuration variable session_replication_role. Simply enabled triggers will fire when the replication role is "origin" (the default) or "local". Triggers configured as `ENABLE`

`REPLICA` will only fire if the session is in "replica" mode, and triggers configured as `ENABLE ALWAYS` will fire regardless of the current replication mode.

`DISABLE/ENABLE [ REPLICA | ALWAYS ] RULE`

These forms configure the firing of rewrite rules belonging to the table. A disabled rule is still known to the system, but is not applied during query rewriting. The semantics are as for disabled/enabled triggers. This configuration is ignored for `ON SELECT` rules, which are always applied in order to keep views working even if the current session is in a non-default replication role.

`CLUSTER ON`

This form selects the default index for future CLUSTER operations. It does not actually re-cluster the table.

`SET WITHOUT CLUSTER`

This form removes the most recently used CLUSTER index specification from the table. This affects future cluster operations that don't specify an index.

`SET WITH OIDS`

This form adds an `oid` system column to the table (see 第 5.4 节). It does nothing if the table already has OIDs.

Note that this is not equivalent to `ADD COLUMN oid oid`; that would add a normal column that happened to be named `oid`, not a system column.

`SET WITHOUT OIDS`

This form removes the `oid` system column from the table. This is exactly equivalent to `DROP COLUMN oid RESTRICT`, except that it will not complain if there is already no `oid` column.

`SET ( storage_parameter = value [, ... ] )`

This form changes one or more storage parameters for the table. See Storage Parameters for details on the available parameters. Note that the table contents will not be modified immediately by this command; depending on the parameter you might need to rewrite the table to get the desired effects. That can be done with VACUUM FULL, CLUSTER or one of the forms of `ALTER TABLE` that forces a table rewrite.

> **注意**
>
> While `CREATE TABLE` allows `OIDS` to be specified in the `WITH (storage_parameter)` syntax, `ALTER TABLE` does not treat `OIDS` as a storage parameter. Instead use the `SET WITH OIDS` and `SET WITHOUT OIDS` forms to change OID status.

`RESET ( storage_parameter [, ... ] )`

This form resets one or more storage parameters to their defaults. As with `SET`, a table rewrite might be needed to update the table entirely.

`INHERIT parent_table`

This form adds the target table as a new child of the specified parent table. Subsequently, queries against the parent will include records of the target table.

To be added as a child, the target table must already contain all the same columns as the parent (it could have additional columns, too). The columns must have matching data types, and if they have `NOT NULL` constraints in the parent then they must also have `NOT NULL` constraints in the child.

There must also be matching child-table constraints for all `CHECK` constraints of the parent, except those marked non-inheritable (that is, created with `ALTER TABLE ... ADD CONSTRAINT ... NO INHERIT`) in the parent, which are ignored; all child-table constraints matched must not be marked non-inheritable. Currently `UNIQUE`, `PRIMARY KEY`, and `FOREIGN KEY` constraints are not considered, but this might change in the future.

`NO INHERIT` *`parent_table`*

This form removes the target table from the list of children of the specified parent table. Queries against the parent table will no longer include records drawn from the target table.

`OF` *`type_name`*

This form links the table to a composite type as though `CREATE TABLE OF` had formed it. The table's list of column names and types must precisely match that of the composite type; the presence of an `oid` system column is permitted to differ. The table must not inherit from any other table. These restrictions ensure that `CREATE TABLE OF` would permit an equivalent table definition.

`NOT OF`

This form dissociates a typed table from its type.

`OWNER`

This form changes the owner of the table, sequence, or view to the specified user.

`SET TABLESPACE`

This form changes the table's tablespace to the specified tablespace and moves the data file(s) associated with the table to the new tablespace. Indexes on the table, if any, are not moved; but they can be moved separately with additional `SET TABLESPACE` commands. See also CREATE TABLESPACE.

`RENAME`

The `RENAME` forms change the name of a table (or an index, sequence, or view), the name of an individual column in a table, or the name of a constraint of the table. There is no effect on the stored data.

`SET SCHEMA`

This form moves the table into another schema. Associated indexes, constraints, and sequences owned by table columns are moved as well.

All the actions except `RENAME` and `SET SCHEMA` can be combined into a list of multiple alterations to apply in parallel. For example, it is possible to add several columns and/or alter the type of several columns in a single command. This is particularly useful with large tables, since only one pass over the table need be made.

You must own the table to use `ALTER TABLE`. To change the schema of a table, you must also have `CREATE` privilege on the new schema. To add the table as a new child of a parent table, you must own the parent table as well. To alter the owner, you must also

be a direct or indirect member of the new owning role, and that role must have `CREATE` privilege on the table's schema. (These restrictions enforce that altering the owner doesn't do anything you couldn't do by dropping and recreating the table. However, a superuser can alter ownership of any table anyway.) To add a column or alter a column type or use the `OF` clause, you must also have `USAGE` privilege on the data type.

Parameters

`name`

The name (optionally schema-qualified) of an existing table to alter. If `ONLY` is specified before the table name, only that table is altered. If `ONLY` is not specified, the table and all its descendant tables (if any) are altered. Optionally, `*` can be specified after the table name to explicitly indicate that descendant tables are included.

`column_name`

Name of a new or existing column.

`new_column_name`

New name for an existing column.

`new_name`

New name for the table.

`type`

Data type of the new column, or new data type for an existing column.

`table_constraint`

New table constraint for the table.

`constraint_name`

Name of an existing constraint to drop.

`CASCADE`

Automatically drop objects that depend on the dropped column or constraint (for example, views referencing the column).

`RESTRICT`

Refuse to drop the column or constraint if there are any dependent objects. This is the default behavior.

`trigger_name`

Name of a single trigger to disable or enable.

`ALL`

Disable or enable all triggers belonging to the table. (This requires superuser privilege if any of the triggers are internally generated constraint triggers such as those that are used to implement foreign key constraints or deferrable uniqueness and exclusion constraints.)

`USER`

Disable or enable all triggers belonging to the table except for internally generated constraint triggers such as those that are used to implement foreign key constraints or deferrable uniqueness and exclusion constraints.

`index_name`

The index name on which the table should be marked for clustering.

`storage_parameter`

The name of a table storage parameter.

`value`

The new value for a table storage parameter. This might be a number or a word depending on the parameter.

`parent_table`

A parent table to associate or de-associate with this table.

`new_owner`

The user name of the new owner of the table.

`new_tablespace`

The name of the tablespace to which the table will be moved.

`new_schema`

The name of the schema to which the table will be moved.

Notes

The key word `COLUMN` is noise and can be omitted.

When a column is added with `ADD COLUMN`, all existing rows in the table are initialized with the column's default value (NULL if no `DEFAULT` clause is specified).

Adding a column with a non-null default or changing the type of an existing column will require the entire table and indexes to be rewritten. As an exception, if the `USING` clause does not change the column contents and the old type is either binary coercible to the new type or an unconstrained domain over the new type, a table rewrite is not needed, but any indexes on the affected columns must still be rebuilt. Adding or removing a system `oid` column also requires rewriting the entire table. Table and/or index rebuilds may take a significant amount of time for a large table; and will temporarily require as much as double the disk space.

Adding a `CHECK` or `NOT NULL` constraint requires scanning the table to verify that existing rows meet the constraint, but does not require a table rewrite.

The main reason for providing the option to specify multiple changes in a single `ALTER TABLE` is that multiple table scans or rewrites can thereby be combined into a single pass over the table.

The `DROP COLUMN` form does not physically remove the column, but simply makes it invisible to SQL operations. Subsequent insert and update operations in the table will store a null value for the column. Thus, dropping a column is quick but it will

not immediately reduce the on-disk size of your table, as the space occupied by the dropped column is not reclaimed. The space will be reclaimed over time as existing rows are updated. (These statements do not apply when dropping the system `oid` column; that is done with an immediate rewrite.)

To force immediate reclamation of space occupied by a dropped column, you can execute one of the forms of `ALTER TABLE` that performs a rewrite of the whole table. This results in reconstructing each row with the dropped column replaced by a null value.

The rewriting forms of `ALTER TABLE` are not MVCC-safe. After a table rewrite, the table will appear empty to concurrent transactions, if they are using a snapshot taken before the rewrite occurred. See 第 13.5 节 for more details.

The `USING` option of `SET DATA TYPE` can actually specify any expression involving the old values of the row; that is, it can refer to other columns as well as the one being converted. This allows very general conversions to be done with the `SET DATA TYPE` syntax. Because of this flexibility, the `USING` expression is not applied to the column's default value (if any); the result might not be a constant expression as required for a default. This means that when there is no implicit or assignment cast from old to new type, `SET DATA TYPE` might fail to convert the default even though a `USING` clause is supplied. In such cases, drop the default with `DROP DEFAULT`, perform the `ALTER TYPE`, and then use `SET DEFAULT` to add a suitable new default. Similar considerations apply to indexes and constraints involving the column.

If a table has any descendant tables, it is not permitted to add, rename, or change the type of a column, or rename an inherited constraint in the parent table without doing the same to the descendants. That is, `ALTER TABLE ONLY` will be rejected. This ensures that the descendants always have columns matching the parent.

A recursive `DROP COLUMN` operation will remove a descendant table's column only if the descendant does not inherit that column from any other parents and never had an independent definition of the column. A nonrecursive `DROP COLUMN` (i.e., `ALTER TABLE ONLY ... DROP COLUMN`) never removes any descendant columns, but instead marks them as independently defined rather than inherited.

The `TRIGGER`, `CLUSTER`, `OWNER`, and `TABLESPACE` actions never recurse to descendant tables; that is, they always act as though `ONLY` were specified. Adding a constraint recurses only for `CHECK` constraints that are not marked `NO INHERIT`.

Changing any part of a system catalog table is not permitted.

Refer to CREATE TABLE for a further description of valid parameters. 第 5 章 has further information on inheritance.

Examples

To add a column of type `varchar` to a table:

```
ALTER TABLE distributors ADD COLUMN address varchar(30);
```

To drop a column from a table:

```
ALTER TABLE distributors DROP COLUMN address RESTRICT;
```

To change the types of two existing columns in one operation:

```
ALTER TABLE distributors
    ALTER COLUMN address TYPE varchar(80),
```

```
    ALTER COLUMN name TYPE varchar(100);
```

To change an integer column containing UNIX timestamps to `timestamp with time zone` via a `USING` clause:

```
ALTER TABLE foo
    ALTER COLUMN foo_timestamp SET DATA TYPE timestamp with time
 zone
    USING
        timestamp with time zone 'epoch' + foo_timestamp * interval
 '1 second';
```

The same, when the column has a default expression that won't automatically cast to the new data type:

```
ALTER TABLE foo
    ALTER COLUMN foo_timestamp DROP DEFAULT,
    ALTER COLUMN foo_timestamp TYPE timestamp with time zone
    USING
        timestamp with time zone 'epoch' + foo_timestamp * interval
 '1 second',
    ALTER COLUMN foo_timestamp SET DEFAULT now();
```

To rename an existing column:

```
ALTER TABLE distributors RENAME COLUMN address TO city;
```

To rename an existing table:

```
ALTER TABLE distributors RENAME TO suppliers;
```

To rename an existing constraint:

```
ALTER TABLE distributors RENAME CONSTRAINT zipchk TO zip_check;
```

To add a not-null constraint to a column:

```
ALTER TABLE distributors ALTER COLUMN street SET NOT NULL;
```

To remove a not-null constraint from a column:

```
ALTER TABLE distributors ALTER COLUMN street DROP NOT NULL;
```

To add a check constraint to a table and all its children:

```
ALTER TABLE distributors ADD CONSTRAINT zipchk CHECK (char_length(
zipcode) = 5);
```

To add a check constraint only to a table and not to its children:

```
ALTER TABLE distributors ADD CONSTRAINT zipchk CHECK (char_length(
zipcode) = 5) NO INHERIT;
```

(The check constraint will not be inherited by future children, either.)

To remove a check constraint from a table and all its children:

```
ALTER TABLE distributors DROP CONSTRAINT zipchk;
```

To remove a check constraint from one table only:

```
ALTER TABLE ONLY distributors DROP CONSTRAINT zipchk;
```

(The check constraint remains in place for any child tables.)

To add a foreign key constraint to a table:

```
ALTER TABLE distributors ADD CONSTRAINT distfk FOREIGN KEY (
address) REFERENCES addresses (address);
```

To add a (multicolumn) unique constraint to a table:

```
ALTER TABLE distributors ADD CONSTRAINT dist_id_zipcode_key UNIQUE
 (dist_id, zipcode);
```

To add an automatically named primary key constraint to a table, noting that a table can only ever have one primary key:

```
ALTER TABLE distributors ADD PRIMARY KEY (dist_id);
```

To move a table to a different tablespace:

```
ALTER TABLE distributors SET TABLESPACE fasttablespace;
```

To move a table to a different schema:

```
ALTER TABLE myschema.distributors SET SCHEMA yourschema;
```

To recreate a primary key constraint, without blocking updates while the index is rebuilt:

```
CREATE UNIQUE INDEX CONCURRENTLY dist_id_temp_idx ON distributors (
dist_id);
ALTER TABLE distributors DROP CONSTRAINT distributors_pkey,
    ADD CONSTRAINT distributors_pkey PRIMARY KEY USING INDEX dist_
id_temp_idx;
```

Compatibility

The forms `ADD` (without `USING INDEX`), `DROP`, `SET DEFAULT`, and `SET DATA TYPE` (without `USING`) conform with the SQL standard. The other forms are PostgreSQL extensions of the SQL standard. Also, the ability to specify more than one manipulation in a single `ALTER TABLE` command is an extension.

`ALTER TABLE DROP COLUMN` can be used to drop the only column of a table, leaving a zero-column table. This is an extension of SQL, which disallows zero-column tables.

See Also

CREATE TABLE

ALTER TABLESPACE

ALTER TABLESPACE — 更改一个表空间的定义

大纲

```
ALTER TABLESPACE name RENAME TO new_name
ALTER TABLESPACE name OWNER TO new_owner
ALTER TABLESPACE name SET ( tablespace_option = value [, ... ] )
ALTER TABLESPACE name RESET ( tablespace_option [, ... ] )
```

描述

`ALTER TABLESPACE`更改表空间的定义。

必须拥有该表空间才能使用`ALTER TABLESPACE`。要更改所有者，还必须是新所有者角色的直接或间接成员。（请注意，超级用户会自动拥有这些权限。）

参数

name

现有表空间的名称。

new_name

该表空间的新名称。新名称不能以`pg_`开头，因为这类名称保留给系统表空间使用。

new_owner

该表空间的新拥有者。

tablespace_parameter

要设置或重置的表空间参数。目前可用的参数只有`seq_page_cost`和`random_page_cost`。为特定表空间设置其中任一值，
会覆盖规划器对读取该表空间中表的数据页成本的通常估计；
该估计由同名配置参数确定（参见seq_page_cost、random_page_cost）。
如果一个表空间位于比 I/O 子系统其余部分更快或更慢的磁盘上，这可能很有用。

示例

将表空间`index_space`重命名为`fast_raid`：

```
ALTER TABLESPACE index_space RENAME TO fast_raid;
```

更改表空间`index_space`的拥有者：

```
ALTER TABLESPACE index_space OWNER TO mary;
```

兼容性

在 SQL 标准中没有 `ALTER TABLESPACE`语句。

另见

CREATE TABLESPACE, DROP TABLESPACE

ALTER TEXT SEARCH CONFIGURATION

ALTER TEXT SEARCH CONFIGURATION — 更改一个文本搜索配置的定义

大纲

```
ALTER TEXT SEARCH CONFIGURATION name
    ADD MAPPING FOR token_type [, ... ] WITH dictionary_name [, ...
 ]
ALTER TEXT SEARCH CONFIGURATION name
    ALTER MAPPING FOR token_type [, ... ] WITH dictionary_name [, .
.. ]
ALTER TEXT SEARCH CONFIGURATION name
    ALTER MAPPING REPLACE old_dictionary WITH new_dictionary
ALTER TEXT SEARCH CONFIGURATION name
    ALTER MAPPING FOR token_type [, ... ] REPLACE old_dictionary
 WITH new_dictionary
ALTER TEXT SEARCH CONFIGURATION name
    DROP MAPPING [ IF EXISTS ] FOR token_type [, ... ]
ALTER TEXT SEARCH CONFIGURATION name RENAME TO new_name
ALTER TEXT SEARCH CONFIGURATION name OWNER TO new_owner
ALTER TEXT SEARCH CONFIGURATION name SET SCHEMA new_schema
```

描述

`ALTER TEXT SEARCH CONFIGURATION` 更改一个文本搜索配置的定义。你可以修改其从记号类型到字典的映射，或者更改该配置的名称或拥有者。

要使用`ALTER TEXT SEARCH CONFIGURATION`，你必须是该配置的拥有者。

参数

name

一个现有文本搜索配置的名称（可以是模式限定的）。

token_type

由该配置的解析器发出的记号类型的名称。

dictionary_name

用于指定记号类型的文本搜索字典名称。如果列出了多个字典，将按指定顺序依次查阅它们。

old_dictionary

在映射中要替换的文本搜索字典的名称。

new_dictionary

被用来替代*old_dictionary* 的文本搜索字典的名称。

new_name

该文本搜索配置的新名称。

new_owner

该文本搜索配置的新拥有者。

new_schema

该文本搜索配置的新模式。

`ADD MAPPING FOR`形式为指定的词元类型安装要查询的字典列表；如果任何词元类型已经存在映射，则会报错。`ALTER MAPPING FOR`形式执行相同操作，但会先移除这些词元类型的现有映射。`ALTER MAPPING REPLACE`形式在任何出现旧字典的地方用*new_dictionary*替换*old_dictionary*。当出现`FOR`时，只对指定的词元类型执行此操作；不出现时，则对该配置的所有映射执行。`DROP MAPPING`形式移除指定词元类型的所有字典，使这些类型的词元被文本搜索配置忽略。如果词元类型没有映射，则会报错，除非出现`IF EXISTS`。

示例

以下示例在`my_config`中使用`english`的任何地方，都将`english`字典替换为`swedish`字典。

```
ALTER TEXT SEARCH CONFIGURATION my_config
  ALTER MAPPING REPLACE english WITH swedish;
```

兼容性

在 SQL 标准中没有 `ALTER TEXT SEARCH CONFIGURATION` 语句。

另见

CREATE TEXT SEARCH CONFIGURATION, DROP TEXT SEARCH CONFIGURATION

ALTER TEXT SEARCH DICTIONARY

ALTER TEXT SEARCH DICTIONARY — 更改一个文本搜索字典的定义

大纲

```
ALTER TEXT SEARCH DICTIONARY name (
    option [ = value ] [, ... ]
)
ALTER TEXT SEARCH DICTIONARY name RENAME TO new_name
ALTER TEXT SEARCH DICTIONARY name OWNER TO new_owner
ALTER TEXT SEARCH DICTIONARY name SET SCHEMA new_schema
```

描述

`ALTER TEXT SEARCH DICTIONARY`更改文本搜索字典的定义。你可以更改该字典的模板相关选项，也可以更改该字典的名称或拥有者。

要使用`ALTER TEXT SEARCH DICTIONARY`，你必须是该字典的拥有者。

参数

name

一个现有文本搜索字典的名称（可以是模式限定的）。

option

要为此字典设置的模板相关选项的名称。

value

模板相关选项要使用的新值。如果省略等号和值，
则会从该字典中移除该选项之前的设置，从而允许使用默认值。

new_name

该文本搜索字典的新名称。

new_owner

该文本搜索字典的新拥有者。

new_schema

该文本搜索字典的新模式。

模板相关选项可以以任意顺序出现。

示例

下面的示例命令更改了一个基于 Snowball 的字典的停用词列表。其他参数保持不变。

```
ALTER TEXT SEARCH DICTIONARY my_dict ( StopWords = newrussian );
```

下面的示例命令将语言选项更改为`dutch`，并完全移除了停用词选项。

```
ALTER TEXT SEARCH DICTIONARY my_dict ( language = dutch, Stop
Words );
```

下面的示例命令“更新”了该字典的定义，但实际上并没有做任何更改。

```
ALTER TEXT SEARCH DICTIONARY my_dict ( dummy );
```

（之所以可行，是因为选项移除代码在不存在该选项时也不会报错。）
这种技巧在修改该字典的配置文件时很有用：`ALTER` 会强制现有数据库会话重新读取配置文件，而如果它们先前已经读取过这些文件，本来是不会再次读取的。

兼容性

在 SQL 标准中没有 `ALTER TEXT SEARCH DICTIONARY`语句。

另见

CREATE TEXT SEARCH DICTIONARY, DROP TEXT SEARCH DICTIONARY

ALTER TEXT SEARCH PARSER

ALTER TEXT SEARCH PARSER — 更改一个全文检索解析器的定义

大纲

```
ALTER TEXT SEARCH PARSER name RENAME TO new_name
ALTER TEXT SEARCH PARSER name SET SCHEMA new_schema
```

描述

`ALTER TEXT SEARCH PARSER`更改全文检索解析器的定义。当前唯一支持的功能是更改该解析器的名称。

要使用`ALTER TEXT SEARCH PARSER`，你必须是超级用户。

参数

name

一个现有全文检索解析器的名称（可以是模式限定的）。

new_name

该全文检索解析器的新名称。

new_schema

该全文检索解析器的新模式。

兼容性

SQL 标准中没有`ALTER TEXT SEARCH PARSER`语句。

另见

CREATE TEXT SEARCH PARSER, DROP TEXT SEARCH PARSER

ALTER TEXT SEARCH TEMPLATE

ALTER TEXT SEARCH TEMPLATE — 更改一个文本搜索模板的定义

大纲

```
ALTER TEXT SEARCH TEMPLATE name RENAME TO new_name
ALTER TEXT SEARCH TEMPLATE name SET SCHEMA new_schema
```

描述

`ALTER TEXT SEARCH TEMPLATE`更改文本搜索模板的定义。当前唯一支持的功能是更改该模板的名称。

要使用 `ALTER TEXT SEARCH TEMPLATE`，你必须是超级用户。

参数

name

一个现有文本搜索模板的名称（可以是模式限定的）。

new_name

该文本搜索模板的新名称。

new_schema

该文本搜索模板的新模式。

兼容性

SQL 标准中没有 `ALTER TEXT SEARCH TEMPLATE` 语句。

另见

CREATE TEXT SEARCH TEMPLATE, DROP TEXT SEARCH TEMPLATE

ALTER TRIGGER

ALTER TRIGGER — 更改触发器的定义

大纲

```
ALTER TRIGGER name ON table_name RENAME TO new_name
```

描述

`ALTER TRIGGER`更改现有触发器的属性。`RENAME`子句更改给定触发器的名称，但不以其他方式更改触发器定义。

若要更改触发器的属性，你必须拥有该触发器所作用的表。

参数

name

要修改的现有触发器的名称。

table_name

该触发器所作用的表的名称。

new_name

该触发器的新名称。

注解

临时启用或禁用触发器的能力由ALTER TABLE提供，而不是由`ALTER TRIGGER`提供，因为`ALTER TRIGGER`没有便捷的方式来表达一次启用或禁用表中所有触发器的选项。

示例

要重命名一个现有触发器：

```
ALTER TRIGGER emp_stamp ON emp RENAME TO emp_track_chgs;
```

兼容性

`ALTER TRIGGER`是 PostgreSQL 对 SQL 标准的扩展。

参见

ALTER TABLE

ALTER TYPE

ALTER TYPE — 更改类型的定义

大纲

```
ALTER TYPE name action [, ... ]
ALTER TYPE name OWNER TO new_owner
ALTER TYPE name RENAME ATTRIBUTE attribute_name TO new_attribute_
name [ CASCADE | RESTRICT ]
ALTER TYPE name RENAME TO new_name
ALTER TYPE name SET SCHEMA new_schema
ALTER TYPE name ADD VALUE new_enum_value [ { BEFORE | AFTER }
 existing_enum_value ]
```

其中*action*为以下之一：

```
    ADD ATTRIBUTE attribute_name data_type [ COLLATE collation ] [
 CASCADE | RESTRICT ]
    DROP ATTRIBUTE [ IF EXISTS ] attribute_name [ CASCADE |
 RESTRICT ]
    ALTER ATTRIBUTE attribute_name [ SET DATA ] TYPE data_type [
 COLLATE collation ] [ CASCADE | RESTRICT ]
```

描述

`ALTER TYPE`更改现有类型的定义。有几种子形式：

`ADD ATTRIBUTE`

该形式使用与CREATE TYPE相同的语法向复合类型添加新属性。

`DROP ATTRIBUTE [ IF EXISTS ]`

这种形式从复合类型中删除一个属性。如果指定了`IF EXISTS`而该属性不存在，则不会抛出错误，而是发出一条提示。

`SET DATA TYPE`

这种形式更改复合类型中某个属性的数据类型。

`OWNER`

这种形式更改类型的拥有者。

`RENAME`

该形式更改类型的名称或复合类型中单个属性的名称。

`SET SCHEMA`

这种形式将类型移动到另一个模式中。

`ADD VALUE [ BEFORE | AFTER ]`

该形式向枚举类型添加一个新值。如果没有使用 `BEFORE` 或 `AFTER` 指定新值在枚举排序中的位置，新项会被添加到值列表的末尾。

CASCADE

自动将该操作传播到正在修改的类型的类型化表及其后代。

RESTRICT

如果正在修改的类型是某个类型化表的类型，则拒绝该操作。这是默认行为。

ADD ATTRIBUTE、DROP ATTRIBUTE 和 ALTER ATTRIBUTE 操作可以组合成一个包含多项修改的列表，以便并行应用。例如，可以在一条命令中添加多个属性和/或更改多个属性的数据类型。

要使用ALTER TYPE，必须拥有该类型。要更改类型的模式，还必须在新模式上拥有CREATE权限。要更改所有者，还必须是新所有者角色的直接或间接成员，
并且该角色必须在类型所在模式上拥有CREATE权限。
（这些限制确保更改所有者不会执行任何无法通过删除并重新创建类型完成的操作。不过，超级用户无论如何都可以更改任何类型的所有权。）要添加属性或更改属性类型，
还必须在该属性的数据类型上拥有USAGE权限。

参数

name

要修改的现有类型的名称（可以是模式限定的）。

new_name

该类型的新名称。

new_owner

该类型新的拥有者的用户名。

new_schema

该类型的新模式。

attribute_name

要添加、修改或删除的属性名称。

new_attribute_name

要重命名的属性的新名称。

data_type

要添加的属性的数据类型，或者要修改的属性的新类型。

new_enum_value

要添加到枚举类型值列表中的新值。与所有枚举字面量一样，它必须加引号。

existing_enum_value

一个现有枚举值；在枚举类型的排序顺序中，新值会紧邻该值，被添加到它之前或之后。与所有枚举字面量一样，它必须加引号。

注解

ALTER TYPE ... ADD VALUE（向枚举类型添加新值的形式）不能在事务块内执行。

涉及新增枚举值的比较有时会比只涉及枚举类型原有成员的比较慢。通常只有在使用`BEFORE`或`AFTER`将新值的排序位置设置在列表末尾以外时，才会发生这种情况。但是，即使新值添加在末尾，有时也会发生这种情况（如果自枚举类型最初创建以来 OID 计数器发生了“回绕”）。速度下降通常并不明显；但如果这确实重要，可以通过删除并重新创建枚举类型，或转储并重新加载数据库，恢复最佳性能。

示例

要重命名一个数据类型：

```
ALTER TYPE electronic_mail RENAME TO email;
```

要将类型`email`的拥有者改为 `joe`：

```
ALTER TYPE email OWNER TO joe;
```

要将类型`email`所在的模式改为 `customers`：

```
ALTER TYPE email SET SCHEMA customers;
```

向类型添加新属性：

```
ALTER TYPE compfoo ADD ATTRIBUTE f3 int;
```

要在枚举类型的特定排序位置添加一个新值：

```
ALTER TYPE colors ADD VALUE 'orange' AFTER 'red';
```

兼容性

添加和删除属性的这些变体属于 SQL 标准；其他变体则是 PostgreSQL 扩展。

另见

CREATE TYPE, DROP TYPE

ALTER USER

ALTER USER — 更改数据库角色

大纲

```
ALTER USER name [ [ WITH ] option [ ... ]

其中option可以是:

      SUPERUSER | NOSUPERUSER
    | CREATEDB | NOCREATEDB
    | CREATEROLE | NOCREATEROLE
    | CREATEUSER | NOCREATEUSER
    | INHERIT | NOINHERIT
    | LOGIN | NOLOGIN
    | REPLICATION | NOREPLICATION
    | BYPASSRLS | NOBYPASSRLS
    | CONNECTION LIMIT connlimit
    | [ ENCRYPTED | UNENCRYPTED ] PASSWORD 'password'
    | VALID UNTIL 'timestamp'

ALTER USER name RENAME TO new_name

ALTER USER name [ IN DATABASE database_name ] SET configuration_
parameter { TO | = } { value | DEFAULT }
ALTER USER name [ IN DATABASE database_name ] SET configuration_
parameter FROM CURRENT
ALTER USER name [ IN DATABASE database_name ] RESET configuration_
parameter
ALTER USER name [ IN DATABASE database_name ] RESET ALL
```

描述

`ALTER USER`现为ALTER ROLE的别名。

兼容性

`ALTER USER`语句是 PostgreSQL的一种扩展。SQL 标准将用户的定义交由具体实现决定。

另见

ALTER ROLE

ALTER USER MAPPING

ALTER USER MAPPING — 更改用户映射的定义

大纲

```
ALTER USER MAPPING FOR { user_name | USER | CURRENT_USER | PUBLIC }
    SERVER server_name
    OPTIONS ( [ ADD | SET | DROP ] option ['value'] [, ... ] )
```

描述

`ALTER USER MAPPING`更改用户映射的定义。

外部服务器的拥有者可以为该服务器上的任何用户修改用户映射。此外，
如果某个用户已被授予该服务器上的`USAGE`权限，
那么该用户也可以修改其自己用户名对应的用户映射。

参数

user_name

映射的用户名。`CURRENT_USER`和`USER`与当前用户名称匹配。`PUBLIC`
用于匹配系统中当前及未来的所有用户名。

server_name

该用户映射所属服务器的名称。

`OPTIONS ( [ ADD | SET | DROP ] option ['value'] [, ... ] )`

更改该用户映射的选项。新选项会覆盖先前指定的任何选项。`ADD`、`SET`和`DROP` 指定要
执行的动作。如果未显式指定操作，则假定为`ADD`。选项名必须唯一；
这些选项还会由该服务器的外部数据包装器进行验证。

示例

更改用户映射`bob`、服务器`foo`的密码：

```
ALTER USER MAPPING FOR bob SERVER foo OPTIONS (SET password
 'public');
```

兼容性

`ALTER USER MAPPING`符合 ISO/IEC 9075-9 (SQL/MED)。这里有一个细微的语法问题：该标准省略了`FOR`关键字。由于`CREATE USER MAPPING`和`DROP USER MAPPING`都在类似位置使用`FOR`，而 IBM DB2（另一个主要的 SQL/MED 实现）也要求在`ALTER USER MAPPING`中使用它，因此为了保持一致性和互操作性，PostgreSQL 在这里偏离了标准。

另见

CREATE USER MAPPING, DROP USER MAPPING

ALTER VIEW

ALTER VIEW — 更改视图的定义

大纲

```
ALTER VIEW [ IF EXISTS ] name ALTER [ COLUMN ] column_name SET
 DEFAULT expression
ALTER VIEW [ IF EXISTS ] name ALTER [ COLUMN ] column_name DROP
 DEFAULT
ALTER VIEW [ IF EXISTS ] name OWNER TO new_owner
ALTER VIEW [ IF EXISTS ] name RENAME TO new_name
ALTER VIEW [ IF EXISTS ] name SET SCHEMA new_schema
ALTER VIEW [ IF EXISTS ] name SET ( view_option_name [= view_
option_value] [, ... ] )
ALTER VIEW [ IF EXISTS ] name RESET ( view_option_name [, ... ] )
```

描述

`ALTER VIEW`更改视图的各种辅助属性。（如果要修改视图的定义查询，请使用 `CREATE OR REPLACE VIEW`。）

要使用`ALTER VIEW`，必须拥有该视图。要更改视图的模式，还必须在新模式上拥有`CREATE`权限。要更改所有者，还必须是新所有者角色的直接或间接成员，
并且该角色必须在视图所在模式上拥有`CREATE`权限。
（这些限制确保更改所有者不会执行任何无法通过删除并重新创建视图完成的操作。不过，超级用户无论如何都可以更改任何视图的所有权。）

参数

name

现有视图的名称（可以带模式限定）。

`IF EXISTS`

如果视图不存在，则不抛出错误，而是发出一条提示。

`SET/DROP DEFAULT`

这些形式用于设置或移除列的默认值。如果在视图上的 `INSERT` 语句中没有为某一列指定值，那么在应用该视图的 `ON INSERT` 规则之前，会先将与该视图列关联的默认值插入该 `INSERT` 语句。

new_owner

视图新拥有者的用户名。

new_name

该视图的新名称。

new_schema

该视图的新模式。

view_option_name

要设置或重置的视图选项的名称。

view_option_value

视图选项的新值。

注解

出于历史原因，视图也可以使用 `ALTER TABLE`；但允许用于视图的 `ALTER TABLE` 变体，仅限于与上面所示形式等价的那些。

示例

将视图 `foo` 重命名为 `bar`：

```
ALTER VIEW foo RENAME TO bar;
```

兼容性

`ALTER VIEW` 是 PostgreSQL 对 SQL 标准的扩展。

另见

CREATE VIEW, DROP VIEW

ANALYZE

ANALYZE — 收集数据库的统计信息

大纲

```
ANALYZE [ VERBOSE ] [ table_name [ ( column_name [, ...] ) ] ]
```

描述

ANALYZE收集数据库中各表内容的统计信息，并将结果存储到pg_statistic 系统目录中。随后，查询规划器会使用这些统计信息来帮助确定查询的最高效执行计划。

如果不带参数，ANALYZE 会检查当前数据库中的每个表。带有参数时，ANALYZE 只检查该表。还可以提供列名列表，这时只会收集这些列的统计信息。

参数

VERBOSE

启用进度消息显示。

table_name

要分析的特定表的名称（可带模式限定）。如果省略，
则分析当前数据库中的所有普通表（但不包括外部表）。

column_name

要分析的特定列的名称。默认为所有列。

输出

指定VERBOSE时，ANALYZE会输出进度消息，指示当前正在处理哪个表，
同时还会打印这些表的各种统计信息。

注解

只有在显式选中时才会分析外部表。并非所有外部数据包装器都支持ANALYZE。
如果该表的包装器不支持ANALYZE，命令会打印一条警告且不执行任何操作。

在默认的PostgreSQL配置中，自动清理守护进程（见第 23.1.6 节）会在表首次装载数据时，
以及在常规运行过程中数据发生变化时，自动分析这些表。当自动清理被禁用时，
最好定期运行ANALYZE，或者在对表内容做出重大更改之后立即运行。
准确的统计信息有助于规划器选择最合适的查询计划，从而提高查询处理速度。
对于以读取为主的数据库，一个常见策略是在每天使用率较低的时段运行一次
VACUUM和ANALYZE。（如果更新活动很频繁，这样做仍然不够。）

ANALYZE只需获取目标表上的读锁，因此可以与该表上的其他活动并行运行。

ANALYZE收集的统计信息通常包括每列中某些高频值的列表，
以及显示每列大致数据分布的直方图。如果ANALYZE认为它们没有意义（例如，
唯一键列中没有高频值），或者列的数据类型不支持相应的操作符，
则可能省略其中一项或两项。更多统计信息见第 23 章。

对于大型表，ANALYZE会对表内容进行随机采样，而不是检查每一行。
这使得即使是很大的表，也能在较短时间内完成分析。不过要注意，

这些统计信息只是近似值，并且即使实际表内容没有变化，每次运行ANALYZE时统计信息也会略有变化。这可能导致EXPLAIN中显示的规划器估算代价略有变化。在少数情况下，这种非确定性会导致规划器在运行ANALYZE之后改用不同的查询计划。要避免这种情况，可以按下文所述提高ANALYZE收集的统计信息量。

可以通过调整配置变量default_statistics_target来控制分析程度，也可以针对单个列使用ALTER TABLE ... ALTER COLUMN ... SET STATISTICS设置每列的统计信息目标（参见ALTER TABLE）。目标值会设置高频值列表中的最大条目数，以及直方图中的最大桶数。默认目标值是 100，但可以把它调高或调低，以在规划器估算精度、ANALYZE耗费的时间以及pg_statistic占用的空间之间作出权衡。特别地，将统计信息目标设置为零会禁用该列的统计信息收集。对于那些从不出现在查询 WHERE、GROUP BY或ORDER BY子句中的列，这样做可能很有用，因为规划器不会用到这些列的统计信息。

被分析列中最大的统计信息目标决定了为了生成统计信息而需要采样的表行数。增加该目标会导致执行ANALYZE所需的时间和空间按比例增加。

ANALYZE估算的值之一是每一列中出现的非重复值数量。由于只检查了部分行，即使使用可能的最大统计信息目标，这种估计有时也可能相当不精确。如果这种不精确导致查询计划不佳，就可以手工确定一个更精确的值，然后用 ALTER TABLE ... ALTER COLUMN ... SET (n_distinct = ...)把该值设置进去（参见ALTER TABLE）。

如果被分析的表有一个或多个子表，ANALYZE会两次收集统计信息：一次只针对父表中的行，另一次针对父表及其所有子表中的行。规划遍历整个继承树的查询时，需要第二组统计信息。不过，自动清理守护进程在决定是否为该表触发自动分析时，只会考虑对父表本身的插入或更新。如果该表很少被插入或更新，那么除非手工运行 ANALYZE，否则继承统计信息就不会保持最新。

如果被分析的表完全为空，ANALYZE将不会为该表记录新的统计信息。任何现有统计信息都会被保留。

兼容性

SQL 标准中没有ANALYZE语句。

另见

VACUUM, vacuumdb, 第 18.4.4 节, 第 23.1.6 节

BEGIN

BEGIN — 开始一个事务块

大纲

```
BEGIN [ WORK | TRANSACTION ] [ transaction_mode [, ...] ]
```

其中 *transaction_mode* 是下列之一:

```
    ISOLATION LEVEL { SERIALIZABLE | REPEATABLE READ | READ
 COMMITTED | READ UNCOMMITTED }
    READ WRITE | READ ONLY
    [ NOT ] DEFERRABLE
```

描述

`BEGIN`启动一个事务块，也就是说，在 `BEGIN`命令之后的所有语句都会在同一个事务中执行，直到显式给出COMMIT或ROLLBACK为止。默认情况下（不使用`BEGIN`），PostgreSQL以“自动提交”模式执行事务，也就是说，每条语句都在自己的事务中执行，并在语句结束时隐式提交（如果执行成功，否则回滚）。

在事务块中执行语句通常会更快，因为事务的启动和提交需要大量的 CPU 和磁盘活动。在进行多个相关更改时，
把多条语句放在一个事务中执行也有助于保证一致性：其他会话将无法看到那些相关更新尚未全部完成时的中间状态。

如果指定了隔离级别、读写模式或可延迟模式，新事务就会具有这些特征，就像执行了SET TRANSACTION一样。

参数

`WORK`
`TRANSACTION`

可选关键字，没有任何作用。

关于本语句其他参数含义的信息，参见SET TRANSACTION。

注解

START TRANSACTION与`BEGIN`具有相同功能。

使用COMMIT或ROLLBACK来结束事务块。

如果在已经处于事务块中时发出`BEGIN`，将会产生一条警告消息。事务状态不会受影响。
若要在事务块中实现事务嵌套，请使用保存点（参见SAVEPOINT）。

出于向后兼容的考虑，连续多个*transaction_modes*之间的逗号可以省略。

示例

开始一个事务块:

```
BEGIN;
```

兼容性

`BEGIN`是PostgreSQL的一种语言扩展。它等价于 SQL 标准命令START TRANSACTION，后者的参考页中包含更多兼容性信息。

`DEFERRABLE` *`transaction_mode`* 是PostgreSQL的一种语言扩展。

另外，在嵌入式 SQL 中，关键字`BEGIN`用于不同的目的。在移植数据库应用程序时，应当谨慎处理事务语义。

另见

COMMIT, ROLLBACK, START TRANSACTION, SAVEPOINT

CHECKPOINT

CHECKPOINT — 强制执行一次事务日志检查点

大纲

```
CHECKPOINT
```

描述

检查点是事务日志序列中的一个位置，到这一位置时，所有数据文件都已更新，
以反映日志中的信息。所有数据文件都会被刷写到磁盘。
有关检查点期间会发生什么的更多细节，参见第 29.4 节。

`CHECKPOINT`命令会在发出该命令时强制立即执行检查点，
而不是等待系统安排的常规检查点（由第 18.5.2 节中的设置控制）。`CHECKPOINT`并非设计用于正常运行期间。

如果在恢复期间执行`CHECKPOINT`命令，它将强制执行一个重启点（见第 29.4 节），而不是写入一个新的检查点。

只有超级用户才能调用`CHECKPOINT`。

兼容性

`CHECKPOINT`命令是 PostgreSQL的一种语言扩展。

CLOSE

CLOSE — 关闭一个游标

大纲

```
CLOSE { name | ALL }
```

描述

CLOSE释放与一个打开的游标相关联的资源。游标关闭后，
不允许再对其执行任何后续操作。当不再需要游标时，应将其关闭。

每个不可保持的打开游标都会在事务通过COMMIT或 ROLLBACK结束时被隐式关闭。可保持游标会在创建它的事务通过ROLLBACK中止时被隐式关闭。如果创建它的事务成功提交，
则该可保持游标会一直保持打开状态，直到显式执行 CLOSE，或者客户端断开连接。

参数

name

要关闭的打开游标的名称。

ALL

关闭所有打开的游标。

注解

PostgreSQL没有显式的OPEN游标语句；游标在声明时即被视为打开。
请使用DECLARE语句来声明游标。

可以通过查询pg_cursors 系统视图查看所有可用游标。

如果在某个保存点之后关闭了游标，而该保存点后来又被回滚，则CLOSE不会被回滚；
也就是说，该游标仍然保持关闭状态。

示例

关闭游标liahona：

```
CLOSE liahona;
```

兼容性

CLOSE完全符合 SQL 标准。CLOSE ALL是PostgreSQL的一种扩展。

另见

DECLARE, FETCH, MOVE

CLUSTER

CLUSTER — 按照一个索引对表进行聚簇

大纲

```
CLUSTER [VERBOSE] table_name [ USING index_name ]
CLUSTER [VERBOSE]
```

描述

`CLUSTER` 指示 PostgreSQL 按照 *`index_name`* 指定的索引，对 *`table_name`* 指定的表进行聚簇。该索引必须已经定义在 *`table_name`* 上。

当一个表被聚簇时，它会根据索引信息在物理上重新排序。
聚簇是一次性操作：之后如果表再被更新，这些更改不会再次被聚簇。也就是说，
系统不会试图按照索引顺序存储新行或更新后的行。（如果需要，
可以定期再次执行该命令来重新聚簇。此外，将表的 `FILLFACTOR` 存储参数设置为小于 100%，有助于在更新期间保持聚簇顺序，因为如果页面上有足够空间，更新后的行会保留在同一页面中。）

当一个表被聚簇时，PostgreSQL 会记住该表是按哪个索引聚簇的。形式 `CLUSTER` *`table_name`* 会使用与之前相同的索引对表重新聚簇。你也可以使用ALTER TABLE的 `CLUSTER` 或 `SET WITHOUT CLUSTER` 形式，设置未来聚簇操作要使用的索引，或者清除任何先前的设置。

不带任何参数的 `CLUSTER` 会对当前数据库中所有此前已聚簇且属于调用用户的表重新执行聚簇；如果由超级用户调用，则会对所有此前已聚簇的表重新执行聚簇。这种形式的 `CLUSTER` 不能在事务块内执行。

当对一个表执行聚簇时，会在该表上获取 `ACCESS EXCLUSIVE` 锁。这会阻止任何其他数据库操作（包括读和写）在 `CLUSTER` 完成前访问该表。

参数

`table_name`

表的名称（可能是模式限定的）。

`index_name`

索引的名称。

`VERBOSE`

在每个表被聚簇时打印进度报告。

注解

当你在表中随机访问单行时，数据在表中的实际顺序并不重要。但是，
如果访问某些数据比其他数据更频繁，并且存在将这些数据放在一起的索引，使用`CLUSTER`
就会有益。如果要从表中获取一段索引值范围，或者获取有多行匹配的单个索引值，
`CLUSTER`会有帮助，因为一旦索引确定第一个匹配行所在的表页，
其他匹配行很可能已在同一个表页中，从而减少磁盘访问并加快查询。

`CLUSTER` 可以通过在指定索引上进行索引扫描，或者（如果该索引是 B-树）先执行顺序扫描再排序，来对表重新排序。它会根据规划器代价参数和可用的统计信息，
尝试选择速度更快的方法。

使用索引扫描时，会创建一个表的临时副本，其中表数据按索引顺序排列。
还会为表上的每个索引创建临时副本。因此，
所需的磁盘空闲空间至少应等于表大小与索引大小之和。

使用顺序扫描加排序时，还会创建临时排序文件，
因此临时空间需求峰值可达表大小的两倍再加上索引大小。这种方法通常比索引扫描更快，
但如果无法接受其磁盘空间需求，可以临时将enable_sort设置为`off`来禁用这种选择。

建议在聚簇之前将maintenance_work_
mem设置为一个合理的大值（但不超过可专门用于`CLUSTER`操作的内存量）。

因为规划器会记录有关表中数据顺序的统计信息，建议在新近聚簇过的表上运行ANALYZE。
否则，规划器可能会选择很差的查询计划。

因为 `CLUSTER` 会记住哪些索引已被设为聚簇索引，你可以第一次先手工聚簇需要聚簇的表，
然后设置一个定期运行的维护脚本，执行不带任何参数的 `CLUSTER`，这样这些表就会被周期
性地重新聚簇。

示例

按照索引 `employees_ind` 对表 `employees` 进行聚簇：

```
CLUSTER employees USING employees_ind;
```

使用之前用过的同一个索引对 `employees` 表进行聚簇：

```
CLUSTER employees;
```

对数据库中此前已聚簇过的所有表执行聚簇：

```
CLUSTER;
```

兼容性

SQL 标准中没有 `CLUSTER` 语句。

语法

```
CLUSTER index_name ON table_name
```

也受支持，以兼容 8.3 之前的 PostgreSQL 版本。

另见

clusterdb

COMMENT

COMMENT — define or change the comment of an object

大纲

```
COMMENT ON
{
  AGGREGATE agg_name (agg_type [, ...] ) |
  CAST (source_type AS target_type) |
  COLLATION object_name |
  COLUMN relation_name.column_name |
  CONSTRAINT constraint_name ON table_name |
  CONVERSION object_name |
  DATABASE object_name |
  DOMAIN object_name |
  EXTENSION object_name |
  FOREIGN DATA WRAPPER object_name |
  FOREIGN TABLE object_name |
  FUNCTION function_name ( [ [ argmode ] [ argname ] argtype [, ...
] ] ) |
  INDEX object_name |
  LARGE OBJECT large_object_oid |
  OPERATOR operator_name (left_type, right_type) |
  OPERATOR CLASS object_name USING index_method |
  OPERATOR FAMILY object_name USING index_method |
  [ PROCEDURAL ] LANGUAGE object_name |
  ROLE object_name |
  RULE rule_name ON table_name |
  SCHEMA object_name |
  SEQUENCE object_name |
  SERVER object_name |
  TABLE object_name |
  TABLESPACE object_name |
  TEXT SEARCH CONFIGURATION object_name |
  TEXT SEARCH DICTIONARY object_name |
  TEXT SEARCH PARSER object_name |
  TEXT SEARCH TEMPLATE object_name |
  TRIGGER trigger_name ON table_name |
  TYPE object_name |
  VIEW object_name
} IS 'text'
```

Description

`COMMENT` stores a comment about a database object.

Only one comment string is stored for each object, so to modify a comment, issue a new `COMMENT` command for the same object. To remove a comment, write `NULL` in place of the text string. Comments are automatically dropped when their object is dropped.

For most kinds of object, only the object's owner can set the comment. Roles don't have owners, so the rule for `COMMENT ON ROLE` is that you must be superuser to comment on a superuser role, or have the `CREATEROLE` privilege to comment on non-superuser roles. Of course, a superuser can comment on anything.

Comments can be viewed using psql's `\d` family of commands. Other user interfaces to retrieve comments can be built atop the same built-in functions that psql uses, namely `obj_description`, `col_description`, and `shobj_description` (see 表 9.55).

Parameters

`object_name`
`relation_name.column_name`
`agg_name`
`constraint_name`
`function_name`
`operator_name`
`rule_name`
`trigger_name`

The name of the object to be commented. Names of tables, aggregates, collations, conversions, domains, foreign tables, functions, indexes, operators, operator classes, operator families, sequences, text search objects, types, and views can be schema-qualified. When commenting on a column, *`relation_name`* must refer to a table, view, composite type, or foreign table.

`agg_type`

An input data type on which the aggregate function operates. To reference a zero-argument aggregate function, write `*` in place of the list of input data types.

`source_type`

The name of the source data type of the cast.

`target_type`

The name of the target data type of the cast.

`argmode`

The mode of a function argument: `IN`, `OUT`, `INOUT`, or `VARIADIC`. If omitted, the default is `IN`. Note that `COMMENT ON FUNCTION` does not actually pay any attention to `OUT` arguments, since only the input arguments are needed to determine the function's identity. So it is sufficient to list the `IN`, `INOUT`, and `VARIADIC` arguments.

`argname`

The name of a function argument. Note that `COMMENT ON FUNCTION` does not actually pay any attention to argument names, since only the argument data types are needed to determine the function's identity.

`argtype`

The data type(s) of the function's arguments (optionally schema-qualified), if any.

`large_object_oid`

The OID of the large object.

`left_type`
`right_type`

The data type(s) of the operator's arguments (optionally schema-qualified). Write `NONE` for the missing argument of a prefix or postfix operator.

`PROCEDURAL`

This is a noise word.

`text`

The new comment, written as a string literal; or `NULL` to drop the comment.

Notes

There is presently no security mechanism for viewing comments: any user connected to a database can see all the comments for objects in that database. For shared objects such as databases, roles, and tablespaces, comments are stored globally so any user connected to any database in the cluster can see all the comments for shared objects. Therefore, don't put security-critical information in comments.

Examples

Attach a comment to the table `mytable`:

```
COMMENT ON TABLE mytable IS 'This is my table.';
```

Remove it again:

```
COMMENT ON TABLE mytable IS NULL;
```

Some more examples:

```
COMMENT ON AGGREGATE my_aggregate (double precision) IS 'Computes
 sample variance';
COMMENT ON CAST (text AS int4) IS 'Allow casts from text to int4';
COMMENT ON COLLATION "fr_CA" IS 'Canadian French';
COMMENT ON COLUMN my_table.my_column IS 'Employee ID number';
COMMENT ON CONVERSION my_conv IS 'Conversion to UTF8';
COMMENT ON CONSTRAINT bar_col_cons ON bar IS 'Constrains column
 col';
COMMENT ON DATABASE my_database IS 'Development Database';
COMMENT ON DOMAIN my_domain IS 'Email Address Domain';
COMMENT ON EXTENSION hstore IS 'implements the hstore data type';
COMMENT ON FOREIGN DATA WRAPPER mywrapper IS 'my foreign data
 wrapper';
COMMENT ON FOREIGN TABLE my_foreign_table IS 'Employee Information
 in other database';
COMMENT ON FUNCTION my_function (timestamp) IS 'Returns Roman
 Numeral';
COMMENT ON INDEX my_index IS 'Enforces uniqueness on employee ID';
COMMENT ON LANGUAGE plpython IS 'Python support for stored
 procedures';
COMMENT ON LARGE OBJECT 346344 IS 'Planning document';
COMMENT ON OPERATOR ^ (text, text) IS 'Performs intersection of two
 texts';
COMMENT ON OPERATOR - (NONE, integer) IS 'Unary minus';
COMMENT ON OPERATOR CLASS int4ops USING btree IS '4 byte integer
 operators for btrees';
COMMENT ON OPERATOR FAMILY integer_ops USING btree IS 'all integer
 operators for btrees';
COMMENT ON ROLE my_role IS 'Administration group for finance
 tables';
```

```
COMMENT ON RULE my_rule ON my_table IS 'Logs updates of employee
 records';
COMMENT ON SCHEMA my_schema IS 'Departmental data';
COMMENT ON SEQUENCE my_sequence IS 'Used to generate primary keys';
COMMENT ON SERVER myserver IS 'my foreign server';
COMMENT ON TABLE my_schema.my_table IS 'Employee Information';
COMMENT ON TABLESPACE my_tablespace IS 'Tablespace for indexes';
COMMENT ON TEXT SEARCH CONFIGURATION my_config IS 'Special word
 filtering';
COMMENT ON TEXT SEARCH DICTIONARY swedish IS 'Snowball stemmer for
 Swedish language';
COMMENT ON TEXT SEARCH PARSER my_parser IS 'Splits text into
 words';
COMMENT ON TEXT SEARCH TEMPLATE snowball IS 'Snowball stemmer';
COMMENT ON TRIGGER my_trigger ON my_table IS 'Used for RI';
COMMENT ON TYPE complex IS 'Complex number data type';
COMMENT ON VIEW my_view IS 'View of departmental costs';
```

Compatibility

There is no `COMMENT` command in the SQL standard.

COMMIT

COMMIT — 提交当前事务

大纲

```
COMMIT [ WORK | TRANSACTION ]
```

描述

`COMMIT`提交当前事务。该事务所做的所有更改都会对其他会话可见，并且如果发生崩溃，也能保证其持久性。

参数

`WORK`
`TRANSACTION`

可选关键字，没有任何作用。

注解

使用ROLLBACK中止事务。

在事务之外发出 `COMMIT` 不会造成危害，但会产生一条警告消息。

示例

提交当前事务，并使所有更改永久生效：

```
COMMIT;
```

兼容性

SQL 标准只规定了 `COMMIT` 和 `COMMIT WORK` 这两种形式。在其他方面，此命令完全符合标准。

另见

BEGIN, ROLLBACK

COMMIT PREPARED

COMMIT PREPARED — 提交一个先前为两阶段提交而预备的事务

大纲

`COMMIT PREPARED` *`transaction_id`*

描述

`COMMIT PREPARED`提交一个处于预备状态的事务。

参数

`transaction_id`

要提交的事务的事务标识符。

注解

要提交预备事务，执行者必须是最初执行该事务的同一用户，或者是超级用户；但不必处在执行该事务的同一会话中。

这个命令不能在事务块内执行。该预备事务会被立即提交。

当前所有处于预备状态的事务都列在 `pg_prepared_xacts` 系统视图中。

示例

提交事务标识符为`foobar`的事务：

```
COMMIT PREPARED 'foobar';
```

兼容性

`COMMIT PREPARED`是 PostgreSQL扩展。它旨在供外部事务管理系统使用，其中有些系统已被标准覆盖（例如 X/Open XA），但这些系统的 SQL 侧并未标准化。

另见

PREPARE TRANSACTION, ROLLBACK PREPARED

COPY

COPY — copy data between a file and a table

大纲

```
COPY table_name [ ( column_name [, ...] ) ]
    FROM { 'filename' | STDIN }
    [ [ WITH ] ( option [, ...] ) ]

COPY { table_name [ ( column_name [, ...] ) ] | ( query ) }
    TO { 'filename' | STDOUT }
    [ [ WITH ] ( option [, ...] ) ]

where option can be one of:

    FORMAT format_name
    OIDS [ boolean ]
    DELIMITER 'delimiter_character'
    NULL 'null_string'
    HEADER [ boolean ]
    QUOTE 'quote_character'
    ESCAPE 'escape_character'
    FORCE_QUOTE { ( column_name [, ...] ) | * }
    FORCE_NOT_NULL ( column_name [, ...] )
    ENCODING 'encoding_name'
```

Description

`COPY` moves data between PostgreSQL tables and standard file-system files. `COPY TO` copies the contents of a table to a file, while `COPY FROM` copies data from a file to a table (appending the data to whatever is in the table already). `COPY TO` can also copy the results of a `SELECT` query.

If a list of columns is specified, `COPY` will only copy the data in the specified columns to or from the file. If there are any columns in the table that are not in the column list, `COPY FROM` will insert the default values for those columns.

`COPY` with a file name instructs the PostgreSQL server to directly read from or write to a file. The file must be accessible to the server and the name must be specified from the viewpoint of the server. When `STDIN` or `STDOUT` is specified, data is transmitted via the connection between the client and the server.

Parameters

`table_name`

The name (optionally schema-qualified) of an existing table.

`column_name`

An optional list of columns to be copied. If no column list is specified, all columns of the table will be copied.

`query`

A SELECT or VALUES command whose results are to be copied. Note that parentheses are required around the query.

filename

The absolute path name of the input or output file. Windows users might need to use an `E''` string and double any backslashes used in the path name.

`STDIN`

Specifies that input comes from the client application.

`STDOUT`

Specifies that output goes to the client application.

boolean

Specifies whether the selected option should be turned on or off. You can write `TRUE`, `ON`, or `1` to enable the option, and `FALSE`, `OFF`, or `0` to disable it. The *boolean* value can also be omitted, in which case `TRUE` is assumed.

`FORMAT`

Selects the data format to be read or written: `text`, `csv` (Comma Separated Values), or `binary`. The default is `text`.

`OIDS`

Specifies copying the OID for each row. (An error is raised if `OIDS` is specified for a table that does not have OIDs, or in the case of copying a *query*.)

`DELIMITER`

Specifies the character that separates columns within each row (line) of the file. The default is a tab character in text format, a comma in `CSV` format. This must be a single one-byte character. This option is not allowed when using `binary` format.

`NULL`

Specifies the string that represents a null value. The default is `\N` (backslash-N) in text format, and an unquoted empty string in `CSV` format. You might prefer an empty string even in text format for cases where you don't want to distinguish nulls from empty strings. This option is not allowed when using `binary` format.

> **注意**
>
> When using `COPY FROM`, any data item that matches this string will be stored as a null value, so you should make sure that you use the same string as you used with `COPY TO`.

`HEADER`

Specifies that the file contains a header line with the names of each column in the file. On output, the first line contains the column names from the table, and on input, the first line is ignored. This option is allowed only when using `CSV` format.

`QUOTE`

Specifies the quoting character to be used when a data value is quoted. The default is double-quote. This must be a single one-byte character. This option is allowed only when using `CSV` format.

`ESCAPE`

Specifies the character that should appear before a data character that matches the `QUOTE` value. The default is the same as the `QUOTE` value (so that the quoting character is doubled if it appears in the data). This must be a single one-byte character. This option is allowed only when using `CSV` format.

`FORCE_QUOTE`

Forces quoting to be used for all non-`NULL` values in each specified column. `NULL` output is never quoted. If `*` is specified, non-`NULL` values will be quoted in all columns. This option is allowed only in `COPY TO`, and only when using `CSV` format.

`FORCE_NOT_NULL`

Do not match the specified columns' values against the null string. In the default case where the null string is empty, this means that empty values will be read as zero-length strings rather than nulls, even when they are not quoted. This option is allowed only in `COPY FROM`, and only when using `CSV` format.

`ENCODING`

Specifies that the file is encoded in the *`encoding_name`*. If this option is omitted, the current client encoding is used. See the Notes below for more details.

Outputs

On successful completion, a `COPY` command returns a command tag of the form

`COPY count`

The *`count`* is the number of rows copied.

Notes

`COPY` can only be used with plain tables, not with views. However, you can write `COPY (SELECT * FROM viewname) TO ...`.

`COPY` only deals with the specific table named; it does not copy data to or from child tables. Thus for example `COPY table TO` shows the same data as `SELECT * FROM ONLY table`. But `COPY (SELECT * FROM table) TO ...` can be used to dump all of the data in an inheritance hierarchy.

You must have select privilege on the table whose values are read by `COPY TO`, and insert privilege on the table into which values are inserted by `COPY FROM`. It is sufficient to have column privileges on the column(s) listed in the command.

Files named in a `COPY` command are read or written directly by the server, not by the client application. Therefore, they must reside on or be accessible to the database server machine, not the client. They must be accessible to and readable or writable by the PostgreSQL user (the user ID the server runs as), not the client. `COPY` naming a file is only allowed to database superusers, since it allows reading or writing any file that the server has privileges to access.

Do not confuse `COPY` with the psql instruction `\copy`. `\copy` invokes `COPY FROM STDIN` or `COPY TO STDOUT`, and then fetches/stores the data in a file accessible to the psql client. Thus, file accessibility and access rights depend on the client rather than the server when `\copy` is used.

It is recommended that the file name used in `COPY` always be specified as an absolute path. This is enforced by the server in the case of `COPY TO`, but for `COPY FROM` you

do have the option of reading from a file specified by a relative path. The path will be interpreted relative to the working directory of the server process (normally the cluster's data directory), not the client's working directory.

`COPY FROM` will invoke any triggers and check constraints on the destination table. However, it will not invoke rules.

`COPY` input and output is affected by `DateStyle`. To ensure portability to other PostgreSQL installations that might use non-default `DateStyle` settings, `DateStyle` should be set to `ISO` before using `COPY TO`. It is also a good idea to avoid dumping data with `IntervalStyle` set to `sql_standard`, because negative interval values might be misinterpreted by a server that has a different setting for `IntervalStyle`.

Input data is interpreted according to `ENCODING` option or the current client encoding, and output data is encoded in `ENCODING` or the current client encoding, even if the data does not pass through the client but is read from or written to a file directly by the server.

`COPY` stops operation at the first error. This should not lead to problems in the event of a `COPY TO`, but the target table will already have received earlier rows in a `COPY FROM`. These rows will not be visible or accessible, but they still occupy disk space. This might amount to a considerable amount of wasted disk space if the failure happened well into a large copy operation. You might wish to invoke `VACUUM` to recover the wasted space.

File Formats

Text Format

When the `text` format is used, the data read or written is a text file with one line per table row. Columns in a row are separated by the delimiter character. The column values themselves are strings generated by the output function, or acceptable to the input function, of each attribute's data type. The specified null string is used in place of columns that are null. `COPY FROM` will raise an error if any line of the input file contains more or fewer columns than are expected. If `OIDS` is specified, the OID is read or written as the first column, preceding the user data columns.

End of data can be represented by a single line containing just backslash-period (`\.`). An end-of-data marker is not necessary when reading from a file, since the end of file serves perfectly well; it is needed only when copying data to or from client applications using pre-3.0 client protocol.

Backslash characters (`\`) can be used in the `COPY` data to quote data characters that might otherwise be taken as row or column delimiters. In particular, the following characters must be preceded by a backslash if they appear as part of a column value: backslash itself, newline, carriage return, and the current delimiter character.

The specified null string is sent by `COPY TO` without adding any backslashes; conversely, `COPY FROM` matches the input against the null string before removing backslashes. Therefore, a null string such as `\N` cannot be confused with the actual data value `\N` (which would be represented as `\\N`).

The following special backslash sequences are recognized by `COPY FROM`:

Sequence	Represents
`\b`	Backspace (ASCII 8)
`\f`	Form feed (ASCII 12)

Sequence	Represents
`\n`	Newline (ASCII 10)
`\r`	Carriage return (ASCII 13)
`\t`	Tab (ASCII 9)
`\v`	Vertical tab (ASCII 11)
`\digits`	Backslash followed by one to three octal digits specifies the character with that numeri c code
`\xdigits`	Backslash `x` followed by one or two hex digits specifies the character with that numeric code

Presently, `COPY TO` will never emit an octal or hex-digits backslash sequence, but it does use the other sequences listed above for those control characters.

Any other backslashed character that is not mentioned in the above table will be taken to represent itself. However, beware of adding backslashes unnecessarily, since that might accidentally produce a string matching the end-of-data marker (`\.`) or the null string (`\N` by default). These strings will be recognized before any other backslash processing is done.

It is strongly recommended that applications generating `COPY` data convert data newlines and carriage returns to the `\n` and `\r` sequences respectively. At present it is possible to represent a data carriage return by a backslash and carriage return, and to represent a data newline by a backslash and newline. However, these representations might not be accepted in future releases. They are also highly vulnerable to corruption if the `COPY` file is transferred across different machines (for example, from Unix to Windows or vice versa).

`COPY TO` will terminate each row with a Unix-style newline (“`\n`”). Servers running on Microsoft Windows instead output carriage return/newline (“`\r\n`”), but only for `COPY` to a server file; for consistency across platforms, `COPY TO STDOUT` always sends “`\n`” regardless of server platform. `COPY FROM` can handle lines ending with newlines, carriage returns, or carriage return/newlines. To reduce the risk of error due to un-backslashed newlines or carriage returns that were meant as data, `COPY FROM` will complain if the line endings in the input are not all alike.

CSV Format

This format option is used for importing and exporting the Comma Separated Value (`CSV`) file format used by many other programs, such as spreadsheets. Instead of the escaping rules used by PostgreSQL's standard text format, it produces and recognizes the common CSV escaping mechanism.

The values in each record are separated by the `DELIMITER` character. If the value contains the delimiter character, the `QUOTE` character, the `NULL` string, a carriage return, or line feed character, then the whole value is prefixed and suffixed by the `QUOTE` character, and any occurrence within the value of a `QUOTE` character or the `ESCAPE` character is preceded by the escape character. You can also use `FORCE_QUOTE` to force quotes when outputting non-`NULL` values in specific columns.

The `CSV` format has no standard way to distinguish a `NULL` value from an empty string. PostgreSQL's `COPY` handles this by quoting. A `NULL` is output as the `NULL` parameter string and is not quoted, while a non-`NULL` value matching the `NULL` parameter string is quoted. For example, with the default settings, a `NULL` is written as an unquoted empty string, while an empty string data value is written with double quotes (`""`). Reading values follows similar rules. You can use `FORCE_NOT_NULL` to prevent `NULL` input comparisons for specific columns.

Because backslash is not a special character in the `CSV` format, `\.`, the end-of-data marker, could also appear as a data value. To avoid any misinterpretation, a `\.` data value appearing as a lone entry on a line is automatically quoted on output, and on input, if quoted, is not interpreted as the end-of-data marker. If you are loading a file created by another application that has a single unquoted column and might have a value of `\.`, you might need to quote that value in the input file.

> **注意**
>
> In `CSV` format, all characters are significant. A quoted value surrounded by white space, or any characters other than `DELIMITER`, will include those characters. This can cause errors if you import data from a system that pads `CSV` lines with white space out to some fixed width. If such a situation arises you might need to preprocess the `CSV` file to remove the trailing white space, before importing the data into PostgreSQL.

> **注意**
>
> CSV format will both recognize and produce CSV files with quoted values containing embedded carriage returns and line feeds. Thus the files are not strictly one line per table row like text-format files.

> **注意**
>
> Many programs produce strange and occasionally perverse CSV files, so the file format is more a convention than a standard. Thus you might encounter some files that cannot be imported using this mechanism, and `COPY` might produce files that other programs cannot process.

Binary Format

The `binary` format option causes all data to be stored/read as binary format rather than as text. It is somewhat faster than the text and `CSV` formats, but a binary-format file is less portable across machine architectures and PostgreSQL versions. Also, the binary format is very data type specific; for example it will not work to output binary data from a `smallint` column and read it into an `integer` column, even though that would work fine in text format.

The `binary` file format consists of a file header, zero or more tuples containing the row data, and a file trailer. Headers and data are in network byte order.

> **注意**
>
> PostgreSQL releases before 7.4 used a different binary file format.

File Header

The file header consists of 15 bytes of fixed fields, followed by a variable-length header extension area. The fixed fields are:

Signature

11-byte sequence `PGCOPY\n\377\r\n\0` — note that the zero byte is a required part of the signature. (The signature is designed to allow easy identification of files that have been munged by a non-8-bit-clean transfer. This signature will be changed by end-of-line-translation filters, dropped zero bytes, dropped high bits, or parity changes.)

Flags field

32-bit integer bit mask to denote important aspects of the file format. Bits are numbered from 0 (LSB) to 31 (MSB). Note that this field is stored in network byte order (most significant byte first), as are all the integer fields used in the file format. Bits 16-31 are reserved to denote critical file format issues; a reader should abort if it finds an unexpected bit set in this range. Bits 0-15 are reserved to signal backwards-compatible format issues; a reader should simply ignore any unexpected bits set in this range. Currently only one flag bit is defined, and the rest must be zero:

Bit 16

if 1, OIDs are included in the data; if 0, not

Header extension area length

32-bit integer, length in bytes of remainder of header, not including self. Currently, this is zero, and the first tuple follows immediately. Future changes to the format might allow additional data to be present in the header. A reader should silently skip over any header extension data it does not know what to do with.

The header extension area is envisioned to contain a sequence of self-identifying chunks. The flags field is not intended to tell readers what is in the extension area. Specific design of header extension contents is left for a later release.

This design allows for both backwards-compatible header additions (add header extension chunks, or set low-order flag bits) and non-backwards-compatible changes (set high-order flag bits to signal such changes, and add supporting data to the extension area if needed).

Tuples

Each tuple begins with a 16-bit integer count of the number of fields in the tuple. (Presently, all tuples in a table will have the same count, but that might not always be true.) Then, repeated for each field in the tuple, there is a 32-bit length word followed by that many bytes of field data. (The length word does not include itself, and can be zero.) As a special case, -1 indicates a NULL field value. No value bytes follow in the NULL case.

There is no alignment padding or any other extra data between fields.

Presently, all data values in a binary-format file are assumed to be in binary format (format code one). It is anticipated that a future extension might add a header field that allows per-column format codes to be specified.

To determine the appropriate binary format for the actual tuple data you should consult the PostgreSQL source, in particular the `*send` and `*recv` functions for each column's data type (typically these functions are found in the `src/backend/utils/adt/` directory of the source distribution).

If OIDs are included in the file, the OID field immediately follows the field-count word. It is a normal field except that it's not included in the field-count. In particular it has

a length word – this will allow handling of 4-byte vs. 8-byte OIDs without too much pain, and will allow OIDs to be shown as null if that ever proves desirable.

File Trailer

The file trailer consists of a 16-bit integer word containing -1. This is easily distinguished from a tuple's field-count word.

A reader should report an error if a field-count word is neither -1 nor the expected number of columns. This provides an extra check against somehow getting out of sync with the data.

Examples

The following example copies a table to the client using the vertical bar (`|`) as the field delimiter:

```
COPY country TO STDOUT (DELIMITER '|');
```

To copy data from a file into the `country` table:

```
COPY country FROM '/usr1/proj/bray/sql/country_data';
```

To copy into a file just the countries whose names start with 'A':

```
COPY (SELECT * FROM country WHERE country_name LIKE 'A%') TO '/
usr1/proj/bray/sql/a_list_countries.copy';
```

Here is a sample of data suitable for copying into a table from `STDIN`:

```
AF      AFGHANISTAN
AL      ALBANIA
DZ      ALGERIA
ZM      ZAMBIA
ZW      ZIMBABWE
```

Note that the white space on each line is actually a tab character.

The following is the same data, output in binary format. The data is shown after filtering through the Unix utility `od -c`. The table has three columns; the first has type `char(2)`, the second has type `text`, and the third has type `integer`. All the rows have a null value in the third column.

```
0000000   P   G   C   O   P   Y  \n 377  \r  \n  \0  \0  \0  \0  \0
  \0
0000020  \0  \0  \0  \0 003  \0  \0  \0 002   A   F  \0  \0  \0 013
   A
0000040   F   G   H   A   N   I   S   T   A   N 377 377 377 377  \0
 003
0000060  \0  \0  \0 002   A   L  \0  \0  \0 007   A   L   B   A   N
   I
0000100   A 377 377 377 377  \0 003  \0  \0  \0 002   D   Z  \0  \0
  \0
0000120 007   A   L   G   E   R   I   A 377 377 377 377  \0 003  \0
  \0
0000140  \0 002   Z   M  \0  \0  \0 006   Z   A   M   B   I   A 377
 377
0000160 377 377  \0 003  \0  \0  \0 002   Z   W  \0  \0  \0  \b   Z
   I
0000200   M   B   A   B   W   E 377 377 377 377 377 377
```

Compatibility

There is no COPY statement in the SQL standard.

The following syntax was used before PostgreSQL version 9.0 and is still supported:

```
COPY table_name [ ( column_name [, ...] ) ]
    FROM { 'filename' | STDIN }
    [ [ WITH ]
          [ BINARY ]
          [ OIDS ]
          [ DELIMITER [ AS ] 'delimiter' ]
          [ NULL [ AS ] 'null string' ]
          [ CSV [ HEADER ]
                [ QUOTE [ AS ] 'quote' ]
                [ ESCAPE [ AS ] 'escape' ]
                [ FORCE NOT NULL column_name [, ...] ] ] ]

COPY { table_name [ ( column_name [, ...] ) ] | ( query ) }
    TO { 'filename' | STDOUT }
    [ [ WITH ]
          [ BINARY ]
          [ OIDS ]
          [ DELIMITER [ AS ] 'delimiter' ]
          [ NULL [ AS ] 'null string' ]
          [ CSV [ HEADER ]
                [ QUOTE [ AS ] 'quote' ]
                [ ESCAPE [ AS ] 'escape' ]
                [ FORCE QUOTE { column_name [, ...] | * } ] ] ]
```

Note that in this syntax, BINARY and CSV are treated as independent keywords, not as arguments of a FORMAT option.

The following syntax was used before PostgreSQL version 7.3 and is still supported:

```
COPY [ BINARY ] table_name [ WITH OIDS ]
    FROM { 'filename' | STDIN }
    [ [USING] DELIMITERS 'delimiter' ]
    [ WITH NULL AS 'null string' ]

COPY [ BINARY ] table_name [ WITH OIDS ]
    TO { 'filename' | STDOUT }
    [ [USING] DELIMITERS 'delimiter' ]
    [ WITH NULL AS 'null string' ]
```

CREATE AGGREGATE

CREATE AGGREGATE — define a new aggregate function

大纲

```
CREATE AGGREGATE name ( input_data_type [ , ... ] ) (
    SFUNC = sfunc,
    STYPE = state_data_type
    [ , FINALFUNC = ffunc ]
    [ , INITCOND = initial_condition ]
    [ , SORTOP = sort_operator ]
)

or the old syntax

CREATE AGGREGATE name (
    BASETYPE = base_type,
    SFUNC = sfunc,
    STYPE = state_data_type
    [ , FINALFUNC = ffunc ]
    [ , INITCOND = initial_condition ]
    [ , SORTOP = sort_operator ]
)
```

Description

`CREATE AGGREGATE` defines a new aggregate function. Some basic and commonly-used aggregate functions are included with the distribution; they are documented in 第 9.20 节. If one defines new types or needs an aggregate function not already provided, then `CREATE AGGREGATE` can be used to provide the desired features.

If a schema name is given (for example, `CREATE AGGREGATE myschema.myagg ...`) then the aggregate function is created in the specified schema. Otherwise it is created in the current schema.

An aggregate function is identified by its name and input data type(s). Two aggregates in the same schema can have the same name if they operate on different input types. The name and input data type(s) of an aggregate must also be distinct from the name and input data type(s) of every ordinary function in the same schema.

An aggregate function is made from one or two ordinary functions: a state transition function *sfunc*, and an optional final calculation function *ffunc*. These are used as follows:

```
sfunc( internal-state, next-data-values ) ---> next-internal-state
ffunc( internal-state ) ---> aggregate-value
```

PostgreSQL creates a temporary variable of data type *stype* to hold the current internal state of the aggregate. At each input row, the aggregate argument value(s) are calculated and the state transition function is invoked with the current state value and the new argument value(s) to calculate a new internal state value. After all the rows have been processed, the final function is invoked once to calculate the aggregate's return value. If there is no final function then the ending state value is returned as-is.

An aggregate function can provide an initial condition, that is, an initial value for the internal state value. This is specified and stored in the database as a value of type `text`, but it must be a valid external representation of a constant of the state value data type. If it is not supplied then the state value starts out null.

If the state transition function is declared "strict", then it cannot be called with null inputs. With such a transition function, aggregate execution behaves as follows. Rows with any null input values are ignored (the function is not called and the previous state value is retained). If the initial state value is null, then at the first row with all-nonnull input values, the first argument value replaces the state value, and the transition function is invoked at subsequent rows with all-nonnull input values. This is handy for implementing aggregates like `max`. Note that this behavior is only available when *`state_data_type`* is the same as the first *`input_data_type`*. When these types are different, you must supply a nonnull initial condition or use a nonstrict transition function.

If the state transition function is not strict, then it will be called unconditionally at each input row, and must deal with null inputs and null transition values for itself. This allows the aggregate author to have full control over the aggregate's handling of null values.

If the final function is declared "strict", then it will not be called when the ending state value is null; instead a null result will be returned automatically. (Of course this is just the normal behavior of strict functions.) In any case the final function has the option of returning a null value. For example, the final function for `avg` returns null when it sees there were zero input rows.

Aggregates that behave like `MIN` or `MAX` can sometimes be optimized by looking into an index instead of scanning every input row. If this aggregate can be so optimized, indicate it by specifying a sort operator. The basic requirement is that the aggregate must yield the first element in the sort ordering induced by the operator; in other words:

```
SELECT agg(col) FROM tab;
```

must be equivalent to:

```
SELECT col FROM tab ORDER BY col USING sortop LIMIT 1;
```

Further assumptions are that the aggregate ignores null inputs, and that it delivers a null result if and only if there were no non-null inputs. Ordinarily, a data type's `<` operator is the proper sort operator for `MIN`, and `>` is the proper sort operator for `MAX`. Note that the optimization will never actually take effect unless the specified operator is the "less than" or "greater than" strategy member of a B-tree index operator class.

To be able to create an aggregate function, you must have `USAGE` privilege on the argument types, the state type, and the return type, as well as `EXECUTE` privilege on the transition and final functions.

Parameters

`name`

The name (optionally schema-qualified) of the aggregate function to create.

`input_data_type`

An input data type on which this aggregate function operates. To create a zero-argument aggregate function, write `*` in place of the list of input data types. (An example of such an aggregate is `count(*)`.)

base_type

In the old syntax for `CREATE AGGREGATE`, the input data type is specified by a `basetype` parameter rather than being written next to the aggregate name. Note that this syntax allows only one input parameter. To define a zero-argument aggregate function, specify the `basetype` as `"ANY"` (not `*`).

sfunc

The name of the state transition function to be called for each input row. For an *N*-argument aggregate function, the *sfunc* must take *N*+1 arguments, the first being of type *state_data_type* and the rest matching the declared input data type(s) of the aggregate. The function must return a value of type *state_data_type*. This function takes the current state value and the current input data value(s), and returns the next state value.

state_data_type

The data type for the aggregate's state value.

ffunc

The name of the final function called to compute the aggregate's result after all input rows have been traversed. The function must take a single argument of type *state_data_type*. The return data type of the aggregate is defined as the return type of this function. If *ffunc* is not specified, then the ending state value is used as the aggregate's result, and the return type is *state_data_type*.

initial_condition

The initial setting for the state value. This must be a string constant in the form accepted for the data type *state_data_type*. If not specified, the state value starts out null.

sort_operator

The associated sort operator for a `MIN`- or `MAX`-like aggregate. This is just an operator name (possibly schema-qualified). The operator is assumed to have the same input data types as the aggregate (which must be a single-argument aggregate).

The parameters of `CREATE AGGREGATE` can be written in any order, not just the order illustrated above.

Examples

See 第 35.10 节.

Compatibility

`CREATE AGGREGATE` is a PostgreSQL language extension. The SQL standard does not provide for user-defined aggregate functions.

See Also

ALTER AGGREGATE, DROP AGGREGATE

CREATE CAST

CREATE CAST — 定义一种新的类型转换

大纲

```
CREATE CAST (source_type AS target_type)
    WITH FUNCTION function_name [ (argument_type [, ...]) ]
    [ AS ASSIGNMENT | AS IMPLICIT ]

CREATE CAST (source_type AS target_type)
    WITHOUT FUNCTION
    [ AS ASSIGNMENT | AS IMPLICIT ]

CREATE CAST (source_type AS target_type)
    WITH INOUT
    [ AS ASSIGNMENT | AS IMPLICIT ]
```

描述

`CREATE CAST`定义一种新的类型转换。类型转换规定如何在两种数据类型之间执行转换。例如，

```
SELECT CAST(42 AS float8);
```

通过调用一个预先指定的函数（此处是 `float8(int4)`）把整型常量 42 转换成 `float8`类型。（如果没有定义合适的类型转换，转换就会失败。）

两种类型可以是二进制可强制转换的，这意味着无需调用任何函数，就可以“免费”执行转换。这要求对应的值使用相同的内部表示。例如，`text`和`varchar`这两种类型在两个方向上都是二进制可强制转换的。二进制可强制转换性不一定是对称关系。例如，在当前实现中，从`xml`到`text`的类型转换可以免费执行，但反方向则需要一个至少执行语法检查的函数。（双向都二进制可强制转换的两种类型也称为二进制兼容。）

使用`WITH INOUT`语法，你可以把一种类型转换定义为基于 I/O 的类型转换。基于 I/O 的类型转换通过调用源数据类型的输出函数，并将得到的字符串传给目标数据类型的输入函数来执行。在许多常见情况下，这项特性避免了为转换单独编写类型转换函数的必要。基于 I/O 的类型转换与常规的基于函数的类型转换行为相同，只是实现方式不同。

默认情况下，只有显式请求类型转换时才会调用一种类型转换，也就是显式使用`CAST(x AS typename)`或 `x::typename` 这种构造。

如果一种类型转换被标记为`AS ASSIGNMENT`，那么在把值赋给目标数据类型的列时就可以隐式调用它。例如，假设`foo.f1` 是一个`text`类型的列，那么如果从`integer`到 `text`的类型转换被标记为`AS ASSIGNMENT`，则：

```
INSERT INTO foo (f1) VALUES (42);
```

就会被允许，否则不会。（我们通常用术语赋值类型转换来描述这种类型转换。）

如果一种类型转换被标记为`AS IMPLICIT`，那么无论是在赋值上下文中还是在表达式内部，都可以在任何上下文中隐式调用它。（我们通常用术语隐式类型转换来描述这种类型转换。）例如，考虑这个查询：

```
SELECT 2 + 4.0;
```

解析器最初分别把这两个常量标记为integer和 numeric类型。系统目录中没有integer + numeric操作符，但有一个 numeric + numeric操作符。因此，如果存在一种从integer到numeric的可用类型转换，并且被标记为AS IMPLICIT — 实际上确实如此 — 该查询就会成功。解析器将应用该隐式类型转换，并把该查询解析为如同它被写成了：

```
SELECT CAST ( 2 AS numeric ) + 4.0;
```

现在，系统目录还提供了一种从numeric到integer的类型转换。
如果该类型转换被标记为AS IMPLICIT — 实际上并没有 — 那么解析器就必须在上面的解释方式和另一种方案之间作出选择：把numeric常量转换成integer，然后应用 integer + integer操作符。由于它不知道该偏向哪一种选择，就会放弃并将该查询判定为有歧义。两种类型转换中只有一种是隐式的，正是借此我们让解析器倾向于把混合了 numeric和integer的表达式解析为 numeric；系统对此并没有内置知识。

把类型转换标记为隐式时应当保持保守。过多的隐式类型转换路径可能导致 PostgreSQL对命令作出令人意外的解释，或者因为存在多种可能的解释而根本无法解析命令。
一个好的经验法则是，只有对同一一般类型分类中且能保留信息的类型间转换，
才让它可以被隐式调用。例如，从int2到int4的类型转换可以合理地设为隐式，但从float8到int4的类型转换大概应仅限赋值使用。跨类型分类的类型转换，例如从text到int4，最好只允许显式调用。

> **注意**
>
> 有时出于可用性或标准兼容性的原因，
> 有必要在一组类型之间提供多种隐式类型转换，
> 这会带来像上面那样无法避免的歧义。解析器有一种基于
> 类型分类和首选类型的后备启发式规则，在这种情况下有助于提供期望的行为。
> 详见CREATE TYPE。

要能够创建一种类型转换，你必须拥有源数据类型或目标数据类型之一，
并且对另一种类型具有USAGE权限。要创建一种二进制强制转换，你必须是超级用户。
（之所以有这一限制，是因为错误的二进制强制转换很容易使服务器崩溃。）

参数

source_type

该类型转换的源数据类型的名称。

target_type

该类型转换的目标数据类型的名称。

function_name(*argument_type* [, ...])

用于执行该类型转换的函数。函数名可以用模式限定；如果没有，
则会在模式搜索路径中查找该函数。
该函数的结果数据类型必须与类型转换的目标类型一致。其参数见下文。

WITHOUT FUNCTION

表示源类型对目标类型是二进制可强制转换的，因此执行该类型转换不需要函数。

WITH INOUT

表示该类型转换是一种基于 I/O 的类型转换，其执行方式是调用源数据类型的输出函数，
并将得到的字符串传给目标数据类型的输入函数。

`AS ASSIGNMENT`

表示该类型转换可以在赋值上下文中隐式调用。

`AS IMPLICIT`

表示该类型转换可以在任何上下文中隐式调用。

类型转换实现函数可以有一到三个参数。第一个参数类型必须与源类型相同，
或者可以由源类型进行二进制强制转换得到。第二个参数（如果有）必须是 `integer`类型；
它接收与目标类型关联的类型修饰符，如果没有则为`-1`。第三个参数（如果有）
必须是`boolean` 类型；如果该类型转换是显式类型转换，它接收`true`，否则接收`false`。
（奇怪的是，SQL 标准在某些情况下要求显式类型转换和隐式类型转换具有不同的行为。这个参数是为必须实现这类类型转换的函数提供的。
不建议你把自己的数据类型设计成需要关心这一点。）

类型转换函数的返回类型必须与类型转换的目标类型相同，
或者对该目标类型是二进制可强制转换的。

通常，类型转换的源数据类型和目标数据类型必须不同。不过，
如果它具有一个接受多个参数的类型转换实现函数，
则允许声明源类型和目标类型相同的类型转换。
这用于在系统目录中表示特定类型的长度强制函数。
所命名的函数用于将该类型的值强制为其第二个参数给定的类型修饰符值。

当一种类型转换的源类型和目标类型不同，且其函数接受多个参数时，
它支持在一个步骤中同时完成从一种类型到另一种类型的转换并应用长度强制。
如果没有这样的条目，
对使用类型修饰符的类型进行强制就需要两个类型转换步骤：先在数据类型之间进行转换，
再应用该修饰符。

当前，到域类型或从域类型的类型转换都没有效果。
到域或从域的类型转换都会使用与其底层类型关联的类型转换。

注解

使用DROP CAST删除用户定义的转换。

请记住，如果你希望能够双向转换类型，就需要在两个方向上分别显式声明类型转换。

通常没有必要在用户定义类型与标准字符串类型（`text`、`varchar`和`char(n)`，
以及被定义为属于字符串分类的用户定义类型）之间创建类型转换。
PostgreSQL会自动为此提供基于 I/O 的类型转换。转换到字符串类型的自动类型转换被视为赋值类型转换，而从字符串类型出发的自动类型转换则只允许显式调用。
你可以声明自己的类型转换来替代自动类型转换，从而覆盖这种行为，
但通常这样做的唯一原因，是希望该转换比标准的仅赋值或仅显式设置更容易调用。
另一种可能的原因是，你希望该转换的行为不同于该类型的 I/O 函数；但这已经足够反常，你应该三思这是不是一个好主意。（确实有少数内置类型在转换行为上有所不同，大多是由于 SQL 标准的要求。）

在 PostgreSQL 7.3 之前，每个与某个数据类型同名、返回该数据类型、并接受一个不同类型参数的函数都会自动成为类型转换函数。由于模式的引入，
以及为了能在系统目录中表示二进制可强制转换的类型转换，这一惯例已被放弃。
内建的类型转换函数仍遵循这一命名方案，但它们还必须作为类型转换显示在系统目录 `pg_cast` 中。

虽然这不是强制要求，但仍建议你继续遵循这种老惯例，
即按目标数据类型为类型转换实现函数命名。
许多用户已经习惯于用函数风格的记法进行类型转换，也就是*typename*(*x*)。

这种记法实际上无非就是调用类型转换实现函数；它不会被特别当作类型转换处理。
如果你的转换函数没有按这种惯例命名，那么用户会感到意外。由于 PostgreSQL允许同一函数名按不同参数类型重载，
因此让来自不同类型的多个转换函数都使用目标类型的名称并不存在困难。

> **注意**
>
> 实际上，前一段说得过于简单了：有两种情况下，
> 即使一个函数调用形式没有匹配到实际存在的函数，也会被当作类型转换请求。
> 如果函数调用 *name*(*x*)不能与任何现有函数精确匹配，但*name*是一个数据类型名，并且 `pg_cast`为从*x*的类型到该类型提供了二进制强制转换，那么该调用会被解释为二进制强制转换。作出这一例外，是为了让二进制强制转换即使没有任何函数，
> 也能使用函数语法调用。同样，如果没有`pg_cast`项，
> 但该类型转换的目标或源是字符串类型，则该调用会被解释为基于 I/O 的类型转换。
> 这一例外允许基于 I/O 的类型转换使用函数语法调用。

> **注意**
>
> 还有一个例外中的例外：从复合类型到字符串类型的基于 I/O 的类型转换不能使用函数语法调用，而必须写成显式类型转换语法（`CAST`或`::`记法）。增加这一例外，是因为在引入自动提供的基于 I/O 的类型转换之后，如果原意是函数调用或列引用，就太容易意外地触发这种类型转换了。

示例

要使用函数`int4(bigint)`创建一种从类型`bigint`到类型`int4`的赋值类型转换：

```
CREATE CAST (bigint AS int4) WITH FUNCTION int4(bigint) AS
 ASSIGNMENT;
```

（在系统中这种类型转换已经被预定义。）

兼容性

`CREATE CAST`命令符合SQL标准，不过 SQL 没有对二进制可强制转换的类型或实现函数的额外参数作出规定。`AS IMPLICIT`也是 PostgreSQL的扩展。

另见

CREATE FUNCTION, CREATE TYPE, DROP CAST

CREATE COLLATION

CREATE COLLATION — 定义一种新排序规则

大纲

```
CREATE COLLATION name (
    [ LOCALE = locale, ]
    [ LC_COLLATE = lc_collate, ]
    [ LC_CTYPE = lc_ctype ]
)
CREATE COLLATION name FROM existing_collation
```

描述

`CREATE COLLATION`使用指定的操作系统区域设置，或者通过复制现有排序规则，定义一个新的排序规则。

要创建排序规则，你必须拥有目标模式上的 `CREATE`权限。

参数

name

排序规则的名称。排序规则名称可以带有模式限定。如果没有，排序规则将在当前模式中定义。排序规则名称在该模式中必须唯一。（系统目录中可能包含其他编码的同名排序规则，但如果数据库编码不匹配，则这些排序规则将被忽略。）

locale

这是同时设置 `LC_COLLATE` 和 `LC_CTYPE` 的快捷方式。如果指定了此选项，则不能指定这两个参数中的任何一个。

lc_collate

为 `LC_COLLATE` 区域类别使用指定的操作系统区域设置。该区域设置必须适用于当前数据库编码。（有关精确规则，请参见CREATE DATABASE。）

lc_ctype

为 `LC_CTYPE` 区域类别使用指定的操作系统区域设置。该区域设置必须适用于当前数据库编码。（有关精确规则，请参见CREATE DATABASE。）

existing_collation

要复制的现有排序规则的名称。新的排序规则将具有与现有排序规则相同的属性，但它是一个独立对象。

注解

使用`DROP COLLATION`可移除用户定义的排序规则。

关于 PostgreSQL 中排序规则支持的更多信息，见第 22.2 节。

示例

创建排序规则，使用操作系统区域设置`fr_FR.utf8`（假设当前数据库编码为`UTF8`）：

```
CREATE COLLATION french (LOCALE = 'fr_FR.utf8');
```

从现有排序规则创建排序规则：

```
CREATE COLLATION german FROM "de_DE";
```

这在应用程序中使用与操作系统无关的排序规则名称时非常方便。

兼容性

在 SQL 标准中有一个`CREATE COLLATION`语句，但它仅限于复制现有排序规则。创建新排序规则的语法是 PostgreSQL扩展。

另见

ALTER COLLATION, DROP COLLATION

CREATE CONVERSION

CREATE CONVERSION — 定义一个新的编码转换

大纲

```
CREATE [ DEFAULT ] CONVERSION name
    FOR source_encoding TO dest_encoding FROM function_name
```

描述

CREATE CONVERSION 定义字符集编码之间的新转换。此外，被标记为 DEFAULT 的转换可用于客户端与服务器之间的自动编码转换。为此目的，必须定义两个转换：从编码 A 到 B 以及从编码 B 到 A。

要创建一个转换，你必须对该函数具有EXECUTE权限，并在目标模式上具有CREATE权限。

参数

DEFAULT

DEFAULT子句表示该转换是这一特定源编码到目标编码的默认转换。对于某个编码对，在一个模式中应该只有一个默认转换。

name

转换的名称。转换名可以是模式限定的；如果不是，则该转换定义在当前模式中。转换名在一个模式中必须唯一。

source_encoding

源编码名称。

dest_encoding

目标编码名称。

function_name

用于执行该转换的函数。函数名可以是模式限定的；如果不是，则会在搜索路径中查找该函数。

该函数必须具有以下签名：

```
conv_proc(
    integer,  -- source encoding ID
    integer,  -- destination encoding ID
    cstring,  -- source string (null terminated C string)
    internal, -- destination (fill with a null terminated C
 string)
    integer   -- source string length
) RETURNS void;
```

注解

使用DROP CONVERSION可以删除用户定义的转换。

创建转换所要求的权限在未来版本中可能会更改。

示例

使用`myfunc`创建从编码`UTF8`到 `LATIN1`的转换：

```
CREATE CONVERSION myconv FOR 'UTF8' TO 'LATIN1' FROM myfunc;
```

兼容性

`CREATE CONVERSION`是 PostgreSQL的一个扩展。SQL 标准中没有`CREATE CONVERSION`语句，但有一个在目的和语法上都非常相似的 `CREATE TRANSLATION`语句。

另见

ALTER CONVERSION, CREATE FUNCTION, DROP CONVERSION

CREATE DATABASE

CREATE DATABASE — 创建一个新数据库

大纲

```
CREATE DATABASE name
    [ [ WITH ] [ OWNER [=] user_name ]
           [ TEMPLATE [=] template ]
           [ ENCODING [=] encoding ]
           [ LC_COLLATE [=] lc_collate ]
           [ LC_CTYPE [=] lc_ctype ]
           [ TABLESPACE [=] tablespace_name ]
           [ CONNECTION LIMIT [=] connlimit ] ]
```

描述

`CREATE DATABASE` 创建一个新的 PostgreSQL 数据库。

要创建数据库，你必须是超级用户，或者拥有特殊的 `CREATEDB` 权限。见CREATE USER。

默认情况下，新数据库通过克隆标准系统数据库 `template1` 来创建。可以通过写成 `TEMPLATE name` 指定其他模板。特别是，写成 `TEMPLATE template0` 时，可以创建一个全新的数据库，它只包含你的 PostgreSQL 版本预定义的标准对象。如果你希望避免复制任何可能已添加到 `template1` 中的站点本地附加对象，这会很有用。

参数

name

要创建的数据库名称。

user_name

新数据库所有者的角色名称，或使用 `DEFAULT` 以使用默认值（即执行命令的用户）。要创建由其他角色拥有的数据库，你必须是该角色的直接或间接成员，或为超级用户。

template

用于创建新数据库的模板名称，或者指定 `DEFAULT` 以使用默认模板（`template1`）。

encoding

新数据库要使用的字符集编码。可指定字符串常量（例如 `'SQL_ASCII'`）、整数编码编号，或者指定 `DEFAULT` 以使用默认编码（即模板数据库的编码）。PostgreSQL 服务器支持的字符集见第 22.3.1 节。其他限制见下文。

lc_collate

新数据库中使用的排序规则（`LC_COLLATE`）。这会影响字符串的排序顺序，例如在带有 ORDER BY 的查询中，以及文本列索引中使用的顺序。默认值是使用模板数据库的排序规则。请参见下文的其他限制。

lc_ctype

新数据库中使用的字符分类（`LC_CTYPE`）。这会影响字符的分类，例如小写、大写和数字。默认值是使用模板数据库的字符分类。请参见下文的其他限制。

tablespace_name

将与新数据库关联的表空间名称，或者指定 DEFAULT 以使用模板数据库的表空间。该表空间将成为在此数据库中创建对象时使用的默认表空间。详见CREATE TABLESPACE。

connlimit

可对该数据库建立的并发连接数。-1（默认值）表示不受限制。

可选参数可以按任意顺序书写，不一定要按照上面的顺序。

注解

CREATE DATABASE 不能在事务块内执行。

形如 “could not initialize database directory” 的错误，多半与数据目录权限不足、磁盘已满或其他文件系统问题有关。

使用DROP DATABASE删除数据库。

程序createdb是这个命令的一个包装器程序，为方便使用而提供。

尽管可以通过把某个数据库名指定为模板，从而复制它而不是复制 template1，但这（至少目前）并不打算作为一种通用的“COPY DATABASE”功能。主要限制是，在复制模板数据库期间，不能有任何其他会话连接到该数据库。CREATE DATABASE 在启动时如果发现存在任何其他连接，就会失败；否则，在 CREATE DATABASE 完成之前，将阻止对模板数据库建立新的连接。详见第 21.3 节。

为新数据库指定的字符集编码必须与所选区域设置（LC_COLLATE 和 LC_CTYPE）兼容。如果区域设置为 C（或等价的 POSIX），则允许所有编码；但对于其他区域设置，只有一种编码能够正常工作。（不过，在 Windows 上，UTF-8 编码可与任何区域设置一起使用。）CREATE DATABASE 允许超级用户不考虑区域设置而指定 SQL_ASCII 编码，但这种选择已弃用；如果数据库中存储了与该区域设置不兼容编码的数据，字符串函数的行为可能会出错。

编码和区域设置必须与模板数据库的设置一致，除非使用 template0 作为模板。这是因为其他数据库可能包含与指定编码不匹配的数据，或者包含排序顺序会受 LC_COLLATE 和 LC_CTYPE 影响的索引。复制这样的数据会导致数据库在新设置下被视为损坏。不过，已知 template0 不包含任何会受此影响的数据或索引。

CONNECTION LIMIT 选项只是近似地被强制执行；如果两个新会话几乎同时启动，而该数据库只剩下一个连接“槽”，则两者都可能失败。此外，该限制对超级用户无效。

示例

要创建一个新数据库：

```
CREATE DATABASE lusiadas;
```

要创建一个由用户 salesapp 拥有、默认表空间为 salesspace 的数据库 sales：

```
CREATE DATABASE sales OWNER salesapp TABLESPACE salesspace;
```

要创建支持 ISO-8859-1 字符集的数据库music：

```
CREATE DATABASE music ENCODING 'LATIN1' TEMPLATE template0;
```

在这个示例中，只有当template1的编码不是 ISO-8859-1 时，才需要TEMPLATE template0子句。注意，更改编码可能还需要选择新的LC_COLLATE和LC_CTYPE设置。

兼容性

SQL 标准中没有 `CREATE DATABASE` 语句。数据库相当于目录，而目录的创建由实现定义。

参见

ALTER DATABASE, DROP DATABASE

CREATE DOMAIN

CREATE DOMAIN — 定义一个新域

大纲

```
CREATE DOMAIN name [ AS ] data_type
    [ COLLATE collation ]
    [ DEFAULT expression ]
    [ constraint [ ... ] ]
```

其中 *constraint* 为:

```
[ CONSTRAINT constraint_name ]
{ NOT NULL | NULL | CHECK (expression) }
```

描述

`CREATE DOMAIN`创建一个新域。域本质上是一种带有可选约束（即对允许值集合的限制）的数据类型。定义域的用户将成为其拥有者。

如果给定了模式名（例如`CREATE DOMAIN myschema.mydomain ...`），则该域会在指定模式中创建。否则它会在当前模式中创建。
域名在其所在模式中的现有类型和域之间必须唯一。

域适合把字段上的常见约束抽象到单一位置进行维护。例如，
若有多个表都包含电子邮件地址列，并且都需要同一个 CHECK 约束来验证地址语法，那么定义一个域会比在每个表上分别设置该约束更合适。

要创建域，你必须对其底层类型拥有`USAGE`权限。

参数

name

要创建的域名（可选地带模式限定）。

data_type

该域的底层数据类型。它可以包含数组说明符。

collation

该域的可选排序规则。如果未指定排序规则，则使用底层数据类型的默认排序规则。
如果指定了`COLLATE`，则底层类型必须是一种支持排序规则的数据类型。

`DEFAULT` *expression*

`DEFAULT`子句为该域数据类型的列指定默认值。
该值可以是任意不含变量的表达式（但不允许子查询）。
默认表达式的数据类型必须与该域的数据类型匹配。如果未指定默认值，
则默认值为空值。

默认表达式会在任何未为该列指定值的插入操作中使用。
如果为某个特定列定义了默认值，它就会覆盖与该域关联的任何默认值。反过来，
域默认值又会覆盖与底层数据类型关联的任何默认值。

`CONSTRAINT constraint_name`

约束的可选名称。如果未指定，系统会生成一个名称。

`NOT NULL`

该域的值通常不允许为空值。不过，
如果给该域赋一个已成为空值的匹配域类型值（例如通过左外连接，或 `INSERT INTO tab (domcol) VALUES ((SELECT domcol FROM tab WHERE false))`），带此约束的域仍可能取到空值。

`NULL`

该域的值允许为空值。这是默认行为。

该子句仅用于与非标准 SQL 数据库兼容。不鼓励在新应用中使用它。

`CHECK (expression)`

`CHECK`子句指定该域的值必须满足的完整性约束或测试。
每个约束都必须是一个产生布尔结果的表达式。它应使用关键字`VALUE`
来引用被测试的值。

当前，`CHECK`表达式不能包含子查询，也不能引用 `VALUE`之外的其他变量。

示例

这个示例创建`us_postal_code`数据类型，然后在一个表定义中使用该类型。
这里使用正则表达式测试来验证该值看起来是否为一个合法的美国邮政编码：

```
CREATE DOMAIN us_postal_code AS TEXT
CHECK(
   VALUE ~ '^\d{5}$'
OR VALUE ~ '^\d{5}-\d{4}$'
);

CREATE TABLE us_snail_addy (
  address_id SERIAL PRIMARY KEY,
  street1 TEXT NOT NULL,
  street2 TEXT,
  street3 TEXT,
  city TEXT NOT NULL,
  postal us_postal_code NOT NULL
);
```

兼容性

命令`CREATE DOMAIN`符合 SQL 标准。

另见

ALTER DOMAIN, DROP DOMAIN

CREATE EXTENSION

CREATE EXTENSION — 安装一个扩展

大纲

```
CREATE EXTENSION [ IF NOT EXISTS ] extension_name
    [ WITH ] [ SCHEMA schema_name ]
             [ VERSION version ]
             [ FROM old_version ]
```

描述

CREATE EXTENSION将一个新扩展装载到当前数据库中。同名扩展不能已经被装载。

装载一个扩展本质上就是运行该扩展的脚本文件。该脚本通常会创建新的 SQL 对象，例如函数、数据类型、操作符以及索引支持方法。CREATE EXTENSION还会记录所有已创建对象的标识，这样在发出DROP EXTENSION时就可以将它们一并删除。

加载扩展需要与创建其组件对象所需的权限相同。对于大多数扩展，
这意味着需要超级用户或数据库所有者权限。运行 CREATE EXTENSION 的用户将成为扩展的所有者，用于后续的权限检查，以及扩展脚本创建的任何对象的所有者。

参数

IF NOT EXISTS

如果已存在同名扩展，则不要抛出错误。这种情况下会发出一条提示。注意，
这并不保证现有扩展与根据当前可用脚本文件本应创建出的扩展有任何相似之处。

extension_name

要安装的扩展名称。PostgreSQL将根据 SHAREDIR/extension/*extension_name*.control 文件中的详细信息创建该扩展。

schema_name

如果该扩展允许其内容被重定位，则这是安装该扩展所含对象的模式名称。
指定的模式必须已经存在。如果未指定，并且扩展控制文件中也未指定模式，
则使用当前默认的对象创建模式。

请记住，扩展本身不被视为属于任何模式：扩展的名称是非限定名，
并且必须在整个数据库范围内唯一。但是，属于该扩展的对象可以位于模式中。

version

要安装的扩展版本。它可以写成标识符，也可以写成字符串常量。
默认版本由扩展控制文件指定。

old_version

FROM *old_version* 必须在且仅在尝试安装一个替换“旧式”模块的扩展时指定，该模块仅是一组未打包为扩展的对象集合。此选项会导致 CREATE EXTENSION 运行一个替代安装脚本，将现有对象吸收进扩展中，而不是创建新对象。请确保 SCHEMA 指定的是包含这些现有对象的模式。

*old_version*的值由扩展作者决定，如果存在多个可升级为扩展的旧式模块版本，该值可能会有所不同。对于PostgreSQL 9.1 之前版本提供的标准附加模块，将模块更新为扩展形式时应使用unpackaged作为*old_version*的值。

注解

在使用 CREATE EXTENSION 将扩展装载到数据库之前，必须先安装好该扩展的支持文件。关于安装 PostgreSQL 随附扩展的信息，可见额外提供的模块。

当前可供装载的扩展可从系统视图 pg_available_extensions 或 pg_available_extension_versions 中查出。

关于编写新扩展的信息，见第 35.15 节。

示例

将 hstore 扩展安装到当前数据库中：

```
CREATE EXTENSION hstore;
```

将 PostgreSQL 9.1 之前版本中安装的hstore更新为扩展形式：

```
CREATE EXTENSION hstore SCHEMA public FROM unpackaged;
```

请注意指定安装现有hstore对象时所用的模式。

兼容性

CREATE EXTENSION 是 PostgreSQL 的扩展。

另见

ALTER EXTENSION, DROP EXTENSION

CREATE FOREIGN DATA WRAPPER

CREATE FOREIGN DATA WRAPPER — 定义一个新的外部数据包装器

大纲

```
CREATE FOREIGN DATA WRAPPER name
    [ HANDLER handler_function | NO HANDLER ]
    [ VALIDATOR validator_function | NO VALIDATOR ]
    [ OPTIONS ( option 'value' [, ... ] ) ]
```

描述

`CREATE FOREIGN DATA WRAPPER`创建一个新的外部数据包装器。定义该外部数据包装器的用户将成为它的拥有者。

外部数据包装器名称在数据库内必须唯一。

只有超级用户能够创建外部数据包装器。

参数

name

要创建的外部数据包装器的名称。

`HANDLER` *handler_function*

handler_function 是一个预先注册的函数的名称，该函数将被调用以取得外部表所需的执行函数。该处理器函数不能接受任何参数，并且其返回类型必须是 `fdw_handler`。

可以创建没有处理器函数的外部数据包装器，但使用这种包装器的外部表只能被声明，不能被访问。

`VALIDATOR` *validator_function*

*validator_function*是一个预先注册的函数的名称，该函数将被调用以检查提供给该外部数据包装器的通用选项，以及用于外部服务器、
用户映射和使用该外部数据包装器的外部表的选项。如果未指定验证器函数或者指定了`NO VALIDATOR`，则在创建时不会检查这些选项。（外部数据包装器可能会在运行时忽略无效的选项声明，或者拒绝它们，这取决于其实现。）
验证器函数必须接受两个参数：一个参数的类型为 `text[]`，其中包含系统目录中存储的选项数组；另一个参数的类型为`oid`，其值是包含这些选项的系统目录的 OID。返回类型会被忽略；该函数应当使用 `ereport(ERROR)`报告无效选项。

`OPTIONS (` *option* `'`*value*`' [, ... ] )`

这个子句为新的外部数据包装器指定选项。
允许的选项名和值都依赖于各个外部数据包装器，
并通过该外部数据包装器的验证器函数进行验证。选项名必须唯一。

注解

目前，外部数据包装器功能还很初级。不支持更新外部表，
查询优化也很原始（而且大部分也留给了包装器本身）。

提供了一个内建的外部数据包装器验证器函数：`postgresql_fdw_validator`，它接受与 libpq 连接参数对应的选项。

示例

创建一个无用的外部数据包装器`dummy`：

```
CREATE FOREIGN DATA WRAPPER dummy;
```

用处理器函数`file_fdw_handler`创建一个外部数据包装器 `file`：

```
CREATE FOREIGN DATA WRAPPER file HANDLER file_fdw_handler;
```

用一些选项创建一个外部数据包装器`mywrapper`：

```
CREATE FOREIGN DATA WRAPPER mywrapper
    OPTIONS (debug 'true');
```

兼容性

`CREATE FOREIGN DATA WRAPPER`符合 ISO/IEC 9075-9 (SQL/MED)，但`HANDLER`和`VALIDATOR`子句属于扩展，而标准中的 `LIBRARY`和`LANGUAGE`子句尚未实现。

不过需要注意，SQL/MED 功能整体上尚未完全符合该标准。

另见

ALTER FOREIGN DATA WRAPPER, DROP FOREIGN DATA WRAPPER, CREATE SERVER, CREATE USER MAPPING

CREATE FOREIGN TABLE

CREATE FOREIGN TABLE — 定义一个新外部表

大纲

```
CREATE FOREIGN TABLE [ IF NOT EXISTS ] table_name ( [
  { column_name data_type [ OPTIONS ( option 'value' [, ... ] ) ] [
 NULL | NOT NULL ] }
    [, ... ]
] )
  SERVER server_name
[ OPTIONS ( option 'value' [, ... ] ) ]
```

描述

`CREATE FOREIGN TABLE`将在当前数据库中创建一个新的外部表。该表归发出该命令的用户所有。

如果给出了模式名（例如 `CREATE FOREIGN TABLE myschema.mytable ...`），则表将在指定模式中创建。否则，它将在当前模式中创建。外部表名必须与同一模式中任何其他关系（表、序列、索引、视图或外部表）的名称不同。

`CREATE FOREIGN TABLE`还会自动创建一种数据类型，用以表示与该外部表一行对应的复合类型。因此，外部表名不能与同一模式中任何已有数据类型同名。

要创建表，必须拥有表中所用所有列类型上的`USAGE`权限。

参数

`IF NOT EXISTS`

已经存在同名关系时不要抛出错误。这种情况下会发出一个提示。注意，已存在的关系不保证与原本将要创建的关系有任何相似之处。

`table_name`

要创建的表名（可选地带有模式限定）。

`column_name`

要在新表中创建的列名。

`data_type`

该列的数据类型，可以包含数组说明符。有关PostgreSQL支持的数据类型的更多信息，参见第 8 章。

`NOT NULL`

该列不允许包含空值。

`NULL`

该列允许包含空值。这是默认行为。

此子句仅用于与非标准 SQL 数据库兼容。在新应用程序中不建议使用。

server_name

要用于外部表的现有服务器的名称。

OPTIONS (*option* '*value*' [, ...])

与新外部表或其某一列关联的选项。允许的选项名称和值取决于每个外部数据包装器，并通过外部数据包装器的验证函数进行验证。
不允许重复的选项名称（尽管表选项和列选项具有相同名称是可以接受的）。

示例

创建通过服务器film_server访问的外部表films：

```
CREATE FOREIGN TABLE films (
    code        char(5) NOT NULL,
    title       varchar(40) NOT NULL,
    did         integer NOT NULL,
    date_prod   date,
    kind        varchar(10),
    len         interval hour to minute
)
SERVER film_server;
```

兼容性

CREATE FOREIGN TABLE 命令在很大程度上符合 SQL 标准；然而，与 CREATE TABLE 类似，允许使用 NULL 约束和零列外部表。

另见

ALTER FOREIGN TABLE, DROP FOREIGN TABLE, CREATE TABLE, CREATE SERVER

CREATE FUNCTION

CREATE FUNCTION — 定义一个新函数

大纲

```
CREATE [ OR REPLACE ] FUNCTION
    name ( [ [ argmode ] [ argname ] argtype [ { DEFAULT | = }
 default_expr ] [, ...] ] )
    [ RETURNS rettype
      | RETURNS TABLE ( column_name column_type [, ...] ) ]
  { LANGUAGE lang_name
    | WINDOW
    | IMMUTABLE | STABLE | VOLATILE | [ NOT ] LEAKPROOF
    | CALLED ON NULL INPUT | RETURNS NULL ON NULL INPUT | STRICT
    | [ EXTERNAL ] SECURITY INVOKER | [ EXTERNAL ] SECURITY DEFINER
    | COST execution_cost
    | ROWS result_rows
    | SET configuration_parameter { TO value | = value | FROM
 CURRENT }
    | AS 'definition'
    | AS 'obj_file', 'link_symbol'
  } ...
    [ WITH ( attribute [, ...] ) ]
```

描述

CREATE FUNCTION定义一个新函数。CREATE OR REPLACE FUNCTION将创建一个新函数，或者替换现有定义。要定义函数，用户必须具有该语言上的USAGE权限。

如果包含模式名，则函数将在指定的模式中创建。否则将在当前模式中创建。
新函数的名称不能与同一模式中具有相同输入参数类型的任何现有函数匹配。然而，
不同参数类型的函数可以共享一个名称（这称为重载）。

要替换一个现有函数的当前定义，可以使用CREATE OR REPLACE FUNCTION。但不能用这种方式更改函数的名称或者参数类型（如果尝试这样做，
实际上就会创建一个新的不同函数）。此外，CREATE OR REPLACE FUNCTION也不允许更改现有函数的返回类型。要做到这一点，必须删除该函数并重新创建。（使用OUT参数时，
这意味着除非删除该函数，否则不能更改任何OUT参数的类型。）

当CREATE OR REPLACE FUNCTION被用来替换一个现有函数时，该函数的拥有权和权限不会改变。所有其他的函数属性会按照该命令中指定的或者隐含的值赋值。
必须拥有（包括成为拥有角色的成员）该函数才能替换它。

如果删除函数后再重新创建，新函数就不再是旧函数的同一实体；
你将必须删除引用旧函数的现有规则、视图、触发器等。使用CREATE OR REPLACE FUNCTION可以在不破坏引用该函数的对象的情况下更改函数定义。此外，ALTER FUNCTION还可用于更改现有函数的大多数辅助属性。

创建该函数的用户将成为该函数的拥有者。

要创建一个函数，你必须拥有参数类型和返回类型上的USAGE权限。

参数

name

要创建的函数名称（可以被模式限定）。

argmode

参数的模式可以是：IN、OUT、INOUT或者VARIADIC。如果省略，则默认为IN。只有OUT参数可以跟在VARIADIC参数之后。此外，OUT和INOUT参数不能与RETURNS TABLE记法一起使用。

argname

参数的名称。某些语言（包括 SQL 和 PL/pgSQL）允许在函数体中使用该名称。对于其他语言，就函数本身而言，输入参数的名称只是额外文档；但你可以在调用函数时使用输入参数名来提高可读性（见第 4.3 节）。无论如何，输出参数的名称很重要，因为它定义了结果行类型中的列名。（如果省略输出参数的名称，系统将选择一个默认列名。）

argtype

该函数参数（如果有）的数据类型（可以是模式限定的）。参数类型可以是基础类型、复合类型或者域类型，也可以引用一个表列的类型。

根据实现语言的不同，也可能允许指定诸如cstring这样的“伪类型”。伪类型表示实际参数类型要么没有被完整指定，要么不属于普通 SQL 数据类型集合。

写成*table_name*.*column_name*%TYPE即可引用一个列的类型。使用这种特性有时有助于让函数独立于表定义的变化。

default_expr

如果未指定该参数，则用作默认值的表达式。该表达式必须能被强制转换为该参数的类型。只有输入参数（包括INOUT）才能有默认值。所有跟在具有默认值参数之后的输入参数也都必须有默认值。

rettype

该函数的返回数据类型（可以是模式限定的）。返回类型可以是基础类型、复合类型或者域类型，也可以引用一个表列的类型。根据实现语言的不同，也可能允许指定诸如cstring这样的“伪类型”。如果函数不应该返回值，请把返回类型指定为void。

当存在OUT或INOUT参数时，可以省略RETURNS子句。如果写出该子句，它必须与输出参数所隐含的结果类型一致：如果有多个输出参数，则为RECORD；如果只有一个输出参数，则为该输出参数的类型。

SETOF修饰符表示该函数将返回一组项，而不是单个项。

写成*table_name*.*column_name*%TYPE即可引用一个列的类型。

column_name

RETURNS TABLE语法中输出列的名称。这实际上是声明一个具名OUT参数的另一种方式，只不过RETURNS TABLE还隐含了RETURNS SETOF。

column_type

RETURNS TABLE语法中的输出列的数据类型。

lang_name

实现该函数所用语言的名称。可以是SQL、C、internal，也可以是用户定义的过程语言。出于向后兼容，名称可以用单引号括起。

WINDOW

WINDOW表示该函数是窗口函数而不是普通函数。目前这只对用 C 编写的函数有用。在替换现有函数定义时，不能更改WINDOW属性。

IMMUTABLE
STABLE
VOLATILE

这些属性会告诉查询优化器该函数的行为。最多只能指定其中一个。
如果这些属性都没有出现，则默认假定为VOLATILE。

IMMUTABLE表示该函数不能修改数据库，并且在给定相同参数值时总会返回相同结果；
也就是说，它不会执行数据库查找，
也不会以其他方式使用未直接出现在其参数列表中的信息。如果给出此选项，
任何使用全常量参数对该函数的调用都可以立即替换为该函数值。

STABLE表示该函数不能修改数据库，并且在一次表扫描内，
对于相同参数值会一致地返回相同结果，但其结果可能在不同 SQL 语句之间发生变化。
这适用于结果依赖于数据库查找、参数变量（例如当前时区）等的函数。
（对于希望查询由当前命令修改过的行的AFTER触发器，这样做并不合适。）另请注意，
current_timestamp函数族也属于稳定函数，因为它们的值在一个事务内不会变化。

VOLATILE表示该函数的值即使在一次表扫描内也可能发生变化，
因此无法进行任何优化。从这个意义上说，真正不稳定的数据库函数相对较少；
一些例子是random()、currval()、timeofday()。但请注意，
任何有副作用的函数都必须归类为不稳定，即使其结果相当可预测，也必须如此，
以防其调用被优化掉；例如setval()。

更多细节见第 35.6 节。

LEAKPROOF

LEAKPROOF表示该函数没有副作用。除返回值外，它不会泄露其参数的任何信息。
例如，对某些参数值会抛出错误而对另一些不会，或者在错误消息中包含参数值的函数，
都不是防泄漏的。这会影响系统如何执行针对使用security_barrier
选项创建的视图或启用了行级安全的表的查询。为了防止数据被无意暴露，
系统会先强制执行安全策略和安全屏障视图中的条件，
再执行查询本身中包含非防泄漏函数的用户提供条件。
被标记为防泄漏的函数和操作符被视为可信，
因此可以在安全策略和安全屏障视图的条件之前执行。此外，不接受参数的函数，
或者没有从安全屏障视图或表中接收到任何参数的函数，即使未标记为防泄漏，
也可以在安全条件之前执行。参见CREATE VIEW和第 37.4 节。此选项只能由超级用户设置。

CALLED ON NULL INPUT
RETURNS NULL ON NULL INPUT
STRICT

CALLED ON NULL INPUT（默认）表示当某些参数为空值时，仍会正常调用该函数。如果有需要，则由函数作者负责检查空值并作出适当响应。

RETURNS NULL ON NULL INPUT或STRICT表示只要任一参数为空值，该函数总是返回空值。如果指定了这个选项，那么在参数中出现空值时不会执行该函数，
而是自动假定结果为空值。

[EXTERNAL] SECURITY INVOKER
[EXTERNAL] SECURITY DEFINER

SECURITY INVOKER表示该函数将以调用它的用户的权限执行。这是默认设置。
SECURITY DEFINER指定该函数将以创建它的用户的权限执行。

为了符合 SQL，允许使用关键字EXTERNAL。但它是可选的，因为与 SQL 不同，这个特性适用于所有函数，而不仅仅是外部函数。

execution_cost

一个正数，给出该函数的估计执行代价，单位为cpu_operator_cost。
如果该函数返回一个集合，则这是每个返回行的代价。如果未指定代价，则对 C 语言和内部函数假定为 1 个单位，对其他所有语言的函数假定为 100 个单位。较大的值会让规划器尽量避免对该函数进行不必要的频繁求值。

result_rows

一个正数，给出规划器应预计该函数返回的行数。
只有函数被声明为返回集合时才允许使用此项。默认假定为 1000 行。

configuration_parameter
value

SET选项会在进入函数时将指定的配置参数设为给定值，
并在函数退出时恢复为先前的值。SET FROM CURRENT会把执行CREATE FUNCTION时该参数的当前值保存下来，作为进入函数时要应用的值。

如果函数附带了SET选项，那么在函数内针对同一变量执行的SET LOCAL命令，其效果会被限制在该函数内部：函数退出时，配置参数先前的值仍会被恢复。不过，普通的SET命令（不带LOCAL）会覆盖SET选项，就像它会覆盖先前的SET LOCAL命令一样：这类命令的效果会在函数退出后继续保持，除非当前事务被回滚。

关于允许的参数名和值的更多信息，见SET和第 18 章。

definition

一个定义该函数的字符串常量，其含义取决于所用语言。它可以是一个内部函数名称、一个对象文件的路径、一个 SQL 命令，或者用一种过程语言编写的文本。

使用美元引用（见第 4.1.2.4 节）来书写函数定义字符串通常会更有帮助，而不是使用普通的单引号语法。如果没有美元引用，
函数定义中的任何单引号或者反斜线都必须用双写来转义。

obj_file, *link_symbol*

当 C 语言源代码中的函数名称与 SQL 函数名称不同时，这种形式的AS子句用于可动态加载的 C 语言函数。字符串*obj_file*是包含该动态可加载对象的文件名，而字符串*link_symbol*是函数的链接符号，即 C 语言源代码中函数的名称。如果省略链接符号，则假定它与正在定义的 SQL 函数名称相同。

当重复的CREATE FUNCTION调用引用同一个对象文件时，该文件在每个会话中只装载一次。要卸载并重新装载该文件（例如在开发期间），请启动一个新会话。

attribute

用于指定函数可选信息的历史方式。以下属性可以出现在此处：

isStrict

等同于STRICT或RETURNS NULL ON NULL INPUT。

isCachable

isCachable是IMMUTABLE的已废弃等价写法；出于向后兼容的原因，
目前仍接受它。

属性名称不区分大小写。

有关编写函数的详细信息，请参阅第 35.3 节。

重载

PostgreSQL允许函数重载；也就是说，只要输入参数类型不同，
同一个名称就可以用于多个不同的函数。但是，所有函数的 C 名称都必须不同，因此必须为重载的 C 函数指定不同的 C 名称（例如将参数类型作为 C 名称的一部分）。

如果两个函数具有相同的名称和输入参数类型，它们被认为相同（不考虑任何`OUT`参数）。
因此这些声明会冲突：

```
CREATE FUNCTION foo(int) ...
CREATE FUNCTION foo(int, out text) ...
```

参数类型列表不同的函数在创建时不会被视为冲突，但如果提供了默认值，
则在使用时可能发生冲突。例如，考虑下面这些声明：

```
CREATE FUNCTION foo(int) ...
CREATE FUNCTION foo(int, int default 42) ...
```

调用`foo(10)`会失败，因为系统无法确定应该调用哪个函数。

注解

允许使用完整的SQL类型语法来声明函数参数和返回值。不过，`CREATE FUNCTION`会丢弃带圆括号的类型修饰符（例如`numeric`类型的精度字段）。因此，`CREATE FUNCTION foo (varchar(10)) ...`与`CREATE FUNCTION foo (varchar) ...`完全等同。

在用`CREATE OR REPLACE FUNCTION`替换现有函数时，更改参数名会受到限制。不能更改已经分配给任何输入参数的名称（但可以给先前没有名称的参数补上名称）。
如果输出参数多于一个，也不能更改输出参数的名称，
因为那会改变描述函数结果的匿名复合类型的列名。这些限制是为了确保函数被替换时，
已有的函数调用不会停止工作。

如果一个函数被声明为带有`VARIADIC`参数的`STRICT`函数，
则严格性检查测试的是可变参数数组作为一个整体是否非空。如果该数组包含空值元素，
仍会调用该函数。

示例

下面给出一些简单示例，帮助你开始使用。有关更多信息和示例，请参见第 35.3 节。

```
CREATE FUNCTION add(integer, integer) RETURNS integer
    AS 'select $1 + $2;'
    LANGUAGE SQL
    IMMUTABLE
    RETURNS NULL ON NULL INPUT;
```

在PL/pgSQL中，使用参数名把一个整数加 1：

```
CREATE OR REPLACE FUNCTION increment(i integer) RETURNS integer AS
 $$
        BEGIN
                RETURN i + 1;
        END;
$$ LANGUAGE plpgsql;
```

返回一个包含多个输出参数的记录：

```
CREATE FUNCTION dup(in int, out f1 int, out f2 text)
```

```
    AS $$ SELECT $1, CAST($1 AS text) || ' is text' $$
    LANGUAGE SQL;

SELECT * FROM dup(42);
```

你也可以用一个显式命名的复合类型，更详细地表达同样的意思：

```
CREATE TYPE dup_result AS (f1 int, f2 text);

CREATE FUNCTION dup(int) RETURNS dup_result
    AS $$ SELECT $1, CAST($1 AS text) || ' is text' $$
    LANGUAGE SQL;

SELECT * FROM dup(42);
```

另一种返回多列的方法是使用TABLE函数：

```
CREATE FUNCTION dup(int) RETURNS TABLE(f1 int, f2 text)
    AS $$ SELECT $1, CAST($1 AS text) || ' is text' $$
    LANGUAGE SQL;

SELECT * FROM dup(42);
```

不过，TABLE函数与前面的示例不同，因为它实际返回的是一组记录，而不只是单条记录。

安全地编写 SECURITY DEFINER函数

因为SECURITY DEFINER函数要以创建它的用户的权限执行，所以必须小心确保该函数不会被滥用。出于安全考虑，应将search_path设置为排除任何可被不受信任用户写入的模式。这可以防止恶意用户创建对象来遮蔽该函数使用的对象。
在这方面尤其重要的是临时表模式；默认情况下它最先被搜索，而且通常任何人都可写。一个安全的安排是强制把临时模式放到搜索顺序的最后。要做到这一点，应把pg_temp写成search_path中的最后一项。下面这个函数展示了安全用法：

```
CREATE FUNCTION check_password(uname TEXT, pass TEXT)
RETURNS BOOLEAN AS $$
DECLARE passed BOOLEAN;
BEGIN
        SELECT  (pwd = $2) INTO passed
        FROM    pwds
        WHERE   username = $1;

        RETURN passed;
END;
$$  LANGUAGE plpgsql
    SECURITY DEFINER
    -- 设置一个安全的 search_path: 受信的模式，然后是 'pg_temp'。
    SET search_path = admin, pg_temp;
```

在PostgreSQL 8.3 版本之前，SET选项尚不可用，因此较旧的函数可能包含相当复杂的逻辑来保存、设置和恢复search_path。使用SET选项更容易实现此目的。

还需注意，默认情况下，新创建的函数会将执行权限授予PUBLIC（有关更多信息，请参见GRANT）。通常，你会希望将安全定义器函数的使用限制为某些用户。为此，必须撤销默认的PUBLIC权限，然后有选择地授予执行权限。
为避免新函数在一段时间内对所有人可访问，应在同一个事务中创建函数并设置权限。例如：

```
BEGIN;
```

```
CREATE FUNCTION check_password(uname TEXT, pass TEXT) ... SECURITY
 DEFINER;
REVOKE ALL ON FUNCTION check_password(uname TEXT, pass TEXT) FROM
 PUBLIC;
GRANT EXECUTE ON FUNCTION check_password(uname TEXT, pass TEXT) TO
 admins;
COMMIT;
```

兼容性

`CREATE FUNCTION`命令在 SQL:1999 及更高版本中定义。PostgreSQL版本与其相似，但并不完全兼容。这些属性和可用的不同语言都不具备可移植性。

为与某些其他数据库系统兼容，*`argmode`*可以写在*`argname`*之前或之后，但只有前一种写法符合标准。

SQL 标准没有规定参数默认值。使用`DEFAULT`关键字的语法来自 Oracle，并且在某种程度上符合标准的精神：SQL/PSM 将它用于变量默认值。使用=的语法则用于 T-SQL 和 Firebird。

另见

ALTER FUNCTION, DROP FUNCTION, GRANT, LOAD, REVOKE, createlang

CREATE GROUP

CREATE GROUP — 定义一个新的数据库角色

大纲

```
CREATE GROUP name [ [ WITH ] option [ ... ] ]
```

其中*option*可以是：

```
      SUPERUSER | NOSUPERUSER
    | CREATEDB | NOCREATEDB
    | CREATEROLE | NOCREATEROLE
    | CREATEUSER | NOCREATEUSER
    | INHERIT | NOINHERIT
    | LOGIN | NOLOGIN
    | [ ENCRYPTED | UNENCRYPTED ] PASSWORD 'password'
    | VALID UNTIL 'timestamp'
    | IN ROLE role_name [, ...]
    | IN GROUP role_name [, ...]
    | ROLE role_name [, ...]
    | ADMIN role_name [, ...]
    | USER role_name [, ...]
    | SYSID uid
```

描述

`CREATE GROUP` 现在是CREATE ROLE的别名。

兼容性

SQL 标准中没有 `CREATE GROUP` 语句。

另见

CREATE ROLE

CREATE INDEX

CREATE INDEX — 定义一个新索引

大纲

```
CREATE [ UNIQUE ] INDEX [ CONCURRENTLY ] [ name ] ON table_name [
 USING method ]
    ( { column_name | ( expression ) } [ COLLATE collation ] [
 opclass ] [ ASC | DESC ] [ NULLS { FIRST | LAST } ] [, ...] )
    [ WITH ( storage_parameter = value [, ... ] ) ]
    [ TABLESPACE tablespace_name ]
    [ WHERE predicate ]
```

描述

`CREATE INDEX`在指定表的指定列上构建一个索引，索引主要用于提升数据库性能（但使用不当也可能导致性能下降）。

索引的键字段指定为列名，或者指定为写在圆括号中的表达式。
如果索引方法支持多列索引，则可以指定多个字段。

索引字段可以是根据表行中一个或多个列值计算得到的表达式。
该特性可用于根据基础数据的某种变换来快速访问数据。例如，在`upper(col)`
上计算的索引可让子句 `WHERE upper(col) = 'JIM'`使用索引。

PostgreSQL提供了索引方法 B-树、hash、GiST、SP-GiST 以及 GIN。用户也可以定义自己的索引方法，但这相当复杂。

当`WHERE`子句存在时，会创建一个部分索引。部分索引只包含表中一部分行的索引项，
通常这一部分比表的其余部分更适合建立索引。例如，
如果一个表同时包含已开票和未开票订单，而未开票订单只占整个表的一小部分，
但这一部分又经常被访问，就可以只对这部分创建索引来提升性能。另一种可能的应用是将
`WHERE`与`UNIQUE`结合使用，以便在表的一个子集上强制唯一性。更多讨论请见第 11.8 节。

`WHERE`子句中使用的表达式只能引用底层表的列，但它可以使用所有列，
而不仅仅是被索引的列。当前，`WHERE`中也禁止使用子查询和聚合表达式。
同样的限制也适用于作为表达式的索引字段。

所有在索引定义中使用的函数和操作符都必须是“不可变的”，也就是说，
它们的结果只能依赖其参数，
而不能受任何外部因素影响（例如另一个表的内容或当前时间）。
这种限制确保索引的行为定义明确。要在索引表达式或`WHERE`子句中使用用户定义的函数，
记得在创建该函数时将其标记为不可变。

参数

`UNIQUE`

使系统在创建索引时（如果数据已经存在）以及每次添加数据时，检查表中的重复值。
任何会导致重复项的插入或更新操作都会报错。

`CONCURRENTLY`

使用此选项时，PostgreSQL将在构建索引时不获取任何会阻止对表进行并发插入、
更新或删除的锁；而标准索引构建会阻塞对表的写入（但不会阻塞读取），
直到构建完成。使用此选项时有几个注意事项需要了解–请参见并发构建索引。

name

要创建的索引名称。此处不能包含模式名称；索引始终创建在其父表所在的同一模式中。如果省略名称，PostgreSQL会根据父表名称和被索引的列名选择合适的名称。

table_name

要建立索引的表名（可以是模式限定名）。

method

要使用的索引方法的名称。选择包括`btree`、`hash`、`gist`、`spgist`和`gin`。默认方法是`btree`。

column_name

一个表列的名称。

expression

一个基于表中一个或多个列的表达式。通常必须像语法中所示那样写在外围圆括号中。不过，如果该表达式是函数调用形式，则可以省略圆括号。

collation

将用于该索引的排序规则名称。默认情况下，索引使用被索引列声明的排序规则，或者被索引表达式的结果排序规则。对于涉及使用非默认排序规则表达式的查询，使用非默认排序规则的索引可能会很有用。

opclass

一个操作符类的名称。详见下文。

`ASC`

指定升序排序（默认）。

`DESC`

指定降序排序。

`NULLS FIRST`

指定把空值排序在非空值前面。在指定`DESC`时，这是默认行为。

`NULLS LAST`

指定把空值排序在非空值后面。在没有指定`DESC`时，这是默认行为。

storage_parameter

索引方法专用存储参数的名称。有关详细信息，请参见索引存储参数。

tablespace_name

在其中创建索引的表空间。如果未指定，将查阅default_tablespace；对于临时表上的索引，则查阅temp_tablespaces。

predicate

部分索引的约束表达式。

索引存储参数

可选的WITH子句为索引指定存储参数。每一种索引方法都有其各自允许的存储参数集合。B-树、hash、GiST 和 SP-GiST 索引方法都接受以下参数：

fillfactor

索引的填充因子是一个百分比，用于确定索引方法将尝试把索引页填充到多满。对于 B-树，在初始构建索引期间，以及向右扩展索引（添加新的最大键值）时，叶页都会填充到该百分比。如果之后页面变得完全填满，就会进行拆分，导致索引效率逐渐下降。B-树使用默认填充因子 90，但可以选择 10 到 100 之间的任意整数值。如果表是静态的，填充因子 100 最适合将索引的物理大小降至最低；但对于频繁更新的表，较小的填充因子更适合减少页面拆分的需要。其他索引方法以不同但大致类似的方式使用填充因子；默认填充因子因方法而异。

GiST 索引还接受以下参数：

buffering

确定是否使用第 53.3.1 节中描述的缓冲构建技术来构建索引。设置为OFF时禁用，设置为ON时启用，设置为AUTO时初始禁用，但一旦索引大小达到effective_cache_size，就会动态启用。默认值为AUTO。

GIN 索引接受不同的参数：

fastupdate

此设置控制第 55.3.1 节中描述的快速更新技术的使用。这是一个布尔参数：ON启用快速更新，OFF禁用快速更新。（如第 18.1 节中所述，允许使用ON和OFF的其他拼写形式。）默认值为ON。

> **注意**
>
> 通过ALTER INDEX关闭fastupdate 会阻止后续插入进入待处理索引项列表，但这本身不会刷新现有条目。之后可能需要对该表执行VACUUM，以确保待处理列表被清空。

并发构建索引

创建索引可能会干扰数据库的正常运行。通常 PostgreSQL会锁住要建立索引的表，阻止其写入，并通过一次扫描完成整个索引构建。其他事务仍可读取该表，但如果它们试图在表中插入、更新或删除行，就会阻塞直到索引构建完成。如果系统是在线生产数据库，这可能产生严重影响。对非常大的表建立索引可能需要很多小时，即便是较小的表，索引构建也可能在一段对生产系统而言不可接受的时间内阻止写入者操作。

PostgreSQL支持在不阻止写入的情况下构建索引。这种方法通过在CREATE INDEX中指定 CONCURRENTLY选项来启用。使用该选项时，PostgreSQL必须对该表执行两次扫描，此外还必须等待所有现有、可能修改或使用该索引的事务结束。因此，这种方法比标准索引构建需要更多总工作量，完成时间也明显更长。不过，由于它允许在构建索引期间继续进行正常操作，所以这种方法适合在生产环境中新增索引。当然，创建索引带来的额外 CPU 和 I/O 负载也可能拖慢其他操作。

在并发索引构建中，索引实际上会在一个事务中录入系统目录，然后在另外两个事务中执行两次表扫描。每次表扫描之前，索引构建都必须等待已修改该表的现有事务结束。第二次扫描之后，索引构建必须等待所有持有早于第二次扫描的快照（参见第 13 章）的事务结束，其中包括其

他表上并发索引构建任一阶段使用的事务。最后，索引才可以标记为可用，CREATE INDEX命令随之结束。不过即便如此，索引也可能无法立即用于查询：在最坏情况下，只要还存在早于索引构建开始的事务，就不能使用它。

如果在扫描表时出现问题，例如唯一索引中的唯一性冲突，CREATE INDEX命令将失败，但会留下一个“无效”索引。由于该索引可能不完整，查询时会忽略它；但是，它仍会带来更新开销。该psql \d命令会将此类索引报告为INVALID：

```
postgres=# \d tab
       Table "public.tab"
 Column |  Type   | Modifiers
--------+---------+-----------
 col    | integer |
Indexes:
    "idx" btree (col) INVALID
```

在这种情况下，建议的恢复方法是删除索引，然后重新尝试执行CREATE INDEX CONCURRENTLY。（另一种可能性是重建索引，使用REINDEX。但是，由于REINDEX不支持并发构建，此选项不太有吸引力。）

并发构建唯一索引时的另一项注意事项是，在第二次表扫描开始时，唯一性约束就已经开始对其他事务生效了。这意味着在该索引可供使用之前，其他查询就可能报告约束违规，甚至在索引构建最终失败的情况下也是如此。另外，如果第二次扫描确实失败了，那个“无效”索引之后仍会继续强制执行其唯一性约束。

也支持并发构建表达式索引和部分索引。计算这些表达式时发生的错误，可能导致与上文所述唯一性约束违规类似的行为。

普通索引构建允许同一表上的其他普通索引构建并行进行，但一张表上一次只能进行一个并发索引构建。在这两种情况下，期间都不允许对该表进行其他类型的模式修改。另一个区别是，普通CREATE INDEX命令可以在事务块中执行，而CREATE INDEX CONCURRENTLY不能。

注解

关于索引何时能被使用、何时不被使用以及什么情况下它们有用的信息请见第 11 章。

> **小心**
>
> 哈希索引操作目前不会写入 WAL 日志，因此如果数据库崩溃时还有未写入的更改，哈希索引可能需要用REINDEX重建。此外，在初始基础备份之后，对哈希索引的更改不会通过流复制或基于文件的复制进行复制，因此之后使用这些索引的查询会得到错误的答案。哈希索引在时间点恢复期间也不能被正确恢复。由于这些原因，目前不鼓励使用哈希索引。

目前，只有 B-树、GiST 和 GIN 索引方法支持多列索引。默认最多可指定 32 个字段。（构建PostgreSQL时可以更改此限制。）目前只有 B-树支持唯一索引。

索引的每一列都可以指定一个操作符类。操作符类标识索引用于该列的操作符。例如，四字节整数上的 B-树索引会使用int4_ops类；此操作符类包含四字节整数的比较函数。实际上，列数据类型的默认操作符类通常已足够。操作符类存在的主要原因是，某些数据类型可能有多个有意义的排序方式。例如，我们可能希望按绝对值或实部对复数数据类型排序。我们可以为该数据类型定义两个操作符类，并在创建索引时选择合适的类。有关操作符类的更多信息，请参见第 11.9 节和第 35.14 节。

对于支持有序扫描的索引方法（当前只有 B-树），可以指定可选子句 ASC、DESC、NULLS FIRST 和/或NULLS LAST来修改索引的排序顺序。由于有序索引可以向前或向后扫描，因此创建单列DESC索引通常并无用处 — 常规索引已经提供了这种排序顺序。这些选项的价值在于可以创建与混合排序查询所要求顺序相匹配的多列索引，例如 SELECT ... ORDER BY x ASC, y DESC。如果需要在依靠索引避免排序步骤的查询中支持“空值排在低位”，而不是默认的“空值排在高位”行为，那么NULLS选项就很有用。

对于大多数索引方法，索引的创建速度取决于maintenance_work_mem的设置。
较大的值将会减少索引创建所需的时间，
当然不要把它设置得超过实际可用的内存量（那会迫使机器进行交换）。

使用DROP INDEX删除索引。

早期版本的PostgreSQL还提供过一种 R-tree 索引方法。该方法已经被移除，因为它相对于 GiST 方法并无明显优势。如果指定了USING rtree，CREATE INDEX 会将其解释为USING gist，以简化旧数据库向 GiST 的转换。

示例

创建 B-树索引，列为title，所在表为films：

```
CREATE UNIQUE INDEX title_idx ON films (title);
```

要在表达式lower(title)上创建一个索引，以便高效执行不区分大小写的搜索：

```
CREATE INDEX ON films ((lower(title)));
```

（在这个示例中，索引名称被省略，因此系统会选择一个名字，通常为films_lower_idx。）

要创建一个具有非默认排序规则的索引：

```
CREATE INDEX title_idx_german ON films (title COLLATE "de_DE");
```

要创建一个具有非默认空值排序顺序的索引：

```
CREATE INDEX title_idx_nulls_low ON films (title NULLS FIRST);
```

要创建一个具有非默认填充因子的索引：

```
CREATE UNIQUE INDEX title_idx ON films (title) WITH (fillfactor =
 70);
```

要创建一个禁用快速更新的GIN索引：

```
CREATE INDEX gin_idx ON documents_table USING GIN (locations) WITH
 (fastupdate = off);
```

要在表films的列code上创建一个索引，并让该索引驻留在表空间indexspace中：

```
CREATE INDEX code_idx ON films (code) TABLESPACE indexspace;
```

要在点属性上创建一个 GiST 索引，以便能够在转换函数的结果上高效地使用 box 操作符：

```
CREATE INDEX pointloc
    ON points USING gist (box(location,location));
SELECT * FROM points
    WHERE box(location,location) && '(0,0),(1,1)'::box;
```

要在不阻止对表执行写操作的情况下创建索引：

```
CREATE INDEX CONCURRENTLY sales_quantity_index ON sales_table (
quantity);
```

兼容性

`CREATE INDEX`是 PostgreSQL的语言扩展。SQL 标准中没有关于索引的规定。

另见

ALTER INDEX, DROP INDEX

CREATE LANGUAGE

CREATE LANGUAGE — 定义一种新的过程语言

大纲

```
CREATE [ OR REPLACE ] [ PROCEDURAL ] LANGUAGE name
CREATE [ OR REPLACE ] [ TRUSTED ] [ PROCEDURAL ] LANGUAGE name
    HANDLER call_handler [ INLINE inline_handler ] [
 VALIDATOR valfunction ]
```

描述

`CREATE LANGUAGE`在PostgreSQL数据库中注册一种新的过程语言。随后，可以用这种新语言定义函数和触发器过程。

> **注意**
>
> 从PostgreSQL 9.1 开始，大多数过程语言都已变成“扩展”，因此应使用CREATE EXTENSION而不是`CREATE LANGUAGE`来安装。现在直接使用`CREATE LANGUAGE`应仅限于扩展安装脚本。如果数据库中有一个“裸”语言（例如升级的结果），可以使用`CREATE EXTENSION langname FROM unpackaged`将其转换为扩展。

`CREATE LANGUAGE`实际上是将该语言名称与负责执行用这种语言编写的函数的处理器函数关联起来。关于语言处理器的更多信息，参见第 49 章。

`CREATE LANGUAGE`命令有两种形式。第一种形式中，用户只提供所需语言的名称，PostgreSQL服务器会查询`pg_pltemplate`系统目录，以确定正确的参数。第二种形式中，用户在提供语言名称的同时也提供语言参数。第二种形式可用来创建未在`pg_pltemplate`中定义的语言，但这种做法被认为已经过时。

当服务器在`pg_pltemplate`目录中找到给定语言名称的条目时，即使命令包含语言参数，也会使用目录中的数据。这种行为简化了旧转储文件的装载，因为它们很可能包含关于语言支持函数的过时信息。

通常，用户必须具有PostgreSQL超级用户权限才能注册新语言。不过，如果语言列在`pg_pltemplate`目录中，并且标记为允许数据库所有者创建（`tmpldbacreate`为真），数据库所有者就可以在该数据库中注册这种新语言。默认情况下，数据库所有者可以创建受信任的语言，但超级用户可以通过修改`pg_pltemplate`的内容来调整这一点。语言的创建者成为其所有者，以后可以删除它、重命名它或将它分配给新的所有者。

`CREATE OR REPLACE LANGUAGE`会创建新语言，或替换现有定义。如果语言已经存在，则会根据指定值或从`pg_pltemplate`获取的值更新其参数，但不会改变语言的所有权和权限设置，并且假定用该语言编写的任何现有函数仍然有效。除了满足创建语言的常规权限要求外，用户还必须是超级用户或现有语言的所有者。`REPLACE`主要用于确保语言存在。如果语言具有`pg_pltemplate`条目，则`REPLACE`实际上不会改变现有定义的任何内容，除非出现少见的情况：该`pg_pltemplate`条目在语言创建后被修改过。

参数

TRUSTED

TRUSTED指定该语言不会让用户获得其原本无权访问的数据访问能力。
如果在注册该语言时省略这个关键字，只有拥有
PostgreSQL超级用户权限的用户才能使用该语言创建新函数。

PROCEDURAL

这是一个噪声词。

name

新过程语言的名称。该名称不能与数据库中其他语言的名称重复。

为了向后兼容，名称可以用单引号括起来。

HANDLER *call_handler*

call_handler 是一个先前注册的函数名称，它会被调用来执行该过程语言中的函数。过程语言的调用处理器必须使用某种编译型语言（例如 C）编写，并采用版本 1 调用约定。它必须在PostgreSQL 中注册为一个不带参数且返回language_handler类型的函数。该返回类型是一种占位符类型，仅用于把该函数标识为调用处理器。

INLINE *inline_handler*

inline_handler 是一个先前注册的函数名称，它会被调用来执行该语言中的匿名代码块（DO命令）。如果没有指定*inline_handler* 函数，则该语言不支持匿名代码块。该处理器函数必须接受一个 internal类型的参数，该参数将是 DO命令的内部表示，并且它通常返回 void。处理器的返回值会被忽略。

VALIDATOR *valfunction*

valfunction 是一个先前注册的函数名称，在创建该语言中的新函数时会调用它来验证该新函数。如果没有指定验证器函数，那么新函数在创建时不会被检查。
验证器函数必须接受一个oid类型的参数，它将是待创建函数的 OID，并且它通常返回 void。

验证器函数通常会检查函数体的语法正确性，但它也可以检查函数的其他属性，
例如该语言是否支持某些参数类型。要报告错误，验证器函数应使用ereport()函数。
函数的返回值会被忽略。

TRUSTED选项和支持函数名称会被忽略，前提是服务器在pg_pltemplate
中有指定语言名称的条目。

注解

createlang程序是对CREATE LANGUAGE命令的一个简单包装。它使从 shell 命令行安装过程语言变得更加容易。

使用DROP LANGUAGE删除过程语言，或者更好的方法是使用droplang程序。

系统目录pg_language（见第 45.27 节）记录当前已安装语言的信息。此外，createlang有一个可以列出已安装语言的选项。

要用某种过程语言创建函数，用户必须拥有该语言的 USAGE权限。默认情况下，对于受信任的语言，USAGE会授予PUBLIC（即所有人）。如果需要，也可以撤销这项授权。

过程语言是各个数据库本地的。不过，可以将某种语言安装到 `template1`数据库中，这样它就会在之后创建的所有数据库中自动可用。

如果服务器在`pg_pltemplate`中没有该语言的条目，则调用处理器函数、内联处理器函数（如果有）和验证器函数（如果有）必须已经存在。但如果有条目，则这些函数不必事先存在；如果数据库中不存在它们，就会自动定义它们。（如果安装中没有实现该语言的共享库，这可能导致`CREATE LANGUAGE`失败。）

在 7.3 之前的PostgreSQL版本中，必须将处理器函数声明为返回占位符类型`opaque`，而不是`language_handler`。为了支持装载旧转储文件，`CREATE LANGUAGE`会接受声明为返回`opaque`的函数，但会发出提示，并将函数声明的返回类型改为`language_handler`。

示例

创建任何标准过程语言的首选方式很简单：

```
CREATE LANGUAGE plperl;
```

对于 `pg_pltemplate` 目录中未知的语言，需要类似以下的命令序列：

```
CREATE FUNCTION plsample_call_handler() RETURNS language_handler
    AS '$libdir/plsample'
    LANGUAGE C;
CREATE LANGUAGE plsample
    HANDLER plsample_call_handler;
```

兼容性

`CREATE LANGUAGE`是一种 PostgreSQL扩展。

另见

ALTER LANGUAGE, CREATE FUNCTION, DROP LANGUAGE, GRANT, REVOKE, createlang, droplang

CREATE OPERATOR

CREATE OPERATOR — 定义一个新的操作符

大纲

```
CREATE OPERATOR name (
    PROCEDURE = function_name
    [, LEFTARG = left_type ] [, RIGHTARG = right_type ]
    [, COMMUTATOR = com_op ] [, NEGATOR = neg_op ]
    [, RESTRICT = res_proc ] [, JOIN = join_proc ]
    [, HASHES ] [, MERGES ]
)
```

描述

`CREATE OPERATOR`定义一个新的操作符 *name*。定义该操作符的用户将成为其拥有者。如果给出了模式名，该操作符将在指定的模式中创建；否则将在当前模式中创建。

操作符名称是一个字符序列，最多包含 `NAMEDATALEN`-1 个字符（默认为 63 个），字符取自以下列表：

+ - * / < > = ~ ! @ # % ^ & | ` ?

名称的选择有以下几项限制：

- `--`和`/*`不能出现在操作符名称的任何位置，因为它们会被视为注释的开始。
- 多字符操作符名称不能以`+`或`-` 结尾，除非该名称中还至少包含下列字符之一：

  ~ ! @ # % ^ & | ` ?

  例如，`@-`是允许的操作符名称，而 `*-`则不允许。这一限制使 PostgreSQL能够在不要求记号之间必须有空格的情况下解析符合 SQL 规范的命令。
- 使用`=>`作为操作符名称的做法已被弃用。未来版本可能完全禁止这种用法。

输入`!=`时会被映射为`<>`，因此这两个名称始终等效。

必须至少定义`LEFTARG`和`RIGHTARG`中的一个。对于二元操作符，必须同时定义两者。
对于右一元操作符，只应定义`LEFTARG`；对于左一元操作符，只应定义`RIGHTARG`。

*function_name*过程必须事先已用`CREATE FUNCTION`定义，并且其定义必须接受正确数量（一个或两个）的指定类型参数。

其他子句用于指定可选的操作符优化属性。其含义详见第 35.13 节。

要创建操作符，你必须在参数类型和返回类型上具有 `USAGE`权限，并在底层函数上具有`EXECUTE`权限。如果指定了交换子或求反器操作符，你还必须拥有这些操作符。

参数

name

要定义的操作符名称。允许使用的字符请见上文。该名称可以带模式限定，例如`CREATE OPERATOR myschema.+ (...)`。如果不带模式限定，该操作符将在当前模式中创建。同一模式中的两个操作符如果针对不同的数据类型进行操作，可以具有相同的名称。这被称为重载。

function_name

用于实现该操作符的函数。

left_type

该操作符左操作数的数据类型（如果有）。对于左一元操作符，应省略此选项。

right_type

该操作符右操作数的数据类型（如果有）。对于右一元操作符，应省略此选项。

com_op

该操作符的交换子。

neg_op

该操作符的求反器。

res_proc

该操作符的限制选择度估计函数。

join_proc

该操作符的连接选择度估算函数。

HASHES

表示该操作符可以支持哈希连接。

MERGES

表示该操作符可以支持归并连接。

要在*com_op*或其他可选参数中给出带模式限定的操作符名称，请使用OPERATOR()语法，例如：

```
COMMUTATOR = OPERATOR(myschema.===) ,
```

注解

更多信息请参见第 35.12 节。

无法在CREATE OPERATOR中指定一个操作符的词法优先级，因为解析器的优先级行为是硬写在代码中的。详见第 4.1.6 节。

废弃的选项SORT1、SORT2、LTCMP以及GTCMP
以前被用来指定与可归并连接的操作符相关联的排序操作符名称。现在已不再需要这样做，因为相关操作符的信息会通过查找 B-树操作符族来获得。如果给出这些选项中的任意一个，除了会隐式将MERGES设为真之外，其余都会被忽略。

使用DROP OPERATOR从数据库中删除用户定义的操作符。使用ALTER OPERATOR修改数据库中的操作符。

示例

以下命令定义一个新的面积相等操作符，适用的数据类型为 box：

```
CREATE OPERATOR === (
```

```
    LEFTARG = box,
    RIGHTARG = box,
    PROCEDURE = area_equal_procedure,
    COMMUTATOR = ===,
    NEGATOR = !==,
    RESTRICT = area_restriction_procedure,
    JOIN = area_join_procedure,
    HASHES, MERGES
);
```

兼容性

`CREATE OPERATOR`是 PostgreSQL扩展。在 SQL 标准中没有用户定义操作符的规定。

另见

ALTER OPERATOR, CREATE OPERATOR CLASS, DROP OPERATOR

CREATE OPERATOR CLASS

CREATE OPERATOR CLASS — 定义一个新的操作符类

大纲

```
CREATE OPERATOR CLASS name [ DEFAULT ] FOR TYPE data_type
  USING index_method [ FAMILY family_name ] AS
  {  OPERATOR strategy_number operator_name [ ( op_type, op_type )
 ] [ FOR SEARCH | FOR ORDER BY sort_family_name ]
   | FUNCTION support_number [ ( op_type [ , op_type ] ) ]
 function_name ( argument_type [, ...] )
   | STORAGE storage_type
  } [, ... ]
```

描述

`CREATE OPERATOR CLASS`创建一个新的操作符类。操作符类定义特定数据类型如何用于索引。它指定某些操作符为该数据类型和索引方法承担特定角色或“策略”。
它还指定为索引列选择该操作符类时，索引方法要使用的支持函数。在创建操作符类之前，必须先定义它使用的所有操作符和函数。

如果给出了一个模式名称，那么该操作符类会被创建在指定模式中。否则，
它会被创建在当前模式中。
同一模式中的两个操作符类只有在被用于不同的索引方法时才可以具有相同的名称。

定义操作符类的用户将成为其拥有者。目前，创建者必须是超级用户。（之所以有此限制，是因为错误的操作符类定义可能会使服务器混乱，甚至崩溃。）

`CREATE OPERATOR CLASS`当前不会检查操作符类定义是否包括该索引方法所要求的所有操作符和函数，也不会检查这些操作符和函数是否构成一个自洽的集合。
定义一个合法的操作符类是用户的责任。

相关的操作符类可以分组为操作符族。要把一个新的操作符类加入现有操作符族，可以在`CREATE OPERATOR CLASS`中指定`FAMILY`选项。如果没有这个选项，新类会被放入一个与其同名的操作符族中（如果该操作符族尚不存在，就会创建它）。

进一步的信息可参考第 35.14 节。

参数

name

要创建的操作符类的名称。该名称可以被模式限定。

`DEFAULT`

如果出现，该操作符类将成为其数据类型的默认操作符类。
对于某个特定的数据类型和索引方法，最多只能有一个默认操作符类。

data_type

该操作符类对应的列数据类型。

index_method

该操作符类对应的索引方法名称。

family_name

要把该操作符类加入其中的现有操作符族名称。如果没有指定，
则使用一个与该操作符类同名的操作符族（如果它还不存在则创建）。

strategy_number

与该操作符类关联的操作符在此索引方法中的策略号。

operator_name

与该操作符类关联的操作符名称（可以是模式限定的）。

op_type

在OPERATOR子句中，表示该操作符的操作数数据类型，或者用NONE
表示一个左一元或右一元操作符。在通常情况下，
如果操作数数据类型与操作符类的数据类型相同，则可以省略操作数数据类型。

在FUNCTION子句中，表示该函数意图支持的操作数数据类型；
如果它不同于函数的输入数据类型（对于 B-树比较函数和哈希函数），或者不同于该类的数据类型（对于 B-树排序支持函数，以及 GiST、SP-GiST 和 GIN 操作符类中的所有函数），则需要在这里指定。上述默认值都是正确的，因此无需将*op_type*
指定在FUNCTION子句中；唯一的例外是某个 B-树排序支持函数打算支持跨数据类型比较的情形。

sort_family_name

一个现有btree操作符族的名称（可以是模式限定的），
它描述与一种排序操作符相关联的排序顺序。

如果既没有指定FOR SEARCH，也没有指定 FOR ORDER BY，则默认值是FOR SEARCH。

support_number

与操作符类关联的函数在索引方法中的支持函数编号。

function_name

作为该操作符类的索引方法支持函数的函数名称（可带模式限定）。

argument_type

该函数的参数数据类型。

storage_type

实际存储在索引中的数据类型。通常它与列数据类型相同，但是有些索引方法（目前是 GiST 和 GIN）允许它们不同。除非索引方法允许使用不同类型，否则必须省略STORAGE子句。

OPERATOR、FUNCTION和STORAGE 子句可以以任何顺序出现。

注解

因为索引机制在使用函数之前不会检查函数的访问权限，
将一个函数或者操作符包括在一个操作符类中，相当于授予其公共执行权限。
对适合放入操作符类的那类函数而言，这通常不是问题。

操作符不应该由 SQL 函数定义。SQL 函数很有可能会被内联到调用查询中，这会妨碍优化器识别该查询匹配一个索引。

在PostgreSQL 8.4 之前，OPERATOR子句可以包含RECHECK选项。现在不再支持该选项，因为索引操作符是否“有损”会在运行时动态确定。
这样可以高效处理操作符可能有损也可能无损的情况。

示例

下面的示例为数据类型_int4（int4数组）定义了一个 GiST 索引操作符类。完整示例见intarray模块。

```
CREATE OPERATOR CLASS gist__int_ops
    DEFAULT FOR TYPE _int4 USING gist AS
        OPERATOR        3       &&,
        OPERATOR        6       = (anyarray, anyarray),
        OPERATOR        7       @>,
        OPERATOR        8       <@,
        OPERATOR        20      @@ (_int4, query_int),
        FUNCTION        1       g_int_consistent (internal, _int4,
 int, oid, internal),
        FUNCTION        2       g_int_union (internal, internal),
        FUNCTION        3       g_int_compress (internal),
        FUNCTION        4       g_int_decompress (internal),
        FUNCTION        5       g_int_penalty (internal, internal,
 internal),
        FUNCTION        6       g_int_picksplit (internal,
 internal),
        FUNCTION        7       g_int_same (_int4, _int4, internal)
;
```

兼容性

CREATE OPERATOR CLASS是一种 PostgreSQL扩展。在 SQL 标准中没有 CREATE OPERATOR CLASS语句。

另见

ALTER OPERATOR CLASS, DROP OPERATOR CLASS, CREATE OPERATOR FAMILY, ALTER OPERATOR FAMILY

CREATE OPERATOR FAMILY

CREATE OPERATOR FAMILY — 定义一个新的操作符族

大纲

```
CREATE OPERATOR FAMILY name USING index_method
```

描述

`CREATE OPERATOR FAMILY`创建一个新的操作符族。操作符族定义了一组相关的操作符类，以及可能还有一些与这些操作符类兼容、
但对任何单个索引的正常工作都不是必需的额外操作符和支持函数。
（那些对索引必不可少的操作符和函数应归入相应的操作符类，
而不是作为操作符族中的“松散”成员。通常，单一数据类型的操作符属于操作符类，
而跨数据类型的操作符则可以作为操作符族中的松散成员，
该操作符族包含这两种数据类型对应的操作符类。）

新的操作符族最初是空的。应当通过随后发出的`CREATE OPERATOR CLASS`命令向其中添加所包含的操作符类，并可选择通过`ALTER OPERATOR FAMILY`命令添加“松散”操作符及其对应的支持函数。

如果给出了模式名称，该操作符族会被创建在指定的模式中。否则，
它会被创建在当前模式中。只有当同一模式中的两个操作符族用于不同的索引方法时，
它们才能具有相同的名称。

定义操作符族的用户将成为其拥有者。当前，创建该对象的用户必须是超级用户。
（之所以这样限制，是因为错误的操作符族定义可能会使服务器混乱，甚至导致崩溃。）

更多信息请参见第 35.14 节。

参数

name

要创建的操作符族名称。该名称可以是模式限定的。

index_method

该操作符族所对应的索引方法名称。

兼容性

`CREATE OPERATOR FAMILY`是一种 PostgreSQL扩展。SQL 标准中没有`CREATE OPERATOR FAMILY`语句。

另见

ALTER OPERATOR FAMILY, DROP OPERATOR FAMILY, CREATE OPERATOR CLASS, ALTER OPERATOR CLASS, DROP OPERATOR CLASS

CREATE ROLE

CREATE ROLE — 定义一个新的数据库角色

大纲

```
CREATE ROLE name [ [ WITH ] option [ ... ] ]
```

其中*option*可以是：

```
      SUPERUSER | NOSUPERUSER
    | CREATEDB | NOCREATEDB
    | CREATEROLE | NOCREATEROLE
    | CREATEUSER | NOCREATEUSER
    | INHERIT | NOINHERIT
    | LOGIN | NOLOGIN
    | REPLICATION | NOREPLICATION
    | CONNECTION LIMIT connlimit
    | [ ENCRYPTED | UNENCRYPTED ] PASSWORD 'password'
    | VALID UNTIL 'timestamp'
    | IN ROLE role_name [, ...]
    | IN GROUP role_name [, ...]
    | ROLE role_name [, ...]
    | ADMIN role_name [, ...]
    | USER role_name [, ...]
    | SYSID uid
```

描述

`CREATE ROLE`向 PostgreSQL数据库集簇中新增一个角色。角色是一种可以拥有数据库对象并拥有数据库权限的实体；根据其使用方式，它既可以视为“用户”，也可以视为“组”，或者兼具这两种身份。有关用户管理和认证的信息，请参见第 20 章和第 19 章。要使用该命令，你必须拥有 `CREATEROLE`权限，或者是数据库超级用户。

注意，角色是在数据库集簇级别定义的，因此在该集簇中的所有数据库里都有效。

参数

name

新角色的名称。

`SUPERUSER`
`NOSUPERUSER`

这些子句决定新角色是否为“超级用户”；超级用户可以绕过数据库中的所有访问限制。超级用户身份具有危险性，只有在确有需要时才应使用。要创建新的超级用户，你自己必须先是超级用户。若未指定，默认值是 `NOSUPERUSER`。

`CREATEDB`
`NOCREATEDB`

这些子句定义角色创建数据库的能力。如果指定`CREATEDB`，则允许正在定义的角色创建新数据库。指定`NOCREATEDB`会禁止角色创建数据库。如果未指定，默认值为`NOCREATEDB`。

CREATEROLE
NOCREATEROLE

这些子句决定是否允许角色创建新角色（即执行CREATE ROLE）。具有CREATEROLE权限的角色还可以更改和删除其他角色。如果未指定，默认值为NOCREATEROLE。

CREATEUSER
NOCREATEUSER

这些子句是 SUPERUSER 和 NOSUPERUSER 的一种已过时但仍被接受的写法。注意，它们不等价于 CREATEROLE，尽管可能有人会天真地这么认为！

INHERIT
NOINHERIT

这些子句决定角色是否“继承”其所属角色的权限。具有INHERIT属性的角色可以自动使用直接或间接所属的所有角色获得的任何数据库权限。没有INHERIT时，成为另一个角色的成员只授予对该角色执行SET ROLE的能力；另一个角色的权限只有在执行该操作后才可用。如果未指定，默认值为INHERIT。

LOGIN
NOLOGIN

这些子句决定是否允许角色登录；也就是说，在客户端连接期间，是否可以将该角色作为初始会话授权名称。具有LOGIN属性的角色可以视为用户。没有该属性的角色可用于管理数据库权限，但通常意义上不属于用户。如果未指定，默认值为NOLOGIN，但通过其另一种拼写CREATE USER调用CREATE ROLE时除外。

REPLICATION
NOREPLICATION

这些子句决定是否允许角色发起流复制，以及让系统进入和退出备份模式。具有REPLICATION属性的角色权限非常高，因此该属性只应赋予实际用于复制的角色。如果未指定，默认值为NOREPLICATION。

CONNECTION LIMIT *connlimit*

如果该角色可以登录，此项指定该角色可建立多少并发连接。-1（默认值）表示不限制。注意，只有普通连接会计入此限制。

PASSWORD *password*

设置角色的密码。（密码只对具有LOGIN属性的角色有用，但即使没有该属性，你仍然可以为角色定义密码。）如果你不打算使用密码认证，可以省略此选项。如果未指定密码，密码将被设为 null，并且该用户的密码认证将始终失败。也可以显式写为PASSWORD NULL。

ENCRYPTED
UNENCRYPTED

这些关键字控制密码是否以加密形式存储在系统目录中。（如果两者都未指定，默认行为由配置参数password_encryption决定。）如果给出的密码字符串已经是 MD5 加密格式，那么无论指定的是ENCRYPTED还是 UNENCRYPTED，它都会按原样以加密形式存储（因为系统无法解密指定的已加密密码字符串）。这样便可在转储/恢复期间重新装载已加密的密码。

VALID UNTIL'*timestamp*'

VALID UNTIL子句设置一个日期和时间，在此之后角色的密码将不再有效。如果省略此子句，密码将始终有效。

`IN ROLE` *`role_name`*

`IN ROLE`子句列出一个或多个现有角色，新角色会立即作为新成员加入这些角色。（请注意，无法通过此选项将新角色添加为管理员；要实现这一点，请使用单独的`GRANT`命令。）

`IN GROUP` *`role_name`*

`IN GROUP`是`IN ROLE`的已废弃写法。

`ROLE` *`role_name`*

`ROLE`子句列出一个或多个现有角色，这些角色会自动作为新角色的成员加入。（实际上，这会使新角色成为一个“组”。）

`ADMIN` *`role_name`*

`ADMIN`子句类似于`ROLE`，但指定的角色会以`WITH ADMIN OPTION`加入新角色，从而获得向其他人授予该角色成员资格的权利。

`USER` *`role_name`*

`USER`子句是`ROLE`子句的已废弃写法。

`SYSID` *`uid`*

`SYSID`子句会被忽略，但为向后兼容而接受。

注解

使用ALTER ROLE更改角色属性，并使用DROP ROLE删除角色。`CREATE ROLE`指定的所有属性都可以由后续的`ALTER ROLE`命令修改。

向作为组使用的角色添加和移除成员，推荐使用GRANT和REVOKE。

`VALID UNTIL`子句只为密码定义过期时间，而不是为角色本身定义过期时间。特别是，使用非基于密码的认证方式登录时，不会强制执行该过期时间。

`INHERIT`属性控制可授予权限的继承（即数据库对象的访问权限和角色成员资格）。它不适用于`CREATE ROLE`和`ALTER ROLE`设置的特殊角色属性。例如，即使设置了`INHERIT`，成为拥有`CREATEDB`权限的角色成员也不会立即获得创建数据库的能力；在创建数据库之前，必须先通过SET ROLE成为该角色。

`INHERIT`属性出于向后兼容的原因而成为默认值：在PostgreSQL的早期版本中，用户总能访问其所属组的所有权限。不过，`NOINHERIT`更接近 SQL 标准规定的语义。

请谨慎对待`CREATEROLE`权限。`CREATEROLE`角色的权限不存在继承这一概念。这意味着，即使某个角色没有特定权限但被允许创建其他角色，它仍可轻易创建一个权限不同于自身的角色（创建具有超级用户权限的角色除外）。例如，如果角色“user”具有`CREATEROLE`权限但没有`CREATEDB`权限，它仍然可以创建一个具有`CREATEDB`权限的新角色。因此，应将拥有`CREATEROLE`权限的角色视为几乎等同超级用户的角色。

PostgreSQL包含一个程序createuser，其功能与`CREATE ROLE`相同（实际上它就是调用此命令），但它可以从命令 shell 中运行。

`CONNECTION LIMIT`选项只是近似地被强制执行；如果两个新会话在几乎同一时刻启动，而此时该角色只剩下一个连接“槽”，则两者都可能失败。此外，该限制从不对超级用户强制执行。

用此命令指定未加密密码时必须谨慎。密码会以明文形式传输到服务器，
也可能被记录在客户端命令历史或服务器日志中。而命令createuser传输的是加密密码。
此外，psql包含 `\password`命令，可用于稍后安全地更改密码。

示例

创建一个可登录但没有密码的角色：

```
CREATE ROLE jonathan LOGIN;
```

创建一个带密码的角色：

```
CREATE USER davide WITH PASSWORD 'jw8s0F4';
```

（`CREATE USER`与`CREATE ROLE`相同，只是它隐含指定了`LOGIN`。）

创建一个密码有效期截止到 2004 年底的角色。当 2005 年的第一秒过去后，该密码就不再有效。

```
CREATE ROLE miriam WITH LOGIN PASSWORD 'jw8s0F4' VALID UNTIL '2005
-01-01';
```

创建一个可以创建数据库并管理角色的角色：

```
CREATE ROLE admin WITH CREATEDB CREATEROLE;
```

兼容性

该`CREATE ROLE`语句在 SQL 标准中有定义，但该标准只要求以下语法

```
CREATE ROLE name [ WITH ADMIN role_name ]
```

多个初始管理员，以及`CREATE ROLE`的所有其他选项，都是PostgreSQL扩展。

SQL　标准定义了用户和角色的概念，但将它们视为不同的概念，并将定义用户的所有命令留给各数据库实现自行规定。在PostgreSQL中，我们选择将用户和角色统一为一种实体。
因此，与标准相比，角色具有更多可选属性。

将用户赋予 `NOINHERIT` 属性，而将角色赋予 `INHERIT` 属性，最接近 SQL 标准规定的行为。

参见

SET ROLE, ALTER ROLE, DROP ROLE, GRANT, REVOKE, createuser

CREATE RULE

CREATE RULE — 定义一条新的重写规则

大纲

```
CREATE [ OR REPLACE ] RULE name AS ON event
    TO table_name [ WHERE condition ]
    DO [ ALSO | INSTEAD ] { NOTHING | command | ( command ; command
 ... ) }
```

描述

CREATE RULE定义一条应用于指定表或视图的新规则。CREATE OR REPLACE RULE将创建一条新规则，或者替换同一个表上同名的现有规则。

PostgreSQL规则系统允许定义在数据库表上执行插入、更新或删除时的替代动作。
粗略地说，当在某个给定表上执行某个命令时，一条规则会导致额外的命令被执行。
另一种情况是，INSTEAD 规则可以用另一个命令替换给定命令，或者让某个命令完全不执行。规则也用于实现 SQL 视图。必须认识到，规则实际上是一种命令转换机制，或者说是命令宏。这种转换发生在命令开始执行之前。
如果你真正需要的是对每个物理行独立触发的操作，那么你很可能应该使用触发器，
而不是规则。有关规则系统的更多信息见第 37 章。

目前，ON SELECT规则必须是无条件的INSTEAD规则，其动作必须由单个SELECT命令组成。因此，ON SELECT规则实际上会把表变为视图，其可见内容是该规则的SELECT命令返回的行，而不是表中原来存储的内容（如果有）。直接编写CREATE VIEW命令，被认为比创建真实表并为其定义ON SELECT规则的风格更好。

可以通过定义ON INSERT、ON UPDATE 和ON DELETE规则（或其中足以满足你需求的任意子集），来营造出可更新视图的效果，
从而将视图上的更新动作替换为对其他表的适当更新。如果想支持INSERT RETURNING等功能，那么务必在这些规则中的每一条里都放入合适的RETURNING子句。或者，也可以使用INSTEAD OF触发器（见CREATE TRIGGER）来实现可更新视图。

如果你尝试对视图更新使用有条件规则，
这里有一个陷阱：对于你希望允许在该视图上执行的每一种动作，必须都有一条无条件的INSTEAD规则。如果该规则是有条件的，或者不是 INSTEAD规则，那么系统仍会拒绝执行该更新动作，因为它认为在某些情况下最终可能还是会尝试在该视图的虚拟表上执行该动作。
如果你想用有条件规则处理所有有用的情况，可以加上一条无条件的 DO INSTEAD NOTHING规则，以确保系统明白它永远不会被要求去更新这个虚拟表。然后，
把这些有条件规则改成非 INSTEAD规则；在它们适用的情况下，它们会附加在默认的INSTEAD NOTHING动作之上。（不过，这种方法当前仍无法支持 RETURNING查询。）

参数

name

要创建的规则的名称。它必须与同一个表上任何其他规则的名称相区分。
同一个表上同一种事件类型的多条规则会按名称的字母顺序应用。

event

事件是SELECT、INSERT、UPDATE或者 DELETE之一。

table_name

规则适用的表或者视图的名称（可以是模式限定的）。

condition

任意的SQL条件表达式（返回 boolean）。条件表达式不能引用NEW和 OLD之外的任何表，并且不能包含聚合函数。

INSTEAD

INSTEAD表示这些命令应当替代原始命令执行。

ALSO

ALSO表示这些命令应当在原始命令之外执行。

如果既未指定ALSO，也未指定 INSTEAD，则默认值为ALSO。

command

组成规则动作的命令。可用的命令有SELECT、INSERT、UPDATE、DELETE或者NOTIFY。

在*condition*和 *command*中，可以使用特殊表名 NEW和OLD来引用目标表中的值。在ON INSERT和ON UPDATE规则中，NEW可用于引用待插入或待更新的新行。在 ON UPDATE和ON DELETE规则中，OLD可用于引用将被更新或删除的现有行。

注解

要在表上创建或者修改规则，必须是表的拥有者。

在视图上的INSERT、UPDATE或 DELETE规则中，可以添加一个输出视图各列的 RETURNING子句。如果该规则分别由 INSERT RETURNING、UPDATE RETURNING 或DELETE RETURNING命令触发，那么这个子句将用于计算输出。当规则由不带RETURNING的命令触发时，该规则的 RETURNING子句会被忽略。当前实现只允许无条件的 INSTEAD规则包含RETURNING；此外，对于同一事件的所有规则，总共最多只能有一个RETURNING子句。（这可以确保只有一个候选RETURNING子句可用于计算结果。）
如果任何可用规则中都没有 RETURNING子句，那么视图上的RETURNING查询将被拒绝。

避免循环规则非常重要。例如，尽管下面的两条规则定义都被 PostgreSQL所接受，SELECT命令将导致 PostgreSQL报告错误，因为规则会发生递归展开：

```
CREATE RULE "_RETURN" AS
    ON SELECT TO t1
    DO INSTEAD
        SELECT * FROM t2;

CREATE RULE "_RETURN" AS
    ON SELECT TO t2
    DO INSTEAD
        SELECT * FROM t1;

SELECT * FROM t1;
```

当前，如果一个规则动作包含一个NOTIFY命令，该NOTIFY命令将被无条件执行，
也就是说，即使没有任何应当应用该规则的行，也会发出NOTIFY。例如：

```
CREATE RULE notify_me AS ON UPDATE TO mytable DO ALSO NOTIFY
 mytable;

UPDATE mytable SET name = 'foo' WHERE id = 42;
```

在UPDATE期间会发送一个NOTIFY事件，无论是否存在匹配条件id = 42的行。这是一个实现限制，未来版本中可能会修复。

兼容性

`CREATE RULE`是一种 PostgreSQL语言扩展，整个查询重写系统也是如此。

CREATE SCHEMA

CREATE SCHEMA — 定义一个新模式

大纲

```
CREATE SCHEMA schema_name [ AUTHORIZATION user_name ] [ schema_
element [ ... ] ]
CREATE SCHEMA AUTHORIZATION user_name [ schema_element [ ... ] ]
```

描述

CREATE SCHEMA在当前数据库中创建一个新模式。该模式名必须与当前数据库中任何现有模式的名称不同。

模式本质上是一个命名空间：它包含具名对象（表、数据类型、函数和操作符），这些对象的名称可以与其他模式中的对象重名。访问具名对象时，可以在其名称前加上模式名作为前缀来“限定”名称，或者设置一个包含所需模式的搜索路径。指定非限定对象名的CREATE命令会在当前模式中创建该对象（即搜索路径最前面的模式，可通过函数current_schema确定）。

CREATE SCHEMA还可以选择包含子命令，以便在新模式中创建对象。这些子命令基本上会被当作在创建模式之后单独发出的命令来处理。不过，如果使用了AUTHORIZATION子句，则所有创建的对象都将归该用户所有。

参数

schema_name

要创建的模式名称。如果省略，*user_name*将被用作模式名。该名称不能以pg_开头，因为这类名称保留给系统模式。

user_name

将拥有新模式的用户的角色名。如果省略，则默认为执行该命令的用户。要创建由另一个角色拥有的模式，你必须是该角色的直接或间接成员，或者是超级用户。

schema_element

定义要在该模式中创建的对象的 SQL 语句。当前，只有CREATE TABLE、CREATE VIEW、CREATE INDEX、CREATE SEQUENCE、CREATE TRIGGER和GRANT可作为CREATE SCHEMA中的子句使用。其他类型的对象可以在模式创建之后通过单独的命令创建。

注解

要创建一个模式，执行该命令的用户必须拥有当前数据库的CREATE 权限（当然，超级用户可以绕过这项检查）。

示例

创建一个模式：

```
CREATE SCHEMA myschema;
```

为用户joe创建一个模式，该模式也将被命名为 joe：

```
CREATE SCHEMA AUTHORIZATION joe;
```

创建一个模式并且在其中创建一个表和视图：

```
CREATE SCHEMA hollywood
    CREATE TABLE films (title text, release date, awards text[])
    CREATE VIEW winners AS
        SELECT title, release FROM films WHERE awards IS NOT NULL;
```

请注意，各个子命令末尾都不带分号。

下面是实现相同结果的一种等效写法：

```
CREATE SCHEMA hollywood;
CREATE TABLE hollywood.films (title text, release date, awards
 text[]);
CREATE VIEW hollywood.winners AS
    SELECT title, release FROM hollywood.films WHERE awards IS NOT
 NULL;
```

兼容性

SQL 标准允许在CREATE SCHEMA中使用 DEFAULT CHARACTER SET子句，并允许使用比PostgreSQL当前所接受的更多子命令类型。

SQL 标准规定，CREATE SCHEMA中的子命令可以按任意顺序出现。当前的PostgreSQL实现并不能处理子命令中所有前向引用的情况；有时可能需要重新排列子命令的顺序，以避免前向引用。

根据 SQL 标准，模式的拥有者总是拥有其中的所有对象。PostgreSQL允许模式包含由模式拥有者之外的用户所拥有的对象。只有当模式拥有者把其模式上的CREATE权限授予其他人，或者超级用户选择在该模式中创建对象时，才会发生这种情况。

另见

ALTER SCHEMA, DROP SCHEMA

CREATE SEQUENCE

CREATE SEQUENCE — 定义一个新的序列发生器

大纲

```
CREATE [ TEMPORARY | TEMP ] SEQUENCE name [ INCREMENT [ BY ]
 increment ]
    [ MINVALUE minvalue | NO MINVALUE ] [ MAXVALUE maxvalue | NO
 MAXVALUE ]
    [ START [ WITH ] start ] [ CACHE cache ] [ [ NO ] CYCLE ]
    [ OWNED BY { table_name.column_name | NONE } ]
```

描述

`CREATE SEQUENCE`创建一个新的序列号发生器。这会创建并初始化一个名为*name*的特殊单行表。该发生器归发出该命令的用户所有。

如果给定了模式名，则序列会在指定模式中创建。否则，它会在当前模式中创建。
临时序列存在于一个特殊模式中，因此创建临时序列时不能指定模式名。
序列名必须不同于同一模式中任何其他关系（表、序列、索引、视图、物化视图或外部表）的名称。

创建序列后，可以使用函数 `nextval`、`currval`以及 `setval`来操作该序列。这些函数的文档见第 9.16 节。

虽然不能直接更新序列，但可以使用如下查询：

```
SELECT * FROM name;
```

来查看序列的参数和当前状态。特别是，序列的 `last_value`字段会显示任一会话最近分配的值。（当然，如果其他会话正在调用`nextval`，那么到该值被打印出来时，
它可能已经过时。）

参数

`TEMPORARY` 或 `TEMP`

如果指定该项，则该序列对象只为当前会话创建，并会在会话退出时自动删除。
临时序列存在期间，已有的同名永久序列在本会话中不可见，
除非使用模式限定名来引用它们。

name

要创建的序列的名称（可以是模式限定的）。

increment

可选子句`INCREMENT BY` *increment*指定加到当前序列值上以生成新值的增量。正值生成升序序列，负值生成降序序列。默认值为 1。

minvalue
`NO MINVALUE`

可选子句`MINVALUE` *minvalue*确定序列可生成的最小值。如果未提供此子句，或者指定了 `NO MINVALUE`，则使用默认值。升序序列和降序序列的默认值分别为 1 和 $-2^{63}-1$。

maxvalue
NO MAXVALUE

可选子句MAXVALUE *maxvalue*确定序列的最大值。如果未提供此子句，或者指定了 NO MAXVALUE，则使用默认值。升序序列和降序序列的默认值分别为 2^{63}-1 和 -1。

start

可选子句START WITH *start* 允许序列从任意值开始。升序序列的默认起始值为*minvalue*，降序序列则为*maxvalue*。

cache

可选子句CACHE *cache*指定要预分配并存储在内存中以加快访问的序列号数量。最小值为 1（一次只能生成一个值，即不缓存），这也是默认值。

CYCLE
NO CYCLE

CYCLE选项允许序列在升序序列达到*maxvalue*、或降序序列达到*minvalue*时分别回绕。如果达到该限制，下一个生成的数字将分别是*minvalue*或*maxvalue*。

如果指定NO CYCLE，则当序列达到其限值后，任何对 nextval的调用都会返回错误。如果既未指定 CYCLE也未指定NO CYCLE，则默认使用NO CYCLE。

OWNED BY *table_name.column_name*
OWNED BY NONE

OWNED BY选项使序列与特定表列关联，从而在删除该列（或整个表）时，也自动删除该序列。指定表必须与序列具有相同的所有者，并且位于同一模式中。默认值OWNED BY NONE表示不存在这种关联。

注解

使用DROP SEQUENCE删除序列。

序列基于bigint算术，因此其范围不能超过八字节整数的范围（-9223372036854775808 到 9223372036854775807）。

如果对一个将由多个会话并发使用的序列对象设置了大于 1 的*cache*，则可能得到意想不到的结果。每个会话会在一次访问该序列对象时分配并缓存连续的序列值，并相应增加该序列对象的last_value。随后，该会话中接下来的 *cache*-1 次nextval 调用会直接返回预分配的值，而不接触序列对象。因此，任何在会话中已分配但未使用的数字都会在会话结束时丢失，从而在序列中造成“空洞”。

此外，虽然多个会话保证会分配到互不相同的序列值，但从所有会话整体来看，这些值的生成顺序可能会乱序。例如，当*cache*设置为 10 时，会话 A 可能保留值 1..10 并返回nextval=1；然后会话 B 可能保留值 11..20 并返回nextval=11，而此时会话 A 尚未生成 nextval=2。因此，当*cache*设置为 1 时，可以安全地假定 nextval值按顺序生成；当*cache*设置大于 1 时，只应假定 nextval值彼此不同，而不应假定它们严格按顺序生成。此外，last_value会反映任一会话最近保留的值，不管该值是否已经由nextval返回。

另一个需要注意的问题是，在这种序列上执行的setval 不会立刻被其他会话察觉，直到它们用完自己缓存的所有预分配值为止。

示例

创建一个名为serial的升序序列，从 101 开始：

```
CREATE SEQUENCE serial START 101;
```

从该序列中取出下一个数字：

```
SELECT nextval('serial');

 nextval
---------
     101
```

从该序列中取出下一个数字：

```
SELECT nextval('serial');

 nextval
---------
     102
```

在INSERT命令中使用该序列：

```
INSERT INTO distributors VALUES (nextval('serial'), 'nothing');
```

在一次COPY FROM之后更新序列值：

```
BEGIN;
COPY distributors FROM 'input_file';
SELECT setval('serial', max(id)) FROM distributors;
END;
```

兼容性

CREATE SEQUENCE符合SQL 标准，但下列情况除外：

- 不支持标准的AS <data type>表达式。
- 获取下一个值是使用nextval()函数，而不是标准的 NEXT VALUE FOR表达式完成的。
- OWNED BY子句是PostgreSQL扩展。

另见

ALTER SEQUENCE, DROP SEQUENCE

CREATE SERVER

CREATE SERVER — 定义一个新的外部服务器

大纲

```
CREATE SERVER server_name [ TYPE 'server_type' ] [ VERSION 'server_
version' ]
    FOREIGN DATA WRAPPER fdw_name
    [ OPTIONS ( option 'value' [, ... ] ) ]
```

描述

`CREATE SERVER`定义一个新的外部服务器。定义该服务器的用户将成为其拥有者。

外部服务器通常封装外部数据包装器用来访问外部数据资源的连接信息。
还可以通过用户映射指定额外的、特定于用户的连接信息。

服务器名称在数据库内必须唯一。

要创建服务器，必须在所使用的外部数据包装器上具有`USAGE`权限。

参数

server_name

要创建的外部服务器的名称。

server_type

可选的服务器类型。

server_version

可选的服务器版本。

fdw_name

管理该服务器的外部数据包装器的名称。

`OPTIONS ( option 'value' [, ... ] )`

这个子句指定服务器的选项。这些选项通常定义该服务器的连接细节，
但实际的名称和值取决于该服务器的外部数据包装器。

注解

在使用 dblink 模块（见dblink）时，可以将外部服务器的名称用作dblink_connect函数的一个参数，以指示连接参数。那里还有更多示例。要以这种方式使用它，
必须在该外部服务器上具有 `USAGE`权限。

示例

创建一个使用内建外部数据包装器`default`的服务器`foo`：

```
CREATE SERVER foo FOREIGN DATA WRAPPER "default";
```

创建一个使用外部数据包装器`pgsql`的服务器`myserver`：

```
CREATE SERVER myserver FOREIGN DATA WRAPPER pgsql OPTIONS (host
 'foo', dbname 'foodb', port '5432');
```

兼容性

`CREATE SERVER`符合 ISO/IEC 9075-9 (SQL/MED)。

另见

ALTER SERVER, DROP SERVER, CREATE FOREIGN DATA WRAPPER, CREATE USER MAPPING

CREATE TABLE

CREATE TABLE — define a new table

大纲

```
CREATE [ [ GLOBAL | LOCAL ] { TEMPORARY | TEMP } | UNLOGGED ] TABLE
 [ IF NOT EXISTS ] table_name ( [
  { column_name data_type [ COLLATE collation ] [ column_constraint
 [ ... ] ]
    | table_constraint
    | LIKE source_table [ like_option ... ] }
    [, ... ]
] )
[ INHERITS ( parent_table [, ... ] ) ]
[ WITH ( storage_parameter [= value] [, ... ] ) | WITH OIDS |
 WITHOUT OIDS ]
[ ON COMMIT { PRESERVE ROWS | DELETE ROWS | DROP } ]
[ TABLESPACE tablespace_name ]

CREATE [ [ GLOBAL | LOCAL ] { TEMPORARY | TEMP } | UNLOGGED ] TABLE
 [ IF NOT EXISTS ] table_name
    OF type_name [ (
  { column_name WITH OPTIONS [ column_constraint [ ... ] ]
    | table_constraint }
    [, ... ]
) ]
[ WITH ( storage_parameter [= value] [, ... ] ) | WITH OIDS |
 WITHOUT OIDS ]
[ ON COMMIT { PRESERVE ROWS | DELETE ROWS | DROP } ]
[ TABLESPACE tablespace_name ]

where column_constraint is:

[ CONSTRAINT constraint_name ]
{ NOT NULL |
  NULL |
  CHECK ( expression ) [ NO INHERIT ] |
  DEFAULT default_expr |
  UNIQUE index_parameters |
  PRIMARY KEY index_parameters |
  REFERENCES reftable [ ( refcolumn ) ] [ MATCH FULL | MATCH
 PARTIAL | MATCH SIMPLE ]
    [ ON DELETE action ] [ ON UPDATE action ] }
[ DEFERRABLE | NOT DEFERRABLE ] [ INITIALLY DEFERRED | INITIALLY
 IMMEDIATE ]

and table_constraint is:

[ CONSTRAINT constraint_name ]
{ CHECK ( expression ) [ NO INHERIT ] |
  UNIQUE ( column_name [, ... ] ) index_parameters |
  PRIMARY KEY ( column_name [, ... ] ) index_parameters |
  EXCLUDE [ USING index_method ] ( exclude_element WITH operator [,
 ... ] ) index_parameters [ WHERE ( predicate ) ] |
```

```
  FOREIGN KEY ( column_name [, ... ] ) REFERENCES reftable [ (
 refcolumn [, ... ] ) ]
    [ MATCH FULL | MATCH PARTIAL | MATCH SIMPLE ] [ ON
 DELETE action ] [ ON UPDATE action ] }
[ DEFERRABLE | NOT DEFERRABLE ] [ INITIALLY DEFERRED | INITIALLY
 IMMEDIATE ]

and like_option is:

{ INCLUDING | EXCLUDING } { DEFAULTS | CONSTRAINTS | INDEXES |
 STORAGE | COMMENTS | ALL }

index_parameters in UNIQUE, PRIMARY KEY, and EXCLUDE constraints
 are:

[ WITH ( storage_parameter [= value] [, ... ] ) ]
[ USING INDEX TABLESPACE tablespace_name ]

exclude_element in an EXCLUDE constraint is:

{ column_name | ( expression ) } [ opclass ] [ ASC | DESC ] [ NULLS
 { FIRST | LAST } ]
```

Description

`CREATE TABLE` will create a new, initially empty table in the current database. The table will be owned by the user issuing the command.

If a schema name is given (for example, `CREATE TABLE myschema.mytable ...`) then the table is created in the specified schema. Otherwise it is created in the current schema. Temporary tables exist in a special schema, so a schema name cannot be given when creating a temporary table. The name of the table must be distinct from the name of any other table, sequence, index, view, or foreign table in the same schema.

`CREATE TABLE` also automatically creates a data type that represents the composite type corresponding to one row of the table. Therefore, tables cannot have the same name as any existing data type in the same schema.

The optional constraint clauses specify constraints (tests) that new or updated rows must satisfy for an insert or update operation to succeed. A constraint is an SQL object that helps define the set of valid values in the table in various ways.

There are two ways to define constraints: table constraints and column constraints. A column constraint is defined as part of a column definition. A table constraint definition is not tied to a particular column, and it can encompass more than one column. Every column constraint can also be written as a table constraint; a column constraint is only a notational convenience for use when the constraint only affects one column.

To be able to create a table, you must have `USAGE` privilege on all column types or the type in the `OF` clause, respectively.

Parameters

`TEMPORARY` or `TEMP`

If specified, the table is created as a temporary table. Temporary tables are automatically dropped at the end of a session, or optionally at the end of the current transaction (see `ON COMMIT` below). Existing permanent tables with the same name are not visible to the current session while the temporary table exists,

unless they are referenced with schema-qualified names. Any indexes created on a temporary table are automatically temporary as well.

The autovacuum daemon cannot access and therefore cannot vacuum or analyze temporary tables. For this reason, appropriate vacuum and analyze operations should be performed via session SQL commands. For example, if a temporary table is going to be used in complex queries, it is wise to run `ANALYZE` on the temporary table after it is populated.

Optionally, `GLOBAL` or `LOCAL` can be written before `TEMPORARY` or `TEMP`. This presently makes no difference in PostgreSQL and is deprecated; see Compatibility.

`UNLOGGED`

If specified, the table is created as an unlogged table. Data written to unlogged tables is not written to the write-ahead log (see 第 29 章), which makes them considerably faster than ordinary tables. However, they are not crash-safe: an unlogged table is automatically truncated after a crash or unclean shutdown. The contents of an unlogged table are also not replicated to standby servers. Any indexes created on an unlogged table are automatically unlogged as well; however, unlogged GiST indexes are currently not supported and cannot be created on an unlogged table.

`IF NOT EXISTS`

Do not throw an error if a relation with the same name already exists. A notice is issued in this case. Note that there is no guarantee that the existing relation is anything like the one that would have been created.

`table_name`

The name (optionally schema-qualified) of the table to be created.

`OF` *`type_name`*

Creates a typed table, which takes its structure from the specified composite type (name optionally schema-qualified). A typed table is tied to its type; for example the table will be dropped if the type is dropped (with `DROP TYPE ... CASCADE`).

When a typed table is created, then the data types of the columns are determined by the underlying composite type and are not specified by the `CREATE TABLE` command. But the `CREATE TABLE` command can add defaults and constraints to the table and can specify storage parameters.

`column_name`

The name of a column to be created in the new table.

`data_type`

The data type of the column. This can include array specifiers. For more information on the data types supported by PostgreSQL, refer to 第 8 章.

`COLLATE` *`collation`*

The `COLLATE` clause assigns a collation to the column (which must be of a collatable data type). If not specified, the column data type's default collation is used.

`INHERITS (` *`parent_table`* `[, ... ] )`

The optional `INHERITS` clause specifies a list of tables from which the new table automatically inherits all columns.

Use of INHERITS creates a persistent relationship between the new child table and its parent table(s). Schema modifications to the parent(s) normally propagate to children as well, and by default the data of the child table is included in scans of the parent(s).

If the same column name exists in more than one parent table, an error is reported unless the data types of the columns match in each of the parent tables. If there is no conflict, then the duplicate columns are merged to form a single column in the new table. If the column name list of the new table contains a column name that is also inherited, the data type must likewise match the inherited column(s), and the column definitions are merged into one. If the new table explicitly specifies a default value for the column, this default overrides any defaults from inherited declarations of the column. Otherwise, any parents that specify default values for the column must all specify the same default, or an error will be reported.

CHECK constraints are merged in essentially the same way as columns: if multiple parent tables and/or the new table definition contain identically-named CHECK constraints, these constraints must all have the same check expression, or an error will be reported. Constraints having the same name and expression will be merged into one copy. A constraint marked NO INHERIT in a parent will not be considered. Notice that an unnamed CHECK constraint in the new table will never be merged, since a unique name will always be chosen for it.

Column STORAGE settings are also copied from parent tables.

LIKE *source_table* [*like_option* ...]

The LIKE clause specifies a table from which the new table automatically copies all column names, their data types, and their not-null constraints.

Unlike INHERITS, the new table and original table are completely decoupled after creation is complete. Changes to the original table will not be applied to the new table, and it is not possible to include data of the new table in scans of the original table.

Default expressions for the copied column definitions will be copied only if INCLUDING DEFAULTS is specified. The default behavior is to exclude default expressions, resulting in the copied columns in the new table having null defaults. Note that copying defaults that call database-modification functions, such as nextval, may create a functional linkage between the original and new tables.

Not-null constraints are always copied to the new table. CHECK constraints will be copied only if INCLUDING CONSTRAINTS is specified. No distinction is made between column constraints and table constraints.

Indexes, PRIMARY KEY, UNIQUE, and EXCLUDE constraints on the original table will be created on the new table only if INCLUDING INDEXES is specified. Names for the new indexes and constraints are chosen according to the default rules, regardless of how the originals were named. (This behavior avoids possible duplicate-name failures for the new indexes.)

STORAGE settings for the copied column definitions will be copied only if INCLUDING STORAGE is specified. The default behavior is to exclude STORAGE settings, resulting in the copied columns in the new table having type-specific default settings. For more on STORAGE settings, see 第 56.2 节.

Comments for the copied columns, constraints, and indexes will be copied only if INCLUDING COMMENTS is specified. The default behavior is to exclude comments, resulting in the copied columns and constraints in the new table having no comments.

`INCLUDING ALL` is an abbreviated form of `INCLUDING DEFAULTS INCLUDING CONSTRAINTS INCLUDING INDEXES INCLUDING STORAGE INCLUDING COMMENTS`.

Note that unlike `INHERITS`, columns and constraints copied by `LIKE` are not merged with similarly named columns and constraints. If the same name is specified explicitly or in another `LIKE` clause, an error is signaled.

The `LIKE` clause can also be used to copy column definitions from views, foreign tables, or composite types. Inapplicable options (e.g., `INCLUDING INDEXES` from a view) are ignored.

`CONSTRAINT` *`constraint_name`*

An optional name for a column or table constraint. If the constraint is violated, the constraint name is present in error messages, so constraint names like `col must be positive` can be used to communicate helpful constraint information to client applications. (Double-quotes are needed to specify constraint names that contain spaces.) If a constraint name is not specified, the system generates a name.

`NOT NULL`

The column is not allowed to contain null values.

`NULL`

The column is allowed to contain null values. This is the default.

This clause is only provided for compatibility with non-standard SQL databases. Its use is discouraged in new applications.

`CHECK (` *`expression`* `) [ NO INHERIT ]`

The `CHECK` clause specifies an expression producing a Boolean result which new or updated rows must satisfy for an insert or update operation to succeed. Expressions evaluating to TRUE or UNKNOWN succeed. Should any row of an insert or update operation produce a FALSE result an error exception is raised and the insert or update does not alter the database. A check constraint specified as a column constraint should reference that column's value only, while an expression appearing in a table constraint can reference multiple columns.

Currently, `CHECK` expressions cannot contain subqueries nor refer to variables other than columns of the current row.

A constraint marked with `NO INHERIT` will not propagate to child tables.

`DEFAULT` *`default_expr`*

The `DEFAULT` clause assigns a default data value for the column whose column definition it appears within. The value is any variable-free expression (subqueries and cross-references to other columns in the current table are not allowed). The data type of the default expression must match the data type of the column.

The default expression will be used in any insert operation that does not specify a value for the column. If there is no default for a column, then the default is null.

`UNIQUE` (column constraint)
`UNIQUE (` *`column_name`* `[, ... ] )` (table constraint)

The `UNIQUE` constraint specifies that a group of one or more columns of a table can contain only unique values. The behavior of the unique table constraint is the

same as that for column constraints, with the additional capability to span multiple columns.

For the purpose of a unique constraint, null values are not considered equal.

Each unique table constraint must name a set of columns that is different from the set of columns named by any other unique or primary key constraint defined for the table. (Otherwise it would just be the same constraint listed twice.)

`PRIMARY KEY` (column constraint)
`PRIMARY KEY ( column_name [, ... ] )` (table constraint)

The `PRIMARY KEY` constraint specifies that a column or columns of a table can contain only unique (non-duplicate), nonnull values. Only one primary key can be specified for a table, whether as a column constraint or a table constraint.

The primary key constraint should name a set of columns that is different from the set of columns named by any unique constraint defined for the same table. (Otherwise, the unique constraint is redundant and will be discarded.)

`PRIMARY KEY` enforces the same data constraints as a combination of `UNIQUE` and `NOT NULL`, but identifying a set of columns as the primary key also provides metadata about the design of the schema, since a primary key implies that other tables can rely on this set of columns as a unique identifier for rows.

`EXCLUDE [ USING index_method ] ( exclude_element WITH operator [, ... ] ) index_parameters [ WHERE ( predicate ) ]`

The `EXCLUDE` clause defines an exclusion constraint, which guarantees that if any two rows are compared on the specified column(s) or expression(s) using the specified operator(s), not all of these comparisons will return `TRUE`. If all of the specified operators test for equality, this is equivalent to a `UNIQUE` constraint, although an ordinary unique constraint will be faster. However, exclusion constraints can specify constraints that are more general than simple equality. For example, you can specify a constraint that no two rows in the table contain overlapping circles (see 第 8.8 节) by using the `&&` operator.

Exclusion constraints are implemented using an index, so each specified operator must be associated with an appropriate operator class (see 第 11.9 节) for the index access method *index_method*. The operators are required to be commutative. Each *exclude_element* can optionally specify an operator class and/or ordering options; these are described fully under CREATE INDEX.

The access method must support `amgettuple` (see 第 52 章); at present this means GIN cannot be used. Although it's allowed, there is little point in using B-tree or hash indexes with an exclusion constraint, because this does nothing that an ordinary unique constraint doesn't do better. So in practice the access method will always be GiST or SP-GiST.

The *predicate* allows you to specify an exclusion constraint on a subset of the table; internally this creates a partial index. Note that parentheses are required around the predicate.

`REFERENCES reftable [ ( refcolumn ) ] [ MATCH matchtype ] [ ON DELETE action ] [ ON UPDATE action ]` (column constraint)
`FOREIGN KEY ( column_name [, ... ] ) REFERENCES reftable [ ( refcolumn [, ... ] ) ] [ MATCH matchtype ] [ ON DELETE action ] [ ON UPDATE action ]` (table constraint)

These clauses specify a foreign key constraint, which requires that a group of one or more columns of the new table must only contain values that match values in

the referenced column(s) of some row of the referenced table. If *refcolumn* is omitted, the primary key of the *reftable* is used. The referenced columns must be the columns of a non-deferrable unique or primary key constraint in the referenced table. Note that foreign key constraints cannot be defined between temporary tables and permanent tables.

A value inserted into the referencing column(s) is matched against the values of the referenced table and referenced columns using the given match type. There are three match types: `MATCH FULL`, `MATCH PARTIAL`, and `MATCH SIMPLE`, which is also the default. `MATCH FULL` will not allow one column of a multicolumn foreign key to be null unless all foreign key columns are null. `MATCH SIMPLE` allows some foreign key columns to be null while other parts of the foreign key are not null. `MATCH PARTIAL` is not yet implemented.

In addition, when the data in the referenced columns is changed, certain actions are performed on the data in this table's columns. The `ON DELETE` clause specifies the action to perform when a referenced row in the referenced table is being deleted. Likewise, the `ON UPDATE` clause specifies the action to perform when a referenced column in the referenced table is being updated to a new value. If the row is updated, but the referenced column is not actually changed, no action is done. Referential actions other than the `NO ACTION` check cannot be deferred, even if the constraint is declared deferrable. There are the following possible actions for each clause:

`NO ACTION`

Produce an error indicating that the deletion or update would create a foreign key constraint violation. If the constraint is deferred, this error will be produced at constraint check time if there still exist any referencing rows. This is the default action.

`RESTRICT`

Produce an error indicating that the deletion or update would create a foreign key constraint violation. This is the same as `NO ACTION` except that the check is not deferrable.

`CASCADE`

Delete any rows referencing the deleted row, or update the value of the referencing column to the new value of the referenced column, respectively.

`SET NULL`

Set the referencing column(s) to null.

`SET DEFAULT`

Set the referencing column(s) to their default values.

If the referenced column(s) are changed frequently, it might be wise to add an index to the foreign key column so that referential actions associated with the foreign key column can be performed more efficiently.

`DEFERRABLE`
`NOT DEFERRABLE`

This controls whether the constraint can be deferred. A constraint that is not deferrable will be checked immediately after every command. Checking of constraints that are deferrable can be postponed until the end of the transaction

(using the SET CONSTRAINTS command). `NOT DEFERRABLE` is the default. Currently, only `UNIQUE`, `PRIMARY KEY`, `EXCLUDE`, and `REFERENCES` (foreign key) constraints accept this clause. `NOT NULL` and `CHECK` constraints are not deferrable.

`INITIALLY IMMEDIATE`
`INITIALLY DEFERRED`

If a constraint is deferrable, this clause specifies the default time to check the constraint. If the constraint is `INITIALLY IMMEDIATE`, it is checked after each statement. This is the default. If the constraint is `INITIALLY DEFERRED`, it is checked only at the end of the transaction. The constraint check time can be altered with the SET CONSTRAINTS command.

`WITH ( storage_parameter [= value] [, ... ] )`

This clause specifies optional storage parameters for a table or index; see Storage Parameters for more information. The `WITH` clause for a table can also include `OIDS=TRUE` (or just `OIDS`) to specify that rows of the new table should have OIDs (object identifiers) assigned to them, or `OIDS=FALSE` to specify that the rows should not have OIDs. If `OIDS` is not specified, the default setting depends upon the default_with_oids configuration parameter. (If the new table inherits from any tables that have OIDs, then `OIDS=TRUE` is forced even if the command says `OIDS=FALSE`.)

If `OIDS=FALSE` is specified or implied, the new table does not store OIDs and no OID will be assigned for a row inserted into it. This is generally considered worthwhile, since it will reduce OID consumption and thereby postpone the wraparound of the 32-bit OID counter. Once the counter wraps around, OIDs can no longer be assumed to be unique, which makes them considerably less useful. In addition, excluding OIDs from a table reduces the space required to store the table on disk by 4 bytes per row (on most machines), slightly improving performance.

To remove OIDs from a table after it has been created, use ALTER TABLE.

`WITH OIDS`
`WITHOUT OIDS`

These are obsolescent syntaxes equivalent to `WITH (OIDS)` and `WITH (OIDS=FALSE)`, respectively. If you wish to give both an `OIDS` setting and storage parameters, you must use the `WITH ( ... )` syntax; see above.

`ON COMMIT`

The behavior of temporary tables at the end of a transaction block can be controlled using `ON COMMIT`. The three options are:

`PRESERVE ROWS`

No special action is taken at the ends of transactions. This is the default behavior.

`DELETE ROWS`

All rows in the temporary table will be deleted at the end of each transaction block. Essentially, an automatic TRUNCATE is done at each commit.

`DROP`

The temporary table will be dropped at the end of the current transaction block.

`TABLESPACE tablespace_name`

The *tablespace_name* is the name of the tablespace in which the new table is to be created. If not specified, default_tablespace is consulted, or temp_tablespaces if the table is temporary.

`USING INDEX TABLESPACE tablespace_name`

This clause allows selection of the tablespace in which the index associated with a `UNIQUE`, `PRIMARY KEY`, or `EXCLUDE` constraint will be created. If not specified, default_tablespace is consulted, or temp_tablespaces if the table is temporary.

Storage Parameters

The `WITH` clause can specify storage parameters for tables, and for indexes associated with a `UNIQUE`, `PRIMARY KEY`, or `EXCLUDE` constraint. Storage parameters for indexes are documented in CREATE INDEX. The storage parameters currently available for tables are listed below. For each parameter, unless noted, there is an additional parameter with the same name prefixed with `toast.`, which can be used to control the behavior of the table's secondary TOAST table, if any (see 第 56.2 节 for more information about TOAST). Note that the TOAST table inherits the `autovacuum_*` values from its parent table, if there are no `toast.autovacuum_*` settings set.

`fillfactor` (`integer`)

The fillfactor for a table is a percentage between 10 and 100. 100 (complete packing) is the default. When a smaller fillfactor is specified, `INSERT` operations pack table pages only to the indicated percentage; the remaining space on each page is reserved for updating rows on that page. This gives `UPDATE` a chance to place the updated copy of a row on the same page as the original, which is more efficient than placing it on a different page. For a table whose entries are never updated, complete packing is the best choice, but in heavily updated tables smaller fillfactors are appropriate. This parameter cannot be set for TOAST tables.

`autovacuum_enabled`, `toast.autovacuum_enabled` (`boolean`)

Enables or disables the autovacuum daemon on a particular table. If true, the autovacuum daemon will initiate a `VACUUM` operation on a particular table when the number of updated or deleted tuples exceeds `autovacuum_vacuum_threshold` plus `autovacuum_vacuum_scale_factor` times the number of live tuples currently estimated to be in the relation. Similarly, it will initiate an `ANALYZE` operation when the number of inserted, updated or deleted tuples exceeds `autovacuum_analyze_threshold` plus `autovacuum_analyze_scale_factor` times the number of live tuples currently estimated to be in the relation. If false, this table will not be autovacuumed, except to prevent transaction Id wraparound. See 第 23.1.5 节 for more about wraparound prevention. Observe that this variable inherits its value from the autovacuum setting.

`autovacuum_vacuum_threshold`, `toast.autovacuum_vacuum_threshold` (`integer`)

Minimum number of updated or deleted tuples before initiate a `VACUUM` operation on a particular table.

`autovacuum_vacuum_scale_factor`, `toast.autovacuum_vacuum_scale_factor` (`float4`)

Multiplier for `reltuples` to add to `autovacuum_vacuum_threshold`.

`autovacuum_analyze_threshold` (`integer`)

Minimum number of inserted, updated, or deleted tuples before initiate an `ANALYZE` operation on a particular table.

`autovacuum_analyze_scale_factor` (`float4`)

Multiplier for `reltuples` to add to `autovacuum_analyze_threshold`.

`autovacuum_vacuum_cost_delay`, `toast.autovacuum_vacuum_cost_delay` (`integer`)

Custom autovacuum_vacuum_cost_delay parameter.

`autovacuum_vacuum_cost_limit`, `toast.autovacuum_vacuum_cost_limit` (`integer`)

Custom autovacuum_vacuum_cost_limit parameter.

`autovacuum_freeze_min_age`, `toast.autovacuum_freeze_min_age` (`integer`)

Custom vacuum_freeze_min_age parameter. Note that autovacuum will ignore attempts to set a per-table `autovacuum_freeze_min_age` larger than the half system-wide autovacuum_freeze_max_age setting.

`autovacuum_freeze_max_age`, `toast.autovacuum_freeze_max_age` (`integer`)

Custom autovacuum_freeze_max_age parameter. Note that autovacuum will ignore attempts to set a per-table `autovacuum_freeze_max_age` larger than the system-wide setting (it can only be set smaller).

`autovacuum_freeze_table_age`, `toast.autovacuum_freeze_table_age` (`integer`)

Custom vacuum_freeze_table_age parameter.

Notes

Using OIDs in new applications is not recommended: where possible, using a `SERIAL` or other sequence generator as the table's primary key is preferred. However, if your application does make use of OIDs to identify specific rows of a table, it is recommended to create a unique constraint on the `oid` column of that table, to ensure that OIDs in the table will indeed uniquely identify rows even after counter wraparound. Avoid assuming that OIDs are unique across tables; if you need a database-wide unique identifier, use the combination of `tableoid` and row OID for the purpose.

> **提示**
>
> The use of `OIDS=FALSE` is not recommended for tables with no primary key, since without either an OID or a unique data key, it is difficult to identify specific rows.

PostgreSQL automatically creates an index for each unique constraint and primary key constraint to enforce uniqueness. Thus, it is not necessary to create an index explicitly for primary key columns. (See CREATE INDEX for more information.)

Unique constraints and primary keys are not inherited in the current implementation. This makes the combination of inheritance and unique constraints rather dysfunctional.

A table cannot have more than 1600 columns. (In practice, the effective limit is usually lower because of tuple-length constraints.)

Examples

Create table `films` and table `distributors`:

```
CREATE TABLE films (
    code        char(5) CONSTRAINT firstkey PRIMARY KEY,
    title       varchar(40) NOT NULL,
    did         integer NOT NULL,
    date_prod   date,
    kind        varchar(10),
    len         interval hour to minute
);

CREATE TABLE distributors (
     did    integer PRIMARY KEY DEFAULT nextval('serial'),
     name   varchar(40) NOT NULL CHECK (name <> '')
);
```

Create a table with a 2-dimensional array:

```
CREATE TABLE array_int (
    vector  int[][]
);
```

Define a unique table constraint for the table `films`. Unique table constraints can be defined on one or more columns of the table:

```
CREATE TABLE films (
    code        char(5),
    title       varchar(40),
    did         integer,
    date_prod   date,
    kind        varchar(10),
    len         interval hour to minute,
    CONSTRAINT production UNIQUE(date_prod)
);
```

Define a check column constraint:

```
CREATE TABLE distributors (
    did     integer CHECK (did > 100),
    name    varchar(40)
);
```

Define a check table constraint:

```
CREATE TABLE distributors (
    did     integer,
    name    varchar(40)
    CONSTRAINT con1 CHECK (did > 100 AND name <> '')
);
```

Define a primary key table constraint for the table `films`:

```
CREATE TABLE films (
    code        char(5),
    title       varchar(40),
    did         integer,
    date_prod   date,
    kind        varchar(10),
    len         interval hour to minute,
    CONSTRAINT code_title PRIMARY KEY(code,title)
);
```

Define a primary key constraint for table `distributors`. The following two examples are equivalent, the first using the table constraint syntax, the second the column constraint syntax:

```
CREATE TABLE distributors (
    did     integer,
    name    varchar(40),
    PRIMARY KEY(did)
);

CREATE TABLE distributors (
    did     integer PRIMARY KEY,
    name    varchar(40)
);
```

Assign a literal constant default value for the column `name`, arrange for the default value of column `did` to be generated by selecting the next value of a sequence object, and make the default value of `modtime` be the time at which the row is inserted:

```
CREATE TABLE distributors (
    name      varchar(40) DEFAULT 'Luso Films',
    did       integer DEFAULT nextval('distributors_serial'),
    modtime   timestamp DEFAULT current_timestamp
);
```

Define two `NOT NULL` column constraints on the table `distributors`, one of which is explicitly given a name:

```
CREATE TABLE distributors (
    did     integer CONSTRAINT no_null NOT NULL,
    name    varchar(40) NOT NULL
);
```

Define a unique constraint for the `name` column:

```
CREATE TABLE distributors (
    did     integer,
    name    varchar(40) UNIQUE
);
```

The same, specified as a table constraint:

```
CREATE TABLE distributors (
    did     integer,
    name    varchar(40),
    UNIQUE(name)
);
```

Create the same table, specifying 70% fill factor for both the table and its unique index:

```
CREATE TABLE distributors (
```

```
    did     integer,
    name    varchar(40),
    UNIQUE(name) WITH (fillfactor=70)
)
WITH (fillfactor=70);
```

Create table `circles` with an exclusion constraint that prevents any two circles from overlapping:

```
CREATE TABLE circles (
    c circle,
    EXCLUDE USING gist (c WITH &&)
);
```

Create table `cinemas` in tablespace `diskvol1`:

```
CREATE TABLE cinemas (
        id serial,
        name text,
        location text
) TABLESPACE diskvol1;
```

Create a composite type and a typed table:

```
CREATE TYPE employee_type AS (name text, salary numeric);

CREATE TABLE employees OF employee_type (
    PRIMARY KEY (name),
    salary WITH OPTIONS DEFAULT 1000
);
```

Compatibility

The `CREATE TABLE` command conforms to the SQL standard, with exceptions listed below.

Temporary Tables

Although the syntax of `CREATE TEMPORARY TABLE` resembles that of the SQL standard, the effect is not the same. In the standard, temporary tables are defined just once and automatically exist (starting with empty contents) in every session that needs them. PostgreSQL instead requires each session to issue its own `CREATE TEMPORARY TABLE` command for each temporary table to be used. This allows different sessions to use the same temporary table name for different purposes, whereas the standard's approach constrains all instances of a given temporary table name to have the same table structure.

The standard's definition of the behavior of temporary tables is widely ignored. PostgreSQL's behavior on this point is similar to that of several other SQL databases.

The SQL standard also distinguishes between global and local temporary tables, where a local temporary table has a separate set of contents for each SQL module within each session, though its definition is still shared across sessions. Since PostgreSQL does not support SQL modules, this distinction is not relevant in PostgreSQL.

For compatibility's sake, PostgreSQL will accept the `GLOBAL` and `LOCAL` keywords in a temporary table declaration, but they currently have no effect. Use of these keywords is discouraged, since future versions of PostgreSQL might adopt a more standard-compliant interpretation of their meaning.

The `ON COMMIT` clause for temporary tables also resembles the SQL standard, but has some differences. If the `ON COMMIT` clause is omitted, SQL specifies that the default behavior is `ON COMMIT DELETE ROWS`. However, the default behavior in PostgreSQL is `ON COMMIT PRESERVE ROWS`. The `ON COMMIT DROP` option does not exist in SQL.

Non-deferred Uniqueness Constraints

When a `UNIQUE` or `PRIMARY KEY` constraint is not deferrable, PostgreSQL checks for uniqueness immediately whenever a row is inserted or modified. The SQL standard says that uniqueness should be enforced only at the end of the statement; this makes a difference when, for example, a single command updates multiple key values. To obtain standard-compliant behavior, declare the constraint as `DEFERRABLE` but not deferred (i.e., `INITIALLY IMMEDIATE`). Be aware that this can be significantly slower than immediate uniqueness checking.

Column Check Constraints

The SQL standard says that `CHECK` column constraints can only refer to the column they apply to; only `CHECK` table constraints can refer to multiple columns. PostgreSQL does not enforce this restriction; it treats column and table check constraints alike.

`EXCLUDE` Constraint

The `EXCLUDE` constraint type is a PostgreSQL extension.

`NULL` "Constraint"

The `NULL` "constraint" (actually a non-constraint) is a PostgreSQL extension to the SQL standard that is included for compatibility with some other database systems (and for symmetry with the `NOT NULL` constraint). Since it is the default for any column, its presence is simply noise.

Inheritance

Multiple inheritance via the `INHERITS` clause is a PostgreSQL language extension. SQL:1999 and later define single inheritance using a different syntax and different semantics. SQL:1999-style inheritance is not yet supported by PostgreSQL.

Zero-column Tables

PostgreSQL allows a table of no columns to be created (for example, `CREATE TABLE foo();`). This is an extension from the SQL standard, which does not allow zero-column tables. Zero-column tables are not in themselves very useful, but disallowing them creates odd special cases for `ALTER TABLE DROP COLUMN`, so it seems cleaner to ignore this spec restriction.

`LIKE` Clause

While a `LIKE` clause exists in the SQL standard, many of the options that PostgreSQL accepts for it are not in the standard, and some of the standard's options are not implemented by PostgreSQL.

`WITH` Clause

The `WITH` clause is a PostgreSQL extension; neither storage parameters nor OIDs are in the standard.

Tablespaces

The PostgreSQL concept of tablespaces is not part of the standard. Hence, the clauses `TABLESPACE` and `USING INDEX TABLESPACE` are extensions.

Typed Tables

Typed tables implement a subset of the SQL standard. According to the standard, a typed table has columns corresponding to the underlying composite type as well as one other column that is the "self-referencing column". PostgreSQL does not support these self-referencing columns explicitly, but the same effect can be had using the OID feature.

See Also

ALTER TABLE, DROP TABLE, CREATE TABLESPACE, CREATE TYPE

CREATE TABLE AS

CREATE TABLE AS — 根据查询结果定义一个新表

大纲

```
CREATE [ [ GLOBAL | LOCAL ] { TEMPORARY | TEMP } | UNLOGGED ]
 TABLE table_name
    [ (column_name [, ...] ) ]
    [ WITH ( storage_parameter [= value] [, ... ] ) | WITH OIDS |
 WITHOUT OIDS ]
    [ ON COMMIT { PRESERVE ROWS | DELETE ROWS | DROP } ]
    [ TABLESPACE tablespace_name ]
    AS query
    [ WITH [ NO ] DATA ]
```

描述

`CREATE TABLE AS`创建一个表，并用 `SELECT`命令计算得到的数据填充该表。该表的列具有与`SELECT`输出列对应的名称和数据类型（但可以通过显式给出新的列名列表来覆盖列名）。

`CREATE TABLE AS`与创建视图有些相似，但实际上差异很大：它会创建一个新表，并且只执行一次该查询来完成新表的初始填充。这个新表不会跟踪该查询所引用源表的后续变化。相比之下，视图每次被查询时都会重新执行其定义中的 `SELECT`语句。

参数

`GLOBAL` 或 `LOCAL`

出于兼容性考虑，这些关键字会被忽略。它们的用法已废弃；详见CREATE TABLE。

`TEMPORARY` 或 `TEMP`

如果指定，将把该表创建为临时表。详见CREATE TABLE。

`UNLOGGED`

如果指定，该表将创建为不记录 WAL 的表。详见CREATE TABLE。

`table_name`

要创建的表名（可选地带模式限定）。

`column_name`

新表中某一列的名称。如果未提供列名，则采用查询输出列的列名。

`WITH ( storage_parameter [= value] [, ... ] )`

此子句为新表指定可选存储参数；有关更多信息，请参见Storage Parameters。`WITH`子句还可以包含`OIDS=TRUE`（或仅包含`OIDS`），以指定应为新表的行分配 OID（对象标识符），或者包含`OIDS=FALSE`，以指定这些行不应有 OID。更多信息请参见CREATE TABLE。

`WITH OIDS`
`WITHOUT OIDS`

这些是分别等同于`WITH (OIDS)`和`WITH (OIDS=FALSE)`的过时语法。如果希望同时指定`OIDS`设置和存储参数，则必须使用`WITH ( ... )`语法；见上文。

`ON COMMIT`

临时表在事务块结束时的行为可以使用以下选项控制：`ON COMMIT`。三个选项是：

`PRESERVE ROWS`

在事务结束时不采取特殊动作。这是默认行为。

`DELETE ROWS`

临时表中的所有行将在每个事务块结束时删除。本质上，每次提交时都会执行一次自动的TRUNCATE。

`DROP`

临时表会在当前事务块结束时被删除。

`TABLESPACE` *`tablespace_name`*

*`tablespace_name`*是要创建新表的表空间名称。如果未指定，则查询default_tablespace；如果表是临时表，则查询temp_tablespaces。

`query`

可以是SELECT、TABLE或VALUES命令，也可以是运行已预备的`SELECT`、`TABLE`或`VALUES`查询的EXECUTE命令。

`WITH [ NO ] DATA`

这个子句指定是否将查询产生的数据复制到新表中。如果不复制，则只复制表结构。默认会复制数据。

注解

这个命令在功能上类似于SELECT INTO，但更推荐使用它，因为它更不容易与`SELECT INTO`语法的其他用法相混淆。此外，`CREATE TABLE AS`提供的功能是`SELECT INTO`所提供功能的超集。

在 PostgreSQL 8.0 之前，`CREATE TABLE AS`创建的表总是包含 OID。从 PostgreSQL 8.0 起，`CREATE TABLE AS`命令允许用户显式指定是否包含 OID。如果未显式指定是否存在OID，则使用default_with_oids配置变量。而从 PostgreSQL 8.1 起，该变量默认为 false，因此默认行为与 8.0 之前的发行版不再相同。需要在`CREATE TABLE AS`所创建表中包含 OID 的应用，应显式指定`WITH (OIDS)`以确保所需行为。

示例

创建一个新表`films_recent`，其中只包含表 `films`中的近期条目：

```
CREATE TABLE films_recent AS
  SELECT * FROM films WHERE date_prod >= '2002-01-01';
```

要完整复制一个表，也可以使用`TABLE`命令的简写形式：

```
CREATE TABLE films2 AS
  TABLE films;
```

创建一个新的临时表`films_recent`，其中只包含来自以下表的近期条目：`films`，使用预备语句。新表具有 OID，并将在提交时删除：

```
PREPARE recentfilms(date) AS
```

```
  SELECT * FROM films WHERE date_prod > $1;
CREATE TEMP TABLE films_recent WITH (OIDS) ON COMMIT DROP AS
  EXECUTE recentfilms('2002-01-01');
```

兼容性

`CREATE TABLE AS`符合SQL标准。以下是非标准扩展：

- 标准要求在子查询子句外围加括号；在 PostgreSQL中，这些括号是可选的。
- 在标准中，`WITH [ NO ] DATA`子句是必需的；在 PostgreSQL 中它是可选的。
- PostgreSQL处理临时表的方式与标准有相当大的不同；详见CREATE TABLE。
- `WITH`子句是PostgreSQL扩展；标准中既没有存储参数，也没有 OID。
- PostgreSQL的表空间概念不是标准的一部分。因此，`TABLESPACE`子句是一种扩展。

另见

CREATE TABLE, EXECUTE, SELECT, SELECT INTO, VALUES

CREATE TABLESPACE

CREATE TABLESPACE — 定义一个新表空间

大纲

```
CREATE TABLESPACE tablespace_name [ OWNER user_name ] LOCATION
 'directory'
```

描述

`CREATE TABLESPACE`注册一个新的集簇级别的表空间。该表空间的名称必须不同于数据库集簇中任何现有表空间的名称。

表空间允许超级用户在文件系统中定义一个替代位置，使数据库对象（如表和索引）的数据文件可以存放在那里。

具有适当权限的用户可以将 *tablespace_name*传递给 `CREATE DATABASE`、`CREATE TABLE`、`CREATE INDEX`或`ADD CONSTRAINT`，从而将这些对象的数据文件存储在指定的表空间中。

参数

tablespace_name

要创建的表空间名称。名称不能以`pg_`开头，因为这类名称保留给系统表空间使用。

user_name

将成为该表空间所有者的用户名。若省略，则默认为执行该命令的用户。
只有超级用户可以创建表空间，但他们可以将表空间的所有权赋予非超级用户。

directory

将用于表空间的目录。该目录应为空，并且必须归PostgreSQL系统用户所有。
必须使用绝对路径名指定该目录。

注解

表空间仅在支持符号链接的系统上受支持。

`CREATE TABLESPACE`不能在事务块内执行。

示例

创建一个表空间`dbspace`，位置为`/data/dbs`：

```
CREATE TABLESPACE dbspace LOCATION '/data/dbs';
```

创建一个表空间`indexspace`，位置为`/data/indexes`，所有者为用户`genevieve`：

```
CREATE TABLESPACE indexspace OWNER genevieve LOCATION '/data/
indexes';
```

兼容性

`CREATE TABLESPACE`是一种PostgreSQL扩展。

另见

CREATE DATABASE, CREATE TABLE, CREATE INDEX, DROP TABLESPACE, ALTER TABLESPACE

CREATE TEXT SEARCH CONFIGURATION

CREATE TEXT SEARCH CONFIGURATION — 定义一个新的文本搜索配置

大纲

```
CREATE TEXT SEARCH CONFIGURATION name (
    PARSER = parser_name |
    COPY = source_config
)
```

描述

`CREATE TEXT SEARCH CONFIGURATION` 创建一个新的文本搜索配置。文本搜索配置指定一个能把字符串拆分为记号的文本搜索解析器，
以及一些可用于判断哪些记号对搜索有意义的词典。

如果只指定了解析器，那么新文本搜索配置最初没有从记号类型到词典的映射，
因而会忽略所有词。要让该配置真正可用，必须使用后续的`ALTER TEXT SEARCH CONFIGURATION`命令创建映射。另一种方式是复制一个现有的文本搜索配置。

如果给出了一个模式名称，则文本搜索配置会被创建在指定的模式中。
否则它将会被创建在当前模式中。

定义该文本搜索配置的用户会成为其拥有者。

更多信息请参见第 12 章。

参数

name

要创建的文本搜索配置的名称。该名称可以是模式限定的。

parser_name

此配置要使用的文本搜索解析器的名称。

source_config

要复制的现有文本搜索配置的名称。

注解

`PARSER`和`COPY`选项是互斥的，因为当一个已有的配置被复制时，
它的解析器选择也会一并被复制。

兼容性

在 SQL 标准中没有 `CREATE TEXT SEARCH CONFIGURATION`语句。

另见

ALTER TEXT SEARCH CONFIGURATION, DROP TEXT SEARCH CONFIGURATION

CREATE TEXT SEARCH DICTIONARY

CREATE TEXT SEARCH DICTIONARY — 定义一个新的文本搜索字典

大纲

```
CREATE TEXT SEARCH DICTIONARY name (
    TEMPLATE = template
    [, option = value [, ... ]]
)
```

描述

`CREATE TEXT SEARCH DICTIONARY`创建一个新的文本搜索字典。文本搜索字典指定一种在搜索时识别哪些词值得关注、哪些词不值得关注的方式。字典依赖于文本搜索模板，后者规定了实际执行这项工作的函数。通常，字典会提供一些选项，用来控制模板函数的具体行为。

如果给出了一个模式名称，那么该文本搜索字典会被创建在指定的模式中。否则它会被创建在当前模式中。

定义文本搜索字典的用户将成为其拥有者。

更多信息请参见第 12 章。

参数

name

要创建的文本搜索字典的名称。该名称可以被模式限定。

template

用于定义该字典基本行为的文本搜索模板名称。

option

要为此字典设置的模板相关选项的名称。

value

模板相关选项要使用的值。如果该值不是简单的标识符或数字，则必须用引号括起（当然，如果你愿意，也始终可以加引号）。

这些选项可以按任意顺序出现。

示例

下面的示例命令创建了一个基于 Snowball 且使用非标准停用词列表的字典。

```
CREATE TEXT SEARCH DICTIONARY my_russian (
    template = snowball,
    language = russian,
    stopwords = myrussian
);
```

兼容性

在 SQL 标准中没有 `CREATE TEXT SEARCH DICTIONARY`语句。

另见

ALTER TEXT SEARCH DICTIONARY, DROP TEXT SEARCH DICTIONARY

CREATE TEXT SEARCH PARSER

CREATE TEXT SEARCH PARSER — 定义一个新的全文检索解析器

大纲

```
CREATE TEXT SEARCH PARSER name (
    START = start_function ,
    GETTOKEN = gettoken_function ,
    END = end_function ,
    LEXTYPES = lextypes_function
    [, HEADLINE = headline_function ]
)
```

描述

`CREATE TEXT SEARCH PARSER`创建一个新的全文检索解析器。全文检索解析器定义了一种方法，用于将文本字符串拆分成记号并为这些记号指定类型（类别）。
解析器本身并没有太大用处，必须与一些全文检索字典一起绑定到一个全文检索配置中，
才能用于搜索。

如果给出了一个模式名称，那么全文检索解析器将被创建在指定的模式中。
否则它会被创建在当前模式中。

要使用`CREATE TEXT SEARCH PARSER`，你必须是超级用户。之所以有此限制，是因为错误的全文检索解析器定义可能会让服务器陷入混乱，甚至崩溃。

更多信息请参见第 12 章。

参数

name

要创建的全文检索解析器的名称。该名称可以是模式限定的。

start_function

该解析器的启动函数的名称。

gettoken_function

该解析器的获取下一个记号的函数名称。

end_function

该解析器的结束函数的名称。

lextypes_function

该解析器的 lextypes 函数的名称（该函数返回它所产生的记号类型集合的信息）。

headline_function

该解析器的 headline 函数的名称（该函数对一组记号生成摘要）。

如有必要，函数名称可以是模式限定的。这里未给出参数类型，
因为每类函数的参数列表都是预先确定的。除 headline 函数外，其余函数都是必需的。

这些参数可以按任意顺序出现，不必与上面展示的顺序一致。

兼容性

在 SQL 标准中没有 `CREATE TEXT SEARCH PARSER`语句。

另见

ALTER TEXT SEARCH PARSER, DROP TEXT SEARCH PARSER

CREATE TEXT SEARCH TEMPLATE

CREATE TEXT SEARCH TEMPLATE — 定义一个新的全文检索模板

大纲

```
CREATE TEXT SEARCH TEMPLATE name (
    [ INIT = init_function , ]
    LEXIZE = lexize_function
)
```

描述

`CREATE TEXT SEARCH TEMPLATE`创建一个新的全文检索模板。全文检索模板定义实现全文检索字典的函数。模板本身并无直接用途，必须先实例化为字典后才能使用。
字典通常会指定要传递给模板函数的参数。

如果给出了模式名称，则全文检索模板会被创建在指定模式中。
否则它会被创建在当前模式中。

要使用`CREATE TEXT SEARCH TEMPLATE`，你必须是超级用户。之所以有此限制，是因为错误的全文检索模板定义可能使服务器陷入混乱，甚至崩溃。将模板与字典分离的原因在于，模板封装了定义字典时那些“不安全”的方面。而在定义字典时可设置的参数，
对非特权用户来说是安全的，因此创建字典不必是特权操作。

更多信息请参见第 12 章。

参数

name

要创建的全文检索模板名称。该名称可以是模式限定的。

init_function

该模板的 init 函数的名称。

lexize_function

该模板的 lexize 函数名称。

如有必要，函数名称可以是模式限定的。这里没有给出参数类型，
因为每一类函数的参数列表都是预先确定的。lexize 函数是必需的，但 init 函数是可选的。

这些参数可以按任意顺序出现，不必局限于上面显示的顺序。

兼容性

在 SQL 标准中没有`CREATE TEXT SEARCH TEMPLATE`语句。

另见

ALTER TEXT SEARCH TEMPLATE, DROP TEXT SEARCH TEMPLATE

CREATE TRIGGER

CREATE TRIGGER — define a new trigger

大纲

```
CREATE [ CONSTRAINT ] TRIGGER name { BEFORE | AFTER | INSTEAD OF }
 { event [ OR ... ] }
    ON table_name
    [ FROM referenced_table_name ]
    [ NOT DEFERRABLE | [ DEFERRABLE ] { INITIALLY IMMEDIATE |
 INITIALLY DEFERRED } ]
    [ FOR [ EACH ] { ROW | STATEMENT } ]
    [ WHEN ( condition ) ]
    EXECUTE PROCEDURE function_name ( arguments )

where event can be one of:

    INSERT
    UPDATE [ OF column_name [, ... ] ]
    DELETE
    TRUNCATE
```

Description

`CREATE TRIGGER` creates a new trigger. The trigger will be associated with the specified table or view and will execute the specified function *`function_name`* when certain events occur.

The trigger can be specified to fire before the operation is attempted on a row (before constraints are checked and the `INSERT`, `UPDATE`, or `DELETE` is attempted); or after the operation has completed (after constraints are checked and the `INSERT`, `UPDATE`, or `DELETE` has completed); or instead of the operation (in the case of inserts, updates or deletes on a view). If the trigger fires before or instead of the event, the trigger can skip the operation for the current row, or change the row being inserted (for `INSERT` and `UPDATE` operations only). If the trigger fires after the event, all changes, including the effects of other triggers, are "visible" to the trigger.

A trigger that is marked `FOR EACH ROW` is called once for every row that the operation modifies. For example, a `DELETE` that affects 10 rows will cause any `ON DELETE` triggers on the target relation to be called 10 separate times, once for each deleted row. In contrast, a trigger that is marked `FOR EACH STATEMENT` only executes once for any given operation, regardless of how many rows it modifies (in particular, an operation that modifies zero rows will still result in the execution of any applicable `FOR EACH STATEMENT` triggers).

Triggers that are specified to fire `INSTEAD OF` the trigger event must be marked `FOR EACH ROW`, and can only be defined on views. `BEFORE` and `AFTER` triggers on a view must be marked as `FOR EACH STATEMENT`.

In addition, triggers may be defined to fire for `TRUNCATE`, though only `FOR EACH STATEMENT`.

The following table summarizes which types of triggers may be used on tables and views:

When	Event	Row-level	Statement-level
BEFORE	INSERT/ UPDATE/DELETE	Tables	Tables and views
	TRUNCATE	–	Tables
AFTER	INSERT/ UPDATE/DELETE	Tables	Tables and views
	TRUNCATE	–	Tables
INSTEAD OF	INSERT/ UPDATE/DELETE	Views	–
	TRUNCATE	–	–

Also, a trigger definition can specify a Boolean WHEN condition, which will be tested to see whether the trigger should be fired. In row-level triggers the WHEN condition can examine the old and/or new values of columns of the row. Statement-level triggers can also have WHEN conditions, although the feature is not so useful for them since the condition cannot refer to any values in the table.

If multiple triggers of the same kind are defined for the same event, they will be fired in alphabetical order by name.

When the CONSTRAINT option is specified, this command creates a constraint trigger. This is the same as a regular trigger except that the timing of the trigger firing can be adjusted using SET CONSTRAINTS. Constraint triggers must be AFTER ROW triggers. They can be fired either at the end of the statement causing the triggering event, or at the end of the containing transaction; in the latter case they are said to be deferred. A pending deferred-trigger firing can also be forced to happen immediately by using SET CONSTRAINTS. Constraint triggers are expected to raise an exception when the constraints they implement are violated.

SELECT does not modify any rows so you cannot create SELECT triggers. Rules and views are more appropriate in such cases.

Refer to 第 36 章 for more information about triggers.

Parameters

name

The name to give the new trigger. This must be distinct from the name of any other trigger for the same table. The name cannot be schema-qualified – the trigger inherits the schema of its table. For a constraint trigger, this is also the name to use when modifying the trigger's behavior using SET CONSTRAINTS.

BEFORE
AFTER
INSTEAD OF

Determines whether the function is called before, after, or instead of the event. A constraint trigger can only be specified as AFTER.

event

One of INSERT, UPDATE, DELETE, or TRUNCATE; this specifies the event that will fire the trigger. Multiple events can be specified using OR.

For UPDATE events, it is possible to specify a list of columns using this syntax:

```
UPDATE OF column_name1 [, column_name2 ... ]
```

The trigger will only fire if at least one of the listed columns is mentioned as a target of the `UPDATE` command.

`INSTEAD OF UPDATE` events do not support lists of columns.

`table_name`

The name (optionally schema-qualified) of the table or view the trigger is for.

`referenced_table_name`

The (possibly schema-qualified) name of another table referenced by the constraint. This option is used for foreign-key constraints and is not recommended for general use. This can only be specified for constraint triggers.

```
DEFERRABLE
NOT DEFERRABLE
INITIALLY IMMEDIATE
INITIALLY DEFERRED
```

The default timing of the trigger. See the CREATE TABLE documentation for details of these constraint options. This can only be specified for constraint triggers.

```
FOR EACH ROW
FOR EACH STATEMENT
```

This specifies whether the trigger procedure should be fired once for every row affected by the trigger event, or just once per SQL statement. If neither is specified, `FOR EACH STATEMENT` is the default. Constraint triggers can only be specified `FOR EACH ROW`.

`condition`

A Boolean expression that determines whether the trigger function will actually be executed. If `WHEN` is specified, the function will only be called if the *`condition`* returns `true`. In `FOR EACH ROW` triggers, the `WHEN` condition can refer to columns of the old and/or new row values by writing `OLD.`*`column_name`* or `NEW.`*`column_name`* respectively. Of course, `INSERT` triggers cannot refer to `OLD` and `DELETE` triggers cannot refer to `NEW`.

`INSTEAD OF` triggers do not support `WHEN` conditions.

Currently, `WHEN` expressions cannot contain subqueries.

Note that for constraint triggers, evaluation of the `WHEN` condition is not deferred, but occurs immediately after the row update operation is performed. If the condition does not evaluate to true then the trigger is not queued for deferred execution.

`function_name`

A user-supplied function that is declared as taking no arguments and returning type `trigger`, which is executed when the trigger fires.

`arguments`

An optional comma-separated list of arguments to be provided to the function when the trigger is executed. The arguments are literal string constants. Simple names and numeric constants can be written here, too, but they will all be converted to strings. Please check the description of the implementation language

of the trigger function to find out how these arguments can be accessed within the function; it might be different from normal function arguments.

Notes

To create a trigger on a table, the user must have the `TRIGGER` privilege on the table. The user must also have `EXECUTE` privilege on the trigger function.

Use DROP TRIGGER to remove a trigger.

A column-specific trigger (one defined using the `UPDATE OF` *`column_name`* syntax) will fire when any of its columns are listed as targets in the `UPDATE` command's `SET` list. It is possible for a column's value to change even when the trigger is not fired, because changes made to the row's contents by `BEFORE UPDATE` triggers are not considered. Conversely, a command such as `UPDATE ... SET x = x ...` will fire a trigger on column `x`, even though the column's value did not change.

In a `BEFORE` trigger, the `WHEN` condition is evaluated just before the function is or would be executed, so using `WHEN` is not materially different from testing the same condition at the beginning of the trigger function. Note in particular that the `NEW` row seen by the condition is the current value, as possibly modified by earlier triggers. Also, a `BEFORE` trigger's `WHEN` condition is not allowed to examine the system columns of the `NEW` row (such as `oid`), because those won't have been set yet.

In an `AFTER` trigger, the `WHEN` condition is evaluated just after the row update occurs, and it determines whether an event is queued to fire the trigger at the end of statement. So when an `AFTER` trigger's `WHEN` condition does not return true, it is not necessary to queue an event nor to re-fetch the row at end of statement. This can result in significant speedups in statements that modify many rows, if the trigger only needs to be fired for a few of the rows.

In PostgreSQL versions before 7.3, it was necessary to declare trigger functions as returning the placeholder type `opaque`, rather than `trigger`. To support loading of old dump files, `CREATE TRIGGER` will accept a function declared as returning `opaque`, but it will issue a notice and change the function's declared return type to `trigger`.

Examples

Execute the function `check_account_update` whenever a row of the table `accounts` is about to be updated:

```
CREATE TRIGGER check_update
    BEFORE UPDATE ON accounts
    FOR EACH ROW
    EXECUTE PROCEDURE check_account_update();
```

The same, but only execute the function if column `balance` is specified as a target in the `UPDATE` command:

```
CREATE TRIGGER check_update
    BEFORE UPDATE OF balance ON accounts
    FOR EACH ROW
    EXECUTE PROCEDURE check_account_update();
```

This form only executes the function if column `balance` has in fact changed value:

```
CREATE TRIGGER check_update
    BEFORE UPDATE ON accounts
```

```
    FOR EACH ROW
    WHEN (OLD.balance IS DISTINCT FROM NEW.balance)
    EXECUTE PROCEDURE check_account_update();
```

Call a function to log updates of `accounts`, but only if something changed:

```
CREATE TRIGGER log_update
    AFTER UPDATE ON accounts
    FOR EACH ROW
    WHEN (OLD.* IS DISTINCT FROM NEW.*)
    EXECUTE PROCEDURE log_account_update();
```

Execute the function `view_insert_row` for each row to insert rows into the tables underlying a view:

```
CREATE TRIGGER view_insert
    INSTEAD OF INSERT ON my_view
    FOR EACH ROW
    EXECUTE PROCEDURE view_insert_row();
```

第 36.4 节 contains a complete example of a trigger function written in C.

Compatibility

The `CREATE TRIGGER` statement in PostgreSQL implements a subset of the SQL standard. The following functionality is currently missing:

- SQL allows you to define aliases for the "old" and "new" rows or tables for use in the definition of the triggered action (e.g., `CREATE TRIGGER ... ON tablename REFERENCING OLD ROW AS somename NEW ROW AS othername ...`). Since PostgreSQL allows trigger procedures to be written in any number of user-defined languages, access to the data is handled in a language-specific way.
- PostgreSQL only allows the execution of a user-defined function for the triggered action. The standard allows the execution of a number of other SQL commands, such as `CREATE TABLE`, as the triggered action. This limitation is not hard to work around by creating a user-defined function that executes the desired commands.

SQL specifies that multiple triggers should be fired in time-of-creation order. PostgreSQL uses name order, which was judged to be more convenient.

SQL specifies that `BEFORE DELETE` triggers on cascaded deletes fire after the cascaded `DELETE` completes. The PostgreSQL behavior is for `BEFORE DELETE` to always fire before the delete action, even a cascading one. This is considered more consistent. There is also nonstandard behavior if `BEFORE` triggers modify rows or prevent updates during an update that is caused by a referential action. This can lead to constraint violations or stored data that does not honor the referential constraint.

The ability to specify multiple actions for a single trigger using `OR` is a PostgreSQL extension of the SQL standard.

The ability to fire triggers for `TRUNCATE` is a PostgreSQL extension of the SQL standard, as is the ability to define statement-level triggers on views.

`CREATE CONSTRAINT TRIGGER` is a PostgreSQL extension of the SQL standard.

See Also

CREATE FUNCTION, ALTER TRIGGER, DROP TRIGGER, SET CONSTRAINTS

CREATE TYPE

CREATE TYPE — 定义一种新的数据类型

大纲

```
CREATE TYPE name AS
    ( [ attribute_name data_type [ COLLATE collation ] [, ... ] ] )

CREATE TYPE name AS ENUM
    ( [ 'label' [, ... ] ] )

CREATE TYPE name AS RANGE (
    SUBTYPE = subtype
    [ , SUBTYPE_OPCLASS = subtype_operator_class ]
    [ , COLLATION = collation ]
    [ , CANONICAL = canonical_function ]
    [ , SUBTYPE_DIFF = subtype_diff_function ]
)

CREATE TYPE name (
    INPUT = input_function,
    OUTPUT = output_function
    [ , RECEIVE = receive_function ]
    [ , SEND = send_function ]
    [ , TYPMOD_IN = type_modifier_input_function ]
    [ , TYPMOD_OUT = type_modifier_output_function ]
    [ , ANALYZE = analyze_function ]
    [ , INTERNALLENGTH = { internallength | VARIABLE } ]
    [ , PASSEDBYVALUE ]
    [ , ALIGNMENT = alignment ]
    [ , STORAGE = storage ]
    [ , LIKE = like_type ]
    [ , CATEGORY = category ]
    [ , PREFERRED = preferred ]
    [ , DEFAULT = default ]
    [ , ELEMENT = element ]
    [ , DELIMITER = delimiter ]
    [ , COLLATABLE = collatable ]
)

CREATE TYPE name
```

描述

`CREATE TYPE`在当前数据库中注册一种新的数据类型。定义该类型的用户将成为其拥有者。

如果给出了模式名，则该类型会在指定模式中创建；否则会在当前模式中创建。
类型名必须不同于同一模式中任何现有类型或域的名称。（由于表也具有关联的数据类型，类型名还必须不同于同一模式中任何现有表的名称。）

如上面的语法概要所示，`CREATE TYPE`有五种形式。它们分别用于创建复合类型、枚举类型、范围类型、基础类型或shell 类型。下文将依次讨论前四种。shell 类型只是稍后定义某种类型时使用的占位符；它通过执行除类型名外不带任何参数的`CREATE TYPE`来创建。正如相应小节所述，创建范围类型和基础类型时，需要用 shell 类型作为前向引用。

复合类型

第一种形式的CREATE TYPE创建复合类型。复合类型由属性名和数据类型列表指定。如果某个属性的数据类型支持排序规则，还可以指定该属性的排序规则。
复合类型本质上与表的行类型相同，但如果目的只是定义一种类型，使用CREATE TYPE 就不必实际创建表。例如，独立的复合类型可用作函数的参数类型或返回类型。

要创建复合类型，必须对所有属性类型都拥有USAGE 权限。

枚举类型

如第 8.7 节中所述，第二种形式的 CREATE TYPE创建枚举（enum）类型。枚举类型接受由一个或多个带引号标签构成的列表，其中每个标签的长度都必须小于 NAMEDATALEN字节（在标准 PostgreSQL构建中为 64 字节）。

范围类型

如第 8.16 节中所述，第三种形式的 CREATE TYPE创建范围类型。

范围类型的*subtype*可以是任何带有关联 B-树操作符类的类型（该操作符类用于确定范围类型值的顺序）。通常使用子类型默认的 B-树操作符类来确定顺序；若要使用非默认操作符类，可用*subtype_opclass*指定其名称。如果子类型支持排序规则，
而你希望在范围排序中使用非默认排序规则，可用 *collation*选项指定所需排序规则。

可选的*canonical*函数必须接受一个正在定义的范围类型值作为参数，
并返回同一类型的值。在适用时，它用于将范围值转换为规范形式。更多信息见第 8.16.8 节。创建 *canonical*函数有些棘手，因为它必须在声明范围类型之前定义。为此，必须先创建一种 shell 类型，它除了名称和拥有者外没有任何属性，只是一个占位符类型。这可通过执行不带任何附加参数的命令CREATE TYPE *name*完成。然后就可以把该 shell 类型用作参数类型和结果类型来声明该函数，最后再用同一名称声明范围类型。这样会自动用有效的范围类型替换 shell 类型条目。

可选的*subtype_diff* 函数必须接受两个*subtype*类型的值作为参数，并返回一个表示这两个给定值之差的double precision值。虽然这是可选的，但提供该函数可显著提高该范围类型列上 GiST 索引的效率。详见第 8.16.8 节。

基础类型

第四种形式的CREATE TYPE创建一种新的基础类型（标量类型）。要创建新的基础类型，你必须是超级用户。（这样限制是因为错误的类型定义可能使服务器陷入混乱，
甚至导致其崩溃。）

这些参数可以按任意顺序出现，不必局限于上面展示的顺序，而且大多数都是可选的。
在定义该类型之前，必须先注册两个或更多函数（使用 CREATE FUNCTION）。支持函数 *input_function*和 *output_function*是必需的；*receive_function*、*send_function*、*type_modifier_input_function*、*type_modifier_output_function*、*analyze_function*则是可选的。通常这些函数必须用 C 或其他低级语言编写。

*input_function*将类型的外部文本表示转换为该类型的操作符和函数所使用的内部表示。*output_function* 执行相反的转换。输入函数可以声明为接受一个cstring 参数，或者接受三个参数，类型分别为cstring、oid、integer。第一个参数是以 C 字符串表示的输入文本，第二个参数是该类型自身的 OID（数组类型例外，此时传入的是其元素类型的 OID），第三个参数是在已知情况下目标列的 typmod（未知则传入 -1）。输入函数必须返回该数据类型本身的值。通常输入函数应声明为 STRICT；否则，在读取 NULL 输入值时会以 NULL 作为第一个参数调用它。除非函数抛出错误，否则在这种情况下仍必须返回 NULL。（这种情况主要是为了支持可能需要拒绝 NULL 输入的域输入函数。）输出函数必须声明为接受一个新数据类型参数，并且必须返回cstring类型。对于 NULL 值不会调用输出函数。

可选的*receive_function*把类型的外部二进制表示转换为内部表示。如果未提供此函数，该类型就不能参与二进制输入。外部二进制表示应选择为既能低成本转换为内部形式，又具有合理可移植性。（例如，标准整数数据类型把网络字节序用作外部二进制表示，而内部表示则使用机器的本地字节序。）接收函数应进行充分检查以确保值有效。它可以声明为接受一个`internal`参数，或者接受三个参数，类型分别为`internal`、`oid`、`integer`。第一个参数是指向保存已接收字节串的`StringInfo`缓冲区的指针；其余可选参数与文本输入函数相同。接收函数必须返回该数据类型本身的值。通常，接收函数应声明为 STRICT；否则，在读取 NULL 输入值时会以 NULL 作为第一个参数调用它。除非函数抛出错误，否则在这种情况下仍必须返回 NULL。（这种情况主要是为了支持可能需要拒绝 NULL 输入的域接收函数。）类似地，可选的 *send_function*把内部表示转换为外部二进制表示。如果未提供此函数，该类型就不能参与二进制输出。发送函数必须声明为接受一个新数据类型参数，并且必须返回 `bytea`类型。对于 NULL 值不会调用发送函数。

读到这里，你可能会问：既然新类型本身还没创建，输入和输出函数怎么能声明为返回或接受这个新类型呢？答案是，应先把该类型定义为一种 shell 类型，它除了名称和拥有者外没有任何属性，只是一个占位符类型。这可通过执行不带任何附加参数的命令 `CREATE TYPE` *name* 完成。然后就可以定义引用该 shell 类型的 I/O 函数。最后，再用带完整定义的`CREATE TYPE`替换该 shell 条目，生成一个完整且有效的类型定义，此后新类型就能正常使用。

如果该类型支持修饰符，也就是附加在类型声明上的可选约束，例如 `char(5)`或`numeric(30,2)`，就需要可选的*type_modifier_input_function*和 *type_modifier_output_function*。PostgreSQL允许用户定义类型接受一个或多个简单常量或标识符作为修饰符。不过，这些信息必须能够打包成单个非负整数值，以便存储在系统目录中。声明的修饰符会以`cstring`数组的形式传递给*type_modifier_input_function*。它必须检查这些值是否有效（若无效则抛出错误），若有效则返回一个非负`integer`值，该值将作为列的“typmod”存储。如果该类型没有*type_modifier_input_function*，就会拒绝类型修饰符。*type_modifier_output_function* 则把内部整数 typmod 值转换回适合用户显示的正确形式。它必须返回一个`cstring`值，即精确追加到类型名后的字符串；例如，`numeric`的该函数可能返回 `(30,2)`。允许省略*type_modifier_output_function*；在这种情况下，默认显示格式只是把存储的 typmod 整数值放在圆括号中。

可选的*analyze_function* 为该数据类型的列执行类型专用的统计信息收集。默认情况下，如果该类型有默认的 B-树操作符类，`ANALYZE`将尝试使用该类型的“equals”和“less-than”操作符收集统计信息。对于非标量类型，这种行为很可能不合适，因此可以通过指定自定义分析函数来覆盖。分析函数必须声明为接受一个 `internal`参数并返回`boolean`结果。分析函数的详细 API 见`src/include/commands/vacuum.h`。

虽然新类型内部表示的细节只有 I/O 函数以及你为该类型编写的其他函数才知道，但仍有若干内部表示属性必须向 PostgreSQL声明。其中最重要的是 *internallength*。基础类型可以是定长的，此时*internallength*为正整数；也可以是变长的，此时将*internallength* 设为 `VARIABLE`。（在内部，这通过把 `typlen`设为 -1 表示。）所有变长类型的内部表示都必须以一个 4 字节整数开头，用来给出该类型该值的总长度。

可选标志`PASSEDBYVALUE`表示该数据类型的值按值传递，而不是按引用传递。内部表示大于`Datum`类型大小（多数机器上为 4 字节，少数为 8 字节）的类型不能按值传递。

*alignment*参数指定该数据类型所需的存储对齐方式。允许的值分别对应按 1、2、4 或 8 字节边界对齐。注意，变长类型的对齐至少必须为 4，因为它们的第一个组成部分必然是一个`int4`。

*storage*参数允许为变长数据类型选择存储策略。（定长类型只允许 `plain`。）`plain`表示该类型数据始终内联存储且不压缩。`extended`表示系统会先尝试压缩较长的数据值，如果仍然过长，就把该值移出主表行。`external`允许把值移出主表，但系统不会尝试压缩它。`main`允许压缩，但不鼓励把值移出主表。

（采用这种存储策略的数据项在没有其他办法让一行适配时仍可能被移出主表，但与extended和external 数据项相比，它们会被优先保留在主表中。）

like_type
参数提供了指定数据类型基本表示属性的另一种方法：从某个现有类型复制这些属性。*internallength*、*passedbyvalue*、*alignment*和 *storage*的值都从指定类型复制而来。（虽然可以通过同时给出LIKE子句和这些选项来覆盖其中某些值，但通常不建议这样做。）当新类型的底层实现以某种方式“借用”现有类型时，以这种方式指定表示属性尤其有用。

*category*和 *preferred*参数可用于在存在歧义时帮助控制应用哪一种隐式类型转换。每种数据类型都属于一个由单个 ASCII 字符命名的类别，并且在其类别内要么是“首选”的，要么不是。当这一规则有助于解析重载函数或操作符时，解析器会优先转换为首选类型（但只会从同一类别中的其他类型转换）。更多细节见第 10 章。对于与任何其他类型之间都没有隐式类型转换的类型，保持这些设置的默认值就足够了。不过，对于一组彼此存在隐式类型转换的相关类型，把它们都标记为属于同一类别，并选择一两个“最通用”的类型作为该类别的首选类型，通常会有帮助。*category*
参数在将用户定义类型加入现有内置类别（例如数值类型或字符串类型）时尤其有用。不过，也可以创建全新的纯用户定义类型类别。为这种类别命名时，可选择任一非大写字母的 ASCII 字符。

如果用户希望该数据类型列的默认值不是空值，可以指定默认值。用 DEFAULT关键字指定默认值。（该默认值可以被附加到具体列上的显式DEFAULT子句覆盖。）

若要表明一种类型是数组类型，可用ELEMENT 关键字指定数组元素的类型。例如，要定义由 4 字节整数（int4）构成的数组，可指定 ELEMENT = int4。更多关于数组类型的细节见下文。

若要指定该类型数组在外部表示中用于分隔各值的分隔符，可把 *delimiter*设为特定字符。默认分隔符是逗号（,）。注意，这个分隔符关联的是数组元素类型，而不是数组类型本身。

如果可选的布尔参数 *collatable*为真，则该类型的列定义和表达式可以通过COLLATE子句携带排序规则信息。是否实际使用这些排序规则信息取决于操作该类型的函数实现；仅仅把类型标记为支持排序规则并不会自动实现这一点。

数组类型

每当创建用户定义类型时，PostgreSQL 都会自动创建一个关联的数组类型，其名称由元素类型名前加一个下划线组成；必要时还会截断，以保持其长度小于 NAMEDATALEN字节。（如果这样生成的名称与现有类型名冲突，就会重复这一过程，直到找到不冲突的名称。）这种隐式创建的数组类型是变长的，并使用内置输入/输出函数 array_in和array_out。该数组类型会跟踪其元素类型的拥有者或模式的任何变化，并在元素类型被删除时一并删除。

如果系统会自动创建正确的数组类型，你可能会合理地问，为什么还需要ELEMENT选项。ELEMENT唯一有用的情况是：你正在创建一种定长类型，而它在内部恰好是若干相同元素组成的数组，并且除了为整个类型提供的操作之外，你还希望允许通过下标直接访问这些元素。例如，类型point在内部就表示为两个浮点数，可以用point[0]和 point[1]访问。注意，这种机制只适用于内部形式恰好是一串相同定长字段的定长类型。可用下标访问的变长类型必须采用array_in和array_out使用的通用内部表示。由于历史原因（也就是说这显然不对，但现在改已经太晚了），定长数组类型的下标从零开始，而变长数组则从一开始。

参数

name

要创建的类型名称（可选地带有模式限定）。

attribute_name

复合类型的一个属性（列）的名称。

data_type

将成为复合类型一列的现有数据类型名称。

collation

要与复合类型的某一列或范围类型关联的现有排序规则名称。

label

表示枚举类型某个值所关联文本标签的字符串字面量。

subtype

范围类型所表示范围的元素类型名称。

subtype_operator_class

子类型的 B-树操作符类名称。

canonical_function

范围类型规范化函数的名称。

subtype_diff_function

子类型差分函数的名称。

input_function

将数据从类型的外部文本形式转换为内部形式的函数名。

output_function

将数据从类型的内部形式转换为外部文本形式的函数名。

receive_function

将数据从类型的外部二进制形式转换成内部形式的函数名。

send_function

将数据从类型的内部形式转换为外部二进制形式的函数名。

type_modifier_input_function

将类型的修饰符数组转换为内部形式的函数名。

type_modifier_output_function

将类型修饰符的内部形式转换为外部文本形式的函数名。

analyze_function

为该数据类型执行统计分析的函数名。

internallength

一个数字常量，用于指定新类型内部表示的字节长度。默认假定它是变长的。

alignment

该数据类型的存储对齐需求。如果被指定，它必须是 `char`、`int2`、`int4`或者`double`。默认是 `int4`。

storage

该数据类型的存储策略。如果被指定，必须是 `plain`、`external`、`extended`或者`main`。默认是`plain`。

like_type

与新类型具有相同表示形式的现有数据类型名称。除非在本 `CREATE TYPE`命令的其他位置显式覆盖，否则 *internallength*、*passedbyvalue*、*alignment*和 *storage*的值都会从该类型复制。

category

该类型的类别码（单个 ASCII 字符）。默认值是表示“用户定义类型”的`'U'`。其他标准类别码见表 45.51。你也可以选择其他 ASCII 字符来创建自定义类别。

preferred

若该类型是其类型类别中的首选类型，则为真，否则为假。默认值为假。
在现有类型类别中创建新的首选类型时要格外小心，
因为这可能导致出人意料的行为变化。

default

该数据类型的默认值。若省略，默认值为 null。

element

正在创建的类型是数组；该参数指定数组元素类型。

delimiter

在由该类型构成的数组中各值之间使用的分隔符字符。

collatable

如果该类型的操作可以使用排序规则信息，则为真。默认为假。

注解

由于数据类型一旦创建，其使用方式就不再受限制，因此创建基础类型或范围类型，
相当于对类型定义中提到的那些函数授予公共执行权限。
这对适合用于类型定义的那类函数来说通常不是问题。但如果设计一种类型时，
需要在把它转换为外部形式或从外部形式转换回来时使用“秘密”信息，就应当三思。

在PostgreSQL 8.3 之前，自动生成的数组类型名称总是恰好等于元素类型名称前加一个下划线字符（`_`）。（因此，类型名称的长度限制比其他名称少一个字符。）
虽然现在通常仍是这样，但在名称达到最大长度或与以下划线开头的用户类型名冲突时，
数组类型名称可能与此不同。因此，依赖这一约定编写代码的做法已经弃用。请改用 `pg_type.typarray` 来定位与给定类型关联的数组类型。

建议避免使用以下划线开头的类型名和表名。
虽然服务器会改变生成的数组类型名称以避免与用户给定的名称冲突，
但仍然存在混淆风险，特别是对旧客户端软件而言，
它们可能会假定以下划线开头的类型名总是表示数组。

在PostgreSQL 8.2 之前，不存在 shell 类型创建语法`CREATE TYPE name`。创建新基础类型的做法是先创建它的输入函数。在这种做法下，PostgreSQL会首先把新数据类型名视为输入函数的返回类型。此时 shell 类型会被隐式创建，然后就可以在其余 I/O 函数的定义中引用它。这种做法仍然有效，但已弃用，并且可能在未来某个版本中被禁止。另外，为避免由于函数定义中的简单拼写错误而意外使系统目录充满 shell 类型，只有在输入函数用 C 编写时，才会以这种方式创建 shell 类型。

在PostgreSQL 7.3 以前的版本中，通常完全不创建 shell 类型，而是将函数中对类型名的前向引用替换为占位伪类型 `opaque`。在 7.3 以前，`cstring`参数和结果也必须声明为`opaque`。为了支持载入旧转储文件，`CREATE TYPE`会接受使用`opaque`声明的 I/O 函数，但会发出通知，并修改函数声明以使用正确的类型。

示例

这个示例创建一种复合类型，并在函数定义中使用它：

```
CREATE TYPE compfoo AS (f1 int, f2 text);

CREATE FUNCTION getfoo() RETURNS SETOF compfoo AS $$
    SELECT fooid, fooname FROM foo
$$ LANGUAGE SQL;
```

这个示例创建一种枚举类型，并在表定义中使用它：

```
CREATE TYPE bug_status AS ENUM ('new', 'open', 'closed');

CREATE TABLE bug (
    id serial,
    description text,
    status bug_status
);
```

这个示例创建一种范围类型：

```
CREATE TYPE float8_range AS RANGE (subtype = float8, subtype_diff =
 float8mi);
```

这个示例创建基础类型`box`，然后在表定义中使用它：

```
CREATE TYPE box;

CREATE FUNCTION my_box_in_function(cstring) RETURNS box AS ... ;
CREATE FUNCTION my_box_out_function(box) RETURNS cstring AS ... ;

CREATE TYPE box (
    INTERNALLENGTH = 16,
    INPUT = my_box_in_function,
    OUTPUT = my_box_out_function
);

CREATE TABLE myboxes (
    id integer,
    description box
);
```

如果`box`的内部结构是由四个 `float4`元素构成的数组，则也可以改为这样写：

```
CREATE TYPE box (
```

```
    INTERNALLENGTH = 16,
    INPUT = my_box_in_function,
    OUTPUT = my_box_out_function,
    ELEMENT = float4
);
```

这样就能通过下标访问 box 值的各个分量。除此之外，该类型的行为与前例相同。

这个示例创建一种大对象类型，并在表定义中使用它：

```
CREATE TYPE bigobj (
    INPUT = lo_filein, OUTPUT = lo_fileout,
    INTERNALLENGTH = VARIABLE
);
CREATE TABLE big_objs (
    id integer,
    obj bigobj
);
```

更多示例（包括配套的输入和输出函数）请见第 35.11 节。

兼容性

第一种`CREATE TYPE`形式，即创建复合类型的形式，符合SQL标准。其他形式都是PostgreSQL扩展。SQL标准中的`CREATE TYPE`语句还定义了PostgreSQL尚未实现的其他形式。

支持创建零属性的复合类型，是 PostgreSQL对标准的一种特有背离（类似于`CREATE TABLE`中的同类情况）。

另见

ALTER TYPE, CREATE DOMAIN, CREATE FUNCTION, DROP TYPE

CREATE USER

CREATE USER — 定义一个新的数据库角色

大纲

```
CREATE USER name [ [ WITH ] option [ ... ] ]
```

其中*option*可以是：

```
      SUPERUSER | NOSUPERUSER
    | CREATEDB | NOCREATEDB
    | CREATEROLE | NOCREATEROLE
    | CREATEUSER | NOCREATEUSER
    | INHERIT | NOINHERIT
    | LOGIN | NOLOGIN
    | REPLICATION | NOREPLICATION
    | CONNECTION LIMIT connlimit
    | [ ENCRYPTED | UNENCRYPTED ] PASSWORD 'password'
    | VALID UNTIL 'timestamp'
    | IN ROLE role_name [, ...]
    | IN GROUP role_name [, ...]
    | ROLE role_name [, ...]
    | ADMIN role_name [, ...]
    | USER role_name [, ...]
    | SYSID uid
```

描述

`CREATE USER`现为CREATE ROLE的别名。唯一的区别在于：当命令写为`CREATE USER`时，默认假定为`LOGIN`；而当命令写为 `CREATE ROLE`时，默认假定为`NOLOGIN`。

兼容性

`CREATE USER`语句是 PostgreSQL的一种扩展。SQL 标准将用户的定义留给具体实现决定。

另见

CREATE ROLE

CREATE USER MAPPING

CREATE USER MAPPING — 定义用户到外部服务器的新映射

大纲

```
CREATE USER MAPPING FOR { user_name | USER | CURRENT_USER |
 PUBLIC }
    SERVER server_name
    [ OPTIONS ( option 'value' [ , ... ] ) ]
```

描述

`CREATE USER MAPPING`定义用户到外部服务器的映射。用户映射通常封装连接信息，外部数据包装器会将其与外部服务器所封装的信息结合起来，以访问外部数据资源。

外部服务器的拥有者可以为该服务器上的任何用户创建用户映射。此外，
如果某个用户已被授予该服务器上的`USAGE`权限，
那么该用户也可以为自己的用户名创建用户映射。

参数

user_name

被映射到外部服务器的现有用户的名称。`CURRENT_USER` 和`USER`都匹配当前用户的名称。当指定 `PUBLIC`时，会创建一个所谓的公共映射；当没有适用的特定用户映射时，就会使用它。

server_name

将为其创建用户映射的现有服务器的名称。

`OPTIONS ( option 'value' [, ... ] )`

该子句指定用户映射的选项。这些选项通常定义该映射实际的用户名和密码。
选项名必须唯一。允许的选项名和值取决于该服务器的外部数据包装器。

示例

为用户`bob`、服务器`foo`创建一个用户映射：

```
CREATE USER MAPPING FOR bob SERVER foo OPTIONS (user 'bob',
 password 'secret');
```

兼容性

`CREATE USER MAPPING`符合 ISO/IEC 9075-9（SQL/MED）。

另见

ALTER USER MAPPING, DROP USER MAPPING, CREATE FOREIGN DATA WRAPPER, CREATE SERVER

CREATE VIEW

CREATE VIEW — define a new view

大纲

```
CREATE [ OR REPLACE ] [ TEMP | TEMPORARY ] VIEW name [ ( column_
name [, ...] ) ]
    [ WITH ( view_option_name [= view_option_value] [, ... ] ) ]
    AS query
```

Description

`CREATE VIEW` defines a view of a query. The view is not physically materialized. Instead, the query is run every time the view is referenced in a query.

`CREATE OR REPLACE VIEW` is similar, but if a view of the same name already exists, it is replaced. The new query must generate the same columns that were generated by the existing view query (that is, the same column names in the same order and with the same data types), but it may add additional columns to the end of the list. The calculations giving rise to the output columns may be completely different.

If a schema name is given (for example, `CREATE VIEW myschema.myview ...`) then the view is created in the specified schema. Otherwise it is created in the current schema. Temporary views exist in a special schema, so a schema name cannot be given when creating a temporary view. The name of the view must be distinct from the name of any other view, table, sequence, index or foreign table in the same schema.

Parameters

`TEMPORARY` or `TEMP`

If specified, the view is created as a temporary view. Temporary views are automatically dropped at the end of the current session. Existing permanent relations with the same name are not visible to the current session while the temporary view exists, unless they are referenced with schema-qualified names.

If any of the tables referenced by the view are temporary, the view is created as a temporary view (whether `TEMPORARY` is specified or not).

name

The name (optionally schema-qualified) of a view to be created.

column_name

An optional list of names to be used for columns of the view. If not given, the column names are deduced from the query.

`WITH ( view_option_name [= view_option_value] [, ... ] )`

This clause specifies optional parameters for a view; currently, the only supported parameter name is `security_barrier`, which should be enabled when a view is intended to provide row-level security. See 第 37.4 节 for full details.

query

A SELECT or VALUES command which will provide the columns and rows of the view.

Notes

Currently, views are read only: the system will not allow an insert, update, or delete on a view. You can get the effect of an updatable view by creating `INSTEAD` triggers on the view, which must convert attempted inserts, etc. on the view into appropriate actions on other tables. For more information see CREATE TRIGGER. Another possibility is to create rules (see CREATE RULE), but in practice triggers are easier to understand and use correctly.

Use the DROP VIEW statement to drop views.

Be careful that the names and types of the view's columns will be assigned the way you want. For example:

```
CREATE VIEW vista AS SELECT 'Hello World';
```

is bad form in two ways: the column name defaults to `?column?`, and the column data type defaults to `unknown`. If you want a string literal in a view's result, use something like:

```
CREATE VIEW vista AS SELECT text 'Hello World' AS hello;
```

Access to tables referenced in the view is determined by permissions of the view owner. In some cases, this can be used to provide secure but restricted access to the underlying tables. However, not all views are secure against tampering; see 第 37. 4 节 for details. Functions called in the view are treated the same as if they had been called directly from the query using the view. Therefore the user of a view must have permissions to call all functions used by the view.

When `CREATE OR REPLACE VIEW` is used on an existing view, only the view's defining SELECT rule is changed. Other view properties, including ownership, permissions, and non-SELECT rules, remain unchanged. You must own the view to replace it (this includes being a member of the owning role).

Examples

Create a view consisting of all comedy films:

```
CREATE VIEW comedies AS
    SELECT *
    FROM films
    WHERE kind = 'Comedy';
```

This will create a view containing the columns that are in the `film` table at the time of view creation. Though `*` was used to create the view, columns added later to the table will not be part of the view.

Compatibility

The SQL standard specifies some additional capabilities for the `CREATE VIEW` statement:

```
CREATE VIEW name [ ( column_name [, ...] ) ]
    AS query
    [ WITH [ CASCADED | LOCAL ] CHECK OPTION ]
```

The optional clauses for the full SQL command are:

`CHECK OPTION`

This option has to do with updatable views. All `INSERT` and `UPDATE` commands on the view will be checked to ensure data satisfy the view-defining condition (that is, the new data would be visible through the view). If they do not, the update will be rejected.

`LOCAL`

Check for integrity on this view.

`CASCADED`

Check for integrity on this view and on any dependent view. `CASCADED` is assumed if neither `CASCADED` nor `LOCAL` is specified.

`CREATE OR REPLACE VIEW` is a PostgreSQL language extension. So is the concept of a temporary view.

See Also

ALTER VIEW, DROP VIEW

DEALLOCATE

DEALLOCATE — 释放一个预备语句

大纲

```
DEALLOCATE [ PREPARE ] { name | ALL }
```

描述

`DEALLOCATE`用于释放一个先前准备好的 SQL 语句。如果不显式释放预备语句，则它会在会话结束时被释放。

更多关于预备语句的信息，请参见PREPARE。

参数

`PREPARE`

此关键字会被忽略。

`name`

要释放的预备语句的名称。

`ALL`

释放所有预备语句。

兼容性

SQL 标准中包含一个`DEALLOCATE`语句，但它只用于嵌入式 SQL。

另见

EXECUTE, PREPARE

DECLARE

DECLARE — 定义一个游标

大纲

```
DECLARE name [ BINARY ] [ INSENSITIVE ] [ [ NO ] SCROLL ]
    CURSOR [ { WITH | WITHOUT } HOLD ] FOR query
```

描述

DECLARE允许用户创建游标，游标可用于从较大的查询中一次取出少量行。游标创建后，可使用FETCH从中取出行。

> **注意**
>
> 本页面描述的是 SQL 命令层面的游标用法。如果想要在 PL/pgSQL函数中使用游标，规则会有所不同 – 见第 39.7 节。

参数

name

要创建的游标名称。它必须与该会话中任何其他活动游标的名称不同。

BINARY

使游标以二进制格式而不是文本格式返回数据。

INSENSITIVE

表示从游标取出的数据不应受到游标创建后对其底层表所做更新的影响。在PostgreSQL中，这是默认行为；因此该关键字没有效果，仅为兼容 SQL 标准而接受它。

SCROLL
NO SCROLL

SCROLL指定游标可用于以非顺序方式（例如向后）取出行。根据查询执行计划的复杂程度，指定SCROLL可能会给查询执行带来性能开销。NO SCROLL指定游标不能以非顺序方式取出行。默认情况下只在某些情形下允许滚动，这与显式指定 SCROLL并不相同。详见下文注解。

WITH HOLD
WITHOUT HOLD

WITH HOLD指定在创建该游标的事务成功提交后，仍可继续使用该游标。WITHOUT HOLD指定该游标不能在创建它的事务之外使用。如果既未指定WITHOUT HOLD也未指定WITH HOLD，默认值是WITHOUT HOLD。

query

一个提供游标要返回的行的SELECT或者VALUES命令。

关键字BINARY、INSENSITIVE和SCROLL可以按任意顺序出现。

注解

普通游标以文本格式返回数据，就像SELECT产生的结果一样。BINARY 选项指定游标应以二进制格式返回数据。这减少了服务器和客户端两端的转换工作量，但代价是程序员需要付出更多精力来处理与平台相关的二进制数据格式。举例来说，如果某个查询从一个整型列返回值 1，那么默认游标会返回字符串 1，而二进制游标则会返回一个 4 字节字段，其中包含该值的内部表示（采用大端字节序）。

应谨慎使用二进制游标。许多应用程序（包括 psql）都没有准备好处理二进制游标，而是期望返回的数据为文本格式。

> **注意**
>
> 当客户端应用使用“扩展查询”协议发出FETCH命令时，Bind 协议消息会指定数据应以文本格式还是二进制格式提取。这一选择会覆盖定义游标时所指定的方式。因此，在使用扩展查询协议时，二进制游标这一概念实际上已经过时了 – 任何游标都可以按文本或二进制方式处理。

除非指定了WITH HOLD，否则该命令创建的游标只能在当前事务中使用。因此，DECLARE若不带WITH HOLD，在事务块之外就毫无用处：游标只能存活到该语句执行完成。所以，如果在事务块之外使用这种命令，PostgreSQL会报错。
可使用BEGIN和COMMIT（或ROLLBACK）来定义事务块。

如果指定了WITH HOLD，且创建游标的事务成功提交，那么在同一会话中的后续事务里仍可继续访问该游标。（但如果创建事务被中止，游标会被移除。）使用WITH HOLD创建的游标会在对其发出显式CLOSE命令时关闭，或在会话结束时关闭。在当前实现中，这种游标所表示的行会被复制到临时文件或内存区域中，以便它们在后续事务中仍然可用。

当查询包括FOR UPDATE或FOR SHARE时，不能指定WITH HOLD。

在定义将用于向后取出行的游标时，应指定SCROLL选项。这是 SQL 标准所要求的。不过，为了兼容早期版本，如果游标的查询计划足够简单，以致支持向后取出不需要额外开销，PostgreSQL也允许在未指定SCROLL的情况下向后取出行。不过，建议应用开发者不要依赖于从未用SCROLL创建的游标中向后取出行。如果指定了NO SCROLL，那么无论如何都不允许向后取出行。

当查询包含FOR UPDATE或FOR SHARE时，同样不允许向后取出行。因此在这种情况下不能指定SCROLL。

> **小心**
>
> 可滚动游标和WITH HOLD游标如果调用了任何易变函数（参见第 35. 6 节），可能产生意外结果。重新取出先前已取出的行时，函数可能会再次执行，导致结果与第一次不同。对于这种情况，一个变通方法是将游标声明为WITH HOLD，并在读取其中任何行之前提交事务。这样会强制将游标的整个输出物化到临时存储中，使易变函数对每行恰好执行一次。

如果游标的查询包含FOR UPDATE或FOR SHARE，则返回的行会在首次取出时被锁定，与带有这些选项的普通SELECT命令相同。此外，返回的行将是最新版本；因此，这些选项提供了相当于 SQL 标准所称的“敏感游标”的行为。（将INSENSITIVE与FOR UPDATE或FOR SHARE一起指定会报错。）

> **小心**
>
> 通常都建议使用FOR UPDATE，如果游标打算与 UPDATE ... WHERE CURRENT OF 或 DELETE ... WHERE CURRENT OF一起使用。使用FOR UPDATE可以防止其他会话在这些行被取出之后、被更新之前更改它们。如果不使用FOR UPDATE，而某一行在游标创建后已经被更改，那么后续的WHERE CURRENT OF命令将不会有任何效果。
>
> 使用FOR UPDATE的另一个原因是：如果没有它，而后续的WHERE CURRENT OF 所针对的游标查询不符合 SQL 标准关于“简单可更新”的规则，则该命令可能失败。（特别是，游标必须只引用一个表，且不能使用分组或ORDER BY）。对于并非简单可更新的游标，是否可用取决于计划选择的细节，可能能工作，也可能不能。
> 因此在最坏情况下，应用可能在测试中可用，却会在生产中失败。
>
> 不将FOR UPDATE与WHERE CURRENT OF一起使用的主要原因，是你需要游标可滚动，或者需要它对后续更新不敏感（也就是说，继续显示旧数据）。如果这是需求，请务必仔细注意上面的警告。

SQL 标准只为嵌入式SQL中的游标作出规定。PostgreSQL服务器没有为游标实现OPEN语句；游标在声明时即被视为打开。不过，ECPG是PostgreSQL的嵌入式 SQL 预处理器，它支持标准 SQL 的游标约定，包括涉及DECLARE和OPEN语句的那些约定。

你可以通过查询pg_cursors 系统视图查看所有可用游标。

示例

声明一个游标：

```
DECLARE liahona CURSOR FOR SELECT * FROM films;
```

更多游标用法示例见FETCH。

兼容性

SQL 标准规定，游标默认是否对底层数据的并发更新敏感，由实现决定。在PostgreSQL中，游标默认不敏感，可以通过指定FOR UPDATE使其敏感。其他产品的行为可能不同。

SQL 标准只允许在嵌入式SQL和模块中使用游标。PostgreSQL允许以交互方式使用游标。

二进制游标是PostgreSQL的一种扩展。

另见

CLOSE, FETCH, MOVE

DELETE

DELETE — 删除表中的行

大纲

```
[ WITH [ RECURSIVE ] with_query [, ...] ]
DELETE FROM [ ONLY ] table_name [ * ] [ [ AS ] alias ]
    [ USING using_list ]
    [ WHERE condition | WHERE CURRENT OF cursor_name ]
    [ RETURNING * | output_expression [ [ AS ] output_name ] [, ...
] ]
```

描述

DELETE从指定表中删除满足 WHERE子句的行。如果省略WHERE 子句，效果就是删除表中的所有行。结果是一个合法但为空的表。

> **提示**
>
> TRUNCATE是一种PostgreSQL扩展，它提供了一种更快的机制来移除表中的所有行。

有两种方法可以利用数据库中其他表所包含的信息来删除一个表中的行：使用子选择，或者在USING子句中指定附加表。哪种技术更合适取决于具体情况。

可选的RETURNING子句使DELETE 基于每个实际被删除的行计算并返回一个或多个值。可以计算使用该表列和/或USING中提到的其他表列的任意表达式。RETURNING 列表的语法与SELECT 的输出列表相同。

要从一个表中删除行，必须具有该表上的DELETE权限，以及USING 子句中出现的任何表或者其值会在 *condition*中读取的任何表上的SELECT权限。

参数

with_query

WITH子句允许你指定一个或多个子查询，这些子查询可在DELETE查询中按名称引用。详见第 7.8 节和SELECT。

table_name

要从中删除行的表名（可以是模式限定的）。如果在表名前指定了 ONLY，只会从所提及表中删除匹配行。如果未指定 ONLY，还会删除继承自该表的任何表中的匹配行。可选地，可以在表名后指定*，以显式指示包含后代表。

alias

目标表的替代名称。提供别名时，它会完全隐藏该表的实际名称。例如，给定DELETE FROM foo AS f，DELETE语句的其余部分必须将该表称为 f，而不是foo。

using_list

一个表表达式列表，允许其他表的列出现在WHERE条件中。这类似于SELECT语句的FROM Clause中可以指定的表列表；例如，可以为表名指定别名。不要把目标表重复写在*using_list*中，除非打算进行自连接。

condition

一个返回boolean类型值的表达式。只有使这个表达式返回true的行才会被删除。

cursor_name

在WHERE CURRENT OF条件中使用的游标名称。要删除的行是最近从该游标取出的那一行。该游标必须是针对DELETE目标表的非分组查询。注意，WHERE CURRENT OF不能与布尔条件一起指定。有关游标使用的更多信息，请参阅DECLARE中关于WHERE CURRENT OF的说明。

output_expression

每行删除之后，由DELETE命令计算并返回的表达式。该表达式可以使用*table_name*所指定的表或USING中列出的表中的任何列名。写成*可返回所有列。

output_name

返回列所使用的名称。

输出

在成功完成时，一个DELETE命令会返回以下形式的命令标签：

DELETE *count*

*count*是被删除的行数。注意，当删除被BEFORE DELETE触发器抑制时，这个数量可能小于匹配*condition* 的行数。如果*count*为 0，则该查询没有删除任何行（这不被视为错误）。

如果DELETE命令包含RETURNING 子句，其结果将类似于一个SELECT语句，其中包含RETURNING列表中定义的列和值，并在该命令删除的行上进行计算。

注解

PostgreSQL允许通过在USING子句中指定其他表，在WHERE条件中引用这些表的列。例如，要删除由给定制片人制作的所有电影，可以这样做：

```
DELETE FROM films USING producers
  WHERE producer_id = producers.id AND producers.name = 'foo';
```

本质上，这里发生的是在films和 producers之间进行连接，并将所有成功连接到的films行标记为删除。这种语法不是标准的。更标准的写法是：

```
DELETE FROM films
  WHERE producer_id IN (SELECT id FROM producers WHERE name =
 'foo');
```

在某些情况下，连接形式比子选择形式更容易书写或者执行更快。

示例

删除所有电影，但音乐剧除外：

```
DELETE FROM films WHERE kind <> 'Musical';
```

清空表films：

```
DELETE FROM films;
```

删除已完成的任务，并返回被删除行的完整详情：

```
DELETE FROM tasks WHERE status = 'DONE' RETURNING *;
```

删除游标c_tasks当前所定位的 tasks行：

```
DELETE FROM tasks WHERE CURRENT OF c_tasks;
```

兼容性

这个命令符合SQL标准，不过 USING和RETURNING子句是 PostgreSQL扩展，在 DELETE中使用WITH也是扩展。

DISCARD

DISCARD — 丢弃会话状态

大纲

```
DISCARD { ALL | PLANS | TEMPORARY | TEMP }
```

描述

DISCARD释放与数据库会话关联的内部资源。这些资源通常在会话结束时释放。

DISCARD TEMP删除当前会话中创建的所有临时表。DISCARD PLANS释放所有内部缓存的查询计划。DISCARD ALL将会话重置为其初始状态，丢弃临时资源并重置会话级的配置更改。

参数

TEMPORARY 或 TEMP

删除当前会话中创建的所有临时表。

PLANS

释放所有缓存的查询计划。

ALL

释放与当前会话关联的所有临时资源，并将会话重置为初始状态。目前，这与执行以下语句序列具有相同效果：

```
SET SESSION AUTHORIZATION DEFAULT;
RESET ALL;
DEALLOCATE ALL;
CLOSE ALL;
UNLISTEN *;
SELECT pg_advisory_unlock_all();
DISCARD PLANS;
DISCARD SEQUENCES;
DISCARD TEMP;
```

注解

DISCARD ALL不能在事务块内部执行。

兼容性

DISCARD是PostgreSQL扩展。

DO

DO — 执行匿名代码块

大纲

```
DO [ LANGUAGE lang_name ] code
```

描述

DO执行匿名代码块，换言之，就是执行一个以过程语言编写的瞬时匿名函数。

该代码块会被当作一个不带参数、返回void的函数体。它只会被解析并执行一次。

可选的LANGUAGE子句可以写在代码块之前，也可以写在之后。

参数

code

要执行的过程语言代码。与CREATE FUNCTION一样，它必须以字符串字面值形式指定。建议使用美元引用的字符串字面值。

lang_name

编写该代码所用的过程语言名称。如果省略，默认值为plpgsql。

注解

要使用的过程语言必须已经通过CREATE LANGUAGE安装到当前数据库中。plpgsql默认已安装，而其他语言则不会。

用户必须拥有该过程语言的USAGE权限；如果该语言是不受信任的，则必须是超级用户。这与用该语言创建函数时的权限要求相同。

示例

将模式public中所有视图上的全部权限授予角色webuser：

```
DO $$DECLARE r record;
BEGIN
    FOR r IN SELECT table_schema, table_name FROM information_
schema.tables
             WHERE table_type = 'VIEW' AND table_schema = 'public'
    LOOP
        EXECUTE 'GRANT ALL ON ' || quote_ident(r.table_schema) ||
 '.' || quote_ident(r.table_name) || ' TO webuser';
    END LOOP;
END$$;
```

兼容性

SQL 标准中没有DO语句。

另见

CREATE LANGUAGE

DROP AGGREGATE

DROP AGGREGATE — 移除一个聚合函数

大纲

```
DROP AGGREGATE [ IF EXISTS ] name ( argtype [ , ... ] ) [ CASCADE |
 RESTRICT ]
```

描述

`DROP AGGREGATE`移除一个现有聚合函数。要执行此命令，当前用户必须是该聚合函数的拥有者。

参数

`IF EXISTS`

如果该聚合函数不存在，则不要抛出错误。这种情况下会发出一个提示。

`name`

现有聚合函数的名称（可以被模式限定）。

`argtype`

聚合函数作用于其上的输入数据类型。要引用一个零参数聚合函数，请在参数说明列表的位置写`*`。

`CASCADE`

自动删除依赖于该聚合函数的对象。

`RESTRICT`

如果有任何对象依赖于该聚合函数，则拒绝删除它。这是默认值。

示例

要移除用于类型`integer`的聚合函数`myavg`：

```
DROP AGGREGATE myavg(integer);
```

兼容性

SQL 标准中没有`DROP AGGREGATE`语句。

另见

ALTER AGGREGATE, CREATE AGGREGATE

DROP CAST

DROP CAST — 移除一个类型转换

大纲

```
DROP CAST [ IF EXISTS ] (source_type AS target_type) [ CASCADE |
 RESTRICT ]
```

描述

`DROP CAST`移除一个先前定义的类型转换。

要删除一种类型转换，你必须拥有其源数据类型或目标数据类型。
这与创建该类型转换所需的权限相同。

参数

`IF EXISTS`

如果该类型转换不存在，则不要抛出错误。这种情况下会发出一条提示。

`source_type`

该类型转换的源数据类型的名称。

`target_type`

该类型转换的目标数据类型的名称。

`CASCADE`
`RESTRICT`

这些关键字没有任何作用，因为没有对象依赖于类型转换。

示例

要删除从类型`text`到类型`int`的类型转换：

```
DROP CAST (text AS int);
```

兼容性

`DROP CAST`命令符合 SQL 标准。

另见

CREATE CAST

DROP COLLATION

DROP COLLATION — 删除一个排序规则

大纲

```
DROP COLLATION [ IF EXISTS ] name [ CASCADE | RESTRICT ]
```

描述

`DROP COLLATION`删除一个先前定义的排序规则。要删除一个排序规则，你必须拥有该排序规则。

参数

`IF EXISTS`

如果该排序规则不存在，则不要抛出错误。这种情况下会发出一个提示。

`name`

排序规则的名称。该名称可以使用模式限定。

`CASCADE`

自动删除依赖于该排序规则的对象。

`RESTRICT`

如果有任何对象依赖于该排序规则，则拒绝删除该排序规则。这是默认值。

示例

删除名为`german`的排序规则：

```
DROP COLLATION german;
```

兼容性

`DROP COLLATION`命令符合SQL标准，但`IF EXISTS`选项是PostgreSQL扩展。

另见

ALTER COLLATION, CREATE COLLATION

DROP CONVERSION

DROP CONVERSION — 移除一个转换

大纲

```
DROP CONVERSION [ IF EXISTS ] name [ CASCADE | RESTRICT ]
```

描述

`DROP CONVERSION`移除一个先前定义的转换。要能够删除一个转换，你必须拥有该转换。

参数

`IF EXISTS`

如果该转换不存在，则不要抛出错误。这种情况下会发出一个提示。

`name`

转换的名称。转换名可以是模式限定的。

`CASCADE`
`RESTRICT`

这些关键字没有任何作用，因为没有对象依赖于转换。

示例

移除名为`myname`的转换：

```
DROP CONVERSION myname;
```

兼容性

SQL 标准中没有`DROP CONVERSION`语句，但有一个 `DROP TRANSLATION`语句，它与`CREATE TRANSLATION`语句配套，而后者又类似于 PostgreSQL 中的`CREATE CONVERSION`语句。

另见

ALTER CONVERSION, CREATE CONVERSION

DROP DATABASE

DROP DATABASE — 移除一个数据库

大纲

```
DROP DATABASE [ IF EXISTS ] name
```

描述

`DROP DATABASE`删除一个数据库。它会移除数据库的系统目录条目，并删除包含数据的目录。只有数据库拥有者才能执行此命令。此外，当你或任何其他人连接到目标数据库时，都不能执行它。（请连接到`postgres`或任何其他数据库来发出此命令。）

`DROP DATABASE`无法撤销。请谨慎使用！

参数

`IF EXISTS`

如果该数据库不存在，则不要抛出错误。这种情况下会发出一个提示。

name

要移除的数据库名称。

注解

`DROP DATABASE`不能在事务块内执行。

由于该命令不能在连接到目标数据库时执行，因此改用程序dropdb可能更方便，它是此命令的一个包装器。

兼容性

SQL 标准中没有`DROP DATABASE`语句。

另见

CREATE DATABASE

DROP DOMAIN

DROP DOMAIN — 移除一个域

大纲

```
DROP DOMAIN [ IF EXISTS ] name [, ...] [ CASCADE | RESTRICT ]
```

描述

`DROP DOMAIN`移除一个域。只有域的拥有者才能移除它。

参数

`IF EXISTS`

如果该域不存在，则不会抛出错误，而是发出一条提示。

`name`

要移除的现有域的名称（可以是模式限定的）。

`CASCADE`

自动删除依赖于该域的对象（例如表列）。

`RESTRICT`

如果有任何对象依赖于该域，则拒绝删除它。这是默认值。

示例

要移除域`box`：

```
DROP DOMAIN box;
```

兼容性

此命令符合 SQL 标准，但`IF EXISTS`选项是 PostgreSQL扩展。

另见

CREATE DOMAIN, ALTER DOMAIN

DROP EXTENSION

DROP EXTENSION — 移除一个扩展

大纲

```
DROP EXTENSION [ IF EXISTS ] name [, ...] [ CASCADE | RESTRICT ]
```

描述

`DROP EXTENSION`从数据库中移除扩展。删除扩展也会删除它所包含的对象。

要使用`DROP EXTENSION`，必须拥有该扩展。

参数

`IF EXISTS`

如果该扩展不存在，则不要抛出错误。这种情况下会发出一个提示。

`name`

一个已安装扩展的名称。

`CASCADE`

自动删除依赖于该扩展的对象。

`RESTRICT`

如果有任何对象依赖于扩展，则拒绝删除它（其自身成员对象以及同一`DROP`命令中列出的其他扩展除外）。这是默认行为。

示例

要从当前数据库中移除扩展`hstore`：

```
DROP EXTENSION hstore;
```

如果数据库中使用了`hstore`的任何对象，例如某些表具有`hstore`类型的列，则该命令会失败。加上`CASCADE`选项可以强制一并移除这些依赖对象。

兼容性

`DROP EXTENSION`是PostgreSQL扩展。

另见

CREATE EXTENSION, ALTER EXTENSION

DROP FOREIGN DATA WRAPPER

DROP FOREIGN DATA WRAPPER — 移除一个外部数据包装器

大纲

```
DROP FOREIGN DATA WRAPPER [ IF EXISTS ] name [ CASCADE | RESTRICT ]
```

描述

`DROP FOREIGN DATA WRAPPER`移除一个现有的外部数据包装器。要执行此命令，当前用户必须是该外部数据包装器的拥有者。

参数

`IF EXISTS`

如果该外部数据包装器不存在，则不要抛出错误。这种情况下会发出一个提示。

`name`

现有外部数据包装器的名称。

`CASCADE`

自动删除依赖于该外部数据包装器的对象（例如服务器）。

`RESTRICT`

如果有任何对象依赖于该外部数据包装器，则拒绝删除它。这是默认值。

示例

删除外部数据包装器`dbi`：

```
DROP FOREIGN DATA WRAPPER dbi;
```

兼容性

`DROP FOREIGN DATA WRAPPER`符合 ISO/IEC 9075-9（SQL/MED）。`IF EXISTS`子句是PostgreSQL扩展。

另见

CREATE FOREIGN DATA WRAPPER, ALTER FOREIGN DATA WRAPPER

DROP FOREIGN TABLE

DROP FOREIGN TABLE — 移除一个外部表

大纲

```
DROP FOREIGN TABLE [ IF EXISTS ] name [, ...] [ CASCADE |
 RESTRICT ]
```

描述

`DROP FOREIGN TABLE`移除一个外部表。只有外部表的拥有者才能移除它。

参数

`IF EXISTS`

如果该外部表不存在，则不要抛出错误。这种情况下会发出一个提示。

name

要删除的外部表名称（可以被模式限定）。

`CASCADE`

自动删除依赖于该外部表的对象（例如视图）。

`RESTRICT`

如果有任何对象依赖于该外部表，则拒绝删除它。这是默认值。

示例

要销毁两个外部表`films`和`distributors`：

```
DROP FOREIGN TABLE films, distributors;
```

兼容性

此命令符合 ISO/IEC 9075-9（SQL/MED），但标准每条命令只允许删除一个外部表，而`IF EXISTS`选项是PostgreSQL扩展。

另见

ALTER FOREIGN TABLE, CREATE FOREIGN TABLE

DROP FUNCTION

DROP FUNCTION — 移除一个函数

大纲

```
DROP FUNCTION [ IF EXISTS ] name ( [ [ argmode ] [ argname ]
 argtype [, ...] ] )
    [ CASCADE | RESTRICT ]
```

描述

`DROP FUNCTION`移除一个现有函数的定义。要执行此命令，用户必须是该函数的拥有者。必须指定该函数的参数类型，因为可能存在多个名称相同但参数列表不同的函数。

参数

`IF EXISTS`

如果该函数不存在，则不要抛出错误。这种情况下会发出一个提示。

`name`

现有函数的名称（可以被模式限定）。

`argmode`

参数的模式：`IN`、`OUT`、`INOUT`或`VARIADIC`。如果省略，默认值为`IN`。注意，`DROP FUNCTION`实际上并不关注`OUT`参数，因为确定函数标识只需要输入参数。因此，只列出`IN`、`INOUT`和 `VARIADIC`参数就足够了。

`argname`

一个参数的名称。注意，`DROP FUNCTION` 实际上并不关注参数名，因为确定函数标识只需要参数数据类型。

`argtype`

函数参数的数据类型（如果有，可以被模式限定）。

`CASCADE`

自动删除依赖于该函数的对象（例如操作符或触发器）。

`RESTRICT`

如果有任何对象依赖于该函数，则拒绝删除它。这是默认值。

示例

此命令删除平方根函数：

```
DROP FUNCTION sqrt(integer);
```

兼容性

`DROP FUNCTION`语句在 SQL 标准中有定义，但与该命令不兼容。

另见

CREATE FUNCTION, ALTER FUNCTION

DROP GROUP

DROP GROUP — 移除一个数据库角色

大纲

```
DROP GROUP [ IF EXISTS ] name [, ...]
```

描述

`DROP GROUP` 现在是DROP ROLE的别名。

兼容性

SQL 标准中没有 `DROP GROUP` 语句。

另见

DROP ROLE

DROP INDEX

DROP INDEX — 移除一个索引

大纲

```
DROP INDEX [ CONCURRENTLY ] [ IF EXISTS ] name [, ...] [ CASCADE |
 RESTRICT ]
```

描述

`DROP INDEX`从数据库系统中删除一个现有索引。要执行此命令，你必须是该索引的拥有者。

参数

`CONCURRENTLY`

删除索引时，不阻塞该索引所在表上的并发查询、插入、更新和删除。普通的`DROP INDEX` 会对该表获取独占锁，阻塞其他访问，直到索引删除完成。使用该选项时，命令会改为等待相冲突的事务完成。

使用此选项时有若干限制需要注意。只能指定一个索引名，并且不支持 `CASCADE`选项。（因此，支撑`UNIQUE` 或`PRIMARY KEY`约束的索引不能以这种方式删除。）另外，普通的`DROP INDEX`可以在事务块内执行，但`DROP INDEX CONCURRENTLY`不可以。

`IF EXISTS`

如果该索引不存在，则不要抛出错误。这种情况下会发出一条提示。

name

要移除的索引名称（可以被模式限定）。

`CASCADE`

自动删除依赖于该索引的对象。

`RESTRICT`

如果有任何对象依赖于该索引，则拒绝删除它。这是默认值。

示例

此命令会删除索引`title_idx`：

```
DROP INDEX title_idx;
```

兼容性

`DROP INDEX`是PostgreSQL 的一种语言扩展。SQL 标准中没有关于索引的规定。

另见

CREATE INDEX

DROP LANGUAGE

DROP LANGUAGE — 移除一种过程语言

大纲

```
DROP [ PROCEDURAL ] LANGUAGE [ IF EXISTS ] name [ CASCADE |
 RESTRICT ]
```

描述

`DROP LANGUAGE`移除一种先前注册的过程语言的定义。要使用`DROP LANGUAGE`，必须是超级用户或该语言的所有者。

> **注意**
>
> 从PostgreSQL 9.1 起，大多数过程语言都已被实现为“扩展”，因此应当使用DROP EXTENSION而不是`DROP LANGUAGE`来移除它们。

参数

`IF EXISTS`

如果该语言不存在，则不要抛出错误。这种情况下会发出一个提示。

name

现有过程语言的名称。为向后兼容，名称可以用单引号括起。

`CASCADE`

自动删除依赖于该语言的对象（例如以该语言编写的函数）。

`RESTRICT`

如果有任何对象依赖于该语言，则拒绝删除它。这是默认值。

示例

此命令移除过程语言`plsample`：

```
DROP LANGUAGE plsample;
```

兼容性

SQL 标准中没有`DROP LANGUAGE`语句。

另见

ALTER LANGUAGE, CREATE LANGUAGE, droplang

DROP OPERATOR

DROP OPERATOR — 移除一个操作符

大纲

```
DROP OPERATOR [ IF EXISTS ] name ( { left_type | NONE } , { right_
type | NONE } ) [ CASCADE | RESTRICT ]
```

描述

`DROP OPERATOR`从数据库系统中删除一个现有操作符。要执行此命令，你必须是该操作符的拥有者。

参数

`IF EXISTS`

如果该操作符不存在，则不要抛出错误。这种情况下会发出一条提示。

name

一个现有操作符的名称（可以是模式限定的）。

left_type

操作符左操作数的数据类型；如果该操作符没有左操作数，请写 `NONE`。

right_type

操作符右操作数的数据类型；如果操作符没有右操作数，请写`NONE`。

`CASCADE`

自动删除依赖于该操作符的对象。

`RESTRICT`

如果有任何对象依赖于该操作符，则拒绝删除它。这是默认值。

示例

删除幂操作符`a^b`，其类型为 `integer`：

```
DROP OPERATOR ^ (integer, integer);
```

删除前缀一元按位取反操作符 `~b`，其类型为 `bit`：

```
DROP OPERATOR ~ (none, bit);
```

删除后缀一元阶乘操作符 `x!`，其类型为 `bigint`：

```
DROP OPERATOR ! (bigint, none);
```

兼容性

SQL 标准中没有`DROP OPERATOR`语句。

另见

CREATE OPERATOR, ALTER OPERATOR

DROP OPERATOR CLASS

DROP OPERATOR CLASS — 移除一个操作符类

大纲

```
DROP OPERATOR CLASS [ IF EXISTS ] name USING index_method [ CASCADE
 | RESTRICT ]
```

描述

`DROP OPERATOR CLASS`删除一个现有操作符类。要执行此命令，你必须是该操作符类的拥有者。

`DROP OPERATOR CLASS`不会删除该类所引用的任何操作符或函数。如果有索引依赖于该操作符类，你将需要指定`CASCADE` 才能完成删除。

参数

`IF EXISTS`

如果该操作符类不存在，则不要抛出错误。这种情况下会发出一个提示。

`name`

一个现有操作符类的名称（可以被模式限定）。

`index_method`

该操作符类所属索引访问方法的名称。

`CASCADE`

自动删除依赖于该操作符类的对象。

`RESTRICT`

如果有任何对象依赖于该操作符类，则拒绝删除它。这是默认值。

注解

`DROP OPERATOR CLASS`不会删除包含该类的操作符族，即使该族中已经没有任何其他成员（尤其是在该族由`CREATE OPERATOR CLASS`隐式创建的情况下）。空的操作符族并无害处，但为了整洁起见，你也许会希望用 `DROP OPERATOR FAMILY`删除该族；或者更好的做法是，一开始就直接使用`DROP OPERATOR FAMILY`。

示例

移除 B-树操作符类`widget_ops`：

```
DROP OPERATOR CLASS widget_ops USING btree;
```

如果仍有索引使用该操作符类，此命令将不会成功。加上`CASCADE`
可以在删除操作符类的同时删除这类索引。

兼容性

SQL 标准中没有`DROP OPERATOR CLASS`语句。

另见

ALTER OPERATOR CLASS, CREATE OPERATOR CLASS, DROP OPERATOR FAMILY

DROP OPERATOR FAMILY

DROP OPERATOR FAMILY — 移除一个操作符族

大纲

```
DROP OPERATOR FAMILY [ IF EXISTS ] name USING index_method [
 CASCADE | RESTRICT ]
```

描述

`DROP OPERATOR FAMILY`删除一个现有的操作符族。要执行此命令，你必须是该操作符族的拥有者。

`DROP OPERATOR FAMILY`还会删除该族中包含的所有操作符类，但不会删除该族引用的任何操作符或函数。如果有任何索引依赖于该族内的操作符类，你需要指定`CASCADE`才能完成删除。

参数

`IF EXISTS`

如果该操作符族不存在，则不要抛出错误。这种情况下会发出一个提示。

`name`

一个现有操作符族的名称（可以是模式限定的）。

`index_method`

该操作符族所对应的索引访问方法名称。

`CASCADE`

自动删除依赖于该操作符族的对象。

`RESTRICT`

如果有任何对象依赖于该操作符族，则拒绝删除它。这是默认值。

示例

移除 B-tree 操作符族`float_ops`：

```
DROP OPERATOR FAMILY float_ops USING btree;
```

如果仍有索引使用该族内的操作符类，此命令将不会成功。加上`CASCADE`可以在删除操作符族的同时删除这类索引。

兼容性

SQL 标准中没有`DROP OPERATOR FAMILY`语句。

另见

ALTER OPERATOR FAMILY, CREATE OPERATOR FAMILY, ALTER OPERATOR CLASS, CREATE OPERATOR CLASS, DROP OPERATOR CLASS

DROP OWNED

DROP OWNED — 移除一个数据库角色所拥有的数据库对象

大纲

```
DROP OWNED BY { name | CURRENT_USER | SESSION_USER } [, ...] [
 CASCADE | RESTRICT ]
```

描述

`DROP OWNED`删除当前数据库中由指定角色之一拥有的所有对象。同时，授予给这些角色的、针对当前数据库中的对象和共享对象（数据库、表空间）的任何权限也会被撤销。

参数

name

一个角色的名称；该角色拥有的对象将被删除，并且授予给该角色的权限将被撤销。

`CASCADE`

自动删除依赖于该受影响对象的对象。

`RESTRICT`

如果有其他数据库对象依赖于某个受影响对象，则拒绝删除该角色拥有的对象。这是默认值。

注解

`DROP OWNED`经常被用来为移除一个或多个角色做准备。因为`DROP OWNED`只影响当前数据库中的对象，所以通常需要在包含待移除角色所拥有对象的每个数据库中执行此命令。

使用`CASCADE`选项可能会使该命令递归到由其他用户拥有的对象。

REASSIGN OWNED命令提供了另一种选择，它会重新分配由一个或多个角色拥有的所有数据库对象的拥有关系。不过，`REASSIGN OWNED`不处理针对其他对象的权限。

由这些角色拥有的数据库和表空间不会被移除。

更多讨论请见第 20.4 节。

兼容性

`DROP OWNED`命令是PostgreSQL扩展。

另见

REASSIGN OWNED, DROP ROLE

DROP ROLE

DROP ROLE — 移除一个数据库角色

大纲

```
DROP ROLE [ IF EXISTS ] name [, ...]
```

描述

`DROP ROLE`移除指定的角色。要删除超级用户角色，你自己也必须是超级用户；要删除非超级用户角色，必须拥有`CREATEROLE`权限。

如果某角色仍在该集簇的任何数据库中被引用，就不能移除它，否则会报错。
在删除角色之前，必须删除它拥有的所有对象（或重新分配其拥有关系），
并撤销该角色在其他对象上被授予的任何权限。REASSIGN OWNED和DROP OWNED命令可用于此目的；更多讨论参见第 20.4 节。

不过，不必手工移除与该角色相关的角色成员资格；`DROP ROLE`会自动撤销目标角色在其他角色中的任何成员资格，以及其他角色在目标角色中的任何成员资格。
其他角色不会被删除，也不会受到影响。

参数

`IF EXISTS`

如果该角色不存在，则不要抛出错误。这种情况下会发出一个提示。

name

要移除的角色名称。

注解

PostgreSQL包含一个程序dropuser，
它与此命令具有相同功能（实际上它就是调用此命令），但可以从命令 shell 中运行。

示例

删除一个角色：

```
DROP ROLE jonathan;
```

兼容性

SQL 标准定义了`DROP ROLE`，但它一次只允许删除一个角色，并且规定的权限要求也不同于PostgreSQL。

另见

CREATE ROLE, ALTER ROLE, SET ROLE

DROP RULE

DROP RULE — 移除一条重写规则

大纲

```
DROP RULE [ IF EXISTS ] name ON table_name [ CASCADE | RESTRICT ]
```

描述

`DROP RULE`移除一条重写规则。

参数

`IF EXISTS`

如果该规则不存在，则不要抛出错误。这种情况下会发出一个提示。

`name`

要移除的规则名称。

`table_name`

该规则适用的表或视图名称（可以是模式限定的）。

`CASCADE`

自动删除依赖于该规则的对象。

`RESTRICT`

如果有任何对象依赖于该规则，则拒绝删除它。这是默认值。

示例

要移除重写规则`newrule`：

```
DROP RULE newrule ON mytable;
```

兼容性

SQL 标准中没有`DROP RULE`语句。

另见

CREATE RULE

DROP SCHEMA

DROP SCHEMA — 移除一个模式

大纲

```
DROP SCHEMA [ IF EXISTS ] name [, ...] [ CASCADE | RESTRICT ]
```

描述

`DROP SCHEMA`从数据库中移除一个或多个模式。

一个模式只能由其拥有者或超级用户删除。注意，
即使该模式的拥有者并不拥有模式内的某些对象，
也仍然可以删除该模式（从而删除其中的全部对象）。

参数

`IF EXISTS`

如果该模式不存在，则不要抛出错误。这种情况下会发出一个提示。

`name`

模式名称。

`CASCADE`

自动删除模式中包含的对象（表、函数等）。

`RESTRICT`

如果该模式中含有任何对象，则拒绝删除它。这是默认值。

示例

要从数据库中移除模式`mystuff`及其包含的全部内容：

```
DROP SCHEMA mystuff CASCADE;
```

兼容性

`DROP SCHEMA`完全符合 SQL 标准，但标准每条命令只允许删除一个模式，另外`IF EXISTS`选项是PostgreSQL扩展。

另见

ALTER SCHEMA, CREATE SCHEMA

DROP SEQUENCE

DROP SEQUENCE — 移除一个序列

大纲

```
DROP SEQUENCE [ IF EXISTS ] name [, ...] [ CASCADE | RESTRICT ]
```

描述

`DROP SEQUENCE`移除序列。只有该序列的拥有者或超级用户才能删除它。

参数

`IF EXISTS`

如果该序列不存在，则不要抛出错误。这种情况下会发出一个提示。

`name`

要删除的序列名称（可以是模式限定的）。

`CASCADE`

自动删除依赖于该序列的对象。

`RESTRICT`

如果有任何对象依赖于该序列，则拒绝删除它。这是默认值。

示例

要移除序列`serial`：

```
DROP SEQUENCE serial;
```

兼容性

`DROP SEQUENCE`符合SQL标准，但标准每条命令只允许删除一个序列，而`IF EXISTS`选项是PostgreSQL扩展。

另见

CREATE SEQUENCE, ALTER SEQUENCE

DROP SERVER

DROP SERVER — 移除一个外部服务器描述符

大纲

```
DROP SERVER [ IF EXISTS ] name [ CASCADE | RESTRICT ]
```

描述

`DROP SERVER`移除一个现有外部服务器描述符。要执行此命令，当前用户必须是该服务器的拥有者。

参数

`IF EXISTS`

如果该服务器不存在，则不要抛出错误。这种情况下会发出一个提示。

`name`

现有服务器的名称。

`CASCADE`

自动删除依赖于该服务器的对象（例如用户映射）。

`RESTRICT`

如果有任何对象依赖于该服务器，则拒绝删除它。这是默认值。

示例

如果服务器`foo`存在，则删除它：

```
DROP SERVER IF EXISTS foo;
```

兼容性

`DROP SERVER`符合 ISO/IEC 9075-9（SQL/MED）。`IF EXISTS`子句是PostgreSQL扩展。

另见

CREATE SERVER, ALTER SERVER

DROP TABLE

DROP TABLE — 移除一个表

大纲

```
DROP TABLE [ IF EXISTS ] name [, ...] [ CASCADE | RESTRICT ]
```

描述

`DROP TABLE`从数据库中移除表。只有表拥有者、模式拥有者以及超级用户才能删除表。若要清空表中的行而不销毁该表，请使用DELETE或TRUNCATE。

`DROP TABLE`总会移除目标表上存在的所有索引、规则、触发器和约束。但是，要删除一个被视图或另一张表的外键约束引用的表，必须指定`CASCADE`。（`CASCADE`会将依赖于该表的视图整个删除，但在外键场景下，它只会删除外键约束，不会把另一张表整体删除。）

参数

`IF EXISTS`

如果该表不存在，则不要抛出错误。这种情况下会发出一个提示。

name

要删除的表名称（可选地使用模式限定）。

`CASCADE`

自动删除依赖于该表的对象（例如视图）。

`RESTRICT`

如果有任何对象依赖于该表，则拒绝删除它。这是默认值。

示例

要删除两个表`films`和`distributors`：

```
DROP TABLE films, distributors;
```

兼容性

此命令符合 SQL 标准，只是标准每条命令只允许删除一个表，且`IF EXISTS`选项是PostgreSQL扩展。

另见

ALTER TABLE, CREATE TABLE

DROP TABLESPACE

DROP TABLESPACE — 移除一个表空间

大纲

```
DROP TABLESPACE [ IF EXISTS ] name
```

描述

`DROP TABLESPACE`从系统中移除一个表空间。

一个表空间只能由其拥有者或超级用户删除。删除前，该表空间中必须不含任何数据库对象。即使当前数据库中没有对象正在使用该表空间，其他数据库中的对象也可能仍然驻留在该表空间中。此外，如果任何活动会话的temp_tablespaces设置中列出了该表空间，那么由于该表空间中可能仍有临时文件，`DROP`也可能失败。

参数

`IF EXISTS`

如果该表空间不存在，则不要抛出错误。这种情况下会发出一个提示。

name

表空间的名称。

注解

`DROP TABLESPACE`不能在事务块内执行。

示例

要从系统中移除表空间`mystuff`：

```
DROP TABLESPACE mystuff;
```

兼容性

`DROP TABLESPACE`是一种PostgreSQL扩展。

另见

CREATE TABLESPACE, ALTER TABLESPACE

DROP TEXT SEARCH CONFIGURATION

DROP TEXT SEARCH CONFIGURATION — 移除一个文本检索配置

大纲

```
DROP TEXT SEARCH CONFIGURATION [ IF EXISTS ] name [ CASCADE |
 RESTRICT ]
```

描述

`DROP TEXT SEARCH CONFIGURATION`删除一个现有文本检索配置。要执行此命令，你必须是该配置的拥有者。

参数

`IF EXISTS`

如果该文本检索配置不存在，则不要抛出错误。这种情况下会发出一个提示。

`name`

现有文本检索配置的名称（可以被模式限定）。

`CASCADE`

自动删除依赖于该文本检索配置的对象。

`RESTRICT`

如果有任何对象依赖于该文本检索配置，则拒绝删除该配置。这是默认值。

示例

移除文本检索配置`my_english`：

```
DROP TEXT SEARCH CONFIGURATION my_english;
```

如果已有索引在`to_tsvector`调用中引用了该配置，此命令将不会成功。加上`CASCADE`可以在删除该文本检索配置的同时删除这类索引。

兼容性

SQL 标准中没有`DROP TEXT SEARCH CONFIGURATION`语句。

另见

ALTER TEXT SEARCH CONFIGURATION, CREATE TEXT SEARCH CONFIGURATION

DROP TEXT SEARCH DICTIONARY

DROP TEXT SEARCH DICTIONARY — 移除一个文本检索词典

大纲

```
DROP TEXT SEARCH DICTIONARY [ IF EXISTS ] name [ CASCADE |
 RESTRICT ]
```

描述

`DROP TEXT SEARCH DICTIONARY`删除一个现有文本检索词典。要执行此命令，你必须是该词典的拥有者。

参数

`IF EXISTS`

如果该文本检索词典不存在，则不要抛出错误。这种情况下会发出一个提示。

`name`

一个现有文本检索词典的名称（可以被模式限定）。

`CASCADE`

自动删除依赖于该文本检索词典的对象。

`RESTRICT`

如果有任何对象依赖于该文本检索词典，则拒绝删除它。这是默认值。

示例

移除文本检索词典`english`：

```
DROP TEXT SEARCH DICTIONARY english;
```

如果已有文本检索配置使用该词典，此命令将不会成功。加上`CASCADE`可以在删除词典的同时删除这类配置。

兼容性

SQL 标准中没有`DROP TEXT SEARCH DICTIONARY`语句。

另见

ALTER TEXT SEARCH DICTIONARY, CREATE TEXT SEARCH DICTIONARY

DROP TEXT SEARCH PARSER

DROP TEXT SEARCH PARSER — 移除一个文本检索解析器

大纲

```
DROP TEXT SEARCH PARSER [ IF EXISTS ] name [ CASCADE | RESTRICT ]
```

描述

`DROP TEXT SEARCH PARSER`删除一个现有文本检索解析器。要使用此命令，你必须是超级用户。

参数

`IF EXISTS`

如果该文本检索解析器不存在，则不要抛出错误。这种情况下会发出一个提示。

`name`

一个现有文本检索解析器的名称（可以被模式限定）。

`CASCADE`

自动删除依赖于该文本检索解析器的对象。

`RESTRICT`

如果有任何对象依赖于该文本检索解析器，则拒绝删除它。这是默认值。

示例

移除文本检索解析器`my_parser`：

```
DROP TEXT SEARCH PARSER my_parser;
```

如果已有文本检索配置使用该解析器，此命令将不会成功。加上`CASCADE`可以在删除解析器的同时删除这类配置。

兼容性

SQL 标准中没有`DROP TEXT SEARCH PARSER`语句。

另见

ALTER TEXT SEARCH PARSER, CREATE TEXT SEARCH PARSER

DROP TEXT SEARCH TEMPLATE

DROP TEXT SEARCH TEMPLATE — 移除一个文本检索模板

大纲

```
DROP TEXT SEARCH TEMPLATE [ IF EXISTS ] name [ CASCADE | RESTRICT ]
```

描述

`DROP TEXT SEARCH TEMPLATE`删除一个现有文本检索模板。要使用此命令，你必须是超级用户。

参数

`IF EXISTS`

如果该文本检索模板不存在，则不要抛出错误。这种情况下会发出一个提示。

`name`

现有文本检索模板的名称（可以被模式限定）。

`CASCADE`

自动删除依赖于该文本检索模板的对象。

`RESTRICT`

如果有任何对象依赖于该文本检索模板，则拒绝删除该模板。这是默认值。

示例

移除文本检索模板`thesaurus`：

```
DROP TEXT SEARCH TEMPLATE thesaurus;
```

如果已有文本检索词典使用该模板，此命令将不会成功。加上`CASCADE` 可以在删除该模板的同时删除这类词典。

兼容性

SQL 标准中没有`DROP TEXT SEARCH TEMPLATE`语句。

另见

ALTER TEXT SEARCH TEMPLATE, CREATE TEXT SEARCH TEMPLATE

DROP TRIGGER

DROP TRIGGER — 移除一个触发器

大纲

```
DROP TRIGGER [ IF EXISTS ] name ON table_name [ CASCADE |
 RESTRICT ]
```

描述

`DROP TRIGGER`移除一个现有触发器的定义。要执行此命令，当前用户必须是定义该触发器的表的所有者。

参数

`IF EXISTS`

如果该触发器不存在，则不要抛出错误。这种情况下会发出一个提示。

`name`

要移除的触发器的名称。

`table_name`

定义该触发器的表的名称（可以是模式限定的）。

`CASCADE`

自动删除依赖于该触发器的对象。

`RESTRICT`

如果有任何对象依赖于该触发器，则拒绝删除它。这是默认值。

示例

删除表`films`上的触发器`if_dist_exists`：

```
DROP TRIGGER if_dist_exists ON films;
```

兼容性

PostgreSQL中的`DROP TRIGGER`语句与 SQL 标准不兼容。在 SQL 标准中，触发器名称不是表局部的，因此该命令仅为 `DROP TRIGGER name`。

参见

CREATE TRIGGER

DROP TYPE

DROP TYPE — 移除一个数据类型

大纲

```
DROP TYPE [ IF EXISTS ] name [, ...] [ CASCADE | RESTRICT ]
```

描述

`DROP TYPE`移除一个用户定义的数据类型。只有该类型的拥有者才能移除它。

参数

`IF EXISTS`

如果该类型不存在，则不会抛出错误，而是发出一条提示。

`name`

要移除的数据类型的名称（可以是模式限定的）。

`CASCADE`

自动删除依赖于该类型的对象（例如表列、函数和操作符）。

`RESTRICT`

如果有任何对象依赖于该类型，则拒绝删除它。这是默认值。

示例

要移除数据类型`box`：

```
DROP TYPE box;
```

兼容性

此命令与 SQL 标准中的对应命令相似，但`IF EXISTS`选项是 PostgreSQL扩展。不过请注意，`CREATE TYPE`命令的大部分内容，以及 PostgreSQL中的数据类型扩展机制，都不同于 SQL 标准。

另见

ALTER TYPE, CREATE TYPE

DROP USER

DROP USER — 移除一个数据库角色

大纲

```
DROP USER [ IF EXISTS ] name [, ...]
```

描述

`DROP USER`只是DROP ROLE的另一种拼写。

兼容性

`DROP USER`语句是 PostgreSQL的一种扩展。SQL 标准将用户的定义留给具体实现决定。

另见

DROP ROLE

DROP USER MAPPING

DROP USER MAPPING — 删除外部服务器的用户映射

大纲

```
DROP USER MAPPING [ IF EXISTS ] FOR { user_name | USER | CURRENT_
USER | PUBLIC } SERVER server_name
```

描述

`DROP USER MAPPING`删除外部服务器上的一个现有用户映射。

外部服务器的拥有者可以删除该服务器上任何用户的用户映射。此外，
如果某个用户已被授予该服务器上的`USAGE`权限，
那么该用户也可以删除其自己用户名对应的用户映射。

参数

`IF EXISTS`

如果该用户映射不存在，则不要报错。这种情况下会发出一个提示。

`user_name`

该映射的用户名。`CURRENT_USER`和`USER`都匹配当前用户的名称。`PUBLIC`
用于匹配系统中当前及未来的所有用户名。

`server_name`

该用户映射所属服务器的名称。

示例

如果用户`bob`在服务器`foo`上的用户映射存在，则将其删除：

```
DROP USER MAPPING IF EXISTS FOR bob SERVER foo;
```

兼容性

`DROP USER MAPPING`符合 ISO/IEC 9075-9（SQL/MED）。`IF EXISTS`子句是PostgreSQL扩展。

另见

CREATE USER MAPPING, ALTER USER MAPPING

DROP VIEW

DROP VIEW — 移除一个视图

大纲

```
DROP VIEW [ IF EXISTS ] name [, ...] [ CASCADE | RESTRICT ]
```

描述

`DROP VIEW`删除一个现有视图。要执行此命令，你必须是该视图的拥有者。

参数

`IF EXISTS`

如果该视图不存在，则不要抛出错误。这种情况下会发出一个提示。

name

要移除的视图名称（可选地使用模式限定）。

`CASCADE`

自动删除依赖于该视图的对象（例如其他视图）。

`RESTRICT`

如果有任何对象依赖于该视图，则拒绝删除它。这是默认值。

示例

这个命令将移除名为`kinds`的视图：

```
DROP VIEW kinds;
```

兼容性

此命令符合 SQL 标准，只是标准每条命令只允许删除一个视图，且`IF EXISTS`选项是PostgreSQL扩展。

另见

ALTER VIEW, CREATE VIEW

END

END — 提交当前事务

大纲

```
END [ WORK | TRANSACTION ]
```

描述

`END`提交当前事务。该事务所做的所有更改都会对其他会话可见，并且如果发生崩溃，也能保证其持久性。该命令是PostgreSQL扩展，等效于COMMIT。

参数

`WORK`
`TRANSACTION`

可选关键字，没有任何作用。

注解

使用ROLLBACK中止事务。

在事务块之外发出`END`不会造成任何影响，但会产生一条警告消息。

示例

提交当前事务，并使所有更改永久生效：

```
END;
```

兼容性

命令`END`是PostgreSQL扩展，提供了与 SQL 标准规定的COMMIT等效的功能。

另见

BEGIN, COMMIT, ROLLBACK

EXECUTE

EXECUTE — 执行一个预备语句

大纲

```
EXECUTE name [ ( parameter [, ...] ) ]
```

描述

EXECUTE用于执行一个先前创建的预备语句。由于预备语句只在一个会话期间内存在，因此该预备语句必须由当前会话中较早执行的PREPARE语句创建。

如果创建该语句的PREPARE语句指定了某些参数，那么传递给EXECUTE语句的参数集合就必须与之兼容，否则将引发错误。注意，预备语句（不同于函数）不会根据其参数的类型或个数进行重载；预备语句的名称在一个数据库会话内必须唯一。

有关预备语句的创建和用法的更多信息，请参见PREPARE。

参数

name

要执行的预备语句名称。

parameter

预备语句中某个参数的实际值。它必须是一个表达式，并且其求值结果必须与该参数的数据类型兼容，该数据类型在创建预备语句时已确定。

输出

EXECUTE返回的命令标签是该预备语句的命令标签，而不是EXECUTE。

示例

示例位于示例一节，详见PREPARE文档。

兼容性

SQL 标准包含EXECUTE语句，但它只用于嵌入式 SQL。这里的EXECUTE语句还使用了稍有不同的语法。

另见

DEALLOCATE, PREPARE

EXPLAIN

EXPLAIN — 显示一个语句的执行计划

大纲

```
EXPLAIN [ ( option [, ...] ) ] statement
EXPLAIN [ ANALYZE ] [ VERBOSE ] statement
```

其中*option*为以下之一：

```
    ANALYZE [ boolean ]
    VERBOSE [ boolean ]
    COSTS [ boolean ]
    BUFFERS [ boolean ]
    TIMING [ boolean ]
    FORMAT { TEXT | XML | JSON | YAML }
```

描述

这个命令显示PostgreSQL规划器为给定语句生成的执行计划。
执行计划会显示将如何扫描该语句引用的表 – 例如普通顺序扫描、索引扫描等 –，如果引用了多个表，还会显示将使用哪些连接算法来汇集每个输入表中的所需行。

显示结果中最关键的部分是语句执行代价的估计值，
它是规划器对运行该语句需要多长时间的猜测（以任意代价单位衡量，
但按惯例表示磁盘页面抓取次数）。实际上会显示两个数字：返回第一行之前的启动代价，
以及返回全部行的总代价。对大多数查询来说，总代价才是关键；但在某些场景中，例如
EXISTS中的子查询，
规划器会选择启动代价最小而不是总代价最小的计划（因为无论如何执行器都会在得到一行
后停止）。此外，如果你用LIMIT子句限制返回的行数，
规划器会在端点代价之间作适当插值，以估计哪个计划实际上最便宜。

ANALYZE选项会让该语句被实际执行，而不仅仅是生成计划。
随后会把实际运行时间统计信息加入显示结果，
包括每个计划节点中耗费的总时间（以毫秒计）以及它实际返回的总行数。
这有助于判断规划器的估计是否接近实际情况。

> **重要**
>
> 请记住，使用 ANALYZE 选项时，语句会被实际执行。虽然 EXPLAIN 会丢弃 SELECT 返回的任何输出，但语句的其他副作用仍会照常发生。如果希望使用 EXPLAIN ANALYZE 分析 INSERT, UPDATE, DELETE, CREATE TABLE AS 或 EXECUTE 语句而不让命令影响数据，可以采用以下方法：
>
> ```
> BEGIN;
> EXPLAIN ANALYZE ...;
> ROLLBACK;
> ```

不使用括号包围选项列表时，只能指定ANALYZE和VERBOSE选项，
而且只能按这个顺序指定。在PostgreSQL 9.0 之前，只支持这种不带括号的语法。预计所有新选项都将仅在带括号的语法中得到支持。

参数

ANALYZE

执行该命令并显示实际运行时间及其他统计信息。此参数默认为FALSE。

VERBOSE

显示有关计划的附加信息。具体包括：计划树中每个节点的输出列列表、模式限定的表名和函数名、总是用其范围表别名标注表达式中的变量，以及总是打印显示统计信息的每个触发器名称。此参数默认为FALSE。

COSTS

包含每个计划节点的估计启动代价和总代价，以及估计行数和每行的估计宽度。此参数默认为TRUE。

BUFFERS

包含缓冲区使用信息。具体来说，会包括共享块命中、读取、写脏和写出的数量，本地块命中、读取、写脏和写出的数量，以及临时块读取和写出的数量。hit表示该块在需要时已经在缓存中找到，因此避免了一次读取。共享块包含普通表和索引的数据；本地块包含临时表和索引的数据；临时块则包含排序、哈希、Materialize 计划节点等场景使用的短期工作数据。dirtied块数表示此查询修改的、先前未修改过的块数；written块数表示在查询处理期间该后端从缓存中逐出的、先前已被写脏的块数。某个上层节点显示的块数包含其所有子节点使用的块数。在文本格式中，只打印非零值。此参数只能在同时启用ANALYZE时使用。默认为FALSE。

TIMING

在输出中包含实际的启动时间以及在该节点中耗费的时间。反复读取系统时钟的开销在某些系统上可能会显著拖慢查询，因此当只需要实际行数而不需要精确时间时，把此参数设置为FALSE可能会有用。此参数只能在同时启用ANALYZE时使用。此参数默认为TRUE。

FORMAT

指定输出格式，可以是 TEXT、XML、JSON 或 YAML。非文本输出包含与文本输出相同的信息，但更容易被程序解析。此参数默认为TEXT。

boolean

指定所选选项应开启还是关闭。可以写TRUE、ON或1来启用选项，写FALSE、OFF或0来禁用它。*boolean*值也可以省略，在这种情况下假定其值为TRUE。

statement

任何你希望查看其执行计划的SELECT、INSERT、UPDATE、DELETE、VALUES、EXECUTE、DECLARE或CREATE TABLE AS语句。

输出

该命令的结果是为*statement*选择的计划的文本描述，并可选择附带执行统计信息。第 14.1 节描述了所提供的信息。

注解

为了让PostgreSQL查询规划器在优化查询时能够做出相当有根据的决策，查询中用到的所有表的pg_statistic数据都应保持最新。通常autovacuum 守护进程会自

动处理这一点。但如果某个表最近内容发生了大量变化，
你可能需要手工执行一次ANALYZE，而不是等待 autovacuum 跟上这些变化。

为了度量执行计划中每个节点的运行时代价，当前的EXPLAIN ANALYZE实现会给查询执行增加性能分析开销。因此，对一个查询运行EXPLAIN ANALYZE有时会比正常执行该查询慢得多。开销大小取决于查询的性质以及所用平台。
最坏情况出现在那些自身每次执行耗时极少的计划节点上，
以及获取当前时间的操作系统调用相对较慢的机器上。

示例

要显示一个只有单个integer列且包含 10000 行的表上的简单查询计划：

```
EXPLAIN SELECT * FROM foo;

                       QUERY PLAN
-----------------------------------------------------------
 Seq Scan on foo  (cost=0.00..155.00 rows=10000 width=4)
(1 row)
```

下面是同一查询，但使用 JSON 输出格式：

```
EXPLAIN (FORMAT JSON) SELECT * FROM foo;
           QUERY PLAN
--------------------------------
 [                              +
   {                            +
     "Plan": {                  +
       "Node Type": "Seq Scan",+
       "Relation Name": "foo", +
       "Alias": "foo",         +
       "Startup Cost": 0.00,   +
       "Total Cost": 155.00,   +
       "Plan Rows": 10000,     +
       "Plan Width": 4         +
     }                          +
   }                            +
 ]
(1 row)
```

如果存在索引，并且我们使用了带有可索引WHERE条件的查询，EXPLAIN
可能会显示不同的计划：

```
EXPLAIN SELECT * FROM foo WHERE i = 4;

                         QUERY PLAN
-------------------------------------------------------------
 Index Scan using fi on foo  (cost=0.00..5.98 rows=1 width=4)
   Index Cond: (i = 4)
(2 rows)
```

下面是同一查询，但采用 YAML 格式：

```
EXPLAIN (FORMAT YAML) SELECT * FROM foo WHERE i='4';
          QUERY PLAN
--------------------------------
 - Plan:                       +
     Node Type: "Index Scan"   +
```

```
     Scan Direction: "Forward"+
     Index Name: "fi"         +
     Relation Name: "foo"     +
     Alias: "foo"             +
     Startup Cost: 0.00       +
     Total Cost: 5.98         +
     Plan Rows: 1             +
     Plan Width: 4            +
     Index Cond: "(i = 4)"
(1 row)
```

XML 格式留给读者自行练习。

下面是同一计划，但隐藏了代价估计：

```
EXPLAIN (COSTS FALSE) SELECT * FROM foo WHERE i = 4;

        QUERY PLAN
----------------------------
 Index Scan using fi on foo
   Index Cond: (i = 4)
(2 rows)
```

下面是使用聚合函数的查询的查询计划示例：

```
EXPLAIN SELECT sum(i) FROM foo WHERE i < 10;

                            QUERY PLAN
-------------------------------------------------------------------
--
 Aggregate  (cost=23.93..23.93 rows=1 width=4)
   ->  Index Scan using fi on foo  (cost=0.00..23.92 rows=6 width=
4)
         Index Cond: (i < 10)
(3 rows)
```

下面是使用 `EXPLAIN EXECUTE` 显示预备查询执行计划的示例：

```
PREPARE query(int, int) AS SELECT sum(bar) FROM test
    WHERE id > $1 AND id < $2
    GROUP BY foo;

EXPLAIN ANALYZE EXECUTE query(100, 200);

                                                        QUERY PLAN
-------------------------------------------------------------------
-----------------------------------------------------
 HashAggregate  (cost=39.53..39.53 rows=1 width=8) (actual time=0.
661..0.672 rows=7 loops=1)
   ->  Index Scan using test_pkey on test  (cost=0.00..32.97 rows=
1311 width=8) (actual time=0.050..0.395 rows=99 loops=1)
         Index Cond: ((id > $1) AND (id < $2))
 Total runtime: 0.851 ms
(4 rows)
```

当然，这里显示的具体数字取决于所涉及表的实际内容。还要注意，由于规划器的改进，这些数字甚至所选查询策略在PostgreSQL的不同版本之间都可能发生变化。此外，`ANALYZE` 命令使用随机采样来估计数据统计信息；因此，即使表中数据的实际分布没有变化，在重新运行一次ANALYZE之后，代价估计也可能发生变化。

兼容性

SQL 标准中没有定义`EXPLAIN`语句。

另见

ANALYZE

FETCH

FETCH — 使用游标从查询中检索行

大纲

```
FETCH [ direction [ FROM | IN ] ] cursor_name
```

其中*direction*可以为空，或为以下之一：

```
    NEXT
    PRIOR
    FIRST
    LAST
    ABSOLUTE count
    RELATIVE count
    count
    ALL
    FORWARD
    FORWARD count
    FORWARD ALL
    BACKWARD
    BACKWARD count
    BACKWARD ALL
```

描述

FETCH使用先前创建的游标检索行。

游标带有一个关联位置，FETCH会使用该位置。游标位置可以位于查询结果的第一行之前、结果中的任意一行上，或者位于结果的最后一行之后。游标创建时位于第一行之前。提取若干行后，游标会定位在最近提取的那一行上。如果FETCH 越过了可用行的末尾，那么游标会留在最后一行之后；如果是向后提取，则会留在第一行之前。FETCH ALL或者 FETCH BACKWARD ALL总是会让游标位于最后一行之后或第一行之前。

NEXT、PRIOR、FIRST、LAST、ABSOLUTE、RELATIVE 这些形式会在适当地移动游标后提取一行。如果不存在这样的行，则返回空结果，并根据情况将游标定位在第一行之前或最后一行之后。

使用FORWARD和BACKWARD的形式会在向前或向后的方向上提取指定数量的行，并将游标定位在最后返回的那一行上（如果*count*超过可用行数，则定位在所有行之后或之前）。

RELATIVE 0、FORWARD 0以及 BACKWARD 0都会请求提取当前行而不移动游标，也就是重新提取最近一次提取的行。除非游标位于第一行之前或最后一行之后，否则该操作都会成功；在这两种情况下，不会返回任何行。

> **注意**
>
> 本页面描述的是 SQL 命令层面上的游标用法。如果想要在 PL/pgSQL函数中使用游标，规则会有所不同 – 请见第 39.7 节。

参数

direction

direction 定义抓取方向和要抓取的行数。它可以是以下值之一：

NEXT

提取下一行。如果省略*direction*，这将是默认值。

PRIOR

提取前一行。

FIRST

提取查询的第一行（与ABSOLUTE 1相同）。

LAST

提取查询的最后一行（与ABSOLUTE -1相同）。

ABSOLUTE *count*

提取查询的第*count*行；如果*count*为负，则提取从末尾算起的第abs(*count*)行。如果 *count*超出范围，则定位在第一行之前或最后一行之后。特别地，ABSOLUTE 0会定位在第一行之前。

RELATIVE *count*

抓取后面的第*count*行，或前面的第abs(*count*)行（当*count*为负时）。RELATIVE 0会重新抓取当前行（如果存在）。

count

提取接下来的*count*行（与FORWARD *count*相同）。

ALL

提取所有剩余行（与FORWARD ALL相同）。

FORWARD

提取下一行（与NEXT相同）。

FORWARD *count*

提取接下来的*count*行。FORWARD 0会重新提取当前行。

FORWARD ALL

提取所有剩余行。

BACKWARD

提取前一行（与PRIOR相同）。

BACKWARD *count*

提取前面的*count*行（反向扫描）。BACKWARD 0会重新提取当前行。

BACKWARD ALL

提取之前的所有行（反向扫描）。

count

*count*是一个可以带符号的整数常量，决定抓取位置或抓取行数。对于FORWARD和BACKWARD，指定负的*count*等同于交换FORWARD和BACKWARD的方向。

cursor_name

一个已打开游标的名称。

输出

成功完成时，FETCH命令会返回如下形式的命令标签：

FETCH *count*

其中*count*是提取到的行数（可能为零）。注意，在psql中，命令标签实际上不会显示，因为psql会改为显示提取到的行。

注解

如果打算使用除FETCH NEXT或带正数计数的 FETCH FORWARD之外的任何FETCH 变体，则应当在声明游标时使用SCROLL选项。对于简单查询，PostgreSQL允许从未使用SCROLL 声明的游标向后提取，但最好不要依赖这种行为。如果游标声明为 NO SCROLL，则不允许向后提取。

ABSOLUTE提取并不比通过相对移动到达目标行更快：无论如何，
底层实现都必须遍历所有中间行。
负值的绝对提取甚至更糟：必须先把查询读到末尾以找到最后一行，
然后再从那里反向遍历。不过，回卷到查询起始处（如FETCH ABSOLUTE 0）是很快的。

DECLARE用于定义游标。使用MOVE可以在不检索数据的情况下改变游标位置。

示例

下面的示例展示了如何使用游标遍历一个表：

```
BEGIN WORK;

-- 建立一个游标:
DECLARE liahona SCROLL CURSOR FOR SELECT * FROM films;

-- 从游标 liahona 中提取前 5 行:
FETCH FORWARD 5 FROM liahona;

 code  |          title          | did | date_prod  |   kind   |
 len
-------+-------------------------+-----+------------+----------+---
----
 BL101 | The Third Man           | 101 | 1949-12-23 | Drama    |
 01:44
 BL102 | The African Queen       | 101 | 1951-08-11 | Romantic |
 01:43
 JL201 | Une Femme est une Femme | 102 | 1961-03-12 | Romantic |
 01:25
```

```
 P_301 | Vertigo                 | 103 | 1958-11-14 | Action   |
 02:08
 P_302 | Becket                  | 103 | 1964-02-03 | Drama    |
 02:28

-- 提取前一行:
FETCH PRIOR FROM liahona;

 code  |  title  | did | date_prod  |  kind  |  len
-------+---------+-----+------------+--------+-------
 P_301 | Vertigo | 103 | 1958-11-14 | Action | 02:08

-- 关闭游标并结束事务:
CLOSE liahona;
COMMIT WORK;
```

兼容性

SQL 标准只为嵌入式 SQL 定义了`FETCH`。这里描述的`FETCH`变体会像 `SELECT`结果那样返回数据，而不是把数据放入主变量中。除此之外，`FETCH`与 SQL 标准完全向上兼容。

涉及`FORWARD`和`BACKWARD`的 `FETCH`形式，以及形式`FETCH` *`count`*和`FETCH ALL`（其中隐含了`FORWARD`）都是 PostgreSQL扩展。

SQL 标准只允许在游标名之前使用`FROM`；允许使用 `IN`，或者将两者都完全省略，都是扩展。

另见

CLOSE, DECLARE, MOVE

GRANT

GRANT — 定义访问权限

大纲

```
GRANT { { SELECT | INSERT | UPDATE | DELETE | TRUNCATE | REFERENCES
 | TRIGGER }
    [, ...] | ALL [ PRIVILEGES ] }
    ON { [ TABLE ] table_name [, ...]
         | ALL TABLES IN SCHEMA schema_name [, ...] }
    TO { [ GROUP ] role_name | PUBLIC } [, ...] [ WITH GRANT
 OPTION ]

GRANT { { SELECT | INSERT | UPDATE | REFERENCES } ( column_name [,
 ...] )
    [, ...] | ALL [ PRIVILEGES ] ( column_name [, ...] ) }
    ON [ TABLE ] table_name [, ...]
    TO { [ GROUP ] role_name | PUBLIC } [, ...] [ WITH GRANT
 OPTION ]

GRANT { { USAGE | SELECT | UPDATE }
    [, ...] | ALL [ PRIVILEGES ] }
    ON { SEQUENCE sequence_name [, ...]
         | ALL SEQUENCES IN SCHEMA schema_name [, ...] }
    TO { [ GROUP ] role_name | PUBLIC } [, ...] [ WITH GRANT
 OPTION ]

GRANT { { CREATE | CONNECT | TEMPORARY | TEMP } [, ...] | ALL [
 PRIVILEGES ] }
    ON DATABASE database_name [, ...]
    TO { [ GROUP ] role_name | PUBLIC } [, ...] [ WITH GRANT
 OPTION ]

GRANT { USAGE | ALL [ PRIVILEGES ] }
    ON DOMAIN domain_name [, ...]
    TO { [ GROUP ] role_name | PUBLIC } [, ...] [ WITH GRANT
 OPTION ]

GRANT { USAGE | ALL [ PRIVILEGES ] }
    ON FOREIGN DATA WRAPPER fdw_name [, ...]
    TO { [ GROUP ] role_name | PUBLIC } [, ...] [ WITH GRANT
 OPTION ]

GRANT { USAGE | ALL [ PRIVILEGES ] }
    ON FOREIGN SERVER server_name [, ...]
    TO { [ GROUP ] role_name | PUBLIC } [, ...] [ WITH GRANT
 OPTION ]

GRANT { EXECUTE | ALL [ PRIVILEGES ] }
    ON { FUNCTION function_name ( [ [ argmode ] [ arg_name ] arg_
type [, ...] ] ) [, ...]
         | ALL FUNCTIONS IN SCHEMA schema_name [, ...] }
    TO { [ GROUP ] role_name | PUBLIC } [, ...] [ WITH GRANT
 OPTION ]
```

```
GRANT { USAGE | ALL [ PRIVILEGES ] }
    ON LANGUAGE lang_name [, ...]
    TO { [ GROUP ] role_name | PUBLIC } [, ...] [ WITH GRANT
 OPTION ]

GRANT { { SELECT | UPDATE } [, ...] | ALL [ PRIVILEGES ] }
    ON LARGE OBJECT loid [, ...]
    TO { [ GROUP ] role_name | PUBLIC } [, ...] [ WITH GRANT
 OPTION ]

GRANT { { CREATE | USAGE } [, ...] | ALL [ PRIVILEGES ] }
    ON SCHEMA schema_name [, ...]
    TO { [ GROUP ] role_name | PUBLIC } [, ...] [ WITH GRANT
 OPTION ]

GRANT { CREATE | ALL [ PRIVILEGES ] }
    ON TABLESPACE tablespace_name [, ...]
    TO { [ GROUP ] role_name | PUBLIC } [, ...] [ WITH GRANT
 OPTION ]

GRANT { USAGE | ALL [ PRIVILEGES ] }
    ON TYPE type_name [, ...]
    TO { [ GROUP ] role_name | PUBLIC } [, ...] [ WITH GRANT
 OPTION ]

GRANT role_name [, ...] TO role_name [, ...] [ WITH ADMIN OPTION ]
```

描述

GRANT命令有两种基本变体：一种用于授予数据库对象（表、列、视图、外部表、序列、数据库、外部数据包装器、外部服务器、函数、过程语言、模式或表空间）上的权限，另一种用于授予角色的成员资格。这两种变体在许多方面很相似，但差别也足够大，因此分别介绍。

在数据库对象上 GRANT

这种 GRANT 命令变体将数据库对象上的特定权限授予一个或多个角色。如果此前已经授予过某些权限，新授予的权限会加到现有权限之上。

还可以选择在一个或多个模式中，对同一类型的所有对象授予权限。目前仅对表、序列和函数支持这一功能（但请注意，ALL TABLES被认为包括视图和外部表）。

关键字 PUBLIC 表示要把权限授予所有角色，包括以后可能创建的角色。PUBLIC 可以视为一个隐式定义的组，并且始终包含所有角色。任何特定角色实际拥有的权限，是直接授予给它的权限、授予给它当前所属任一角色的权限，以及授予给 PUBLIC 的权限之和。

如果指定了 WITH GRANT OPTION，权限接收者随后可以再把该权限授予其他人。没有授予选项时，接收者不能这样做。授予选项不能授予给 PUBLIC。

没有必要向对象拥有者（通常是创建它的用户）授予权限，因为拥有者默认拥有全部权限。（不过，出于安全考虑，拥有者也可以选择撤销自己的某些权限。）

删除对象或以任何方式更改其定义的权利，不被视为一种可授予的权限；它是拥有者固有的，不能被授予或撤销。（不过，可以通过授予或撤销拥有该对象的角色的成员资格，获得类似效果；见下文。）拥有者还隐式拥有该对象上的全部授予选项。

PostgreSQL 会将某些类型对象上的默认权限授予PUBLIC。默认情况下，不会在表、表列、序列、外部数据包装器、外部服务器、大对象、模式或表空间上向PUBLIC授予任何权限。

对于其他类型的对象，授予PUBLIC的默认权限如下：数据库上的CONNECT和TEMPORARY（创建临时表）权限；函数上的EXECUTE权限；以及语言和数据类型（包括域）上的USAGE权限。当然，对象所有者可以使用REVOKE撤销默认权限和显式授予的权限。（为了尽可能安全，请在创建对象的同一事务中执行REVOKE，这样其他用户就没有可以使用该对象的时间窗口。）此外，还可以使用ALTER DEFAULT PRIVILEGES命令更改这些初始默认权限设置。

可用权限如下：

SELECT

允许使用SELECT读取指定表、视图或序列的任何列，或列出的特定列。还允许使用COPY TO。在UPDATE或DELETE中引用现有列值时也需要此权限。对于序列，此权限还允许使用currval函数。对于大对象，此权限允许读取该对象。

INSERT

允许使用INSERT向指定表插入新行。如果列出了特定列，则INSERT命令只能给这些列赋值（其他列因此会获得默认值）。还允许使用COPY FROM。

UPDATE

允许使用UPDATE更新指定表的任何列，或列出的特定列。（实际上，任何非简单的UPDATE命令还需要SELECT权限，因为它必须引用表列，以确定要更新的行和/或计算列的新值。）SELECT ... FOR UPDATE和SELECT ... FOR SHARE除了需要SELECT权限外，也要求至少在一列上具有此权限。对于序列，此权限允许使用nextval和setval函数。对于大对象，此权限允许写入或截断该对象。

DELETE

允许使用DELETE从指定表删除一行。（实际上，任何非简单的DELETE命令还需要SELECT权限，因为它必须引用表列，以确定要删除的行。）

TRUNCATE

允许在指定表上使用TRUNCATE。

REFERENCES

要创建外键约束，必须在引用列和被引用列上都拥有此权限。此权限可以授予表的所有列，也可以只授予特定列。

TRIGGER

允许在指定表上创建触发器。（参见CREATE TRIGGER语句。）

CREATE

对于数据库，允许在数据库中创建新的模式。

对于模式，允许在模式中创建新对象。要重命名现有对象，必须拥有该对象，并且在包含它的模式上具有此权限。

对于表空间，允许在其中创建表、索引和临时文件，也允许创建以该表空间为默认表空间的数据库。（注意，撤销此权限不会改变现有对象的存放位置。）

CONNECT

允许用户连接到指定数据库。此权限在连接启动时检查（此外还会检查pg_hba.conf施加的任何限制）。

TEMPORARY
TEMP

允许在使用指定数据库时创建临时表。

EXECUTE

允许使用指定的函数，以及基于该函数实现的任何操作符。
这是唯一适用于函数的权限类型。（此语法也适用于聚合函数。）

USAGE

对于过程语言，允许使用指定语言创建以该语言编写的函数。
这是唯一适用于过程语言的权限类型。

对于模式，允许访问指定模式中包含的对象（假定也满足对象自身的权限要求）。
本质上，这允许被授权者“查找”模式内的对象。没有此权限，仍然可能看到对象名称，
例如通过查询系统表。此外，撤销此权限后，
现有后端中可能仍有先前已执行过这种查找的语句，
因此这并不是阻止对象访问的完全安全的方法。

对于序列，此权限允许使用`currval`和`nextval`函数。

对于类型和域，此权限允许在创建表、函数和其他模式对象时使用该类型或域。（注意，
它不控制该类型的一般“使用”，例如在查询中出现该类型的值。
它仅阻止创建依赖该类型的对象。
此权限的主要目的是控制哪些用户可以创建对某个类型的依赖，
因为这些依赖可能使所有者以后无法更改该类型。）

对于外部数据包装器，此权限使被授权者能够使用该外部数据包装器创建新的服务器。

对于服务器，此权限使被授权者能够创建、修改和删除该服务器关联的自己的用户映射。
此外，它还使被授权者能够查询该服务器及相关用户映射的选项。

ALL PRIVILEGES

一次授予所有可用权限。`PRIVILEGES`关键字在PostgreSQL中是可选的，但在严格 SQL 中是必需的。

其他命令所需的权限列在各自命令的参考页面上。

角色上的 GRANT

这种`GRANT`命令变体把一个角色的成员资格授予一个或多个其他角色。
角色成员资格之所以重要，是因为它会把授予该角色的权限传递给其每个成员。

如果指定了`WITH ADMIN OPTION`，成员就可以将该角色的成员资格继续授予其他人，也可以撤销该角色的成员资格。没有管理选项时，普通用户不能这样做。
一个角色不被认为在其自身上持有`WITH ADMIN OPTION`，但是在会话用户与该角色相符的数据库会话中，它可以把自身角色的成员资格授予其他角色，或撤销这种资格。
数据库超级用户可以向任何人授予或撤销任何角色的成员资格。具有`CREATEROLE`
权限的角色可以授予或撤销任何非超级用户角色的成员资格。

与权限不同，角色成员资格不能授予`PUBLIC`。还要注意，
这种形式的命令不允许使用无实际作用的`GROUP`一词。

注解

REVOKE命令用于撤销访问权限。

从 PostgreSQL 8.1 起，用户和组的概念已统一为一种称为角色的单一实体。因此，不再需要使用关键字 GROUP 来标识被授权者是用户还是组。GROUP 仍可出现在命令中，但它只是一个噪声词。

如果用户对某一列本身，或者对其所在整张表拥有该权限，就可以在该列上执行 SELECT、INSERT 等操作。在表级授予某项权限后，再在单列上撤销该权限，并不会产生你可能期望的效果：表级授权不会受到列级操作的影响。

当对象的非拥有者试图在该对象上执行 GRANT 时，如果该用户在该对象上完全没有任何权限，命令会立即失败。只要有某项权限可用，命令就会继续执行，
但只会授予那些该用户持有授予选项的权限。如果未持有任何授予选项，GRANT ALL PRIVILEGES 形式会发出警告；而其他形式如果命令中特别列出的任一权限未持有其授予选项，也会发出警告。（原则上，这些说明也适用于对象拥有者；
但由于拥有者总是被视为持有全部授予选项，这种情况实际上不会发生。）

需要注意，数据库超级用户可以访问所有对象，而不受对象权限设置的影响。这可类比于 Unix 系统中的 root 权限。和 root 一样，除非绝对必要，否则不宜以超级用户身份操作。

如果超级用户选择执行 GRANT 或 REVOKE 命令，该命令会像由受影响对象的拥有者发出一样执行。特别是，通过这种命令授予的权限看起来会像是由对象拥有者授予的。
（对于角色成员资格，则看起来像是由被授予成员资格的角色本身授予的。）

GRANT 和 REVOKE 也可以由并非受影响对象拥有者的角色执行，只要该角色是拥有该对象之角色的成员，或者是持有该对象上 WITH GRANT OPTION 权限之角色的成员。在这种情况下，权限会记录为由实际拥有该对象的角色，或者由持有 WITH GRANT OPTION 权限的角色授予。例如，如果表 t1 由角色 g1 拥有，而角色 u1 是它的成员，那么 u1 可以把 t1 上的权限授予给 u2，但这些权限看起来会像是直接由 g1 授予的。角色 g1 的任何其他成员之后都可以撤销这些权限。

如果执行 GRANT 的角色通过多条角色成员资格路径间接持有所需权限，则系统不会指明会被记录为执行该授权的是哪一个上层角色。在这种情况下，最佳做法是使用 SET ROLE 切换成你希望作为其身份执行 GRANT 的那个具体角色。

在表上授予权限，并不会自动把权限扩展到该表使用的任何序列，包括绑定到 SERIAL 列的序列。序列上的权限必须单独设置。

使用psql的 \dp 命令可以获取表和列的现有权限信息。例如：

```
=> \dp mytable
                            Access privileges
 Schema |  Name   | Type  |   Access privileges   | Column access
 privileges
--------+---------+-------+-----------------------+----------------
----------
 public | mytable | table | miriam=arwdDxt/miriam | col1:
                          : =r/miriam             :   miriam_rw=rw/
miriam
                          : admin=arw/miriam
(1 row)
```

由 \dp 显示的条目解释如下：

```
rolename=xxxx -- privileges granted to a role
        =xxxx -- privileges granted to PUBLIC

            r -- SELECT ("read")
            w -- UPDATE ("write")
            a -- INSERT ("append")
            d -- DELETE
```

```
            D -- TRUNCATE
            x -- REFERENCES
            t -- TRIGGER
            X -- EXECUTE
            U -- USAGE
            C -- CREATE
            c -- CONNECT
            T -- TEMPORARY
      arwdDxt -- ALL PRIVILEGES (for tables, varies for other
 objects)
            * -- grant option for preceding privilege

        /yyyy -- role that granted this privilege
```

上例中的显示结果会出现在用户 `miriam` 创建表 `mytable` 并执行以下命令之后：

```
GRANT SELECT ON mytable TO PUBLIC;
GRANT SELECT, UPDATE, INSERT ON mytable TO admin;
GRANT SELECT (col1), UPDATE (col1) ON mytable TO miriam_rw;
```

对于非表对象，还有其他`\d`命令可以显示其权限。

如果某个对象的“Access privileges”列为空，表示该对象具有默认权限（即其权限列为null）。默认权限始终包括所有者的全部权限，并且可能根据对象类型包含授予`PUBLIC`的某些权限，如上所述。在对象上首次执行`GRANT`或`REVOKE`时，会先实例化默认权限（例如生成`{miriam=arwdDxt/miriam}`），然后根据指定请求修改它们。同样，“Column access privileges”中也只会显示具有非默认权限的列的条目。（注意：此处的“默认权限”始终指该对象类型的内置默认权限。权限受`ALTER DEFAULT PRIVILEGES`命令影响的对象，始终会显示显式权限条目，其中包含`ALTER`的效果。）

注意，访问权限显示中不会标记所有者隐含的授权选项。只有在显式向某人授予授权选项时，才会出现`*`。

示例

将表 `films` 上的插入权限授予所有用户：

```
GRANT INSERT ON films TO PUBLIC;
```

将视图 `kinds` 上的所有可用权限授予用户 `manuel`：

```
GRANT ALL PRIVILEGES ON kinds TO manuel;
```

请注意，如果上述命令由超级用户或 `kinds` 的拥有者执行，确实会授予所有权限；但如果由其他人执行，则只会授予该执行者持有授予选项的那些权限。

将角色 `admins` 的成员资格授予用户 `joe`：

```
GRANT admins TO joe;
```

兼容性

根据 SQL 标准，`ALL PRIVILEGES` 中的 `PRIVILEGES` 关键字是必需的。SQL 标准也不支持每条命令对多个对象设置权限。

PostgreSQL 允许对象拥有者撤销自己的普通权限：例如，表拥有者可以通过撤销自己的 `INSERT`、`UPDATE`、`DELETE` 和 `TRUNCATE` 权限，使该表对自己变成只读。这在 SQL 标准中是不可能的。原因是 PostgreSQL 把拥有者的权限视为拥有者授予给自己的；因此他们也可以

撤销这些权限。在 SQL 标准中，拥有者的权限由一个假定实体“_SYSTEM”授予。由于拥有者并不是“_SYSTEM”，因此不能撤销这些权利。

根据 SQL 标准，授予选项可以授予给 `PUBLIC`；PostgreSQL 只支持将授予选项授予给角色。

SQL 标准还为其他种类的对象提供 `USAGE` 权限：字符集、排序规则、翻译。

在 SQL 标准中，序列只有 `USAGE` 这一项权限，它控制 `NEXT VALUE FOR` 表达式的使用；该表达式等价于 PostgreSQL 中的 `nextval` 函数。序列上的 `SELECT` 和 `UPDATE` 权限都是 PostgreSQL 扩展。把序列的 `USAGE` 权限应用到 `currval` 函数上也是 PostgreSQL 扩展（该函数本身也是扩展）。

数据库、表空间、模式和语言上的权限都是 PostgreSQL 扩展。

参见

REVOKE, ALTER DEFAULT PRIVILEGES

INSERT

INSERT — 在表中插入新行

大纲

```
[ WITH [ RECURSIVE ] with_query [, ...] ]
INSERT INTO table_name [ ( column_name [, ...] ) ]
    { DEFAULT VALUES | VALUES ( { expression | DEFAULT } [, ...] )
 [, ...] | query }
    [ RETURNING * | output_expression [ [ AS ] output_name ] [, ...
] ]
```

描述

INSERT将新行插入表中。可以插入由值表达式指定的一行或多行，
也可以插入由查询产生的零行或多行。

目标列名可以按任意顺序列出。如果根本未给出列名列表，
则默认使用按声明顺序排列的全部列；或者如果VALUES子句或 *query*只提供了*N*列，则默认使用按声明顺序排列的前*N*个列名。VALUES子句或*query* 提供的值，会按从左到右的顺序与显式或隐式列列表对应起来。

任何未出现在显式或隐式列列表中的列都会填入默认值；如果没有声明默认值，
则填入空值。

如果任何列的表达式数据类型不正确，将尝试自动进行类型转换。

可选的RETURNING子句使INSERT 基于每个实际插入的行计算并返回一个或多个值。这主要用于获取由默认值提供的值，例如 serial 序列号。不过，也允许使用任何引用该表列的表达式。RETURNING列表的语法与 SELECT的输出列表相同。

要向表中插入行，必须具有该表上的INSERT权限。如果指定了列列表，
则只需要对所列的列具有INSERT权限。使用RETURNING子句要求具有SELECT权限，涵盖RETURNING中提到的所有列。如果使用*query*子句从查询中插入行，
当然需要在查询使用的任何表或列上具有SELECT权限。

参数

with_query

WITH子句允许指定一个或多个子查询，这些子查询可以在INSERT查询中按名称引用。详见第 7.8 节和SELECT。

query（SELECT语句）本身也可以包含 WITH子句。在这种情况下，*query*中可以引用两组 *with_query*，但由于第二组嵌套得更近，它具有更高的优先级。

table_name

现有表的名称（可选地使用模式限定）。

column_name

名为*table_name*的表中某一列的名称。如有需要，列名可以附带子字段名或数组下标。（只向组合列的部分字段插入值时，其余字段将为空值。）

`DEFAULT VALUES`

所有列都会填入各自的默认值。

expression

赋给相应列的表达式或值。

`DEFAULT`

相应列将填入其默认值。

query

提供要插入行的查询（`SELECT`语句）。其语法说明请参见SELECT。

output_expression

`INSERT`命令在插入每行后要计算并返回的表达式。表达式可以使用*table_name*所指定表中的任何列名。写`*`可返回所插入行的所有列。

output_name

用于返回列的名称。

输出

成功完成时，`INSERT` 命令返回以下形式的命令标签：

`INSERT` *oid count*

其中，*count* 是插入的行数。如果 *count* 恰好为一，且目标表有 OID，则 *oid* 表示分配给插入行的 OID。该单行必须是插入的，而不是更新的。否则 *oid* 为零。

如果`INSERT`命令包含`RETURNING` 子句，则结果将类似于一个`SELECT`语句，其中包含`RETURNING`列表定义的列和值，它们是基于该命令插入或更新的行计算出来的。

示例

向表`films`插入一行：

```
INSERT INTO films VALUES
    ('UA502', 'Bananas', 105, '1971-07-13', 'Comedy', '82 minutes')
;
```

在这个示例中，省略了`len`列，因此它将取默认值：

```
INSERT INTO films (code, title, did, date_prod, kind)
    VALUES ('T_601', 'Yojimbo', 106, '1961-06-16', 'Drama');
```

这个示例对日期列使用`DEFAULT`子句，而不是显式指定值：

```
INSERT INTO films VALUES
    ('UA502', 'Bananas', 105, DEFAULT, 'Comedy', '82 minutes');
INSERT INTO films (code, title, did, date_prod, kind)
    VALUES ('T_601', 'Yojimbo', 106, DEFAULT, 'Drama');
```

要插入一行，其所有列都由默认值构成：

```
INSERT INTO films DEFAULT VALUES;
```

使用多行VALUES语法插入多行：

```
INSERT INTO films (code, title, did, date_prod, kind) VALUES
    ('B6717', 'Tampopo', 110, '1985-02-10', 'Comedy'),
    ('HG120', 'The Dinner Game', 140, DEFAULT, 'Comedy');
```

这个示例从与films列布局相同的 tmp_films表中选出一些行并插入到 films表：

```
INSERT INTO films SELECT * FROM tmp_films WHERE date_prod < '2004
-05-07';
```

这个示例向数组列插入值：

```
-- Create an empty 3x3 gameboard for noughts-and-crosses
INSERT INTO tictactoe (game, board[1:3][1:3])
    VALUES (1, '{{" "," "," "},{" "," "," "},{" "," "," "}}');
-- The subscripts in the above example aren't really needed
INSERT INTO tictactoe (game, board)
    VALUES (2, '{{X," "," "},{" ",O," "},{" ",X," "}}');
```

向表distributors插入一行，并返回由 DEFAULT子句生成的序号：

```
INSERT INTO distributors (did, dname) VALUES (DEFAULT, 'XYZ
 Widgets')
   RETURNING did;
```

为负责 Acme Corporation 账户的销售人员增加销量计数，并把整个更新后的行连同当前时间记录到日志表中：

```
WITH upd AS (
  UPDATE employees SET sales_count = sales_count + 1 WHERE id =
    (SELECT sales_person FROM accounts WHERE name = 'Acme
 Corporation')
    RETURNING *
)
INSERT INTO employees_log SELECT *, current_timestamp FROM upd;
```

兼容性

INSERT符合 SQL 标准，但 RETURNING子句是 PostgreSQL扩展，在 INSERT中使用WITH的能力也是扩展。此外，标准不允许省略列名列表却又不是所有列都由VALUES 子句或*query*填充的情况。

*query*子句可能存在的限制见SELECT。

LISTEN

LISTEN — 监听通知

大纲

```
LISTEN channel
```

描述

LISTEN将当前会话注册为名为*channel*的通知通道的监听者。
如果当前会话已经在该通知通道上注册为监听者，则不会执行任何操作。

每当命令NOTIFY *channel*被执行时，无论是由当前会话还是由另一个连接到同一数据库的会话执行，所有当前正在监听该通知通道的会话都会收到通知，
而每个会话随后都会通知其所连接的客户端应用。

会话可以使用UNLISTEN命令取消在给定通知通道上的监听注册。会话结束时，
其监听注册会被自动清除。

客户端应用必须采用何种方法来检测通知事件，取决于它所使用的 PostgreSQL应用程序编程接口。使用 libpq库时，应用程序将LISTEN 作为普通 SQL 命令发出，然后必须定期调用 PQnotifies函数，以判断是否已经收到任何通知事件。其他接口，例如libpgtcl，则提供了处理通知事件的更高层方法；事实上，在libpgtcl中，应用程序员甚至不应直接发出LISTEN或UNLISTEN。详见所使用接口的相关文档。

NOTIFY中对LISTEN和NOTIFY的使用有更详细的讨论。

参数

channel

通知通道的名称（任意标识符）。

注解

LISTEN会在事务提交时生效。如果在随后被回滚的事务中执行了LISTEN或 UNLISTEN，则被监听的通知通道集合不会发生变化。

执行过LISTEN的事务不能为两阶段提交做准备。

示例

在psql中配置并执行一组 listen/notify 操作：

```
LISTEN virtual;
NOTIFY virtual;
Asynchronous notification "virtual" received from server process
 with PID 8448.
```

兼容性

SQL 标准中没有LISTEN语句。

另见

NOTIFY, UNLISTEN

LOAD

LOAD — 载入共享库文件

大纲

```
LOAD 'filename'
```

描述

该命令将一个共享库文件载入PostgreSQL服务器的地址空间。如果该文件已经载入过，则此命令不执行任何操作。包含 C 函数的共享库文件会在调用其中某个函数时自动载入。因此，显式执行`LOAD`通常只在要载入的是通过“钩子”修改服务器行为、而不是提供一组函数的库时才有需要。

文件名的指定方式与CREATE FUNCTION中共享库名的指定方式相同；特别地，可以依赖搜索路径以及系统标准共享库文件名扩展名的自动添加。关于此主题的更多信息，见第 35.9 节。

非超级用户只能将`LOAD`用于位于`$libdir/plugins/`的库文件 – 指定的*filename*必须以该字符串原样开头。（数据库管理员有责任确保只在该目录中安装“安全的”库。）

兼容性

`LOAD`是PostgreSQL的一种扩展。

另见

CREATE FUNCTION

LOCK

LOCK — 锁定表

大纲

```
LOCK [ TABLE ] [ ONLY ] name [ * ] [, ...] [ IN lockmode MODE ] [
 NOWAIT ]
```

其中*lockmode*为以下之一：

```
    ACCESS SHARE | ROW SHARE | ROW EXCLUSIVE | SHARE UPDATE
 EXCLUSIVE
    | SHARE | SHARE ROW EXCLUSIVE | EXCLUSIVE | ACCESS EXCLUSIVE
```

描述

LOCK TABLE获取一个表级锁；如有必要，会等待任何冲突锁被释放。如果指定了NOWAIT，LOCK TABLE就不会等待获取所需的锁：如果无法立即获得，命令将被中止并报错。锁一旦获得，就会一直持有到当前事务结束。（没有UNLOCK TABLE命令；锁总是在事务结束时释放。）

为引用表的命令自动获取锁时，PostgreSQL始终使用限制最少的可用锁模式。LOCK TABLE用于可能需要更严格锁定的情况。例如，假设一个应用在READ COMMITTED隔离级别运行事务，并且需要确保表中的数据在事务期间保持稳定。为此，可以在查询之前对表获取SHARE锁。这会阻止并发数据更改，确保后续对表的读取能看到已提交数据的稳定视图，因为SHARE锁模式与写入者获取的ROW EXCLUSIVE锁冲突，而你的LOCK TABLE *name* IN SHARE MODE语句会等待，直到所有并发持有ROW EXCLUSIVE模式锁的事务提交或回滚。因此，一旦获得该锁，就不存在尚未提交的写入；而且在释放该锁之前，也不会开始任何写入。

若要在REPEATABLE READ或SERIALIZABLE隔离级别的事务中达到类似效果，你必须在执行任何SELECT或数据修改语句之前执行LOCK TABLE语句。REPEATABLE READ或SERIALIZABLE事务的数据视图，会在其第一条SELECT或数据修改语句开始时冻结。在事务稍后再执行LOCK TABLE仍然可以阻止并发写入 – 但它不能保证该事务读取到的是最新已提交的值。

如果这类事务还要修改表中的数据，那么它应使用SHARE ROW EXCLUSIVE锁模式，而不是SHARE模式。这样可以确保同一时间只有一个这类事务在运行。
否则就可能发生死锁：两个事务都可能先获得SHARE模式，
然后都无法再获得实际执行更新所需的ROW EXCLUSIVE模式。（注意，事务自己的锁永远不会互相冲突，因此事务在持有SHARE模式时仍可获得 ROW EXCLUSIVE模式，但前提是没有其他人持有SHARE模式。）为避免死锁，要确保所有事务都按相同顺序对相同对象获取锁；
如果同一对象需要多种锁模式，则事务应始终先获取限制最严格的模式。

关于锁模式和锁策略的更多信息，请参见第 13.3 节。

参数

name

要锁定的现有表的名称（可选模式限定）。如果在表名前指定了 ONLY，则只有该表会被锁定。如果未指定ONLY，则该表及其所有后代表（如果有）都会被锁定。
也可以在表名后指定*，以显式表明包含后代表。

命令LOCK TABLE a, b;等效于 LOCK TABLE a; LOCK TABLE b;。这些表会按 LOCK TABLE命令中指定的顺序逐个锁定。

lockmode

锁模式指定该锁会与哪些锁冲突。锁模式见第 13.3 节。

如果未指定锁模式，则使用限制最严格的ACCESS EXCLUSIVE模式。

NOWAIT

指定LOCK TABLE不等待任何冲突锁被释放：如果指定的锁无法在不等待的情况下立即获得，事务就会中止。

注解

LOCK TABLE ... IN ACCESS SHARE MODE要求对目标表具有SELECT权限。所有其他形式的LOCK 都要求具有表级UPDATE、DELETE或TRUNCATE权限。

LOCK TABLE在事务块外毫无用处：锁只会一直持有到该语句结束。因此，如果在事务块外使用LOCK，PostgreSQL会报告错误。请使用BEGIN和COMMIT（或ROLLBACK）来定义事务块。

LOCK TABLE只处理表级锁，因此名称中带有ROW的模式其实都不准确。这些模式名称通常应理解为：用户打算在被锁定的表中获取行级锁。此外，ROW EXCLUSIVE模式本身也是一种可共享的表锁。请记住，就LOCK TABLE而言，所有锁模式的语义完全相同，差别只在于哪些模式彼此冲突。关于如何获取真正的行级锁，请参阅第 13.3.2 节以及FOR UPDATE/FOR SHARE Clause一节（位于SELECT参考文档中）。

示例

在准备向外键表执行插入时，在主键表上获取一个SHARE锁：

```
BEGIN WORK;
LOCK TABLE films IN SHARE MODE;
SELECT id FROM films
    WHERE name = 'Star Wars: Episode I - The Phantom Menace';
-- 如果未返回记录则执行 ROLLBACK
INSERT INTO films_user_comments VALUES
    (_id_, 'GREAT! I was waiting for it for so long!');
COMMIT WORK;
```

在准备执行删除操作时，在主键表上获取一个SHARE ROW EXCLUSIVE锁：

```
BEGIN WORK;
LOCK TABLE films IN SHARE ROW EXCLUSIVE MODE;
DELETE FROM films_user_comments WHERE id IN
    (SELECT id FROM films WHERE rating < 5);
DELETE FROM films WHERE rating < 5;
COMMIT WORK;
```

兼容性

SQL 标准中没有LOCK TABLE，而是使用SET TRANSACTION 来指定事务的并发级别。PostgreSQL也支持这一点；详见SET TRANSACTION。

除ACCESS SHARE、ACCESS EXCLUSIVE和 SHARE UPDATE EXCLUSIVE锁模式外，PostgreSQL的锁模式和LOCK TABLE语法与Oracle中的对应语法兼容。

MOVE

MOVE — 定位游标

大纲

```
MOVE [ direction [ FROM | IN ] ] cursor_name
```

其中*direction*可以为空，或为以下之一：

```
    NEXT
    PRIOR
    FIRST
    LAST
    ABSOLUTE count
    RELATIVE count
    count
    ALL
    FORWARD
    FORWARD count
    FORWARD ALL
    BACKWARD
    BACKWARD count
    BACKWARD ALL
```

描述

MOVE在不检索任何数据的情况下重新定位游标。MOVE与FETCH命令的工作方式完全相同，只是它只定位游标而不返回行。

MOVE命令的参数与FETCH命令完全相同；有关语法和用法的细节，请参阅FETCH。

输出

成功完成时，MOVE命令会返回如下形式的命令标签：

```
MOVE count
```

其中*count*是用相同参数执行 FETCH命令时本应返回的行数（可能为零）。

示例

```
BEGIN WORK;
DECLARE liahona CURSOR FOR SELECT * FROM films;

-- 跳过前 5 行:
MOVE FORWARD 5 IN liahona;
MOVE 5

-- 从游标 liahona 中提取第 6 行:
FETCH 1 FROM liahona;
 code  | title  | did | date_prod  |  kind  |  len
-------+--------+-----+------------+--------+-------
 P_303 | 48 Hrs | 103 | 1982-10-22 | Action | 01:37
(1 row)
```

```
-- 关闭游标 liahona 并结束事务:
CLOSE liahona;
COMMIT WORK;
```

兼容性

SQL 标准中没有MOVE语句。

另见

CLOSE, DECLARE, FETCH

NOTIFY

NOTIFY — 发出一个通知

大纲

```
NOTIFY channel [ , payload ]
```

描述

NOTIFY命令向每个客户端应用发送一个通知事件，并可附带一个可选的“载荷”字符串；这些客户端应用此前都已在当前数据库中针对指定的通道名执行过LISTEN *channel*。

NOTIFY为访问同一个PostgreSQL数据库的一组进程提供了一种简单的进程间通信机制。通知中可以附带载荷字符串；如果需要传递结构化数据，也可以借助数据库中的表，将附加数据从通知发送者传递给一个或多个监听者，从而构建用于传递结构化数据的更高层机制。

传递给客户端的通知事件信息包括：通知通道名、发出通知的会话对应的服务器进程PID，以及载荷字符串。如果未指定载荷字符串，则该字符串为空串。

在某个数据库中使用哪些通道名以及各自含义，由数据库设计者自行决定。常见的做法是让通道名与数据库中的某个表同名，而通知事件基本上意味着：“我改动了这张表，去看看有什么新变化”。不过，NOTIFY和LISTEN命令并不会强制这种关联。例如，数据库设计者可以使用多个不同的通道名，来标识同一张表上的不同类型变更；或者也可以利用载荷字符串区分不同情况。

当NOTIFY用于表明某张特定表发生变化时，一种很有用的编程技巧是把NOTIFY放进由表更新触发的规则中。这样一来，每当表发生变化时就会自动发出通知，应用程序员也不容易忘记这么做。

NOTIFY会以几种重要方式与 SQL 事务交互。首先，如果在事务内部执行NOTIFY，那么只有在事务提交之后，通知事件才会被递送；如果事务被中止，则其中所有命令都不会生效，NOTIFY也不例外。这种行为是合理的，但如果你期望通知立即送达，可能会感到意外。其次，如果某个正在监听的会话在事务内部收到了通知信号，那么在该事务结束（提交或中止）之前，通知事件都不会递送给它所连接的客户端。原因同样在于：如果通知在事务内部就已递送，而该事务后来又被中止，我们会希望通知也能被撤销，但服务器一旦把通知发送给客户端，就无法再把它“收回”。因此，通知事件只会在事务之间递送。由此得出的结论是，使用NOTIFY做实时信号的应用，应尽量让事务保持短小。

如果在同一个事务中，针对同一通道名多次发送完全相同载荷字符串的通知，那么数据库服务器可以决定只递送一个通知。另一方面，载荷字符串不同的通知始终会作为不同通知递送。类似地，来自不同事务的通知也绝不会被折叠为一个通知。除了会丢弃重复通知中较后的那些实例之外，NOTIFY还保证来自同一事务的通知会按发送顺序递送；来自不同事务的消息，则会按事务提交顺序递送。

执行NOTIFY的客户端自己同时也在监听同一通知通道，这是很常见的情形。在这种情况下，它会像其他监听会话一样收到一个通知事件。根据应用逻辑，这可能导致无用功，例如再次读取自己刚刚写入更新的数据库表。要避免这种额外工作，可以检查通知事件消息中提供的发出通知的服务器进程PID，是否与当前会话自身的PID（可通过libpq获得）相同。如果二者相同，就说明这是当前会话自己发出的通知回送给自己，可以直接忽略。

参数

channel

要发信号的通知通道名称（任意标识符）。

payload

随通知一起传递的“载荷”字符串。它必须指定为简单的字符串字面值。在默认配置下，它必须短于 8000 字节。（如果需要传递二进制数据或大量信息，最好把它们放入数据库表中，并发送该记录的键。）

注解

系统中有一个队列，用来保存那些已经发送但尚未被所有监听会话处理的通知。
如果这个队列被塞满，那么调用NOTIFY的事务会在提交时失败。
该队列非常大（标准安装中为 8GB），几乎足以满足所有用例。不过，如果某个会话执行了LISTEN后又长时间停留在一个事务里，就无法进行清理。一旦队列占用达到一半，
你就会在日志文件中看到警告，指出究竟是哪个会话阻碍了清理。在这种情况下，
应确保该会话结束其当前事务，以便清理能够继续进行。

执行过NOTIFY的事务不能为两阶段提交做准备。

pg_notify

要发送通知，也可以使用函数pg_notify(text, text)。该函数的第一个参数是通道名，第二个参数是载荷。如果需要处理非常量的通道名和载荷，它会比NOTIFY命令更容易使用。

示例

在psql中配置并执行一组 listen/notify 操作：

```
LISTEN virtual;
NOTIFY virtual;
Asynchronous notification "virtual" received from server process
 with PID 8448.
NOTIFY virtual, 'This is the payload';
Asynchronous notification "virtual" with payload "This is the
 payload" received from server process with PID 8448.

LISTEN foo;
SELECT pg_notify('fo' || 'o', 'pay' || 'load');
Asynchronous notification "foo" with payload "payload" received
 from server process with PID 14728.
```

兼容性

SQL 标准中没有NOTIFY语句。

另见

LISTEN, UNLISTEN

PREPARE

PREPARE — 为执行准备一个语句

大纲

```
PREPARE name [ ( data_type [, ...] ) ] AS statement
```

描述

PREPARE 创建一个预备语句。预备语句是一种服务器端对象，可用于优化性能。执行 PREPARE 语句时，指定的语句会被解析、分析并重写。随后发出 EXECUTE 命令时，该预备语句会被规划并执行。这种分工避免了重复的解析分析工作，
同时又允许执行计划依赖于所提供的特定参数值。

预备语句可以带参数，也就是在执行时会代入语句中的值。创建预备语句时，
可以按位置引用参数，例如 $1、$2 等。也可以选择指定相应的参数数据类型列表。如果某个参数的数据类型未指定，或被声明为 unknown，则其类型会从该参数第一次被引用时所在的上下文中推断出来（如果可能）。执行该语句时，需要在 EXECUTE 语句中为这些参数提供实际值。有关详情，参见EXECUTE。

预备语句只在当前数据库会话期间存在。会话结束时，预备语句就会被遗忘，
因此再次使用前必须重新创建。
这也意味着单个预备语句不能由多个并发的数据库客户端共用；不过，
每个客户端都可以创建自己要使用的预备语句。预备语句也可以用
DEALLOCATE命令手工释放。

当单个会话要执行大量相似语句时，预备语句能带来最大的性能优势。
如果语句在规划或重写时比较复杂，例如查询涉及许多表的连接，或者需要应用多条规则，
那么这种性能差异会特别明显。如果语句相对容易规划和重写，但执行本身代价较高，
则预备语句带来的性能优势就不那么明显。

参数

name

为这个特定预备语句指定的任意名称。它在单个会话内必须唯一，
随后用于执行或释放先前已准备的语句。

data_type

预备语句中某个参数的数据类型。如果某个参数的数据类型未指定，或被指定为
unknown，则其类型会从该参数第一次被引用时所在的上下文中推断出来。
要在预备语句本身中引用参数，可使用 $1、$2 等。

statement

任何SELECT、INSERT、UPDATE、DELETE或VALUES语句。

注解

如果一条预备语句被执行的次数足够多，服务器最终可能会决定保存并复用一个通用计划，
而不是每次都重新规划。如果预备语句没有参数，这将立即发生；否则，
只有当通用计划看起来不比依赖具体参数值的计划昂贵多少时，才会使用通用计划。通常，
只有当查询的性能估计对所提供的具体参数值相当不敏感时，才会选择通用计划。

要检查PostgreSQL为预备语句使用的查询计划，请使用EXPLAIN。如果使用的是通用计划，其中会包含参数符号`$n`，而自定义计划中会代入当前的实际参数值。

有关查询规划以及 PostgreSQL 为此收集的统计信息的更多内容，请参见ANALYZE文档。

可以通过查询`pg_prepared_statements` 系统视图，查看当前会话中所有可用的预备语句。

示例

为一个`INSERT` 语句创建预备语句，然后执行它：

```
PREPARE fooplan (int, text, bool, numeric) AS
    INSERT INTO foo VALUES($1, $2, $3, $4);
EXECUTE fooplan(1, 'Hunter Valley', 't', 200.00);
```

为一个 `SELECT` 语句创建预备语句，然后执行它：

```
PREPARE usrrptplan (int) AS
    SELECT * FROM users u, logs l WHERE u.usrid=$1 AND u.usrid=l.
usrid
    AND l.date = $2;
EXECUTE usrrptplan(1, current_date);
```

注意，第二个参数的数据类型未指定，因此会从 `$2` 所在的使用上下文中推断出来。

兼容性

SQL 标准包含`PREPARE`语句，但它只用于嵌入式 SQL。这里的`PREPARE`语句在语法上也略有不同。

另见

DEALLOCATE, EXECUTE

PREPARE TRANSACTION

PREPARE TRANSACTION — 为两阶段提交准备当前事务

大纲

PREPARE TRANSACTION *transaction_id*

描述

PREPARE TRANSACTION为两阶段提交准备当前事务。执行此命令后，该事务将不再与当前会话相关联；相反，它的状态会被完整地存储到磁盘上，
因此即使在请求提交之前数据库发生崩溃，它也极有可能成功提交。

事务进入预备状态后，稍后可以分别使用COMMIT PREPARED或ROLLBACK PREPARED来提交或回滚。发出这些命令的不必是执行原始事务的那个会话，任何会话都可以这样做。

从发出该命令的会话来看，PREPARE TRANSACTION颇似ROLLBACK：执行之后，将不再有活动的当前事务，并且该预备事务的效果也不再可见。（如果该事务随后被提交，
这些效果会再次可见。）

如果PREPARE TRANSACTION因任何原因失败，它就等同于执行了一次ROLLBACK：当前事务会被取消。

参数

transaction_id

一个任意标识符，后续可通过它在COMMIT PREPARED或ROLLBACK PREPARED中标识该事务。该标识符必须写成字符串字面值，长度必须小于 200 字节，并且不能与当前任何已处于预备状态的事务标识符相同。

注解

PREPARE TRANSACTION并非供应用程序或交互式会话使用。它的目的是让外部事务管理器能够跨多个数据库或其他事务性资源执行原子性的全局事务。除非是在编写事务管理器，
否则通常不应使用PREPARE TRANSACTION。

该命令必须在事务块内部使用。请用BEGIN启动事务块。

当前不允许对执行过以下任一操作的事务执行PREPARE：涉及临时表的操作、创建任何WITH HOLD游标，或执行过LISTEN、UNLISTEN或NOTIFY。
这些特性与当前会话绑定得过于紧密，因此在要进入预备状态的事务中没有意义。

如果该事务曾用SET修改过任何运行时参数（且未使用LOCAL选项），那么这些效果在执行PREPARE TRANSACTION之后仍会保留，并且不会受到后续任何COMMIT PREPARED或ROLLBACK PREPARED的影响。因此，仅就这一点而言，PREPARE TRANSACTION的行为更像COMMIT而不是ROLLBACK。

当前所有处于预备状态的事务都列在pg_prepared_xacts系统视图中。

小心

让事务长时间停留在预备状态并不明智。这会妨碍VACUUM回收存储空间，
在极端情况下甚至可能导致数据库为防止事务 ID 回卷而关闭（见第 23.1.

5 节）。还要记住，该事务会继续持有它原本持有的所有锁。此功能的预期用法是：一旦外部事务管理器确认其他数据库也已准备好提交，就尽快提交或回滚该预备事务。

如果没有设置外部事务管理器来跟踪预备事务并确保它们能被及时结束，最好将max_prepared_transactions设为零，从而禁用预备事务功能。
这样可以防止意外创建预备事务，而这些事务随后可能被遗忘并最终引发问题。

示例

为两阶段提交准备当前事务，并使用`foobar`作为事务标识符：

```
PREPARE TRANSACTION 'foobar';
```

兼容性

`PREPARE TRANSACTION`是PostgreSQL扩展。它供外部事务管理系统使用，其中有些系统受标准约束（例如 X/Open XA），但这些系统的 SQL 侧并未标准化。

另见

COMMIT PREPARED, ROLLBACK PREPARED

REASSIGN OWNED

REASSIGN OWNED — 更改一个数据库角色所拥有的数据库对象的所有权

大纲

`REASSIGN OWNED BY old_role [, ...] TO new_role`

描述

`REASSIGN OWNED`指示系统将任何 *old_roles*所拥有的数据库对象的所有权更改为 *new_role*。

参数

old_role

一个角色的名称。该角色在当前数据库中拥有的所有对象的所有权，都将被重新分配给 *new_role*。

new_role

将成为受影响对象新拥有者的角色名称。

注解

`REASSIGN OWNED`常用于为移除一个或多个角色做准备。由于`REASSIGN OWNED`只影响当前数据库中的对象，因此通常需要在每个包含待移除角色所拥有对象的数据库中执行此命令。

`REASSIGN OWNED` 要求对源角色和目标角色都具有权限。

DROP OWNED命令提供了另一种选择，它会直接删除一个或多个角色所拥有的全部数据库对象。

`REASSIGN OWNED`命令不会影响授予 *old_roles*的、在非其所拥有的对象上的任何权限。如需撤销这类权限，请使用`DROP OWNED`。

`REASSIGN OWNED`命令不会影响该角色所拥有的任何数据库的所有权。要重新分配这类所有权，请使用ALTER DATABASE。

更多讨论请见第 20.4 节。

兼容性

`REASSIGN OWNED`命令是 PostgreSQL扩展。

另见

DROP OWNED, DROP ROLE, ALTER DATABASE

REINDEX

REINDEX — 重建索引

大纲

```
REINDEX { INDEX | TABLE | DATABASE | SYSTEM } name [ FORCE ]
```

描述

REINDEX使用索引所属表中存储的数据重建索引，并替换索引的旧副本。
以下几种场景适合使用REINDEX：

- 一个索引已经损坏，不再包含有效数据。尽管理论上这不该发生，但在实践中索引可能因软件缺陷或硬件故障而损坏。REINDEX提供了一种恢复方法。
- 一个索引已经“膨胀”，也就是说其中包含许多空页或几乎为空的页。在 PostgreSQL 中，B-树索引在某些不常见的访问模式下可能出现这种情况。REINDEX 可通过写入一个不含死页的新版本索引来减少索引的空间消耗。详见第 23.2 节。
- 你修改了某个索引的存储参数（例如 fillfactor），并希望确保该更改已经完全生效。
- 使用 CONCURRENTLY 选项构建索引失败后，会留下一个“无效”索引。这类索引没有用处，但用 REINDEX 重建它们可能很方便。注意，REINDEX 不会进行并发构建。要在不干扰生产运行的情况下构建索引，应删除该索引并重新执行 CREATE INDEX CONCURRENTLY 命令。

参数

INDEX

重新创建指定的索引。

TABLE

重新创建指定表的所有索引。如果该表有一个辅助“TOAST”表，也会对其重新索引。

DATABASE

重新创建当前数据库中的所有索引。共享系统目录上的索引也会被处理。这种形式的REINDEX不能在事务块内执行。

SYSTEM

重新创建当前数据库内系统目录上的所有索引。共享系统目录上的索引也包含在内。用户表上的索引不会被处理。这种形式的 REINDEX不能在事务块内执行。

name

要重新索引的特定索引、表或数据库的名称。索引名和表名可以带模式限定。目前，REINDEX DATABASE和 REINDEX SYSTEM只能对当前数据库重新索引，所以其参数必须与当前数据库名匹配。

VERBOSE

在每个索引被重建时打印进度报告。

注解

如果怀疑某个用户表上的索引已经损坏，可以使用 REINDEX INDEX或REINDEX TABLE 直接重建该索引，或者重建该表上的所有索引。

如果需要从系统表上的索引损坏中恢复，情况就更复杂了。在这种情况下，重要的是系统本身没有使用任何可疑索引。（事实上，在这种场景下，你可能会发现服务器进程在启动时立即崩溃，因为它依赖损坏的索引。）要安全恢复，必须用-P选项启动服务器，该选项会阻止服务器在查找系统目录时使用索引。

一种做法是关闭服务器，并在命令行中包含-P选项来启动单用户 PostgreSQL 服务器。然后可以根据希望重建的范围，执行REINDEX DATABASE、REINDEX SYSTEM、REINDEX TABLE 或REINDEX INDEX。如果拿不准，就使用REINDEX SYSTEM来重建该数据库中的所有系统索引。然后退出单用户服务器会话并重新启动常规服务器。关于如何与单用户服务器接口交互的更多信息，参见postgres参考页。

另一种方法是启动一个常规服务器会话，并在其命令行选项中包含 -P。具体做法因客户端而异，但对于所有基于 libpq的客户端，都可以在启动客户端之前将环境变量PGOPTIONS设置为-P。注意，尽管这种方法不需要阻止其他客户端，但在修复完成之前，阻止其他用户连接到受损数据库可能仍然更稳妥。

REINDEX类似于删除并重新创建索引，因为索引内容都是从头重建的。不过，两者在锁方面的考量相当不同。REINDEX会阻止该索引所属表上的写入，但不阻止读取。它还会对正在处理的特定索引获取独占锁，从而阻塞试图使用该索引的读取。相比之下，DROP INDEX会短暂地对父表获取独占锁，同时阻塞写入和读取。随后的CREATE INDEX会阻止写入但不阻止读取；由于索引不存在，读取不会尝试使用它，因此不会发生阻塞，但读取可能被迫使用代价高昂的顺序扫描。

对单个索引或表重新索引，要求用户是该索引或表的拥有者。对数据库重新索引，则要求用户是该数据库的拥有者（因此，拥有者可以重建其他用户所拥有表的索引）。当然，超级用户始终可以重新索引任何对象。

在 PostgreSQL 8.1 之前，REINDEX DATABASE只处理系统索引，而不是如其名称所暗示的处理所有索引。为减少意外，这一点已被更改。旧行为可通过REINDEX SYSTEM使用。

在 PostgreSQL 7.4 之前，REINDEX TABLE不会自动处理 TOAST 表，因此那些表必须用单独的命令重新索引。现在仍然可以这样做，但已是多余的。

示例

重建单个索引：

```
REINDEX INDEX my_index;
```

重建表my_table上的所有索引：

```
REINDEX TABLE my_table;
```

在不假定系统索引已经有效的情况下，重建某个数据库中的所有索引：

```
$ export PGOPTIONS="-P"
$ psql broken_db
...
broken_db=> REINDEX DATABASE broken_db;
broken_db=> \q
```

兼容性

在 SQL 标准中没有REINDEX命令。

RELEASE SAVEPOINT

RELEASE SAVEPOINT — 销毁一个先前定义的保存点

大纲

```
RELEASE [ SAVEPOINT ] savepoint_name
```

描述

`RELEASE SAVEPOINT` 销毁先前在当前事务中定义的保存点。

销毁保存点后，就不能再将它用作回滚点，但除此之外没有其他用户可见的行为。它不会撤销保存点建立之后执行的命令的效果。（要这样做，请参见ROLLBACK TO SAVEPOINT。）在不再需要保存点时将其销毁，可以让系统在事务结束之前回收一些资源。

`RELEASE SAVEPOINT` 还会销毁在指定保存点建立之后建立的所有保存点。

参数

savepoint_name

要销毁的保存点名称。

注解

指定一个此前未定义的保存点名称是一种错误。

当事务处于已中止状态时，不能释放保存点。

如果有多个保存点使用相同名称，则只会释放最近定义的那个。

示例

要建立保存点，并在稍后销毁它：

```
BEGIN;
    INSERT INTO table1 VALUES (3);
    SAVEPOINT my_savepoint;
    INSERT INTO table1 VALUES (4);
    RELEASE SAVEPOINT my_savepoint;
COMMIT;
```

上述事务将同时插入 3 和 4。

兼容性

该命令符合SQL标准。标准规定关键字`SAVEPOINT`是强制的，但PostgreSQL允许省略它。

另见

BEGIN, COMMIT, ROLLBACK, ROLLBACK TO SAVEPOINT, SAVEPOINT

RESET

RESET — 将一个运行时参数的值恢复为默认值

大纲

```
RESET configuration_parameter
RESET ALL
```

描述

`RESET`将运行时参数恢复为其默认值。`RESET`是

```
SET configuration_parameter TO DEFAULT
```

的另一种拼写。详见SET。

默认值指的是：在当前会话中，如果从未对该参数执行过`SET`，它本应具有的值。
该值的实际来源可能是编译时内置的默认值、配置文件、命令行选项，
或者针对特定数据库或特定用户的默认设置。这与将其定义为“该参数在会话开始时的值”略有不同，因为如果该值来自配置文件，那么它会被重置为配置文件当前指定的值。
详见第 18 章。

`RESET`的事务行为和`SET`相同：它的效果会在事务回滚时被撤销。

参数

configuration_parameter

可设置的运行时参数名称。可用参数见第 18 章以及SET参考页。

`ALL`

将所有可设置的运行时参数重置为默认值。

示例

将`timezone`配置参数设置为其默认值：

```
RESET timezone;
```

兼容性

`RESET`是一种 PostgreSQL扩展。

另见

SET, SHOW

REVOKE

REVOKE — 撤销访问权限

大纲

```
REVOKE [ GRANT OPTION FOR ]
    { { SELECT | INSERT | UPDATE | DELETE | TRUNCATE | REFERENCES |
 TRIGGER }
    [, ...] | ALL [ PRIVILEGES ] }
    ON { [ TABLE ] table_name [, ...]
         | ALL TABLES IN SCHEMA schema_name [, ...] }
    FROM { [ GROUP ] role_name | PUBLIC } [, ...]
    [ CASCADE | RESTRICT ]

REVOKE [ GRANT OPTION FOR ]
    { { SELECT | INSERT | UPDATE | REFERENCES } ( column_name [, ..
.] )
    [, ...] | ALL [ PRIVILEGES ] ( column_name [, ...] ) }
    ON [ TABLE ] table_name [, ...]
    FROM { [ GROUP ] role_name | PUBLIC } [, ...]
    [ CASCADE | RESTRICT ]

REVOKE [ GRANT OPTION FOR ]
    { { USAGE | SELECT | UPDATE }
    [, ...] | ALL [ PRIVILEGES ] }
    ON { SEQUENCE sequence_name [, ...]
         | ALL SEQUENCES IN SCHEMA schema_name [, ...] }
    FROM { [ GROUP ] role_name | PUBLIC } [, ...]
    [ CASCADE | RESTRICT ]

REVOKE [ GRANT OPTION FOR ]
    { { CREATE | CONNECT | TEMPORARY | TEMP } [, ...] | ALL [
 PRIVILEGES ] }
    ON DATABASE database_name [, ...]
    FROM { [ GROUP ] role_name | PUBLIC } [, ...]
    [ CASCADE | RESTRICT ]

REVOKE [ GRANT OPTION FOR ]
    { USAGE | ALL [ PRIVILEGES ] }
    ON DOMAIN domain_name [, ...]
    FROM { [ GROUP ] role_name | PUBLIC } [, ...]
    [ CASCADE | RESTRICT ]

REVOKE [ GRANT OPTION FOR ]
    { USAGE | ALL [ PRIVILEGES ] }
    ON FOREIGN DATA WRAPPER fdw_name [, ...]
    FROM { [ GROUP ] role_name | PUBLIC } [, ...]
    [ CASCADE | RESTRICT ]

REVOKE [ GRANT OPTION FOR ]
    { USAGE | ALL [ PRIVILEGES ] }
    ON FOREIGN SERVER server_name [, ...]
    FROM { [ GROUP ] role_name | PUBLIC } [, ...]
    [ CASCADE | RESTRICT ]
```

```
REVOKE [ GRANT OPTION FOR ]
    { EXECUTE | ALL [ PRIVILEGES ] }
    ON { FUNCTION function_name ( [ [ argmode ] [ arg_name ] arg_
type [, ...] ) [, ...]
         | ALL FUNCTIONS IN SCHEMA schema_name [, ...] }
    FROM { [ GROUP ] role_name | PUBLIC } [, ...]
    [ CASCADE | RESTRICT ]

REVOKE [ GRANT OPTION FOR ]
    { USAGE | ALL [ PRIVILEGES ] }
    ON LANGUAGE lang_name [, ...]
    FROM { [ GROUP ] role_name | PUBLIC } [, ...]
    [ CASCADE | RESTRICT ]

REVOKE [ GRANT OPTION FOR ]
    { { SELECT | UPDATE } [, ...] | ALL [ PRIVILEGES ] }
    ON LARGE OBJECT loid [, ...]
    FROM { [ GROUP ] role_name | PUBLIC } [, ...]
    [ CASCADE | RESTRICT ]

REVOKE [ GRANT OPTION FOR ]
    { { CREATE | USAGE } [, ...] | ALL [ PRIVILEGES ] }
    ON SCHEMA schema_name [, ...]
    FROM { [ GROUP ] role_name | PUBLIC } [, ...]
    [ CASCADE | RESTRICT ]

REVOKE [ GRANT OPTION FOR ]
    { CREATE | ALL [ PRIVILEGES ] }
    ON TABLESPACE tablespace_name [, ...]
    FROM { [ GROUP ] role_name | PUBLIC } [, ...]
    [ CASCADE | RESTRICT ]

REVOKE [ GRANT OPTION FOR ]
    { USAGE | ALL [ PRIVILEGES ] }
    ON TYPE type_name [, ...]
    FROM { [ GROUP ] role_name | PUBLIC } [, ...]
    [ CASCADE | RESTRICT ]

REVOKE [ ADMIN OPTION FOR ]
    role_name [, ...] FROM role_name [, ...]
    [ CASCADE | RESTRICT ]
```

描述

REVOKE 命令从一个或多个角色那里撤销先前授予的权限。关键字 PUBLIC 指的是由所有角色组成的隐式定义组。

有关各种权限类型的含义，请参见GRANT命令的描述。

请注意，任何特定角色实际拥有的权限，是直接授予给它的权限、授予给它当前所属任一角色的权限，以及授予给 PUBLIC 的权限之和。因此，例如，从 PUBLIC 撤销 SELECT 权限，并不一定意味着所有角色都失去了该对象上的 SELECT 权限：那些被直接授予该权限或通过其他角色获得该权限的角色仍然拥有它。类似地，从某个用户撤销 SELECT 权限，如果 PUBLIC 或其所属的其他角色仍然拥有 SELECT 权限，该用户仍可能使用 SELECT。

如果指定了 GRANT OPTION FOR，则只撤销该权限的授予选项，而不撤销权限本身。否则，权限和授予选项都会被撤销。

如果某个用户持有带授予选项的权限，并且已经将其授予其他用户，
那么其他用户持有的这些权限称为依赖权限。
若正在撤销第一个用户持有的该权限或其授予选项，并且存在依赖权限，则在指定 CASCADE 时这些依赖权限也会被一并撤销；否则，该撤销操作会失败。此递归撤销只影响那些通过某条可追溯到本 REVOKE 命令目标用户的用户链授予的权限。因此，如果受影响用户还通过其他用户获得了同一权限，那么他们实际上可能仍保有该权限。

在撤销某个表上的权限时，该表每一列上的对应列权限（若有）也会自动被撤销。反过来，
如果某个角色已被授予表级权限，那么从单独列上撤销同一权限不会产生任何效果。

在撤销角色成员资格时，GRANT OPTION 改称为 ADMIN OPTION，但行为类似。另请注意，
这种形式的命令不允许使用噪声词 GROUP。

注解

使用psql的 \dp 命令可以显示现有表和列上授予的权限。有关格式的信息，参见GRANT。对于非表对象，也有其他 \d 命令可以显示它们的权限。

用户只能撤销由自己直接授予的权限。例如，如果用户 A 已将某项带授予选项的权限授予用户 B，而用户 B 又将其授予用户 C，则用户 A 不能直接从 C 撤销该权限。相反，用户 A 可以从 B 撤销授予选项，并使用 CASCADE 选项，这样该权限就会进一步从 C 撤销。再例如，如果 A 和 B 都将同一权限授予了 C，A 可以撤销他自己授予的那份，但不能撤销 B 授予的那份，因此 C 实际上仍会拥有该权限。

当对象的非拥有者尝试对该对象执行 REVOKE 时，如果该用户在该对象上完全没有任何权限，命令将立即失败。只要该用户至少拥有某项权限，命令就会继续执行，
但只会撤销那些该用户持有授予选项的权限。如果未持有任何授予选项，REVOKE ALL PRIVILEGES 形式会发出警告；而其他形式如果命令中特别列出的任一权限未持有其授予选项，也会发出警告。（原则上，这些说明也适用于对象拥有者；
但由于拥有者始终被视为持有全部授予选项，这种情况实际上不会发生。）

如果超级用户选择执行 GRANT 或 REVOKE 命令，则该命令会像由受影响对象的拥有者发出那样执行。由于所有权限最终都来自对象拥有者（可能经由授予选项链间接传递），
超级用户可以撤销所有权限，但如上所述，这可能需要使用 CASCADE。

REVOKE 也可以由并非受影响对象拥有者的角色执行，只要该角色是拥有该对象之角色的成员，或者是持有该对象上 WITH GRANT OPTION 权限之角色的成员。在这种情况下，该命令会视同由实际拥有该对象的角色，或持有该对象上 WITH GRANT OPTION 权限的角色发出。例如，如果表 t1 由角色 g1 拥有，而角色 u1 是其成员，那么 u1 可以撤销 t1 上那些记录为由 g1 授出的权限。这包括由 u1 以及角色 g1 的其他成员作出的授权。

如果执行 REVOKE 的角色通过多条角色成员资格路径间接持有权限，则系统不会指明将使用哪一个上层角色来执行该命令。在这种情况下，最佳做法是使用 SET ROLE 切换成你希望作为其身份执行 REVOKE 的那个具体角色。否则，可能会撤销掉并非你本意要撤销的权限，或者根本没有撤销任何权限。

示例

撤销表 films 上授予所有用户的插入权限：

```
REVOKE INSERT ON films FROM PUBLIC;
```

撤销用户 manuel 在视图 kinds 上的所有权限：

```
REVOKE ALL PRIVILEGES ON kinds FROM manuel;
```

请注意，这实际上意味着“撤销所有由我授予的权限”。

撤销用户 joe 在角色 admins 中的成员资格：

```
REVOKE admins FROM joe;
```

兼容性

GRANT命令的兼容性注解同样适用于 `REVOKE`。按照标准，关键词 `RESTRICT` 或 `CASCADE` 是必需的，但 PostgreSQL 默认假定为 `RESTRICT`。

另见

GRANT

ROLLBACK

ROLLBACK — 中止当前事务

大纲

```
ROLLBACK [ WORK | TRANSACTION ]
```

描述

`ROLLBACK`回滚当前事务，并丢弃该事务所做的全部更新。

参数

`WORK`
`TRANSACTION`

可选关键字，没有任何作用。

注解

使用COMMIT成功结束事务。

在事务之外发出`ROLLBACK`并无害处，但会产生一条警告消息。

示例

要中止所有更改：

```
ROLLBACK;
```

兼容性

SQL 标准只规定了 `ROLLBACK` 和 `ROLLBACK WORK` 两种形式。除此之外，该命令完全符合标准。

另见

BEGIN, COMMIT, ROLLBACK TO SAVEPOINT

ROLLBACK PREPARED

ROLLBACK PREPARED — 回滚一个先前为两阶段提交而预备的事务

大纲

`ROLLBACK PREPARED` *`transaction_id`*

描述

`ROLLBACK PREPARED`回滚一个处于预备状态的事务。

参数

`transaction_id`

要回滚的事务的事务标识符。

注解

要回滚预备事务，执行者必须是最初执行该事务的同一用户，或者是超级用户；但不必处在执行该事务的同一会话中。

这个命令不能在事务块内执行。该预备事务会被立即回滚。

当前所有处于预备状态的事务都列在 `pg_prepared_xacts` 系统视图中。

示例

回滚事务标识符为`foobar`的事务：

```
ROLLBACK PREPARED 'foobar';
```

兼容性

`ROLLBACK PREPARED`是 PostgreSQL扩展。它旨在供外部事务管理系统使用，其中有些系统已被标准覆盖（例如 X/Open XA），但这些系统的 SQL 侧并未标准化。

另见

PREPARE TRANSACTION, COMMIT PREPARED

ROLLBACK TO SAVEPOINT

ROLLBACK TO SAVEPOINT — 回滚到一个保存点

大纲

ROLLBACK [WORK | TRANSACTION] TO [SAVEPOINT] *savepoint_name*

描述

回滚该保存点建立后执行的所有命令。该保存点仍然有效，如有需要，
之后还可以再次回滚到它。

ROLLBACK TO SAVEPOINT会隐式销毁在指定保存点之后建立的所有保存点。

参数

savepoint_name

要回滚到的保存点名称。

注解

使用RELEASE SAVEPOINT可以销毁一个保存点，而不丢弃在它建立之后执行的命令所产生的效果。

指定一个尚未建立的保存点名称是一种错误。

就保存点而言，游标带有一些非事务性的行为。凡是在某个保存点内打开的游标，
在回滚该保存点时都会被关闭。如果先前打开的游标在某个随后又被回滚的保存点内受到了FETCH或MOVE命令的影响，那么该游标会保留在FETCH使其指向的位置上（也就是说，由FETCH引起的游标移动不会被回滚）。关闭游标同样不会因回滚而撤销。不过，
如果游标查询导致了其他副作用（例如该查询调用的易变函数带来的副作用），
且这些副作用发生在后来被回滚的保存点期间，那么它们会被回滚。
如果某个游标的执行导致事务中止，该游标会进入不可执行状态，因此即使事务可以通过ROLLBACK TO SAVEPOINT恢复，该游标也不能再使用。

示例

要撤销在my_savepoint建立后执行的命令的效果：

```
ROLLBACK TO SAVEPOINT my_savepoint;
```

游标位置不受保存点回滚的影响：

```
BEGIN;

DECLARE foo CURSOR FOR SELECT 1 UNION SELECT 2;

SAVEPOINT foo;

FETCH 1 FROM foo;
 ?column?
----------
        1
```

```
ROLLBACK TO SAVEPOINT foo;

FETCH 1 FROM foo;
 ?column?
----------
        2

COMMIT;
```

兼容性

SQL标准规定关键字SAVEPOINT是强制的，但 PostgreSQL和Oracle允许省略它。SQL 只允许在ROLLBACK之后使用WORK作为噪声词，不允许使用TRANSACTION。此外，SQL 还有一个可选子句 AND [NO] CHAIN，而PostgreSQL当前尚不支持。除此之外，该命令符合 SQL 标准。

另见

BEGIN, COMMIT, RELEASE SAVEPOINT, ROLLBACK, SAVEPOINT

SAVEPOINT

SAVEPOINT — 在当前事务中定义一个新的保存点

大纲

```
SAVEPOINT savepoint_name
```

描述

`SAVEPOINT`在当前事务中建立一个新的保存点。

保存点是事务中的一种特殊标记，它允许回滚在其建立之后执行的所有命令，
从而将事务状态恢复到建立保存点时的状态。

参数

savepoint_name

赋予新保存点的名称。

注解

使用ROLLBACK TO SAVEPOINT回滚到保存点。使用RELEASE SAVEPOINT销毁保存点，同时保留其建立后执行的命令所产生的效果。

保存点只能在事务块内部建立。一个事务中可以定义多个保存点。

示例

要建立一个保存点，并在稍后撤销其建立后执行的所有命令的效果：

```
BEGIN;
    INSERT INTO table1 VALUES (1);
    SAVEPOINT my_savepoint;
    INSERT INTO table1 VALUES (2);
    ROLLBACK TO SAVEPOINT my_savepoint;
    INSERT INTO table1 VALUES (3);
COMMIT;
```

上述事务将插入值 1 和 3，而不会插入 2。

要建立保存点，并在稍后销毁它：

```
BEGIN;
    INSERT INTO table1 VALUES (3);
    SAVEPOINT my_savepoint;
    INSERT INTO table1 VALUES (4);
    RELEASE SAVEPOINT my_savepoint;
COMMIT;
```

上述事务将同时插入 3 和 4。

兼容性

SQL 要求在建立另一个同名保存点时自动销毁原有保存点。在 PostgreSQL中，旧保存点会被保留，不过在回滚或释放时只会使用最近建立的那个。（如果使用 `RELEASE SAVEPOINT`释

放较新的保存点，较旧的保存点将再次可供 `ROLLBACK TO SAVEPOINT` 和 `RELEASE SAVEPOINT` 使用。）除此之外，`SAVEPOINT` 完全符合 SQL。

另见

BEGIN, COMMIT, RELEASE SAVEPOINT, ROLLBACK, ROLLBACK TO SAVEPOINT

SECURITY LABEL

SECURITY LABEL — define or change a security label applied to an object

大纲

```
SECURITY LABEL [ FOR provider ] ON
{
  TABLE object_name |
  COLUMN table_name.column_name |
  AGGREGATE agg_name (agg_type [, ...] ) |
  DATABASE object_name |
  DOMAIN object_name |
  FOREIGN TABLE object_name
  FUNCTION function_name ( [ [ argmode ] [ argname ] argtype [, ...
] ] ) |
  LARGE OBJECT large_object_oid |
  [ PROCEDURAL ] LANGUAGE object_name |
  ROLE object_name |
  SCHEMA object_name |
  SEQUENCE object_name |
  TABLESPACE object_name |
  TYPE object_name |
  VIEW object_name
} IS 'label'
```

Description

`SECURITY LABEL` applies a security label to a database object. An arbitrary number of security labels, one per label provider, can be associated with a given database object. Label providers are loadable modules which register themselves by using the function `register_label_provider`.

> **注意**
>
> `register_label_provider` is not an SQL function; it can only be called from C code loaded into the backend.

The label provider determines whether a given label is valid and whether it is permissible to assign that label to a given object. The meaning of a given label is likewise at the discretion of the label provider. PostgreSQL places no restrictions on whether or how a label provider must interpret security labels; it merely provides a mechanism for storing them. In practice, this facility is intended to allow integration with label-based mandatory access control (MAC) systems such as SE-Linux. Such systems make all access control decisions based on object labels, rather than traditional discretionary access control (DAC) concepts such as users and groups.

Parameters

object_name
table_name.column_name
agg_name
function_name

The name of the object to be labeled. Names of tables, aggregates, domains, foreign tables, functions, sequences, types, and views can be schema-qualified.

provider

The name of the provider with which this label is to be associated. The named provider must be loaded and must consent to the proposed labeling operation. If exactly one provider is loaded, the provider name may be omitted for brevity.

arg_type

An input data type on which the aggregate function operates. To reference a zero-argument aggregate function, write `*` in place of the list of input data types.

argmode

The mode of a function argument: `IN`, `OUT`, `INOUT`, or `VARIADIC`. If omitted, the default is `IN`. Note that `SECURITY LABEL ON FUNCTION` does not actually pay any attention to `OUT` arguments, since only the input arguments are needed to determine the function's identity. So it is sufficient to list the `IN`, `INOUT`, and `VARIADIC` arguments.

argname

The name of a function argument. Note that `SECURITY LABEL ON FUNCTION` does not actually pay any attention to argument names, since only the argument data types are needed to determine the function's identity.

argtype

The data type(s) of the function's arguments (optionally schema-qualified), if any.

large_object_oid

The OID of the large object.

`PROCEDURAL`

This is a noise word.

label

The new security label, written as a string literal; or `NULL` to drop the security label.

Examples

The following example shows how the security label of a table might be changed.

```
SECURITY LABEL FOR selinux ON TABLE mytable IS 'system_u:object_r:
sepgsql_table_t:s0';
```

Compatibility

There is no `SECURITY LABEL` command in the SQL standard.

See Also

sepgsql, dummy_seclabel

SELECT

SELECT, TABLE, WITH — retrieve rows from a table or view

大纲

```
[ WITH [ RECURSIVE ] with_query [, ...] ]
SELECT [ ALL | DISTINCT [ ON ( expression [, ...] ) ] ]
    * | expression [ [ AS ] output_name ] [, ...]
    [ FROM from_item [, ...] ]
    [ WHERE condition ]
    [ GROUP BY expression [, ...] ]
    [ HAVING condition [, ...] ]
    [ WINDOW window_name AS ( window_definition ) [, ...] ]
    [ { UNION | INTERSECT | EXCEPT } [ ALL | DISTINCT ] select ]
    [ ORDER BY expression [ ASC | DESC | USING operator ] [ NULLS {
 FIRST | LAST } ] [, ...] ]
    [ LIMIT { count | ALL } ]
    [ OFFSET start [ ROW | ROWS ] ]
    [ FETCH { FIRST | NEXT } [ count ] { ROW | ROWS } ONLY ]
    [ FOR { UPDATE | SHARE } [ OF table_name [, ...] ] [ NOWAIT ] [
...] ]

where from_item can be one of:

    [ ONLY ] table_name [ * ] [ [ AS ] alias [ ( column_alias [, ..
.] ) ] ]
    ( select ) [ AS ] alias [ ( column_alias [, ...] ) ]
    with_query_name [ [ AS ] alias [ ( column_alias [, ...] ) ] ]
    function_name ( [ argument [, ...] ] ) [ AS ] alias [ ( column_
alias [, ...] | column_definition [, ...] ) ]
    function_name ( [ argument [, ...] ] ) AS ( column_definition [
, ...] )
    from_item [ NATURAL ] join_type from_item [ ON join_condition |
 USING ( join_column [, ...] ) ]

and with_query is:

    with_query_name [ ( column_name [, ...] ) ] AS ( select |
 values | insert | update | delete )

TABLE [ ONLY ] table_name [ * ]
```

Description

`SELECT` retrieves rows from zero or more tables. The general processing of `SELECT` is as follows:

1. All queries in the `WITH` list are computed. These effectively serve as temporary tables that can be referenced in the `FROM` list. A `WITH` query that is referenced more than once in `FROM` is computed only once. (See `WITH` Clause below.)

2. All elements in the `FROM` list are computed. (Each element in the `FROM` list is a real or virtual table.) If more than one element is specified in the `FROM` list, they are cross-joined together. (See `FROM` Clause below.)

3. If the `WHERE` clause is specified, all rows that do not satisfy the condition are eliminated from the output. (See `WHERE` Clause below.)

4. If the `GROUP BY` clause is specified, the output is combined into groups of rows that match on one or more values. If the `HAVING` clause is present, it eliminates groups that do not satisfy the given condition. (See `GROUP BY` Clause and `HAVING` Clause below.)

5. The actual output rows are computed using the `SELECT` output expressions for each selected row or row group. (See `SELECT` List below.)

6. `SELECT DISTINCT` eliminates duplicate rows from the result. `SELECT DISTINCT ON` eliminates rows that match on all the specified expressions. `SELECT ALL` (the default) will return all candidate rows, including duplicates. (See `DISTINCT` Clause below.)

7. Using the operators `UNION`, `INTERSECT`, and `EXCEPT`, the output of more than one `SELECT` statement can be combined to form a single result set. The `UNION` operator returns all rows that are in one or both of the result sets. The `INTERSECT` operator returns all rows that are strictly in both result sets. The `EXCEPT` operator returns the rows that are in the first result set but not in the second. In all three cases, duplicate rows are eliminated unless `ALL` is specified. The noise word `DISTINCT` can be added to explicitly specify eliminating duplicate rows. Notice that `DISTINCT` is the default behavior here, even though `ALL` is the default for `SELECT` itself. (See `UNION` Clause, `INTERSECT` Clause, and `EXCEPT` Clause below.)

8. If the `ORDER BY` clause is specified, the returned rows are sorted in the specified order. If `ORDER BY` is not given, the rows are returned in whatever order the system finds fastest to produce. (See `ORDER BY` Clause below.)

9. If the `LIMIT` (or `FETCH FIRST`) or `OFFSET` clause is specified, the `SELECT` statement only returns a subset of the result rows. (See `LIMIT` Clause below.)

10. If `FOR UPDATE` or `FOR SHARE` is specified, the `SELECT` statement locks the selected rows against concurrent updates. (See `FOR UPDATE`/`FOR SHARE` Clause below.)

You must have `SELECT` privilege on each column used in a `SELECT` command. The use of `FOR UPDATE` or `FOR SHARE` requires `UPDATE` privilege as well (for at least one column of each table so selected).

Parameters

`WITH` Clause

The `WITH` clause allows you to specify one or more subqueries that can be referenced by name in the primary query. The subqueries effectively act as temporary tables or views for the duration of the primary query. Each subquery can be a `SELECT`, `VALUES`, `INSERT`, `UPDATE` or `DELETE` statement. When writing a data-modifying statement (`INSERT`, `UPDATE` or `DELETE`) in `WITH`, it is usual to include a `RETURNING` clause. It is the output of `RETURNING`, not the underlying table that the statement modifies, that forms the temporary table that is read by the primary query. If `RETURNING` is omitted, the statement is still executed, but it produces no output so it cannot be referenced as a table by the primary query.

A name (without schema qualification) must be specified for each `WITH` query. Optionally, a list of column names can be specified; if this is omitted, the column names are inferred from the subquery.

If RECURSIVE is specified, it allows a SELECT subquery to reference itself by name. Such a subquery must have the form

non_recursive_term UNION [ALL | DISTINCT] *recursive_term*

where the recursive self-reference must appear on the right-hand side of the UNION . Only one recursive self-reference is permitted per query. Recursive data-modifying statements are not supported, but you can use the results of a recursive SELECT query in a data-modifying statement. See 第 7.8 节 for an example.

Another effect of RECURSIVE is that WITH queries need not be ordered: a query can reference another one that is later in the list. (However, circular references, or mutual recursion, are not implemented.) Without RECURSIVE, WITH queries can only reference sibling WITH queries that are earlier in the WITH list.

A key property of WITH queries is that they are evaluated only once per execution of the primary query, even if the primary query refers to them more than once. In particular, data-modifying statements are guaranteed to be executed once and only once, regardless of whether the primary query reads all or any of their output.

The primary query and the WITH queries are all (notionally) executed at the same time. This implies that the effects of a data-modifying statement in WITH cannot be seen from other parts of the query, other than by reading its RETURNING output. If two such data-modifying statements attempt to modify the same row, the results are unspecified.

See 第 7.8 节 for additional information.

FROM Clause

The FROM clause specifies one or more source tables for the SELECT. If multiple sources are specified, the result is the Cartesian product (cross join) of all the sources. But usually qualification conditions are added to restrict the returned rows to a small subset of the Cartesian product.

The FROM clause can contain the following elements:

table_name

The name (optionally schema-qualified) of an existing table or view. If ONLY is specified before the table name, only that table is scanned. If ONLY is not specified, the table and all its descendant tables (if any) are scanned. Optionally, * can be specified after the table name to explicitly indicate that descendant tables are included.

alias

A substitute name for the FROM item containing the alias. An alias is used for brevity or to eliminate ambiguity for self-joins (where the same table is scanned multiple times). When an alias is provided, it completely hides the actual name of the table or function; for example given FROM foo AS f, the remainder of the SELECT must refer to this FROM item as f not foo. If an alias is written, a column alias list can also be written to provide substitute names for one or more columns of the table.

select

A sub-SELECT can appear in the FROM clause. This acts as though its output were created as a temporary table for the duration of this single SELECT command. Note that the sub-SELECT must be surrounded by parentheses, and an alias must be provided for it. A VALUES command can also be used here.

with_query_name

A WITH query is referenced by writing its name, just as though the query's name were a table name. (In fact, the WITH query hides any real table of the same name for the purposes of the primary query. If necessary, you can refer to a real table of the same name by schema-qualifying the table's name.) An alias can be provided in the same way as for a table.

function_name

Function calls can appear in the FROM clause. (This is especially useful for functions that return result sets, but any function can be used.) This acts as though its output were created as a temporary table for the duration of this single SELECT command. An alias can also be used. If an alias is written, a column alias list can also be written to provide substitute names for one or more attributes of the function's composite return type. If the function has been defined as returning the record data type, then an alias or the key word AS must be present, followed by a column definition list in the form (*column_name data_type* [, ...]). The column definition list must match the actual number and types of columns returned by the function.

join_type

One of

- [INNER] JOIN
- LEFT [OUTER] JOIN
- RIGHT [OUTER] JOIN
- FULL [OUTER] JOIN
- CROSS JOIN

For the INNER and OUTER join types, a join condition must be specified, namely exactly one of NATURAL, ON *join_condition*, or USING (*join_column* [, ...]). See below for the meaning. For CROSS JOIN, none of these clauses can appear.

A JOIN clause combines two FROM items. Use parentheses if necessary to determine the order of nesting. In the absence of parentheses, JOINs nest left-to-right. In any case JOIN binds more tightly than the commas separating FROM items.

CROSS JOIN and INNER JOIN produce a simple Cartesian product, the same result as you get from listing the two items at the top level of FROM, but restricted by the join condition (if any). CROSS JOIN is equivalent to INNER JOIN ON (TRUE), that is, no rows are removed by qualification. These join types are just a notational convenience, since they do nothing you couldn't do with plain FROM and WHERE.

LEFT OUTER JOIN returns all rows in the qualified Cartesian product (i.e., all combined rows that pass its join condition), plus one copy of each row in the left-hand table for which there was no right-hand row that passed the join condition. This left-hand row is extended to the full width of the joined table by inserting null values for the right-hand columns. Note that only the JOIN clause's own condition is considered while deciding which rows have matches. Outer conditions are applied afterwards.

Conversely, RIGHT OUTER JOIN returns all the joined rows, plus one row for each unmatched right-hand row (extended with nulls on the left). This is just a notational convenience, since you could convert it to a LEFT OUTER JOIN by switching the left and right inputs.

`FULL OUTER JOIN` returns all the joined rows, plus one row for each unmatched left-hand row (extended with nulls on the right), plus one row for each unmatched right-hand row (extended with nulls on the left).

`ON` *`join_condition`*

`join_condition` is an expression resulting in a value of type `boolean` (similar to a `WHERE` clause) that specifies which rows in a join are considered to match.

`USING ( join_column [, ...] )`

A clause of the form `USING ( a, b, ... )` is shorthand for `ON left_table.a = right_table.a AND left_table.b = right_table.b ...`. Also, `USING` implies that only one of each pair of equivalent columns will be included in the join output, not both.

`NATURAL`

`NATURAL` is shorthand for a `USING` list that mentions all columns in the two tables that have matching names. If there are no common column names, `NATURAL` is equivalent to `ON TRUE`.

`WHERE` Clause

The optional `WHERE` clause has the general form

`WHERE condition`

where *`condition`* is any expression that evaluates to a result of type `boolean`. Any row that does not satisfy this condition will be eliminated from the output. A row satisfies the condition if it returns true when the actual row values are substituted for any variable references.

`GROUP BY` Clause

The optional `GROUP BY` clause has the general form

`GROUP BY expression [, ...]`

`GROUP BY` will condense into a single row all selected rows that share the same values for the grouped expressions. *`expression`* can be an input column name, or the name or ordinal number of an output column (`SELECT` list item), or an arbitrary expression formed from input-column values. In case of ambiguity, a `GROUP BY` name will be interpreted as an input-column name rather than an output column name.

Aggregate functions, if any are used, are computed across all rows making up each group, producing a separate value for each group (whereas without `GROUP BY`, an aggregate produces a single value computed across all the selected rows). When `GROUP BY` is present, it is not valid for the `SELECT` list expressions to refer to ungrouped columns except within aggregate functions or if the ungrouped column is functionally dependent on the grouped columns, since there would otherwise be more than one possible value to return for an ungrouped column. A functional dependency exists if the grouped columns (or a subset thereof) are the primary key of the table containing the ungrouped column.

`HAVING` Clause

The optional `HAVING` clause has the general form

`HAVING condition`

where *condition* is the same as specified for the WHERE clause.

HAVING eliminates group rows that do not satisfy the condition. HAVING is different from WHERE: WHERE filters individual rows before the application of GROUP BY, while HAVING filters group rows created by GROUP BY. Each column referenced in *condition* must unambiguously reference a grouping column, unless the reference appears within an aggregate function.

The presence of HAVING turns a query into a grouped query even if there is no GROUP BY clause. This is the same as what happens when the query contains aggregate functions but no GROUP BY clause. All the selected rows are considered to form a single group, and the SELECT list and HAVING clause can only reference table columns from within aggregate functions. Such a query will emit a single row if the HAVING condition is true, zero rows if it is not true.

WINDOW Clause

The optional WINDOW clause has the general form

```
WINDOW window_name AS ( window_definition ) [, ...]
```

where *window_name* is a name that can be referenced from OVER clauses or subsequent window definitions, and *window_definition* is

```
[ existing_window_name ]
[ PARTITION BY expression [, ...] ]
[ ORDER BY expression [ ASC | DESC | USING operator ] [ NULLS {
 FIRST | LAST } ] [, ...] ]
[ frame_clause ]
```

If an *existing_window_name* is specified it must refer to an earlier entry in the WINDOW list; the new window copies its partitioning clause from that entry, as well as its ordering clause if any. In this case the new window cannot specify its own PARTITION BY clause, and it can specify ORDER BY only if the copied window does not have one. The new window always uses its own frame clause; the copied window must not specify a frame clause.

The elements of the PARTITION BY list are interpreted in much the same fashion as elements of a GROUP BY Clause, except that they are always simple expressions and never the name or number of an output column. Another difference is that these expressions can contain aggregate function calls, which are not allowed in a regular GROUP BY clause. They are allowed here because windowing occurs after grouping and aggregation.

Similarly, the elements of the ORDER BY list are interpreted in much the same fashion as elements of an ORDER BY Clause, except that the expressions are always taken as simple expressions and never the name or number of an output column.

The optional *frame_clause* defines the window frame for window functions that depend on the frame (not all do). The window frame is a set of related rows for each row of the query (called the current row). The *frame_clause* can be one of

```
{ RANGE | ROWS } frame_start
{ RANGE | ROWS } BETWEEN frame_start AND frame_end
```

where *frame_start* and *frame_end* can be one of

```
UNBOUNDED PRECEDING
value PRECEDING
CURRENT ROW
```

value FOLLOWING
UNBOUNDED FOLLOWING

If *frame_end* is omitted it defaults to CURRENT ROW. Restrictions are that *frame_start* cannot be UNBOUNDED FOLLOWING, *frame_end* cannot be UNBOUNDED PRECEDING, and the *frame_end* choice cannot appear earlier in the above list than the *frame_start* choice – for example RANGE BETWEEN CURRENT ROW AND *value* PRECEDING is not allowed.

The default framing option is RANGE UNBOUNDED PRECEDING, which is the same as RANGE BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW; it sets the frame to be all rows from the partition start up through the current row's last peer in the ORDER BY ordering (which means all rows if there is no ORDER BY). In general, UNBOUNDED PRECEDING means that the frame starts with the first row of the partition, and similarly UNBOUNDED FOLLOWING means that the frame ends with the last row of the partition (regardless of RANGE or ROWS mode). In ROWS mode, CURRENT ROW means that the frame starts or ends with the current row; but in RANGE mode it means that the frame starts or ends with the current row's first or last peer in the ORDER BY ordering. The *value* PRECEDING and *value* FOLLOWING cases are currently only allowed in ROWS mode. They indicate that the frame starts or ends with the row that many rows before or after the current row. *value* must be an integer expression not containing any variables, aggregate functions, or window functions. The value must not be null or negative; but it can be zero, which selects the current row itself.

Beware that the ROWS options can produce unpredictable results if the ORDER BY ordering does not order the rows uniquely. The RANGE options are designed to ensure that rows that are peers in the ORDER BY ordering are treated alike; any two peer rows will be both in or both not in the frame.

The purpose of a WINDOW clause is to specify the behavior of window functions appearing in the query's SELECT List or ORDER BY Clause. These functions can reference the WINDOW clause entries by name in their OVER clauses. A WINDOW clause entry does not have to be referenced anywhere, however; if it is not used in the query it is simply ignored. It is possible to use window functions without any WINDOW clause at all, since a window function call can specify its window definition directly in its OVER clause. However, the WINDOW clause saves typing when the same window definition is needed for more than one window function.

Window functions are described in detail in 第 3.5 节, 第 4.2.8 节, and 第 7.2.4 节.

SELECT List

The SELECT list (between the key words SELECT and FROM) specifies expressions that form the output rows of the SELECT statement. The expressions can (and usually do) refer to columns computed in the FROM clause.

Just as in a table, every output column of a SELECT has a name. In a simple SELECT this name is just used to label the column for display, but when the SELECT is a sub-query of a larger query, the name is seen by the larger query as the column name of the virtual table produced by the sub-query. To specify the name to use for an output column, write AS *output_name* after the column's expression. (You can omit AS, but only if the desired output name does not match any PostgreSQL keyword (see 附录 C). For protection against possible future keyword additions, it is recommended that you always either write AS or double-quote the output name.) If you do not specify a column name, a name is chosen automatically by PostgreSQL. If the column's expression is a simple column reference then the chosen name is the same as that column's name. In more complex cases a function or type name may be used, or the system may fall back on a generated name such as ?column?.

An output column's name can be used to refer to the column's value in `ORDER BY` and `GROUP BY` clauses, but not in the `WHERE` or `HAVING` clauses; there you must write out the expression instead.

Instead of an expression, `*` can be written in the output list as a shorthand for all the columns of the selected rows. Also, you can write *`table_name`*`.*` as a shorthand for the columns coming from just that table. In these cases it is not possible to specify new names with `AS`; the output column names will be the same as the table columns' names.

`DISTINCT` Clause

If `SELECT DISTINCT` is specified, all duplicate rows are removed from the result set (one row is kept from each group of duplicates). `SELECT ALL` specifies the opposite: all rows are kept; that is the default.

`SELECT DISTINCT ON (` *`expression`* `[, ...] )` keeps only the first row of each set of rows where the given expressions evaluate to equal. The `DISTINCT ON` expressions are interpreted using the same rules as for `ORDER BY` (see above). Note that the "first row" of each set is unpredictable unless `ORDER BY` is used to ensure that the desired row appears first. For example:

```
SELECT DISTINCT ON (location) location, time, report
    FROM weather_reports
    ORDER BY location, time DESC;
```

retrieves the most recent weather report for each location. But if we had not used `ORDER BY` to force descending order of time values for each location, we'd have gotten a report from an unpredictable time for each location.

The `DISTINCT ON` expression(s) must match the leftmost `ORDER BY` expression(s). The `ORDER BY` clause will normally contain additional expression(s) that determine the desired precedence of rows within each `DISTINCT ON` group.

`UNION` Clause

The `UNION` clause has this general form:

`select_statement` `UNION [ ALL | DISTINCT ]` *`select_statement`*

`select_statement` is any `SELECT` statement without an `ORDER BY`, `LIMIT`, `FOR UPDATE`, or `FOR SHARE` clause. (`ORDER BY` and `LIMIT` can be attached to a subexpression if it is enclosed in parentheses. Without parentheses, these clauses will be taken to apply to the result of the `UNION`, not to its right-hand input expression.)

The `UNION` operator computes the set union of the rows returned by the involved `SELECT` statements. A row is in the set union of two result sets if it appears in at least one of the result sets. The two `SELECT` statements that represent the direct operands of the `UNION` must produce the same number of columns, and corresponding columns must be of compatible data types.

The result of `UNION` does not contain any duplicate rows unless the `ALL` option is specified. `ALL` prevents elimination of duplicates. (Therefore, `UNION ALL` is usually significantly quicker than `UNION`; use `ALL` when you can.) `DISTINCT` can be written to explicitly specify the default behavior of eliminating duplicate rows.

Multiple `UNION` operators in the same `SELECT` statement are evaluated left to right, unless otherwise indicated by parentheses.

Currently, `FOR UPDATE` and `FOR SHARE` cannot be specified either for a `UNION` result or for any input of a `UNION`.

`INTERSECT` Clause

The `INTERSECT` clause has this general form:

```
select_statement INTERSECT [ ALL | DISTINCT ] select_statement
```

`select_statement` is any `SELECT` statement without an `ORDER BY`, `LIMIT`, `FOR UPDATE`, or `FOR SHARE` clause.

The `INTERSECT` operator computes the set intersection of the rows returned by the involved `SELECT` statements. A row is in the intersection of two result sets if it appears in both result sets.

The result of `INTERSECT` does not contain any duplicate rows unless the `ALL` option is specified. With `ALL`, a row that has *m* duplicates in the left table and *n* duplicates in the right table will appear min(*m*,*n*) times in the result set. `DISTINCT` can be written to explicitly specify the default behavior of eliminating duplicate rows.

Multiple `INTERSECT` operators in the same `SELECT` statement are evaluated left to right, unless parentheses dictate otherwise. `INTERSECT` binds more tightly than `UNION`. That is, `A UNION B INTERSECT C` will be read as `A UNION (B INTERSECT C)`.

Currently, `FOR UPDATE` and `FOR SHARE` cannot be specified either for an `INTERSECT` result or for any input of an `INTERSECT`.

`EXCEPT` Clause

The `EXCEPT` clause has this general form:

```
select_statement EXCEPT [ ALL | DISTINCT ] select_statement
```

`select_statement` is any `SELECT` statement without an `ORDER BY`, `LIMIT`, `FOR UPDATE`, or `FOR SHARE` clause.

The `EXCEPT` operator computes the set of rows that are in the result of the left `SELECT` statement but not in the result of the right one.

The result of `EXCEPT` does not contain any duplicate rows unless the `ALL` option is specified. With `ALL`, a row that has *m* duplicates in the left table and *n* duplicates in the right table will appear max(*m*-*n*,0) times in the result set. `DISTINCT` can be written to explicitly specify the default behavior of eliminating duplicate rows.

Multiple `EXCEPT` operators in the same `SELECT` statement are evaluated left to right, unless parentheses dictate otherwise. `EXCEPT` binds at the same level as `UNION`.

Currently, `FOR UPDATE` and `FOR SHARE` cannot be specified either for an `EXCEPT` result or for any input of an `EXCEPT`.

`ORDER BY` Clause

The optional `ORDER BY` clause has this general form:

```
ORDER BY expression [ ASC | DESC | USING operator ] [ NULLS { FIRST
 | LAST } ] [, ...]
```

The `ORDER BY` clause causes the result rows to be sorted according to the specified expression(s). If two rows are equal according to the leftmost expression, they are

compared according to the next expression and so on. If they are equal according to all specified expressions, they are returned in an implementation-dependent order.

Each *expression* can be the name or ordinal number of an output column (SELECT list item), or it can be an arbitrary expression formed from input-column values.

The ordinal number refers to the ordinal (left-to-right) position of the output column. This feature makes it possible to define an ordering on the basis of a column that does not have a unique name. This is never absolutely necessary because it is always possible to assign a name to an output column using the AS clause.

It is also possible to use arbitrary expressions in the ORDER BY clause, including columns that do not appear in the SELECT output list. Thus the following statement is valid:

```
SELECT name FROM distributors ORDER BY code;
```

A limitation of this feature is that an ORDER BY clause applying to the result of a UNION, INTERSECT, or EXCEPT clause can only specify an output column name or number, not an expression.

If an ORDER BY expression is a simple name that matches both an output column name and an input column name, ORDER BY will interpret it as the output column name. This is the opposite of the choice that GROUP BY will make in the same situation. This inconsistency is made to be compatible with the SQL standard.

Optionally one can add the key word ASC (ascending) or DESC (descending) after any expression in the ORDER BY clause. If not specified, ASC is assumed by default. Alternatively, a specific ordering operator name can be specified in the USING clause. An ordering operator must be a less-than or greater-than member of some B-tree operator family. ASC is usually equivalent to USING < and DESC is usually equivalent to USING >. (But the creator of a user-defined data type can define exactly what the default sort ordering is, and it might correspond to operators with other names.)

If NULLS LAST is specified, null values sort after all non-null values; if NULLS FIRST is specified, null values sort before all non-null values. If neither is specified, the default behavior is NULLS LAST when ASC is specified or implied, and NULLS FIRST when DESC is specified (thus, the default is to act as though nulls are larger than non-nulls). When USING is specified, the default nulls ordering depends on whether the operator is a less-than or greater-than operator.

Note that ordering options apply only to the expression they follow; for example ORDER BY x, y DESC does not mean the same thing as ORDER BY x DESC, y DESC.

Character-string data is sorted according to the collation that applies to the column being sorted. That can be overridden at need by including a COLLATE clause in the *expression*, for example ORDER BY mycolumn COLLATE "en_US". For more information see 第 4.2.10 节 and 第 22.2 节.

LIMIT Clause

The LIMIT clause consists of two independent sub-clauses:

```
LIMIT { count | ALL }
OFFSET start
```

count specifies the maximum number of rows to return, while *start* specifies the number of rows to skip before starting to return rows. When both are specified, *start* rows are skipped before starting to count the *count* rows to be returned.

If the *count* expression evaluates to NULL, it is treated as LIMIT ALL, i.e., no limit. If *start* evaluates to NULL, it is treated the same as OFFSET 0.

SQL:2008 introduced a different syntax to achieve the same result, which PostgreSQL also supports. It is:

```
OFFSET start { ROW | ROWS }
FETCH { FIRST | NEXT } [ count ] { ROW | ROWS } ONLY
```

In this syntax, to write anything except a simple integer constant for *start* or *count*, you must write parentheses around it. If *count* is omitted in a FETCH clause, it defaults to 1. ROW and ROWS as well as FIRST and NEXT are noise words that don't influence the effects of these clauses. According to the standard, the OFFSET clause must come before the FETCH clause if both are present; but PostgreSQL is laxer and allows either order.

When using LIMIT, it is a good idea to use an ORDER BY clause that constrains the result rows into a unique order. Otherwise you will get an unpredictable subset of the query's rows – you might be asking for the tenth through twentieth rows, but tenth through twentieth in what ordering? You don't know what ordering unless you specify ORDER BY.

The query planner takes LIMIT into account when generating a query plan, so you are very likely to get different plans (yielding different row orders) depending on what you use for LIMIT and OFFSET. Thus, using different LIMIT/OFFSET values to select different subsets of a query result will give inconsistent results unless you enforce a predictable result ordering with ORDER BY. This is not a bug; it is an inherent consequence of the fact that SQL does not promise to deliver the results of a query in any particular order unless ORDER BY is used to constrain the order.

It is even possible for repeated executions of the same LIMIT query to return different subsets of the rows of a table, if there is not an ORDER BY to enforce selection of a deterministic subset. Again, this is not a bug; determinism of the results is simply not guaranteed in such a case.

FOR UPDATE/FOR SHARE Clause

The FOR UPDATE clause has this form:

```
FOR UPDATE [ OF table_name [, ...] ] [ NOWAIT ]
```

The closely related FOR SHARE clause has this form:

```
FOR SHARE [ OF table_name [, ...] ] [ NOWAIT ]
```

FOR UPDATE causes the rows retrieved by the SELECT statement to be locked as though for update. This prevents them from being modified or deleted by other transactions until the current transaction ends. That is, other transactions that attempt UPDATE, DELETE, or SELECT FOR UPDATE of these rows will be blocked until the current transaction ends. Also, if an UPDATE, DELETE, or SELECT FOR UPDATE from another transaction has already locked a selected row or rows, SELECT FOR UPDATE will wait for the other transaction to complete, and will then lock and return the updated row (or no row, if the row was deleted). Within a REPEATABLE READ or SERIALIZABLE transaction, however, an error will be thrown if a row to be locked has changed since the transaction started. For further discussion see 第 13 章.

FOR SHARE behaves similarly, except that it acquires a shared rather than exclusive lock on each retrieved row. A shared lock blocks other transactions from performing UPDATE, DELETE, or SELECT FOR UPDATE on these rows, but it does not prevent them from performing SELECT FOR SHARE.

To prevent the operation from waiting for other transactions to commit, use the `NOWAIT` option. With `NOWAIT`, the statement reports an error, rather than waiting, if a selected row cannot be locked immediately. Note that `NOWAIT` applies only to the row-level lock(s) – the required `ROW SHARE` table-level lock is still taken in the ordinary way (see 第 13 章). You can use LOCK with the `NOWAIT` option first, if you need to acquire the table-level lock without waiting.

If specific tables are named in `FOR UPDATE` or `FOR SHARE`, then only rows coming from those tables are locked; any other tables used in the `SELECT` are simply read as usual. A `FOR UPDATE` or `FOR SHARE` clause without a table list affects all tables used in the statement. If `FOR UPDATE` or `FOR SHARE` is applied to a view or sub-query, it affects all tables used in the view or sub-query. However, `FOR UPDATE`/`FOR SHARE` do not apply to `WITH` queries referenced by the primary query. If you want row locking to occur within a `WITH` query, specify `FOR UPDATE` or `FOR SHARE` within the `WITH` query.

Multiple `FOR UPDATE` and `FOR SHARE` clauses can be written if it is necessary to specify different locking behavior for different tables. If the same table is mentioned (or implicitly affected) by both `FOR UPDATE` and `FOR SHARE` clauses, then it is processed as `FOR UPDATE`. Similarly, a table is processed as `NOWAIT` if that is specified in any of the clauses affecting it.

`FOR UPDATE` and `FOR SHARE` cannot be used in contexts where returned rows cannot be clearly identified with individual table rows; for example they cannot be used with aggregation.

When `FOR UPDATE` or `FOR SHARE` appears at the top level of a `SELECT` query, the rows that are locked are exactly those that are returned by the query; in the case of a join query, the rows locked are those that contribute to returned join rows. In addition, rows that satisfied the query conditions as of the query snapshot will be locked, although they will not be returned if they were updated after the snapshot and no longer satisfy the query conditions. If a `LIMIT` is used, locking stops once enough rows have been returned to satisfy the limit (but note that rows skipped over by `OFFSET` will get locked). Similarly, if `FOR UPDATE` or `FOR SHARE` is used in a cursor's query, only rows actually fetched or stepped past by the cursor will be locked.

When `FOR UPDATE` or `FOR SHARE` appears in a sub-`SELECT`, the rows locked are those returned to the outer query by the sub-query. This might involve fewer rows than inspection of the sub-query alone would suggest, since conditions from the outer query might be used to optimize execution of the sub-query. For example,

```
SELECT * FROM (SELECT * FROM mytable FOR UPDATE) ss WHERE col1 = 5;
```

will lock only rows having `col1 = 5`, even though that condition is not textually within the sub-query.

小心

Avoid locking a row and then modifying it within a later savepoint or PL/pgSQL exception block. A subsequent rollback would cause the lock to be lost. For example:

```
BEGIN;
SELECT * FROM mytable WHERE key = 1 FOR UPDATE;
SAVEPOINT s;
UPDATE mytable SET ... WHERE key = 1;
ROLLBACK TO s;
```

After the `ROLLBACK`, the row is effectively unlocked, rather than returned to its pre-savepoint state of being locked but not modified. This hazard occurs if a

row locked in the current transaction is updated or deleted, or if a shared lock is upgraded to exclusive: in all these cases, the former lock state is forgotten. If the transaction is then rolled back to a state between the original locking command and the subsequent change, the row will appear not to be locked at all. This is an implementation deficiency which will be addressed in a future release of PostgreSQL.

小心

It is possible for a `SELECT` command running at the `READ COMMITTED` transaction isolation level and using `ORDER BY` and `FOR UPDATE/SHARE` to return rows out of order. This is because `ORDER BY` is applied first. The command sorts the result, but might then block trying to obtain a lock on one or more of the rows. Once the `SELECT` unblocks, some of the ordering column values might have been modified, leading to those rows appearing to be out of order (though they are in order in terms of the original column values). This can be worked around at need by placing the `FOR UPDATE/SHARE` clause in a sub-query, for example

```
SELECT * FROM (SELECT * FROM mytable FOR UPDATE) ss ORDER BY
 column1;
```

Note that this will result in locking all rows of `mytable`, whereas `FOR UPDATE` at the top level would lock only the actually returned rows. This can make for a significant performance difference, particularly if the `ORDER BY` is combined with `LIMIT` or other restrictions. So this technique is recommended only if concurrent updates of the ordering columns are expected and a strictly sorted result is required.

At the `REPEATABLE READ` or `SERIALIZABLE` transaction isolation level this would cause a serialization failure (with a `SQLSTATE` of `'40001'`), so there is no possibility of receiving rows out of order under these isolation levels.

`TABLE` Command

The command

```
TABLE name
```

is completely equivalent to

```
SELECT * FROM name
```

It can be used as a top-level command or as a space-saving syntax variant in parts of complex queries.

Examples

To join the table `films` with the table `distributors`:

```
SELECT f.title, f.did, d.name, f.date_prod, f.kind
    FROM distributors d, films f
    WHERE f.did = d.did

       title       | did |     name     | date_prod  |   kind
```

```
------------------+-----+--------------+------------+----------
 The Third Man     | 101 | British Lion | 1949-12-23 | Drama
 The African Queen | 101 | British Lion | 1951-08-11 | Romantic
 ...
```

To sum the column `len` of all films and group the results by `kind`:

```
SELECT kind, sum(len) AS total FROM films GROUP BY kind;

   kind   | total
----------+-------
 Action   | 07:34
 Comedy   | 02:58
 Drama    | 14:28
 Musical  | 06:42
 Romantic | 04:38
```

To sum the column `len` of all films, group the results by `kind` and show those group totals that are less than 5 hours:

```
SELECT kind, sum(len) AS total
    FROM films
    GROUP BY kind
    HAVING sum(len) < interval '5 hours';

   kind   | total
----------+-------
 Comedy   | 02:58
 Romantic | 04:38
```

The following two examples are identical ways of sorting the individual results according to the contents of the second column (`name`):

```
SELECT * FROM distributors ORDER BY name;
SELECT * FROM distributors ORDER BY 2;

 did |       name
-----+------------------
 109 | 20th Century Fox
 110 | Bavaria Atelier
 101 | British Lion
 107 | Columbia
 102 | Jean Luc Godard
 113 | Luso films
 104 | Mosfilm
 103 | Paramount
 106 | Toho
 105 | United Artists
 111 | Walt Disney
 112 | Warner Bros.
 108 | Westward
```

The next example shows how to obtain the union of the tables `distributors` and `actors`, restricting the results to those that begin with the letter W in each table. Only distinct rows are wanted, so the key word `ALL` is omitted.

```
distributors:               actors:
 did |     name              id |     name
-----+--------------        ----+----------------
```

```
 108 | Westward              1 | Woody Allen
 111 | Walt Disney           2 | Warren Beatty
 112 | Warner Bros.          3 | Walter Matthau
 ...                        ...

SELECT distributors.name
    FROM distributors
    WHERE distributors.name LIKE 'W%'
UNION
SELECT actors.name
    FROM actors
    WHERE actors.name LIKE 'W%';

      name
----------------
 Walt Disney
 Walter Matthau
 Warner Bros.
 Warren Beatty
 Westward
 Woody Allen
```

This example shows how to use a function in the `FROM` clause, both with and without a column definition list:

```
CREATE FUNCTION distributors(int) RETURNS SETOF distributors AS $$
    SELECT * FROM distributors WHERE did = $1;
$$ LANGUAGE SQL;

SELECT * FROM distributors(111);
 did |    name
-----+-------------
 111 | Walt Disney

CREATE FUNCTION distributors_2(int) RETURNS SETOF record AS $$
    SELECT * FROM distributors WHERE did = $1;
$$ LANGUAGE SQL;

SELECT * FROM distributors_2(111) AS (f1 int, f2 text);
 f1  |     f2
-----+-------------
 111 | Walt Disney
```

This example shows how to use a simple `WITH` clause:

```
WITH t AS (
    SELECT random() as x FROM generate_series(1, 3)
  )
SELECT * FROM t
UNION ALL
SELECT * FROM t

         x
--------------------
  0.534150459803641
  0.520092216785997
 0.0735620250925422
  0.534150459803641
  0.520092216785997
```

```
 0.0735620250925422
```

Notice that the WITH query was evaluated only once, so that we got two sets of the same three random values.

This example uses WITH RECURSIVE to find all subordinates (direct or indirect) of the employee Mary, and their level of indirectness, from a table that shows only direct subordinates:

```
WITH RECURSIVE employee_recursive(distance, employee_name, manager_
name) AS (
    SELECT 1, employee_name, manager_name
    FROM employee
    WHERE manager_name = 'Mary'
  UNION ALL
    SELECT er.distance + 1, e.employee_name, e.manager_name
    FROM employee_recursive er, employee e
    WHERE er.employee_name = e.manager_name
  )
SELECT distance, employee_name FROM employee_recursive;
```

Notice the typical form of recursive queries: an initial condition, followed by UNION, followed by the recursive part of the query. Be sure that the recursive part of the query will eventually return no tuples, or else the query will loop indefinitely. (See 第 7.8 节 for more examples.)

Compatibility

Of course, the SELECT statement is compatible with the SQL standard. But there are some extensions and some missing features.

Omitted FROM Clauses

PostgreSQL allows one to omit the FROM clause. It has a straightforward use to compute the results of simple expressions:

```
SELECT 2+2;

 ?column?
----------
        4
```

Some other SQL databases cannot do this except by introducing a dummy one-row table from which to do the SELECT.

Note that if a FROM clause is not specified, the query cannot reference any database tables. For example, the following query is invalid:

```
SELECT distributors.* WHERE distributors.name = 'Westward';
```

PostgreSQL releases prior to 8.1 would accept queries of this form, and add an implicit entry to the query's FROM clause for each table referenced by the query. This is no longer allowed.

Omitting the AS Key Word

In the SQL standard, the optional key word AS can be omitted before an output column name whenever the new column name is a valid column name (that is, not the same as

any reserved keyword). PostgreSQL is slightly more restrictive: `AS` is required if the new column name matches any keyword at all, reserved or not. Recommended practice is to use `AS` or double-quote output column names, to prevent any possible conflict against future keyword additions.

In `FROM` items, both the standard and PostgreSQL allow `AS` to be omitted before an alias that is an unreserved keyword. But this is impractical for output column names, because of syntactic ambiguities.

`ONLY` and Inheritance

The SQL standard requires parentheses around the table name when writing `ONLY`, for example `SELECT * FROM ONLY (tab1), ONLY (tab2) WHERE ...`. PostgreSQL considers these parentheses to be optional.

PostgreSQL allows a trailing `*` to be written to explicitly specify the non-`ONLY` behavior of including child tables. The standard does not allow this.

(These points apply equally to all SQL commands supporting the `ONLY` option.)

Namespace Available to `GROUP BY` and `ORDER BY`

In the SQL-92 standard, an `ORDER BY` clause can only use output column names or numbers, while a `GROUP BY` clause can only use expressions based on input column names. PostgreSQL extends each of these clauses to allow the other choice as well (but it uses the standard's interpretation if there is ambiguity). PostgreSQL also allows both clauses to specify arbitrary expressions. Note that names appearing in an expression will always be taken as input-column names, not as output-column names.

SQL:1999 and later use a slightly different definition which is not entirely upward compatible with SQL-92. In most cases, however, PostgreSQL will interpret an `ORDER BY` or `GROUP BY` expression the same way SQL:1999 does.

Functional Dependencies

PostgreSQL recognizes functional dependency (allowing columns to be omitted from `GROUP BY`) only when a table's primary key is included in the `GROUP BY` list. The SQL standard specifies additional conditions that should be recognized.

`WINDOW` Clause Restrictions

The SQL standard provides additional options for the window *`frame_clause`*. PostgreSQL currently supports only the options listed above.

`LIMIT` and `OFFSET`

The clauses `LIMIT` and `OFFSET` are PostgreSQL-specific syntax, also used by MySQL. The SQL:2008 standard has introduced the clauses `OFFSET ... FETCH {FIRST|NEXT} ...` for the same functionality, as shown above in `LIMIT` Clause. This syntax is also used by IBM DB2. (Applications written for Oracle frequently use a workaround involving the automatically generated `rownum` column, which is not available in PostgreSQL, to implement the effects of these clauses.)

`FOR UPDATE` and `FOR SHARE`

Although `FOR UPDATE` appears in the SQL standard, the standard allows it only as an option of `DECLARE CURSOR`. PostgreSQL allows it in any `SELECT` query as well as in

sub-`SELECT`s, but this is an extension. The `FOR SHARE` variant, and the `NOWAIT` option, do not appear in the standard.

Data-Modifying Statements in `WITH`

PostgreSQL allows `INSERT`, `UPDATE`, and `DELETE` to be used as `WITH` queries. This is not found in the SQL standard.

Nonstandard Clauses

The clause `DISTINCT ON` is not defined in the SQL standard.

SELECT INTO

SELECT INTO — 根据查询结果定义一个新表

大纲

```
[ WITH [ RECURSIVE ] with_query [, ...] ]
SELECT [ ALL | DISTINCT [ ON ( expression [, ...] ) ] ]
    * | expression [ [ AS ] output_name ] [, ...]
    INTO [ TEMPORARY | TEMP | UNLOGGED ] [ TABLE ] new_table
    [ FROM from_item [, ...] ]
    [ WHERE condition ]
    [ GROUP BY expression [, ...] ]
    [ HAVING condition [, ...] ]
    [ WINDOW window_name AS ( window_definition ) [, ...] ]
    [ { UNION | INTERSECT | EXCEPT } [ ALL | DISTINCT ] select ]
    [ ORDER BY expression [ ASC | DESC | USING operator ] [ NULLS {
 FIRST | LAST } ] [, ...] ]
    [ LIMIT { count | ALL } ]
    [ OFFSET start [ ROW | ROWS ] ]
    [ FETCH { FIRST | NEXT } [ count ] { ROW | ROWS } ONLY ]
    [ FOR { UPDATE | SHARE } [ OF table_name [, ...] ] [ NOWAIT ] [
...] ]
```

描述

`SELECT INTO`创建一个新表，并用查询计算得到的数据填充该表。与普通的`SELECT`不同，这些数据不会返回给客户端。新表的列具有与`SELECT`输出列对应的名称和数据类型。

参数

`TEMPORARY` 或 `TEMP`

如果指定，该表将创建为临时表。详见CREATE TABLE。

`UNLOGGED`

如果指定，该表将创建为不记录 WAL 的表。详见CREATE TABLE。

`new_table`

要创建的表的名称（可选地带模式限定）。

所有其他参数都在SELECT中有详细说明。

注解

CREATE TABLE AS在功能上与 `SELECT INTO`类似。`CREATE TABLE AS` 是推荐使用的语法，因为这种形式的`SELECT INTO`在ECPG 或PL/pgSQL中不可用，因为它们对 `INTO`子句有不同的解释。此外，`CREATE TABLE AS`提供的功能是 `SELECT INTO`所提供功能的超集。

在 PostgreSQL 8.1 之前，`SELECT INTO` 创建的表默认包含 OID。在 PostgreSQL 8.1 中不再如此 – 要在新表中包含 OID，必须启用default_with_oids配置变量。或者，也可以在 `CREATE TABLE AS` 中使用 `WITH OIDS` 子句。

示例

创建一个新表`films_recent`，它只包含表 `films`中的最近条目：

```
SELECT * INTO films_recent FROM films WHERE date_prod >= '2002-01
-01';
```

兼容性

SQL 标准使用`SELECT INTO`表示将值选入宿主程序的标量变量，而不是创建一个新表。这确实是 ECPG（见第 33 章）和 PL/pgSQL（见第 39 章）中的用法。PostgreSQL使用`SELECT INTO`表示创建表则是历史遗留用法。对于新代码，最好为此目的使用`CREATE TABLE AS`。

另见

CREATE TABLE AS

SET

SET — 更改运行时参数

大纲

```
SET [ SESSION | LOCAL ] configuration_parameter { TO | = } { value
 | 'value' | DEFAULT }
SET [ SESSION | LOCAL ] TIME ZONE { timezone | LOCAL | DEFAULT }
```

描述

SET命令用于更改运行时配置参数。许多在第 18 章中列出的运行时参数都可以通过 SET即时更改。（有些参数需要超级用户权限才能更改，
还有一些参数在服务器启动或会话开始之后便无法再更改。）SET
只影响当前会话所使用的值。

如果在一个随后被中止的事务中发出SET（或者等效的SET SESSION），那么在事务回滚时，SET命令的效果就会消失。一旦该事务提交，这些效果就会持续到会话结束，
除非被另一个SET覆盖。

SET LOCAL的效果只持续到当前事务结束，无论事务是否提交。一种特殊情况是：在同一个事务中先执行 SET，再执行SET LOCAL。这样SET LOCAL设置的值会一直生效到事务结束，但在此之后（如果该事务被提交），SET设置的值将生效。

SET或SET LOCAL的效果，也会因为回滚到早于该命令的保存点而被取消。

如果在一个针对同一变量带有SET选项的函数内使用 SET LOCAL（见CREATE FUNCTION），那么SET LOCAL命令的效果会在函数退出时消失；也就是说，无论如何都会恢复函数被调用时生效的值。这使得SET LOCAL 可以在函数内部用于动态或重复地更改某个参数，同时仍能方便地利用 SET选项保存并恢复调用者的值。不过，普通的 SET命令会覆盖任何外围函数的SET选项；除非回滚，否则它的效果会一直保留。

> **注意**
>
> 在PostgreSQL 8.0 到 8.2 版本中，SET LOCAL的效果会因为释放一个更早的保存点，或者成功退出某个PL/pgSQL异常块而被取消。由于这种行为被认为不够直观，
> 因此后来对其进行了更改。

参数

SESSION

指定该命令对当前会话生效。（如果既没有出现SESSION，也没有出现LOCAL，
这就是默认情况。）

LOCAL

指定该命令只对当前事务生效。在COMMIT或 ROLLBACK之后，会话级设置会再次生效。请注意，如果在BEGIN块之外执行SET LOCAL，它看起来不会有任何效果，因为事务会立即结束。

configuration_parameter

可设置的运行时参数名称。可用参数见第 18 章以及下文。

value

参数的新值。根据具体参数的不同，值可以指定为字符串常量、标识符、数字，或由这些构成的逗号分隔列表。也可以写成DEFAULT，表示把该参数重置为其默认值（即当前会话中如果从未执行过 SET，它本应具有的值）。

除了第 18 章中记载的配置参数外，还有几个参数只能通过SET命令调整，或者具有特殊语法：

SCHEMA

SET SCHEMA '*value*'是 SET search_path TO *value*的别名。使用这种语法时只能指定一个模式。

NAMES

SET NAMES *value*是 SET client_encoding TO *value*的别名。

SEED

设置随机数生成器（即函数 random）的内部种子。允许的值是 -1 到 1 之间的浮点数，这些值随后会乘以 2^{31}-1。

种子也可以通过调用函数setseed来设置：

SELECT setseed(*value*);

TIME ZONE

SET TIME ZONE *value*是以下写法的别名：SET timezone TO *value*。语法SET TIME ZONE允许使用特殊语法指定时区。下面是一些有效值的示例：

'PST8PDT'

加利福尼亚州伯克利所使用的时区。

'Europe/Rome'

意大利所使用的时区。

-7

比 UTC 向西 7 小时的时区（等同于 PDT）。正值表示位于 UTC 以东。

INTERVAL '-08:00' HOUR TO MINUTE

比 UTC 向西 8 小时的时区（等同于 PST）。

LOCAL
DEFAULT

将时区设置为本地时区（即服务器timezone的默认值）。

关于时区的更多信息，见第 8.5.3 节。

注解

函数set_config提供了等效功能；见第 9.26 节。此外，也可以对 pg_settings 系统视图执行 UPDATE，以实现与SET等效的操作。

示例

设置模式搜索路径：

```
SET search_path TO my_schema, public;
```

将日期风格设置为传统POSTGRES风格，并采用“日在月之前”的输入惯例：

```
SET datestyle TO postgres, dmy;
```

设置加利福尼亚州伯克利的时区：

```
SET TIME ZONE 'PST8PDT';
```

将时区设置为意大利所使用的时区：

```
SET TIME ZONE 'Europe/Rome';
```

兼容性

`SET TIME ZONE`扩展了 SQL 标准定义的语法。标准只允许使用数值形式的时区偏移量，而PostgreSQL 允许更灵活的时区指定方式。其他所有`SET`特性都是 PostgreSQL扩展。

另见

RESET, SHOW

SET CONSTRAINTS

SET CONSTRAINTS — 为当前事务设置约束检查时机

大纲

```
SET CONSTRAINTS { ALL | name [, ...] } { DEFERRED | IMMEDIATE }
```

描述

SET CONSTRAINTS设置当前事务中约束检查的行为。IMMEDIATE约束会在每条语句结束时检查。DEFERRED约束要等到事务提交时才检查。每个约束都有各自的IMMEDIATE或DEFERRED模式。

创建约束时，它会被赋予以下三种特性之一：DEFERRABLE INITIALLY DEFERRED、DEFERRABLE INITIALLY IMMEDIATE或者 NOT DEFERRABLE。第三类始终是IMMEDIATE，不会受到 SET CONSTRAINTS命令的影响。前两类在每个事务开始时都处于其指定的模式，但其行为可以在事务中通过 SET CONSTRAINTS修改。

带约束名称列表的SET CONSTRAINTS只会修改这些约束的模式（它们都必须是可延迟的）。每个约束名称都可以带模式限定。如果未指定模式名称，就会使用当前模式搜索路径查找第一个匹配的名称。SET CONSTRAINTS ALL会修改所有可延迟约束的模式。

当SET CONSTRAINTS将某个约束的模式从 DEFERRED改为IMMEDIATE时，新模式具有追溯效力：任何原本会在事务结束时才检查的未决数据修改，都会改为在执行SET CONSTRAINTS命令期间检查。如果违反了任何此类约束，SET CONSTRAINTS就会失败（并且不会改变该约束的模式）。因此，可以利用SET CONSTRAINTS强制在事务中的特定点执行约束检查。

当前，只有UNIQUE、PRIMARY KEY、REFERENCES（外键）以及EXCLUDE 约束受到这个设置的影响。NOT NULL和CHECK约束总是在一行被插入或修改时立即检查（不是在语句结束时）。未声明为DEFERRABLE的唯一约束和排他约束也会立即检查。

被声明为“约束触发器”的触发器，其引发也受此设置控制 – 它们会在相关约束应当被检查的同时引发。

注解

因为PostgreSQL并不要求约束名称在同一模式内唯一（只要求在每个表内唯一），所以指定的约束名称有可能匹配到多个约束。在这种情况下，SET CONSTRAINTS会作用于所有匹配项。对于未带模式限定的名称，一旦在搜索路径中的某个模式里找到一个或多个匹配项，就不会再搜索路径中更靠后的模式。

这个命令只会改变当前事务中约束的行为。因此，如果在事务块（BEGIN/COMMIT对）之外执行该命令，它看起来不会有任何效果。

兼容性

这个命令符合 SQL 标准定义的行为，但有一个限制：在 PostgreSQL中，它不会应用在NOT NULL和CHECK约束上。此外，PostgreSQL会立即检查不可延迟的唯一约束，而不是像标准所暗示的那样在语句结束时检查。

SET ROLE

SET ROLE — 设置当前会话的当前用户标识符

大纲

```
SET [ SESSION | LOCAL ] ROLE role_name
SET [ SESSION | LOCAL ] ROLE NONE
RESET ROLE
```

描述

该命令把当前 SQL 会话的当前用户标识符设置为 *role_name*。角色名可以写成标识符或字符串字面量。在 SET ROLE 之后，SQL 命令的权限检查会视同指定角色是最初登录的角色。

当前会话用户必须是指定的 *role_name* 角色的成员。（如果会话用户是超级用户，则可以选择任意角色。）

SESSION 和 LOCAL 修饰符的作用与常规的SET命令相同。

NONE 和 RESET 形式会把当前用户标识符重置为当前会话用户标识符。这些形式可以由任何用户执行。

注解

使用这个命令，既可以增加权限，也可以限制自己的权限。如果会话用户角色具有 INHERITS 属性，它会自动拥有每个可通过 SET ROLE 切换到的角色的全部权限；在这种情况下，SET ROLE 实际上会去掉直接分配给会话用户及其所属其他角色的全部权限，只保留指定角色可用的权限。另一方面，如果会话用户角色具有 NOINHERITS 属性，SET ROLE 会放弃直接分配给会话用户的权限，转而获得指定角色可用的权限。

特别是，当超级用户选择通过 SET ROLE 切换到非超级用户角色时，会失去其超级用户权限。

SET ROLE的效果与SET SESSION AUTHORIZATION相近，但其中涉及的权限检查完全不同。此外，SET SESSION AUTHORIZATION会决定后续 SET ROLE命令允许使用哪些角色，而通过 SET ROLE更改角色并不会改变后续 SET ROLE允许使用的角色集合。

SET ROLE不会处理角色的ALTER ROLE设置所指定的会话变量；这只会在登录期间发生。

SET ROLE不能在 SECURITY DEFINER函数中使用。

示例

```
SELECT SESSION_USER, CURRENT_USER;

 session_user | current_user
--------------+--------------
 peter        | peter

SET ROLE 'paul';

SELECT SESSION_USER, CURRENT_USER;

 session_user | current_user
--------------+--------------
```

```
 peter         | paul
```

兼容性

PostgreSQL允许使用标识符语法（"rolename"），而 SQL 标准要求将角色名写成字符串字面量。SQL 不允许在事务中执行这个命令；PostgreSQL不施加这一限制，因为没有理由这样做。SESSION和LOCAL修饰符，以及 RESET语法，都是PostgreSQL扩展。

另见

SET SESSION AUTHORIZATION

SET SESSION AUTHORIZATION

SET SESSION AUTHORIZATION — 设置当前会话的会话用户标识符和当前用户标识符

大纲

```
SET [ SESSION | LOCAL ] SESSION AUTHORIZATION user_name
SET [ SESSION | LOCAL ] SESSION AUTHORIZATION DEFAULT
RESET SESSION AUTHORIZATION
```

描述

这个命令将当前 SQL 会话的会话用户标识符和当前用户标识符设置为 *user_name*。用户名可以写成一个标识符或者一个字符串字面量。例如，
可以使用这个命令临时变为一个非特权用户，之后再切换回超级用户。

会话用户标识符初始时被设置为客户端提供的（可能已认证的）用户名。
当前用户标识符通常与会话用户标识符相同，但可能在 `SECURITY DEFINER`函数和类似机制的上下文中临时更改；它也可以通过SET ROLE更改。当前用户标识符用于权限检查。

只有当初始会话用户（即已认证用户）最初具有超级用户权限时，
才能更改会话用户标识符。否则，只有当该命令指定的是已认证用户名时，才会被接受。

`SESSION` 和 `LOCAL` 修饰符的作用与常规的SET命令相同。

`DEFAULT`和`RESET`形式会将会话用户标识符和当前用户标识符重置为最初已认证的用户名。这些形式可以由任何用户执行。

注解

`SET SESSION AUTHORIZATION`不能在一个 `SECURITY DEFINER`函数中使用。

示例

```
SELECT SESSION_USER, CURRENT_USER;

 session_user | current_user
--------------+--------------
 peter        | peter

SET SESSION AUTHORIZATION 'paul';

SELECT SESSION_USER, CURRENT_USER;

 session_user | current_user
--------------+--------------
 paul         | paul
```

兼容性

SQL 标准允许在字面值*user_name*的位置上使用某些其他表达式，但这些选项在实践中并不重要。PostgreSQL允许使用标识符语法（`"username"`），而 SQL 标准不允许。SQL 不允许在事务中使用这个命令；PostgreSQL并不做此限制，因为没有理由这样做。`SESSION`和`LOCAL`修饰符以及`RESET`语法都是PostgreSQL扩展。

标准把执行这个命令所需的权限留给实现定义。

另见

SET ROLE

SET TRANSACTION

SET TRANSACTION — 设置当前事务的特性

大纲

```
SET TRANSACTION transaction_mode [, ...]
SET TRANSACTION SNAPSHOT snapshot_id
SET SESSION CHARACTERISTICS AS TRANSACTION transaction_mode [, ...]
```

其中 *transaction_mode* 是下列之一：

```
    ISOLATION LEVEL { SERIALIZABLE | REPEATABLE READ | READ
 COMMITTED | READ UNCOMMITTED }
    READ WRITE | READ ONLY
    [ NOT ] DEFERRABLE
```

描述

SET TRANSACTION命令设置当前事务的特性。它对任何后续事务都没有影响。SET SESSION CHARACTERISTICS设置一个会话中后续事务的默认事务特性。对于单个事务，这些默认值可以用 SET TRANSACTION覆盖。

可用的事务特性是事务隔离级别、事务访问模式（读/写或只读）以及可延迟模式。此外，还可以选择一个快照，不过它只能用于当前事务，不能作为会话默认值。

一个事务的隔离级别决定当其他事务并发运行时该事务能看见什么数据：

READ COMMITTED

一条语句只能看到在它开始之前已提交的行。这是默认值。

REPEATABLE READ

当前事务中的所有语句都只能看到在该事务中执行第一条查询或数据修改语句之前已提交的行。

SERIALIZABLE

当前事务中的所有语句都只能看到在该事务中执行第一条查询或数据修改语句之前已提交的行。如果并发的可串行化事务之间出现某种读写模式，
而这种情况不可能在这些事务的任何串行（一次执行一个）执行中发生，
那么其中一个事务将以 serialization_failure错误回滚。

SQL 标准还定义了一个额外的级别：READ UNCOMMITTED。在 PostgreSQL中，READ UNCOMMITTED被视为 READ COMMITTED。

在事务执行第一条查询或数据修改语句（SELECT，INSERT，DELETE，UPDATE，FETCH，或 COPY）之后，事务隔离级别就不能再更改。有关事务隔离和并发控制的更多信息，请参见第 13 章。

事务访问模式决定事务是读/写还是只读。读/写是默认值。当事务为只读时，下列 SQL 命令会被禁止：INSERT、UPDATE、DELETE 以及 COPY FROM，前提是它们要写入的表不是临时表；所有 CREATE、ALTER 和 DROP 命令；COMMENT、GRANT、REVOKE、TRUNCATE；以及 EXPLAIN ANALYZE 和 EXECUTE，前提是它们要执行的命令属于上述列表。这是一个较高层面的只读概念，并不会阻止所有写入磁盘的行为。

只有当事务同时是SERIALIZABLE和 READ ONLY时，DEFERRABLE 事务属性才会生效。当为一个事务同时选择这三个属性时，该事务在首次获取其快照时可能会阻塞；在此之后，它便可以运行，而无需承担普通 SERIALIZABLE事务的常规开销，也不会有促成串行化失败或因串行化失败而被取消的风险。这种模式非常适合长时间运行的报表或备份。

SET TRANSACTION SNAPSHOT命令允许一个新事务使用与现有事务相同的快照运行。已有事务必须用 pg_export_snapshot函数导出其快照（参见第 9.26.5 节）。该函数会返回一个快照标识符，必须将其提供给SET TRANSACTION SNAPSHOT以指定要导入的快照。该标识符在此命令中必须写成字符串字面量，例如 '000003A1-1'。SET TRANSACTION SNAPSHOT只能在事务开始时执行，也就是在事务的第一条查询或数据修改语句（SELECT、INSERT、DELETE、UPDATE、FETCH 或 COPY）之前。此外，事务还必须已经设置为 SERIALIZABLE 或 REPEATABLE READ 隔离级别（否则，快照会被立即丢弃，因为 READ COMMITTED 模式会为每条命令获取一个新快照）。如果导入事务使用 SERIALIZABLE 隔离级别，则导出快照的事务也必须使用该隔离级别。此外，非只读的可串行化事务不能从只读事务导入快照。

注解

如果执行SET TRANSACTION之前没有 START TRANSACTION或者 BEGIN，它看起来不会有任何效果，因为事务会立即结束。

可以不使用SET TRANSACTION，而是在 BEGIN或START TRANSACTION中指定所需的 *transaction_modes*。但是，对于SET TRANSACTION SNAPSHOT，这种方式不可用。

会话默认的事务模式也可以通过配置参数default_transaction_isolation、default_transaction_read_only和default_transaction_deferrable来设置（实际上，SET SESSION CHARACTERISTICS只是用 SET设置这些变量的一种更冗长的等价写法）。这意味着可以通过配置文件、ALTER DATABASE等方式设置默认值。详见第 18 章。

示例

要以与某个现有事务相同的快照开始一个新事务，先从该现有事务导出快照。这样会返回快照标识符，例如：

```
BEGIN TRANSACTION ISOLATION LEVEL REPEATABLE READ;
SELECT pg_export_snapshot();
 pg_export_snapshot
--------------------
 000003A1-1
(1 row)
```

然后在新开启事务的开始处，通过SET TRANSACTION SNAPSHOT命令给出该快照标识符：

```
BEGIN TRANSACTION ISOLATION LEVEL REPEATABLE READ;
SET TRANSACTION SNAPSHOT '000003A1-1';
```

兼容性

这些命令由SQL标准定义，但 DEFERRABLE事务模式和 SET TRANSACTION SNAPSHOT这种形式除外，它们是 PostgreSQL扩展。

SERIALIZABLE是标准中的默认事务隔离级别。在 PostgreSQL中，默认值通常是 READ COMMITTED，但你可以按上述方式修改。

在 SQL 标准中，还可以用这些命令设置另一项事务特性：诊断区域的大小。这个概念特定于嵌入式 SQL，因此没有在 PostgreSQL服务器中实现。

SQL 标准要求在连续的*transaction_modes*之间有逗号，但出于历史原因，PostgreSQL允许省略逗号。

SHOW

SHOW — 显示一个运行时参数的值

大纲

```
SHOW name
SHOW ALL
```

描述

SHOW将显示运行时参数的当前设置。这些变量可以使用SET语句来设置，也可以通过编辑postgresql.conf配置文件、使用 PGOPTIONS环境变量（在使用 libpq或基于libpq的应用时），或者在启动postgres服务器时通过命令行标志来设置。详见第 18 章。

参数

name

运行时参数的名称。可用参数的说明见第 18 章以及SET参考页。此外，还有几个可以显示但不能设置的参数：

SERVER_VERSION

显示服务器的版本号。

SERVER_ENCODING

显示服务器端字符集编码。目前，该参数可以显示但不能设置，因为编码在数据库创建时就已确定。

LC_COLLATE

显示数据库用于排序规则（文本排序）的区域设置。目前，该参数可以显示但不能设置，因为该设置在数据库创建时就已确定。

LC_CTYPE

显示数据库用于字符分类的区域设置。目前，该参数可以显示但不能设置，因为该设置在数据库创建时就已确定。

IS_SUPERUSER

如果当前角色具有超级用户权限，则为真。

ALL

显示所有配置参数的值及其描述。

注解

函数current_setting也会产生等效的输出，见第 9.26 节。另外，pg_settings 系统视图也会给出相同的信息。

示例

显示参数DateStyle的当前设置：

```
SHOW DateStyle;
 DateStyle
-----------
 ISO, MDY
(1 row)
```

显示参数geqo的当前设置：

```
SHOW geqo;
 geqo
------
 on
(1 row)
```

显示所有设置：

```
SHOW ALL;
            name          | setting |                 description
--------------------------+---------+-------------------------------
------------------
 allow_system_table_mods | off     | Allows modifications of the
 structure of ...
    .
    .
    .
 xmloption               | content | Sets whether XML data in
 implicit parsing ...
 zero_damaged_pages      | off     | Continues processing past
 damaged page headers.
(196 rows)
```

兼容性

SHOW命令是一种 PostgreSQL扩展。

另见

SET, RESET

START TRANSACTION

START TRANSACTION — 开始一个事务块

大纲

```
START TRANSACTION [ transaction_mode [, ...] ]
```

其中 *transaction_mode* 是下列之一：

```
    ISOLATION LEVEL { SERIALIZABLE | REPEATABLE READ | READ
 COMMITTED | READ UNCOMMITTED }
    READ WRITE | READ ONLY
    [ NOT ] DEFERRABLE
```

描述

该命令开始一个新的事务块。如果指定了隔离级别、读写模式或可延迟模式，
新事务就会具有这些特性，就像执行了SET TRANSACTION一样。它与BEGIN命令具有相同功能。

参数

关于本语句参数含义的信息，参见SET TRANSACTION。

兼容性

在 SQL 标准中，并不需要发出START TRANSACTION 来开始一个事务块：
任何 SQL 命令都会隐式开始一个事务块。PostgreSQL的行为可以看作是：对每条不跟在START TRANSACTION（或BEGIN）之后的命令，都会在其后隐式发出一个COMMIT，因此这种行为通常被称为“自动提交”。
其他关系型数据库系统也可能为了方便而提供自动提交特性。

DEFERRABLE *transaction_mode* 是PostgreSQL的一种语言扩展。

SQL 标准要求在连续的*transaction_modes*之间有逗号，但出于历史原因，PostgreSQL允许省略逗号。

另见SET TRANSACTION的兼容性小节。

另见

BEGIN, COMMIT, ROLLBACK, SAVEPOINT, SET TRANSACTION

TRUNCATE

TRUNCATE — 清空一个表或一组表

大纲

```
TRUNCATE [ TABLE ] [ ONLY ] name [ * ] [, ... ]
    [ RESTART IDENTITY | CONTINUE IDENTITY ] [ CASCADE | RESTRICT ]
```

描述

TRUNCATE可以快速移除一组表中的所有行。它的效果与对每个表执行不带条件的DELETE相同，但由于它并不实际扫描这些表，因此速度更快。此外，它会立即回收磁盘空间，而不是要求后续再执行一次VACUUM操作。这一点对于大表尤其有用。

参数

name

要截断的表名（可以是模式限定的）。如果在表名前指定了 ONLY，则只截断该表。如果未指定ONLY，则该表及其所有后代表（如果有）都会被截断。也可以在表名后指定 *，以显式表明包含后代表。

RESTART IDENTITY

自动重新启动由被截断表列拥有的 sequence。

CONTINUE IDENTITY

不更改 sequence 的值。这是默认值。

CASCADE

自动截断所有以外键引用任一已命名表的表，以及任何由于 CASCADE而被加入该组的表。

RESTRICT

如果任一表具有来自命令中未列出表的外键引用，则拒绝截断。这是默认值。

注解

要截断一个表，你必须拥有该表上的TRUNCATE权限。

TRUNCATE会在其操作的每个表上获取 ACCESS EXCLUSIVE锁，这会阻塞该表上的所有其他并发操作。当指定RESTART IDENTITY时，任何需要重新启动的 sequence 也会同样被排他锁定。如果需要对某个表进行并发访问，则应改用 DELETE命令。

TRUNCATE不能用于被其他表通过外键引用的表，除非所有这类表也在同一条命令中被截断。因为在这种情况下检查有效性将需要扫描表，而该命令的意义恰恰在于避免这样做。CASCADE 选项可用于自动包含所有依赖表 – 但使用该选项时一定要非常小心，否则你可能会丢失并非有意删除的数据！

TRUNCATE不会引发这些表上可能存在的任何 ON DELETE触发器，但会引发 ON TRUNCATE触发器。如果这些表中的任何一个定义了ON TRUNCATE触发器，那么所有 BEFORE TRUNCATE触发器都会在任何截断发生之前引发，而所有AFTER TRUNCATE触发器都会在最

后一次截断完成且所有 sequence 被重置之后引发。触发器会按照表的处理顺序引发（先是命令中列出的表，然后是由于级联而加入的表）。

`TRUNCATE`不是多版本并发控制（MVCC）安全的。截断之后，
如果并发事务使用的是在截断发生前取得的快照，该表对这些并发事务而言将表现为空。
详见第 13.5 节。

就表中数据而言，`TRUNCATE`是事务安全的：如果外围事务没有提交，截断将被安全地回滚。

当指定`RESTART IDENTITY`时，隐含的 `ALTER SEQUENCE RESTART`操作也会以事务方式执行；也就是说，如果外围事务没有提交，它们也会被回滚。这与 `ALTER SEQUENCE RESTART`通常的行为不同。请注意，如果在事务回滚前又对这些重启后的 sequence 执行了额外操作，这些操作对 sequence 本身的影响会被回滚，但对`currval()`的影响不会被回滚。也就是说，事务结束后，`currval()`仍将反映失败事务内取得的最后一个 sequence 值，即使 sequence 本身可能已经不再与之保持一致。这与失败事务之后`currval()`的通常行为类似。

示例

截断表`bigtable`和 `fattable`：

```
TRUNCATE bigtable, fattable;
```

同样的操作，并重置所有相关联的 sequence 生成器：

```
TRUNCATE bigtable, fattable RESTART IDENTITY;
```

截断表`othertable`，并级联到任何通过外键约束引用`othertable`的表：

```
TRUNCATE othertable CASCADE;
```

兼容性

SQL:2008 标准包括了一个`TRUNCATE`命令，语法是`TRUNCATE TABLE` *`tablename`*。子句 `CONTINUE IDENTITY/RESTART IDENTITY` 在该标准中也有出现，但其含义虽相关却略有不同。该命令的某些并发行为在标准中被留作实现定义，因此必要时应结合上述注解并与其他实现进行比较。

UNLISTEN

UNLISTEN — 停止监听通知

大纲

```
UNLISTEN { channel | * }
```

描述

UNLISTEN用于移除现有的 NOTIFY事件注册。UNLISTEN会取消当前 PostgreSQL会话作为名为*channel*的通知通道监听者的任何现有注册。特殊通配符*
会取消当前会话的所有监听注册。

NOTIFY中对LISTEN和 NOTIFY的使用有更详细的讨论。

参数

channel

通知通道的名称（任意标识符）。

*

该会话当前的所有监听注册都会被清除。

注解

可以取消监听自己并未监听的通道，不会出现警告或错误。

每个会话结束时，都会自动执行UNLISTEN *。

执行过UNLISTEN的事务不能为两阶段提交做准备。

示例

建立一个监听注册：

```
LISTEN virtual;
NOTIFY virtual;
Asynchronous notification "virtual" received from server process
 with PID 8448.
```

执行UNLISTEN之后，后续的NOTIFY 消息将被忽略：

```
UNLISTEN virtual;
NOTIFY virtual;
-- no NOTIFY event is received
```

兼容性

SQL 标准中没有UNLISTEN语句。

另见

LISTEN, NOTIFY

UPDATE

UPDATE — 更新表中的行

大纲

```
[ WITH [ RECURSIVE ] with_query [, ...] ]
UPDATE [ ONLY ] table_name [ * ] [ [ AS ] alias ]
    SET { column_name = { expression | DEFAULT } |
          ( column_name [, ...] ) = ( { expression | DEFAULT } [, .
..] ) } [, ...]
    [ FROM from_list ]
    [ WHERE condition | WHERE CURRENT OF cursor_name ]
    [ RETURNING * | output_expression [ [ AS ] output_name ] [, ...
] ]
```

描述

UPDATE会更改所有满足条件的行中指定列的值。只需在SET子句中提及需要修改的列；未被显式修改的列会保留其原有值。

有两种方法可以利用数据库中其他表所包含的信息来修改一个表：使用子选择，或者在FROM子句中指定附加表。哪种技术更合适取决于具体情况。

可选的RETURNING子句使UPDATE 基于每个实际更新的行计算并返回一个或多个值。可以计算任何使用该表列和/或 FROM中提到的其他表列的表达式。默认使用该表列的新值（更新后值），但也可以请求旧值（更新前值）。RETURNING列表的语法与SELECT 的输出列表相同。

必须拥有该表上的UPDATE权限，或者至少拥有所列待更新列的该权限。还必须拥有SELECT权限，覆盖在*expressions*或*condition*中读取其值的所有列。

参数

with_query

WITH子句允许你指定一个或多个子查询，这些子查询可在UPDATE查询中按名称引用。详见第 7.8 节和SELECT。

table_name

要更新的表名（可以是模式限定的）。如果在表名前指定了 ONLY，只会更新所提及表中的匹配行。如果未指定 ONLY，还会更新继承自该表的任何表中的匹配行。可选地，可以在表名后指定*，以显式指示包含后代表。

alias

目标表的替代名称。提供别名时，它会完全隐藏该表的实际名称。例如，给定UPDATE foo AS f，UPDATE语句的其余部分必须将该表称为 f，而不是foo。

column_name

由*table_name*命名的表中的列名。如有需要，列名可以使用子字段名或数组下标进行限定。指定目标列时不要包含表名 – 例如，UPDATE tab SET tab.col = 1是无效的。

expression

要赋给该列的表达式。该表达式可以使用该表中这一列和其他列的旧值。

DEFAULT

把该列设置为其默认值（如果没有为它指定特定的默认表达式，则该值为 NULL）。

from_list

一个表表达式列表，允许其他表的列出现在WHERE条件以及更新表达式中。这类似于SELECT语句的FROM Clause中可以指定的表列表。注意，除非打算进行自连接（此时它必须在*from_list*中带有别名出现），否则目标表不能出现在*from_list*中。

condition

一个返回boolean类型值的表达式。只有使这个表达式返回true的行才会被更新。

cursor_name

要在WHERE CURRENT OF条件中使用的游标名。要被更新的行是最近一次从该游标中取出的那一行。该游标必须是针对UPDATE目标表的非分组查询。注意，WHERE CURRENT OF不能与布尔条件同时指定。参见DECLARE，了解将游标用于WHERE CURRENT OF的更多信息。

output_expression

在每一行被更新后，由UPDATE命令计算并返回的表达式。该表达式可以使用由*table_name* 命名的表或FROM中列出的表中的任何列名。写成 *可返回所有列。

output_name

用于返回列的名称。

输出

在成功完成时，一个UPDATE命令会返回以下形式的命令标签：

UPDATE *count*

*count*是被更新的行数，包括值没有改变的匹配行。注意，当更新被BEFORE UPDATE 触发器抑制时，这个数量可能小于匹配 *condition*的行数。如果 *count*为 0，则该查询没有更新任何行（这不被视为错误）。

如果UPDATE命令包含RETURNING 子句，其结果将类似于一个SELECT语句，其中包含RETURNING列表中定义的列和值，并在该命令更新的行上进行计算。

注解

当存在FROM子句时，本质上会将目标表与 *from_list*中提到的表连接起来，而连接的每一条输出行都代表对目标表的一次更新操作。使用 FROM时，应确保对每一个要修改的目标行，连接至多生成一条输出行。换言之，一条目标行不应与其他表中的多于一行成功连接。
如果发生这种情况，则只会使用其中某一条连接行来更新目标行，
但具体使用哪一条并不容易预测。

因为存在这种不确定性，所以仅在子选择中引用其他表会更安全，
尽管这种写法通常比使用连接更难阅读、也更慢。

示例

把表films的kind 列中的单词Drama改为Dramatic：

```
UPDATE films SET kind = 'Dramatic' WHERE kind = 'Drama';
```

在表weather的一行中调整温度项，并将降水量重置为其默认值：

```
UPDATE weather SET temp_lo = temp_lo+1, temp_hi = temp_lo+15, prcp
 = DEFAULT
  WHERE city = 'San Francisco' AND date = '2003-07-03';
```

执行同样的操作，并返回更新后的各项：

```
UPDATE weather SET temp_lo = temp_lo+1, temp_hi = temp_lo+15, prcp
 = DEFAULT
  WHERE city = 'San Francisco' AND date = '2003-07-03'
  RETURNING temp_lo, temp_hi, prcp;
```

使用另一种列列表语法完成同样的更新：

```
UPDATE weather SET (temp_lo, temp_hi, prcp) = (temp_lo+1, temp_lo
+15, DEFAULT)
  WHERE city = 'San Francisco' AND date = '2003-07-03';
```

将负责 Acme Corporation 账户的销售人员的销量计数加一，使用 FROM 子句语法：

```
UPDATE employees SET sales_count = sales_count + 1 FROM accounts
  WHERE accounts.name = 'Acme Corporation'
  AND employees.id = accounts.sales_person;
```

执行同样的操作，在 WHERE 子句中使用子选择：

```
UPDATE employees SET sales_count = sales_count + 1 WHERE id =
  (SELECT sales_person FROM accounts WHERE name = 'Acme
 Corporation');
```

尝试插入一个新库存项及其库存量。如果该项已存在，则改为更新现有项的库存量。
要在不致使整个事务失败的情况下做到这一点，请使用保存点：

```
BEGIN;
-- 其他操作
SAVEPOINT sp1;
INSERT INTO wines VALUES('Chateau Lafite 2003', '24');
-- 假设上面的语句因唯一键冲突而失败，
-- 那么现在发出这些命令：
ROLLBACK TO sp1;
UPDATE wines SET stock = stock + 24 WHERE winename = 'Chateau
 Lafite 2003';
-- 继续执行其他操作，最后
COMMIT;
```

更改 kind 列，该列位于表 films 中，所更改的行是游标 c_films 当前定位的行：

```
UPDATE films SET kind = 'Dramatic' WHERE CURRENT OF c_films;
```

兼容性

这个命令符合SQL标准，不过 FROM和RETURNING子句是 PostgreSQL扩展，把WITH 与UPDATE一起使用的能力也是扩展。

某些其他数据库系统提供一种FROM选项，要求在 FROM中再次列出目标表。PostgreSQL 并不是这样解释FROM的。在移植使用这种扩展的应用时要小心。

根据标准，列列表语法应当允许从一个单一行值表达式（例如子查询）给一组列赋值：

```
UPDATE accounts SET (contact_last_name, contact_first_name) =
    (SELECT last_name, first_name FROM salesmen
     WHERE salesmen.id = accounts.sales_id);
```

这一点目前尚未实现--源必须是一个独立表达式列表。

VACUUM

VACUUM — 垃圾收集并按需分析数据库

大纲

```
VACUUM [ ( { FULL | FREEZE | VERBOSE | ANALYZE } [, ...] ) ] [
 table_name [ (column_name [, ...] ) ] ]
VACUUM [ FULL ] [ FREEZE ] [ VERBOSE ] [ table_name ]
VACUUM [ FULL ] [ FREEZE ] [ VERBOSE ] ANALYZE [ table_name [ (
column_name [, ...] ) ] ]
```

描述

VACUUM回收死元组占用的存储空间。在正常的 PostgreSQL运行中，被删除或因更新而过时的元组并不会从其表中物理移除；它们会一直保留，直到执行 VACUUM。因此有必要定期执行VACUUM，尤其是对频繁更新的表。

不带参数时，VACUUM会处理当前数据库中当前用户有权清理的每个表。带参数时，VACUUM则只处理该表。

VACUUM ANALYZE会对每个选定表先执行 VACUUM，再执行ANALYZE。这种便捷的组合形式很适合例行维护脚本使用。关于其处理细节，参见ANALYZE。

普通的VACUUM（不带FULL）只是回收空间并使其可被重用。
这种形式的命令可以与表的正常读写并行运行，因为它不会获得独占锁。不过，
在大多数情况下，额外空间不会返还给操作系统；它只是保留在同一张表内供再次使用。
VACUUM FULL会把表的全部内容重写到一个没有额外空闲空间的新磁盘文件中，从而让未使用的空间能够返还给操作系统。这种形式要慢得多，并且在处理每个表时都需要独占锁。

当选项列表用圆括号括起来时，选项可以按任意顺序书写。如果不加圆括号，
则必须严格按照上面所示的顺序指定选项。带圆括号的语法是在PostgreSQL 9.0 中加入的；不带圆括号的语法已被弃用。

参数

FULL

选择“完全”清理，它可以回收更多空间，但耗时更长，并且会独占锁定该表。
这种方法还需要额外的磁盘空间，因为它会写出该表的一个新副本，
并且在操作完成之前不会释放旧副本。通常，
只有当需要从表内回收大量空间时才应使用这种方法。

FREEZE

选择激进的元组“冻结”。指定FREEZE 等价于执行一个将vacuum_freeze_min_age参数设为零的 VACUUM。

VERBOSE

为每个表输出详细的清理活动报告。

ANALYZE

更新规划器用来确定查询最高效执行方式的统计信息。

table_name

要清理的特定表的名称（可选地带模式限定）。默认为当前数据库中的所有表。

column_name

要分析的特定列名。默认会分析所有列。如果指定了列列表，则隐含指定ANALYZE。

输出

指定VERBOSE时，VACUUM会输出进度消息，指示当前正在处理哪个表，
同时还会打印这些表的各种统计信息。

注解

要清理一个表，通常调用者必须是该表的拥有者或超级用户。不过，
数据库拥有者可以清理其数据库中的所有表，但共享系统目录除外。
（对共享系统目录的这一限制意味着，真正意义上的全数据库 VACUUM 只能由超级用户执行。）VACUUM 会跳过调用用户无权清理的任何表。

VACUUM不能在一个事务块内被执行。

对于带有GIN索引的表，VACUUM（任何形式）还会通过把待处理的索引项移动到主 GIN索引结构中的适当位置，来完成所有挂起的索引插入。详见第 55.3.1 节。

我们建议对活跃的生产数据库频繁执行清理（至少每晚一次），以移除死行。
在添加或删除大量行之后，对受影响的表执行VACUUM ANALYZE命令可能是个好主意。这会用所有近期变更的结果更新系统目录，使
PostgreSQL查询规划器在规划查询时能够做出更好的选择。

FULL选项不建议在日常场景中使用，但在某些特殊情况下可能很有用。例如，
当你删除或更新了表中的绝大多数行，并希望该表在物理上收缩以占用更少磁盘空间、
让表扫描更快时，这个选项就比较合适。VACUUM FULL通常会比普通 VACUUM更大幅度地收缩表。

VACUUM会显著增加 I/O 流量，这可能导致其他活动会话性能变差。因此，有时建议使用基于代价的清理延迟特性。详见第 18.4.4 节。

PostgreSQL提供了一个“自动清理（autovacuum）”机制，可以自动执行常规清理维护。
有关自动与手动清理的更多信息，参见第 23.1 节。

示例

清理单个表onek，对其进行分析以供优化器使用，并打印详细的清理活动报告：

```
VACUUM (VERBOSE, ANALYZE) onek;
```

兼容性

在SQL标准中没有VACUUM语句。

另见

vacuumdb, 第 18.4.4 节, 第 23.1.6 节

VALUES

VALUES — 计算一组行

大纲

```
VALUES ( expression [, ...] ) [, ...]
    [ ORDER BY sort_expression [ ASC | DESC | USING operator ] [, .
..] ]
    [ LIMIT { count | ALL } ]
    [ OFFSET start [ ROW | ROWS ] ]
    [ FETCH { FIRST | NEXT } [ count ] { ROW | ROWS } ONLY ]
```

描述

VALUES计算由值表达式指定的一个行值或一组行值。
它最常用于在更大的命令中生成一个“常量表”，但也可以单独使用。

当指定多于一行时，所有行都必须具有相同数量的元素。
结果表各列的数据类型由出现在该列中的表达式的显式类型或推断类型组合决定，
所用规则与 UNION相同（见第 10.5 节）。

在更大的命令中，从语法上说，凡是允许出现SELECT的地方，也都允许出现VALUES。
因为在语法中它被当作 SELECT处理，所以可以对VALUES 命令使用ORDER BY、LIMIT（或者等效的FETCH FIRST）以及OFFSET子句。

参数

expression

要在结果表（行集合）中相应位置计算并插入的常量或表达式。在出现在INSERT顶层的VALUES列表中，*expression*可以用 DEFAULT替代，以表示应插入目标列的默认值。当VALUES出现在其他上下文中时，不能使用 DEFAULT。

sort_expression

一个指示结果行如何排序的表达式或整数常量。该表达式可以将VALUES结果的列引用为column1、column2等。详细信息参见ORDER BY Clause。

operator

一个排序操作符。详细信息参见ORDER BY Clause。

count

返回的最大行数。详细信息参见LIMIT Clause。

start

开始返回行之前要跳过的行数。详细信息参见LIMIT Clause。

注解

应避免使用行数非常多的VALUES列表，因为这可能导致内存不足错误或性能不佳。出现在INSERT中的 VALUES属于一种特殊情况（因为所需的列类型可以从INSERT的目标表得知，

并且无须通过扫描 VALUES列表来推断），因此它能处理比其他上下文中实际可行的更大的列表。

示例

一个独立的VALUES命令：

```
VALUES (1, 'one'), (2, 'two'), (3, 'three');
```

这将返回一个两列三行的表。它实际上等效于：

```
SELECT 1 AS column1, 'one' AS column2
UNION ALL
SELECT 2, 'two'
UNION ALL
SELECT 3, 'three';
```

更常见的是，VALUES会用在更大的 SQL 命令中。最常见的用法是在INSERT中：

```
INSERT INTO films (code, title, did, date_prod, kind)
    VALUES ('T_601', 'Yojimbo', 106, '1961-06-16', 'Drama');
```

在INSERT的上下文中，VALUES列表中的项可以写成DEFAULT，
表示此处应使用该列的默认值，而不是显式指定一个值：

```
INSERT INTO films VALUES
    ('UA502', 'Bananas', 105, DEFAULT, 'Comedy', '82 minutes'),
    ('T_601', 'Yojimbo', 106, DEFAULT, 'Drama', DEFAULT);
```

VALUES也可以用在原本可以编写子SELECT的地方，例如在FROM子句中：

```
SELECT f.*
  FROM films f, (VALUES('MGM', 'Horror'), ('UA', 'Sci-Fi')) AS t (
studio, kind)
  WHERE f.studio = t.studio AND f.kind = t.kind;

UPDATE employees SET salary = salary * v.increase
  FROM (VALUES(1, 200000, 1.2), (2, 400000, 1.4)) AS v (depno,
 target, increase)
  WHERE employees.depno = v.depno AND employees.sales >= v.target;
```

注意，在FROM子句中使用VALUES时，需要有一个AS子句，这一点与SELECT相同。虽然AS子句不要求为所有列都指定名称，但这样做是一个好习惯。（在PostgreSQL中，VALUES的默认列名是 column1、column2等，但在其他数据库系统中，这些名称可能不同。）

当在INSERT中使用VALUES时，所有值都会自动转换为相应目标列的数据类型。
当它用于其他上下文时，可能有必要指定正确的数据类型。
如果各项都是带引号的字符串常量，只需转换第一项，
就足以为所有项确定假定的数据类型：

```
SELECT * FROM machines
WHERE ip_address IN (VALUES('192.168.0.1'::inet), ('192.168.0.10'),
 ('192.168.1.43'));
```

> **提示**
>
> 对于简单的IN测试，最好使用IN的标量列表形式，而不是写一个像上面那样的VALUES查询。标量列表方法写法更简洁，而且通常效率更高。

兼容性

`VALUES`符合 SQL 标准。`LIMIT`和`OFFSET`是 PostgreSQL扩展，另见SELECT。

另见

INSERT, SELECT

PostgreSQL Client Applications

This part contains reference information for PostgreSQL client applications and utilities. Not all of these commands are of general utility; some might require special privileges. The common feature of these applications is that they can be run on any host, independent of where the database server resides.

When specified on the command line, user and database names have their case preserved – the presence of spaces or special characters might require quoting. Table names and other identifiers do not have their case preserved, except where documented, and might require quoting.

目录

clusterdb 1350
createdb 1353
createlang 1356
createuser 1359
dropdb 1363
droplang 1366
dropuser 1369
ecpg 1371
pg_basebackup 1373
pg_config 1378
pg_dump 1381
pg_dumpall 1392
pg_receivexlog 1397
pg_restore 1400
psql 1408
reindexdb 1441
vacuumdb 1444

clusterdb

clusterdb — 聚簇一个PostgreSQL数据库

大纲

`clusterdb` [*connection-option*...] [`--verbose`|`-v`] [`--table`|`-t` *table*] [*dbname*]

`clusterdb` [*connection-option*...] [`--verbose`|`-v`] `--all`|`-a`

描述

clusterdb是一个用于重新聚簇PostgreSQL 数据库中各表的工具。它会找出此前已经聚簇过的表，并按照上次使用的同一索引再次对其进行聚簇。从未聚簇过的表不会受到影响。

clusterdb是 SQL 命令CLUSTER的一个包装器。使用该工具聚簇数据库，与使用其他访问服务器的方法聚簇数据库，并没有实质区别。

选项

clusterdb接受以下命令行参数：

`-a`
`--all`

聚簇所有数据库。

`[-d]` *dbname*
`[--dbname=]`*dbname*

指定要聚簇的数据库名。如果未指定并且没有使用 `-a`（或`--all`），则从环境变量 `PGDATABASE` 读取数据库名。如果该环境变量也未设置，则使用为该连接指定的用户名作为数据库名。

`-e`
`--echo`

回显clusterdb生成并发送到服务器的命令。

`-t` *table*
`--table=`*table*

仅对*table*执行聚簇。

`-v`
`--verbose`

在处理过程中打印详细信息。

`-q`
`--quiet`

不显示进度消息。

`-V`
`--version`

打印clusterdb的版本并退出。

`-?`
`--help`

显示有关clusterdb命令行参数的帮助信息，并退出。

clusterdb还接受以下用于连接参数的命令行参数：

`-h` *`host`*
`--host=`*`host`*

指定运行服务器的机器的主机名。如果该值以斜杠开头，则它将被用作 Unix 域套接字的目录。

`-p` *`port`*
`--port=`*`port`*

指定服务器监听连接的 TCP 端口，或本地 Unix 域套接字文件扩展名。

`-U` *`username`*
`--username=`*`username`*

用于连接的用户名。

`-w`
`--no-password`

绝不发出密码提示。如果服务器要求密码认证，而又无法通过 `.pgpass` 文件等其他方式获得密码，则连接尝试将失败。这个选项在批处理作业和脚本中很有用，
因为这些场景下没有用户可以输入密码。

`-W`
`--password`

强制clusterdb在连接到数据库之前提示输入密码。

这个选项并非必不可少，因为如果服务器要求密码认证，
clusterdb会自动提示输入密码。不过，clusterdb会浪费一次连接尝试，
才知道服务器需要密码。在某些情况下，为了避免这次额外的连接尝试，提前指定
`-W`是值得的。

`--maintenance-db=`*`dbname`*

连接到该数据库，以发现还有哪些其他数据库需要聚簇。如果未指定，将使用`postgres`数据库；如果它不存在，则使用`template1`。

环境

`PGDATABASE`
`PGHOST`
`PGPORT`
`PGUSER`

默认连接参数

该工具和大多数其他PostgreSQL工具一样，也使用 libpq支持的环境变量（见第 31.14 节）。

诊断

如果遇到问题，请参见CLUSTER和psql中关于潜在问题和错误消息的讨论。
数据库服务器必须在目标主机上运行。此外，
libpq前端库所使用的任何默认连接设置和环境变量都会生效。

示例

要聚簇数据库`test`：

```
$ clusterdb test
```

要聚簇名为`xyzzy`的数据库中的单个表`foo`：

```
$ clusterdb --table foo xyzzy
```

参见

CLUSTER

createdb

createdb — 创建新的PostgreSQL数据库

大纲

`createdb` [*connection-option*...] [*option*...] [*dbname* [*description*]]

描述

createdb创建一个新的PostgreSQL数据库。

通常，执行此命令的数据库用户会成为新数据库的所有者。不过，
如果执行用户具有相应的权限，也可以通过`-O`选项指定不同的所有者。

createdb是SQL命令CREATE DATABASE的包装器。通过此工具创建数据库与通过其他访问服务器的方法创建数据库，在效果上没有区别。

选项

createdb接受以下命令行参数：

dbname

指定要创建的数据库名称。
该名称必须在此数据库集簇中的所有PostgreSQL数据库中唯一。默认情况下，
会创建一个与当前系统用户同名的数据库。

description

指定要与新创建的数据库关联的注释。

`-D` *tablespace*
`--tablespace=`*tablespace*

指定数据库的默认表空间。（该名称将按双引号标识符处理。）

`-e`
`--echo`

回显createdb生成并发送到服务器的命令。

`-E` *encoding*
`--encoding=`*encoding*

指定该数据库使用的字符集编码。PostgreSQL服务器支持的字符集见第 22.3.1 节。

`-l` *locale*
`--locale=`*locale*

指定此数据库要使用的区域设置。这相当于同时指定`--lc-collate`和`--lc-ctype`。

`--lc-collate=`*locale*

指定该数据库要使用的 LC_COLLATE 设置。

`--lc-ctype=`*locale*

指定该数据库要使用的 LC_CTYPE 设置。

`-O` *owner*
`--owner=`*owner*

指定将成为新数据库拥有者的数据库用户。（该名称按双引号标识符处理。）

`-T` *template*
`--template=`*template*

指定用于构建该数据库的模板数据库。（该名称按双引号标识符处理。）

`-V`
`--version`

打印createdb的版本并退出。

`-?`
`--help`

显示有关createdb命令行参数的帮助信息，并退出。

`-D`、`-l`、`-E`、`-O`和`-T`选项对应底层 SQL 命令CREATE DATABASE的选项；有关这些选项的更多信息请参见该命令。

createdb还接受以下用于连接参数的命令行参数：

`-h` *host*
`--host=`*host*

指定服务器所在机器的主机名。如果该值以斜杠开头，则它将被用作 Unix 域套接字的目录。

`-p` *port*
`--port=`*port*

指定服务器监听连接所用的 TCP 端口或本地 Unix 域套接字文件扩展名。

`-U` *username*
`--username=`*username*

以该用户名进行连接。

`-w`
`--no-password`

从不发出密码提示。如果服务器要求密码认证，而密码又无法通过诸如`.pgpass`文件等其他方式获得，则连接尝试将失败。
该选项适合没有用户在场输入密码的批处理作业和脚本。

`-W`
`--password`

强制createdb在连接到数据库之前提示输入密码。

这个选项并非必不可少，因为如果服务器要求密码认证，
createdb会自动提示输入密码。不过，createdb会浪费一次连接尝试，
才知道服务器需要密码。在某些情况下，为了避免这次额外的连接尝试，提前指定
`-W`是值得的。

`--maintenance-db=`*dbname*

指定在创建新数据库时要连接到的数据库名称。如果未指定，将使用`postgres`数据库；若该数据库不存在（或者它正是要创建的新数据库），则改用`template1`。

环境

PGDATABASE

如果设置了该变量，其值就是要创建的数据库名称，除非被命令行覆盖。

PGHOST
PGPORT
PGUSER

默认连接参数。如果命令行和PGDATABASE都没有指定要创建的数据库名称，则PGUSER也会决定该名称。

与大多数其他PostgreSQL工具一样，该工具也使用libpq支持的环境变量（见第 31.14 节）。

诊断

如果遇到问题，请参阅CREATE DATABASE和psql中关于潜在问题和错误消息的讨论。数据库服务器必须在目标主机上运行。此外，由libpq前端库使用的任何默认连接设置和环境变量也都会生效。

示例

要使用默认数据库服务器创建demo数据库：

```
$ createdb demo
```

要使用主机eden上、端口为 5000 的服务器，并以 LATIN1编码方案创建demo数据库，可以看看下面的底层命令：

```
$ createdb -p 5000 -h eden -E LATIN1 -e demo
CREATE DATABASE demo ENCODING 'LATIN1';
```

参见

dropdb, CREATE DATABASE

createlang

createlang — 安装一个 PostgreSQL 过程语言

大纲

`createlang` [*connection-option*...] *langname* [*dbname*]

`createlang` [*connection-option*...] `--list` | `-l` [*dbname*]

描述

createlang 是一个向 PostgreSQL 数据库中添加过程语言的工具。

createlang 只是对CREATE EXTENSION SQL 命令的一个包装。

> **小心**
>
> createlang 已被弃用，在未来的 PostgreSQL 版本中可能会被移除。建议直接使用 `CREATE EXTENSION` 命令。

选项

createlang 接受下列命令行参数：

langname

指定要安装的过程语言的名称。（该名称会被转换为小写。）

[-d] *dbname*
[--dbname=]*dbname*

指定要向哪个数据库添加该语言。默认使用与当前系统用户同名的数据库。

-e
--echo

在执行 SQL 命令时把它们显示出来。

-l
--list

显示目标数据库中已安装语言的列表。

-V
--version

打印 createlang 的版本并退出。

-?
--help

显示有关 createlang 命令行参数的帮助并退出。

createlang 还接受下列用于连接参数的命令行参数：

`-h` *`host`*
`--host=`*`host`*

指定服务器所在主机的主机名。如果该值以斜杠开头，则将其用作 Unix 域套接字所在目录。

`-p` *`port`*
`--port=`*`port`*

指定服务器监听连接所用的 TCP 端口或本地 Unix 域套接字文件扩展名。

`-U` *`username`*
`--username=`*`username`*

要用来连接的用户名。

`-w`
`--no-password`

绝不提示输入密码。如果服务器要求密码认证，而密码又无法通过诸如`.pgpass`文件等其他方式获得，则连接尝试将失败。
该选项可用于没有用户在场输入密码的批处理作业和脚本。

`-W`
`--password`

强制 createlang 在连接数据库之前提示输入密码。

该选项绝非必需，因为如果服务器要求密码认证，createlang会自动提示输入密码。不过，createlang会浪费一次连接尝试来发现服务器需要密码。在某些情况下，输入`-W`可以避免这次额外的连接尝试。

环境

`PGDATABASE`
`PGHOST`
`PGPORT`
`PGUSER`

默认连接参数

和大部分其他 PostgreSQL 工具一样，这个工具也使用 libpq 支持的环境变量（参见第 31.14 节）。

诊断

大多数错误消息都是不言自明的。如果不然，可以带 `--echo` 选项运行 createlang，然后查看相应的 SQL 命令了解详情。此外，libpq 前端库使用的任何默认连接设置和环境变量也都适用。

注解

使用droplang可移除一个语言。

示例

要将语言 `pltcl` 安装到数据库 `template1` 中：

```
$ createlang pltcl template1
```

注意，将该语言安装到 `template1` 中后，之后创建的数据库中也会自动安装它。

参见

droplang, CREATE EXTENSION, CREATE LANGUAGE

createuser

createuser — 定义一个新的PostgreSQL用户账户

大纲

`createuser` [*connection-option*...] [*option*...] [*username*]

描述

createuser创建一个新的PostgreSQL用户（更准确地说，是一个角色）。
只有超级用户以及具有`CREATEROLE`权限的用户才能创建新用户，因此调用
createuser的用户必须能够以超级用户或具有`CREATEROLE`权限的用户身份连接。

如果希望创建新的超级用户，必须以超级用户身份连接，而不能仅凭`CREATEROLE`权限。
超级用户意味着能够绕过数据库内所有访问权限检查，因此不应轻易授予超级用户身份。

createuser是SQL命令CREATE　ROLE的包装器。通过此工具创建用户与通过其他访问服务器的方法创建用户，在效果上没有区别。

选项

createuser接受以下命令行参数：

username

指定要创建的PostgreSQL用户的名称。
该名称必须不同于此PostgreSQL安装中所有现有角色的名称。

`-c` *number*
`--connection-limit=`*number*

为新用户设置最大连接数。默认不设限制。

`-d`
`--createdb`

允许新用户创建数据库。

`-D`
`--no-createdb`

不允许新用户创建数据库。这是默认设置。

`-e`
`--echo`

回显createuser生成并发送给服务器的命令。

`-E`
`--encrypted`

加密存储在数据库中的用户密码。如果未指定，则使用默认的密码行为。

`-i`
`--inherit`

新角色将自动继承其所属角色的权限。这是默认设置。

-I
--no-inherit

新角色将不会自动继承其所属角色的权限。

--interactive

如果命令行中未指定用户名，则提示输入用户名；此外，
还会提示输入命令行中未指定的 -d/-D、-r/-R、-s/-S选项。（直到 PostgreSQL 9.1 为止，这都是默认行为。）

-l
--login

允许新用户登录（也就是说，该用户名可用作初始会话用户标识符）。这是默认设置。

-L
--no-login

不允许新用户登录。（不具有登录权限的角色仍可用于管理数据库权限。）

-N
--unencrypted

不加密存储在数据库中的用户密码。如果未指定，则使用默认的密码行为。

-P
--pwprompt

如果给出此选项，createuser会提示输入新用户的密码。如果不打算使用密码认证，
则不需要这样做。

-r
--createrole

允许新用户创建新角色（即该用户将具有CREATEROLE权限）。

-R
--no-createrole

不允许新用户创建新角色。这是默认设置。

-s
--superuser

新用户将成为超级用户。

-S
--no-superuser

新用户将不是超级用户。这是默认设置。

-V
--version

打印createuser版本并退出。

--replication

新用户将具有REPLICATION权限，关于该权限的更完整说明见CREATE ROLE。

`--no-replication`

新用户将不具有`REPLICATION`权限，关于该权限的更完整说明见CREATE ROLE。

`-?`
`--help`

显示有关createuser命令行参数的帮助并退出。

createuser还接受以下用于连接参数的命令行参数：

`-h` *`host`*
`--host=`*`host`*

指定服务器所在主机的主机名。如果该值以斜杠开头，则将其用作 Unix 域套接字所在目录。

`-p` *`port`*
`--port=`*`port`*

指定服务器监听连接所用的 TCP 端口或本地 Unix 域套接字文件扩展名。

`-U` *`username`*
`--username=`*`username`*

指定连接时使用的用户名（不是要创建的用户名）。

`-w`
`--no-password`

绝不提示输入密码。如果服务器要求密码认证，而密码又无法通过诸如`.pgpass`文件等其他方式获得，则连接尝试将失败。
该选项可用于没有用户在场输入密码的批处理作业和脚本。

`-W`
`--password`

强制createuser提示输入密码（用于连接到服务器，而不是新用户的密码）。

该选项绝非必需，因为如果服务器要求密码认证，createuser会自动提示输入密码。不过，createuser会浪费一次连接尝试来发现服务器需要密码。在某些情况下，输入`-W`可以避免这次额外的连接尝试。

环境

`PGHOST`
`PGPORT`
`PGUSER`

默认连接参数

与大多数其他PostgreSQL工具一样，此工具也使用 libpq支持的环境变量（见第 31.14 节）。

诊断

若遇到困难，请参见CREATE ROLE和psql，其中讨论了潜在问题和错误消息。数据库服务器必须在目标主机上运行。此外，
libpq前端库使用的任何默认连接设置和环境变量也都会生效。

示例

要在默认数据库服务器上创建用户joe：

```
$ createuser joe
```

要在默认数据库服务器上创建用户joe，并提示输入一些额外属性：

```
$ createuser --interactive joe
Shall the new role be a superuser? (y/n) n
Shall the new role be allowed to create databases? (y/n) n
Shall the new role be allowed to create more new roles? (y/n) n
```

要通过主机eden、端口 5000 上的服务器创建同一个用户joe，并显式指定各项属性，同时查看底层命令：

```
$ createuser -h eden -p 5000 -S -D -R -e joe
CREATE ROLE joe NOSUPERUSER NOCREATEDB NOCREATEROLE INHERIT LOGIN;
```

创建用户joe作为超级用户，并立即分配密码：

```
$ createuser -P -s -e joe
Enter password for new role: xyzzy
Enter it again: xyzzy
CREATE ROLE joe PASSWORD 'md5b5f5ba1a423792b526f799ae4eb3d59e'
 SUPERUSER CREATEDB CREATEROLE INHERIT LOGIN;
```

上面的示例中，输入密码时实际上不会回显新密码，但为清晰起见，
我们显示了所输入的内容。如你所见，密码在发送给客户端之前会被加密。如果使用了
--unencrypted选项，密码
将出现在回显的命令中（也可能出现在服务器日志以及其他地方），因此在这种情况下，
如果还有其他人能看到你的屏幕，就不要使用-e。

参见

dropuser, CREATE ROLE

dropdb

dropdb — 移除一个PostgreSQL数据库

大纲

`dropdb` [*connection-option*...] [*option*...] *dbname*

描述

dropdb删除一个现有的 PostgreSQL数据库。执行此命令的用户必须是数据库超级用户或该数据库的拥有者。

dropdb是SQL命令DROP DATABASE的包装器。通过此工具删除数据库与通过其他访问服务器的方法删除数据库，在效果上没有区别。

选项

dropdb接受以下命令行参数：

dbname

指定要移除的数据库名称。

`-e`
`--echo`

回显dropdb生成并发送给服务器的命令。

`-i`
`--interactive`

在执行任何破坏性操作之前发出确认提示。

`-V`
`--version`

打印dropdb版本并退出。

`--if-exists`

如果数据库不存在，则不要抛出错误。这种情况下会发出一个提示。

`-?`
`--help`

显示有关dropdb命令行参数的帮助并退出。

dropdb也接受下列用于连接参数的命令行参数：

`-h` *host*
`--host=`*host*

指定服务器所在机器的主机名。如果该值以斜杠开头，则它将被用作 Unix 域套接字的目录。

`-p` *port*
`--port=`*port*

指定服务器监听连接所用的 TCP 端口或本地 Unix 域套接字文件扩展名。

-U *username*
--username=*username*

指定用于连接的用户名。

-w
--no-password

从不发出密码提示。如果服务器要求密码认证，而密码又无法通过诸如 .pgpass文件等其他方式获得，则连接尝试将失败。
该选项适合没有用户在场输入密码的批处理作业和脚本。

-W
--password

强制dropdb在连接数据库之前提示输入密码。

这个选项并非必不可少，因为如果服务器要求密码认证，dropdb会自动提示输入密码。不过，dropdb会浪费一次连接尝试，才知道服务器需要密码。在某些情况下，为了避免这次额外的连接尝试，提前指定-W是值得的。

--maintenance-db=*dbname*

指定在删除目标数据库时要连接到的数据库名称。如果未指定，将使用 postgres数据库；若该数据库不存在（或者它正是要被删除的数据库），则改用template1。
这也可以是一个

环境

PGHOST
PGPORT
PGUSER

默认连接参数

与大多数其他PostgreSQL工具一样，此工具也使用 libpq支持的环境变量（见第 31. 14 节）。

诊断

若遇到困难，请参见DROP DATABASE和psql，其中讨论了潜在问题和错误消息。数据库服务器必须运行在目标主机上。此外，
libpq前端库使用的任何默认连接设置和环境变量也都会生效。

示例

要在默认数据库服务器上删除数据库demo：

```
$ dropdb demo
```

要使用主机eden、端口 5000 上的服务器删除数据库demo，并带确认提示以及查看底层命令：

```
$ dropdb -p 5000 -h eden -i -e demo
Database "demo" will be permanently deleted.
Are you sure? (y/n) y
DROP DATABASE demo;
```

另见

createdb, DROP DATABASE

droplang

droplang — 移除一个 PostgreSQL 过程语言

大纲

`droplang` [*connection-option*...] *langname* [*dbname*]

`droplang` [*connection-option*...] `--list` | `-l` [*dbname*]

描述

droplang 是一个从 PostgreSQL 数据库中移除现有过程语言的工具。

droplang 只是对DROP EXTENSION SQL 命令的一个包装。

> **小心**
>
> droplang 已被弃用，在未来的 PostgreSQL 版本中可能会被移除。建议直接使用 `DROP EXTENSION` 命令。

选项

droplang 接受下列命令行参数：

langname

指定要移除的过程语言的名称。（该名称会被转换为小写。）

[-d] *dbname*
[--dbname=]*dbname*

指定要从哪个数据库中移除该语言。默认使用与当前系统用户同名的数据库。

-e
--echo

在执行 SQL 命令时把它们显示出来。

-l
--list

显示目标数据库中已安装语言的列表。

-V
--version

打印 droplang 的版本并退出。

-?
--help

显示有关 droplang 命令行参数的帮助并退出。

droplang 还接受下列用于连接参数的命令行参数：

`-h` *`host`*
`--host=`*`host`*

指定服务器所在主机的主机名。如果该值以斜杠开头，则将其用作 Unix 域套接字所在目录。

`-p` *`port`*
`--port=`*`port`*

指定服务器监听连接所用的 TCP 端口或本地 Unix 域套接字文件扩展名。

`-U` *`username`*
`--username=`*`username`*

要用来连接的用户名。

`-w`
`--no-password`

绝不提示输入密码。如果服务器要求密码认证，而密码又无法通过诸如`.pgpass`文件等其他方式获得，则连接尝试将失败。
该选项可用于没有用户在场输入密码的批处理作业和脚本。

`-W`
`--password`

强制 droplang 在连接数据库之前提示输入密码。

该选项绝非必需，因为如果服务器要求密码认证，droplang会自动提示输入密码。不过，droplang会浪费一次连接尝试来发现服务器需要密码。在某些情况下，输入`-W`可以避免这次额外的连接尝试。

环境

`PGDATABASE`
`PGHOST`
`PGPORT`
`PGUSER`

默认连接参数

和大部分其他 PostgreSQL 工具一样，这个工具也使用 libpq 支持的环境变量（参见第 31.14 节）。

诊断

大多数错误消息都是不言自明的。如果不然，可以带 `--echo` 选项运行 droplang，然后查看相应的 SQL 命令了解详情。此外，libpq 前端库使用的任何默认连接设置和环境变量也都适用。

注解

使用createlang可添加一个语言。

示例

要移除语言 `pltcl`：

```
$ droplang pltcl dbname
```

参见

createlang, DROP EXTENSION, DROP LANGUAGE

dropuser

dropuser — 移除一个PostgreSQL用户账户

大纲

`dropuser` [*connection-option*...] [*option*...] [*username*]

描述

dropuser移除现有的PostgreSQL用户。只有超级用户和拥有`CREATEROLE`权限的用户才能移除PostgreSQL用户。（要移除超级用户，你自己也必须是超级用户。）

dropuser是SQL命令DROP ROLE的包装器。通过此工具删除用户与通过其他访问服务器的方法删除用户，在效果上没有区别。

选项

dropuser接受下列命令行参数：

username

指定要移除的PostgreSQL用户名。如果命令行中未指定名称，且使用了`-i`/`--interactive`选项，则会提示输入名称。

`-e`
`--echo`

回显dropuser生成并发送给服务器的命令。

`-i`
`--interactive`

在实际移除该用户之前提示确认；如果命令行中未指定用户名，也会提示输入用户名。

`-V`
`--version`

打印dropuser版本并退出。

`--if-exists`

如果该用户不存在，则不要抛出错误。这种情况下会发出一个提示。

`-?`
`--help`

显示有关dropuser命令行参数的帮助并退出。

dropuser也接受下列用于连接参数的命令行参数：

`-h` *host*
`--host=`*host*

指定服务器运行所在主机的主机名。如果该值以斜杠开头，则它将被用作 Unix 域套接字所在目录。

`-p port`
`--port=port`

指定服务器监听连接所用的 TCP 端口或本地 Unix 域套接字文件扩展名。

`-U username`
`--username=username`

用于连接的用户名（不是要删除的用户名）。

`-w`
`--no-password`

从不发出密码提示。如果服务器要求密码认证，而密码又无法通过诸如 `.pgpass`文件之类的其他方式获得，则连接尝试将失败。
该选项适合没有用户在场输入密码的批处理作业和脚本。

`-W`
`--password`

强制dropuser在连接到数据库之前提示输入密码。

这个选项绝非必需，因为如果服务器要求密码认证，dropuser会自动提示输入密码。不过，dropuser会浪费一次连接尝试来发现服务器需要密码。在某些情况下，输入`-W`值得一试，以避免这次额外的连接尝试。

环境

`PGHOST`
`PGPORT`
`PGUSER`

默认连接参数

与大多数其他PostgreSQL工具一样，此工具也使用 libpq支持的环境变量（见第 31.14 节）。

诊断

若遇到困难，请参见DROP ROLE和psql，其中讨论了潜在问题和错误消息。数据库服务器必须在目标主机上运行。此外，libpq前端库使用的任何默认连接设置和环境变量也都适用。

示例

要从默认数据库服务器上移除用户`joe`：

```
$ dropuser joe
```

要使用主机`eden`、端口 5000 上的服务器移除用户`joe`，并带确认提示以及查看底层命令：

```
$ dropuser -p 5000 -h eden -i -e joe
Role "joe" will be permanently removed.
Are you sure? (y/n) y
DROP ROLE joe;
```

另见

createuser, DROP ROLE

ecpg

ecpg — 嵌入式 SQL C 预处理器

大纲

ecpg [*option*...] *file*...

描述

ecpg是供 C 程序使用的嵌入式 SQL 预处理器。它会把嵌入了 SQL 语句的 C 程序转换成普通 C 代码，方法是将 SQL 调用替换为特殊的函数调用。生成的输出文件随后可以由任何 C 编译器工具链处理。

ecpg会将命令行中给出的每个输入文件转换成对应的 C 输出文件。若输入文件名没有任何扩展名，则假定其扩展名为.pgc。随后会把该文件的扩展名替换为 .c，以构造输出文件名。不过，也可以用-o选项覆盖输出文件名。

本参考页不描述嵌入式 SQL 语言。有关该主题的更多信息，见第 33 章。

选项

ecpg 接受以下命令行参数：

-c

自动从 SQL 代码生成某些 C 代码。目前，这适用于EXEC SQL TYPE。

-C *mode*

设置兼容模式。*mode*可以是INFORMIX或INFORMIX_SE。

-D *symbol*

定义一个 C 预处理器符号。

-i

也解析系统 include 文件。

-I *directory*

指定一个附加的 include 路径，用于查找通过EXEC SQL INCLUDE包含的文件。默认路径依次为 .（当前目录）、/usr/local/include、在编译时定义的PostgreSQL include 目录（默认：/usr/local/pgsql/include），以及 /usr/include。

-o *filename*

指定ecpg应将全部输出写入给定的*filename*。

-r *option*

选择运行时行为。*option* 可以是以下值之一：

no_indicator

不使用指示器，而改用特殊值表示空值。历史上曾有数据库采用这种方式。

`prepare`

在使用前预备所有语句。libecpg会维护一个已预备语句的缓存；如果某条语句再次执行，就会复用它。若缓存已满，libecpg会释放使用次数最少的语句。

`questionmarks`

出于兼容性原因，允许使用问号作为占位符。这在很久以前曾是默认行为。

`-t`

打开事务自动提交。在此模式下，除非 SQL 命令位于显式事务块中，否则每条 SQL 命令都会自动提交。在默认模式下，只有发出`EXEC SQL COMMIT`时才会提交命令。

`-v`

打印附加信息，包括版本和 include 路径。

`--version`

打印ecpg版本并退出。

`-?`
`--help`

显示关于ecpg命令行参数的帮助信息，并退出。

注解

在编译预处理后的 C 代码文件时，编译器必须能够在PostgreSQL的 include 目录中找到ECPG头文件。因此，调用编译器时你可能需要使用`-I`选项（例如`-I/usr/local/pgsql/include`）。

使用嵌入式 SQL 的 C 程序必须与`libecpg`库链接，例如可使用链接器选项`-L/usr/local/pgsql/lib -lecpg`。

适合该安装的这两个目录的值可以通过pg_config查出。

示例

如果你有一个名为`prog1.pgc`的嵌入式 SQL C 源文件，就可以使用下列命令序列创建一个可执行程序：

```
ecpg prog1.pgc
cc -I/usr/local/pgsql/include -c prog1.c
cc -o prog1 prog1.o -L/usr/local/pgsql/lib -lecpg
```

pg_basebackup

pg_basebackup — 获取PostgreSQL集簇的基础备份

大纲

`pg_basebackup` [*option*...]

描述

pg_basebackup用于获取正在运行的PostgreSQL数据库集簇的基础备份。
备份过程不会影响数据库的其他客户端，并且该备份既可用于时间点恢复（见第 24.3 节），也可用作日志传送或流复制备库的起点（见第 25.2 节）。

pg_basebackup创建数据库集簇文件的二进制副本，
同时确保系统自动进入和退出备份模式。备份始终针对整个数据库集簇；
无法只备份单个数据库或数据库对象。要备份单个数据库，必须使用pg_dump之类的工具。

备份通过使用复制协议的普通PostgreSQL连接进行。
建立连接时必须使用超级用户或具有`REPLICATION`权限（参见第 20.2 节）的用户，并且`pg_hba.conf`必须明确允许复制连接。服务器还必须将max_wal_senders设置得足够高，
以便至少为备份保留一个可用会话。

可以同时运行多个`pg_basebackup`，但从性能角度来看，最好只执行一次备份，
然后复制其结果。

pg_basebackup不仅可以从主库获取基础备份，也可以从备库获取。要从备库获取备份，
需要将备库配置为能够接受复制连接（即设置`max_wal_senders`和hot_standby，并配置基于主机的认证）。还需要在主库上启用full_page_writes。

注意，从备库获取在线备份时有一些限制：

- 备份历史文件不会在被备份的数据库集簇中创建。
- 无法保证备份所需的所有 WAL 文件都会在备份结束时被归档。如果你打算将该备份用于归档恢复，并希望确保届时所有必需文件都可用，就需要使用`-x` 选项将它们包含在备份中。
- 如果备库在在线备份过程中被提升为主库，则备份会失败。
- 备份所需的所有 WAL 记录都必须包含足够的整页写入，因此必须在主库上启用`full_page_writes`，并且不能把pg_compresslog之类的工具用作`archive_command`，从 WAL 文件中移除整页写入。

选项

以下命令行选项控制输出的位置和格式。

`-D` *directory*
`--pgdata=`*directory*

设置写入输出的目标目录。如果该目录不存在，pg_basebackup会创建它（以及所有缺失的父目录）。如果该目录已经存在，则必须为空。

当备份采用 tar 格式时，目标目录可以指定为`-`（短横线），从而将 tar 文件写到`stdout`。

此选项是必需的。

`-F` *`format`*
`--format=`*`format`*

选择输出格式。*`format`*可以是以下值之一：

`p`
`plain`

将输出写为普通文件，其布局与源服务器的数据目录和表空间相同。当集簇没有额外表空间时，整个数据库都会放在目标目录中。如果集簇包含额外表空间，则主数据目录会放在目标目录中，而其他所有表空间都会放在与服务器上相同的绝对路径中。

这是默认格式。

`t`
`tar`

将输出写为目标目录中的 tar 文件。主数据目录的内容会写入名为`base.tar`的文件中，而每个其他表空间都会写入一个以该表空间 OID 命名的独立 tar 文件中。

如果目标目录指定为`-`（短横线），tar 内容将写入标准输出，适合通过管道传给例如 gzip。只有当集簇没有额外表空间时，才允许这样做。

`-x`
`--xlog`

使用此选项等价于使用方法为`fetch`的`-X`。

`-X` *`method`*
`--xlog-method=`*`method`*

在备份中包含所需的事务日志（WAL）文件。这将包括备份期间生成的所有事务日志。如果指定了此选项，就可以在解包后的目录中直接启动 postmaster，而无需查阅日志归档，从而使输出成为一个完全独立的备份。

支持以下收集事务日志的方法：

`f`
`fetch`

在备份结束时收集事务日志文件。因此，wal_keep_segments参数必须设置得足够高，以确保在备份结束前不会移除所需的日志数据。如果在传输这些数据之前它们已经被回收，则备份会失败并且无法使用。

`s`
`stream`

在创建备份的同时流式传输事务日志。这会打开与服务器的第二个连接，并在执行备份时并行开始流式传输事务日志。因此，它会占用由max_wal_senders参数配置的两个槽。只要客户端能够跟上接收到的事务日志，使用此模式就不需要在主库上额外保留事务日志。

`-z`
`--gzip`

启用 tar 文件输出的 gzip 压缩，使用默认压缩级别。压缩仅在使用 tar 格式时可用。

`-Z level`
`--compress=level`

启用 tar 文件输出的 gzip 压缩，并指定压缩级别（0 到 9，其中 0 表示不压缩，9 表示最佳压缩）。压缩仅在使用 tar 格式时可用。

以下命令行选项控制备份的生成和程序的运行。

`-c fast|spread`
`--checkpoint=fast|spread`

将检查点模式设置为 fast 或 spread（默认）。

`-l label`
`--label=label`

设置备份标签。如果未指定，则使用默认值“`pg_basebackup base backup`”。

`-P`
`--progress`

启用进度报告。打开此选项后，会在备份过程中给出一个近似的进度报告。
由于数据库在备份过程中可能发生变化，因此这只是近似值，最终未必恰好结束在 `100%`。特别是当备份中包含 WAL 时，总数据量无法预先估计；在这种情况下，一旦进度超过不含 WAL 时的总估计值，估计目标大小就会继续增加。

启用此选项后，备份会先遍历整个数据库以统计其大小，然后再返回发送实际内容。
这可能使备份稍微耗时更长，尤其会使首次发送数据之前的等待时间更长。

`-v`
`--verbose`

启用详细模式。它会在启动和关闭过程中输出一些额外步骤；如果同时启用了进度报告，还会显示当前正在处理的确切文件名。

以下命令行选项控制数据库连接参数。

`-h host`
`--host=host`

指定服务器运行所在机器的主机名。如果该值以斜线开头，则它会被用作 Unix 域套接字的目录。默认值取自`PGHOST`环境变量（如果已设置）；否则会尝试使用 Unix 域套接字连接。

`-p port`
`--port=port`

指定服务器监听连接所使用的 TCP 端口，或本地 Unix 域套接字文件扩展名。默认使用`PGPORT`环境变量中的值（如果已设置），否则使用编译时确定的默认值。

`-s interval`
`--status-interval=interval`

指定向服务器回送状态包的时间间隔（秒）。如果在服务器上配置了复制超时，
并且以流式传输事务日志（使用`--xlog=stream`），这就是必需的；
同时它也更便于监视进度。值为零将完全禁用状态更新。默认值是 10 秒。

`-U username`
`--username=username`

指定连接时使用的用户名。

`-w`
`--no-password`

禁止发出密码提示。如果服务器要求密码认证，而又无法通过其他方式（例如`.pgpass`文件）获得密码，则连接尝试将失败。此选项对于批处理作业和脚本很有用，因为那种场景下通常没有用户在场输入密码。

`-W`
`--password`

强制pg_basebackup在连接数据库之前提示输入密码。

此选项绝非必需，因为如果服务器要求密码认证，pg_basebackup会自动提示输入密码。不过，pg_basebackup会浪费一次连接尝试来发现服务器需要密码。在某些情况下，输入`-W`以避免额外的连接尝试是值得的。

其他选项也可用：

`-V`
`--version`

输出pg_basebackup的版本并退出。

`-?`
`--help`

显示pg_basebackup命令行参数的帮助并退出。

环境

与大多数其他PostgreSQL工具一样，此工具也使用libpq支持的环境变量（见第 31. 14 节）。

注解

备份将包括数据目录和表空间中的所有文件，包括配置文件以及第三方放在这些目录中的任何额外文件。数据目录中只允许普通文件和目录，不允许符号链接和特殊设备文件。

由于 PostgreSQL 管理表空间的方式，无论何时恢复备份，所有附加表空间的路径都必须一致。不过，主数据目录可以重定位到任何位置。

示例

要为服务器`mydbserver`创建一个基础备份，并将其存储到本地目录`/usr/local/pgsql/data`中：

```
$ pg_basebackup -h mydbserver -D /usr/local/pgsql/data
```

要为本地服务器创建一个备份，为每个表空间各生成一个压缩的 tar 文件，并将其存储在目录`backup`中，同时在运行期间显示进度报告：

```
$ pg_basebackup -D backup -Ft -z -P
```

要为一个仅包含单个表空间的本地数据库创建备份，并使用bzip2进行压缩：

```
$ pg_basebackup -D - -Ft | bzip2 > backup.tar.bz2
```

（如果该数据库中有多个表空间，此命令将失败。）

另见

pg_dump

pg_config

pg_config — 获取已安装的PostgreSQL版本的信息

大纲

`pg_config` [*option*...]

描述

pg_config工具打印当前已安装的PostgreSQL版本的配置参数。例如，希望与PostgreSQL进行接口的软件包可以使用它，以便找到所需的头文件和库。

选项

要使用 pg_config，请提供以下一个或多个选项：

`--bindir`

打印用户可执行文件所在的位置。例如，可用它来查找`psql`程序。通常这也是`pg_config`程序所在的位置。

`--docdir`

打印文档文件所在的位置。

`--htmldir`

打印 HTML 文档文件所在的位置。

`--includedir`

打印客户端接口的 C 头文件所在的位置。

`--pkgincludedir`

打印其他 C 头文件所在的位置。

`--includedir-server`

打印服务器编程所用 C 头文件所在的位置。

`--libdir`

打印目标代码库所在的位置。

`--pkglibdir`

打印动态可加载模块所在的位置，或服务器将搜索这些模块的位置。（其他与体系结构相关的数据文件也可能安装在该目录中。）

`--localedir`

打印区域设置支持文件所在的位置。（如果在构建PostgreSQL时未配置区域设置支持，则这将是一个空字符串。）

`--mandir`

打印手册页所在的位置。

`--sharedir`

打印与体系结构无关的支持文件所在的位置。

`--sysconfdir`

打印系统范围的配置文件所在的位置。

`--pgxs`

打印扩展 makefile 的位置。

`--configure`

打印为构建PostgreSQL而运行`configure`脚本时给定的选项。
这可用于重现完全相同的配置，或者查明某个二进制软件包是用哪些选项构建的。
（但请注意，二进制软件包通常包含供应商特定的自定义补丁。）另请参见下面的示例。

`--cc`

打印构建PostgreSQL时所用的`CC`变量值。这显示所使用的 C 编译器。

`--cppflags`

打印构建PostgreSQL时所用的`CPPFLAGS`变量值。这显示预处理阶段所需的 C 编译器开关（通常是`-I`开关）。

`--cflags`

打印构建PostgreSQL时所用的`CFLAGS`变量值。这显示 C 编译器开关。

`--cflags_sl`

打印构建PostgreSQL时所用的`CFLAGS_SL`变量值。这显示构建共享库时使用的额外 C 编译器开关。

`--ldflags`

打印构建PostgreSQL时所用的`LDFLAGS`变量值。这显示链接器开关。

`--ldflags_ex`

打印构建PostgreSQL时所用的`LDFLAGS_EX`变量值。
这显示仅用于构建可执行文件的链接器开关。

`--ldflags_sl`

打印构建PostgreSQL时所用的`LDFLAGS_SL`变量值。
这显示仅用于构建共享库的链接器开关。

`--libs`

打印构建PostgreSQL时所用的`LIBS`变量值。这通常包含外部库的`-l`开关，
这些外部库会被链接进PostgreSQL。

`--version`

打印PostgreSQL的版本。

`-?`
`--help`

显示有关pg_config命令行参数的帮助，然后退出。

如果指定多个选项，信息将按选项顺序打印，每行一项。如果不指定选项，
则打印所有可用信息，并附带标签。

注解

选项`--includedir-server`是在PostgreSQL 7.2 中加入的。在更早的发行版中，服务器头文件与客户端头文件安装在同一位置，可以通过选项`--includedir`查询。
要让你的包同时处理这两种情况，可以先尝试较新的选项并测试退出状态以判断是否成功。

选项`--docdir`、`--pkgincludedir`、`--localedir`、`--mandir`、`--sharedir`、`--sysconfdir`、`--cc`、`--cppflags`、`--cflags`、`--cflags_sl`、`--ldflags`、`--ldflags_sl`以及`--libs`是在PostgreSQL 8.1 中加入的。选项`--htmldir`是在PostgreSQL 8.4 中加入的。选项`--ldflags_ex`是在PostgreSQL 9.0 中加入的。

在 PostgreSQL 7.1 之前的发行版中，还没有 `pg_config`，当时不存在获取等效配置信息的方法。

示例

要重现当前 PostgreSQL 安装的构建配置，请运行下列命令：

```
eval ./configure `pg_config --configure`
```

`pg_config --configure`的输出包含 shell 引号，因此带空格的参数能够被正确表示。因此，要得到正确的结果，需要使用`eval`。

pg_dump

pg_dump — extract a PostgreSQL database into a script file or other archive file

大纲

`pg_dump` [*connection-option*...] [*option*...] [*dbname*]

Description

pg_dump is a utility for backing up a PostgreSQL database. It makes consistent backups even if the database is being used concurrently. pg_dump does not block other users accessing the database (readers or writers).

Dumps can be output in script or archive file formats. Script dumps are plain-text files containing the SQL commands required to reconstruct the database to the state it was in at the time it was saved. To restore from such a script, feed it to psql. Script files can be used to reconstruct the database even on other machines and other architectures; with some modifications, even on other SQL database products.

The alternative archive file formats must be used with pg_restore to rebuild the database. They allow pg_restore to be selective about what is restored, or even to reorder the items prior to being restored. The archive file formats are designed to be portable across architectures.

When used with one of the archive file formats and combined with pg_restore, pg_dump provides a flexible archival and transfer mechanism. pg_dump can be used to backup an entire database, then pg_restore can be used to examine the archive and/or select which parts of the database are to be restored. The most flexible output file format is the "custom" format (`-Fc`). It allows for selection and reordering of all archived items, and is compressed by default.

While running pg_dump, one should examine the output for any warnings (printed on standard error), especially in light of the limitations listed below.

Options

The following command-line options control the content and format of the output.

dbname

Specifies the name of the database to be dumped. If this is not specified, the environment variable `PGDATABASE` is used. If that is not set, the user name specified for the connection is used.

`-a`
`--data-only`

Dump only the data, not the schema (data definitions). Table data, large objects, and sequence values are dumped.

This option is similar to, but for historical reasons not identical to, specifying `--section=data`.

`-b`
`--blobs`

Include large objects in the dump. This is the default behavior except when `--schema`, `--table`, or `--schema-only` is specified. The `-b` switch is therefore only

useful to add large objects to dumps where a specific schema or table has been requested. Note that blobs are considered data and therefore will be included when --data-only is used, but not when --schema-only is.

`-c`
`--clean`

Output commands to clean (drop) database objects prior to outputting the commands for creating them. (Restore might generate some harmless error messages, if any objects were not present in the destination database.)

This option is only meaningful for the plain-text format. For the archive formats, you can specify the option when you call `pg_restore`.

`-C`
`--create`

Begin the output with a command to create the database itself and reconnect to the created database. (With a script of this form, it doesn't matter which database in the destination installation you connect to before running the script.) If `--clean` is also specified, the script drops and recreates the target database before reconnecting to it.

This option is only meaningful for the plain-text format. For the archive formats, you can specify the option when you call `pg_restore`.

`-E` *`encoding`*
`--encoding=`*`encoding`*

Create the dump in the specified character set encoding. By default, the dump is created in the database encoding. (Another way to get the same result is to set the `PGCLIENTENCODING` environment variable to the desired dump encoding.)

`-f` *`file`*
`--file=`*`file`*

Send output to the specified file. This parameter can be omitted for file based output formats, in which case the standard output is used. It must be given for the directory output format however, where it specifies the target directory instead of a file. In this case the directory is created by `pg_dump` and must not exist before.

`-F` *`format`*
`--format=`*`format`*

Selects the format of the output. *`format`* can be one of the following:

`p`
`plain`

Output a plain-text SQL script file (the default).

`c`
`custom`

Output a custom-format archive suitable for input into pg_restore. Together with the directory output format, this is the most flexible output format in that it allows manual selection and reordering of archived items during restore. This format is also compressed by default.

`d`
`directory`

Output a directory-format archive suitable for input into pg_restore. This will create a directory with one file for each table and blob being dumped, plus a so-called Table of Contents file describing the dumped objects in a machine-readable format that pg_restore can read. A directory format archive can be manipulated with standard Unix tools; for example, files in an uncompressed archive can be compressed with the gzip tool. This format is compressed by default.

`t`
`tar`

Output a `tar`-format archive suitable for input into pg_restore. The tar format is compatible with the directory format: extracting a tar-format archive produces a valid directory-format archive. However, the tar format does not support compression. Also, when using tar format the relative order of table data items cannot be changed during restore.

`-i`
`--ignore-version`

A deprecated option that is now ignored.

`-n` *`schema`*
`--schema=`*`schema`*

Dump only schemas matching *`schema`*; this selects both the schema itself, and all its contained objects. When this option is not specified, all non-system schemas in the target database will be dumped. Multiple schemas can be selected by writing multiple `-n` switches. Also, the *`schema`* parameter is interpreted as a pattern according to the same rules used by psql's `\d` commands (see Patterns), so multiple schemas can also be selected by writing wildcard characters in the pattern. When using wildcards, be careful to quote the pattern if needed to prevent the shell from expanding the wildcards; see Examples.

> **注意**
>
> When `-n` is specified, pg_dump makes no attempt to dump any other database objects that the selected schema(s) might depend upon. Therefore, there is no guarantee that the results of a specific-schema dump can be successfully restored by themselves into a clean database.

> **注意**
>
> Non-schema objects such as blobs are not dumped when `-n` is specified. You can add blobs back to the dump with the `--blobs` switch.

`-N` *`schema`*
`--exclude-schema=`*`schema`*

Do not dump any schemas matching the *`schema`* pattern. The pattern is interpreted according to the same rules as for `-n`. `-N` can be given more than once to exclude schemas matching any of several patterns.

When both `-n` and `-N` are given, the behavior is to dump just the schemas that match at least one `-n` switch but no `-N` switches. If `-N` appears without `-n`, then schemas matching `-N` are excluded from what is otherwise a normal dump.

`-o`
`--oids`

Dump object identifiers (OIDs) as part of the data for every table. Use this option if your application references the OID columns in some way (e.g., in a foreign key constraint). Otherwise, this option should not be used.

`-O`
`--no-owner`

Do not output commands to set ownership of objects to match the original database. By default, pg_dump issues `ALTER OWNER` or `SET SESSION AUTHORIZATION` statements to set ownership of created database objects. These statements will fail when the script is run unless it is started by a superuser (or the same user that owns all of the objects in the script). To make a script that can be restored by any user, but will give that user ownership of all the objects, specify `-O`.

This option is only meaningful for the plain-text format. For the archive formats, you can specify the option when you call `pg_restore`.

`-R`
`--no-reconnect`

This option is obsolete but still accepted for backwards compatibility.

`-s`
`--schema-only`

Dump only the object definitions (schema), not data.

This option is the inverse of `--data-only`. It is similar to, but for historical reasons not identical to, specifying `--section=pre-data --section=post-data`.

(Do not confuse this with the `--schema` option, which uses the word "schema" in a different meaning.)

To exclude table data for only a subset of tables in the database, see `--exclude-table-data`.

`-S` *`username`*
`--superuser=`*`username`*

Specify the superuser user name to use when disabling triggers. This is only relevant if `--disable-triggers` is used. (Usually, it's better to leave this out, and instead start the resulting script as superuser.)

`-t` *`table`*
`--table=`*`table`*

Dump only tables (or views or sequences or foreign tables) matching *`table`*. Multiple tables can be selected by writing multiple `-t` switches. Also, the *`table`* parameter is interpreted as a pattern according to the same rules used by psql's `\d` commands (see Patterns), so multiple tables can also be selected by writing wildcard characters in the pattern. When using wildcards, be careful to quote the pattern if needed to prevent the shell from expanding the wildcards; see Examples.

The `-n` and `-N` switches have no effect when `-t` is used, because tables selected by `-t` will be dumped regardless of those switches, and non-table objects will not be dumped.

> **注意**
>
> When `-t` is specified, pg_dump makes no attempt to dump any other database objects that the selected table(s) might depend upon. Therefore, there is no guarantee that the results of a specific-table dump can be successfully restored by themselves into a clean database.

> **注意**
>
> The behavior of the `-t` switch is not entirely upward compatible with pre-8.2 PostgreSQL versions. Formerly, writing `-t tab` would dump all tables named `tab`, but now it just dumps whichever one is visible in your default search path. To get the old behavior you can write `-t '*.tab'`. Also, you must write something like `-t sch.tab` to select a table in a particular schema, rather than the old locution of `-n sch -t tab`.

`-T` *`table`*
`--exclude-table=`*`table`*

Do not dump any tables matching the *`table`* pattern. The pattern is interpreted according to the same rules as for `-t`. `-T` can be given more than once to exclude tables matching any of several patterns.

When both `-t` and `-T` are given, the behavior is to dump just the tables that match at least one `-t` switch but no `-T` switches. If `-T` appears without `-t`, then tables matching `-T` are excluded from what is otherwise a normal dump.

`-v`
`--verbose`

Specifies verbose mode. This will cause pg_dump to output detailed object comments and start/stop times to the dump file, and progress messages to standard error.

`-V`
`--version`

Print the pg_dump version and exit.

`-x`
`--no-privileges`
`--no-acl`

Prevent dumping of access privileges (grant/revoke commands).

`-Z` *`0..9`*
`--compress=`*`0..9`*

Specify the compression level to use. Zero means no compression. For the custom archive format, this specifies compression of individual table-data segments, and

the default is to compress at a moderate level. For plain text output, setting a nonzero compression level causes the entire output file to be compressed, as though it had been fed through gzip; but the default is not to compress. The tar archive format currently does not support compression at all.

`--binary-upgrade`

This option is for use by in-place upgrade utilities. Its use for other purposes is not recommended or supported. The behavior of the option may change in future releases without notice.

`--column-inserts`
`--attribute-inserts`

Dump data as `INSERT` commands with explicit column names (`INSERT INTO` *`table`* `(`*`column`*`, ...) VALUES ...`). This will make restoration very slow; it is mainly useful for making dumps that can be loaded into non-PostgreSQL databases. However, since this option generates a separate command for each row, an error in reloading a row causes only that row to be lost rather than the entire table contents.

`--disable-dollar-quoting`

This option disables the use of dollar quoting for function bodies, and forces them to be quoted using SQL standard string syntax.

`--disable-triggers`

This option is only relevant when creating a data-only dump. It instructs pg_dump to include commands to temporarily disable triggers on the target tables while the data is reloaded. Use this if you have referential integrity checks or other triggers on the tables that you do not want to invoke during data reload.

Presently, the commands emitted for `--disable-triggers` must be done as superuser. So, you should also specify a superuser name with `-S`, or preferably be careful to start the resulting script as a superuser.

This option is only meaningful for the plain-text format. For the archive formats, you can specify the option when you call `pg_restore`.

`--exclude-table-data=`*`table`*

Do not dump data for any tables matching the *`table`* pattern. The pattern is interpreted according to the same rules as for `-t`. `--exclude-table-data` can be given more than once to exclude tables matching any of several patterns. This option is useful when you need the definition of a particular table even though you do not need the data in it.

To exclude data for all tables in the database, see `--schema-only`.

`--inserts`

Dump data as `INSERT` commands (rather than `COPY`). This will make restoration very slow; it is mainly useful for making dumps that can be loaded into non-PostgreSQL databases. However, since this option generates a separate command for each row, an error in reloading a row causes only that row to be lost rather than the entire table contents. Note that the restore might fail altogether if you have rearranged column order. The `--column-inserts` option is safe against column order changes, though even slower.

`--lock-wait-timeout=timeout`

Do not wait forever to acquire shared table locks at the beginning of the dump. Instead fail if unable to lock a table within the specified *timeout*. The timeout may be specified in any of the formats accepted by `SET statement_timeout`. (Allowed values vary depending on the server version you are dumping from, but an integer number of milliseconds is accepted by all versions since 7.3. This option is ignored when dumping from a pre-7.3 server.)

`--no-security-labels`

Do not dump security labels.

`--no-tablespaces`

Do not output commands to select tablespaces. With this option, all objects will be created in whichever tablespace is the default during restore.

This option is only meaningful for the plain-text format. For the archive formats, you can specify the option when you call `pg_restore`.

`--no-unlogged-table-data`

Do not dump the contents of unlogged tables. This option has no effect on whether or not the table definitions (schema) are dumped; it only suppresses dumping the table data. Data in unlogged tables is always excluded when dumping from a standby server.

`--quote-all-identifiers`

Force quoting of all identifiers. This option is recommended when dumping a database from a server whose PostgreSQL major version is different from pg_dump's, or when the output is intended to be loaded into a server of a different major version. By default, pg_dump quotes only identifiers that are reserved words in its own major version. This sometimes results in compatibility issues when dealing with servers of other versions that may have slightly different sets of reserved words. Using `--quote-all-identifiers` prevents such issues, at the price of a harder-to-read dump script.

`--section=sectionname`

Only dump the named section. The section name can be `pre-data`, `data`, or `post-data`. This option can be specified more than once to select multiple sections. The default is to dump all sections.

The data section contains actual table data, large-object contents, and sequence values. Post-data items include definitions of indexes, triggers, rules, and constraints other than validated check constraints. Pre-data items include all other data definition items.

`--serializable-deferrable`

Use a `serializable` transaction for the dump, to ensure that the snapshot used is consistent with later database states; but do this by waiting for a point in the transaction stream at which no anomalies can be present, so that there isn't a risk of the dump failing or causing other transactions to roll back with a `serialization_failure`. See 第 13 章 for more information about transaction isolation and concurrency control.

This option is not beneficial for a dump which is intended only for disaster recovery. It could be useful for a dump used to load a copy of the database for reporting

or other read-only load sharing while the original database continues to be updated. Without it the dump may reflect a state which is not consistent with any serial execution of the transactions eventually committed. For example, if batch processing techniques are used, a batch may show as closed in the dump without all of the items which are in the batch appearing.

This option will make no difference if there are no read-write transactions active when pg_dump is started. If read-write transactions are active, the start of the dump may be delayed for an indeterminate length of time. Once running, performance with or without the switch is the same.

`--use-set-session-authorization`

Output SQL-standard `SET SESSION AUTHORIZATION` commands instead of `ALTER OWNER` commands to determine object ownership. This makes the dump more standards-compatible, but depending on the history of the objects in the dump, might not restore properly. Also, a dump using `SET SESSION AUTHORIZATION` will certainly require superuser privileges to restore correctly, whereas `ALTER OWNER` requires lesser privileges.

`-?`
`--help`

Show help about pg_dump command line arguments, and exit.

The following command-line options control the database connection parameters.

`-h` *`host`*
`--host=`*`host`*

Specifies the host name of the machine on which the server is running. If the value begins with a slash, it is used as the directory for the Unix domain socket. The default is taken from the `PGHOST` environment variable, if set, else a Unix domain socket connection is attempted.

`-p` *`port`*
`--port=`*`port`*

Specifies the TCP port or local Unix domain socket file extension on which the server is listening for connections. Defaults to the `PGPORT` environment variable, if set, or a compiled-in default.

`-U` *`username`*
`--username=`*`username`*

User name to connect as.

`-w`
`--no-password`

Never issue a password prompt. If the server requires password authentication and a password is not available by other means such as a `.pgpass` file, the connection attempt will fail. This option can be useful in batch jobs and scripts where no user is present to enter a password.

`-W`
`--password`

Force pg_dump to prompt for a password before connecting to a database.

This option is never essential, since pg_dump will automatically prompt for a password if the server demands password authentication. However, pg_dump will waste a connection attempt finding out that the server wants a password. In some cases it is worth typing `-W` to avoid the extra connection attempt.

`--role=rolename`

Specifies a role name to be used to create the dump. This option causes pg_dump to issue a `SET ROLE rolename` command after connecting to the database. It is useful when the authenticated user (specified by `-U`) lacks privileges needed by pg_dump, but can switch to a role with the required rights. Some installations have a policy against logging in directly as a superuser, and use of this option allows dumps to be made without violating the policy.

Environment

`PGDATABASE`
`PGHOST`
`PGOPTIONS`
`PGPORT`
`PGUSER`

Default connection parameters.

This utility, like most other PostgreSQL utilities, also uses the environment variables supported by libpq (see 第 31.14 节).

Diagnostics

pg_dump internally executes `SELECT` statements. If you have problems running pg_dump, make sure you are able to select information from the database using, for example, psql. Also, any default connection settings and environment variables used by the libpq front-end library will apply.

The database activity of pg_dump is normally collected by the statistics collector. If this is undesirable, you can set parameter `track_counts` to false via `PGOPTIONS` or the `ALTER USER` command.

Notes

If your database cluster has any local additions to the `template1` database, be careful to restore the output of pg_dump into a truly empty database; otherwise you are likely to get errors due to duplicate definitions of the added objects. To make an empty database without any local additions, copy from `template0` not `template1`, for example:

```
CREATE DATABASE foo WITH TEMPLATE template0;
```

When a data-only dump is chosen and the option `--disable-triggers` is used, pg_dump emits commands to disable triggers on user tables before inserting the data, and then commands to re-enable them after the data has been inserted. If the restore is stopped in the middle, the system catalogs might be left in the wrong state.

The dump file produced by pg_dump does not contain the statistics used by the optimizer to make query planning decisions. Therefore, it is wise to run `ANALYZE` after restoring from a dump file to ensure optimal performance; see 第 23.1.3 节 and 第 23.1.6 节 for more information. The dump file also does not contain any `ALTER DATABASE` .

`.. SET` commands; these settings are dumped by pg_dumpall, along with database users and other installation-wide settings.

Because pg_dump is used to transfer data to newer versions of PostgreSQL, the output of pg_dump can be expected to load into PostgreSQL server versions newer than pg_dump's version. pg_dump can also dump from PostgreSQL servers older than its own version. (Currently, servers back to version 7.0 are supported.) However, pg_dump cannot dump from PostgreSQL servers newer than its own major version; it will refuse to even try, rather than risk making an invalid dump. Also, it is not guaranteed that pg_dump's output can be loaded into a server of an older major version – not even if the dump was taken from a server of that version. Loading a dump file into an older server may require manual editing of the dump file to remove syntax not understood by the older server. Use of the `--quote-all-identifiers` option is recommended in cross-version cases, as it can prevent problems arising from varying reserved-word lists in different PostgreSQL versions.

Examples

To dump a database called `mydb` into a SQL-script file:

```
$ pg_dump mydb > db.sql
```

To reload such a script into a (freshly created) database named `newdb`:

```
$ psql -d newdb -f db.sql
```

To dump a database into a custom-format archive file:

```
$ pg_dump -Fc mydb > db.dump
```

To dump a database into a directory-format archive:

```
$ pg_dump -Fd mydb -f dumpdir
```

To reload an archive file into a (freshly created) database named `newdb`:

```
$ pg_restore -d newdb db.dump
```

To dump a single table named `mytab`:

```
$ pg_dump -t mytab mydb > db.sql
```

To dump all tables whose names start with `emp` in the `detroit` schema, except for the table named `employee_log`:

```
$ pg_dump -t 'detroit.emp*' -T detroit.employee_log mydb > db.sql
```

To dump all schemas whose names start with `east` or `west` and end in `gsm`, excluding any schemas whose names contain the word `test`:

```
$ pg_dump -n 'east*gsm' -n 'west*gsm' -N '*test*' mydb > db.sql
```

The same, using regular expression notation to consolidate the switches:

```
$ pg_dump -n '(east|west)*gsm' -N '*test*' mydb > db.sql
```

To dump all database objects except for tables whose names begin with `ts_`:

```
$ pg_dump -T 'ts_*' mydb > db.sql
```

To specify an upper-case or mixed-case name in `-t` and related switches, you need to double-quote the name; else it will be folded to lower case (see Patterns). But double

quotes are special to the shell, so in turn they must be quoted. Thus, to dump a single table with a mixed-case name, you need something like

```
$ pg_dump -t '"MixedCaseName"' mydb > mytab.sql
```

See Also

pg_dumpall, pg_restore, psql

pg_dumpall

pg_dumpall — 将一个PostgreSQL数据库集簇转储为脚本文件

大纲

`pg_dumpall` [*connection-option*...] [*option*...]

描述

pg_dumpall是一个实用程序，用于写出（“转储”）
一个集簇中的所有PostgreSQL数据库到一个脚本文件中。该脚本文件包含SQL命令，可作为psql的输入来恢复这些数据库。它通过对集簇中的每个数据库调用pg_dump来实现这一点。pg_dumpall还会转储所有数据库共有的全局对象。（pg_dump不会保存这些对象。）
目前包括数据库用户和组、表空间，以及适用于整个数据库的访问权限等属性的信息。

由于pg_dumpall会读取所有数据库中的表，因此通常必须以数据库超级用户身份连接，
才能生成完整的转储。另外，要执行保存的脚本，也需要具备超级用户权限，
这样才能添加用户和组并创建数据库。

SQL 脚本将写入标准输出。使用 [-f|file] 选项或 shell 操作符将其重定向到文件。

pg_dumpall需要多次连接到PostgreSQL服务器（每个数据库一次）。如果使用密码认证，它每次都会提示输入密码。在这种情况下准备一个`~/.pgpass`文件会比较方便。详见第 31.15 节。

选项

以下命令行选项控制输出的内容和格式。

`-a`
`--data-only`

只转储数据，不转储模式（数据定义）。

`-c`
`--clean`

包含用于在重新创建数据库之前清理（删除）数据库的 SQL 命令。也会添加用于角色和表空间的`DROP`命令。

`-f` *filename*
`--file=`*filename*

将输出发送到指定文件。如果省略此选项，则使用标准输出。

`-g`
`--globals-only`

只转储全局对象（角色和表空间），不转储数据库。

`-o`
`--oids`

将对象标识符（OID）作为每个表的数据的一部分进行转储。
如果应用程序以某种方式引用OID列（例如在外键约束中），请使用此选项。否则，
不应使用此选项。

`-O`
`--no-owner`

不要输出用于将对象所有权设置为与原始数据库一致的命令。默认情况下，pg_dumpall会发出`ALTER OWNER`或`SET SESSION AUTHORIZATION`语句，以设置已创建模式元素的所有权。除非该脚本由超级用户（或拥有脚本中全部对象的同一用户）启动，否则这些语句在运行时会失败。若要创建一个可由任意用户恢复、并让该用户获得所有对象所有权的脚本，请指定`-O`。

`-r`
`--roles-only`

只转储角色，不转储数据库或表空间。

`-s`
`--schema-only`

只转储对象定义（模式），不转储数据。

`-S` *`username`*
`--superuser=`*`username`*

指定在禁用触发器时要使用的超级用户名。只有在使用`--disable-triggers`时才相关。（通常更好的做法是省略此选项，而以超级用户身份运行生成的脚本。）

`-t`
`--tablespaces-only`

只转储表空间，不转储数据库或角色。

`-v`
`--verbose`

指定详细模式。这会让pg_dumpall把开始/停止时间写入转储文件，并将进度消息输出到标准错误。它还会启用pg_dump的详细输出。

`-V`
`--version`

打印pg_dumpall的版本并退出。

`-x`
`--no-privileges`
`--no-acl`

阻止转储访问权限（GRANT/REVOKE 命令）。

`--binary-upgrade`

此选项供就地升级工具使用。不建议也不支持将其用于其他用途。该选项的行为在将来的发行版中可能会在不另行通知的情况下发生变化。

`--column-inserts`
`--attribute-inserts`

将数据转储为带有显式列名的`INSERT`命令（`INSERT INTO` *`table`* `(`*`column`*`, ...) VALUES ...`）。这会使恢复变得非常缓慢；它主要用于生成可以装入非 PostgreSQL 数据库的转储。

`--disable-dollar-quoting`

此选项禁用函数体中的 dollar quoting，并强制改用 SQL 标准字符串语法对它们进行引用。

`--disable-triggers`

此选项只在创建包含数据但不包含模式的转储时才相关。它指示pg_dumpall在输出中包含一些命令，以便在恢复数据时临时禁用目标表上的触发器。如果这些表上存在不希望在数据恢复期间触发的引用完整性检查或其他触发器，请使用此选项。

目前，为`--disable-triggers`输出的这些命令必须由超级用户执行。因此，还应通过`-S`指定一个超级用户名，或者更好的做法是确保以超级用户身份运行生成的脚本。

`--inserts`

将数据转储为`INSERT`命令（而不是`COPY`）。这会使恢复非常缓慢；它主要用于生成可以装入非 PostgreSQL 数据库的转储。注意，如果重新安排了列顺序，恢复可能会彻底失败。`--column-inserts`选项可以避免列顺序变化带来的问题，但速度更慢。

`--lock-wait-timeout=`*`timeout`*

在转储开始时，不要无限等待获取共享表锁。如果无法在指定的*`timeout`*内锁定某个表，就让转储失败。超时可以用`SET statement_timeout`接受的任意格式指定。允许的值因被转储的服务器版本而异，但从 7.3 起，所有版本都接受以毫秒为单位的整数。对 7.3 之前的服务器进行转储时，此选项会被忽略。

`--no-security-labels`

不要转储安全标签。

`--no-tablespaces`

不输出创建表空间的命令，也不输出用于为对象选择表空间的命令。使用此选项时，所有对象在恢复时都会创建在当时默认的表空间中。

`--no-unlogged-table-data`

不要转储不记录 WAL 的表的内容。此选项不影响是否转储表定义（模式）；它只会抑制转储表数据。

`--quote-all-identifiers`

强制为所有标识符加引号。当从其PostgreSQL主版本与pg_dumpall不同的服务器转储数据库时，或者当输出打算装入另一主版本服务器中时，建议使用此选项。默认情况下，pg_dumpall只会给在其自身主版本中属于保留字的标识符加引号。处理其他版本服务器时，这有时会带来兼容性问题，因为它们的保留字集合可能略有不同。使用`--quote-all-identifiers`可以避免这类问题，但代价是转储脚本更难阅读。

`--use-set-session-authorization`

输出符合 SQL 标准的`SET SESSION AUTHORIZATION`命令，而不是用`ALTER OWNER`命令来确定对象所有权。这会让转储更符合标准，但根据转储中对象的历史，可能无法正确恢复。

`-?`
`--help`

显示关于pg_dumpall命令行参数的帮助信息，并退出。

以下命令行选项控制数据库连接参数。

-h *host*
--host=*host*

指定运行数据库服务器的机器的主机名。如果该值以斜杠开头，则将其用作 Unix 域套接字的目录。默认值取自PGHOST环境变量（如果已设置）；否则会尝试使用 Unix 域套接字连接。

-i
--ignore-version

一个已弃用、现在会被忽略的选项。

-l *dbname*
--database=*dbname*

指定连接来转储全局对象并发现还应转储哪些数据库的数据库名。如果未指定，则使用postgres数据库；如果该数据库不存在，则使用template1。

-p *port*
--port=*port*

指定服务器监听连接的 TCP 端口，或本地 Unix 域套接字文件扩展名。默认值取自PGPORT环境变量（如果已设置），否则使用编译时默认值。

-U *username*
--username=*username*

用于连接的用户名。

-w
--no-password

绝不提示输入密码。如果服务器要求密码认证，而又无法通过.pgpass文件等其他方式获得密码，则连接尝试会失败。该选项适用于批处理作业和脚本，因为这些场景下没有用户可以输入密码。

-W
--password

强制pg_dumpall在连接数据库之前提示输入密码。

此选项从来都不是必需的，因为如果服务器要求密码认证，pg_dumpall会自动提示输入密码。不过，pg_dumpall会浪费一次连接尝试来发现服务器需要密码。在某些情况下，键入-W以避免这次额外的连接尝试是值得的。

注意，对每个要转储的数据库都会再次提示输入密码。通常，最好设置一个~/.pgpass文件，而不是依赖手工输入密码。

--role=*rolename*

指定一个角色名，用于创建转储。此选项会使pg_dumpall在连接数据库后发出SET ROLE *rolename*命令。当已认证用户（由-U指定）缺少pg_dumpall所需权限，但可以切换到具备所需权限的角色时，这很有用。
有些安装环境不允许直接以超级用户身份登录，而使用此选项可以在不违反该策略的情况下完成转储。

环境

`PGHOST`
`PGOPTIONS`
`PGPORT`
`PGUSER`

默认连接参数

与大多数其他PostgreSQL工具一样，此实用程序也使用libpq支持的环境变量（见第 31.14 节）。

注解

由于pg_dumpall在内部调用pg_dump，因此某些诊断消息会提到pg_dump。

恢复完成后，最好对每个数据库运行`ANALYZE`，以便优化器获得有用的统计信息。也可以运行`vacuumdb -a -z`来分析所有数据库。

pg_dumpall要求在恢复之前，所有必需的表空间目录都已经存在；否则，位于非默认位置的数据库在创建时将会失败。

示例

要转储所有数据库：

```
$ pg_dumpall > db.out
```

要从此文件重新装载数据库，可以使用：

```
$ psql -f db.out postgres
```

（这里连接哪个数据库并不重要，因为 pg_dumpall 创建的脚本文件会包含适当的命令，用于创建并连接到已保存的数据库。）

另见

有关可能出现的错误情况，请参见pg_dump。

pg_receivexlog

pg_receivexlog — streams transaction logs from a PostgreSQL cluster

大纲

`pg_receivexlog` [*option*...]

Description

pg_receivexlog is used to stream transaction log from a running PostgreSQL cluster. The transaction log is streamed using the streaming replication protocol, and is written to a local directory of files. This directory can be used as the archive location for doing a restore using point-in-time recovery (see 第 24.3 节).

pg_receivexlog streams the transaction log in real time as it's being generated on the server, and does not wait for segments to complete like archive_command does. For this reason, it is not necessary to set archive_timeout when using pg_receivexlog.

The transaction log is streamed over a regular PostgreSQL connection, and uses the replication protocol. The connection must be made with a superuser or a user having `REPLICATION` permissions (see 第 20.2 节), and `pg_hba.conf` must explicitly permit the replication connection. The server must also be configured with max_wal_senders set high enough to leave at least one session available for the stream.

If the connection is lost, or if it cannot be initially established, with a non-fatal error, pg_receivexlog will retry the connection indefinitely, and reestablish streaming as soon as possible. To avoid this behavior, use the `-n` parameter.

Options

The following command-line options control the location and format of the output.

`-D` *directory*
`--directory=`*directory*

Directory to write the output to.

This parameter is required.

The following command-line options control the running of the program.

`-n`
`--no-loop`

Don't loop on connection errors. Instead, exit right away with an error.

`-v`
`--verbose`

Enables verbose mode.

The following command-line options control the database connection parameters.

`-h` *host*
`--host=`*host*

Specifies the host name of the machine on which the server is running. If the value begins with a slash, it is used as the directory for the Unix domain socket. The

default is taken from the `PGHOST` environment variable, if set, else a Unix domain socket connection is attempted.

`-p` *`port`*
`--port=`*`port`*

Specifies the TCP port or local Unix domain socket file extension on which the server is listening for connections. Defaults to the `PGPORT` environment variable, if set, or a compiled-in default.

`-s` *`interval`*
`--status-interval=`*`interval`*

Specifies the number of seconds between status packets sent back to the server. This is required if replication timeout is configured on the server, and allows for easier monitoring. A value of zero disables the status updates completely. The default value is 10 seconds.

`-U` *`username`*
`--username=`*`username`*

User name to connect as.

`-w`
`--no-password`

Never issue a password prompt. If the server requires password authentication and a password is not available by other means such as a `.pgpass` file, the connection attempt will fail. This option can be useful in batch jobs and scripts where no user is present to enter a password.

`-W`
`--password`

Force pg_receivexlog to prompt for a password before connecting to a database.

This option is never essential, since pg_receivexlog will automatically prompt for a password if the server demands password authentication. However, pg_receivexlog will waste a connection attempt finding out that the server wants a password. In some cases it is worth typing `-W` to avoid the extra connection attempt.

Other options are also available:

`-V`
`--version`

Print the pg_receivexlog version and exit.

`-?`
`--help`

Show help about pg_receivexlog command line arguments, and exit.

Environment

This utility, like most other PostgreSQL utilities, uses the environment variables supported by libpq (see 第 31.14 节).

Notes

When using pg_receivexlog instead of archive_command, the server will continue to recycle transaction log files even if the backups are not properly archived, since there is no command that fails. This can be worked around by having an archive_command that fails when the file has not been properly archived yet, for example:

```
archive_command = 'sleep 5 && test -f /mnt/server/archivedir/%f'
```

The initial timeout is necessary because pg_receivexlog works using asynchronous replication and can therefore be slightly behind the master.

Examples

To stream the transaction log from the server at `mydbserver` and store it in the local directory `/usr/local/pgsql/archive`:

```
$ pg_receivexlog -h mydbserver -D /usr/local/pgsql/archive
```

See Also

pg_basebackup

pg_restore

pg_restore — restore a PostgreSQL database from an archive file created by pg_dump

大纲

`pg_restore` [*connection-option*...] [*option*...] [*filename*]

Description

pg_restore is a utility for restoring a PostgreSQL database from an archive created by pg_dump in one of the non-plain-text formats. It will issue the commands necessary to reconstruct the database to the state it was in at the time it was saved. The archive files also allow pg_restore to be selective about what is restored, or even to reorder the items prior to being restored. The archive files are designed to be portable across architectures.

pg_restore can operate in two modes. If a database name is specified, pg_restore connects to that database and restores archive contents directly into the database. Otherwise, a script containing the SQL commands necessary to rebuild the database is created and written to a file or standard output. This script output is equivalent to the plain text output format of pg_dump. Some of the options controlling the output are therefore analogous to pg_dump options.

Obviously, pg_restore cannot restore information that is not present in the archive file. For instance, if the archive was made using the "dump data as `INSERT` commands" option, pg_restore will not be able to load the data using `COPY` statements.

Options

pg_restore accepts the following command line arguments.

filename

Specifies the location of the archive file (or directory, for a directory-format archive) to be restored. If not specified, the standard input is used.

`-a`
`--data-only`

Restore only the data, not the schema (data definitions). Table data, large objects, and sequence values are restored, if present in the archive.

This option is similar to, but for historical reasons not identical to, specifying `--section=data`.

`-c`
`--clean`

Clean (drop) database objects before recreating them. (This might generate some harmless error messages, if any objects were not present in the destination database.)

`-C`
`--create`

Create the database before restoring into it. If `--clean` is also specified, drop and recreate the target database before connecting to it.

When this option is used, the database named with `-d` is used only to issue the initial `DROP DATABASE` and `CREATE DATABASE` commands. All data is restored into the database name that appears in the archive.

`-d` *`dbname`*
`--dbname=`*`dbname`*

Connect to database *`dbname`* and restore directly into the database.

`-e`
`--exit-on-error`

Exit if an error is encountered while sending SQL commands to the database. The default is to continue and to display a count of errors at the end of the restoration.

`-f` *`filename`*
`--file=`*`filename`*

Specify output file for generated script, or for the listing when used with `-l`. Default is the standard output.

`-F` *`format`*
`--format=`*`format`*

Specify format of the archive. It is not necessary to specify the format, since pg_restore will determine the format automatically. If specified, it can be one of the following:

`c`
`custom`

The archive is in the custom format of pg_dump.

`d`
`directory`

The archive is a directory archive.

`t`
`tar`

The archive is a `tar` archive.

`-i`
`--ignore-version`

A deprecated option that is now ignored.

`-I` *`index`*
`--index=`*`index`*

Restore definition of named index only.

`-j` *`number-of-jobs`*
`--jobs=`*`number-of-jobs`*

Run the most time-consuming parts of pg_restore — those which load data, create indexes, or create constraints — using multiple concurrent jobs. This option can dramatically reduce the time to restore a large database to a server running on a multiprocessor machine.

Each job is one process or one thread, depending on the operating system, and uses a separate connection to the server.

The optimal value for this option depends on the hardware setup of the server, of the client, and of the network. Factors include the number of CPU cores and the disk setup. A good place to start is the number of CPU cores on the server, but values larger than that can also lead to faster restore times in many cases. Of course, values that are too high will lead to decreased performance because of thrashing.

Only the custom archive format is supported with this option. The input file must be a regular file (not, for example, a pipe). This option is ignored when emitting a script rather than connecting directly to a database server. Also, multiple jobs cannot be used together with the option `--single-transaction`.

`-l`
`--list`

List the contents of the archive. The output of this operation can be used as input to the `-L` option. Note that if filtering switches such as `-n` or `-t` are used with `-l`, they will restrict the items listed.

`-L` *`list-file`*
`--use-list=`*`list-file`*

Restore only those archive elements that are listed in *`list-file`*, and restore them in the order they appear in the file. Note that if filtering switches such as `-n` or `-t` are used with `-L`, they will further restrict the items restored.

`list-file` is normally created by editing the output of a previous `-l` operation. Lines can be moved or removed, and can also be commented out by placing a semicolon (`;`) at the start of the line. See below for examples.

`-n` *`namespace`*
`--schema=`*`schema`*

Restore only objects that are in the named schema. This can be combined with the `-t` option to restore just a specific table.

`-O`
`--no-owner`

Do not output commands to set ownership of objects to match the original database. By default, pg_restore issues `ALTER OWNER` or `SET SESSION AUTHORIZATION` statements to set ownership of created schema elements. These statements will fail unless the initial connection to the database is made by a superuser (or the same user that owns all of the objects in the script). With `-O`, any user name can be used for the initial connection, and this user will own all the created objects.

`-P` *`function-name(argtype [, ...])`*
`--function=`*`function-name(argtype [, ...])`*

Restore the named function only. Be careful to spell the function name and arguments exactly as they appear in the dump file's table of contents.

`-R`
`--no-reconnect`

This option is obsolete but still accepted for backwards compatibility.

`-s`
`--schema-only`

Restore only the schema (data definitions), not data, to the extent that schema entries are present in the archive.

This option is the inverse of `--data-only`. It is similar to, but for historical reasons not identical to, specifying `--section=pre-data --section=post-data`.

(Do not confuse this with the `--schema` option, which uses the word "schema" in a different meaning.)

`-S` *`username`*
`--superuser=`*`username`*

Specify the superuser user name to use when disabling triggers. This is only relevant if `--disable-triggers` is used.

`-t` *`table`*
`--table=`*`table`*

Restore definition and/or data of named table only. This can be combined with the `-n` option to specify a schema.

`-T` *`trigger`*
`--trigger=`*`trigger`*

Restore named trigger only.

`-v`
`--verbose`

Specifies verbose mode.

`-V`
`--version`

Print the pg_restore version and exit.

`-x`
`--no-privileges`
`--no-acl`

Prevent restoration of access privileges (grant/revoke commands).

`-1`
`--single-transaction`

Execute the restore as a single transaction (that is, wrap the emitted commands in `BEGIN`/`COMMIT`). This ensures that either all the commands complete successfully, or no changes are applied. This option implies `--exit-on-error`.

`--disable-triggers`

This option is only relevant when performing a data-only restore. It instructs pg_restore to execute commands to temporarily disable triggers on the target tables while the data is reloaded. Use this if you have referential integrity checks or other triggers on the tables that you do not want to invoke during data reload.

Presently, the commands emitted for `--disable-triggers` must be done as superuser. So, you should also specify a superuser name with `-S`, or preferably run pg_restore as a PostgreSQL superuser.

`--no-data-for-failed-tables`

By default, table data is restored even if the creation command for the table failed (e.g., because it already exists). With this option, data for such a table is skipped. This behavior is useful if the target database already contains the desired table contents. For example, auxiliary tables for PostgreSQL extensions such as PostGIS might already be loaded in the target database; specifying this option prevents duplicate or obsolete data from being loaded into them.

This option is effective only when restoring directly into a database, not when producing SQL script output.

`--no-security-labels`

Do not output commands to restore security labels, even if the archive contains them.

`--no-tablespaces`

Do not output commands to select tablespaces. With this option, all objects will be created in whichever tablespace is the default during restore.

`--section=`*`sectionname`*

Only restore the named section. The section name can be `pre-data`, `data`, or `post-data`. This option can be specified more than once to select multiple sections. The default is to restore all sections.

The data section contains actual table data as well as large-object definitions. Post-data items consist of definitions of indexes, triggers, rules and constraints other than validated check constraints. Pre-data items consist of all other data definition items.

`--use-set-session-authorization`

Output SQL-standard `SET SESSION AUTHORIZATION` commands instead of `ALTER OWNER` commands to determine object ownership. This makes the dump more standards-compatible, but depending on the history of the objects in the dump, might not restore properly.

`-?`
`--help`

Show help about pg_restore command line arguments, and exit.

pg_restore also accepts the following command line arguments for connection parameters:

`-h` *`host`*
`--host=`*`host`*

Specifies the host name of the machine on which the server is running. If the value begins with a slash, it is used as the directory for the Unix domain socket. The default is taken from the `PGHOST` environment variable, if set, else a Unix domain socket connection is attempted.

`-p` *`port`*
`--port=`*`port`*

Specifies the TCP port or local Unix domain socket file extension on which the server is listening for connections. Defaults to the `PGPORT` environment variable, if set, or a compiled-in default.

`-U` *`username`*
`--username=`*`username`*

User name to connect as.

`-w`
`--no-password`

Never issue a password prompt. If the server requires password authentication and a password is not available by other means such as a `.pgpass` file, the connection attempt will fail. This option can be useful in batch jobs and scripts where no user is present to enter a password.

`-W`
`--password`

Force pg_restore to prompt for a password before connecting to a database.

This option is never essential, since pg_restore will automatically prompt for a password if the server demands password authentication. However, pg_restore will waste a connection attempt finding out that the server wants a password. In some cases it is worth typing `-W` to avoid the extra connection attempt.

`--role=`*`rolename`*

Specifies a role name to be used to perform the restore. This option causes pg_restore to issue a `SET ROLE` *`rolename`* command after connecting to the database. It is useful when the authenticated user (specified by `-U`) lacks privileges needed by pg_restore, but can switch to a role with the required rights. Some installations have a policy against logging in directly as a superuser, and use of this option allows restores to be performed without violating the policy.

Environment

`PGHOST`
`PGOPTIONS`
`PGPORT`
`PGUSER`

Default connection parameters

This utility, like most other PostgreSQL utilities, also uses the environment variables supported by libpq (see 第 31.14 节).

Diagnostics

When a direct database connection is specified using the `-d` option, pg_restore internally executes SQL statements. If you have problems running pg_restore, make sure you are able to select information from the database using, for example, psql. Also, any default connection settings and environment variables used by the libpq front-end library will apply.

Notes

If your installation has any local additions to the `template1` database, be careful to load the output of pg_restore into a truly empty database; otherwise you are likely to get errors due to duplicate definitions of the added objects. To make an empty database without any local additions, copy from `template0` not `template1`, for example:

```
CREATE DATABASE foo WITH TEMPLATE template0;
```

The limitations of pg_restore are detailed below.

- When restoring data to a pre-existing table and the option `--disable-triggers` is used, pg_restore emits commands to disable triggers on user tables before inserting the data, then emits commands to re-enable them after the data has been inserted. If the restore is stopped in the middle, the system catalogs might be left in the wrong state.
- pg_restore cannot restore large objects selectively; for instance, only those for a specific table. If an archive contains large objects, then all large objects will be restored, or none of them if they are excluded via `-L`, `-t`, or other options.

See also the pg_dump documentation for details on limitations of pg_dump.

Once restored, it is wise to run `ANALYZE` on each restored table so the optimizer has useful statistics; see 第 23.1.3 节 and 第 23.1.6 节 for more information.

Examples

Assume we have dumped a database called `mydb` into a custom-format dump file:

```
$ pg_dump -Fc mydb > db.dump
```

To drop the database and recreate it from the dump:

```
$ dropdb mydb
$ pg_restore -C -d postgres db.dump
```

The database named in the `-d` switch can be any database existing in the cluster; pg_restore only uses it to issue the `CREATE DATABASE` command for `mydb`. With `-C`, data is always restored into the database name that appears in the dump file.

To reload the dump into a new database called `newdb`:

```
$ createdb -T template0 newdb
$ pg_restore -d newdb db.dump
```

Notice we don't use `-C`, and instead connect directly to the database to be restored into. Also note that we clone the new database from `template0` not `template1`, to ensure it is initially empty.

To reorder database items, it is first necessary to dump the table of contents of the archive:

```
$ pg_restore -l db.dump > db.list
```

The listing file consists of a header and one line for each item, e.g.:

```
;
; Archive created at Mon Sep 14 13:55:39 2009
;     dbname: DBDEMOS
;     TOC Entries: 81
;     Compression: 9
;     Dump Version: 1.10-0
;     Format: CUSTOM
;     Integer: 4 bytes
;     Offset: 8 bytes
;     Dumped from database version: 8.3.5
```

```
;     Dumped by pg_dump version: 8.3.8
;
;
; Selected TOC Entries:
;
3; 2615 2200 SCHEMA - public pasha
1861; 0 0 COMMENT - SCHEMA public pasha
1862; 0 0 ACL - public pasha
317; 1247 17715 TYPE public composite pasha
319; 1247 25899 DOMAIN public domain0 pasha
```

Semicolons start a comment, and the numbers at the start of lines refer to the internal archive ID assigned to each item.

Lines in the file can be commented out, deleted, and reordered. For example:

```
10; 145433 TABLE map_resolutions postgres
;2; 145344 TABLE species postgres
;4; 145359 TABLE nt_header postgres
6; 145402 TABLE species_records postgres
;8; 145416 TABLE ss_old postgres
```

could be used as input to pg_restore and would only restore items 10 and 6, in that order:

```
$ pg_restore -L db.list db.dump
```

See Also

pg_dump, pg_dumpall, psql

psql

psql — PostgreSQL interactive terminal

大纲

psql [*option*...] [*dbname* [*username*]]

Description

psql is a terminal-based front-end to PostgreSQL. It enables you to type in queries interactively, issue them to PostgreSQL, and see the query results. Alternatively, input can be from a file. In addition, it provides a number of meta-commands and various shell-like features to facilitate writing scripts and automating a wide variety of tasks.

Options

-a
--echo-all

Print all nonempty input lines to standard output as they are read. (This does not apply to lines read interactively.) This is equivalent to setting the variable ECHO to all.

-A
--no-align

Switches to unaligned output mode. (The default output mode is otherwise aligned.)

-c *command*
--command=*command*

Specifies that psql is to execute one command string, *command*, and then exit. This is useful in shell scripts. Start-up files (psqlrc and ~/.psqlrc) are ignored with this option.

command must be either a command string that is completely parsable by the server (i.e., it contains no psql-specific features), or a single backslash command. Thus you cannot mix SQL and psql meta-commands with this option. To achieve that, you could pipe the string into psql, for example: echo '\x \\ SELECT * FROM foo;' | psql. (\\ is the separator meta-command.)

If the command string contains multiple SQL commands, they are processed in a single transaction, unless there are explicit BEGIN/COMMIT commands included in the string to divide it into multiple transactions. This is different from the behavior when the same string is fed to psql's standard input. Also, only the result of the last SQL command is returned.

Because of these legacy behaviors, putting more than one command in the -c string often has unexpected results. It's better to feed multiple commands to psql's standard input, either using echo as illustrated above, or via a shell here-document, for example:

```
psql <<EOF
\x
SELECT * FROM foo;
```

```
EOF
```

`-d dbname`
`--dbname=dbname`

Specifies the name of the database to connect to. This is equivalent to specifying *dbname* as the first non-option argument on the command line.

If this parameter contains an = sign or starts with a valid URI prefix (`postgresql://` or `postgres://`), it is treated as a *conninfo* string. See 第 31.1 节 for more information.

`-e`
`--echo-queries`

Copy all SQL commands sent to the server to standard output as well. This is equivalent to setting the variable `ECHO` to `queries`.

`-E`
`--echo-hidden`

Echo the actual queries generated by `\d` and other backslash commands. You can use this to study psql's internal operations. This is equivalent to setting the variable `ECHO_HIDDEN` to `on`.

`-f filename`
`--file=filename`

Use the file *filename* as the source of commands instead of reading commands interactively. After the file is processed, psql terminates. This is in many ways equivalent to the meta-command `\i`.

If *filename* is - (hyphen), then standard input is read until an EOF indication or `\q` meta-command. Note however that Readline is not used in this case (much as if `-n` had been specified).

Using this option is subtly different from writing `psql < filename`. In general, both will do what you expect, but using `-f` enables some nice features such as error messages with line numbers. There is also a slight chance that using this option will reduce the start-up overhead. On the other hand, the variant using the shell's input redirection is (in theory) guaranteed to yield exactly the same output you would have received had you entered everything by hand.

`-F separator`
`--field-separator=separator`

Use *separator* as the field separator for unaligned output. This is equivalent to `\pset fieldsep` or `\f`.

`-h hostname`
`--host=hostname`

Specifies the host name of the machine on which the server is running. If the value begins with a slash, it is used as the directory for the Unix-domain socket.

`-H`
`--html`

Turn on HTML tabular output. This is equivalent to `\pset format html` or the `\H` command.

`-l`
`--list`

List all available databases, then exit. Other non-connection options are ignored. This is similar to the meta-command `\list`.

`-L` *`filename`*
`--log-file=`*`filename`*

Write all query output into file *`filename`*, in addition to the normal output destination.

`-n`
`--no-readline`

Do not use Readline for line editing and do not use the command history. This can be useful to turn off tab expansion when cutting and pasting.

`-o` *`filename`*
`--output=`*`filename`*

Put all query output into file *`filename`*. This is equivalent to the command `\o`.

`-p` *`port`*
`--port=`*`port`*

Specifies the TCP port or the local Unix-domain socket file extension on which the server is listening for connections. Defaults to the value of the `PGPORT` environment variable or, if not set, to the port specified at compile time, usually 5432.

`-P` *`assignment`*
`--pset=`*`assignment`*

Specifies printing options, in the style of `\pset`. Note that here you have to separate name and value with an equal sign instead of a space. For example, to set the output format to LaTeX, you could write `-P format=latex`.

`-q`
`--quiet`

Specifies that psql should do its work quietly. By default, it prints welcome messages and various informational output. If this option is used, none of this happens. This is useful with the `-c` option. This is equivalent to setting the variable `QUIET` to `on`.

`-R` *`separator`*
`--record-separator=`*`separator`*

Use *`separator`* as the record separator for unaligned output. This is equivalent to the `\pset recordsep` command.

`-s`
`--single-step`

Run in single-step mode. That means the user is prompted before each command is sent to the server, with the option to cancel execution as well. Use this to debug scripts.

`-S`
`--single-line`

Runs in single-line mode where a newline terminates an SQL command, as a semicolon does.

> **注意**
>
> This mode is provided for those who insist on it, but you are not necessarily encouraged to use it. In particular, if you mix SQL and meta-commands on a line the order of execution might not always be clear to the inexperienced user.

`-t`
`--tuples-only`

Turn off printing of column names and result row count footers, etc. This is equivalent to the `\t` command.

`-T` *`table_options`*
`--table-attr=`*`table_options`*

Specifies options to be placed within the HTML `table` tag. See `\pset` for details.

`-U` *`username`*
`--username=`*`username`*

Connect to the database as the user *`username`* instead of the default. (You must have permission to do so, of course.)

`-v` *`assignment`*
`--set=`*`assignment`*
`--variable=`*`assignment`*

Perform a variable assignment, like the `\set` meta-command. Note that you must separate name and value, if any, by an equal sign on the command line. To unset a variable, leave off the equal sign. To set a variable with an empty value, use the equal sign but leave off the value. These assignments are done during a very early stage of start-up, so variables reserved for internal purposes might get overwritten later.

`-V`
`--version`

Print the psql version and exit.

`-w`
`--no-password`

Never issue a password prompt. If the server requires password authentication and a password is not available by other means such as a `.pgpass` file, the connection attempt will fail. This option can be useful in batch jobs and scripts where no user is present to enter a password.

Note that this option will remain set for the entire session, and so it affects uses of the meta-command `\connect` as well as the initial connection attempt.

`-W`
`--password`

Force psql to prompt for a password before connecting to a database.

This option is never essential, since psql will automatically prompt for a password if the server demands password authentication. However, psql will waste a connection attempt finding out that the server wants a password. In some cases it is worth typing `-W` to avoid the extra connection attempt.

Note that this option will remain set for the entire session, and so it affects uses of the meta-command `\connect` as well as the initial connection attempt.

`-x`
`--expanded`

Turn on the expanded table formatting mode. This is equivalent to the `\x` command.

`-X,`
`--no-psqlrc`

Do not read the start-up file (neither the system-wide `psqlrc` file nor the user's `~/.psqlrc` file).

`-z`
`--field-separator-zero`

Set the field separator for unaligned output to a zero byte.

`-0`
`--record-separator-zero`

Set the record separator for unaligned output to a zero byte. This is useful for interfacing, for example, with `xargs -0`.

`-1`
`--single-transaction`

When psql executes a script with the `-f` option, adding this option wraps `BEGIN` /`COMMIT` around the script to execute it as a single transaction. This ensures that either all the commands complete successfully, or no changes are applied.

If the script itself uses `BEGIN`, `COMMIT`, or `ROLLBACK`, this option will not have the desired effects. Also, if the script contains any command that cannot be executed inside a transaction block, specifying this option will cause that command (and hence the whole transaction) to fail.

`-?`
`--help`

Show help about psql command line arguments, and exit.

Exit Status

psql returns 0 to the shell if it finished normally, 1 if a fatal error of its own occurs (e.g. out of memory, file not found), 2 if the connection to the server went bad and the session was not interactive, and 3 if an error occurred in a script and the variable `ON_ERROR_STOP` was set.

Usage

Connecting to a Database

psql is a regular PostgreSQL client application. In order to connect to a database you need to know the name of your target database, the host name and port number of the server, and what user name you want to connect as. psql can be told about those parameters via command line options, namely `-d`, `-h`, `-p`, and `-U` respectively. If an argument is found that does not belong to any option it will be interpreted

as the database name (or the user name, if the database name is already given). Not all of these options are required; there are useful defaults. If you omit the host name, psql will connect via a Unix-domain socket to a server on the local host, or via TCP/IP to `localhost` on machines that don't have Unix-domain sockets. The default port number is determined at compile time. Since the database server uses the same default, you will not have to specify the port in most cases. The default user name is your operating-system user name, as is the default database name. Note that you cannot just connect to any database under any user name. Your database administrator should have informed you about your access rights.

When the defaults aren't quite right, you can save yourself some typing by setting the environment variables `PGDATABASE`, `PGHOST`, `PGPORT` and/or `PGUSER` to appropriate values. (For additional environment variables, see 第 31.14 节.) It is also convenient to have a `~/.pgpass` file to avoid regularly having to type in passwords. See 第 31.15 节 for more information.

An alternative way to specify connection parameters is in a *`conninfo`* string or a URI, which is used instead of a database name. This mechanism give you very wide control over the connection. For example:

```
$ psql "service=myservice sslmode=require"
$ psql postgresql://dbmaster:5433/mydb?sslmode=require
```

This way you can also use LDAP for connection parameter lookup as described in 第 31.17 节. See 第 31.1 节 for more information on all the available connection options.

If the connection could not be made for any reason (e.g., insufficient privileges, server is not running on the targeted host, etc.), psql will return an error and terminate.

If at least one of standard input or standard output are a terminal, then psql sets the client encoding to "auto", which will detect the appropriate client encoding from the locale settings (`LC_CTYPE` environment variable on Unix systems). If this doesn't work out as expected, the client encoding can be overridden using the environment variable `PGCLIENTENCODING`.

Entering SQL Commands

In normal operation, psql provides a prompt with the name of the database to which psql is currently connected, followed by the string `=>`. For example:

```
$ psql testdb
psql (9.2.24)
Type "help" for help.

testdb=>
```

At the prompt, the user can type in SQL commands. Ordinarily, input lines are sent to the server when a command-terminating semicolon is reached. An end of line does not terminate a command. Thus commands can be spread over several lines for clarity. If the command was sent and executed without error, the results of the command are displayed on the screen.

Whenever a command is executed, psql also polls for asynchronous notification events generated by LISTEN and NOTIFY.

Meta-Commands

Anything you enter in psql that begins with an unquoted backslash is a psql meta-command that is processed by psql itself. These commands make psql more useful

for administration or scripting. Meta-commands are often called slash or backslash commands.

The format of a psql command is the backslash, followed immediately by a command verb, then any arguments. The arguments are separated from the command verb and each other by any number of whitespace characters.

To include whitespace in an argument you can quote it with single quotes. To include a single quote in an argument, write two single quotes within single-quoted text. Anything contained in single quotes is furthermore subject to C-like substitutions for `\n` (new line), `\t` (tab), `\b` (backspace), `\r` (carriage return), `\f` (form feed), `\digits` (octal), and `\xdigits` (hexadecimal). A backslash preceding any other character within single-quoted text quotes that single character, whatever it is.

Within an argument, text that is enclosed in backquotes (`) is taken as a command line that is passed to the shell. The output of the command (with any trailing newline removed) replaces the backquoted text.

If an unquoted colon (`:`) followed by a psql variable name appears within an argument, it is replaced by the variable's value, as described in SQL Interpolation.

Some commands take an SQL identifier (such as a table name) as argument. These arguments follow the syntax rules of SQL: Unquoted letters are forced to lowercase, while double quotes (`"`) protect letters from case conversion and allow incorporation of whitespace into the identifier. Within double quotes, paired double quotes reduce to a single double quote in the resulting name. For example, `FOO"BAR"BAZ` is interpreted as `fooBARbaz`, and `"A weird"" name"` becomes `A weird" name`.

Parsing for arguments stops at the end of the line, or when another unquoted backslash is found. An unquoted backslash is taken as the beginning of a new meta-command. The special sequence `\\` (two backslashes) marks the end of arguments and continues parsing SQL commands, if any. That way SQL and psql commands can be freely mixed on a line. But in any case, the arguments of a meta-command cannot continue beyond the end of the line.

The following meta-commands are defined:

`\a`

If the current table output format is unaligned, it is switched to aligned. If it is not unaligned, it is set to unaligned. This command is kept for backwards compatibility. See `\pset` for a more general solution.

`\c` or `\connect [ -reuse-previous=on|off ] [ dbname [ username ] [ host ] [ port ] | conninfo ]`

Establishes a new connection to a PostgreSQL server. The connection parameters to use can be specified either using a positional syntax, or using `conninfo` connection strings as detailed in 第 31.1.1 节.

Where the command omits database name, user, host, or port, the new connection can reuse values from the previous connection. By default, values from the previous connection are reused except when processing a `conninfo` string. Passing a first argument of `-reuse-previous=on` or `-reuse-previous=off` overrides that default. When the command neither specifies nor reuses a particular parameter, the libpq default is used. Specifying any of *dbname*, *username*, *host* or *port* as – is equivalent to omitting that parameter.

If the new connection is successfully made, the previous connection is closed. If the connection attempt failed (wrong user name, access denied, etc.), the previous

connection will only be kept if psql is in interactive mode. When executing a non-interactive script, processing will immediately stop with an error. This distinction was chosen as a user convenience against typos on the one hand, and a safety mechanism that scripts are not accidentally acting on the wrong database on the other hand.

Examples:

```
=> \c mydb myuser host.dom 6432
=> \c service=foo
=> \c "host=localhost port=5432 dbname=mydb connect_timeout=10
 sslmode=disable"
=> \c postgresql://tom@localhost/mydb?application_name=myapp
```

`\C [ title ]`

Sets the title of any tables being printed as the result of a query or unset any such title. This command is equivalent to `\pset title title`. (The name of this command derives from "caption", as it was previously only used to set the caption in an HTML table.)

`\cd [ directory ]`

Changes the current working directory to *directory*. Without argument, changes to the current user's home directory.

> **提示**
>
> To print your current working directory, use `\! pwd`.

`\conninfo`

Outputs information about the current database connection.

`\copy { table [(column_list)] | (query) } { from | to } { filename | stdin | stdout | pstdin | pstdout } [with] [binary] [oids] [delimiter [as] 'character'] [null [as] 'string'] [csv [header] [quote [as] 'character'] [escape [as] 'character'] [force quote column_list | *] [force not null column_list]]`

Performs a frontend (client) copy. This is an operation that runs an SQL COPY command, but instead of the server reading or writing the specified file, psql reads or writes the file and routes the data between the server and the local file system. This means that file accessibility and privileges are those of the local user, not the server, and no SQL superuser privileges are required.

The syntax of the command is similar to that of the SQL COPY command. Note that, because of this, special parsing rules apply to the `\copy` command. In particular, the variable substitution rules and backslash escapes do not apply.

`\copy ... from stdin | to stdout` reads/writes based on the command input and output respectively. All rows are read from the same source that issued the command, continuing until `\.` is read or the stream reaches EOF. Output is sent to the same place as command output. To read/write from psql's standard input or output, use `pstdin` or `pstdout`. This option is useful for populating tables in-line within a SQL script file.

> **提示**
>
> This operation is not as efficient as the SQL `COPY` command because all data must pass through the client/server connection. For large amounts of data the SQL command might be preferable.

`\copyright`

Shows the copyright and distribution terms of PostgreSQL.

`\d[S+] [ pattern ]`

For each relation (table, view, index, sequence, or foreign table) or composite type matching the *pattern*, show all columns, their types, the tablespace (if not the default) and any special attributes such as `NOT NULL` or defaults. Associated indexes, constraints, rules, and triggers are also shown. For foreign tables, the associated foreign server is shown as well. ("Matching the pattern" is defined in Patterns below.)

For some types of relation, `\d` shows additional information for each column: column values for sequences, indexed expression for indexes and foreign data wrapper options for foreign tables.

The command form `\d+` is identical, except that more information is displayed: any comments associated with the columns of the table are shown, as is the presence of OIDs in the table, the view definition if the relation is a view.

By default, only user-created objects are shown; supply a pattern or the `S` modifier to include system objects.

> **注意**
>
> If `\d` is used without a *pattern* argument, it is equivalent to `\dtvsE` which will show a list of all visible tables, views, sequences and foreign tables. This is purely a convenience measure.

`\da[S] [ pattern ]`

Lists aggregate functions, together with their return type and the data types they operate on. If *pattern* is specified, only aggregates whose names match the pattern are shown. By default, only user-created objects are shown; supply a pattern or the `S` modifier to include system objects.

`\db[+] [ pattern ]`

Lists tablespaces. If *pattern* is specified, only tablespaces whose names match the pattern are shown. If `+` is appended to the command name, each object is listed with its associated permissions.

`\dc[S+] [ pattern ]`

Lists conversions between character-set encodings. If *pattern* is specified, only conversions whose names match the pattern are listed. By default, only user-created objects are shown; supply a pattern or the `S` modifier to include system

objects. If + is appended to the command name, each object is listed with its associated description.

`\dC[+] [ pattern ]`

Lists type casts. If *pattern* is specified, only casts whose source or target types match the pattern are listed. If + is appended to the command name, each object is listed with its associated description.

`\dd[S] [ pattern ]`

Shows the descriptions of objects of type `constraint`, `operator class`, `operator family`, `rule`, and `trigger`. All other comments may be viewed by the respective backslash commands for those object types.

`\dd` displays descriptions for objects matching the *pattern*, or of visible objects of the appropriate type if no argument is given. But in either case, only objects that have a description are listed. By default, only user-created objects are shown; supply a pattern or the `S` modifier to include system objects.

Descriptions for objects can be created with the COMMENT SQL command.

`\ddp [ pattern ]`

Lists default access privilege settings. An entry is shown for each role (and schema, if applicable) for which the default privilege settings have been changed from the built-in defaults. If *pattern* is specified, only entries whose role name or schema name matches the pattern are listed.

The ALTER DEFAULT PRIVILEGES command is used to set default access privileges. The meaning of the privilege display is explained under GRANT.

`\dD[S+] [ pattern ]`

Lists domains. If *pattern* is specified, only domains whose names match the pattern are shown. By default, only user-created objects are shown; supply a pattern or the `S` modifier to include system objects. If + is appended to the command name, each object is listed with its associated permissions and description.

```
\dE[S+] [ pattern ]
\di[S+] [ pattern ]
\ds[S+] [ pattern ]
\dt[S+] [ pattern ]
\dv[S+] [ pattern ]
```

In this group of commands, the letters `E`, `i`, `s`, `t`, and `v` stand for foreign table, index, sequence, table, and view, respectively. You can specify any or all of these letters, in any order, to obtain a listing of objects of these types. For example, `\dit` lists indexes and tables. If + is appended to the command name, each object is listed with its physical size on disk and its associated description, if any. If *pattern* is specified, only objects whose names match the pattern are listed. By default, only user-created objects are shown; supply a pattern or the `S` modifier to include system objects.

`\des[+] [ pattern ]`

Lists foreign servers (mnemonic: "external servers"). If *pattern* is specified, only those servers whose name matches the pattern are listed. If the form `\des+` is used, a full description of each server is shown, including the server's ACL, type, version, options, and description.

`\det[+] [ pattern ]`

Lists foreign tables (mnemonic: "external tables"). If *pattern* is specified, only entries whose table name or schema name matches the pattern are listed. If the form `\det+` is used, generic options and the foreign table description are also displayed.

`\deu[+] [ pattern ]`

Lists user mappings (mnemonic: "external users"). If *pattern* is specified, only those mappings whose user names match the pattern are listed. If the form `\deu+` is used, additional information about each mapping is shown.

> **小心**
>
> `\deu+` might also display the user name and password of the remote user, so care should be taken not to disclose them.

`\dew[+] [ pattern ]`

Lists foreign-data wrappers (mnemonic: "external wrappers"). If *pattern* is specified, only those foreign-data wrappers whose name matches the pattern are listed. If the form `\dew+` is used, the ACL, options, and description of the foreign-data wrapper are also shown.

`\df[antwS+] [ pattern ]`

Lists functions, together with their arguments, return types, and function types, which are classified as "agg" (aggregate), "normal", "trigger", or "window". To display only functions of specific type(s), add the corresponding letters `a`, `n`, `t`, or `w` to the command. If *pattern* is specified, only functions whose names match the pattern are shown. If the form `\df+` is used, additional information about each function, including volatility, language, source code and description, is shown. By default, only user-created objects are shown; supply a pattern or the `S` modifier to include system objects.

> **提示**
>
> To look up functions taking arguments or returning values of a specific type, use your pager's search capability to scroll through the `\df` output.

`\dF[+] [ pattern ]`

Lists text search configurations. If *pattern* is specified, only configurations whose names match the pattern are shown. If the form `\dF+` is used, a full description of each configuration is shown, including the underlying text search parser and the dictionary list for each parser token type.

`\dFd[+] [ pattern ]`

Lists text search dictionaries. If *pattern* is specified, only dictionaries whose names match the pattern are shown. If the form `\dFd+` is used, additional information is shown about each selected dictionary, including the underlying text search template and the option values.

`\dFp[+] [ pattern ]`

Lists text search parsers. If *pattern* is specified, only parsers whose names match the pattern are shown. If the form `\dFp+` is used, a full description of each parser is shown, including the underlying functions and the list of recognized token types.

`\dFt[+] [ pattern ]`

Lists text search templates. If *pattern* is specified, only templates whose names match the pattern are shown. If the form `\dFt+` is used, additional information is shown about each template, including the underlying function names.

`\dg[+] [ pattern ]`

Lists database roles. (Since the concepts of "users" and "groups" have been unified into "roles", this command is now equivalent to `\du`.) If *pattern* is specified, only those roles whose names match the pattern are listed. If the form `\dg+` is used, additional information is shown about each role; currently this adds the comment for each role.

`\dl`

This is an alias for `\lo_list`, which shows a list of large objects.

`\dL[S+] [ pattern ]`

Lists procedural languages. If *pattern* is specified, only languages whose names match the pattern are listed. By default, only user-created languages are shown; supply the `S` modifier to include system objects. If `+` is appended to the command name, each language is listed with its call handler, validator, access privileges, and whether it is a system object.

`\dn[S+] [ pattern ]`

Lists schemas (namespaces). If *pattern* is specified, only schemas whose names match the pattern are listed. By default, only user-created objects are shown; supply a pattern or the `S` modifier to include system objects. If `+` is appended to the command name, each object is listed with its associated permissions and description, if any.

`\do[S] [ pattern ]`

Lists operators with their operand and return types. If *pattern* is specified, only operators whose names match the pattern are listed. By default, only user-created objects are shown; supply a pattern or the `S` modifier to include system objects.

`\dO[S+] [ pattern ]`

Lists collations. If *pattern* is specified, only collations whose names match the pattern are listed. By default, only user-created objects are shown; supply a pattern or the `S` modifier to include system objects. If `+` is appended to the command name, each collation is listed with its associated description, if any. Note that only collations usable with the current database's encoding are shown, so the results may vary in different databases of the same installation.

`\dp [ pattern ]`

Lists tables, views and sequences with their associated access privileges. If *pattern* is specified, only tables, views and sequences whose names match the pattern are listed.

The GRANT and REVOKE commands are used to set access privileges. The meaning of the privilege display is explained under GRANT.

`\drds [ role-pattern [ database-pattern ] ]`

Lists defined configuration settings. These settings can be role-specific, database-specific, or both. *role-pattern* and *database-pattern* are used to select specific roles and databases to list, respectively. If omitted, or if `*` is specified, all settings are listed, including those not role-specific or database-specific, respectively.

The ALTER ROLE and ALTER DATABASE commands are used to define per-role and per-database configuration settings.

`\dT[S+] [ pattern ]`

Lists data types. If *pattern* is specified, only types whose names match the pattern are listed. If `+` is appended to the command name, each type is listed with its internal name and size, its allowed values if it is an `enum` type, and its associated permissions. By default, only user-created objects are shown; supply a pattern or the `S` modifier to include system objects.

`\du[+] [ pattern ]`

Lists database roles. (Since the concepts of "users" and "groups" have been unified into "roles", this command is now equivalent to `\dg`.) If *pattern* is specified, only those roles whose names match the pattern are listed. If the form `\du+` is used, additional information is shown about each role; currently this adds the comment for each role.

`\dx[+] [ pattern ]`

Lists installed extensions. If *pattern* is specified, only those extensions whose names match the pattern are listed. If the form `\dx+` is used, all the objects belonging to each matching extension are listed.

`\e` or `\edit [ filename ] [ line_number ]`

If *filename* is specified, the file is edited; after the editor exits, its content is copied back to the query buffer. If no *filename* is given, the current query buffer is copied to a temporary file which is then edited in the same fashion.

The new query buffer is then re-parsed according to the normal rules of psql, where the whole buffer is treated as a single line. (Thus you cannot make scripts this way. Use `\i` for that.) This means that if the query ends with (or contains) a semicolon, it is immediately executed. Otherwise it will merely wait in the query buffer; type semicolon or `\g` to send it, or `\r` to cancel.

If a line number is specified, psql will position the cursor on the specified line of the file or query buffer. Note that if a single all-digits argument is given, psql assumes it is a line number, not a file name.

> **提示**
>
> See under Environment for how to configure and customize your editor.

`\echo text [ ... ]`

Prints the arguments to the standard output, separated by one space and followed by a newline. This can be useful to intersperse information in the output of scripts. For example:

```
=> \echo `date`
Tue Oct 26 21:40:57 CEST 1999
```

If the first argument is an unquoted `-n` the trailing newline is not written.

> **提示**
>
> If you use the `\o` command to redirect your query output you might wish to use `\qecho` instead of this command.

`\ef [ function_description [ line_number ] ]`

This command fetches and edits the definition of the named function, in the form of a `CREATE OR REPLACE FUNCTION` command. Editing is done in the same way as for `\edit`. After the editor exits, the updated command waits in the query buffer; type semicolon or `\g` to send it, or `\r` to cancel.

The target function can be specified by name alone, or by name and arguments, for example `foo(integer, text)`. The argument types must be given if there is more than one function of the same name.

If no function is specified, a blank `CREATE FUNCTION` template is presented for editing.

If a line number is specified, psql will position the cursor on the specified line of the function body. (Note that the function body typically does not begin on the first line of the file.)

> **提示**
>
> See under Environment for how to configure and customize your editor.

`\encoding [ encoding ]`

Sets the client character set encoding. Without an argument, this command shows the current encoding.

`\f [ string ]`

Sets the field separator for unaligned query output. The default is the vertical bar (`|`). See also `\pset` for a generic way of setting output options.

`\g [ filename ]`
`\g [ |command ]`

Sends the current query input buffer to the server, and optionally stores the query's output in *filename* or pipes the output to the shell command *command*. A bare `\g` is virtually equivalent to a semicolon. A `\g` with argument is a "one-shot" alternative to the `\o` command.

`\h` or `\help [ command ]`

Gives syntax help on the specified SQL command. If *command* is not specified, then psql will list all the commands for which syntax help is available. If *command* is an asterisk (`*`), then syntax help on all SQL commands is shown.

> **注意**
>
> To simplify typing, commands that consists of several words do not have to be quoted. Thus it is fine to type **`\help alter table`**.

`\H` or `\html`

Turns on HTML query output format. If the HTML format is already on, it is switched back to the default aligned text format. This command is for compatibility and convenience, but see `\pset` about setting other output options.

`\i` or `\include` *`filename`*

Reads input from the file *`filename`* and executes it as though it had been typed on the keyboard.

If *`filename`* is – (hyphen), then standard input is read until an EOF indication or `\q` meta-command. This can be used to intersperse interactive input with input from files. Note that Readline behavior will be used only if it is active at the outermost level.

> **注意**
>
> If you want to see the lines on the screen as they are read you must set the variable `ECHO` to `all`.

`\ir` or `\include_relative` *`filename`*

The `\ir` command is similar to `\i`, but resolves relative file names differently. When executing in interactive mode, the two commands behave identically. However, when invoked from a script, `\ir` interprets file names relative to the directory in which the script is located, rather than the current working directory.

`\l` (or `\list`)
`\l+` (or `\list+`)

List the names, owners, character set encodings, and access privileges of all the databases in the server. If `+` is appended to the command name, database sizes, default tablespaces, and descriptions are also displayed. (Size information is only available for databases that the current user can connect to.)

`\lo_export` *`loid filename`*

Reads the large object with OID *`loid`* from the database and writes it to *`filename`*. Note that this is subtly different from the server function `lo_export`, which acts with the permissions of the user that the database server runs as and on the server's file system.

> **提示**
>
> Use `\lo_list` to find out the large object's OID.

`\lo_import` *`filename`* `[` *`comment`* `]`

Stores the file into a PostgreSQL large object. Optionally, it associates the given comment with the object. Example:

```
foo=> \lo_import '/home/peter/pictures/photo.xcf' 'a picture of
 me'
lo_import 152801
```

The response indicates that the large object received object ID 152801, which can be used to access the newly-created large object in the future. For the sake of readability, it is recommended to always associate a human-readable comment with every object. Both OIDs and comments can be viewed with the `\lo_list` command.

Note that this command is subtly different from the server-side `lo_import` because it acts as the local user on the local file system, rather than the server's user and file system.

`\lo_list`

Shows a list of all PostgreSQL large objects currently stored in the database, along with any comments provided for them.

`\lo_unlink` *`loid`*

Deletes the large object with OID *`loid`* from the database.

> **提示**
>
> Use `\lo_list` to find out the large object's OID.

`\o` or `\out [` *`filename`* `]`
`\o` or `\out [ |`*`command`* `]`

Arranges to save future query results to the file *`filename`* or pipe future results to the shell command *`command`*. If no argument is specified, the query output is reset to the standard output.

"Query results" includes all tables, command responses, and notices obtained from the database server, as well as output of various backslash commands that query the database (such as `\d`), but not error messages.

> **提示**
>
> To intersperse text output in between query results, use `\qecho`.

`\p` or `\print`

Print the current query buffer to the standard output.

`\password [` *`username`* `]`

Changes the password of the specified user (by default, the current user). This command prompts for the new password, encrypts it, and sends it to the server as an `ALTER ROLE` command. This makes sure that the new password does not appear in cleartext in the command history, the server log, or elsewhere.

`\prompt [` *`text`* `]` *`name`*

Prompts the user to supply text, which is assigned to the variable *`name`*. An optional prompt string, *`text`*, can be specified. (For multiword prompts, surround the text with single quotes.)

By default, `\prompt` uses the terminal for input and output. However, if the `-f` command line switch was used, `\prompt` uses standard input and standard output.

`\pset` *`option`* `[` *`value`* `]`

This command sets options affecting the output of query result tables. *`option`* indicates which option is to be set. The semantics of *`value`* vary depending on the selected option. For some options, omitting *`value`* causes the option to be toggled or unset, as described under the particular option. If no such behavior is mentioned, then omitting *`value`* just results in the current setting being displayed.

Adjustable printing options are:

`border`

The *`value`* must be a number. In general, the higher the number the more borders and lines the tables will have, but this depends on the particular format. In HTML format, this will translate directly into the `border=...` attribute; in the other formats only values 0 (no border), 1 (internal dividing lines), and 2 (table frame) make sense.

`columns`

Sets the target width for the `wrapped` format, and also the width limit for determining whether output is wide enough to require the pager or switch to the vertical display in expanded auto mode. Zero (the default) causes the target width to be controlled by the environment variable `COLUMNS`, or the detected screen width if `COLUMNS` is not set. In addition, if `columns` is zero then the `wrapped` format only affects screen output. If `columns` is nonzero then file and pipe output is wrapped to that width as well.

`expanded` (or `x`)

If *`value`* is specified it must be either `on` or `off`, which will enable or disable expanded mode, or `auto`. If *`value`* is omitted the command toggles between the on and off settings. When expanded mode is enabled, query results are displayed in two columns, with the column name on the left and the data on the right. This mode is useful if the data wouldn't fit on the screen in the normal "horizontal" mode. In the auto setting, the expanded mode is used whenever the query output is wider than the screen, otherwise the regular mode is used. The auto setting is only effective in the aligned and wrapped formats. In other formats, it always behaves as if the expanded mode is off.

`fieldsep`

Specifies the field separator to be used in unaligned output format. That way one can create, for example, tab- or comma-separated output, which other programs might prefer. To set a tab as field separator, type `\pset fieldsep '\t'`. The default field separator is `'|'` (a vertical bar).

`fieldsep_zero`

Sets the field separator to use in unaligned output format to a zero byte.

`footer`

If *`value`* is specified it must be either `on` or `off` which will enable or disable display of the table footer (the `(`*`n`*` rows)` count). If *`value`* is omitted the command toggles footer display on or off.

`format`

Sets the output format to one of `unaligned`, `aligned`, `wrapped`, `html`, `latex`, or `troff-ms`. Unique abbreviations are allowed. (That would mean one letter is enough.)

`unaligned` format writes all columns of a row on one line, separated by the currently active field separator. This is useful for creating output that might be intended to be read in by other programs (for example, tab-separated or comma-separated format).

`aligned` format is the standard, human-readable, nicely formatted text output; this is the default.

`wrapped` format is like `aligned` but wraps wide data values across lines to make the output fit in the target column width. The target width is determined as described under the `columns` option. Note that psql will not attempt to wrap column header titles; therefore, `wrapped` format behaves the same as `aligned` if the total width needed for column headers exceeds the target.

The `html`, `latex`, and `troff-ms` formats put out tables that are intended to be included in documents using the respective mark-up language. They are not complete documents! (This might not be so dramatic in HTML, but in LaTeX you must have a complete document wrapper.)

`linestyle`

Sets the border line drawing style to one of `ascii`, `old-ascii` or `unicode`. Unique abbreviations are allowed. (That would mean one letter is enough.) The default setting is `ascii`. This option only affects the `aligned` and `wrapped` output formats.

`ascii` style uses plain ASCII characters. Newlines in data are shown using a `+` symbol in the right-hand margin. When the `wrapped` format wraps data from one line to the next without a newline character, a dot (`.`) is shown in the right-hand margin of the first line, and again in the left-hand margin of the following line.

`old-ascii` style uses plain ASCII characters, using the formatting style used in PostgreSQL 8.4 and earlier. Newlines in data are shown using a `:` symbol in place of the left-hand column separator. When the data is wrapped from one line to the next without a newline character, a `;` symbol is used in place of the left-hand column separator.

`unicode` style uses Unicode box-drawing characters. Newlines in data are shown using a carriage return symbol in the right-hand margin. When the data is wrapped from one line to the next without a newline character, an ellipsis symbol is shown in the right-hand margin of the first line, and again in the left-hand margin of the following line.

When the `border` setting is greater than zero, this option also determines the characters with which the border lines are drawn. Plain ASCII characters work everywhere, but Unicode characters look nicer on displays that recognize them.

`null`

Sets the string to be printed in place of a null value. The default is to print nothing, which can easily be mistaken for an empty string. For example, one might prefer `\pset null '(null)'`.

`numericlocale`

If *value* is specified it must be either `on` or `off` which will enable or disable display of a locale-specific character to separate groups of digits to the left of the decimal marker. If *value* is omitted the command toggles between regular and locale-specific numeric output.

`pager`

Controls use of a pager program for query and psql help output. If the environment variable `PAGER` is set, the output is piped to the specified program. Otherwise a platform-dependent default (such as `more`) is used.

When the `pager` option is `off`, the pager program is not used. When the `pager` option is `on`, the pager is used when appropriate, i.e., when the output is to a terminal and will not fit on the screen. The `pager` option can also be set to `always`, which causes the pager to be used for all terminal output regardless of whether it fits on the screen. `\pset pager` without a *value* toggles pager use on and off.

`recordsep`

Specifies the record (line) separator to use in unaligned output format. The default is a newline character.

`recordsep_zero`

Sets the record separator to use in unaligned output format to a zero byte.

`tableattr` (or `T`)

Specifies attributes to be placed inside the HTML `table` tag in `html` output format. This could for example be `cellpadding` or `bgcolor`. Note that you probably don't want to specify `border` here, as that is already taken care of by `\pset border`. If no *value* is given, the table attributes are unset.

`title`

Sets the table title for any subsequently printed tables. This can be used to give your output descriptive tags. If no *value* is given, the title is unset.

`tuples_only` (or `t`)

If *value* is specified it must be either `on` or `off` which will enable or disable tuples-only mode. If *value* is omitted the command toggles between regular and tuples-only output. Regular output includes extra information such as column headers, titles, and various footers. In tuples-only mode, only actual table data is shown.

Illustrations of how these different formats look can be seen in the Examples section.

提示

There are various shortcut commands for `\pset`. See `\a`, `\C`, `\H`, `\t`, `\T`, and `\x`.

> **注意**
>
> It is an error to call `\pset` without any arguments. In the future this case might show the current status of all printing options.

`\q` or `\quit`

Quits the psql program. In a script file, only execution of that script is terminated.

`\qecho` *`text`* `[ ... ]`

This command is identical to `\echo` except that the output will be written to the query output channel, as set by `\o`.

`\r` or `\reset`

Resets (clears) the query buffer.

`\s [` *`filename`* `]`

Print psql's command line history to *`filename`*. If *`filename`* is omitted, the history is written to the standard output (using the pager if appropriate). This command is not available if psql was built without Readline support.

`\set [` *`name`* `[` *`value`* `[ ... ] ] ]`

Sets the psql variable *`name`* to *`value`*, or if more than one value is given, to the concatenation of all of them. If only one argument is given, the variable is set with an empty value. To unset a variable, use the `\unset` command.

`\set` without any arguments displays the names and values of all currently-set psql variables.

Valid variable names can contain letters, digits, and underscores. See the section Variables below for details. Variable names are case-sensitive.

Although you are welcome to set any variable to anything you want, psql treats several variables as special. They are documented in the section about variables.

> **注意**
>
> This command is unrelated to the SQL command SET.

`\setenv` *`name`* `[` *`value`* `]`

Sets the environment variable *`name`* to *`value`*, or if the *`value`* is not supplied, unsets the environment variable. Example:

```
testdb=> \setenv PAGER less
testdb=> \setenv LESS -imx4F
```

`\sf[+]` *`function_description`*

This command fetches and shows the definition of the named function, in the form of a `CREATE OR REPLACE FUNCTION` command. The definition is printed to the current query output channel, as set by `\o`.

The target function can be specified by name alone, or by name and arguments, for example `foo(integer, text)`. The argument types must be given if there is more than one function of the same name.

If + is appended to the command name, then the output lines are numbered, with the first line of the function body being line 1.

`\t`

Toggles the display of output column name headings and row count footer. This command is equivalent to `\pset tuples_only` and is provided for convenience.

`\T` *`table_options`*

Specifies attributes to be placed within the `table` tag in HTML output format. This command is equivalent to `\pset tableattr` *`table_options`*.

`\timing [` *`on`* `|` *`off`* `]`

Without parameter, toggles a display of how long each SQL statement takes, in milliseconds. With parameter, sets same.

`\unset` *`name`*

Unsets (deletes) the psql variable *`name`*.

`\w` or `\write` *`filename`*
`\w` or `\write |`*`command`*

Outputs the current query buffer to the file *`filename`* or pipes it to the shell command *`command`*.

`\x [` *`on`* `|` *`off`* `|` *`auto`* `]`

Sets or toggles expanded table formatting mode. As such it is equivalent to `\pset expanded`.

`\z [` *`pattern`* `]`

Lists tables, views and sequences with their associated access privileges. If a *`pattern`* is specified, only tables, views and sequences whose names match the pattern are listed.

This is an alias for `\dp` (“display privileges”).

`\! [` *`command`* `]`

Escapes to a separate shell or executes the shell command *`command`*. The arguments are not further interpreted; the shell will see them as-is.

`\?`

Shows help information about the backslash commands.

Patterns

The various `\d` commands accept a *`pattern`* parameter to specify the object name(s) to be displayed. In the simplest case, a pattern is just the exact name of the object. The characters within a pattern are normally folded to lower case, just as in SQL names; for example, `\dt FOO` will display the table named `foo`. As in SQL names, placing double quotes around a pattern stops folding to lower case. Should you need to include an actual double quote character in a pattern, write it as a pair of double

quotes within a double-quote sequence; again this is in accord with the rules for SQL quoted identifiers. For example, `\dt "FOO""BAR"` will display the table named `FOO"BAR` (not `foo"bar`). Unlike the normal rules for SQL names, you can put double quotes around just part of a pattern, for instance `\dt FOO"FOO"BAR` will display the table named `fooFOObar`.

Whenever the *`pattern`* parameter is omitted completely, the `\d` commands display all objects that are visible in the current schema search path – this is equivalent to using `*` as the pattern. (An object is said to be visible if its containing schema is in the search path and no object of the same kind and name appears earlier in the search path. This is equivalent to the statement that the object can be referenced by name without explicit schema qualification.) To see all objects in the database regardless of visibility, use `*.*` as the pattern.

Within a pattern, `*` matches any sequence of characters (including no characters) and `?` matches any single character. (This notation is comparable to Unix shell file name patterns.) For example, `\dt int*` displays tables whose names begin with `int`. But within double quotes, `*` and `?` lose these special meanings and are just matched literally.

A pattern that contains a dot (`.`) is interpreted as a schema name pattern followed by an object name pattern. For example, `\dt foo*.*bar*` displays all tables whose table name includes `bar` that are in schemas whose schema name starts with `foo`. When no dot appears, then the pattern matches only objects that are visible in the current schema search path. Again, a dot within double quotes loses its special meaning and is matched literally.

Advanced users can use regular-expression notations such as character classes, for example `[0-9]` to match any digit. All regular expression special characters work as specified in 第 9.7.3 节, except for `.` which is taken as a separator as mentioned above, `*` which is translated to the regular-expression notation `.*`, `?` which is translated to `.`, and `$` which is matched literally. You can emulate these pattern characters at need by writing `?` for `.`, `(R+|)` for `R*`, or `(R|)` for `R?`. `$` is not needed as a regular-expression character since the pattern must match the whole name, unlike the usual interpretation of regular expressions (in other words, `$` is automatically appended to your pattern). Write `*` at the beginning and/or end if you don't wish the pattern to be anchored. Note that within double quotes, all regular expression special characters lose their special meanings and are matched literally. Also, the regular expression special characters are matched literally in operator name patterns (i.e., the argument of `\do`).

Advanced Features

Variables

psql provides variable substitution features similar to common Unix command shells. Variables are simply name/value pairs, where the value can be any string of any length. The name must consist of letters (including non-Latin letters), digits, and underscores.

To set a variable, use the psql meta-command `\set`. For example,

```
testdb=> \set foo bar
```

sets the variable `foo` to the value `bar`. To retrieve the content of the variable, precede the name with a colon, for example:

```
testdb=> \echo :foo
bar
```

This works in both regular SQL commands and meta-commands; there is more detail in SQL Interpolation, below.

If you call `\set` without a second argument, the variable is set, with an empty string as value. To unset (i.e., delete) a variable, use the command `\unset`. To show the values of all variables, call `\set` without any argument.

> **注意**
>
> The arguments of `\set` are subject to the same substitution rules as with other commands. Thus you can construct interesting references such as `\set :foo 'something'` and get "soft links" or "variable variables" of Perl or PHP fame, respectively. Unfortunately (or fortunately?), there is no way to do anything useful with these constructs. On the other hand, `\set bar :foo` is a perfectly valid way to copy a variable.

A number of these variables are treated specially by psql. They represent certain option settings that can be changed at run time by altering the value of the variable, or in some cases represent changeable state of psql. Although you can use these variables for other purposes, this is not recommended, as the program behavior might grow really strange really quickly. By convention, all specially treated variables' names consist of all upper-case ASCII letters (and possibly digits and underscores). To ensure maximum compatibility in the future, avoid using such variable names for your own purposes. A list of all specially treated variables follows.

`AUTOCOMMIT`

When `on` (the default), each SQL command is automatically committed upon successful completion. To postpone commit in this mode, you must enter a `BEGIN` or `START TRANSACTION` SQL command. When `off` or unset, SQL commands are not committed until you explicitly issue `COMMIT` or `END`. The autocommit-off mode works by issuing an implicit `BEGIN` for you, just before any command that is not already in a transaction block and is not itself a `BEGIN` or other transaction-control command, nor a command that cannot be executed inside a transaction block (such as `VACUUM`).

> **注意**
>
> In autocommit-off mode, you must explicitly abandon any failed transaction by entering `ABORT` or `ROLLBACK`. Also keep in mind that if you exit the session without committing, your work will be lost.

> **注意**
>
> The autocommit-on mode is PostgreSQL's traditional behavior, but autocommit-off is closer to the SQL spec. If you prefer autocommit-off, you might wish to set it in the system-wide `psqlrc` file or your `~/.psqlrc` file.

`COMP_KEYWORD_CASE`

Determines which letter case to use when completing an SQL key word. If set to `lower` or `upper`, the completed word will be in lower or upper case, respectively.

If set to `preserve-lower` or `preserve-upper` (the default), the completed word will be in the case of the word already entered, but words being completed without anything entered will be in lower or upper case, respectively.

`DBNAME`

The name of the database you are currently connected to. This is set every time you connect to a database (including program start-up), but can be unset.

`ECHO`

If set to `all`, all nonempty input lines are printed to standard output as they are read. (This does not apply to lines read interactively.) To select this behavior on program start-up, use the switch `-a`. If set to `queries`, psql prints each query to standard output as it is sent to the server. The switch for this is `-e`.

`ECHO_HIDDEN`

When this variable is set to `on` and a backslash command queries the database, the query is first shown. This feature helps you to study PostgreSQL internals and provide similar functionality in your own programs. (To select this behavior on program start-up, use the switch `-E`.) If you set the variable to the value `noexec`, the queries are just shown but are not actually sent to the server and executed.

`ENCODING`

The current client character set encoding.

`FETCH_COUNT`

If this variable is set to an integer value > 0, the results of `SELECT` queries are fetched and displayed in groups of that many rows, rather than the default behavior of collecting the entire result set before display. Therefore only a limited amount of memory is used, regardless of the size of the result set. Settings of 100 to 1000 are commonly used when enabling this feature. Keep in mind that when using this feature, a query might fail after having already displayed some rows.

> **提示**
>
> Although you can use any output format with this feature, the default `aligned` format tends to look bad because each group of `FETCH_COUNT` rows will be formatted separately, leading to varying column widths across the row groups. The other output formats work better.

`HISTCONTROL`

If this variable is set to `ignorespace`, lines which begin with a space are not entered into the history list. If set to a value of `ignoredups`, lines matching the previous history line are not entered. A value of `ignoreboth` combines the two options. If unset, or if set to any other value than those above, all lines read in interactive mode are saved on the history list.

> **注意**
>
> This feature was shamelessly plagiarized from Bash.

`HISTFILE`

The file name that will be used to store the history list. The default value is `~/.psql_history`. For example, putting:

```
\set HISTFILE ~/.psql_history- :DBNAME
```

in `~/.psqlrc` will cause psql to maintain a separate history for each database.

> **注意**
>
> This feature was shamelessly plagiarized from Bash.

`HISTSIZE`

The number of commands to store in the command history. The default value is 500.

> **注意**
>
> This feature was shamelessly plagiarized from Bash.

`HOST`

The database server host you are currently connected to. This is set every time you connect to a database (including program start-up), but can be unset.

`IGNOREEOF`

If unset, sending an EOF character (usually **Control+D**) to an interactive session of psql will terminate the application. If set to a numeric value, that many EOF characters are ignored before the application terminates. If the variable is set but has no numeric value, the default is 10.

> **注意**
>
> This feature was shamelessly plagiarized from Bash.

`LASTOID`

The value of the last affected OID, as returned from an `INSERT` or `\lo_import` command. This variable is only guaranteed to be valid until after the result of the next SQL command has been displayed.

`ON_ERROR_ROLLBACK`

When set to `on`, if a statement in a transaction block generates an error, the error is ignored and the transaction continues. When set to `interactive`, such errors are only ignored in interactive sessions, and not when reading script files. When unset or set to `off`, a statement in a transaction block that generates an error aborts the entire transaction. The error rollback mode works by issuing an implicit `SAVEPOINT` for you, just before each command that is in a transaction block, and then rolling back to the savepoint if the command fails.

`ON_ERROR_STOP`

By default, command processing continues after an error. When this variable is set to `on`, processing will instead stop immediately. In interactive mode, psql will return to the command prompt; otherwise, psql will exit, returning error code 3 to distinguish this case from fatal error conditions, which are reported using error code 1. In either case, any currently running scripts (the top-level script, if any, and any other scripts which it may have in invoked) will be terminated immediately. If the top-level command string contained multiple SQL commands, processing will stop with the current command.

`PORT`

The database server port to which you are currently connected. This is set every time you connect to a database (including program start-up), but can be unset.

`PROMPT1`
`PROMPT2`
`PROMPT3`

These specify what the prompts psql issues should look like. See Prompting below.

`QUIET`

Setting this variable to `on` is equivalent to the command line option `-q`. It is probably not too useful in interactive mode.

`SINGLELINE`

Setting this variable to `on` is equivalent to the command line option `-S`.

`SINGLESTEP`

Setting this variable to `on` is equivalent to the command line option `-s`.

`USER`

The database user you are currently connected as. This is set every time you connect to a database (including program start-up), but can be unset.

`VERBOSITY`

This variable can be set to the values `default`, `verbose`, or `terse` to control the verbosity of error reports.

SQL Interpolation

A key feature of psql variables is that you can substitute (“interpolate”) them into regular SQL statements, as well as the arguments of meta-commands. Furthermore, psql provides facilities for ensuring that variable values used as SQL literals and identifiers are properly quoted. The syntax for interpolating a value without any quoting is to prepend the variable name with a colon (`:`). For example,

```
testdb=> \set foo 'my_table'
testdb=> SELECT * FROM :foo;
```

would query the table `my_table`. Note that this may be unsafe: the value of the variable is copied literally, so it can contain unbalanced quotes, or even backslash commands. You must make sure that it makes sense where you put it.

When a value is to be used as an SQL literal or identifier, it is safest to arrange for it to be quoted. To quote the value of a variable as an SQL literal, write a colon followed by the variable name in single quotes. To quote the value as an SQL identifier, write a colon followed by the variable name in double quotes. These constructs deal correctly with quotes and other special characters embedded within the variable value. The previous example would be more safely written this way:

```
testdb=> \set foo 'my_table'
testdb=> SELECT * FROM :"foo";
```

Variable interpolation will not be performed within quoted SQL literals and identifiers. Therefore, a construction such as `':foo'` doesn't work to produce a quoted literal from a variable's value (and it would be unsafe if it did work, since it wouldn't correctly handle quotes embedded in the value).

One example use of this mechanism is to copy the contents of a file into a table column. First load the file into a variable and then interpolate the variable's value as a quoted string:

```
testdb=> \set content `cat my_file.txt`
testdb=> INSERT INTO my_table VALUES (:'content');
```

(Note that this still won't work if `my_file.txt` contains NUL bytes. psql does not support embedded NUL bytes in variable values.)

Since colons can legally appear in SQL commands, an apparent attempt at interpolation (that is, `:name`, `:'name'`, or `:"name"`) is not replaced unless the named variable is currently set. In any case, you can escape a colon with a backslash to protect it from substitution.

The colon syntax for variables is standard SQL for embedded query languages, such as ECPG. The colon syntaxes for array slices and type casts are PostgreSQL extensions, which can sometimes conflict with the standard usage. The colon-quote syntax for escaping a variable's value as an SQL literal or identifier is a psql extension.

Prompting

The prompts psql issues can be customized to your preference. The three variables `PROMPT1`, `PROMPT2`, and `PROMPT3` contain strings and special escape sequences that describe the appearance of the prompt. Prompt 1 is the normal prompt that is issued when psql requests a new command. Prompt 2 is issued when more input is expected during command entry, for example because the command was not terminated with a semicolon or a quote was not closed. Prompt 3 is issued when you are running an SQL `COPY FROM STDIN` command and you need to type in a row value on the terminal.

The value of the selected prompt variable is printed literally, except where a percent sign (`%`) is encountered. Depending on the next character, certain other text is substituted instead. Defined substitutions are:

`%M`

The full host name (with domain name) of the database server, or `[local]` if the connection is over a Unix domain socket, or `[local:`*`/dir/name`*`]`, if the Unix domain socket is not at the compiled in default location.

`%m`

The host name of the database server, truncated at the first dot, or `[local]` if the connection is over a Unix domain socket.

`%>`

The port number at which the database server is listening.

`%n`

The database session user name. (The expansion of this value might change during a database session as the result of the command `SET SESSION AUTHORIZATION`.)

`%/`

The name of the current database.

`%~`

Like `%/`, but the output is `~` (tilde) if the database is your default database.

`%#`

If the session user is a database superuser, then a `#`, otherwise a `>`. (The expansion of this value might change during a database session as the result of the command `SET SESSION AUTHORIZATION`.)

`%R`

In prompt 1 normally `=`, but `^` if in single-line mode, or `!` if the session is disconnected from the database (which can happen if `\connect` fails). In prompt 2 `%R` is replaced by a character that depends on why psql expects more input: – if the command simply wasn't terminated yet, but `*` if there is an unfinished `/* ... */` comment, a single quote if there is an unfinished quoted string, a double quote if there is an unfinished quoted identifier, a dollar sign if there is an unfinished dollar-quoted string, or `(` if there is an unmatched left parenthesis. In prompt 3 `%R` doesn't produce anything.

`%x`

Transaction status: an empty string when not in a transaction block, or `*` when in a transaction block, or `!` when in a failed transaction block, or `?` when the transaction state is indeterminate (for example, because there is no connection).

`%`*`digits`*

The character with the indicated octal code is substituted.

`%:`*`name`*`:`

The value of the psql variable *`name`*. See the section Variables for details.

`%` ` *`command`* `

The output of *`command`*, similar to ordinary "back-tick" substitution.

`%[ ... %]`

Prompts can contain terminal control characters which, for example, change the color, background, or style of the prompt text, or change the title of the terminal window. In order for the line editing features of Readline to work properly, these non-printing control characters must be designated as invisible by surrounding them with `%[` and `%]`. Multiple pairs of these can occur within the prompt. For example:

```
testdb=> \set PROMPT1 '%[%033[1;33;40m%]%n@%/%R%[%033[0m%]%# '
```

results in a boldfaced (`1;`) yellow-on-black (`33;40`) prompt on VT100-compatible, color-capable terminals.

To insert a percent sign into your prompt, write `%%`. The default prompts are `'%/%R%# '` for prompts 1 and 2, and `'>> '` for prompt 3.

> **注意**
>
> This feature was shamelessly plagiarized from tcsh.

Command-Line Editing

psql supports the Readline library for convenient line editing and retrieval. The command history is automatically saved when psql exits and is reloaded when psql starts up. Tab-completion is also supported, although the completion logic makes no claim to be an SQL parser. If for some reason you do not like the tab completion, you can turn it off by putting this in a file named `.inputrc` in your home directory:

```
$if psql
set disable-completion on
$endif
```

(This is not a psql but a Readline feature. Read its documentation for further details.)

Environment

`COLUMNS`

If `\pset columns` is zero, controls the width for the `wrapped` format and width for determining if wide output requires the pager or should be switched to the vertical format in expanded auto mode.

`PAGER`

If the query results do not fit on the screen, they are piped through this command. Typical values are `more` or `less`. The default is platform-dependent. Use of the pager can be disabled by setting `PAGER` to empty, or by using pager-related options of the `\pset` command.

`PGDATABASE`
`PGHOST`
`PGPORT`
`PGUSER`

Default connection parameters (see 第 31.14 节).

`PSQL_EDITOR`
`EDITOR`
`VISUAL`

Editor used by the `\e` and `\ef` commands. The variables are examined in the order listed; the first that is set is used.

The built-in default editors are `vi` on Unix systems and `notepad.exe` on Windows systems.

`PSQL_EDITOR_LINENUMBER_ARG`

When `\e` or `\ef` is used with a line number argument, this variable specifies the command-line argument used to pass the starting line number to the user's editor. For editors such as Emacs or vi, this is a plus sign. Include a trailing space in the value of the variable if there needs to be space between the option name and the line number. Examples:

```
PSQL_EDITOR_LINENUMBER_ARG='+'
PSQL_EDITOR_LINENUMBER_ARG='--line '
```

The default is `+` on Unix systems (corresponding to the default editor `vi`, and useful for many other common editors); but there is no default on Windows systems.

`PSQL_HISTORY`

Alternative location for the command history file. Tilde (`~`) expansion is performed.

`PSQLRC`

Alternative location of the user's `.psqlrc` file. Tilde (`~`) expansion is performed.

`SHELL`

Command executed by the `\!` command.

`TMPDIR`

Directory for storing temporary files. The default is `/tmp`.

This utility, like most other PostgreSQL utilities, also uses the environment variables supported by libpq (see 第 31.14 节).

Files

`psqlrc` and `~/.psqlrc`

Unless it is passed an `-X` or `-c` option, psql attempts to read and execute commands from the system-wide startup file (`psqlrc`) and then the user's personal startup file (`~/.psqlrc`), after connecting to the database but before accepting normal commands. These files can be used to set up the client and/or the server to taste, typically with `\set` and `SET` commands.

The system-wide startup file is named `psqlrc` and is sought in the installation's "system configuration" directory, which is most reliably identified by running `pg_config --sysconfdir`. By default this directory will be `../etc/` relative to the directory containing the PostgreSQL executables. The name of this directory can be set explicitly via the `PGSYSCONFDIR` environment variable.

The user's personal startup file is named `.psqlrc` and is sought in the invoking user's home directory. On Windows, which lacks such a concept, the personal startup file is named `%APPDATA%\postgresql\psqlrc.conf`. The location of the user's startup file can be set explicitly via the `PSQLRC` environment variable.

Both the system-wide startup file and the user's personal startup file can be made psql-version-specific by appending a dash and the PostgreSQL major or minor release number to the file name, for example `~/.psqlrc-9.2` or `~/.psqlrc-9.2.5`. The most specific version-matching file will be read in preference to a non-version-specific file.

`.psql_history`

The command-line history is stored in the file `~/.psql_history`, or `%APPDATA%\postgresql\psql_history` on Windows.

The location of the history file can be set explicitly via the `PSQL_HISTORY` environment variable.

Notes

- In an earlier life psql allowed the first argument of a single-letter backslash command to start directly after the command, without intervening whitespace. As of PostgreSQL 8.4 this is no longer allowed.
- psql is only guaranteed to work smoothly with servers of the same version. That does not mean other combinations will fail outright, but subtle and not-so-subtle problems might come up. Backslash commands are particularly likely to fail if the server is of a newer version than psql itself. However, backslash commands of the `\d` family should work with servers of versions back to 7.4, though not necessarily with servers newer than psql itself.

Notes for Windows Users

psql is built as a "console application". Since the Windows console windows use a different encoding than the rest of the system, you must take special care when using 8-bit characters within psql. If psql detects a problematic console code page, it will warn you at startup. To change the console code page, two things are necessary:

- Set the code page by entering **`cmd.exe /c chcp 1252`**. (1252 is a code page that is appropriate for German; replace it with your value.) If you are using Cygwin, you can put this command in `/etc/profile`.
- Set the console font to `Lucida Console`, because the raster font does not work with the ANSI code page.

Examples

The first example shows how to spread a command over several lines of input. Notice the changing prompt:

```
testdb=> CREATE TABLE my_table (
testdb(>  first integer not null default 0,
testdb(>  second text)
testdb-> ;
CREATE TABLE
```

Now look at the table definition again:

```
testdb=> \d my_table
             Table "my_table"
 Attribute |  Type   |      Modifier
-----------+---------+--------------------
 first     | integer | not null default 0
 second    | text    |
```

Now we change the prompt to something more interesting:

```
testdb=> \set PROMPT1 '%n@%m %~%R%# '
```

```
peter@localhost testdb=>
```

Let's assume you have filled the table with data and want to take a look at it:

```
peter@localhost testdb=> SELECT * FROM my_table;
 first | second
-------+--------
     1 | one
     2 | two
     3 | three
     4 | four
(4 rows)
```

You can display tables in different ways by using the `\pset` command:

```
peter@localhost testdb=> \pset border 2
Border style is 2.
peter@localhost testdb=> SELECT * FROM my_table;
+-------+--------+
| first | second |
+-------+--------+
|     1 | one    |
|     2 | two    |
|     3 | three  |
|     4 | four   |
+-------+--------+
(4 rows)

peter@localhost testdb=> \pset border 0
Border style is 0.
peter@localhost testdb=> SELECT * FROM my_table;
first second
----- ------
    1 one
    2 two
    3 three
    4 four
(4 rows)

peter@localhost testdb=> \pset border 1
Border style is 1.
peter@localhost testdb=> \pset format unaligned
Output format is unaligned.
peter@localhost testdb=> \pset fieldsep ","
Field separator is ",".
peter@localhost testdb=> \pset tuples_only
Showing only tuples.
peter@localhost testdb=> SELECT second, first FROM my_table;
one,1
two,2
three,3
four,4
```

Alternatively, use the short commands:

```
peter@localhost testdb=> \a \t \x
Output format is aligned.
Tuples only is off.
Expanded display is on.
```

```
peter@localhost testdb=> SELECT * FROM my_table;
-[ RECORD 1 ]-
first  | 1
second | one
-[ RECORD 2 ]-
first  | 2
second | two
-[ RECORD 3 ]-
first  | 3
second | three
-[ RECORD 4 ]-
first  | 4
second | four
```

reindexdb

reindexdb — 重建一个PostgreSQL数据库中的索引

大纲

`reindexdb` [*connection-option*...] [`--table` | `-t` *table*] [`--index` | `-i` *index*] [*dbname*]

`reindexdb` [*connection-option*...] `-a` | `--all`

`reindexdb` [*connection-option*...] `-s` | `--system` [*dbname*]

描述

reindexdb是用于重建PostgreSQL数据库中索引的工具。

reindexdb 是 SQL 命令REINDEX的一个包装器。通过该工具重建数据库索引与通过其他方法访问服务器来重建索引，在效果上没有区别。

选项

reindexdb接受以下命令行参数：

`-a`
`--all`

重建所有数据库的索引。

[`-d`] *dbname*
[`--dbname=`]*dbname*

指定要重建索引的数据库名称。如果未指定并且没有使用 `-a`（或`--all`），则从环境变量`PGDATABASE`中读取数据库名称。如果该变量未设置，则使用为连接指定的用户名。

`-e`
`--echo`

回显reindexdb生成并发送给服务器的命令。

`-i` *index*
`--index=`*index*

仅重建 *index*。

`-q`
`--quiet`

不显示进度消息。

`-s`
`--system`

重建数据库的系统目录上的索引。

`-t` *table*
`--table=`*table*

仅重建以下对象的索引：*table*。

`-V`
`--version`

打印reindexdb的版本并退出。

`-?`
`--help`

显示关于reindexdb命令行参数的帮助信息，并退出。

reindexdb还接受以下用于连接参数的命令行参数：

`-h` *`host`*
`--host=`*`host`*

指定服务器所在机器的主机名。如果该值以斜杠开头，则它将被用作 Unix 域套接字的目录。

`-p` *`port`*
`--port=`*`port`*

指定服务器监听连接所用的 TCP 端口或本地 Unix 域套接字文件扩展名。

`-U` *`username`*
`--username=`*`username`*

要作为其身份连接的用户名。

`-w`
`--no-password`

永远不发出密码提示。如果服务器要求密码认证，而又无法通过其他方式获取密码，例如通过`.pgpass`文件，则连接尝试会失败。
此选项可用于没有用户可输入密码的批处理作业和脚本。

`-W`
`--password`

强制reindexdb在连接数据库前提示输入密码。

该选项并非必需，因为如果服务器要求密码认证，reindexdb会自动提示输入密码。不过，reindexdb会先浪费一次连接尝试来确认服务器需要密码。在某些场景下，值得使用 `-W`，以避免这次额外的连接尝试。

`--maintenance-db=`*`dbname`*

连接到该数据库，以发现还有哪些其他数据库需要重建索引。如果未指定，将使用`postgres`数据库；如果它不存在，则使用`template1`。

环境

`PGDATABASE`
`PGHOST`
`PGPORT`
`PGUSER`

默认连接参数

与大多数其他PostgreSQL工具一样，该工具也使用libpq支持的环境变量（见第 31. 14 节）。

诊断

如果遇到困难，请参阅REINDEX和psql中关于潜在问题和错误消息的讨论。
数据库服务器必须运行在目标主机上。此外，
libpq前端库所使用的任何默认连接设置和环境变量也都会生效。

注解

reindexdb 可能需要多次连接到 PostgreSQL 服务器，每次都要求输入密码。在这种情况下，使用 `~/.pgpass` 文件会很方便。更多信息见第 31.15 节。

示例

要重建数据库`test`中的索引：

```
$ reindexdb test
```

要重建名为`abcd`的数据库中表`foo`的索引和索引`bar`：

```
$ reindexdb --table foo --index bar abcd
```

另见

REINDEX

vacuumdb

vacuumdb — 清理并分析一个 PostgreSQL 数据库

大纲

vacuumdb [*connection-option*...] [*option*...] [--table | -t *table* [(*column* [,...])]] [*dbname*]

vacuumdb [*connection-option*...] [*option*...] -a | --all

描述

vacuumdb 是用于清理 PostgreSQL 数据库的工具。vacuumdb 还会生成供 PostgreSQL 查询优化器使用的内部统计信息。

vacuumdb是 SQL 命令VACUUM的包装器。通过该工具或通过访问服务器的其他方式来清理和分析数据库，在效果上没有区别。

选项

vacuumdb接受下列命令行参数：

-a
--all

清理所有数据库。

[-d] *dbname*
[--dbname=]*dbname*

指定要清理或分析的数据库名称。如果未指定并且没有使用 -a（或--all），则从环境变量 PGDATABASE 中读取数据库名。如果该变量未设置，则使用连接时指定的用户名。

-e
--echo

回显 vacuumdb 生成并发送给服务器的命令。

-f
--full

执行“完全”清理。

-F
--freeze

激进地“冻结”元组。

-q
--quiet

不显示进度消息。

-t *table* [(*column* [,...])]
--table=*table* [(*column* [,...])]

仅清理或分析 *table*。只有结合 --analyze 或 --analyze-only 选项时才能指定列名。

> **提示**
>
> 如果指定了列，你很可能需要在 shell 中转义括号。（见下面的示例。）

`-v`
`--verbose`

在处理过程中输出详细信息。

`-V`
`--version`

输出 vacuumdb 的版本并退出。

`-z`
`--analyze`

同时计算供优化器使用的统计信息。

`-Z`
`--analyze-only`

仅计算供优化器使用的统计信息（不执行清理）。

`-?`
`--help`

显示 vacuumdb 命令行参数的帮助并退出。

vacuumdb还接受下列用于连接参数的命令行参数：

`-h` *`host`*
`--host=`*`host`*

指定服务器所在机器的主机名。如果该值以斜杠开头，则它会被用作 Unix 域套接字目录。

`-p` *`port`*
`--port=`*`port`*

指定服务器监听连接所用的 TCP 端口，或本地 Unix 域套接字文件扩展名。

`-U` *`username`*
`--username=`*`username`*

连接时使用的用户名。

`-w`
`--no-password`

绝不提示输入密码。如果服务器要求密码认证，而又无法通过其他方式获得密码，例如 `.pgpass` 文件，则连接尝试将失败。此选项可用于无人输入密码的批处理作业和脚本。

`-W`
`--password`

强制 vacuumdb 在连接数据库之前提示输入密码。

这个选项绝非必需，因为如果服务器要求密码认证，vacuumdb 会自动提示输入密码。不过，vacuumdb 会先浪费一次连接尝试，才得知服务器需要密码。在某些情况下，输入 `-W` 以避免这次额外的连接尝试是值得的。

`--maintenance-db=`*`dbname`*

连接到该数据库，以发现还有哪些其他数据库需要清理。如果未指定，将使用 `postgres` 数据库；若该数据库不存在，则使用 `template1`。

环境

`PGDATABASE`
`PGHOST`
`PGPORT`
`PGUSER`

默认连接参数

与大多数其他 PostgreSQL 工具一样，此工具也使用 libpq 支持的环境变量（见第 31.14 节）。

诊断

如遇困难，请参见VACUUM和psql中关于潜在问题和错误消息的讨论。
数据库服务器必须在目标主机上运行。此外，libpq 前端库所使用的任何默认连接设置和环境变量也都会生效。

注解

vacuumdb可能需要多次连接到PostgreSQL服务器，每次都要询问密码。在这种情况下，使用`~/.pgpass`文件会比较方便。更多信息见第 31.15 节。

示例

清理数据库 `test`：

```
$ vacuumdb test
```

清理并为优化器分析数据库 `bigdb`：

```
$ vacuumdb --analyze bigdb
```

清理单个表 `foo`，它位于数据库 `xyzzy` 中，并为优化器分析该表的单列 `bar`：

```
$ vacuumdb --analyze --verbose --table 'foo(bar)' xyzzy
```

另见

VACUUM

PostgreSQL Server Applications

This part contains reference information for PostgreSQL server applications and support utilities. These commands can only be run usefully on the host where the database server resides. Other utility programs are listed in PostgreSQL Client Applications.

目录

initdb 1448
pg_controldata 1451
pg_ctl 1452
pg_resetxlog 1457
postgres 1459
postmaster 1466

initdb

initdb — 创建一个新的PostgreSQL数据库集簇

大纲

initdb [*option*...] [--pgdata | -D] *directory*

描述

initdb创建一个新的PostgreSQL数据库集簇。
数据库集簇是由单个服务器实例管理的一组数据库。

创建数据库集簇包括创建用于存放数据库数据的目录，
生成共享系统目录表（属于整个集簇而不是某个特定数据库的表），以及创建template1
和postgres数据库。以后创建新数据库时，会复制template1数据库中的所有内容。
（因此，安装在template1中的任何东西都会自动复制到以后创建的每个数据库中。）
postgres数据库是一个默认数据库，供用户、工具程序和第三方应用程序使用。

虽然initdb会尝试创建指定的数据目录，但如果所需数据目录的父目录归 root 所有，它可能没有足够的权限。要在这种环境中初始化，可先由 root 创建一个空的数据目录，然后用chown将该目录的所有权赋予数据库用户账户，再用 su切换为该数据库用户来运行initdb。

initdb必须以将拥有服务器进程的用户身份运行，因为服务器需要访问 initdb创建的文件和目录。由于服务器不能以 root 身份运行，因此也绝不能以 root 身份运行initdb。（实际上它会拒绝这样做。）

initdb会初始化数据库集簇的默认区域设置和字符集编码。字符集编码、排序顺序（LC_COLLATE）和字符集分类（LC_CTYPE，例如大写、小写、数字）
可以在创建数据库时单独设置。initdb为template1数据库确定这些设置，
它们将作为所有其他数据库的默认值。

要更改默认排序顺序或字符集分类，请使用--lc-collate和--lc-ctype选项。使用C
或POSIX之外的排序顺序也会带来性能损失。因此，在运行initdb
时选择正确的区域设置很重要。

其余区域设置类别可以在以后启动服务器时更改。
还可以使用--locale设置所有区域设置类别的默认值，包括排序顺序和字符集分类。
所有服务器区域设置值（lc_*）都可以通过SHOW ALL显示。更多细节见第 22.1 节。

要修改默认编码，请使用--encoding。更多细节见第 22.3 节。

选项

-A *authmethod*
--auth=*authmethod*

此选项指定pg_hba.conf中本地用户使用的认证方法（host和local行）。
除非信任系统上的所有本地用户，否则不要使用trust。为便于安装，默认值为trust。

--auth-host=*authmethod*

该选项指定pg_hba.conf中本地用户通过 TCP/IP 连接时（host 行）使用的认证方法。

`--auth-local=authmethod`

该选项指定pg_hba.conf中本地用户通过 Unix 域套接字连接时（local 行）使用的认证方法。

`-D directory`
`--pgdata=directory`

该选项指定数据库集簇应存放的目录。这是initdb所需的唯一信息，但也可以通过设置PGDATA环境变量来省去显式写出它；这通常更方便，因为数据库服务器（postgres）之后也可以通过同一变量找到数据目录。

`-E encoding`
`--encoding=encoding`

选择模板数据库的编码。除非后来覆盖，
否则这也将是你以后创建的任何数据库的默认编码。默认值从区域设置导出，
如果不行则为 SQL_ASCII。PostgreSQL服务器支持的字符集在第 22.3.1 节中描述。

`--locale=locale`

设置数据库集簇的默认区域设置。如果未指定该选项，区域设置将继承自 initdb运行时所在的环境。区域设置支持见第 22.1 节。

`--lc-collate=locale`
`--lc-ctype=locale`
`--lc-messages=locale`
`--lc-monetary=locale`
`--lc-numeric=locale`
`--lc-time=locale`

类似于--locale，但只在指定的类别中设置区域设置。

`--no-locale`

等价于--locale=C。

`--pwfile=filename`

使initdb从文件中读取数据库超级用户的密码。文件的第一行会被当作密码。

`-T CFG`
`--text-search-config=CFG`

设置默认文本检索配置。更多信息见default_text_search_config。

`-U username`
`--username=username`

选择数据库超级用户的用户名。默认值是运行initdb的有效用户的名称。
超级用户的名称本身并不重要，不过即使操作系统用户名称不同，
也可以选择沿用惯常的名称 postgres。

`-W`
`--pwprompt`

使initdb提示输入要赋给数据库超级用户的密码。如果不打算使用密码认证，
这一点并不重要。否则，在设置密码之前将无法使用密码认证。

`-X directory`
`--xlogdir=directory`

该选项指定事务日志应存放的目录。

还可以使用以下较少用到的选项：

`-d`
`--debug`

打印引导后端的调试输出，以及少量普通用户通常不感兴趣的其他消息。引导后端是`initdb`用来创建系统目录表的程序。该选项会产生大量极其乏味的输出。

`-L` *`directory`*

指定`initdb`初始化数据库集簇时应到哪里查找其输入文件。通常不需要这样做。若需要显式指定其位置，系统会提示。

`-n`
`--noclean`

默认情况下，如果`initdb`发现某个错误使其无法完整创建数据库集簇，就会删除它在发现无法完成任务之前可能已创建的所有文件。该选项会禁止这种清理，因此对调试有用。

其他选项：

`-V`
`--version`

打印initdb版本并退出。

`-?`
`--help`

显示有关initdb命令行参数的帮助并退出。

环境

`PGDATA`

指定数据库集簇应存放的目录；可使用`-D`选项覆盖。

`TZ`

指定所创建数据库集簇应使用的时区，使用完整的时区名称。

与大多数其他PostgreSQL工具程序一样，此工具也使用libpq支持的环境变量（参见第 31.14 节）。

注解

也可以通过`pg_ctl initdb`调用`initdb`。

参见

pg_ctl, postgres

pg_controldata

pg_controldata — 显示 PostgreSQL 数据库集簇的控制信息

大纲

`pg_controldata` [*option*] [*datadir*]

描述

`pg_controldata`会打印在`initdb`期间初始化的信息，例如系统目录版本。
它还会显示有关预写式日志和检查点处理的信息。这些信息是整个集簇范围内的，
不针对任何单个数据库。

该工具只能由初始化该集簇的用户运行，因为它需要对数据目录具有读取权限。
可以在命令行中指定数据目录，也可以使用环境变量`PGDATA`。
该工具支持`-V`和`--version`选项，用于输出 pg_controldata的版本并退出。它还支持`-?`和`--help`选项，用于输出可用参数。

环境

`PGDATA`

默认数据目录位置

pg_ctl

pg_ctl — 初始化、启动、停止或控制PostgreSQL服务器

大纲

`pg_ctl init[db]` [`-s`] [`-D` *datadir*] [`-o` *initdb-options*]

`pg_ctl start` [`-w`] [`-t` *seconds*] [`-s`] [`-D` *datadir*] [`-l` *filename*] [`-o` *options*] [`-p` *path*] [`-c`]

`pg_ctl stop` [`-W`] [`-t` *seconds*] [`-s`] [`-D` *datadir*] [`-m s[mart]|f[ast]|i[mmediate]`]

`pg_ctl restart` [`-w`] [`-t` *seconds*] [`-s`] [`-D` *datadir*] [`-c`] [`-m s[mart] | f[ast] | i[mmediate]`] [`-o` *options*]

`pg_ctl reload` [`-s`] [`-D` *datadir*]

`pg_ctl status` [`-D` *datadir*]

`pg_ctl promote` [`-s`] [`-D` *datadir*]

`pg_ctl kill` *signal_name process_id*

`pg_ctl register` [`-N` *servicename*] [`-U` *username*] [`-P` *password*] [`-D` *datadir*] [`-S a[uto]|d[emand]`] [`-w`] [`-t` *seconds*] [`-s`] [`-o` *options*]

`pg_ctl unregister` [`-N` *servicename*]

描述

pg_ctl是一个实用工具，用于初始化PostgreSQL数据库集簇，启动、停止或重启PostgreSQL数据库服务器（postgres），或者显示正在运行服务器的状态。虽然服务器也可以手工启动，但pg_ctl将重定向日志输出、正确地与终端和进程组脱离等任务封装了起来。它还提供了便于实施受控关闭的选项。

`init`或`initdb`模式会创建一个新的PostgreSQL数据库集簇，也就是由单个服务器实例管理的一组数据库。该模式会调用`initdb`命令。详见initdb。

`start`模式会启动一个新服务器。服务器在后台启动，其标准输入连接到`/dev/null`（在Windows 上则是`nul`）。在类 Unix 系统上，默认情况下，服务器的标准输出和标准错误会被发送到pg_ctl的标准输出（而不是标准错误）。因此，pg_ctl的标准输出应被重定向到文件，或者通过管道传给另一个进程，例如rotatelogs这样的日志轮转程序；否则`postgres`会在后台将其输出写入控制终端，并且不会脱离 shell 的进程组。在 Windows 上，默认情况下，服务器的标准输出和标准错误会被发送到终端。使用`-l`将服务器输出追加到日志文件，可以改变这些默认行为。建议使用`-l`或输出重定向。

`stop`模式会关闭在指定数据目录中运行的服务器。可以使用`-m`选项选择三种不同的关闭方法。“Smart”模式（默认值）等待所有活动客户端断开，以及所有在线备份结束。如果服务器处于热备状态，那么在所有客户端断开后，恢复和流复制都会终止。“Fast”模式不等待客户端断开，并会终止正在进行的在线备份。所有活动事务都会回滚，客户端会被强制断开，然后服务器关闭。“Immediate”模式会立即中止所有服务器进程，而不执行干净关闭。这样会导致服务器在下次启动时进入一次崩溃恢复周期。

`restart`模式实际上就是先执行停止再执行启动。这使得可以修改`postgres`的命令行选项。

reload模式只是向postgres服务器进程发送一个SIGHUP信号，使其重新读取配置文件（postgresql.conf、pg_hba.conf等）。
这样就可以修改那些无需完全重启服务器即可生效的配置文件选项。

status模式会检查指定数据目录中是否有服务器正在运行。若有，则显示服务器的PID以及调用它时使用的命令行选项。若服务器未运行，进程返回退出状态 3。

在promote模式下，会指示在指定数据目录中运行的备库退出恢复，并开始进行读写操作。

kill模式向指定进程发送信号。这在没有内置kill命令的Microsoft Windows上尤其有用。使用--help可查看受支持的信号名称列表。

register模式允许在Microsoft Windows上注册一个系统服务。-S选项允许选择服务启动类型，可以是“auto”（系统启动时自动启动服务）或“demand”（按需启动服务）。

unregister模式会在Microsoft Windows上注销一个系统服务。这会撤销register命令的效果。

选项

-c
--core-file

在支持的平台上，通过解除对核心转储文件施加的任何软资源限制，
尝试允许服务器在崩溃时生成核心转储文件。
这样就可以从失败的服务器进程中获得栈跟踪，从而有助于调试或诊断问题。

-D *datadir*
--pgdata *datadir*

指定数据库文件所在的文件系统位置。若省略此选项，则使用环境变量PGDATA。

-l *filename*
--log *filename*

将服务器日志输出追加到*filename*。如果该文件不存在，就会创建它。umask被设置为077，因此默认情况下其他用户无法访问该日志文件。

-m *mode*
--mode *mode*

指定关闭模式。*mode*可以是smart、fast或immediate，
也可以是这三者之一的首字母。若省略此选项，则使用smart。

-o *options*

指定要直接传递给postgres命令的选项。

通常应将选项放在单引号或双引号中，以确保它们作为一个整体被传递。

-o *initdb-options*

指定要直接传递给initdb命令的选项。

通常应将选项放在单引号或双引号中，以确保它们作为一个整体被传递。

-p *path*

指定postgres可执行程序的位置。默认情况下，postgres可执行程序取自与pg_ctl相同的目录；如果那里没有，则取自硬编码的安装目录。除非采用了某些非常规方式，并收到找不到postgres可执行程序的错误，否则通常不需要使用此选项。

在init模式中，此选项同样指定initdb可执行程序的位置。

-s
--silent

只打印错误，不打印信息性消息。

-t
--timeout

等待启动或关闭完成时最多等待多少秒。默认值为环境变量PGCTLTIMEOUT的值；如果未设置该环境变量，则默认为 60 秒。

-V
--version

打印pg_ctl的版本并退出。

-w

等待启动或关闭完成。对关闭来说，等待是默认选项；但对启动来说不是。等待启动时，pg_ctl会反复尝试连接服务器。等待关闭时，pg_ctl会等待服务器移除其PID文件。pg_ctl会根据启动或关闭是否成功返回相应的退出代码。

-W

不等待启动或关闭完成。对 start 和 restart 模式来说，这是默认行为。

-?
--help

显示有关pg_ctl命令行参数的帮助并退出。

用于 Windows 的选项

-N *servicename*

要注册的系统服务名称。该名称会同时用作服务名和显示名。

-P *password*

用于启动该服务的用户密码。

-S *start-type*

要注册的系统服务的启动类型。start-type 可以是auto、demand，或者两者之一的首字母。若省略此选项，则使用auto。

-U *username*

用于启动该服务的用户名。对于域用户，请使用DOMAIN\username格式。

环境

PGCTLTIMEOUT

等待启动或关闭完成时，默认的等待秒数上限。如果未设置，默认值为 60 秒。

PGDATA

默认数据目录位置。

pg_ctl与大多数其他PostgreSQL工具一样，也使用libpq所支持的环境变量（参见第 31.14 节）。其他服务器相关变量见postgres。

文件

postmaster.pid

数据目录中该文件的存在与否，用于帮助pg_ctl确定服务器当前是否正在运行。

postmaster.opts

如果该文件存在于数据目录中，pg_ctl（在restart模式下）会将该文件的内容作为选项传递给postgres，除非被-o选项覆盖。该文件的内容也会在status模式下显示出来。

示例

启动服务器

要启动服务器：

```
$ pg_ctl start
```

要启动服务器并等待其开始接受连接：

```
$ pg_ctl -w start
```

要使用端口 5433 启动服务器，并在未启用fsync的情况下运行，可使用：

```
$ pg_ctl -o "-F -p 5433" start
```

停止服务器

要停止服务器，可使用：

```
$ pg_ctl stop
```

-m选项允许控制服务器如何关闭：

```
$ pg_ctl stop -m fast
```

重启服务器

重启服务器几乎等同于先停止服务器再重新启动，只不过pg_ctl会保存并重用传给前一个运行实例的命令行选项。以最简单的形式重启服务器，可使用：

```
$ pg_ctl restart
```

要重启服务器并等待其关闭和重启完成：

```
$ pg_ctl -w restart
```

要使用端口 5433 重启服务器，并在重启时禁用fsync，可使用：

```
$ pg_ctl -o "-F -p 5433" restart
```

显示服务器状态

下面是pg_ctl状态输出的示例：

```
$ pg_ctl status
pg_ctl: server is running (PID: 13718)
/usr/local/pgsql/bin/postgres "-D" "/usr/local/pgsql/data" "-p"
 "5433" "-B" "128"
```

这是在重启模式下会调用的命令行。

参见

initdb, postgres

pg_resetxlog

pg_resetxlog — 重置一个 PostgreSQL 数据库集簇的事务日志及其他控制信息

大纲

pg_resetxlog [-f] [-n] [-o *oid*] [-x *xid*] [-e *xid_epoch*] [-m *mxid*] [-O *mxoff*] [-l *timelineid*,*fileid*,*seg*] *datadir*

描述

pg_resetxlog 清空事务日志（WAL），并有选择地重置 pg_control 文件中存储的一些其他控制信息。当这些文件损坏时，有时就需要这项功能。它应当只作为最后的手段使用，即当服务器因这种损坏而无法启动时。

运行这条命令之后，应当就能启动服务器了，但请记住，由于部分提交的事务，数据库中可能包含不一致的数据。你应当立即转储数据、运行 initdb 并重新装载。重新装载之后，检查不一致之处并按需修复。

这个工具只能由安装服务器的用户运行，因为它需要对数据目录的读写访问权限。出于安全原因，你必须在命令行上指定数据目录。pg_resetxlog 不使用环境变量 PGDATA。

如果 pg_resetxlog 抱怨说无法确定 pg_control 的有效数据，你可以通过指定 -f（强制）选项迫使它继续执行。在这种情况下，缺失的数据会用貌似合理的值代替。大多数字段都可以期望是匹配的，但下一个 OID、下一个事务 ID 及其纪元、下一个多事务 ID 及偏移量、WAL 起始地址这些字段可能需要人工协助。这些字段可以用下面讨论的选项来设置。如果你无法为所有这些字段确定正确的值，仍可以使用 -f，但恢复出的数据库必须受到比平常更多的怀疑：必须立即转储并重新装载。在转储之前不要在数据库中执行任何修改数据的操作，因为任何此类操作都很可能使损坏加剧。

-o、-x、-e、-m、-O 和 -l 选项允许手动设置下一个 OID、下一个事务 ID、下一个事务 ID 的纪元、下一个和最旧的多事务 ID、下一个多事务偏移量以及 WAL 起始地址。只有当 pg_resetxlog 无法通过读取 pg_control 确定合适的值时，才需要这些选项。安全值可以按如下方法确定：

- 下一个事务 ID（-x）的安全值可以通过在数据目录下的 pg_clog 目录中查找数值最大的文件名、加一、再乘以 1048576 来确定。注意文件名为十六进制。通常最简单的做法也是用十六进制指定选项值。例如，如果 0011 是 pg_clog 中最大的项，则 -x 0x1200000 可行（五个尾随零提供正确的乘数）。

- 下一个多事务 ID 的安全值（-m）可以通过在数据目录下的 pg_multixact/offsets 目录中查找数值最大的文件名、加一、再乘以 65536 来确定。如上所述，文件名为十六进制，因此最简单的做法是用十六进制指定选项值并加上四个零。

- 下一个多事务偏移量（-O）的安全值可以通过在数据目录下的 pg_multixact/members 目录中查找数值最大的文件名、加一、再乘以 65536 来确定。如上所述，文件名为十六进制，因此最简单的做法是用十六进制指定选项值并加上四个零。

- WAL 起始地址（-l）应当大于数据目录下 pg_xlog 目录中现存的任何 WAL 段文件名。这些名称同样是十六进制的，并且分为三部分：第一部分是“时间线 ID”（范围为 0 到 255），第二部分范围为 0 到 511，第三部分范围为 0到 240。不要为第三部分选择大于 255（0xFF）的值；而应将第二部分加一并把第三部分重置为 0。例如，如果 000000010000003200000004A 是 pg_xlog 中最大的项，-l 0x1,0x32,0x4B 即可；但如果最大的项是 000000010000003A000000FF，则应选择 -l 0x1,0x3B,0x0 或更高。

> **注意**
>
> `pg_resetxlog` 本身会查看 `pg_xlog` 中的文件，并选择一个超出最后一个现存文件名的默认 `-l` 设置。因此，只有当你知道存在当前不在 `pg_xlog` 中的 WAL 段文件（例如离线归档中的条目），或者 `pg_xlog` 的内容已完全丢失时，才需要手动调整 `-l`。

- 没有同样简单的方法来确定一个超出数据库中最大 OID 的下一个 OID，但幸运的是，下一个 OID 设置得是否正确并不关键。
- 事务 ID 纪元实际上并不存储在数据库中的任何地方（由 `pg_resetxlog` 设置的字段除外），因此就数据库本身而言，任何值都可以。你可能需要调整这个值以确保 Slony-I 等复制系统正常工作—— 如果需要，应该可以从下游复制数据库的状态中获得合适的值。

`-n`（不操作）选项指示 `pg_resetxlog` 打印从 `pg_control` 重建的值，然后不做任何修改直接退出。这主要是一种调试工具，但在真正让 `pg_resetxlog` 执行之前，也可以用作健全性检查。

`-V` 和 `--version` 选项打印 pg_resetxlog 的版本并退出。`-?` 和 `--help` 选项显示支持的参数，并退出。

注解

服务器正在运行时不得使用这条命令。如果 `pg_resetxlog` 在数据目录中发现服务器锁文件，它将拒绝启动。如果服务器崩溃过，可能会留下锁文件；
这种情况下你可以删除锁文件以允许 `pg_resetxlog` 运行。但在这样做之前，请务必再三确认没有仍然存活的服务器进程。

`pg_resetxlog` 只能用于同一大版本的服务器。

postgres

postgres — PostgreSQL数据库服务器

大纲

postgres [*option*...]

描述

postgres是 PostgreSQL数据库服务器。客户端应用要访问数据库时，会（通过网络或本地）连接到一个正在运行的postgres实例。该postgres
实例随后会启动一个独立的服务器进程来处理该连接。

一个postgres实例始终只管理一个数据库集簇的数据。
数据库集簇是一组存储在同一文件系统位置（“数据区域”）中的数据库。
只要使用不同的数据区域和不同的通信端口（见下文），一个系统上就可以同时运行多个postgres实例。当 postgres启动时，它需要知道数据区域的位置。该位置必须通过 -D选项或PGDATA环境变量指定；没有默认值。通常，-D或 PGDATA会直接指向由initdb创建的数据区域目录。其他可能的文件布局见第 18.2 节。

默认情况下，postgres在前台启动，并将日志消息打印到标准错误流。在实际应用中，postgres 应作为后台进程启动，例如在系统启动时启动。

postgres命令也可以在单用户模式下调用。
这种模式的主要用途是在initdb执行引导期间使用。有时也会用它来调试或进行灾难恢复；
但请注意，运行单用户服务器其实并不适合调试服务器，
因为不会发生真实的进程间通信和锁定。从 shell 以单用户模式调用时，用户可以输入查询，
结果会打印到屏幕上，但其形式对开发者更有用，而非面向终端用户。在单用户模式下，
会话用户会被设置为 ID 为 1 的用户，并向该用户隐式授予超级用户权限。该用户实际上不必存在，因此单用户模式可用于手工恢复系统目录遭受某些意外损坏的情况。

选项

postgres接受以下命令行参数。有关这些选项的详细讨论，请参阅第 18 章。通过设置配置文件，你可以省去键入其中大多数选项的麻烦。某些（安全的）
选项还可以由连接的客户端以应用相关的方式设置，并且仅对该会话生效。例如，
如果设置了环境变量PGOPTIONS，那么基于libpq的客户端就会将该字符串传给服务器，
服务器会将其解释为 postgres命令行选项。

通用选项

-A 0|1

启用运行时断言检查，这是一种帮助检测编程错误的调试辅助手段。只有在编译PostgreSQL 时启用了断言，此选项才可用；如果是这样，默认值为开。

-B *nbuffers*

设置供服务器进程使用的共享缓冲区数量。该参数的默认值由initdb自动选择。
指定此选项等效于设置shared_buffers配置参数。

-c *name=value*

设置一个命名的运行时参数。PostgreSQL支持的配置参数见第 18 章。大多数其他命令行选项实际上都是这种参数赋值的简写形式。-c可以出现多次以设置多个参数。

`-C` *name*

打印指定运行时参数的值并退出。（详情见上面的`-c`选项。）
这个选项可用于正在运行的服务器，它会返回 `postgresql.conf`中的值，并应用本次调用中提供的任何参数修改。它不反映集簇启动时提供的参数。

该选项供其他与服务器实例交互的程序（例如pg_ctl）查询配置参数值之用。
面向用户的应用则应使用SHOW或`pg_settings`视图。

`-d` *debug-level*

设置调试级别。该值设得越高，写入服务器日志的调试输出就越多。取值范围是 1 到 5。还可以针对某个特定会话传入`-d 0`，从而阻止父`postgres`进程的服务器日志级别传播到该会话。

`-D` *datadir*

指定数据目录或配置文件在文件系统中的位置。详见第 18.2 节。

`-e`

将默认日期风格设置为“European”，即输入日期字段采用 `DMY`顺序。这也会在某些日期输出格式中使日期显示为日在月前。更多信息见第 8.5 节。

`-F`

禁用`fsync`调用以提高性能，但在系统崩溃时会有数据损坏的风险。
指定此选项等效于禁用fsync配置参数。使用前请先阅读详细文档！

`-h` *hostname*

指定`postgres`用来侦听来自客户端应用的 TCP/IP 连接的 IP 主机名或地址。该值也可以是逗号分隔的地址列表，或者用`*`指定侦听所有可用接口。空值表示不侦听任何 IP 地址，在这种情况下只能使用 Unix 域套接字连接到服务器。默认只侦听 localhost。指定此选项等效于设置listen_addresses配置参数。

`-i`

允许远程客户端通过 TCP/IP（Internet 域）连接。没有此选项时，只接受本地连接。该选项等效于在 `postgresql.conf`中，或通过`-h`将 `listen_addresses`设置为`*`。

该选项已被弃用，因为它无法访问listen_addresses的全部功能。
通常最好直接设置`listen_addresses`。

`-k` *directory*

指定`postgres`用来侦听客户端应用连接的 Unix 域套接字所在目录。默认值通常是 `/tmp`，但可以在编译时更改。

`-l`

启用使用SSL的安全连接。若要使用此选项，PostgreSQL必须在编译时启用 SSL支持。关于使用SSL的更多信息，请参阅第 17.9 节。

`-N` *max-connections*

设置该服务器将接受的最大客户端连接数。该参数的默认值由initdb自动选择。
指定此选项等效于设置max_connections配置参数。

`-o` *extra-options*

在*`extra-options`*中指定的命令行形式选项，会传给由这个`postgres`
进程启动的所有服务器进程。如果选项字符串中包含空格，整个字符串必须加引号。

此选项已过时；服务器进程的所有命令行选项都可以直接在postgres命令行上指定。

-p *port*

指定postgres 用来侦听客户端应用连接的 TCP/IP 端口或本地 Unix 域套接字文件扩展名。默认值取自PGPORT环境变量；如果PGPORT未设置，
则默认值就是在编译期间确定的值（通常为 5432）。如果你指定了非默认端口，那么所有客户端应用都必须通过命令行选项或PGPORT指定相同的端口。

-s

在每条命令结束时打印时间信息和其他统计信息。
这对基准测试或者调整缓冲区数量很有用。

-S *work-mem*

指定内部排序和散列操作在转而使用临时磁盘文件之前可使用的内存量。有关work_mem配置参数的说明，请参阅第 18.4.1 节。

-V
--version

打印postgres的版本并退出。

--*name=value*

设置一个命名的运行时参数；这是-c的较短形式。

--describe-config

该选项会以制表符分隔的COPY格式导出服务器内部配置变量、说明以及默认值。
它主要是为管理工具设计的。

-?
--help

显示postgres命令行参数的帮助并退出。

半内部选项

这里描述的选项主要用于调试，并且在某些情况下可协助恢复严重损坏的数据库。
在生产数据库环境中没有理由使用它们。这里列出这些选项，仅供PostgreSQL 系统开发者使用。此外，这些选项在未来版本中可能会更改或被移除，且不另行通知。

-f { s | i | o | b | t | n | m | h }

禁止使用某些扫描和连接方法：s和i 分别禁用顺序扫描和索引扫描，o、b和t 分别禁用仅索引扫描、位图索引扫描和 TID 扫描，而n、m和h 则分别禁用嵌套循环连接、归并连接和哈希连接。

顺序扫描和嵌套循环连接都无法被完全禁用；-fs和 -fn选项只是在优化器有其他选择时，尽量不使用这些计划类型。

-n

此选项用于调试导致服务器进程异常终止的问题。在这种情况下，
通常的策略是通知所有其他服务器进程必须终止，然后重新初始化共享内存和信号量。
这是因为出错的服务器进程可能在终止前破坏了某些共享状态。此选项指定postgres不重新初始化共享数据结构。
经验丰富的系统程序员随后可以使用调试器检查共享内存和信号量的状态。

`-O`

允许修改系统表的结构。该选项由 `initdb`使用。

`-P`

读取系统表时忽略系统索引，但在修改表时仍然更新这些索引。
这在从损坏的系统索引中恢复时很有用。

`-t pa[rser] | pl[anner] | e[xecutor]`

打印与各个主要系统模块相关的每个查询的时间统计信息。
该选项不能与`-s`选项一起使用。

`-T`

此选项用于调试导致服务器进程异常终止的问题。在这种情况下，
通常的策略是通知所有其他服务器进程必须终止，然后重新初始化共享内存和信号量。
这是因为出错的服务器进程可能在终止前破坏了某些共享状态。此选项指定`postgres`
通过发送`SIGSTOP`信号暂停所有其他服务器进程，但不使它们终止。
这允许系统程序员手工收集所有服务器进程的核心转储。

`-v` *`protocol`*

指定特定会话要使用的前端/后端协议版本号。该选项仅供内部使用。

`-W` *`seconds`*

新服务器进程在完成认证过程后，会延迟这么多秒。
这是为了给调试器附着到该服务器进程提供机会。

用于单用户模式的选项

以下选项仅适用于单用户模式。

`--single`

选择单用户模式。这必须是命令行上的第一个参数。

`database`

指定要访问的数据库名称。这必须是命令行上的最后一个参数。如果省略，
则默认为用户名。

`-E`

回显所有命令。

`-j`

禁用换行符作为语句分隔符。

`-r` *`filename`*

将所有服务器日志输出发送到*`filename`*。只有在正常的多用户模式下，
此选项会被忽略，所有进程都使用 stderr。

环境

`PGCLIENTENCODING`

客户端使用的默认字符编码。（客户端可以单独覆盖它。）
该值也可以在配置文件中设置。

PGDATA

默认数据目录位置

PGDATESTYLE

DateStyle运行时参数的默认值。（该环境变量的用法已被弃用。）

PGPORT

默认端口号（最好在配置文件中设置）

诊断

提到semget或 shmget的失败消息，很可能表明你需要配置内核，以提供足够的共享内存和信号量。更多讨论见第 17.4 节。你也许可以通过降低shared_buffers来减少PostgreSQL的共享内存消耗，以及/或通过降低max_connections来减少信号量消耗，
从而推迟重新配置内核。

表明另一个服务器已经在运行的失败消息应仔细检查，
例如可根据你的系统使用下面的命令：

```
$ ps ax | grep postgres
```

或

```
$ ps -ef | grep postgres
```

如果你确信没有冲突的服务器在运行，可以删除消息中提到的锁文件，然后重试。

表明无法绑定端口的失败消息，可能意味着该端口已经被某个非PostgreSQL进程占用。
你在终止postgres 后如果立即使用同一端口重新启动它，也可能会得到这个错误；在这种情况下，你只需等待几秒钟，直到操作系统关闭该端口后再重试。最后，
如果你指定了一个操作系统认为是保留的端口号，也可能会收到这个错误。例如，
许多版本的 Unix 认为低于 1024 的端口号是“受信任的”，并且只允许 Unix 超级用户访问它们。

注解

实用命令pg_ctl可用于安全、便捷地启动和关闭postgres服务器。

只要有可能，就不要使用 SIGKILL杀死主 postgres服务器。这样会阻止 postgres在终止前释放其持有的系统资源（例如共享内存和信号量）。这可能会导致新的postgres
实例在启动时出现问题。

要正常终止postgres服务器，可以使用 SIGTERM、SIGINT或 SIGQUIT信号。第一个会在退出前等待所有客户端终止，第二个会强制断开所有客户端，
而第三个会在不进行正确关闭的情况下立即退出，从而在重启期间导致一次恢复运行。

SIGHUP信号会重新加载服务器配置文件。也可以向单个服务器进程发送 SIGHUP，但这通常没有意义。

要取消一个正在运行的查询，可以向执行该命令的进程发送SIGINT信号。

postgres服务器使用SIGTERM 通知从属服务器进程正常退出，并使用SIGQUIT 让它们在不进行正常清理的情况下终止。用户不应该使用这些信号。向某个服务器进程发送SIGKILL
也是不明智的 – 主postgres进程会将其解释为一次崩溃，并作为其标准崩溃恢复过程的一部分强制所有同级进程退出。

缺陷

--选项在FreeBSD或OpenBSD上不起作用。请改用-c。这是受影响操作系统中的一个缺陷；如果这个问题未被修复，未来版本的PostgreSQL 将提供变通方案。

用法

要启动单用户模式服务器，可以使用类似下面的命令：

postgres --single -D /usr/local/pgsql/data *other-options* my_database

使用-D提供正确的数据库目录路径，或者确保已经设置了环境变量PGDATA。还要指定你想要操作的那个数据库的名称。

通常，单用户模式服务器将换行视为命令输入终止符；它不像psql那样会对分号进行智能处理。要让一个命令跨越多行，必须在除最后一个换行之外的每个换行前输入反斜线。

但是，如果你使用-j命令行开关，换行并不会终止命令输入。在这种情况下，服务器会读取标准输入直到文件末尾（EOF）标记，然后把输入作为单个命令字符串处理。在这种情况下，反斜线-换行不会被特殊处理。

要退出会话，输入EOF（通常是**Control+D**）。如果你使用了-j，则需要连续输入两次EOF才能退出。

请注意，单用户模式服务器不提供复杂的行编辑功能（例如没有命令历史）。单用户模式也不会执行任何后台处理，例如自动检查点。

示例

要使用默认值在后台启动postgres，输入：

```
$ nohup postgres >logfile 2>&1 </dev/null &
```

要用指定端口启动postgres，例如 1234：

```
$ postgres -p 1234
```

要使用psql连接到该服务器，请用 -p 选项指定这个端口：

```
$ psql -p 1234
```

或者设置环境变量PGPORT：

```
$ export PGPORT=1234
$ psql
```

命名的运行时参数可以采用以下任一种形式设置：

```
$ postgres -c work_mem=1234
$ postgres --work-mem=1234
```

两种形式都会覆盖 postgresql.conf中可能存在的 work_mem设置。请注意，参数名中的下划线在命令行中既可以写成下划线，也可以写成连字符。除了短期实验之外，与其依赖命令行开关来设置参数，通常更好的做法是直接编辑 postgresql.conf中的设置。

另见

initdb, pg_ctl

postmaster

postmaster — PostgreSQL数据库服务器

大纲

`postmaster` [*`option`*...]

描述

`postmaster`是`postgres`的一个已弃用别名。

另见

postgres

部分 VII. 内部

本部分包含一些可能对PostgreSQL开发人员有用的各类信息。

目录

44. PostgreSQL 内部概述 1472
44.1. 查询的路径 1472
44.2. 连接是如何建立的 1472
44.3. 解析器阶段 1473
44.3.1. 解析器 1473
44.3.2. 转换过程 1473
44.4. PostgreSQL 规则系统 1474
44.5. 规划器/优化器 1474
44.5.1. 生成可能的计划 1474
44.6. 执行器 1475
45. 系统目录 1477
45.1. 概述 1477
45.2. pg_aggregate 1478
45.3. pg_am 1479
45.4. pg_amop 1481
45.5. pg_amproc 1482
45.6. pg_attrdef 1482
45.7. pg_attribute 1482
45.8. pg_authid 1485
45.9. pg_auth_members 1486
45.10. pg_cast 1486
45.11. pg_class 1488
45.12. pg_constraint 1490
45.13. pg_collation 1493
45.14. pg_conversion 1493
45.15. pg_database 1494
45.16. pg_db_role_setting 1495
45.17. pg_default_acl 1496
45.18. pg_depend 1496
45.19. pg_description 1497
45.20. pg_enum 1498
45.21. pg_extension 1498
45.22. pg_foreign_data_wrapper 1499
45.23. pg_foreign_server 1500
45.24. pg_foreign_table 1500
45.25. pg_index 1500
45.26. pg_inherits 1502
45.27. pg_language 1503
45.28. pg_largeobject 1504
45.29. pg_largeobject_metadata 1504
45.30. pg_namespace 1505
45.31. pg_opclass 1505
45.32. pg_operator 1506
45.33. pg_opfamily 1506
45.34. pg_pltemplate 1507
45.35. pg_proc 1508
45.36. pg_range 1511
45.37. pg_rewrite 1511
45.38. pg_seclabel 1512
45.39. pg_shdepend 1513
45.40. pg_shdescription 1514
45.41. pg_shseclabel 1514
45.42. pg_statistic 1515
45.43. pg_tablespace 1516

45.44. pg_trigger 1517
45.45. pg_ts_config 1518
45.46. pg_ts_config_map 1519
45.47. pg_ts_dict 1519
45.48. pg_ts_parser 1520
45.49. pg_ts_template 1520
45.50. pg_type 1521
45.51. pg_user_mapping 1525
45.52. 系统视图 1525
45.53. pg_available_extensions 1526
45.54. pg_available_extension_versions 1527
45.55. pg_cursors 1527
45.56. pg_group 1528
45.57. pg_indexes 1528
45.58. pg_locks 1529
45.59. pg_prepared_statements 1531
45.60. pg_prepared_xacts 1532
45.61. pg_roles 1532
45.62. pg_rules 1533
45.63. pg_seclabels 1534
45.64. pg_settings 1534
45.65. pg_shadow 1536
45.66. pg_stats 1537
45.67. pg_tables 1538
45.68. pg_timezone_abbrevs 1539
45.69. pg_timezone_names 1539
45.70. pg_user 1540
45.71. pg_user_mappings 1540
45.72. pg_views 1541
46. Frontend/Backend Protocol 1542
46.1. Overview 1542
46.1.1. Messaging Overview 1542
46.1.2. Extended Query Overview 1543
46.1.3. Formats and Format Codes 1543
46.2. Message Flow 1544
46.2.1. Start-up 1544
46.2.2. Simple Query 1546
46.2.3. Extended Query 1547
46.2.4. Function Call 1550
46.2.5. COPY Operations 1551
46.2.6. Asynchronous Operations 1552
46.2.7. Canceling Requests in Progress 1553
46.2.8. Termination 1554
46.2.9. SSL Session Encryption 1554
46.3. Streaming Replication Protocol 1555
46.4. Message Data Types 1558
46.5. Message Formats 1559
46.6. Error and Notice Message Fields 1574
46.7. Summary of Changes since Protocol 2.0 1575
47. PostgreSQL 编码约定 1577
47.1. 格式 1577
47.2. 在服务器内部报告错误 1577
47.3. 错误消息风格指南 1579
48. 本地语言支持 1584
48.1. 给翻译者 1584
48.1.1. 要求 1584
48.1.2. 概念 1584

48.1.3. 创建和维护消息目录 1585
48.1.4. 编辑 PO 文件 1585
48.2. 给程序员 1586
48.2.1. 实现机制 1586
48.2.2. 消息编写指南 1587
49. 编写过程语言调用处理器 1589
50. Writing A Foreign Data Wrapper 1592
50.1. Foreign Data Wrapper Functions 1592
50.2. Foreign Data Wrapper Callback Routines 1592
50.3. Foreign Data Wrapper Helper Functions 1595
50.4. Foreign Data Wrapper Query Planning 1596
51. 遗传查询优化器 1598
51.1. 将查询处理看成是一个复杂的优化问题 1598
51.2. 遗传算法 1598
51.3. PostgreSQL 中的遗传查询优化（GEQO） 1599
51.3.1. 使用GEQO生成可能的计划 1599
51.3.2. PostgreSQL GEQO的未来实现任务 1600
51.4. 进一步阅读 1600
52. 索引访问方法接口定义 1601
52.1. 索引的目录条目 1601
52.2. 索引访问方法函数 1602
52.3. 索引扫描 1605
52.4. 索引锁定注意事项 1606
52.5. 索引唯一性检查 1607
52.6. 索引代价估算函数 1608
53. GiST 索引 1611
53.1. 简介 1611
53.2. 可扩展性 1611
53.3. 实现 1617
53.3.1. GiST 缓冲构建 1617
53.4. 示例 1618
54. SP-GiST 索引 1619
54.1. 简介 1619
54.2. 可扩展性 1619
54.3. 实现 1625
54.3.1. SP-GiST 限制 1625
54.3.2. 无结点标签的 SP-GiST 1625
54.3.3. “全部相同”的内部元组 1626
54.4. 示例 1626
55. GIN Indexes 1627
55.1. Introduction 1627
55.2. Extensibility 1627
55.3. Implementation 1629
55.3.1. GIN Fast Update Technique 1630
55.3.2. Partial Match Algorithm 1630
55.4. GIN Tips and Tricks 1630
55.5. Limitations 1631
55.6. Examples 1631
56. Database Physical Storage 1633
56.1. Database File Layout 1633
56.2. TOAST 1635
56.3. Free Space Map 1637
56.4. Visibility Map 1637
56.5. The Initialization Fork 1637
56.6. Database Page Layout 1638
57. BKI 后端接口 1641
57.1. BKI 文件格式 1641
57.2. BKI 命令 1641

57.3. 引导 BKI 文件的结构 1642
57.4. 示例 1643
58. 规划器如何使用统计信息 1644
58.1. 行估计示例 1644
58.2. 规划器统计信息与安全 1648

第 44 章 PostgreSQL 内部概述

作者

本章最初源自[sim98] Stefan Simkovics 在维也纳技术大学完成的硕士论文，该论文由 O.Univ.Prof.Dr. Georg Gottlob 和 Univ.Ass. Mag. Katrin Seyr 指导。

本章概述 PostgreSQL 后端的内部结构。阅读以下各节之后，你应该能够大致了解一个查询是如何被处理的。本章并不试图详细描述 PostgreSQL 的内部运作，因为那样的文档会非常庞大。相反，本章旨在帮助读者理解，从后端收到一个查询开始，
到将结果返回给客户端为止，内部通常有哪些操作步骤。

44.1. 查询的路径

这里将简要概述一个查询为得到结果必须经过的各个阶段。

1. 必须先建立从应用程序到 PostgreSQL 服务器的连接。应用程序将查询发送给服务器，并等待接收服务器返回的结果。

2. 解析器阶段检查应用程序传来的查询是否具有正确的语法，并创建一棵查询树。

3. 重写系统接收由解析器阶段创建的查询树，并查找任何可应用于该查询树的规则（存储在系统目录中）。它执行这些规则体中给出的转换。

   重写系统的一个应用是实现视图。每当对一个视图（即虚拟表）发出查询时，
   重写系统都会将用户的查询重写成一个改为访问视图定义中给出的基表的查询。

4. 规划器/优化器接收（重写后的）查询树，并创建一个查询计划，将作为执行器的输入。

   它首先生成所有能得到同一结果的可能路径。例如，
   如果待扫描的某个关系上有一个索引，
   那么该扫描就有两条路径：一种是简单的顺序扫描，另一种是使用该索引。
   接着会估算执行每条路径的代价，并选择代价最低的路径。
   代价最低的路径会被展开成一个完整的计划，供执行器使用。

5. 执行器会递归地遍历计划树，并以计划所表示的方式提取行。
   执行器在扫描关系时会使用存储系统，执行排序和连接，计算限定条件，
   最后返回得到的行。

在后续各节中，我们将更详细地介绍上述各项内容，以便更好地理解 PostgreSQL 的内部控制和数据结构。

44.2. 连接是如何建立的

PostgreSQL 使用一种简单的“每用户一个进程”客户端/服务器模型。在这种模型中，一个客户端进程恰好连接到一个服务器进程。由于事先不知道会建立多少个连接，
因此必须使用一个主进程，在每次收到连接请求时派生一个新的服务器进程。
这个主进程称为 `postgres`，在指定的 TCP/IP 端口上监听传入连接。每当检测到连接请求时，`postgres` 进程就会派生一个新的服务器进程。服务器任务之间使用信号量和共享内存通信，以确保并发数据访问期间的数据完整性。

客户端进程可以是任何理解 PostgreSQL 协议（见第 46 章）的程序。很多客户端基于 C 语言库 libpq，但该协议也有若干独立实现，例如 Java 的 JDBC 驱动。

一旦连接建立起来，客户端进程就可以向后端（服务器）发送查询。
查询以纯文本方式传输，也就是说，前端（客户端）不进行解析。服务器解析查询，
创建一个执行计划，执行该计划，并通过已建立的连接把检索到的行返回给客户端。

44.3. 解析器阶段

解析器阶段由两部分组成：

- 解析器定义在 `gram.y` 和 `scan.l` 中，它使用 Unix 工具 bison 和 flex 构建。
- 转换过程，它会对解析器返回的数据结构做修改和补充。

44.3.1. 解析器

解析器必须检查查询字符串（它以纯 ASCII 文本形式到达）是否具有有效语法。如果语法正确，就会构造出一棵语法解析树并返回；否则会返回错误。
解析器和词法分析器是使用著名的 Unix 工具 bison 和 flex 实现的。

词法分析器定义在文件 `scan.l` 中，负责识别标识符、SQL 关键字等。每找到一个关键字或标识符，就会生成一个词元并交给解析器。

解析器定义在文件 `gram.y` 中，由一组语法规则和动作组成，每当某条规则被触发时，对应动作就会执行。动作中的代码（实际上是 C 代码）用于构造语法解析树。

文件 `scan.l` 会被转换成 C 源文件 `scan.c`，所用程序是 flex；而 `gram.y` 则会被转换成 `gram.c`，所用程序是 bison。这些转换完成之后，就可以使用普通的 C 编译器来构建解析器。绝不要修改这些生成出来的 C 文件，因为下一次调用 flex 或 bison 时，它们都会被覆盖。

> **注意**
>
> 上述转换和编译通常都是借助 makefiles 自动完成的，这些文件随 PostgreSQL 源代码发行版一同提供。

对 bison 的详细描述，或者对 `gram.y` 中给出的语法规则的说明，都超出了本文的范围。有许多书籍和文档专门讨论 flex 和 bison。在开始研究之前，你应该先熟悉 bison，然后再去看 `gram.y` 中给出的语法，否则你将无法理解其中发生了什么。

44.3.2. 转换过程

解析器阶段仅依据 SQL 语法结构的固定规则创建一棵语法解析树。它不会在系统目录中做任何查找，因此不可能理解所请求操作的详细语义。解析器完成之后，
转换过程会接收解析器返回的树，并进行必要的语义解释，以理解查询引用了哪些表、
函数和操作符。为表示这些信息而构建的数据结构称为查询树。

之所以将原始解析与语义分析分开，是因为系统目录查找只能在事务内部进行，
而我们不希望在刚收到查询字符串时就立刻启动事务。
原始解析阶段已经足以识别事务控制命令（`BEGIN`、`ROLLBACK` 等），因此这些命令可以在无需进一步分析的情况下被正确执行。一旦我们知道当前处理的是实际查询（例如 `SELECT` 或 `UPDATE`），如果尚未处于事务中，就可以启动事务。只有在那之后才能调用转换过程。

转换过程创建的查询树在大多数地方的结构都与原始语法解析树相似，
但在细节上有许多不同。例如，语法解析树中的一个 `FuncCall` 节点表示某个在语法上看起来像函数调用的东西。它可能被转换成 `FuncExpr` 节点，也可能被转换成 `Aggref` 节点，这取决于被引用的名称最终被判定为普通函数还是聚合函数。此外，关于列和表达式结果的实际数据类型的信息也会被加入到查询树中。

44.4. PostgreSQL 规则系统

PostgreSQL 提供了一个强大的规则系统，用于定义视图以及处理有歧义的视图更新。最初，PostgreSQL 的规则系统由两种实现组成：

- 第一种实现使用行级处理，并且深度实现于执行器内部。每当访问到单独一行时，规则系统就会被调用。这种实现在 1995 年被移除，当时 Berkeley Postgres 项目的最后一个官方发行版被转换成了 Postgres95。
- 规则系统的第二种实现是一种称为查询重写的技术。重写系统是位于解析器阶段与规划器/优化器之间的一个模块。这种技术至今仍在使用。

关于查询重写器，第 37 章中已有相当详细的讨论，因此这里没有必要再展开。我们只指出一点：重写器的输入和输出都是查询树，也就是说，树的表示形式以及语义细节层次都不会发生变化。重写可以被看作某种形式的宏展开。

44.5. 规划器/优化器

规划器/优化器的任务是创建一个最优的执行计划。给定的一个 SQL 查询（因此也就是一棵查询树）实际上可以用多种不同方式执行，而这些方式都会产生同样的结果集。如果在计算上可行，查询优化器将考察这些可能执行计划中的每一种，并最终选择预计运行最快的那个执行计划。

> **注意**
>
> 在某些情况下，考察查询的每一种可能执行方式会耗费过多时间和内存。尤其是在执行涉及大量连接操作的查询时更是如此。为了在合理时间内确定一个合理的（不一定最优的）查询计划，PostgreSQL 会使用遗传查询优化器（见第 51 章），前提是连接数量超过某个阈值（见geqo_threshold）。

规划器的搜索过程实际上使用一种称为路径的数据结构，它只是计划的简化表示，仅包含规划器做出决策所需的信息。在确定出代价最低的路径之后，会构建一棵完整的计划树传递给执行器。这棵树以足够细致的方式表示了期望的执行计划，使执行器能够运行它。在本节余下部分，我们将忽略路径与计划之间的区别。

44.5.1. 生成可能的计划

规划器/优化器首先为查询中使用的每个单独关系（表）生成扫描计划。可能的计划由每个关系上可用的索引决定。对一个关系总是可以执行顺序扫描，因此一定会生成顺序扫描计划。假设某个关系上定义了一个索引（例如一个 B-树索引），并且查询中包含限制条件 `relation.attribute OPR constant`。如果 `relation.attribute` 恰好匹配该 B-树索引的键，而且 `OPR` 是该索引操作符类中列出的操作符之一，那么就会生成另一个使用该 B-树索引扫描该关系的计划。如果还存在其他索引，并且查询中的限制条件恰好匹配某个索引的键，则还会考虑更多计划。对于那些排序顺序能够匹配查询 `ORDER BY` 子句（如果存在）或者可能有助于进行归并连接（见下文）的索引，也会生成索引扫描计划。

如果查询需要连接两个或更多关系，那么会在找出所有可行的单关系扫描计划之后，再考虑关系连接计划。有以下三种可用的连接策略：

- 嵌套循环连接：左关系中找到的每一行，都会使右关系被扫描一次。这种策略实现起来很容易，但可能非常耗时。（不过，如果右关系可以通过索引扫描，那么这也可能是一种不错的策略。可以把左关系当前行中的值作为右关系索引扫描的键。）

- 归并连接：在连接开始之前，每个关系都会先按照连接属性排序。
  然后两个关系并行扫描，将匹配的行组合成连接结果行。这种连接方式很有吸引力，
  因为每个关系只需扫描一次。所需的排序既可以通过显式排序步骤完成，
  也可以利用连接键上的索引按适当顺序扫描关系来完成。

- 哈希连接：首先扫描右关系，并使用其连接属性作为哈希键将其装入一个哈希表。
  接着扫描左关系，并将找到的每一行中的相应值作为哈希键，
  用来在该哈希表中定位匹配的行。

当查询涉及两个以上的关系时，最终结果必须由一棵连接步骤树构建出来，
每个连接步骤都有两个输入。规划器会考察不同的可能连接顺序，以找出代价最低的那个。

如果查询使用的关系少于geqo_threshold个，就会执行一次近乎穷举的搜索，
以找出最佳连接顺序。对于任意两个关系，只要在 `WHERE` 限定条件中存在相应的连接子句（即存在类似 `where rel1.attr1=rel2.attr2` 这样的限制），规划器就会优先考虑它们之间的连接。没有连接子句的连接对只有在别无选择时才会被考虑，也就是说，
某个关系与其他任何关系之间都没有可用的连接子句。对于规划器所考虑的每一对连接，
都会生成所有可能的计划，并选择其中估计代价最低的那个。

当超过 `geqo_threshold` 时，被考虑的连接顺序将由启发式方法决定，如第 51 章中所述。
除此之外，处理过程与前面相同。

最终完成的计划树由基表的顺序扫描或索引扫描，再加上所需的嵌套循环、
归并或哈希连接节点，以及任何需要的辅助步骤（例如排序节点或聚合函数计算节点）
组成。这些计划节点类型中的大多数还具有执行选择（丢弃不满足指定布尔条件的行）
和投影（根据给定列值计算派生列集，也就是在需要时计算标量表达式）的额外能力。
规划器的职责之一，就是把来自 `WHERE` 子句的选择条件以及所需输出表达式的计算附加到计划树中最合适的节点上。

44.6. 执行器

执行器接收由规划器/优化器创建的计划，并递归地处理它，以提取所需的行集合。
这本质上是一种按需拉取的流水线机制。每当某个计划节点被调用时，它必须再交付一行，
或者报告自己已经完成了行的交付。

为了给出一个具体示例，假设顶层节点是一个 `MergeJoin` 节点。在执行归并之前，必须先取到两行（每个子计划各一行）。因此执行器会递归调用自身去处理这些子计划（从挂接在 `lefttree` 上的子计划开始）。新的顶层节点，也就是左子计划的顶层节点，假设是一个 `Sort` 节点，那么又需要通过递归来取得一个输入行。`Sort` 的子节点可能是一个 `SeqScan` 节点，它表示真正去读取一个表。执行该节点会使执行器从表中取出一行，并把它返回给调用者节点。`Sort` 节点会反复调用它的子节点，以取得所有待排序的行。当输入耗尽时（子节点返回的是 NULL 而不是一行），`Sort` 代码就执行排序，并最终能够返回它的第一条输出行，也就是排序后的第一行。它会把其余行保存起来，
以便在后续请求中按排序后的顺序交付这些行。

`MergeJoin` 节点同样会向它的右子计划请求第一行。然后它比较这两行，看它们是否可以连接；如果可以，就向调用者返回一条连接结果行。在下一次调用时，
或者在当前这一对输入无法连接时立即，
它会前进到某一个表的下一行或另一个表的下一行（取决于比较结果），
并再次检查是否匹配。最终，其中一个子计划会耗尽，`MergeJoin` 节点就返回 NULL，以表明无法再形成更多连接结果行。

复杂查询可能涉及许多层计划节点，但总体方法是相同的：每个节点在每次被调用时，
都会计算并返回它的下一条输出行。
每个节点还负责应用规划器分配给它的任何选择或投影表达式。

执行器机制用于求值全部四种基本 SQL 查询类型：`SELECT`、`INSERT`、`UPDATE` 和 `DELETE`。对于 `SELECT`，顶层执行器代码只需要把查询计划树返回的每一行发送给客户端。

对于 INSERT，每个返回的行会被插入到为该 INSERT 指定的目标表中。这项工作是在一个名为 ModifyTable 的特殊顶层计划节点中完成的。（一个简单的 INSERT ... VALUES 命令会创建一棵极为简单的计划树，它只包含一个计算单条结果行的 Result 节点，以及位于其上执行插入的 ModifyTable。但 INSERT ... SELECT 可能需要执行器机制的全部能力。）对于 UPDATE，规划器会安排每个计算得到的行包含所有被更新列的值，再加上原始目标行的 TID（tuple ID，即元组 ID 或行 ID）；这些数据会被送入 ModifyTable 节点，该节点利用这些信息创建一个新的更新后行，并将旧行标记为已删除。对于 DELETE，计划实际返回的唯一列就是 TID，而 ModifyTable 节点只需利用该 TID 访问每个目标行并将其标记为已删除。

第 45 章 系统目录

系统目录是关系型数据库管理系统存放模式元数据的地方，
例如表和列的信息以及内部记账信息。PostgreSQL 的系统目录是普通表。你可以删除并重建这些表、增加列、插入和更新数值，并借此把系统严重搞坏。通常，
不应手工修改系统目录，总是有 SQL 命令可用于完成这些操作。（例如，`CREATE DATABASE` 会向 `pg_database` 目录中插入一行 – 并且实际上会在磁盘上创建该数据库。）
对于某些特别深奥的操作存在一些例外，例如增加索引访问方法。

45.1. 概述

表 45.1列出了系统目录。每个目录更详细的文档见后文。

大多数系统目录在创建数据库时都会从模板数据库复制过来，因此之后是数据库专有的。
少数目录在物理上由一个集簇中的所有数据库共享；这些目录会在各自的说明中注明。

表 45.1. 系统目录

Catalog Name	Purpose
`pg_aggregate`	aggregate functions
`pg_am`	index access methods
`pg_amop`	access method operators
`pg_amproc`	access method support procedures
`pg_attrdef`	column default values
`pg_attribute`	table columns (“attributes”)
`pg_authid`	authorization identifiers (roles)
`pg_auth_members`	authorization identifier membership relati onships
`pg_cast`	casts (data type conversions)
`pg_class`	tables, indexes, sequences, views (“relati ons”)
`pg_constraint`	check constraints, unique constraints, primary key constraints, foreign key constraints
`pg_collation`	collations (locale information)
`pg_conversion`	encoding conversion information
`pg_database`	databases within this database cluster
`pg_db_role_setting`	per-role and per-database settings
`pg_default_acl`	default privileges for object types
`pg_depend`	dependencies between database objects
`pg_description`	descriptions or comments on database object s
`pg_enum`	enum label and value definitions
`pg_extension`	installed extensions
`pg_foreign_data_wrapper`	foreign-data wrapper definitions
`pg_foreign_server`	foreign server definitions
`pg_foreign_table`	additional foreign table information
`pg_index`	additional index information
`pg_inherits`	table inheritance hierarchy
`pg_language`	languages for writing functions

Catalog Name	Purpose
pg_largeobject	data pages for large objects
pg_largeobject_metadata	metadata for large objects
pg_namespace	schemas
pg_opclass	access method operator classes
pg_operator	operators
pg_opfamily	access method operator families
pg_pltemplate	template data for procedural languages
pg_proc	functions and procedures
pg_range	范围类型的信息
pg_rewrite	query rewrite rules
pg_seclabel	security labels on database objects
pg_shdepend	dependencies on shared objects
pg_shdescription	comments on shared objects
pg_shseclabel	共享数据库对象上的安全标签
pg_statistic	planner statistics
pg_tablespace	tablespaces within this database cluster
pg_trigger	triggers
pg_ts_config	text search configurations
pg_ts_config_map	text search configurations' token mappings
pg_ts_dict	text search dictionaries
pg_ts_parser	text search parsers
pg_ts_template	text search templates
pg_type	data types
pg_user_mapping	mappings of users to foreign servers

45.2. pg_aggregate

目录pg_aggregate存储关于聚合函数的信息。
聚合函数是对一组值（典型的是每个匹配查询条件的行中的同一个列的值）进行操作的函数，它返回从这些值中计算出的单个值。典型的聚合函数是 sum、count和max。pg_aggregate里的每个项都是一个pg_proc项的扩展。pg_proc项记载该聚合的名字、输入和输出数据类型，以及其他一些和普通函数类似的信息。

表 45.2. pg_aggregate 列

Name	Type	References	Description
aggfnoid	regproc	pg_proc.oid	pg_proc OID of the aggregate function
aggtransfn	regproc	pg_proc.oid	Transition function
aggfinalfn	regproc	pg_proc.oid	Final function (zero if none)
aggsortop	oid	pg_operator.oid	Associated sort operat or (zero if none)
aggtranstype	oid	pg_type.oid	Data type of the aggregate function's internal transition (state) data

Name	Type	References	Description
agginitval	text		The initial value of the transition state. This is a text field containing the initial value in its external string repres entation. If this field is null, the transition state value starts out null.

新的聚合函数可通过CREATE AGGREGATE命令注册。更多关于编写聚合函数以及转移函数的含义等信息请参见第 35.10 节。

45.3. pg_am

目录pg_am存储关于索引访问方法的信息。
系统支持的每种索引访问方法在这个目录中都有一行。此目录的内容在第 52 章中详细讨论。

表 45.3. pg_am 列

Name	Type	References	Description
amname	name		Name of the access method
amstrategies	int2		Number of operat or strategies for this access method, or zero if access method does not have a fixed set of operator strate gies
amsupport	int2		Number of suppor t routines for this access method
amcanorder	bool		Does the access method suppor t ordered scans sorted by the indexed column's value?
amcanorderbyop	bool		Does the access method support ordered scans sorted by the result of an operator on the indexed column?
amcanbackward	bool		Does the access method support backward scanning?
amcanunique	bool		Does the access method support unique indexes?
amcanmulticol	bool		Does the access method support multicolumn indexes?
amoptionalkey	bool		Does the access method support a scan without any

Name	Type	References	Description
			constraint for the first index column?
amsearcharray	bool		访问方法是否支持 ScalarArrayOpExpr 搜索?
amsearchnulls	bool		Does the access method support IS NULL/NOT NULL searches?
amstorage	bool		Can index storage data type differ from column data type?
amclusterable	bool		Can an index of this type be clustered on?
ampredlocks	bool		Does an index of this type manage fine-grained predicate locks?
amkeytype	oid	pg_type.oid	Type of data stored in index, or zero if not a fixed type
aminsert	regproc	pg_proc.oid	“Insert this tuple” function
ambeginscan	regproc	pg_proc.oid	“Prepare for index scan” function
amgettuple	regproc	pg_proc.oid	“Next valid tuple” function, or zero if none
amgetbitmap	regproc	pg_proc.oid	“Fetch all valid tuples” function, or zero if none
amrescan	regproc	pg_proc.oid	“(Re)start index scan” function
amendscan	regproc	pg_proc.oid	“Clean up after index scan” function
ammarkpos	regproc	pg_proc.oid	“Mark current scan position” function
amrestrpos	regproc	pg_proc.oid	“Restore marked scan position” function
ambuild	regproc	pg_proc.oid	“Build new index” function
ambuildempty	regproc	pg_proc.oid	“Build empty index” function
ambulkdelete	regproc	pg_proc.oid	Bulk-delete function
amvacuumcleanup	regproc	pg_proc.oid	Post-VACUUM cleanup function
amcanreturn	regproc	pg_proc.oid	检查索引列是否支持仅索引扫描的函数。如果从不支持仅索引扫描，可以为零。
amcostestimate	regproc	pg_proc.oid	Function to estimate cost of an index scan

Name	Type	References	Description
amoptions	regproc	pg_proc.oid	Function to parse and validate reloptions for an index

45.4. pg_amop

目录pg_amop存储关于与访问方法操作符族相关的操作符信息。
对于操作符族中的每个成员操作符，此目录中都有一行。一个成员可以是搜索操作符，
也可以是排序操作符。一个操作符可以出现在多个族中，
但在同一个族中既不能出现在多个搜索位置，也不能出现在多个排序位置（虽然不太可能，
但允许一个操作符同时用于搜索和排序）。

表 45.4. pg_amop 列

Name	Type	References	Description
amopfamily	oid	pg_opfamily.oid	The operator family this entry is for
amoplefttype	oid	pg_type.oid	Left-hand input data type of operator
amoprighttype	oid	pg_type.oid	Right-hand input data type of operator
amopstrategy	int2		Operator strategy number
amoppurpose	char		Operator purpose, either s for search or o for ordering
amopopr	oid	pg_operator.oid	OID of the operator
amopmethod	oid	pg_am.oid	Index access method operator family is for
amopsortfamily	oid	pg_opfamily.oid	The btree operator family this entry sorts according to, if an ordering operator; zero if a search operat or

“搜索”操作符项表明可以搜索该操作符族的一个索引来找到所有满足 WHERE *indexed_column operator constant* 的行。显然，这样的操作符必须返回布尔值，并且其左输入类型必须匹配索引列的数据类型。

“排序”操作符项表明可以扫描该操作符族的一个索引，按照 ORDER BY *indexed_column operator constant* 所表示的顺序返回行。这样的操作符可以返回任何可排序的数据类型，但其左输入类型同样必须匹配索引列的数据类型。ORDER BY 的确切语义由 amopsortfamily 列指定，该列必须引用操作符结果类型的一个 btree 操作符族。

> **注意**
>
> 当前假定排序操作符的排序顺序是被引用操作符族的默认值，即 ASC NULLS LAST。将来可能会通过增加额外的列来显式指定排序选项，从而放宽这一假定。

一个项的amopmethod必须和其所属操作符族的opfmethod相匹配（此处加入amopmethod是出于性能原因而对目录结构故意做出的反规范化）。此外，amoplefttype和amoprighttype也必须匹配被引用的pg_operator项的oprleft和oprright域。

45.5. pg_amproc

目录pg_amproc存储关于访问方法操作符族相关的支持函数。
属于一个操作符族的每一个支持函数在这个目录中都有一行。

表 45.5. pg_amproc 列

Name	Type	References	Description
amprocfamily	oid	pg_opfamily.oid	The operator family this entry is for
amproclefttype	oid	pg_type.oid	Left-hand input data type of associated operator
amprocrighttype	oid	pg_type.oid	Right-hand input data type of associated operator
amprocnum	int2		Support procedure number
amproc	regproc	pg_proc.oid	OID of the procedure

amproclefttype 和 amprocrighttype 字段的通常解释是，它们标识某个特定支持过程所支持的操作符的左、右输入类型。For some access methods these match the input data type(s) of the support procedure itself, for others not. There is a notion of “default” support procedures for an index, which are those with amproclefttype and amprocrighttype both equal to the index opclass's opcintype.

45.6. pg_attrdef

目录pg_attrdef存储列默认值。列的主要信息存储在pg_attribute中（见下文）。
只有在创建表或添加列时显式指定默认值的列，才会在这里有一项。

表 45.6. pg_attrdef 列

Name	Type	References	Description
adrelid	oid	pg_class.oid	The table this column belongs to
adnum	int2	pg_attribute.attnum	The number of the column
adbin	pg_node_tree		The internal repres entation of the column default value
adsrc	text		A human-readable representation of the default value

adsrc字段是历史遗留，最好不要使用，因为它不会跟踪可能影响默认值表示的外部变化。
显示默认值时，更好的方法是反向编译adbin字段（例如使用pg_get_expr）。

45.7. pg_attribute

目录pg_attribute存储关于表列的信息。数据库中每个表的每一列，在pg_attribute中都恰好有一行。（其中也包括索引的属性项，实际上，凡是在pg_class中有项的对象，都有对应的属性项。）

术语属性等同于列，这里使用它只是出于历史原因。

表 45.7. pg_attribute 列

Name	Type	References	Description
attrelid	oid	pg_class.oid	The table this column belongs to
attname	name		The column name
atttypid	oid	pg_type.oid	The data type of this column
attstattarget	int4		attstattarget controls the level of detail of statistics accumulated for this column by ANALYZE. A zero value indicates that no statistics should be collected. A negative value says to use the system default statistics target. The exact meaning of positive values is data type-dependent. For scalar data types, attstattarget is both the target number of "most common values" to collect, and the target number of histogram bins to create.
attlen	int2		A copy of pg_type.typlen of this column's type
attnum	int2		The number of the column. Ordinary columns are numbered from 1 up. System columns, such as oid, have (arbitrary) negative numbers.
attndims	int4		Number of dimensions, if the column is an array type; otherwise 0. (Presently, the number of dimensions of an array is not enforced, so any nonzero value effectively means "it's an array".)
attcacheoff	int4		Always -1 in storage, but when loaded into a row descriptor in memory this might be updated to cache the offset of the attribute within the row
atttypmod	int4		atttypmod records type-specific data supplied at table

Name	Type	References	Description
			creation time (for example, the maximum length of a `varchar` column). It is passed to type-specific input functions and length coercion functions. The value will generally be -1 for types that do not need `atttypmod`.
`attbyval`	`bool`		A copy of `pg_type.typbyval` of this column's type
`attstorage`	`char`		Normally a copy of `pg_type.typstorage` of this column's type. For TOASTable data types, this can be altered after column creation to control storage policy.
`attalign`	`char`		A copy of `pg_type.typalign` of this column's type
`attnotnull`	`bool`		This represents a not-null constraint. It is possible to change this column to enable or disable the constraint.
`atthasdef`	`bool`		This column has a default value, in which case there will be a corresponding entry in the `pg_attrdef` catalog that actually defines the value.
`attisdropped`	`bool`		This column has been dropped and is no longer valid. A dropped column is still physically present in the table, but is ignored by the parser and so cannot be accessed via SQL.
`attislocal`	`bool`		This column is defined locally in the relation. Note that a column can be locally defined and inherited simultaneously.
`attinhcount`	`int4`		The number of direct ancestors this column has. A column with a nonzero number of

Name	Type	References	Description
			ancestors cannot be dropped nor rename d.
attcollation	oid	pg_collation.oid	The defined collation of the column, or zero if the column is not of a collatable data type.
attacl	aclitem[]		Column-level access privileges, if any have been granted specif ically on this column
attoptions	text[]		Attribute-level option s, as “keyword= value” strings
attfdwoptions	text[]		属性级的外部数据包装器选项，以“keyword= value”形式的字符串

在被删除列的 pg_attribute 条目中，atttypid 被重置为零，但 attlen 以及其他从 pg_type 复制的字段仍然有效。这种安排用于应对被删除列的数据类型后来也被删除、因而不再有相应 pg_type 行的情况。attlen 和其他字段可用于解释表中一行的内容。

45.8. pg_authid

目录pg_authid包含关于数据库授权标识符（角色）的信息。角色涵盖了“用户”和“组”这两个概念。用户本质上只是设置了rolcanlogin标志的角色。
任何角色（无论是否设置了rolcanlogin）都可以拥有其他角色作为成员；参见pg_auth_members。

由于这个目录包含密码，它不应公开可读。pg_roles是建立在pg_authid之上的公开可读视图，它会隐藏密码字段。

第 20 章包含关于用户和权限管理的详细信息。

由于用户标识符是集簇范围的，pg_authid
在一个集簇的所有数据库之间共享：在一个集簇中只有一份pg_authid拷贝，而不是每个数据库一份。

表 45.8. pg_authid 列

Name	Type	Description
rolname	name	Role name
rolsuper	bool	Role has superuser privileges
rolinherit	bool	Role automatically inheri ts privileges of roles it is a member of
rolcreaterole	bool	Role can create more roles
rolcreatedb	bool	Role can create databases
rolcatupdate	bool	Role can update system catalogs directly. (Even a superuser cannot do this unless this column is true)
rolcanlogin	bool	Role can log in. That is, this role can be given as the initia

Name	Type	Description
		l session authorization identi fier
rolreplication	bool	Role is a replication role. That is, this role can initiate streaming replication (see 第 25.2.5 节) and set/unset the system backup mode using pg_start_backup and pg_stop_backup
rolconnlimit	int4	For roles that can log in, this sets maximum number of concurrent connections this role can make. -1 means no limit.
rolpassword	text	Password (possibly encryp ted); null if none. If the password is encrypted, this column will begin with the string md5 followed by a 32-character hexadecimal MD5 hash. The MD5 hash will be of the user's password concat enated to their user name. For example, if user joe has password xyzzy, PostgreSQL will store the md5 hash of xyzzyjoe. A password that does not follow that format is assumed to be unencrypted.
rolvaliduntil	timestamptz	Password expiry time (only used for password authen tication); null if no expiration

45.9. pg_auth_members

目录pg_auth_members展示了角色之间的成员关系。允许任何无环的关系集合。

由于用户标识符是集簇范围的，pg_auth_members
在一个集簇的所有数据库之间共享：在一个集簇中只有一份pg_auth_members拷贝，
而不是每个数据库一份。

表 45.9. pg_auth_members 列

Name	Type	References	Description
roleid	oid	pg_authid.oid	ID of a role that has a member
member	oid	pg_authid.oid	ID of a role that is a member of roleid
grantor	oid	pg_authid.oid	ID of the role that granted this member ship
admin_option	bool		True if member can grant membership in roleid to others

45.10. pg_cast

目录 pg_cast 存储数据类型转换路径，包括内建路径和用户自定义路径。

需要注意的是，pg_cast并不表示系统知道如何执行的所有类型转换，
它只包括那些不能从某些通用规则推导出的转换。例如，
一个域及其基础类型之间的转换并未显式地在pg_cast中展示。
另一个重要的例外是“自动的基于 I/O 的类型转换”，它们通过数据类型自己的 I/O 函数来转换成（或者转换自）text或其他字符串类型，这些转换也没有显式地在pg_cast中表示。

表 45.10. pg_cast 列

Name	Type	References	Description
castsource	oid	pg_type.oid	OID of the source data type
casttarget	oid	pg_type.oid	OID of the target data type
castfunc	oid	pg_proc.oid	The OID of the function to use to perform this cast. Zero is stored if the cast method doesn't require a function.
castcontext	char		Indicates what contexts the cast can be invoked in. e means only as an explicit cast (using CAST or :: syntax). a means implicitly in assignment to a target column, as well as explicitly. i means implicitly in expressions, as well as the other cases.
castmethod	char		Indicates how the cast is performed. f means that the function specified in the castfunc field is used. i means that the input/output functions are used. b means that the types are binary-coercible, thus no conversion is required.

在pg_cast里列出的类型转换函数必须总是以转换的源类型作为它的第一个参数类型，
并且返回转换的目标类型作为它的结果类型。一个类型转换函数最多有三个参数。
如果出现了第二个参数，必须是integer类型；它接受与目标类型关联的修饰词，
如果没有，就是 -1。如果出现了第三个参数，那么必须是boolean类型；如果该类型转换是一种明确的转换，那么它接受true，否则接受false。

在pg_cast里创建一条源类型和目标类型相同的记录是合理的，
只要相关联的函数接受多过一个参数。这样的记录代表“长度转换函数”，
它们把该类型的值转换为对特定的类型合法的值。

如果一个pg_cast的项有着不同的原类型和目标类型，
并且有一个接收多于一个参数的函数，
那么它会在一个步骤中完成从一种类型到另外一种类型的转换并应用一个长度转换。

如果没有这样的项，使用一个类型修改器的转换涉及两个步骤，
一个是在数据类型之间转换，另外一个是应用修改器。

45.11. pg_class

目录 pg_class 记录表和大多数其他具有列或者类似于表的对象。这包括索引（但另见 pg_index）、序列、视图、组合类型和 TOAST 表；见 relkind。Below, when we mean all of these kinds of objects we speak of “relations”. Not all columns are meaningful for all relation types.

表 45.11. pg_class 列

Name	Type	References	Description
relname	name		Name of the table, index, view, etc.
relnamespace	oid	pg_namespace.oid	The OID of the namespace that contains this relation
reltype	oid	pg_type.oid	The OID of the data type that corresponds to this table's row type, if any (zero for indexes, which have no pg_type entry)
reloftype	oid	pg_type.oid	For typed tables, the OID of the underlying composite type, zero for all other relations
relowner	oid	pg_authid.oid	Owner of the relation
relam	oid	pg_am.oid	If this is an index, the access method used (B-tree, hash, etc.)
relfilenode	oid		Name of the on-disk file of this relation; zero means this is a “mapped” relation whose disk file name is determined by low-level state
reltablespace	oid	pg_tablespace.oid	The tablespace in which this relation is stored. If zero, the database's default tablespace is implied. (Not meaningful if the relation has no on-disk file.)
relpages	int4		Size of the on-disk representation of this table in pages (of size BLCKSZ). This is only an estimate used by the planner. It is updated by VACUUM, ANALYZE, and a few DDL commands such as CREATE INDEX.

Name	Type	References	Description
reltuples	float4		Number of rows in the table. This is only an estimate used by the planner. It is updated by VACUUM, ANALYZE, and a few DDL commands such as CREATE INDEX.
relallvisible	int4		在表的可见性映射中被标记为全可见的页数。这只是一个由规划器使用的估计值。它被VACUUM、ANALYZE以及一些DDL命令，如CREATE INDEX所更新。
reltoastrelid	oid	pg_class.oid	OID of the TOAST table associated with this table, 0 if none. The TOAST table stores large attributes “out of line” in a second ary table.
reltoastidxid	oid	pg_class.oid	For a TOAST table, the OID of its index. 0 if not a TOAST table.
relhasindex	bool		True if this is a table and it has (or recently had) any indexes
relisshared	bool		True if this table is shared across all databases in the cluster. Only certain system catalogs (such as pg_database) are shared.
relpersistence	char		p = permanent table, u = unlogged table, t = temporary table
relkind	char		r = ordinary table, i = index, S = sequence, v = view, c = composite type, t = TOAST table, f = foreign table
relnatts	int2		Number of user columns in the relati on (system columns not counted). There must be this many corresponding entries in pg_attribute. See also pg_attribute.attnum.
relchecks	int2		Number of CHECK constraints on the table; see pg_constr aint catalog

Name	Type	References	Description
relhasoids	bool		True if we generate an OID for each row of the relation
relhaspkey	bool		True if the table has (or once had) a primar y key
relhasrules	bool		True if table has (or once had) rules; see pg_rewrite catalog
relhastriggers	bool		True if table has (or once had) trigge rs; see pg_trigger catalog
relhassubclass	bool		True if table has (or once had) any inheri tance children
relfrozenxid	xid		All transaction IDs before this one have been replaced with a permanent (“frozen”) transaction ID in this table. This is used to track whether the table needs to be vacuumed in order to prevent transaction ID wraparound or to allow pg_clog to be shrunk. Zero (InvalidTransactionId) if the relation is not a table.
relacl	aclitem[]		Access privileges; see GRANT and REVOKE for details
reloptions	text[]		Access-method-specific options, as “keyword=value” strings

pg_class中的一些布尔标志采用延迟维护：当条件成立时，保证它们为真；但当条件不再成立时，可能不会立即将它们重置为假。例如，relhasindex由CREATE INDEX设置，但它从不会被DROP INDEX清除。作为替代，VACUUM会在找到无索引表后清除其relhasindex。这种安排避免了竞争条件并且提高了并发性。

45.12. pg_constraint

目录pg_constraint存储表上的检查约束、主键约束、唯一约束、外键约束和排他约束。（列约束不会被特殊对待。每一个列约束都等价于某种表约束。）非空约束在pg_attribute目录中表示，而不是在这里。

用户定义的约束触发器（使用CREATE CONSTRAINT TRIGGER创建）也会在这个表中产生一项。

域上的检查约束也存储在这里。

表 45.12. pg_constraint 列

Name	Type	References	Description
conname	name		Constraint name (not necessarily unique!)
connamespace	oid	pg_namespace.oid	The OID of the namespace that contains this constraint
contype	char		c = check constraint, f = foreign key constraint, p = primary key constraint, u = unique constraint, t = constraint trigger, x = exclusion constraint
condeferrable	bool		Is the constraint deferrable?
condeferred	bool		Is the constraint deferred by default?
convalidated	bool		Has the constraint been validated? Currently, can only be false for foreign keys
conrelid	oid	pg_class.oid	The table this constraint is on; 0 if not a table constraint
contypid	oid	pg_type.oid	The domain this constraint is on; 0 if not a domain constraint
conindid	oid	pg_class.oid	The index supporting this constraint, if it's a unique, primary key, foreign key, or exclusion constraint; else 0
confrelid	oid	pg_class.oid	If a foreign key, the referenced table; else 0
confupdtype	char		Foreign key update action code: a = no action, r = restrict, c = cascade, n = set null, d = set default
confdeltype	char		Foreign key deletion action code: a = no action, r = restrict, c = cascade, n = set null, d = set default
confmatchtype	char		Foreign key match type: f = full, p = partial, u = simple (unspecified)
conislocal	bool		This constraint is defined locally for the relation. Note that a constraint can be locally defined

Name	Type	References	Description
			and inherited simult aneously.
coninhcount	int4		The number of direct inheritance ancestors this constraint has. A constraint with a nonzero number of ancestors cannot be dropped nor rename d.
connoinherit	bool		为真表示此约束被定义在关系本地。它是一个不可继承约束。
conkey	int2[]	pg_attribute. attnum	If a table constraint (including foreign keys, but not constr aint triggers), list of the constrained columns
confkey	int2[]	pg_attribute. attnum	If a foreign key, list of the referenced columns
conpfeqop	oid[]	pg_operator.oid	If a foreign key, list of the equality operat ors for PK = FK compar isons
conppeqop	oid[]	pg_operator.oid	If a foreign key, list of the equality operators for PK = PK compar isons
conffeqop	oid[]	pg_operator.oid	If a foreign key, list of the equality operat ors for FK = FK compar isons
conexclop	oid[]	pg_operator.oid	If an exclusion constr aint, list of the per-column exclusion operators
conbin	pg_node_tree		If a check constraint, an internal repres entation of the expres sion
consrc	text		If a check constraint, a human-readable representation of the expression

对于排他约束，conkey仅对作为简单列引用的约束元素有用。对于其他情况，conkey中会出现一个 0，必须查阅关联索引来确定被约束的表达式。（因此，conkey与该索引的pg_index.indkey具有相同的内容。）

注意

被引用对象发生变化时，consrc不会更新；例如，它不会跟踪列重命名。最好使用pg_get_constraintdef()提取检查约束的定义，而不是依赖这个字段。

> **注意**
>
> pg_class.relchecks需要和每个关系在此目录中的检查约束数量保持一致。

45.13. pg_collation

目录pg_collation描述可用的排序规则，其本质上是从 SQL 名字到操作系统区域设置类别的映射。更多信息参见第 22.2 节。

表 45.13. pg_collation 列

Name	Type	References	Description
collname	name		Collation name (unique per namesp ace and encoding)
collnamespace	oid	pg_namespace.oid	The OID of the namespace that contains this collation
collowner	oid	pg_authid.oid	Owner of the collation
collencoding	int4		Encoding in which the collation is applic able, or -1 if it works for any encoding
collcollate	name		LC_COLLATE for this collation object
collctype	name		LC_CTYPE for this collation object

注意在这个目录中的唯一键是（collname、collencoding、collnamespace），不仅仅是（collname、collnamespace）。PostgreSQL 通常会忽略所有 collencoding 既不等于当前数据库编码、也不等于 -1 的排序规则，并且禁止创建与 collencoding = -1 的项同名的新项。因此，使用限定的 SQL 名字（*schema.name*）来标识一个排序规则已经足够，即使按目录定义它并不唯一。之所以将该目录定义成这样，是因为 initdb 在集簇初始化时会用系统上所有可用的区域设置填充它，因此它必须能够容纳集簇中未来可能使用到的所有编码的项。

在template0数据库中，创建与数据库编码不匹配的排序规则可能很有用，因为它们可以匹配之后从template0克隆出的数据库的编码。目前这必须手工完成。

45.14. pg_conversion

目录pg_conversion描述编码转换函数。更多信息参见CREATE CONVERSION。

表 45.14. pg_conversion 列

Name	Type	References	Description
conname	name		Conversion name (unique within a namespace)
connamespace	oid	pg_namespace.oid	The OID of the namespace that contains this conver sion

Name	Type	References	Description
conowner	oid	pg_authid.oid	Owner of the conversion
conforencoding	int4		Source encoding ID
contoencoding	int4		Destination encoding ID
conproc	regproc	pg_proc.oid	Conversion procedure
condefault	bool		True if this is the default conversion

45.15. pg_database

目录pg_database存储有关可用数据库的信息。数据库通过CREATE DATABASE命令创建。更多关于其参数的信息请查阅第 21 章。

和大部分系统目录不同，pg_database是在集簇的所有数据库之间共享的：在一个集簇中只有一份pg_database拷贝，而不是每个数据库一份。

表 45.15. pg_database 列

Name	Type	References	Description
datname	name		Database name
datdba	oid	pg_authid.oid	Owner of the database, usually the user who created it
encoding	int4		Character encoding for this database (pg_encoding_to_char() can translate this number to the encoding name)
datcollate	name		LC_COLLATE for this database
datctype	name		LC_CTYPE for this database
datistemplate	bool		If true then this database can be used in the TEMPLATE clause of CREATE DATABASE to create a new database as a clone of this one
datallowconn	bool		If false then no one can connect to this database. This is used to protect the template0 database from being altered.
datconnlimit	int4		Sets maximum number of concurrent connections that can be made to this database. -1 means no limit.

Name	Type	References	Description
datlastsysoid	oid		Last system OID in the database; useful particularly to pg_dump
datfrozenxid	xid		All transaction IDs before this one have been replaced with a permanent (“frozen”) transaction ID in this database. This is used to track whether the database needs to be vacuumed in order to prevent transaction ID wraparound or to allow pg_clog to be shrunk. It is the minimum of the per-table pg_class.relfrozenxid values.
dattablespace	oid	pg_tablespace.oid	The default tablespace for the database. Within this database, all tables for which pg_class.reltablespace is zero will be stored in this tablespace; in particular, all the non-shared system catalogs will be there.
datacl	aclitem[]		Access privileges; see GRANT and REVOKE for details

45.16. pg_db_role_setting

目录 pg_db_role_setting为每个角色与数据库组合记录运行时配置变量的默认值。

和大部分系统目录不同，pg_db_role_setting
是在集簇的所有数据库之间共享的：在一个集簇中只有一份pg_db_role_setting拷贝，
而不是每个数据库一份。

表 45.16. pg_db_role_setting 列

Name	Type	References	Description
setdatabase	oid	pg_database.oid	The OID of the database the setting is applicable to, or zero if not database-specific
setrole	oid	pg_authid.oid	The OID of the role the setting is applicable to, or zero if not role-specific
setconfig	text[]		Defaults for run-time configuration variables

45.17. pg_default_acl

目录pg_default_acl存储要被分配给新创建对象的初始权限。

表 45.17. pg_default_acl 列

Name	Type	References	Description
defaclrole	oid	pg_authid.oid	The OID of the role associated with this entry
defaclnamespace	oid	pg_namespace.oid	与此条目关联的命名空间的 OID，如果没有则为 0
defaclobjtype	char		此条目所针对的对象类型：r = 关系（表、视图），S = 序列，f = 函数，T = 类型
defaclacl	aclitem[]		Access privileges that this type of object should have on creati on

一个pg_default_acl项表示要赋给属于指定用户的对象的初始权限。
目前有两类项：defaclnamespace = 0 的“全局”项，以及引用特定模式的“按模式”项。
如果存在全局项，则它会覆盖该对象类型通常写死的默认权限。如果存在按模式项，
则表示要将这些权限附加到全局默认权限或写死的默认权限之上。

注意，当另一个目录中的 ACL 项为 NULL 时，它表示该对象采用写死的默认权限，而不是当前 pg_default_acl 中可能存在的内容。pg_default_acl 只会在对象创建期间被查阅。

45.18. pg_depend

目录pg_depend记录数据库对象之间的依赖关系。这些信息允许DROP命令查找必须被DROP CASCADE删除的其他对象，或者在DROP RESTRICT情况下阻止删除。

另请参阅pg_shdepend，
它对在一个数据库集簇中共享的对象之间的依赖提供了相似的功能。

表 45.18. pg_depend 列

Name	Type	References	Description
classid	oid	pg_class.oid	The OID of the system catalog the depend ent object is in
objid	oid	any OID column	The OID of the specific dependent object
objsubid	int4		For a table column, this is the column number (the objid and classid refer to the table itself). For all other object types, this column is zero.
refclassid	oid	pg_class.oid	The OID of the system catalog the refere nced object is in

Name	Type	References	Description
refobjid	oid	any OID column	The OID of the specific referenced object
refobjsubid	int4		For a table column, this is the column number (the refobjid and refclassid refer to the table itself). For all other object types, this column is zero.
deptype	char		A code defining the specific semantics of this dependency relationship; see text

在所有情况下，一条pg_depend记录都表示：如果不同时删除依赖对象，就不能删除被引用对象。不过，还存在若干由deptype标识的子类型：

DEPENDENCY_NORMAL（n）

两个独立创建的对象之间的正常关系。依赖对象可以被删除而不影响被引用对象。被引用对象只能通过指定CASCADE来删除，这样依赖对象也会被删除。例如：表列对其数据类型有一个正常依赖关系。

DEPENDENCY_AUTO（a）

依赖对象可以与被引用对象分开删除，并且在被引用对象删除时应自动删除（无论使用RESTRICT还是CASCADE模式）。例如，表上的命名约束会自动依赖于该表，因此删除表时该约束也会消失。

DEPENDENCY_INTERNAL（i）

依赖对象作为被引用对象创建过程的一部分而创建，实际上只是其内部实现的一部分。对依赖对象执行DROP会被直接禁止（我们会告诉用户改为对被引用对象执行DROP）。对被引用对象执行DROP会传播到依赖对象，使其被删除，无论是否指定CASCADE。例如，为实施外键约束而创建的触发器会在内部依赖于该约束的pg_constraint项。

DEPENDENCY_EXTENSION（e）

依赖对象是作为被引用对象的扩展的一个成员（参见pg_extension）。只能通过对被引用对象执行DROP EXTENSION来删除依赖对象。在功能上，这种依赖类型与内部依赖相同，但为了清晰和简化pg_dump而单独区分。

DEPENDENCY_PIN（p）

不存在依赖对象；这种项表示系统本身依赖于被引用对象，因此该对象绝不能删除。这种项只由initdb创建。用于表示依赖对象的各列包含零。

未来可能需要其他依赖关系的变种。

45.19. pg_description

目录pg_description存储对每一个数据库对象可选的描述（注释）。描述可以通过COMMENT操作，并可使用psql的\d命令查看。在pg_description的初始内容中提供了很多内置系统对象的描述。

参见pg_shdescription，它对在一个数据库集簇中共享的对象的描述提供了相似的功能。

表 45.19. pg_description 列

Name	Type	References	Description
objoid	oid	any OID column	The OID of the object this description pertai ns to
classoid	oid	pg_class.oid	The OID of the system catalog this object appears in
objsubid	int4		For a comment on a table column, this is the column number (the objoid and classoid refer to the table itself). For all other object types, this column is zero.
description	text		Arbitrary text that serves as the descri ption of this object

45.20. pg_enum

pg_enum目录包含每一个枚举类型的项，其中包括了值和标签。
一个给定枚举值的内部表示实际上是它在pg_enum中的相关行的OID。

表 45.20. pg_enum 列

Name	Type	References	Description
enumtypid	oid	pg_type.oid	The OID of the pg_type entry owning this enum value
enumsortorder	float4		The sort position of this enum value within its enum type
enumlabel	name		The textual label for this enum value

pg_enum
行的OID值遵循一种特殊的规则：即OID的数值被保证按照其枚举类型的排序顺序进行排序。
即如果两个偶数OID属于同一枚举类型，较小的OID必然具有较小的enumsortorder值。
奇数OID值不需要遵循排序顺序。
这种规则使得枚举比较例程在很多常见情况下可以避免系统目录查找。
创建和修改枚举类型的例程将尝试尽可能地为枚举值分配偶数OID。

当一个枚举类型被创建后，其成员会被分配排序位置 1..*n*。但后来增加的成员可能会被赋予负值或分数值的enumsortorder。对这些值的唯一要求是它们必须排序正确并且保持唯一。

45.21. pg_extension

目录pg_extension存储有关已安装扩展的信息。有关扩展的细节请参见第 35.15 节。

表 45.21. pg_extension 列

Name	Type	References	Description
extname	name		Name of the extens ion

Name	Type	References	Description
extowner	oid	pg_authid.oid	Owner of the extension
extnamespace	oid	pg_namespace.oid	Schema containing the extension's exported objects
extrelocatable	bool		True if extension can be relocated to another schema
extversion	text		Version name for the extension
extconfig	oid[]	pg_class.oid	Array of regclass OIDs for the extension's configuration table(s), or NULL if none
extcondition	text[]		Array of WHERE-clause filter conditions for the extension's configuration table(s), or NULL if none

注意和大部分具有一个“namespace”列的模式不同，extnamespace
不是用来表示扩展属于该模式。扩展的名字从不用模式进行限定。extnamespace
表明该模式包含了该扩展的大部分或全部对象。如果extrelocatable为真，
则该模式事实上必须包含属于此扩展的全部模式限定的对象。

45.22. pg_foreign_data_wrapper

目录pg_foreign_data_wrapper存储外部数据包装器定义。
外部数据包装器是一种访问位于外部服务器上数据的机制。

表 45.22. pg_foreign_data_wrapper 列

Name	Type	References	Description
fdwname	name		Name of the foreign-data wrapper
fdwowner	oid	pg_authid.oid	Owner of the foreign-data wrapper
fdwhandler	oid	pg_proc.oid	References a handler function that is responsible for supplying execution routines for the foreign-data wrapper. Zero if no handler is provided
fdwvalidator	oid	pg_proc.oid	References a validator function that is responsible for checking the validity of the options given to the foreign-data wrapper, as well as options for foreign servers and user mappings using the foreign-data wrapper. Zero if no validator is provided

Name	Type	References	Description
fdwacl	aclitem[]		Access privileges; see GRANT and REVOKE for details
fdwoptions	text[]		Foreign-data wrapper specific options, as “keyword=value” strings

45.23. pg_foreign_server

目录pg_foreign_server存储外部服务器定义。外部服务器定义了外部数据的来源，例如一个远程服务器。外部服务器通过外部数据包装器来访问。

表 45.23. pg_foreign_server 列

Name	Type	References	Description
srvname	name		Name of the foreign server
srvowner	oid	pg_authid.oid	Owner of the foreign server
srvfdw	oid	pg_foreign_data_wrapper.oid	OID of the foreign-data wrapper of this foreign server
srvtype	text		Type of the server (optional)
srvversion	text		Version of the server (optional)
srvacl	aclitem[]		Access privileges; see GRANT and REVOKE for details
srvoptions	text[]		Foreign server specific options, as “keyword=value” strings

45.24. pg_foreign_table

目录pg_foreign_table包含关于外部表的辅助信息。一个外部表和普通表一样，主要由一个pg_class项表示。它的pg_foreign_table项包含外部表所特有的信息。

表 45.24. pg_foreign_table 列

Name	Type	References	Description
ftrelid	oid	pg_class.oid	OID of the pg_class entry for this foreign table
ftserver	oid	pg_foreign_server.oid	OID of the foreign server for this foreign table
ftoptions	text[]		Foreign table options, as “keyword=value” strings

45.25. pg_index

目录pg_index包含关于索引的部分信息。其他信息大部分在pg_class中。

表 45.25. pg_index 列

Name	Type	References	Description
indexrelid	oid	pg_class.oid	The OID of the pg_class entry for this index
indrelid	oid	pg_class.oid	The OID of the pg_class entry for the table this index is for
indnatts	int2		The number of columns in the index (duplicates pg_class.relnatts)
indisunique	bool		If true, this is a unique index
indisprimary	bool		If true, this index represents the primar y key of the table (indisunique should always be true when this is true)
indisexclusion	bool		If true, this index supports an exclusion constraint
indimmediate	bool		If true, the unique ness check is enforc ed immediately on insertion (irrelevant if indisunique is not true)
indisclustered	bool		If true, the table was last clustered on this index
indisvalid	bool		如果此标志和indisr eady都为真，表示此索引当前可用于查询。为假表示索引可能是不完整的：它仍必须由 INSERT/UPDATE 操作修改，但还不能安全地用于查询。如果它是唯一索引，唯一性属性也不成立。
indcheckxmin	bool		If true, queries must not use the index until the xmin of this pg_index row is below their Transaction Xmin event horizon, because the table may contain broken HOT chains with incompatible rows that they can see
indisready	bool		如果为真，表示此索引当前可以用于插入。为假表示索引必须被

Name	Type	References	Description
			INSERT/UPDATE操作忽略。
indkey	int2vector	pg_attribute.attnum	This is an array of indnatts values that indicate which table columns this index indexes. For example a value of 1 3 would mean that the first and the third table columns make up the index key. A zero in this array indicates that the correspondin g index attribute is an expression over the table columns, rather than a simple column reference.
indcollation	oidvector	pg_collation.oid	For each column in the index key, this contains the OID of the collation to use for the index.
indclass	oidvector	pg_opclass.oid	For each column in the index key, this contains the OID of the operator class to use. See pg_opclass for details.
indoption	int2vector		This is an array of indnatts values that store per-column flag bits. The meaning of the bits is defined by the index's access method.
indexprs	pg_node_tree		Expression trees (in nodeToString() representation) for index attributes that are not simple column references. This is a list with one elemen t for each zero entry in indkey. Null if all index attributes are simple references.
indpred	pg_node_tree		Expression tree (in nodeToString() representation) for partial index predic ate. Null if not a partia l index.

45.26. pg_inherits

目录pg_inherits记录有关表继承层次的信息。数据库中每个直接子表在这里都有一项。（非直接继承可以通过顺着项构成的链来确定。）

表 45.26. pg_inherits 列

Name	Type	References	Description
inhrelid	oid	pg_class.oid	The OID of the child table
inhparent	oid	pg_class.oid	The OID of the parent table
inhseqno	int4		If there is more than one direct parent for a child table (multiple inheritance), this number tells the order in which the inherited columns are to be arranged. The count starts at 1.

45.27. pg_language

目录pg_language注册了可用于编写函数或存储过程的语言。
更多关于语言处理器的信息请参阅CREATE LANGUAGE和第 38 章。

表 45.27. pg_language 列

Name	Type	References	Description
lanname	name		Name of the language
lanowner	oid	pg_authid.oid	Owner of the language
lanispl	bool		This is false for internal languages (such as SQL) and true for user-defined languages. Currently, pg_dump still uses this to determine which languages need to be dumped, but this might be replaced by a different mechanism in the future.
lanpltrusted	bool		True if this is a trusted language, which means that it is believed not to grant access to anything outside the normal SQL execution environment. Only superusers can create functions in untrusted languages.
lanplcallfoid	oid	pg_proc.oid	For noninternal languages this references the language handler, which is a special function that is responsible for executing all functions that are written in

Name	Type	References	Description
			the particular langua ge
laninline	oid	pg_proc.oid	This references a function that is respon sible for executing “inline” anonym ous code blocks (DO blocks). Zero if inline blocks are not suppor ted.
lanvalidator	oid	pg_proc.oid	This references a language validator function that is respon sible for checking the syntax and validi ty of new functions when they are create d. Zero if no validator is provided.
lanacl	aclitem[]		Access privileges; see GRANT and REVOKE for details

45.28. pg_largeobject

目录pg_largeobject保存构成“大对象”的数据。
一个大对象在被创建时会被分配一个OID。每个大对象被分解成足够小的段或“页”，以便方便地作为行存储在pg_largeobject中。每页中的数据量被定义为LOBLKSIZE（目前是BLCKSZ/4或是2 kB）。

在PostgreSQL 9.0 之前，大对象没有相关的权限结构。因此，pg_largeobject曾是公开可读的，并且可以用来获得系统中所有大对象的 OID（和内容）。但现在已经不是这样了；可使用pg_largeobject_metadata来获得大对象 OID 的列表。

表 45.28. pg_largeobject 列

Name	Type	References	Description
loid	oid	pg_largeobject_metadata.oid	Identifier of the large object that includes this page
pageno	int4		Page number of this page within its large object (counting from zero)
data	bytea		Actual data stored in the large object. This will never be more than LOBLKSIZE bytes and might be less.

pg_largeobject的每一行保存一个大对象的一个页的数据，从对象内部的字节偏移量（pageno * LOBLKSIZE）开始。现在的实现允许稀疏存储：页面可能丢失，并且可能比LOBLKSIZE字节短（即便不是最后一页）。一个大对象中丢失的区域会被读出为0。

45.29. pg_largeobject_metadata

目录pg_largeobject_metadata保持着与大对象有关的元数据。
真正的大对象数据被存储在pg_largeobject中。

表 45.29. pg_largeobject_metadata 列

Name	Type	References	Description
lomowner	oid	pg_authid.oid	Owner of the large object
lomacl	aclitem[]		Access privileges; see GRANT and REVOKE for details

45.30. pg_namespace

目录pg_namespace存储名字空间。
名字空间是SQL模式之下的结构：每个名字空间拥有一个独立的表、类型等的集合，
且其中没有名字冲突。

表 45.30. pg_namespace 列

Name	Type	References	Description
nspname	name		Name of the namespace
nspowner	oid	pg_authid.oid	Owner of the namespace
nspacl	aclitem[]		Access privileges; see GRANT and REVOKE for details

45.31. pg_opclass

目录 pg_opclass 定义索引访问方法操作符类。每个操作符类为一种特定数据类型和一种特定索引访问方法的索引列定义语义。
操作符类本质上指定一个特定的操作符族适用于一种特定的可索引列数据类型。
该操作符族中实际可用于被索引列的操作符集合，
是那些接受该列数据类型作为其左输入的操作符。

操作符类详见第 35.14 节。

表 45.31. pg_opclass 列

Name	Type	References	Description
opcmethod	oid	pg_am.oid	Index access method operator class is for
opcname	name		Name of this operator class
opcnamespace	oid	pg_namespace.oid	Namespace of this operator class
opcowner	oid	pg_authid.oid	Owner of the operator class
opcfamily	oid	pg_opfamily.oid	Operator family containing the operator class
opcintype	oid	pg_type.oid	Data type that the operator class indexes

Name	Type	References	Description
opcdefault	bool		True if this operator class is the default for opcintype
opckeytype	oid	pg_type.oid	Type of data stored in index, or zero if same as opcintype

一个操作符类的opcmethod必须匹配包含它的操作符族的opfmethod。而且，对于任何给定的opcmethod和opcintype组合，只有不超过一个pg_opclass行的opcdefault值为真。

45.32. pg_operator

目录pg_operator存储关于操作符的信息。详见CREATE OPERATOR和第 35.12 节。

表 45.32. pg_operator 列

Name	Type	References	Description
oprname	name		Name of the operator
oprnamespace	oid	pg_namespace.oid	The OID of the namespace that contains this operator
oprowner	oid	pg_authid.oid	Owner of the operator
oprkind	char		b = infix (“both”), l = prefix (“left”), r = postfix (“right”)
oprcanmerge	bool		This operator supports merge joins
oprcanhash	bool		This operator supports hash joins
oprleft	oid	pg_type.oid	Type of the left operand
oprright	oid	pg_type.oid	Type of the right operand
oprresult	oid	pg_type.oid	Type of the result
oprcom	oid	pg_operator.oid	Commutator of this operator, if any
oprnegate	oid	pg_operator.oid	Negator of this operator, if any
oprcode	regproc	pg_proc.oid	Function that implements this operator
oprrest	regproc	pg_proc.oid	Restriction selectivity estimation function for this operator
oprjoin	regproc	pg_proc.oid	Join selectivity estimation function for this operator

未使用的列包含零。例如，前缀操作符的oprleft为零。

45.33. pg_opfamily

目录pg_opfamily定义了操作符族。每一个操作符族是操作符和相关支持例程的集合，支持例程用于实现一个特定索引访问方法的语义。此外，按照访问方法指定的某种方式，一个族内的操作符都是“兼容的”。操作符族概念允许在索引中使用跨数据类型操作符，并可以使用访问方法语义的知识推导出。

操作符族最终在第 35.14 节中描述。

表 45.33. pg_opfamily 列

Name	Type	References	Description
opfmethod	oid	pg_am.oid	Index access method operator family is for
opfname	name		Name of this operator family
opfnamespace	oid	pg_namespace.oid	Namespace of this operator family
opfowner	oid	pg_authid.oid	Owner of the operator family

定义操作符族的主要信息不在它的pg_opfamily行，而是在相关的pg_amop、pg_amproc和pg_opclass行中。

45.34. pg_pltemplate

目录pg_pltemplate存储过程语言的“模板”信息。有了语言模板，就可以用简单的CREATE LANGUAGE命令在特定数据库中创建该语言，无需指定实现细节。

与大多数系统目录不同，pg_pltemplate
由一个集簇中的所有数据库共享：每个集簇只有一份pg_pltemplate，
而不是每个数据库各有一份。这样，每个数据库都可以在需要时访问这些信息。

表 45.34. pg_pltemplate列

Name	Type	Description
tmplname	name	Name of the language this template is for
tmpltrusted	boolean	True if language is considered trusted
tmpldbacreate	boolean	True if language may be created by a database owner
tmplhandler	text	Name of call handler function
tmplinline	text	Name of anonymous-block handler function, or null if none
tmplvalidator	text	Name of validator function, or null if none
tmpllibrary	text	Path of shared library that implements language
tmplacl	aclitem[]	Access privileges for template (not actually used)

目前没有用于操纵过程语言模板的命令；要更改内置信息，超级用户必须使用普通的INSERT、DELETE或UPDATE命令修改此表。

> **注意**
>
> pg_pltemplate很可能会在PostgreSQL的某个未来版本中移除，
> 改为将这些过程语言的信息保存在各自的扩展安装脚本中。

45.35. pg_proc

目录pg_proc存放有关函数（或过程）的信息。更多信息请参见CREATE FUNCTION和第 35. 3 节。

该表同时包含聚合函数和普通函数的数据。如果proisagg为真，在pg_aggregate中应该有一个相匹配的行。

表 45.35. pg_proc 列

Name	Type	References	Description
proname	name		Name of the function
pronamespace	oid	pg_namespace.oid	The OID of the namespace that contains this function
proowner	oid	pg_authid.oid	Owner of the function
prolang	oid	pg_language.oid	Implementation language or call interface of this function
procost	float4		Estimated execution cost (in units of cpu_operator_cost); if proretset, this is cost per row returned
prorows	float4		Estimated number of result rows (zero if not proretset)
provariadic	oid	pg_type.oid	Data type of the variadic array parameter's elements, or zero if the function does not have a variadic parameter
proisagg	bool		Function is an aggregate function
proiswindow	bool		Function is a window function
prosecdef	bool		Function is a security definer (i.e., a “setuid” function)
proleakproof	bool		该函数没有副作用。除返回值外，不会泄露任何关于参数的信息。任何可能因参数值而报错的函数都不是 leakproof。
proisstrict	bool		Function returns null if any call argument is null. In that case the function won't

Name	Type	References	Description
			actually be called at all. Functions that are not “strict” must be prepared to handle null inputs.
proretset	bool		Function returns a set (i.e., multiple values of the specified data type)
provolatile	char		provolatile tells whether the function's result depends only on its input arguments, or is affected by outside factors. It is i for “immutable” functions, which always deliver the same result for the same inputs. It is s for “stable” functions, whose results (for fixed inputs) do not change within a scan. It is v for “volatile” functions, whose results might change at any time. (Use v also for functions with side-effects, so that calls to them cannot get optimized away.)
pronargs	int2		Number of input arguments
pronargdefaults	int2		Number of arguments that have defaults
prorettype	oid	pg_type.oid	Data type of the return value
proargtypes	oidvector	pg_type.oid	An array with the data types of the function arguments. This includes only input arguments (including INOUT and VARIADIC arguments), and thus represents the call signature of the function.
proallargtypes	oid[]	pg_type.oid	An array with the data types of the function arguments. This includes all arguments (including OUT and INOUT arguments); however, if all the arguments are IN arguments, this field will be null. Note that subscripting is

Name	Type	References	Description
			1-based, whereas for historical reasons `proargtypes` is subscripted from 0.
`proargmodes`	`char[]`		An array with the modes of the function arguments, encoded as `i` for `IN` arguments, `o` for `OUT` arguments, `b` for `INOUT` arguments, `v` for `VARIADIC` arguments, `t` for `TABLE` arguments. If all the arguments are `IN` arguments, this field will be null. Note that subscripts correspond to positions of `proallargtypes` not `proargtypes`.
`proargnames`	`text[]`		An array with the names of the function arguments. Arguments without a name are set to empty strings in the array. If none of the arguments have a name, this field will be null. Note that subscripts correspond to positions of `proallargtypes` not `proargtypes`.
`proargdefaults`	`pg_node_tree`		Expression trees (in `nodeToString()` representation) for default values. This is a list with `pronargdefaults` elements, corresponding to the last *N* input arguments (i.e., the last *N* `proargtypes` positions). If none of the arguments have defaults, this field will be null.
`prosrc`	`text`		This tells the function handler how to invoke the function. It might be the actual source code of the function for interpreted languages, a link symbol, a file name, or just about anything else, depending on the implementation language/call convention.

Name	Type	References	Description
probin	text		Additional information about how to invoke the function. Again, the interpretation is language-specific.
proconfig	text[]		Function's local settings for run-time configuration variables
proacl	aclitem[]		Access privileges; see GRANT and REVOKE for details
protransform	regproc	pg_proc.oid	对该函数的调用可以由此处的另一个函数简化（参见第 35.9.11 节）

对于编译好的函数，包括内置函数和动态装载函数，prosrc包含函数的 C 语言名字（链接符号）。所有其他已知的语言类型，prosrc包含函数的源码文本。
除了对于动态载入的C函数之外，probin不使用，对于动态载入的C函数，
它给定了包含该函数的共享库文件的名称。

45.36. pg_range

目录pg_range存储关于范围类型的信息。它是类型在pg_type中项的补充。

表 45.36. pg_range 列

名称	类型	引用	描述
rngtypid	oid	pg_type.oid	范围类型的OID
rngsubtype	oid	pg_type.oid	该范围类型的元素类型（子类型）的OID
rngcollation	oid	pg_collation.oid	用于范围比较的排序规则的OID，如果没有则为零
rngsubopc	oid	pg_opclass.oid	用于范围比较的子类型的操作符类的OID
rngcanonical	regproc	pg_proc.oid	将一个范围值转换为规范形式的函数的OID，如果没有则为零
rngsubdiff	regproc	pg_proc.oid	返回两个元素值之差并以 double precision 表示的函数的OID；如果没有则为零

rngsubopc（如果元素类型支持排序规则，则还包括 rngcollation）决定范围类型所用的排序顺序。rngcanonical用于元素类型为离散类型的情况。rngsubdiff是可选的，
但应当提供它，以提高范围类型上的 GiST 索引性能。

45.37. pg_rewrite

目录pg_rewrite存储对于表和视图的重写规则。

表 45.37. pg_rewrite 列

Name	Type	References	Description
rulename	name		Rule name
ev_class	oid	pg_class.oid	The table this rule is for
ev_attr	int2		The column this rule is for (currently, always -1 to indicate the whole table)
ev_type	char		Event type that the rule is for: 1 = SELECT, 2 = UPDATE, 3 = INSERT, 4 = DELETE
ev_enabled	char		Controls in which session_replication_role modes the rule fires. O = rule fires in "origin" and "local" modes, D = rule is disabled, R = rule fires in "replica" mode, A = rule fires always.
is_instead	bool		True if the rule is an INSTEAD rule
ev_qual	pg_node_tree		Expression tree (in the form of a nodeToString() representation) for the rule's qualifying condition
ev_action	pg_node_tree		Query tree (in the form of a nodeToString() representation) for the rule's action

注意

如果一个表在这个目录中有任何规则，pg_class.relhasrules必须为真。

45.38. pg_seclabel

目录pg_seclabel存储数据库对象上的安全标签。安全标签可以通过SECURITY LABEL命令操纵。查看安全标签的更简单方法请见第 45.63 节。

同时请见pg_shseclabel，它对集簇共享的数据库对象的安全标签执行相似的功能。

表 45.38. pg_seclabel 列

Name	Type	References	Description
objoid	oid	any OID column	The OID of the object this security label pertains to
classoid	oid	pg_class.oid	The OID of the system catalog this object appears in

Name	Type	References	Description
objsubid	int4		For a security label on a table column, this is the column number (the objoid and classoid refer to the table itself). For all other object types, this column is zero.
provider	text		The label provider associated with this label.
label	text		The security label applied to this object.

45.39. pg_shdepend

目录pg_shdepend记录数据库对象和共享对象之间的依赖关系，例如角色。这些信息使得PostgreSQL可以确保对象在被删除时没有被其他对象引用。

另请参阅pg_depend，它对单个数据库中对象之间的依赖提供了相似的功能。

与大部分其他系统目录不同，pg_shdepend
在整个集簇的所有数据库之间共享：在每一个集簇中只有一个pg_shdepend的拷贝，
而不是每个数据库一份。

表 45.39. pg_shdepend 列

Name	Type	References	Description
dbid	oid	pg_database.oid	The OID of the database the dependent object is in, or zero for a shared object
classid	oid	pg_class.oid	The OID of the system catalog the dependent object is in
objid	oid	any OID column	The OID of the specific dependent object
objsubid	int4		For a table column, this is the column number (the objid and classid refer to the table itself). For all other object types, this column is zero.
refclassid	oid	pg_class.oid	The OID of the system catalog the referenced object is in (must be a shared catalog)
refobjid	oid	any OID column	The OID of the specific referenced object
deptype	char		A code defining the specific semantics of this dependency relationship; see text

在所有情况下，一条pg_shdepend记录都表示：如果不同时删除依赖对象，
就不能删除被引用对象。不过，还存在若干由deptype标识的子类型：

SHARED_DEPENDENCY_OWNER (o)

被引用对象（必须是一个角色）是依赖对象的拥有者。

SHARED_DEPENDENCY_ACL (a)

被引用对象（必须是角色）出现在依赖对象的 ACL（访问控制列表，即权限列表）中。（不会为对象拥有者创建 SHARED_DEPENDENCY_ACL 项，因为对象拥有者无论如何都会有一个 SHARED_DEPENDENCY_OWNER 项。）

SHARED_DEPENDENCY_PIN (p)

不存在依赖对象；这种项表示系统本身依赖于被引用对象，因此该对象绝不能删除。这种项只由initdb创建。用于表示依赖对象的各列包含零。

将来可能还需要其他依赖类型。特别要注意的是，当前定义只支持把角色作为被引用对象。

45.40. pg_shdescription

目录pg_shdescription存储共享数据库对象的可选描述（注释）。
这些描述可以通过COMMENT命令设置或修改，并且可以使用psql的\d命令查看。

另请参阅pg_description，它对单个数据库内对象的描述提供了相似的功能。

与大部分其他系统目录不同，pg_shdescription
在整个集簇的所有数据库之间共享：在每一个集簇中只有一个pg_shdescription的拷贝，而不是每个数据库一份。

表 45.40. pg_shdescription 列

Name	Type	References	Description
objoid	oid	any OID column	The OID of the object this description pertai ns to
classoid	oid	pg_class.oid	The OID of the system catalog this object appears in
description	text		Arbitrary text that serves as the descri ption of this object

45.41. pg_shseclabel

目录pg_shseclabel存储共享数据库对象上的安全标签。安全标签可以通过SECURITY LABEL命令操纵。查看安全标签的更简单方法请见第 45.63 节。

另请参阅pg_seclabel，它对单个数据库中对象的安全标签提供了相似的功能。

与大部分其他系统目录不同，pg_shseclabel
在整个集簇的所有数据库之间共享：在每一个集簇中只有一个pg_shseclabel的拷贝，而不是每个数据库一份。

表 45.41. pg_shseclabel 列

名称	类型	引用	描述
objoid	oid	任意 OID 列	该安全标签依附的对象的OID
classoid	oid	pg_class.oid	该对象所在的系统目录的 OID

名称	类型	引用	描述
provider	text		与该标签相关的标签提供者。
label	text		应用于该对象的安全标签。

45.42. pg_statistic

目录pg_statistic存储有关数据库内容的统计数据。其中的项由ANALYZE创建，查询规划器会使用这些数据来进行查询规划。注意所有的统计数据天然就是近似的，即使它刚刚被更新。

通常，每个已分析的表列都有一个条目，其中stainherit = false。如果表具有继承子表，则还会创建第二个条目，其中stainherit = true。此行表示继承树上列的统计信息，即你可以通过SELECT *column* FROM *table**看到的数据的统计信息，而stainherit = false行表示SELECT *column* FROM ONLY *table*的结果。

pg_statistic也存储关于索引表达式值的统计数据，就好像它们是真正的数据列，但在这种情况中starelid指索引。对一个普通非表达式索引列不会创建项，因为它将是底层表列的项的冗余。当前，索引表达式的项都具有stainherit = false。

由于不同种类的数据可能适合不同种类的统计信息，pg_statistic 在设计上尽量不对所存储的统计信息种类作出假定。只有极为通用的统计信息（例如空值情况）才在 pg_statistic 中有专用的列。其余统计信息都存储在“槽位”中。每个槽位都是一组相关的列，其内容由其中一列的代码编号来标识。更多信息见 src/include/catalog/pg_statistic.h。

pg_statistic不应公开可读，因为即使是表内容的统计信息也可能被认为是敏感的（例如，一个工资列的最大值和最小值就可能非常引人关注）。pg_stats是建立在pg_statistic之上的公开可读视图，它只会显示当前用户可读取的表的信息。

表 45.42. pg_statistic 列

Name	Type	References	Description
starelid	oid	pg_class.oid	The table or index that the described column belongs to
staattnum	int2	pg_attribute.attnum	The number of the described column
stainherit	bool		If true, the stats include inheritance child columns, not just the values in the specified relation
stanullfrac	float4		The fraction of the column's entries that are null
stawidth	int4		The average stored width, in bytes, of nonnull entries
stadistinct	float4		The number of distinct nonnull data values in the column. A value greater than zero is the actual number of distinct values. A

Name	Type	References	Description
			value less than zero is the negative of a multiplier for the number of rows in the table; for example, a column in which about 80% of the values are nonnul l and each nonnull value appears about twice on average could be represented by `stadistinct` = -0. 4. A zero value means the number of distinct values is unknown.
`stakind`*N*	`int2`		A code number indica ting the kind of statis tics stored in the *N* th "slot" of the `pg_statistic` row.
`staop`*N*	`oid`	`pg_operator.oid`	An operator used to derive the statis tics stored in the *N*th "slot". For exampl e, a histogram slot would show the < operator that define s the sort order of the data.
`stanumbers`*N*	`float4[]`		Numerical statistics of the appropriate kind for the *N*th "slot", or null if the slot kind does not involve numerical values
`stavalues`*N*	`anyarray`		Column data values of the appropriate kind for the *N*th "slot", or null if the slot kind does not store any data values. Each array's element values are actually of the specific column's data type, so there is no way to define these columns' type more specifically than `anyarray`.

45.43. pg_tablespace

目录`pg_tablespace`存储关于可用表空间的信息。
表可以被放置在特定表空间中以实现磁盘布局的管理。

与大部分其他系统目录不同，`pg_tablespace`
在整个集簇的所有数据库之间共享：在每一个集簇中只有一个`pg_tablespace`的拷贝，
而不是每个数据库一份。

表 45.43. pg_tablespace 列

Name	Type	References	Description
spcname	name		Tablespace name
spcowner	oid	pg_authid.oid	Owner of the tables pace, usually the user who created it
spcacl	aclitem[]		Access privileges; see GRANT and REVOKE for details
spcoptions	text[]		Tablespace-level options, as “keywor d=value” strings

45.44. pg_trigger

目录pg_trigger存储表和视图上的触发器。详见CREATE TRIGGER。

表 45.44. pg_trigger 列

Name	Type	References	Description
tgrelid	oid	pg_class.oid	The table this trigger is on
tgname	name		Trigger name (must be unique among triggers of same table)
tgfoid	oid	pg_proc.oid	The function to be called
tgtype	int2		Bit mask identifying trigger firing condit ions
tgenabled	char		Controls in which session_replication_ role modes the trigge r fires. O = trigger fires in “origin” and “local” modes, D = trigger is disabled, R = trigger fires in “replic a” mode, A = trigger fires always.
tgisinternal	bool		True if trigger is intern ally generated (usuall y, to enforce the constraint identified by tgconstraint)
tgconstrrelid	oid	pg_class.oid	The table referenced by a referential integr ity constraint
tgconstrindid	oid	pg_class.oid	The index supporting a unique, primary key, or referential integrity constraint
tgconstraint	oid	pg_constraint.oid	The pg_constraint entry associated with the trigger, if any

Name	Type	References	Description
tgdeferrable	bool		True if constraint trigger is deferrable
tginitdeferred	bool		True if constraint trigger is initially deferred
tgnargs	int2		Number of argume nt strings passed to trigger function
tgattr	int2vector	pg_attribute. attnum	Column numbers, if trigger is column-specific; otherwise an empty array
tgargs	bytea		Argument strings to pass to trigger, each NULL-terminated
tgqual	pg_node_tree		Expression tree (in nodeToString() representation) for the trigger's WHEN condition, or null if none

当前，列限定触发器只支持UPDATE事件，因此tgattr只用于这种事件类型。tgtype也可以包含用于其他事件类型的位，但那些事件类型对应的是表级触发器，并且会忽略tgattr。

> **注意**
>
> 当tgconstraint非零时，tgconstrrelid、tgconstrindid、tgdeferrable和tginitdeferred与被引用的pg_constraint 项有很大的冗余。但是，存在将一个不可延迟触发器关联到一个可延迟约束的可能性：外键约束可以有一些可延迟和一些不可延迟触发器。

> **注意**
>
> 如果一个关系在本目录中拥有任何触发器，其pg_class.relhastriggers必须为真。

45.45. pg_ts_config

pg_ts_config系统目录包含表示文本检索配置的条目。一个配置指定某个特定的文本检索解析器，以及针对该解析器每种输出词元类型所定义的词典列表。解析器记录在pg_ts_config条目中，而词元到词典的映射则由pg_ts_config_map中的辅助项定义。

PostgreSQL的文本检索特性在第 12 章中有更详尽的描述。

表 45.45. pg_ts_config 列

Name	Type	References	Description
cfgname	name		Text search config uration name

Name	Type	References	Description
cfgnamespace	oid	pg_namespace.oid	The OID of the namespace that contains this configuration
cfgowner	oid	pg_authid.oid	Owner of the configuration
cfgparser	oid	pg_ts_parser.oid	The OID of the text search parser for this configuration

45.46. pg_ts_config_map

pg_ts_config_map 系统目录中的条目说明了，对于每个文本检索配置所用解析器的每种输出词元类型，应当查询哪些文本检索词典以及查询的顺序。

PostgreSQL的文本检索特性在第 12 章中有更详尽的描述。

表 45.46. pg_ts_config_map 列

Name	Type	References	Description
mapcfg	oid	pg_ts_config.oid	The OID of the pg_ts_config entry owning this map entry
maptokentype	integer		A token type emitted by the configuration's parser
mapseqno	integer		Order in which to consult this entry (lower mapseqnos first)
mapdict	oid	pg_ts_dict.oid	The OID of the text search dictionary to consult

45.47. pg_ts_dict

pg_ts_dict系统目录包含定义文本检索词典的项。一个词典依赖于一个文本检索模板，它指定了所有需要的实现函数，词典本身则为模板支持的用户可设置参数提供值。这种分工允许普通用户创建词典。参数由一个文本串dictinitoption定义，其格式和意义随着模板而变化。

PostgreSQL的文本检索特性在第 12 章中有更详尽的描述。

表 45.47. pg_ts_dict 列

Name	Type	References	Description
dictname	name		Text search dictionary name
dictnamespace	oid	pg_namespace.oid	The OID of the namespace that contains this dictionary
dictowner	oid	pg_authid.oid	Owner of the dictionary

Name	Type	References	Description
dicttemplate	oid	pg_ts_template.oid	The OID of the text search template for this dictionary
dictinitoption	text		Initialization option string for the template

45.48. pg_ts_parser

pg_ts_parser系统目录包含定义文本检索解析器的项。
一个解析器负责将输入文本分割成词位并为每一个词位分配一个词元类型。
由于一个解析器必须用 C 语言级别的函数实现，创建新解析器的工作只限于数据库的超级用户。

PostgreSQL的文本检索特性在第 12 章中有更详尽的描述。

表 45.48. pg_ts_parser 列

Name	Type	References	Description
prsname	name		Text search parser name
prsnamespace	oid	pg_namespace.oid	The OID of the namespace that contains this parser
prsstart	regproc	pg_proc.oid	OID of the parser's startup function
prstoken	regproc	pg_proc.oid	OID of the parser's next-token function
prsend	regproc	pg_proc.oid	OID of the parser's shutdown function
prsheadline	regproc	pg_proc.oid	OID of the parser's headline function
prslextype	regproc	pg_proc.oid	OID of the parser's lextype function

45.49. pg_ts_template

pg_ts_template系统目录包含定义文本检索模板的项。
一个模板是一类文本检索词典的实现骨架。由于一个模板必须用 C 语言级别的函数实现，新模板的创建只限于数据库超级用户。

PostgreSQL的文本检索特性在第 12 章中有更详尽的描述。

表 45.49. pg_ts_template 列

Name	Type	References	Description
tmplname	name		Text search template name
tmplnamespace	oid	pg_namespace.oid	The OID of the namespace that contains this templa te
tmplinit	regproc	pg_proc.oid	OID of the template's initialization function
tmpllexize	regproc	pg_proc.oid	OID of the template's lexize function

45.50. pg_type

目录pg_type存储有关数据类型的信息。基础类型和枚举类型（标量类型）使用CREATE TYPE创建，而域使用CREATE DOMAIN创建。数据库中的每一个表都会有一个自动创建的复合类型，用于表示表的行结构。也可以使用CREATE TYPE AS创建复合类型。

表 45.50. pg_type 列

Name	Type	References	Description
typname	name		数据类型的名字
typnamespace	oid	pg_namespace.oid	包含此类型的名字空间的OID
typowner	oid	pg_authid.oid	类型的拥有者
typlen	int2		对于一个固定尺寸的类型，typlen是该类型内部表示的字节数。对于一个变长类型，typlen为负值。-1表示一个“varlena”类型（具有长度字），-2表示一个以空值结尾的C字符串。
typbyval	bool		typbyval判断内部例程传递这个类型的数值时是通过传值还是传引用。如果typlen不是1、2或4（或者在Datum为8字节的机器上为8），因此typbyval最好是假。变长类型总是传引用。注意即使长度允许传值，typbyval也可以为假。
typtype	char		typtype可以是：b表示一个基础类型，c表示一个复合类型（例如一个表的行类型），d表示一个域，e表示一个枚举类型，p表示一个伪类型，r表示一个范围类型。另请参阅typrelid和typbasetype。
typcategory	char		typcategory是一种任意的数据类型分类，它被分析器用来决定哪种隐式转换“更好”。参见表 45.51。
typispreferred	bool		如果此类型在它的typcategory中是一个更好的转换目标，此列为真
typisdefined	bool		如果此类型已被定义则为真，如果此类型只是一个表示还未定义类型的占位符则为假。当typisdefined为假，除了类型名字、名字空

Name	Type	References	Description
			间和OID之外什么都不能被依赖。
typdelim	char		在分析数组输入时，分隔两个此类型值的字符。注意该分隔符是与数组元素数据类型相关联的，而不是和数组的数据类型关联。
typrelid	oid	pg_class.oid	如果这是一个复合类型（见typtype），那么此列指向pg_class中定义对应表的项（对于自由存在的复合类型，pg_class项并不表示一个表，但不管怎样该类型的pg_attribute项需要链接到它）。对非复合类型此列为零。
typelem	oid	pg_type.oid	如果typelem不为 0，则它标识pg_type中的另一行。此时可以像数组一样对当前类型使用下标，得到typelem类型的值。“真正的”数组类型是变长的（typlen = -1），但某些定长类型（typlen > 0）的typelem也非零，例如name和point。如果定长类型具有typelem，则其内部表示必须是若干个typelem数据类型的值，不含其他数据。变长数组类型的首部由数组子例程定义。
typarray	oid	pg_type.oid	如果typarray不是零，则它标识pg_type中的另一行，这一行是一个将此类型作为元素的“真的”数组类型
typinput	regproc	pg_proc.oid	输入转换函数（文本格式）
typoutput	regproc	pg_proc.oid	输出转换函数（文本格式）
typreceive	regproc	pg_proc.oid	输入转换函数（二进制格式），如果没有则为零
typsend	regproc	pg_proc.oid	输出转换函数（二进制格式），如果没有则为零
typmodin	regproc	pg_proc.oid	类型修改器输入函数，如果类型没有提供修改器则为零
typmodout	regproc	pg_proc.oid	类型修改器输出函数，或为零以使用标准格式
typanalyze	regproc	pg_proc.oid	自定义ANALYZE函数，零表示使用标准函数

Name	Type	References	Description
typalign	char		typalign表示存储此类型值时所要求的对齐方式。它适用于磁盘存储，以及该值在 PostgreSQL 内部的大多数表示形式。当多个值连续存放时，例如在磁盘上完整行的表示中，这种类型的数据前会插入填充，使其从指定边界开始。对齐参考点是该序列中第一个数据的起始位置。 可能的值为： • c = char对齐，即不需要对齐。 • s = short对齐（在大部分机器上为2字节）。 • i = int对齐（在大部分机器上为4字节）。 • d = double对齐（在很多机器上为8字节，但绝不是全部）。 **注意** 对于系统表中使用的类型，关键的是在pg_type定义的大小和对齐与编译器在表示表行的结构体中布局列的方式一致。
typstorage	char		typstorage用于指示 varlena 类型（即typlen = -1）是否支持TOAST，以及这种类型的属性应采用什么默认策略。可能的值为： • p：值必须始终以普通方式存储。 • e：值可以存储在一个“二级”关系中（如果关系有这样的关系，参见pg_class.reltoastrelid）。 • m：值可以被压缩并内联存储。

Name	Type	References	Description
			• x：值可以被压缩并内联存储，或存储在“二级”存储中。 注意，m列也可以被移动到二级存储，但只能作为最后手段（e和x列会先被移动）。
typnotnull	bool		typnotnull表示类型上的一个非空约束。只用于域。
typbasetype	oid	pg_type.oid	如果这是一个域（见typtype），则typbasetype标识该域所基于的类型。如果此类型不是域，则为 0。
typtypmod	int4		域使用typtypmod来记录被应用于它们基础类型的typmod（如果基础类型不使用typmod，则为-1）。如果此类型不是一个域则为-1。
typndims	int4		对于基于数组的域，typndims是数组维度数（即，typbasetype是一个数组类型）。对于不是基于数组的域的类型，此列为零。
typcollation	oid	pg_collation.oid	typcollation指定此类型的排序规则。如果类型不支持排序规则，此列为零。支持排序规则的基础类型在这里会有DEFAULT_COLLATION_OID。基于支持排序规则的类型的域，如果为其指定了排序规则 OID，则它的排序规则 OID 可以不同于其基础类型的排序规则 OID。
typdefaultbin	pg_node_tree		如果typdefaultbin非空，那么它就是该类型默认表达式的nodeToString()表示形式。此列只用于域。
typdefault	text		如果某类型没有相关默认值，那么typdefault为空。如果typdefaultbin不为空，那么typdefault必须包含由typdefaultbin表示的默认表达式的人类可读版本。如果typdefaultbin为空而typdefault不为空，则typdefault是该类型默认值的外部表示形式，它可以被交

Name	Type	References	Description
			给该类型的输入转换器来产生一个常量。
typacl	aclitem[]		访问权限，详见GRANT和REVOKE

表 45.51列出了typcategory的系统定义值。任何未来对此列表的增加都将是大写ASCII字母。所有其他ASCII字符都保留给用户定义的类别。

表 45.51. typcategory Codes

Code	Category
A	Array types
B	Boolean types
C	Composite types
D	Date/time types
E	Enum types
G	Geometric types
I	Network address types
N	Numeric types
P	Pseudo-types
R	范围类型
S	String types
T	Timespan types
U	User-defined types
V	Bit-string types
X	unknown type

45.51. pg_user_mapping

目录pg_user_mapping存储从本地用户到远程的映射。
对这个目录的访问对普通用户有限制，可使用视图pg_user_mappings替代。

表 45.52. pg_user_mapping 列

Name	Type	References	Description
umuser	oid	pg_authid.oid	将要被映射的本地角色的OID，如果用户映射是公共的则为零
umserver	oid	pg_foreign_server.oid	包含此映射的外部服务器的OID
umoptions	text[]		用户映射专用的选项，以“keyword=value”字符串形式

45.52. 系统视图

除了系统目录之外，PostgreSQL还提供了一些内置视图。
一些系统视图提供了对系统目录中一些常用查询的便捷访问。
其他视图则提供了对内部服务器状态的访问。

信息模式（第 34 章）提供了一组与系统视图功能重叠的替代视图。由于信息模式是SQL标准，而这里描述的视图是PostgreSQL特有的，所以如果信息模式提供了所需的全部信息，通常最好使用信息模式。

表 45.53列出了这里描述的系统视图。每个视图的更详细文档见下文。还有一些额外的视图提供了对统计信息收集器结果的访问；它们在表 27.1中描述。

除非另有说明，否则这里描述的所有视图都是只读的。

表 45.53. 系统视图

View Name	Purpose
pg_available_extensions	available extensions
pg_available_extension_versions	available versions of extensions
pg_cursors	open cursors
pg_group	groups of database users
pg_indexes	indexes
pg_locks	currently held locks
pg_prepared_statements	prepared statements
pg_prepared_xacts	prepared transactions
pg_roles	database roles
pg_rules	rules
pg_seclabels	security labels
pg_settings	parameter settings
pg_shadow	database users
pg_stats	planner statistics
pg_tables	tables
pg_timezone_abbrevs	time zone abbreviations
pg_timezone_names	time zone names
pg_user	database users
pg_user_mappings	user mappings
pg_views	views

45.53. pg_available_extensions

pg_available_extensions视图列出了可供安装的扩展。另请参阅pg_extension目录，显示当前已安装的扩展。

表 45.54. pg_available_extensions 列

Name	Type	Description
name	name	Extension name
default_version	text	Name of default version, or NULL if none is specified
installed_version	text	Currently installed version of the extension, or NULL if not installed
comment	text	Comment string from the extension's control file

pg_available_extensions视图是只读的。

45.54. pg_available_extension_versions

pg_available_extension_versions视图列出了可供安装的特定扩展版本。
另请参阅pg_extension目录，显示当前已安装的扩展。

表 45.55. pg_available_extension_versions 列

Name	Type	Description
name	name	Extension name
version	text	Version name
installed	bool	True if this version of this extension is currently installed
superuser	bool	True if only superusers are allowed to install this extens ion
relocatable	bool	True if extension can be relocated to another schema
schema	name	Name of the schema that the extension must be installed into, or NULL if partially or fully relocatable
requires	name[]	Names of prerequisite extens ions, or NULL if none
comment	text	Comment string from the extension's control file

pg_available_extension_versions视图是只读的。

45.55. pg_cursors

该pg_cursors视图列出当前可用的游标。游标可以通过几种方式定义：

- 通过SQL中的DECLARE语句
- 通过前端/后端协议中的Bind消息，如第 46.2.3 节中所述
- 通过服务器编程接口(SPI)，如第 43.1 节中所述

该pg_cursors视图会显示通过上述任意方式创建的游标。
游标仅在定义它的事务持续期间存在，除非将其声明为WITH HOLD。因此，不可保持的游标只会在此视图中保留到创建它的事务结束为止。

> **注意**
>
> 游标在内部用于实现PostgreSQL的某些组件，如过程语言。因此，pg_cursors视图可能包含用户未明确创建的游标。

表 45.56. pg_cursors 列

Name	Type	Description
name	text	The name of the cursor

Name	Type	Description
statement	text	The verbatim query string submitted to declare this cursor
is_holdable	boolean	true if the cursor is holdable (that is, it can be accessed after the transaction that declared the cursor has committed); false otherwise
is_binary	boolean	true if the cursor was declared BINARY; false otherwise
is_scrollable	boolean	true if the cursor is scrollable (that is, it allows rows to be retrieved in a nonsequential manner); false otherwise
creation_time	timestamptz	The time at which the cursor was declared

pg_cursors视图是只读的。

45.56. pg_group

视图pg_group存在是为了向后兼容性：它模拟了在PostgreSQL版本8.1之前存在的目录。它显示了所有被标记为非rolcanlogin的角色的名称和成员，这是对被用作组的角色集合的近似。

表 45.57. pg_group 列

Name	Type	References	Description
groname	name	pg_authid.rolname	Name of the group
grosysid	oid	pg_authid.oid	ID of this group
grolist	oid[]	pg_authid.oid	An array containing the IDs of the roles in this group

45.57. pg_indexes

视图pg_indexes提供了有关数据库中每个索引的有用信息。

表 45.58. pg_indexes 列

Name	Type	References	Description
schemaname	name	pg_namespace.nspname	Name of schema containing table and index
tablename	name	pg_class.relname	Name of table the index is for
indexname	name	pg_class.relname	Name of index
tablespace	name	pg_tablespace.spcname	Name of tablespace containing index (null if default for database)
indexdef	text		Index definition (a reconstructed CREATE INDEX command)

45.58. pg_locks

视图pg_locks提供了对数据库服务器中打开的事务持有的锁的信息的访问。
有关锁定的更多讨论，请参见第 13 章。

pg_locks包含每个活动的可锁定对象、请求的锁模式和相关事务的一行。因此，
如果多个事务正在持有或等待锁定它，同一可锁定对象可能会出现多次。但是，
当前没有任何锁定的对象将不会出现。

有几种不同类型的可锁定对象：整个关系（例如，表），关系的单个页面，
关系的单个元组，事务ID（虚拟和永久ID均包括），
以及一般的数据库对象（由类OID和对象OID标识，与pg_description或 pg_depend中的方式相同）。此外，还可以对具有用户定义含义的数字施加“咨询锁”。

表 45.59. pg_locks 列

Name	Type	References	Description
locktype	text		Type of the lockable object: relation, extend, page, tuple, transactionid, virtualxid, object, userlock, or advisory
database	oid	pg_database.oid	OID of the database in which the object exists, or zero if the object is a shared object, or null if the object is a transaction ID
relation	oid	pg_class.oid	OID of the relation, or null if the object is not a relation or part of a relation
page	integer		Page number within the relation, or null if the object is not a tuple or relation page
tuple	smallint		Tuple number within the page, or null if the object is not a tuple
virtualxid	text		Virtual ID of a transaction, or null if the object is not a virtual transaction ID
transactionid	xid		ID of a transaction, or null if the object is not a transaction ID
classid	oid	pg_class.oid	OID of the system catalog containing the object, or null if the object is not a general database object
objid	oid	any OID column	OID of the object within its system catalog, or null if the

Name	Type	References	Description
			object is not a general database object
objsubid	smallint		Column number targeted by the lock (the classid and objid refer to the table itself), or zero if the target is some other general database object, or null if the target is not a general database object
virtualtransaction	text		Virtual ID of the transaction that is holding or awaiting this lock
pid	integer		Process ID of the server process holding or awaiting this lock, or null if the lock is held by a prepared transaction
mode	text		Name of the lock mode held or desired by this process (see 第 13.3.1 节 and 第 13.2.3 节)
granted	boolean		True if lock is held, false if lock is awaited
fastpath	boolean		如果锁通过快速路径获得则为真，通过主锁表获得则为假

在表示由指定事务持有的锁的行中，granted 为真。为假表示该事务当前正在等待获取此锁，这意味着某个其他事务正在同一可锁定对象上持有冲突的锁模式。等待的事务会休眠，直到另一个锁被释放（或检测到死锁情况）。单个事务一次最多只能等待获取一个锁。

每个事务在其整个持续期内都对其虚拟事务ID持有独占锁。
如果为事务分配了永久ID（通常仅在事务改变数据库状态时才会发生），
它还会对其永久事务ID持有独占锁直到事务结束。
当一个事务发现有必要专门等待另一个事务时，
它会尝试获取另一个事务ID（根据情况是虚拟ID还是永久ID）的共享锁。
只有当另一个事务终止并释放其锁时，这才会成功。

虽然元组是一种可锁定的对象类型，但关于行级锁的信息存储在磁盘上，而不是内存中，
因此行级锁通常不会出现在此视图中。如果一个事务正在等待行级锁，
它通常会出现在视图中，等待当前持有该行锁的永久事务ID。

咨询锁可以在由单个 bigint 值或两个整数值组成的键上获取。bigint 键显示时，其高位一半放在 classid 列，低位一半放在 objid 列，而 objsubid 等于 1。整数键显示时，第一个键放在 classid 列，第二个键放在 objid 列，而 objsubid 等于 2。The actual meaning of the keys is up to the user. Advisory locks are local to each database, so the database column is meaningful for an advisory lock.

pg_locks提供了集簇中所有锁的全局视图，不仅包括与当前数据库相关的锁。虽然它的relation列可以与pg_class.oid 连接来识别被锁定的关系，但这仅对当前数据库中的关系有效（即 database 列为当前数据库的 OID 或零的那些关系）。

pid 列可以连接到 pg_stat_activity 视图的 pid 列，以获得关于持有或等待持有每个锁的会话的更多信息，例如

```
SELECT * FROM pg_locks pl LEFT JOIN pg_stat_activity psa
    ON pl.pid = psa.procpid;
```

Also, if you are using prepared transactions, the `virtualtransaction` column can be joined to the `transaction` column of the `pg_prepared_xacts` view to get more information on prepared transactions that hold locks. (A prepared transaction can never be waiting for a lock, but it continues to hold the locks it acquired while running.) For example:

```
SELECT * FROM pg_locks pl LEFT JOIN pg_prepared_xacts ppx
    ON pl.virtualtransaction = '-1/' || ppx.transaction;
```

`pg_locks`视图显示来自常规锁管理器和谓词锁管理器的数据，这两个是独立的系统；此外，常规锁管理器将其锁分为常规锁和快速路径锁。不能保证这些数据完全一致。
当查询该视图时，快速路径锁的数据（具有`fastpath = true`）从每个后端逐个收集，而不会冻结整个锁管理器的状态，因此在收集信息时可能会发生锁的获取或释放。但请注意，这些锁已知不会与当前放置的任何其他锁发生冲突。在查询所有后端的快速路径锁后，剩余的常规锁管理器将作为一个单元被锁定，
并且所有剩余锁的一致快照将作为一个原子操作收集。解锁常规锁管理器后，
谓词锁管理器类似地被锁定，并且所有谓词锁将作为一个原子操作收集。因此，
除了快速路径锁外，每个锁管理器将提供一致的结果集，
但由于我们不同时锁定两个锁管理器，
因此在询问常规锁管理器后和在询问谓词锁管理器前，可能会发生锁的获取或释放。

如果这个视图被非常频繁地访问，锁定常规和（或）
谓词锁管理器可能会对数据库性能产生一些影响。
锁仅在获取来自锁管理器的数据所需的最短时间内保持，
但这并不能完全消除性能影响的可能性。

45.59. pg_prepared_statements

`pg_prepared_statements` 视图显示当前会话中所有可用的预备语句。关于预备语句的更多信息，参见PREPARE。

`pg_prepared_statements` 为每个预备语句包含一行。创建新的预备语句时，会向该视图添加一行；释放预备语句时（例如通过DEALLOCATE命令）则会移除该行。

表 45.60. pg_prepared_statements 列

Name	Type	Description
`name`	`text`	The identifier of the prepared statement
`statement`	`text`	The query string submitted by the client to create this prepared statement. For prepared statements created via SQL, this is the `PREPARE` statement submitted by the client. For prepared statements created via the frontend/backend protocol, this is the text of the prepared statement itself.
`prepare_time`	`timestamptz`	The time at which the prepared statement was created
`parameter_types`	`regtype[]`	The expected parameter types for the prepared statement in the form of an array of `regtype`. The OID corresponding to an element of this array can be obtained by

Name	Type	Description
		casting the regtype value to oid.
from_sql	boolean	true if the prepared statement was created via the PREPARE SQL statement; false if the statement was prepared via the frontend/backend protocol

pg_prepared_statements视图是只读的。

45.60. pg_prepared_xacts

视图pg_prepared_xacts显示当前准备进行两阶段提交的事务的信息 (详见PREPARE TRANSACTION)。

pg_prepared_xacts 为每个预备事务包含一行。事务提交或回滚时，对应条目会被移除。

表 45.61. pg_prepared_xacts 列

Name	Type	References	Description
transaction	xid		Numeric transaction identifier of the prepared transaction
gid	text		Global transaction identifier that was assigned to the transaction
prepared	timestamp with time zone		Time at which the transaction was prepared for commit
owner	name	pg_authid.rolname	Name of the user that executed the transaction
database	name	pg_database.datname	Name of the database in which the transaction was executed

当访问pg_prepared_xacts视图时，内部事务管理器数据结构会被暂时锁定，
并为视图制作一份副本用以显示。这确保视图生成一致的结果集，
同时不会不必要地阻塞正常操作。尽管如此，如果频繁访问该视图，
可能会对数据库性能产生一些影响。

45.61. pg_roles

视图pg_roles提供数据库角色的信息。它本质上是 pg_authid 的公开可读视图，并隐藏密码字段。

此视图显式显示底层表的 OID 列，因为与其他目录连接时需要此列。

表 45.62. pg_roles 列

Name	Type	References	Description
rolname	name		Role name
rolsuper	bool		Role has superuser privileges

Name	Type	References	Description
rolinherit	bool		Role automatically inherits privileges of roles it is a member of
rolcreaterole	bool		Role can create more roles
rolcreatedb	bool		Role can create databases
rolcatupdate	bool		Role can update system catalogs directly. (Even a superuser cannot do this unless this column is true)
rolcanlogin	bool		Role can log in. That is, this role can be given as the initial session authorization identifier
rolreplication	bool		Role is a replication role. That is, this role can initiate streaming replication (see 第 25.2.5 节) and set/unset the system backup mode using pg_start_backup and pg_stop_backup
rolconnlimit	int4		For roles that can log in, this sets maximum number of concurrent connections this role can make. -1 means no limit.
rolpassword	text		Not the password (always reads as ********)
rolvaliduntil	timestamptz		Password expiry time (only used for password authentication); null if no expiration
rolconfig	text[]		Role-specific defaults for run-time configuration variables
oid	oid	pg_authid.oid	ID of role

45.62. pg_rules

视图pg_rules提供了有关查询重写规则的有用信息。

表 45.63. pg_rules 列

Name	Type	References	Description
schemaname	name	pg_namespace.nspname	Name of schema containing table
tablename	name	pg_class.relname	Name of table the rule is for

Name	Type	References	Description
rulename	name	pg_rewrite.rulename	Name of rule
definition	text		Rule definition (a reconstructed creation command)

pg_rules 视图不包含视图的 ON SELECT 规则；那些可以在 pg_views 中看到。

45.63. pg_seclabels

视图pg_seclabels提供有关安全标签的信息。它是 pg_seclabel目录的更易查询版本。

表 45.64. pg_seclabels 列

Name	Type	References	Description
objoid	oid	any OID column	The OID of the object this security label pertains to
classoid	oid	pg_class.oid	The OID of the system catalog this object appears in
objsubid	int4		For a security label on a table column, this is the column number (the objoid and classoid refer to the table itself). For all other object types, this column is zero.
objtype	text		The type of object to which this label applies, as text.
objnamespace	oid	pg_namespace.oid	The OID of the namespace for this object, if applicable; otherwise NULL.
objname	text		The name of the object to which this label applies, as text.
provider	text	pg_seclabel.provider	The label provider associated with this label.
label	text	pg_seclabel.label	The security label applied to this object.

45.64. pg_settings

视图pg_settings提供对服务器运行时参数的访问。
它实质上是SHOW和SET命令的替代接口。它还提供了一些关于每个参数的事实，
这些事实无法直接从SHOW中获取，例如最小值和最大值。

表 45.65. pg_settings 列

Name	Type	Description
name	text	Run-time configuration parameter name
setting	text	Current value of the parame ter
unit	text	Implicit unit of the parameter
category	text	Logical group of the parame ter
short_desc	text	A brief description of the parameter
extra_desc	text	Additional, more detailed, description of the parameter
context	text	Context required to set the parameter's value (see below)
vartype	text	Parameter type (bool, enum, integer, real, or string)
source	text	Source of the current parame ter value
min_val	text	Minimum allowed value of the parameter (null for non-numeric values)
max_val	text	Maximum allowed value of the parameter (null for non-numeric values)
enumvals	text[]	Allowed values of an enum parameter (null for non-enum values)
boot_val	text	Parameter value assumed at server startup if the parame ter is not otherwise set
reset_val	text	Value that RESET would reset the parameter to in the curren t session
sourcefile	text	Configuration file the curren t value was set in (null for values set from sources other than configuration files, or when examined by a non-superuser); helpful when using include directives in configuration files
sourceline	integer	Line number within the config uration file the current value was set at (null for values set from sources other than configuration files, or when examined by a non-superu ser)

有几种可能的context值。按照更改设置的难度递减的顺序，它们是：

internal

这些设置不能直接更改；它们反映了内部确定的值。
其中一些可能可通过使用不同的配置选项重新构建服务器，或通过更改提供给initdb的选项来进行调整。

postmaster

这些设置只能在服务器启动时应用，因此任何更改都需要重启服务器。
这些设置的值通常存储在 postgresql.conf 文件中，或者在服务器启动时通过命令行传递。当然，任何较低 context 类型的设置也都可以在服务器启动时设置。

sighup

可以在 postgresql.conf 中更改这些设置，而无需重启服务器。向 postmaster 发送 SIGHUP 信号，会使其重新读取 postgresql.conf 并应用这些更改。postmaster 还会将 SIGHUP 信号转发给其子进程，以便它们都采用新值。

backend

可以在 postgresql.conf 中更改这些设置，而无需重启服务器。也可以在连接请求报文中为特定会话设置这些值（例如，通过 libpq 的 PGOPTIONS 环境变量）；任何用户都可以为其会话进行此类更改。但是，在会话启动后，这些设置永远不会更改。如果在 postgresql.conf 中更改它们，请向 postmaster 发送 SIGHUP 信号，使其重新读取 postgresql.conf。新值只会影响随后启动的会话。

superuser

这些设置可以从postgresql.conf中设置，或者通过SET命令在会话中设置；
但只有超级用户可以通过SET更改它们。如果没有使用SET建立会话本地值，那么在postgresql.conf中的更改也会影响到现有会话。

user

这些设置可以从postgresql.conf中设置，或通过SET命令在会话中设置。
任何用户都可以更改其会话本地值。如果没有使用SET建立会话本地值，那么在postgresql.conf中的更改也会影响到现有会话。

有关更改这些参数的各种方法，详见第 18.1 节。

pg_settings视图不能插入或删除，但可以更新。对 pg_settings 的某一行执行 UPDATE，等同于对该命名参数执行SET命令。更改只影响当前会话所用的值。如果在之后被中止的事务中发出了 UPDATE，那么事务回滚时 UPDATE 命令的效果也会消失。一旦外围事务提交，这些效果会持续到会话结束，除非被另一个 UPDATE 或 SET 覆盖。

45.65. pg_shadow

视图pg_shadow存在是为了向后兼容性：它模拟了在PostgreSQL 8.1版本之前存在的目录。它显示了在pg_authid中标记为rolcanlogin的所有角色的属性。

这个名称源于该表不应向公众开放读取，因为它包含密码。pg_user 是 pg_shadow 的公开可读视图，并隐藏密码字段。

表 45.66. pg_shadow 列

Name	Type	References	Description
usename	name	pg_authid.rolname	User name

Name	Type	References	Description
usesysid	oid	pg_authid.oid	ID of this user
usecreatedb	bool		User can create databases
usesuper	bool		User is a superuser
usecatupd	bool		User can update system catalogs. (Even a superuser cannot do this unless this column is true.)
userepl	bool		用户可开启流复制并将系统设置或者取消备份模式。
passwd	text		Password (possibly encrypted); null if none. See pg_authid for details of how encrypted passwords are stored.
valuntil	abstime		Password expiry time (only used for passwo rd authentication)
useconfig	text[]		Session defaults for run-time configuratio n variables

45.66. pg_stats

视图pg_stats提供对存储在pg_statistic目录中信息的访问。
此视图仅允许访问用户具有读取权限的表对应的pg_statistic行，
因此可以安全地允许对此视图进行公共读取访问。

pg_stats也旨在以比底层目录更易读的格式呈现信息— 但其模式必须在为pg_statistic定义新的槽类型时进行扩展。

表 45.67. pg_stats 列

名称	类型	引用	描述
schemaname	name	pg_namespace. nspname	包含表的模式名称
tablename	name	pg_class.relname	表的名称
attname	name	pg_attribute. attname	被此行描述的列名
inherited	bool		如果为true，则此行包括来自子表的值，而不仅仅是指定表中的值
null_frac	real		列项中为空的比例
avg_width	integer		列的条目的平均字节宽度
n_distinct	real		如果大于零，表示列中可区分值的估计个数。如果小于零，是可区分值个数除以行数的负值（当ANALYZE认为可区分值的数量会随着表增长而增加时采用负值

名称	类型	引用	描述
			的形式，而如果认为列具有固定数量的可选值时采用正值的形式）。例如，-1表示一个唯一列，即其中可区分值的个数等于行数。
most_common_vals	anyarray		列中高频值的一个列表（如果没有任何一个值看起来比其他值更常用，此列为空）
most_common_freqs	real[]		高频值的频率列表，即每一个高频值的出现次数除以总行数（如果most_common_vals为空，则此列为空）
histogram_bounds	anyarray		将列值划分成大小接近的组的值列表。如果存在most_common_vals，其中的值会被直方图计算所忽略（如果列类型没有一个<操作符或者most_common_vals等于整个值集合，则此列为空）
correlation	real		物理行顺序和列值逻辑顺序之间的统计关联。其范围从-1到+1。当值接近-1或+1时，在列上的一个索引扫描被认为比值接近0时的代价更低，因为这种情况减少了对磁盘的随机访问（如果列数据类型不具有一个<操作符，则此列为空）
most_common_elems	anyarray		在列值中，最经常出现的非空元素列表（对标量类型为空）
most_common_elem_freqs	real[]		最常用元素值的频度列表，即含有至少一个给定值实例的行的分数。在每个元素的频度之后有二至三个附加值，它们是每个元素频度的最小和最大值，以及可选的空元素的频度（如果most_common_elems为空，则此列为空）
elem_count_histogram	real[]		在列值中可区分非空元素值计数的一个直方图，后面跟随可区分非空元素的平均数（对于标量类型为空）

数组字段中的最大项数可以逐列地用 ALTER TABLE SET STATISTICS 命令设置，或者全局地通过设置default_statistics_target运行时参数来设置。

45.67. pg_tables

视图pg_tables提供了对数据库中每个表的有用信息的访问。

表 45.68. pg_tables 列

Name	Type	References	Description
schemaname	name	pg_namespace.nspname	Name of schema containing table
tablename	name	pg_class.relname	Name of table
tableowner	name	pg_authid.rolname	Name of table's owner
tablespace	name	pg_tablespace.spcname	Name of tablespace containing table (null if default for databa se)
hasindexes	boolean	pg_class.relhasindex	True if table has (or recently had) any indexes
hasrules	boolean	pg_class.relhasrules	True if table has (or once had) rules
hastriggers	boolean	pg_class.relhastriggers	True if table has (or once had) triggers

45.68. pg_timezone_abbrevs

视图pg_timezone_abbrevs提供当前被日期时间输入例程识别的时区缩写列表。
当修改timezone_abbreviations运行时参数时，该视图的内容会发生变化。

表 45.69. pg_timezone_abbrevs 列

Name	Type	Description
abbrev	text	Time zone abbreviation
utc_offset	interval	Offset from UTC (positive means east of Greenwich)
is_dst	boolean	True if this is a daylight-savings abbreviation

虽然大多数时区缩写代表与UTC的固定偏移量，但也有一些在历史上变化过（有关更多信息，请参见第 B.3 节）。在这种情况下，此视图呈现它们当前的含义。

45.69. pg_timezone_names

视图pg_timezone_names提供了一个由SET TIMEZONE识别的时区名称列表，以及它们的相关缩写、UTC偏移和夏令时状态。（从技术上讲，PostgreSQL使用UT1而不是UTC，因为不处理闰秒。）与在pg_timezone_abbrevs中显示的缩写不同，
这些名称中的许多暗示了一组夏令时转换日期规则。因此，
相关信息在本地夏令时边界上发生变化。显示的信息是基于CURRENT_TIMESTAMP
的当前值计算的。

表 45.70. pg_timezone_names 列

Name	Type	Description
name	text	Time zone name
abbrev	text	Time zone abbreviation
utc_offset	interval	Offset from UTC (positive means east of Greenwich)

Name	Type	Description
is_dst	boolean	True if currently observing daylight savings

45.70. pg_user

视图pg_user提供数据库用户的信息。它本质上是 pg_shadow 的公开可读视图，并隐藏密码字段。

表 45.71. pg_user 列

Name	Type	Description
usename	name	User name
usesysid	oid	User ID (arbitrary number used to reference this user)
usecreatedb	bool	User can create databases
usesuper	bool	User is a superuser
usecatupd	bool	User can update system catalogs. (Even a superuser cannot do this unless this column is true.)
passwd	text	Not the password (always reads as ********)
valuntil	abstime	Password expiry time (only used for password authen tication)
useconfig	text[]	Session defaults for run-time configuration variables
userepl	bool	用户可开启流复制并将系统设置或者取消备份模式。

45.71. pg_user_mappings

视图pg_user_mappings提供用户映射的信息。它本质上是 pg_user_mapping 的公开可读视图，但如果用户没有使用相应选项的权限，则会省略选项字段。

表 45.72. pg_user_mappings 列

Name	Type	References	Description
umid	oid	pg_user_mapping.oid	用户映射的OID
srvid	oid	pg_foreign_server.oid	包含该映射的外部服务器的OID
srvname	name	pg_foreign_server.srvname	外部服务器名
umuser	oid	pg_authid.oid	将要被映射的本地角色的OID，如果用户映射是公共的则为零
usename	name		将被映射的本地用户名
umoptions	text[]		用户映射特定的选项，以 “keyword=value” 字符串形式

为了保护作为用户映射选项存储的密码信息，umoptions列会显示为空值，除非符合以下条件之一：

- 当前用户是被映射的用户，并拥有服务器或在其上拥有USAGE权限
- 当前用户是服务器所有者，映射是为PUBLIC而进行的
- 当前用户是超级用户

45.72. pg_views

视图pg_views提供了对数据库中每个视图的有用信息的访问。

表 45.73. pg_views 列

Name	Type	References	Description
schemaname	name	pg_namespace.nspname	Name of schema containing view
viewname	name	pg_class.relname	Name of view
viewowner	name	pg_authid.rolname	Name of view's owner
definition	text		View definition (a reconstructed SELECT query)

第 46 章 Frontend/Backend Protocol

PostgreSQL uses a message-based protocol for communication between frontends and backends (clients and servers). The protocol is supported over TCP/IP and also over Unix-domain sockets. Port number 5432 has been registered with IANA as the customary TCP port number for servers supporting this protocol, but in practice any non-privileged port number can be used.

This document describes version 3.0 of the protocol, implemented in PostgreSQL 7.4 and later. For descriptions of the earlier protocol versions, see previous releases of the PostgreSQL documentation. A single server can support multiple protocol versions. The initial startup-request message tells the server which protocol version the client is attempting to use, and then the server follows that protocol if it is able.

In order to serve multiple clients efficiently, the server launches a new "backend" process for each client. In the current implementation, a new child process is created immediately after an incoming connection is detected. This is transparent to the protocol, however. For purposes of the protocol, the terms "backend" and "server" are interchangeable; likewise "frontend" and "client" are interchangeable.

46.1. Overview

The protocol has separate phases for startup and normal operation. In the startup phase, the frontend opens a connection to the server and authenticates itself to the satisfaction of the server. (This might involve a single message, or multiple messages depending on the authentication method being used.) If all goes well, the server then sends status information to the frontend, and finally enters normal operation. Except for the initial startup-request message, this part of the protocol is driven by the server.

During normal operation, the frontend sends queries and other commands to the backend, and the backend sends back query results and other responses. There are a few cases (such as `NOTIFY`) wherein the backend will send unsolicited messages, but for the most part this portion of a session is driven by frontend requests.

Termination of the session is normally by frontend choice, but can be forced by the backend in certain cases. In any case, when the backend closes the connection, it will roll back any open (incomplete) transaction before exiting.

Within normal operation, SQL commands can be executed through either of two sub-protocols. In the "simple query" protocol, the frontend just sends a textual query string, which is parsed and immediately executed by the backend. In the "extended query" protocol, processing of queries is separated into multiple steps: parsing, binding of parameter values, and execution. This offers flexibility and performance benefits, at the cost of extra complexity.

Normal operation has additional sub-protocols for special operations such as `COPY`.

46.1.1. Messaging Overview

All communication is through a stream of messages. The first byte of a message identifies the message type, and the next four bytes specify the length of the rest of the message (this length count includes itself, but not the message-type byte). The remaining contents of the message are determined by the message type. For historical reasons, the very first message sent by the client (the startup message) has no initial message-type byte.

To avoid losing synchronization with the message stream, both servers and clients typically read an entire message into a buffer (using the byte count) before attempting

to process its contents. This allows easy recovery if an error is detected while processing the contents. In extreme situations (such as not having enough memory to buffer the message), the receiver can use the byte count to determine how much input to skip before it resumes reading messages.

Conversely, both servers and clients must take care never to send an incomplete message. This is commonly done by marshaling the entire message in a buffer before beginning to send it. If a communications failure occurs partway through sending or receiving a message, the only sensible response is to abandon the connection, since there is little hope of recovering message-boundary synchronization.

46.1.2. Extended Query Overview

In the extended-query protocol, execution of SQL commands is divided into multiple steps. The state retained between steps is represented by two types of objects: prepared statements and portals. A prepared statement represents the result of parsing and semantic analysis of a textual query string. A prepared statement is not in itself ready to execute, because it might lack specific values for parameters. A portal represents a ready-to-execute or already-partially-executed statement, with any missing parameter values filled in. (For `SELECT` statements, a portal is equivalent to an open cursor, but we choose to use a different term since cursors don't handle non-`SELECT` statements.)

The overall execution cycle consists of a parse step, which creates a prepared statement from a textual query string; a bind step, which creates a portal given a prepared statement and values for any needed parameters; and an execute step that runs a portal's query. In the case of a query that returns rows (`SELECT`, `SHOW`, etc), the execute step can be told to fetch only a limited number of rows, so that multiple execute steps might be needed to complete the operation.

The backend can keep track of multiple prepared statements and portals (but note that these exist only within a session, and are never shared across sessions). Existing prepared statements and portals are referenced by names assigned when they were created. In addition, an “unnamed” prepared statement and portal exist. Although these behave largely the same as named objects, operations on them are optimized for the case of executing a query only once and then discarding it, whereas operations on named objects are optimized on the expectation of multiple uses.

46.1.3. Formats and Format Codes

Data of a particular data type might be transmitted in any of several different formats. As of PostgreSQL 7.4 the only supported formats are “text” and “binary”, but the protocol makes provision for future extensions. The desired format for any value is specified by a format code. Clients can specify a format code for each transmitted parameter value and for each column of a query result. Text has format code zero, binary has format code one, and all other format codes are reserved for future definition.

The text representation of values is whatever strings are produced and accepted by the input/output conversion functions for the particular data type. In the transmitted representation, there is no trailing null character; the frontend must add one to received values if it wants to process them as C strings. (The text format does not allow embedded nulls, by the way.)

Binary representations for integers use network byte order (most significant byte first). For other data types consult the documentation or source code to learn about the binary representation. Keep in mind that binary representations for complex data types might change across server versions; the text format is usually the more portable choice.

46.2. Message Flow

This section describes the message flow and the semantics of each message type. (Details of the exact representation of each message appear in 第 46.5 节.) There are several different sub-protocols depending on the state of the connection: start-up, query, function call, `COPY`, and termination. There are also special provisions for asynchronous operations (including notification responses and command cancellation), which can occur at any time after the start-up phase.

46.2.1. Start-up

To begin a session, a frontend opens a connection to the server and sends a startup message. This message includes the names of the user and of the database the user wants to connect to; it also identifies the particular protocol version to be used. (Optionally, the startup message can include additional settings for run-time parameters.) The server then uses this information and the contents of its configuration files (such as `pg_hba.conf`) to determine whether the connection is provisionally acceptable, and what additional authentication is required (if any).

The server then sends an appropriate authentication request message, to which the frontend must reply with an appropriate authentication response message (such as a password). For all authentication methods except GSSAPI and SSPI, there is at most one request and one response. In some methods, no response at all is needed from the frontend, and so no authentication request occurs. For GSSAPI and SSPI, multiple exchanges of packets may be needed to complete the authentication.

The authentication cycle ends with the server either rejecting the connection attempt (ErrorResponse), or sending AuthenticationOk.

The possible messages from the server in this phase are:

ErrorResponse

The connection attempt has been rejected. The server then immediately closes the connection.

AuthenticationOk

The authentication exchange is successfully completed.

AuthenticationKerberosV5

The frontend must now take part in a Kerberos V5 authentication dialog (not described here, part of the Kerberos specification) with the server. If this is successful, the server responds with an AuthenticationOk, otherwise it responds with an ErrorResponse.

AuthenticationCleartextPassword

The frontend must now send a PasswordMessage containing the password in clear-text form. If this is the correct password, the server responds with an AuthenticationOk, otherwise it responds with an ErrorResponse.

AuthenticationMD5Password

The frontend must now send a PasswordMessage containing the password (with username) encrypted via MD5, then encrypted again using the 4-byte random salt specified in the AuthenticationMD5Password message. If this is the correct password, the server responds with an AuthenticationOk, otherwise it responds with an ErrorResponse. The actual PasswordMessage can be computed in SQL as

`concat('md5', md5(concat(md5(concat(password, username)), random-salt)))`. (Keep in mind the `md5()` function returns its result as a hex string.)

AuthenticationSCMCredential

This response is only possible for local Unix-domain connections on platforms that support SCM credential messages. The frontend must issue an SCM credential message and then send a single data byte. (The contents of the data byte are uninteresting; it's only used to ensure that the server waits long enough to receive the credential message.) If the credential is acceptable, the server responds with an AuthenticationOk, otherwise it responds with an ErrorResponse. (This message type is only issued by pre-9.1 servers. It may eventually be removed from the protocol specification.)

AuthenticationGSS

The frontend must now initiate a GSSAPI negotiation. The frontend will send a PasswordMessage with the first part of the GSSAPI data stream in response to this. If further messages are needed, the server will respond with AuthenticationGSSContinue.

AuthenticationSSPI

The frontend must now initiate a SSPI negotiation. The frontend will send a PasswordMessage with the first part of the SSPI data stream in response to this. If further messages are needed, the server will respond with AuthenticationGSSContinue.

AuthenticationGSSContinue

This message contains the response data from the previous step of GSSAPI or SSPI negotiation (AuthenticationGSS, AuthenticationSSPI or a previous AuthenticationGSSContinue). If the GSSAPI or SSPI data in this message indicates more data is needed to complete the authentication, the frontend must send that data as another PasswordMessage. If GSSAPI or SSPI authentication is completed by this message, the server will next send AuthenticationOk to indicate successful authentication or ErrorResponse to indicate failure.

If the frontend does not support the authentication method requested by the server, then it should immediately close the connection.

After having received AuthenticationOk, the frontend must wait for further messages from the server. In this phase a backend process is being started, and the frontend is just an interested bystander. It is still possible for the startup attempt to fail (ErrorResponse), but in the normal case the backend will send some ParameterStatus messages, BackendKeyData, and finally ReadyForQuery.

During this phase the backend will attempt to apply any additional run-time parameter settings that were given in the startup message. If successful, these values become session defaults. An error causes ErrorResponse and exit.

The possible messages from the backend in this phase are:

BackendKeyData

This message provides secret-key data that the frontend must save if it wants to be able to issue cancel requests later. The frontend should not respond to this message, but should continue listening for a ReadyForQuery message.

ParameterStatus

This message informs the frontend about the current (initial) setting of backend parameters, such as client_encoding or DateStyle. The frontend can ignore this

message, or record the settings for its future use; see 第 46.2.6 节 for more details. The frontend should not respond to this message, but should continue listening for a ReadyForQuery message.

ReadyForQuery

Start-up is completed. The frontend can now issue commands.

ErrorResponse

Start-up failed. The connection is closed after sending this message.

NoticeResponse

A warning message has been issued. The frontend should display the message but continue listening for ReadyForQuery or ErrorResponse.

The ReadyForQuery message is the same one that the backend will issue after each command cycle. Depending on the coding needs of the frontend, it is reasonable to consider ReadyForQuery as starting a command cycle, or to consider ReadyForQuery as ending the start-up phase and each subsequent command cycle.

46.2.2. Simple Query

A simple query cycle is initiated by the frontend sending a Query message to the backend. The message includes an SQL command (or commands) expressed as a text string. The backend then sends one or more response messages depending on the contents of the query command string, and finally a ReadyForQuery response message. ReadyForQuery informs the frontend that it can safely send a new command. (It is not actually necessary for the frontend to wait for ReadyForQuery before issuing another command, but the frontend must then take responsibility for figuring out what happens if the earlier command fails and already-issued later commands succeed.)

The possible response messages from the backend are:

CommandComplete

An SQL command completed normally.

CopyInResponse

The backend is ready to copy data from the frontend to a table; see 第 46.2.5 节.

CopyOutResponse

The backend is ready to copy data from a table to the frontend; see 第 46.2.5 节.

RowDescription

Indicates that rows are about to be returned in response to a `SELECT`, `FETCH`, etc query. The contents of this message describe the column layout of the rows. This will be followed by a DataRow message for each row being returned to the frontend.

DataRow

One of the set of rows returned by a `SELECT`, `FETCH`, etc query.

EmptyQueryResponse

An empty query string was recognized.

ErrorResponse

An error has occurred.

ReadyForQuery

Processing of the query string is complete. A separate message is sent to indicate this because the query string might contain multiple SQL commands. (CommandComplete marks the end of processing one SQL command, not the whole string.) ReadyForQuery will always be sent, whether processing terminates successfully or with an error.

NoticeResponse

A warning message has been issued in relation to the query. Notices are in addition to other responses, i.e., the backend will continue processing the command.

The response to a `SELECT` query (or other queries that return row sets, such as `EXPLAIN` or `SHOW`) normally consists of RowDescription, zero or more DataRow messages, and then CommandComplete. `COPY` to or from the frontend invokes special protocol as described in 第 46.2.5 节. All other query types normally produce only a CommandComplete message.

Since a query string could contain several queries (separated by semicolons), there might be several such response sequences before the backend finishes processing the query string. ReadyForQuery is issued when the entire string has been processed and the backend is ready to accept a new query string.

If a completely empty (no contents other than whitespace) query string is received, the response is EmptyQueryResponse followed by ReadyForQuery.

In the event of an error, ErrorResponse is issued followed by ReadyForQuery. All further processing of the query string is aborted by ErrorResponse (even if more queries remained in it). Note that this might occur partway through the sequence of messages generated by an individual query.

In simple Query mode, the format of retrieved values is always text, except when the given command is a `FETCH` from a cursor declared with the `BINARY` option. In that case, the retrieved values are in binary format. The format codes given in the RowDescription message tell which format is being used.

A frontend must be prepared to accept ErrorResponse and NoticeResponse messages whenever it is expecting any other type of message. See also 第 46.2.6 节 concerning messages that the backend might generate due to outside events.

Recommended practice is to code frontends in a state-machine style that will accept any message type at any time that it could make sense, rather than wiring in assumptions about the exact sequence of messages.

46.2.3. Extended Query

The extended query protocol breaks down the above-described simple query protocol into multiple steps. The results of preparatory steps can be re-used multiple times for improved efficiency. Furthermore, additional features are available, such as the possibility of supplying data values as separate parameters instead of having to insert them directly into a query string.

In the extended protocol, the frontend first sends a Parse message, which contains a textual query string, optionally some information about data types of parameter

placeholders, and the name of a destination prepared-statement object (an empty string selects the unnamed prepared statement). The response is either ParseComplete or ErrorResponse. Parameter data types can be specified by OID; if not given, the parser attempts to infer the data types in the same way as it would do for untyped literal string constants.

> **注意**
>
> A parameter data type can be left unspecified by setting it to zero, or by making the array of parameter type OIDs shorter than the number of parameter symbols (`$n`) used in the query string. Another special case is that a parameter's type can be specified as `void` (that is, the OID of the `void` pseudotype). This is meant to allow parameter symbols to be used for function parameters that are actually OUT parameters. Ordinarily there is no context in which a `void` parameter could be used, but if such a parameter symbol appears in a function's parameter list, it is effectively ignored. For example, a function call such as `foo($1,$2,$3,$4)` could match a function with two IN and two OUT arguments, if `$3` and `$4` are specified as having type `void`.

> **注意**
>
> The query string contained in a Parse message cannot include more than one SQL statement; else a syntax error is reported. This restriction does not exist in the simple-query protocol, but it does exist in the extended protocol, because allowing prepared statements or portals to contain multiple commands would complicate the protocol unduly.

If successfully created, a named prepared-statement object lasts till the end of the current session, unless explicitly destroyed. An unnamed prepared statement lasts only until the next Parse statement specifying the unnamed statement as destination is issued. (Note that a simple Query message also destroys the unnamed statement.) Named prepared statements must be explicitly closed before they can be redefined by another Parse message, but this is not required for the unnamed statement. Named prepared statements can also be created and accessed at the SQL command level, using `PREPARE` and `EXECUTE`.

Once a prepared statement exists, it can be readied for execution using a Bind message. The Bind message gives the name of the source prepared statement (empty string denotes the unnamed prepared statement), the name of the destination portal (empty string denotes the unnamed portal), and the values to use for any parameter placeholders present in the prepared statement. The supplied parameter set must match those needed by the prepared statement. (If you declared any `void` parameters in the Parse message, pass NULL values for them in the Bind message.) Bind also specifies the format to use for any data returned by the query; the format can be specified overall, or per-column. The response is either BindComplete or ErrorResponse.

> **注意**
>
> The choice between text and binary output is determined by the format codes given in Bind, regardless of the SQL command involved. The `BINARY` attribute in cursor declarations is irrelevant when using extended query protocol.

Query planning typically occurs when the Bind message is processed. If the prepared statement has no parameters, or is executed repeatedly, the server might save the created plan and re-use it during subsequent Bind messages for the same prepared statement. However, it will do so only if it finds that a generic plan can be created that is not much less efficient than a plan that depends on the specific parameter values supplied. This happens transparently so far as the protocol is concerned.

If successfully created, a named portal object lasts till the end of the current transaction, unless explicitly destroyed. An unnamed portal is destroyed at the end of the transaction, or as soon as the next Bind statement specifying the unnamed portal as destination is issued. (Note that a simple Query message also destroys the unnamed portal.) Named portals must be explicitly closed before they can be redefined by another Bind message, but this is not required for the unnamed portal. Named portals can also be created and accessed at the SQL command level, using `DECLARE CURSOR` and `FETCH`.

Once a portal exists, it can be executed using an Execute message. The Execute message specifies the portal name (empty string denotes the unnamed portal) and a maximum result-row count (zero meaning "fetch all rows"). The result-row count is only meaningful for portals containing commands that return row sets; in other cases the command is always executed to completion, and the row count is ignored. The possible responses to Execute are the same as those described above for queries issued via simple query protocol, except that Execute doesn't cause ReadyForQuery or RowDescription to be issued.

If Execute terminates before completing the execution of a portal (due to reaching a nonzero result-row count), it will send a PortalSuspended message; the appearance of this message tells the frontend that another Execute should be issued against the same portal to complete the operation. The CommandComplete message indicating completion of the source SQL command is not sent until the portal's execution is completed. Therefore, an Execute phase is always terminated by the appearance of exactly one of these messages: CommandComplete, EmptyQueryResponse (if the portal was created from an empty query string), ErrorResponse, or PortalSuspended.

At completion of each series of extended-query messages, the frontend should issue a Sync message. This parameterless message causes the backend to close the current transaction if it's not inside a `BEGIN/COMMIT` transaction block ("close" meaning to commit if no error, or roll back if error). Then a ReadyForQuery response is issued. The purpose of Sync is to provide a resynchronization point for error recovery. When an error is detected while processing any extended-query message, the backend issues ErrorResponse, then reads and discards messages until a Sync is reached, then issues ReadyForQuery and returns to normal message processing. (But note that no skipping occurs if an error is detected while processing Sync – this ensures that there is one and only one ReadyForQuery sent for each Sync.)

> **注意**
>
> Sync does not cause a transaction block opened with `BEGIN` to be closed. It is possible to detect this situation since the ReadyForQuery message includes transaction status information.

In addition to these fundamental, required operations, there are several optional operations that can be used with extended-query protocol.

The Describe message (portal variant) specifies the name of an existing portal (or an empty string for the unnamed portal). The response is a RowDescription message describing the rows that will be returned by executing the portal; or a NoData message

if the portal does not contain a query that will return rows; or ErrorResponse if there is no such portal.

The Describe message (statement variant) specifies the name of an existing prepared statement (or an empty string for the unnamed prepared statement). The response is a ParameterDescription message describing the parameters needed by the statement, followed by a RowDescription message describing the rows that will be returned when the statement is eventually executed (or a NoData message if the statement will not return rows). ErrorResponse is issued if there is no such prepared statement. Note that since Bind has not yet been issued, the formats to be used for returned columns are not yet known to the backend; the format code fields in the RowDescription message will be zeroes in this case.

> **提示**
>
> In most scenarios the frontend should issue one or the other variant of Describe before issuing Execute, to ensure that it knows how to interpret the results it will get back.

The Close message closes an existing prepared statement or portal and releases resources. It is not an error to issue Close against a nonexistent statement or portal name. The response is normally CloseComplete, but could be ErrorResponse if some difficulty is encountered while releasing resources. Note that closing a prepared statement implicitly closes any open portals that were constructed from that statement.

The Flush message does not cause any specific output to be generated, but forces the backend to deliver any data pending in its output buffers. A Flush must be sent after any extended-query command except Sync, if the frontend wishes to examine the results of that command before issuing more commands. Without Flush, messages returned by the backend will be combined into the minimum possible number of packets to minimize network overhead.

> **注意**
>
> The simple Query message is approximately equivalent to the series Parse, Bind, portal Describe, Execute, Close, Sync, using the unnamed prepared statement and portal objects and no parameters. One difference is that it will accept multiple SQL statements in the query string, automatically performing the bind/describe/execute sequence for each one in succession. Another difference is that it will not return ParseComplete, BindComplete, CloseComplete, or NoData messages.

46.2.4. Function Call

The Function Call sub-protocol allows the client to request a direct call of any function that exists in the database's `pg_proc` system catalog. The client must have execute permission for the function.

> **注意**
>
> The Function Call sub-protocol is a legacy feature that is probably best avoided in new code. Similar results can be accomplished by setting up a prepared

> statement that does `SELECT function($1, ...)`. The Function Call cycle can then be replaced with Bind/Execute.

A Function Call cycle is initiated by the frontend sending a FunctionCall message to the backend. The backend then sends one or more response messages depending on the results of the function call, and finally a ReadyForQuery response message. ReadyForQuery informs the frontend that it can safely send a new query or function call.

The possible response messages from the backend are:

ErrorResponse

An error has occurred.

FunctionCallResponse

The function call was completed and returned the result given in the message. (Note that the Function Call protocol can only handle a single scalar result, not a row type or set of results.)

ReadyForQuery

Processing of the function call is complete. ReadyForQuery will always be sent, whether processing terminates successfully or with an error.

NoticeResponse

A warning message has been issued in relation to the function call. Notices are in addition to other responses, i.e., the backend will continue processing the command.

46.2.5. COPY Operations

The `COPY` command allows high-speed bulk data transfer to or from the server. Copy-in and copy-out operations each switch the connection into a distinct sub-protocol, which lasts until the operation is completed.

Copy-in mode (data transfer to the server) is initiated when the backend executes a `COPY FROM STDIN` SQL statement. The backend sends a CopyInResponse message to the frontend. The frontend should then send zero or more CopyData messages, forming a stream of input data. (The message boundaries are not required to have anything to do with row boundaries, although that is often a reasonable choice.) The frontend can terminate the copy-in mode by sending either a CopyDone message (allowing successful termination) or a CopyFail message (which will cause the `COPY` SQL statement to fail with an error). The backend then reverts to the command-processing mode it was in before the `COPY` started, which will be either simple or extended query protocol. It will next send either CommandComplete (if successful) or ErrorResponse (if not).

In the event of a backend-detected error during copy-in mode (including receipt of a CopyFail message), the backend will issue an ErrorResponse message. If the `COPY` command was issued via an extended-query message, the backend will now discard frontend messages until a Sync message is received, then it will issue ReadyForQuery and return to normal processing. If the `COPY` command was issued in a simple Query message, the rest of that message is discarded and ReadyForQuery is issued. In either case, any subsequent CopyData, CopyDone, or CopyFail messages issued by the frontend will simply be dropped.

The backend will ignore Flush and Sync messages received during copy-in mode. Receipt of any other non-copy message type constitutes an error that will abort the copy-in state as described above. (The exception for Flush and Sync is for the convenience of client libraries that always send Flush or Sync after an Execute message, without checking whether the command to be executed is a `COPY FROM STDIN`.)

Copy-out mode (data transfer from the server) is initiated when the backend executes a `COPY TO STDOUT` SQL statement. The backend sends a CopyOutResponse message to the frontend, followed by zero or more CopyData messages (always one per row), followed by CopyDone. The backend then reverts to the command-processing mode it was in before the `COPY` started, and sends CommandComplete. The frontend cannot abort the transfer (except by closing the connection or issuing a Cancel request), but it can discard unwanted CopyData and CopyDone messages.

In the event of a backend-detected error during copy-out mode, the backend will issue an ErrorResponse message and revert to normal processing. The frontend should treat receipt of ErrorResponse as terminating the copy-out mode.

It is possible for NoticeResponse and ParameterStatus messages to be interspersed between CopyData messages; frontends must handle these cases, and should be prepared for other asynchronous message types as well (see 第 46.2.6 节). Otherwise, any message type other than CopyData or CopyDone may be treated as terminating copy-out mode.

There is another Copy-related mode called Copy-both, which allows high-speed bulk data transfer to and from the server. Copy-both mode is initiated when a backend in walsender mode executes a `START_REPLICATION` statement. The backend sends a CopyBothResponse message to the frontend. Both the backend and the frontend may then send CopyData messages until the connection is terminated. See 第 46.3 节.

The CopyInResponse, CopyOutResponse and CopyBothResponse messages include fields that inform the frontend of the number of columns per row and the format codes being used for each column. (As of the present implementation, all columns in a given `COPY` operation will use the same format, but the message design does not assume this.)

46.2.6. Asynchronous Operations

There are several cases in which the backend will send messages that are not specifically prompted by the frontend's command stream. Frontends must be prepared to deal with these messages at any time, even when not engaged in a query. At minimum, one should check for these cases before beginning to read a query response.

It is possible for NoticeResponse messages to be generated due to outside activity; for example, if the database administrator commands a “fast” database shutdown, the backend will send a NoticeResponse indicating this fact before closing the connection. Accordingly, frontends should always be prepared to accept and display NoticeResponse messages, even when the connection is nominally idle.

ParameterStatus messages will be generated whenever the active value changes for any of the parameters the backend believes the frontend should know about. Most commonly this occurs in response to a `SET` SQL command executed by the frontend, and this case is effectively synchronous – but it is also possible for parameter status changes to occur because the administrator changed a configuration file and then sent the SIGHUP signal to the server. Also, if a `SET` command is rolled back, an appropriate ParameterStatus message will be generated to report the current effective value.

At present there is a hard-wired set of parameters for which ParameterStatus will be generated: they are `server_version`, `server_encoding`, `client_encoding`, `application_name`, `is_superuser`, `session_authorization`, `DateStyle`, `IntervalStyle`, `TimeZone`, `integer_datetimes`, and `standard_conforming_strings`. (`server_encoding`, `TimeZone`, and `integer_datetimes` were not reported by releases before 8.0; `standard_conforming_strings` was not reported by releases before 8.1; `IntervalStyle` was not reported by releases before 8.4; `application_name` was not reported by releases before 9.0.) Note that `server_version`, `server_encoding` and `integer_datetimes` are pseudo-parameters that cannot change after startup. This set might change in the future, or even become configurable. Accordingly, a frontend should simply ignore ParameterStatus for parameters that it does not understand or care about.

If a frontend issues a `LISTEN` command, then the backend will send a NotificationResponse message (not to be confused with NoticeResponse!) whenever a `NOTIFY` command is executed for the same channel name.

> **注意**
>
> At present, NotificationResponse can only be sent outside a transaction, and thus it will not occur in the middle of a command-response series, though it might occur just before ReadyForQuery. It is unwise to design frontend logic that assumes that, however. Good practice is to be able to accept NotificationResponse at any point in the protocol.

46.2.7. Canceling Requests in Progress

During the processing of a query, the frontend might request cancellation of the query. The cancel request is not sent directly on the open connection to the backend for reasons of implementation efficiency: we don't want to have the backend constantly checking for new input from the frontend during query processing. Cancel requests should be relatively infrequent, so we make them slightly cumbersome in order to avoid a penalty in the normal case.

To issue a cancel request, the frontend opens a new connection to the server and sends a CancelRequest message, rather than the StartupMessage message that would ordinarily be sent across a new connection. The server will process this request and then close the connection. For security reasons, no direct reply is made to the cancel request message.

A CancelRequest message will be ignored unless it contains the same key data (PID and secret key) passed to the frontend during connection start-up. If the request matches the PID and secret key for a currently executing backend, the processing of the current query is aborted. (In the existing implementation, this is done by sending a special signal to the backend process that is processing the query.)

The cancellation signal might or might not have any effect – for example, if it arrives after the backend has finished processing the query, then it will have no effect. If the cancellation is effective, it results in the current command being terminated early with an error message.

The upshot of all this is that for reasons of both security and efficiency, the frontend has no direct way to tell whether a cancel request has succeeded. It must continue to wait for the backend to respond to the query. Issuing a cancel simply improves the odds that the current query will finish soon, and improves the odds that it will fail with an error message instead of succeeding.

Since the cancel request is sent across a new connection to the server and not across the regular frontend/backend communication link, it is possible for the cancel request to be issued by any process, not just the frontend whose query is to be canceled. This might provide additional flexibility when building multiple-process applications. It also introduces a security risk, in that unauthorized persons might try to cancel queries. The security risk is addressed by requiring a dynamically generated secret key to be supplied in cancel requests.

46.2.8. Termination

The normal, graceful termination procedure is that the frontend sends a Terminate message and immediately closes the connection. On receipt of this message, the backend closes the connection and terminates.

In rare cases (such as an administrator-commanded database shutdown) the backend might disconnect without any frontend request to do so. In such cases the backend will attempt to send an error or notice message giving the reason for the disconnection before it closes the connection.

Other termination scenarios arise from various failure cases, such as core dump at one end or the other, loss of the communications link, loss of message-boundary synchronization, etc. If either frontend or backend sees an unexpected closure of the connection, it should clean up and terminate. The frontend has the option of launching a new backend by recontacting the server if it doesn't want to terminate itself. Closing the connection is also advisable if an unrecognizable message type is received, since this probably indicates loss of message-boundary sync.

For either normal or abnormal termination, any open transaction is rolled back, not committed. One should note however that if a frontend disconnects while a non-`SELECT` query is being processed, the backend will probably finish the query before noticing the disconnection. If the query is outside any transaction block (`BEGIN` ... `COMMIT` sequence) then its results might be committed before the disconnection is recognized.

46.2.9. SSL Session Encryption

If PostgreSQL was built with SSL support, frontend/backend communications can be encrypted using SSL. This provides communication security in environments where attackers might be able to capture the session traffic. For more information on encrypting PostgreSQL sessions with SSL, see 第 17.9 节.

To initiate an SSL-encrypted connection, the frontend initially sends an SSLRequest message rather than a StartupMessage. The server then responds with a single byte containing `S` or `N`, indicating that it is willing or unwilling to perform SSL, respectively. The frontend might close the connection at this point if it is dissatisfied with the response. To continue after `S`, perform an SSL startup handshake (not described here, part of the SSL specification) with the server. If this is successful, continue with sending the usual StartupMessage. In this case the StartupMessage and all subsequent data will be SSL-encrypted. To continue after `N`, send the usual StartupMessage and proceed without encryption.

The frontend should also be prepared to handle an ErrorMessage response to SSLRequest from the server. This would only occur if the server predates the addition of SSL support to PostgreSQL. (Such servers are now very ancient, and likely do not exist in the wild anymore.) In this case the connection must be closed, but the frontend might choose to open a fresh connection and proceed without requesting SSL.

An initial SSLRequest can also be used in a connection that is being opened to send a CancelRequest message.

While the protocol itself does not provide a way for the server to force SSL encryption, the administrator can configure the server to reject unencrypted sessions as a byproduct of authentication checking.

46.3. Streaming Replication Protocol

To initiate streaming replication, the frontend sends the `replication` parameter in the startup message. This tells the backend to go into walsender mode, wherein a small set of replication commands can be issued instead of SQL statements. Only the simple query protocol can be used in walsender mode. The commands accepted in walsender mode are:

IDENTIFY_SYSTEM

Requests the server to identify itself. Server replies with a result set of a single row, containing three fields:

systemid

The unique system identifier identifying the cluster. This can be used to check that the base backup used to initialize the standby came from the same cluster.

timeline

Current TimelineID. Also useful to check that the standby is consistent with the master.

xlogpos

Current xlog flush location. Useful to get a known location in the transaction log where streaming can start.

START_REPLICATION *XXX*/*XXX*

Instructs server to start streaming WAL, starting at WAL position *XXX*/*XXX*. The server can reply with an error, e.g. if the requested section of WAL has already been recycled. On success, server responds with a CopyBothResponse message, and then starts to stream WAL to the frontend. WAL will continue to be streamed until the connection is broken; no further commands will be accepted.

WAL data is sent as a series of CopyData messages. (This allows other information to be intermixed; in particular the server can send an ErrorResponse message if it encounters a failure after beginning to stream.) The payload in each CopyData message follows this format:

XLogData (B)

Byte1('w')

Identifies the message as WAL data.

Byte8

The starting point of the WAL data in this message, given in XLogRecPtr format.

Byte8

The current end of WAL on the server, given in XLogRecPtr format.

Byte8

The server's system clock at the time of transmission, given in TimestampTz format.

Byte*n*

A section of the WAL data stream.

A single WAL record is never split across two CopyData messages. When a WAL record crosses a WAL page boundary, and is therefore already split using continuation records, it can be split at the page boundary. In other words, the first main WAL record and its continuation records can be sent in different CopyData messages.

Note that all fields within the WAL data and the above-described header will be in the sending server's native format. Endianness, and the format for the timestamp, are unpredictable unless the receiver has verified that the sender's system identifier matches its own `pg_control` contents.

If the WAL sender process is terminated normally (during postmaster shutdown), it will send a CommandComplete message before exiting. This might not happen during an abnormal shutdown, of course.

The receiving process can send replies back to the sender at any time, using one of the following message formats (also in the payload of a CopyData message):

Primary keepalive message (B)

Byte1('k')

Identifies the message as a sender keepalive.

Byte8

The current end of WAL on the server, given in XLogRecPtr format.

Byte8

The server's system clock at the time of transmission, given in TimestampTz format.

Standby status update (F)

Byte1('r')

Identifies the message as a receiver status update.

Byte8

The location of the last WAL byte + 1 received and written to disk in the standby, in XLogRecPtr format.

Byte8

The location of the last WAL byte + 1 flushed to disk in the standby, in XLogRecPtr format.

Byte8

The location of the last WAL byte + 1 applied in the standby, in XLogRecPtr format.

Byte8

The server's system clock at the time of transmission, given in TimestampTz format.

Hot Standby feedback message (F)

Byte1('h')

Identifies the message as a Hot Standby feedback message.

Byte8

The server's system clock at the time of transmission, given in TimestampTz format.

Byte4

The standby's current xmin. This may be 0, if the standby is sending notification that Hot Standby feedback will no longer be sent on this connection. Later non-zero messages may reinitiate the feedback mechanism.

Byte4

The standby's current epoch.

BASE_BACKUP [`LABEL 'label'`] [`PROGRESS`] [`FAST`] [`WAL`] [`NOWAIT`]

Instructs the server to start streaming a base backup. The system will automatically be put in backup mode before the backup is started, and taken out of it when the backup is complete. The following options are accepted:

`LABEL 'label'`

Sets the label of the backup. If none is specified, a backup label of `base backup` will be used. The quoting rules for the label are the same as a standard SQL string with standard_conforming_strings turned on.

`PROGRESS`

Request information required to generate a progress report. This will send back an approximate size in the header of each tablespace, which can be used to calculate how far along the stream is done. This is calculated by enumerating all the file sizes once before the transfer is even started, and may as such have a negative impact on the performance - in particular it may take longer before the first data is streamed. Since the database files can change during the backup, the size is only approximate and may both grow and shrink between the time of approximation and the sending of the actual files.

`FAST`

Request a fast checkpoint.

`WAL`

Include the necessary WAL segments in the backup. This will include all the files between start and stop backup in the `pg_xlog` directory of the base directory tar file.

`NOWAIT`

By default, the backup will wait until the last required xlog segment has been archived, or emit a warning if log archiving is not enabled. Specifying `NOWAIT` disables both the waiting and the warning, leaving the client responsible for ensuring the required log is available.

When the backup is started, the server will first send two ordinary result sets, followed by one or more CopyResponse results.

The first ordinary result set contains the starting position of the backup, given in XLogRecPtr format as a single column in a single row.

The second ordinary result set has one row for each tablespace. The fields in this row are:

spcoid

The oid of the tablespace, or `NULL` if it's the base directory.

spclocation

The full path of the tablespace directory, or `NULL` if it's the base directory.

size

The approximate size of the tablespace, if progress report has been requested; otherwise it's `NULL`.

After the second regular result set, one or more CopyResponse results will be sent, one for PGDATA and one for each additional tablespace other than `pg_default` and `pg_global`. The data in the CopyResponse results will be a tar format (following the "ustar interchange format" specified in the POSIX 1003.1-2008 standard) dump of the tablespace contents, except that the two trailing blocks of zeroes specified in the standard are omitted. After the tar data is complete, a final ordinary result set will be sent.

The tar archive for the data directory and each tablespace will contain all files in the directories, regardless of whether they are PostgreSQL files or other files added to the same directory. The only excluded files are:

- `postmaster.pid`
- `postmaster.opts`
- `pg_xlog`, including subdirectories. If the backup is run with WAL files included, a synthesized version of `pg_xlog` will be included, but it will only contain the files necessary for the backup to work, not the rest of the contents.

Owner, group and file mode are set if the underlying file system on the server supports it.

Once all tablespaces have been sent, a final regular result set will be sent. This result set contains the end position of the backup, given in XLogRecPtr format as a single column in a single row.

46.4. Message Data Types

This section describes the base data types used in messages.

Int*n*(*i*)

An *n*-bit integer in network byte order (most significant byte first). If *i* is specified it is the exact value that will appear, otherwise the value is variable. Eg. Int16, Int32(42).

Int$n$[k]

An array of k n-bit integers, each in network byte order. The array length k is always determined by an earlier field in the message. Eg. Int16[M].

String(s)

A null-terminated string (C-style string). There is no specific length limitation on strings. If s is specified it is the exact value that will appear, otherwise the value is variable. Eg. String, String("user").

> **注意**
>
> There is no predefined limit on the length of a string that can be returned by the backend. Good coding strategy for a frontend is to use an expandable buffer so that anything that fits in memory can be accepted. If that's not feasible, read the full string and discard trailing characters that don't fit into your fixed-size buffer.

Byte$n$(c)

Exactly n bytes. If the field width n is not a constant, it is always determinable from an earlier field in the message. If c is specified it is the exact value. Eg. Byte2, Byte1('\n').

46.5. Message Formats

This section describes the detailed format of each message. Each is marked to indicate that it can be sent by a frontend (F), a backend (B), or both (F & B). Notice that although each message includes a byte count at the beginning, the message format is defined so that the message end can be found without reference to the byte count. This aids validity checking. (The CopyData message is an exception, because it forms part of a data stream; the contents of any individual CopyData message cannot be interpretable on their own.)

AuthenticationOk (B)

Byte1('R')

Identifies the message as an authentication request.

Int32(8)

Length of message contents in bytes, including self.

Int32(0)

Specifies that the authentication was successful.

AuthenticationKerberosV5 (B)

Byte1('R')

Identifies the message as an authentication request.

Int32(8)

Length of message contents in bytes, including self.

Int32(2)

Specifies that Kerberos V5 authentication is required.

AuthenticationCleartextPassword (B)

Byte1('R')

Identifies the message as an authentication request.

Int32(8)

Length of message contents in bytes, including self.

Int32(3)

Specifies that a clear-text password is required.

AuthenticationMD5Password (B)

Byte1('R')

Identifies the message as an authentication request.

Int32(12)

Length of message contents in bytes, including self.

Int32(5)

Specifies that an MD5-encrypted password is required.

Byte4

The salt to use when encrypting the password.

AuthenticationSCMCredential (B)

Byte1('R')

Identifies the message as an authentication request.

Int32(8)

Length of message contents in bytes, including self.

Int32(6)

Specifies that an SCM credentials message is required.

AuthenticationGSS (B)

Byte1('R')

Identifies the message as an authentication request.

Int32(8)

Length of message contents in bytes, including self.

Int32(7)

Specifies that GSSAPI authentication is required.

AuthenticationSSPI (B)

Byte1('R')

Identifies the message as an authentication request.

Int32(8)

Length of message contents in bytes, including self.

Int32(9)

Specifies that SSPI authentication is required.

AuthenticationGSSContinue (B)

Byte1('R')

Identifies the message as an authentication request.

Int32

Length of message contents in bytes, including self.

Int32(8)

Specifies that this message contains GSSAPI or SSPI data.

Byte$n$

GSSAPI or SSPI authentication data.

BackendKeyData (B)

Byte1('K')

Identifies the message as cancellation key data. The frontend must save these values if it wishes to be able to issue CancelRequest messages later.

Int32(12)

Length of message contents in bytes, including self.

Int32

The process ID of this backend.

Int32

The secret key of this backend.

Bind (F)

Byte1('B')

Identifies the message as a Bind command.

Int32

Length of message contents in bytes, including self.

String

The name of the destination portal (an empty string selects the unnamed portal).

String

The name of the source prepared statement (an empty string selects the unnamed prepared statement).

Int16

The number of parameter format codes that follow (denoted C below). This can be zero to indicate that there are no parameters or that the parameters all use the default format (text); or one, in which case the specified format code is applied to all parameters; or it can equal the actual number of parameters.

Int16[C]

The parameter format codes. Each must presently be zero (text) or one (binary).

Int16

The number of parameter values that follow (possibly zero). This must match the number of parameters needed by the query.

Next, the following pair of fields appear for each parameter:

Int32

The length of the parameter value, in bytes (this count does not include itself). Can be zero. As a special case, -1 indicates a NULL parameter value. No value bytes follow in the NULL case.

Byte$n$

The value of the parameter, in the format indicated by the associated format code. n is the above length.

After the last parameter, the following fields appear:

Int16

The number of result-column format codes that follow (denoted R below). This can be zero to indicate that there are no result columns or that the result columns should all use the default format (text); or one, in which case the specified format code is applied to all result columns (if any); or it can equal the actual number of result columns of the query.

Int16[R]

The result-column format codes. Each must presently be zero (text) or one (binary).

BindComplete (B)

Byte1('2')

Identifies the message as a Bind-complete indicator.

Int32(4)

Length of message contents in bytes, including self.

CancelRequest (F)

Int32(16)

Length of message contents in bytes, including self.

Int32(80877102)

The cancel request code. The value is chosen to contain `1234` in the most significant 16 bits, and `5678` in the least significant 16 bits. (To avoid confusion, this code must not be the same as any protocol version number.)

Int32

The process ID of the target backend.

Int32

The secret key for the target backend.

Close (F)

Byte1('C')

Identifies the message as a Close command.

Int32

Length of message contents in bytes, including self.

Byte1

'`S`' to close a prepared statement; or '`P`' to close a portal.

String

The name of the prepared statement or portal to close (an empty string selects the unnamed prepared statement or portal).

CloseComplete (B)

Byte1('3')

Identifies the message as a Close-complete indicator.

Int32(4)

Length of message contents in bytes, including self.

CommandComplete (B)

Byte1('C')

Identifies the message as a command-completed response.

Int32

Length of message contents in bytes, including self.

String

The command tag. This is usually a single word that identifies which SQL command was completed.

For an `INSERT` command, the tag is `INSERT` *`oid`* *`rows`*, where *`rows`* is the number of rows inserted. *`oid`* is the object ID of the inserted row if *`rows`* is 1 and the target table has OIDs; otherwise *`oid`* is 0.

For a `DELETE` command, the tag is `DELETE` *`rows`* where *`rows`* is the number of rows deleted.

For an `UPDATE` command, the tag is `UPDATE rows` where *`rows`* is the number of rows updated.

For a `SELECT` or `CREATE TABLE AS` command, the tag is `SELECT rows` where *`rows`* is the number of rows retrieved.

For a `MOVE` command, the tag is `MOVE rows` where *`rows`* is the number of rows the cursor's position has been changed by.

For a `FETCH` command, the tag is `FETCH rows` where *`rows`* is the number of rows that have been retrieved from the cursor.

For a `COPY` command, the tag is `COPY rows` where *`rows`* is the number of rows copied. (Note: the row count appears only in PostgreSQL 8.2 and later.)

CopyData (F & B)

Byte1('d')

Identifies the message as `COPY` data.

Int32

Length of message contents in bytes, including self.

Byte*n*

Data that forms part of a `COPY` data stream. Messages sent from the backend will always correspond to single data rows, but messages sent by frontends might divide the data stream arbitrarily.

CopyDone (F & B)

Byte1('c')

Identifies the message as a `COPY`-complete indicator.

Int32(4)

Length of message contents in bytes, including self.

CopyFail (F)

Byte1('f')

Identifies the message as a `COPY`-failure indicator.

Int32

Length of message contents in bytes, including self.

String

An error message to report as the cause of failure.

CopyInResponse (B)

Byte1('G')

Identifies the message as a Start Copy In response. The frontend must now send copy-in data (if not prepared to do so, send a CopyFail message).

Int32

Length of message contents in bytes, including self.

Int8

0 indicates the overall `COPY` format is textual (rows separated by newlines, columns separated by separator characters, etc). 1 indicates the overall copy format is binary (similar to DataRow format). See COPY for more information.

Int16

The number of columns in the data to be copied (denoted *N* below).

Int16[*N*]

The format codes to be used for each column. Each must presently be zero (text) or one (binary). All must be zero if the overall copy format is textual.

CopyOutResponse (B)

Byte1('H')

Identifies the message as a Start Copy Out response. This message will be followed by copy-out data.

Int32

Length of message contents in bytes, including self.

Int8

0 indicates the overall `COPY` format is textual (rows separated by newlines, columns separated by separator characters, etc). 1 indicates the overall copy format is binary (similar to DataRow format). See COPY for more information.

Int16

The number of columns in the data to be copied (denoted *N* below).

Int16[*N*]

The format codes to be used for each column. Each must presently be zero (text) or one (binary). All must be zero if the overall copy format is textual.

CopyBothResponse (B)

Byte1('W')

Identifies the message as a Start Copy Both response. This message is used only for Streaming Replication.

Int32

Length of message contents in bytes, including self.

Int8

0 indicates the overall `COPY` format is textual (rows separated by newlines, columns separated by separator characters, etc). 1 indicates the overall copy format is binary (similar to DataRow format). See COPY for more information.

Int16

The number of columns in the data to be copied (denoted *N* below).

Int16[*N*]

The format codes to be used for each column. Each must presently be zero (text) or one (binary). All must be zero if the overall copy format is textual.

DataRow (B)

Byte1('D')

Identifies the message as a data row.

Int32

Length of message contents in bytes, including self.

Int16

The number of column values that follow (possibly zero).

Next, the following pair of fields appear for each column:

Int32

The length of the column value, in bytes (this count does not include itself). Can be zero. As a special case, -1 indicates a NULL column value. No value bytes follow in the NULL case.

Byte*n*

The value of the column, in the format indicated by the associated format code. *n* is the above length.

Describe (F)

Byte1('D')

Identifies the message as a Describe command.

Int32

Length of message contents in bytes, including self.

Byte1

'S' to describe a prepared statement; or 'P' to describe a portal.

String

The name of the prepared statement or portal to describe (an empty string selects the unnamed prepared statement or portal).

EmptyQueryResponse (B)

Byte1('I')

Identifies the message as a response to an empty query string. (This substitutes for CommandComplete.)

Int32(4)

Length of message contents in bytes, including self.

ErrorResponse (B)

Byte1('E')

Identifies the message as an error.

Int32

Length of message contents in bytes, including self.

The message body consists of one or more identified fields, followed by a zero byte as a terminator. Fields can appear in any order. For each field there is the following:

Byte1

A code identifying the field type; if zero, this is the message terminator and no string follows. The presently defined field types are listed in 第 46.6 节. Since more field types might be added in future, frontends should silently ignore fields of unrecognized type.

String

The field value.

Execute (F)

Byte1('E')

Identifies the message as an Execute command.

Int32

Length of message contents in bytes, including self.

String

The name of the portal to execute (an empty string selects the unnamed portal).

Int32

Maximum number of rows to return, if portal contains a query that returns rows (ignored otherwise). Zero denotes “no limit” .

Flush (F)

Byte1('H')

Identifies the message as a Flush command.

Int32(4)

Length of message contents in bytes, including self.

FunctionCall (F)

Byte1('F')

Identifies the message as a function call.

Int32

Length of message contents in bytes, including self.

Int32

Specifies the object ID of the function to call.

Int16

The number of argument format codes that follow (denoted *C* below). This can be zero to indicate that there are no arguments or that the arguments all use the default format (text); or one, in which case the specified format code is applied to all arguments; or it can equal the actual number of arguments.

Int16[*C*]

The argument format codes. Each must presently be zero (text) or one (binary).

Int16

Specifies the number of arguments being supplied to the function.

Next, the following pair of fields appear for each argument:

Int32

The length of the argument value, in bytes (this count does not include itself). Can be zero. As a special case, -1 indicates a NULL argument value. No value bytes follow in the NULL case.

Byte*n*

The value of the argument, in the format indicated by the associated format code. *n* is the above length.

After the last argument, the following field appears:

Int16

The format code for the function result. Must presently be zero (text) or one (binary).

FunctionCallResponse (B)

Byte1('V')

Identifies the message as a function call result.

Int32

Length of message contents in bytes, including self.

Int32

The length of the function result value, in bytes (this count does not include itself). Can be zero. As a special case, -1 indicates a NULL function result. No value bytes follow in the NULL case.

Byte*n*

The value of the function result, in the format indicated by the associated format code. *n* is the above length.

NoData (B)

Byte1('n')

Identifies the message as a no-data indicator.

Int32(4)

Length of message contents in bytes, including self.

NoticeResponse (B)

Byte1('N')

Identifies the message as a notice.

Int32

Length of message contents in bytes, including self.

The message body consists of one or more identified fields, followed by a zero byte as a terminator. Fields can appear in any order. For each field there is the following:

Byte1

A code identifying the field type; if zero, this is the message terminator and no string follows. The presently defined field types are listed in 第 46.6 节. Since more field types might be added in future, frontends should silently ignore fields of unrecognized type.

String

The field value.

NotificationResponse (B)

Byte1('A')

Identifies the message as a notification response.

Int32

Length of message contents in bytes, including self.

Int32

The process ID of the notifying backend process.

String

The name of the channel that the notify has been raised on.

String

The “payload” string passed from the notifying process.

ParameterDescription (B)

Byte1('t')

Identifies the message as a parameter description.

Int32

Length of message contents in bytes, including self.

Int16

The number of parameters used by the statement (can be zero).

Then, for each parameter, there is the following:

Int32

Specifies the object ID of the parameter data type.

ParameterStatus (B)

Byte1('S')

Identifies the message as a run-time parameter status report.

Int32

Length of message contents in bytes, including self.

String

The name of the run-time parameter being reported.

String

The current value of the parameter.

Parse (F)

Byte1('P')

Identifies the message as a Parse command.

Int32

Length of message contents in bytes, including self.

String

The name of the destination prepared statement (an empty string selects the unnamed prepared statement).

String

The query string to be parsed.

Int16

The number of parameter data types specified (can be zero). Note that this is not an indication of the number of parameters that might appear in the query string, only the number that the frontend wants to prespecify types for.

Then, for each parameter, there is the following:

Int32

Specifies the object ID of the parameter data type. Placing a zero here is equivalent to leaving the type unspecified.

ParseComplete (B)

Byte1('1')

Identifies the message as a Parse-complete indicator.

Int32(4)

Length of message contents in bytes, including self.

PasswordMessage (F)

Byte1('p')

Identifies the message as a password response. Note that this is also used for GSSAPI and SSPI response messages (which is really a design error, since the contained data is not a null-terminated string in that case, but can be arbitrary binary data).

Int32

Length of message contents in bytes, including self.

String

The password (encrypted, if requested).

PortalSuspended (B)

Byte1('s')

Identifies the message as a portal-suspended indicator. Note this only appears if an Execute message's row-count limit was reached.

Int32(4)

Length of message contents in bytes, including self.

Query (F)

Byte1('Q')

Identifies the message as a simple query.

Int32

Length of message contents in bytes, including self.

String

The query string itself.

ReadyForQuery (B)

Byte1('Z')

Identifies the message type. ReadyForQuery is sent whenever the backend is ready for a new query cycle.

Int32(5)

Length of message contents in bytes, including self.

Byte1

Current backend transaction status indicator. Possible values are '`I`' if idle (not in a transaction block); '`T`' if in a transaction block; or '`E`' if in a failed transaction block (queries will be rejected until block is ended).

RowDescription (B)

Byte1('T')

Identifies the message as a row description.

Int32

Length of message contents in bytes, including self.

Int16

Specifies the number of fields in a row (can be zero).

Then, for each field, there is the following:

String

The field name.

Int32

If the field can be identified as a column of a specific table, the object ID of the table; otherwise zero.

Int16

If the field can be identified as a column of a specific table, the attribute number of the column; otherwise zero.

Int32

The object ID of the field's data type.

Int16

The data type size (see `pg_type.typlen`). Note that negative values denote variable-width types.

Int32

The type modifier (see `pg_attribute.atttypmod`). The meaning of the modifier is type-specific.

Int16

The format code being used for the field. Currently will be zero (text) or one (binary). In a RowDescription returned from the statement variant of Describe, the format code is not yet known and will always be zero.

SSLRequest (F)

Int32(8)

Length of message contents in bytes, including self.

Int32(80877103)

The SSL request code. The value is chosen to contain `1234` in the most significant 16 bits, and `5679` in the least significant 16 bits. (To avoid confusion, this code must not be the same as any protocol version number.)

StartupMessage (F)

Int32

Length of message contents in bytes, including self.

Int32(196608)

The protocol version number. The most significant 16 bits are the major version number (3 for the protocol described here). The least significant 16 bits are the minor version number (0 for the protocol described here).

The protocol version number is followed by one or more pairs of parameter name and value strings. A zero byte is required as a terminator after the last name/value pair. Parameters can appear in any order. `user` is required, others are optional. Each parameter is specified as:

String

The parameter name. Currently recognized names are:

`user`

The database user name to connect as. Required; there is no default.

`database`

The database to connect to. Defaults to the user name.

`options`

Command-line arguments for the backend. (This is deprecated in favor of setting individual run-time parameters.)

In addition to the above, any run-time parameter that can be set at backend start time might be listed. Such settings will be applied during backend start (after parsing the command-line options if any). The values will act as session defaults.

String

The parameter value.

Sync (F)

Byte1('S')

Identifies the message as a Sync command.

Int32(4)

Length of message contents in bytes, including self.

Terminate (F)

Byte1('X')

Identifies the message as a termination.

Int32(4)

Length of message contents in bytes, including self.

46.6. Error and Notice Message Fields

This section describes the fields that can appear in ErrorResponse and NoticeResponse messages. Each field type has a single-byte identification token. Note that any given field type should appear at most once per message.

`S`

Severity: the field contents are `ERROR`, `FATAL`, or `PANIC` (in an error message), or `WARNING`, `NOTICE`, `DEBUG`, `INFO`, or `LOG` (in a notice message), or a localized translation of one of these. Always present.

`C`

Code: the SQLSTATE code for the error (see 附录 A). Not localizable. Always present.

`M`

Message: the primary human-readable error message. This should be accurate but terse (typically one line). Always present.

`D`

Detail: an optional secondary error message carrying more detail about the problem. Might run to multiple lines.

`H`

Hint: an optional suggestion what to do about the problem. This is intended to differ from Detail in that it offers advice (potentially inappropriate) rather than hard facts. Might run to multiple lines.

`P`

Position: the field value is a decimal ASCII integer, indicating an error cursor position as an index into the original query string. The first character has index 1, and positions are measured in characters not bytes.

`p`

Internal position: this is defined the same as the `P` field, but it is used when the cursor position refers to an internally generated command rather than the one submitted by the client. The `q` field will always appear when this field appears.

`q`

Internal query: the text of a failed internally-generated command. This could be, for example, a SQL query issued by a PL/pgSQL function.

`W`

Where: an indication of the context in which the error occurred. Presently this includes a call stack traceback of active procedural language functions and internally-generated queries. The trace is one entry per line, most recent first.

`F`

File: the file name of the source-code location where the error was reported.

`L`

Line: the line number of the source-code location where the error was reported.

R

Routine: the name of the source-code routine reporting the error.

The client is responsible for formatting displayed information to meet its needs; in particular it should break long lines as needed. Newline characters appearing in the error message fields should be treated as paragraph breaks, not line breaks.

46.7. Summary of Changes since Protocol 2.0

This section provides a quick checklist of changes, for the benefit of developers trying to update existing client libraries to protocol 3.0.

The initial startup packet uses a flexible list-of-strings format instead of a fixed format. Notice that session default values for run-time parameters can now be specified directly in the startup packet. (Actually, you could do that before using the `options` field, but given the limited width of `options` and the lack of any way to quote whitespace in the values, it wasn't a very safe technique.)

All messages now have a length count immediately following the message type byte (except for startup packets, which have no type byte). Also note that PasswordMessage now has a type byte.

ErrorResponse and NoticeResponse ('`E`' and '`N`') messages now contain multiple fields, from which the client code can assemble an error message of the desired level of verbosity. Note that individual fields will typically not end with a newline, whereas the single string sent in the older protocol always did.

The ReadyForQuery ('`Z`') message includes a transaction status indicator.

The distinction between BinaryRow and DataRow message types is gone; the single DataRow message type serves for returning data in all formats. Note that the layout of DataRow has changed to make it easier to parse. Also, the representation of binary values has changed: it is no longer directly tied to the server's internal representation.

There is a new “extended query” sub-protocol, which adds the frontend message types Parse, Bind, Execute, Describe, Close, Flush, and Sync, and the backend message types ParseComplete, BindComplete, PortalSuspended, ParameterDescription, NoData, and CloseComplete. Existing clients do not have to concern themselves with this sub-protocol, but making use of it might allow improvements in performance or functionality.

`COPY` data is now encapsulated into CopyData and CopyDone messages. There is a well-defined way to recover from errors during `COPY`. The special “`\.`” last line is not needed anymore, and is not sent during `COPY OUT`. (It is still recognized as a terminator during `COPY IN`, but its use is deprecated and will eventually be removed.) Binary `COPY` is supported. The CopyInResponse and CopyOutResponse messages include fields indicating the number of columns and the format of each column.

The layout of FunctionCall and FunctionCallResponse messages has changed. FunctionCall can now support passing NULL arguments to functions. It also can handle passing parameters and retrieving results in either text or binary format. There is no longer any reason to consider FunctionCall a potential security hole, since it does not offer direct access to internal server data representations.

The backend sends ParameterStatus ('`S`') messages during connection startup for all parameters it considers interesting to the client library. Subsequently, a ParameterStatus message is sent whenever the active value changes for any of these parameters.

The RowDescription ('`T`') message carries new table OID and column number fields for each column of the described row. It also shows the format code for each column.

The CursorResponse ('`P`') message is no longer generated by the backend.

The NotificationResponse ('`A`') message has an additional string field, which can carry a “payload” string passed from the `NOTIFY` event sender.

The EmptyQueryResponse ('`I`') message used to include an empty string parameter; this has been removed.

第 47 章 PostgreSQL 编码约定

47.1. 格式

源代码格式使用 4 列制表宽度，并保留制表符（即，不会把制表符展开为空格）。每个逻辑缩进层级都对应再增加一个制表位。

布局规则（花括号位置等）遵循 BSD 约定。特别是，`if`、`while`、`switch` 等控制语句所对应代码块的花括号都单独占一行。

应限制行长度，使代码在 80 列窗口中可读。（这并不是说绝不能超过 80 列。例如，仅仅为了让代码保持在 80 列内而任意拆开一条很长的错误消息字符串，未必会提升可读性。）

不要使用 C++ 风格注释（`//` 注释）。严格的 ANSI C 编译器不接受这种注释。同样，也不要使用在块中途声明新变量这样的 C++ 扩展。

多行注释块的首选样式是

```
/*
 * comment text begins here
 * and continues here
 */
```

注意，从第 1 列开始的注释块会被 pgindent 原样保留，但它会把缩进的注释块当作普通文本重新排版。如果想保留缩进块中的换行，请像下面这样加上横线：

```
    /*----------
     * comment text begins here
     * and continues here
     *----------
     */
```

尽管提交的补丁并不绝对必须遵循这些格式规则，但这样做是个好主意。
你的代码会在下一个版本发布前经过 pgindent 处理，因此按另一套格式约定把它写得再漂亮也没有意义。补丁的一个经验法则是“让新代码看起来像周围现有的代码”。

`src/tools` 目录中包含可供 emacs、xemacs 或 vim 编辑器使用的示例设置文件，以帮助确保它们按这些约定格式化代码。

文本浏览工具 more 和 less 可以这样调用：

```
more -x4
less -x4
```

以便正确显示制表符。

47.2. 在服务器内部报告错误

服务器代码内生成的错误、警告和日志消息应使用 `ereport`，或其更老的近亲 `elog` 来创建。这一函数的用法相当复杂，因此需要一些解释。

每条消息都必须包含两个元素：严重性级别（范围从 `DEBUG` 到 `PANIC`）以及主消息文本。此外还可以有可选元素，其中最常见的是遵循 SQL 规范 SQLSTATE 约定的错误标识符代码。`ereport` 本身只是一个包装函数，主要为了语法上的便利，使消息生成在 C 源代码中看起来像一次函数调用。`ereport` 唯一直接接受的参数是严重性级别。主消息文本以及任何可选消息元素都是通过在 `ereport` 调用中调用辅助函数（例如 `errmsg`）来生成的。

对`ereport`的一次典型调用可能如下：

```
ereport(ERROR,
        (errcode(ERRCODE_DIVISION_BY_ZERO),
         errmsg("division by zero")));
```

这指定了错误严重性级别ERROR（一种普通错误）。errcode调用使用定义在src/include/utils/errcodes.h中的一个宏指定 SQLSTATE 错误代码。errmsg调用提供主消息文本。注意辅助函数调用外面多了一层圆括号 – 它们虽然烦人，但在语法上是必需的。

这里有一个更复杂的示例：

```
ereport(ERROR,
        (errcode(ERRCODE_AMBIGUOUS_FUNCTION),
         errmsg("function %s is not unique",
                func_signature_string(funcname, nargs,
                                      NIL, actual_arg_types)),
         errhint("Unable to choose a best candidate function. "
                 "You might need to add explicit typecasts.")));
```

这展示了如何用格式代码把运行时值嵌入消息文本中。此外还提供了一条可选的“提示”消息。

适用于ereport的辅助例程有：

- errcode(sqlerrcode) 为该条件指定 SQLSTATE 错误标识符代码。如果不调用这个例程，则默认错误标识符在错误严重性级别为 ERROR 或更高时为 ERRCODE_INTERNAL_ERROR，在错误级别为 WARNING 时为 ERRCODE_WARNING，否则（对于 NOTICE 及以下）为 ERRCODE_SUCCESSFUL_COMPLETION。虽然这些默认值常常很方便，但在省略 errcode() 调用之前，始终要先想想它们是否合适。

- errmsg(const char *msg, ...) 指定主错误消息文本，以及可能要插入其中的运行时值。插入项通过 sprintf 风格的格式代码指定。除了 sprintf 接受的标准格式代码外，还可以使用格式代码 %m 插入 strerror 为 errno 当前值返回的错误消息。[1] %m 不需要在 errmsg 的参数列表中有任何对应项。注意，在处理格式代码之前，消息字符串会先经过 gettext 以便可能进行本地化。

- errmsg_internal(const char *msg, ...) 与 errmsg 相同，只是消息字符串不会被翻译，也不会被收入国际化消息字典。这应当用于那些“不可能发生”、大概不值得为之投入翻译精力的情况。

- errmsg_plural(const char *fmt_singular, const char *fmt_plural, unsigned long n, ...) 类似于 errmsg，但支持消息的各种复数形式。*fmt_singular* 是英文单数格式，*fmt_plural* 是英文复数格式，*n* 是决定需要哪种复数形式的整数值，其余参数按所选格式字符串进行格式化。更多信息见第 48.2.2 节。

- errdetail(const char *msg, ...) 提供一条可选的“详情”消息；当有额外信息但似乎不适合放在主消息中时，可使用它。消息字符串的处理方式与 errmsg 完全相同。

- errdetail_internal(const char *msg, ...) 与 errdetail 相同，只是消息字符串不会被翻译，也不会被收入国际化消息字典。这应当用于那些不值得投入翻译精力的详情消息，例如它们对大多数用户来说技术性太强而无甚用处。

- errdetail_plural(const char *fmt_singular, const char *fmt_plural, unsigned long n, ...) 类似于 errdetail，但支持消息的各种复数形式。更多信息见第 48.2.2 节。

- errdetail_log(const char *msg, ...) 与 errdetail 相同，只是该字符串只会写入服务器日志，绝不会发送给客户端。如果同时使用 errdetail（或其上面的某个等价函

[1] 也就是说，是到达 ereport 调用点时的那个值；辅助报告例程内部对 errno 的更改不会影响它。如果显式写出 strerror(errno) 作为 errmsg 的参数列表内容，就不是这样了；因此不要这么做。

数）和 errdetail_log，那么一条字符串会发往客户端，另一条会发往日志。对于那些因安全性过于敏感或内容过于庞大而不适合放入发给客户端的报告中的错误细节，这很有用。

- errhint(const char *msg, ...) 提供一条可选的“提示”消息；它用于给出如何修复该问题的建议，而不是说明究竟哪里出了错的事实细节。消息字符串的处理方式与 errmsg 完全相同。

- errcontext(const char *msg, ...) 通常不会直接在 ereport 消息处调用；它用于 error_context_stack 回调函数中，提供错误发生时的上下文信息，例如 PL 函数中的当前位置。消息字符串的处理方式与 errmsg 完全相同。与其他辅助函数不同，它在每次 ereport 调用中都可以调用多次；这样提供的各个字符串会以换行分隔并连接起来。

- errposition(int cursorpos) 指定错误在查询字符串中的文本位置。目前，它只对在查询处理的词法和语法分析阶段检测到的错误有用。

- errcode_for_file_access() 是一个便捷函数，用于为与文件访问相关的系统调用失败选择合适的 SQLSTATE 错误标识符。它使用保存下来的 errno 来确定要生成哪种错误代码。通常应将它与主错误消息文本中的 %m 结合使用。

- errcode_for_socket_access() 是一个便捷函数，用于为与套接字相关的系统调用失败选择合适的 SQLSTATE 错误标识符。

- 可以调用 errhidestmt(bool hide_stmt) 来指定抑制 postmaster 日志中消息的 STATEMENT: 部分。一般来说，如果消息文本本身已经包含当前语句，这么做就是合适的。

还有一个较旧的函数elog，至今仍被大量使用。一个elog调用：

```
elog(level, "format string", ...);
```

完全等价于：

```
ereport(level, (errmsg_internal("format string", ...)));
```

注意，SQLSTATE 错误代码总会取默认值，而且消息字符串不会被翻译。因此，elog只应用于内部错误和低层调试日志。凡是普通用户可能感兴趣的消息，都应通过ereport。尽管如此，系统中仍有足够多的内部“不可能发生”错误检查，因此elog依然被广泛使用；对这类消息来说，由于记法更简洁，它更受青睐。

关于如何编写良好的错误消息，可参见第 47.3 节。

47.3. 错误消息风格指南

提供这份风格指南，是希望在 PostgreSQL 生成的所有消息中保持一致、对用户友好的风格。

各部分写什么

主消息应简短、陈述事实，并避免提及具体函数名等实现细节。“简短”的意思是“在正常情况下应能放在一行里”。如果为了保持主消息简短，或者你觉得有必要提及特定失败的系统调用之类的实现细节，可使用详情消息。主消息和详情消息都应当陈述事实。提示消息则用于给出如何修复问题的建议，尤其是在该建议未必总是适用时。

例如，不要写：

```
IpcMemoryCreate: shmget(key=%d, size=%u, 0%o) failed: %m
(plus a long addendum that is basically a hint)
```

而要写：

```
Primary:    could not create shared memory segment: %m
Detail:     Failed syscall was shmget(key=%d, size=%u, 0%o).
Hint:       the addendum
```

原理：保持主消息简短，有助于使其切中要点，也便于客户端假定一行足以容纳错误消息，从而安排屏幕空间。详情消息和提示消息可以归入详细模式，或者放到弹出的错误详情窗口中。此外，为节省空间，详情消息和提示消息通常会从服务器日志中省略。最好避免提及实现细节，因为用户反正也不知道这些细节。

格式

不要在消息文本中写入任何特定的格式假设。应预期客户端和服务器日志会按照各自需要折行。对于较长的消息，可以使用换行字符（\n）来表示建议的分段。不要让消息以换行结束。不要使用制表符或其他格式控制字符。（在错误上下文显示中，会自动添加换行以分隔函数调用等上下文层次。）

原理：消息不一定显示在终端类设备上。在 GUI 显示或浏览器中，这些格式指令充其量也只是被忽略。

引号

英文文本在适合引用时应使用双引号。其他语言的文本应一致地使用一种引号，这种引号应符合该语言的出版习惯以及其他程序的计算机输出习惯。

原理：选择双引号而不是单引号多少有些武断，但通常更符合首选用法。有人建议按照 SQL 约定，根据对象类型来选择引号的种类（即字符串用单引号，标识符用双引号）。但这是语言内部的技术问题，很多用户甚至并不熟悉；它也无法扩展到其他类型的被引术语，更无法翻译到其他语言中，而且本身也没什么意义。

引号的用法

对文件名、用户提供的标识符以及其他可能包含单词的变量，始终使用引号界定。对于不会包含单词的变量（例如操作符名），则不要加引号。

后端中有些函数会在需要时自行给其输出加上双引号（例如 `format_type_be()`）。不要再对这类函数的输出额外加引号。

原理：对象的名称在嵌入消息时可能造成歧义。对于插入的名称从哪里开始、到哪里结束，要保持一致。但也不要用不必要或重复的引号把消息弄得杂乱无章。

语法和标点

主错误消息与详情/提示消息的规则不同：

主错误消息：首字母不要大写。消息末尾不要加句号。更不要想着在消息末尾加感叹号。

详情消息和提示消息：使用完整句子，并且每条都用句号结束。句子的第一个词要首字母大写。如果后面还有一句，在句号后放两个空格（针对英文文本；对其他语言可能并不合适）。

错误上下文字符串：首字母不要大写，字符串末尾不要加句号。上下文字符串通常不应是完整句子。

原理：避免使用标点，能让客户端应用更容易把消息嵌入各种语法上下文中。主消息往往本来也不是语法完整的句子。（而如果它们长到不止一句，

就应当拆分成主消息和详情消息。）不过，详情消息和提示消息更长，
也可能需要包含多个句子。为了保持一致，即便只有一个句子，
它们也应遵循完整句子的风格。

大写与小写

消息措辞应使用小写，包括主错误消息的首字母。如果消息中出现 SQL 命令或关键字，则使用大写形式。

原理：这样更容易让所有消息看起来一致，因为有些消息是完整句子，而有些则不是。

避免被动语态

使用主动语态。有施动者时，用完整句子（“A could not do B”）。如果施动者就是程序本身，则使用无主语的电报式写法，但不要把程序写成“I”。

原理：程序不是人。不要假装它是。

现在时与过去时

如果一次尝试做某事失败了，但下次仍可能成功（也许在修复某个问题之后），
就使用过去时。如果失败显然是永久性的，就使用现在时。

下面两种句式在语义上有明显差别：

```
could not open file "%s": %m
```

和：

```
cannot open file "%s"
```

第一种表示尝试打开文件失败了。消息应给出原因，例如“磁盘已满”或“文件不存在”。
过去时更合适，因为下次磁盘可能就不满了，或者所请求的文件可能已经存在。

第二种形式表示，程序中根本不存在打开该命名文件的功能，或者从概念上就不可能。
现在时更合适，因为这种情况会无限期持续下去。

原理：诚然，普通用户未必能仅凭消息的时态得出什么重要结论，但既然语言提供了语法，
我们就应当正确使用它。

对象类型

在引用对象名称时，要说明它是什么类型的对象。

原理：否则没人会知道“foo.bar.baz”指的是什么。

括号

方括号只应用于：（1）命令概要中表示可选参数，或（2）表示数组下标。

原理：其他任何用法都不符合广为人知的习惯用法，而且会让人困惑。

组装错误消息

当消息中包含其他地方生成的文本时，应按以下方式嵌入：

```
could not open file %s: %m
```

原理：要把这种文本拼进一个流畅的单句中，同时照顾所有可能的错误代码会很困难，
因此需要某种标点。也有人建议把嵌入的文本放在圆括号里，
但如果嵌入的文本很可能是消息中最重要的部分（而这往往确实如此），
那样就显得不自然了。

错误原因

消息总应说明错误发生的原因。例如：

```
BAD:    could not open file %s
BETTER: could not open file %s (I/O failure)
```

如果不知道原因，最好去修复代码。

函数名

不要在错误文本中包含报告该错误的例程名。需要时，我们有其他机制可以找出它，
而且对大多数用户来说，这项信息没有帮助。如果去掉函数名后错误文本就不够清楚，
请重新措辞。

```
BAD:    pg_atoi: error in "z": cannot parse "z"
BETTER: invalid input syntax for integer: "z"
```

也要避免提及被调用函数的名称；应改为说明代码想要做什么：

```
BAD:    open() failed: %m
BETTER: could not open file %s: %m
```

如果确实有必要，可在详情消息中提及系统调用。（在某些情况下，
在详情消息中提供传给该系统调用的实际值也许是合适的信息。）

原理：用户并不知道那些函数都做了什么。

应避免的词语

Unable. “Unable” 几乎就是被动语态。更好的做法是视情况使用 “cannot” 或 “could not” 。

Bad. 像 “bad result” 这样的错误消息，实在很难被合理理解。最好写出结果为什么是 “bad” ，例如 “invalid format” 。

Illegal. “Illegal” 指的是违法，其余情况应使用 “invalid” 。更好的是，直接说明为什么无效。

Unknown. 尽量避免 “unknown” 。考虑 “error: unknown response” 。如果你都不知道响应是什么，又如何知道它有错？ “Unrecognized” 往往是更好的选择。此外，
一定要包含被抱怨的值。

```
BAD:    unknown node type
BETTER: unrecognized node type: 42
```

Find vs. Exists. 如果程序为了定位某个资源使用了非平凡算法（例如路径搜索），而该算法失败了，那么说程序没能 “find” 该资源是公平的。另一方面，如果资源的预期位置已知，
但程序无法在那里访问它，那么就应说该资源并不 “exist” 。在这种情况下使用 “find”
显得含糊，也会混淆问题。

May、Can 与 Might. “May” 暗示许可（例如，"You may borrow my rake."），在文档或错误消息中用途不大。 “Can” 暗示能力（例如，"I can lift that log."），而 “might” 暗示可能性（例如，"It might rain today."）。使用恰当的词能澄清含义，也有助于翻译。

Contractions. 避免使用缩略形式，例如“can't”；应改用“cannot”。

完整拼写

单词应完整拼写。例如，避免使用：

- spec
- stats
- parens
- auth
- xact

原理：这会提升一致性。

本地化

请记住，错误消息文本需要被翻译成其他语言。请遵循第 48.2.2 节中的指导，避免给翻译者制造麻烦。

第 48 章 本地语言支持

Peter Eisentraut

48.1. 给翻译者

PostgreSQL 程序（服务器和客户端）可以用你所偏好的语言发出消息 – 前提是这些消息已经被翻译。创建和维护翻译后的消息集，需要那些精通自己语言并愿意为 PostgreSQL 项目作出贡献的人的帮助。做这件事完全不必是程序员。本节说明如何提供帮助。

48.1.1. 要求

我们不会评判你的语言能力 – 本节讨论的是软件工具。理论上，你只需要一个文本编辑器。但这只有在你不打算试用自己翻译的消息时才成立，而这种情况不太可能发生。
配置你的源代码树时，请务必使用 `--enable-nls` 选项。这还会检查 libintl 库和 `msgfmt` 程序，而终端用户无论如何也需要它们。要试用你的工作成果，请遵循安装说明中适用的部分。

如果你想开始一项新的翻译工作，或者想执行一次消息目录合并（后文会描述），
则分别需要 GNU 兼容实现的 `xgettext` 和 `msgmerge` 程序。今后我们会尽量做到：如果你使用打包的源码发布版，就不需要 `xgettext` 了。（如果从 Git 工作，你仍然需要它。）目前推荐使用 GNU Gettext 0.10.36 或更高版本。

你本地的 gettext 实现应该自带文档。其中有些内容可能与下文重复，但要了解更多细节，你应当去查阅那些文档。

48.1.2. 概念

原始（英语）消息及其（可能存在的）译文对应项，以成对形式保存在消息目录中；
每个程序以及每种目标语言各有一份消息目录（尽管相关程序可以共享同一个消息目录）。
消息目录有两种文件格式：第一种是“PO”文件（Portable Object，可移植对象），它是带有特殊语法的纯文本文件，由翻译者编辑。第二种是“MO”文件（Machine Object，机器对象），它是根据相应的 PO 文件生成的二进制文件，在国际化后的程序运行时使用。翻译者不需要处理 MO 文件；事实上几乎没有人需要。

消息目录文件的扩展名，毫不意外地是 `.po` 或 `.mo`。基本名则视情况而定，要么是其所对应的程序名，要么是该文件所对应的语言。这多少有些令人困惑。例子包括 `psql.po`（psql 的 PO 文件）或 `fr.mo`（法语的 MO 文件）。

PO 文件的格式示例如下：

```
# comment

msgid "original string"
msgstr "translated string"

msgid "more original"
msgstr "another translated"
"string can be broken up like this"

...
```

msgid 行是从程序源代码中提取出来的。（并非一定如此，但这是最常见的做法。）msgstr 行起初为空，由翻译者填入有意义的字符串。字符串可以包含 C 风格的转义字符，也可以像上例那样跨多行延续。（下一行必须从行首开始。）

字符引入注释。如果 # 字符后面紧跟空白，那么这就是由翻译者维护的注释。也可能存在自动注释，即 # 后紧跟非空白字符的注释。这些注释由各种处理 PO 文件的工具维护，目的是帮助翻译者。

```
#. automatic comment
#: filename.c:1023
#, flags, flags
```

#. 风格的注释是从使用该消息的源文件中提取出来的。程序员可能已经为翻译者插入了信息，例如期望的对齐方式。#: 注释指出该消息在源代码中使用的精确位置。翻译者不必查看程序源代码，但如果对正确译法有疑问，也可以去看。#, 注释包含以某种方式描述该消息的标志。目前有两种标志：如果某条消息可能因程序源代码的更改而过期，则会设置 `fuzzy`。翻译者随后可以核实这一点，并在可能时移除 fuzzy 标志。注意，fuzzy 消息不会提供给终端用户。另一个标志是 `c-format`，它表示该消息是一个 `printf` 风格的格式模板。这意味着译文也应该是一个格式字符串，并具有相同数量和类型的占位符。
有一些工具会据此验证格式字符串，它们就是根据 c-format 标志来检查的。

48.1.3. 创建和维护消息目录

好，那么该如何创建一个“空白”消息目录呢？首先，
进入包含你想翻译其消息的程序的目录。如果那里有一个 `nls.mk` 文件，就说明这个程序已经为翻译做好准备了。

如果已经有一些 `.po` 文件，那么就说明已经有人做过部分翻译工作。这些文件命名为 *language*`.po`，其中 *language* 是 ISO 639-1 两字母语言代码（小写）[1]，例如法语对应 `fr.po`。如果某种语言确实需要进行多套翻译工作，这些文件也可以命名为 *language*_*region*`.po`，其中 *region* 是 ISO 3166-1 两字母国家代码（大写）[2]，例如巴西葡萄牙语对应 `pt_BR.po`。如果你找到了自己想要的语言，就可以直接开始编辑那个文件。

如果你需要开始一项新的翻译工作，先运行以下命令：

```
gmake init-po
```

这会创建一个文件*progname*`.pot`。（`.pot`用来与已经“投入使用”的 PO 文件区分开来；`T` 代表“template”。）把这个文件复制为*language*`.po`并进行编辑。
为了让系统知道新语言已经可用，还要编辑文件`nls.mk`，
并在类似下面的行中添加该语言（或语言加国家）代码：

```
AVAIL_LANGUAGES := de fr
```

（当然也可能有其他语言。）

随着底层程序或库发生变化，程序员可能会修改或增加消息。
这种情况下你不需要从头开始。相反，请运行以下命令：

```
gmake update-po
```

它会创建一个新的空白消息目录文件（也就是你最初使用的那个 pot 文件），并将它与现有的 PO 文件合并。如果合并算法对某条消息拿不准，就会像上文所述那样把它标记为“fuzzy”。新的 PO 文件会以 `.po.new` 作为扩展名保存。

48.1.4. 编辑 PO 文件

PO 文件可以用普通文本编辑器编辑。翻译者只应修改 msgstr 指令后引号之间的内容、添加注释，以及调整 fuzzy 标志。Emacs 里（不出意外地）有一个 PO 模式，相当有用。

[1] http://www.loc.gov/standards/iso639-2/php/English_list.php
[2] http://www.iso.org/iso/country_names_and_code_elements

PO 文件不必完全填满。如果没有可用译文（或译文为空），软件会自动回退到原始字符串。把不完整的翻译提交到源码树中也没有问题；这样别人就有机会接手你的工作。不过，完成一次合并之后，建议优先清除 fuzzy 条目。记住，fuzzy 条目不会被安装；它们只用来作为可能正确译文的参考。

编辑译文时要记住以下几点：

- 务必确保如果原文以换行结束，译文也同样以换行结束。制表符等也是如此。
- 如果原文是一个 `printf` 格式字符串，译文也必须如此。译文还需要按相同顺序保留相同的格式说明符。有时语言自身的规则会让这一点变得不可能，或者至少很别扭。在这种情况下，你可以像下面这样修改格式说明符：

  ```
  msgstr "Die Datei %2$s hat %1$u Zeichen."
  ```

  这样一来，第一个占位符实际上会使用参数列表中的第二个参数。*digits*`$` 必须紧跟在 `%` 后面，位于其他任何格式修饰符之前。（这个特性确实存在于 `printf` 函数族中。你以前可能没听说过它，因为在消息国际化之外，它几乎没有用武之地。）
- 如果原始字符串包含语言错误，就报告它（或者你自己在程序源代码中修正它），然后照常翻译。等程序源代码更新之后，修正过的字符串就可以被合并进来。如果原始字符串包含事实性错误，就报告它（或者你自己修正它），并且不要翻译它。相反，你可以在 PO 文件里为该字符串加上一条注释。
- 保持原始字符串的风格和语气。特别是，那些不是完整句子的消息（`cannot open file %s`）通常不应以大写字母开头（如果你的语言区分大小写），也不应以句号结尾（如果你的语言使用标点符号）。阅读第 47.3 节可能会有帮助。
- 如果你不知道某条消息是什么意思，或者它有歧义，就到开发者邮件列表里提问。很可能说英语的终端用户也同样看不懂它，或者会觉得它有歧义，因此最好直接改进那条消息。

48.2. 给程序员

48.2.1. 实现机制

本节描述如何在 PostgreSQL 发行版中的程序或库里实现本地语言支持。目前它只适用于 C 程序。

为程序添加 NLS 支持

1. 把下面的代码插入到该程序的启动序列中：

   ```
   #ifdef ENABLE_NLS
   #include <locale.h>
   #endif

   ...

   #ifdef ENABLE_NLS
   setlocale(LC_ALL, "");
   bindtextdomain("progname", LOCALEDIR);
   textdomain("progname");
   #endif
   ```

   （其中 *progname* 实际上可以自由选择。）

2. 凡是遇到可能需要翻译的消息，都需要插入对 `gettext()` 的调用。例如：

```
fprintf(stderr, "panic level %d\n", lvl);
```

将改成：

```
fprintf(stderr, gettext("panic level %d\n"), lvl);
```

（如果没有配置 NLS 支持，`gettext` 会被定义成一个空操作。）

这样往往会增加很多杂乱内容。一个常用的简写是：

```
#define _(x) gettext(x)
```

如果该程序的大部分通信都是通过一个或少数几个函数完成的，例如后端中的 `ereport()`，另一种可行的解决办法是让这个函数在内部对所有输入字符串调用 `gettext`。

3. 在程序源代码所在目录中添加一个`nls.mk`文件。这个文件会作为 makefile 读取。这里需要设置以下变量：

   `CATALOG_NAME`

   程序名，即 `textdomain()` 调用中提供的名称。

   `AVAIL_LANGUAGES`

   已提供的翻译列表 – 最初为空。

   `GETTEXT_FILES`

   包含可翻译字符串的文件列表，也就是那些使用 `gettext` 或其他替代方案标记过的文件。最终，这会包含该程序几乎所有的源文件。如果这个列表太长，可以让第一个“文件”是一个 `+`，第二个词则是一个文件名，该文件每行包含一个文件名。

   `GETTEXT_TRIGGERS`

   为翻译者生成消息目录的工具需要知道哪些函数调用包含可翻译字符串。默认只认识 `gettext()` 调用。如果你使用了 `_` 或其他标识符，就需要在这里列出它们。如果可翻译字符串不是第一个参数，条目就需要写成 `func:2` 这样的形式（表示第二个参数）。如果你有一个支持复数形式消息的函数，条目应写成 `func:1,2` 这样（分别标识单数和复数消息参数）。

构建系统会自动处理消息目录的构建和安装。

48.2.2. 消息编写指南

下面列出一些便于翻译的消息编写指南。

- 不要像下面这样在运行时拼接句子：

  ```
  printf("Files were %s.\n", flag ? "copied" : "removed");
  ```

  句子中的词序在其他语言里可能完全不同。另外，即使你记得对每个片段都调用 `gettext()`，这些片段分开来看也可能难以妥善翻译。最好稍微重复一点代码，让每条待翻译消息都成为一个语义完整的整体。只有数字、文件名之类的运行时变量才应该在运行时插入消息文本中。

- 出于类似的原因，下面这样也行不通：

  ```
  printf("copied %d file%s", n, n!=1 ? "s" : "");
  ```

因为它假定了复数形式的构造方式。如果你以为可以这样解决：

```
if (n==1)
    printf("copied 1 file");
else
    printf("copied %d files", n):
```

那就会失望了。有些语言的复数形式不止两种，而且规则相当特殊。
通常最好从设计上彻底避开这个问题，例如这样：

```
printf("number of copied files: %d", n);
```

如果你确实想构造一条正确处理复数的消息，也有相应支持，只是有点别扭。在 `ereport()` 中生成主错误消息或详细错误消息时，可以写成这样：

```
errmsg_plural("copied %d file",
              "copied %d files",
              n,
              n)
```

第一个参数是适用于英语单数形式的格式字符串，
第二个参数是适用于英语复数形式的格式字符串，
第三个参数是决定使用哪种复数形式的整数控制值。
后续参数会像往常一样按照格式字符串进行格式化。（通常，
这个复数控制值本身也会是要格式化的值之一，所以必须写两次。）在英语中，
只需要区分 *n* 是否为 1；而在其他语言中，可能存在许多不同的复数形式。翻译者会把这两个英语形式视为一组，并有机会提供多个替代字符串，再根据运行时的 *n* 值选择合适的那个。

如果你需要对一条不是直接用于 `errmsg` 或 `errdetail` 报告的消息做复数处理，就必须使用底层函数 `ngettext`。参见 gettext 文档。

- 如果你想向翻译者传达某些信息，例如一条消息应如何与其他输出对齐，
  就在该字符串出现之前放置一个以 `translator` 开头的注释，例如：

  ```
  /* translator: This message is not what it seems to be. */
  ```

  这些注释会被复制到消息目录文件中，这样翻译者就能看到它们。

第 49 章 编写过程语言调用处理器

凡是不是使用当前针对编译型语言的“版本 1”接口编写的函数，在被调用时都会经过该语言专用的调用处理器函数。这包括用户定义过程语言中的函数、用 SQL 编写的函数，以及使用版本 0 编译语言接口的函数。调用处理器负责以恰当的方式执行该函数，例如解释所提供的源文本。本章概述如何编写新的过程语言调用处理器。

过程语言的调用处理器是一个“普通”函数，必须使用诸如 C 这样的编译型语言、按照版本 1 接口编写，并在 PostgreSQL 中注册为不接受参数且返回 `language_handler` 类型。这个特殊伪类型会将该函数标识为调用处理器，并阻止它在 SQL 命令中被直接调用。有关 C 语言调用约定和动态装载的更多细节，见第 35.9 节。

调用处理器与其他函数的调用方式相同：它接收一个指向 `FunctionCallInfoData struct` 的指针，其中包含参数值和被调用函数的信息，并应返回一个 `Datum` 结果（如果要返回 SQL 空值，还需要设置 `FunctionCallInfoData` 结构体的 `isnull` 字段）。调用处理器与普通被调用函数的区别在于，`FunctionCallInfoData` 结构体的 `flinfo->fn_oid` 字段包含的是实际要调用的函数的 OID，而不是调用处理器自身的 OID。调用处理器必须使用该字段确定应执行哪个函数。此外，传入的参数列表也是按照目标函数的声明设置的，而不是按照调用处理器的声明。

调用处理器必须从系统目录 `pg_proc` 中取出该函数的条目，并分析被调用函数的参数类型和返回类型。该函数的 `CREATE FUNCTION` 命令中的 `AS` 子句内容，保存在 `pg_proc` 对应行的 `prosrc` 列中。这里通常是一段过程语言源文本，但理论上也可以是其他内容，例如某个文件的路径名，或者任何能详细告诉调用处理器该做什么的信息。

同一个函数在执行一条 SQL 语句期间往往会被调用很多次。调用处理器可以利用 `flinfo->fn_extra` 字段，避免重复查找被调用函数的信息。该字段起初为 `NULL`，但调用处理器可以把它设置为指向与被调用函数有关的信息。在后续调用中，如果 `flinfo->fn_extra` 已经不是 `NULL`，就可以直接使用它并跳过信息查找步骤。调用处理器必须确保 `flinfo->fn_extra` 指向的内存至少能存活到当前查询结束，因为 `FmgrInfo` 数据结构可能会保留这么久。一种做法是在 `flinfo->fn_mcxt` 指定的内存上下文中分配这些额外数据；这类数据通常会与 `FmgrInfo` 本身具有相同的生命周期。不过，处理器也可以选择使用生命周期更长的内存上下文，以便跨查询缓存函数定义信息。

过程语言函数作为触发器调用时，不会按通常方式传入参数，但 `FunctionCallInfoData` 的 `context` 字段会指向一个 `TriggerData` 结构体，而不是像普通函数调用那样为 `NULL`。语言调用处理器应提供让过程语言函数获取触发器信息的机制。

下面是用 C 编写的过程语言处理器模板：

```
#include "postgres.h"
#include "executor/spi.h"
#include "commands/trigger.h"
#include "fmgr.h"
#include "access/heapam.h"
#include "utils/syscache.h"
#include "catalog/pg_proc.h"
#include "catalog/pg_type.h"

#ifdef PG_MODULE_MAGIC
PG_MODULE_MAGIC;
#endif

PG_FUNCTION_INFO_V1(plsample_call_handler);

Datum
plsample_call_handler(PG_FUNCTION_ARGS)
```

```
{
    Datum          retval;

    if (CALLED_AS_TRIGGER(fcinfo))
    {
        /*
         * 作为触发器过程调用
         */
        TriggerData    *trigdata = (TriggerData *) fcinfo->context;

        retval = ...
    }
    else
    {
        /*
         * 作为函数调用
         */

        retval = ...
    }

    return retval;
}
```

只需在省略号处添加几千行代码，就能完成这个调用处理器。

将处理器函数编译为可加载模块后（参见第 35.9.6 节），可用以下命令注册示例过程语言：

```
CREATE FUNCTION plsample_call_handler() RETURNS language_handler
    AS 'filename'
    LANGUAGE C;
CREATE LANGUAGE plsample
    HANDLER plsample_call_handler;
```

虽然仅提供调用处理器就足以创建一种最小可用的过程语言，
但还可以额外提供另外两个函数，以便让该语言更易于使用。它们是验证器和内联处理器。
可以提供验证器，以便在CREATE FUNCTION期间执行与该语言相关的检查。也可以提供内联处理器，以便让该语言支持通过DO命令执行匿名代码块。

如果过程语言提供了验证器，它必须声明为一个接受单个 `oid` 参数的函数。验证器的返回值会被忽略，因此习惯上将其声明为返回 `void`。在 `CREATE FUNCTION` 命令创建或更新了一个用该过程语言编写的函数之后，验证器会在该命令结束时被调用。传入的 OID 是该函数在 `pg_proc` 中对应行的 OID。验证器必须按通常方式取出这一行，并执行适当的检查。首先，应调用 `CheckFunctionValidatorAccess()`，以诊断那些用户无法通过 `CREATE FUNCTION` 实现的对验证器的显式调用。典型检查包括确认该语言支持该函数的参数类型和结果类型，以及确认函数体在该语言中语法正确。如果验证器认为该函数没有问题，
就应直接返回；如果发现错误，则应通过常规的 `ereport()` 错误报告机制加以报告。抛出错误会强制事务回滚，从而阻止错误的函数定义被提交。

验证器函数通常应遵守check_function_bodies参数：关闭该参数时，
应跳过代价高昂或依赖上下文的检查。如果语言允许在编译时执行代码，
验证器必须禁止会引发这种执行的检查。特别是，pg_dump 会关闭此参数，以便装载过程语言函数，而无需担心副作用或函数体对其他数据库对象的依赖。（因此，
调用处理器不应假定验证器已经完整检查了函数。
验证器的目的不是让调用处理器省略检查，而是在 `CREATE FUNCTION` 命令存在明显错误时立即通知用户。）虽然具体检查哪些内容主要由验证器函数自行决定，但要注意，核心 `CREATE FUNCTION` 代码只有在 `check_function_bodies` 开启时，才执行函数附带的 `SET` 子句。因此，关闭 `check_function_bodies` 时，必须跳过那些结果可能受 GUC 参数影响的检查，以免重新装载转储时误报错误。

如果过程语言提供了内联处理器，它必须声明为一个接受单个 `internal` 参数的函数。内联处理器的返回值会被忽略，因此习惯上将其声明为返回 `void`。当执行指定了该过程语言的 `DO` 语句时，就会调用内联处理器。实际传入的参数是一个指向 `InlineCodeBlock` 结构体的指针，其中包含 `DO` 语句参数的相关信息，特别是待执行的匿名代码块文本。内联处理器应执行这段代码并返回。

建议你将以上所有函数声明以及 `CREATE LANGUAGE` 命令本身都封装进一个扩展中，这样只需执行一条简单的 `CREATE EXTENSION` 命令就足以安装该语言。关于如何编写扩展，见第 35.15 节。

标准发行版中附带的各过程语言，
在尝试编写自己的语言调用处理器时都是很好的参考资料。请查看源码树中的 `src/pl` 子目录。CREATE LANGUAGE参考页中也包含一些有用的细节。

第 50 章 Writing A Foreign Data Wrapper

All operations on a foreign table are handled through its foreign data wrapper, which consists of a set of functions that the core server calls. The foreign data wrapper is responsible for fetching data from the remote data source and returning it to the PostgreSQL executor. This chapter outlines how to write a new foreign data wrapper.

The foreign data wrappers included in the standard distribution are good references when trying to write your own. Look into the `contrib/file_fdw` subdirectory of the source tree. The CREATE FOREIGN DATA WRAPPER reference page also has some useful details.

> **注意**
>
> The SQL standard specifies an interface for writing foreign data wrappers. However, PostgreSQL does not implement that API, because the effort to accommodate it into PostgreSQL would be large, and the standard API hasn't gained wide adoption anyway.

50.1. Foreign Data Wrapper Functions

The FDW author needs to implement a handler function, and optionally a validator function. Both functions must be written in a compiled language such as C, using the version-1 interface. For details on C language calling conventions and dynamic loading, see 第 35.9 节.

The handler function simply returns a struct of function pointers to callback functions that will be called by the planner, executor, and various maintenance commands. Most of the effort in writing an FDW is in implementing these callback functions. The handler function must be registered with PostgreSQL as taking no arguments and returning the special pseudo-type `fdw_handler`. The callback functions are plain C functions and are not visible or callable at the SQL level. The callback functions are described in 第 50.2 节.

The validator function is responsible for validating options given in `CREATE` and `ALTER` commands for its foreign data wrapper, as well as foreign servers, user mappings, and foreign tables using the wrapper. The validator function must be registered as taking two arguments, a text array containing the options to be validated, and an OID representing the type of object the options are associated with (in the form of the OID of the system catalog the object would be stored in, either `ForeignDataWrapperRelationId`, `ForeignServerRelationId`, `UserMappingRelationId`, or `ForeignTableRelationId`). If no validator function is supplied, options are not checked at object creation time or object alteration time.

50.2. Foreign Data Wrapper Callback Routines

The FDW handler function returns a palloc'd `FdwRoutine` struct containing pointers to the following callback functions:

```
void
```

```
GetForeignRelSize (PlannerInfo *root,
                   RelOptInfo *baserel,
                   Oid foreigntableid);
```

Obtain relation size estimates for a foreign table. This is called at the beginning of planning for a query involving a foreign table. `root` is the planner's global information about the query; `baserel` is the planner's information about this table; and `foreigntableid` is the `pg_class` OID of the foreign table. (`foreigntableid` could be obtained from the planner data structures, but it's passed explicitly to save effort.)

This function should update `baserel->rows` to be the expected number of rows returned by the table scan, after accounting for the filtering done by the restriction quals. The initial value of `baserel->rows` is just a constant default estimate, which should be replaced if at all possible. The function may also choose to update `baserel->width` if it can compute a better estimate of the average result row width.

See 第 50.4 节 for additional information.

```
void
GetForeignPaths (PlannerInfo *root,
                 RelOptInfo *baserel,
                 Oid foreigntableid);
```

Create possible access paths for a scan on a foreign table. This is called during query planning. The parameters are the same as for `GetForeignRelSize`, which has already been called.

This function must generate at least one access path (`ForeignPath` node) for a scan on the foreign table and must call `add_path` to add each such path to `baserel->pathlist`. It's recommended to use `create_foreignscan_path` to build the `ForeignPath` nodes. The function can generate multiple access paths, e.g., a path which has valid `pathkeys` to represent a pre-sorted result. Each access path must contain cost estimates, and can contain any FDW-private information that is needed to identify the specific scan method intended.

See 第 50.4 节 for additional information.

```
ForeignScan *
GetForeignPlan (PlannerInfo *root,
                RelOptInfo *baserel,
                Oid foreigntableid,
                ForeignPath *best_path,
                List *tlist,
                List *scan_clauses);
```

Create a `ForeignScan` plan node from the selected foreign access path. This is called at the end of query planning. The parameters are as for `GetForeignRelSize`, plus the selected `ForeignPath` (previously produced by `GetForeignPaths`), the target list to be emitted by the plan node, and the restriction clauses to be enforced by the plan node.

This function must create and return a `ForeignScan` plan node; it's recommended to use `make_foreignscan` to build the `ForeignScan` node.

See 第 50.4 节 for additional information.

```
void
ExplainForeignScan (ForeignScanState *node,
                    ExplainState *es);
```

Print additional EXPLAIN output for a foreign table scan. This can just return if there is no need to print anything. Otherwise, it should call ExplainPropertyText and related functions to add fields to the EXPLAIN output. The flag fields in es can be used to determine what to print, and the state of the ForeignScanState node can be inspected to provide run-time statistics in the EXPLAIN ANALYZE case.

```
void
BeginForeignScan (ForeignScanState *node,
                  int eflags);
```

Begin executing a foreign scan. This is called during executor startup. It should perform any initialization needed before the scan can start, but not start executing the actual scan (that should be done upon the first call to IterateForeignScan). The ForeignScanState node has already been created, but its fdw_state field is still NULL. Information about the table to scan is accessible through the ForeignScanState node (in particular, from the underlying ForeignScan plan node, which contains any FDW-private information provided by GetForeignPlan).

Note that when (eflags & EXEC_FLAG_EXPLAIN_ONLY) is true, this function should not perform any externally-visible actions; it should only do the minimum required to make the node state valid for ExplainForeignScan and EndForeignScan.

```
TupleTableSlot *
IterateForeignScan (ForeignScanState *node);
```

Fetch one row from the foreign source, returning it in a tuple table slot (the node's ScanTupleSlot should be used for this purpose). Return NULL if no more rows are available. The tuple table slot infrastructure allows either a physical or virtual tuple to be returned; in most cases the latter choice is preferable from a performance standpoint. Note that this is called in a short-lived memory context that will be reset between invocations. Create a memory context in BeginForeignScan if you need longer-lived storage, or use the es_query_cxt of the node's EState.

The rows returned must match the column signature of the foreign table being scanned. If you choose to optimize away fetching columns that are not needed, you should insert nulls in those column positions.

Note that PostgreSQL's executor doesn't care whether the rows returned violate the NOT NULL constraints which were defined on the foreign table columns - but the planner does care, and may optimize queries incorrectly if NULL values are present in a column declared not to contain them. If a NULL value is encountered when the user has declared that none should be present, it may be appropriate to raise an error (just as you would need to do in the case of a data type mismatch).

```
void
ReScanForeignScan (ForeignScanState *node);
```

Restart the scan from the beginning. Note that any parameters the scan depends on may have changed value, so the new scan does not necessarily return exactly the same rows.

```
void
EndForeignScan (ForeignScanState *node);
```

End the scan and release resources. It is normally not important to release palloc'd memory, but for example open files and connections to remote servers should be cleaned up.

```
bool
AnalyzeForeignTable (Relation relation,
```

```
                        AcquireSampleRowsFunc *func,
                        BlockNumber *totalpages);
```

This function is called when ANALYZE is executed on a foreign table. If the FDW can collect statistics for this foreign table, it should return `true`, and provide a pointer to a function that will collect sample rows from the table in *`func`*, plus the estimated size of the table in pages in *`totalpages`*. Otherwise, return `false`. If the FDW does not support collecting statistics for any tables, the `AnalyzeForeignTable` pointer can be set to `NULL`.

If provided, the sample collection function must have the signature

```
int
AcquireSampleRowsFunc (Relation relation, int elevel,
                       HeapTuple *rows, int targrows,
                       double *totalrows,
                       double *totaldeadrows);
```

A random sample of up to *`targrows`* rows should be collected from the table and stored into the caller-provided *`rows`* array. The actual number of rows collected must be returned. In addition, store estimates of the total numbers of live and dead rows in the table into the output parameters *`totalrows`* and *`totaldeadrows`*. (Set *`totaldeadrows`* to zero if the FDW does not have any concept of dead rows.)

The `FdwRoutine` struct type is declared in `src/include/foreign/fdwapi.h`, which see for additional details.

50.3. Foreign Data Wrapper Helper Functions

Several helper functions are exported from the core server so that authors of foreign data wrappers can get easy access to attributes of FDW-related objects, such as FDW options. To use any of these functions, you need to include the header file `foreign/foreign.h` in your source file. That header also defines the struct types that are returned by these functions.

```
ForeignDataWrapper *
GetForeignDataWrapper(Oid fdwid);
```

This function returns a `ForeignDataWrapper` object for the foreign-data wrapper with the given OID. A `ForeignDataWrapper` object contains properties of the FDW (see `foreign/foreign.h` for details).

```
ForeignServer *
GetForeignServer(Oid serverid);
```

This function returns a `ForeignServer` object for the foreign server with the given OID. A `ForeignServer` object contains properties of the server (see `foreign/foreign.h` for details).

```
UserMapping *
GetUserMapping(Oid userid, Oid serverid);
```

This function returns a `UserMapping` object for the user mapping of the given role on the given server. (If there is no mapping for the specific user, it will return the mapping for `PUBLIC`, or throw error if there is none.) A `UserMapping` object contains properties of the user mapping (see `foreign/foreign.h` for details).

```
ForeignTable *
GetForeignTable(Oid relid);
```

This function returns a `ForeignTable` object for the foreign table with the given OID. A `ForeignTable` object contains properties of the foreign table (see `foreign/foreign.h` for details).

```
List *
GetForeignColumnOptions(Oid relid, AttrNumber attnum);
```

This function returns the per-column FDW options for the column with the given foreign table OID and attribute number, in the form of a list of `DefElem`. NIL is returned if the column has no options.

Some object types have name-based lookup functions in addition to the OID-based ones:

```
ForeignDataWrapper *
GetForeignDataWrapperByName(const char *name, bool missing_ok);
```

This function returns a `ForeignDataWrapper` object for the foreign-data wrapper with the given name. If the wrapper is not found, return NULL if missing_ok is true, otherwise raise an error.

```
ForeignServer *
GetForeignServerByName(const char *name, bool missing_ok);
```

This function returns a `ForeignServer` object for the foreign server with the given name. If the server is not found, return NULL if missing_ok is true, otherwise raise an error.

50.4. Foreign Data Wrapper Query Planning

The FDW callback functions `GetForeignRelSize`, `GetForeignPaths`, and `GetForeignPlan` must fit into the workings of the PostgreSQL planner. Here are some notes about what they must do.

The information in `root` and `baserel` can be used to reduce the amount of information that has to be fetched from the foreign table (and therefore reduce the cost). `baserel->baserestrictinfo` is particularly interesting, as it contains restriction quals (`WHERE` clauses) that should be used to filter the rows to be fetched. (The FDW itself is not required to enforce these quals, as the core executor can check them instead.) `baserel->reltargetlist` can be used to determine which columns need to be fetched; but note that it only lists columns that have to be emitted by the `ForeignScan` plan node, not columns that are used in qual evaluation but not output by the query.

Various private fields are available for the FDW planning functions to keep information in. Generally, whatever you store in FDW private fields should be palloc'd, so that it will be reclaimed at the end of planning.

`baserel->fdw_private` is a `void` pointer that is available for FDW planning functions to store information relevant to the particular foreign table. The core planner does not touch it except to initialize it to NULL when the `baserel` node is created. It is useful for passing information forward from `GetForeignRelSize` to `GetForeignPaths` and/or `GetForeignPaths` to `GetForeignPlan`, thereby avoiding recalculation.

`GetForeignPaths` can identify the meaning of different access paths by storing private information in the `fdw_private` field of `ForeignPath` nodes. `fdw_private` is declared as a `List` pointer, but could actually contain anything since the core planner does not touch it. However, best practice is to use a representation that's dumpable by `nodeToString`, for use with debugging support available in the backend.

GetForeignPlan can examine the fdw_private field of the selected ForeignPath node, and can generate fdw_exprs and fdw_private lists to be placed in the ForeignScan plan node, where they will be available at execution time. Both of these lists must be represented in a form that copyObject knows how to copy. The fdw_private list has no other restrictions and is not interpreted by the core backend in any way. The fdw_exprs list, if not NIL, is expected to contain expression trees that are intended to be executed at run time. These trees will undergo post-processing by the planner to make them fully executable.

In GetForeignPlan, generally the passed-in target list can be copied into the plan node as-is. The passed scan_clauses list contains the same clauses as baserel->baserestrictinfo, but may be re-ordered for better execution efficiency. In simple cases the FDW can just strip RestrictInfo nodes from the scan_clauses list (using extract_actual_clauses) and put all the clauses into the plan node's qual list, which means that all the clauses will be checked by the executor at run time. More complex FDWs may be able to check some of the clauses internally, in which case those clauses can be removed from the plan node's qual list so that the executor doesn't waste time rechecking them.

As an example, the FDW might identify some restriction clauses of the form *foreign_variable* = *sub_expression*, which it determines can be executed on the remote server given the locally-evaluated value of the *sub_expression*. The actual identification of such a clause should happen during GetForeignPaths, since it would affect the cost estimate for the path. The path's fdw_private field would probably include a pointer to the identified clause's RestrictInfo node. Then GetForeignPlan would remove that clause from scan_clauses, but add the *sub_expression* to fdw_exprs to ensure that it gets massaged into executable form. It would probably also put control information into the plan node's fdw_private field to tell the execution functions what to do at run time. The query transmitted to the remote server would involve something like WHERE *foreign_variable* = $1, with the parameter value obtained at run time from evaluation of the fdw_exprs expression tree.

The FDW should always construct at least one path that depends only on the table's restriction clauses. In join queries, it might also choose to construct path(s) that depend on join clauses, for example *foreign_variable* = *local_variable*. Such clauses will not be found in baserel->baserestrictinfo but must be sought in the relation's join lists. A path using such a clause is called a "parameterized path". It must identify the other relations used in the selected join clause(s) with a suitable value of param_info; use get_baserel_parampathinfo to compute that value. In GetForeignPlan, the *local_variable* portion of the join clause would be added to fdw_exprs, and then at run time the case works the same as for an ordinary restriction clause.

第 51 章 遗传查询优化器

Martin Utesch, University of Mining and Technology

作者

由 Martin Utesch <utesch@aut.tu-freiberg.de> 为德国弗赖贝格矿业和技术大学自动控制研究所编写。

51.1. 将查询处理看成是一个复杂的优化问题

在所有关系操作符中，最难处理和优化的是连接。随着查询中连接数目的增加，可能的查询计划数量会呈指数增长。为了处理单个连接而支持多种连接方法（例如 PostgreSQL 中的嵌套循环、哈希连接和归并连接），以及作为关系访问路径的多种索引（例如 PostgreSQL 中的 B-树、哈希、GiST 和 GIN），也进一步增加了优化工作量。

常规的PostgreSQL查询优化器会在可选策略空间中执行近似穷举搜索。该算法最早出现在 IBM 的 System R 数据库中，能够产生近乎最优的连接顺序；但当查询中的连接数变得很大时，它可能耗费极大的时间和内存空间。因此，普通的 PostgreSQL查询优化器并不适合需要连接大量表的查询。

德国弗赖贝格矿业和技术大学自动控制研究所在尝试将 PostgreSQL 用作一个用于电网维护的基于知识的决策支持系统后端时遇到了一些问题。该 DBMS 需要为该基于知识系统中的推理机处理大型连接查询。这些查询中的连接数量使得使用普通查询优化器变得不可行。

下文将介绍一种遗传算法的实现，它以适合涉及大量连接的查询的高效方式来解决连接顺序问题。

51.2. 遗传算法

遗传算法（GA）是一种通过随机化搜索进行工作的启发式优化方法。优化问题的可能解集合被视为由若干个体组成的种群。个体对其环境的适应程度由其适应度表示。

一个个体在搜索空间中的坐标由染色体表示，本质上是一组字符串。基因是染色体的一个片段，它编码某个待优化参数的值。基因的典型编码可以是二进制或整数。

通过模拟重组、变异和选择这些进化操作，可以找到平均适应度高于前代的新一代搜索点。

根据comp.ai.genetic FAQ 中的说法，再怎么强调也不过分：GA并不是为了求解问题而进行的纯粹随机搜索。GA会使用随机过程，但其结果显然并非随机的（优于随机）。

图 51.1. 遗传算法的结构图

P(t)	时刻 t 的祖代
P''(t)	时刻 t 的子代

```
+=========================================+
|>>>>>>>>>>>  Algorithm GA  <<<<<<<<<<<<<<|
+=========================================+
| INITIALIZE t := 0                       |
+=========================================+
| INITIALIZE P(t)                         |
+=========================================+
| evaluate FITNESS of P(t)                |
+=========================================+
| while not STOPPING CRITERION do         |
|   +-------------------------------------+
|   | P'(t)  := RECOMBINATION{P(t)}       |
|   +-------------------------------------+
|   | P''(t) := MUTATION{P'(t)}           |
|   +-------------------------------------+
|   | P(t+1) := SELECTION{P''(t) + P(t)}  |
|   +-------------------------------------+
|   | evaluate FITNESS of P''(t)          |
|   +-------------------------------------+
|   | t := t + 1                          |
+===+=====================================+
```

51.3. PostgreSQL 中的遗传查询优化（GEQO）

GEQO模块把查询优化问题视为著名的旅行商问题（TSP）来处理。
可能的查询计划被编码为整数串。每个串表示查询中从一个关系到下一个关系的连接顺序。
例如，连接树

```
   /\
  /\ 2
 /\ 3
4  1
```

会被编码为整数串 '4-1-3-2'，这意味着先连接关系 '4' 和 '1'，再连接 '3'，最后连接 '2'；其中1、2、3、4 是PostgreSQL优化器内部的关系 ID。

PostgreSQL 中 GEQO 实现的具体特征包括：

- 采用稳态 GA（替换种群中适应度最低的个体，而不是整代替换），能够快速收敛到更优的查询计划。这对于在合理时间内处理查询至关重要；
- 采用边重组交叉，它特别适合在利用 GA 求解 TSP 时将边损失保持在较低水平；
- 不使用变异作为遗传操作符，因此无须借助修复机制来生成合法的 TSP 回路。

GEQO模块的部分内容改编自 D. Whitley 的 Genitor 算法。

GEQO模块使PostgreSQL查询优化器能够通过非穷举搜索有效支持大型连接查询。

51.3.1. 使用GEQO生成可能的计划

GEQO 规划过程使用标准规划器代码来生成各个关系扫描的计划，然后再用遗传方法生成连接计划。如上所示，每个候选连接计划都表示为一个连接基本关系的顺序。在初始阶段，GEQO代码只是随机生成一些可能的连接序列。对于每一个待考察的连接序列，
都会调用标准规划器代码来估算按该序列执行查询的代价。（对于连接序列中的每一步，

都会考虑全部三种可能的连接策略；并且所有预先确定的关系扫描计划也都可用。
估计代价取这些可能性中最低者。）
估计代价较低的连接序列会被认为比代价较高的序列“更适合”。
遗传算法会丢弃最不适合的候选，然后通过组合更适合候选的基因来生成新的候选，
也就是从已知低代价连接序列中随机选取片段，构造新的序列以供考察。
这个过程会重复进行，直到已经考察的连接序列数量达到预设值；然后，
在搜索过程中任何时刻找到的最佳序列都会被用来生成最终计划。

由于初始种群选择以及后续对最佳候选的“变异”都包含随机选择，
因此这一过程天生就是非确定性的。为了避免所选计划发生令人意外的变化，每次运行 GEQO 算法时，都会用当前的geqo_seed参数设置重新初始化其随机数生成器。只要 `geqo_seed`、其他 GEQO 参数以及统计信息等其他规划器输入保持不变，对于给定查询就会生成相同的计划。若想试验不同的搜索路径，可以尝试修改 `geqo_seed`。

51.3.2. PostgreSQL GEQO的未来实现任务

为了改进遗传算法的参数设置，仍有一些工作要做。在文件`src/backend/optimizer/geqo/geqo_main.c`中的例程`gimme_pool_size`和`gimme_number_generations`里，我们必须为参数设置找到一种折中，以满足两个相互竞争的需求：

- 查询计划的最优性
- 计算时间

在当前实现中，
每个候选连接序列的适应度都是通过从头运行标准规划器的连接选择和代价估算代码来估算的。只要不同候选使用了相似的连接子序列，就会有大量工作被重复执行。
如果能保留子连接的代价估算，就可以显著加快这一过程。问题在于，
必须避免为了保存这种状态而消耗不合理数量的内存。

从更基础的层面看，用一个为 TSP 设计的 GA 算法来解决查询优化问题是否合适，也并不明确。在 TSP 情况下，与任何子串（部分巡回）相关的代价都独立于巡回的其余部分，但对于查询优化显然并非如此。因此，边重组交叉是否是最有效的变异过程，仍然值得怀疑。

51.4. 进一步阅读

下列资源包含关于遗传算法的更多信息：

- 《Hitch-Hiker 演化计算指南》[1]（comp.ai.genetic[2] FAQ）
- 《演化计算及其在艺术和设计中的应用》[3]，Craig Reynolds 著
- [elma04]
- [fong]

[1] http://www.aip.de/~ast/EvolCompFAQ/
[2] news://comp.ai.genetic
[3] http://www.red3d.com/cwr/evolve.html

第 52 章 索引访问方法接口定义

本章定义 PostgreSQL 核心系统与管理各个索引类型的索引访问方法之间的接口。除这里规定的内容之外，核心系统对索引一无所知，
因此可以通过编写附加代码来开发全新的索引类型。

PostgreSQL 中所有索引在技术上都称为二级索引；也就是说，索引与它所描述的表文件在物理上是分离的。每个索引都存储为独立的物理关系，因此在 pg_class 目录中都有一个条目。索引的内容完全由其索引访问方法控制。实践中，
所有索引访问方法都会把索引划分为标准大小的页面，
以便使用常规的存储管理器和缓冲区管理器访问索引内容。
（现有的所有索引访问方法还都使用第 56.6 节中描述的标准页面布局，而且它们都对索引元组头使用相同的格式；但这些决定并不是访问方法必须遵循的。）

索引本质上是从一些数据键值到行版本（元组）在索引父表中的元组标识符（TIDs）
的映射。一个 TID 由块号以及该块中的项号组成（见第 56.6 节）。这些信息足以从表中取出某个特定的行版本。索引并不直接知道在 MVCC 之下同一逻辑行可能会存在多个现存版本；
对索引来说，每个元组都是一个独立对象，需要它自己的索引条目。因此，
对一行的更新总会为该行创建全新的索引条目，即使键值并未改变也是如此。（HOT 元组是这一说法的例外；但索引同样不直接处理它们。）当死元组自身被回收时（通过清理），
它们对应的索引条目也会被回收。

52.1. 索引的目录条目

每种索引访问方法都由系统目录 pg_am 中的一行描述（参见第 45.3 节）。pg_am 行的主要内容是一些指向 pg_proc 条目的引用，它们标识了该访问方法所提供的索引访问函数。这些函数的 API 将在本章后面定义。此外，pg_am 行还指定了访问方法的一些固定属性，例如它是否支持多列索引。目前没有任何专门用于创建或删除 pg_am 条目的支持；能够编写新访问方法的人，理应有能力自己插入合适的行。

要使一种索引访问方法真正可用，还必须在 pg_opfamily、pg_opclass、pg_amop 和 pg_amproc 中定义一个或多个操作符族和操作符类。这些条目使规划器能够确定该访问方法的索引可以使用哪些查询限定条件。关于操作符族和操作符类的说明见第 35.14 节；阅读本章前应先掌握这些内容。

单个索引由两个系统目录条目共同定义：一个 pg_class 条目把它描述为一个物理关系，另一个 pg_index 条目给出索引的逻辑内容，也就是它包含哪些索引列，以及由相关操作符类定义的这些列的语义。索引列（键值）既可以是底层表的简单列，
也可以是基于表行的表达式。
索引访问方法通常并不关心索引键值来自何处（它拿到的总是预先计算好的键值），
但会非常关心 pg_index 中的操作符类信息。这两个目录条目都可以作为传递给索引上所有操作的 Relation 数据结构的一部分来访问。

pg_am 的某些标志列还有一些不那么直观的含义。amcanunique 的要求见第 52.5 节。amcanmulticol 标志表明该访问方法支持多列索引，而 amoptionalkey 表明它允许在第一索引列上没有给出可索引限制子句时进行扫描。当 amcanmulticol 为假时，amoptionalkey 实质上表示该访问方法是否支持没有任何限制子句的全索引扫描。支持多个索引列的访问方法必须支持在第一列之后省略任意一个或全部列限制条件的扫描；
但它们可以要求第一索引列必须出现某种限制条件，这通过把 amoptionalkey 设为假来表示。索引 AM 可能把 amoptionalkey 设为假的一个原因，是它不索引 NULL 值。由于大多数可索引操作符都是严格的（strict），因此对 NULL 输入不可能返回 TRUE，所以乍看之下不存储 NULL 值的索引条目似乎很有吸引力：无论如何，这些条目似乎都不可能被索引扫描返回。然而，当某个索引扫描对给定索引列没有限制子句时，这个论证就不成立了。
实践中这意味着，若索引把 amoptionalkey 设为真，就必须索引 NULL 值，因为规划器可能决定在完全没有扫描键的情况下使用这种索引。与此相关的另一个限制是，
支持多个索引列的索引访问方法必须支持对第一列之后各列中的 NULL 值建立索引，因为规

划器会假定该索引可用于不限制这些列的查询。例如，考虑一个在 (a,b) 上的索引以及查询 `WHERE a = 4`。系统会假定该索引可用于扫描满足 `a = 4` 的行；如果索引省略了 `b` 为 NULL 的行，这个假定就是错误的。不过，省略第一索引列为 NULL 的行是可以的。对 NULL 值建立索引的索引访问方法还可以设置 `amsearchnulls`，表示它支持把 `IS NULL` 和 `IS NOT NULL` 子句作为搜索条件。

52.2. 索引访问方法函数

索引访问方法必须提供如下索引构建和维护函数：

```
IndexBuildResult *
ambuild (Relation heapRelation,
         Relation indexRelation,
         IndexInfo *indexInfo);
```

构建新索引。索引关系已在物理上创建，但内容为空。必须填入访问方法所需的固定数据，以及表中所有已有元组对应的条目。通常，`ambuild`函数会调用`IndexBuildHeapScan()`扫描表中现有的元组，并计算需要插入索引的键。函数必须返回一个通过 palloc 分配的结构体，其中包含新索引的统计信息。

```
void
ambuildempty (Relation indexRelation);
```

构建一个空索引，并将其写入给定关系的初始化分支（INIT_FORKNUM）。只有不记录 WAL 的表才会调用此方法；
写入初始化分支的空索引会在每次服务器重启时复制到主关系分支上。

```
bool
aminsert (Relation indexRelation,
          Datum *values,
          bool *isnull,
          ItemPointer heap_tid,
          Relation heapRelation,
          IndexUniqueCheck checkUnique);
```

将一个新元组插入已有索引。`values`和`isnull`数组给出要索引的键值，`heap_tid`则是要索引的 TID。如果访问方法支持唯一索引（其`pg_am.amcanunique`标志为 true），则`checkUnique`指明要执行的唯一性检查类型，具体取决于唯一约束是否可延迟；
详情参见第 52.5 节。通常，访问方法只在执行唯一性检查时才需要`heapRelation`参数（因为此时必须查看堆以确认元组是否存活）。

只有当`checkUnique`为`UNIQUE_CHECK_PARTIAL`时，该函数的布尔返回值才有意义。在这种情况下，TRUE 表示已知新条目唯一，而 FALSE 表示它可能不唯一（并且必须安排一次延迟唯一性检查）。对于其他情况，建议始终返回 FALSE。

某些索引可能不会为所有元组建立索引。如果某个元组不应被索引，`aminsert` 应当直接返回而不做任何事。

```
IndexBulkDeleteResult *
ambulkdelete (IndexVacuumInfo *info,
              IndexBulkDeleteResult *stats,
              IndexBulkDeleteCallback callback,
              void *callback_state);
```

从索引中删除元组。这是一个 “批量删除” 操作，
旨在通过扫描整个索引并检查每个条目是否应被删除来实现。必须调用传入的 `callback` 函数，其调用形式为 `callback(TID, callback_state) returns bool`，以确定由其引用的 TID 所标识的某个索引条目是否应删除。该函数必须返回 NULL，或者返

回一个通过 palloc 分配的结构体，其中包含此次删除操作影响的统计信息。如果不需要向 amvacuumcleanup 传递信息，返回 NULL 也是可以的。

由于 maintenance_work_mem 有限，当待删除的元组很多时，ambulkdelete 可能需要被调用多次。参数 stats 是此前对该索引上一次调用的结果（在一次 VACUUM 操作中的第一次调用时它为 NULL）。这使得 AM 可以在整个操作过程中累积统计信息。通常，如果传入的 stats 非 NULL，ambulkdelete 会修改并返回同一个结构体。

```
IndexBulkDeleteResult *
amvacuumcleanup (IndexVacuumInfo *info,
                 IndexBulkDeleteResult *stats);
```

在一次 VACUUM 操作（零次或多次 ambulkdelete 调用）之后执行清理。它不一定要做返回索引统计信息之外的事情，但也可能执行批量清理，例如回收空索引页面。stats 是最后一次 ambulkdelete 调用返回的结果；如果因为没有需要删除的元组而没有调用 ambulkdelete，则为 NULL。如果结果不是 NULL，它必须是一个通过 palloc 分配的结构体。其中的统计信息将用于更新 pg_class，并在指定了 VERBOSE 时由 VACUUM 报告。如果索引在整个 VACUUM 操作期间完全没有变化，返回 NULL 也是可以的；否则应返回正确的统计信息。

从PostgreSQL 8.4 开始，amvacuumcleanup 也会在 ANALYZE 操作完成时被调用。在这种情况下，stats 总是 NULL，而且任何返回值都会被忽略。这种情况可以通过检查 info->analyze_only 来区分。我们建议访问方法在这种调用中除了做插入后的清理外什么都不做，而且只在自动清理工作进程中这样做。

```
bool
amcanreturn (Relation indexRelation, int attno);
```

检查索引能否通过以 IndexTuple 形式返回某个索引条目的被索引列值，在给定列上支持仅索引扫描。属性编号从 1 开始，也就是说第一列的 attno 为 1。若支持则返回 TRUE，否则返回 FALSE。如果访问方法根本不支持仅索引扫描，那么pg_am行中的amcanreturn字段可以设为零。

```
void
amcostestimate (PlannerInfo *root,
                IndexPath *path,
                double loop_count,
                Cost *indexStartupCost,
                Cost *indexTotalCost,
                Selectivity *indexSelectivity,
                double *indexCorrelation,
                double *indexPages);
```

估计一次索引扫描的代价。该函数将在后面的第 52.6 节中详细讨论。

```
bytea *
amoptions (ArrayType *reloptions,
           bool validate);
```

解析并验证索引的 reloptions 数组。只有当该索引存在非 NULL 的 reloptions 数组时才会调用此函数。*reloptions* 是一个 text 数组，其中的条目形如 *name=value*。该函数应构造一个 bytea 值，并将其复制到索引 relcache 条目的 rd_options 字段中。这个 bytea 值中的数据内容由访问方法自行定义；大多数标准访问方法使用结构体 StdRdOptions。当 *validate* 为真时，若存在未识别选项或无效取值，函数应报告合适的错误消息；当 *validate* 为假时，无效条目应被静默忽略。（当装载已存储在 pg_catalog 中的选项时，*validate* 为假；此时只有在访问方法改变了选项规则时才可能发现无效条目，而忽略过时条目是合适的。）如果希望采用默认行为，返回 NULL 也是可以的。

索引的目的当然是支持扫描那些匹配可索引 WHERE 条件的元组，这种条件常被称为限定词或扫描键。关于索引扫描的语义，将在下面的第 52.3 节中更详细地说明。一种索引访问方

法可以支持“普通”索引扫描、“位图”索引扫描，或者两者都支持。
索引访问方法必须或可以提供的扫描相关函数如下：

```
IndexScanDesc
ambeginscan (Relation indexRelation,
             int nkeys,
             int norderbys);
```

为一次索引扫描做准备。nkeys 和 norderbys 参数表示扫描中将使用的限定条件和排序操作符数量，这些信息可能有助于空间分配。请注意，此时还没有提供扫描键的实际值。
结果必须是一个通过 palloc 分配的结构体。出于实现上的原因，索引访问方法必须通过调用 RelationGetIndexScan() 来创建这个结构体。大多数情况下，ambeginscan 除了做这次调用以及也许获取一些锁之外，不会做太多工作；索引扫描启动中真正有意思的部分在 amrescan 中。

```
void
amrescan (IndexScanDesc scan,
          ScanKey keys,
          int nkeys,
          ScanKey orderbys,
          int norderbys);
```

开始或重新开始一次索引扫描，并且可以使用新的扫描键。
（若要用之前传入的键重新开始，则为 keys 和/或 orderbys 传递 NULL。）请注意，键的数量或排序操作符的数量都不允许超过传给 ambeginscan 的数量。实践中，这种重启特性通常用于嵌套循环连接选中了新的外层元组，因此需要新的键比较值，
但扫描键结构体本身保持不变的场景。

```
boolean
amgettuple (IndexScanDesc scan,
            ScanDirection direction);
```

沿给定方向（在索引中向前或向后）取出给定扫描中的下一个元组。若取得元组则返回 TRUE，若没有剩余匹配元组则返回 FALSE。返回 TRUE 时，元组 TID 存入scan结构体中。注意，“成功”仅表示索引中包含与扫描键匹配的条目，并不表示该元组一定仍存在于堆中，
或者能够通过调用方的快照测试。成功时，amgettuple还必须将scan->xs_recheck设为 TRUE 或 FALSE。FALSE 表示可以确定该索引条目匹配扫描键。TRUE 表示无法确定，因此在取出堆元组后必须重新检查扫描键所代表的条件。这一规定支持“有损”索引操作符。注意，重新检查仅针对扫描条件；部分索引谓词（如果有）永远不会由amgettuple
的调用方重新检查。

如果索引支持仅索引扫描（即 amcanreturn 对它返回 TRUE），那么成功时 AM 还必须检查 scan->xs_want_itup；若其为真，就必须以如下形式返回该索引条目的原始被索引数据：把 IndexTuple 指针存入 scan->xs_itup，其元组描述符为 scan->xs_itupdesc。（指针所引用数据的管理责任在访问方法一侧。这份数据必须至少保持有效，直到该扫描下一次调用 amgettuple、amrescan 或 amendscan。）

只有当访问方法支持“普通”索引扫描时，才需要提供 amgettuple 函数。如果不支持，其 pg_am 行中的 amgettuple 字段必须设为零。

```
int64
amgetbitmap (IndexScanDesc scan,
             TIDBitmap *tbm);
```

取出给定扫描中的所有元组，并将其加入调用者提供的 TIDBitmap 中（也就是把这组元组 ID 与位图中已有的集合做 OR）。返回值是取得的元组数量（这可能只是近似计数，例如某些 AM 不会检测重复项）。在把元组 ID 插入位图时，amgetbitmap 可以指出某些具体的元组 ID 需要重新检查扫描条件。这类似于 amgettuple 的 xs_recheck 输出参数。注意：在当前实现中，对这一特性的支持与位图自身的有损存储支持混在一起，

因此调用者会对需要重检的元组同时重新检查扫描条件和部分索引谓词（如果有）。不过，这并不一定永远如此。amgetbitmap 和 amgettuple 不能在同一次索引扫描中同时使用；使用 amgetbitmap 时还有其他限制，详见第 52.3 节。

只有当访问方法支持“位图”索引扫描时，才需要提供 amgetbitmap 函数。如果不支持，其 pg_am 行中的 amgetbitmap 字段必须设为零。

```
void
amendscan (IndexScanDesc scan);
```

结束一次扫描并释放资源。scan 结构体本身不应被释放，但必须释放访问方法内部获取的所有锁、解除所有钉住状态。

```
void
ammarkpos (IndexScanDesc scan);
```

标记当前扫描位置。访问方法只需要为每次扫描支持一个被记住的扫描位置。

```
void
amrestrpos (IndexScanDesc scan);
```

将扫描恢复到最近一次标记的位置。

按照惯例，索引访问方法函数的 pg_proc 条目应显示正确的参数个数，但把所有参数都声明为 internal 类型（因为大多数参数的类型对 SQL 而言是未知的，而且我们也不希望用户直接调用这些函数）。返回类型根据实际情况声明为 void、internal 或 boolean。唯一的例外是 amoptions，它应当正确地声明为接受 text[] 和 bool 并返回 bytea。这一规定允许客户端代码执行 amoptions 来测试选项设置的有效性。

52.3. 索引扫描

在索引扫描中，索引访问方法负责返回它所知道的、匹配扫描键的所有元组的 TID。访问方法不负责从索引父表中实际取出这些元组，
也不负责判断它们是否能通过扫描的可见性测试或其他条件。

扫描键是如下形式的 WHERE 子句的内部表示：*index_key operator constant*。其中，索引键是该索引的一列，而操作符是与该索引列关联的操作符族成员之一。
一次索引扫描可以有零个或多个扫描键，这些扫描键会被隐式地以 AND 连接，也就是说，返回的元组预期应满足所有给定条件。

对于某个特定查询，访问方法可以报告该索引是有损的，或者要求重新检查。
这意味着索引扫描会返回所有通过扫描键的条目，
并可能额外返回一些未通过扫描键的条目。随后，
核心系统的索引扫描机制会再次把索引条件应用到堆元组上，以验证它是否真的应被选中。
如果没有指定重检选项，索引扫描就必须精确返回匹配条目的集合。

请注意，确保正确找出通过全部给定扫描键的所有且仅有的条目，
这项工作完全由访问方法负责。此外，
核心系统只是简单地把所有与索引键和操作符族匹配的 WHERE 子句交给访问方法，而不会做任何语义分析来判断它们是否冗余或互相矛盾。例如，给定 WHERE x > 4 AND x > 14，其中 x 是一个 B-树索引列，那么由 B-树的 amrescan 函数负责识别第一个扫描键是冗余的并可被丢弃。amrescan 期间需要做多少预处理，取决于索引访问方法需要在多大程度上把扫描键归约成一种“正规化”形式。

某些访问方法会按明确定义的顺序返回索引条目，另一些则不会。实际上，
访问方法支持有序输出有两种不同方式：

- 总是按数据自然顺序返回条目的访问方法（例如 btree）应把 pg_am.amcanorder 设为真。目前，这类访问方法必须对其等值和排序操作符使用与 btree 兼容的策略号。

- 支持排序操作符的访问方法应把 pg_am.amcanorderbyop 设为真。这表示索引能够按满足 ORDER BY *index_key operator constant* 的顺序返回条目。前面已经提到，这种形式的扫描修饰符可以传给 amrescan。

amgettuple 函数有一个 direction 参数，它可以是 ForwardScanDirection（通常情况）或 BackwardScanDirection。如果 amrescan 之后的第一次调用指定了 BackwardScanDirection，那么匹配条件的索引条目集合就要按从后到前的方向扫描，而不是通常的从前到后，因此 amgettuple 必须返回索引中最后一个匹配元组，而不是通常情况下的第一个。（这只会发生在把 amcanorder 设为真的访问方法上。）第一次调用之后，amgettuple 必须准备好从最近一次返回条目的位置继续按任一方向推进扫描。（但如果 pg_am.amcanbackward 为假，则后续所有调用的方向都必须与第一次相同。）

支持有序扫描的访问方法必须支持在扫描中“标记”一个位置，并在之后返回到该标记位置。同一个位置可能会被恢复多次。不过，每次扫描只需要记住一个位置；新的 ammarkpos 调用会覆盖先前标记的位置。不支持有序扫描的访问方法仍应在 pg_am 中提供标记和恢复函数，但让这些函数在被调用时报错也是可以的。

无论是扫描位置还是标记位置（如果有），
在面对索引中的并发插入或删除时都必须保持一致。
如果某个新插入的条目没有被一次扫描返回，
而假如该条目在扫描开始前就已存在则本应被返回，这也是可以接受的；同样，
即使某个条目在第一次扫描通过时未被返回，
后续重新扫描或反向扫描时返回它也是可以接受的。类似地，
并发删除也可能会或者不会反映在扫描结果中。重要的是，
插入或删除本身不能导致扫描漏掉，或者重复返回那些本身并未被插入或删除的条目。

如果索引存储的是原始被索引数据值（而不是它们的某种有损表示），
那么它就适合支持仅索引扫描，在这种扫描中索引返回的是实际数据，而不只是堆元组的 TID。只有当可见性映射显示该 TID 位于一个全部可见的页面上时，这才可行；否则仍然必须访问堆元组来检查 MVCC 可见性。不过，这并不是访问方法需要关心的事。

除了使用 amgettuple 外，索引扫描也可以使用 amgetbitmap，在一次调用中取出所有元组。与 amgettuple 相比，这可能明显更高效，因为它可以避免访问方法内部的反复加锁和解锁。原则上，amgetbitmap 应具有与重复调用 amgettuple 相同的效果，但为了简化实现，我们施加了若干限制。首先，amgetbitmap 一次性返回全部元组，因此不支持标记或恢复扫描位置。其次，元组是通过一个位图返回的，没有特定顺序，这也是为什么 amgetbitmap 不带 direction 参数的原因。（这种扫描也永远不会提供排序操作符。）此外，使用 amgetbitmap 时也没有仅索引扫描的支持，因为没有办法返回索引元组的内容。最后，amgetbitmap 不保证对返回的元组做任何锁定，其影响见第 52.4 节。

请注意，如果访问方法的内部实现并不适合某一个 API，那么它可以只实现 amgetbitmap 而不实现 amgettuple，反之亦然。

52.4. 索引锁定注意事项

索引访问方法必须处理多个进程并发更新同一个索引的情况。PostgreSQL 核心系统会在索引扫描期间对索引获取 AccessShareLock，并在更新索引时（包括普通的 VACUUM）获取 RowExclusiveLock。由于这些锁类型彼此不冲突，所以访问方法必须自行处理它可能需要的任何细粒度锁定。对整个索引的排他锁只会在索引创建、销毁或 REINDEX 期间获取。

构建一种支持并发更新的索引类型，通常需要对所需行为进行大量而细致的分析。对于 B-树和 hash 索引类型，可以阅读 src/backend/access/nbtree/README 和 src/backend/access/hash/README 中涉及的设计决策。

除了索引自身内部一致性的要求外，并发更新还会带来父表（即堆）
与索引之间一致性的问题。由于 PostgreSQL 把对堆的访问和更新与对索引的访问和更新分离开来，因此存在一些窗口期，在这些窗口期内索引可能与堆不一致。
我们通过以下规则处理这个问题：

- 先创建新的堆条目，再创建它对应的索引条目。
  （因此并发索引扫描很可能看不到这个堆条目。这是可以接受的，
  因为索引读取者本来也不会关心未提交的行。但也请见第 52.5 节。）

- 当某个堆条目将要被删除（由 VACUUM 执行）时，必须先删除它的所有索引条目。

- 索引扫描必须在保存 amgettuple 最近一次返回条目的索引页面上保持钉住状态，而 ambulkdelete 不能从被其他后端钉住的页面中删除条目。下面会解释为什么需要这条规则。

如果没有第三条规则，索引读取者就有可能在某个索引条目被 VACUUM 删除之前先看到它，然后在对应的堆条目已被 VACUUM 删除之后才到达那里。如果读取者到达时，该项号仍未被重新使用，就不会造成严重问题，因为空项槽位会被 heap_fetch() 忽略。但如果第三个后端已经把这个项槽位重新用作别的东西呢？在使用 MVCC 兼容快照时不会有问题，因为该槽位的新占用者一定太新，无法通过快照测试。然而，对于非 MVCC 兼容的快照（例如 SnapshotNow），就可能错误地接受并返回一行实际上并不匹配扫描键的数据。我们可以要求在所有情况下都对堆行重新检查扫描键，以防范这种情形，但那样代价太高。于是，
我们改为把索引页面的钉住状态作为一种代理，
表示读取者从索引条目到匹配堆条目的访问过程可能仍“在进行中”。让 ambulkdelete 因这种钉住状态而阻塞，能够保证 VACUUM 不会在读取者处理完之前删除对应的堆条目。这种方案运行时开销很小，只有在真正发生冲突的少数情况下才会增加阻塞开销。

这种解决方案要求索引扫描是“同步的”：我们必须在扫描到相应索引条目之后立刻取出对应的堆元组。由于多种原因，这会比较昂贵。相反，“异步”扫描可以先从索引中收集许多 TID，再在稍后某个时刻访问堆元组，这样索引锁定的开销要小得多，
也能实现更高效的堆访问模式。根据前面的分析，对于非 MVCC 兼容快照，我们必须采用同步方式；而对于使用 MVCC 快照的查询，异步扫描则是可行的。

在 amgetbitmap 索引扫描中，访问方法不会为任何返回的元组保持索引页面的钉住状态。因此，只有把这种扫描与 MVCC 兼容的快照一起使用才是安全的。

当未设置 ampredlocks 标志时，在可串行化事务中使用该索引访问方法的任何扫描，都会在整个索引上获取一个非阻塞谓词锁。
这会与并发可串行化事务向该索引插入任何元组形成读写冲突。
如果在一组并发可串行化事务之间检测到某些特定模式的读写冲突，为了保护数据完整性，
其中某个事务可能会被取消。设置该标志则表示该索引访问方法实现了更细粒度的谓词锁，
这通常会降低这类事务取消的频率。

52.5. 索引唯一性检查

PostgreSQL 使用唯一索引来强制执行 SQL 唯一性约束；所谓唯一索引，就是不允许存在多个具有相同键值条目的索引。支持这一特性的访问方法会把 pg_am.amcanunique 设为真。（目前只有 B-树支持这一点。）

由于 MVCC 的存在，索引中在物理上总是必须允许存在重复条目：这些条目可能指向同一个逻辑行的连续版本。我们真正想强制的行为是，任何 MVCC 快照都不能同时包含两行拥有相同索引键的记录。在向唯一索引插入一行新数据时，必须检查以下几种情况：

- 如果某一条冲突的有效行已被当前事务删除，那么这是允许的。（特别是，因为一次 UPDATE 总是在插入新版本前删除旧版本，这就允许对某一行执行不改变键值的 UPDATE。）

- 如果某条冲突的行是由一个尚未提交的事务插入的，那么当前准备插入的事务必须等待，
  看看那个事务是否提交。如果它回滚，就不存在冲突；
  如果它提交并且没有再次删除那条冲突行，就发生了唯一性违背。（实际上，
  我们只是等待另一个事务结束，然后把可见性检查整个重新做一遍。）

- 类似地，如果某条冲突的有效行是由一个尚未提交的事务删除的，
  那么当前准备插入的事务必须等待该事务提交或中止，然后重新执行测试。

此外，就在按照上述规则报告唯一性违背之前，
访问方法必须重新检查正在插入那一行的存活性。如果它已经是提交后死亡的状态，
就不应报告违背。（这种情况不会出现在插入由当前事务刚创建的行这一普通场景中，但在 `CREATE UNIQUE INDEX CONCURRENTLY` 期间却可能发生。）

我们要求索引访问方法自行应用这些测试，
这意味着它必须深入堆中检查那些根据索引内容显示为具有重复键的行的提交状态。
毫无疑问，这样做既丑陋又不够模块化，
但它避免了重复工作：如果我们单独再做一次探测，那么在寻找新行索引条目插入位置时，
查找冲突行的索引搜索实际上就会被重复执行。更何况，
除非把冲突检查作为插入新索引条目动作的一个组成部分，
否则也没有明显的方法可以避免竞争条件。

如果唯一约束是可延迟的，情况会更复杂：我们需要能够为新行插入一个索引条目，
但把任何唯一性违背错误延迟到语句结束时甚至更晚才报告。
为了避免对索引进行不必要的重复搜索，
索引访问方法应在初始插入期间执行一次初步唯一性检查。
如果这表明确实不存在冲突的存活元组，那么事情就结束了。否则，
我们会安排在真正强制约束时再做一次重检。若在重检时，
插入的元组与另外某个具有相同键值的元组都仍然存活，就必须报告错误。（注意，
就此用途而言，“存活”实际上是指“索引条目 HOT 链中至少有一个元组是存活的”。）为实现这一点，传给 `aminsert` 函数的 `checkUnique` 参数会取以下值之一：

- `UNIQUE_CHECK_NO` 表示不应执行唯一性检查（这不是唯一索引）。

- `UNIQUE_CHECK_YES` 表示这是一个不可延迟的唯一索引，必须按前述方式立即执行唯一性检查。

- `UNIQUE_CHECK_PARTIAL` 表示该唯一约束是可延迟的。PostgreSQL 会使用此模式为每一行插入索引条目。访问方法必须允许索引中出现重复条目，并通过让 `aminsert` 返回 FALSE来报告任何潜在的重复。对于每一行返回 FALSE的情况，系统都会安排一次延迟重检。

  访问方法必须识别任何可能违反唯一约束的行，但报告误报并不算错误。
  这使得检查可以在不等待其他事务结束的情况下完成；
  此处报告的冲突不会被当作错误处理，而会在之后重新检查，
  到那时它们可能已不再冲突。

- `UNIQUE_CHECK_EXISTING` 表示这是对某一行的延迟重检，此前该行被报告为可能违反唯一性。虽然这同样是通过调用 `aminsert` 来实现，但访问方法在这种情况下不得插入新的索引条目。该索引条目已经存在。相反，
  访问方法必须检查是否存在另一个存活的索引条目；如果存在，
  并且目标行本身也仍然存活，就应报告错误。

  建议在一次 `UNIQUE_CHECK_EXISTING` 调用中，访问方法进一步验证目标行在索引中确实已有现存条目；若没有，就报告错误。这样做是个好主意，因为传给 `aminsert` 的索引元组值会被重新计算。如果索引定义涉及并非真正 immutable 的函数，我们就可能在索引的错误区域上做检查。确认重检时确实找到了目标行，
  可以验证我们扫描的仍是原始插入时使用的同一组元组值。

52.6. 索引代价估算函数

`amcostestimate` 函数会收到描述某种可能索引扫描方式的信息，其中包括已经确定可用于该索引的 WHERE 子句和 ORDER BY 子句列表。它必须返回访问该索引的代价估算，以及 WHERE 子句选择率的估计值（也就是在索引扫描期间将从父表中检索出的行所占比例）。对于简单情况，代价估算器几乎所有工作都可以通过调用优化器中的标准例程来完成；
之所以提供 `amcostestimate` 函数，是为了让索引访问方法能够提供与索引类型有关的专门知识，以便在可能时改进标准估计。

每个 amcostestimate 函数都必须具有如下签名：

```
void
amcostestimate (PlannerInfo *root,
                IndexPath *path,
                double loop_count,
                Cost *indexStartupCost,
                Cost *indexTotalCost,
                Selectivity *indexSelectivity,
                double *indexCorrelation);
```

前三个参数是输入参数：

root

规划器关于当前正在处理查询的信息。

path

当前正在考虑的索引访问路径。除代价和选择率字段外，其余字段都有效。

loop_count

在代价估算中应计入的索引扫描重复次数。
当考虑在嵌套循环连接内部使用参数化扫描时，这个参数通常会大于 1。请注意，代价估算仍应只针对一次扫描；更大的 *loop_count* 只表示可以适当考虑多次扫描之间的一些缓存效应。

最后四个参数是按引用传递的输出参数：

**indexStartupCost*

设为索引启动处理的代价。

**indexTotalCost*

设为索引处理的总代价。

**indexSelectivity*

设为索引选择率。

**indexCorrelation*

设为索引扫描顺序与底层表顺序之间的相关系数。

请注意，代价估算函数必须用 C 编写，而不能用 SQL 或任何可用的过程语言，因为它们必须访问规划器/优化器的内部数据结构。

索引访问代价应使用 src/backend/optimizer/path/costsize.c 所采用的参数来计算：顺序磁盘块读取的代价为 seq_page_cost，非顺序读取的代价为 random_page_cost，处理一条索引行的代价通常应取为 cpu_index_tuple_cost。此外，在索引处理期间调用的任何比较操作符（特别是对 indexquals 本身的求值）都应计入适当倍数的 cpu_operator_cost。

访问代价应包括与扫描索引本身有关的全部磁盘和 CPU 代价，但不包括取出或处理由索引标识出的父表行的代价。

“启动代价”是整个扫描总代价中必须在开始取第一行之前先付出的那一部分。
对大多数索引来说，这可以视为零；但启动代价较高的索引类型可能希望把它设为非零。

indexSelectivity 应设为在索引扫描期间将从父表中检索出的行的估计比例。对于有损查询，这个值通常会高于实际通过给定限定条件的行比例。

indexCorrelation 应设为索引顺序与表顺序之间的相关性（范围从 -1.0 到 1.0）。该值用于调整从父表取行代价的估计。

当 *loop_count* 大于 1 时，返回的数字应是该索引任意一次扫描的期望平均值。

代价估算

一个典型的代价估算器会按如下步骤进行：

1. 基于给定的限定条件，估计并返回将被访问的父表行比例。
   如果没有任何与索引类型相关的专门知识，可以使用优化器的标准函数 `clauselist_selectivity()`：

   ```
   *indexSelectivity = clauselist_selectivity(root, path-
   >indexquals,
                                              path->indexinfo-
   >rel->relid,
                                              JOIN_INNER, NULL);
   ```

2. 估计扫描期间将访问的索引行数。对许多索引类型来说，这等于 *indexSelectivity* 乘以索引中的行数，但也可能更多。（请注意，索引的页面数和行数可以从 `path->indexinfo` 结构体中取得。）

3. 估计扫描期间将读取的索引页面数。它可能仅仅是 *indexSelectivity* 乘以索引总页面数。

4. 计算索引访问代价。一个通用估计器可能会这样做：

   ```
   /*
    * 通用假设是索引页面将按顺序读取，
    * 因此每页代价为 seq_page_cost，而非 random_page_cost。
    * 此外，还要计入在每个索引行上对 indexquals 求值的代价。
    * 假定所有代价都在扫描过程中逐步付出。
    */
   cost_qual_eval(&index_qual_cost, path->indexquals, root);
   *indexStartupCost = index_qual_cost.startup;
   *indexTotalCost = seq_page_cost * numIndexPages +
       (cpu_index_tuple_cost + index_qual_cost.per_tuple) * num
   IndexTuples;
   ```

   不过，上述做法没有考虑重复索引扫描之间索引读取的摊销效果。

5. 估计索引的相关性。对于单列上的简单有序索引，这个值可以从 pg_statistic 中取得。如果相关性未知，保守估计应为零（即无相关性）。

代价估算器函数的示例可在 `src/backend/utils/adt/selfuncs.c` 中找到。

第 53 章 GiST 索引

53.1. 简介

GiST是 Generalized Search Tree（通用搜索树）的缩写。它是一种平衡的树形访问方法，可作为实现任意索引方案的基本模板。B-树、R 树以及许多其他索引方案都可以在GiST中实现。

GiST的一个优点是，它使数据类型所属领域的专家而不是数据库专家，能够开发带有适当访问方法的自定义数据类型。

这里的一些内容来自加州大学伯克利分校的 GiST 索引项目网站[1]以及 Marcel Kornacker 的论文 Access Methods for Next-Generation Database Systems[2]。PostgreSQL中的GiST 实现主要由 Teodor Sigaev 和 Oleg Bartunov 维护，他们的网站[3]上还有更多信息。

53.2. 可扩展性

传统上，实现一种新的索引访问方法意味着大量艰难的工作。必须理解数据库的内部机制，例如锁管理器和预写式日志。GiST接口具有很高的抽象层次，只要求访问方法实现者实现被访问数据类型的语义。GiST层本身会处理并发、日志记录以及树结构的搜索。

这种可扩展性不应与其他标准搜索树在可处理数据方面的可扩展性相混淆。例如，PostgreSQL支持可扩展的 B-树和 hash 索引。这意味着你可以用PostgreSQL在任意数据类型上构建 B-树或 hash 索引。但 B-树只支持范围谓词（<、=、>），而 hash 索引只支持等值查询。

因此，如果你用PostgreSQL的 B-树为一个图像集合建立索引，你只能发出诸如“imagex 是否等于 imagey”、“imagex 是否小于 imagey”以及“imagex 是否大于 imagey”之类的查询。取决于你如何在这种上下文中定义“等于”、“小于”和“大于”，这可能仍然有用。不过，使用基于GiST的索引，你就可以构造出能够提出特定领域问题的查询方式，例如“找出所有马的图片”或者“找出所有曝光过度的图片”。

要让一个GiST访问方法运行起来，只需实现几个用户定义的方法，这些方法定义了树中键的行为。当然，要支持复杂查询，这些方法本身也必须足够巧妙；但对于所有标准查询（B-树、R 树等），它们都相对直接。简而言之，GiST把可扩展性与通用性、代码复用以及清晰的接口结合了起来。

一个GiST索引操作符类必须提供七个方法，另外还有一个可选方法。通过正确实现`same`、`consistent`和`union`方法可以保证索引的正确性，而索引的效率（大小与速度）则取决于`penalty`和`picksplit`方法。另外两个基本方法是`compress`和`decompress`，它们允许索引的内部树数据使用与其所索引数据不同的类型。叶子必须是被索引数据类型，而其他树节点可以是任意 C 结构体（但这里仍必须遵守PostgreSQL的数据类型规则，关于变长数据可参见`varlena`）。如果树的内部数据类型在 SQL 层存在，可以使用`CREATE OPERATOR CLASS`命令的`STORAGE`选项。可选的第八个方法是`distance`，若操作符类希望支持有序扫描（最近邻搜索），则需要它。

`consistent`

> 给定一个索引项`p`和一个查询值`q`，该函数判断该索引项是否与该查询“一致”；也就是说，该索引项所代表的某一行是否可能使谓词“*indexed_column indexable_operator q*”为真。对于叶子索引项，这等同于测试该可索引条件；而对于内部树节点，这决定是否有必要扫描该树节点所表示的索引子树。当结果为`true`时，

[1] http://gist.cs.berkeley.edu/
[2] http://www.sai.msu.su/~megera/postgres/gist/papers/concurrency/access-methods-for-next-generation.pdf.gz
[3] http://www.sai.msu.su/~megera/postgres/gist/

还必须返回一个recheck标志。它表示该谓词是确定为真，还是仅可能为真。如果recheck = false，则该索引已经精确测试了谓词条件；如果recheck = true，则该行只是候选匹配。在这种情况下，系统会自动针对实际行值计算*indexable_operator*，以判断它是否真的匹配。这种约定使GiST能够同时支持无损和有损的索引结构。

该函数的SQL声明必须如下所示：

```
CREATE OR REPLACE FUNCTION my_consistent(internal, data_type,
 smallint, oid, internal)
RETURNS bool
AS 'MODULE_PATHNAME'
LANGUAGE C STRICT;
```

而 C 模块中的对应代码则可以遵循如下框架：

```
Datum       my_consistent(PG_FUNCTION_ARGS);
PG_FUNCTION_INFO_V1(my_consistent);

Datum
my_consistent(PG_FUNCTION_ARGS)
{
    GISTENTRY  *entry = (GISTENTRY *) PG_GETARG_POINTER(0);
    data_type  *query = PG_GETARG_DATA_TYPE_P(1);
    StrategyNumber strategy = (StrategyNumber) PG_GETARG_UINT16(
2);
    /* Oid subtype = PG_GETARG_OID(3); */
    bool       *recheck = (bool *) PG_GETARG_POINTER(4);
    data_type  *key = DatumGetDataType(entry->key);
    bool        retval;

    /*
     * 根据 strategy、key 和 query 确定返回值。
     *
     * 使用 GIST_LEAF(entry) 判断当前调用位于索引树的哪个位置。
     * 例如，支持 = 操作符时这很有用（可以在非叶节点检查
     * union() 是否非空，在叶节点检查是否相等）。
     */

    *recheck = true;        /* 如果检查是精确的，则为 false */

    PG_RETURN_BOOL(retval);
}
```

这里，key是索引中的一个元素，而query是在该索引中查找的值。StrategyNumber参数指示应用的是操作符类中的哪个操作符，它对应于CREATE OPERATOR CLASS命令中的某个操作符编号。取决于你在该类中包含了哪些操作符，query的数据类型可能会随操作符而变化，但上面的框架假设它不会变化。

union

该方法用于汇总树中的信息。给定一组项，该函数生成一个新的索引项，用来表示所有给定项。

该函数的SQL声明必须如下所示：

```
CREATE OR REPLACE FUNCTION my_union(internal, internal)
RETURNS internal
AS 'MODULE_PATHNAME'
LANGUAGE C STRICT;
```

而 C 模块中的对应代码则可以遵循如下框架：

```
Datum       my_union(PG_FUNCTION_ARGS);
PG_FUNCTION_INFO_V1(my_union);

Datum
my_union(PG_FUNCTION_ARGS)
{
    GistEntryVector *entryvec = (GistEntryVector *) PG_GETARG_
POINTER(0);
    GISTENTRY  *ent = entryvec->vector;
    data_type  *out,
               *tmp,
               *old;
    int         numranges,
                i = 0;

    numranges = entryvec->n;
    tmp = DatumGetDataType(ent[0].key);
    out = tmp;

    if (numranges == 1)
    {
        out = data_type_deep_copy(tmp);

        PG_RETURN_DATA_TYPE_P(out);
    }

    for (i = 1; i < numranges; i++)
    {
        old = out;
        tmp = DatumGetDataType(ent[i].key);
        out = my_union_implementation(out, tmp);
    }

    PG_RETURN_DATA_TYPE_P(out);
}
```

如你所见，在这个框架里，我们处理的是一种满足`union(X, Y, Z) = union(union(X, Y), Z)`的数据类型。对于不满足这一性质的数据类型，只需在这个GiST支持方法中实现正确的 union 算法即可。

`union`实现函数应返回一个指向新近通过`palloc()`分配的内存的指针。
不能原样返回输入值。

`compress`

将一个数据项转换成适合在索引页中物理存储的格式。

该函数的SQL声明必须如下所示：

```
CREATE OR REPLACE FUNCTION my_compress(internal)
RETURNS internal
AS 'MODULE_PATHNAME'
LANGUAGE C STRICT;
```

而 C 模块中的对应代码则可以遵循如下框架：

```
Datum       my_compress(PG_FUNCTION_ARGS);
```

```
PG_FUNCTION_INFO_V1(my_compress);

Datum
my_compress(PG_FUNCTION_ARGS)
{
    GISTENTRY  *entry = (GISTENTRY *) PG_GETARG_POINTER(0);
    GISTENTRY  *retval;

    if (entry->leafkey)
    {
        /* 将 entry->key 替换为压缩后的形式 */
        compressed_data_type *compressed_data = palloc(sizeof(
compressed_data_type));

        /* 根据 entry->key 填充 *compressed_data ... */

        retval = palloc(sizeof(GISTENTRY));
        gistentryinit(*retval, PointerGetDatum(compressed_data),
                      entry->rel, entry->page, entry->offset,
 FALSE);
    }
    else
    {
        /* 通常无需对非叶项做任何处理 */
        retval = entry;
    }

    PG_RETURN_POINTER(retval);
}
```

当然，为了压缩叶子节点，你必须把*compressed_data_type*改成要转换成的具体类型。

根据你的需要，你可能还需要注意在其中压缩NULL值，例如像gist_circle_compress那样存储(Datum) 0。

decompress

compress方法的逆操作。将数据项的索引表示转换成数据库能够操作的格式。

该SQL声明必须如下所示：

```
CREATE OR REPLACE FUNCTION my_decompress(internal)
RETURNS internal
AS 'MODULE_PATHNAME'
LANGUAGE C STRICT;
```

C 模块中相应的代码可以采用以下框架：

```
Datum       my_decompress(PG_FUNCTION_ARGS);
PG_FUNCTION_INFO_V1(my_decompress);

Datum
my_decompress(PG_FUNCTION_ARGS)
{
    PG_RETURN_POINTER(PG_GETARG_POINTER(0));
}
```

上述框架适用于不需要解压的情况。

penalty

返回一个值，指示把新项插入树中特定分支的“代价”。项会沿着树中penalty最小的路径插入。penalty返回的值应为非负；如果返回负值，它将被按零处理。

该函数的SQL声明必须如下所示：

```
CREATE OR REPLACE FUNCTION my_penalty(internal, internal,
 internal)
RETURNS internal
AS 'MODULE_PATHNAME'
LANGUAGE C STRICT;  -- 某些情况下 penalty 函数不必是严格函数
```

而 C 模块中的对应代码则可以遵循如下框架：

```
Datum       my_penalty(PG_FUNCTION_ARGS);
PG_FUNCTION_INFO_V1(my_penalty);

Datum
my_penalty(PG_FUNCTION_ARGS)
{
    GISTENTRY  *origentry = (GISTENTRY *) PG_GETARG_POINTER(0);
    GISTENTRY  *newentry = (GISTENTRY *) PG_GETARG_POINTER(1);
    float      *penalty = (float *) PG_GETARG_POINTER(2);
    data_type  *orig = DatumGetDataType(origentry->key);
    data_type  *new = DatumGetDataType(newentry->key);

    *penalty = my_penalty_implementation(orig, new);
    PG_RETURN_POINTER(penalty);
}
```

penalty函数对于索引的良好性能至关重要。
它会在插入时用于决定在树中应沿着哪个分支向下，以便选择把新项加到哪里。
在查询时，索引越平衡，查找就越快。

picksplit

当索引页必须分裂时，该函数决定页面上的哪些项留在旧页中，哪些移到新页中。

该函数的 SQL 声明必须类似如下形式：

```
CREATE OR REPLACE FUNCTION my_picksplit(internal, internal)
RETURNS internal
AS 'MODULE_PATHNAME'
LANGUAGE C STRICT;
```

而 C 模块中匹配的代码可以遵循如下框架：

```
Datum       my_picksplit(PG_FUNCTION_ARGS);
PG_FUNCTION_INFO_V1(my_picksplit);

Datum
my_picksplit(PG_FUNCTION_ARGS)
{
    GistEntryVector *entryvec = (GistEntryVector *) PG_GETARG_
POINTER(0);
    OffsetNumber maxoff = entryvec->n - 1;
    GISTENTRY  *ent = entryvec->vector;
    GIST_SPLITVEC *v = (GIST_SPLITVEC *) PG_GETARG_POINTER(1);
    int         i,
```

```
                    nbytes;
        OffsetNumber *left,
                    *right;
        data_type  *tmp_union;
        data_type  *unionL;
        data_type  *unionR;
        GISTENTRY **raw_entryvec;

        /* 这里的代码将项划分到 left 和 right 两个数组中 */
    }
```

和penalty一样，picksplit函数对于索引的良好性能至关重要。设计合适的penalty和picksplit实现，正是实现高性能GiST索引的难点所在。

same

如果两个索引项相同则返回真，否则返回假。

该函数的SQL声明必须如下所示：

```
CREATE OR REPLACE FUNCTION my_same(internal, internal, internal)
RETURNS internal
AS 'MODULE_PATHNAME'
LANGUAGE C STRICT;
```

而 C 模块中的对应代码则可以遵循如下框架：

```
Datum       my_same(PG_FUNCTION_ARGS);
PG_FUNCTION_INFO_V1(my_same);

Datum
my_same(PG_FUNCTION_ARGS)
{
    prefix_range *v1 = PG_GETARG_PREFIX_RANGE_P(0);
    prefix_range *v2 = PG_GETARG_PREFIX_RANGE_P(1);
    bool       *result = (bool *) PG_GETARG_POINTER(2);

    *result = my_eq(v1, v2);
    PG_RETURN_POINTER(result);
}
```

出于历史原因，same函数并不是直接返回一个布尔结果；相反，
它必须把该标志存储到第三个参数指示的位置。

distance

给定一个索引项p和一个查询值q，该函数确定索引项与查询值之间的“距离”。
如果操作符类包含任何排序操作符，就必须提供此函数。
使用排序操作符的查询会优先返回“距离”值最小的索引项，
因此结果必须与该操作符的语义一致。对于叶子索引项，结果仅表示到该索引项的距离；
对于内部树节点，结果必须是其任意子项可能具有的最小距离。

该函数的SQL声明必须如下所示：

```
CREATE OR REPLACE FUNCTION my_distance(internal, data_type,
 smallint, oid)
RETURNS float8
AS 'MODULE_PATHNAME'
LANGUAGE C STRICT;
```

而 C 模块中的对应代码则可以遵循如下框架：

```
Datum       my_distance(PG_FUNCTION_ARGS);
PG_FUNCTION_INFO_V1(my_distance);

Datum
my_distance(PG_FUNCTION_ARGS)
{
    GISTENTRY  *entry = (GISTENTRY *) PG_GETARG_POINTER(0);
    data_type  *query = PG_GETARG_DATA_TYPE_P(1);
    StrategyNumber strategy = (StrategyNumber) PG_GETARG_UINT16(
2);
    /* Oid subtype = PG_GETARG_OID(3); */
    data_type  *key = DatumGetDataType(entry->key);
    double      retval;

    /*
     * 根据 strategy、key 和 query 确定返回值。
     */

    PG_RETURN_FLOAT8(retval);
}
```

distance函数的参数与consistent函数的参数相同，只是不使用 recheck 标志。到叶子索引项的距离必须始终精确确定，因为元组一旦返回就无法再重新排序。
在确定到内部树节点的距离时允许有一定近似，
只要结果永不大于任一子节点的实际距离即可。因此，例如在几何应用中，
到包围盒的距离通常就足够了。结果值可以是任意有限的float8值。
（无穷大和负无穷在内部用于处理空值等情况，因此不建议distance
函数返回这些值。）

所有 GiST 支持方法通常都在短生命周期的内存上下文中被调用；也就是说，每处理完一个元组，CurrentMemoryContext都会被重置。因此通常无需过分担心释放所有通过 palloc 分配的内容。不过，在某些情况下，让支持方法在重复调用之间缓存数据是有用的。
要做到这一点，可将寿命更长的数据分配在fcinfo->flinfo->fn_mcxt中，并在fcinfo->flinfo->fn_extra中保存指向它的指针。
这类数据会在一次索引操作期间存活（例如一次 GiST 索引扫描、索引构建或索引元组插入）。在替换fn_extra值时要注意 pfree 旧值，否则泄漏会在整个操作期间不断累积。

53.3. 实现

53.3.1. GiST 缓冲构建

仅靠把所有元组逐个插入来构建大型 GiST 索引往往很慢，因为如果索引元组分散在整个索引中，而索引又大到无法放入缓存，插入时就需要大量随机 I/O。从版本 9.2 开始，PostgreSQL 支持一种更高效的、基于缓冲的 GiST 索引构建方法，对于非有序数据集，它可以显著减少所需的随机 I/O 次数。对于顺序良好的数据集，收益较小或者根本没有，因为一次只有少量页面会接收新元组，而这些页面即使整个索引放不进缓存，也能够放进缓存。

不过，缓冲索引构建需要更频繁地调用penalty函数，这会消耗一些额外的 CPU 资源。此外，缓冲区需要临时磁盘空间，最多可达最终索引的大小。
缓冲还可能正面或负面地影响最终索引的质量。这种影响取决于多种因素，
例如输入数据的分布以及操作符类的实现。

默认情况下，当索引大小达到effective_cache_size时，GiST 索引构建会切换到缓冲方法。也可以通过 CREATE INDEX 命令的BUFFERING参数手工强制启用或禁止缓冲。默认行为在大多数情况下都不错，但如果输入数据是有序的，关闭缓冲模式可能会略微加快构建速度。

53.4. 示例

PostgreSQL源代码发行包包含了若干使用GiST实现的索引方法示例。
核心系统目前提供了文本检索支持（为`tsvector`和`tsquery`建立索引），
并为某些内置几何数据类型提供了与 R 树等价的功能（见`src/backend/access/gist/gistproc.c`）。下列`contrib`模块中也包含GiST操作符类：

`btree_gist`

为多种数据类型提供与 B-树等价的功能

`cube`

多维立方体的索引

`hstore`

用于存储（键，值）对的模块

`intarray`

一维 int4 值数组的 RD 树

`ltree`

树状结构的索引

`pg_trgm`

基于三字符组匹配的文本相似度

`seg`

“float 范围” 的索引

第 54 章 SP-GiST 索引

54.1. 简介

SP-GiST 是空间分区 GiST 的缩写。SP-GiST 支持分区搜索树，这使得开发多种不同的非平衡数据结构成为可能，例如四叉树、k-d 树以及后缀树。这些结构的共同特征是，它们会反复将搜索空间划分为不必等大的分区。与这种划分规则良好匹配的搜索可以非常快。

这些常见数据结构最初是为内存中使用而开发的。在主存中，
它们通常被设计成一组由指针链接的动态分配结点。由于这些指针链可能相当长，
直接存储到磁盘上并不合适，因为那会需要过多的磁盘访问。相比之下，
基于磁盘的数据结构应当具有较高的扇出，以尽量减少 I/O。SP-GiST 要解决的难题是，如何以这样的方式将搜索树结点映射到磁盘页：即使搜索遍历了许多结点，
也只需访问少数几个磁盘页。

像 GiST 一样，SP-GiST 的目标是让数据类型领域专家而非数据库专家，能够针对自定义数据类型开发合适的访问方法。

这里的一些信息来自普渡大学的 SP-GiST 索引项目网站[1]。SP-GiST 在 PostgreSQL 中的实现主要由 Teodor Sigaev 和 Oleg Bartunov 维护，他们的网站[2]上还有更多信息。

54.2. 可扩展性

SP-GiST 提供了一个高度抽象的接口，访问方法开发者只需实现特定数据类型所需的方法。SP-GiST 核心负责将树结构高效映射到磁盘并执行搜索，同时也处理并发与日志记录方面的问题。

SP-GiST 树的叶子元组包含与被索引列相同数据类型的值。位于根层的叶子元组始终包含原始的被索引数据值，而位于较低层的叶子元组可能只包含压缩表示，例如一个后缀。
在这种情况下，
操作符类支持函数必须能够利用为到达叶子层而经过的内部元组中累积的信息，
重建出原始值。

内部元组更为复杂，因为它们是搜索树中的分支点。每个内部元组都包含一个或多个结点，
代表相似叶子值的分组。一个结点包含一个向下链接，
它要么指向更低层级的另一个内部元组，要么指向一小组位于同一索引页上的叶子元组。
每个结点都有一个描述它的标签；例如在后缀树中，
结点标签可以是字符串值的下一个字符。内部元组还可以选择带有一个描述其所有成员的
前缀值。在后缀树中，这可以是所表示字符串的公共前缀。前缀值不一定真的是前缀，
它也可以是操作符类需要的任何数据；例如在四叉树中，
它可以存储划分四个象限所依据的中心点。这样，四叉树的内部元组还会包含四个结点，
对应于该中心点周围的四个象限。

某些树算法需要知道当前元组所在的层级（或深度），因此 SP-GiST 核心允许操作符类在沿树向下遍历时管理层级计数。它还支持在需要时增量重建所表示的值。

> **注意**
>
> SP-GiST 核心代码负责处理值为 null 的索引项。虽然 SP-GiST 索引会为被索引列中的 null 值存储索引项，但索引操作符类代码看不到这些项：值为 null 的索引项或搜索条件绝不会传给操作符类方法。（这里假定 SP-GiST 操作符是严格的，因此对 null 值不可能返回真。）所以这里不再讨论 null 值。

[1] http://www.cs.purdue.edu/spgist/
[2] http://www.sai.msu.su/~megera/wiki/spgist_dev

SP-GiST 的索引操作符类必须提供五个用户定义方法。五个必需方法都遵循这样的约定：接受两个 internal 参数，第一个参数是指向某个 C 结构体的指针，其中包含该支持方法的输入值；第二个参数也是指向某个 C 结构体的指针，方法必须将输出值写入其中。四个必需方法只返回 void，因为它们的全部结果都体现在输出结构体中；但 leaf_consistent 还返回一个 boolean 结果。这些方法不得修改其输入结构体中的任何字段。在所有情况下，调用用户定义方法之前，输出结构体都会先被清零。

五个用户定义方法是：

config

返回索引实现的静态信息，包括前缀和结点标签数据类型的 OID。

该SQL声明必须如下所示：

```
CREATE FUNCTION my_config(internal, internal) RETURNS void ...
```

第一个参数是一个指向spgConfigInC 结构体的指针，其中包含该函数的输入数据。第二个参数是一个指向spgConfigOutC 结构体的指针，函数必须将结果数据填入其中。

```
typedef struct spgConfigIn
{
    Oid         attType;        /* 要被索引的数据类型 */
} spgConfigIn;

typedef struct spgConfigOut
{
    Oid         prefixType;     /* 内部元组前缀的数据类型 */
    Oid         labelType;      /* 内部元组结点标签的数据类型 */
    bool        canReturnData;  /* 操作符类能重建原始数据 */
    bool        longValuesOK;   /* 操作符类能处理大小 > 1 页的值 */
} spgConfigOut;
```

attType的传入是为了支持多态索引操作符类；对于普通的固定数据类型操作符类，它始终具有相同的值，因此可以忽略。

对于不使用前缀的操作符类，可以将 prefixType 设为 VOIDOID。同样，对于不使用结点标签的操作符类，可以将 labelType 设为 VOIDOID。如果操作符类能够重建最初提供的索引值，则应将 canReturnData 设为真。只有在 attType 是变长类型，并且该操作符类能够通过反复取后缀来切分长值时，才应将 longValuesOK 设为真（参见第 54.3.1 节）。

choose

为向内部元组插入新值选择一种方法。

该SQL声明必须如下所示：

```
CREATE FUNCTION my_choose(internal, internal) RETURNS void ...
```

第一个参数是一个指向spgChooseInC 结构体的指针，其中包含该函数的输入数据。第二个参数是一个指向spgChooseOutC 结构体的指针，函数必须将结果数据填入其中。

```
typedef struct spgChooseIn
{
    Datum       datum;          /* 要被索引的原始 datum */
    Datum       leafDatum;      /* 当前要存储在叶子中的 datum */
    int         level;          /* 当前层级（从零开始计） */

    /* 来自当前内部元组的数据 */
```

```
    bool        allTheSame;     /* 元组被标记为全部相同？ */
    bool        hasPrefix;      /* 元组有前缀？ */
    Datum       prefixDatum;    /* 如果有，前缀值 */
    int         nNodes;         /* 内部元组中的结点数 */
    Datum      *nodeLabels;     /* 结点标签值（如果没有则为 NULL）
 */
} spgChooseIn;

typedef enum spgChooseResultType
{
    spgMatchNode = 1,           /* 下降到现有结点 */
    spgAddNode,                 /* 向内部元组添加一个结点 */
    spgSplitTuple               /* 拆分内部元组（修改其前缀） */
} spgChooseResultType;

typedef struct spgChooseOut
{
    spgChooseResultType resultType;     /* 动作代码，见上文 */
    union
    {
        struct                  /* spgMatchNode 的结果 */
        {
            int         nodeN;      /* 下降到该结点（索引从 0 开始）
 */
            int         levelAdd;   /* 层级增加这么多 */
            Datum       restDatum;  /* 新的叶子 datum */
        }           matchNode;
        struct                  /* spgAddNode 的结果 */
        {
            Datum       nodeLabel;  /* 新结点的标签 */
            int         nodeN;      /* 在哪里插入它（索引从 0 开始）
 */
        }           addNode;
        struct                  /* spgSplitTuple 的结果 */
        {
            /* 构造只有一个结点的新内部元组所需的信息 */
            bool        prefixHasPrefix;    /* 元组应有前缀？ */
            Datum       prefixPrefixDatum;  /* 如果有，前缀值 */
            Datum       nodeLabel;          /* 结点的标签 */

            /* 构造包含所有旧结点的新下层内部元组所需的信息 */
            bool        postfixHasPrefix;   /* 元组应有前缀？ */
            Datum       postfixPrefixDatum; /* 如果有，前缀值 */
        }           splitTuple;
    }           result;
} spgChooseOut;
```

datum是将要插入索引的原始 datum。leafDatum最初与datum相同，但在树的较低层可能发生变化，如果choose或picksplit方法对它进行了修改。
当插入搜索到达叶页时，leafDatum的当前值将存储到新创建的叶子元组中。level是当前内部元组的层级，根层为零。allTheSame为真，
表示当前内部元组被标记为包含多个等价结点（参见第 54.3.3 节）。hasPrefix为真时，表示当前内部元组包含前缀；若是如此，prefixDatum就是该前缀值。nNodes是内部元组中包含的子结点数量，而nodeLabels是它们的标签值数组；如果没有标签，则为 NULL。

choose 函数可以判定：新值要么匹配某个现有子结点，要么必须添加一个新子结点，要么与该元组的前缀不一致，因此必须拆分该内部元组以创建限制性更弱的前缀。

如果新值匹配某个现有子结点，则将 resultType 设为 spgMatchNode。将 nodeN 设为该结点在结点数组中的索引（从零开始）。将 levelAdd 设为通过该结点向下下降所导致的 level 增量；如果操作符类不使用层级，则保持为零。若操作符类不会在层级之间修改 datum，则将 restDatum 设为与 datum 相等；否则，将它设为下一层要用作 leafDatum 的修改后值。

如果必须添加一个新子结点，则将 resultType 设为 spgAddNode。将 nodeLabel 设为新结点要使用的标签，并将 nodeN 设为该结点应插入到结点数组中的位置索引（从零开始）。添加该结点后，choose 函数会使用修改后的内部元组再次被调用；这次调用应返回 spgMatchNode。

如果新值与该元组的前缀不一致，则将 resultType 设为 spgSplitTuple。这个动作会把所有现有结点移动到一个新的较低层内部元组中，并用一个仅含单个结点、链接到该新下层内部元组的元组替换现有内部元组。将 prefixHasPrefix 设为指示新的上层元组是否应有前缀；若应有，则将 prefixPrefixDatum 设为该前缀值。这个新前缀值必须比原来的限制性更弱，以便能够接受将要索引的新值，并且它不应比原来的前缀更长。将 nodeLabel 设为指向新的较低层内部元组的那个结点所使用的标签。将postfixHasPrefix设为指示新的较低层内部元组是否应有前缀；若应有，则将 postfixPrefixDatum 设为该前缀值。这两个前缀与附加标签的组合，必须与原始前缀具有相同的含义，因为没有机会修改被移动到新下层元组中的结点标签，也不能更改任何子索引项。结点拆分完成后，choose 函数会用替换后的内部元组再次被调用。这次调用通常会得到 spgAddNode 结果，因为拆分步骤中加入的结点标签多半不会匹配新值；因此之后还会有第三次调用，它最终返回 spgMatchNode，让插入下降到叶子层。

picksplit

决定如何在一组叶子元组上创建一个新的内部元组。

该SQL声明必须如下所示：

```
CREATE FUNCTION my_picksplit(internal, internal) RETURNS void ..
.
```

第一个参数是一个指向spgPickSplitInC 结构体的指针，其中包含该函数的输入数据。第二个参数是一个指向spgPickSplitOutC 结构体的指针，函数必须将结果数据填入其中。

```
typedef struct spgPickSplitIn
{
    int         nTuples;        /* 叶子元组的数量 */
    Datum      *datums;         /* 它们的 datum（长度为 nTuples 的
数组） */
    int         level;          /* 当前层级（从零开始计） */
} spgPickSplitIn;

typedef struct spgPickSplitOut
{
    bool        hasPrefix;      /* 新内部元组应有前缀? */
    Datum       prefixDatum;    /* 如果有，前缀值 */

    int         nNodes;         /* 新内部元组的结点数 */
    Datum      *nodeLabels;     /* 它们的标签（或为 NULL 表示无标签）
 */

    int        *mapTuplesToNodes;   /* 每个叶子元组对应的结点索引 */
    Datum      *leafTupleDatums;    /* 每个新叶子元组中存储的 datum
 */
```

```
} spgPickSplitOut;
```

nTuples是所提供的叶子元组数量。datums是这些元组的 datum 值数组。level是所有这些叶子元组当前共同的层级，它将成为新内部元组的层级。

将 hasPrefix 设为指示新的内部元组是否应有前缀；若应有，则将 prefixDatum 设为该前缀值。将 nNodes 设为新内部元组将包含的结点数，并将 nodeLabels 设为这些结点的标签值数组。（如果结点不需要标签，则将 nodeLabels 设为 NULL；详情参见第 54.3.2 节。）将 mapTuplesToNodes 设为一个数组，其中给出每个叶子元组应分配到的结点索引（从零开始）。将 leafTupleDatums 设为要存储在新叶子元组中的值数组（如果操作符类不会在层级之间修改 datum，这些值就与输入的 datums 相同）。注意，picksplit 函数负责为 nodeLabels、mapTuplesToNodes 和 leafTuple Datums 数组执行 palloc。

如果提供了多于一个叶子元组，则期望 picksplit 函数把它们划分到多于一个结点中；否则就无法把叶子元组拆分到多个页上，而这正是此操作的最终目的。因此，如果 picksplit 最终把所有叶子元组都放进同一个结点，SP-GiST 核心代码会覆盖这一决定，生成一个内部元组，并将叶子元组随机分配到多个标签相同的结点上。
这样的元组会被标记为 allTheSame，以表明发生了这种情况。choose 和 inner_consistent 函数必须对这种内部元组做出恰当处理。更多信息见第 54.3.3 节。

只有在 config 函数将 longValuesOK 设为真，并且提供了一个大于一页的输入值时，picksplit 才会应用到单个叶子元组。在这种情况下，这个操作的目的是剥离一个前缀，并产生一个新的、更短的叶子 datum 值。该调用会重复进行，直到生成足够短、能够放入一页的叶子 datum。更多信息见第 54.3.1 节。

inner_consistent

在树搜索期间返回需要继续跟随的一组结点（分支）。

该SQL声明必须如下所示：

```
CREATE FUNCTION my_inner_consistent(internal, internal) RETURNS
 void ...
```

第一个参数是一个指向spgInnerConsistentInC 结构体的指针，其中包含该函数的输入数据。第二个参数是一个指向spgInnerConsistentOutC 结构体的指针，函数必须将结果数据填入其中。

```
typedef struct spgInnerConsistentIn
{
    ScanKey     scankeys;        /* 操作符和比较值的数组 */
    int         nkeys;           /* 数组长度 */

    Datum       reconstructedValue;     /* 在父元组处重建的值 */
    int         level;           /* 当前层级（从零开始计） */
    bool        returnData;      /* 必须返回原始数据？ */

    /* 来自当前内部元组的数据 */
    bool        allTheSame;      /* 元组被标记为全部相同？ */
    bool        hasPrefix;       /* 元组有前缀？ */
    Datum       prefixDatum;     /* 如果有，前缀值 */
    int         nNodes;          /* 内部元组中的结点数 */
    Datum      *nodeLabels;      /* 结点标签值（如果没有则为 NULL)
 */
} spgInnerConsistentIn;

typedef struct spgInnerConsistentOut
{
```

```
    int         nNodes;          /* 需要访问的子结点数 */
    int        *nodeNumbers;     /* 它们在结点数组中的索引 */
    int        *levelAdds;       /* 对每个结点层级增加这么多 */
    Datum      *reconstructedValues;     /* 关联的重建值 */
} spgInnerConsistentOut;
```

数组scankeys的长度为nkeys，它描述索引搜索条件。这些条件用 AND 组合 – 只有满足全部条件的索引项才是我们关心的。（注意，nkeys= 0 表示所有索引项都满足该查询。）通常一致性检查函数只关心每个数组元素的sk_strategy和sk_argument字段，它们分别给出可索引操作符和比较值。特别地，无需检查sk_flags以判断比较值是否为 NULL，因为 SP-GiST 核心代码会过滤掉此类条件。reconstructedValue是为父元组重建的值；以下情况下它为(Datum) 0：位于根层，或者inner_consistent函数没有在父层提供该值。level是当前内部元组的层级，根层为零。returnData为true表示本查询需要重建数据；这要求config函数将canReturnData设为真。allTheSame为真，表示当前内部元组被标记为“全部相同”；在这种情况下，所有结点都具有相同的标签（如果有），因此要么全部匹配该查询，要么全部不匹配（参见第 54.3.3 节）。hasPrefix为真时，表示当前内部元组包含前缀；若是如此，prefixDatum就是该前缀值。nNodes是内部元组中包含的子结点数量，而nodeLabels是它们的标签值数组；如果结点没有标签，则为 NULL。

nNodes 必须设为搜索需要访问的子结点数量，并且 nodeNumbers 必须设为这些结点索引的数组。如果操作符类跟踪层级，则将 levelAdds 设为一个数组，其中给出下降到每个待访问结点时所需增加的层数。（这些增量常常对所有结点都相同，但并非必然如此，所以这里使用数组。）如果需要值重建，则将 reconstructedValues 设为一个数组，其中包含为每个待访问子结点重建的值；否则，将 reconstructedValues 保持为 NULL。注意，inner_consistent 函数负责在当前内存上下文中为 nodeNumbers、levelAdds 和 reconstructedValues 数组执行 palloc。

leaf_consistent

如果叶子元组满足查询，则返回 true。

该SQL声明必须如下所示：

```
CREATE FUNCTION my_leaf_consistent(internal, internal) RETURNS
 bool ...
```

第一个参数是一个指向spgLeafConsistentInC 结构体的指针，其中包含该函数的输入数据。第二个参数是一个指向spgLeafConsistentOutC 结构体的指针，函数必须将结果数据填入其中。

```
typedef struct spgLeafConsistentIn
{
    ScanKey     scankeys;        /* 操作符和比较值的数组 */
    int         nkeys;           /* 数组长度 */

    Datum       reconstructedValue;      /* 在父元组处重建的值 */
    int         level;           /* 当前层级（从零开始计） */
    bool        returnData;      /* 必须返回原始数据？ */

    Datum       leafDatum;       /* 叶子元组中的 datum */
} spgLeafConsistentIn;

typedef struct spgLeafConsistentOut
{
    Datum       leafValue;       /* 重建出的原始数据（如果有） */
    bool        recheck;         /* 如果必须重新检查操作符则设为真 */
```

```
} spgLeafConsistentOut;
```

数组scankeys的长度为nkeys，它描述索引搜索条件。这些条件用 AND 组合 – 只有满足全部条件的索引项才满足该查询。（注意，nkeys= 0 表示所有索引项都满足该查询。）通常一致性检查函数只关心每个数组元素的sk_strategy和sk_argument字段，它们分别给出可索引操作符和比较值。特别地，无需检查sk_flags以判断比较值是否为 NULL，因为 SP-GiST 核心代码会过滤掉此类条件。reconstructedValue是为父元组重建的值；以下情况下它为(Datum) 0：位于根层，或者inner_consistent函数没有在父层提供该值。level是当前叶子元组的层级，根层为零。returnData为true表示本查询需要重建数据；这要求config函数将canReturnData设为真。leafDatum是当前叶子元组中存储的键值。

如果叶子元组匹配查询，则该函数必须返回 true，否则返回 false。在返回 true 的情况下，如果 returnData 为 true，则必须将 leafValue 设为最初为该叶子元组提供并建立索引的值。此外，如果匹配结果不确定，因而必须将操作符重新应用到实际的堆元组上以验证匹配，则可以将 recheck 设为 true。

所有 SP-GiST 支持方法通常都在一个短生命周期的内存上下文中调用；也就是说，处理完每个元组后，CurrentMemoryContext 都会被重置。因此，通常不必太担心是否 pfree 了你用 palloc 分配的所有内容。（config 方法是个例外：它应尽量避免内存泄漏。不过通常 config 方法只需把常量赋入传入的参数结构体即可。）

如果被索引列属于支持排序规则的数据类型，则索引排序规则会通过标准的 PG_GET_COLLATION() 机制传递给所有支持方法。

54.3. 实现

本节介绍实现细节以及其他一些对 SP-GiST 操作符类实现者有用的技巧。

54.3.1. SP-GiST 限制

单个叶子元组和内部元组都必须能放入单个索引页中（默认 8KB）。因此，在对变长数据类型的值建立索引时，只有像后缀树这类方法才能支持长值：树的每一层都包含足够短、能放进一页的前缀，而最终叶子层也包含足够短、能放进一页的后缀。
只有当操作符类准备好保证这一点时，才应将 longValuesOK 设为 TRUE。否则，SP-GiST 核心会拒绝为过大、无法放入索引页的值建立索引的请求。

同样，确保内部元组不会增长到大得无法放入索引页，也是操作符类的责任；这限制了单个内部元组中可使用的子结点数量，以及前缀值的最大大小。

另一个限制是，当内部元组的某个结点指向一组叶子元组时，这些元组必须全部位于同一个索引页上。（这是一个设计决策，目的是减少寻道，并节省把这类元组链接成链时所需链接占用的空间。）
如果一组叶子元组增长到单页无法容纳，就会执行拆分并插入一个中间内部元组。要解决这个问题，新的内部元组必须把叶子值集合划分成多个结点组。如果操作符类的 picksplit 函数做不到这一点，SP-GiST 核心就会诉诸第 54.3.3 节中描述的非常规措施。

54.3.2. 无结点标签的 SP-GiST

某些树算法对每个内部元组都使用固定的一组结点；例如在四叉树中，总是恰好有四个结点，对应于内部元组中心点周围的四个象限。在这种情况下，代码通常按编号处理结点，因此不需要显式的结点标签。
为了省略结点标签（从而节省一些空间），picksplit 函数可以为 nodeLabels 数组返回 NULL。随后对 choose 和 inner_consistent 的调用中，nodeLabels 也将为 NULL。原则上，同一个索引中可以对某些内部元组使用结点标签，而对另一些省略。

当处理具有无标签结点的内部元组时，choose 返回 spgAddNode 是错误的，因为在这种情况下结点集合应被视为固定不变。另外，spgSplitTuple 动作中没有生成无标签结点的规定，因为预期随后还需要执行 spgAddNode 动作。

54.3.3. “全部相同”的内部元组

当 picksplit 无法把提供的叶子值划分为至少两个结点类别时，SP-GiST 核心可以覆盖操作符类 picksplit 函数的结果。在这种情况下，会创建一个新的内部元组，其中有多个结点，而每个结点都具有相同的标签（如果有）；这个标签就是 picksplit 给它唯一使用的那个结点分配的标签。叶子值会被随机分配到这些等价结点中。该内部元组会设置 allTheSame 标志，用来提醒 choose 和 inner_consistent 函数：这个元组的结点集合并不是它们通常会预期的那种。

在处理 allTheSame 元组时，choose 返回 spgMatchNode 表示新值可以分配给任意一个等价结点；核心代码会忽略给出的 nodeN 值，并随机下降到其中一个结点（以保持树的平衡）。choose 返回 spgAddNode 则属于错误，因为那会使结点不再全部等价；如果待插入的值与现有结点不匹配，就必须使用 spgSplitTuple 动作。

在处理 allTheSame 元组时，inner_consistent 函数应当要么把全部结点返回为继续索引搜索的目标，要么一个也不返回，因为它们都是等价的。这是否需要编写特殊情况代码，取决于 inner_consistent 函数平常对这些结点含义做了多大程度的假定。

54.4. 示例

PostgreSQL 源代码发行版中包含了若干 SP-GiST 索引操作符类示例。核心系统目前为文本列提供后缀树，并为点提供两类树：四叉树和 k-d 树。要查看代码，请参阅 src/backend/access/spgist/。

第 55 章 GIN Indexes

55.1. Introduction

GIN stands for Generalized Inverted Index. GIN is designed for handling cases where the items to be indexed are composite values, and the queries to be handled by the index need to search for element values that appear within the composite items. For example, the items could be documents, and the queries could be searches for documents containing specific words.

We use the word item to refer to a composite value that is to be indexed, and the word key to refer to an element value. GIN always stores and searches for keys, not item values per se.

A GIN index stores a set of (key, posting list) pairs, where a posting list is a set of row IDs in which the key occurs. The same row ID can appear in multiple posting lists, since an item can contain more than one key. Each key value is stored only once, so a GIN index is very compact for cases where the same key appears many times.

GIN is generalized in the sense that the GIN access method code does not need to know the specific operations that it accelerates. Instead, it uses custom strategies defined for particular data types. The strategy defines how keys are extracted from indexed items and query conditions, and how to determine whether a row that contains some of the key values in a query actually satisfies the query.

One advantage of GIN is that it allows the development of custom data types with the appropriate access methods, by an expert in the domain of the data type, rather than a database expert. This is much the same advantage as using GiST.

The GIN implementation in PostgreSQL is primarily maintained by Teodor Sigaev and Oleg Bartunov. There is more information about GIN on their website[1].

55.2. Extensibility

The GIN interface has a high level of abstraction, requiring the access method implementer only to implement the semantics of the data type being accessed. The GIN layer itself takes care of concurrency, logging and searching the tree structure.

All it takes to get a GIN access method working is to implement four (or five) user-defined methods, which define the behavior of keys in the tree and the relationships between keys, indexed items, and indexable queries. In short, GIN combines extensibility with generality, code reuse, and a clean interface.

The four methods that an operator class for GIN must provide are:

`int compare(Datum a, Datum b)`

Compares two keys (not indexed items!) and returns an integer less than zero, zero, or greater than zero, indicating whether the first key is less than, equal to, or greater than the second. Null keys are never passed to this function.

`Datum *extractValue(Datum itemValue, int32 *nkeys, bool **nullFlags)`

Returns a palloc'd array of keys given an item to be indexed. The number of returned keys must be stored into `*nkeys`. If any of the keys can be null, also palloc an array of `*nkeys` booleans, store its address at `*nullFlags`, and set these null

[1] http://www.sai.msu.su/~megera/wiki/Gin

flags as needed. `*nullFlags` can be left `NULL` (its initial value) if all keys are non-null. The return value can be `NULL` if the item contains no keys.

`Datum *extractQuery(Datum query, int32 *nkeys, StrategyNumber n, bool **pmatch, Pointer **extra_data, bool **nullFlags, int32 *searchMode)`

Returns a palloc'd array of keys given a value to be queried; that is, `query` is the value on the right-hand side of an indexable operator whose left-hand side is the indexed column. `n` is the strategy number of the operator within the operator class (see 第 35.14.2 节). Often, `extractQuery` will need to consult `n` to determine the data type of `query` and the method it should use to extract key values. The number of returned keys must be stored into `*nkeys`. If any of the keys can be null, also palloc an array of `*nkeys` booleans, store its address at `*nullFlags`, and set these null flags as needed. `*nullFlags` can be left NULL (its initial value) if all keys are non-null. The return value can be NULL if the `query` contains no keys.

`searchMode` is an output argument that allows `extractQuery` to specify details about how the search will be done. If `*searchMode` is set to `GIN_SEARCH_MODE_DEFAULT` (which is the value it is initialized to before call), only items that match at least one of the returned keys are considered candidate matches. If `*searchMode` is set to `GIN_SEARCH_MODE_INCLUDE_EMPTY`, then in addition to items containing at least one matching key, items that contain no keys at all are considered candidate matches. (This mode is useful for implementing is-subset-of operators, for example.) If `*searchMode` is set to `GIN_SEARCH_MODE_ALL`, then all non-null items in the index are considered candidate matches, whether they match any of the returned keys or not. (This mode is much slower than the other two choices, since it requires scanning essentially the entire index, but it may be necessary to implement corner cases correctly. An operator that needs this mode in most cases is probably not a good candidate for a GIN operator class.) The symbols to use for setting this mode are defined in `access/gin.h`.

`pmatch` is an output argument for use when partial match is supported. To use it, `extractQuery` must allocate an array of `*nkeys` booleans and store its address at `*pmatch`. Each element of the array should be set to TRUE if the corresponding key requires partial match, FALSE if not. If `*pmatch` is set to NULL then GIN assumes partial match is not required. The variable is initialized to NULL before call, so this argument can simply be ignored by operator classes that do not support partial match.

`extra_data` is an output argument that allows `extractQuery` to pass additional data to the `consistent` and `comparePartial` methods. To use it, `extractQuery` must allocate an array of `*nkeys` pointers and store its address at `*extra_data`, then store whatever it wants to into the individual pointers. The variable is initialized to NULL before call, so this argument can simply be ignored by operator classes that do not require extra data. If `*extra_data` is set, the whole array is passed to the `consistent` method, and the appropriate element to the `comparePartial` method.

`bool consistent(bool check[], StrategyNumber n, Datum query, int32 nkeys, Pointer extra_data[], bool *recheck, Datum queryKeys[], bool nullFlags[])`

Returns TRUE if an indexed item satisfies the query operator with strategy number `n` (or might satisfy it, if the recheck indication is returned). This function does not have direct access to the indexed item's value, since GIN does not store items explicitly. Rather, what is available is knowledge about which key values extracted from the query appear in a given indexed item. The `check` array has length `nkeys`, which is the same as the number of keys previously returned by `extractQuery`

for this `query` datum. Each element of the `check` array is TRUE if the indexed item contains the corresponding query key, ie, if (check[i] == TRUE) the i-th key of the `extractQuery` result array is present in the indexed item. The original `query` datum is passed in case the `consistent` method needs to consult it, and so are the `queryKeys[]` and `nullFlags[]` arrays previously returned by `extractQuery`. `extra_data` is the extra-data array returned by `extractQuery`, or NULL if none.

When `extractQuery` returns a null key in `queryKeys[]`, the corresponding `check[]` element is TRUE if the indexed item contains a null key; that is, the semantics of `check[]` are like `IS NOT DISTINCT FROM`. The `consistent` function can examine the corresponding `nullFlags[]` element if it needs to tell the difference between a regular value match and a null match.

On success, `*recheck` should be set to TRUE if the heap tuple needs to be rechecked against the query operator, or FALSE if the index test is exact. That is, a FALSE return value guarantees that the heap tuple does not match the query; a TRUE return value with `*recheck` set to FALSE guarantees that the heap tuple does match the query; and a TRUE return value with `*recheck` set to TRUE means that the heap tuple might match the query, so it needs to be fetched and rechecked by evaluating the query operator directly against the originally indexed item.

Optionally, an operator class for GIN can supply a fifth method:

```
int comparePartial(Datum partial_key, Datum key, StrategyNumber n,
 Pointer extra_data)
```

Compare a partial-match query key to an index key. Returns an integer whose sign indicates the result: less than zero means the index key does not match the query, but the index scan should continue; zero means that the index key does match the query; greater than zero indicates that the index scan should stop because no more matches are possible. The strategy number `n` of the operator that generated the partial match query is provided, in case its semantics are needed to determine when to end the scan. Also, `extra_data` is the corresponding element of the extra-data array made by `extractQuery`, or NULL if none. Null keys are never passed to this function.

To support "partial match" queries, an operator class must provide the `comparePartial` method, and its `extractQuery` method must set the `pmatch` parameter when a partial-match query is encountered. See 第 55.3.2 节 for details.

The actual data types of the various `Datum` values mentioned above vary depending on the operator class. The item values passed to `extractValue` are always of the operator class's input type, and all key values must be of the class's `STORAGE` type. The type of the `query` argument passed to `extractQuery` and `consistent` is whatever is specified as the right-hand input type of the class member operator identified by the strategy number. This need not be the same as the item type, so long as key values of the correct type can be extracted from it.

55.3. Implementation

Internally, a GIN index contains a B-tree index constructed over keys, where each key is an element of one or more indexed items (a member of an array, for example) and where each tuple in a leaf page contains either a pointer to a B-tree of heap pointers (a "posting tree"), or a simple list of heap pointers (a "posting list") when the list is small enough to fit into a single index tuple along with the key value.

As of PostgreSQL 9.1, NULL key values can be included in the index. Also, placeholder NULLs are included in the index for indexed items that are NULL or contain no keys

according to `extractValue`. This allows searches that should find empty items to do so.

Multicolumn GIN indexes are implemented by building a single B-tree over composite values (column number, key value). The key values for different columns can be of different types.

55.3.1. GIN Fast Update Technique

Updating a GIN index tends to be slow because of the intrinsic nature of inverted indexes: inserting or updating one heap row can cause many inserts into the index (one for each key extracted from the indexed item). As of PostgreSQL 8.4, GIN is capable of postponing much of this work by inserting new tuples into a temporary, unsorted list of pending entries. When the table is vacuumed, or if the pending list becomes too large (larger than work_mem), the entries are moved to the main GIN data structure using the same bulk insert techniques used during initial index creation. This greatly improves GIN index update speed, even counting the additional vacuum overhead. Moreover the overhead work can be done by a background process instead of in foreground query processing.

The main disadvantage of this approach is that searches must scan the list of pending entries in addition to searching the regular index, and so a large list of pending entries will slow searches significantly. Another disadvantage is that, while most updates are fast, an update that causes the pending list to become "too large" will incur an immediate cleanup cycle and thus be much slower than other updates. Proper use of autovacuum can minimize both of these problems.

If consistent response time is more important than update speed, use of pending entries can be disabled by turning off the `FASTUPDATE` storage parameter for a GIN index. See CREATE INDEX for details.

55.3.2. Partial Match Algorithm

GIN can support "partial match" queries, in which the query does not determine an exact match for one or more keys, but the possible matches fall within a reasonably narrow range of key values (within the key sorting order determined by the `compare` support method). The `extractQuery` method, instead of returning a key value to be matched exactly, returns a key value that is the lower bound of the range to be searched, and sets the `pmatch` flag true. The key range is then scanned using the `comparePartial` method. `comparePartial` must return zero for a matching index key, less than zero for a non-match that is still within the range to be searched, or greater than zero if the index key is past the range that could match.

55.4. GIN Tips and Tricks

Create vs. insert

Insertion into a GIN index can be slow due to the likelihood of many keys being inserted for each item. So, for bulk insertions into a table it is advisable to drop the GIN index and recreate it after finishing bulk insertion.

As of PostgreSQL 8.4, this advice is less necessary since delayed indexing is used (see 第 55.3.1 节 for details). But for very large updates it may still be best to drop and recreate the index.

maintenance_work_mem

Build time for a GIN index is very sensitive to the `maintenance_work_mem` setting; it doesn't pay to skimp on work memory during index creation.

work_mem

During a series of insertions into an existing GIN index that has `FASTUPDATE` enabled, the system will clean up the pending-entry list whenever the list grows larger than `work_mem`. To avoid fluctuations in observed response time, it's desirable to have pending-list cleanup occur in the background (i.e., via autovacuum). Foreground cleanup operations can be avoided by increasing `work_mem` or making autovacuum more aggressive. However, enlarging `work_mem` means that if a foreground cleanup does occur, it will take even longer.

gin_fuzzy_search_limit

The primary goal of developing GIN indexes was to create support for highly scalable full-text search in PostgreSQL, and there are often situations when a full-text search returns a very large set of results. Moreover, this often happens when the query contains very frequent words, so that the large result set is not even useful. Since reading many tuples from the disk and sorting them could take a lot of time, this is unacceptable for production. (Note that the index search itself is very fast.)

To facilitate controlled execution of such queries, GIN has a configurable soft upper limit on the number of rows returned: the `gin_fuzzy_search_limit` configuration parameter. It is set to 0 (meaning no limit) by default. If a non-zero limit is set, then the returned set is a subset of the whole result set, chosen at random.

"Soft" means that the actual number of returned results could differ somewhat from the specified limit, depending on the query and the quality of the system's random number generator.

From experience, values in the thousands (e.g., 5000 – 20000) work well.

55.5. Limitations

GIN assumes that indexable operators are strict. This means that `extractValue` will not be called at all on a NULL item value (instead, a placeholder index entry is created automatically), and `extractQuery` will not be called on a NULL query value either (instead, the query is presumed to be unsatisfiable). Note however that NULL key values contained within a non-null composite item or query value are supported.

55.6. Examples

The PostgreSQL source distribution includes GIN operator classes for `tsvector` and for one-dimensional arrays of all internal types. Prefix searching in `tsvector` is implemented using the GIN partial match feature. The following `contrib` modules also contain GIN operator classes:

`btree_gin`

B-tree equivalent functionality for several data types

`hstore`

Module for storing (key, value) pairs

`intarray`

Enhanced support for `int[]`

`pg_trgm`

Text similarity using trigram matching

第 56 章 Database Physical Storage

This chapter provides an overview of the physical storage format used by PostgreSQL databases.

56.1. Database File Layout

This section describes the storage format at the level of files and directories.

All the data needed for a database cluster is stored within the cluster's data directory, commonly referred to as `PGDATA` (after the name of the environment variable that can be used to define it). A common location for `PGDATA` is `/var/lib/pgsql/data` . Multiple clusters, managed by different server instances, can exist on the same machine.

The `PGDATA` directory contains several subdirectories and control files, as shown in 表 56.1. In addition to these required items, the cluster configuration files `postgresql.conf`, `pg_hba.conf`, and `pg_ident.conf` are traditionally stored in `PGDATA` (although in PostgreSQL 8.0 and later, it is possible to keep them elsewhere).

表 56.1. Contents of PGDATA

Item	Description
`PG_VERSION`	A file containing the major version number of PostgreSQL
`base`	Subdirectory containing per-database subdir ectories
`global`	Subdirectory containing cluster-wide tables, such as `pg_database`
`pg_clog`	Subdirectory containing transaction commit status data
`pg_multixact`	Subdirectory containing multitransaction status data (used for shared row locks)
`pg_notify`	Subdirectory containing LISTEN/NOTIFY status data
`pg_serial`	Subdirectory containing information about committed serializable transactions
`pg_snapshots`	Subdirectory containing exported snapshots
`pg_stat_tmp`	Subdirectory containing temporary files for the statistics subsystem
`pg_subtrans`	Subdirectory containing subtransaction status data
`pg_tblspc`	Subdirectory containing symbolic links to tablespaces
`pg_twophase`	Subdirectory containing state files for prepar ed transactions
`pg_xlog`	Subdirectory containing WAL (Write Ahead Log) files
`postmaster.opts`	A file recording the command-line options the server was last started with
`postmaster.pid`	A lock file recording the current postmaster process ID (PID), cluster data directory path, postmaster start timestamp, port number,

Item	Description
	Unix-domain socket directory path (empty on Windows), first valid listen_address (IP addres s or `*`, or empty if not listening on TCP), and shared memory segment ID (this file is not present after server shutdown)

For each database in the cluster there is a subdirectory within `PGDATA/base`, named after the database's OID in `pg_database`. This subdirectory is the default location for the database's files; in particular, its system catalogs are stored there.

Each table and index is stored in a separate file. For ordinary relations, these files are named after the table or index's filenode number, which can be found in `pg_class.relfilenode`. But for temporary relations, the file name is of the form `t`*`BBB`*`_`*`FFF`*, where *`BBB`* is the backend ID of the backend which created the file, and *`FFF`* is the filenode number. In either case, in addition to the main file (a/k/a main fork), each table and index has a free space map (see 第 56.3 节), which stores information about free space available in the relation. The free space map is stored in a file named with the filenode number plus the suffix `_fsm`. Tables also have a visibility map, stored in a fork with the suffix `_vm`, to track which pages are known to have no dead tuples. The visibility map is described further in 第 56.4 节. Unlogged tables and indexes have a third fork, known as the initialization fork, which is stored in a fork with the suffix `_init` (see 第 56.5 节).

> **小心**
>
> Note that while a table's filenode often matches its OID, this is not necessarily the case; some operations, like `TRUNCATE`, `REINDEX`, `CLUSTER` and some forms of `ALTER TABLE`, can change the filenode while preserving the OID. Avoid assuming that filenode and table OID are the same. Also, for certain system catalogs including `pg_class` itself, `pg_class.relfilenode` contains zero. The actual filenode number of these catalogs is stored in a lower-level data structure, and can be obtained using the `pg_relation_filenode()` function.

When a table or index exceeds 1 GB, it is divided into gigabyte-sized segments. The first segment's file name is the same as the filenode; subsequent segments are named filenode.1, filenode.2, etc. This arrangement avoids problems on platforms that have file size limitations. (Actually, 1 GB is just the default segment size. The segment size can be adjusted using the configuration option `--with-segsize` when building PostgreSQL.) In principle, free space map and visibility map forks could require multiple segments as well, though this is unlikely to happen in practice.

A table that has columns with potentially large entries will have an associated TOAST table, which is used for out-of-line storage of field values that are too large to keep in the table rows proper. `pg_class.reltoastrelid` links from a table to its TOAST table, if any. See 第 56.2 节 for more information.

The contents of tables and indexes are discussed further in 第 56.6 节.

Tablespaces make the scenario more complicated. Each user-defined tablespace has a symbolic link inside the `PGDATA/pg_tblspc` directory, which points to the physical tablespace directory (i.e., the location specified in the tablespace's `CREATE TABLESPACE` command). This symbolic link is named after the tablespace's OID. Inside the physical tablespace directory there is a subdirectory with a name that depends on the PostgreSQL server version, such as `PG_9.0_201008051`. (The reason for using this subdirectory is so that successive versions of the database can use the same `CREATE`

TABLESPACE location value without conflicts.) Within the version-specific subdirectory, there is a subdirectory for each database that has elements in the tablespace, named after the database's OID. Tables and indexes are stored within that directory, using the filenode naming scheme. The pg_default tablespace is not accessed through pg_tblspc, but corresponds to PGDATA/base. Similarly, the pg_global tablespace is not accessed through pg_tblspc, but corresponds to PGDATA/global.

The pg_relation_filepath() function shows the entire path (relative to PGDATA) of any relation. It is often useful as a substitute for remembering many of the above rules. But keep in mind that this function just gives the name of the first segment of the main fork of the relation – you may need to append a segment number and/or _fsm, _vm, or _init to find all the files associated with the relation.

Temporary files (for operations such as sorting more data than can fit in memory) are created within PGDATA/base/pgsql_tmp, or within a pgsql_tmp subdirectory of a tablespace directory if a tablespace other than pg_default is specified for them. The name of a temporary file has the form pgsql_tmp*PPP*.*NNN*, where *PPP* is the PID of the owning backend and *NNN* distinguishes different temporary files of that backend.

56.2. TOAST

This section provides an overview of TOAST (The Oversized-Attribute Storage Technique).

PostgreSQL uses a fixed page size (commonly 8 kB), and does not allow tuples to span multiple pages. Therefore, it is not possible to store very large field values directly. To overcome this limitation, large field values are compressed and/or broken up into multiple physical rows. This happens transparently to the user, with only small impact on most of the backend code. The technique is affectionately known as TOAST (or "the best thing since sliced bread").

Only certain data types support TOAST – there is no need to impose the overhead on data types that cannot produce large field values. To support TOAST, a data type must have a variable-length (varlena) representation, in which the first 32-bit word of any stored value contains the total length of the value in bytes (including itself). TOAST does not constrain the rest of the representation. All the C-level functions supporting a TOAST-able data type must be careful to handle TOASTed input values. (This is normally done by invoking PG_DETOAST_DATUM before doing anything with an input value, but in some cases more efficient approaches are possible.)

TOAST usurps two bits of the varlena length word (the high-order bits on big-endian machines, the low-order bits on little-endian machines), thereby limiting the logical size of any value of a TOAST-able data type to 1 GB (2^{30} - 1 bytes). When both bits are zero, the value is an ordinary un-TOASTed value of the data type, and the remaining bits of the length word give the total datum size (including length word) in bytes. When the highest-order or lowest-order bit is set, the value has only a single-byte header instead of the normal four-byte header, and the remaining bits give the total datum size (including length byte) in bytes. As a special case, if the remaining bits are all zero (which would be impossible for a self-inclusive length), the value is a pointer to out-of-line data stored in a separate TOAST table. (The size of a TOAST pointer is given in the second byte of the datum.) Values with single-byte headers aren't aligned on any particular boundary, either. Lastly, when the highest-order or lowest-order bit is clear but the adjacent bit is set, the content of the datum has been compressed and must be decompressed before use. In this case the remaining bits of the length word give the total size of the compressed datum, not the original data. Note that compression is also possible for out-of-line data but the varlena header does not tell whether it has occurred – the content of the TOAST pointer tells that, instead.

If any of the columns of a table are TOAST-able, the table will have an associated TOAST table, whose OID is stored in the table's `pg_class.reltoastrelid` entry. Out-of-line TOASTed values are kept in the TOAST table, as described in more detail below.

The compression technique used is a fairly simple and very fast member of the LZ family of compression techniques. See `src/backend/utils/adt/pg_lzcompress.c` for the details.

Out-of-line values are divided (after compression if used) into chunks of at most `TOAST_MAX_CHUNK_SIZE` bytes (by default this value is chosen so that four chunk rows will fit on a page, making it about 2000 bytes). Each chunk is stored as a separate row in the TOAST table for the owning table. Every TOAST table has the columns `chunk_id` (an OID identifying the particular TOASTed value), `chunk_seq` (a sequence number for the chunk within its value), and `chunk_data` (the actual data of the chunk). A unique index on `chunk_id` and `chunk_seq` provides fast retrieval of the values. A pointer datum representing an out-of-line TOASTed value therefore needs to store the OID of the TOAST table in which to look and the OID of the specific value (its `chunk_id`). For convenience, pointer datums also store the logical datum size (original uncompressed data length) and actual stored size (different if compression was applied). Allowing for the varlena header bytes, the total size of a TOAST pointer datum is therefore 18 bytes regardless of the actual size of the represented value.

The TOAST code is triggered only when a row value to be stored in a table is wider than `TOAST_TUPLE_THRESHOLD` bytes (normally 2 kB). The TOAST code will compress and/or move field values out-of-line until the row value is shorter than `TOAST_TUPLE_TARGET` bytes (also normally 2 kB) or no more gains can be had. During an UPDATE operation, values of unchanged fields are normally preserved as-is; so an UPDATE of a row with out-of-line values incurs no TOAST costs if none of the out-of-line values change.

The TOAST code recognizes four different strategies for storing TOAST-able columns:

- `PLAIN` prevents either compression or out-of-line storage; furthermore it disables use of single-byte headers for varlena types. This is the only possible strategy for columns of non-TOAST-able data types.
- `EXTENDED` allows both compression and out-of-line storage. This is the default for most TOAST-able data types. Compression will be attempted first, then out-of-line storage if the row is still too big.
- `EXTERNAL` allows out-of-line storage but not compression. Use of `EXTERNAL` will make substring operations on wide `text` and `bytea` columns faster (at the penalty of increased storage space) because these operations are optimized to fetch only the required parts of the out-of-line value when it is not compressed.
- `MAIN` allows compression but not out-of-line storage. (Actually, out-of-line storage will still be performed for such columns, but only as a last resort when there is no other way to make the row small enough to fit on a page.)

Each TOAST-able data type specifies a default strategy for columns of that data type, but the strategy for a given table column can be altered with `ALTER TABLE SET STORAGE`.

This scheme has a number of advantages compared to a more straightforward approach such as allowing row values to span pages. Assuming that queries are usually qualified by comparisons against relatively small key values, most of the work of the executor will be done using the main row entry. The big values of TOASTed attributes will only be pulled out (if selected at all) at the time the result set is sent to the client. Thus, the main table is much smaller and more of its rows fit in the shared

buffer cache than would be the case without any out-of-line storage. Sort sets shrink also, and sorts will more often be done entirely in memory. A little test showed that a table containing typical HTML pages and their URLs was stored in about half of the raw data size including the TOAST table, and that the main table contained only about 10% of the entire data (the URLs and some small HTML pages). There was no run time difference compared to an un-TOASTed comparison table, in which all the HTML pages were cut down to 7 kB to fit.

56.3. Free Space Map

Each heap and index relation, except for hash indexes, has a Free Space Map (FSM) to keep track of available space in the relation. It's stored alongside the main relation data in a separate relation fork, named after the filenode number of the relation, plus a `_fsm` suffix. For example, if the filenode of a relation is 12345, the FSM is stored in a file called `12345_fsm`, in the same directory as the main relation file.

The Free Space Map is organized as a tree of FSM pages. The bottom level FSM pages store the free space available on each heap (or index) page, using one byte to represent each such page. The upper levels aggregate information from the lower levels.

Within each FSM page is a binary tree, stored in an array with one byte per node. Each leaf node represents a heap page, or a lower level FSM page. In each non-leaf node, the higher of its children's values is stored. The maximum value in the leaf nodes is therefore stored at the root.

See `src/backend/storage/freespace/README` for more details on how the FSM is structured, and how it's updated and searched. The pg_freespacemap module can be used to examine the information stored in free space maps.

56.4. Visibility Map

Each heap relation has a Visibility Map (VM) to keep track of which pages contain only tuples that are known to be visible to all active transactions. It's stored alongside the main relation data in a separate relation fork, named after the filenode number of the relation, plus a `_vm` suffix. For example, if the filenode of a relation is 12345, the VM is stored in a file called `12345_vm`, in the same directory as the main relation file. Note that indexes do not have VMs.

The visibility map simply stores one bit per heap page. A set bit means that all tuples on the page are known to be visible to all transactions. This means that the page does not contain any tuples that need to be vacuumed. This information can also be used by index-only scans to answer queries using only the index tuple.

The map is conservative in the sense that we make sure that whenever a bit is set, we know the condition is true, but if a bit is not set, it might or might not be true. Visibility map bits are only set by vacuum, but are cleared by any data-modifying operations on a page.

56.5. The Initialization Fork

Each unlogged table, and each index on an unlogged table, has an initialization fork. The initialization fork is an empty table or index of the appropriate type. When an unlogged table must be reset to empty due to a crash, the initialization fork is copied over the main fork, and any other forks are erased (they will be recreated automatically as needed).

56.6. Database Page Layout

This section provides an overview of the page format used within PostgreSQL tables and indexes.[1] Sequences and TOAST tables are formatted just like a regular table.

In the following explanation, a byte is assumed to contain 8 bits. In addition, the term item refers to an individual data value that is stored on a page. In a table, an item is a row; in an index, an item is an index entry.

Every table and index is stored as an array of pages of a fixed size (usually 8 kB, although a different page size can be selected when compiling the server). In a table, all the pages are logically equivalent, so a particular item (row) can be stored in any page. In indexes, the first page is generally reserved as a metapage holding control information, and there can be different types of pages within the index, depending on the index access method.

表 56.2 shows the overall layout of a page. There are five parts to each page.

表 56.2. Overall Page Layout

Item	Description
PageHeaderData	24 bytes long. Contains general information about the page, including free space pointe rs.
ItemIdData	Array of (offset,length) pairs pointing to the actual items. 4 bytes per item.
Free space	The unallocated space. New item pointers are allocated from the start of this area, new items from the end.
Items	The actual items themselves.
Special space	Index access method specific data. Different methods store different data. Empty in ordina ry tables.

The first 24 bytes of each page consists of a page header (PageHeaderData). Its format is detailed in 表 56.3. The first two fields track the most recent WAL entry related to this page. Next is a 2-byte field containing flag bits. This is followed by three 2-byte integer fields (`pd_lower`, `pd_upper`, and `pd_special`). These contain byte offsets from the page start to the start of unallocated space, to the end of unallocated space, and to the start of the special space. The next 2 bytes of the page header, `pd_pagesize_version`, store both the page size and a version indicator. Beginning with PostgreSQL 8.3 the version number is 4; PostgreSQL 8.1 and 8.2 used version number 3; PostgreSQL 8.0 used version number 2; PostgreSQL 7.3 and 7.4 used version number 1; prior releases used version number 0. (The basic page layout and header format has not changed in most of these versions, but the layout of heap row headers has.) The page size is basically only present as a cross-check; there is no support for having more than one page size in an installation. The last field is a hint that shows whether pruning the page is likely to be profitable: it tracks the oldest un-pruned XMAX on the page.

表 56.3. PageHeaderData Layout

Field	Type	Length	Description
pd_lsn	XLogRecPtr	8 bytes	LSN: next byte after last byte of xlog

[1] Actually, index access methods need not use this page format. All the existing index methods do use this basic format, but the data kept on index metapages usually doesn't follow the item layout rules.

Field	Type	Length	Description
			record for last change to this page
pd_tli	uint16	2 bytes	TimeLineID of last change (only its lowest 16 bits)
pd_flags	uint16	2 bytes	Flag bits
pd_lower	LocationIndex	2 bytes	Offset to start of free space
pd_upper	LocationIndex	2 bytes	Offset to end of free space
pd_special	LocationIndex	2 bytes	Offset to start of special space
pd_pagesize_version	uint16	2 bytes	Page size and layout version number information
pd_prune_xid	TransactionId	4 bytes	Oldest unpruned XMAX on page, or zero if none

All the details can be found in `src/include/storage/bufpage.h`.

Following the page header are item identifiers (`ItemIdData`), each requiring four bytes. An item identifier contains a byte-offset to the start of an item, its length in bytes, and a few attribute bits which affect its interpretation. New item identifiers are allocated as needed from the beginning of the unallocated space. The number of item identifiers present can be determined by looking at `pd_lower`, which is increased to allocate a new identifier. Because an item identifier is never moved until it is freed, its index can be used on a long-term basis to reference an item, even when the item itself is moved around on the page to compact free space. In fact, every pointer to an item (`ItemPointer`, also known as `CTID`) created by PostgreSQL consists of a page number and the index of an item identifier.

The items themselves are stored in space allocated backwards from the end of unallocated space. The exact structure varies depending on what the table is to contain. Tables and sequences both use a structure named `HeapTupleHeaderData`, described below.

The final section is the “special section” which can contain anything the access method wishes to store. For example, b-tree indexes store links to the page's left and right siblings, as well as some other data relevant to the index structure. Ordinary tables do not use a special section at all (indicated by setting `pd_special` to equal the page size).

All table rows are structured in the same way. There is a fixed-size header (occupying 23 bytes on most machines), followed by an optional null bitmap, an optional object ID field, and the user data. The header is detailed in 表 56.4. The actual user data (columns of the row) begins at the offset indicated by `t_hoff`, which must always be a multiple of the MAXALIGN distance for the platform. The null bitmap is only present if the HEAP_HASNULL bit is set in `t_infomask`. If it is present it begins just after the fixed header and occupies enough bytes to have one bit per data column (that is, `t_natts` bits altogether). In this list of bits, a 1 bit indicates not-null, a 0 bit is a null. When the bitmap is not present, all columns are assumed not-null. The object ID is only present if the HEAP_HASOID bit is set in `t_infomask`. If present, it appears just before the `t_hoff` boundary. Any padding needed to make `t_hoff` a MAXALIGN multiple will appear between the null bitmap and the object ID. (This in turn ensures that the object ID is suitably aligned.)

表 56.4. HeapTupleHeaderData Layout

Field	Type	Length	Description
t_xmin	TransactionId	4 bytes	insert XID stamp
t_xmax	TransactionId	4 bytes	delete XID stamp
t_cid	CommandId	4 bytes	insert and/or delete CID stamp (overlays with t_xvac)
t_xvac	TransactionId	4 bytes	XID for VACUUM operation moving a row version
t_ctid	ItemPointerData	6 bytes	current TID of this or newer row version
t_infomask2	uint16	2 bytes	number of attributes, plus various flag bits
t_infomask	uint16	2 bytes	various flag bits
t_hoff	uint8	1 byte	offset to user data

All the details can be found in `src/include/access/htup.h`.

Interpreting the actual data can only be done with information obtained from other tables, mostly `pg_attribute`. The key values needed to identify field locations are `attlen` and `attalign`. There is no way to directly get a particular attribute, except when there are only fixed width fields and no null values. All this trickery is wrapped up in the functions heap_getattr, fastgetattr and heap_getsysattr.

To read the data you need to examine each attribute in turn. First check whether the field is NULL according to the null bitmap. If it is, go to the next. Then make sure you have the right alignment. If the field is a fixed width field, then all the bytes are simply placed. If it's a variable length field (attlen = -1) then it's a bit more complicated. All variable-length data types share the common header structure `struct varlena`, which includes the total length of the stored value and some flag bits. Depending on the flags, the data can be either inline or in a TOAST table; it might be compressed, too (see 第 56.2 节).

第 57 章 BKI 后端接口

后端接口（BKI）文件是使用一种特殊语言编写的脚本，PostgreSQL 后端在“引导”模式下运行时能够理解这种语言。引导模式允许从零开始创建并填充系统目录，而普通 SQL 命令要求目录已经存在。因此，BKI 文件可以用于最初创建数据库系统。（它们可能也没有其他用途。）

initdb 在创建新的数据库集簇时，使用 BKI 文件完成部分工作。initdb 使用的输入文件作为构建和安装 PostgreSQL 的一部分，由名为 `genbki.pl` 的程序创建。该程序读取源码树 `src/include/catalog/` 目录中采用特殊格式的 C 头文件。创建的 BKI 文件名为 `postgres.bki`，通常安装在安装树的 `share` 子目录中。

相关信息可以在 initdb 的文档中找到。

57.1. BKI 文件格式

本节描述 PostgreSQL 后端如何解释 BKI 文件。如果手边有一份 `postgres.bki` 文件作为示例，这段说明会更容易理解。

BKI 输入由一系列命令组成。根据命令语法的不同，每条命令由若干词元组成。词元通常由空白分隔，但如果不会产生歧义，也可以不分隔。没有专门的命令分隔符；语法上不可能属于前一条命令的下一个词元，就会开始一条新命令。（通常为了清晰，你会把新命令写在新的一行。）词元可以是某些关键字、特殊字符（圆括号、逗号等）、数字，或者双引号字符串。所有内容都区分大小写。

以 `#` 开头的行会被忽略。

57.2. BKI 命令

`create` *`tablename`* *`tableoid`* [`bootstrap`] [`shared_relation`] [`without_oids`] [`rowtype_oid` *`oid`*] (*`name1`* = *`type1`* [, *`name2`* = *`type2`*, ...])

创建一个名为 *`tablename`*、OID 为 *`tableoid`* 的表，其列定义在括号中给出。

`bootstrap.c` 直接支持以下列类型：`bool`、`bytea`、`char`（1 字节）、`name`、`int2`、`int4`、`regproc`、`regclass`、`regtype`、`text`、`oid`、`tid`、`xid`、`cid`、`int2vector`、`oidvector`、`_int4`（数组）、`_text`（数组）、`_oid`（数组）、`_char`（数组）以及 `_aclitem`（数组）。虽然也可以创建包含其他类型列的表，但这必须等到 `pg_type` 创建完成并填入合适条目之后才能做到。（实际上，这意味着引导目录中只能使用这些列类型，而非引导目录则可以包含任何内置类型。）

指定 `bootstrap` 时，表只会在磁盘上创建；不会在 `pg_class`、`pg_attribute` 等目录中为它写入任何信息。因此，在通过 `insert` 命令以较为原始的方式补入这些条目之前，普通 SQL 操作无法访问该表。这个选项用于创建 `pg_class` 等目录自身。

如果指定了 `shared_relation`，该表就会作为共享表创建。除非指定 `without_oids`，否则该表将具有 OID。该表的行类型 OID（即 `pg_type` 中对应的 OID）还可以通过 `rowtype_oid` 子句显式指定；如果不指定，就会自动为其生成一个 OID。（如果指定了 `bootstrap`，那么 `rowtype_oid` 子句其实没有用，但仍然可以写上，作为文档说明。）

`open` *`tablename`*

打开名为 *`tablename`* 的表，以便插入数据。当前任何已打开的表都会被关闭。

`close` [*`tablename`*]

关闭当前打开的表。可以给出表名进行交叉校验，但并非必须。

`insert [OID = oid_value] ( value1 value2 ... )`

向当前打开的表插入一行新数据，使用 *value1*、*value2* 等作为各列的值，使用 *oid_value* 作为其 OID。如果 *oid_value* 为零（0）或省略该子句，并且该表具有 OID，则分配下一个可用 OID。

NULL 值可以用特殊关键字 `_null_` 表示。包含空格的值必须用双引号括起来。

`declare [unique] index indexname indexoid on tablename using amname ( opclass1 name1 [, ...] )`

在名为 *tablename* 的表上，使用 *amname* 访问方法，创建一个名为 *indexname*、OID 为 *indexoid* 的索引。要建立索引的字段分别称为 *name1*、*name2* 等；使用的操作符类分别为 *opclass1*、*opclass2* 等。该命令会创建索引文件并写入相应的目录条目，但不会初始化索引内容。

`declare toast toasttableoid toastindexoid on tablename`

为名为 *tablename* 的表创建一个 TOAST 表。该 TOAST 表的 OID 设为 *toasttableoid*，其索引的 OID 设为 *toastindexoid*。与 `declare index` 一样，索引填充会被延后。

`build indices`

填充之前已声明的索引。

57.3. 引导 BKI 文件的结构

在 `open` 命令可以使用之前，它所依赖的那些表必须已经存在，并且其中已经有了将要被打开之表对应的条目。（这些最基本的表包括 `pg_class`、`pg_attribute`、`pg_proc` 和 `pg_type`。）为了让这些表本身也能够被填充，带 `bootstrap` 选项的 `create` 命令会隐式打开刚创建的表，以便插入数据。

同样，`declare index` 和 `declare toast` 命令也必须等到它们所需的系统目录已经创建并填充完毕之后才能使用。

因此，`postgres.bki` 文件的结构必须如下：

1. 对某个关键表执行 `create bootstrap`
2. 用 `insert` 插入至少足以描述这些关键表的数据
3. `close`
4. 对其他关键表重复上述步骤。
5. 对某个非关键表执行 `create`（不带 `bootstrap`）
6. `open`
7. 用 `insert` 插入所需数据
8. `close`
9. 对其他非关键表重复上述步骤。
10. 定义索引和 TOAST 表。
11. `build indices`

无疑还存在其他未文档化的顺序依赖。

57.4. 示例

以下命令序列会创建表test_table，其 OID 为 420，包含两列cola和colb，类型分别为int4和text，并向表中插入两行：

```
create test_table 420 (cola = int4, colb = text)
open test_table
insert OID=421 ( 1 "value1" )
insert OID=422 ( 2 _null_ )
close test_table
```

第 58 章 规划器如何使用统计信息

本章建立在第 14.1 节和第 14.2 节所介绍的内容之上，进一步说明规划器如何利用系统统计信息来估计查询各部分可能返回的行数。这是规划过程中的重要组成部分，
为代价计算提供了大量原始材料。

本章的目的不是详细说明代码，而是概述其工作方式。
这或许能让随后想要阅读代码的人更容易入门。

58.1. 行估计示例

下面的示例使用 PostgreSQL 回归测试数据库中的表。所示输出取自版本 8.3，更早（或更晚）的版本行为可能有所不同。另外，由于 ANALYZE 在生成统计信息时使用随机采样，每次重新执行 ANALYZE 后，结果都会略有变化。

让我们从一个很简单的查询开始：

```
EXPLAIN SELECT * FROM tenk1;

                         QUERY PLAN
-------------------------------------------------------------
 Seq Scan on tenk1  (cost=0.00..458.00 rows=10000 width=244)
```

关于规划器如何确定tenk1的基数，第 14.2 节中已经介绍过；这里为了完整起见再重复一次。页数和行数是从pg_class中查得的：

```
SELECT relpages, reltuples FROM pg_class WHERE relname = 'tenk1';

 relpages | reltuples
----------+-----------
      358 |     10000
```

这些数字反映的是该表最近一次VACUUM或ANALYZE时的情况。随后，
规划器会取得该表当前实际的页数（这是一项开销很小的操作，无需扫描全表）。
如果该值与relpages不同，就会相应地缩放reltuples，从而得到当前的行数估计。
在上面的示例中，relpages的值是最新的，因此行数估计与reltuples相同。

接着看一个包含范围条件的示例，其条件位于WHERE子句中：

```
EXPLAIN SELECT * FROM tenk1 WHERE unique1 < 1000;

                                    QUERY PLAN
-------------------------------------------------------------------
-------------
 Bitmap Heap Scan on tenk1  (cost=24.06..394.64 rows=1007 width=
244)
   Recheck Cond: (unique1 < 1000)
   ->  Bitmap Index Scan on tenk1_unique1  (cost=0.00..23.80 rows=
1007 width=0)
         Index Cond: (unique1 < 1000)
```

规划器检查WHERE子句中的条件，并查找操作符<的选择率函数，所在的系统目录是pg_operator。此函数保存在oprrest列中，本例的条目为scalarltsel。scalarltsel函数取得unique1的直方图，来源为pg_statistics。手动查询时，查看更简单的pg_stats视图更方便：

```
SELECT histogram_bounds FROM pg_stats
WHERE tablename='tenk1' AND attname='unique1';
```

```
                    histogram_bounds
-------------------------------------------------------
 {0,993,1997,3050,4040,5036,5957,7057,8029,9016,9995}
```

接下来，计算“< 1000”在直方图中所占的比例，这就是选择率。直方图把范围划分为等频率的桶，因此只需找到目标值所在的桶，计入该桶的部分和前面所有桶的全部。值 1000 显然在第二个桶（993-1997）中。假设每个桶内部的值呈线性分布，可以按如下方式计算选择率：

```
selectivity = (1 + (1000 - bucket[2].min)/(bucket[2].max - bucket[
2].min))/num_buckets
            = (1 + (1000 - 993)/(1997 - 993))/10
            = 0.100697
```

即一个完整的桶加上第二个桶中的线性比例，再除以桶数。用选择率乘以表的基数，即可计算估计行数。这里所用的表为tenk1：

```
rows = rel_cardinality * selectivity
     = 10000 * 0.100697
     = 1007  (rounding off)
```

接下来，考虑一个包含等值条件的示例，其条件位于WHERE子句中：

```
EXPLAIN SELECT * FROM tenk1 WHERE stringu1 = 'CRAAAA';

                        QUERY PLAN
----------------------------------------------------------
 Seq Scan on tenk1  (cost=0.00..483.00 rows=30 width=244)
   Filter: (stringu1 = 'CRAAAA'::name)
```

规划器同样会检查WHERE子句中的条件，查找操作符=的选择率函数，即eqsel。对于等值估计，直方图没有用；应该使用高频值（MCV）列表来确定选择率。查看 MCV，并同时查看后面会用到的其他一些列：

```
SELECT null_frac, n_distinct, most_common_vals, most_common_freqs
 FROM pg_stats
WHERE tablename='tenk1' AND attname='stringu1';

null_frac         | 0
n_distinct        | 676
most_common_vals  | {EJAAAA,BBAAAA,CRAAAA,FCAAAA,FEAAAA,GSAAAA,
JOAAAA,MCAAAA,NAAAAA,WGAAAA}
most_common_freqs | {0.00333333,0.003,0.003,0.003,0.003,0.003,0.
003,0.003,0.003,0.003}
```

由于CRAAAA出现在 MCV 列表中，选择率就是最常见值频率（MCF）列表中的对应条目：

```
selectivity = mcf[3]
            = 0.003
```

与之前一样，估计行数就是这个值与表的基数的乘积。这里所用的表为tenk1：

```
rows = 10000 * 0.003
     = 30
```

现在考虑同一个查询，但使用一个不在MCV列表中的常量：

```
EXPLAIN SELECT * FROM tenk1 WHERE stringu1 = 'xxx';

                        QUERY PLAN
```

```
-----------------------------------------------------------
 Seq Scan on tenk1  (cost=0.00..483.00 rows=15 width=244)
   Filter: (stringu1 = 'xxx'::name)
```

这是一个完全不同的问题：当值不在MCV列表中时，如何估计选择率。
方法是利用该值不在列表中的事实，再结合所有MCV的频率信息：

```
selectivity = (1 - sum(mvf))/(num_distinct - num_mcv)
            = (1 - (0.00333333 + 0.003 + 0.003 + 0.003 + 0.003 + 0.
003 +
                    0.003 + 0.003 + 0.003 + 0.003))/(676 - 10)
            = 0.0014559
```

也就是说，把所有MCV的频率加起来，用一减去这个总和，再除以其他非重复值的数量。
这相当于假设该列中不属于任何 MCV 的那部分值，均匀分布在其他所有非重复值上。注意，这里没有空值，所以不必考虑它们（否则还应从分子中减去空值比例）。
然后照常计算估计行数：

```
rows = 10000 * 0.0014559
     = 15  (rounding off)
```

前面条件为`unique1 < 1000`的示例，过度简化了`scalarltsel`的实际行为。现在已经看过 MCV 的使用示例，可以补充一些细节。前面的示例就其涉及的范围而言是正确的，因为`unique1`是唯一列，所以没有 MCV（显然，没有哪个值会比其他值更常见）。对于非唯一列，通常既有直方图，也有 MCV 列表，而且直方图不包括该列总体中由 MCV 表示的那一部分。这样处理可以使估计更准确。在这种情况下，`scalarltsel`直接将条件（例如“< 1000”）应用于 MCV 列表中的每个值，并将满足条件的 MCV 的频率相加。这可以精确估计表中 MCV 所代表部分的选择率。然后按前面所述的方法使用直方图，估计表中非 MCV 部分的选择率，再将两个值结合起来，估计整体选择率。例如，考虑以下查询：

```
EXPLAIN SELECT * FROM tenk1 WHERE stringu1 < 'IAAAAA';

                        QUERY PLAN
-------------------------------------------------------------
 Seq Scan on tenk1  (cost=0.00..483.00 rows=3077 width=244)
   Filter: (stringu1 < 'IAAAAA'::name)
```

我们已经看过`stringu1`的 MCV 信息，下面是它的直方图：

```
SELECT histogram_bounds FROM pg_stats
WHERE tablename='tenk1' AND attname='stringu1';

                                histogram_bounds
-----------------------------------------------------------------
-------------
 {AAAAAA,CQAAAA,FRAAAA,IBAAAA,KRAAAA,NFAAAA,PSAAAA,SGAAAA,VAAAAA,
XLAAAA,ZZAAAA}
```

检查 MCV 列表可以发现，条件`stringu1 < 'IAAAAA'`被前六个条目满足，而后四个条目不满足，因此总体中 MCV 部分的选择率为：

```
selectivity = sum(relevant mvfs)
            = 0.00333333 + 0.003 + 0.003 + 0.003 + 0.003 + 0.003
            = 0.01833333
```

将所有 MCF 相加，还可以得知 MCV 所代表的部分占总体的比例为 0.03033333，因此直方图所代表的部分占比为 0.96966667（这里同样没有空值，否则还需要将其排除）。可以看到，值`IAAAAA`接近第三个直方图桶的末尾。规划器对不同字符的频率作出一些粗略假设，
估计直方图所代表的总体中有 0.298387 的部分小于`IAAAAA`。然后将 MCV 和非 MCV 两部分的估计结合起来：

```
selectivity = mcv_selectivity + histogram_selectivity * histogram_
fraction
            = 0.01833333 + 0.298387 * 0.96966667
            = 0.307669

rows        = 10000 * 0.307669
            = 3077  (rounding off)
```

在这个特定的示例中，MCV 列表带来的修正相当小，因为该列的分布实际上相当均匀（统计信息显示这些特定值比其他值更常见，主要是采样误差造成的）。更常见的情况是，某些值明显比其他值更常见；这时，这个复杂过程能够有效提高准确性，因为最常见值的选择率是精确求得的。

现在考虑另一种情形，其中多个条件出现在WHERE子句中：

```
EXPLAIN SELECT * FROM tenk1 WHERE unique1 < 1000 AND stringu1 =
 'xxx';

                                   QUERY PLAN
-------------------------------------------------------------------
-------------
 Bitmap Heap Scan on tenk1  (cost=23.80..396.91 rows=1 width=244)
   Recheck Cond: (unique1 < 1000)
   Filter: (stringu1 = 'xxx'::name)
   ->  Bitmap Index Scan on tenk1_unique1  (cost=0.00..23.80 rows=
1007 width=0)
         Index Cond: (unique1 < 1000)
```

规划器假定这两个条件相互独立，因此可以将各个子句的选择率相乘：

```
selectivity = selectivity(unique1 < 1000) * selectivity(stringu1 =
 'xxx')
            = 0.100697 * 0.0014559
            = 0.0001466

rows        = 10000 * 0.0001466
            = 1  (rounding off)
```

注意，位图索引扫描的估计返回行数只反映用于索引的条件。这一点很重要，因为它会影响后续堆访问的代价估计。

最后，来看一个涉及连接的查询：

```
EXPLAIN SELECT * FROM tenk1 t1, tenk2 t2
WHERE t1.unique1 < 50 AND t1.unique2 = t2.unique2;

                                       QUERY PLAN
-------------------------------------------------------------------
-------------------
 Nested Loop  (cost=4.64..456.23 rows=50 width=488)
   ->  Bitmap Heap Scan on tenk1 t1  (cost=4.64..142.17 rows=50
 width=244)
         Recheck Cond: (unique1 < 50)
         ->  Bitmap Index Scan on tenk1_unique1  (cost=0.00..4.63
 rows=50 width=0)
               Index Cond: (unique1 < 50)
   ->  Index Scan using tenk2_unique2 on tenk2 t2  (cost=0.00..6.27
 rows=1 width=244)
         Index Cond: (unique2 = t1.unique2)
```

对于tenk1，unique1 < 50这一限制会在嵌套循环连接之前求值。处理方法与之前的范围查询示例类似。这次，值 50 落在unique1直方图的第一个桶中：

```
selectivity = (0 + (50 - bucket[1].min)/(bucket[1].max - bucket[1].
min))/num_buckets
            = (0 + (50 - 0)/(993 - 0))/10
            = 0.005035

rows        = 10000 * 0.005035
            = 50  (rounding off)
```

连接的限制条件是t2.unique2 = t1.unique2。操作符就是熟悉的=，不过选择率函数来自oprjoin列，该列属于pg_operator，本例中为eqjoinsel。eqjoinsel
查询两个表的统计信息，这两个表是tenk2和tenk1：

```
SELECT tablename, null_frac,n_distinct, most_common_vals FROM pg_
stats
WHERE tablename IN ('tenk1', 'tenk2') AND attname='unique2';

tablename  | null_frac | n_distinct | most_common_vals
-----------+-----------+------------+------------------
 tenk1     |         0 |         -1 |
 tenk2     |         0 |         -1 |
```

本例中，没有MCV信息可用于unique2，因为所有值看起来都是唯一的。
因此使用的算法仅依赖于两个关系的非重复值数量及其空值比例：

```
selectivity = (1 - null_frac1) * (1 - null_frac2) * min(1/num_
distinct1, 1/num_distinct2)
            = (1 - 0) * (1 - 0) / max(10000, 10000)
            = 0.0001
```

也就是说，对两个关系分别用一减去空值比例，再除以两者非重复值数量中的较大者。
预计连接输出的行数，通过将两个输入的笛卡尔积基数乘以选择率来计算：

```
rows = (outer_cardinality * inner_cardinality) * selectivity
     = (50 * 10000) * 0.0001
     = 50
```

如果这两列存在 MCV 列表，eqjoinsel就会通过直接比较 MCV 列表，确定由 MCV 表示的那部分列值总体中的连接选择率。剩余部分总体的估计则沿用这里展示的相同方法。

请注意，我们把inner_cardinality写成 10000，也就是tenk2未经修改的大小。单看EXPLAIN输出，似乎连接行数估计来自 50 * 1，也就是外表行数乘以对tenk2执行每次内表索引扫描得到的估计行数。
但事实并非如此：连接关系的大小是在考虑任何具体连接计划之前估计的。如果一切正常，这两种连接大小估算方式会得到大致相同的答案，但由于舍入误差和其他因素，它们有时会出现明显偏差。

如果想了解更多细节，表大小（在任何WHERE子句之前）的估计是在src/backend/optimizer/util/plancat.c中完成的。子句选择率的一般逻辑位于src/backend/optimizer/path/clausesel.c。按操作符区分的选择率函数大多位于src/backend/utils/adt/selfuncs.c中。

58.2. 规划器统计信息与安全

对表pg_statistic的访问仅限于超级用户，
这样普通用户就无法借此了解其他用户表中的内容。
有些选择率估算函数会使用用户提供的操作符（查询中出现的操作符或与之相关的操作符）

来分析已存储的统计信息。例如，为了判断某个已存储的高频值是否适用，
选择率估算器必须运行合适的=操作符，将查询中的常量与已存储的值进行比较。因此，pg_statistic中的数据有可能被传递给用户定义的操作符。
精心构造的操作符可以故意泄露传给它的操作数（例如记录到日志中，或写入另一张表），
也可能无意中通过在错误消息中显示其值而泄露它们；无论哪种情况，都可能把pg_statistic中的数据暴露给本不应看到这些数据的用户。

为了防止这种情况，以下规则适用于所有内置选择率估算函数。在规划查询时，
若要使用已存储的统计信息，当前用户必须对该表或相关列具有SELECT权限，
或者所用操作符必须是LEAKPROOF（更准确地说，是该操作符所基于的函数必须如此）。
否则，选择率估算器会像不存在可用统计信息一样工作，
规划器则会按默认或后备假设继续。

如果用户对表或列不具备所需权限，那么在许多情况下，
查询最终都会收到一个权限被拒绝的错误，因此这一机制在实际中是不可见的。
但如果用户是通过安全屏障视图读取数据，
规划器可能会想要检查一个用户本来无权访问的底层表的统计信息。在这种情况下，
操作符应当是 leakproof，否则统计信息不会被使用。对此没有直接反馈，除非计划可能不是最优的。如果怀疑发生了这种情况，可以尝试让权限更高的用户运行该查询，
看看是否会得到不同的计划。

此限制只适用于规划器需要对来自pg_statistic
的一个或多个值执行用户定义操作符的情况。因此，不论访问权限如何，
规划器仍可使用通用统计信息，例如空值比例或列中非重复值的数量。

第三方扩展中的选择率估算函数如果可能会用用户定义的操作符处理统计信息，
也应遵循同样的安全规则。可参考 PostgreSQL 源代码。

部分 VIII. 附录

目录

A. PostgreSQL错误代码 1670
B. 日期/时间支持 1678
B.1. 日期/时间输入解释 1678
B.2. 日期/时间关键字 1679
B.3. 日期/时间配置文件 1680
B.4. 历法的历史 1681
C. SQL 关键字 1683
D. SQL 符合性 1703
D.1. 支持的特性 1704
D.2. 不支持的特性 1717
E. 发布说明 1730
E.1. Release 9.2.24 1730
E.1.1. Migration to Version 9.2.24 1730
E.1.2. Changes 1730
E.2. Release 9.2.23 1731
E.2.1. Migration to Version 9.2.23 1732
E.2.2. Changes 1732
E.3. Release 9.2.22 1733
E.3.1. Migration to Version 9.2.22 1733
E.3.2. Changes 1734
E.4. Release 9.2.21 1737
E.4.1. Migration to Version 9.2.21 1737
E.4.2. Changes 1737
E.5. Release 9.2.20 1740
E.5.1. Migration to Version 9.2.20 1740
E.5.2. Changes 1740
E.6. Release 9.2.19 1743
E.6.1. Migration to Version 9.2.19 1743
E.6.2. Changes 1743
E.7. Release 9.2.18 1745
E.7.1. Migration to Version 9.2.18 1745
E.7.2. Changes 1745
E.8. Release 9.2.17 1748
E.8.1. Migration to Version 9.2.17 1748
E.8.2. Changes 1748
E.9. Release 9.2.16 1749
E.9.1. Migration to Version 9.2.16 1749
E.9.2. Changes 1749
E.10. Release 9.2.15 1750
E.10.1. Migration to Version 9.2.15 1751
E.10.2. Changes 1751
E.11. Release 9.2.14 1754
E.11.1. Migration to Version 9.2.14 1754
E.11.2. Changes 1755
E.12. Release 9.2.13 1758
E.12.1. Migration to Version 9.2.13 1758
E.12.2. Changes 1759
E.13. Release 9.2.12 1759
E.13.1. Migration to Version 9.2.12 1759
E.13.2. Changes 1759
E.14. Release 9.2.11 1760
E.14.1. Migration to Version 9.2.11 1760
E.14.2. Changes 1760
E.15. Release 9.2.10 1764
E.15.1. Migration to Version 9.2.10 1764

E.15.2. Changes 1764
E.16. Release 9.2.9 1771
E.16.1. Migration to Version 9.2.9 1771
E.16.2. Changes 1771
E.17. Release 9.2.8 1775
E.17.1. Migration to Version 9.2.8 1775
E.17.2. Changes 1775
E.18. Release 9.2.7 1776
E.18.1. Migration to Version 9.2.7 1776
E.18.2. Changes 1776
E.19. Release 9.2.6 1781
E.19.1. Migration to Version 9.2.6 1781
E.19.2. Changes 1781
E.20. Release 9.2.5 1783
E.20.1. Migration to Version 9.2.5 1783
E.20.2. Changes 1783
E.21. Release 9.2.4 1786
E.21.1. Migration to Version 9.2.4 1786
E.21.2. Changes 1786
E.22. Release 9.2.3 1789
E.22.1. Migration to Version 9.2.3 1789
E.22.2. Changes 1789
E.23. Release 9.2.2 1792
E.23.1. Migration to Version 9.2.2 1792
E.23.2. Changes 1792
E.24. Release 9.2.1 1797
E.24.1. Migration to Version 9.2.1 1797
E.24.2. Changes 1798
E.25. Release 9.2 1799
E.25.1. Overview 1799
E.25.2. Migration to Version 9.2 1799
E.25.3. Changes 1802
E.26. Release 9.1.24 1817
E.26.1. Migration to Version 9.1.24 1817
E.26.2. Changes 1817
E.27. Release 9.1.23 1818
E.27.1. Migration to Version 9.1.23 1819
E.27.2. Changes 1819
E.28. Release 9.1.22 1821
E.28.1. Migration to Version 9.1.22 1821
E.28.2. Changes 1821
E.29. Release 9.1.21 1822
E.29.1. Migration to Version 9.1.21 1822
E.29.2. Changes 1822
E.30. Release 9.1.20 1824
E.30.1. Migration to Version 9.1.20 1824
E.30.2. Changes 1824
E.31. Release 9.1.19 1827
E.31.1. Migration to Version 9.1.19 1827
E.31.2. Changes 1828
E.32. Release 9.1.18 1831
E.32.1. Migration to Version 9.1.18 1831
E.32.2. Changes 1831
E.33. Release 9.1.17 1832
E.33.1. Migration to Version 9.1.17 1832
E.33.2. Changes 1832
E.34. Release 9.1.16 1832
E.34.1. Migration to Version 9.1.16 1833

E.34.2. Changes 1833
E.35. Release 9.1.15 1837
E.35.1. Migration to Version 9.1.15 1837
E.35.2. Changes 1837
E.36. Release 9.1.14 1843
E.36.1. Migration to Version 9.1.14 1843
E.36.2. Changes 1843
E.37. Release 9.1.13 1846
E.37.1. Migration to Version 9.1.13 1846
E.37.2. Changes 1846
E.38. Release 9.1.12 1847
E.38.1. Migration to Version 9.1.12 1847
E.38.2. Changes 1847
E.39. Release 9.1.11 1851
E.39.1. Migration to Version 9.1.11 1851
E.39.2. Changes 1851
E.40. Release 9.1.10 1853
E.40.1. Migration to Version 9.1.10 1853
E.40.2. Changes 1853
E.41. Release 9.1.9 1855
E.41.1. Migration to Version 9.1.9 1855
E.41.2. Changes 1856
E.42. Release 9.1.8 1858
E.42.1. Migration to Version 9.1.8 1858
E.42.2. Changes 1858
E.43. Release 9.1.7 1860
E.43.1. Migration to Version 9.1.7 1860
E.43.2. Changes 1860
E.44. Release 9.1.6 1863
E.44.1. Migration to Version 9.1.6 1864
E.44.2. Changes 1864
E.45. Release 9.1.5 1865
E.45.1. Migration to Version 9.1.5 1865
E.45.2. Changes 1865
E.46. Release 9.1.4 1868
E.46.1. Migration to Version 9.1.4 1868
E.46.2. Changes 1868
E.47. Release 9.1.3 1872
E.47.1. Migration to Version 9.1.3 1872
E.47.2. Changes 1872
E.48. Release 9.1.2 1877
E.48.1. Migration to Version 9.1.2 1877
E.48.2. Changes 1877
E.49. Release 9.1.1 1881
E.49.1. Migration to Version 9.1.1 1881
E.49.2. Changes 1881
E.50. Release 9.1 1882
E.50.1. Overview 1882
E.50.2. Migration to Version 9.1 1882
E.50.3. Changes 1885
E.51. Release 9.0.23 1900
E.51.1. Migration to Version 9.0.23 1900
E.51.2. Changes 1900
E.52. Release 9.0.22 1903
E.52.1. Migration to Version 9.0.22 1903
E.52.2. Changes 1903
E.53. Release 9.0.21 1904
E.53.1. Migration to Version 9.0.21 1904

E.53.2. Changes 1904
E.54. Release 9.0.20 1905
E.54.1. Migration to Version 9.0.20 1905
E.54.2. Changes 1905
E.55. Release 9.0.19 1908
E.55.1. Migration to Version 9.0.19 1908
E.55.2. Changes 1908
E.56. Release 9.0.18 1914
E.56.1. Migration to Version 9.0.18 1914
E.56.2. Changes 1914
E.57. Release 9.0.17 1916
E.57.1. Migration to Version 9.0.17 1916
E.57.2. Changes 1916
E.58. Release 9.0.16 1917
E.58.1. Migration to Version 9.0.16 1918
E.58.2. Changes 1918
E.59. Release 9.0.15 1921
E.59.1. Migration to Version 9.0.15 1921
E.59.2. Changes 1922
E.60. Release 9.0.14 1923
E.60.1. Migration to Version 9.0.14 1923
E.60.2. Changes 1923
E.61. Release 9.0.13 1925
E.61.1. Migration to Version 9.0.13 1925
E.61.2. Changes 1925
E.62. Release 9.0.12 1927
E.62.1. Migration to Version 9.0.12 1927
E.62.2. Changes 1927
E.63. Release 9.0.11 1929
E.63.1. Migration to Version 9.0.11 1929
E.63.2. Changes 1929
E.64. Release 9.0.10 1932
E.64.1. Migration to Version 9.0.10 1932
E.64.2. Changes 1932
E.65. Release 9.0.9 1933
E.65.1. Migration to Version 9.0.9 1933
E.65.2. Changes 1933
E.66. Release 9.0.8 1935
E.66.1. Migration to Version 9.0.8 1935
E.66.2. Changes 1935
E.67. Release 9.0.7 1937
E.67.1. Migration to Version 9.0.7 1937
E.67.2. Changes 1937
E.68. Release 9.0.6 1941
E.68.1. Migration to Version 9.0.6 1941
E.68.2. Changes 1941
E.69. Release 9.0.5 1944
E.69.1. Migration to Version 9.0.5 1944
E.69.2. Changes 1944
E.70. Release 9.0.4 1948
E.70.1. Migration to Version 9.0.4 1948
E.70.2. Changes 1949
E.71. Release 9.0.3 1951
E.71.1. Migration to Version 9.0.3 1951
E.71.2. Changes 1951
E.72. Release 9.0.2 1952
E.72.1. Migration to Version 9.0.2 1952
E.72.2. Changes 1952

E.73. Release 9.0.1 1955
E.73.1. Migration to Version 9.0.1 1955
E.73.2. Changes 1955
E.74. Release 9.0 1956
E.74.1. Overview 1956
E.74.2. Migration to Version 9.0 1957
E.74.3. Changes 1960
E.75. Release 8.4.22 1977
E.75.1. Migration to Version 8.4.22 1978
E.75.2. Changes 1978
E.76. Release 8.4.21 1980
E.76.1. Migration to Version 8.4.21 1980
E.76.2. Changes 1980
E.77. Release 8.4.20 1981
E.77.1. Migration to Version 8.4.20 1981
E.77.2. Changes 1981
E.78. Release 8.4.19 1984
E.78.1. Migration to Version 8.4.19 1984
E.78.2. Changes 1985
E.79. Release 8.4.18 1986
E.79.1. Migration to Version 8.4.18 1986
E.79.2. Changes 1986
E.80. Release 8.4.17 1987
E.80.1. Migration to Version 8.4.17 1987
E.80.2. Changes 1987
E.81. Release 8.4.16 1989
E.81.1. Migration to Version 8.4.16 1989
E.81.2. Changes 1989
E.82. Release 8.4.15 1990
E.82.1. Migration to Version 8.4.15 1990
E.82.2. Changes 1990
E.83. Release 8.4.14 1992
E.83.1. Migration to Version 8.4.14 1992
E.83.2. Changes 1993
E.84. Release 8.4.13 1993
E.84.1. Migration to Version 8.4.13 1993
E.84.2. Changes 1993
E.85. Release 8.4.12 1995
E.85.1. Migration to Version 8.4.12 1995
E.85.2. Changes 1995
E.86. Release 8.4.11 1997
E.86.1. Migration to Version 8.4.11 1997
E.86.2. Changes 1997
E.87. Release 8.4.10 2000
E.87.1. Migration to Version 8.4.10 2000
E.87.2. Changes 2000
E.88. Release 8.4.9 2002
E.88.1. Migration to Version 8.4.9 2002
E.88.2. Changes 2002
E.89. Release 8.4.8 2006
E.89.1. Migration to Version 8.4.8 2006
E.89.2. Changes 2006
E.90. Release 8.4.7 2007
E.90.1. Migration to Version 8.4.7 2008
E.90.2. Changes 2008
E.91. Release 8.4.6 2009
E.91.1. Migration to Version 8.4.6 2009
E.91.2. Changes 2009

E.92. Release 8.4.5 2011
E.92.1. Migration to Version 8.4.5 2011
E.92.2. Changes 2011
E.93. Release 8.4.4 2014
E.93.1. Migration to Version 8.4.4 2015
E.93.2. Changes 2015
E.94. Release 8.4.3 2017
E.94.1. Migration to Version 8.4.3 2017
E.94.2. Changes 2017
E.95. Release 8.4.2 2020
E.95.1. Migration to Version 8.4.2 2020
E.95.2. Changes 2020
E.96. Release 8.4.1 2023
E.96.1. Migration to Version 8.4.1 2023
E.96.2. Changes 2024
E.97. Release 8.4 2025
E.97.1. Overview 2025
E.97.2. Migration to Version 8.4 2026
E.97.3. Changes 2029
E.98. Release 8.3.23 2046
E.98.1. Migration to Version 8.3.23 2046
E.98.2. Changes 2046
E.99. Release 8.3.22 2047
E.99.1. Migration to Version 8.3.22 2047
E.99.2. Changes 2048
E.100. Release 8.3.21 2049
E.100.1. Migration to Version 8.3.21 2050
E.100.2. Changes 2050
E.101. Release 8.3.20 2050
E.101.1. Migration to Version 8.3.20 2050
E.101.2. Changes 2050
E.102. Release 8.3.19 2052
E.102.1. Migration to Version 8.3.19 2052
E.102.2. Changes 2052
E.103. Release 8.3.18 2054
E.103.1. Migration to Version 8.3.18 2054
E.103.2. Changes 2054
E.104. Release 8.3.17 2056
E.104.1. Migration to Version 8.3.17 2056
E.104.2. Changes 2056
E.105. Release 8.3.16 2058
E.105.1. Migration to Version 8.3.16 2058
E.105.2. Changes 2058
E.106. Release 8.3.15 2060
E.106.1. Migration to Version 8.3.15 2060
E.106.2. Changes 2061
E.107. Release 8.3.14 2061
E.107.1. Migration to Version 8.3.14 2062
E.107.2. Changes 2062
E.108. Release 8.3.13 2063
E.108.1. Migration to Version 8.3.13 2063
E.108.2. Changes 2063
E.109. Release 8.3.12 2064
E.109.1. Migration to Version 8.3.12 2065
E.109.2. Changes 2065
E.110. Release 8.3.11 2067
E.110.1. Migration to Version 8.3.11 2067
E.110.2. Changes 2067

E.111. Release 8.3.10 2069
E.111.1. Migration to Version 8.3.10 2069
E.111.2. Changes 2069
E.112. Release 8.3.9 2071
E.112.1. Migration to Version 8.3.9 2071
E.112.2. Changes 2071
E.113. Release 8.3.8 2073
E.113.1. Migration to Version 8.3.8 2073
E.113.2. Changes 2074
E.114. Release 8.3.7 2075
E.114.1. Migration to Version 8.3.7 2075
E.114.2. Changes 2075
E.115. Release 8.3.6 2077
E.115.1. Migration to Version 8.3.6 2077
E.115.2. Changes 2077
E.116. Release 8.3.5 2079
E.116.1. Migration to Version 8.3.5 2079
E.116.2. Changes 2079
E.117. Release 8.3.4 2081
E.117.1. Migration to Version 8.3.4 2081
E.117.2. Changes 2081
E.118. Release 8.3.3 2083
E.118.1. Migration to Version 8.3.3 2083
E.118.2. Changes 2083
E.119. Release 8.3.2 2084
E.119.1. Migration to Version 8.3.2 2084
E.119.2. Changes 2084
E.120. Release 8.3.1 2086
E.120.1. Migration to Version 8.3.1 2086
E.120.2. Changes 2086
E.121. Release 8.3 2088
E.121.1. Overview 2088
E.121.2. Migration to Version 8.3 2089
E.121.3. Changes 2092
E.122. Release 8.2.23 2105
E.122.1. Migration to Version 8.2.23 2105
E.122.2. Changes 2105
E.123. Release 8.2.22 2107
E.123.1. Migration to Version 8.2.22 2107
E.123.2. Changes 2107
E.124. Release 8.2.21 2109
E.124.1. Migration to Version 8.2.21 2109
E.124.2. Changes 2109
E.125. Release 8.2.20 2110
E.125.1. Migration to Version 8.2.20 2110
E.125.2. Changes 2110
E.126. Release 8.2.19 2111
E.126.1. Migration to Version 8.2.19 2111
E.126.2. Changes 2111
E.127. Release 8.2.18 2113
E.127.1. Migration to Version 8.2.18 2113
E.127.2. Changes 2113
E.128. Release 8.2.17 2115
E.128.1. Migration to Version 8.2.17 2115
E.128.2. Changes 2115
E.129. Release 8.2.16 2116
E.129.1. Migration to Version 8.2.16 2116
E.129.2. Changes 2116

E.130. Release 8.2.15 2118
E.130.1. Migration to Version 8.2.15 2118
E.130.2. Changes 2118
E.131. Release 8.2.14 2120
E.131.1. Migration to Version 8.2.14 2120
E.131.2. Changes 2120
E.132. Release 8.2.13 2121
E.132.1. Migration to Version 8.2.13 2122
E.132.2. Changes 2122
E.133. Release 8.2.12 2123
E.133.1. Migration to Version 8.2.12 2123
E.133.2. Changes 2123
E.134. Release 8.2.11 2124
E.134.1. Migration to Version 8.2.11 2124
E.134.2. Changes 2124
E.135. Release 8.2.10 2125
E.135.1. Migration to Version 8.2.10 2125
E.135.2. Changes 2125
E.136. Release 8.2.9 2127
E.136.1. Migration to Version 8.2.9 2127
E.136.2. Changes 2127
E.137. Release 8.2.8 2127
E.137.1. Migration to Version 8.2.8 2128
E.137.2. Changes 2128
E.138. Release 8.2.7 2129
E.138.1. Migration to Version 8.2.7 2129
E.138.2. Changes 2129
E.139. Release 8.2.6 2131
E.139.1. Migration to Version 8.2.6 2131
E.139.2. Changes 2131
E.140. Release 8.2.5 2133
E.140.1. Migration to Version 8.2.5 2133
E.140.2. Changes 2133
E.141. Release 8.2.4 2134
E.141.1. Migration to Version 8.2.4 2134
E.141.2. Changes 2134
E.142. Release 8.2.3 2135
E.142.1. Migration to Version 8.2.3 2135
E.142.2. Changes 2135
E.143. Release 8.2.2 2135
E.143.1. Migration to Version 8.2.2 2135
E.143.2. Changes 2135
E.144. Release 8.2.1 2136
E.144.1. Migration to Version 8.2.1 2136
E.144.2. Changes 2136
E.145. Release 8.2 2137
E.145.1. Overview 2137
E.145.2. Migration to Version 8.2 2138
E.145.3. Changes 2140
E.146. Release 8.1.23 2152
E.146.1. Migration to Version 8.1.23 2152
E.146.2. Changes 2152
E.147. Release 8.1.22 2154
E.147.1. Migration to Version 8.1.22 2154
E.147.2. Changes 2154
E.148. Release 8.1.21 2155
E.148.1. Migration to Version 8.1.21 2155
E.148.2. Changes 2155

E.149. Release 8.1.20 2156
E.149.1. Migration to Version 8.1.20 2157
E.149.2. Changes 2157
E.150. Release 8.1.19 2158
E.150.1. Migration to Version 8.1.19 2158
E.150.2. Changes 2158
E.151. Release 8.1.18 2159
E.151.1. Migration to Version 8.1.18 2159
E.151.2. Changes 2159
E.152. Release 8.1.17 2160
E.152.1. Migration to Version 8.1.17 2160
E.152.2. Changes 2160
E.153. Release 8.1.16 2161
E.153.1. Migration to Version 8.1.16 2161
E.153.2. Changes 2161
E.154. Release 8.1.15 2162
E.154.1. Migration to Version 8.1.15 2162
E.154.2. Changes 2162
E.155. Release 8.1.14 2163
E.155.1. Migration to Version 8.1.14 2163
E.155.2. Changes 2163
E.156. Release 8.1.13 2164
E.156.1. Migration to Version 8.1.13 2165
E.156.2. Changes 2165
E.157. Release 8.1.12 2165
E.157.1. Migration to Version 8.1.12 2165
E.157.2. Changes 2165
E.158. Release 8.1.11 2167
E.158.1. Migration to Version 8.1.11 2167
E.158.2. Changes 2167
E.159. Release 8.1.10 2169
E.159.1. Migration to Version 8.1.10 2169
E.159.2. Changes 2169
E.160. Release 8.1.9 2169
E.160.1. Migration to Version 8.1.9 2170
E.160.2. Changes 2170
E.161. Release 8.1.8 2170
E.161.1. Migration to Version 8.1.8 2170
E.161.2. Changes 2170
E.162. Release 8.1.7 2170
E.162.1. Migration to Version 8.1.7 2170
E.162.2. Changes 2171
E.163. Release 8.1.6 2171
E.163.1. Migration to Version 8.1.6 2171
E.163.2. Changes 2171
E.164. Release 8.1.5 2172
E.164.1. Migration to Version 8.1.5 2172
E.164.2. Changes 2172
E.165. Release 8.1.4 2173
E.165.1. Migration to Version 8.1.4 2173
E.165.2. Changes 2173
E.166. Release 8.1.3 2175
E.166.1. Migration to Version 8.1.3 2175
E.166.2. Changes 2175
E.167. Release 8.1.2 2176
E.167.1. Migration to Version 8.1.2 2176
E.167.2. Changes 2176
E.168. Release 8.1.1 2177

E.168.1. Migration to Version 8.1.1 2178
E.168.2. Changes 2178
E.169. Release 8.1 2178
E.169.1. Overview 2179
E.169.2. Migration to Version 8.1 2180
E.169.3. Additional Changes 2183
E.170. Release 8.0.26 2193
E.170.1. Migration to Version 8.0.26 2194
E.170.2. Changes 2194
E.171. Release 8.0.25 2195
E.171.1. Migration to Version 8.0.25 2195
E.171.2. Changes 2195
E.172. Release 8.0.24 2196
E.172.1. Migration to Version 8.0.24 2196
E.172.2. Changes 2196
E.173. Release 8.0.23 2197
E.173.1. Migration to Version 8.0.23 2198
E.173.2. Changes 2198
E.174. Release 8.0.22 2199
E.174.1. Migration to Version 8.0.22 2199
E.174.2. Changes 2199
E.175. Release 8.0.21 2200
E.175.1. Migration to Version 8.0.21 2200
E.175.2. Changes 2200
E.176. Release 8.0.20 2200
E.176.1. Migration to Version 8.0.20 2200
E.176.2. Changes 2201
E.177. Release 8.0.19 2201
E.177.1. Migration to Version 8.0.19 2201
E.177.2. Changes 2201
E.178. Release 8.0.18 2202
E.178.1. Migration to Version 8.0.18 2202
E.178.2. Changes 2202
E.179. Release 8.0.17 2203
E.179.1. Migration to Version 8.0.17 2203
E.179.2. Changes 2203
E.180. Release 8.0.16 2203
E.180.1. Migration to Version 8.0.16 2203
E.180.2. Changes 2203
E.181. Release 8.0.15 2205
E.181.1. Migration to Version 8.0.15 2205
E.181.2. Changes 2205
E.182. Release 8.0.14 2206
E.182.1. Migration to Version 8.0.14 2207
E.182.2. Changes 2207
E.183. Release 8.0.13 2207
E.183.1. Migration to Version 8.0.13 2207
E.183.2. Changes 2207
E.184. Release 8.0.12 2208
E.184.1. Migration to Version 8.0.12 2208
E.184.2. Changes 2208
E.185. Release 8.0.11 2208
E.185.1. Migration to Version 8.0.11 2208
E.185.2. Changes 2208
E.186. Release 8.0.10 2209
E.186.1. Migration to Version 8.0.10 2209
E.186.2. Changes 2209
E.187. Release 8.0.9 2209

E.187.1. Migration to Version 8.0.9 2209
E.187.2. Changes 2209
E.188. Release 8.0.8 2210
E.188.1. Migration to Version 8.0.8 2210
E.188.2. Changes 2210
E.189. Release 8.0.7 2211
E.189.1. Migration to Version 8.0.7 2211
E.189.2. Changes 2211
E.190. Release 8.0.6 2212
E.190.1. Migration to Version 8.0.6 2212
E.190.2. Changes 2213
E.191. Release 8.0.5 2213
E.191.1. Migration to Version 8.0.5 2213
E.191.2. Changes 2214
E.192. Release 8.0.4 2214
E.192.1. Migration to Version 8.0.4 2214
E.192.2. Changes 2214
E.193. Release 8.0.3 2216
E.193.1. Migration to Version 8.0.3 2216
E.193.2. Changes 2216
E.194. Release 8.0.2 2217
E.194.1. Migration to Version 8.0.2 2217
E.194.2. Changes 2217
E.195. Release 8.0.1 2219
E.195.1. Migration to Version 8.0.1 2219
E.195.2. Changes 2219
E.196. Release 8.0 2220
E.196.1. Overview 2220
E.196.2. Migration to Version 8.0 2221
E.196.3. Deprecated Features 2223
E.196.4. Changes 2223
E.197. Release 7.4.30 2236
E.197.1. Migration to Version 7.4.30 2236
E.197.2. Changes 2236
E.198. Release 7.4.29 2237
E.198.1. Migration to Version 7.4.29 2237
E.198.2. Changes 2237
E.199. Release 7.4.28 2238
E.199.1. Migration to Version 7.4.28 2238
E.199.2. Changes 2239
E.200. Release 7.4.27 2239
E.200.1. Migration to Version 7.4.27 2239
E.200.2. Changes 2239
E.201. Release 7.4.26 2240
E.201.1. Migration to Version 7.4.26 2240
E.201.2. Changes 2240
E.202. Release 7.4.25 2241
E.202.1. Migration to Version 7.4.25 2241
E.202.2. Changes 2241
E.203. Release 7.4.24 2242
E.203.1. Migration to Version 7.4.24 2242
E.203.2. Changes 2242
E.204. Release 7.4.23 2242
E.204.1. Migration to Version 7.4.23 2242
E.204.2. Changes 2242
E.205. Release 7.4.22 2243
E.205.1. Migration to Version 7.4.22 2243
E.205.2. Changes 2243

E.206. Release 7.4.21 2243
E.206.1. Migration to Version 7.4.21 2243
E.206.2. Changes 2243
E.207. Release 7.4.20 2244
E.207.1. Migration to Version 7.4.20 2244
E.207.2. Changes 2244
E.208. Release 7.4.19 2245
E.208.1. Migration to Version 7.4.19 2245
E.208.2. Changes 2245
E.209. Release 7.4.18 2246
E.209.1. Migration to Version 7.4.18 2246
E.209.2. Changes 2246
E.210. Release 7.4.17 2246
E.210.1. Migration to Version 7.4.17 2247
E.210.2. Changes 2247
E.211. Release 7.4.16 2247
E.211.1. Migration to Version 7.4.16 2247
E.211.2. Changes 2247
E.212. Release 7.4.15 2247
E.212.1. Migration to Version 7.4.15 2248
E.212.2. Changes 2248
E.213. Release 7.4.14 2248
E.213.1. Migration to Version 7.4.14 2248
E.213.2. Changes 2248
E.214. Release 7.4.13 2249
E.214.1. Migration to Version 7.4.13 2249
E.214.2. Changes 2249
E.215. Release 7.4.12 2250
E.215.1. Migration to Version 7.4.12 2250
E.215.2. Changes 2250
E.216. Release 7.4.11 2250
E.216.1. Migration to Version 7.4.11 2251
E.216.2. Changes 2251
E.217. Release 7.4.10 2251
E.217.1. Migration to Version 7.4.10 2251
E.217.2. Changes 2251
E.218. Release 7.4.9 2252
E.218.1. Migration to Version 7.4.9 2252
E.218.2. Changes 2252
E.219. Release 7.4.8 2253
E.219.1. Migration to Version 7.4.8 2253
E.219.2. Changes 2254
E.220. Release 7.4.7 2255
E.220.1. Migration to Version 7.4.7 2255
E.220.2. Changes 2255
E.221. Release 7.4.6 2256
E.221.1. Migration to Version 7.4.6 2256
E.221.2. Changes 2256
E.222. Release 7.4.5 2257
E.222.1. Migration to Version 7.4.5 2257
E.222.2. Changes 2257
E.223. Release 7.4.4 2257
E.223.1. Migration to Version 7.4.4 2257
E.223.2. Changes 2257
E.224. Release 7.4.3 2258
E.224.1. Migration to Version 7.4.3 2258
E.224.2. Changes 2258
E.225. Release 7.4.2 2259

E.225.1. Migration to Version 7.4.2 2259
E.225.2. Changes 2260
E.226. Release 7.4.1 2261
E.226.1. Migration to Version 7.4.1 2261
E.226.2. Changes 2261
E.227. Release 7.4 2262
E.227.1. Overview 2262
E.227.2. Migration to Version 7.4 2264
E.227.3. Changes 2265
E.228. Release 7.3.21 2278
E.228.1. Migration to Version 7.3.21 2278
E.228.2. Changes 2278
E.229. Release 7.3.20 2279
E.229.1. Migration to Version 7.3.20 2279
E.229.2. Changes 2279
E.230. Release 7.3.19 2279
E.230.1. Migration to Version 7.3.19 2279
E.230.2. Changes 2279
E.231. Release 7.3.18 2280
E.231.1. Migration to Version 7.3.18 2280
E.231.2. Changes 2280
E.232. Release 7.3.17 2280
E.232.1. Migration to Version 7.3.17 2280
E.232.2. Changes 2280
E.233. Release 7.3.16 2280
E.233.1. Migration to Version 7.3.16 2281
E.233.2. Changes 2281
E.234. Release 7.3.15 2281
E.234.1. Migration to Version 7.3.15 2281
E.234.2. Changes 2281
E.235. Release 7.3.14 2282
E.235.1. Migration to Version 7.3.14 2282
E.235.2. Changes 2282
E.236. Release 7.3.13 2282
E.236.1. Migration to Version 7.3.13 2283
E.236.2. Changes 2283
E.237. Release 7.3.12 2283
E.237.1. Migration to Version 7.3.12 2283
E.237.2. Changes 2283
E.238. Release 7.3.11 2284
E.238.1. Migration to Version 7.3.11 2284
E.238.2. Changes 2284
E.239. Release 7.3.10 2284
E.239.1. Migration to Version 7.3.10 2284
E.239.2. Changes 2285
E.240. Release 7.3.9 2286
E.240.1. Migration to Version 7.3.9 2286
E.240.2. Changes 2286
E.241. Release 7.3.8 2287
E.241.1. Migration to Version 7.3.8 2287
E.241.2. Changes 2287
E.242. Release 7.3.7 2287
E.242.1. Migration to Version 7.3.7 2287
E.242.2. Changes 2287
E.243. Release 7.3.6 2287
E.243.1. Migration to Version 7.3.6 2288
E.243.2. Changes 2288
E.244. Release 7.3.5 2288

E.244.1. Migration to Version 7.3.5 2288
E.244.2. Changes 2288
E.245. Release 7.3.4 2289
E.245.1. Migration to Version 7.3.4 2289
E.245.2. Changes 2289
E.246. Release 7.3.3 2290
E.246.1. Migration to Version 7.3.3 2290
E.246.2. Changes 2290
E.247. Release 7.3.2 2292
E.247.1. Migration to Version 7.3.2 2292
E.247.2. Changes 2292
E.248. Release 7.3.1 2293
E.248.1. Migration to Version 7.3.1 2293
E.248.2. Changes 2294
E.249. Release 7.3 2294
E.249.1. Overview 2294
E.249.2. Migration to Version 7.3 2295
E.249.3. Changes 2296
E.250. Release 7.2.8 2306
E.250.1. Migration to Version 7.2.8 2306
E.250.2. Changes 2306
E.251. Release 7.2.7 2307
E.251.1. Migration to Version 7.2.7 2307
E.251.2. Changes 2307
E.252. Release 7.2.6 2307
E.252.1. Migration to Version 7.2.6 2307
E.252.2. Changes 2307
E.253. Release 7.2.5 2308
E.253.1. Migration to Version 7.2.5 2308
E.253.2. Changes 2308
E.254. Release 7.2.4 2308
E.254.1. Migration to Version 7.2.4 2308
E.254.2. Changes 2308
E.255. Release 7.2.3 2309
E.255.1. Migration to Version 7.2.3 2309
E.255.2. Changes 2309
E.256. Release 7.2.2 2309
E.256.1. Migration to Version 7.2.2 2309
E.256.2. Changes 2309
E.257. Release 7.2.1 2310
E.257.1. Migration to Version 7.2.1 2310
E.257.2. Changes 2310
E.258. Release 7.2 2311
E.258.1. Overview 2311
E.258.2. Migration to Version 7.2 2311
E.258.3. Changes 2312
E.259. Release 7.1.3 2321
E.259.1. Migration to Version 7.1.3 2321
E.259.2. Changes 2321
E.260. Release 7.1.2 2321
E.260.1. Migration to Version 7.1.2 2321
E.260.2. Changes 2321
E.261. Release 7.1.1 2321
E.261.1. Migration to Version 7.1.1 2321
E.261.2. Changes 2322
E.262. Release 7.1 2322
E.262.1. Migration to Version 7.1 2323
E.262.2. Changes 2323

E.263. Release 7.0.3 2326
E.263.1. Migration to Version 7.0.3 2326
E.263.2. Changes 2326
E.264. Release 7.0.2 2327
E.264.1. Migration to Version 7.0.2 2327
E.264.2. Changes 2327
E.265. Release 7.0.1 2327
E.265.1. Migration to Version 7.0.1 2328
E.265.2. Changes 2328
E.266. Release 7.0 2328
E.266.1. Migration to Version 7.0 2329
E.266.2. Changes 2329
E.267. Release 6.5.3 2336
E.267.1. Migration to Version 6.5.3 2336
E.267.2. Changes 2336
E.268. Release 6.5.2 2336
E.268.1. Migration to Version 6.5.2 2336
E.268.2. Changes 2336
E.269. Release 6.5.1 2337
E.269.1. Migration to Version 6.5.1 2337
E.269.2. Changes 2337
E.270. Release 6.5 2337
E.270.1. Migration to Version 6.5 2338
E.270.2. Changes 2339
E.271. Release 6.4.2 2342
E.271.1. Migration to Version 6.4.2 2343
E.271.2. Changes 2343
E.272. Release 6.4.1 2343
E.272.1. Migration to Version 6.4.1 2343
E.272.2. Changes 2343
E.273. Release 6.4 2344
E.273.1. Migration to Version 6.4 2344
E.273.2. Changes 2344
E.274. Release 6.3.2 2348
E.274.1. Changes 2348
E.275. Release 6.3.1 2349
E.275.1. Changes 2349
E.276. Release 6.3 2350
E.276.1. Migration to Version 6.3 2351
E.276.2. Changes 2351
E.277. Release 6.2.1 2355
E.277.1. Migration from version 6.2 to version 6.2.1 2355
E.277.2. Changes 2355
E.278. Release 6.2 2355
E.278.1. Migration from version 6.1 to version 6.2 2356
E.278.2. Migration from version 1.x to version 6.2 2356
E.278.3. Changes 2356
E.279. Release 6.1.1 2358
E.279.1. Migration from version 6.1 to version 6.1.1 2358
E.279.2. Changes 2358
E.280. Release 6.1 2359
E.280.1. Migration to Version 6.1 2359
E.280.2. Changes 2359
E.281. Release 6.0 2361
E.281.1. Migration from version 1.09 to version 6.0 2361
E.281.2. Migration from pre-1.09 to version 6.0 2361
E.281.3. Changes 2362
E.282. Release 1.09 2364

E.283. Release 1.02 2364
E.283.1. Migration from version 1.02 to version 1.02.1 2364
E.283.2. Dump/Reload Procedure 2364
E.283.3. Changes 2365
E.284. Release 1.01 2365
E.284.1. Migration from version 1.0 to version 1.01 2365
E.284.2. Changes 2367
E.285. Release 1.0 2368
E.285.1. Changes 2368
E.286. Postgres95 Release 0.03 2369
E.286.1. Changes 2369
E.287. Postgres95 Release 0.02 2372
E.287.1. Changes 2372
E.288. Postgres95 Release 0.01 2372
F. 额外提供的模块与扩展 2373
F.1. adminpack 2374
F.2. auth_delay 2376
F.2.1. 配置参数 2376
F.2.2. 作者 2376
F.3. auto_explain 2377
F.3.1. Configuration Parameters 2377
F.3.2. Example 2378
F.3.3. Author 2379
F.4. btree_gin 2380
F.4.1. 用法示例 2380
F.4.2. 作者 2380
F.5. btree_gist 2381
F.5.1. 用法示例 2381
F.5.2. 作者 2382
F.6. chkpass 2383
F.6.1. 作者 2383
F.7. citext – 大小写不敏感的字符串类型 2384
F.7.1. 原理 2384
F.7.2. 如何使用 2384
F.7.3. 字符串比较行为 2384
F.7.4. 限制 2385
F.7.5. 作者 2385
F.8. cube – 多维立方体数据类型 2386
F.8.1. 语法 2386
F.8.2. 精度 2386
F.8.3. 用法 2386
F.8.4. 默认规则 2387
F.8.5. 注意 2388
F.8.6. 致谢 2388
F.9. dblink 2389
F.10. dict_int – 用于整数的示例全文搜索词典 2421
F.10.1. 配置 2421
F.10.2. 用法 2421
F.11. dict_xsyn – 示例同义词全文检索词典 2422
F.11.1. 配置 2422
F.11.2. 用法 2422
F.12. dummy_seclabel 2424
F.12.1. 缘由 2424
F.12.2. 用法 2424
F.12.3. 作者 2424
F.13. earthdistance – 计算大圆距离 2425
F.13.1. 基于立方体的地球距离 2425
F.13.2. 基于点的地球距离 2426

F.14. file_fdw 2427
F.15. fuzzystrmatch – 确定字符串的相似性和距离 2429
F.15.1. Soundex 2429
F.15.2. Levenshtein 2429
F.15.3. Metaphone 2430
F.15.4. Double Metaphone 2430
F.16. hstore 2432
F.16.1. `hstore` External Representation 2432
F.16.2. `hstore` Operators and Functions 2432
F.16.3. Indexes 2435
F.16.4. Examples 2435
F.16.5. Statistics 2436
F.16.6. Compatibility 2436
F.16.7. Authors 2437
F.17. intagg – 整数聚合器和枚举器 2438
F.17.1. 函数 2438
F.17.2. 使用示例 2438
F.18. intarray 2440
F.18.1. `intarray` 函数和操作符 2440
F.18.2. 索引支持 2441
F.18.3. 示例 2441
F.18.4. 基准测试 2442
F.18.5. 作者 2442
F.19. isn 2443
F.19.1. 数据类型 2443
F.19.2. 类型转换 2443
F.19.3. 函数和操作符 2444
F.19.4. 示例 2444
F.19.5. 参考文献 2445
F.19.6. 作者 2446
F.20. lo – 管理大对象 2447
F.20.1. 原理 2447
F.20.2. 如何使用 2447
F.20.3. 限制 2447
F.20.4. 作者 2447
F.21. ltree 2448
F.21.1. 定义 2448
F.21.2. 操作符和函数 2449
F.21.3. 索引 2451
F.21.4. 示例 2451
F.21.5. 作者 2453
F.22. pageinspect 2454
F.22.1. 函数 2454
F.23. passwordcheck 2456
F.24. pg_buffercache 2457
F.24.1. `pg_buffercache` 视图 2457
F.24.2. 示例输出 2457
F.24.3. 作者 2458
F.25. pgcrypto 2459
F.25.1. General Hashing Functions 2459
F.25.2. Password Hashing Functions 2459
F.25.3. PGP Encryption Functions 2461
F.25.4. Raw Encryption Functions 2466
F.25.5. Random-Data Functions 2467
F.25.6. Notes 2467
F.25.7. Author 2469
F.26. pg_freespacemap 2470
F.26.1. 函数 2470

F.26.2. 示例输出 2470
F.26.3. 作者 2471
F.27. pgrowlocks – 显示表的行锁信息 2472
F.27.1. 概述 2472
F.27.2. 示例输出 2472
F.27.3. 作者 2472
F.28. pg_stat_statements 2473
F.28.1. The `pg_stat_statements` View 2473
F.28.2. Functions 2475
F.28.3. Configuration Parameters 2475
F.28.4. Sample Output 2475
F.28.5. Authors 2476
F.29. pgstattuple – 获取元组级统计信息 2477
F.29.1. 函数 2477
F.29.2. 作者 2478
F.30. pg_trgm 2479
F.30.1. 三字符组（Trigram 或 Trigraph）概念 2479
F.30.2. 函数和操作符 2479
F.30.3. 索引支持 2479
F.30.4. 文本检索集成 2480
F.30.5. 参考 2481
F.30.6. 作者 2481
F.31. seg – 用于线段或浮点区间的数据类型 2482
F.31.1. 原理 2482
F.31.2. 语法 2482
F.31.3. 精度 2483
F.31.4. 用法 2483
F.31.5. 注意 2484
F.31.6. 致谢 2484
F.32. sepgsql 2485
F.32.1. Overview 2485
F.32.2. Installation 2485
F.32.3. Regression Tests 2486
F.32.4. GUC Parameters 2487
F.32.5. Features 2488
F.32.6. Sepgsql Functions 2491
F.32.7. Limitations 2491
F.32.8. External Resources 2492
F.32.9. Author 2492
F.33. spi 2493
F.33.1. refint – 用于实现引用完整性的函数 2493
F.33.2. timetravel – 实现时间旅行的函数 2493
F.33.3. autoinc – 用于字段自动递增的函数 2494
F.33.4. insert_username – 用于跟踪谁修改了表的函数 2494
F.33.5. moddatetime – 用于跟踪最后修改时间的函数 2494
F.34. sslinfo 2495
F.34.1. 提供的函数 2495
F.34.2. 作者 2496
F.35. tablefunc 2497
F.35.1. 提供的函数 2497
F.35.2. 作者 2505
F.36. tcn 2506
F.37. test_parser 2507
F.37.1. 用法 2507
F.38. tsearch2 2509
F.38.1. 可移植性问题 2509
F.38.2. 转换 8.3 之前的安装 2509
F.38.3. 参考文献 2510

F.39. unaccent 2511
F.39.1. 配置 2511
F.39.2. 用法 2511
F.39.3. 函数 2512
F.40. uuid-ossp 2513
F.40.1. `uuid-ossp` 函数 2513
F.40.2. 作者 2513
F.41. xml2 2514
F.41.1. 弃用说明 2514
F.41.2. 函数说明 2514
F.41.3. `xpath_table` 2515
F.41.4. XSLT 函数 2517
F.41.5. 作者 2517
G. 额外提供的程序 2518
G.1. 客户端应用 2518
G.2. 服务器应用 2532
H. 外部项目 2548
H.1. 客户端接口 2548
H.2. 管理工具 2548
H.3. 过程语言 2548
H.4. 扩展 2549
I. 源代码仓库 2550
I.1. 通过 Git 获取源代码 2550
J. Documentation 2551
J.1. DocBook 2551
J.2. Tool Sets 2551
J.2.1. Linux RPM Installation 2552
J.2.2. FreeBSD Installation 2552
J.2.3. Debian Packages 2553
J.2.4. Manual Installation from Source 2553
J.2.5. Detection by `configure` 2555
J.3. Building The Documentation 2556
J.3.1. HTML 2556
J.3.2. Manpages 2556
J.3.3. Print Output via JadeTeX 2556
J.3.4. Overflow Text 2557
J.3.5. Print Output via RTF 2557
J.3.6. Plain Text Files 2559
J.3.7. Syntax Check 2559
J.4. Documentation Authoring 2559
J.4.1. Emacs/PSGML 2559
J.4.2. Other Emacs Modes 2560
J.5. Style Guide 2560
J.5.1. Reference Pages 2560
K. 缩略词 2563

附录 A. PostgreSQL错误代码

PostgreSQL 服务器发出的所有消息都被分配了五字符错误代码，遵循 SQL 标准对“SQLSTATE”代码的约定。需要知道发生了哪一种错误条件的应用，通常应测试错误代码，而不是查看错误消息的文本内容。这些错误代码在不同 PostgreSQL 版本之间更不容易发生变化，也不会因错误消息的本地化而改变。请注意，PostgreSQL 产生的错误代码中，只有一部分由 SQL 标准定义；还有一些用于标准未定义情况的附加错误代码，是新设的或从其他数据库借用而来的。

根据标准，错误代码的前两个字符表示错误类别，后三个字符表示该类别中的特定条件。因此，即使应用无法识别某个具体错误代码，仍可能从错误类别推断出应采取的操作。

表 A.1列出了 PostgreSQL 9.2.24 中定义的所有错误代码。（其中有些当前实际上并未使用，不过它们由 SQL 标准定义。）其中也显示了错误类别。对于每个错误类别，都有一个后三个字符为 `000` 的“标准”错误代码。该代码只用于那些属于该类别但没有分配更具体代码的错误情况。

“条件名称”列中显示的符号也是 PL/pgSQL 中使用的条件名称。条件名称既可以写成大写，也可以写成小写。（注意，PL/pgSQL 不识别警告条件名称，也就是说不识别 00、01 和 02 这几个类别。）

表 A.1. PostgreSQL错误代码

错误代码	条件名称
Class 00 – Successful Completion	
00000	successful_completion
Class 01 – Warning	
01000	warning
0100C	dynamic_result_sets_returned
01008	implicit_zero_bit_padding
01003	null_value_eliminated_in_set_function
01007	privilege_not_granted
01006	privilege_not_revoked
01004	string_data_right_truncation
01P01	deprecated_feature
Class 02 – No Data (this is also a warning class per the SQL standard)	
02000	no_data
02001	no_additional_dynamic_result_sets_returned
Class 03 – SQL Statement Not Yet Complete	
03000	sql_statement_not_yet_complete
Class 08 – Connection Exception	
08000	connection_exception
08003	connection_does_not_exist
08006	connection_failure
08001	sqlclient_unable_to_establish_sqlconnection
08004	sqlserver_rejected_establishment_of_sqlconnection
08007	transaction_resolution_unknown

错误代码	条件名称
08P01	protocol_violation
Class 09 – Triggered Action Exception	
09000	triggered_action_exception
Class 0A – Feature Not Supported	
0A000	feature_not_supported
Class 0B – Invalid Transaction Initiation	
0B000	invalid_transaction_initiation
Class 0F – Locator Exception	
0F000	locator_exception
0F001	invalid_locator_specification
Class 0L – Invalid Grantor	
0L000	invalid_grantor
0LP01	invalid_grant_operation
Class 0P – Invalid Role Specification	
0P000	invalid_role_specification
Class 0Z – Diagnostics Exception	
0Z000	diagnostics_exception
0Z002	stacked_diagnostics_accessed_without_active_handler
Class 20 – Case Not Found	
20000	case_not_found
Class 21 – Cardinality Violation	
21000	cardinality_violation
Class 22 – Data Exception	
22000	data_exception
2202E	array_subscript_error
22021	character_not_in_repertoire
22008	datetime_field_overflow
22012	division_by_zero
22005	error_in_assignment
2200B	escape_character_conflict
22022	indicator_overflow
22015	interval_field_overflow
2201E	invalid_argument_for_logarithm
22014	invalid_argument_for_ntile_function
22016	invalid_argument_for_nth_value_function
2201F	invalid_argument_for_power_function
2201G	invalid_argument_for_width_bucket_function
22018	invalid_character_value_for_cast
22007	invalid_datetime_format
22019	invalid_escape_character
2200D	invalid_escape_octet

错误代码	条件名称
22025	invalid_escape_sequence
22P06	nonstandard_use_of_escape_character
22010	invalid_indicator_parameter_value
22023	invalid_parameter_value
2201B	invalid_regular_expression
2201W	invalid_row_count_in_limit_clause
2201X	invalid_row_count_in_result_offset_clause
22009	invalid_time_zone_displacement_value
2200C	invalid_use_of_escape_character
2200G	most_specific_type_mismatch
22004	null_value_not_allowed
22002	null_value_no_indicator_parameter
22003	numeric_value_out_of_range
22026	string_data_length_mismatch
22001	string_data_right_truncation
22011	substring_error
22027	trim_error
22024	unterminated_c_string
2200F	zero_length_character_string
22P01	floating_point_exception
22P02	invalid_text_representation
22P03	invalid_binary_representation
22P04	bad_copy_file_format
22P05	untranslatable_character
2200L	not_an_xml_document
2200M	invalid_xml_document
2200N	invalid_xml_content
2200S	invalid_xml_comment
2200T	invalid_xml_processing_instruction
Class 23 – Integrity Constraint Violation	
23000	integrity_constraint_violation
23001	restrict_violation
23502	not_null_violation
23503	foreign_key_violation
23505	unique_violation
23514	check_violation
23P01	exclusion_violation
Class 24 – Invalid Cursor State	
24000	invalid_cursor_state
Class 25 – Invalid Transaction State	
25000	invalid_transaction_state
25001	active_sql_transaction

错误代码	条件名称
25002	branch_transaction_already_active
25008	held_cursor_requires_same_isolation_level
25003	inappropriate_access_mode_for_branch_transaction
25004	inappropriate_isolation_level_for_branch_transaction
25005	no_active_sql_transaction_for_branch_transaction
25006	read_only_sql_transaction
25007	schema_and_data_statement_mixing_not_supported
25P01	no_active_sql_transaction
25P02	in_failed_sql_transaction
Class 26 – Invalid SQL Statement Name	
26000	invalid_sql_statement_name
Class 27 – Triggered Data Change Violation	
27000	triggered_data_change_violation
Class 28 – Invalid Authorization Specification	
28000	invalid_authorization_specification
28P01	invalid_password
Class 2B – Dependent Privilege Descriptors Still Exist	
2B000	dependent_privilege_descriptors_still_exist
2BP01	dependent_objects_still_exist
Class 2D – Invalid Transaction Termination	
2D000	invalid_transaction_termination
Class 2F – SQL Routine Exception	
2F000	sql_routine_exception
2F005	function_executed_no_return_statement
2F002	modifying_sql_data_not_permitted
2F003	prohibited_sql_statement_attempted
2F004	reading_sql_data_not_permitted
Class 34 – Invalid Cursor Name	
34000	invalid_cursor_name
Class 38 – External Routine Exception	
38000	external_routine_exception
38001	containing_sql_not_permitted
38002	modifying_sql_data_not_permitted
38003	prohibited_sql_statement_attempted
38004	reading_sql_data_not_permitted
Class 39 – External Routine Invocation Exception	
39000	external_routine_invocation_exception
39001	invalid_sqlstate_returned

错误代码	条件名称
39004	null_value_not_allowed
39P01	trigger_protocol_violated
39P02	srf_protocol_violated
Class 3B – Savepoint Exception	
3B000	savepoint_exception
3B001	invalid_savepoint_specification
Class 3D – Invalid Catalog Name	
3D000	invalid_catalog_name
Class 3F – Invalid Schema Name	
3F000	invalid_schema_name
Class 40 – Transaction Rollback	
40000	transaction_rollback
40002	transaction_integrity_constraint_violation
40001	serialization_failure
40003	statement_completion_unknown
40P01	deadlock_detected
Class 42 – Syntax Error or Access Rule Violation	
42000	syntax_error_or_access_rule_violation
42601	syntax_error
42501	insufficient_privilege
42846	cannot_coerce
42803	grouping_error
42P20	windowing_error
42P19	invalid_recursion
42830	invalid_foreign_key
42602	invalid_name
42622	name_too_long
42939	reserved_name
42804	datatype_mismatch
42P18	indeterminate_datatype
42P21	collation_mismatch
42P22	indeterminate_collation
42809	wrong_object_type
42703	undefined_column
42883	undefined_function
42P01	undefined_table
42P02	undefined_parameter
42704	undefined_object
42701	duplicate_column
42P03	duplicate_cursor
42P04	duplicate_database
42723	duplicate_function

错误代码	条件名称
42P05	duplicate_prepared_statement
42P06	duplicate_schema
42P07	duplicate_table
42712	duplicate_alias
42710	duplicate_object
42702	ambiguous_column
42725	ambiguous_function
42P08	ambiguous_parameter
42P09	ambiguous_alias
42P10	invalid_column_reference
42611	invalid_column_definition
42P11	invalid_cursor_definition
42P12	invalid_database_definition
42P13	invalid_function_definition
42P14	invalid_prepared_statement_definition
42P15	invalid_schema_definition
42P16	invalid_table_definition
42P17	invalid_object_definition
Class 44 – WITH CHECK OPTION Violation	
44000	with_check_option_violation
Class 53 – Insufficient Resources	
53000	insufficient_resources
53100	disk_full
53200	out_of_memory
53300	too_many_connections
53400	configuration_limit_exceeded
Class 54 – Program Limit Exceeded	
54000	program_limit_exceeded
54001	statement_too_complex
54011	too_many_columns
54023	too_many_arguments
Class 55 – Object Not In Prerequisite State	
55000	object_not_in_prerequisite_state
55006	object_in_use
55P02	cant_change_runtime_param
55P03	lock_not_available
Class 57 – Operator Intervention	
57000	operator_intervention
57014	query_canceled
57P01	admin_shutdown
57P02	crash_shutdown
57P03	cannot_connect_now
57P04	database_dropped

错误代码	条件名称
Class 58 – System Error (errors external to PostgreSQL itself)	
58000	system_error
58030	io_error
58P01	undefined_file
58P02	duplicate_file
Class F0 – Configuration File Error	
F0000	config_file_error
F0001	lock_file_exists
Class HV – Foreign Data Wrapper Error (SQL/MED)	
HV000	fdw_error
HV005	fdw_column_name_not_found
HV002	fdw_dynamic_parameter_value_needed
HV010	fdw_function_sequence_error
HV021	fdw_inconsistent_descriptor_information
HV024	fdw_invalid_attribute_value
HV007	fdw_invalid_column_name
HV008	fdw_invalid_column_number
HV004	fdw_invalid_data_type
HV006	fdw_invalid_data_type_descriptors
HV091	fdw_invalid_descriptor_field_identifier
HV00B	fdw_invalid_handle
HV00C	fdw_invalid_option_index
HV00D	fdw_invalid_option_name
HV090	fdw_invalid_string_length_or_buffer_length
HV00A	fdw_invalid_string_format
HV009	fdw_invalid_use_of_null_pointer
HV014	fdw_too_many_handles
HV001	fdw_out_of_memory
HV00P	fdw_no_schemas
HV00J	fdw_option_name_not_found
HV00K	fdw_reply_handle
HV00Q	fdw_schema_not_found
HV00R	fdw_table_not_found
HV00L	fdw_unable_to_create_execution
HV00M	fdw_unable_to_create_reply
HV00N	fdw_unable_to_establish_connection
Class P0 – PL/pgSQL Error	
P0000	plpgsql_error
P0001	raise_exception
P0002	no_data_found
P0003	too_many_rows

错误代码	条件名称
Class XX – Internal Error	
XX000	internal_error
XX001	data_corrupted
XX002	index_corrupted

附录 B. 日期/时间支持

PostgreSQL 对所有日期/时间输入都使用一个内部的启发式解析器。日期和时间以字符串形式输入，并被拆分为不同的字段，同时初步判断每个字段中可能包含哪类信息。
每个字段都会被解释，并被赋予数值、忽略或拒绝。
解析器内部为所有文本字段都维护了查找表，包括月份、星期几以及时区。

本附录包含关于这些查找表内容的信息，并描述了解析器解码日期和时间时所使用的步骤。

B.1. 日期/时间输入解释

日期/时间输入字符串按照下面的过程进行解码。

1. 将输入字符串拆分为词元，并把每个词元归类为字符串、时间、时区或数字。
   a. 如果数字词元包含冒号（`:`），则它是一个时间字符串。
      把其后的所有数字和冒号都包含进来。
   b. 如果数字词元包含连字符（`-`）、斜杠（`/`）或两个以上的点（`.`），
      则它是一个日期字符串，并且可能带有文本形式的月份。如果已经见过日期词元，
      则它会被解释为时区名称（例如 `America/New_York`）。
   c. 如果词元仅由数字组成，那么它要么是单个字段，要么是 ISO 8601 的拼接式日期（例如，`19990113` 表示 1999 年 1 月 13 日）或时间（例如，`141516` 表示 14:15:16）。
   d. 如果词元以加号（`+`）或减号（`-`）开头，那么它要么是数字时区，
      要么是特殊字段。
2. 如果词元是一个纯字母字符串，则将其与可能的字符串匹配：
   a. 先把该词元作为时区缩写进行一次二分查找表查询。
   b. 如果没有找到，则搜索内部表，将该词元识别为特殊字符串（例如 `today`）、星期名（例如 `Thursday`）、月份名（例如 `January`）或噪声词（例如 `at`、`on`）。
   c. 如果仍然没有找到，则抛出错误。
3. 当词元是一个数字或数字字段时：
   a. 如果有八位或六位数字，并且之前还没有读取到其他日期字段，
      则将其解释为“拼接式日期”（例如 `19990118` 或 `990118`）。其解释方式分别为 `YYYYMMDD` 或 `YYMMDD`。
   b. 如果词元是三位数字，并且已经读取到了年份，则将其解释为一年中的第几天。
   c. 如果是四位或六位数字，并且已经读取到了年份，则将其解释为时间（`HHMM` 或 `HHMMSS`）。
   d. 如果有三位或更多位数字，并且尚未找到任何日期字段，
      则将其解释为年份（这会强制其余日期字段按 yy-mm-dd 的顺序解释）。
   e. 否则，假定日期字段顺序遵循 `DateStyle` 设置：mm-dd-yy、dd-mm-yy 或 yy-mm-dd。如果发现月份字段或日字段超出范围，则抛出错误。
4. 如果指定了 BC，则将年份取反并加一后再用于内部存储。（格里高利历中没有 0 年，因此从数值上说，公元前 1 年会变成年 0。）
5. 如果没有指定 BC，并且年份字段的长度为两位，则将年份调整为四位。如果该字段小于 70，则加上 2000，否则加上 1900。

> **提示**
>
> 格里高利历公元 1-99 年可以使用带前导零的四位数字输入（例如，`0099` 表示公元 99 年）。

B.2. 日期/时间关键字

表 B.1展示了哪些词元会被识别为月份名称。

表 B.1. 月份名称

月份	缩写
January	Jan
February	Feb
March	Mar
April	Apr
May	
June	Jun
July	Jul
August	Aug
September	Sep, Sept
October	Oct
November	Nov
December	Dec

表 B.2展示了哪些词元会被识别为星期名称。

表 B.2. 星期名称

星期	缩写
Sunday	Sun
Monday	Mon
Tuesday	Tue, Tues
Wednesday	Wed, Weds
Thursday	Thu, Thur, Thurs
Friday	Fri
Saturday	Sat

表 B.3展示了用于各种修饰用途的词元。

表 B.3. 日期/时间字段修饰符

标识符	说明
`AM`	时间早于 12:00
`AT`	忽略
`JULIAN`, `JD`, `J`	下一个字段是儒略日
`ON`	忽略
`PM`	时间为 12:00 或之后

标识符	说明
T	下一个字段是时间

B.3. 日期/时间配置文件

由于时区缩写没有得到良好的标准化，PostgreSQL 提供了一种定制服务器所接受缩写集合的方法。运行时参数timezone_abbreviations决定当前使用的缩写集合。
任何数据库用户都可以修改此参数，但它的可选值由数据库管理员控制；
这些值实际上是安装目录中 `.../share/timezonesets/` 下的配置文件名。管理员可以通过在该目录中添加或修改文件，制定本地时区缩写策略。

如果文件名完全由字母组成，那么 `timezone_abbreviations` 可以被设置为 `.../share/timezonesets/` 中找到的任意文件名。（禁止在 `timezone_abbreviations` 中使用非字母字符，不仅可以防止读取目标目录之外的文件，也能防止读取编辑器备份文件和其他无关文件。）

时区缩写文件可以包含空行，以及以下字符开头的注释：`#`。
非注释行必须采用以下格式之一：

```
zone_abbreviation offset
zone_abbreviation offset D
zone_abbreviation time_zone_name
@INCLUDE file_name
@OVERRIDE
```

`zone_abbreviation` 就是要定义的缩写。*`offset`* 是一个整数，表示相对于 UTC 的等效偏移秒数，正数表示格林尼治以东，负数表示以西。例如，-18000 表示格林尼治以西五小时，即北美东海岸标准时间。`D` 表示该时区名称代表当地夏令时间，而非标准时间。

也可以给出 *`time_zone_name`*，引用 IANA 时区数据库中定义的时区名。系统会查阅该时区的定义，判断其中是否正在或曾经使用该缩写；如果是，
则采用合适的含义：即正在确定其值的时间戳所处时刻使用的含义；
如果该时刻并未使用此缩写，则取此前最近使用的含义；如果只在该时刻之后使用过，
则取最早的含义。这对于处理含义曾经发生变化的缩写至关重要。
也允许用一个并未出现该缩写的时区名来定义缩写；此时，
使用缩写就等价于写出该时区名。

> **提示**
>
> 如果某个缩写相对 UTC 的偏移从未改变过，那么定义它时最好使用简单的整数 *`offset`*，因为这类缩写的处理成本远低于那些需要查阅时区定义的缩写。

`@INCLUDE` 语法允许包含 `.../share/timezonesets/` 目录中的另一个文件。包含可以嵌套，但深度有限。

`@OVERRIDE` 语法表示文件中后续的条目可以覆盖之前的条目（通常是从被包含文件中取得的条目）。如果没有此指令，同一时区缩写的冲突定义会被视为错误。

在未经修改的安装中，`Default` 文件包含世界上大多数地区中所有互不冲突的时区缩写。另外还为澳大利亚和印度这两个地区提供了 `Australia` 和 `India` 文件：这些文件会先包含 `Default` 文件，然后按需添加或修改缩写。

作为参考，标准安装还包含 `Africa.txt`、`America.txt` 等文件，其中包含 IANA 时区数据库中已知正在使用的每个时区缩写的信息。
可以按需把这些文件中找到的时区名称定义复制并粘贴到自定义配置文件中。注意，

这些文件不能直接作为 `timezone_abbreviations` 设置来引用，因为它们的名称中带有点号。

> **注意**
>
> 如果在读取时区缩写集时发生错误，则不会应用新值，而会保留旧集合。
> 如果该错误发生在数据库启动期间，则数据库启动会失败。

> **小心**
>
> 配置文件中定义的时区缩写会覆盖 PostgreSQL 内置的非时区含义。例如，`Australia` 配置文件定义了 `SAT`（南澳大利亚标准时间）。当该文件处于活动状态时，`SAT` 将不会被识别为 Saturday 的缩写。

> **小心**
>
> 如果你修改了 `.../share/timezonesets/` 中的文件，就需要自行做好备份 – 普通的数据库转储不会包含这个目录。

B.4. 历法的历史

SQL 标准指出，“在‘日期时间字面量’的定义中，‘日期时间值’受格里高利历中日期和时间自然规则的约束”。PostgreSQL 遵循 SQL 标准的做法，即便对于该历法尚未实际使用之前的年份，也完全按照格里高利历计算日期。这一规则称为外推格里高利历。

儒略历由 Julius Caesar 于公元前 45 年引入。在 1582 年各国开始改用格里高利历之前，它在西方世界一直被广泛使用。在儒略历中，回归年近似为 365 1/4 天 = 365.25 天。这会在大约 128 年里累计 1 天的误差。

不断累积的历法误差促使教皇格里高利十三世按照特伦托会议的指示改革历法。
在格里高利历中，回归年近似为 365 + 97 / 400 天 = 365.2425 天。因此，大约要经过 3300 年，回归年与格里高利历之间才会累积出 1 天的偏差。

365+97/400 这一近似值是通过每 400 年设置 97 个闰年来实现的，具体规则如下：

每个能被 4 整除的年份都是闰年。
不过，每个能被 100 整除的年份都不是闰年。
但是，每个能被 400 整除的年份仍然是闰年。

因此，1700、1800、1900、2100 和 2200 都不是闰年，但 1600、2000 和 2400 是闰年。相比之下，较早的儒略历规定所有能被 4 整除的年份都是闰年。

1582 年 2 月发布的教皇诏书规定，应从 1582 年 10 月删去 10 天，使 10 月 15 日紧接在 10 月 4 日之后。这一改革在意大利、波兰、葡萄牙和西班牙得到执行。其他天主教国家不久后也跟进，但新教国家不愿改变，而希腊正教国家直到 20 世纪初才改用格里高利历。大不列颠及其属地（包括今天的美国）在 1752 年实施了这项改革。因此，1752 年 9 月 2 日之后紧接着就是 1752 年 9 月 14 日。这就是为什么提供 `cal` 程序的 Unix 系统会产生如下输出：

```
$ cal 9 1752
   September 1752
 S  M Tu  W Th  F  S
       1  2 14 15 16
```

```
17 18 19 20 21 22 23
24 25 26 27 28 29 30
```

不过，当然，这个历法只对大不列颠及其属地有效，对其他地方并不适用。
由于尝试跟踪不同地方在不同时间实际采用的历法既困难又容易引起混淆，PostgreSQL 并不这样做，而是对所有日期都遵循格里高利历规则，尽管这种做法在历史上并不准确。

世界上不同地区发展出了不同的历法，其中许多都早于格里高利体系。例如，
中国历法的起源可以追溯到公元前 14 世纪。传说黄帝在公元前 2637 年发明了这种历法。中华人民共和国在民用方面使用格里高利历。中国历法则用于确定节日。

儒略日系统是另一种历法，它与儒略历并无关系，只是名称容易让人误以为二者有关。
儒略日系统由法国学者 Joseph Justus Scaliger（1540-1609）发明，其名称大概取自他的父亲、意大利学者 Julius Caesar Scaliger（1484-1558）。在儒略日系统中，每一天都有一个连续编号，从 JD 0 开始（有时称为起始儒略日）。JD 0 对应儒略历公元前 4713 年 1 月 1 日，或格里高利历公元前 4714 年 11 月 24 日。儒略日计数最常用于天文学家标记夜间观测，因此一天从 UTC 正午开始，到下一个 UTC 正午结束，而不是从午夜到午夜：JD 0 表示从 UTC 公元前 4714 年 11 月 24 日正午，到 UTC 公元前 4714 年 11 月 25 日正午的 24 小时。

虽然 PostgreSQL 支持在日期输入和输出中使用儒略日记法（并且也会在某些内部日期/时间计算中使用儒略日），但它并不遵循日期从正午算到正午这一约定。PostgreSQL 将儒略日视为从午夜到午夜。

附录 C. SQL 关键字

表 C.1列出了在 SQL 标准以及PostgreSQL 9.2.24中被视为关键字的所有词元。背景信息可参见第 4.1.1 节。（由于篇幅所限，这里只列出了 SQL 标准最近两个版本的情况，以及用于历史比较的 SQL-92；这些版本与其他中间标准版本之间的差异很小。）

SQL 区分保留关键字和非保留关键字。根据标准，保留关键字才是真正的关键字，它们绝不允许作为标识符使用。非保留关键字只在特定上下文中具有特殊含义，
在其他上下文中则可以作为标识符使用。大多数非保留关键字实际上是 SQL 规定的内置表和内置函数名称。非保留关键字这一概念，
本质上只是为了声明某个词在某些上下文中附带了预定义含义。

在PostgreSQL解析器中，情况要更复杂一些。这里的词元分属若干不同类别，
从绝不能用作标识符的词元，到在解析器中完全没有特殊地位、
只是被当作普通标识符处理的词元都有。（后一类通常就是 SQL 规定的函数名。）即使是保留关键字，在PostgreSQL中也不是绝对保留的，它们仍然可以用作列标签（例如`SELECT 55 AS CHECK`，尽管`CHECK`是一个保留关键字）。

在表 C.1中针对PostgreSQL的那一列里，我们把解析器明确识别、但又允许作为列名或表名的那些关键字归类为“非保留”。
某些原本属于非保留的关键字不能用作函数名或数据类型名，因此会另外加以标记。
（这类词大多表示具有特殊语法的内置函数或数据类型。函数或类型本身仍然可用，
但用户不能重新定义它们。）不允许作为列名或表名的词元则标为“保留”。
有些保留关键字仍允许作为函数名或数据类型名，这一点也会在表中标出。
如果没有这样的标记，保留关键字就只允许作为“AS”列标签名使用。

一般来说，如果你在某条命令中把这里列出的关键字当作标识符使用，
并因此遇到意外的解析器错误，应尝试为该标识符加上引号，看看问题是否消失。

在研读表 C.1之前，有一点非常重要：某个关键字在PostgreSQL中不被保留，并不意味着与该词相关的特性尚未实现；反过来，某个关键字的存在也不意味着相应特性一定存在。

表 C.1. SQL 关键字

关键字	PostgreSQL	SQL:2011	SQL:2008	SQL-92
A		非保留	非保留	
ABORT	非保留			
ABS		保留	保留	
ABSENT		非保留	非保留	
ABSOLUTE	非保留	非保留	非保留	保留
ACCESS	非保留			
ACCORDING		非保留	非保留	
ACTION	非保留	非保留	非保留	保留
ADA		非保留	非保留	非保留
ADD	非保留	非保留	非保留	保留
ADMIN	非保留	非保留	非保留	
AFTER	非保留	非保留	非保留	
AGGREGATE	非保留			
ALL	保留	保留	保留	保留
ALLOCATE		保留	保留	保留
ALSO	非保留			
ALTER	非保留	保留	保留	保留
ALWAYS	非保留	非保留	非保留	

关键字	PostgreSQL	SQL:2011	SQL:2008	SQL-92
ANALYSE	保留			
ANALYZE	保留			
AND	保留	保留	保留	保留
ANY	保留	保留	保留	保留
ARE		保留	保留	保留
ARRAY	保留	保留	保留	
ARRAY_AGG		保留	保留	
ARRAY_MAX_CARDINALITY		保留		
AS	保留	保留	保留	保留
ASC	保留	非保留	非保留	保留
ASENSITIVE		保留	保留	
ASSERTION	非保留	非保留	非保留	保留
ASSIGNMENT	非保留	非保留	非保留	
ASYMMETRIC	保留	保留	保留	
AT	非保留	保留	保留	保留
ATOMIC		保留	保留	
ATTRIBUTE	非保留	非保留	非保留	
ATTRIBUTES		非保留	非保留	
AUTHORIZATION	保留（可用作函数或类型）	保留	保留	保留
AVG		保留	保留	保留
BACKWARD	非保留			
BASE64		非保留	非保留	
BEFORE	非保留	非保留	非保留	
BEGIN	非保留	保留	保留	保留
BEGIN_FRAME		保留		
BEGIN_PARTITION		保留		
BERNOULLI		非保留	非保留	
BETWEEN	非保留（不能用作函数或类型）	保留	保留	保留
BIGINT	非保留（不能用作函数或类型）	保留	保留	
BINARY	保留（可用作函数或类型）	保留	保留	
BIT	非保留（不能用作函数或类型）			保留
BIT_LENGTH				保留
BLOB		保留	保留	
BLOCKED		非保留	非保留	
BOM		非保留	非保留	
BOOLEAN	非保留（不能用作函数或类型）	保留	保留	
BOTH	保留	保留	保留	保留
BREADTH		非保留	非保留	

关键字	PostgreSQL	SQL:2011	SQL:2008	SQL-92
BY	非保留	保留	保留	保留
C		非保留	非保留	非保留
CACHE	非保留			
CALL		保留	保留	
CALLED	非保留	保留	保留	
CARDINALITY		保留	保留	
CASCADE	非保留	非保留	非保留	保留
CASCADED	非保留	保留	保留	保留
CASE	保留	保留	保留	保留
CAST	保留	保留	保留	保留
CATALOG	非保留	非保留	非保留	保留
CATALOG_NAME		非保留	非保留	非保留
CEIL		保留	保留	
CEILING		保留	保留	
CHAIN	非保留	非保留	非保留	
CHAR	非保留（不能用作函数或类型）	保留	保留	保留
CHARACTER	非保留（不能用作函数或类型）	保留	保留	保留
CHARACTERISTICS	非保留	非保留	非保留	
CHARACTERS		非保留	非保留	
CHARACTER_LENGTH		保留	保留	保留
CHARACTER_SET_CATALOG		非保留	非保留	非保留
CHARACTER_SET_NAME		非保留	非保留	非保留
CHARACTER_SET_SCHEMA		非保留	非保留	非保留
CHAR_LENGTH		保留	保留	保留
CHECK	保留	保留	保留	保留
CHECKPOINT	非保留			
CLASS	非保留			
CLASS_ORIGIN		非保留	非保留	非保留
CLOB		保留	保留	
CLOSE	非保留	保留	保留	保留
CLUSTER	非保留			
COALESCE	非保留（不能用作函数或类型）	保留	保留	保留
COBOL		非保留	非保留	非保留
COLLATE	保留	保留	保留	保留
COLLATION	保留（可用作函数或类型）	非保留	非保留	保留
COLLATION_CATALOG		非保留	非保留	非保留

关键字	PostgreSQL	SQL:2011	SQL:2008	SQL-92
COLLATION_NAME		非保留	非保留	非保留
COLLATION_SCHEMA		非保留	非保留	非保留
COLLECT		保留	保留	
COLUMN	保留	保留	保留	保留
COLUMNS		非保留	非保留	
COLUMN_NAME		非保留	非保留	非保留
COMMAND_FUNCTION		非保留	非保留	非保留
COMMAND_FUNCTION_CODE		非保留	非保留	
COMMENT	非保留			
COMMENTS	非保留			
COMMIT	非保留	保留	保留	保留
COMMITTED	非保留	非保留	非保留	非保留
CONCURRENTLY	保留（可用作函数或类型）			
CONDITION		保留	保留	
CONDITION_NUMBER		非保留	非保留	非保留
CONFIGURATION	非保留			
CONNECT		保留	保留	保留
CONNECTION	非保留	非保留	非保留	保留
CONNECTION_NAME		非保留	非保留	非保留
CONSTRAINT	保留	保留	保留	保留
CONSTRAINTS	非保留	非保留	非保留	保留
CONSTRAINT_CATALOG		非保留	非保留	非保留
CONSTRAINT_NAME		非保留	非保留	非保留
CONSTRAINT_SCHEMA		非保留	非保留	非保留
CONSTRUCTOR		非保留	非保留	
CONTAINS		保留	非保留	
CONTENT	非保留	非保留	非保留	
CONTINUE	非保留	非保留	非保留	保留
CONTROL		非保留	非保留	
CONVERSION	非保留			
CONVERT		保留	保留	保留
COPY	非保留			
CORR		保留	保留	
CORRESPONDING		保留	保留	保留
COST	非保留			
COUNT		保留	保留	保留
COVAR_POP		保留	保留	

关键字	PostgreSQL	SQL:2011	SQL:2008	SQL-92
COVAR_SAMP		保留	保留	
CREATE	保留	保留	保留	保留
CROSS	保留（可用作函数或类型）	保留	保留	保留
CSV	非保留			
CUBE	非保留	保留	保留	
CUME_DIST		保留	保留	
CURRENT	非保留	保留	保留	保留
CURRENT_CATALOG	保留	保留	保留	
CURRENT_DATE	保留	保留	保留	保留
CURRENT_DEFAULT_TRANSFORM_GROUP		保留	保留	
CURRENT_PATH		保留	保留	
CURRENT_ROLE	保留	保留	保留	
CURRENT_ROW		保留		
CURRENT_SCHEMA	保留（可用作函数或类型）	保留	保留	
CURRENT_TIME	保留	保留	保留	保留
CURRENT_TIMESTAMP	保留	保留	保留	保留
CURRENT_TRANSFORM_GROUP_FOR_TYPE		保留	保留	
CURRENT_USER	保留	保留	保留	保留
CURSOR	非保留	保留	保留	保留
CURSOR_NAME		非保留	非保留	非保留
CYCLE	非保留	保留	保留	
DATA	非保留	非保留	非保留	非保留
DATABASE	非保留			
DATALINK		保留	保留	
DATE		保留	保留	保留
DATETIME_INTERVAL_CODE		非保留	非保留	非保留
DATETIME_INTERVAL_PRECISION		非保留	非保留	非保留
DAY	非保留	保留	保留	保留
DB		非保留	非保留	
DEALLOCATE	非保留	保留	保留	保留
DEC	非保留（不能用作函数或类型）	保留	保留	保留
DECIMAL	非保留（不能用作函数或类型）	保留	保留	保留
DECLARE	非保留	保留	保留	保留
DEFAULT	保留	保留	保留	保留

关键字	PostgreSQL	SQL:2011	SQL:2008	SQL-92
DEFAULTS	非保留	非保留	非保留	
DEFERRABLE	保留	非保留	非保留	保留
DEFERRED	非保留	非保留	非保留	保留
DEFINED		非保留	非保留	
DEFINER	非保留	非保留	非保留	
DEGREE		非保留	非保留	
DELETE	非保留	保留	保留	保留
DELIMITER	非保留			
DELIMITERS	非保留			
DENSE_RANK		保留	保留	
DEPTH		非保留	非保留	
DEREF		保留	保留	
DERIVED		非保留	非保留	
DESC	保留	非保留	非保留	保留
DESCRIBE		保留	保留	保留
DESCRIPTOR		非保留	非保留	保留
DETERMINISTIC		保留	保留	
DIAGNOSTICS		非保留	非保留	保留
DICTIONARY	非保留			
DISABLE	非保留			
DISCARD	非保留			
DISCONNECT		保留	保留	保留
DISPATCH		非保留	非保留	
DISTINCT	保留	保留	保留	保留
DLNEWCOPY		保留	保留	
DLPREVIOUSCOPY		保留	保留	
DLURLCOMPLETE		保留	保留	
DLURLCOMPLETEONLY		保留	保留	
DLURLCOMPLETEWRITE		保留	保留	
DLURLPATH		保留	保留	
DLURLPATHONLY		保留	保留	
DLURLPATHWRITE		保留	保留	
DLURLSCHEME		保留	保留	
DLURLSERVER		保留	保留	
DLVALUE		保留	保留	
DO	保留			
DOCUMENT	非保留	非保留	非保留	
DOMAIN	非保留	非保留	非保留	保留
DOUBLE	非保留	保留	保留	保留
DROP	非保留	保留	保留	保留
DYNAMIC		保留	保留	

关键字	PostgreSQL	SQL:2011	SQL:2008	SQL-92
DYNAMIC_FUNCTION		非保留	非保留	非保留
DYNAMIC_FUNCTION_CODE		非保留	非保留	
EACH	非保留	保留	保留	
ELEMENT		保留	保留	
ELSE	保留	保留	保留	保留
EMPTY		非保留	非保留	
ENABLE	非保留			
ENCODING	非保留	非保留	非保留	
ENCRYPTED	非保留			
END	保留	保留	保留	保留
END-EXEC		保留	保留	保留
END_FRAME		保留		
END_PARTITION		保留		
ENFORCED		非保留		
ENUM	非保留			
EQUALS		保留	非保留	
ESCAPE	非保留	保留	保留	保留
EVERY		保留	保留	
EXCEPT	保留	保留	保留	保留
EXCEPTION				保留
EXCLUDE	非保留	非保留	非保留	
EXCLUDING	非保留	非保留	非保留	
EXCLUSIVE	非保留			
EXEC		保留	保留	保留
EXECUTE	非保留	保留	保留	保留
EXISTS	非保留（不能用作函数或类型）	保留	保留	保留
EXP		保留	保留	
EXPLAIN	非保留			
EXPRESSION		非保留		
EXTENSION	非保留			
EXTERNAL	非保留	保留	保留	保留
EXTRACT	非保留（不能用作函数或类型）	保留	保留	保留
FALSE	保留	保留	保留	保留
FAMILY	非保留			
FETCH	保留	保留	保留	保留
FILE		非保留	非保留	
FILTER		保留	保留	
FINAL		非保留	非保留	
FIRST	非保留	非保留	非保留	保留
FIRST_VALUE		保留	保留	

关键字	PostgreSQL	SQL:2011	SQL:2008	SQL-92
FLAG		非保留	非保留	
FLOAT	非保留（不能用作函数或类型）	保留	保留	保留
FLOOR		保留	保留	
FOLLOWING	非保留	非保留	非保留	
FOR	保留	保留	保留	保留
FORCE	非保留			
FOREIGN	保留	保留	保留	保留
FORTRAN		非保留	非保留	非保留
FORWARD	非保留			
FOUND		非保留	非保留	保留
FRAME_ROW		保留		
FREE		保留	保留	
FREEZE	保留（可用作函数或类型）			
FROM	保留	保留	保留	保留
FS		非保留	非保留	
FULL	保留（可用作函数或类型）	保留	保留	保留
FUNCTION	非保留	保留	保留	
FUNCTIONS	非保留			
FUSION		保留	保留	
G		非保留	非保留	
GENERAL		非保留	非保留	
GENERATED		非保留	非保留	
GET		保留	保留	保留
GLOBAL	非保留	保留	保留	保留
GO		非保留	非保留	保留
GOTO		非保留	非保留	保留
GRANT	保留	保留	保留	保留
GRANTED	非保留	非保留	非保留	
GREATEST	非保留（不能用作函数或类型）			
GROUP	保留	保留	保留	保留
GROUPING	非保留（不能用作函数或类型）	保留	保留	
GROUPS		保留		
HANDLER	非保留			
HAVING	保留	保留	保留	保留
HEADER	非保留			
HEX		非保留	非保留	
HIERARCHY		非保留	非保留	
HOLD	非保留	保留	保留	
HOUR	非保留	保留	保留	保留
ID		非保留	非保留	

关键字	PostgreSQL	SQL:2011	SQL:2008	SQL-92
IDENTITY	非保留	保留	保留	保留
IF	非保留			
IGNORE		非保留	非保留	
ILIKE	保留（可用作函数或类型）			
IMMEDIATE	非保留	非保留	非保留	保留
IMMEDIATELY		非保留		
IMMUTABLE	非保留			
IMPLEMENTATION		非保留	非保留	
IMPLICIT	非保留			
IMPORT	非保留	保留	保留	
IN	保留	保留	保留	保留
INCLUDING	非保留	非保留	非保留	
INCREMENT	非保留	非保留	非保留	
INDENT		非保留	非保留	
INDEX	非保留			
INDEXES	非保留			
INDICATOR		保留	保留	保留
INHERIT	非保留			
INHERITS	非保留			
INITIALLY	保留	非保留	非保留	保留
INLINE	非保留			
INNER	保留（可用作函数或类型）	保留	保留	保留
INOUT	非保留（不能用作函数或类型）	保留	保留	
INPUT	非保留	非保留	非保留	保留
INSENSITIVE	非保留	保留	保留	保留
INSERT	非保留	保留	保留	保留
INSTANCE		非保留	非保留	
INSTANTIABLE		非保留	非保留	
INSTEAD	非保留	非保留	非保留	
INT	非保留（不能用作函数或类型）	保留	保留	保留
INTEGER	非保留（不能用作函数或类型）	保留	保留	保留
INTEGRITY		非保留	非保留	
INTERSECT	保留	保留	保留	保留
INTERSECTION		保留	保留	
INTERVAL	非保留（不能用作函数或类型）	保留	保留	保留
INTO	保留	保留	保留	保留
INVOKER	非保留	非保留	非保留	
IS	保留（可用作函数或类型）	保留	保留	保留

关键字	PostgreSQL	SQL:2011	SQL:2008	SQL-92
ISNULL	保留（可用作函数或类型）			
ISOLATION	非保留	非保留	非保留	保留
JOIN	保留（可用作函数或类型）	保留	保留	保留
K		非保留	非保留	
KEY	非保留	非保留	非保留	保留
KEY_MEMBER		非保留	非保留	
KEY_TYPE		非保留	非保留	
LABEL	非保留			
LAG		保留	保留	
LANGUAGE	非保留	保留	保留	保留
LARGE	非保留	保留	保留	
LC_COLLATE	非保留			
LC_CTYPE	非保留			
LAST	非保留	非保留	非保留	保留
LAST_VALUE		保留	保留	
LATERAL		保留	保留	
LEAD		保留	保留	
LEADING	保留	保留	保留	保留
LEAKPROOF	非保留			
LEAST	非保留（不能用作函数或类型）			
LEFT	保留（可用作函数或类型）	保留	保留	保留
LENGTH		非保留	非保留	非保留
LEVEL	非保留	非保留	非保留	保留
LIBRARY		非保留	非保留	
LIKE	保留（可用作函数或类型）	保留	保留	保留
LIKE_REGEX		保留	保留	
LIMIT	保留	非保留	非保留	
LINK		非保留	非保留	
LISTEN	非保留			
LN		保留	保留	
LOAD	非保留			
LOCAL	非保留	保留	保留	保留
LOCALTIME	保留	保留	保留	
LOCALTIMESTAMP	保留	保留	保留	
LOCATION	非保留	非保留	非保留	
LOCATOR		非保留	非保留	
LOCK	非保留			
LOWER		保留	保留	保留
M		非保留	非保留	
MAP		非保留	非保留	

关键字	PostgreSQL	SQL:2011	SQL:2008	SQL-92
MAPPING	非保留	非保留	非保留	
MATCH	非保留	保留	保留	保留
MATCHED		非保留	非保留	
MAX		保留	保留	保留
MAXVALUE	非保留	非保留	非保留	
MAX_CARDINALITY			保留	
MEMBER		保留	保留	
MERGE		保留	保留	
MESSAGE_LENGTH		非保留	非保留	非保留
MESSAGE_OCTET_LENGTH		非保留	非保留	非保留
MESSAGE_TEXT		非保留	非保留	非保留
METHOD		保留	保留	
MIN		保留	保留	保留
MINUTE	非保留	保留	保留	保留
MINVALUE	非保留	非保留	非保留	
MOD		保留	保留	
MODE	非保留			
MODIFIES		保留	保留	
MODULE		保留	保留	保留
MONTH	非保留	保留	保留	保留
MORE		非保留	非保留	非保留
MOVE	非保留			
MULTISET		保留	保留	
MUMPS		非保留	非保留	非保留
NAME	非保留	非保留	非保留	非保留
NAMES	非保留	非保留	非保留	保留
NAMESPACE		非保留	非保留	
NATIONAL	非保留（不能用作函数或类型）	保留	保留	保留
NATURAL	保留（可用作函数或类型）	保留	保留	保留
NCHAR	非保留（不能用作函数或类型）	保留	保留	保留
NCLOB		保留	保留	
NESTING		非保留	非保留	
NEW		保留	保留	
NEXT	非保留	非保留	非保留	保留
NFC		非保留	非保留	
NFD		非保留	非保留	
NFKC		非保留	非保留	
NFKD		非保留	非保留	
NIL		非保留	非保留	
NO	非保留	保留	保留	保留

关键字	PostgreSQL	SQL:2011	SQL:2008	SQL-92
NONE	非保留（不能用作函数或类型）	保留	保留	
NORMALIZE		保留	保留	
NORMALIZED		非保留	非保留	
NOT	保留	保留	保留	保留
NOTHING	非保留			
NOTIFY	非保留			
NOTNULL	保留（可用作函数或类型）			
NOWAIT	非保留			
NTH_VALUE		保留	保留	
NTILE		保留	保留	
NULL	保留	保留	保留	保留
NULLABLE		非保留	非保留	非保留
NULLIF	非保留（不能用作函数或类型）	保留	保留	保留
NULLS	非保留	非保留	非保留	
NUMBER		非保留	非保留	非保留
NUMERIC	非保留（不能用作函数或类型）	保留	保留	保留
OBJECT	非保留	非保留	非保留	
OCCURRENCES_REGEX		保留	保留	
OCTETS		非保留	非保留	
OCTET_LENGTH		保留	保留	保留
OF	非保留	保留	保留	保留
OFF	非保留	非保留	非保留	
OFFSET	保留	保留	保留	
OIDS	非保留			
OLD		保留	保留	
ON	保留	保留	保留	保留
ONLY	保留	保留	保留	保留
OPEN		保留	保留	保留
OPERATOR	非保留			
OPTION	非保留	非保留	非保留	保留
OPTIONS	非保留	非保留	非保留	
OR	保留	保留	保留	保留
ORDER	保留	保留	保留	保留
ORDERING		非保留	非保留	
ORDINALITY	非保留	非保留	非保留	
OTHERS		非保留	非保留	
OUT	非保留（不能用作函数或类型）	保留	保留	
OUTER	保留（可用作函数或类型）	保留	保留	保留

关键字	PostgreSQL	SQL:2011	SQL:2008	SQL-92
OUTPUT		非保留	非保留	保留
OVER	非保留	保留	保留	
OVERLAPS	保留（可用作函数或类型）	保留	保留	保留
OVERLAY	非保留（不能用作函数或类型）	保留	保留	
OVERRIDING		非保留	非保留	
OWNED	非保留			
OWNER	非保留			
P		非保留	非保留	
PAD		非保留	非保留	保留
PARAMETER		保留	保留	
PARAMETER_MODE		非保留	非保留	
PARAMETER_NAME		非保留	非保留	
PARAMETER_ ORDINAL_POSITI ON		非保留	非保留	
PARAMETER_ SPECIFIC_ CATALOG		非保留	非保留	
PARAMETER_ SPECIFIC_NAME		非保留	非保留	
PARAMETER_ SPECIFIC_ SCHEMA		非保留	非保留	
PARSER	非保留			
PARTIAL	非保留	非保留	非保留	保留
PARTITION	非保留	保留	保留	
PASCAL		非保留	非保留	非保留
PASSING	非保留	非保留	非保留	
PASSTHROUGH		非保留	非保留	
PASSWORD	非保留			
PATH		非保留	非保留	
PERCENT		保留		
PERCENTILE_ CONT		保留	保留	
PERCENTILE_ DISC		保留	保留	
PERCENT_RANK		保留	保留	
PERIOD		保留		
PERMISSION		非保留	非保留	
PLACING	保留	非保留	非保留	
PLANS	非保留			
PLI		非保留	非保留	非保留
PORTION		保留		
POSITION	非保留（不能用作函数或类型）	保留	保留	保留

关键字	PostgreSQL	SQL:2011	SQL:2008	SQL-92
POSITION_REGEX		保留	保留	
POWER		保留	保留	
PRECEDES		保留		
PRECEDING	非保留	非保留	非保留	
PRECISION	非保留（不能用作函数或类型）	保留	保留	保留
PREPARE	非保留	保留	保留	保留
PREPARED	非保留			
PRESERVE	非保留	非保留	非保留	保留
PRIMARY	保留	保留	保留	保留
PRIOR	非保留	非保留	非保留	保留
PRIVILEGES	非保留	非保留	非保留	保留
PROCEDURAL	非保留			
PROCEDURE	非保留	保留	保留	保留
PUBLIC		非保留	非保留	保留
QUOTE	非保留			
RANGE	非保留	保留	保留	
RANK		保留	保留	
READ	非保留	非保留	非保留	保留
READS		保留	保留	
REAL	非保留（不能用作函数或类型）	保留	保留	保留
REASSIGN	非保留			
RECHECK	非保留			
RECOVERY		非保留	非保留	
RECURSIVE	非保留	保留	保留	
REF	非保留	保留	保留	
REFERENCES	保留	保留	保留	保留
REFERENCING		保留	保留	
REGR_AVGX		保留	保留	
REGR_AVGY		保留	保留	
REGR_COUNT		保留	保留	
REGR_INTERCEPT		保留	保留	
REGR_R2		保留	保留	
REGR_SLOPE		保留	保留	
REGR_SXX		保留	保留	
REGR_SXY		保留	保留	
REGR_SYY		保留	保留	
REINDEX	非保留			
RELATIVE	非保留	非保留	非保留	保留
RELEASE	非保留	保留	保留	
RENAME	非保留			
REPEATABLE	非保留	非保留	非保留	非保留
REPLACE	非保留			

关键字	PostgreSQL	SQL:2011	SQL:2008	SQL-92
REPLICA	非保留			
REQUIRING		非保留	非保留	
RESET	非保留			
RESPECT		非保留	非保留	
RESTART	非保留	非保留	非保留	
RESTORE		非保留	非保留	
RESTRICT	非保留	非保留	非保留	保留
RESULT		保留	保留	
RETURN		保留	保留	
RETURNED_CARDINALITY		非保留	非保留	
RETURNED_LENGTH		非保留	非保留	非保留
RETURNED_OCTET_LENGTH		非保留	非保留	非保留
RETURNED_SQLSTATE		非保留	非保留	非保留
RETURNING	保留	非保留	非保留	
RETURNS	非保留	保留	保留	
REVOKE	非保留	保留	保留	保留
RIGHT	保留（可用作函数或类型）	保留	保留	保留
ROLE	非保留	非保留	非保留	
ROLLBACK	非保留	保留	保留	保留
ROLLUP	非保留	保留	保留	
ROUTINE		非保留	非保留	
ROUTINE_CATALOG		非保留	非保留	
ROUTINE_NAME		非保留	非保留	
ROUTINE_SCHEMA		非保留	非保留	
ROW	非保留（不能用作函数或类型）	保留	保留	
ROWS	非保留	保留	保留	保留
ROW_COUNT		非保留	非保留	非保留
ROW_NUMBER		保留	保留	
RULE	非保留			
SAVEPOINT	非保留	保留	保留	
SCALE		非保留	非保留	非保留
SCHEMA	非保留	非保留	非保留	保留
SCHEMA_NAME		非保留	非保留	非保留
SCOPE		保留	保留	
SCOPE_CATALOG		非保留	非保留	
SCOPE_NAME		非保留	非保留	
SCOPE_SCHEMA		非保留	非保留	
SCROLL	非保留	保留	保留	保留

关键字	PostgreSQL	SQL:2011	SQL:2008	SQL-92
SEARCH	非保留	保留	保留	
SECOND	非保留	保留	保留	保留
SECTION		非保留	非保留	保留
SECURITY	非保留	非保留	非保留	
SELECT	保留	保留	保留	保留
SELECTIVE		非保留	非保留	
SELF		非保留	非保留	
SENSITIVE		保留	保留	
SEQUENCE	非保留	非保留	非保留	
SEQUENCES	非保留			
SERIALIZABLE	非保留	非保留	非保留	非保留
SERVER	非保留	非保留	非保留	
SERVER_NAME		非保留	非保留	非保留
SESSION	非保留	非保留	非保留	保留
SESSION_USER	保留	保留	保留	保留
SET	非保留	保留	保留	保留
SETOF	非保留（不能用作函数或类型）			
SETS	非保留	非保留	非保留	
SHARE	非保留			
SHOW	非保留			
SIMILAR	保留（可用作函数或类型）	保留	保留	
SIMPLE	非保留	非保留	非保留	
SIZE		非保留	非保留	保留
SMALLINT	非保留（不能用作函数或类型）	保留	保留	保留
SNAPSHOT	非保留			
SOME	保留	保留	保留	保留
SOURCE		非保留	非保留	
SPACE		非保留	非保留	保留
SPECIFIC		保留	保留	
SPECIFICTYPE		保留	保留	
SPECIFIC_NAME		非保留	非保留	
SQL	非保留	保留	保留	保留
SQLCODE				保留
SQLERROR				保留
SQLEXCEPTION		保留	保留	
SQLSTATE		保留	保留	保留
SQLWARNING		保留	保留	
SQRT		保留	保留	
STABLE	非保留			
STANDALONE	非保留	非保留	非保留	
START	非保留	保留	保留	

关键字	PostgreSQL	SQL:2011	SQL:2008	SQL-92
STATE		非保留	非保留	
STATEMENT	非保留	非保留	非保留	
STATIC		保留	保留	
STATISTICS	非保留			
STDDEV_POP		保留	保留	
STDDEV_SAMP		保留	保留	
STDIN	非保留			
STDOUT	非保留			
STORAGE	非保留			
STRICT	非保留			
STRIP	非保留	非保留	非保留	
STRUCTURE		非保留	非保留	
STYLE		非保留	非保留	
SUBCLASS_ ORIGIN		非保留	非保留	非保留
SUBMULTISET		保留	保留	
SUBSTRING	非保留（不能用作函数或类型）	保留	保留	保留
SUBSTRING_ REGEX		保留	保留	
SUCCEEDS		保留		
SUM		保留	保留	保留
SYMMETRIC	保留	保留	保留	
SYSID	非保留			
SYSTEM	非保留	保留	保留	
SYSTEM_TIME		保留		
SYSTEM_USER		保留	保留	保留
T		非保留	非保留	
TABLE	保留	保留	保留	保留
TABLES	非保留			
TABLESAMPLE	保留（可用作函数或类型）	保留	保留	
TABLESPACE	非保留			
TABLE_NAME		非保留	非保留	非保留
TEMP	非保留			
TEMPLATE	非保留			
TEMPORARY	非保留	非保留	非保留	保留
TEXT	非保留			
THEN	保留	保留	保留	保留
TIES		非保留	非保留	
TIME	非保留（不能用作函数或类型）	保留	保留	保留
TIMESTAMP	非保留（不能用作函数或类型）	保留	保留	保留
TIMEZONE_HOUR		保留	保留	保留

关键字	PostgreSQL	SQL:2011	SQL:2008	SQL-92
TIMEZONE_MINUTE		保留	保留	保留
TO	保留	保留	保留	保留
TOKEN		非保留	非保留	
TOP_LEVEL_COUNT		非保留	非保留	
TRAILING	保留	保留	保留	保留
TRANSACTION	非保留	非保留	非保留	保留
TRANSACTIONS_COMMITTED		非保留	非保留	
TRANSACTIONS_ROLLED_BACK		非保留	非保留	
TRANSACTION_ACTIVE		非保留	非保留	
TRANSFORM	非保留	非保留	非保留	
TRANSFORMS		非保留	非保留	
TRANSLATE		保留	保留	保留
TRANSLATE_REGEX		保留	保留	
TRANSLATION		保留	保留	保留
TREAT	非保留（不能用作函数或类型）	保留	保留	
TRIGGER	非保留	保留	保留	
TRIGGER_CATALOG		非保留	非保留	
TRIGGER_NAME		非保留	非保留	
TRIGGER_SCHEMA		非保留	非保留	
TRIM	非保留（不能用作函数或类型）	保留	保留	保留
TRIM_ARRAY		保留	保留	
TRUE	保留	保留	保留	保留
TRUNCATE	非保留	保留	保留	
TRUSTED	非保留			
TYPE	非保留	非保留	非保留	非保留
TYPES	非保留			
UESCAPE		保留	保留	
UNBOUNDED	非保留	非保留	非保留	
UNCOMMITTED	非保留	非保留	非保留	非保留
UNDER		非保留	非保留	
UNENCRYPTED	非保留			
UNION	保留	保留	保留	保留
UNIQUE	保留	保留	保留	保留
UNKNOWN	非保留	保留	保留	保留
UNLINK		非保留	非保留	
UNLISTEN	非保留			
UNLOGGED	非保留			

关键字	PostgreSQL	SQL:2011	SQL:2008	SQL-92
UNNAMED		非保留	非保留	非保留
UNNEST		保留	保留	
UNTIL	非保留			
UNTYPED		非保留	非保留	
UPDATE	非保留	保留	保留	保留
UPPER		保留	保留	保留
URI		非保留	非保留	
USAGE		非保留	非保留	保留
USER	保留	保留	保留	保留
USER_DEFINED_TYPE_CATALOG		非保留	非保留	
USER_DEFINED_TYPE_CODE		非保留	非保留	
USER_DEFINED_TYPE_NAME		非保留	非保留	
USER_DEFINED_TYPE_SCHEMA		非保留	非保留	
USING	保留	保留	保留	保留
VACUUM	非保留			
VALID	非保留	非保留	非保留	
VALIDATE	非保留			
VALIDATOR	非保留			
VALUE	非保留	保留	保留	保留
VALUES	非保留（不能用作函数或类型）	保留	保留	保留
VALUE_OF		保留		
VARBINARY		保留	保留	
VARCHAR	非保留（不能用作函数或类型）	保留	保留	保留
VARIADIC	保留			
VARYING	非保留	保留	保留	保留
VAR_POP		保留	保留	
VAR_SAMP		保留	保留	
VERBOSE	保留（可用作函数或类型）			
VERSION	非保留	非保留	非保留	
VERSIONING		保留		
VIEW	非保留	非保留	非保留	保留
VOLATILE	非保留			
WHEN	保留	保留	保留	保留
WHENEVER		保留	保留	保留
WHERE	保留	保留	保留	保留
WHITESPACE	非保留	非保留	非保留	
WIDTH_BUCKET		保留	保留	
WINDOW	保留	保留	保留	

关键字	PostgreSQL	SQL:2011	SQL:2008	SQL-92
WITH	保留	保留	保留	保留
WITHIN	非保留	保留	保留	
WITHOUT	非保留	保留	保留	
WORK	非保留	非保留	非保留	保留
WRAPPER	非保留	非保留	非保留	
WRITE	非保留	非保留	非保留	保留
XML	非保留	保留	保留	
XMLAGG		保留	保留	
XMLATTRIBUTES	非保留（不能用作函数或类型）	保留	保留	
XMLBINARY		保留	保留	
XMLCAST		保留	保留	
XMLCOMMENT		保留	保留	
XMLCONCAT	非保留（不能用作函数或类型）	保留	保留	
XMLDECLARATION		非保留	非保留	
XMLDOCUMENT		保留	保留	
XMLELEMENT	非保留（不能用作函数或类型）	保留	保留	
XMLEXISTS	非保留（不能用作函数或类型）	保留	保留	
XMLFOREST	非保留（不能用作函数或类型）	保留	保留	
XMLITERATE		保留	保留	
XMLNAMESPACES		保留	保留	
XMLPARSE	非保留（不能用作函数或类型）	保留	保留	
XMLPI	非保留（不能用作函数或类型）	保留	保留	
XMLQUERY		保留	保留	
XMLROOT	非保留（不能用作函数或类型）			
XMLSCHEMA		非保留	非保留	
XMLSERIALIZE	非保留（不能用作函数或类型）	保留	保留	
XMLTABLE		保留	保留	
XMLTEXT		保留	保留	
XMLVALIDATE		保留	保留	
YEAR	非保留	保留	保留	保留
YES	非保留	非保留	非保留	
ZONE	非保留	非保留	非保留	保留

附录 D. SQL 符合性

本节尝试概述PostgreSQL在多大程度上符合当前的 SQL 标准。下文并不是一份完整的符合性声明，但会在合理且对用户有用的范围内，尽可能详细地介绍主要主题。

SQL 标准的正式名称是 ISO/IEC 9075“Database Language SQL”。该标准会不时发布修订版，最近一次更新出现在 2011 年。2011 年版称为 ISO/IEC 9075:2011，也简称 SQL:2011。此前的版本依次为 SQL:2008、SQL:2003、SQL:1999 和 SQL-92。每个版本都会取代前一版本，因此宣称符合更早版本并无官方意义。PostgreSQL 的开发目标是在不违背传统特性或常识的前提下，尽可能符合标准的最新正式版本。SQL 标准要求的许多特性都得到了支持，虽然有时语法或功能会略有不同。随着时间推移，仍可期待它在符合性方面继续推进。

SQL-92为符合性定义了三个特性集：入门（Entry）、中级（Intermediate）和完整（Full）。多数宣称符合SQL标准的数据库管理系统，实际上只符合入门（Entry）级别，因为中级（Intermediate）和完整（Full）级别中的全部特性要么过于庞杂，要么与既有行为相冲突。

从SQL:1999开始，SQL 标准不再沿用SQL-92中那种效果不佳且过于宽泛的三级分类，而是定义了大量独立特性。其中很大一个子集构成了核心（“Core”）特性，也就是每个符合 SQL 标准的实现都必须提供的特性。其余特性则完全是可选的。某些可选特性被组合在一起，形成“包”，SQL 实现可以声明符合这些包，从而表明符合特定的特性组。

从SQL:2003开始的各版标准还被拆分为若干部分，每一部分都有一个简称。请注意，这些部分的编号并不是连续的。

- ISO/IEC 9075-1 框架（SQL/Framework）
- ISO/IEC 9075-2 基础（SQL/Foundation）
- ISO/IEC 9075-3 调用级接口（SQL/CLI）
- ISO/IEC 9075-4 持久化存储模块（SQL/PSM）
- ISO/IEC 9075-9 外部数据管理（SQL/MED）
- ISO/IEC 9075-10 对象语言绑定（SQL/OLB）
- ISO/IEC 9075-11 信息与定义模式（SQL/Schemata）
- ISO/IEC 9075-13 使用 Java 语言的例程和类型（SQL/JRT）
- ISO/IEC 9075-14 XML 相关规范（SQL/XML）

PostgreSQL核心部分涵盖第 1、2、9、11 和 14 部分。第 3 部分由 ODBC 驱动实现，第 13 部分由 PL/Java 插件实现，但目前尚未核验这些组件的确切符合性。PostgreSQL目前还没有第 4 和 10 部分的实现。

PostgreSQL 支持 SQL:2011 的大多数主要特性。在达到完整 Core 符合性所要求的 179 个强制特性中，PostgreSQL 至少符合其中 160 个。此外，还有一长串已支持的可选特性。值得一提的是，在写作本文时，还没有任何数据库管理系统的现行版本宣称完全符合 SQL:2011 的核心要求。

接下来的两节先列出PostgreSQL已支持的特性，再列出SQL:2011中定义但PostgreSQL尚未支持的特性。

这两个列表都只是近似描述：某个被列为已支持的特性，
可能仍有少量细节并不完全符合标准；反之，一个被列为不支持的特性，
其很大一部分实际上也可能已经实现。关于哪些内容可用、哪些内容不可用，
文档正文始终是最准确的信息来源。

注意

带连字符的特性代码表示子特性。因此，如果某个特定子特性不受支持，
则即使该主特性的其他子特性受支持，该主特性也会被列为不受支持。

D.1. 支持的特性

标识符	包	描述	注释
B012		Embedded C	
B021		Direct SQL	
E011	Core	Numeric data types	
E011-01	Core	INTEGER and SMALLINT data types	
E011-02	Core	REAL, DOUBLE PRECISION, and FLOAT data types	
E011-03	Core	DECIMAL and NUMERIC data types	
E011-04	Core	Arithmetic operators	
E011-05	Core	Numeric comparison	
E011-06	Core	Implicit casting among the numeric data types	
E021	Core	Character data types	
E021-01	Core	CHARACTER data type	
E021-02	Core	CHARACTER VARYING data type	
E021-03	Core	Character literals	
E021-04	Core	CHARACTER_LENGTH function	trims trailing spaces from CHARACTER values before counting
E021-05	Core	OCTET_LENGTH function	
E021-06	Core	SUBSTRING function	
E021-07	Core	Character concatenation	
E021-08	Core	UPPER and LOWER functions	
E021-09	Core	TRIM function	
E021-10	Core	Implicit casting among the character string types	
E021-11	Core	POSITION function	
E021-12	Core	Character comparison	
E031	Core	Identifiers	

标识符	包	描述	注释
E031-01	Core	Delimited identifiers	
E031-02	Core	Lower case identifiers	
E031-03	Core	Trailing underscore	
E051	Core	Basic query specif ication	
E051-01	Core	SELECT DISTINCT	
E051-02	Core	GROUP BY clause	
E051-04	Core	GROUP BY can contain columns not in <select list>	
E051-05	Core	Select list items can be renamed	
E051-06	Core	HAVING clause	
E051-07	Core	Qualified * in select list	
E051-08	Core	Correlation names in the FROM clause	
E051-09	Core	Rename columns in the FROM clause	
E061	Core	Basic predicates and search conditions	
E061-01	Core	Comparison predicate	
E061-02	Core	BETWEEN predicate	
E061-03	Core	IN predicate with list of values	
E061-04	Core	LIKE predicate	
E061-05	Core	LIKE predicate ESCAPE clause	
E061-06	Core	NULL predicate	
E061-07	Core	Quantified compar ison predicate	
E061-08	Core	EXISTS predicate	
E061-09	Core	Subqueries in compar ison predicate	
E061-11	Core	Subqueries in IN predicate	
E061-12	Core	Subqueries in quanti fied comparison predicate	
E061-13	Core	Correlated subqueries	
E061-14	Core	Search condition	
E071	Core	Basic query expres sions	
E071-01	Core	UNION DISTINCT table operator	
E071-02	Core	UNION ALL table operator	
E071-03	Core	EXCEPT DISTINCT table operator	

标识符	包	描述	注释
E071-05	Core	Columns combined via table operators need not have exactly the same data type	
E071-06	Core	Table operators in subqueries	
E081	Core	Basic Privileges	
E081-01	Core	SELECT privilege	
E081-02	Core	DELETE privilege	
E081-03	Core	INSERT privilege at the table level	
E081-04	Core	UPDATE privilege at the table level	
E081-05	Core	UPDATE privilege at the column level	
E081-06	Core	REFERENCES privilege at the table level	
E081-07	Core	REFERENCES privilege at the column level	
E081-08	Core	WITH GRANT OPTION	
E081-09	Core	USAGE privilege	
E081-10	Core	EXECUTE privilege	
E091	Core	Set functions	
E091-01	Core	AVG	
E091-02	Core	COUNT	
E091-03	Core	MAX	
E091-04	Core	MIN	
E091-05	Core	SUM	
E091-06	Core	ALL quantifier	
E091-07	Core	DISTINCT quantifier	
E101	Core	Basic data manipu lation	
E101-01	Core	INSERT statement	
E101-03	Core	Searched UPDATE statement	
E101-04	Core	Searched DELETE statement	
E111	Core	Single row SELECT statement	
E121	Core	Basic cursor support	
E121-01	Core	DECLARE CURSOR	
E121-02	Core	ORDER BY columns need not be in select list	
E121-03	Core	Value expressions in ORDER BY clause	
E121-04	Core	OPEN statement	
E121-06	Core	Positioned UPDATE statement	

标识符	包	描述	注释
E121-07	Core	Positioned DELETE statement	
E121-08	Core	CLOSE statement	
E121-10	Core	FETCH statement implicit NEXT	
E121-17	Core	WITH HOLD cursors	
E131	Core	Null value support (nulls in lieu of values)	
E141	Core	Basic integrity constr aints	
E141-01	Core	NOT NULL constraints	
E141-02	Core	UNIQUE constraints of NOT NULL columns	
E141-03	Core	PRIMARY KEY constr aints	
E141-04	Core	Basic FOREIGN KEY constraint with the NO ACTION default for both referential delete action and referential update action	
E141-06	Core	CHECK constraints	
E141-07	Core	Column defaults	
E141-08	Core	NOT NULL inferred on PRIMARY KEY	
E141-10	Core	Names in a foreign key can be specified in any order	
E151	Core	Transaction support	
E151-01	Core	COMMIT statement	
E151-02	Core	ROLLBACK statement	
E152	Core	Basic SET TRANSA CTION statement	
E152-01	Core	SET TRANSACTION statement: ISOLATION LEVEL SERIALIZABLE clause	
E152-02	Core	SET TRANSACTION statement: READ ONLY and READ WRITE clauses	
E161	Core	SQL comments using leading double minus	
E171	Core	SQLSTATE support	
F021	Core	Basic information schema	
F021-01	Core	COLUMNS view	
F021-02	Core	TABLES view	
F021-03	Core	VIEWS view	
F021-04	Core	TABLE_CONSTRAINTS view	

标识符	包	描述	注释
F021-05	Core	REFERENTIAL_CONSTRAINTS view	
F021-06	Core	CHECK_CONSTRAINTS view	
F031	Core	Basic schema manipulation	
F031-01	Core	CREATE TABLE statement to create persistent base tables	
F031-02	Core	CREATE VIEW statement	
F031-03	Core	GRANT statement	
F031-04	Core	ALTER TABLE statement: ADD COLUMN clause	
F031-13	Core	DROP TABLE statement: RESTRICT clause	
F031-16	Core	DROP VIEW statement: RESTRICT clause	
F031-19	Core	REVOKE statement: RESTRICT clause	
F032		CASCADE drop behavior	
F033		ALTER TABLE statement: DROP COLUMN clause	
F034		Extended REVOKE statement	
F034-01		REVOKE statement performed by other than the owner of a schema object	
F034-02		REVOKE statement: GRANT OPTION FOR clause	
F034-03		REVOKE statement to revoke a privilege that the grantee has WITH GRANT OPTION	
F041	Core	Basic joined table	
F041-01	Core	Inner join (but not necessarily the INNER keyword)	
F041-02	Core	INNER keyword	
F041-03	Core	LEFT OUTER JOIN	
F041-04	Core	RIGHT OUTER JOIN	
F041-05	Core	Outer joins can be nested	
F041-07	Core	The inner table in a left or right outer join can also be used in an inner join	

标识符	包	描述	注释
F041-08	Core	All comparison operat ors are supported (rather than just =)	
F051	Core	Basic date and time	
F051-01	Core	DATE data type (includ ing support of DATE literal)	
F051-02	Core	TIME data type (includ ing support of TIME literal) with fractional seconds precision of at least 0	
F051-03	Core	TIMESTAMP data type (including support of TIMESTAMP literal) with fractional second s precision of at least 0 and 6	
F051-04	Core	Comparison predicate on DATE, TIME, and TIMESTAMP data types	
F051-05	Core	Explicit CAST betwee n datetime types and character string types	
F051-06	Core	CURRENT_DATE	
F051-07	Core	LOCALTIME	
F051-08	Core	LOCALTIMESTAMP	
F052	Enhanced datetime facilities	Intervals and dateti me arithmetic	
F053		OVERLAPS predicate	
F081	Core	UNION and EXCEPT in views	
F111		Isolation levels other than SERIALIZABLE	
F111-01		READ UNCOMMITTED isolation level	
F111-02		READ COMMITTED isolation level	
F111-03		REPEATABLE READ isolation level	
F131	Core	Grouped operations	
F131-01	Core	WHERE, GROUP BY, and HAVING clauses supported in queries with grouped views	
F131-02	Core	Multiple tables suppor ted in queries with grouped views	
F131-03	Core	Set functions suppor ted in queries with grouped views	
F131-04	Core	Subqueries with GROUP BY and	

标识符	包	描述	注释
		HAVING clauses and grouped views	
F131-05	Core	Single row SELECT with GROUP BY and HAVING clauses and grouped views	
F171		Multiple schemas per user	
F191	Enhanced integrity management	Referential delete actions	
F200		TRUNCATE TABLE statement	
F201	Core	CAST function	
F221	Core	Explicit defaults	
F222		INSERT statement: DEFAULT VALUES clause	
F231		Privilege tables	
F231-01		TABLE_PRIVILEGES view	
F231-02		COLUMN_PRIVILEGES view	
F231-03		USAGE_PRIVILEGES view	
F251		Domain support	
F261	Core	CASE expression	
F261-01	Core	Simple CASE	
F261-02	Core	Searched CASE	
F261-03	Core	NULLIF	
F261-04	Core	COALESCE	
F262		Extended CASE expres sion	
F271		Compound character literals	
F281		LIKE enhancements	
F302		INTERSECT table operator	
F302-01		INTERSECT DISTINCT table operator	
F302-02		INTERSECT ALL table operator	
F304		EXCEPT ALL table operator	
F311-01	Core	CREATE SCHEMA	
F311-02	Core	CREATE TABLE for persistent base tables	
F311-03	Core	CREATE VIEW	
F311-05	Core	GRANT statement	
F321		User authorization	
F361		Subprogram support	

标识符	包	描述	注释
F381		Extended schema manipulation	
F381-01		ALTER TABLE statem ent: ALTER COLUMN clause	
F381-02		ALTER TABLE statem ent: ADD CONSTRAINT clause	
F381-03		ALTER TABLE statem ent: DROP CONSTR AINT clause	
F382		Alter column data type	
F383		Set column not null clause	
F391		Long identifiers	
F392		Unicode escapes in identifiers	
F393		Unicode escapes in literals	
F401		Extended joined table	
F401-01		NATURAL JOIN	
F401-02		FULL OUTER JOIN	
F401-04		CROSS JOIN	
F402		Named column joins for LOBs, arrays, and multisets	
F411	Enhanced datetime facilities	Time zone specificatio n	differences regarding literal interpretation
F421		National character	
F431		Read-only scrollable cursors	
F431-01		FETCH with explicit NEXT	
F431-02		FETCH FIRST	
F431-03		FETCH LAST	
F431-04		FETCH PRIOR	
F431-05		FETCH ABSOLUTE	
F431-06		FETCH RELATIVE	
F441		Extended set function support	
F442		Mixed column refere nces in set functions	
F471	Core	Scalar subquery values	
F481	Core	Expanded NULL predicate	
F491	Enhanced integrity management	Constraint manage ment	
F501	Core	Features and confor mance views	

标识符	包	描述	注释
F501-01	Core	SQL_FEATURES view	
F501-02	Core	SQL_SIZING view	
F501-03	Core	SQL_LANGUAGES view	
F502		Enhanced docume ntation tables	
F502-01		SQL_SIZING_PROFILES view	
F502-02		SQL_IMPLEMENTATI ON_INFO view	
F502-03		SQL_PACKAGES view	
F531		Temporary tables	
F555	Enhanced datetime facilities	Enhanced seconds precision	
F561		Full value expressions	
F571		Truth value tests	
F591		Derived tables	
F611		Indicator data types	
F641		Row and table constr uctors	
F651		Catalog name qualif iers	
F661		Simple tables	
F672		Retrospective check constraints	
F690		Collation support	but no character set support
F692		Extended collation support	
F701	Enhanced integrity management	Referential update actions	
F711		ALTER domain	
F731		INSERT column privil eges	
F761		Session management	
F762		CURRENT_CATALOG	
F763		CURRENT_SCHEMA	
F771		Connection manage ment	
F781		Self-referencing operations	
F791		Insensitive cursors	
F801		Full set function	
F850		Top-level <order by clause> in <query expression>	
F851		<order by clause> in subqueries	
F852		Top-level <order by clause> in views	

标识符	包	描述	注释
F855		Nested <order by clause> in <query expression>	
F856		Nested <fetch first clause> in <query expression>	
F857		Top-level <fetch first clause> in <query expression>	
F858		<fetch first clause> in subqueries	
F859		Top-level <fetch first clause> in views	
F860		<fetch first row count> in <fetch first clause>	
F861		Top-level <result offset clause> in <query expression>	
F862		<result offset clause> in subqueries	
F863		Nested <result offset clause> in <query expression>	
F864		Top-level <result offset clause> in views	
F865		<offset row count> in <result offset clause>	
S071	Enhanced object support	SQL paths in function and type name resolu tion	
S092		Arrays of user-define d types	
S095		Array constructors by query	
S096		Optional array bounds	
S098		ARRAY_AGG	
S111	Enhanced object support	ONLY in query expres sions	
S201		SQL-invoked routines on arrays	
S201-01		Array parameters	
S201-02		Array as result type of functions	
S211	Enhanced object support	User-defined cast functions	
T031		BOOLEAN data type	
T071		BIGINT data type	
T121		WITH (excluding RECURSIVE) in query expression	

标识符	包	描述	注释
T122		WITH (excluding RECURSIVE) in subquery	
T131		Recursive query	
T132		Recursive query in subquery	
T141		SIMILAR predicate	
T151		DISTINCT predicate	
T152		DISTINCT predicate with negation	
T171		LIKE clause in table definition	
T172		AS subquery clause in table definition	
T173		Extended LIKE clause in table definition	
T191	Enhanced integrity management	Referential action RESTRICT	
T201	Enhanced integrity management	Comparable data types for referential constraints	
T211-01	Active database, Enhanced integrity management	Triggers activated on UPDATE, INSERT, or DELETE of one base table	
T211-02	Active database, Enhanced integrity management	BEFORE triggers	
T211-03	Active database, Enhanced integrity management	AFTER triggers	
T211-04	Active database, Enhanced integrity management	FOR EACH ROW triggers	
T211-05	Active database, Enhanced integrity management	Ability to specify a search condition that must be true before the trigger is invoked	
T211-07	Active database, Enhanced integrity management	TRIGGER privilege	
T212	Enhanced integrity management	Enhanced trigger capability	
T213		INSTEAD OF triggers	
T231		Sensitive cursors	
T241		START TRANSACTION statement	
T271		Savepoints	
T281		SELECT privilege with column granularity	
T312		OVERLAY function	

标识符	包	描述	注释
T321-01	Core	User-defined functi ons with no overlo ading	
T321-03	Core	Function invocation	
T321-06	Core	ROUTINES view	
T321-07	Core	PARAMETERS view	
T323		Explicit security for external routines	
T331		Basic roles	
T341		Overloading of SQL-invoked functions and procedures	
T351		Bracketed SQL comments (/*...*/ comments)	
T441		ABS and MOD functi ons	
T461		Symmetric BETWEEN predicate	
T501		Enhanced EXISTS predicate	
T551		Optional key words for default syntax	
T581		Regular expression substring function	
T591		UNIQUE constraints of possibly null columns	
T614		NTILE function	
T615		LEAD and LAG functi ons	
T617		FIRST_VALUE and LAST_VALUE function	
T621		Enhanced numeric functions	
T631	Core	IN predicate with one list element	
T651		SQL-schema statem ents in SQL routines	
T655		Cyclically dependent routines	
X010		XML type	
X011		Arrays of XML type	
X016		Persistent XML values	
X020		XMLConcat	
X031		XMLElement	
X032		XMLForest	
X034		XMLAgg	
X035		XMLAgg: ORDER BY option	
X036		XMLComment	

标识符	包	描述	注释
X037		XMLPI	
X040		Basic table mapping	
X041		Basic table mapping: nulls absent	
X042		Basic table mapping: null as nil	
X043		Basic table mapping: table as forest	
X044		Basic table mapping: table as element	
X045		Basic table mapping: with target namespace	
X046		Basic table mapping: data mapping	
X047		Basic table mapping: metadata mapping	
X048		Basic table mapping: base64 encoding of binary strings	
X049		Basic table mapping: hex encoding of binary strings	
X050		Advanced table mapping	
X051		Advanced table mapping: nulls absent	
X052		Advanced table mapping: null as nil	
X053		Advanced table mapping: table as forest	
X054		Advanced table mapping: table as element	
X055		Advanced table mapping: target namespace	
X056		Advanced table mapping: data mapping	
X057		Advanced table mapping: metadata mapping	
X058		Advanced table mapping: base64 encoding of binary strings	
X059		Advanced table mapping: hex encoding of binary strings	

标识符	包	描述	注释
X060		XMLParse: Character string input and CONTENT option	
X061		XMLParse: Character string input and DOCUMENT option	
X070		XMLSerialize: Character string serialization and CONTENT option	
X071		XMLSerialize: Character string serialization and DOCUMENT option	
X072		XMLSerialize: Character string serialization	
X090		XML document predicate	
X120		XML parameters in SQL routines	
X121		XML parameters in external routines	
X400		Name and identifier mapping	
X410		Alter column data type: XML type	

D.2. 不支持的特性

下列在SQL:2011中定义的特性，在本版本的PostgreSQL中尚未实现。在少数情况下，可用等效功能替代。

标识符	包	描述	注释
B011		Embedded Ada	
B013		Embedded COBOL	
B014		Embedded Fortran	
B015		Embedded MUMPS	
B016		Embedded Pascal	
B017		Embedded PL/I	
B031		Basic dynamic SQL	
B032		Extended dynamic SQL	
B032-01		<describe input statement>	
B033		Untyped SQL-invoked function arguments	
B034		Dynamic specification of cursor attributes	
B035		Non-extended descriptor names	
B041		Extensions to embedded SQL exception declarations	

标识符	包	描述	注释
B051		Enhanced execution rights	
B111		Module language Ada	
B112		Module language C	
B113		Module language COBOL	
B114		Module language Fortran	
B115		Module language MUMPS	
B116		Module language Pascal	
B117		Module language PL/I	
B121		Routine language Ada	
B122		Routine language C	
B123		Routine language COBOL	
B124		Routine language Fortran	
B125		Routine language MUMPS	
B126		Routine language Pascal	
B127		Routine language PL/I	
B128		Routine language SQL	
B211		Module language Ada: VARCHAR and NUMERIC support	
B221		Routine language Ada: VARCHAR and NUMERIC support	
E153	Core	Updatable queries with subqueries	
E182	Core	Module language	
F054		TIMESTAMP in DATE type precedence list	
F121		Basic diagnostics management	
F121-01		GET DIAGNOSTICS statement	
F121-02		SET TRANSACTION statement: DIAGNO STICS SIZE clause	
F122		Enhanced diagnostics management	
F123		All diagnostics	
F181	Core	Multiple module support	
F202		TRUNCATE TABLE: identity column restar t option	

标识符	包	描述	注释
F263		Comma-separated predicates in simple CASE expression	
F291		UNIQUE predicate	
F301		CORRESPONDING in query expressions	
F311	Core	Schema definition statement	
F311-04	Core	CREATE VIEW: WITH CHECK OPTION	
F312		MERGE statement	
F313		Enhanced MERGE statement	
F314		MERGE statement with DELETE branch	
F341		Usage tables	no ROUTINE_*_USAGE tables
F384		Drop identity property clause	
F385		Drop column genera tion expression clause	
F386		Set identity column generation clause	
F394		Optional normal form specification	
F403		Partitioned joined tables	
F451		Character set defini tion	
F461		Named character sets	
F492		Optional table constr aint enforcement	
F521	Enhanced integrity management	Assertions	
F671	Enhanced integrity management	Subqueries in CHECK	intentionally omitted
F693		SQL-session and client module collat ions	
F695		Translation support	
F696		Additional translation documentation	
F721		Deferrable constraints	foreign and unique keys only
F741		Referential MATCH types	no partial match yet
F751		View CHECK enhanc ements	
F812	Core	Basic flagging	
F813		Extended flagging	
F821		Local table references	

标识符	包	描述	注释
F831		Full cursor update	
F831-01		Updatable scrollable cursors	
F831-02		Updatable ordered cursors	
F841		LIKE_REGEX predicate	
F842		OCCURENCES_REGEX function	
F843		POSITION_REGEX function	
F844		SUBSTRING_REGEX function	
F845		TRANSLATE_REGEX function	
F846		Octet support in regular expression operators	
F847		Nonconstant regular expressions	
F866		FETCH FIRST clause: PERCENT option	
F867		FETCH FIRST clause: WITH TIES option	
S011	Core	Distinct data types	
S011-01	Core	USER_DEFINED_TYPES view	
S023	Basic object support	Basic structured types	
S024	Enhanced object support	Enhanced structured types	
S025		Final structured types	
S026		Self-referencing structured types	
S027		Create method by specific method name	
S028		Permutable UDT options list	
S041	Basic object support	Basic reference types	
S043	Enhanced object support	Enhanced reference types	
S051	Basic object support	Create table of type	partially supported
S081	Enhanced object support	Subtables	
S091		Basic array support	partially supported
S091-01		Arrays of built-in data types	
S091-02		Arrays of distinct types	
S091-03		Array expressions	
S094		Arrays of reference types	

标识符	包	描述	注释
S097		Array element assign ment	
S151	Basic object support	Type predicate	
S161	Enhanced object support	Subtype treatment	
S162		Subtype treatment for references	
S202		SQL-invoked routines on multisets	
S231	Enhanced object support	Structured type locato rs	
S232		Array locators	
S233		Multiset locators	
S241		Transform functions	
S242		Alter transform statem ent	
S251		User-defined orderi ngs	
S261		Specific type method	
S271		Basic multiset support	
S272		Multisets of user-defined types	
S274		Multisets of reference types	
S275		Advanced multiset support	
S281		Nested collection types	
S291		Unique constraint on entire row	
S301		Enhanced UNNEST	
S401		Distinct types based on array types	
S402		Distinct types based on distinct types	
S403		MAX_CARDINALITY	
S404		TRIM_ARRAY	
T011		Timestamp in Inform ation Schema	
T021		BINARY and VARBIN ARY data types	
T022		Advanced support for BINARY and VARBIN ARY data types	
T023		Compound binary literal	
T024		Spaces in binary litera ls	
T041	Basic object support	Basic LOB data type support	

标识符	包	描述	注释
T041-01	Basic object support	BLOB data type	
T041-02	Basic object support	CLOB data type	
T041-03	Basic object support	POSITION, LENGTH, LOWER, TRIM, UPPER, and SUBSTRING functions for LOB data types	
T041-04	Basic object support	Concatenation of LOB data types	
T041-05	Basic object support	LOB locator: non-holdable	
T042		Extended LOB data type support	
T043		Multiplier T	
T044		Multiplier P	
T051		Row types	
T052		MAX and MIN for row types	
T053		Explicit aliases for all-fields reference	
T061		UCS support	
T101		Enhanced nullability determination	
T111		Updatable joins, unions, and columns	
T174		Identity columns	
T175		Generated columns	
T176		Sequence generator support	
T177		Sequence generator support: simple restart option	
T178		Identity columns: simple restart option	
T180		System-versioned tables	
T181		Application-time period tables	
T211	Active database, Enhanced integrity management	Basic trigger capability	
T211-06	Active database, Enhanced integrity management	Support for run-time rules for the interaction of triggers and constraints	
T211-08	Active database, Enhanced integrity management	Multiple triggers for the same event are executed in the order in which they were created in the catalog	intentionally omitted
T251		SET TRANSACTION statement: LOCAL option	

标识符	包	描述	注释
T261		Chained transactions	
T272		Enhanced savepoint management	
T285		Enhanced derived column names	
T301		Functional depend encies	partially supported
T321	Core	Basic SQL-invoked routines	
T321-02	Core	User-defined stored procedures with no overloading	
T321-04	Core	CALL statement	
T321-05	Core	RETURN statement	
T322	PSM	Declared data type attributes	
T324		Explicit security for SQL routines	
T325		Qualified SQL parame ter references	
T326		Table functions	
T332		Extended roles	mostly supported
T431	OLAP	Extended grouping capabilities	
T432		Nested and concat enated GROUPING SETS	
T433		Multiargument GROUPING function	
T434		GROUP BY DISTINCT	
T471		Result sets return value	
T472		DESCRIBE CURSOR	
T491		LATERAL derived table	
T495		Combined data change and retrieval	different syntax
T502		Period predicates	
T511		Transaction counts	
T521		Named arguments in CALL statement	
T522		Default values for IN parameters of SQL-invoked procedures	
T541		Updatable table references	
T561		Holdable locators	
T571		Array-returning extern al SQL-invoked functi ons	

标识符	包	描述	注释
T572		Multiset-returning external SQL-invoked functions	
T601		Local cursor refere nces	
T611	OLAP	Elementary OLAP operations	most forms supported
T612		Advanced OLAP operations	some forms suppor ted
T613		Sampling	
T616		Null treatment option for LEAD and LAG functions	
T618		NTH_VALUE function	function exists, but some options missing
T619		Nested window functi ons	
T620		WINDOW clause: GROUPS option	
T641		Multiple column assignment	only some syntax variants supported
T652		SQL-dynamic statem ents in SQL routines	
T653		SQL-schema statem ents in external routin es	
T654		SQL-dynamic statem ents in external routin es	
M001		Datalinks	
M002		Datalinks via SQL/CLI	
M003		Datalinks via Embedd ed SQL	
M004		Foreign data support	partially supported
M005		Foreign schema support	
M006		GetSQLString routine	
M007		TransmitRequest	
M009		GetOpts and GetStatis tics routines	
M010		Foreign data wrapper support	
M011		Datalinks via Ada	
M012		Datalinks via C	
M013		Datalinks via COBOL	
M014		Datalinks via Fortran	
M015		Datalinks via M	
M016		Datalinks via Pascal	
M017		Datalinks via PL/I	

标识符	包	描述	注释
M018		Foreign data wrappe r interface routines in Ada	
M019		Foreign data wrapper interface routines in C	
M020		Foreign data wrappe r interface routines in COBOL	
M021		Foreign data wrappe r interface routines in Fortran	
M022		Foreign data wrappe r interface routines in MUMPS	
M023		Foreign data wrappe r interface routines in Pascal	
M024		Foreign data wrappe r interface routines in PL/I	
M030		SQL-server foreign data support	
M031		Foreign data wrapper general routines	
X012		Multisets of XML type	
X013		Distinct types of XML type	
X014		Attributes of XML type	
X015		Fields of XML type	
X025		XMLCast	
X030		XMLDocument	
X038		XMLText	
X065		XMLParse: BLOB input and CONTENT option	
X066		XMLParse: BLOB input and DOCUMENT option	
X068		XMLSerialize: BOM	
X069		XMLSerialize: INDENT	
X073		XMLSerialize: BLOB serialization and CONTENT option	
X074		XMLSerialize: BLOB serialization and DOCUMENT option	
X075		XMLSerialize: BLOB serialization	
X076		XMLSerialize: VERSIO N	
X077		XMLSerialize: explicit ENCODING option	

标识符	包	描述	注释
X078		XMLSerialize: explicit XML declaration	
X080		Namespaces in XML publishing	
X081		Query-level XML namespace declar ations	
X082		XML namespace declarations in DML	
X083		XML namespace declarations in DDL	
X084		XML namespace declarations in compound statements	
X085		Predefined namesp ace prefixes	
X086		XML namespace declarations in XMLTable	
X091		XML content predicate	
X096		XMLExists	
X100		Host language suppor t for XML: CONTENT option	
X101		Host language suppor t for XML: DOCUMENT option	
X110		Host language suppor t for XML: VARCHAR mapping	
X111		Host language suppor t for XML: CLOB mapping	
X112		Host language suppor t for XML: BLOB mapping	
X113		Host language suppor t for XML: STRIP WHITESPACE option	
X114		Host language suppor t for XML: PRESERVE WHITESPACE option	
X131		Query-level XMLBIN ARY clause	
X132		XMLBINARY clause in DML	
X133		XMLBINARY clause in DDL	
X134		XMLBINARY clause in compound statements	
X135		XMLBINARY clause in subqueries	

标识符	包	描述	注释
X141		IS VALID predicate: data-driven case	
X142		IS VALID predicate: ACCORDING TO clause	
X143		IS VALID predicate: ELEMENT clause	
X144		IS VALID predicate: schema location	
X145		IS VALID predicate outside check constr aints	
X151		IS VALID predicate with DOCUMENT option	
X152		IS VALID predicate with CONTENT option	
X153		IS VALID predicate with SEQUENCE option	
X155		IS VALID predicate: NAMESPACE without ELEMENT clause	
X157		IS VALID predicate: NO NAMESPACE with ELEMENT clause	
X160		Basic Information Schema for registered XML Schemas	
X161		Advanced Information Schema for registered XML Schemas	
X170		XML null handling options	
X171		NIL ON NO CONTENT option	
X181		XML(DOCUMENT(UNTYPED)) type	
X182		XML(DOCUMENT(ANY)) type	
X190		XML(SEQUENCE) type	
X191		XML(DOCUMENT(XMLSCHEMA)) type	
X192		XML(CONTENT(XMLSCHEMA)) type	
X200		XMLQuery	
X201		XMLQuery: RETURN ING CONTENT	
X202		XMLQuery: RETURN ING SEQUENCE	
X203		XMLQuery: passing a context item	
X204		XMLQuery: initializing an XQuery variable	

标识符	包	描述	注释
X205		XMLQuery: EMPTY ON EMPTY option	
X206		XMLQuery: NULL ON EMPTY option	
X211		XML 1.1 support	
X221		XML passing mechanism BY VALUE	
X222		XML passing mechanism BY REF	
X231		XML(CONTENT(UNTYPED)) type	
X232		XML(CONTENT(ANY)) type	
X241		RETURNING CONTENT in XML publishing	
X242		RETURNING SEQUENCE in XML publishing	
X251		Persistent XML values of XML(DOCUMENT(UNTYPED)) type	
X252		Persistent XML values of XML(DOCUMENT(ANY)) type	
X253		Persistent XML values of XML(CONTENT(UNTYPED)) type	
X254		Persistent XML values of XML(CONTENT(ANY)) type	
X255		Persistent XML values of XML(SEQUENCE) type	
X256		Persistent XML values of XML(DOCUMENT(XMLSCHEMA)) type	
X257		Persistent XML values of XML(CONTENT(XMLSCHEMA)) type	
X260		XML type: ELEMENT clause	
X261		XML type: NAMESPACE without ELEMENT clause	
X263		XML type: NO NAMESPACE with ELEMENT clause	
X264		XML type: schema location	
X271		XMLValidate: data-driven case	
X272		XMLValidate: ACCORDING TO clause	

标识符	包	描述	注释
X273		XMLValidate: ELEMENT clause	
X274		XMLValidate: schema location	
X281		XMLValidate: with DOCUMENT option	
X282		XMLValidate with CONTENT option	
X283		XMLValidate with SEQUENCE option	
X284		XMLValidate NAMESPACE without ELEMENT clause	
X286		XMLValidate: NO NAMESPACE with ELEMENT clause	
X300		XMLTable	
X301		XMLTable: derived column list option	
X302		XMLTable: ordinality column option	
X303		XMLTable: column default option	
X304		XMLTable: passing a context item	
X305		XMLTable: initializing an XQuery variable	

附录 E. 发布说明

发布说明包含每个PostgreSQL发布中的重要变更，并将主要特性和迁移问题列在最前面。发布说明不包含只影响少数用户的变更，也不包含仅限内部实现、因而对用户不可见的变更。例如，优化器几乎在每次发布中都会得到改进，但用户通常只会把这些改进感知为查询速度更快。

每个发布的完整变更列表都可以通过查看对应版本的Git 日志获得。`pgsql-committers` 邮件列表[1]也记录了所有源代码变更。另有一个Web 界面[2]可显示特定文件的变更。

每个条目旁边标出的姓名表示该条目的主要开发者。当然，所有变更都经历了社区讨论和补丁审查，因此每个条目实际上都是社区共同努力的成果。

E.1. Release 9.2.24

Release date: 2017-11-09

This release contains a variety of fixes from 9.2.23. For information about new features in the 9.2 major release, see 第 E.25 节.

This is expected to be the last PostgreSQL release in the 9.2.X series. Users are encouraged to update to a newer release branch soon.

E.1.1. Migration to Version 9.2.24

A dump/restore is not required for those running 9.2.X.

However, if you are upgrading from a version earlier than 9.2.22, see 第 E.3 节.

E.1.2. Changes

- Fix sample server-start scripts to become `$PGUSER` before opening `$PGLOG` (Noah Misch)

  Previously, the postmaster log file was opened while still running as root. The database owner could therefore mount an attack against another system user by making `$PGLOG` be a symbolic link to some other file, which would then become corrupted by appending log messages.

  By default, these scripts are not installed anywhere. Users who have made use of them will need to manually recopy them, or apply the same changes to their modified versions. If the existing `$PGLOG` file is root-owned, it will need to be removed or renamed out of the way before restarting the server with the corrected script. (CVE-2017-12172)

- Properly reject attempts to convert infinite float values to type `numeric` (Tom Lane, KaiGai Kohei)

  Previously the behavior was platform-dependent.

- Fix corner-case crashes when columns have been added to the end of a view (Tom Lane)

[1] https://archives.postgresql.org/pgsql-committers/
[2] https://git.postgresql.org/gitweb?p=postgresql.git;a=summary

- Record proper dependencies when a view or rule contains `FieldSelect` or `FieldStore` expression nodes (Tom Lane)

  Lack of these dependencies could allow a column or data type `DROP` to go through when it ought to fail, thereby causing later uses of the view or rule to get errors. This patch does not do anything to protect existing views/rules, only ones created in the future.

- Correctly detect hashability of range data types (Tom Lane)

  The planner mistakenly assumed that any range type could be hashed for use in hash joins or hash aggregation, but actually it must check whether the range's subtype has hash support. This does not affect any of the built-in range types, since they're all hashable anyway.

- Fix low-probability loss of `NOTIFY` messages due to XID wraparound (Marko Tiikkaja, Tom Lane)

  If a session executed no queries, but merely listened for notifications, for more than 2 billion transactions, it started to miss some notifications from concurrently-committing transactions.

- Prevent low-probability crash in processing of nested trigger firings (Tom Lane)

- Correctly restore the umask setting when file creation fails in `COPY` or `lo_export()` (Peter Eisentraut)

- Give a better error message for duplicate column names in `ANALYZE` (Nathan Bossart)

- Fix libpq to not require user's home directory to exist (Tom Lane)

  In v10, failure to find the home directory while trying to read `~/.pgpass` was treated as a hard error, but it should just cause that file to not be found. Both v10 and previous release branches made the same mistake when reading `~/.pg_service.conf`, though this was less obvious since that file is not sought unless a service name is specified.

- Fix libpq to guard against integer overflow in the row count of a `PGresult` (Michael Paquier)

- Sync our copy of the timezone library with IANA release tzcode2017c (Tom Lane)

  This fixes various issues; the only one likely to be user-visible is that the default DST rules for a POSIX-style zone name, if no `posixrules` file exists in the timezone data directory, now match current US law rather than what it was a dozen years ago.

- Update time zone data files to tzdata release 2017c for DST law changes in Fiji, Namibia, Northern Cyprus, Sudan, Tonga, and Turks & Caicos Islands, plus historical corrections for Alaska, Apia, Burma, Calcutta, Detroit, Ireland, Namibia, and Pago Pago.

E.2. Release 9.2.23

Release date: 2017-08-31

This release contains a small number of fixes from 9.2.22. For information about new features in the 9.2 major release, see 第 E.25 节.

The PostgreSQL community will stop releasing updates for the 9.2.X release series in September 2017. Users are encouraged to update to a newer release branch soon.

E.2.1. Migration to Version 9.2.23

A dump/restore is not required for those running 9.2.X.

However, if you are upgrading from a version earlier than 9.2.22, see 第 E.3 节.

E.2.2. Changes

- Show foreign tables in `information_schema.table_privileges` view (Peter Eisentraut)

  All other relevant `information_schema` views include foreign tables, but this one ignored them.

  Since this view definition is installed by initdb, merely upgrading will not fix the problem. If you need to fix this in an existing installation, you can, as a superuser, do this in psql:

```
SET search_path TO information_schema;
CREATE OR REPLACE VIEW table_privileges AS
    SELECT CAST(u_grantor.rolname AS sql_identifier) AS grantor,
           CAST(grantee.rolname AS sql_identifier) AS grantee,
           CAST(current_database() AS sql_identifier) AS table_
catalog,
           CAST(nc.nspname AS sql_identifier) AS table_schema,
           CAST(c.relname AS sql_identifier) AS table_name,
           CAST(c.prtype AS character_data) AS privilege_type,
           CAST(
             CASE WHEN
                  -- object owner always has grant options
                  pg_has_role(grantee.oid, c.relowner, 'USAGE')
                  OR c.grantable
                  THEN 'YES' ELSE 'NO' END AS yes_or_no) AS is_
grantable,
           CAST(CASE WHEN c.prtype = 'SELECT' THEN 'YES' ELSE
 'NO' END AS yes_or_no) AS with_hierarchy

    FROM (
            SELECT oid, relname, relnamespace, relkind, relowner,
 (aclexplode(coalesce(relacl, acldefault('r', relowner)))).* FROM
 pg_class
        ) AS c (oid, relname, relnamespace, relkind, relowner,
 grantor, grantee, prtype, grantable),
        pg_namespace nc,
        pg_authid u_grantor,
        (
          SELECT oid, rolname FROM pg_authid
          UNION ALL
          SELECT 0::oid, 'PUBLIC'
        ) AS grantee (oid, rolname)

    WHERE c.relnamespace = nc.oid
          AND c.relkind IN ('r', 'v', 'f')
          AND c.grantee = grantee.oid
          AND c.grantor = u_grantor.oid
```

```
        AND c.prtype IN ('INSERT', 'SELECT', 'UPDATE',
 'DELETE', 'TRUNCATE', 'REFERENCES', 'TRIGGER')
        AND (pg_has_role(u_grantor.oid, 'USAGE')
             OR pg_has_role(grantee.oid, 'USAGE')
             OR grantee.rolname = 'PUBLIC');
```

This must be repeated in each database to be fixed, including `template0`.

- Clean up handling of a fatal exit (e.g., due to receipt of SIGTERM) that occurs while trying to execute a `ROLLBACK` of a failed transaction (Tom Lane)

  This situation could result in an assertion failure. In production builds, the exit would still occur, but it would log an unexpected message about "cannot drop active portal".

- Remove assertion that could trigger during a fatal exit (Tom Lane)

- Correctly identify columns that are of a range type or domain type over a composite type or domain type being searched for (Tom Lane)

  Certain `ALTER` commands that change the definition of a composite type or domain type are supposed to fail if there are any stored values of that type in the database, because they lack the infrastructure needed to update or check such values. Previously, these checks could miss relevant values that are wrapped inside range types or sub-domains, possibly allowing the database to become inconsistent.

- Change ecpg's parser to allow `RETURNING` clauses without attached C variables (Michael Meskes)

  This allows ecpg programs to contain SQL constructs that use `RETURNING` internally (for example, inside a CTE) rather than using it to define values to be returned to the client.

- Improve selection of compiler flags for PL/Perl on Windows (Tom Lane)

  This fix avoids possible crashes of PL/Perl due to inconsistent assumptions about the width of `time_t` values. A side-effect that may be visible to extension developers is that `_USE_32BIT_TIME_T` is no longer defined globally in PostgreSQL Windows builds. This is not expected to cause problems, because type `time_t` is not used in any PostgreSQL API definitions.

E.3. Release 9.2.22

Release date: 2017-08-10

This release contains a variety of fixes from 9.2.21. For information about new features in the 9.2 major release, see 第 E.25 节.

The PostgreSQL community will stop releasing updates for the 9.2.X release series in September 2017. Users are encouraged to update to a newer release branch soon.

E.3.1. Migration to Version 9.2.22

A dump/restore is not required for those running 9.2.X.

However, if you use foreign data servers that make use of user passwords for authentication, see the first changelog entry below.

Also, if you are upgrading from a version earlier than 9.2.20, see 第 E.5 节.

E.3.2. Changes

- Further restrict visibility of `pg_user_mappings.umoptions`, to protect passwords stored as user mapping options (Noah Misch)

  The fix for CVE-2017-7486 was incorrect: it allowed a user to see the options in her own user mapping, even if she did not have `USAGE` permission on the associated foreign server. Such options might include a password that had been provided by the server owner rather than the user herself. Since `information_schema.user_mapping_options` does not show the options in such cases, `pg_user_mappings` should not either. (CVE-2017-7547)

  By itself, this patch will only fix the behavior in newly initdb'd databases. If you wish to apply this change in an existing database, you will need to do the following:

  1. Restart the postmaster after adding `allow_system_table_mods = true` to `postgresql.conf`. (In versions supporting `ALTER SYSTEM`, you can use that to make the configuration change, but you'll still need a restart.)

  2. In each database of the cluster, run the following commands as superuser:

     ```
     SET search_path = pg_catalog;
     CREATE OR REPLACE VIEW pg_user_mappings AS
         SELECT
             U.oid       AS umid,
             S.oid       AS srvid,
             S.srvname   AS srvname,
             U.umuser    AS umuser,
             CASE WHEN U.umuser = 0 THEN
                 'public'
             ELSE
                 A.rolname
             END AS usename,
             CASE WHEN (U.umuser <> 0 AND A.rolname = current_user
                         AND (pg_has_role(S.srvowner, 'USAGE')
                              OR has_server_privilege(S.oid,
      'USAGE')))
                        OR (U.umuser = 0 AND pg_has_role(S.
     srvowner, 'USAGE'))
                        OR (SELECT rolsuper FROM pg_authid WHERE
      rolname = current_user)
                        THEN U.umoptions
                    ELSE NULL END AS umoptions
         FROM pg_user_mapping U
              LEFT JOIN pg_authid A ON (A.oid = U.umuser) JOIN
             pg_foreign_server S ON (U.umserver = S.oid);
     ```

  3. Do not forget to include the `template0` and `template1` databases, or the vulnerability will still exist in databases you create later. To fix `template0`, you'll need to temporarily make it accept connections. In PostgreSQL 9.5 and later, you can use

     ```
     ALTER DATABASE template0 WITH ALLOW_CONNECTIONS true;
     ```

     and then after fixing `template0`, undo that with

     ```
     ALTER DATABASE template0 WITH ALLOW_CONNECTIONS false;
     ```

     In prior versions, instead use

```
UPDATE pg_database SET datallowconn = true WHERE datname =
 'template0';
UPDATE pg_database SET datallowconn = false WHERE datname =
 'template0';
```

4. Finally, remove the `allow_system_table_mods` configuration setting, and again restart the postmaster.

- Disallow empty passwords in all password-based authentication methods (Heikki Linnakangas)

  libpq ignores empty password specifications, and does not transmit them to the server. So, if a user's password has been set to the empty string, it's impossible to log in with that password via psql or other libpq-based clients. An administrator might therefore believe that setting the password to empty is equivalent to disabling password login. However, with a modified or non-libpq-based client, logging in could be possible, depending on which authentication method is configured. In particular the most common method, `md5`, accepted empty passwords. Change the server to reject empty passwords in all cases. (CVE-2017-7546)

- On Windows, retry process creation if we fail to reserve the address range for our shared memory in the new process (Tom Lane, Amit Kapila)

  This is expected to fix infrequent child-process-launch failures that are probably due to interference from antivirus products.

- Fix low-probability corruption of shared predicate-lock hash table in Windows builds (Thomas Munro, Tom Lane)

- Avoid logging clean closure of an SSL connection as though it were a connection reset (Michael Paquier)

- Prevent sending SSL session tickets to clients (Tom Lane)

  This fix prevents reconnection failures with ticket-aware client-side SSL code.

- Fix code for setting tcp_keepalives_idle on Solaris (Tom Lane)

- Fix statistics collector to honor inquiry messages issued just after a postmaster shutdown and immediate restart (Tom Lane)

  Statistics inquiries issued within half a second of the previous postmaster shutdown were effectively ignored.

- Ensure that the statistics collector's receive buffer size is at least 100KB (Tom Lane)

  This reduces the risk of dropped statistics data on older platforms whose default receive buffer size is less than that.

- Fix possible creation of an invalid WAL segment when a standby is promoted just after it processes an `XLOG_SWITCH` WAL record (Andres Freund)

- Fix SIGHUP and SIGUSR1 handling in walsender processes (Petr Jelinek, Andres Freund)

- Fix unnecessarily slow restarts of walreceiver processes due to race condition in postmaster (Tom Lane)

- Fix cases where an `INSERT` or `UPDATE` assigns to more than one element of a column that is of domain-over-array type (Tom Lane)

- Move autogenerated array types out of the way during `ALTER ... RENAME` (Vik Fearing)

  Previously, we would rename a conflicting autogenerated array type out of the way during `CREATE`; this fix extends that behavior to renaming operations.

- Ensure that `ALTER USER ... SET` accepts all the syntax variants that `ALTER ROLE ... SET` does (Peter Eisentraut)

- Properly update dependency info when changing a datatype I/O function's argument or return type from `opaque` to the correct type (Heikki Linnakangas)

  `CREATE TYPE` updates I/O functions declared in this long-obsolete style, but it forgot to record a dependency on the type, allowing a subsequent `DROP TYPE` to leave broken function definitions behind.

- Reduce memory usage when `ANALYZE` processes a `tsvector` column (Heikki Linnakangas)

- Fix unnecessary precision loss and sloppy rounding when multiplying or dividing `money` values by integers or floats (Tom Lane)

- Tighten checks for whitespace in functions that parse identifiers, such as `regprocedurein()` (Tom Lane)

  Depending on the prevailing locale, these functions could misinterpret fragments of multibyte characters as whitespace.

- Use relevant `#define` symbols from Perl while compiling PL/Perl (Ashutosh Sharma, Tom Lane)

  This avoids portability problems, typically manifesting as a "handshake" mismatch during library load, when working with recent Perl versions.

- In psql, fix failure when `COPY FROM STDIN` is ended with a keyboard EOF signal and then another `COPY FROM STDIN` is attempted (Thomas Munro)

  This misbehavior was observed on BSD-derived platforms (including macOS), but not on most others.

- Fix pg_dump to not emit invalid SQL for an empty operator class (Daniel Gustafsson)

- Fix pg_dump output to stdout on Windows (Kuntal Ghosh)

  A compressed plain-text dump written to stdout would contain corrupt data due to failure to put the file descriptor into binary mode.

- Fix `pg_get_ruledef()` to print correct output for the `ON SELECT` rule of a view whose columns have been renamed (Tom Lane)

  In some corner cases, pg_dump relies on `pg_get_ruledef()` to dump views, so that this error could result in dump/reload failures.

- Fix dumping of function expressions in the `FROM` clause in cases where the expression does not deparse into something that looks like a function call (Tom Lane)

- Fix pg_basebackup output to stdout on Windows (Haribabu Kommi)

  A backup written to stdout would contain corrupt data due to failure to put the file descriptor into binary mode.

- Fix pg_upgrade to ensure that the ending WAL record does not have wal_level = `minimum` (Bruce Momjian)

  This condition could prevent upgraded standby servers from reconnecting.

- Always use `-fPIC`, not `-fpic`, when building shared libraries with gcc (Tom Lane)

  This supports larger extension libraries on platforms where it makes a difference.

- Fix unescaped-braces issue in our build scripts for Microsoft MSVC, to avoid a warning or error from recent Perl versions (Andrew Dunstan)

- In MSVC builds, handle the case where the openssl library is not within a `VC` subdirectory (Andrew Dunstan)

- In MSVC builds, add proper include path for libxml2 header files (Andrew Dunstan)

  This fixes a former need to move things around in standard Windows installations of libxml2.

- In MSVC builds, recognize a Tcl library that is named `tcl86.lib` (Noah Misch)

E.4. Release 9.2.21

Release date: 2017-05-11

This release contains a variety of fixes from 9.2.20. For information about new features in the 9.2 major release, see 第 E.25 节.

The PostgreSQL community will stop releasing updates for the 9.2.X release series in September 2017. Users are encouraged to update to a newer release branch soon.

E.4.1. Migration to Version 9.2.21

A dump/restore is not required for those running 9.2.X.

However, if you use foreign data servers that make use of user passwords for authentication, see the first changelog entry below.

Also, if you are upgrading from a version earlier than 9.2.20, see 第 E.5 节.

E.4.2. Changes

- Restrict visibility of `pg_user_mappings.umoptions`, to protect passwords stored as user mapping options (Michael Paquier, Feike Steenbergen)

  The previous coding allowed the owner of a foreign server object, or anyone he has granted server `USAGE` permission to, to see the options for all user mappings associated with that server. This might well include passwords for other users. Adjust the view definition to match the behavior of `information_schema.user_mapping_options`, namely that these options are visible to the user being mapped, or if the mapping is for `PUBLIC` and the current user is the server owner, or if the current user is a superuser. (CVE-2017-7486)

  By itself, this patch will only fix the behavior in newly initdb'd databases. If you wish to apply this change in an existing database, follow the corrected procedure shown in the changelog entry for CVE-2017-7547, in 第 E.3 节.

- Prevent exposure of statistical information via leaky operators (Peter Eisentraut)

Some selectivity estimation functions in the planner will apply user-defined operators to values obtained from `pg_statistic`, such as most common values and histogram entries. This occurs before table permissions are checked, so a nefarious user could exploit the behavior to obtain these values for table columns he does not have permission to read. To fix, fall back to a default estimate if the operator's implementation function is not certified leak-proof and the calling user does not have permission to read the table column whose statistics are needed. At least one of these criteria is satisfied in most cases in practice. (CVE-2017-7484)

- Fix possible corruption of "init forks" of unlogged indexes (Robert Haas, Michael Paquier)

  This could result in an unlogged index being set to an invalid state after a crash and restart. Such a problem would persist until the index was dropped and rebuilt.

- Fix incorrect reconstruction of `pg_subtrans` entries when a standby server replays a prepared but uncommitted two-phase transaction (Tom Lane)

  In most cases this turned out to have no visible ill effects, but in corner cases it could result in circular references in `pg_subtrans`, potentially causing infinite loops in queries that examine rows modified by the two-phase transaction.

- Ensure parsing of queries in extension scripts sees the results of immediately-preceding DDL (Julien Rouhaud, Tom Lane)

  Due to lack of a cache flush step between commands in an extension script file, non-utility queries might not see the effects of an immediately preceding catalog change, such as `ALTER TABLE ... RENAME`.

- Skip tablespace privilege checks when `ALTER TABLE ... ALTER COLUMN TYPE` rebuilds an existing index (Noah Misch)

  The command failed if the calling user did not currently have `CREATE` privilege for the tablespace containing the index. That behavior seems unhelpful, so skip the check, allowing the index to be rebuilt where it is.

- Fix `ALTER TABLE ... VALIDATE CONSTRAINT` to not recurse to child tables when the constraint is marked `NO INHERIT` (Amit Langote)

  This fix prevents unwanted "constraint does not exist" failures when no matching constraint is present in the child tables.

- Fix `VACUUM` to account properly for pages that could not be scanned due to conflicting page pins (Andrew Gierth)

  This tended to lead to underestimation of the number of tuples in the table. In the worst case of a small heavily-contended table, `VACUUM` could incorrectly report that the table contained no tuples, leading to very bad planning choices.

- Ensure that bulk-tuple-transfer loops within a hash join are interruptible by query cancel requests (Tom Lane, Thomas Munro)

- Fix `cursor_to_xml()` to produce valid output with *`tableforest`* = false (Thomas Munro, Peter Eisentraut)

  Previously it failed to produce a wrapping `<table>` element.

- Improve performance of `pg_timezone_names` view (Tom Lane, David Rowley)

- Fix sloppy handling of corner-case errors from `lseek()` and `close()` (Tom Lane)

Neither of these system calls are likely to fail in typical situations, but if they did, `fd.c` could get quite confused.

- Fix incorrect check for whether postmaster is running as a Windows service (Michael Paquier)

  This could result in attempting to write to the event log when that isn't accessible, so that no logging happens at all.

- Fix ecpg to support `COMMIT PREPARED` and `ROLLBACK PREPARED` (Masahiko Sawada)

- Fix a double-free error when processing dollar-quoted string literals in ecpg (Michael Meskes)

- In pg_dump, fix incorrect schema and owner marking for comments and security labels of some types of database objects (Giuseppe Broccolo, Tom Lane)

  In simple cases this caused no ill effects; but for example, a schema-selective restore might omit comments it should include, because they were not marked as belonging to the schema of their associated object.

- Avoid emitting an invalid list file in `pg_restore -l` when SQL object names contain newlines (Tom Lane)

  Replace newlines by spaces, which is sufficient to make the output valid for `pg_restore -L`'s purposes.

- Fix pg_upgrade to transfer comments and security labels attached to "large objects" (blobs) (Stephen Frost)

  Previously, blobs were correctly transferred to the new database, but any comments or security labels attached to them were lost.

- Improve error handling in `contrib/adminpack`'s `pg_file_write()` function (Noah Misch)

  Notably, it failed to detect errors reported by `fclose()`.

- In `contrib/dblink`, avoid leaking the previous unnamed connection when establishing a new unnamed connection (Joe Conway)

- Support OpenSSL 1.1.0 (Heikki Linnakangas, Andreas Karlsson, Tom Lane)

  This is a back-patch of work previously done in newer branches; it's needed since many platforms are adopting newer OpenSSL versions.

- Support Tcl 8.6 in MSVC builds (Álvaro Herrera)

- Sync our copy of the timezone library with IANA release tzcode2017b (Tom Lane)

  This fixes a bug affecting some DST transitions in January 2038.

- Update time zone data files to tzdata release 2017b for DST law changes in Chile, Haiti, and Mongolia, plus historical corrections for Ecuador, Kazakhstan, Liberia, and Spain. Switch to numeric abbreviations for numerous time zones in South America, the Pacific and Indian oceans, and some Asian and Middle Eastern countries.

  The IANA time zone database previously provided textual abbreviations for all time zones, sometimes making up abbreviations that have little or no currency among the local population. They are in process of reversing that policy in favor of using

numeric UTC offsets in zones where there is no evidence of real-world use of an English abbreviation. At least for the time being, PostgreSQL will continue to accept such removed abbreviations for timestamp input. But they will not be shown in the `pg_timezone_names` view nor used for output.

- Use correct daylight-savings rules for POSIX-style time zone names in MSVC builds (David Rowley)

  The Microsoft MSVC build scripts neglected to install the `posixrules` file in the timezone directory tree. This resulted in the timezone code falling back to its built-in rule about what DST behavior to assume for a POSIX-style time zone name. For historical reasons that still corresponds to the DST rules the USA was using before 2007 (i.e., change on first Sunday in April and last Sunday in October). With this fix, a POSIX-style zone name will use the current and historical DST transition dates of the `US/Eastern` zone. If you don't want that, remove the `posixrules` file, or replace it with a copy of some other zone file (see 第 8.5.3 节). Note that due to caching, you may need to restart the server to get such changes to take effect.

E.5. Release 9.2.20

Release date: 2017-02-09

This release contains a variety of fixes from 9.2.19. For information about new features in the 9.2 major release, see 第 E.25 节.

E.5.1. Migration to Version 9.2.20

A dump/restore is not required for those running 9.2.X.

However, if your installation has been affected by the bug described in the first changelog entry below, then after updating you may need to take action to repair corrupted indexes.

Also, if you are upgrading from a version earlier than 9.2.11, see 第 E.14 节.

E.5.2. Changes

- Fix a race condition that could cause indexes built with `CREATE INDEX CONCURRENTLY` to be corrupt (Pavan Deolasee, Tom Lane)

  If `CREATE INDEX CONCURRENTLY` was used to build an index that depends on a column not previously indexed, then rows updated by transactions that ran concurrently with the `CREATE INDEX` command could have received incorrect index entries. If you suspect this may have happened, the most reliable solution is to rebuild affected indexes after installing this update.

- Unconditionally WAL-log creation of the "init fork" for an unlogged table (Michael Paquier)

  Previously, this was skipped when wal_level = `minimal`, but actually it's necessary even in that case to ensure that the unlogged table is properly reset to empty after a crash.

- Fix WAL page header validation when re-reading segments (Takayuki Tsunakawa, Amit Kapila)

  In corner cases, a spurious "out-of-sequence TLI" error could be reported during recovery.

- If the stats collector dies during hot standby, restart it (Takayuki Tsunakawa)
- Check for interrupts while hot standby is waiting for a conflicting query (Simon Riggs)
- Avoid constantly respawning the autovacuum launcher in a corner case (Amit Khandekar)

  This fix avoids problems when autovacuum is nominally off and there are some tables that require freezing, but all such tables are already being processed by autovacuum workers.
- Fix check for when an extension member object can be dropped (Tom Lane)

  Extension upgrade scripts should be able to drop member objects, but this was disallowed for serial-column sequences, and possibly other cases.
- Make sure `ALTER TABLE` preserves index tablespace assignments when rebuilding indexes (Tom Lane, Michael Paquier)

  Previously, non-default settings of default_tablespace could result in broken indexes.
- Prevent dropping a foreign-key constraint if there are pending trigger events for the referenced relation (Tom Lane)

  This avoids "could not find trigger *NNN*" or "relation *NNN* has no triggers" errors.
- Fix processing of OID column when a table with OIDs is associated to a parent with OIDs via `ALTER TABLE ... INHERIT` (Amit Langote)

  The OID column should be treated the same as regular user columns in this case, but it wasn't, leading to odd behavior in later inheritance changes.
- Check for serializability conflicts before reporting constraint-violation failures (Thomas Munro)

  When using serializable transaction isolation, it is desirable that any error due to concurrent transactions should manifest as a serialization failure, thereby cueing the application that a retry might succeed. Unfortunately, this does not reliably happen for duplicate-key failures caused by concurrent insertions. This change ensures that such an error will be reported as a serialization error if the application explicitly checked for the presence of a conflicting key (and did not find it) earlier in the transaction.
- Ensure that column typmods are determined accurately for multi-row `VALUES` constructs (Tom Lane)

  This fixes problems occurring when the first value in a column has a determinable typmod (e.g., length for a `varchar` value) but later values don't share the same limit.
- Throw error for an unfinished Unicode surrogate pair at the end of a Unicode string (Tom Lane)

  Normally, a Unicode surrogate leading character must be followed by a Unicode surrogate trailing character, but the check for this was missed if the leading character was the last character in a Unicode string literal (`U&'...'`) or Unicode identifier (`U&"..."`).
- Ensure that a purely negative text search query, such as `!foo`, matches empty `tsvector`s (Tom Dunstan)

Such matches were found by GIN index searches, but not by sequential scans or GiST index searches.

- Prevent crash when `ts_rewrite()` replaces a non-top-level subtree with an empty query (Artur Zakirov)
- Fix performance problems in `ts_rewrite()` (Tom Lane)
- Fix `ts_rewrite()`'s handling of nested NOT operators (Tom Lane)
- Fix `array_fill()` to handle empty arrays properly (Tom Lane)
- Fix one-byte buffer overrun in `quote_literal_cstr()` (Heikki Linnakangas)

  The overrun occurred only if the input consisted entirely of single quotes and/or backslashes.

- Prevent multiple calls of `pg_start_backup()` and `pg_stop_backup()` from running concurrently (Michael Paquier)

  This avoids an assertion failure, and possibly worse things, if someone tries to run these functions in parallel.

- Avoid discarding `interval`-to-`interval` casts that aren't really no-ops (Tom Lane)

  In some cases, a cast that should result in zeroing out low-order `interval` fields was mistakenly deemed to be a no-op and discarded. An example is that casting from `INTERVAL MONTH` to `INTERVAL YEAR` failed to clear the months field.

- Fix pg_dump to dump user-defined casts and transforms that use built-in functions (Stephen Frost)
- Fix possible pg_basebackup failure on standby server when including WAL files (Amit Kapila, Robert Haas)
- Ensure that the Python exception objects we create for PL/Python are properly reference-counted (Rafa de la Torre, Tom Lane)

  This avoids failures if the objects are used after a Python garbage collection cycle has occurred.

- Fix PL/Tcl to support triggers on tables that have `.tupno` as a column name (Tom Lane)

  This matches the (previously undocumented) behavior of PL/Tcl's `spi_exec` and `spi_execp` commands, namely that a magic `.tupno` column is inserted only if there isn't a real column named that.

- Allow DOS-style line endings in `~/.pgpass` files, even on Unix (Vik Fearing)

  This change simplifies use of the same password file across Unix and Windows machines.

- Fix one-byte buffer overrun if ecpg is given a file name that ends with a dot (Takayuki Tsunakawa)
- Fix psql's tab completion for `ALTER DEFAULT PRIVILEGES` (Gilles Darold, Stephen Frost)
- In psql, treat an empty or all-blank setting of the `PAGER` environment variable as meaning “no pager” (Tom Lane)

Previously, such a setting caused output intended for the pager to vanish entirely.

- Improve `contrib/dblink`'s reporting of low-level libpq errors, such as out-of-memory (Joe Conway)
- On Windows, ensure that environment variable changes are propagated to DLLs built with debug options (Christian Ullrich)
- Sync our copy of the timezone library with IANA release tzcode2016j (Tom Lane)

  This fixes various issues, most notably that timezone data installation failed if the target directory didn't support hard links.
- Update time zone data files to tzdata release 2016j for DST law changes in northern Cyprus (adding a new zone Asia/Famagusta), Russia (adding a new zone Europe/Saratov), Tonga, and Antarctica/Casey. Historical corrections for Italy, Kazakhstan, Malta, and Palestine. Switch to preferring numeric zone abbreviations for Tonga.

E.6. Release 9.2.19

Release date: 2016-10-27

This release contains a variety of fixes from 9.2.18. For information about new features in the 9.2 major release, see 第 E.25 节.

E.6.1. Migration to Version 9.2.19

A dump/restore is not required for those running 9.2.X.

However, if you are upgrading from a version earlier than 9.2.11, see 第 E.14 节.

E.6.2. Changes

- Fix EvalPlanQual rechecks involving CTE scans (Tom Lane)

  The recheck would always see the CTE as returning no rows, typically leading to failure to update rows that were recently updated.
- Fix improper repetition of previous results from hashed aggregation in a subquery (Andrew Gierth)

  The test to see if we can reuse a previously-computed hash table of the aggregate state values neglected the possibility of an outer query reference appearing in an aggregate argument expression. A change in the value of such a reference should lead to recalculating the hash table, but did not.
- Fix `EXPLAIN` to emit valid XML when track_io_timing is on (Markus Winand)

  Previously the XML output-format option produced syntactically invalid tags such as `<I/O-Read-Time>`. That is now rendered as `<I-O-Read-Time>`.
- Suppress printing of zeroes for unmeasured times in `EXPLAIN` (Maksim Milyutin)

  Certain option combinations resulted in printing zero values for times that actually aren't ever measured in that combination. Our general policy in `EXPLAIN` is not to print such fields at all, so do that consistently in all cases.
- Fix timeout length when `VACUUM` is waiting for exclusive table lock so that it can truncate the table (Simon Riggs)

The timeout was meant to be 50 milliseconds, but it was actually only 50 microseconds, causing `VACUUM` to give up on truncation much more easily than intended. Set it to the intended value.

- Fix bugs in merging inherited `CHECK` constraints while creating or altering a table (Tom Lane, Amit Langote)

  Allow identical `CHECK` constraints to be added to a parent and child table in either order. Prevent merging of a valid constraint from the parent table with a `NOT VALID` constraint on the child. Likewise, prevent merging of a `NO INHERIT` child constraint with an inherited constraint.

- Remove artificial restrictions on the values accepted by `numeric_in()` and `numeric_recv()` (Tom Lane)

  We allow numeric values up to the limit of the storage format (more than `1e100000`), so it seems fairly pointless that `numeric_in()` rejected scientific-notation exponents above 1000. Likewise, it was silly for `numeric_recv()` to reject more than 1000 digits in an input value.

- Avoid very-low-probability data corruption due to testing tuple visibility without holding buffer lock (Thomas Munro, Peter Geoghegan, Tom Lane)

- Fix file descriptor leakage when truncating a temporary relation of more than 1GB (Andres Freund)

- Disallow starting a standalone backend with `standby_mode` turned on (Michael Paquier)

  This can't do anything useful, since there will be no WAL receiver process to fetch more WAL data; and it could result in misbehavior in code that wasn't designed with this situation in mind.

- Don't try to share SSL contexts across multiple connections in libpq (Heikki Linnakangas)

  This led to assorted corner-case bugs, particularly when trying to use different SSL parameters for different connections.

- Avoid corner-case memory leak in libpq (Tom Lane)

  The reported problem involved leaking an error report during `PQreset()`, but there might be related cases.

- Make ecpg's `--help` and `--version` options work consistently with our other executables (Haribabu Kommi)

- In pg_dump, never dump range constructor functions (Tom Lane)

  This oversight led to pg_upgrade failures with extensions containing range types, due to duplicate creation of the constructor functions.

- Fix `contrib/intarray/bench/bench.pl` to print the results of the `EXPLAIN` it does when given the `-e` option (Daniel Gustafsson)

- Update Windows time zone mapping to recognize some time zone names added in recent Windows versions (Michael Paquier)

- Prevent failure of obsolete dynamic time zone abbreviations (Tom Lane)

If a dynamic time zone abbreviation does not match any entry in the referenced time zone, treat it as equivalent to the time zone name. This avoids unexpected failures when IANA removes abbreviations from their time zone database, as they did in tzdata release 2016f and seem likely to do again in the future. The consequences were not limited to not recognizing the individual abbreviation; any mismatch caused the `pg_timezone_abbrevs` view to fail altogether.

- Update time zone data files to tzdata release 2016h for DST law changes in Palestine and Turkey, plus historical corrections for Turkey and some regions of Russia. Switch to numeric abbreviations for some time zones in Antarctica, the former Soviet Union, and Sri Lanka.

  The IANA time zone database previously provided textual abbreviations for all time zones, sometimes making up abbreviations that have little or no currency among the local population. They are in process of reversing that policy in favor of using numeric UTC offsets in zones where there is no evidence of real-world use of an English abbreviation. At least for the time being, PostgreSQL will continue to accept such removed abbreviations for timestamp input. But they will not be shown in the `pg_timezone_names` view nor used for output.

  In this update, `AMT` is no longer shown as being in use to mean Armenia Time. Therefore, we have changed the `Default` abbreviation set to interpret it as Amazon Time, thus UTC-4 not UTC+4.

E.7. Release 9.2.18

Release date: 2016-08-11

This release contains a variety of fixes from 9.2.17. For information about new features in the 9.2 major release, see 第 E.25 节.

E.7.1. Migration to Version 9.2.18

A dump/restore is not required for those running 9.2.X.

However, if you are upgrading from a version earlier than 9.2.11, see 第 E.14 节.

E.7.2. Changes

- Fix possible mis-evaluation of nested `CASE-WHEN` expressions (Heikki Linnakangas, Michael Paquier, Tom Lane)

  A `CASE` expression appearing within the test value subexpression of another `CASE` could become confused about whether its own test value was null or not. Also, inlining of a SQL function implementing the equality operator used by a `CASE` expression could result in passing the wrong test value to functions called within a `CASE` expression in the SQL function's body. If the test values were of different data types, a crash might result; moreover such situations could be abused to allow disclosure of portions of server memory. (CVE-2016-5423)

- Fix client programs' handling of special characters in database and role names (Noah Misch, Nathan Bossart, Michael Paquier)

  Numerous places in vacuumdb and other client programs could become confused by database and role names containing double quotes or backslashes. Tighten up quoting rules to make that safe. Also, ensure that when a conninfo string is used

as a database name parameter to these programs, it is correctly treated as such throughout.

Fix handling of paired double quotes in psql's `\connect` and `\password` commands to match the documentation.

Introduce a new `-reuse-previous` option in psql's `\connect` command to allow explicit control of whether to re-use connection parameters from a previous connection. (Without this, the choice is based on whether the database name looks like a conninfo string, as before.) This allows secure handling of database names containing special characters in pg_dumpall scripts.

pg_dumpall now refuses to deal with database and role names containing carriage returns or newlines, as it seems impractical to quote those characters safely on Windows. In future we may reject such names on the server side, but that step has not been taken yet.

These are considered security fixes because crafted object names containing special characters could have been used to execute commands with superuser privileges the next time a superuser executes pg_dumpall or other routine maintenance operations. (CVE-2016-5424)

- Fix corner-case misbehaviors for `IS NULL`/`IS NOT NULL` applied to nested composite values (Andrew Gierth, Tom Lane)

  The SQL standard specifies that `IS NULL` should return TRUE for a row of all null values (thus `ROW(NULL,NULL) IS NULL` yields TRUE), but this is not meant to apply recursively (thus `ROW(NULL, ROW(NULL,NULL)) IS NULL` yields FALSE). The core executor got this right, but certain planner optimizations treated the test as recursive (thus producing TRUE in both cases), and `contrib/postgres_fdw` could produce remote queries that misbehaved similarly.

- Make the `inet` and `cidr` data types properly reject IPv6 addresses with too many colon-separated fields (Tom Lane)

- Prevent crash in `close_ps()` (the `point ## lseg` operator) for NaN input coordinates (Tom Lane)

  Make it return NULL instead of crashing.

- Fix several one-byte buffer over-reads in `to_number()` (Peter Eisentraut)

  In several cases the `to_number()` function would read one more character than it should from the input string. There is a small chance of a crash, if the input happens to be adjacent to the end of memory.

- Avoid unsafe intermediate state during expensive paths through `heap_update()` (Masahiko Sawada, Andres Freund)

  Previously, these cases locked the target tuple (by setting its XMAX) but did not WAL-log that action, thus risking data integrity problems if the page were spilled to disk and then a database crash occurred before the tuple update could be completed.

- Avoid crash in `postgres -C` when the specified variable has a null string value (Michael Paquier)

- Avoid consuming a transaction ID during `VACUUM` (Alexander Korotkov)

  Some cases in `VACUUM` unnecessarily caused an XID to be assigned to the current transaction. Normally this is negligible, but if one is up against the XID wraparound limit, consuming more XIDs during anti-wraparound vacuums is a very bad thing.

- Avoid canceling hot-standby queries during `VACUUM FREEZE` (Simon Riggs, Álvaro Herrera)

  `VACUUM FREEZE` on an otherwise-idle master server could result in unnecessary cancellations of queries on its standby servers.

- When a manual `ANALYZE` specifies a column list, don't reset the table's `changes_since_analyze` counter (Tom Lane)

  If we're only analyzing some columns, we should not prevent routine auto-analyze from happening for the other columns.

- Fix `ANALYZE`'s overestimation of `n_distinct` for a unique or nearly-unique column with many null entries (Tom Lane)

  The nulls could get counted as though they were themselves distinct values, leading to serious planner misestimates in some types of queries.

- Prevent autovacuum from starting multiple workers for the same shared catalog (Álvaro Herrera)

  Normally this isn't much of a problem because the vacuum doesn't take long anyway; but in the case of a severely bloated catalog, it could result in all but one worker uselessly waiting instead of doing useful work on other tables.

- Prevent infinite loop in GiST index build for geometric columns containing NaN component values (Tom Lane)

- Fix `contrib/btree_gin` to handle the smallest possible `bigint` value correctly (Peter Eisentraut)

- Teach libpq to correctly decode server version from future servers (Peter Eisentraut)

  It's planned to switch to two-part instead of three-part server version numbers for releases after 9.6. Make sure that `PQserverVersion()` returns the correct value for such cases.

- Fix ecpg's code for `unsigned long long` array elements (Michael Meskes)

- In pg_dump with both `-c` and `-C` options, avoid emitting an unwanted `CREATE SCHEMA public` command (David Johnston, Tom Lane)

- Make pg_basebackup accept `-Z 0` as specifying no compression (Fujii Masao)

- Fix makefiles' rule for building AIX shared libraries to be safe for parallel make (Noah Misch)

- Fix TAP tests and MSVC scripts to work when build directory's path name contains spaces (Michael Paquier, Kyotaro Horiguchi)

- Make regression tests safe for Danish and Welsh locales (Jeff Janes, Tom Lane)

  Change some test data that triggered the unusual sorting rules of these locales.

- Update our copy of the timezone code to match IANA's tzcode release 2016c (Tom Lane)

  This is needed to cope with anticipated future changes in the time zone data files. It also fixes some corner-case bugs in coping with unusual time zones.

- Update time zone data files to tzdata release 2016f for DST law changes in Kemerovo and Novosibirsk, plus historical corrections for Azerbaijan, Belarus, and Morocco.

E.8. Release 9.2.17

Release date: 2016-05-12

This release contains a variety of fixes from 9.2.16. For information about new features in the 9.2 major release, see 第 E.25 节.

E.8.1. Migration to Version 9.2.17

A dump/restore is not required for those running 9.2.X.

However, if you are upgrading from a version earlier than 9.2.11, see 第 E.14 节.

E.8.2. Changes

- Clear the OpenSSL error queue before OpenSSL calls, rather than assuming it's clear already; and make sure we leave it clear afterwards (Peter Geoghegan, Dave Vitek, Peter Eisentraut)

  This change prevents problems when there are multiple connections using OpenSSL within a single process and not all the code involved follows the same rules for when to clear the error queue. Failures have been reported specifically when a client application uses SSL connections in libpq concurrently with SSL connections using the PHP, Python, or Ruby wrappers for OpenSSL. It's possible for similar problems to arise within the server as well, if an extension module establishes an outgoing SSL connection.

- Fix "failed to build any *N*-way joins" planner error with a full join enclosed in the right-hand side of a left join (Tom Lane)

- Fix incorrect handling of equivalence-class tests in multilevel nestloop plans (Tom Lane)

  Given a three-or-more-way equivalence class of variables, such as `X.X = Y.Y = Z.Z`, it was possible for the planner to omit some of the tests needed to enforce that all the variables are actually equal, leading to join rows being output that didn't satisfy the `WHERE` clauses. For various reasons, erroneous plans were seldom selected in practice, so that this bug has gone undetected for a long time.

- Fix possible misbehavior of `TH`, `th`, and `Y,YYY` format codes in `to_timestamp()` (Tom Lane)

  These could advance off the end of the input string, causing subsequent format codes to read garbage.

- Fix dumping of rules and views in which the *array* argument of a *value operator* `ANY (`*array*`)` construct is a sub-SELECT (Tom Lane)

- Make pg_regress use a startup timeout from the `PGCTLTIMEOUT` environment variable, if that's set (Tom Lane)

  This is for consistency with a behavior recently added to pg_ctl; it eases automated testing on slow machines.

- Fix pg_upgrade to correctly restore extension membership for operator families containing only one operator class (Tom Lane)

  In such a case, the operator family was restored into the new database, but it was no longer marked as part of the extension. This had no immediate ill effects, but

would cause later pg_dump runs to emit output that would cause (harmless) errors on restore.

- Back-port 9.4-era memory-barrier code changes into 9.2 and 9.3 (Tom Lane)

  These changes were not originally needed in pre-9.4 branches, but we recently back-patched a fix that expected the barrier code to work properly. Only IA64 (when using icc), HPPA, and Alpha platforms are affected.

- Reduce the number of SysV semaphores used by a build configured with `--disable-spinlocks` (Tom Lane)

- Rename internal function `strtoi()` to `strtoint()` to avoid conflict with a NetBSD library function (Thomas Munro)

- Fix reporting of errors from `bind()` and `listen()` system calls on Windows (Tom Lane)

- Reduce verbosity of compiler output when building with Microsoft Visual Studio (Christian Ullrich)

- Avoid possibly-unsafe use of Windows' `FormatMessage()` function (Christian Ullrich)

  Use the `FORMAT_MESSAGE_IGNORE_INSERTS` flag where appropriate. No live bug is known to exist here, but it seems like a good idea to be careful.

- Update time zone data files to tzdata release 2016d for DST law changes in Russia and Venezuela. There are new zone names `Europe/Kirov` and `Asia/Tomsk` to reflect the fact that these regions now have different time zone histories from adjacent regions.

E.9. Release 9.2.16

Release date: 2016-03-31

This release contains a variety of fixes from 9.2.15. For information about new features in the 9.2 major release, see 第 E.25 节.

E.9.1. Migration to Version 9.2.16

A dump/restore is not required for those running 9.2.X.

However, if you are upgrading from a version earlier than 9.2.11, see 第 E.14 节.

E.9.2. Changes

- Fix incorrect handling of NULL index entries in indexed `ROW()` comparisons (Tom Lane)

  An index search using a row comparison such as `ROW(a, b) > ROW('x', 'y')` would stop upon reaching a NULL entry in the `b` column, ignoring the fact that there might be non-NULL `b` values associated with later values of `a`.

- Avoid unlikely data-loss scenarios due to renaming files without adequate `fsync()` calls before and after (Michael Paquier, Tomas Vondra, Andres Freund)

- Correctly handle cases where `pg_subtrans` is close to XID wraparound during server startup (Jeff Janes)

- Fix corner-case crash due to trying to free `localeconv()` output strings more than once (Tom Lane)
- Fix parsing of affix files for `ispell` dictionaries (Tom Lane)

  The code could go wrong if the affix file contained any characters whose byte length changes during case-folding, for example `I` in Turkish UTF8 locales.
- Avoid use of `sscanf()` to parse `ispell` dictionary files (Artur Zakirov)

  This dodges a portability problem on FreeBSD-derived platforms (including macOS).
- Avoid a crash on old Windows versions (before 7SP1/2008R2SP1) with an AVX2-capable CPU and a Postgres build done with Visual Studio 2013 (Christian Ullrich)

  This is a workaround for a bug in Visual Studio 2013's runtime library, which Microsoft have stated they will not fix in that version.
- Fix psql's tab completion logic to handle multibyte characters properly (Kyotaro Horiguchi, Robert Haas)
- Fix psql's tab completion for `SECURITY LABEL` (Tom Lane)

  Pressing TAB after `SECURITY LABEL` might cause a crash or offering of inappropriate keywords.
- Make pg_ctl accept a wait timeout from the `PGCTLTIMEOUT` environment variable, if none is specified on the command line (Noah Misch)

  This eases testing of slower buildfarm members by allowing them to globally specify a longer-than-normal timeout for postmaster startup and shutdown.
- Fix incorrect test for Windows service status in pg_ctl (Manuel Mathar)

  The previous set of minor releases attempted to fix pg_ctl to properly determine whether to send log messages to Window's Event Log, but got the test backwards.
- Fix pgbench to correctly handle the combination of `-C` and `-M prepared` options (Tom Lane)
- In PL/Perl, properly translate empty Postgres arrays into empty Perl arrays (Alex Hunsaker)
- Make PL/Python cope with function names that aren't valid Python identifiers (Jim Nasby)
- Fix multiple mistakes in the statistics returned by `contrib/pgstattuple`'s `pgstatindex()` function (Tom Lane)
- Remove dependency on `psed` in MSVC builds, since it's no longer provided by core Perl (Michael Paquier, Andrew Dunstan)
- Update time zone data files to tzdata release 2016c for DST law changes in Azerbaijan, Chile, Haiti, Palestine, and Russia (Altai, Astrakhan, Kirov, Sakhalin, Ulyanovsk regions), plus historical corrections for Lithuania, Moldova, and Russia (Kaliningrad, Samara, Volgograd).

E.10. Release 9.2.15

Release date: 2016-02-11

This release contains a variety of fixes from 9.2.14. For information about new features in the 9.2 major release, see 第 E.25 节.

E.10.1. Migration to Version 9.2.15

A dump/restore is not required for those running 9.2.X.

However, if you are upgrading from a version earlier than 9.2.11, see 第 E.14 节.

E.10.2. Changes

- Fix infinite loops and buffer-overrun problems in regular expressions (Tom Lane)

  Very large character ranges in bracket expressions could cause infinite loops in some cases, and memory overwrites in other cases. (CVE-2016-0773)

- Perform an immediate shutdown if the `postmaster.pid` file is removed (Tom Lane)

  The postmaster now checks every minute or so that `postmaster.pid` is still there and still contains its own PID. If not, it performs an immediate shutdown, as though it had received SIGQUIT. The main motivation for this change is to ensure that failed buildfarm runs will get cleaned up without manual intervention; but it also serves to limit the bad effects if a DBA forcibly removes `postmaster.pid` and then starts a new postmaster.

- In `SERIALIZABLE` transaction isolation mode, serialization anomalies could be missed due to race conditions during insertions (Kevin Grittner, Thomas Munro)

- Fix failure to emit appropriate WAL records when doing `ALTER TABLE ... SET TABLESPACE` for unlogged relations (Michael Paquier, Andres Freund)

  Even though the relation's data is unlogged, the move must be logged or the relation will be inaccessible after a standby is promoted to master.

- Fix possible misinitialization of unlogged relations at the end of crash recovery (Andres Freund, Michael Paquier)

- Fix `ALTER COLUMN TYPE` to reconstruct inherited check constraints properly (Tom Lane)

- Fix `REASSIGN OWNED` to change ownership of composite types properly (Álvaro Herrera)

- Fix `REASSIGN OWNED` and `ALTER OWNER` to correctly update granted-permissions lists when changing owners of data types, foreign data wrappers, or foreign servers (Bruce Momjian, Álvaro Herrera)

- Fix `REASSIGN OWNED` to ignore foreign user mappings, rather than fail (Álvaro Herrera)

- Add more defenses against bad planner cost estimates for GIN index scans when the index's internal statistics are very out-of-date (Tom Lane)

- Make planner cope with hypothetical GIN indexes suggested by an index advisor plug-in (Julien Rouhaud)

- Fix dumping of whole-row Vars in `ROW()` and `VALUES()` lists (Tom Lane)

- Fix possible internal overflow in `numeric` division (Dean Rasheed)

- Fix enforcement of restrictions inside parentheses within regular expression lookahead constraints (Tom Lane)

  Lookahead constraints aren't allowed to contain backrefs, and parentheses within them are always considered non-capturing, according to the manual. However, the code failed to handle these cases properly inside a parenthesized subexpression, and would give unexpected results.

- Conversion of regular expressions to indexscan bounds could produce incorrect bounds from regexps containing lookahead constraints (Tom Lane)

- Fix regular-expression compiler to handle loops of constraint arcs (Tom Lane)

  The code added for CVE-2007-4772 was both incomplete, in that it didn't handle loops involving more than one state, and incorrect, in that it could cause assertion failures (though there seem to be no bad consequences of that in a non-assert build). Multi-state loops would cause the compiler to run until the query was canceled or it reached the too-many-states error condition.

- Improve memory-usage accounting in regular-expression compiler (Tom Lane)

  This causes the code to emit "regular expression is too complex" errors in some cases that previously used unreasonable amounts of time and memory.

- Improve performance of regular-expression compiler (Tom Lane)

- Make `%h` and `%r` escapes in `log_line_prefix` work for messages emitted due to `log_connections` (Tom Lane)

  Previously, `%h`/`%r` started to work just after a new session had emitted the "connection received" log message; now they work for that message too.

- On Windows, ensure the shared-memory mapping handle gets closed in child processes that don't need it (Tom Lane, Amit Kapila)

  This oversight resulted in failure to recover from crashes whenever `logging_collector` is turned on.

- Fix possible failure to detect socket EOF in non-blocking mode on Windows (Tom Lane)

  It's not entirely clear whether this problem can happen in pre-9.5 branches, but if it did, the symptom would be that a walsender process would wait indefinitely rather than noticing a loss of connection.

- Avoid leaking a token handle during SSPI authentication (Christian Ullrich)

- In psql, ensure that libreadline's idea of the screen size is updated when the terminal window size changes (Merlin Moncure)

  Previously, libreadline did not notice if the window was resized during query output, leading to strange behavior during later input of multiline queries.

- Fix psql's `\det` command to interpret its pattern argument the same way as other `\d` commands with potentially schema-qualified patterns do (Reece Hart)

- Avoid possible crash in psql's `\c` command when previous connection was via Unix socket and command specifies a new hostname and same username (Tom Lane)

- In `pg_ctl start -w`, test child process status directly rather than relying on heuristics (Tom Lane, Michael Paquier)

Previously, pg_ctl relied on an assumption that the new postmaster would always create `postmaster.pid` within five seconds. But that can fail on heavily-loaded systems, causing pg_ctl to report incorrectly that the postmaster failed to start.

Except on Windows, this change also means that a `pg_ctl start -w` done immediately after another such command will now reliably fail, whereas previously it would report success if done within two seconds of the first command.

- In `pg_ctl start -w`, don't attempt to use a wildcard listen address to connect to the postmaster (Kondo Yuta)

  On Windows, pg_ctl would fail to detect postmaster startup if `listen_addresses` is set to `0.0.0.0` or `::`, because it would try to use that value verbatim as the address to connect to, which doesn't work. Instead assume that `127.0.0.1` or `::1`, respectively, is the right thing to use.

- In pg_ctl on Windows, check service status to decide where to send output, rather than checking if standard output is a terminal (Michael Paquier)

- In pg_dump and pg_basebackup, adopt the GNU convention for handling tar-archive members exceeding 8GB (Tom Lane)

  The POSIX standard for `tar` file format does not allow archive member files to exceed 8GB, but most modern implementations of tar support an extension that fixes that. Adopt this extension so that pg_dump with `-Ft` no longer fails on tables with more than 8GB of data, and so that pg_basebackup can handle files larger than 8GB. In addition, fix some portability issues that could cause failures for members between 4GB and 8GB on some platforms. Potentially these problems could cause unrecoverable data loss due to unreadable backup files.

- Fix assorted corner-case bugs in pg_dump's processing of extension member objects (Tom Lane)

- Make pg_dump mark a view's triggers as needing to be processed after its rule, to prevent possible failure during parallel pg_restore (Tom Lane)

- Ensure that relation option values are properly quoted in pg_dump (Kouhei Sutou, Tom Lane)

  A reloption value that isn't a simple identifier or number could lead to dump/reload failures due to syntax errors in CREATE statements issued by pg_dump. This is not an issue with any reloption currently supported by core PostgreSQL, but extensions could allow reloptions that cause the problem.

- Fix pg_upgrade's file-copying code to handle errors properly on Windows (Bruce Momjian)

- Install guards in pgbench against corner-case overflow conditions during evaluation of script-specified division or modulo operators (Fabien Coelho, Michael Paquier)

- Fix failure to localize messages emitted by pg_receivexlog and pg_recvlogical (Ioseph Kim)

- Avoid dump/reload problems when using both plpython2 and plpython3 (Tom Lane)

  In principle, both versions of PL/Python can be used in the same database, though not in the same session (because the two versions of libpython cannot safely be used concurrently). However, pg_restore and pg_upgrade both do things that can

fall foul of the same-session restriction. Work around that by changing the timing of the check.

- Fix PL/Python regression tests to pass with Python 3.5 (Peter Eisentraut)
- Prevent certain PL/Java parameters from being set by non-superusers (Noah Misch)

  This change mitigates a PL/Java security bug (CVE-2016-0766), which was fixed in PL/Java by marking these parameters as superuser-only. To fix the security hazard for sites that update PostgreSQL more frequently than PL/Java, make the core code aware of them also.
- Improve libpq's handling of out-of-memory situations (Michael Paquier, Amit Kapila, Heikki Linnakangas)
- Fix order of arguments in ecpg-generated `typedef` statements (Michael Meskes)
- Use `%g` not `%f` format in ecpg's `PGTYPESnumeric_from_double()` (Tom Lane)
- Fix ecpg-supplied header files to not contain comments continued from a preprocessor directive line onto the next line (Michael Meskes)

  Such a comment is rejected by ecpg. It's not yet clear whether ecpg itself should be changed.
- Ensure that `contrib/pgcrypto`'s `crypt()` function can be interrupted by query cancel (Andreas Karlsson)
- Accept flex versions later than 2.5.x (Tom Lane, Michael Paquier)

  Now that flex 2.6.0 has been released, the version checks in our build scripts needed to be adjusted.
- Install our `missing` script where PGXS builds can find it (Jim Nasby)

  This allows sane behavior in a PGXS build done on a machine where build tools such as bison are missing.
- Ensure that `dynloader.h` is included in the installed header files in MSVC builds (Bruce Momjian, Michael Paquier)
- Add variant regression test expected-output file to match behavior of current libxml2 (Tom Lane)

  The fix for libxml2's CVE-2015-7499 causes it not to output error context reports in some cases where it used to do so. This seems to be a bug, but we'll probably have to live with it for some time, so work around it.
- Update time zone data files to tzdata release 2016a for DST law changes in Cayman Islands, Metlakatla, and Trans-Baikal Territory (Zabaykalsky Krai), plus historical corrections for Pakistan.

E.11. Release 9.2.14

Release date: 2015-10-08

This release contains a variety of fixes from 9.2.13. For information about new features in the 9.2 major release, see 第 E.25 节.

E.11.1. Migration to Version 9.2.14

A dump/restore is not required for those running 9.2.X.

However, if you are upgrading from a version earlier than 9.2.11, see 第 E.14 节.

E.11.2. Changes

- Fix `contrib/pgcrypto` to detect and report too-short `crypt()` salts (Josh Kupershmidt)

  Certain invalid salt arguments crashed the server or disclosed a few bytes of server memory. We have not ruled out the viability of attacks that arrange for presence of confidential information in the disclosed bytes, but they seem unlikely. (CVE-2015-5288)

- Fix subtransaction cleanup after a portal (cursor) belonging to an outer subtransaction fails (Tom Lane, Michael Paquier)

  A function executed in an outer-subtransaction cursor could cause an assertion failure or crash by referencing a relation created within an inner subtransaction.

- Fix insertion of relations into the relation cache "init file" (Tom Lane)

  An oversight in a patch in the most recent minor releases caused `pg_trigger_tgrelid_tgname_index` to be omitted from the init file. Subsequent sessions detected this, then deemed the init file to be broken and silently ignored it, resulting in a significant degradation in session startup time. In addition to fixing the bug, install some guards so that any similar future mistake will be more obvious.

- Avoid O(N^2) behavior when inserting many tuples into a SPI query result (Neil Conway)

- Improve `LISTEN` startup time when there are many unread notifications (Matt Newell)

- Back-patch 9.3-era addition of per-resource-owner lock caches (Jeff Janes)

  This substantially improves performance when pg_dump tries to dump a large number of tables.

- Disable SSL renegotiation by default (Michael Paquier, Andres Freund)

  While use of SSL renegotiation is a good idea in theory, we have seen too many bugs in practice, both in the underlying OpenSSL library and in our usage of it. Renegotiation will be removed entirely in 9.5 and later. In the older branches, just change the default value of `ssl_renegotiation_limit` to zero (disabled).

- Lower the minimum values of the `*_freeze_max_age` parameters (Andres Freund)

  This is mainly to make tests of related behavior less time-consuming, but it may also be of value for installations with limited disk space.

- Limit the maximum value of `wal_buffers` to 2GB to avoid server crashes (Josh Berkus)

- Fix rare internal overflow in multiplication of `numeric` values (Dean Rasheed)

- Guard against hard-to-reach stack overflows involving record types, range types, `json`, `jsonb`, `tsquery`, `ltxtquery` and `query_int` (Noah Misch)

- Fix handling of `DOW` and `DOY` in datetime input (Greg Stark)

  These tokens aren't meant to be used in datetime values, but previously they resulted in opaque internal error messages rather than "invalid input syntax".

- Add more query-cancel checks to regular expression matching (Tom Lane)
- Add recursion depth protections to regular expression, `SIMILAR TO`, and `LIKE` matching (Tom Lane)

  Suitable search patterns and a low stack depth limit could lead to stack-overrun crashes.
- Fix potential infinite loop in regular expression execution (Tom Lane)

  A search pattern that can apparently match a zero-length string, but actually doesn't match because of a back reference, could lead to an infinite loop.
- In regular expression execution, correctly record match data for capturing parentheses within a quantifier even when the match is zero-length (Tom Lane)
- Fix low-memory failures in regular expression compilation (Andreas Seltenreich)
- Fix low-probability memory leak during regular expression execution (Tom Lane)
- Fix rare low-memory failure in lock cleanup during transaction abort (Tom Lane)
- Fix "unexpected out-of-memory situation during sort" errors when using tuplestores with small `work_mem` settings (Tom Lane)
- Fix very-low-probability stack overrun in `qsort` (Tom Lane)
- Fix "invalid memory alloc request size" failure in hash joins with large `work_mem` settings (Tomas Vondra, Tom Lane)
- Fix assorted planner bugs (Tom Lane)

  These mistakes could lead to incorrect query plans that would give wrong answers, or to assertion failures in assert-enabled builds, or to odd planner errors such as "could not devise a query plan for the given query", "could not find pathkey item to sort", "plan should not reference subplan's variable", or "failed to assign all NestLoopParams to plan nodes". Thanks are due to Andreas Seltenreich and Piotr Stefaniak for fuzz testing that exposed these problems.
- Improve planner's performance for `UPDATE`/`DELETE` on large inheritance sets (Tom Lane, Dean Rasheed)
- Ensure standby promotion trigger files are removed at postmaster startup (Michael Paquier, Fujii Masao)

  This prevents unwanted promotion from occurring if these files appear in a database backup that is used to initialize a new standby server.
- During postmaster shutdown, ensure that per-socket lock files are removed and listen sockets are closed before we remove the `postmaster.pid` file (Tom Lane)

  This avoids race-condition failures if an external script attempts to start a new postmaster as soon as `pg_ctl stop` returns.
- Fix postmaster's handling of a startup-process crash during crash recovery (Tom Lane)

  If, during a crash recovery cycle, the startup process crashes without having restored database consistency, we'd try to launch a new startup process, which typically would just crash again, leading to an infinite loop.
- Do not print a `WARNING` when an autovacuum worker is already gone when we attempt to signal it, and reduce log verbosity for such signals (Tom Lane)

- Prevent autovacuum launcher from sleeping unduly long if the server clock is moved backwards a large amount (Álvaro Herrera)
- Ensure that cleanup of a GIN index's pending-insertions list is interruptable by cancel requests (Jeff Janes)
- Allow all-zeroes pages in GIN indexes to be reused (Heikki Linnakangas)

  Such a page might be left behind after a crash.
- Fix handling of all-zeroes pages in SP-GiST indexes (Heikki Linnakangas)

  `VACUUM` attempted to recycle such pages, but did so in a way that wasn't crash-safe.
- Fix off-by-one error that led to otherwise-harmless warnings about "apparent wraparound" in subtrans/multixact truncation (Thomas Munro)
- Fix misreporting of `CONTINUE` and `MOVE` statement types in PL/pgSQL's error context messages (Pavel Stehule, Tom Lane)
- Fix PL/Perl to handle non-ASCII error message texts correctly (Alex Hunsaker)
- Fix PL/Python crash when returning the string representation of a `record` result (Tom Lane)
- Fix some places in PL/Tcl that neglected to check for failure of `malloc()` calls (Michael Paquier, Álvaro Herrera)
- In `contrib/isn`, fix output of ISBN-13 numbers that begin with 979 (Fabien Coelho)

  EANs beginning with 979 (but not 9790) are considered ISBNs, but they must be printed in the new 13-digit format, not the 10-digit format.
- Fix `contrib/sepgsql`'s handling of `SELECT INTO` statements (Kohei KaiGai)
- Improve libpq's handling of out-of-memory conditions (Michael Paquier, Heikki Linnakangas)
- Fix memory leaks and missing out-of-memory checks in ecpg (Michael Paquier)
- Fix psql's code for locale-aware formatting of numeric output (Tom Lane)

  The formatting code invoked by `\pset numericlocale on` did the wrong thing for some uncommon cases such as numbers with an exponent but no decimal point. It could also mangle already-localized output from the `money` data type.
- Prevent crash in psql's `\c` command when there is no current connection (Noah Misch)
- Make pg_dump handle inherited `NOT VALID` check constraints correctly (Tom Lane)
- Fix selection of default zlib compression level in pg_dump's directory output format (Andrew Dunstan)
- Ensure that temporary files created during a pg_dump run with tar-format output are not world-readable (Michael Paquier)
- Fix pg_dump and pg_upgrade to support cases where the `postgres` or `template1` database is in a non-default tablespace (Marti Raudsepp, Bruce Momjian)
- Fix pg_dump to handle object privileges sanely when dumping from a server too old to have a particular privilege type (Tom Lane)

When dumping data types from pre-9.2 servers, and when dumping functions or procedural languages from pre-7.3 servers, pg_dump would produce `GRANT /REVOKE` commands that revoked the owner's grantable privileges and instead granted all privileges to `PUBLIC`. Since the privileges involved are just `USAGE` and `EXECUTE`, this isn't a security problem, but it's certainly a surprising representation of the older systems' behavior. Fix it to leave the default privilege state alone in these cases.

- Fix pg_dump to dump shell types (Tom Lane)

  Shell types (that is, not-yet-fully-defined types) aren't useful for much, but nonetheless pg_dump should dump them.

- Fix assorted minor memory leaks in pg_dump and other client-side programs (Michael Paquier)

- Fix spinlock assembly code for PPC hardware to be compatible with AIX's native assembler (Tom Lane)

  Building with gcc didn't work if gcc had been configured to use the native assembler, which is becoming more common.

- On AIX, test the `-qlonglong` compiler option rather than just assuming it's safe to use (Noah Misch)

- On AIX, use `-Wl,-brtllib` link option to allow symbols to be resolved at runtime (Noah Misch)

  Perl relies on this ability in 5.8.0 and later.

- Avoid use of inline functions when compiling with 32-bit xlc, due to compiler bugs (Noah Misch)

- Use `librt` for `sched_yield()` when necessary, which it is on some Solaris versions (Oskari Saarenmaa)

- Fix Windows `install.bat` script to handle target directory names that contain spaces (Heikki Linnakangas)

- Make the numeric form of the PostgreSQL version number (e.g., `90405`) readily available to extension Makefiles, as a variable named `VERSION_NUM` (Michael Paquier)

- Update time zone data files to tzdata release 2015g for DST law changes in Cayman Islands, Fiji, Moldova, Morocco, Norfolk Island, North Korea, Turkey, and Uruguay. There is a new zone name `America/Fort_Nelson` for the Canadian Northern Rockies.

E.12. Release 9.2.13

Release date: 2015-06-12

This release contains a small number of fixes from 9.2.12. For information about new features in the 9.2 major release, see 第 E.25 节.

E.12.1. Migration to Version 9.2.13

A dump/restore is not required for those running 9.2.X.

However, if you are upgrading from a version earlier than 9.2.11, see 第 E.14 节.

E.12.2. Changes

- Fix rare failure to invalidate relation cache init file (Tom Lane)

  With just the wrong timing of concurrent activity, a `VACUUM FULL` on a system catalog might fail to update the "init file" that's used to avoid cache-loading work for new sessions. This would result in later sessions being unable to access that catalog at all. This is a very ancient bug, but it's so hard to trigger that no reproducible case had been seen until recently.

- Avoid deadlock between incoming sessions and `CREATE/DROP DATABASE` (Tom Lane)

  A new session starting in a database that is the target of a `DROP DATABASE` command, or is the template for a `CREATE DATABASE` command, could cause the command to wait for five seconds and then fail, even if the new session would have exited before that.

E.13. Release 9.2.12

Release date: 2015-06-04

This release contains a small number of fixes from 9.2.11. For information about new features in the 9.2 major release, see 第 E.25 节.

E.13.1. Migration to Version 9.2.12

A dump/restore is not required for those running 9.2.X.

However, if you are upgrading from a version earlier than 9.2.11, see 第 E.14 节.

E.13.2. Changes

- Avoid failures while `fsync`'ing data directory during crash restart (Abhijit Menon-Sen, Tom Lane)

  In the previous minor releases we added a patch to `fsync` everything in the data directory after a crash. Unfortunately its response to any error condition was to fail, thereby preventing the server from starting up, even when the problem was quite harmless. An example is that an unwritable file in the data directory would prevent restart on some platforms; but it is common to make SSL certificate files unwritable by the server. Revise this behavior so that permissions failures are ignored altogether, and other types of failures are logged but do not prevent continuing.

- Fix `pg_get_functiondef()` to show functions' `LEAKPROOF` property, if set (Jeevan Chalke)

- Remove configure's check prohibiting linking to a threaded libpython on OpenBSD (Tom Lane)

  The failure this restriction was meant to prevent seems to not be a problem anymore on current OpenBSD versions.

- Allow libpq to use TLS protocol versions beyond v1 (Noah Misch)

For a long time, libpq was coded so that the only SSL protocol it would allow was TLS v1. Now that newer TLS versions are becoming popular, allow it to negotiate the highest commonly-supported TLS version with the server. (PostgreSQL servers were already capable of such negotiation, so no change is needed on the server side.) This is a back-patch of a change already released in 9.4.0.

E.14. Release 9.2.11

Release date: 2015-05-22

This release contains a variety of fixes from 9.2.10. For information about new features in the 9.2 major release, see 第 E.25 节.

E.14.1. Migration to Version 9.2.11

A dump/restore is not required for those running 9.2.X.

However, if you use `contrib/citext`'s `regexp_matches()` functions, see the changelog entry below about that.

Also, if you are upgrading from a version earlier than 9.2.10, see 第 E.15 节.

E.14.2. Changes

- Avoid possible crash when client disconnects just before the authentication timeout expires (Benkocs Norbert Attila)

  If the timeout interrupt fired partway through the session shutdown sequence, SSL-related state would be freed twice, typically causing a crash and hence denial of service to other sessions. Experimentation shows that an unauthenticated remote attacker could trigger the bug somewhat consistently, hence treat as security issue. (CVE-2015-3165)

- Improve detection of system-call failures (Noah Misch)

  Our replacement implementation of `snprintf()` failed to check for errors reported by the underlying system library calls; the main case that might be missed is out-of-memory situations. In the worst case this might lead to information exposure, due to our code assuming that a buffer had been overwritten when it hadn't been. Also, there were a few places in which security-relevant calls of other system library functions did not check for failure.

  It remains possible that some calls of the `*printf()` family of functions are vulnerable to information disclosure if an out-of-memory error occurs at just the wrong time. We judge the risk to not be large, but will continue analysis in this area. (CVE-2015-3166)

- In `contrib/pgcrypto`, uniformly report decryption failures as “Wrong key or corrupt data” (Noah Misch)

  Previously, some cases of decryption with an incorrect key could report other error message texts. It has been shown that such variance in error reports can aid attackers in recovering keys from other systems. While it's unknown whether `pgcrypto`'s specific behaviors are likewise exploitable, it seems better to avoid the risk by using a one-size-fits-all message. (CVE-2015-3167)

- Fix incorrect declaration of `contrib/citext`'s `regexp_matches()` functions (Tom Lane)

These functions should return `setof text[]`, like the core functions they are wrappers for; but they were incorrectly declared as returning just `text[]`. This mistake had two results: first, if there was no match you got a scalar null result, whereas what you should get is an empty set (zero rows). Second, the `g` flag was effectively ignored, since you would get only one result array even if there were multiple matches.

While the latter behavior is clearly a bug, there might be applications depending on the former behavior; therefore the function declarations will not be changed by default until PostgreSQL 9.5. In pre-9.5 branches, the old behavior exists in version 1.0 of the `citext` extension, while we have provided corrected declarations in version 1.1 (which is not installed by default). To adopt the fix in pre-9.5 branches, execute `ALTER EXTENSION citext UPDATE TO '1.1'` in each database in which `citext` is installed. (You can also "update" back to 1.0 if you need to undo that.) Be aware that either update direction will require dropping and recreating any views or rules that use `citext`'s `regexp_matches()` functions.

- Fix incorrect checking of deferred exclusion constraints after a HOT update (Tom Lane)

  If a new row that potentially violates a deferred exclusion constraint is HOT-updated (that is, no indexed columns change and the row can be stored back onto the same table page) later in the same transaction, the exclusion constraint would be reported as violated when the check finally occurred, even if the row(s) the new row originally conflicted with had been deleted.

- Fix planning of star-schema-style queries (Tom Lane)

  Sometimes, efficient scanning of a large table requires that index parameters be provided from more than one other table (commonly, dimension tables whose keys are needed to index a large fact table). The planner should be able to find such plans, but an overly restrictive search heuristic prevented it.

- Prevent improper reordering of antijoins (NOT EXISTS joins) versus other outer joins (Tom Lane)

  This oversight in the planner has been observed to cause "could not find RelOptInfo for given relids" errors, but it seems possible that sometimes an incorrect query plan might get past that consistency check and result in silently-wrong query output.

- Fix incorrect matching of subexpressions in outer-join plan nodes (Tom Lane)

  Previously, if textually identical non-strict subexpressions were used both above and below an outer join, the planner might try to re-use the value computed below the join, which would be incorrect because the executor would force the value to NULL in case of an unmatched outer row.

- Fix GEQO planner to cope with failure of its join order heuristic (Tom Lane)

  This oversight has been seen to lead to "failed to join all relations together" errors in queries involving `LATERAL`, and that might happen in other cases as well.

- Fix possible deadlock at startup when `max_prepared_transactions` is too small (Heikki Linnakangas)

- Don't archive useless preallocated WAL files after a timeline switch (Heikki Linnakangas)

- Avoid "cannot GetMultiXactIdMembers() during recovery" error (Álvaro Herrera)

- Recursively `fsync()` the data directory after a crash (Abhijit Menon-Sen, Robert Haas)

  This ensures consistency if another crash occurs shortly later. (The second crash would have to be a system-level crash, not just a database crash, for there to be a problem.)

- Fix autovacuum launcher's possible failure to shut down, if an error occurs after it receives SIGTERM (Álvaro Herrera)

- Cope with unexpected signals in `LockBufferForCleanup()` (Andres Freund)

  This oversight could result in spurious errors about "multiple backends attempting to wait for pincount 1" .

- Fix crash when doing `COPY IN` to a table with check constraints that contain whole-row references (Tom Lane)

  The known failure case only crashes in 9.4 and up, but there is very similar code in 9.3 and 9.2, so back-patch those branches as well.

- Avoid waiting for WAL flush or synchronous replication during commit of a transaction that was read-only so far as the user is concerned (Andres Freund)

  Previously, a delay could occur at commit in transactions that had written WAL due to HOT page pruning, leading to undesirable effects such as sessions getting stuck at startup if all synchronous replicas are down. Sessions have also been observed to get stuck in catchup interrupt processing when using synchronous replication; this will fix that problem as well.

- Fix crash when manipulating hash indexes on temporary tables (Heikki Linnakangas)

- Fix possible failure during hash index bucket split, if other processes are modifying the index concurrently (Tom Lane)

- Check for interrupts while analyzing index expressions (Jeff Janes)

  `ANALYZE` executes index expressions many times; if there are slow functions in such an expression, it's desirable to be able to cancel the `ANALYZE` before that loop finishes.

- Ensure `tableoid` of a foreign table is reported correctly when a `READ COMMITTED` recheck occurs after locking rows in `SELECT FOR UPDATE`, `UPDATE`, or `DELETE` (Etsuro Fujita)

- Add the name of the target server to object description strings for foreign-server user mappings (Álvaro Herrera)

- Recommend setting `include_realm` to 1 when using Kerberos/GSSAPI/SSPI authentication (Stephen Frost)

  Without this, identically-named users from different realms cannot be distinguished. For the moment this is only a documentation change, but it will become the default setting in PostgreSQL 9.5.

- Remove code for matching IPv4 `pg_hba.conf` entries to IPv4-in-IPv6 addresses (Tom Lane)

  This hack was added in 2003 in response to a report that some Linux kernels of the time would report IPv4 connections as having IPv4-in-IPv6 addresses. However,

the logic was accidentally broken in 9.0. The lack of any field complaints since then shows that it's not needed anymore. Now we have reports that the broken code causes crashes on some systems, so let's just remove it rather than fix it. (Had we chosen to fix it, that would make for a subtle and potentially security-sensitive change in the effective meaning of IPv4 `pg_hba.conf` entries, which does not seem like a good thing to do in minor releases.)

- Report WAL flush, not insert, position in `IDENTIFY_SYSTEM` replication command (Heikki Linnakangas)

  This avoids a possible startup failure in pg_receivexlog.

- While shutting down service on Windows, periodically send status updates to the Service Control Manager to prevent it from killing the service too soon; and ensure that pg_ctl will wait for shutdown (Krystian Bigaj)

- Reduce risk of network deadlock when using libpq's non-blocking mode (Heikki Linnakangas)

  When sending large volumes of data, it's important to drain the input buffer every so often, in case the server has sent enough response data to cause it to block on output. (A typical scenario is that the server is sending a stream of NOTICE messages during `COPY FROM STDIN`.) This worked properly in the normal blocking mode, but not so much in non-blocking mode. We've modified libpq to opportunistically drain input when it can, but a full defense against this problem requires application cooperation: the application should watch for socket read-ready as well as write-ready conditions, and be sure to call `PQconsumeInput()` upon read-ready.

- In libpq, fix misparsing of empty values in URI connection strings (Thomas Fanghaenel)

- Fix array handling in ecpg (Michael Meskes)

- Fix psql to sanely handle URIs and conninfo strings as the first parameter to `\connect` (David Fetter, Andrew Dunstan, Álvaro Herrera)

  This syntax has been accepted (but undocumented) for a long time, but previously some parameters might be taken from the old connection instead of the given string, which was agreed to be undesirable.

- Suppress incorrect complaints from psql on some platforms that it failed to write `~/.psql_history` at exit (Tom Lane)

  This misbehavior was caused by a workaround for a bug in very old (pre-2006) versions of libedit. We fixed it by removing the workaround, which will cause a similar failure to appear for anyone still using such versions of libedit. Recommendation: upgrade that library, or use libreadline.

- Fix pg_dump's rule for deciding which casts are system-provided casts that should not be dumped (Tom Lane)

- In pg_dump, fix failure to honor `-Z` compression level option together with `-Fd` (Michael Paquier)

- Make pg_dump consider foreign key relationships between extension configuration tables while choosing dump order (Gilles Darold, Michael Paquier, Stephen Frost)

  This oversight could result in producing dumps that fail to reload because foreign key constraints are transiently violated.

- Fix dumping of views that are just `VALUES(...)` but have column aliases (Tom Lane)

- In pg_upgrade, force timeline 1 in the new cluster (Bruce Momjian)

  This change prevents upgrade failures caused by bogus complaints about missing WAL history files.

- In pg_upgrade, check for improperly non-connectable databases before proceeding (Bruce Momjian)

- In pg_upgrade, quote directory paths properly in the generated `delete_old_cluster` script (Bruce Momjian)

- In pg_upgrade, preserve database-level freezing info properly (Bruce Momjian)

  This oversight could cause missing-clog-file errors for tables within the `postgres` and `template1` databases.

- Run pg_upgrade and pg_resetxlog with restricted privileges on Windows, so that they don't fail when run by an administrator (Muhammad Asif Naeem)

- Improve handling of `readdir()` failures when scanning directories in initdb and pg_basebackup (Marco Nenciarini)

- Fix failure in pg_receivexlog (Andres Freund)

  A patch merge mistake in 9.2.10 led to "could not create archive status file" errors.

- Fix slow sorting algorithm in `contrib/intarray` (Tom Lane)

- Fix compile failure on Sparc V8 machines (Rob Rowan)

- Update time zone data files to tzdata release 2015d for DST law changes in Egypt, Mongolia, and Palestine, plus historical changes in Canada and Chile. Also adopt revised zone abbreviations for the America/Adak zone (HST/HDT not HAST/HADT).

E.15. Release 9.2.10

Release date: 2015-02-05

This release contains a variety of fixes from 9.2.9. For information about new features in the 9.2 major release, see 第 E.25 节.

E.15.1. Migration to Version 9.2.10

A dump/restore is not required for those running 9.2.X.

However, if you are a Windows user and are using the "Norwegian (Bokmål)" locale, manual action is needed after the upgrade to replace any "Norwegian (Bokmål)_Norway" locale names stored in PostgreSQL system catalogs with the plain-ASCII alias "Norwegian_Norway". For details see http://wiki.postgresql.org/wiki/Changes_To_Norwegian_Locale

Also, if you are upgrading from a version earlier than 9.2.9, see 第 E.16 节.

E.15.2. Changes

- Fix buffer overruns in `to_char()` (Bruce Momjian)

  When `to_char()` processes a numeric formatting template calling for a large number of digits, PostgreSQL would read past the end of a buffer. When processing

a crafted timestamp formatting template, PostgreSQL would write past the end of a buffer. Either case could crash the server. We have not ruled out the possibility of attacks that lead to privilege escalation, though they seem unlikely. (CVE-2015-0241)

- Fix buffer overrun in replacement `*printf()` functions (Tom Lane)

  PostgreSQL includes a replacement implementation of `printf` and related functions. This code will overrun a stack buffer when formatting a floating point number (conversion specifiers `e`, `E`, `f`, `F`, `g` or `G`) with requested precision greater than about 500. This will crash the server, and we have not ruled out the possibility of attacks that lead to privilege escalation. A database user can trigger such a buffer overrun through the `to_char()` SQL function. While that is the only affected core PostgreSQL functionality, extension modules that use printf-family functions may be at risk as well.

  This issue primarily affects PostgreSQL on Windows. PostgreSQL uses the system implementation of these functions where adequate, which it is on other modern platforms. (CVE-2015-0242)

- Fix buffer overruns in `contrib/pgcrypto` (Marko Tiikkaja, Noah Misch)

  Errors in memory size tracking within the `pgcrypto` module permitted stack buffer overruns and improper dependence on the contents of uninitialized memory. The buffer overrun cases can crash the server, and we have not ruled out the possibility of attacks that lead to privilege escalation. (CVE-2015-0243)

- Fix possible loss of frontend/backend protocol synchronization after an error (Heikki Linnakangas)

  If any error occurred while the server was in the middle of reading a protocol message from the client, it could lose synchronization and incorrectly try to interpret part of the message's data as a new protocol message. An attacker able to submit crafted binary data within a command parameter might succeed in injecting his own SQL commands this way. Statement timeout and query cancellation are the most likely sources of errors triggering this scenario. Particularly vulnerable are applications that use a timeout and also submit arbitrary user-crafted data as binary query parameters. Disabling statement timeout will reduce, but not eliminate, the risk of exploit. Our thanks to Emil Lenngren for reporting this issue. (CVE-2015-0244)

- Fix information leak via constraint-violation error messages (Stephen Frost)

  Some server error messages show the values of columns that violate a constraint, such as a unique constraint. If the user does not have `SELECT` privilege on all columns of the table, this could mean exposing values that the user should not be able to see. Adjust the code so that values are displayed only when they came from the SQL command or could be selected by the user. (CVE-2014-8161)

- Lock down regression testing's temporary installations on Windows (Noah Misch)

  Use SSPI authentication to allow connections only from the OS user who launched the test suite. This closes on Windows the same vulnerability previously closed on other platforms, namely that other users might be able to connect to the test postmaster. (CVE-2014-0067)

- Cope with the Windows locale named “Norwegian (Bokmål)” (Heikki Linnakangas)

  Non-ASCII locale names are problematic since it's not clear what encoding they should be represented in. Map the troublesome locale name to a plain-ASCII alias, “Norwegian_Norway” .

- Avoid possible data corruption if `ALTER DATABASE SET TABLESPACE` is used to move a database to a new tablespace and then shortly later move it back to its original tablespace (Tom Lane)
- Avoid corrupting tables when `ANALYZE` inside a transaction is rolled back (Andres Freund, Tom Lane, Michael Paquier)

  If the failing transaction had earlier removed the last index, rule, or trigger from the table, the table would be left in a corrupted state with the relevant `pg_class` flags not set though they should be.
- Ensure that unlogged tables are copied correctly during `CREATE DATABASE` or `ALTER DATABASE SET TABLESPACE` (Pavan Deolasee, Andres Freund)
- Fix `DROP`'s dependency searching to correctly handle the case where a table column is recursively visited before its table (Petr Jelinek, Tom Lane)

  This case is only known to arise when an extension creates both a datatype and a table using that datatype. The faulty code might refuse a `DROP EXTENSION` unless `CASCADE` is specified, which should not be required.
- Fix use-of-already-freed-memory problem in EvalPlanQual processing (Tom Lane)

  In `READ COMMITTED` mode, queries that lock or update recently-updated rows could crash as a result of this bug.
- Fix planning of `SELECT FOR UPDATE` when using a partial index on a child table (Kyotaro Horiguchi)

  In `READ COMMITTED` mode, `SELECT FOR UPDATE` must also recheck the partial index's `WHERE` condition when rechecking a recently-updated row to see if it still satisfies the query's `WHERE` condition. This requirement was missed if the index belonged to an inheritance child table, so that it was possible to incorrectly return rows that no longer satisfy the query condition.
- Fix corner case wherein `SELECT FOR UPDATE` could return a row twice, and possibly miss returning other rows (Tom Lane)

  In `READ COMMITTED` mode, a `SELECT FOR UPDATE` that is scanning an inheritance tree could incorrectly return a row from a prior child table instead of the one it should return from a later child table.
- Reject duplicate column names in the referenced-columns list of a `FOREIGN KEY` declaration (David Rowley)

  This restriction is per SQL standard. Previously we did not reject the case explicitly, but later on the code would fail with bizarre-looking errors.
- Restore previous behavior of conversion of domains to JSON (Tom Lane)

  This change causes domains over numeric and boolean to be treated like their base types for purposes of conversion to JSON. It worked like that before 9.3.5 and 9.2.9, but was unintentionally changed while fixing a related problem.
- Fix bugs in raising a `numeric` value to a large integral power (Tom Lane)

  The previous code could get a wrong answer, or consume excessive amounts of time and memory before realizing that the answer must overflow.
- In `numeric_recv()`, truncate away any fractional digits that would be hidden according to the value's `dscale` field (Tom Lane)

A `numeric` value's display scale (`dscale`) should never be less than the number of nonzero fractional digits; but apparently there's at least one broken client application that transmits binary `numeric` values in which that's true. This leads to strange behavior since the extra digits are taken into account by arithmetic operations even though they aren't printed. The least risky fix seems to be to truncate away such "hidden" digits on receipt, so that the value is indeed what it prints as.

- Fix incorrect search for shortest-first regular expression matches (Tom Lane)

  Matching would often fail when the number of allowed iterations is limited by a `?` quantifier or a bound expression.

- Reject out-of-range numeric timezone specifications (Tom Lane)

  Simple numeric timezone specifications exceeding +/- 168 hours (one week) would be accepted, but could then cause null-pointer dereference crashes in certain operations. There's no use-case for such large UTC offsets, so reject them.

- Fix bugs in `tsquery @> tsquery` operator (Heikki Linnakangas)

  Two different terms would be considered to match if they had the same CRC. Also, if the second operand had more terms than the first, it would be assumed not to be contained in the first; which is wrong since it might contain duplicate terms.

- Improve ispell dictionary's defenses against bad affix files (Tom Lane)

- Allow more than 64K phrases in a thesaurus dictionary (David Boutin)

  The previous coding could crash on an oversize dictionary, so this was deemed a back-patchable bug fix rather than a feature addition.

- Fix namespace handling in `xpath()` (Ali Akbar)

  Previously, the `xml` value resulting from an `xpath()` call would not have namespace declarations if the namespace declarations were attached to an ancestor element in the input `xml` value, rather than to the specific element being returned. Propagate the ancestral declaration so that the result is correct when considered in isolation.

- Ensure that whole-row variables expose nonempty column names to functions that pay attention to column names within composite arguments (Tom Lane)

  In some contexts, constructs like `row_to_json(tab.*)` may not produce the expected column names. This is fixed properly as of 9.4; in older branches, just ensure that we produce some nonempty name. (In some cases this will be the underlying table's column name rather than the query-assigned alias that should theoretically be visible.)

- Fix mishandling of system columns, particularly `tableoid`, in FDW queries (Etsuro Fujita)

- Avoid doing *`indexed_column`* `= ANY (`*`array`*`)` as an index qualifier if that leads to an inferior plan (Andrew Gierth)

  In some cases, `= ANY` conditions applied to non-first index columns would be done as index conditions even though it would be better to use them as simple filter conditions.

- Fix planner problems with nested append relations, such as inherited tables within `UNION ALL` subqueries (Tom Lane)

- Fail cleanly when a GiST index tuple doesn't fit on a page, rather than going into infinite recursion (Andrew Gierth)
- Exempt tables that have per-table `cost_limit` and/or `cost_delay` settings from autovacuum's global cost balancing rules (Álvaro Herrera)

  The previous behavior resulted in basically ignoring these per-table settings, which was unintended. Now, a table having such settings will be vacuumed using those settings, independently of what is going on in other autovacuum workers. This may result in heavier total I/O load than before, so such settings should be re-examined for sanity.
- Avoid wholesale autovacuuming when autovacuum is nominally off (Tom Lane)

  Even when autovacuum is nominally off, we will still launch autovacuum worker processes to vacuum tables that are at risk of XID wraparound. However, such a worker process then proceeded to vacuum all tables in the target database, if they met the usual thresholds for autovacuuming. This is at best pretty unexpected; at worst it delays response to the wraparound threat. Fix it so that if autovacuum is turned off, workers only do anti-wraparound vacuums and not any other work.
- During crash recovery, ensure that unlogged relations are rewritten as empty and are synced to disk before recovery is considered complete (Abhijit Menon-Sen, Andres Freund)

  This prevents scenarios in which unlogged relations might contain garbage data following database crash recovery.
- Fix race condition between hot standby queries and replaying a full-page image (Heikki Linnakangas)

  This mistake could result in transient errors in queries being executed in hot standby.
- Fix several cases where recovery logic improperly ignored WAL records for `COMMIT/ABORT PREPARED` (Heikki Linnakangas)

  The most notable oversight was that `recovery_target_xid` could not be used to stop at a two-phase commit.
- Prevent latest WAL file from being archived a second time at completion of crash recovery (Fujii Masao)
- Avoid creating unnecessary `.ready` marker files for timeline history files (Fujii Masao)
- Fix possible null pointer dereference when an empty prepared statement is used and the `log_statement` setting is `mod` or `ddl` (Fujii Masao)
- Change “pgstat wait timeout” warning message to be LOG level, and rephrase it to be more understandable (Tom Lane)

  This message was originally thought to be essentially a can't-happen case, but it occurs often enough on our slower buildfarm members to be a nuisance. Reduce it to LOG level, and expend a bit more effort on the wording: it now reads “using stale statistics instead of current ones because stats collector is not responding”.
- Fix SPARC spinlock implementation to ensure correctness if the CPU is being run in a non-TSO coherency mode, as some non-Solaris kernels do (Andres Freund)
- Warn if macOS's `setlocale()` starts an unwanted extra thread inside the postmaster (Noah Misch)

- Fix processing of repeated `dbname` parameters in `PQconnectdbParams()` (Alex Shulgin)

  Unexpected behavior ensued if the first occurrence of `dbname` contained a connection string or URI to be expanded.

- Ensure that libpq reports a suitable error message on unexpected socket EOF (Marko Tiikkaja, Tom Lane)

  Depending on kernel behavior, libpq might return an empty error string rather than something useful when the server unexpectedly closed the socket.

- Clear any old error message during `PQreset()` (Heikki Linnakangas)

  If `PQreset()` is called repeatedly, and the connection cannot be re-established, error messages from the failed connection attempts kept accumulating in the `PGconn`'s error string.

- Properly handle out-of-memory conditions while parsing connection options in libpq (Alex Shulgin, Heikki Linnakangas)

- Fix array overrun in ecpg's version of `ParseDateTime()` (Michael Paquier)

- In initdb, give a clearer error message if a password file is specified but is empty (Mats Erik Andersson)

- Fix psql's `\s` command to work nicely with libedit, and add pager support (Stepan Rutz, Tom Lane)

  When using libedit rather than readline, `\s` printed the command history in a fairly unreadable encoded format, and on recent libedit versions might fail altogether. Fix that by printing the history ourselves rather than having the library do it. A pleasant side-effect is that the pager is used if appropriate.

  This patch also fixes a bug that caused newline encoding to be applied inconsistently when saving the command history with libedit. Multiline history entries written by older psql versions will be read cleanly with this patch, but perhaps not vice versa, depending on the exact libedit versions involved.

- Improve consistency of parsing of psql's special variables (Tom Lane)

  Allow variant spellings of `on` and `off` (such as `1`/`0`) for `ECHO_HIDDEN` and `ON_ERROR_ROLLBACK`. Report a warning for unrecognized values for `COMP_KEYWORD_CASE`, `ECHO`, `ECHO_HIDDEN`, `HISTCONTROL`, `ON_ERROR_ROLLBACK`, and `VERBOSITY`. Recognize all values for all these variables case-insensitively; previously there was a mishmash of case-sensitive and case-insensitive behaviors.

- Fix psql's expanded-mode display to work consistently when using `border` = 3 and `linestyle` = `ascii` or `unicode` (Stephen Frost)

- Improve performance of pg_dump when the database contains many instances of multiple dependency paths between the same two objects (Tom Lane)

- Fix pg_dumpall to restore its ability to dump from pre-8.1 servers (Gilles Darold)

- Fix possible deadlock during parallel restore of a schema-only dump (Robert Haas, Tom Lane)

- Fix core dump in `pg_dump --binary-upgrade` on zero-column composite type (Rushabh Lathia)

- Prevent WAL files created by `pg_basebackup -x/-X` from being archived again when the standby is promoted (Andres Freund)
- Fix failure of `contrib/auto_explain` to print per-node timing information when doing `EXPLAIN ANALYZE` (Tom Lane)
- Fix upgrade-from-unpackaged script for `contrib/citext` (Tom Lane)
- Fix block number checking in `contrib/pageinspect`'s `get_raw_page()` (Tom Lane)

  The incorrect checking logic could prevent access to some pages in non-main relation forks.
- Fix `contrib/pgcrypto`'s `pgp_sym_decrypt()` to not fail on messages whose length is 6 less than a power of 2 (Marko Tiikkaja)
- Fix file descriptor leak in `contrib/pg_test_fsync` (Jeff Janes)

  This could cause failure to remove temporary files on Windows.
- Handle unexpected query results, especially NULLs, safely in `contrib/tablefunc`'s `connectby()` (Michael Paquier)

  `connectby()` previously crashed if it encountered a NULL key value. It now prints that row but doesn't recurse further.
- Avoid a possible crash in `contrib/xml2`'s `xslt_process()` (Mark Simonetti)

  libxslt seems to have an undocumented dependency on the order in which resources are freed; reorder our calls to avoid a crash.
- Mark some `contrib` I/O functions with correct volatility properties (Tom Lane)

  The previous over-conservative marking was immaterial in normal use, but could cause optimization problems or rejection of valid index expression definitions. Since the consequences are not large, we've just adjusted the function definitions in the extension modules' scripts, without changing version numbers.
- Numerous cleanups of warnings from Coverity static code analyzer (Andres Freund, Tatsuo Ishii, Marko Kreen, Tom Lane, Michael Paquier)

  These changes are mostly cosmetic but in some cases fix corner-case bugs, for example a crash rather than a proper error report after an out-of-memory failure. None are believed to represent security issues.
- Detect incompatible OpenLDAP versions during build (Noah Misch)

  With OpenLDAP versions 2.4.24 through 2.4.31, inclusive, PostgreSQL backends can crash at exit. Raise a warning during configure based on the compile-time OpenLDAP version number, and test the crashing scenario in the `contrib/dblink` regression test.
- In non-MSVC Windows builds, ensure `libpq.dll` is installed with execute permissions (Noah Misch)
- Make pg_regress remove any temporary installation it created upon successful exit (Tom Lane)

  This results in a very substantial reduction in disk space usage during `make check-world`, since that sequence involves creation of numerous temporary installations.

- Support time zone abbreviations that change UTC offset from time to time (Tom Lane)

  Previously, PostgreSQL assumed that the UTC offset associated with a time zone abbreviation (such as `EST`) never changes in the usage of any particular locale. However this assumption fails in the real world, so introduce the ability for a zone abbreviation to represent a UTC offset that sometimes changes. Update the zone abbreviation definition files to make use of this feature in timezone locales that have changed the UTC offset of their abbreviations since 1970 (according to the IANA timezone database). In such timezones, PostgreSQL will now associate the correct UTC offset with the abbreviation depending on the given date.

- Update time zone abbreviations lists (Tom Lane)

  Add CST (China Standard Time) to our lists. Remove references to ADT as "Arabia Daylight Time", an abbreviation that's been out of use since 2007; therefore, claiming there is a conflict with "Atlantic Daylight Time" doesn't seem especially helpful. Fix entirely incorrect GMT offsets for CKT (Cook Islands), FJT, and FJST (Fiji); we didn't even have them on the proper side of the date line.

- Update time zone data files to tzdata release 2015a.

  The IANA timezone database has adopted abbreviations of the form `AxST`/`AxDT` for all Australian time zones, reflecting what they believe to be current majority practice Down Under. These names do not conflict with usage elsewhere (other than ACST for Acre Summer Time, which has been in disuse since 1994). Accordingly, adopt these names into our "Default" timezone abbreviation set. The "Australia" abbreviation set now contains only CST, EAST, EST, SAST, SAT, and WST, all of which are thought to be mostly historical usage. Note that SAST has also been changed to be South Africa Standard Time in the "Default" abbreviation set.

  Also, add zone abbreviations SRET (Asia/Srednekolymsk) and XJT (Asia/Urumqi), and use WSST/WSDT for western Samoa. Also, there were DST law changes in Chile, Mexico, the Turks & Caicos Islands (America/Grand_Turk), and Fiji. There is a new zone Pacific/Bougainville for portions of Papua New Guinea. Also, numerous corrections for historical (pre-1970) time zone data.

E.16. Release 9.2.9

Release date: 2014-07-24

This release contains a variety of fixes from 9.2.8. For information about new features in the 9.2 major release, see 第 E.25 节.

E.16.1. Migration to Version 9.2.9

A dump/restore is not required for those running 9.2.X.

However, this release corrects an index corruption problem in some GiST indexes. See the first changelog entry below to find out whether your installation has been affected and what steps you should take if so.

Also, if you are upgrading from a version earlier than 9.2.6, see 第 E.19 节.

E.16.2. Changes

- Correctly initialize padding bytes in `contrib/btree_gist` indexes on `bit` columns (Heikki Linnakangas)

This error could result in incorrect query results due to values that should compare equal not being seen as equal. Users with GiST indexes on `bit` or `bit varying` columns should `REINDEX` those indexes after installing this update.

- Protect against torn pages when deleting GIN list pages (Heikki Linnakangas)

  This fix prevents possible index corruption if a system crash occurs while the page update is being written to disk.

- Don't clear the right-link of a GiST index page while replaying updates from WAL (Heikki Linnakangas)

  This error could lead to transiently wrong answers from GiST index scans performed in Hot Standby.

- Fix corner-case infinite loop during insertion into an SP-GiST text index (Tom Lane)

- Fix feedback status when hot_standby_feedback is turned off on-the-fly (Simon Riggs)

- Fix possibly-incorrect cache invalidation during nested calls to `ReceiveSharedInvalidMessages` (Andres Freund)

- Fix planner's mishandling of nested PlaceHolderVars generated in nested-nestloop plans (Tom Lane)

  This oversight could result in "variable not found in subplan target lists" errors, or in silently wrong query results.

- Fix "could not find pathkey item to sort" planner failures with `UNION ALL` over subqueries reading from tables with inheritance children (Tom Lane)

- Don't assume a subquery's output is unique if there's a set-returning function in its targetlist (David Rowley)

  This oversight could lead to misoptimization of constructs like `WHERE x IN (SELECT y, generate_series(1,10) FROM t GROUP BY y)`.

- Improve planner to drop constant-NULL inputs of `AND`/`OR` when possible (Tom Lane)

  This change fixes some cases where the more aggressive parameter substitution done by 9.2 and later can lead to a worse plan than older versions produced.

- Fix identification of input type category in `to_json()` and friends (Tom Lane)

  This is known to have led to inadequate quoting of `money` fields in the `JSON` result, and there may have been wrong results for other data types as well.

- Fix failure to detoast fields in composite elements of structured types (Tom Lane)

  This corrects cases where TOAST pointers could be copied into other tables without being dereferenced. If the original data is later deleted, it would lead to errors like "missing chunk number 0 for toast value ..." when the now-dangling pointer is used.

- Fix "record type has not been registered" failures with whole-row references to the output of Append plan nodes (Tom Lane)

- Fix possible crash when invoking a user-defined function while rewinding a cursor (Tom Lane)

- Fix query-lifespan memory leak while evaluating the arguments for a function in `FROM` (Tom Lane)
- Fix session-lifespan memory leaks in regular-expression processing (Tom Lane, Arthur O'Dwyer, Greg Stark)
- Fix data encoding error in `hungarian.stop` (Tom Lane)
- Prevent foreign tables from being created with OIDS when default_with_oids is true (Etsuro Fujita)
- Fix liveness checks for rows that were inserted in the current transaction and then deleted by a now-rolled-back subtransaction (Andres Freund)

  This could cause problems (at least spurious warnings, and at worst an infinite loop) if `CREATE INDEX` or `CLUSTER` were done later in the same transaction.
- Clear `pg_stat_activity.xact_start` during `PREPARE TRANSACTION` (Andres Freund)

  After the `PREPARE`, the originating session is no longer in a transaction, so it should not continue to display a transaction start time.
- Fix `REASSIGN OWNED` to not fail for text search objects (Álvaro Herrera)
- Block signals during postmaster startup (Tom Lane)

  This ensures that the postmaster will properly clean up after itself if, for example, it receives SIGINT while still starting up.
- Fix client host name lookup when processing `pg_hba.conf` entries that specify host names instead of IP addresses (Tom Lane)

  Ensure that reverse-DNS lookup failures are reported, instead of just silently not matching such entries. Also ensure that we make only one reverse-DNS lookup attempt per connection, not one per host name entry, which is what previously happened if the lookup attempts failed.
- Allow the root user to use `postgres -C variable` and `postgres --describe-config` (MauMau)

  The prohibition on starting the server as root does not need to extend to these operations, and relaxing it prevents failure of pg_ctl in some scenarios.
- Secure Unix-domain sockets of temporary postmasters started during `make check` (Noah Misch)

  Any local user able to access the socket file could connect as the server's bootstrap superuser, then proceed to execute arbitrary code as the operating-system user running the test, as we previously noted in CVE-2014-0067. This change defends against that risk by placing the server's socket in a temporary, mode 0700 subdirectory of `/tmp`. The hazard remains however on platforms where Unix sockets are not supported, notably Windows, because then the temporary postmaster must accept local TCP connections.

  A useful side effect of this change is to simplify `make check` testing in builds that override `DEFAULT_PGSOCKET_DIR`. Popular non-default values like `/var/run/postgresql` are often not writable by the build user, requiring workarounds that will no longer be necessary.
- Fix tablespace creation WAL replay to work on Windows (MauMau)

- Fix detection of socket creation failures on Windows (Bruce Momjian)
- On Windows, allow new sessions to absorb values of PGC_BACKEND parameters (such as log_connections) from the configuration file (Amit Kapila)

  Previously, if such a parameter were changed in the file post-startup, the change would have no effect.
- Properly quote executable path names on Windows (Nikhil Deshpande)

  This oversight could cause initdb and pg_upgrade to fail on Windows, if the installation path contained both spaces and `@` signs.
- Fix linking of libpython on macOS (Tom Lane)

  The method we previously used can fail with the Python library supplied by Xcode 5.0 and later.
- Avoid buffer bloat in libpq when the server consistently sends data faster than the client can absorb it (Shin-ichi Morita, Tom Lane)

  libpq could be coerced into enlarging its input buffer until it runs out of memory (which would be reported misleadingly as "lost synchronization with server"). Under ordinary circumstances it's quite far-fetched that data could be continuously transmitted more quickly than the `recv()` loop can absorb it, but this has been observed when the client is artificially slowed by scheduler constraints.
- Ensure that LDAP lookup attempts in libpq time out as intended (Laurenz Albe)
- Fix ecpg to do the right thing when an array of `char *` is the target for a FETCH statement returning more than one row, as well as some other array-handling fixes (Ashutosh Bapat)
- Fix pg_restore's processing of old-style large object comments (Tom Lane)

  A direct-to-database restore from an archive file generated by a pre-9.0 version of pg_dump would usually fail if the archive contained more than a few comments for large objects.
- Fix pg_upgrade for cases where the new server creates a TOAST table but the old version did not (Bruce Momjian)

  This rare situation would manifest as "relation OID mismatch" errors.
- Prevent `contrib/auto_explain` from changing the output of a user's `EXPLAIN` (Tom Lane)

  If `auto_explain` is active, it could cause an `EXPLAIN (ANALYZE, TIMING OFF)` command to nonetheless print timing information.
- Fix query-lifespan memory leak in `contrib/dblink` (MauMau, Joe Conway)
- In `contrib/pgcrypto` functions, ensure sensitive information is cleared from stack variables before returning (Marko Kreen)
- Prevent use of already-freed memory in `contrib/pgstattuple`'s `pgstat_heap()` (Noah Misch)
- In `contrib/uuid-ossp`, cache the state of the OSSP UUID library across calls (Tom Lane)

  This improves the efficiency of UUID generation and reduces the amount of entropy drawn from `/dev/urandom`, on platforms that have that.

- Update time zone data files to tzdata release 2014e for DST law changes in Crimea, Egypt, and Morocco.

E.17. Release 9.2.8

Release date: 2014-03-20

This release contains a variety of fixes from 9.2.7. For information about new features in the 9.2 major release, see 第 E.25 节.

E.17.1. Migration to Version 9.2.8

A dump/restore is not required for those running 9.2.X.

However, if you are upgrading from a version earlier than 9.2.6, see 第 E.19 节.

E.17.2. Changes

- Restore GIN metapages unconditionally to avoid torn-page risk (Heikki Linnakangas)

  Although this oversight could theoretically result in a corrupted index, it is unlikely to have caused any problems in practice, since the active part of a GIN metapage is smaller than a standard 512-byte disk sector.

- Avoid race condition in checking transaction commit status during receipt of a `NOTIFY` message (Marko Tiikkaja)

  This prevents a scenario wherein a sufficiently fast client might respond to a notification before database updates made by the notifier have become visible to the recipient.

- Allow regular-expression operators to be terminated early by query cancel requests (Tom Lane)

  This prevents scenarios wherein a pathological regular expression could lock up a server process uninterruptibly for a long time.

- Remove incorrect code that tried to allow `OVERLAPS` with single-element row arguments (Joshua Yanovski)

  This code never worked correctly, and since the case is neither specified by the SQL standard nor documented, it seemed better to remove it than fix it.

- Avoid getting more than `AccessShareLock` when de-parsing a rule or view (Dean Rasheed)

  This oversight resulted in pg_dump unexpectedly acquiring `RowExclusiveLock` locks on tables mentioned as the targets of `INSERT`/`UPDATE`/`DELETE` commands in rules. While usually harmless, that could interfere with concurrent transactions that tried to acquire, for example, `ShareLock` on those tables.

- Improve performance of index endpoint probes during planning (Tom Lane)

  This change fixes a significant performance problem that occurred when there were many not-yet-committed rows at the end of the index, which is a common situation for indexes on sequentially-assigned values such as timestamps or sequence-generated identifiers.

- Fix walsender's failure to shut down cleanly when client is pg_receivexlog (Fujii Masao)

- Check WAL level and hot standby parameters correctly when doing crash recovery that will be followed by archive recovery (Heikki Linnakangas)
- Fix test to see if hot standby connections can be allowed immediately after a crash (Heikki Linnakangas)
- Prevent interrupts while reporting non-`ERROR` messages (Tom Lane)

  This guards against rare server-process freezeups due to recursive entry to `syslog()`, and perhaps other related problems.
- Fix memory leak in PL/Perl when returning a composite result, including multiple-OUT-parameter cases (Alex Hunsaker)
- Fix tracking of psql script line numbers during `\copy` from out-of-line data (Kumar Rajeev Rastogi, Amit Khandekar)

  `\copy ... from` incremented the script file line number for each data line, even if the data was not coming from the script file. This mistake resulted in wrong line numbers being reported for any errors occurring later in the same script file.
- Prevent intermittent "could not reserve shared memory region" failures on recent Windows versions (MauMau)
- Update time zone data files to tzdata release 2014a for DST law changes in Fiji and Turkey, plus historical changes in Israel and Ukraine.

E.18. Release 9.2.7

Release date: 2014-02-20

This release contains a variety of fixes from 9.2.6. For information about new features in the 9.2 major release, see 第 E.25 节.

E.18.1. Migration to Version 9.2.7

A dump/restore is not required for those running 9.2.X.

However, if you are upgrading from a version earlier than 9.2.6, see 第 E.19 节.

E.18.2. Changes

- Shore up `GRANT ... WITH ADMIN OPTION` restrictions (Noah Misch)

  Granting a role without `ADMIN OPTION` is supposed to prevent the grantee from adding or removing members from the granted role, but this restriction was easily bypassed by doing `SET ROLE` first. The security impact is mostly that a role member can revoke the access of others, contrary to the wishes of his grantor. Unapproved role member additions are a lesser concern, since an uncooperative role member could provide most of his rights to others anyway by creating views or `SECURITY DEFINER` functions. (CVE-2014-0060)
- Prevent privilege escalation via manual calls to PL validator functions (Andres Freund)

  The primary role of PL validator functions is to be called implicitly during `CREATE FUNCTION`, but they are also normal SQL functions that a user can call explicitly. Calling a validator on a function actually written in some other language was not checked for and could be exploited for privilege-escalation purposes. The fix

involves adding a call to a privilege-checking function in each validator function. Non-core procedural languages will also need to make this change to their own validator functions, if any. (CVE-2014-0061)

- Avoid multiple name lookups during table and index DDL (Robert Haas, Andres Freund)

  If the name lookups come to different conclusions due to concurrent activity, we might perform some parts of the DDL on a different table than other parts. At least in the case of `CREATE INDEX`, this can be used to cause the permissions checks to be performed against a different table than the index creation, allowing for a privilege escalation attack. (CVE-2014-0062)

- Prevent buffer overrun with long datetime strings (Noah Misch)

  The `MAXDATELEN` constant was too small for the longest possible value of type `interval`, allowing a buffer overrun in `interval_out()`. Although the datetime input functions were more careful about avoiding buffer overrun, the limit was short enough to cause them to reject some valid inputs, such as input containing a very long timezone name. The ecpg library contained these vulnerabilities along with some of its own. (CVE-2014-0063)

- Prevent buffer overrun due to integer overflow in size calculations (Noah Misch, Heikki Linnakangas)

  Several functions, mostly type input functions, calculated an allocation size without checking for overflow. If overflow did occur, a too-small buffer would be allocated and then written past. (CVE-2014-0064)

- Prevent overruns of fixed-size buffers (Peter Eisentraut, Jozef Mlich)

  Use `strlcpy()` and related functions to provide a clear guarantee that fixed-size buffers are not overrun. Unlike the preceding items, it is unclear whether these cases really represent live issues, since in most cases there appear to be previous constraints on the size of the input string. Nonetheless it seems prudent to silence all Coverity warnings of this type. (CVE-2014-0065)

- Avoid crashing if `crypt()` returns NULL (Honza Horak, Bruce Momjian)

  There are relatively few scenarios in which `crypt()` could return NULL, but `contrib/chkpass` would crash if it did. One practical case in which this could be an issue is if libc is configured to refuse to execute unapproved hashing algorithms (e.g., “FIPS mode”). (CVE-2014-0066)

- Document risks of `make check` in the regression testing instructions (Noah Misch, Tom Lane)

  Since the temporary server started by `make check` uses “trust” authentication, another user on the same machine could connect to it as database superuser, and then potentially exploit the privileges of the operating-system user who started the tests. A future release will probably incorporate changes in the testing procedure to prevent this risk, but some public discussion is needed first. So for the moment, just warn people against using `make check` when there are untrusted users on the same machine. (CVE-2014-0067)

- Fix possible mis-replay of WAL records when some segments of a relation aren't full size (Greg Stark, Tom Lane)

  The WAL update could be applied to the wrong page, potentially many pages past where it should have been. Aside from corrupting data, this error has been observed

to result in significant “bloat” of standby servers compared to their masters, due to updates being applied far beyond where the end-of-file should have been. This failure mode does not appear to be a significant risk during crash recovery, only when initially synchronizing a standby created from a base backup taken from a quickly-changing master.

- Fix bug in determining when recovery has reached consistency (Tomonari Katsumata, Heikki Linnakangas)

  In some cases WAL replay would mistakenly conclude that the database was already consistent at the start of replay, thus possibly allowing hot-standby queries before the database was really consistent. Other symptoms such as “PANIC: WAL contains references to invalid pages” were also possible.

- Fix improper locking of btree index pages while replaying a `VACUUM` operation in hot-standby mode (Andres Freund, Heikki Linnakangas, Tom Lane)

  This error could result in “PANIC: WAL contains references to invalid pages” failures.

- Ensure that insertions into non-leaf GIN index pages write a full-page WAL record when appropriate (Heikki Linnakangas)

  The previous coding risked index corruption in the event of a partial-page write during a system crash.

- When `pause_at_recovery_target` and `recovery_target_inclusive` are both set, ensure the target record is applied before pausing, not after (Heikki Linnakangas)

- Fix race conditions during server process exit (Robert Haas)

  Ensure that signal handlers don't attempt to use the process's `MyProc` pointer after it's no longer valid.

- Fix race conditions in walsender shutdown logic and walreceiver SIGHUP signal handler (Tom Lane)

- Fix unsafe references to `errno` within error reporting logic (Christian Kruse)

  This would typically lead to odd behaviors such as missing or inappropriate `HINT` fields.

- Fix possible crashes from using `ereport()` too early during server startup (Tom Lane)

  The principal case we've seen in the field is a crash if the server is started in a directory it doesn't have permission to read.

- Clear retry flags properly in OpenSSL socket write function (Alexander Kukushkin)

  This omission could result in a server lockup after unexpected loss of an SSL-encrypted connection.

- Fix length checking for Unicode identifiers (`U&"..."` syntax) containing escapes (Tom Lane)

  A spurious truncation warning would be printed for such identifiers if the escaped form of the identifier was too long, but the identifier actually didn't need truncation after de-escaping.

- Allow keywords that are type names to be used in lists of roles (Stephen Frost)

A previous patch allowed such keywords to be used without quoting in places such as role identifiers; but it missed cases where a list of role identifiers was permitted, such as `DROP ROLE`.

- Fix parser crash for `EXISTS(SELECT * FROM zero_column_table)` (Tom Lane)
- Fix possible crash due to invalid plan for nested sub-selects, such as `WHERE (... x IN (SELECT ...) ...) IN (SELECT ...)` (Tom Lane)
- Fix `UPDATE/DELETE` of an inherited target table that has `UNION ALL` subqueries (Tom Lane)

  Without this fix, `UNION ALL` subqueries aren't correctly inserted into the update plans for inheritance child tables after the first one, typically resulting in no update happening for those child table(s).

- Ensure that `ANALYZE` creates statistics for a table column even when all the values in it are "too wide" (Tom Lane)

  `ANALYZE` intentionally omits very wide values from its histogram and most-common-values calculations, but it neglected to do something sane in the case that all the sampled entries are too wide.

- In `ALTER TABLE ... SET TABLESPACE`, allow the database's default tablespace to be used without a permissions check (Stephen Frost)

  `CREATE TABLE` has always allowed such usage, but `ALTER TABLE` didn't get the memo.

- Fix "cannot accept a set" error when some arms of a `CASE` return a set and others don't (Tom Lane)
- Properly distinguish numbers from non-numbers when generating JSON output (Andrew Dunstan)
- Fix checks for all-zero client addresses in pgstat functions (Kevin Grittner)
- Fix possible misclassification of multibyte characters by the text search parser (Tom Lane)

  Non-ASCII characters could be misclassified when using C locale with a multibyte encoding. On Cygwin, non-C locales could fail as well.

- Fix possible misbehavior in `plainto_tsquery()` (Heikki Linnakangas)

  Use `memmove()` not `memcpy()` for copying overlapping memory regions. There have been no field reports of this actually causing trouble, but it's certainly risky.

- Fix placement of permissions checks in `pg_start_backup()` and `pg_stop_backup()` (Andres Freund, Magnus Hagander)

  The previous coding might attempt to do catalog access when it shouldn't.

- Accept `SHIFT_JIS` as an encoding name for locale checking purposes (Tatsuo Ishii)
- Fix `*`-qualification of named parameters in SQL-language functions (Tom Lane)

  Given a composite-type parameter named `foo`, `$1.*` worked fine, but `foo.*` not so much.

- Fix misbehavior of `PQhost()` on Windows (Fujii Masao)

  It should return `localhost` if no host has been specified.

- Improve error handling in libpq and psql for failures during `COPY TO STDOUT/FROM STDIN` (Tom Lane)

  In particular this fixes an infinite loop that could occur in 9.2 and up if the server connection was lost during `COPY FROM STDIN`. Variants of that scenario might be possible in older versions, or with other client applications.

- Fix incorrect translation handling in some psql `\d` commands (Peter Eisentraut, Tom Lane)

- Ensure pg_basebackup's background process is killed when exiting its foreground process (Magnus Hagander)

- Fix possible incorrect printing of filenames in pg_basebackup's verbose mode (Magnus Hagander)

- Avoid including tablespaces inside PGDATA twice in base backups (Dimitri Fontaine, Magnus Hagander)

- Fix misaligned descriptors in ecpg (MauMau)

- In ecpg, handle lack of a hostname in the connection parameters properly (Michael Meskes)

- Fix performance regression in `contrib/dblink` connection startup (Joe Conway)

  Avoid an unnecessary round trip when client and server encodings match.

- In `contrib/isn`, fix incorrect calculation of the check digit for ISMN values (Fabien Coelho)

- Fix `contrib/pg_stat_statement`'s handling of `CURRENT_DATE` and related constructs (Kyotaro Horiguchi)

- Ensure client-code-only installation procedure works as documented (Peter Eisentraut)

- In Mingw and Cygwin builds, install the libpq DLL in the `bin` directory (Andrew Dunstan)

  This duplicates what the MSVC build has long done. It should fix problems with programs like psql failing to start because they can't find the DLL.

- Avoid using the deprecated `dllwrap` tool in Cygwin builds (Marco Atzeri)

- Don't generate plain-text `HISTORY` and `src/test/regress/README` files anymore (Tom Lane)

  These text files duplicated the main HTML and PDF documentation formats. The trouble involved in maintaining them greatly outweighs the likely audience for plain-text format. Distribution tarballs will still contain files by these names, but they'll just be stubs directing the reader to consult the main documentation. The plain-text `INSTALL` file will still be maintained, as there is arguably a use-case for that.

- Update time zone data files to tzdata release 2013i for DST law changes in Jordan and historical changes in Cuba.

In addition, the zones `Asia/Riyadh87`, `Asia/Riyadh88`, and `Asia/Riyadh89` have been removed, as they are no longer maintained by IANA, and never represented actual civil timekeeping practice.

E.19. Release 9.2.6

Release date: 2013-12-05

This release contains a variety of fixes from 9.2.5. For information about new features in the 9.2 major release, see 第 E.25 节.

E.19.1. Migration to Version 9.2.6

A dump/restore is not required for those running 9.2.X.

However, this release corrects a number of potential data corruption issues. See the first two changelog entries below to find out whether your installation has been affected and what steps you can take if so.

Also, if you are upgrading from a version earlier than 9.2.4, see 第 E.21 节.

E.19.2. Changes

- Fix `VACUUM`'s tests to see whether it can update `relfrozenxid` (Andres Freund)

  In some cases `VACUUM` (either manual or autovacuum) could incorrectly advance a table's `relfrozenxid` value, allowing tuples to escape freezing, causing those rows to become invisible once 2^31 transactions have elapsed. The probability of data loss is fairly low since multiple incorrect advancements would need to happen before actual loss occurs, but it's not zero. In 9.2.0 and later, the probability of loss is higher, and it's also possible to get "could not access status of transaction" errors as a consequence of this bug. Users upgrading from releases 9.0.4 or 8.4.8 or earlier are not affected, but all later versions contain the bug.

  The issue can be ameliorated by, after upgrading, vacuuming all tables in all databases while having `vacuum_freeze_table_age` set to zero. This will fix any latent corruption but will not be able to fix all pre-existing data errors. However, an installation can be presumed safe after performing this vacuuming if it has executed fewer than 2^31 update transactions in its lifetime (check this with `SELECT txid_current() < 2^31`).

- Fix initialization of `pg_clog` and `pg_subtrans` during hot standby startup (Andres Freund, Heikki Linnakangas)

  This bug can cause data loss on standby servers at the moment they start to accept hot-standby queries, by marking committed transactions as uncommitted. The likelihood of such corruption is small unless, at the time of standby startup, the primary server has executed many updating transactions since its last checkpoint. Symptoms include missing rows, rows that should have been deleted being still visible, and obsolete versions of updated rows being still visible alongside their newer versions.

  This bug was introduced in versions 9.3.0, 9.2.5, 9.1.10, and 9.0.14. Standby servers that have only been running earlier releases are not at risk. It's recommended that standby servers that have ever run any of the buggy releases be re-cloned from the primary (e.g., with a new base backup) after upgrading.

- Fix dangling-pointer problem in fast-path locking (Tom Lane)

This could lead to corruption of the lock data structures in shared memory, causing "lock already held" and other odd errors.

- Truncate `pg_multixact` contents during WAL replay (Andres Freund)

  This avoids ever-increasing disk space consumption in standby servers.

- Ensure an anti-wraparound `VACUUM` counts a page as scanned when it's only verified that no tuples need freezing (Sergey Burladyan, Jeff Janes)

  This bug could result in failing to advance `relfrozenxid`, so that the table would still be thought to need another anti-wraparound vacuum. In the worst case the database might even shut down to prevent wraparound.

- Fix race condition in GIN index posting tree page deletion (Heikki Linnakangas)

  This could lead to transient wrong answers or query failures.

- Fix "unexpected spgdoinsert() failure" error during SP-GiST index creation (Teodor Sigaev)

- Avoid flattening a subquery whose `SELECT` list contains a volatile function wrapped inside a sub-`SELECT` (Tom Lane)

  This avoids unexpected results due to extra evaluations of the volatile function.

- Fix planner's processing of non-simple-variable subquery outputs nested within outer joins (Tom Lane)

  This error could lead to incorrect plans for queries involving multiple levels of subqueries within `JOIN` syntax.

- Fix incorrect planning in cases where the same non-strict expression appears in multiple `WHERE` and outer `JOIN` equality clauses (Tom Lane)

- Fix planner crash with whole-row reference to a subquery (Tom Lane)

- Fix incorrect generation of optimized MIN()/MAX() plans for inheritance trees (Tom Lane)

  The planner could fail in cases where the MIN()/MAX() argument was an expression rather than a simple variable.

- Fix premature deletion of temporary files (Andres Freund)

- Prevent intra-transaction memory leak when printing range values (Tom Lane)

  This fix actually cures transient memory leaks in any datatype output function, but range types are the only ones known to have had a significant problem.

- Prevent incorrect display of dropped columns in NOT NULL and CHECK constraint violation messages (Michael Paquier and Tom Lane)

- Allow default arguments and named-argument notation for window functions (Tom Lane)

  Previously, these cases were likely to crash.

- Fix possible read past end of memory in rule printing (Peter Eisentraut)

- Fix array slicing of `int2vector` and `oidvector` values (Tom Lane)

Expressions of this kind are now implicitly promoted to regular `int2` or `oid` arrays.

- Fix incorrect behaviors when using a SQL-standard, simple GMT offset timezone (Tom Lane)

  In some cases, the system would use the simple GMT offset value when it should have used the regular timezone setting that had prevailed before the simple offset was selected. This change also causes the `timeofday` function to honor the simple GMT offset zone.

- Prevent possible misbehavior when logging translations of Windows error codes (Tom Lane)

- Properly quote generated command lines in pg_ctl (Naoya Anzai and Tom Lane)

  This fix applies only to Windows.

- Fix pg_dumpall to work when a source database sets `default_transaction_read_only` via `ALTER DATABASE SET` (Kevin Grittner)

  Previously, the generated script would fail during restore.

- Make ecpg search for quoted cursor names case-sensitively (Zoltán Böszörményi)

- Fix ecpg's processing of lists of variables declared `varchar` (Zoltán Böszörményi)

- Make `contrib/lo` defend against incorrect trigger definitions (Marc Cousin)

- Update time zone data files to tzdata release 2013h for DST law changes in Argentina, Brazil, Jordan, Libya, Liechtenstein, Morocco, and Palestine. Also, new timezone abbreviations WIB, WIT, WITA for Indonesia.

E.20. Release 9.2.5

Release date: 2013-10-10

This release contains a variety of fixes from 9.2.4. For information about new features in the 9.2 major release, see 第 E.25 节.

E.20.1. Migration to Version 9.2.5

A dump/restore is not required for those running 9.2.X.

However, if you are upgrading from a version earlier than 9.2.4, see 第 E.21 节.

E.20.2. Changes

- Prevent corruption of multi-byte characters when attempting to case-fold identifiers (Andrew Dunstan)

  PostgreSQL case-folds non-ASCII characters only when using a single-byte server encoding.

- Fix memory leak when creating B-tree indexes on range columns (Heikki Linnakangas)

- Fix checkpoint memory leak in background writer when `wal_level = hot_standby` (Naoya Anzai)

- Fix memory leak caused by `lo_open()` failure (Heikki Linnakangas)

- Fix memory overcommit bug when `work_mem` is using more than 24GB of memory (Stephen Frost)
- Serializable snapshot fixes (Kevin Grittner, Heikki Linnakangas)
- Fix deadlock bug in libpq when using SSL (Stephen Frost)
- Fix possible SSL state corruption in threaded libpq applications (Nick Phillips, Stephen Frost)
- Improve estimate of planner cost when choosing between generic and custom plans (Tom Lane)

  This change will favor generic plans when planning cost is high.
- Properly compute row estimates for boolean columns containing many NULL values (Andrew Gierth)

  Previously tests like `col IS NOT TRUE` and `col IS NOT FALSE` did not properly factor in NULL values when estimating plan costs.
- Fix accounting for qualifier evaluation costs in `UNION ALL` and inheritance queries (Tom Lane)

  This fixes cases where suboptimal query plans could be chosen if some `WHERE` clauses are expensive to calculate.
- Prevent pushing down `WHERE` clauses into unsafe `UNION/INTERSECT` subqueries (Tom Lane)

  Subqueries of a `UNION` or `INTERSECT` that contain set-returning functions or volatile functions in their `SELECT` lists could be improperly optimized, leading to run-time errors or incorrect query results.
- Fix rare case of "failed to locate grouping columns" planner failure (Tom Lane)
- Fix pg_dump of foreign tables with dropped columns (Andrew Dunstan)

  Previously such cases could cause a pg_upgrade error.
- Reorder pg_dump processing of extension-related rules and event triggers (Joe Conway)
- Force dumping of extension tables if specified by `pg_dump -t` or `-n` (Joe Conway)
- Improve view dumping code's handling of dropped columns in referenced tables (Tom Lane)
- Fix `pg_restore -l` with the directory archive to display the correct format name (Fujii Masao)
- Properly record index comments created using `UNIQUE` and `PRIMARY KEY` syntax (Andres Freund)

  This fixes a parallel pg_restore failure.
- Cause `pg_basebackup -x` with an empty xlog directory to throw an error rather than crashing (Magnus Hagander, Haruka Takatsuka)
- Properly guarantee transmission of WAL files before clean switchover (Fujii Masao)

  Previously, the streaming replication connection might close before all WAL files had been replayed on the standby.

- Fix WAL segment timeline handling during recovery (Mitsumasa Kondo, Heikki Linnakangas)

  WAL file recycling during standby recovery could lead to premature recovery completion, resulting in data loss.

- Prevent errors in WAL replay due to references to uninitialized empty pages (Andres Freund)

- Fix `REINDEX TABLE` and `REINDEX DATABASE` to properly revalidate constraints and mark invalidated indexes as valid (Noah Misch)

  `REINDEX INDEX` has always worked properly.

- Avoid deadlocks during insertion into SP-GiST indexes (Teodor Sigaev)

- Fix possible deadlock during concurrent `CREATE INDEX CONCURRENTLY` operations (Tom Lane)

- Fix GiST index lookup crash (Tom Lane)

- Fix `regexp_matches()` handling of zero-length matches (Jeevan Chalke)

  Previously, zero-length matches like '^' could return too many matches.

- Fix crash for overly-complex regular expressions (Heikki Linnakangas)

- Fix regular expression match failures for back references combined with non-greedy quantifiers (Jeevan Chalke)

- Prevent `CREATE FUNCTION` from checking `SET` variables unless function body checking is enabled (Tom Lane)

- Allow `ALTER DEFAULT PRIVILEGES` to operate on schemas without requiring CREATE permission (Tom Lane)

- Loosen restriction on keywords used in queries (Tom Lane)

  Specifically, lessen keyword restrictions for role names, language names, `EXPLAIN` and `COPY` options, and `SET` values. This allows `COPY ... (FORMAT BINARY)` to work as expected; previously `BINARY` needed to be quoted.

- Print proper line number during `COPY` failure (Heikki Linnakangas)

- Fix `pgp_pub_decrypt()` so it works for secret keys with passwords (Marko Kreen)

- Make pg_upgrade use `pg_dump --quote-all-identifiers` to avoid problems with keyword changes between releases (Tom Lane)

- Remove rare inaccurate warning during vacuum of index-less tables (Heikki Linnakangas)

- Ensure that `VACUUM ANALYZE` still runs the ANALYZE phase if its attempt to truncate the file is cancelled due to lock conflicts (Kevin Grittner)

- Avoid possible failure when performing transaction control commands (e.g `ROLLBACK`) in prepared queries (Tom Lane)

- Ensure that floating-point data input accepts standard spellings of “infinity” on all platforms (Tom Lane)

The C99 standard says that allowable spellings are `inf`, `+inf`, `-inf`, `infinity`, `+infinity`, and `-infinity`. Make sure we recognize these even if the platform's `strtod` function doesn't.

- Avoid unnecessary reporting when `track_activities` is off (Tom Lane)
- Expand ability to compare rows to records and arrays (Rafal Rzepecki, Tom Lane)
- Prevent crash when psql's `PSQLRC` variable contains a tilde (Bruce Momjian)
- Add spinlock support for ARM64 (Mark Salter)
- Update time zone data files to tzdata release 2013d for DST law changes in Israel, Morocco, Palestine, and Paraguay. Also, historical zone data corrections for Macquarie Island.

E.21. Release 9.2.4

Release date: 2013-04-04

This release contains a variety of fixes from 9.2.3. For information about new features in the 9.2 major release, see 第 E.25 节.

E.21.1. Migration to Version 9.2.4

A dump/restore is not required for those running 9.2.X.

However, this release corrects several errors in management of GiST indexes. After installing this update, it is advisable to `REINDEX` any GiST indexes that meet one or more of the conditions described below.

Also, if you are upgrading from a version earlier than 9.2.2, see 第 E.23 节.

E.21.2. Changes

- Fix insecure parsing of server command-line switches (Mitsumasa Kondo, Kyotaro Horiguchi)

  A connection request containing a database name that begins with "-" could be crafted to damage or destroy files within the server's data directory, even if the request is eventually rejected. (CVE-2013-1899)

- Reset OpenSSL randomness state in each postmaster child process (Marko Kreen)

  This avoids a scenario wherein random numbers generated by `contrib/pgcrypto` functions might be relatively easy for another database user to guess. The risk is only significant when the postmaster is configured with `ssl` = `on` but most connections don't use SSL encryption. (CVE-2013-1900)

- Make REPLICATION privilege checks test current user not authenticated user (Noah Misch)

  An unprivileged database user could exploit this mistake to call `pg_start_backup()` or `pg_stop_backup()`, thus possibly interfering with creation of routine backups. (CVE-2013-1901)

- Fix GiST indexes to not use "fuzzy" geometric comparisons when it's not appropriate to do so (Alexander Korotkov)

The core geometric types perform comparisons using "fuzzy" equality, but `gist_box_same` must do exact comparisons, else GiST indexes using it might become inconsistent. After installing this update, users should `REINDEX` any GiST indexes on `box`, `polygon`, `circle`, or `point` columns, since all of these use `gist_box_same`.

- Fix erroneous range-union and penalty logic in GiST indexes that use `contrib/btree_gist` for variable-width data types, that is `text`, `bytea`, `bit`, and `numeric` columns (Tom Lane)

  These errors could result in inconsistent indexes in which some keys that are present would not be found by searches, and also in useless index bloat. Users are advised to `REINDEX` such indexes after installing this update.

- Fix bugs in GiST page splitting code for multi-column indexes (Tom Lane)

  These errors could result in inconsistent indexes in which some keys that are present would not be found by searches, and also in indexes that are unnecessarily inefficient to search. Users are advised to `REINDEX` multi-column GiST indexes after installing this update.

- Fix `gist_point_consistent` to handle fuzziness consistently (Alexander Korotkov)

  Index scans on GiST indexes on `point` columns would sometimes yield results different from a sequential scan, because `gist_point_consistent` disagreed with the underlying operator code about whether to do comparisons exactly or fuzzily.

- Fix buffer leak in WAL replay (Heikki Linnakangas)

  This bug could result in "incorrect local pin count" errors during replay, making recovery impossible.

- Ensure we do crash recovery before entering archive recovery, if the database was not stopped cleanly and a `recovery.conf` file is present (Heikki Linnakangas, Kyotaro Horiguchi, Mitsumasa Kondo)

  This is needed to ensure that the database is consistent in certain scenarios, such as initializing a standby server with a filesystem snapshot from a running server.

- Avoid deleting not-yet-archived WAL files during crash recovery (Heikki Linnakangas, Fujii Masao)

- Fix race condition in `DELETE RETURNING` (Tom Lane)

  Under the right circumstances, `DELETE RETURNING` could attempt to fetch data from a shared buffer that the current process no longer has any pin on. If some other process changed the buffer meanwhile, this would lead to garbage `RETURNING` output, or even a crash.

- Fix infinite-loop risk in regular expression compilation (Tom Lane, Don Porter)

- Fix potential null-pointer dereference in regular expression compilation (Tom Lane)

- Fix `to_char()` to use ASCII-only case-folding rules where appropriate (Tom Lane)

  This fixes misbehavior of some template patterns that should be locale-independent, but mishandled "`I`" and "`i`" in Turkish locales.

- Fix unwanted rejection of timestamp `1999-12-31 24:00:00` (Tom Lane)

- Fix SQL-language functions to be safely usable as support functions for range types (Tom Lane)

- Fix logic error when a single transaction does `UNLISTEN` then `LISTEN` (Tom Lane)

  The session wound up not listening for notify events at all, though it surely should listen in this case.

- Fix possible planner crash after columns have been added to a view that's depended on by another view (Tom Lane)

- Fix performance issue in `EXPLAIN (ANALYZE, TIMING OFF)` (Pavel Stehule)

- Remove useless "picksplit doesn't support secondary split" log messages (Josh Hansen, Tom Lane)

  This message seems to have been added in expectation of code that was never written, and probably never will be, since GiST's default handling of secondary splits is actually pretty good. So stop nagging end users about it.

- Remove vestigial secondary-split support in `gist_box_picksplit()` (Tom Lane)

  Not only was this implementation of secondary-split not better than the default implementation, it's actually worse. So remove it and let the default code path handle the case.

- Fix possible failure to send a session's last few transaction commit/abort counts to the statistics collector (Tom Lane)

- Eliminate memory leaks in PL/Perl's `spi_prepare()` function (Alex Hunsaker, Tom Lane)

- Fix pg_dumpall to handle database names containing "=" correctly (Heikki Linnakangas)

- Avoid crash in pg_dump when an incorrect connection string is given (Heikki Linnakangas)

- Ignore invalid indexes in pg_dump and pg_upgrade (Michael Paquier, Bruce Momjian)

  Dumping invalid indexes can cause problems at restore time, for example if the reason the index creation failed was because it tried to enforce a uniqueness condition not satisfied by the table's data. Also, if the index creation is in fact still in progress, it seems reasonable to consider it to be an uncommitted DDL change, which pg_dump wouldn't be expected to dump anyway. pg_upgrade now also skips invalid indexes rather than failing.

- In pg_basebackup, include only the current server version's subdirectory when backing up a tablespace (Heikki Linnakangas)

- Add a server version check in pg_basebackup and pg_receivexlog, so they fail cleanly with version combinations that won't work (Heikki Linnakangas)

- Fix `contrib/dblink` to handle inconsistent settings of `DateStyle` or `IntervalStyle` safely (Daniel Farina, Tom Lane)

  Previously, if the remote server had different settings of these parameters, ambiguous dates might be read incorrectly. This fix ensures that datetime and interval columns fetched by a `dblink` query will be interpreted correctly. Note however that inconsistent settings are still risky, since literal values appearing in SQL commands sent to the remote server might be interpreted differently than they would be locally.

- Fix `contrib/pg_trgm`'s `similarity()` function to return zero for trigram-less strings (Tom Lane)

  Previously it returned `NaN` due to internal division by zero.

- Enable building PostgreSQL with Microsoft Visual Studio 2012 (Brar Piening, Noah Misch)

- Update time zone data files to tzdata release 2013b for DST law changes in Chile, Haiti, Morocco, Paraguay, and some Russian areas. Also, historical zone data corrections for numerous places.

  Also, update the time zone abbreviation files for recent changes in Russia and elsewhere: `CHOT`, `GET`, `IRKT`, `KGT`, `KRAT`, `MAGT`, `MAWT`, `MSK`, `NOVT`, `OMST`, `TKT`, `VLAT`, `WST`, `YAKT`, `YEKT` now follow their current meanings, and `VOLT` (Europe/Volgograd) and `MIST` (Antarctica/Macquarie) are added to the default abbreviations list.

E.22. Release 9.2.3

Release date: 2013-02-07

This release contains a variety of fixes from 9.2.2. For information about new features in the 9.2 major release, see 第 E.25 节.

E.22.1. Migration to Version 9.2.3

A dump/restore is not required for those running 9.2.X.

However, if you are upgrading from a version earlier than 9.2.2, see 第 E.23 节.

E.22.2. Changes

- Prevent execution of `enum_recv` from SQL (Tom Lane)

  The function was misdeclared, allowing a simple SQL command to crash the server. In principle an attacker might be able to use it to examine the contents of server memory. Our thanks to Sumit Soni (via Secunia SVCRP) for reporting this issue. (CVE-2013-0255)

- Fix multiple problems in detection of when a consistent database state has been reached during WAL replay (Fujii Masao, Heikki Linnakangas, Simon Riggs, Andres Freund)

- Fix detection of end-of-backup point when no actual redo work is required (Heikki Linnakangas)

  This mistake could result in incorrect "WAL ends before end of online backup" errors.

- Update minimum recovery point when truncating a relation file (Heikki Linnakangas)

  Once data has been discarded, it's no longer safe to stop recovery at an earlier point in the timeline.

- Fix recycling of WAL segments after changing recovery target timeline (Heikki Linnakangas)

- Properly restore timeline history files from archive on cascading standby servers (Heikki Linnakangas)

- Fix lock conflict detection on hot-standby servers (Andres Freund, Robert Haas)
- Fix missing cancellations in hot standby mode (Noah Misch, Simon Riggs)

  The need to cancel conflicting hot-standby queries would sometimes be missed, allowing those queries to see inconsistent data.
- Prevent recovery pause feature from pausing before users can connect (Tom Lane)
- Fix SQL grammar to allow subscripting or field selection from a sub-SELECT result (Tom Lane)
- Fix performance problems with autovacuum truncation in busy workloads (Jan Wieck)

  Truncation of empty pages at the end of a table requires exclusive lock, but autovacuum was coded to fail (and release the table lock) when there are conflicting lock requests. Under load, it is easily possible that truncation would never occur, resulting in table bloat. Fix by performing a partial truncation, releasing the lock, then attempting to re-acquire the lock and continue. This fix also greatly reduces the average time before autovacuum releases the lock after a conflicting request arrives.
- Improve performance of `SPI_execute` and related functions, thereby improving PL/pgSQL's `EXECUTE` (Heikki Linnakangas, Tom Lane)

  Remove some data-copying overhead that was added in 9.2 as a consequence of revisions in the plan caching mechanism. This eliminates a performance regression compared to 9.1, and also saves memory, especially when the query string to be executed contains many SQL statements.

  A side benefit is that multi-statement query strings are now processed fully serially, that is we complete execution of earlier statements before running parse analysis and planning on the following ones. This eliminates a long-standing issue, in that DDL that should affect the behavior of a later statement will now behave as expected.
- Restore pre-9.2 cost estimates for index usage (Tom Lane)

  An ill-considered change of a fudge factor led to undesirably high cost estimates for use of very large indexes.
- Fix intermittent crash in `DROP INDEX CONCURRENTLY` (Tom Lane)
- Fix potential corruption of shared-memory lock table during `CREATE/DROP INDEX CONCURRENTLY` (Tom Lane)
- Fix `COPY`'s multiple-tuple-insertion code for the case of a tuple larger than page size minus fillfactor (Heikki Linnakangas)

  The previous coding could get into an infinite loop.
- Protect against race conditions when scanning `pg_tablespace` (Stephen Frost, Tom Lane)

  `CREATE DATABASE` and `DROP DATABASE` could misbehave if there were concurrent updates of `pg_tablespace` entries.
- Prevent `DROP OWNED` from trying to drop whole databases or tablespaces (Álvaro Herrera)

  For safety, ownership of these objects must be reassigned, not dropped.

- Fix error in `vacuum_freeze_table_age` implementation (Andres Freund)

  In installations that have existed for more than `vacuum_freeze_min_age` transactions, this mistake prevented autovacuum from using partial-table scans, so that a full-table scan would always happen instead.

- Prevent misbehavior when a `RowExpr` or `XmlExpr` is parse-analyzed twice (Andres Freund, Tom Lane)

  This mistake could be user-visible in contexts such as `CREATE TABLE LIKE INCLUDING INDEXES`.

- Improve defenses against integer overflow in hashtable sizing calculations (Jeff Davis)

- Fix some bugs associated with privileges on datatypes (Tom Lane)

  There were some issues with default privileges for types, and pg_dump failed to dump such privileges at all.

- Fix failure to ignore leftover temporary tables after a server crash (Tom Lane)

- Fix failure to rotate postmaster log files for size reasons on Windows (Jeff Janes, Heikki Linnakangas)

- Reject out-of-range dates in `to_date()` (Hitoshi Harada)

- Fix `pg_extension_config_dump()` to handle extension-update cases properly (Tom Lane)

  This function will now replace any existing entry for the target table, making it usable in extension update scripts.

- Fix PL/pgSQL's reporting of plan-time errors in possibly-simple expressions (Tom Lane)

  The previous coding resulted in sometimes omitting the first line in the `CONTEXT` traceback for the error.

- Fix PL/Python's handling of functions used as triggers on multiple tables (Andres Freund)

- Ensure that non-ASCII prompt strings are translated to the correct code page on Windows (Alexander Law, Noah Misch)

  This bug affected psql and some other client programs.

- Fix possible crash in psql's `\?` command when not connected to a database (Meng Qingzhong)

- Fix possible error if a relation file is removed while pg_basebackup is running (Heikki Linnakangas)

- Tolerate timeline switches while `pg_basebackup -X fetch` is backing up a standby server (Heikki Linnakangas)

- Make pg_dump exclude data of unlogged tables when running on a hot-standby server (Magnus Hagander)

  This would fail anyway because the data is not available on the standby server, so it seems most convenient to assume `--no-unlogged-table-data` automatically.

- Fix pg_upgrade to deal with invalid indexes safely (Bruce Momjian)
- Fix pg_upgrade's -O/-o options (Marti Raudsepp)
- Fix one-byte buffer overrun in libpq's `PQprintTuples` (Xi Wang)

  This ancient function is not used anywhere by PostgreSQL itself, but it might still be used by some client code.
- Make ecpglib use translated messages properly (Chen Huajun)
- Properly install ecpg_compat and pgtypes libraries on MSVC (Jiang Guiqing)
- Include our version of `isinf()` in libecpg if it's not provided by the system (Jiang Guiqing)
- Rearrange configure's tests for supplied functions so it is not fooled by bogus exports from libedit/libreadline (Christoph Berg)
- Ensure Windows build number increases over time (Magnus Hagander)
- Make pgxs build executables with the right `.exe` suffix when cross-compiling for Windows (Zoltan Boszormenyi)
- Add new timezone abbreviation `FET` (Tom Lane)

  This is now used in some eastern-European time zones.

E.23. Release 9.2.2

Release date: 2012-12-06

This release contains a variety of fixes from 9.2.1. For information about new features in the 9.2 major release, see 第 E.25 节.

E.23.1. Migration to Version 9.2.2

A dump/restore is not required for those running 9.2.X.

However, you may need to perform `REINDEX` operations to correct problems in concurrently-built indexes, as described in the first changelog item below.

Also, if you are upgrading from version 9.2.0, see 第 E.24 节.

E.23.2. Changes

- Fix multiple bugs associated with `CREATE/DROP INDEX CONCURRENTLY` (Andres Freund, Tom Lane, Simon Riggs, Pavan Deolasee)

  An error introduced while adding `DROP INDEX CONCURRENTLY` allowed incorrect indexing decisions to be made during the initial phase of `CREATE INDEX CONCURRENTLY`; so that indexes built by that command could be corrupt. It is recommended that indexes built in 9.2.X with `CREATE INDEX CONCURRENTLY` be rebuilt after applying this update.

  In addition, fix `CREATE/DROP INDEX CONCURRENTLY` to use in-place updates when changing the state of an index's `pg_index` row. This prevents race conditions that could cause concurrent sessions to miss updating the target index, thus again resulting in corrupt concurrently-created indexes.

Also, fix various other operations to ensure that they ignore invalid indexes resulting from a failed `CREATE INDEX CONCURRENTLY` command. The most important of these is `VACUUM`, because an auto-vacuum could easily be launched on the table before corrective action can be taken to fix or remove the invalid index.

Also fix `DROP INDEX CONCURRENTLY` to not disable insertions into the target index until all queries using it are done.

Also fix misbehavior if `DROP INDEX CONCURRENTLY` is canceled: the previous coding could leave an un-droppable index behind.

- Correct predicate locking for `DROP INDEX CONCURRENTLY` (Kevin Grittner)

  Previously, SSI predicate locks were processed at the wrong time, possibly leading to incorrect behavior of serializable transactions executing in parallel with the `DROP`.

- Fix buffer locking during WAL replay (Tom Lane)

  The WAL replay code was insufficiently careful about locking buffers when replaying WAL records that affect more than one page. This could result in hot standby queries transiently seeing inconsistent states, resulting in wrong answers or unexpected failures.

- Fix an error in WAL generation logic for GIN indexes (Tom Lane)

  This could result in index corruption, if a torn-page failure occurred.

- Fix an error in WAL replay logic for SP-GiST indexes (Tom Lane)

  This could result in index corruption after a crash, or on a standby server.

- Fix incorrect detection of end-of-base-backup location during WAL recovery (Heikki Linnakangas)

  This mistake allowed hot standby mode to start up before the database reaches a consistent state.

- Properly remove startup process's virtual XID lock when promoting a hot standby server to normal running (Simon Riggs)

  This oversight could prevent subsequent execution of certain operations such as `CREATE INDEX CONCURRENTLY`.

- Avoid bogus "out-of-sequence timeline ID" errors in standby mode (Heikki Linnakangas)

- Prevent the postmaster from launching new child processes after it's received a shutdown signal (Tom Lane)

  This mistake could result in shutdown taking longer than it should, or even never completing at all without additional user action.

- Fix the syslogger process to not fail when `log_rotation_age` exceeds 2^31 milliseconds (about 25 days) (Tom Lane)

- Fix `WaitLatch()` to return promptly when the requested timeout expires (Jeff Janes, Tom Lane)

  With the previous coding, a steady stream of non-wait-terminating interrupts could delay return from `WaitLatch()` indefinitely. This has been shown to be a problem for the autovacuum launcher process, and might cause trouble elsewhere as well.

- Avoid corruption of internal hash tables when out of memory (Hitoshi Harada)
- Prevent file descriptors for dropped tables from being held open past transaction end (Tom Lane)

  This should reduce problems with long-since-dropped tables continuing to occupy disk space.
- Prevent database-wide crash and restart when a new child process is unable to create a pipe for its latch (Tom Lane)

  Although the new process must fail, there is no good reason to force a database-wide restart, so avoid that. This improves robustness when the kernel is nearly out of file descriptors.
- Avoid planner crash with joins to unflattened subqueries (Tom Lane)
- Fix planning of non-strict equivalence clauses above outer joins (Tom Lane)

  The planner could derive incorrect constraints from a clause equating a non-strict construct to something else, for example `WHERE COALESCE(foo, 0) = 0` when `foo` is coming from the nullable side of an outer join. 9.2 showed this type of error in more cases than previous releases, but the basic bug has been there for a long time.
- Fix `SELECT DISTINCT` with index-optimized `MIN`/`MAX` on an inheritance tree (Tom Lane)

  The planner would fail with "failed to re-find MinMaxAggInfo record" given this combination of factors.
- Make sure the planner sees implicit and explicit casts as equivalent for all purposes, except in the minority of cases where there's actually a semantic difference (Tom Lane)
- Include join clauses when considering whether partial indexes can be used for a query (Tom Lane)

  A strict join clause can be sufficient to establish an *x* `IS NOT NULL` predicate, for example. This fixes a planner regression in 9.2, since previous versions could make comparable deductions.
- Limit growth of planning time when there are many indexable join clauses for the same index (Tom Lane)
- Improve planner's ability to prove exclusion constraints from equivalence classes (Tom Lane)
- Fix partial-row matching in hashed subplans to handle cross-type cases correctly (Tom Lane)

  This affects multicolumn `NOT IN` subplans, such as `WHERE (a, b) NOT IN (SELECT x, y FROM ...)` when for instance `b` and `y` are `int4` and `int8` respectively. This mistake led to wrong answers or crashes depending on the specific datatypes involved.
- Fix btree mark/restore functions to handle array keys (Tom Lane)

  This oversight could result in wrong answers from merge joins whose inner side is an index scan using an *indexed_column* `= ANY(`*array*`)` condition.
- Revert patch for taking fewer snapshots (Tom Lane)

The 9.2 change to reduce the number of snapshots taken during query execution led to some anomalous behaviors not seen in previous releases, because execution would proceed with a snapshot acquired before locking the tables used by the query. Thus, for example, a query would not be guaranteed to see updates committed by a preceding transaction even if that transaction had exclusive lock. We'll probably revisit this in future releases, but meanwhile put it back the way it was before 9.2.

- Acquire buffer lock when re-fetching the old tuple for an `AFTER ROW UPDATE/DELETE` trigger (Andres Freund)

  In very unusual circumstances, this oversight could result in passing incorrect data to a trigger `WHEN` condition, or to the precheck logic for a foreign-key enforcement trigger. That could result in a crash, or in an incorrect decision about whether to fire the trigger.

- Fix `ALTER COLUMN TYPE` to handle inherited check constraints properly (Pavan Deolasee)

  This worked correctly in pre-8.4 releases, and now works correctly in 8.4 and later.

- Fix `ALTER EXTENSION SET SCHEMA`'s failure to move some subsidiary objects into the new schema (Álvaro Herrera, Dimitri Fontaine)

- Handle `CREATE TABLE AS EXECUTE` correctly in extended query protocol (Tom Lane)

- Don't modify the input parse tree in `DROP RULE IF NOT EXISTS` and `DROP TRIGGER IF NOT EXISTS` (Tom Lane)

  This mistake would cause errors if a cached statement of one of these types was re-executed.

- Fix `REASSIGN OWNED` to handle grants on tablespaces (Álvaro Herrera)

- Ignore incorrect `pg_attribute` entries for system columns for views (Tom Lane)

  Views do not have any system columns. However, we forgot to remove such entries when converting a table to a view. That's fixed properly for 9.3 and later, but in previous branches we need to defend against existing mis-converted views.

- Fix rule printing to dump `INSERT INTO` *`table`* `DEFAULT VALUES` correctly (Tom Lane)

- Guard against stack overflow when there are too many `UNION`/`INTERSECT`/`EXCEPT` clauses in a query (Tom Lane)

- Prevent platform-dependent failures when dividing the minimum possible integer value by -1 (Xi Wang, Tom Lane)

- Fix possible access past end of string in date parsing (Hitoshi Harada)

- Fix failure to advance XID epoch if XID wraparound happens during a checkpoint and `wal_level` is `hot_standby` (Tom Lane, Andres Freund)

  While this mistake had no particular impact on PostgreSQL itself, it was bad for applications that rely on `txid_current()` and related functions: the TXID value would appear to go backwards.

- Fix `pg_terminate_backend()` and `pg_cancel_backend()` to not throw error for a non-existent target process (Josh Kupershmidt)

This case already worked as intended when called by a superuser, but not so much when called by ordinary users.

- Fix display of `pg_stat_replication.sync_state` at a page boundary (Kyotaro Horiguchi)
- Produce an understandable error message if the length of the path name for a Unix-domain socket exceeds the platform-specific limit (Tom Lane, Andrew Dunstan)

  Formerly, this would result in something quite unhelpful, such as “Non-recoverable failure in name resolution” .
- Fix memory leaks when sending composite column values to the client (Tom Lane)
- Save some cycles by not searching for subtransaction locks at commit (Simon Riggs)

  In a transaction holding many exclusive locks, this useless activity could be quite costly.
- Make pg_ctl more robust about reading the `postmaster.pid` file (Heikki Linnakangas)

  This fixes race conditions and possible file descriptor leakage.
- Fix possible crash in psql if incorrectly-encoded data is presented and the `client_encoding` setting is a client-only encoding, such as SJIS (Jiang Guiqing)
- Make pg_dump dump `SEQUENCE SET` items in the data not pre-data section of the archive (Tom Lane)

  This fixes an undesirable inconsistency between the meanings of `--data-only` and `--section=data`, and also fixes dumping of sequences that are marked as extension configuration tables.
- Fix pg_dump's handling of `DROP DATABASE` commands in `--clean` mode (Guillaume Lelarge)

  Beginning in 9.2.0, `pg_dump --clean` would issue a `DROP DATABASE` command, which was either useless or dangerous depending on the usage scenario. It no longer does that. This change also fixes the combination of `--clean` and `--create` to work sensibly, i.e., emit `DROP DATABASE` then `CREATE DATABASE` before reconnecting to the target database.
- Fix pg_dump for views with circular dependencies and no relation options (Tom Lane)

  The previous fix to dump relation options when a view is involved in a circular dependency didn't work right for the case that the view has no options; it emitted `ALTER VIEW foo SET ()` which is invalid syntax.
- Fix bugs in the `restore.sql` script emitted by pg_dump in `tar` output format (Tom Lane)

  The script would fail outright on tables whose names include upper-case characters. Also, make the script capable of restoring data in `--inserts` mode as well as the regular COPY mode.
- Fix pg_restore to accept POSIX-conformant `tar` files (Brian Weaver, Tom Lane)

  The original coding of pg_dump's `tar` output mode produced files that are not fully conformant with the POSIX standard. This has been corrected for version 9.3. This

patch updates previous branches so that they will accept both the incorrect and the corrected formats, in hopes of avoiding compatibility problems when 9.3 comes out.

- Fix `tar` files emitted by pg_basebackup to be POSIX conformant (Brian Weaver, Tom Lane)
- Fix pg_resetxlog to locate `postmaster.pid` correctly when given a relative path to the data directory (Tom Lane)

  This mistake could lead to pg_resetxlog not noticing that there is an active postmaster using the data directory.
- Fix libpq's `lo_import()` and `lo_export()` functions to report file I/O errors properly (Tom Lane)
- Fix ecpg's processing of nested structure pointer variables (Muhammad Usama)
- Fix ecpg's `ecpg_get_data` function to handle arrays properly (Michael Meskes)
- Prevent pg_upgrade from trying to process TOAST tables for system catalogs (Bruce Momjian)

  This fixes an error seen when the `information_schema` has been dropped and recreated. Other failures were also possible.
- Improve pg_upgrade performance by setting `synchronous_commit` to `off` in the new cluster (Bruce Momjian)
- Make `contrib/pageinspect`'s btree page inspection functions take buffer locks while examining pages (Tom Lane)
- Work around unportable behavior of `malloc(0)` and `realloc(NULL, 0)` (Tom Lane)

  On platforms where these calls return `NULL`, some code mistakenly thought that meant out-of-memory. This is known to have broken pg_dump for databases containing no user-defined aggregates. There might be other cases as well.
- Ensure that `make install` for an extension creates the `extension` installation directory (Cédric Villemain)

  Previously, this step was missed if `MODULEDIR` was set in the extension's Makefile.
- Fix pgxs support for building loadable modules on AIX (Tom Lane)

  Building modules outside the original source tree didn't work on AIX.
- Update time zone data files to tzdata release 2012j for DST law changes in Cuba, Israel, Jordan, Libya, Palestine, Western Samoa, and portions of Brazil.

E.24. Release 9.2.1

Release date: 2012-09-24

This release contains a variety of fixes from 9.2.0. For information about new features in the 9.2 major release, see 第 E.25 节.

E.24.1. Migration to Version 9.2.1

A dump/restore is not required for those running 9.2.X.

However, you may need to perform `REINDEX` and/or `VACUUM` operations to recover from the effects of the data corruption bug described in the first changelog item below.

E.24.2. Changes

- Fix persistence marking of shared buffers during WAL replay (Jeff Davis)

  This mistake can result in buffers not being written out during checkpoints, resulting in data corruption if the server later crashes without ever having written those buffers. Corruption can occur on any server following crash recovery, but it is significantly more likely to occur on standby slave servers since those perform much more WAL replay. There is a low probability of corruption of btree and GIN indexes. There is a much higher probability of corruption of table "visibility maps", which might lead to wrong answers from index-only scans. Table data proper cannot be corrupted by this bug.

  While no index corruption due to this bug is known to have occurred in the field, as a precautionary measure it is recommended that production installations `REINDEX` all btree and GIN indexes at a convenient time after upgrading to 9.2.1.

  Also, it is recommended to perform a `VACUUM` of all tables while having `vacuum_freeze_table_age` set to zero. This will fix any incorrect visibility map data. `vacuum_cost_delay` can be adjusted to reduce the performance impact of vacuuming, while causing it to take longer to finish.

- Fix possible incorrect sorting of output from queries involving `WHERE` *`indexed_column`* `IN (`*`list_of_values`*`)` (Tom Lane)

- Fix planner failure for queries involving `GROUP BY` expressions along with window functions and aggregates (Tom Lane)

- Fix planner's assignment of executor parameters (Tom Lane)

  This error could result in wrong answers from queries that scan the same `WITH` subquery multiple times.

- Improve planner's handling of join conditions in index scans (Tom Lane)

- Improve selectivity estimation for text search queries involving prefixes, i.e. *`word`*`:*` patterns (Tom Lane)

- Fix delayed recognition of permissions changes (Tom Lane)

  A command that needed no locks other than ones its transaction already had might fail to notice a concurrent `GRANT` or `REVOKE` that committed since the start of its transaction.

- Fix `ANALYZE` to not fail when a column is a domain over an array type (Tom Lane)

- Prevent PL/Perl from crashing if a recursive PL/Perl function is redefined while being executed (Tom Lane)

- Work around possible misoptimization in PL/Perl (Tom Lane)

  Some Linux distributions contain an incorrect version of `pthread.h` that results in incorrect compiled code in PL/Perl, leading to crashes if a PL/Perl function calls another one that throws an error.

- Remove unnecessary dependency on pg_config from pg_upgrade (Peter Eisentraut)

- Update time zone data files to tzdata release 2012f for DST law changes in Fiji

E.25. Release 9.2

Release date: 2012-09-10

E.25.1. Overview

This release has been largely focused on performance improvements, though new SQL features are not lacking. Work also continues in the area of replication support. Major enhancements include:

- Allow queries to retrieve data only from indexes, avoiding heap access (index-only scans)
- Allow the planner to generate custom plans for specific parameter values even when using prepared statements
- Improve the planner's ability to use nested loops with inner index scans
- Allow streaming replication slaves to forward data to other slaves (cascading replication)
- Allow pg_basebackup to make base backups from standby servers
- Add a pg_receivexlog tool to archive WAL file changes as they are written
- Add the SP-GiST (Space-Partitioned GiST) index access method
- Add support for range data types
- Add a `JSON` data type
- Add a `security_barrier` option for views
- Allow libpq connection strings to have the format of a URI
- Add a single-row processing mode to libpq for better handling of large result sets

The above items are explained in more detail in the sections below.

E.25.2. Migration to Version 9.2

A dump/restore using pg_dump, or use of pg_upgrade, is required for those wishing to migrate data from any previous release.

Version 9.2 contains a number of changes that may affect compatibility with previous releases. Observe the following incompatibilities:

E.25.2.1. System Catalogs

- Remove the `spclocation` field from `pg_tablespace` (Magnus Hagander)

  This field was duplicative of the symbolic links that actually define tablespace locations, and thus risked errors of omission when moving a tablespace. This change allows tablespace directories to be moved while the server is down, by manually adjusting the symbolic links. To replace this field, we have added `pg_tablespace_location()` to allow querying of the symbolic links.

- Move `tsvector` most-common-element statistics to new `pg_stats` columns (Alexander Korotkov)

Consult `most_common_elems` and `most_common_elem_freqs` for the data formerly available in `most_common_vals` and `most_common_freqs` for a `tsvector` column.

E.25.2.2. Functions

- Remove hstore's => operator (Robert Haas)

  Users should now use `hstore(text, text)`. Since PostgreSQL 9.0, a warning message has been emitted when an operator named => is created because the SQL standard reserves that token for another use.

- Ensure that `xpath()` escapes special characters in string values (Florian Pflug)

  Without this it is possible for the result not to be valid XML.

- Make `pg_relation_size()` and friends return NULL if the object does not exist (Phil Sorber)

  This prevents queries that call these functions from returning errors immediately after a concurrent `DROP`.

- Make `EXTRACT(EPOCH FROM timestamp without time zone)` measure the epoch from local midnight, not UTC midnight (Tom Lane)

  This change reverts an ill-considered change made in release 7.3. Measuring from UTC midnight was inconsistent because it made the result dependent on the `timezone` setting, which computations for `timestamp without time zone` should not be. The previous behavior remains available by casting the input value to `timestamp with time zone`.

- Properly parse time strings with trailing `yesterday`, `today`, and `tomorrow` (Dean Rasheed)

  Previously, `SELECT '04:00:00 yesterday'::timestamp` returned yesterday's date at midnight.

- Fix `to_date()` and `to_timestamp()` to wrap incomplete dates toward 2020 (Bruce Momjian)

  Previously, supplied years and year masks of less than four digits wrapped inconsistently.

E.25.2.3. Object Modification

- Prevent `ALTER DOMAIN` from working on non-domain types (Peter Eisentraut)

  Owner and schema changes were previously possible on non-domain types.

- No longer forcibly lowercase procedural language names in `CREATE FUNCTION` (Robert Haas)

  While unquoted language identifiers are still lowercased, strings and quoted identifiers are no longer forcibly down-cased. Thus for example `CREATE FUNCTION ... LANGUAGE 'C'` will no longer work; it must be spelled `'c'`, or better omit the quotes.

- Change system-generated names of foreign key enforcement triggers (Tom Lane)

  This change ensures that the triggers fire in the correct order in some corner cases involving self-referential foreign key constraints.

E.25.2.4. Command-Line Tools

- Provide consistent backquote, variable expansion, and quoted substring behavior in psql meta-command arguments (Tom Lane)

  Previously, such references were treated oddly when not separated by whitespace from adjacent text. For example `'FOO'BAR` was output as `FOO BAR` (unexpected insertion of a space) and `FOO'BAR'BAZ` was output unchanged (not removing the quotes as most would expect).

- No longer treat clusterdb table names as double-quoted; no longer treat reindexdb table and index names as double-quoted (Bruce Momjian)

  Users must now include double-quotes in the command arguments if quoting is wanted.

- createuser no longer prompts for option settings by default (Peter Eisentraut)

  Use `--interactive` to obtain the old behavior.

- Disable prompting for the user name in dropuser unless `--interactive` is specified (Peter Eisentraut)

E.25.2.5. Server Settings

- Add server parameters for specifying the locations of server-side SSL files (Peter Eisentraut)

  This allows changing the names and locations of the files that were previously hard-coded as `server.crt`, `server.key`, `root.crt`, and `root.crl` in the data directory. The server will no longer examine *`root.crt`* or *`root.crl`* by default; to load these files, the associated parameters must be set to non-default values.

- Remove the `silent_mode` parameter (Heikki Linnakangas)

  Similar behavior can be obtained with `pg_ctl start -l postmaster.log`.

- Remove the `wal_sender_delay` parameter, as it is no longer needed (Tom Lane)

- Remove the `custom_variable_classes` parameter (Tom Lane)

  The checking provided by this setting was dubious. Now any setting can be prefixed by any class name.

E.25.2.6. Monitoring

- Rename `pg_stat_activity.procpid` to `pid`, to match other system tables (Magnus Hagander)

- Create a separate `pg_stat_activity` column to report process state (Scott Mead, Magnus Hagander)

  The previous `query` and `query_start` values now remain available for an idle session, allowing enhanced analysis.

- Rename `pg_stat_activity.current_query` to `query` because it is not cleared when the query completes (Magnus Hagander)

- Change all SQL-level statistics timing values to be `float8` columns measured in milliseconds (Tom Lane)

This change eliminates the designed-in assumption that the values are accurate to microseconds and no more (since the `float8` values can be fractional). The columns affected are `pg_stat_user_functions.total_time`, `pg_stat_user_functions.self_time`, `pg_stat_xact_user_functions.total_time`, and `pg_stat_xact_user_functions.self_time`. The statistics functions underlying these columns now also return `float8` milliseconds, rather than `bigint` microseconds. `contrib/pg_stat_statements'` `total_time` column is now also measured in milliseconds.

E.25.3. Changes

Below you will find a detailed account of the changes between PostgreSQL 9.2 and the previous major release.

E.25.3.1. Server

E.25.3.1.1. Performance

- Allow queries to retrieve data only from indexes, avoiding heap access (Robert Haas, Ibrar Ahmed, Heikki Linnakangas, Tom Lane)

  This feature is often called index-only scans. Heap access can be skipped for heap pages containing only tuples that are visible to all sessions, as reported by the visibility map; so the benefit applies mainly to mostly-static data. The visibility map was made crash-safe as a necessary part of implementing this feature.

- Add the SP-GiST (Space-Partitioned GiST) index access method (Teodor Sigaev, Oleg Bartunov, Tom Lane)

  SP-GiST is comparable to GiST in flexibility, but supports unbalanced partitioned search structures rather than balanced trees. For suitable problems, SP-GiST can be faster than GiST in both index build time and search time.

- Allow group commit to work effectively under heavy load (Peter Geoghegan, Simon Riggs, Heikki Linnakangas)

  Previously, batching of commits became ineffective as the write workload increased, because of internal lock contention.

- Allow uncontended locks to be managed using a new fast-path lock mechanism (Robert Haas)

- Reduce overhead of creating virtual transaction ID locks (Robert Haas)

- Reduce the overhead of serializable isolation level locks (Dan Ports)

- Improve PowerPC and Itanium spinlock performance (Manabu Ori, Robert Haas, Tom Lane)

- Reduce overhead for shared invalidation cache messages (Robert Haas)

- Move the frequently accessed members of the `PGPROC` shared memory array to a separate array (Pavan Deolasee, Heikki Linnakangas, Robert Haas)

- Improve `COPY` performance by adding tuples to the heap in batches (Heikki Linnakangas)

- Improve GiST index performance for geometric data types by producing better trees with less memory allocation overhead (Alexander Korotkov)

- Improve GiST index build times (Alexander Korotkov, Heikki Linnakangas)
- Allow hint bits to be set sooner for temporary and unlogged tables (Robert Haas)
- Allow sorting to be performed by inlined, non-SQL-callable comparison functions (Peter Geoghegan, Robert Haas, Tom Lane)
- Make the number of CLOG buffers scale based on `shared_buffers` (Robert Haas, Simon Riggs, Tom Lane)
- Improve performance of buffer pool scans that occur when tables or databases are dropped (Jeff Janes, Simon Riggs)
- Improve performance of checkpointer's fsync-request queue when many tables are being dropped or truncated (Tom Lane)
- Pass the safe number of file descriptors to child processes on Windows (Heikki Linnakangas)

  This allows Windows sessions to use more open file descriptors than before.

E.25.3.1.2. Process Management

- Create a dedicated background process to perform checkpoints (Simon Riggs)

  Formerly the background writer did both dirty-page writing and checkpointing. Separating this into two processes allows each goal to be accomplished more predictably.

- Improve asynchronous commit behavior by waking the walwriter sooner (Simon Riggs)

  Previously, only `wal_writer_delay` triggered WAL flushing to disk; now filling a WAL buffer also triggers WAL writes.

- Allow the bgwriter, walwriter, checkpointer, statistics collector, log collector, and archiver background processes to sleep more efficiently during periods of inactivity (Peter Geoghegan, Tom Lane)

  This series of changes reduces the frequency of process wake-ups when there is nothing to do, dramatically reducing power consumption on idle servers.

E.25.3.1.3. Optimizer

- Allow the planner to generate custom plans for specific parameter values even when using prepared statements (Tom Lane)

  In the past, a prepared statement always had a single “generic” plan that was used for all parameter values, which was frequently much inferior to the plans used for non-prepared statements containing explicit constant values. Now, the planner attempts to generate custom plans for specific parameter values. A generic plan will only be used after custom plans have repeatedly proven to provide no benefit. This change should eliminate the performance penalties formerly seen from use of prepared statements (including non-dynamic statements in PL/pgSQL).

- Improve the planner's ability to use nested loops with inner index scans (Tom Lane)

  The new “parameterized path” mechanism allows inner index scans to use values from relations that are more than one join level up from the scan. This can greatly improve performance in situations where semantic restrictions (such as outer joins) limit the allowed join orderings.

- Improve the planning API for foreign data wrappers (Etsuro Fujita, Shigeru Hanada, Tom Lane)

  Wrappers can now provide multiple access “paths” for their tables, allowing more flexibility in join planning.

- Recognize self-contradictory restriction clauses for non-table relations (Tom Lane)

  This check is only performed when `constraint_exclusion` is `on`.

- Allow `indexed_col op ANY(ARRAY[...])` conditions to be used in plain index scans and index-only scans (Tom Lane)

  Formerly such conditions could only be used in bitmap index scans.

- Support `MIN`/`MAX` index optimizations on `boolean` columns (Marti Raudsepp)

- Account for set-returning functions in `SELECT` target lists when setting row count estimates (Tom Lane)

- Fix planner to handle indexes with duplicated columns more reliably (Tom Lane)

- Collect and use element-frequency statistics for arrays (Alexander Korotkov, Tom Lane)

  This change improves selectivity estimation for the array `<@`, `&&`, and `@>` operators (array containment and overlaps).

- Allow statistics to be collected for foreign tables (Etsuro Fujita)

- Improve cost estimates for use of partial indexes (Tom Lane)

- Improve the planner's ability to use statistics for columns referenced in subqueries (Tom Lane)

- Improve statistical estimates for subqueries using `DISTINCT` (Tom Lane)

E.25.3.1.4. Authentication

- Do not treat role names and `samerole` specified in `pg_hba.conf` as automatically including superusers (Andrew Dunstan)

  This makes it easier to use `reject` lines with group roles.

- Adjust `pg_hba.conf` processing to handle token parsing more consistently (Brendan Jurd, Álvaro Herrera)

- Disallow empty `pg_hba.conf` files (Tom Lane)

  This was done to more quickly detect misconfiguration.

- Make superuser privilege imply replication privilege (Noah Misch)

  This avoids the need to explicitly assign such privileges.

E.25.3.1.5. Monitoring

- Attempt to log the current query string during a backend crash (Marti Raudsepp)

- Make logging of autovacuum I/O activity more verbose (Greg Smith, Noah Misch)

  This logging is triggered by `log_autovacuum_min_duration`.

- Make WAL replay report failures sooner (Fujii Masao)

  There were some cases where failures were only reported once the server went into master mode.

- Add `pg_xlog_location_diff()` to simplify WAL location comparisons (Euler Taveira de Oliveira)

  This is useful for computing replication lag.

- Support configurable event log application names on Windows (MauMau, Magnus Hagander)

  This allows different instances to use the event log with different identifiers, by setting the `event_source` server parameter, which is similar to how `syslog_ident` works.

- Change "unexpected EOF" messages to `DEBUG1` level, except when there is an open transaction (Magnus Hagander)

  This change reduces log chatter caused by applications that close database connections ungracefully.

E.25.3.1.6. Statistical Views

- Track temporary file sizes and file counts in the `pg_stat_database` system view (Tomas Vondra)
- Add a deadlock counter to the `pg_stat_database` system view (Magnus Hagander)
- Add a server parameter `track_io_timing` to track I/O timings (Ants Aasma, Robert Haas)
- Report checkpoint timing information in `pg_stat_bgwriter` (Greg Smith, Peter Geoghegan)

E.25.3.1.7. Server Settings

- Silently ignore nonexistent schemas specified in `search_path` (Tom Lane)

  This makes it more convenient to use generic path settings, which might include some schemas that don't exist in all databases.

- Allow superusers to set `deadlock_timeout` per-session, not just per-cluster (Noah Misch)

  This allows `deadlock_timeout` to be reduced for transactions that are likely to be involved in a deadlock, thus detecting the failure more quickly. Alternatively, increasing the value can be used to reduce the chances of a session being chosen for cancellation due to a deadlock.

- Add a server parameter `temp_file_limit` to constrain temporary file space usage per session (Mark Kirkwood)

- Allow a superuser to `SET` an extension's superuser-only custom variable before loading the associated extension (Tom Lane)

  The system now remembers whether a `SET` was performed by a superuser, so that proper privilege checking can be done when the extension is loaded.

- Add postmaster `-C` option to query configuration parameters (Bruce Momjian)

This allows pg_ctl to better handle cases where `PGDATA` or `-D` points to a configuration-only directory.

- Replace an empty locale name with the implied value in `CREATE DATABASE` (Tom Lane)

  This prevents cases where `pg_database.datcollate` or `datctype` could be interpreted differently after a server restart.

E.25.3.1.7.1. `postgresql.conf`

- Allow multiple errors in `postgresql.conf` to be reported, rather than just the first one (Alexey Klyukin, Tom Lane)
- Allow a reload of `postgresql.conf` to be processed by all sessions, even if there are some settings that are invalid for particular sessions (Alexey Klyukin)

  Previously, such not-valid-within-session values would cause all setting changes to be ignored by that session.

- Add an `include_if_exists` facility for configuration files (Greg Smith)

  This works the same as `include`, except that an error is not thrown if the file is missing.

- Identify the server time zone during initdb, and set `postgresql.conf` entries `timezone` and `log_timezone` accordingly (Tom Lane)

  This avoids expensive time zone probes during server start.

- Fix `pg_settings` to report `postgresql.conf` line numbers on Windows (Tom Lane)

E.25.3.2. Replication and Recovery

- Allow streaming replication slaves to forward data to other slaves (cascading replication) (Fujii Masao)

  Previously, only the master server could supply streaming replication log files to standby servers.

- Add new `synchronous_commit` mode `remote_write` (Fujii Masao, Simon Riggs)

  This mode waits for the standby server to write transaction data to its own operating system, but does not wait for the data to be flushed to the standby's disk.

- Add a pg_receivexlog tool to archive WAL file changes as they are written, rather than waiting for completed WAL files (Magnus Hagander)
- Allow pg_basebackup to make base backups from standby servers (Jun Ishizuka, Fujii Masao)

  This feature lets the work of making new base backups be off-loaded from the primary server.

- Allow streaming of WAL files while pg_basebackup is performing a backup (Magnus Hagander)

  This allows passing of WAL files to the standby before they are discarded on the primary.

E.25.3.3. Queries

- Cancel the running query if the client gets disconnected (Florian Pflug)

  If the backend detects loss of client connection during a query, it will now cancel the query rather than attempting to finish it.

- Retain column names at run time for row expressions (Andrew Dunstan, Tom Lane)

  This change allows better results when a row value is converted to `hstore` or `json` type: the fields of the resulting value will now have the expected names.

- Improve column labels used for sub-`SELECT` results (Marti Raudsepp)

  Previously, the generic label `?column?` was used.

- Improve heuristics for determining the types of unknown values (Tom Lane)

  The longstanding rule that an unknown constant might have the same type as the value on the other side of the operator using it is now applied when considering polymorphic operators, not only for simple operator matches.

- Warn about creating casts to or from domain types (Robert Haas)

  Such casts have no effect.

- When a row fails a `CHECK` or `NOT NULL` constraint, show the row's contents as error detail (Jan Kundrát)

  This should make it easier to identify which row is problematic when an insert or update is processing many rows.

E.25.3.4. Object Manipulation

- Provide more reliable operation during concurrent DDL (Robert Haas, Noah Misch)

  This change adds locking that should eliminate "cache lookup failed" errors in many scenarios. Also, it is no longer possible to add relations to a schema that is being concurrently dropped, a scenario that formerly led to inconsistent system catalog contents.

- Add `CONCURRENTLY` option to `DROP INDEX` (Simon Riggs)

  This allows index removal without blocking other sessions.

- Allow foreign data wrappers to have per-column options (Shigeru Hanada)

- Improve pretty-printing of view definitions (Andrew Dunstan)

E.25.3.4.1. Constraints

- Allow `CHECK` constraints to be declared `NOT VALID` (Álvaro Herrera)

  Adding a `NOT VALID` constraint does not cause the table to be scanned to verify that existing rows meet the constraint. Subsequently, newly added or updated rows are checked. Such constraints are ignored by the planner when considering `constraint_exclusion`, since it is not certain that all rows meet the constraint.

  The new `ALTER TABLE VALIDATE` command allows `NOT VALID` constraints to be checked for existing rows, after which they are converted into ordinary constraints.

- Allow `CHECK` constraints to be declared `NO INHERIT` (Nikhil Sontakke, Alex Hunsaker, Álvaro Herrera)

  This makes them enforceable only on the parent table, not on child tables.

- Add the ability to rename constraints (Peter Eisentraut)

E.25.3.4.2. `ALTER`

- Reduce need to rebuild tables and indexes for certain `ALTER TABLE` ... `ALTER COLUMN TYPE` operations (Noah Misch)

  Increasing the length limit for a `varchar` or `varbit` column, or removing the limit altogether, no longer requires a table rewrite. Similarly, increasing the allowable precision of a `numeric` column, or changing a column from constrained `numeric` to unconstrained `numeric`, no longer requires a table rewrite. Table rewrites are also avoided in similar cases involving the `interval`, `timestamp`, and `timestamptz` types.

- Avoid having `ALTER TABLE` revalidate foreign key constraints in some cases where it is not necessary (Noah Misch)

- Add `IF EXISTS` options to some `ALTER` commands (Pavel Stehule)

  For example, `ALTER FOREIGN TABLE IF EXISTS foo RENAME TO bar`.

- Add `ALTER FOREIGN DATA WRAPPER` ... `RENAME` and `ALTER SERVER` ... `RENAME` (Peter Eisentraut)

- Add `ALTER DOMAIN` ... `RENAME` (Peter Eisentraut)

  You could already rename domains using `ALTER TYPE`.

- Throw an error for `ALTER DOMAIN` ... `DROP CONSTRAINT` on a nonexistent constraint (Peter Eisentraut)

  An `IF EXISTS` option has been added to provide the previous behavior.

E.25.3.4.3. `CREATE TABLE`

- Allow `CREATE TABLE (LIKE ...)` from foreign tables, views, and composite types (Peter Eisentraut)

  For example, this allows a table to be created whose schema matches a view.

- Fix `CREATE TABLE (LIKE ...)` to avoid index name conflicts when copying index comments (Tom Lane)

- Fix `CREATE TABLE` ... `AS EXECUTE` to handle `WITH NO DATA` and column name specifications (Tom Lane)

E.25.3.4.4. Object Permissions

- Add a `security_barrier` option for views (KaiGai Kohei, Robert Haas)

  This option prevents optimizations that might allow view-protected data to be exposed to users, for example pushing a clause involving an insecure function into the `WHERE` clause of the view. Such views can be expected to perform more poorly than ordinary views.

- Add a new `LEAKPROOF` function attribute to mark functions that can safely be pushed down into `security_barrier` views (KaiGai Kohei)

- Add support for privileges on data types (Peter Eisentraut)

  This adds support for the SQL-conforming `USAGE` privilege on types and domains. The intent is to be able to restrict which users can create dependencies on types, since such dependencies limit the owner's ability to alter the type.

- Check for `INSERT` privileges in `SELECT INTO` / `CREATE TABLE AS` (KaiGai Kohei)

  Because the object is being created by `SELECT INTO` or `CREATE TABLE AS`, the creator would ordinarily have insert permissions; but there are corner cases where this is not true, such as when `ALTER DEFAULT PRIVILEGES` has removed such permissions.

E.25.3.5. Utility Operations

- Allow `VACUUM` to more easily skip pages that cannot be locked (Simon Riggs, Robert Haas)

  This change should greatly reduce the incidence of `VACUUM` getting "stuck" waiting for other sessions.

- Make `EXPLAIN (BUFFERS)` count blocks dirtied and written (Robert Haas)

- Make `EXPLAIN ANALYZE` report the number of rows rejected by filter steps (Marko Tiikkaja)

- Allow `EXPLAIN ANALYZE` to avoid timing overhead when time values are not wanted (Tomas Vondra)

  This is accomplished by setting the new `TIMING` option to `FALSE`.

E.25.3.6. Data Types

- Add support for range data types (Jeff Davis, Tom Lane, Alexander Korotkov)

  A range data type stores a lower and upper bound belonging to its base data type. It supports operations like contains, overlaps, and intersection.

- Add a `JSON` data type (Robert Haas)

  This type stores JSON (JavaScript Object Notation) data with proper validation.

- Add `array_to_json()` and `row_to_json()` (Andrew Dunstan)

- Add a `SMALLSERIAL` data type (Mike Pultz)

  This is like `SERIAL`, except it stores the sequence in a two-byte integer column (`int2`).

- Allow domains to be declared `NOT VALID` (Álvaro Herrera)

  This option can be set at domain creation time, or via `ALTER DOMAIN ... ADD CONSTRAINT ... NOT VALID`. `ALTER DOMAIN ... VALIDATE CONSTRAINT` fully validates the constraint.

- Support more locale-specific formatting options for the `money` data type (Tom Lane)

  Specifically, honor all the POSIX options for ordering of the value, sign, and currency symbol in monetary output. Also, make sure that the thousands separator is only inserted to the left of the decimal point, as required by POSIX.

- Add bitwise "and", "or", and "not" operators for the `macaddr` data type (Brendan Jurd)
- Allow `xpath()` to return a single-element XML array when supplied a scalar value (Florian Pflug)

  Previously, it returned an empty array. This change will also cause `xpath_exists()` to return true, not false, for such expressions.
- Improve XML error handling to be more robust (Florian Pflug)

E.25.3.7. Functions

- Allow non-superusers to use `pg_cancel_backend()` and `pg_terminate_backend()` on other sessions belonging to the same user (Magnus Hagander, Josh Kupershmidt, Dan Farina)

  Previously only superusers were allowed to use these functions.
- Allow importing and exporting of transaction snapshots (Joachim Wieland, Tom Lane)

  This allows multiple transactions to share identical views of the database state. Snapshots are exported via `pg_export_snapshot()` and imported via `SET TRANSACTION SNAPSHOT`. Only snapshots from currently-running transactions can be imported.
- Support `COLLATION FOR` on expressions (Peter Eisentraut)

  This returns a string representing the collation of the expression.
- Add `pg_opfamily_is_visible()` (Josh Kupershmidt)
- Add a `numeric` variant of `pg_size_pretty()` for use with `pg_xlog_location_diff()` (Fujii Masao)
- Add a `pg_trigger_depth()` function (Kevin Grittner)

  This reports the current trigger call depth.
- Allow `string_agg()` to process `bytea` values (Pavel Stehule)
- Fix regular expressions in which a back-reference occurs within a larger quantified subexpression (Tom Lane)

  For example, `^(\w+)( \1)+$`. Previous releases did not check that the back-reference actually matched the first occurrence.

E.25.3.8. Information Schema

- Add information schema views `role_udt_grants`, `udt_privileges`, and `user_defined_types` (Peter Eisentraut)
- Add composite-type attributes to the information schema `element_types` view (Peter Eisentraut)
- Implement `interval_type` columns in the information schema (Peter Eisentraut)

  Formerly these columns read as nulls.

- Implement collation-related columns in the information schema `attributes`, `columns`, `domains`, and `element_types` views (Peter Eisentraut)
- Implement the `with_hierarchy` column in the information schema `table_privileges` view (Peter Eisentraut)
- Add display of sequence `USAGE` privileges to information schema (Peter Eisentraut)
- Make the information schema show default privileges (Peter Eisentraut)

  Previously, non-empty default permissions were not represented in the views.

E.25.3.9. Server-Side Languages

E.25.3.9.1. PL/pgSQL Server-Side Language

- Allow the PL/pgSQL `OPEN` cursor command to supply parameters by name (Yeb Havinga)
- Add a `GET STACKED DIAGNOSTICS` PL/pgSQL command to retrieve exception info (Pavel Stehule)
- Speed up PL/pgSQL array assignment by caching type information (Pavel Stehule)
- Improve performance and memory consumption for long chains of `ELSIF` clauses (Tom Lane)
- Output the function signature, not just the name, in PL/pgSQL error messages (Pavel Stehule)

E.25.3.9.2. PL/Python Server-Side Language

- Add PL/Python SPI cursor support (Jan Urbanski)

  This allows PL/Python to read partial result sets.
- Add result metadata functions to PL/Python (Peter Eisentraut)

  Specifically, this adds result object functions `.colnames`, `.coltypes`, and `.coltypmods`.
- Remove support for Python 2.2 (Peter Eisentraut)

E.25.3.9.3. SQL Server-Side Language

- Allow SQL-language functions to reference parameters by name (Matthew Draper)

  To use this, simply name the function arguments and then reference the argument names in the SQL function body.

E.25.3.10. Client Applications

- Add initdb options `--auth-local` and `--auth-host` (Peter Eisentraut)

  This allows separate control of `local` and `host` `pg_hba.conf` authentication settings. `--auth` still controls both.
- Add `--replication`/`--no-replication` flags to createuser to control replication permission (Fujii Masao)
- Add the `--if-exists` option to dropdb and dropuser (Josh Kupershmidt)

- Give command-line tools the ability to specify the name of the database to connect to, and fall back to `template1` if a `postgres` database connection fails (Robert Haas)

E.25.3.10.1. psql

- Add a display mode to auto-expand output based on the display width (Peter Eisentraut)

  This adds the `auto` option to the `\x` command, which switches to the expanded mode when the normal output would be wider than the screen.

- Allow inclusion of a script file that is named relative to the directory of the file from which it was invoked (Gurjeet Singh)

  This is done with a new command `\ir`.

- Add support for non-ASCII characters in psql variable names (Tom Lane)

- Add support for major-version-specific `.psqlrc` files (Bruce Momjian)

  psql already supported minor-version-specific `.psqlrc` files.

- Provide environment variable overrides for psql history and startup file locations (Andrew Dunstan)

  `PSQL_HISTORY` and `PSQLRC` now determine these file names if set.

- Add a `\setenv` command to modify the environment variables passed to child processes (Andrew Dunstan)

- Name psql's temporary editor files with a `.sql` extension (Peter Eisentraut)

  This allows extension-sensitive editors to select the right mode.

- Allow psql to use zero-byte field and record separators (Peter Eisentraut)

  Various shell tools use zero-byte (NUL) separators, e.g. find.

- Make the `\timing` option report times for failed queries (Magnus Hagander)

  Previously times were reported only for successful queries.

- Unify and tighten psql's treatment of `\copy` and SQL `COPY` (Noah Misch)

  This fix makes failure behavior more predictable and honors `\set ON_ERROR_ROLLBACK`.

E.25.3.10.2. Informational Commands

- Make `\d` on a sequence show the table/column name owning it (Magnus Hagander)

- Show statistics target for columns in `\d+` (Magnus Hagander)

- Show role password expiration dates in `\du` (Fabrízio de Royes Mello)

- Display comments for casts, conversions, domains, and languages (Josh Kupershmidt)

  These are included in the output of `\dC+`, `\dc+`, `\dD+`, and `\dL` respectively.

- Display comments for SQL/MED objects (Josh Kupershmidt)

These are included in the output of `\des+`, `\det+`, and `\dew+` for foreign servers, foreign tables, and foreign data wrappers respectively.

- Change `\dd` to display comments only for object types without their own backslash command (Josh Kupershmidt)

E.25.3.10.3. Tab Completion

- In psql tab completion, complete SQL keywords in either upper or lower case according to the new `COMP_KEYWORD_CASE` setting (Peter Eisentraut)
- Add tab completion support for `EXECUTE` (Andreas Karlsson)
- Allow tab completion of role references in `GRANT`/`REVOKE` (Peter Eisentraut)
- Allow tab completion of file names to supply quotes, when necessary (Noah Misch)
- Change tab completion support for `TABLE` to also include views (Magnus Hagander)

E.25.3.10.4. pg_dump

- Add an `--exclude-table-data` option to pg_dump (Andrew Dunstan)

  This allows dumping of a table's definition but not its data, on a per-table basis.

- Add a `--section` option to pg_dump and pg_restore (Andrew Dunstan)

  Valid values are `pre-data`, `data`, and `post-data`. The option can be given more than once to select two or more sections.

- Make pg_dumpall dump all roles first, then all configuration settings on roles (Phil Sorber)

  This allows a role's configuration settings to mention other roles without generating an error.

- Allow pg_dumpall to avoid errors if the `postgres` database is missing in the new cluster (Robert Haas)
- Dump foreign server user mappings in user name order (Peter Eisentraut)

  This helps produce deterministic dump files.

- Dump operators in a predictable order (Peter Eisentraut)
- Tighten rules for when extension configuration tables are dumped by pg_dump (Tom Lane)
- Make pg_dump emit more useful dependency information (Tom Lane)

  The dependency links included in archive-format dumps were formerly of very limited use, because they frequently referenced objects that appeared nowhere in the dump. Now they represent actual dependencies (possibly indirect) among the dumped objects.

- Improve pg_dump's performance when dumping many database objects (Tom Lane)

E.25.3.11. libpq

- Allow libpq connection strings to have the format of a URI (Alexander Shulgin)

The syntax begins with `postgres://`. This can allow applications to avoid implementing their own parser for URIs representing database connections.

- Add a connection option to disable SSL compression (Laurenz Albe)

  This can be used to remove the overhead of SSL compression on fast networks.

- Add a single-row processing mode for better handling of large result sets (Kyotaro Horiguchi, Marko Kreen)

  Previously, libpq always collected the entire query result in memory before passing it back to the application.

- Add `const` qualifiers to the declarations of the functions `PQconnectdbParams`, `PQconnectStartParams`, and `PQpingParams` (Lionel Elie Mamane)

- Allow the `.pgpass` file to include escaped characters in the password field (Robert Haas)

- Make library functions use `abort()` instead of `exit()` when it is necessary to terminate the process (Peter Eisentraut)

  This choice does not interfere with the normal exit codes used by the program, and generates a signal that can be caught by the caller.

E.25.3.12. Source Code

- Remove dead ports (Peter Eisentraut)

  The following platforms are no longer supported: dgux, nextstep, sunos4, svr4, ultrix4, univel, bsdi.

- Add support for building with MS Visual Studio 2010 (Brar Piening)

- Enable compiling with the MinGW-w64 32-bit compiler (Lars Kanis)

- Install `plpgsql.h` into `include/server` during installation (Heikki Linnakangas)

- Improve the latch facility to include detection of postmaster death (Peter Geoghegan, Heikki Linnakangas, Tom Lane)

  This eliminates one of the main reasons that background processes formerly had to wake up to poll for events.

- Use C flexible array members, where supported (Peter Eisentraut)

- Improve the concurrent transaction regression tests (isolationtester) (Noah Misch)

- Modify thread_test to create its test files in the current directory, rather than `/tmp` (Bruce Momjian)

- Improve flex and bison warning and error reporting (Tom Lane)

- Add memory barrier support (Robert Haas)

  This is currently unused.

- Modify pgindent to use a typedef file (Bruce Momjian)

- Add a hook for processing messages due to be sent to the server log (Martin Pihlak)

- Add object access hooks for `DROP` commands (KaiGai Kohei)

- Centralize `DROP` handling for some object types (KaiGai Kohei)
- Add a pg_upgrade test suite (Peter Eisentraut)
- Sync regular expression code with TCL 8.5.11 and improve internal processing (Tom Lane)
- Move CRC tables to libpgport, and provide them in a separate include file (Daniel Farina)
- Add options to git_changelog for use in major release note creation (Bruce Momjian)
- Support Linux's `/proc/self/oom_score_adj` API (Tom Lane)

E.25.3.13. Additional Modules

- Improve efficiency of dblink by using libpq's new single-row processing mode (Kyotaro Horiguchi, Marko Kreen)

  This improvement does not apply to `dblink_send_query()`/`dblink_get_result()`.

- Support `force_not_null` option in file_fdw (Shigeru Hanada)
- Implement dry-run mode for pg_archivecleanup (Gabriele Bartolini)

  This only outputs the names of files to be deleted.

- Add new pgbench switches `--unlogged-tables`, `--tablespace`, and `--index-tablespace` (Robert Haas)
- Change pg_test_fsync to test for a fixed amount of time, rather than a fixed number of cycles (Bruce Momjian)

  The `-o`/cycles option was removed, and `-s`/seconds added.

- Add a pg_test_timing utility to measure clock monotonicity and timing overhead (Ants Aasma, Greg Smith)
- Add a tcn (triggered change notification) module to generate `NOTIFY` events on table changes (Kevin Grittner)

E.25.3.13.1. pg_upgrade

- Adjust pg_upgrade environment variables (Bruce Momjian)

  Rename data, bin, and port environment variables to begin with `PG`, and support `PGPORTOLD`/`PGPORTNEW`, to replace `PGPORT`.

- Overhaul pg_upgrade logging and failure reporting (Bruce Momjian)

  Create four append-only log files, and delete them on success. Add `-r`/`--retain` option to unconditionally retain these files. Also remove pg_upgrade options `-g`/`-G`/`-l` options as unnecessary, and tighten log file permissions.

- Make pg_upgrade create a script to incrementally generate more accurate optimizer statistics (Bruce Momjian)

  This reduces the time needed to generate minimal cluster statistics after an upgrade.

- Allow pg_upgrade to upgrade an old cluster that does not have a `postgres` database (Bruce Momjian)

- Allow pg_upgrade to handle cases where some old or new databases are missing, as long as they are empty (Bruce Momjian)
- Allow pg_upgrade to handle configuration-only directory installations (Bruce Momjian)
- In pg_upgrade, add `-o/-O` options to pass parameters to the servers (Bruce Momjian)

  This is useful for configuration-only directory installs.
- Change pg_upgrade to use port 50432 by default (Bruce Momjian)

  This helps avoid unintended client connections during the upgrade.
- Reduce cluster locking in pg_upgrade (Bruce Momjian)

  Specifically, only lock the old cluster if link mode is used, and do it right after the schema is restored.

E.25.3.13.2. pg_stat_statements

- Allow pg_stat_statements to aggregate similar queries via SQL text normalization (Peter Geoghegan, Tom Lane)

  Users with applications that use non-parameterized SQL will now be able to monitor query performance without detailed log analysis.
- Add dirtied and written block counts and read/write times to pg_stat_statements (Robert Haas, Ants Aasma)
- Prevent pg_stat_statements from double-counting `PREPARE` and `EXECUTE` commands (Tom Lane)

E.25.3.13.3. sepgsql

- Support `SECURITY LABEL` on global objects (KaiGai Kohei, Robert Haas)

  Specifically, add security labels to databases, tablespaces, and roles.
- Allow sepgsql to honor database labels (KaiGai Kohei)
- Perform sepgsql permission checks during the creation of various objects (KaiGai Kohei)
- Add `sepgsql_setcon()` and related functions to control the sepgsql security domain (KaiGai Kohei)
- Add a user space access cache to sepgsql to improve performance (KaiGai Kohei)

E.25.3.14. Documentation

- Add a rule to optionally build HTML documentation using the stylesheet from the website (Magnus Hagander)

  Use `gmake STYLE=website draft`.
- Improve `EXPLAIN` documentation (Tom Lane)
- Document that user/database names are preserved with double-quoting by command-line tools like vacuumdb (Bruce Momjian)

- Document the actual string returned by the client for MD5 authentication (Cyan Ogilvie)
- Deprecate use of `GLOBAL` and `LOCAL` in `CREATE TEMP TABLE` (Noah Misch)

  PostgreSQL has long treated these keyword as no-ops, and continues to do so; but in future they might mean what the SQL standard says they mean, so applications should avoid using them.

E.26. Release 9.1.24

Release date: 2016-10-27

This release contains a variety of fixes from 9.1.23. For information about new features in the 9.1 major release, see 第 E.50 节.

This is expected to be the last PostgreSQL release in the 9.1.X series. Users are encouraged to update to a newer release branch soon.

E.26.1. Migration to Version 9.1.24

A dump/restore is not required for those running 9.1.X.

However, if you are upgrading from a version earlier than 9.1.16, see 第 E.34 节.

E.26.2. Changes

- Fix EvalPlanQual rechecks involving CTE scans (Tom Lane)

  The recheck would always see the CTE as returning no rows, typically leading to failure to update rows that were recently updated.

- Fix improper repetition of previous results from hashed aggregation in a subquery (Andrew Gierth)

  The test to see if we can reuse a previously-computed hash table of the aggregate state values neglected the possibility of an outer query reference appearing in an aggregate argument expression. A change in the value of such a reference should lead to recalculating the hash table, but did not.

- Fix timeout length when `VACUUM` is waiting for exclusive table lock so that it can truncate the table (Simon Riggs)

  The timeout was meant to be 50 milliseconds, but it was actually only 50 microseconds, causing `VACUUM` to give up on truncation much more easily than intended. Set it to the intended value.

- Remove artificial restrictions on the values accepted by `numeric_in()` and `numeric_recv()` (Tom Lane)

  We allow numeric values up to the limit of the storage format (more than `1e100000`), so it seems fairly pointless that `numeric_in()` rejected scientific-notation exponents above 1000. Likewise, it was silly for `numeric_recv()` to reject more than 1000 digits in an input value.

- Avoid very-low-probability data corruption due to testing tuple visibility without holding buffer lock (Thomas Munro, Peter Geoghegan, Tom Lane)
- Fix file descriptor leakage when truncating a temporary relation of more than 1GB (Andres Freund)

- Disallow starting a standalone backend with `standby_mode` turned on (Michael Paquier)

  This can't do anything useful, since there will be no WAL receiver process to fetch more WAL data; and it could result in misbehavior in code that wasn't designed with this situation in mind.

- Don't try to share SSL contexts across multiple connections in libpq (Heikki Linnakangas)

  This led to assorted corner-case bugs, particularly when trying to use different SSL parameters for different connections.

- Avoid corner-case memory leak in libpq (Tom Lane)

  The reported problem involved leaking an error report during `PQreset()`, but there might be related cases.

- Make ecpg's `--help` and `--version` options work consistently with our other executables (Haribabu Kommi)

- Fix `contrib/intarray/bench/bench.pl` to print the results of the `EXPLAIN` it does when given the `-e` option (Daniel Gustafsson)

- Prevent failure of obsolete dynamic time zone abbreviations (Tom Lane)

  If a dynamic time zone abbreviation does not match any entry in the referenced time zone, treat it as equivalent to the time zone name. This avoids unexpected failures when IANA removes abbreviations from their time zone database, as they did in tzdata release 2016f and seem likely to do again in the future. The consequences were not limited to not recognizing the individual abbreviation; any mismatch caused the `pg_timezone_abbrevs` view to fail altogether.

- Update time zone data files to tzdata release 2016h for DST law changes in Palestine and Turkey, plus historical corrections for Turkey and some regions of Russia. Switch to numeric abbreviations for some time zones in Antarctica, the former Soviet Union, and Sri Lanka.

  The IANA time zone database previously provided textual abbreviations for all time zones, sometimes making up abbreviations that have little or no currency among the local population. They are in process of reversing that policy in favor of using numeric UTC offsets in zones where there is no evidence of real-world use of an English abbreviation. At least for the time being, PostgreSQL will continue to accept such removed abbreviations for timestamp input. But they will not be shown in the `pg_timezone_names` view nor used for output.

  In this update, `AMT` is no longer shown as being in use to mean Armenia Time. Therefore, we have changed the `Default` abbreviation set to interpret it as Amazon Time, thus UTC-4 not UTC+4.

E.27. Release 9.1.23

Release date: 2016-08-11

This release contains a variety of fixes from 9.1.22. For information about new features in the 9.1 major release, see 第 E.50 节.

The PostgreSQL community will stop releasing updates for the 9.1.X release series in September 2016. Users are encouraged to update to a newer release branch soon.

E.27.1. Migration to Version 9.1.23

A dump/restore is not required for those running 9.1.X.

However, if you are upgrading from a version earlier than 9.1.16, see 第 E.34 节.

E.27.2. Changes

- Fix possible mis-evaluation of nested `CASE-WHEN` expressions (Heikki Linnakangas, Michael Paquier, Tom Lane)

  A `CASE` expression appearing within the test value subexpression of another `CASE` could become confused about whether its own test value was null or not. Also, inlining of a SQL function implementing the equality operator used by a `CASE` expression could result in passing the wrong test value to functions called within a `CASE` expression in the SQL function's body. If the test values were of different data types, a crash might result; moreover such situations could be abused to allow disclosure of portions of server memory. (CVE-2016-5423)

- Fix client programs' handling of special characters in database and role names (Noah Misch, Nathan Bossart, Michael Paquier)

  Numerous places in vacuumdb and other client programs could become confused by database and role names containing double quotes or backslashes. Tighten up quoting rules to make that safe. Also, ensure that when a conninfo string is used as a database name parameter to these programs, it is correctly treated as such throughout.

  Fix handling of paired double quotes in psql's `\connect` and `\password` commands to match the documentation.

  Introduce a new `-reuse-previous` option in psql's `\connect` command to allow explicit control of whether to re-use connection parameters from a previous connection. (Without this, the choice is based on whether the database name looks like a conninfo string, as before.) This allows secure handling of database names containing special characters in pg_dumpall scripts.

  pg_dumpall now refuses to deal with database and role names containing carriage returns or newlines, as it seems impractical to quote those characters safely on Windows. In future we may reject such names on the server side, but that step has not been taken yet.

  These are considered security fixes because crafted object names containing special characters could have been used to execute commands with superuser privileges the next time a superuser executes pg_dumpall or other routine maintenance operations. (CVE-2016-5424)

- Fix corner-case misbehaviors for `IS NULL`/`IS NOT NULL` applied to nested composite values (Andrew Gierth, Tom Lane)

  The SQL standard specifies that `IS NULL` should return TRUE for a row of all null values (thus `ROW(NULL,NULL) IS NULL` yields TRUE), but this is not meant to apply recursively (thus `ROW(NULL, ROW(NULL,NULL)) IS NULL` yields FALSE). The core executor got this right, but certain planner optimizations treated the test as recursive (thus producing TRUE in both cases), and `contrib/postgres_fdw` could produce remote queries that misbehaved similarly.

- Make the `inet` and `cidr` data types properly reject IPv6 addresses with too many colon-separated fields (Tom Lane)

- Prevent crash in `close_ps()` (the `point ## lseg` operator) for NaN input coordinates (Tom Lane)

  Make it return NULL instead of crashing.

- Fix several one-byte buffer over-reads in `to_number()` (Peter Eisentraut)

  In several cases the `to_number()` function would read one more character than it should from the input string. There is a small chance of a crash, if the input happens to be adjacent to the end of memory.

- Avoid unsafe intermediate state during expensive paths through `heap_update()` (Masahiko Sawada, Andres Freund)

  Previously, these cases locked the target tuple (by setting its XMAX) but did not WAL-log that action, thus risking data integrity problems if the page were spilled to disk and then a database crash occurred before the tuple update could be completed.

- Avoid consuming a transaction ID during `VACUUM` (Alexander Korotkov)

  Some cases in `VACUUM` unnecessarily caused an XID to be assigned to the current transaction. Normally this is negligible, but if one is up against the XID wraparound limit, consuming more XIDs during anti-wraparound vacuums is a very bad thing.

- Avoid canceling hot-standby queries during `VACUUM FREEZE` (Simon Riggs, Álvaro Herrera)

  `VACUUM FREEZE` on an otherwise-idle master server could result in unnecessary cancellations of queries on its standby servers.

- When a manual `ANALYZE` specifies a column list, don't reset the table's `changes_since_analyze` counter (Tom Lane)

  If we're only analyzing some columns, we should not prevent routine auto-analyze from happening for the other columns.

- Fix `ANALYZE`'s overestimation of `n_distinct` for a unique or nearly-unique column with many null entries (Tom Lane)

  The nulls could get counted as though they were themselves distinct values, leading to serious planner misestimates in some types of queries.

- Prevent autovacuum from starting multiple workers for the same shared catalog (Álvaro Herrera)

  Normally this isn't much of a problem because the vacuum doesn't take long anyway; but in the case of a severely bloated catalog, it could result in all but one worker uselessly waiting instead of doing useful work on other tables.

- Fix `contrib/btree_gin` to handle the smallest possible `bigint` value correctly (Peter Eisentraut)

- Teach libpq to correctly decode server version from future servers (Peter Eisentraut)

  It's planned to switch to two-part instead of three-part server version numbers for releases after 9.6. Make sure that `PQserverVersion()` returns the correct value for such cases.

- Fix ecpg's code for `unsigned long long` array elements (Michael Meskes)

- Make pg_basebackup accept `-Z 0` as specifying no compression (Fujii Masao)

- Revert to the old heuristic timeout for `pg_ctl start -w` (Tom Lane)

  The new method adopted as of release 9.1.20 does not work when `silent_mode` is enabled, so go back to the old way.

- Fix makefiles' rule for building AIX shared libraries to be safe for parallel make (Noah Misch)

- Fix TAP tests and MSVC scripts to work when build directory's path name contains spaces (Michael Paquier, Kyotaro Horiguchi)

- Make regression tests safe for Danish and Welsh locales (Jeff Janes, Tom Lane)

  Change some test data that triggered the unusual sorting rules of these locales.

- Update our copy of the timezone code to match IANA's tzcode release 2016c (Tom Lane)

  This is needed to cope with anticipated future changes in the time zone data files. It also fixes some corner-case bugs in coping with unusual time zones.

- Update time zone data files to tzdata release 2016f for DST law changes in Kemerovo and Novosibirsk, plus historical corrections for Azerbaijan, Belarus, and Morocco.

E.28. Release 9.1.22

Release date: 2016-05-12

This release contains a variety of fixes from 9.1.21. For information about new features in the 9.1 major release, see 第 E.50 节.

The PostgreSQL community will stop releasing updates for the 9.1.X release series in September 2016. Users are encouraged to update to a newer release branch soon.

E.28.1. Migration to Version 9.1.22

A dump/restore is not required for those running 9.1.X.

However, if you are upgrading from a version earlier than 9.1.16, see 第 E.34 节.

E.28.2. Changes

- Clear the OpenSSL error queue before OpenSSL calls, rather than assuming it's clear already; and make sure we leave it clear afterwards (Peter Geoghegan, Dave Vitek, Peter Eisentraut)

  This change prevents problems when there are multiple connections using OpenSSL within a single process and not all the code involved follows the same rules for when to clear the error queue. Failures have been reported specifically when a client application uses SSL connections in libpq concurrently with SSL connections using the PHP, Python, or Ruby wrappers for OpenSSL. It's possible for similar problems to arise within the server as well, if an extension module establishes an outgoing SSL connection.

- Fix "failed to build any *N*-way joins" planner error with a full join enclosed in the right-hand side of a left join (Tom Lane)

- Fix possible misbehavior of `TH`, `th`, and `Y,YYY` format codes in `to_timestamp()` (Tom Lane)

These could advance off the end of the input string, causing subsequent format codes to read garbage.

- Fix dumping of rules and views in which the *array* argument of a *value operator* ANY (*array*) construct is a sub-SELECT (Tom Lane)

- Make pg_regress use a startup timeout from the PGCTLTIMEOUT environment variable, if that's set (Tom Lane)

  This is for consistency with a behavior recently added to pg_ctl; it eases automated testing on slow machines.

- Fix pg_upgrade to correctly restore extension membership for operator families containing only one operator class (Tom Lane)

  In such a case, the operator family was restored into the new database, but it was no longer marked as part of the extension. This had no immediate ill effects, but would cause later pg_dump runs to emit output that would cause (harmless) errors on restore.

- Rename internal function strtoi() to strtoint() to avoid conflict with a NetBSD library function (Thomas Munro)

- Fix reporting of errors from bind() and listen() system calls on Windows (Tom Lane)

- Reduce verbosity of compiler output when building with Microsoft Visual Studio (Christian Ullrich)

- Avoid possibly-unsafe use of Windows' FormatMessage() function (Christian Ullrich)

  Use the FORMAT_MESSAGE_IGNORE_INSERTS flag where appropriate. No live bug is known to exist here, but it seems like a good idea to be careful.

- Update time zone data files to tzdata release 2016d for DST law changes in Russia and Venezuela. There are new zone names Europe/Kirov and Asia/Tomsk to reflect the fact that these regions now have different time zone histories from adjacent regions.

E.29. Release 9.1.21

Release date: 2016-03-31

This release contains a variety of fixes from 9.1.20. For information about new features in the 9.1 major release, see 第 E.50 节.

E.29.1. Migration to Version 9.1.21

A dump/restore is not required for those running 9.1.X.

However, if you are upgrading from a version earlier than 9.1.16, see 第 E.34 节.

E.29.2. Changes

- Fix incorrect handling of NULL index entries in indexed ROW() comparisons (Tom Lane)

An index search using a row comparison such as `ROW(a, b) > ROW('x', 'y')` would stop upon reaching a NULL entry in the `b` column, ignoring the fact that there might be non-NULL `b` values associated with later values of `a`.

- Avoid unlikely data-loss scenarios due to renaming files without adequate `fsync()` calls before and after (Michael Paquier, Tomas Vondra, Andres Freund)
- Correctly handle cases where `pg_subtrans` is close to XID wraparound during server startup (Jeff Janes)
- Fix corner-case crash due to trying to free `localeconv()` output strings more than once (Tom Lane)
- Fix parsing of affix files for `ispell` dictionaries (Tom Lane)

  The code could go wrong if the affix file contained any characters whose byte length changes during case-folding, for example `I` in Turkish UTF8 locales.
- Avoid use of `sscanf()` to parse `ispell` dictionary files (Artur Zakirov)

  This dodges a portability problem on FreeBSD-derived platforms (including macOS).
- Avoid a crash on old Windows versions (before 7SP1/2008R2SP1) with an AVX2-capable CPU and a Postgres build done with Visual Studio 2013 (Christian Ullrich)

  This is a workaround for a bug in Visual Studio 2013's runtime library, which Microsoft have stated they will not fix in that version.
- Fix psql's tab completion logic to handle multibyte characters properly (Kyotaro Horiguchi, Robert Haas)
- Fix psql's tab completion for `SECURITY LABEL` (Tom Lane)

  Pressing TAB after `SECURITY LABEL` might cause a crash or offering of inappropriate keywords.
- Make pg_ctl accept a wait timeout from the `PGCTLTIMEOUT` environment variable, if none is specified on the command line (Noah Misch)

  This eases testing of slower buildfarm members by allowing them to globally specify a longer-than-normal timeout for postmaster startup and shutdown.
- Fix incorrect test for Windows service status in pg_ctl (Manuel Mathar)

  The previous set of minor releases attempted to fix pg_ctl to properly determine whether to send log messages to Window's Event Log, but got the test backwards.
- Fix pgbench to correctly handle the combination of `-C` and `-M prepared` options (Tom Lane)
- In PL/Perl, properly translate empty Postgres arrays into empty Perl arrays (Alex Hunsaker)
- Make PL/Python cope with function names that aren't valid Python identifiers (Jim Nasby)
- Fix multiple mistakes in the statistics returned by `contrib/pgstattuple`'s `pgstatindex()` function (Tom Lane)
- Remove dependency on `psed` in MSVC builds, since it's no longer provided by core Perl (Michael Paquier, Andrew Dunstan)

- Update time zone data files to tzdata release 2016c for DST law changes in Azerbaijan, Chile, Haiti, Palestine, and Russia (Altai, Astrakhan, Kirov, Sakhalin, Ulyanovsk regions), plus historical corrections for Lithuania, Moldova, and Russia (Kaliningrad, Samara, Volgograd).

E.30. Release 9.1.20

Release date: 2016-02-11

This release contains a variety of fixes from 9.1.19. For information about new features in the 9.1 major release, see 第 E.50 节.

E.30.1. Migration to Version 9.1.20

A dump/restore is not required for those running 9.1.X.

However, if you are upgrading from a version earlier than 9.1.16, see 第 E.34 节.

E.30.2. Changes

- Fix infinite loops and buffer-overrun problems in regular expressions (Tom Lane)

  Very large character ranges in bracket expressions could cause infinite loops in some cases, and memory overwrites in other cases. (CVE-2016-0773)

- Perform an immediate shutdown if the `postmaster.pid` file is removed (Tom Lane)

  The postmaster now checks every minute or so that `postmaster.pid` is still there and still contains its own PID. If not, it performs an immediate shutdown, as though it had received SIGQUIT. The main motivation for this change is to ensure that failed buildfarm runs will get cleaned up without manual intervention; but it also serves to limit the bad effects if a DBA forcibly removes `postmaster.pid` and then starts a new postmaster.

- In `SERIALIZABLE` transaction isolation mode, serialization anomalies could be missed due to race conditions during insertions (Kevin Grittner, Thomas Munro)

- Fix failure to emit appropriate WAL records when doing `ALTER TABLE ... SET TABLESPACE` for unlogged relations (Michael Paquier, Andres Freund)

  Even though the relation's data is unlogged, the move must be logged or the relation will be inaccessible after a standby is promoted to master.

- Fix possible misinitialization of unlogged relations at the end of crash recovery (Andres Freund, Michael Paquier)

- Fix `ALTER COLUMN TYPE` to reconstruct inherited check constraints properly (Tom Lane)

- Fix `REASSIGN OWNED` to change ownership of composite types properly (Álvaro Herrera)

- Fix `REASSIGN OWNED` and `ALTER OWNER` to correctly update granted-permissions lists when changing owners of data types, foreign data wrappers, or foreign servers (Bruce Momjian, Álvaro Herrera)

- Fix `REASSIGN OWNED` to ignore foreign user mappings, rather than fail (Álvaro Herrera)

- Add more defenses against bad planner cost estimates for GIN index scans when the index's internal statistics are very out-of-date (Tom Lane)
- Make planner cope with hypothetical GIN indexes suggested by an index advisor plug-in (Julien Rouhaud)
- Fix dumping of whole-row Vars in `ROW()` and `VALUES()` lists (Tom Lane)
- Fix possible internal overflow in `numeric` division (Dean Rasheed)
- Fix enforcement of restrictions inside parentheses within regular expression lookahead constraints (Tom Lane)

  Lookahead constraints aren't allowed to contain backrefs, and parentheses within them are always considered non-capturing, according to the manual. However, the code failed to handle these cases properly inside a parenthesized subexpression, and would give unexpected results.
- Conversion of regular expressions to indexscan bounds could produce incorrect bounds from regexps containing lookahead constraints (Tom Lane)
- Fix regular-expression compiler to handle loops of constraint arcs (Tom Lane)

  The code added for CVE-2007-4772 was both incomplete, in that it didn't handle loops involving more than one state, and incorrect, in that it could cause assertion failures (though there seem to be no bad consequences of that in a non-assert build). Multi-state loops would cause the compiler to run until the query was canceled or it reached the too-many-states error condition.
- Improve memory-usage accounting in regular-expression compiler (Tom Lane)

  This causes the code to emit “regular expression is too complex” errors in some cases that previously used unreasonable amounts of time and memory.
- Improve performance of regular-expression compiler (Tom Lane)
- Make `%h` and `%r` escapes in `log_line_prefix` work for messages emitted due to `log_connections` (Tom Lane)

  Previously, `%h`/`%r` started to work just after a new session had emitted the “connection received” log message; now they work for that message too.
- On Windows, ensure the shared-memory mapping handle gets closed in child processes that don't need it (Tom Lane, Amit Kapila)

  This oversight resulted in failure to recover from crashes whenever `logging_collector` is turned on.
- Fix possible failure to detect socket EOF in non-blocking mode on Windows (Tom Lane)

  It's not entirely clear whether this problem can happen in pre-9.5 branches, but if it did, the symptom would be that a walsender process would wait indefinitely rather than noticing a loss of connection.
- Avoid leaking a token handle during SSPI authentication (Christian Ullrich)
- In psql, ensure that libreadline's idea of the screen size is updated when the terminal window size changes (Merlin Moncure)

  Previously, libreadline did not notice if the window was resized during query output, leading to strange behavior during later input of multiline queries.

- Fix psql's `\det` command to interpret its pattern argument the same way as other `\d` commands with potentially schema-qualified patterns do (Reece Hart)
- Avoid possible crash in psql's `\c` command when previous connection was via Unix socket and command specifies a new hostname and same username (Tom Lane)
- In `pg_ctl start -w`, test child process status directly rather than relying on heuristics (Tom Lane, Michael Paquier)

  Previously, pg_ctl relied on an assumption that the new postmaster would always create `postmaster.pid` within five seconds. But that can fail on heavily-loaded systems, causing pg_ctl to report incorrectly that the postmaster failed to start.

  Except on Windows, this change also means that a `pg_ctl start -w` done immediately after another such command will now reliably fail, whereas previously it would report success if done within two seconds of the first command.
- In `pg_ctl start -w`, don't attempt to use a wildcard listen address to connect to the postmaster (Kondo Yuta)

  On Windows, pg_ctl would fail to detect postmaster startup if `listen_addresses` is set to `0.0.0.0` or `::`, because it would try to use that value verbatim as the address to connect to, which doesn't work. Instead assume that `127.0.0.1` or `::1`, respectively, is the right thing to use.
- In pg_ctl on Windows, check service status to decide where to send output, rather than checking if standard output is a terminal (Michael Paquier)
- In pg_dump and pg_basebackup, adopt the GNU convention for handling tar-archive members exceeding 8GB (Tom Lane)

  The POSIX standard for `tar` file format does not allow archive member files to exceed 8GB, but most modern implementations of tar support an extension that fixes that. Adopt this extension so that pg_dump with `-Ft` no longer fails on tables with more than 8GB of data, and so that pg_basebackup can handle files larger than 8GB. In addition, fix some portability issues that could cause failures for members between 4GB and 8GB on some platforms. Potentially these problems could cause unrecoverable data loss due to unreadable backup files.
- Fix assorted corner-case bugs in pg_dump's processing of extension member objects (Tom Lane)
- Make pg_dump mark a view's triggers as needing to be processed after its rule, to prevent possible failure during parallel pg_restore (Tom Lane)
- Ensure that relation option values are properly quoted in pg_dump (Kouhei Sutou, Tom Lane)

  A reloption value that isn't a simple identifier or number could lead to dump/reload failures due to syntax errors in CREATE statements issued by pg_dump. This is not an issue with any reloption currently supported by core PostgreSQL, but extensions could allow reloptions that cause the problem.
- Fix pg_upgrade's file-copying code to handle errors properly on Windows (Bruce Momjian)
- Install guards in pgbench against corner-case overflow conditions during evaluation of script-specified division or modulo operators (Fabien Coelho, Michael Paquier)
- Prevent certain PL/Java parameters from being set by non-superusers (Noah Misch)

This change mitigates a PL/Java security bug (CVE-2016-0766), which was fixed in PL/Java by marking these parameters as superuser-only. To fix the security hazard for sites that update PostgreSQL more frequently than PL/Java, make the core code aware of them also.

- Improve libpq's handling of out-of-memory situations (Michael Paquier, Amit Kapila, Heikki Linnakangas)
- Fix order of arguments in ecpg-generated `typedef` statements (Michael Meskes)
- Use `%g` not `%f` format in ecpg's `PGTYPESnumeric_from_double()` (Tom Lane)
- Fix ecpg-supplied header files to not contain comments continued from a preprocessor directive line onto the next line (Michael Meskes)

  Such a comment is rejected by ecpg. It's not yet clear whether ecpg itself should be changed.
- Ensure that `contrib/pgcrypto`'s `crypt()` function can be interrupted by query cancel (Andreas Karlsson)
- Accept flex versions later than 2.5.x (Tom Lane, Michael Paquier)

  Now that flex 2.6.0 has been released, the version checks in our build scripts needed to be adjusted.
- Install our `missing` script where PGXS builds can find it (Jim Nasby)

  This allows sane behavior in a PGXS build done on a machine where build tools such as bison are missing.
- Ensure that `dynloader.h` is included in the installed header files in MSVC builds (Bruce Momjian, Michael Paquier)
- Add variant regression test expected-output file to match behavior of current libxml2 (Tom Lane)

  The fix for libxml2's CVE-2015-7499 causes it not to output error context reports in some cases where it used to do so. This seems to be a bug, but we'll probably have to live with it for some time, so work around it.
- Update time zone data files to tzdata release 2016a for DST law changes in Cayman Islands, Metlakatla, and Trans-Baikal Territory (Zabaykalsky Krai), plus historical corrections for Pakistan.

E.31. Release 9.1.19

Release date: 2015-10-08

This release contains a variety of fixes from 9.1.18. For information about new features in the 9.1 major release, see 第 E.50 节.

E.31.1. Migration to Version 9.1.19

A dump/restore is not required for those running 9.1.X.

However, if you are upgrading from a version earlier than 9.1.16, see 第 E.34 节.

E.31.2. Changes

- Fix `contrib/pgcrypto` to detect and report too-short `crypt()` salts (Josh Kupershmidt)

  Certain invalid salt arguments crashed the server or disclosed a few bytes of server memory. We have not ruled out the viability of attacks that arrange for presence of confidential information in the disclosed bytes, but they seem unlikely. (CVE-2015-5288)

- Fix subtransaction cleanup after a portal (cursor) belonging to an outer subtransaction fails (Tom Lane, Michael Paquier)

  A function executed in an outer-subtransaction cursor could cause an assertion failure or crash by referencing a relation created within an inner subtransaction.

- Fix insertion of relations into the relation cache "init file" (Tom Lane)

  An oversight in a patch in the most recent minor releases caused `pg_trigger_tgrelid_tgname_index` to be omitted from the init file. Subsequent sessions detected this, then deemed the init file to be broken and silently ignored it, resulting in a significant degradation in session startup time. In addition to fixing the bug, install some guards so that any similar future mistake will be more obvious.

- Avoid O(N^2) behavior when inserting many tuples into a SPI query result (Neil Conway)

- Improve `LISTEN` startup time when there are many unread notifications (Matt Newell)

- Back-patch 9.3-era addition of per-resource-owner lock caches (Jeff Janes)

  This substantially improves performance when pg_dump tries to dump a large number of tables.

- Disable SSL renegotiation by default (Michael Paquier, Andres Freund)

  While use of SSL renegotiation is a good idea in theory, we have seen too many bugs in practice, both in the underlying OpenSSL library and in our usage of it. Renegotiation will be removed entirely in 9.5 and later. In the older branches, just change the default value of `ssl_renegotiation_limit` to zero (disabled).

- Lower the minimum values of the `*_freeze_max_age` parameters (Andres Freund)

  This is mainly to make tests of related behavior less time-consuming, but it may also be of value for installations with limited disk space.

- Limit the maximum value of `wal_buffers` to 2GB to avoid server crashes (Josh Berkus)

- Fix rare internal overflow in multiplication of `numeric` values (Dean Rasheed)

- Guard against hard-to-reach stack overflows involving record types, range types, `json`, `jsonb`, `tsquery`, `ltxtquery` and `query_int` (Noah Misch)

- Fix handling of `DOW` and `DOY` in datetime input (Greg Stark)

  These tokens aren't meant to be used in datetime values, but previously they resulted in opaque internal error messages rather than "invalid input syntax".

- Add more query-cancel checks to regular expression matching (Tom Lane)

- Add recursion depth protections to regular expression, `SIMILAR TO`, and `LIKE` matching (Tom Lane)

  Suitable search patterns and a low stack depth limit could lead to stack-overrun crashes.

- Fix potential infinite loop in regular expression execution (Tom Lane)

  A search pattern that can apparently match a zero-length string, but actually doesn't match because of a back reference, could lead to an infinite loop.

- Fix low-memory failures in regular expression compilation (Andreas Seltenreich)

- Fix low-probability memory leak during regular expression execution (Tom Lane)

- Fix rare low-memory failure in lock cleanup during transaction abort (Tom Lane)

- Fix "unexpected out-of-memory situation during sort" errors when using tuplestores with small `work_mem` settings (Tom Lane)

- Fix very-low-probability stack overrun in `qsort` (Tom Lane)

- Fix "invalid memory alloc request size" failure in hash joins with large `work_mem` settings (Tomas Vondra, Tom Lane)

- Fix assorted planner bugs (Tom Lane)

  These mistakes could lead to incorrect query plans that would give wrong answers, or to assertion failures in assert-enabled builds, or to odd planner errors such as "could not devise a query plan for the given query", "could not find pathkey item to sort", "plan should not reference subplan's variable", or "failed to assign all NestLoopParams to plan nodes". Thanks are due to Andreas Seltenreich and Piotr Stefaniak for fuzz testing that exposed these problems.

- Use fuzzy path cost tiebreaking rule in all supported branches (Tom Lane)

  This change is meant to avoid platform-specific behavior when alternative plan choices have effectively-identical estimated costs.

- Ensure standby promotion trigger files are removed at postmaster startup (Michael Paquier, Fujii Masao)

  This prevents unwanted promotion from occurring if these files appear in a database backup that is used to initialize a new standby server.

- During postmaster shutdown, ensure that per-socket lock files are removed and listen sockets are closed before we remove the `postmaster.pid` file (Tom Lane)

  This avoids race-condition failures if an external script attempts to start a new postmaster as soon as `pg_ctl stop` returns.

- Fix postmaster's handling of a startup-process crash during crash recovery (Tom Lane)

  If, during a crash recovery cycle, the startup process crashes without having restored database consistency, we'd try to launch a new startup process, which typically would just crash again, leading to an infinite loop.

- Do not print a `WARNING` when an autovacuum worker is already gone when we attempt to signal it, and reduce log verbosity for such signals (Tom Lane)

- Prevent autovacuum launcher from sleeping unduly long if the server clock is moved backwards a large amount (Álvaro Herrera)
- Ensure that cleanup of a GIN index's pending-insertions list is interruptable by cancel requests (Jeff Janes)
- Allow all-zeroes pages in GIN indexes to be reused (Heikki Linnakangas)

  Such a page might be left behind after a crash.
- Fix off-by-one error that led to otherwise-harmless warnings about "apparent wraparound" in subtrans/multixact truncation (Thomas Munro)
- Fix misreporting of `CONTINUE` and `MOVE` statement types in PL/pgSQL's error context messages (Pavel Stehule, Tom Lane)
- Fix PL/Perl to handle non-ASCII error message texts correctly (Alex Hunsaker)
- Fix PL/Python crash when returning the string representation of a `record` result (Tom Lane)
- Fix some places in PL/Tcl that neglected to check for failure of `malloc()` calls (Michael Paquier, Álvaro Herrera)
- In `contrib/isn`, fix output of ISBN-13 numbers that begin with 979 (Fabien Coelho)

  EANs beginning with 979 (but not 9790) are considered ISBNs, but they must be printed in the new 13-digit format, not the 10-digit format.
- Improve libpq's handling of out-of-memory conditions (Michael Paquier, Heikki Linnakangas)
- Fix memory leaks and missing out-of-memory checks in ecpg (Michael Paquier)
- Fix psql's code for locale-aware formatting of numeric output (Tom Lane)

  The formatting code invoked by `\pset numericlocale on` did the wrong thing for some uncommon cases such as numbers with an exponent but no decimal point. It could also mangle already-localized output from the `money` data type.
- Prevent crash in psql's `\c` command when there is no current connection (Noah Misch)
- Fix selection of default zlib compression level in pg_dump's directory output format (Andrew Dunstan)
- Ensure that temporary files created during a pg_dump run with tar-format output are not world-readable (Michael Paquier)
- Fix pg_dump and pg_upgrade to support cases where the `postgres` or `template1` database is in a non-default tablespace (Marti Raudsepp, Bruce Momjian)
- Fix pg_dump to handle object privileges sanely when dumping from a server too old to have a particular privilege type (Tom Lane)

  When dumping functions or procedural languages from pre-7.3 servers, pg_dump would produce `GRANT/REVOKE` commands that revoked the owner's grantable privileges and instead granted all privileges to `PUBLIC`. Since the privileges involved are just `USAGE` and `EXECUTE`, this isn't a security problem, but it's certainly a surprising representation of the older systems' behavior. Fix it to leave the default privilege state alone in these cases.

- Fix pg_dump to dump shell types (Tom Lane)

  Shell types (that is, not-yet-fully-defined types) aren't useful for much, but nonetheless pg_dump should dump them.

- Fix assorted minor memory leaks in pg_dump and other client-side programs (Michael Paquier)

- Fix spinlock assembly code for PPC hardware to be compatible with AIX's native assembler (Tom Lane)

  Building with gcc didn't work if gcc had been configured to use the native assembler, which is becoming more common.

- On AIX, test the `-qlonglong` compiler option rather than just assuming it's safe to use (Noah Misch)

- On AIX, use `-Wl,-brtllib` link option to allow symbols to be resolved at runtime (Noah Misch)

  Perl relies on this ability in 5.8.0 and later.

- Avoid use of inline functions when compiling with 32-bit xlc, due to compiler bugs (Noah Misch)

- Use `librt` for `sched_yield()` when necessary, which it is on some Solaris versions (Oskari Saarenmaa)

- Fix Windows `install.bat` script to handle target directory names that contain spaces (Heikki Linnakangas)

- Make the numeric form of the PostgreSQL version number (e.g., `90405`) readily available to extension Makefiles, as a variable named `VERSION_NUM` (Michael Paquier)

- Update time zone data files to tzdata release 2015g for DST law changes in Cayman Islands, Fiji, Moldova, Morocco, Norfolk Island, North Korea, Turkey, and Uruguay. There is a new zone name `America/Fort_Nelson` for the Canadian Northern Rockies.

E.32. Release 9.1.18

Release date: 2015-06-12

This release contains a small number of fixes from 9.1.17. For information about new features in the 9.1 major release, see 第 E.50 节.

E.32.1. Migration to Version 9.1.18

A dump/restore is not required for those running 9.1.X.

However, if you are upgrading from a version earlier than 9.1.16, see 第 E.34 节.

E.32.2. Changes

- Fix rare failure to invalidate relation cache init file (Tom Lane)

  With just the wrong timing of concurrent activity, a `VACUUM FULL` on a system catalog might fail to update the “init file” that's used to avoid cache-loading work for new

sessions. This would result in later sessions being unable to access that catalog at all. This is a very ancient bug, but it's so hard to trigger that no reproducible case had been seen until recently.

- Avoid deadlock between incoming sessions and `CREATE/DROP DATABASE` (Tom Lane)

  A new session starting in a database that is the target of a `DROP DATABASE` command, or is the template for a `CREATE DATABASE` command, could cause the command to wait for five seconds and then fail, even if the new session would have exited before that.

E.33. Release 9.1.17

Release date: 2015-06-04

This release contains a small number of fixes from 9.1.16. For information about new features in the 9.1 major release, see 第 E.50 节.

E.33.1. Migration to Version 9.1.17

A dump/restore is not required for those running 9.1.X.

However, if you are upgrading from a version earlier than 9.1.16, see 第 E.34 节.

E.33.2. Changes

- Avoid failures while `fsync`'ing data directory during crash restart (Abhijit Menon-Sen, Tom Lane)

  In the previous minor releases we added a patch to `fsync` everything in the data directory after a crash. Unfortunately its response to any error condition was to fail, thereby preventing the server from starting up, even when the problem was quite harmless. An example is that an unwritable file in the data directory would prevent restart on some platforms; but it is common to make SSL certificate files unwritable by the server. Revise this behavior so that permissions failures are ignored altogether, and other types of failures are logged but do not prevent continuing.

- Remove configure's check prohibiting linking to a threaded libpython on OpenBSD (Tom Lane)

  The failure this restriction was meant to prevent seems to not be a problem anymore on current OpenBSD versions.

- Allow libpq to use TLS protocol versions beyond v1 (Noah Misch)

  For a long time, libpq was coded so that the only SSL protocol it would allow was TLS v1. Now that newer TLS versions are becoming popular, allow it to negotiate the highest commonly-supported TLS version with the server. (PostgreSQL servers were already capable of such negotiation, so no change is needed on the server side.) This is a back-patch of a change already released in 9.4.0.

E.34. Release 9.1.16

Release date: 2015-05-22

This release contains a variety of fixes from 9.1.15. For information about new features in the 9.1 major release, see 第 E.50 节.

E.34.1. Migration to Version 9.1.16

A dump/restore is not required for those running 9.1.X.

However, if you use `contrib/citext`'s `regexp_matches()` functions, see the changelog entry below about that.

Also, if you are upgrading from a version earlier than 9.1.14, see 第 E.36 节.

E.34.2. Changes

- Avoid possible crash when client disconnects just before the authentication timeout expires (Benkocs Norbert Attila)

  If the timeout interrupt fired partway through the session shutdown sequence, SSL-related state would be freed twice, typically causing a crash and hence denial of service to other sessions. Experimentation shows that an unauthenticated remote attacker could trigger the bug somewhat consistently, hence treat as security issue. (CVE-2015-3165)

- Improve detection of system-call failures (Noah Misch)

  Our replacement implementation of `snprintf()` failed to check for errors reported by the underlying system library calls; the main case that might be missed is out-of-memory situations. In the worst case this might lead to information exposure, due to our code assuming that a buffer had been overwritten when it hadn't been. Also, there were a few places in which security-relevant calls of other system library functions did not check for failure.

  It remains possible that some calls of the `*printf()` family of functions are vulnerable to information disclosure if an out-of-memory error occurs at just the wrong time. We judge the risk to not be large, but will continue analysis in this area. (CVE-2015-3166)

- In `contrib/pgcrypto`, uniformly report decryption failures as "Wrong key or corrupt data" (Noah Misch)

  Previously, some cases of decryption with an incorrect key could report other error message texts. It has been shown that such variance in error reports can aid attackers in recovering keys from other systems. While it's unknown whether `pgcrypto`'s specific behaviors are likewise exploitable, it seems better to avoid the risk by using a one-size-fits-all message. (CVE-2015-3167)

- Fix incorrect declaration of `contrib/citext`'s `regexp_matches()` functions (Tom Lane)

  These functions should return `setof text[]`, like the core functions they are wrappers for; but they were incorrectly declared as returning just `text[]`. This mistake had two results: first, if there was no match you got a scalar null result, whereas what you should get is an empty set (zero rows). Second, the `g` flag was effectively ignored, since you would get only one result array even if there were multiple matches.

  While the latter behavior is clearly a bug, there might be applications depending on the former behavior; therefore the function declarations will not be changed by default until PostgreSQL 9.5. In pre-9.5 branches, the old behavior exists in version 1.

0 of the `citext` extension, while we have provided corrected declarations in version 1.1 (which is not installed by default). To adopt the fix in pre-9.5 branches, execute `ALTER EXTENSION citext UPDATE TO '1.1'` in each database in which `citext` is installed. (You can also "update" back to 1.0 if you need to undo that.) Be aware that either update direction will require dropping and recreating any views or rules that use `citext`'s `regexp_matches()` functions.

- Fix incorrect checking of deferred exclusion constraints after a HOT update (Tom Lane)

  If a new row that potentially violates a deferred exclusion constraint is HOT-updated (that is, no indexed columns change and the row can be stored back onto the same table page) later in the same transaction, the exclusion constraint would be reported as violated when the check finally occurred, even if the row(s) the new row originally conflicted with had been deleted.

- Prevent improper reordering of antijoins (NOT EXISTS joins) versus other outer joins (Tom Lane)

  This oversight in the planner has been observed to cause "could not find RelOptInfo for given relids" errors, but it seems possible that sometimes an incorrect query plan might get past that consistency check and result in silently-wrong query output.

- Fix incorrect matching of subexpressions in outer-join plan nodes (Tom Lane)

  Previously, if textually identical non-strict subexpressions were used both above and below an outer join, the planner might try to re-use the value computed below the join, which would be incorrect because the executor would force the value to NULL in case of an unmatched outer row.

- Fix GEQO planner to cope with failure of its join order heuristic (Tom Lane)

  This oversight has been seen to lead to "failed to join all relations together" errors in queries involving `LATERAL`, and that might happen in other cases as well.

- Fix possible deadlock at startup when `max_prepared_transactions` is too small (Heikki Linnakangas)

- Don't archive useless preallocated WAL files after a timeline switch (Heikki Linnakangas)

- Avoid "cannot GetMultiXactIdMembers() during recovery" error (Álvaro Herrera)

- Recursively `fsync()` the data directory after a crash (Abhijit Menon-Sen, Robert Haas)

  This ensures consistency if another crash occurs shortly later. (The second crash would have to be a system-level crash, not just a database crash, for there to be a problem.)

- Fix autovacuum launcher's possible failure to shut down, if an error occurs after it receives SIGTERM (Álvaro Herrera)

- Cope with unexpected signals in `LockBufferForCleanup()` (Andres Freund)

  This oversight could result in spurious errors about "multiple backends attempting to wait for pincount 1".

- Avoid waiting for WAL flush or synchronous replication during commit of a transaction that was read-only so far as the user is concerned (Andres Freund)

Previously, a delay could occur at commit in transactions that had written WAL due to HOT page pruning, leading to undesirable effects such as sessions getting stuck at startup if all synchronous replicas are down. Sessions have also been observed to get stuck in catchup interrupt processing when using synchronous replication; this will fix that problem as well.

- Fix crash when manipulating hash indexes on temporary tables (Heikki Linnakangas)
- Fix possible failure during hash index bucket split, if other processes are modifying the index concurrently (Tom Lane)
- Check for interrupts while analyzing index expressions (Jeff Janes)

  `ANALYZE` executes index expressions many times; if there are slow functions in such an expression, it's desirable to be able to cancel the `ANALYZE` before that loop finishes.

- Ensure `tableoid` of a foreign table is reported correctly when a `READ COMMITTED` recheck occurs after locking rows in `SELECT FOR UPDATE`, `UPDATE`, or `DELETE` (Etsuro Fujita)
- Add the name of the target server to object description strings for foreign-server user mappings (Álvaro Herrera)
- Recommend setting `include_realm` to 1 when using Kerberos/GSSAPI/SSPI authentication (Stephen Frost)

  Without this, identically-named users from different realms cannot be distinguished. For the moment this is only a documentation change, but it will become the default setting in PostgreSQL 9.5.

- Remove code for matching IPv4 `pg_hba.conf` entries to IPv4-in-IPv6 addresses (Tom Lane)

  This hack was added in 2003 in response to a report that some Linux kernels of the time would report IPv4 connections as having IPv4-in-IPv6 addresses. However, the logic was accidentally broken in 9.0. The lack of any field complaints since then shows that it's not needed anymore. Now we have reports that the broken code causes crashes on some systems, so let's just remove it rather than fix it. (Had we chosen to fix it, that would make for a subtle and potentially security-sensitive change in the effective meaning of IPv4 `pg_hba.conf` entries, which does not seem like a good thing to do in minor releases.)

- Report WAL flush, not insert, position in `IDENTIFY_SYSTEM` replication command (Heikki Linnakangas)

  This avoids a possible startup failure in pg_receivexlog.

- While shutting down service on Windows, periodically send status updates to the Service Control Manager to prevent it from killing the service too soon; and ensure that pg_ctl will wait for shutdown (Krystian Bigaj)
- Reduce risk of network deadlock when using libpq's non-blocking mode (Heikki Linnakangas)

  When sending large volumes of data, it's important to drain the input buffer every so often, in case the server has sent enough response data to cause it to block on output. (A typical scenario is that the server is sending a stream of NOTICE messages during `COPY FROM STDIN`.) This worked properly in the normal blocking mode,

but not so much in non-blocking mode. We've modified libpq to opportunistically drain input when it can, but a full defense against this problem requires application cooperation: the application should watch for socket read-ready as well as write-ready conditions, and be sure to call `PQconsumeInput()` upon read-ready.

- Fix array handling in ecpg (Michael Meskes)
- Fix psql to sanely handle URIs and conninfo strings as the first parameter to `\connect` (David Fetter, Andrew Dunstan, Álvaro Herrera)

  This syntax has been accepted (but undocumented) for a long time, but previously some parameters might be taken from the old connection instead of the given string, which was agreed to be undesirable.
- Suppress incorrect complaints from psql on some platforms that it failed to write `~/.psql_history` at exit (Tom Lane)

  This misbehavior was caused by a workaround for a bug in very old (pre-2006) versions of libedit. We fixed it by removing the workaround, which will cause a similar failure to appear for anyone still using such versions of libedit. Recommendation: upgrade that library, or use libreadline.
- Fix pg_dump's rule for deciding which casts are system-provided casts that should not be dumped (Tom Lane)
- In pg_dump, fix failure to honor `-Z` compression level option together with `-Fd` (Michael Paquier)
- Make pg_dump consider foreign key relationships between extension configuration tables while choosing dump order (Gilles Darold, Michael Paquier, Stephen Frost)

  This oversight could result in producing dumps that fail to reload because foreign key constraints are transiently violated.
- Fix dumping of views that are just `VALUES(...)` but have column aliases (Tom Lane)
- In pg_upgrade, force timeline 1 in the new cluster (Bruce Momjian)

  This change prevents upgrade failures caused by bogus complaints about missing WAL history files.
- In pg_upgrade, check for improperly non-connectable databases before proceeding (Bruce Momjian)
- In pg_upgrade, quote directory paths properly in the generated `delete_old_cluster` script (Bruce Momjian)
- In pg_upgrade, preserve database-level freezing info properly (Bruce Momjian)

  This oversight could cause missing-clog-file errors for tables within the `postgres` and `template1` databases.
- Run pg_upgrade and pg_resetxlog with restricted privileges on Windows, so that they don't fail when run by an administrator (Muhammad Asif Naeem)
- Improve handling of `readdir()` failures when scanning directories in initdb and pg_basebackup (Marco Nenciarini)
- Fix slow sorting algorithm in `contrib/intarray` (Tom Lane)
- Fix compile failure on Sparc V8 machines (Rob Rowan)

- Update time zone data files to tzdata release 2015d for DST law changes in Egypt, Mongolia, and Palestine, plus historical changes in Canada and Chile. Also adopt revised zone abbreviations for the America/Adak zone (HST/HDT not HAST/HADT).

E.35. Release 9.1.15

Release date: 2015-02-05

This release contains a variety of fixes from 9.1.14. For information about new features in the 9.1 major release, see 第 E.50 节.

E.35.1. Migration to Version 9.1.15

A dump/restore is not required for those running 9.1.X.

However, if you are upgrading from a version earlier than 9.1.14, see 第 E.36 节.

E.35.2. Changes

- Fix buffer overruns in `to_char()` (Bruce Momjian)

  When `to_char()` processes a numeric formatting template calling for a large number of digits, PostgreSQL would read past the end of a buffer. When processing a crafted timestamp formatting template, PostgreSQL would write past the end of a buffer. Either case could crash the server. We have not ruled out the possibility of attacks that lead to privilege escalation, though they seem unlikely. (CVE-2015-0241)

- Fix buffer overrun in replacement `*printf()` functions (Tom Lane)

  PostgreSQL includes a replacement implementation of `printf` and related functions. This code will overrun a stack buffer when formatting a floating point number (conversion specifiers `e`, `E`, `f`, `F`, `g` or `G`) with requested precision greater than about 500. This will crash the server, and we have not ruled out the possibility of attacks that lead to privilege escalation. A database user can trigger such a buffer overrun through the `to_char()` SQL function. While that is the only affected core PostgreSQL functionality, extension modules that use printf-family functions may be at risk as well.

  This issue primarily affects PostgreSQL on Windows. PostgreSQL uses the system implementation of these functions where adequate, which it is on other modern platforms. (CVE-2015-0242)

- Fix buffer overruns in `contrib/pgcrypto` (Marko Tiikkaja, Noah Misch)

  Errors in memory size tracking within the `pgcrypto` module permitted stack buffer overruns and improper dependence on the contents of uninitialized memory. The buffer overrun cases can crash the server, and we have not ruled out the possibility of attacks that lead to privilege escalation. (CVE-2015-0243)

- Fix possible loss of frontend/backend protocol synchronization after an error (Heikki Linnakangas)

  If any error occurred while the server was in the middle of reading a protocol message from the client, it could lose synchronization and incorrectly try to interpret part of the message's data as a new protocol message. An attacker able to submit crafted binary data within a command parameter might succeed in injecting his own SQL commands this way. Statement timeout and query cancellation are the

most likely sources of errors triggering this scenario. Particularly vulnerable are applications that use a timeout and also submit arbitrary user-crafted data as binary query parameters. Disabling statement timeout will reduce, but not eliminate, the risk of exploit. Our thanks to Emil Lenngren for reporting this issue. (CVE-2015-0244)

- Fix information leak via constraint-violation error messages (Stephen Frost)

  Some server error messages show the values of columns that violate a constraint, such as a unique constraint. If the user does not have `SELECT` privilege on all columns of the table, this could mean exposing values that the user should not be able to see. Adjust the code so that values are displayed only when they came from the SQL command or could be selected by the user. (CVE-2014-8161)

- Lock down regression testing's temporary installations on Windows (Noah Misch)

  Use SSPI authentication to allow connections only from the OS user who launched the test suite. This closes on Windows the same vulnerability previously closed on other platforms, namely that other users might be able to connect to the test postmaster. (CVE-2014-0067)

- Avoid possible data corruption if `ALTER DATABASE SET TABLESPACE` is used to move a database to a new tablespace and then shortly later move it back to its original tablespace (Tom Lane)

- Avoid corrupting tables when `ANALYZE` inside a transaction is rolled back (Andres Freund, Tom Lane, Michael Paquier)

  If the failing transaction had earlier removed the last index, rule, or trigger from the table, the table would be left in a corrupted state with the relevant `pg_class` flags not set though they should be.

- Ensure that unlogged tables are copied correctly during `CREATE DATABASE` or `ALTER DATABASE SET TABLESPACE` (Pavan Deolasee, Andres Freund)

- Fix `DROP`'s dependency searching to correctly handle the case where a table column is recursively visited before its table (Petr Jelinek, Tom Lane)

  This case is only known to arise when an extension creates both a datatype and a table using that datatype. The faulty code might refuse a `DROP EXTENSION` unless `CASCADE` is specified, which should not be required.

- Fix use-of-already-freed-memory problem in EvalPlanQual processing (Tom Lane)

  In `READ COMMITTED` mode, queries that lock or update recently-updated rows could crash as a result of this bug.

- Fix planning of `SELECT FOR UPDATE` when using a partial index on a child table (Kyotaro Horiguchi)

  In `READ COMMITTED` mode, `SELECT FOR UPDATE` must also recheck the partial index's `WHERE` condition when rechecking a recently-updated row to see if it still satisfies the query's `WHERE` condition. This requirement was missed if the index belonged to an inheritance child table, so that it was possible to incorrectly return rows that no longer satisfy the query condition.

- Fix corner case wherein `SELECT FOR UPDATE` could return a row twice, and possibly miss returning other rows (Tom Lane)

  In `READ COMMITTED` mode, a `SELECT FOR UPDATE` that is scanning an inheritance tree could incorrectly return a row from a prior child table instead of the one it should return from a later child table.

- Reject duplicate column names in the referenced-columns list of a `FOREIGN KEY` declaration (David Rowley)

  This restriction is per SQL standard. Previously we did not reject the case explicitly, but later on the code would fail with bizarre-looking errors.

- Fix bugs in raising a `numeric` value to a large integral power (Tom Lane)

  The previous code could get a wrong answer, or consume excessive amounts of time and memory before realizing that the answer must overflow.

- In `numeric_recv()`, truncate away any fractional digits that would be hidden according to the value's `dscale` field (Tom Lane)

  A `numeric` value's display scale (`dscale`) should never be less than the number of nonzero fractional digits; but apparently there's at least one broken client application that transmits binary `numeric` values in which that's true. This leads to strange behavior since the extra digits are taken into account by arithmetic operations even though they aren't printed. The least risky fix seems to be to truncate away such "hidden" digits on receipt, so that the value is indeed what it prints as.

- Reject out-of-range numeric timezone specifications (Tom Lane)

  Simple numeric timezone specifications exceeding +/- 168 hours (one week) would be accepted, but could then cause null-pointer dereference crashes in certain operations. There's no use-case for such large UTC offsets, so reject them.

- Fix bugs in `tsquery @> tsquery` operator (Heikki Linnakangas)

  Two different terms would be considered to match if they had the same CRC. Also, if the second operand had more terms than the first, it would be assumed not to be contained in the first; which is wrong since it might contain duplicate terms.

- Improve ispell dictionary's defenses against bad affix files (Tom Lane)

- Allow more than 64K phrases in a thesaurus dictionary (David Boutin)

  The previous coding could crash on an oversize dictionary, so this was deemed a back-patchable bug fix rather than a feature addition.

- Fix namespace handling in `xpath()` (Ali Akbar)

  Previously, the `xml` value resulting from an `xpath()` call would not have namespace declarations if the namespace declarations were attached to an ancestor element in the input `xml` value, rather than to the specific element being returned. Propagate the ancestral declaration so that the result is correct when considered in isolation.

- Fix planner problems with nested append relations, such as inherited tables within `UNION ALL` subqueries (Tom Lane)

- Fail cleanly when a GiST index tuple doesn't fit on a page, rather than going into infinite recursion (Andrew Gierth)

- Exempt tables that have per-table `cost_limit` and/or `cost_delay` settings from autovacuum's global cost balancing rules (Álvaro Herrera)

  The previous behavior resulted in basically ignoring these per-table settings, which was unintended. Now, a table having such settings will be vacuumed using those settings, independently of what is going on in other autovacuum workers. This may result in heavier total I/O load than before, so such settings should be re-examined for sanity.

- Avoid wholesale autovacuuming when autovacuum is nominally off (Tom Lane)

  Even when autovacuum is nominally off, we will still launch autovacuum worker processes to vacuum tables that are at risk of XID wraparound. However, such a worker process then proceeded to vacuum all tables in the target database, if they met the usual thresholds for autovacuuming. This is at best pretty unexpected; at worst it delays response to the wraparound threat. Fix it so that if autovacuum is turned off, workers only do anti-wraparound vacuums and not any other work.

- During crash recovery, ensure that unlogged relations are rewritten as empty and are synced to disk before recovery is considered complete (Abhijit Menon-Sen, Andres Freund)

  This prevents scenarios in which unlogged relations might contain garbage data following database crash recovery.

- Fix race condition between hot standby queries and replaying a full-page image (Heikki Linnakangas)

  This mistake could result in transient errors in queries being executed in hot standby.

- Fix several cases where recovery logic improperly ignored WAL records for `COMMIT/ABORT PREPARED` (Heikki Linnakangas)

  The most notable oversight was that `recovery_target_xid` could not be used to stop at a two-phase commit.

- Avoid creating unnecessary `.ready` marker files for timeline history files (Fujii Masao)

- Fix possible null pointer dereference when an empty prepared statement is used and the `log_statement` setting is `mod` or `ddl` (Fujii Masao)

- Change “pgstat wait timeout” warning message to be LOG level, and rephrase it to be more understandable (Tom Lane)

  This message was originally thought to be essentially a can't-happen case, but it occurs often enough on our slower buildfarm members to be a nuisance. Reduce it to LOG level, and expend a bit more effort on the wording: it now reads “using stale statistics instead of current ones because stats collector is not responding”.

- Fix SPARC spinlock implementation to ensure correctness if the CPU is being run in a non-TSO coherency mode, as some non-Solaris kernels do (Andres Freund)

- Warn if macOS's `setlocale()` starts an unwanted extra thread inside the postmaster (Noah Misch)

- Fix processing of repeated `dbname` parameters in `PQconnectdbParams()` (Alex Shulgin)

  Unexpected behavior ensued if the first occurrence of `dbname` contained a connection string or URI to be expanded.

- Ensure that libpq reports a suitable error message on unexpected socket EOF (Marko Tiikkaja, Tom Lane)

  Depending on kernel behavior, libpq might return an empty error string rather than something useful when the server unexpectedly closed the socket.

- Clear any old error message during `PQreset()` (Heikki Linnakangas)

If `PQreset()` is called repeatedly, and the connection cannot be re-established, error messages from the failed connection attempts kept accumulating in the `PGconn`'s error string.

- Properly handle out-of-memory conditions while parsing connection options in libpq (Alex Shulgin, Heikki Linnakangas)
- Fix array overrun in ecpg's version of `ParseDateTime()` (Michael Paquier)
- In initdb, give a clearer error message if a password file is specified but is empty (Mats Erik Andersson)
- Fix psql's `\s` command to work nicely with libedit, and add pager support (Stepan Rutz, Tom Lane)

  When using libedit rather than readline, `\s` printed the command history in a fairly unreadable encoded format, and on recent libedit versions might fail altogether. Fix that by printing the history ourselves rather than having the library do it. A pleasant side-effect is that the pager is used if appropriate.

  This patch also fixes a bug that caused newline encoding to be applied inconsistently when saving the command history with libedit. Multiline history entries written by older psql versions will be read cleanly with this patch, but perhaps not vice versa, depending on the exact libedit versions involved.
- Improve consistency of parsing of psql's special variables (Tom Lane)

  Allow variant spellings of `on` and `off` (such as `1`/`0`) for `ECHO_HIDDEN` and `ON_ERROR_ROLLBACK`. Report a warning for unrecognized values for `COMP_KEYWORD_CASE`, `ECHO`, `ECHO_HIDDEN`, `HISTCONTROL`, `ON_ERROR_ROLLBACK`, and `VERBOSITY`. Recognize all values for all these variables case-insensitively; previously there was a mishmash of case-sensitive and case-insensitive behaviors.
- Fix psql's expanded-mode display to work consistently when using `border` = 3 and `linestyle` = `ascii` or `unicode` (Stephen Frost)
- Improve performance of pg_dump when the database contains many instances of multiple dependency paths between the same two objects (Tom Lane)
- Fix possible deadlock during parallel restore of a schema-only dump (Robert Haas, Tom Lane)
- Fix core dump in `pg_dump --binary-upgrade` on zero-column composite type (Rushabh Lathia)
- Prevent WAL files created by `pg_basebackup -x/-X` from being archived again when the standby is promoted (Andres Freund)
- Fix upgrade-from-unpackaged script for `contrib/citext` (Tom Lane)
- Fix block number checking in `contrib/pageinspect`'s `get_raw_page()` (Tom Lane)

  The incorrect checking logic could prevent access to some pages in non-main relation forks.
- Fix `contrib/pgcrypto`'s `pgp_sym_decrypt()` to not fail on messages whose length is 6 less than a power of 2 (Marko Tiikkaja)
- Fix file descriptor leak in `contrib/pg_test_fsync` (Jeff Janes)

This could cause failure to remove temporary files on Windows.

- Handle unexpected query results, especially NULLs, safely in `contrib/tablefunc` 's `connectby()` (Michael Paquier)

  `connectby()` previously crashed if it encountered a NULL key value. It now prints that row but doesn't recurse further.

- Avoid a possible crash in `contrib/xml2`'s `xslt_process()` (Mark Simonetti)

  libxslt seems to have an undocumented dependency on the order in which resources are freed; reorder our calls to avoid a crash.

- Mark some `contrib` I/O functions with correct volatility properties (Tom Lane)

  The previous over-conservative marking was immaterial in normal use, but could cause optimization problems or rejection of valid index expression definitions. Since the consequences are not large, we've just adjusted the function definitions in the extension modules' scripts, without changing version numbers.

- Numerous cleanups of warnings from Coverity static code analyzer (Andres Freund, Tatsuo Ishii, Marko Kreen, Tom Lane, Michael Paquier)

  These changes are mostly cosmetic but in some cases fix corner-case bugs, for example a crash rather than a proper error report after an out-of-memory failure. None are believed to represent security issues.

- Detect incompatible OpenLDAP versions during build (Noah Misch)

  With OpenLDAP versions 2.4.24 through 2.4.31, inclusive, PostgreSQL backends can crash at exit. Raise a warning during configure based on the compile-time OpenLDAP version number, and test the crashing scenario in the `contrib/dblink` regression test.

- In non-MSVC Windows builds, ensure `libpq.dll` is installed with execute permissions (Noah Misch)

- Make pg_regress remove any temporary installation it created upon successful exit (Tom Lane)

  This results in a very substantial reduction in disk space usage during `make check-world`, since that sequence involves creation of numerous temporary installations.

- Support time zone abbreviations that change UTC offset from time to time (Tom Lane)

  Previously, PostgreSQL assumed that the UTC offset associated with a time zone abbreviation (such as `EST`) never changes in the usage of any particular locale. However this assumption fails in the real world, so introduce the ability for a zone abbreviation to represent a UTC offset that sometimes changes. Update the zone abbreviation definition files to make use of this feature in timezone locales that have changed the UTC offset of their abbreviations since 1970 (according to the IANA timezone database). In such timezones, PostgreSQL will now associate the correct UTC offset with the abbreviation depending on the given date.

- Update time zone abbreviations lists (Tom Lane)

  Add CST (China Standard Time) to our lists. Remove references to ADT as “Arabia Daylight Time”, an abbreviation that's been out of use since 2007; therefore, claiming there is a conflict with “Atlantic Daylight Time” doesn't seem especially

helpful. Fix entirely incorrect GMT offsets for CKT (Cook Islands), FJT, and FJST (Fiji); we didn't even have them on the proper side of the date line.

- Update time zone data files to tzdata release 2015a.

  The IANA timezone database has adopted abbreviations of the form `AxST`/`AxDT` for all Australian time zones, reflecting what they believe to be current majority practice Down Under. These names do not conflict with usage elsewhere (other than ACST for Acre Summer Time, which has been in disuse since 1994). Accordingly, adopt these names into our "Default" timezone abbreviation set. The "Australia" abbreviation set now contains only CST, EAST, EST, SAST, SAT, and WST, all of which are thought to be mostly historical usage. Note that SAST has also been changed to be South Africa Standard Time in the "Default" abbreviation set.

  Also, add zone abbreviations SRET (Asia/Srednekolymsk) and XJT (Asia/Urumqi), and use WSST/WSDT for western Samoa. Also, there were DST law changes in Chile, Mexico, the Turks & Caicos Islands (America/Grand_Turk), and Fiji. There is a new zone Pacific/Bougainville for portions of Papua New Guinea. Also, numerous corrections for historical (pre-1970) time zone data.

E.36. Release 9.1.14

Release date: 2014-07-24

This release contains a variety of fixes from 9.1.13. For information about new features in the 9.1 major release, see 第 E.50 节.

E.36.1. Migration to Version 9.1.14

A dump/restore is not required for those running 9.1.X.

However, this release corrects an index corruption problem in some GiST indexes. See the first changelog entry below to find out whether your installation has been affected and what steps you should take if so.

Also, if you are upgrading from a version earlier than 9.1.11, see 第 E.39 节.

E.36.2. Changes

- Correctly initialize padding bytes in `contrib/btree_gist` indexes on `bit` columns (Heikki Linnakangas)

  This error could result in incorrect query results due to values that should compare equal not being seen as equal. Users with GiST indexes on `bit` or `bit varying` columns should `REINDEX` those indexes after installing this update.

- Protect against torn pages when deleting GIN list pages (Heikki Linnakangas)

  This fix prevents possible index corruption if a system crash occurs while the page update is being written to disk.

- Don't clear the right-link of a GiST index page while replaying updates from WAL (Heikki Linnakangas)

  This error could lead to transiently wrong answers from GiST index scans performed in Hot Standby.

- Fix feedback status when hot_standby_feedback is turned off on-the-fly (Simon Riggs)

- Fix possibly-incorrect cache invalidation during nested calls to `ReceiveSharedInvalidMessages` (Andres Freund)
- Fix "could not find pathkey item to sort" planner failures with `UNION ALL` over subqueries reading from tables with inheritance children (Tom Lane)
- Don't assume a subquery's output is unique if there's a set-returning function in its targetlist (David Rowley)

  This oversight could lead to misoptimization of constructs like `WHERE x IN (SELECT y, generate_series(1,10) FROM t GROUP BY y)`.
- Fix failure to detoast fields in composite elements of structured types (Tom Lane)

  This corrects cases where TOAST pointers could be copied into other tables without being dereferenced. If the original data is later deleted, it would lead to errors like "missing chunk number 0 for toast value ..." when the now-dangling pointer is used.
- Fix "record type has not been registered" failures with whole-row references to the output of Append plan nodes (Tom Lane)
- Fix possible crash when invoking a user-defined function while rewinding a cursor (Tom Lane)
- Fix query-lifespan memory leak while evaluating the arguments for a function in `FROM` (Tom Lane)
- Fix session-lifespan memory leaks in regular-expression processing (Tom Lane, Arthur O'Dwyer, Greg Stark)
- Fix data encoding error in `hungarian.stop` (Tom Lane)
- Prevent foreign tables from being created with OIDS when default_with_oids is true (Etsuro Fujita)
- Fix liveness checks for rows that were inserted in the current transaction and then deleted by a now-rolled-back subtransaction (Andres Freund)

  This could cause problems (at least spurious warnings, and at worst an infinite loop) if `CREATE INDEX` or `CLUSTER` were done later in the same transaction.
- Clear `pg_stat_activity.xact_start` during `PREPARE TRANSACTION` (Andres Freund)

  After the `PREPARE`, the originating session is no longer in a transaction, so it should not continue to display a transaction start time.
- Fix `REASSIGN OWNED` to not fail for text search objects (Álvaro Herrera)
- Block signals during postmaster startup (Tom Lane)

  This ensures that the postmaster will properly clean up after itself if, for example, it receives SIGINT while still starting up.
- Fix client host name lookup when processing `pg_hba.conf` entries that specify host names instead of IP addresses (Tom Lane)

  Ensure that reverse-DNS lookup failures are reported, instead of just silently not matching such entries. Also ensure that we make only one reverse-DNS lookup

attempt per connection, not one per host name entry, which is what previously happened if the lookup attempts failed.

- Secure Unix-domain sockets of temporary postmasters started during `make check` (Noah Misch)

  Any local user able to access the socket file could connect as the server's bootstrap superuser, then proceed to execute arbitrary code as the operating-system user running the test, as we previously noted in CVE-2014-0067. This change defends against that risk by placing the server's socket in a temporary, mode 0700 subdirectory of `/tmp`. The hazard remains however on platforms where Unix sockets are not supported, notably Windows, because then the temporary postmaster must accept local TCP connections.

  A useful side effect of this change is to simplify `make check` testing in builds that override `DEFAULT_PGSOCKET_DIR`. Popular non-default values like `/var/run/postgresql` are often not writable by the build user, requiring workarounds that will no longer be necessary.

- Fix tablespace creation WAL replay to work on Windows (MauMau)

- Fix detection of socket creation failures on Windows (Bruce Momjian)

- On Windows, allow new sessions to absorb values of PGC_BACKEND parameters (such as log_connections) from the configuration file (Amit Kapila)

  Previously, if such a parameter were changed in the file post-startup, the change would have no effect.

- Properly quote executable path names on Windows (Nikhil Deshpande)

  This oversight could cause initdb and pg_upgrade to fail on Windows, if the installation path contained both spaces and `@` signs.

- Fix linking of libpython on macOS (Tom Lane)

  The method we previously used can fail with the Python library supplied by Xcode 5.0 and later.

- Avoid buffer bloat in libpq when the server consistently sends data faster than the client can absorb it (Shin-ichi Morita, Tom Lane)

  libpq could be coerced into enlarging its input buffer until it runs out of memory (which would be reported misleadingly as "lost synchronization with server"). Under ordinary circumstances it's quite far-fetched that data could be continuously transmitted more quickly than the `recv()` loop can absorb it, but this has been observed when the client is artificially slowed by scheduler constraints.

- Ensure that LDAP lookup attempts in libpq time out as intended (Laurenz Albe)

- Fix ecpg to do the right thing when an array of `char *` is the target for a FETCH statement returning more than one row, as well as some other array-handling fixes (Ashutosh Bapat)

- Fix pg_restore's processing of old-style large object comments (Tom Lane)

  A direct-to-database restore from an archive file generated by a pre-9.0 version of pg_dump would usually fail if the archive contained more than a few comments for large objects.

- In `contrib/pgcrypto` functions, ensure sensitive information is cleared from stack variables before returning (Marko Kreen)

- In `contrib/uuid-ossp`, cache the state of the OSSP UUID library across calls (Tom Lane)

  This improves the efficiency of UUID generation and reduces the amount of entropy drawn from `/dev/urandom`, on platforms that have that.

- Update time zone data files to tzdata release 2014e for DST law changes in Crimea, Egypt, and Morocco.

E.37. Release 9.1.13

Release date: 2014-03-20

This release contains a variety of fixes from 9.1.12. For information about new features in the 9.1 major release, see 第 E.50 节.

E.37.1. Migration to Version 9.1.13

A dump/restore is not required for those running 9.1.X.

However, if you are upgrading from a version earlier than 9.1.11, see 第 E.39 节.

E.37.2. Changes

- Restore GIN metapages unconditionally to avoid torn-page risk (Heikki Linnakangas)

  Although this oversight could theoretically result in a corrupted index, it is unlikely to have caused any problems in practice, since the active part of a GIN metapage is smaller than a standard 512-byte disk sector.

- Avoid race condition in checking transaction commit status during receipt of a `NOTIFY` message (Marko Tiikkaja)

  This prevents a scenario wherein a sufficiently fast client might respond to a notification before database updates made by the notifier have become visible to the recipient.

- Allow regular-expression operators to be terminated early by query cancel requests (Tom Lane)

  This prevents scenarios wherein a pathological regular expression could lock up a server process uninterruptibly for a long time.

- Remove incorrect code that tried to allow `OVERLAPS` with single-element row arguments (Joshua Yanovski)

  This code never worked correctly, and since the case is neither specified by the SQL standard nor documented, it seemed better to remove it than fix it.

- Avoid getting more than `AccessShareLock` when de-parsing a rule or view (Dean Rasheed)

  This oversight resulted in pg_dump unexpectedly acquiring `RowExclusiveLock` locks on tables mentioned as the targets of `INSERT`/`UPDATE`/`DELETE` commands in rules. While usually harmless, that could interfere with concurrent transactions that tried to acquire, for example, `ShareLock` on those tables.

- Improve performance of index endpoint probes during planning (Tom Lane)

This change fixes a significant performance problem that occurred when there were many not-yet-committed rows at the end of the index, which is a common situation for indexes on sequentially-assigned values such as timestamps or sequence-generated identifiers.

- Fix walsender's failure to shut down cleanly when client is pg_receivexlog (Fujii Masao)
- Fix test to see if hot standby connections can be allowed immediately after a crash (Heikki Linnakangas)
- Prevent interrupts while reporting non-`ERROR` messages (Tom Lane)

  This guards against rare server-process freezeups due to recursive entry to `syslog()`, and perhaps other related problems.
- Fix memory leak in PL/Perl when returning a composite result, including multiple-OUT-parameter cases (Alex Hunsaker)
- Prevent intermittent "could not reserve shared memory region" failures on recent Windows versions (MauMau)
- Update time zone data files to tzdata release 2014a for DST law changes in Fiji and Turkey, plus historical changes in Israel and Ukraine.

E.38. Release 9.1.12

Release date: 2014-02-20

This release contains a variety of fixes from 9.1.11. For information about new features in the 9.1 major release, see 第 E.50 节.

E.38.1. Migration to Version 9.1.12

A dump/restore is not required for those running 9.1.X.

However, if you are upgrading from a version earlier than 9.1.11, see 第 E.39 节.

E.38.2. Changes

- Shore up `GRANT ... WITH ADMIN OPTION` restrictions (Noah Misch)

  Granting a role without `ADMIN OPTION` is supposed to prevent the grantee from adding or removing members from the granted role, but this restriction was easily bypassed by doing `SET ROLE` first. The security impact is mostly that a role member can revoke the access of others, contrary to the wishes of his grantor. Unapproved role member additions are a lesser concern, since an uncooperative role member could provide most of his rights to others anyway by creating views or `SECURITY DEFINER` functions. (CVE-2014-0060)
- Prevent privilege escalation via manual calls to PL validator functions (Andres Freund)

  The primary role of PL validator functions is to be called implicitly during `CREATE FUNCTION`, but they are also normal SQL functions that a user can call explicitly. Calling a validator on a function actually written in some other language was not checked for and could be exploited for privilege-escalation purposes. The fix

involves adding a call to a privilege-checking function in each validator function. Non-core procedural languages will also need to make this change to their own validator functions, if any. (CVE-2014-0061)

- Avoid multiple name lookups during table and index DDL (Robert Haas, Andres Freund)

  If the name lookups come to different conclusions due to concurrent activity, we might perform some parts of the DDL on a different table than other parts. At least in the case of `CREATE INDEX`, this can be used to cause the permissions checks to be performed against a different table than the index creation, allowing for a privilege escalation attack. (CVE-2014-0062)

- Prevent buffer overrun with long datetime strings (Noah Misch)

  The `MAXDATELEN` constant was too small for the longest possible value of type `interval`, allowing a buffer overrun in `interval_out()`. Although the datetime input functions were more careful about avoiding buffer overrun, the limit was short enough to cause them to reject some valid inputs, such as input containing a very long timezone name. The ecpg library contained these vulnerabilities along with some of its own. (CVE-2014-0063)

- Prevent buffer overrun due to integer overflow in size calculations (Noah Misch, Heikki Linnakangas)

  Several functions, mostly type input functions, calculated an allocation size without checking for overflow. If overflow did occur, a too-small buffer would be allocated and then written past. (CVE-2014-0064)

- Prevent overruns of fixed-size buffers (Peter Eisentraut, Jozef Mlich)

  Use `strlcpy()` and related functions to provide a clear guarantee that fixed-size buffers are not overrun. Unlike the preceding items, it is unclear whether these cases really represent live issues, since in most cases there appear to be previous constraints on the size of the input string. Nonetheless it seems prudent to silence all Coverity warnings of this type. (CVE-2014-0065)

- Avoid crashing if `crypt()` returns NULL (Honza Horak, Bruce Momjian)

  There are relatively few scenarios in which `crypt()` could return NULL, but `contrib/chkpass` would crash if it did. One practical case in which this could be an issue is if libc is configured to refuse to execute unapproved hashing algorithms (e.g., “FIPS mode”). (CVE-2014-0066)

- Document risks of `make check` in the regression testing instructions (Noah Misch, Tom Lane)

  Since the temporary server started by `make check` uses “trust” authentication, another user on the same machine could connect to it as database superuser, and then potentially exploit the privileges of the operating-system user who started the tests. A future release will probably incorporate changes in the testing procedure to prevent this risk, but some public discussion is needed first. So for the moment, just warn people against using `make check` when there are untrusted users on the same machine. (CVE-2014-0067)

- Fix possible mis-replay of WAL records when some segments of a relation aren't full size (Greg Stark, Tom Lane)

  The WAL update could be applied to the wrong page, potentially many pages past where it should have been. Aside from corrupting data, this error has been observed

to result in significant “bloat” of standby servers compared to their masters, due to updates being applied far beyond where the end-of-file should have been. This failure mode does not appear to be a significant risk during crash recovery, only when initially synchronizing a standby created from a base backup taken from a quickly-changing master.

- Fix bug in determining when recovery has reached consistency (Tomonari Katsumata, Heikki Linnakangas)

  In some cases WAL replay would mistakenly conclude that the database was already consistent at the start of replay, thus possibly allowing hot-standby queries before the database was really consistent. Other symptoms such as “PANIC: WAL contains references to invalid pages” were also possible.

- Fix improper locking of btree index pages while replaying a `VACUUM` operation in hot-standby mode (Andres Freund, Heikki Linnakangas, Tom Lane)

  This error could result in “PANIC: WAL contains references to invalid pages” failures.

- Ensure that insertions into non-leaf GIN index pages write a full-page WAL record when appropriate (Heikki Linnakangas)

  The previous coding risked index corruption in the event of a partial-page write during a system crash.

- When `pause_at_recovery_target` and `recovery_target_inclusive` are both set, ensure the target record is applied before pausing, not after (Heikki Linnakangas)

- Fix race conditions during server process exit (Robert Haas)

  Ensure that signal handlers don't attempt to use the process's `MyProc` pointer after it's no longer valid.

- Fix race conditions in walsender shutdown logic and walreceiver SIGHUP signal handler (Tom Lane)

- Fix unsafe references to `errno` within error reporting logic (Christian Kruse)

  This would typically lead to odd behaviors such as missing or inappropriate `HINT` fields.

- Fix possible crashes from using `ereport()` too early during server startup (Tom Lane)

  The principal case we've seen in the field is a crash if the server is started in a directory it doesn't have permission to read.

- Clear retry flags properly in OpenSSL socket write function (Alexander Kukushkin)

  This omission could result in a server lockup after unexpected loss of an SSL-encrypted connection.

- Fix length checking for Unicode identifiers (`U&"..."` syntax) containing escapes (Tom Lane)

  A spurious truncation warning would be printed for such identifiers if the escaped form of the identifier was too long, but the identifier actually didn't need truncation after de-escaping.

- Allow keywords that are type names to be used in lists of roles (Stephen Frost)

A previous patch allowed such keywords to be used without quoting in places such as role identifiers; but it missed cases where a list of role identifiers was permitted, such as `DROP ROLE`.

- Fix parser crash for `EXISTS(SELECT * FROM zero_column_table)` (Tom Lane)
- Fix possible crash due to invalid plan for nested sub-selects, such as `WHERE (... x IN (SELECT ...) ...) IN (SELECT ...)` (Tom Lane)
- Ensure that `ANALYZE` creates statistics for a table column even when all the values in it are "too wide" (Tom Lane)

  `ANALYZE` intentionally omits very wide values from its histogram and most-common-values calculations, but it neglected to do something sane in the case that all the sampled entries are too wide.

- In `ALTER TABLE ... SET TABLESPACE`, allow the database's default tablespace to be used without a permissions check (Stephen Frost)

  `CREATE TABLE` has always allowed such usage, but `ALTER TABLE` didn't get the memo.

- Fix "cannot accept a set" error when some arms of a `CASE` return a set and others don't (Tom Lane)
- Fix checks for all-zero client addresses in pgstat functions (Kevin Grittner)
- Fix possible misclassification of multibyte characters by the text search parser (Tom Lane)

  Non-ASCII characters could be misclassified when using C locale with a multibyte encoding. On Cygwin, non-C locales could fail as well.

- Fix possible misbehavior in `plainto_tsquery()` (Heikki Linnakangas)

  Use `memmove()` not `memcpy()` for copying overlapping memory regions. There have been no field reports of this actually causing trouble, but it's certainly risky.

- Fix placement of permissions checks in `pg_start_backup()` and `pg_stop_backup()` (Andres Freund, Magnus Hagander)

  The previous coding might attempt to do catalog access when it shouldn't.

- Accept `SHIFT_JIS` as an encoding name for locale checking purposes (Tatsuo Ishii)
- Fix misbehavior of `PQhost()` on Windows (Fujii Masao)

  It should return `localhost` if no host has been specified.

- Improve error handling in libpq and psql for failures during `COPY TO STDOUT/FROM STDIN` (Tom Lane)

  In particular this fixes an infinite loop that could occur in 9.2 and up if the server connection was lost during `COPY FROM STDIN`. Variants of that scenario might be possible in older versions, or with other client applications.

- Fix possible incorrect printing of filenames in pg_basebackup's verbose mode (Magnus Hagander)
- Avoid including tablespaces inside PGDATA twice in base backups (Dimitri Fontaine, Magnus Hagander)

- Fix misaligned descriptors in ecpg (MauMau)
- In ecpg, handle lack of a hostname in the connection parameters properly (Michael Meskes)
- Fix performance regression in `contrib/dblink` connection startup (Joe Conway)

  Avoid an unnecessary round trip when client and server encodings match.
- In `contrib/isn`, fix incorrect calculation of the check digit for ISMN values (Fabien Coelho)
- Ensure client-code-only installation procedure works as documented (Peter Eisentraut)
- In Mingw and Cygwin builds, install the libpq DLL in the `bin` directory (Andrew Dunstan)

  This duplicates what the MSVC build has long done. It should fix problems with programs like psql failing to start because they can't find the DLL.
- Avoid using the deprecated `dllwrap` tool in Cygwin builds (Marco Atzeri)
- Don't generate plain-text `HISTORY` and `src/test/regress/README` files anymore (Tom Lane)

  These text files duplicated the main HTML and PDF documentation formats. The trouble involved in maintaining them greatly outweighs the likely audience for plain-text format. Distribution tarballs will still contain files by these names, but they'll just be stubs directing the reader to consult the main documentation. The plain-text `INSTALL` file will still be maintained, as there is arguably a use-case for that.
- Update time zone data files to tzdata release 2013i for DST law changes in Jordan and historical changes in Cuba.

  In addition, the zones `Asia/Riyadh87`, `Asia/Riyadh88`, and `Asia/Riyadh89` have been removed, as they are no longer maintained by IANA, and never represented actual civil timekeeping practice.

E.39. Release 9.1.11

Release date: 2013-12-05

This release contains a variety of fixes from 9.1.10. For information about new features in the 9.1 major release, see 第 E.50 节.

E.39.1. Migration to Version 9.1.11

A dump/restore is not required for those running 9.1.X.

However, this release corrects a number of potential data corruption issues. See the first two changelog entries below to find out whether your installation has been affected and what steps you can take if so.

Also, if you are upgrading from a version earlier than 9.1.9, see 第 E.41 节.

E.39.2. Changes

- Fix `VACUUM`'s tests to see whether it can update `relfrozenxid` (Andres Freund)

In some cases `VACUUM` (either manual or autovacuum) could incorrectly advance a table's `relfrozenxid` value, allowing tuples to escape freezing, causing those rows to become invisible once 2^31 transactions have elapsed. The probability of data loss is fairly low since multiple incorrect advancements would need to happen before actual loss occurs, but it's not zero. Users upgrading from releases 9.0.4 or 8.4.8 or earlier are not affected, but all later versions contain the bug.

The issue can be ameliorated by, after upgrading, vacuuming all tables in all databases while having `vacuum_freeze_table_age` set to zero. This will fix any latent corruption but will not be able to fix all pre-existing data errors. However, an installation can be presumed safe after performing this vacuuming if it has executed fewer than 2^31 update transactions in its lifetime (check this with `SELECT txid_current() < 2^31`).

- Fix initialization of `pg_clog` and `pg_subtrans` during hot standby startup (Andres Freund, Heikki Linnakangas)

  This bug can cause data loss on standby servers at the moment they start to accept hot-standby queries, by marking committed transactions as uncommitted. The likelihood of such corruption is small unless, at the time of standby startup, the primary server has executed many updating transactions since its last checkpoint. Symptoms include missing rows, rows that should have been deleted being still visible, and obsolete versions of updated rows being still visible alongside their newer versions.

  This bug was introduced in versions 9.3.0, 9.2.5, 9.1.10, and 9.0.14. Standby servers that have only been running earlier releases are not at risk. It's recommended that standby servers that have ever run any of the buggy releases be re-cloned from the primary (e.g., with a new base backup) after upgrading.

- Truncate `pg_multixact` contents during WAL replay (Andres Freund)

  This avoids ever-increasing disk space consumption in standby servers.

- Fix race condition in GIN index posting tree page deletion (Heikki Linnakangas)

  This could lead to transient wrong answers or query failures.

- Avoid flattening a subquery whose `SELECT` list contains a volatile function wrapped inside a sub-`SELECT` (Tom Lane)

  This avoids unexpected results due to extra evaluations of the volatile function.

- Fix planner's processing of non-simple-variable subquery outputs nested within outer joins (Tom Lane)

  This error could lead to incorrect plans for queries involving multiple levels of subqueries within `JOIN` syntax.

- Fix incorrect generation of optimized MIN()/MAX() plans for inheritance trees (Tom Lane)

  The planner could fail in cases where the MIN()/MAX() argument was an expression rather than a simple variable.

- Fix premature deletion of temporary files (Andres Freund)

- Fix possible read past end of memory in rule printing (Peter Eisentraut)

- Fix array slicing of `int2vector` and `oidvector` values (Tom Lane)

Expressions of this kind are now implicitly promoted to regular `int2` or `oid` arrays.

- Fix incorrect behaviors when using a SQL-standard, simple GMT offset timezone (Tom Lane)

  In some cases, the system would use the simple GMT offset value when it should have used the regular timezone setting that had prevailed before the simple offset was selected. This change also causes the `timeofday` function to honor the simple GMT offset zone.

- Prevent possible misbehavior when logging translations of Windows error codes (Tom Lane)

- Properly quote generated command lines in pg_ctl (Naoya Anzai and Tom Lane)

  This fix applies only to Windows.

- Fix pg_dumpall to work when a source database sets `default_transaction_read_only` via `ALTER DATABASE SET` (Kevin Grittner)

  Previously, the generated script would fail during restore.

- Make ecpg search for quoted cursor names case-sensitively (Zoltán Böszörményi)

- Fix ecpg's processing of lists of variables declared `varchar` (Zoltán Böszörményi)

- Make `contrib/lo` defend against incorrect trigger definitions (Marc Cousin)

- Update time zone data files to tzdata release 2013h for DST law changes in Argentina, Brazil, Jordan, Libya, Liechtenstein, Morocco, and Palestine. Also, new timezone abbreviations WIB, WIT, WITA for Indonesia.

E.40. Release 9.1.10

Release date: 2013-10-10

This release contains a variety of fixes from 9.1.9. For information about new features in the 9.1 major release, see 第 E.50 节.

E.40.1. Migration to Version 9.1.10

A dump/restore is not required for those running 9.1.X.

However, if you are upgrading from a version earlier than 9.1.9, see 第 E.41 节.

E.40.2. Changes

- Prevent corruption of multi-byte characters when attempting to case-fold identifiers (Andrew Dunstan)

  PostgreSQL case-folds non-ASCII characters only when using a single-byte server encoding.

- Fix checkpoint memory leak in background writer when `wal_level = hot_standby` (Naoya Anzai)

- Fix memory leak caused by `lo_open()` failure (Heikki Linnakangas)

- Fix memory overcommit bug when `work_mem` is using more than 24GB of memory (Stephen Frost)

- Serializable snapshot fixes (Kevin Grittner, Heikki Linnakangas)
- Fix deadlock bug in libpq when using SSL (Stephen Frost)
- Fix possible SSL state corruption in threaded libpq applications (Nick Phillips, Stephen Frost)
- Properly compute row estimates for boolean columns containing many NULL values (Andrew Gierth)

  Previously tests like `col IS NOT TRUE` and `col IS NOT FALSE` did not properly factor in NULL values when estimating plan costs.
- Prevent pushing down `WHERE` clauses into unsafe `UNION/INTERSECT` subqueries (Tom Lane)

  Subqueries of a `UNION` or `INTERSECT` that contain set-returning functions or volatile functions in their `SELECT` lists could be improperly optimized, leading to run-time errors or incorrect query results.
- Fix rare case of "failed to locate grouping columns" planner failure (Tom Lane)
- Fix pg_dump of foreign tables with dropped columns (Andrew Dunstan)

  Previously such cases could cause a pg_upgrade error.
- Reorder pg_dump processing of extension-related rules and event triggers (Joe Conway)
- Force dumping of extension tables if specified by `pg_dump -t` or `-n` (Joe Conway)
- Improve view dumping code's handling of dropped columns in referenced tables (Tom Lane)
- Fix `pg_restore -l` with the directory archive to display the correct format name (Fujii Masao)
- Properly record index comments created using `UNIQUE` and `PRIMARY KEY` syntax (Andres Freund)

  This fixes a parallel pg_restore failure.
- Properly guarantee transmission of WAL files before clean switchover (Fujii Masao)

  Previously, the streaming replication connection might close before all WAL files had been replayed on the standby.
- Fix WAL segment timeline handling during recovery (Mitsumasa Kondo, Heikki Linnakangas)

  WAL file recycling during standby recovery could lead to premature recovery completion, resulting in data loss.
- Fix `REINDEX TABLE` and `REINDEX DATABASE` to properly revalidate constraints and mark invalidated indexes as valid (Noah Misch)

  `REINDEX INDEX` has always worked properly.
- Fix possible deadlock during concurrent `CREATE INDEX CONCURRENTLY` operations (Tom Lane)
- Fix `regexp_matches()` handling of zero-length matches (Jeevan Chalke)

Previously, zero-length matches like '^' could return too many matches.

- Fix crash for overly-complex regular expressions (Heikki Linnakangas)
- Fix regular expression match failures for back references combined with non-greedy quantifiers (Jeevan Chalke)
- Prevent `CREATE FUNCTION` from checking `SET` variables unless function body checking is enabled (Tom Lane)
- Allow `ALTER DEFAULT PRIVILEGES` to operate on schemas without requiring CREATE permission (Tom Lane)
- Loosen restriction on keywords used in queries (Tom Lane)

  Specifically, lessen keyword restrictions for role names, language names, `EXPLAIN` and `COPY` options, and `SET` values. This allows `COPY ... (FORMAT BINARY)` to work as expected; previously `BINARY` needed to be quoted.
- Fix `pgp_pub_decrypt()` so it works for secret keys with passwords (Marko Kreen)
- Make pg_upgrade use `pg_dump --quote-all-identifiers` to avoid problems with keyword changes between releases (Tom Lane)
- Remove rare inaccurate warning during vacuum of index-less tables (Heikki Linnakangas)
- Ensure that `VACUUM ANALYZE` still runs the ANALYZE phase if its attempt to truncate the file is cancelled due to lock conflicts (Kevin Grittner)
- Avoid possible failure when performing transaction control commands (e.g `ROLLBACK`) in prepared queries (Tom Lane)
- Ensure that floating-point data input accepts standard spellings of "infinity" on all platforms (Tom Lane)

  The C99 standard says that allowable spellings are `inf`, `+inf`, `-inf`, `infinity`, `+infinity`, and `-infinity`. Make sure we recognize these even if the platform's `strtod` function doesn't.
- Expand ability to compare rows to records and arrays (Rafal Rzepecki, Tom Lane)
- Update time zone data files to tzdata release 2013d for DST law changes in Israel, Morocco, Palestine, and Paraguay. Also, historical zone data corrections for Macquarie Island.

E.41. Release 9.1.9

Release date: 2013-04-04

This release contains a variety of fixes from 9.1.8. For information about new features in the 9.1 major release, see 第 E.50 节.

E.41.1. Migration to Version 9.1.9

A dump/restore is not required for those running 9.1.X.

However, this release corrects several errors in management of GiST indexes. After installing this update, it is advisable to `REINDEX` any GiST indexes that meet one or more of the conditions described below.

Also, if you are upgrading from a version earlier than 9.1.6, see 第 E.44 节.

E.41.2. Changes

- Fix insecure parsing of server command-line switches (Mitsumasa Kondo, Kyotaro Horiguchi)

  A connection request containing a database name that begins with "-" could be crafted to damage or destroy files within the server's data directory, even if the request is eventually rejected. (CVE-2013-1899)

- Reset OpenSSL randomness state in each postmaster child process (Marko Kreen)

  This avoids a scenario wherein random numbers generated by `contrib/pgcrypto` functions might be relatively easy for another database user to guess. The risk is only significant when the postmaster is configured with `ssl` = `on` but most connections don't use SSL encryption. (CVE-2013-1900)

- Make REPLICATION privilege checks test current user not authenticated user (Noah Misch)

  An unprivileged database user could exploit this mistake to call `pg_start_backup()` or `pg_stop_backup()`, thus possibly interfering with creation of routine backups. (CVE-2013-1901)

- Fix GiST indexes to not use "fuzzy" geometric comparisons when it's not appropriate to do so (Alexander Korotkov)

  The core geometric types perform comparisons using "fuzzy" equality, but `gist_box_same` must do exact comparisons, else GiST indexes using it might become inconsistent. After installing this update, users should `REINDEX` any GiST indexes on `box`, `polygon`, `circle`, or `point` columns, since all of these use `gist_box_same`.

- Fix erroneous range-union and penalty logic in GiST indexes that use `contrib/btree_gist` for variable-width data types, that is `text`, `bytea`, `bit`, and `numeric` columns (Tom Lane)

  These errors could result in inconsistent indexes in which some keys that are present would not be found by searches, and also in useless index bloat. Users are advised to `REINDEX` such indexes after installing this update.

- Fix bugs in GiST page splitting code for multi-column indexes (Tom Lane)

  These errors could result in inconsistent indexes in which some keys that are present would not be found by searches, and also in indexes that are unnecessarily inefficient to search. Users are advised to `REINDEX` multi-column GiST indexes after installing this update.

- Fix `gist_point_consistent` to handle fuzziness consistently (Alexander Korotkov)

  Index scans on GiST indexes on `point` columns would sometimes yield results different from a sequential scan, because `gist_point_consistent` disagreed with the underlying operator code about whether to do comparisons exactly or fuzzily.

- Fix buffer leak in WAL replay (Heikki Linnakangas)

  This bug could result in "incorrect local pin count" errors during replay, making recovery impossible.

- Fix race condition in `DELETE RETURNING` (Tom Lane)

Under the right circumstances, `DELETE RETURNING` could attempt to fetch data from a shared buffer that the current process no longer has any pin on. If some other process changed the buffer meanwhile, this would lead to garbage `RETURNING` output, or even a crash.

- Fix infinite-loop risk in regular expression compilation (Tom Lane, Don Porter)
- Fix potential null-pointer dereference in regular expression compilation (Tom Lane)
- Fix `to_char()` to use ASCII-only case-folding rules where appropriate (Tom Lane)

  This fixes misbehavior of some template patterns that should be locale-independent, but mishandled "`I`" and "`i`" in Turkish locales.

- Fix unwanted rejection of timestamp `1999-12-31 24:00:00` (Tom Lane)
- Fix logic error when a single transaction does `UNLISTEN` then `LISTEN` (Tom Lane)

  The session wound up not listening for notify events at all, though it surely should listen in this case.

- Fix possible planner crash after columns have been added to a view that's depended on by another view (Tom Lane)
- Remove useless "picksplit doesn't support secondary split" log messages (Josh Hansen, Tom Lane)

  This message seems to have been added in expectation of code that was never written, and probably never will be, since GiST's default handling of secondary splits is actually pretty good. So stop nagging end users about it.

- Fix possible failure to send a session's last few transaction commit/abort counts to the statistics collector (Tom Lane)
- Eliminate memory leaks in PL/Perl's `spi_prepare()` function (Alex Hunsaker, Tom Lane)
- Fix pg_dumpall to handle database names containing "`=`" correctly (Heikki Linnakangas)
- Avoid crash in pg_dump when an incorrect connection string is given (Heikki Linnakangas)
- Ignore invalid indexes in pg_dump and pg_upgrade (Michael Paquier, Bruce Momjian)

  Dumping invalid indexes can cause problems at restore time, for example if the reason the index creation failed was because it tried to enforce a uniqueness condition not satisfied by the table's data. Also, if the index creation is in fact still in progress, it seems reasonable to consider it to be an uncommitted DDL change, which pg_dump wouldn't be expected to dump anyway. pg_upgrade now also skips invalid indexes rather than failing.

- In pg_basebackup, include only the current server version's subdirectory when backing up a tablespace (Heikki Linnakangas)
- Add a server version check in pg_basebackup and pg_receivexlog, so they fail cleanly with version combinations that won't work (Heikki Linnakangas)
- Fix `contrib/pg_trgm`'s `similarity()` function to return zero for trigram-less strings (Tom Lane)

Previously it returned `NaN` due to internal division by zero.

- Update time zone data files to tzdata release 2013b for DST law changes in Chile, Haiti, Morocco, Paraguay, and some Russian areas. Also, historical zone data corrections for numerous places.

  Also, update the time zone abbreviation files for recent changes in Russia and elsewhere: `CHOT`, `GET`, `IRKT`, `KGT`, `KRAT`, `MAGT`, `MAWT`, `MSK`, `NOVT`, `OMST`, `TKT`, `VLAT`, `WST`, `YAKT`, `YEKT` now follow their current meanings, and `VOLT` (Europe/Volgograd) and `MIST` (Antarctica/Macquarie) are added to the default abbreviations list.

E.42. Release 9.1.8

Release date: 2013-02-07

This release contains a variety of fixes from 9.1.7. For information about new features in the 9.1 major release, see 第 E.50 节.

E.42.1. Migration to Version 9.1.8

A dump/restore is not required for those running 9.1.X.

However, if you are upgrading from a version earlier than 9.1.6, see 第 E.44 节.

E.42.2. Changes

- Prevent execution of `enum_recv` from SQL (Tom Lane)

  The function was misdeclared, allowing a simple SQL command to crash the server. In principle an attacker might be able to use it to examine the contents of server memory. Our thanks to Sumit Soni (via Secunia SVCRP) for reporting this issue. (CVE-2013-0255)

- Fix multiple problems in detection of when a consistent database state has been reached during WAL replay (Fujii Masao, Heikki Linnakangas, Simon Riggs, Andres Freund)

- Update minimum recovery point when truncating a relation file (Heikki Linnakangas)

  Once data has been discarded, it's no longer safe to stop recovery at an earlier point in the timeline.

- Fix recycling of WAL segments after changing recovery target timeline (Heikki Linnakangas)

- Fix missing cancellations in hot standby mode (Noah Misch, Simon Riggs)

  The need to cancel conflicting hot-standby queries would sometimes be missed, allowing those queries to see inconsistent data.

- Prevent recovery pause feature from pausing before users can connect (Tom Lane)

- Fix SQL grammar to allow subscripting or field selection from a sub-SELECT result (Tom Lane)

- Fix performance problems with autovacuum truncation in busy workloads (Jan Wieck)

Truncation of empty pages at the end of a table requires exclusive lock, but autovacuum was coded to fail (and release the table lock) when there are conflicting lock requests. Under load, it is easily possible that truncation would never occur, resulting in table bloat. Fix by performing a partial truncation, releasing the lock, then attempting to re-acquire the lock and continue. This fix also greatly reduces the average time before autovacuum releases the lock after a conflicting request arrives.

- Protect against race conditions when scanning `pg_tablespace` (Stephen Frost, Tom Lane)

  `CREATE DATABASE` and `DROP DATABASE` could misbehave if there were concurrent updates of `pg_tablespace` entries.

- Prevent `DROP OWNED` from trying to drop whole databases or tablespaces (Álvaro Herrera)

  For safety, ownership of these objects must be reassigned, not dropped.

- Fix error in `vacuum_freeze_table_age` implementation (Andres Freund)

  In installations that have existed for more than `vacuum_freeze_min_age` transactions, this mistake prevented autovacuum from using partial-table scans, so that a full-table scan would always happen instead.

- Prevent misbehavior when a `RowExpr` or `XmlExpr` is parse-analyzed twice (Andres Freund, Tom Lane)

  This mistake could be user-visible in contexts such as `CREATE TABLE LIKE INCLUDING INDEXES`.

- Improve defenses against integer overflow in hashtable sizing calculations (Jeff Davis)

- Fix failure to ignore leftover temporary tables after a server crash (Tom Lane)

- Reject out-of-range dates in `to_date()` (Hitoshi Harada)

- Fix `pg_extension_config_dump()` to handle extension-update cases properly (Tom Lane)

  This function will now replace any existing entry for the target table, making it usable in extension update scripts.

- Fix PL/Python's handling of functions used as triggers on multiple tables (Andres Freund)

- Ensure that non-ASCII prompt strings are translated to the correct code page on Windows (Alexander Law, Noah Misch)

  This bug affected psql and some other client programs.

- Fix possible crash in psql's `\?` command when not connected to a database (Meng Qingzhong)

- Fix possible error if a relation file is removed while pg_basebackup is running (Heikki Linnakangas)

- Make pg_dump exclude data of unlogged tables when running on a hot-standby server (Magnus Hagander)

This would fail anyway because the data is not available on the standby server, so it seems most convenient to assume `--no-unlogged-table-data` automatically.

- Fix pg_upgrade to deal with invalid indexes safely (Bruce Momjian)
- Fix one-byte buffer overrun in libpq's `PQprintTuples` (Xi Wang)

  This ancient function is not used anywhere by PostgreSQL itself, but it might still be used by some client code.
- Make ecpglib use translated messages properly (Chen Huajun)
- Properly install ecpg_compat and pgtypes libraries on MSVC (Jiang Guiqing)
- Include our version of `isinf()` in libecpg if it's not provided by the system (Jiang Guiqing)
- Rearrange configure's tests for supplied functions so it is not fooled by bogus exports from libedit/libreadline (Christoph Berg)
- Ensure Windows build number increases over time (Magnus Hagander)
- Make pgxs build executables with the right `.exe` suffix when cross-compiling for Windows (Zoltan Boszormenyi)
- Add new timezone abbreviation `FET` (Tom Lane)

  This is now used in some eastern-European time zones.

E.43. Release 9.1.7

Release date: 2012-12-06

This release contains a variety of fixes from 9.1.6. For information about new features in the 9.1 major release, see 第 E.50 节.

E.43.1. Migration to Version 9.1.7

A dump/restore is not required for those running 9.1.X.

However, if you are upgrading from a version earlier than 9.1.6, see 第 E.44 节.

E.43.2. Changes

- Fix multiple bugs associated with `CREATE INDEX CONCURRENTLY` (Andres Freund, Tom Lane)

  Fix `CREATE INDEX CONCURRENTLY` to use in-place updates when changing the state of an index's `pg_index` row. This prevents race conditions that could cause concurrent sessions to miss updating the target index, thus resulting in corrupt concurrently-created indexes.

  Also, fix various other operations to ensure that they ignore invalid indexes resulting from a failed `CREATE INDEX CONCURRENTLY` command. The most important of these is `VACUUM`, because an auto-vacuum could easily be launched on the table before corrective action can be taken to fix or remove the invalid index.

- Fix buffer locking during WAL replay (Tom Lane)

  The WAL replay code was insufficiently careful about locking buffers when replaying WAL records that affect more than one page. This could result in hot standby queries transiently seeing inconsistent states, resulting in wrong answers or unexpected failures.

- Fix an error in WAL generation logic for GIN indexes (Tom Lane)

  This could result in index corruption, if a torn-page failure occurred.

- Properly remove startup process's virtual XID lock when promoting a hot standby server to normal running (Simon Riggs)

  This oversight could prevent subsequent execution of certain operations such as `CREATE INDEX CONCURRENTLY`.

- Avoid bogus "out-of-sequence timeline ID" errors in standby mode (Heikki Linnakangas)

- Prevent the postmaster from launching new child processes after it's received a shutdown signal (Tom Lane)

  This mistake could result in shutdown taking longer than it should, or even never completing at all without additional user action.

- Avoid corruption of internal hash tables when out of memory (Hitoshi Harada)

- Prevent file descriptors for dropped tables from being held open past transaction end (Tom Lane)

  This should reduce problems with long-since-dropped tables continuing to occupy disk space.

- Prevent database-wide crash and restart when a new child process is unable to create a pipe for its latch (Tom Lane)

  Although the new process must fail, there is no good reason to force a database-wide restart, so avoid that. This improves robustness when the kernel is nearly out of file descriptors.

- Fix planning of non-strict equivalence clauses above outer joins (Tom Lane)

  The planner could derive incorrect constraints from a clause equating a non-strict construct to something else, for example `WHERE COALESCE(foo, 0) = 0` when `foo` is coming from the nullable side of an outer join.

- Fix `SELECT DISTINCT` with index-optimized `MIN`/`MAX` on an inheritance tree (Tom Lane)

  The planner would fail with "failed to re-find MinMaxAggInfo record" given this combination of factors.

- Improve planner's ability to prove exclusion constraints from equivalence classes (Tom Lane)

- Fix partial-row matching in hashed subplans to handle cross-type cases correctly (Tom Lane)

  This affects multicolumn `NOT IN` subplans, such as `WHERE (a, b) NOT IN (SELECT x, y FROM ...)` when for instance `b` and `y` are `int4` and `int8` respectively.

This mistake led to wrong answers or crashes depending on the specific datatypes involved.

- Acquire buffer lock when re-fetching the old tuple for an `AFTER ROW UPDATE/DELETE` trigger (Andres Freund)

  In very unusual circumstances, this oversight could result in passing incorrect data to a trigger `WHEN` condition, or to the precheck logic for a foreign-key enforcement trigger. That could result in a crash, or in an incorrect decision about whether to fire the trigger.

- Fix `ALTER COLUMN TYPE` to handle inherited check constraints properly (Pavan Deolasee)

  This worked correctly in pre-8.4 releases, and now works correctly in 8.4 and later.

- Fix `ALTER EXTENSION SET SCHEMA`'s failure to move some subsidiary objects into the new schema (Álvaro Herrera, Dimitri Fontaine)

- Fix `REASSIGN OWNED` to handle grants on tablespaces (Álvaro Herrera)

- Ignore incorrect `pg_attribute` entries for system columns for views (Tom Lane)

  Views do not have any system columns. However, we forgot to remove such entries when converting a table to a view. That's fixed properly for 9.3 and later, but in previous branches we need to defend against existing mis-converted views.

- Fix rule printing to dump `INSERT INTO` *`table`* `DEFAULT VALUES` correctly (Tom Lane)

- Guard against stack overflow when there are too many `UNION`/`INTERSECT`/`EXCEPT` clauses in a query (Tom Lane)

- Prevent platform-dependent failures when dividing the minimum possible integer value by -1 (Xi Wang, Tom Lane)

- Fix possible access past end of string in date parsing (Hitoshi Harada)

- Fix failure to advance XID epoch if XID wraparound happens during a checkpoint and `wal_level` is `hot_standby` (Tom Lane, Andres Freund)

  While this mistake had no particular impact on PostgreSQL itself, it was bad for applications that rely on `txid_current()` and related functions: the TXID value would appear to go backwards.

- Fix display of `pg_stat_replication.sync_state` at a page boundary (Kyotaro Horiguchi)

- Produce an understandable error message if the length of the path name for a Unix-domain socket exceeds the platform-specific limit (Tom Lane, Andrew Dunstan)

  Formerly, this would result in something quite unhelpful, such as “Non-recoverable failure in name resolution” .

- Fix memory leaks when sending composite column values to the client (Tom Lane)

- Make pg_ctl more robust about reading the `postmaster.pid` file (Heikki Linnakangas)

  Fix race conditions and possible file descriptor leakage.

- Fix possible crash in psql if incorrectly-encoded data is presented and the `client_encoding` setting is a client-only encoding, such as SJIS (Jiang Guiqing)
- Make pg_dump dump `SEQUENCE SET` items in the data not pre-data section of the archive (Tom Lane)

  This change fixes dumping of sequences that are marked as extension configuration tables.
- Fix bugs in the `restore.sql` script emitted by pg_dump in `tar` output format (Tom Lane)

  The script would fail outright on tables whose names include upper-case characters. Also, make the script capable of restoring data in `--inserts` mode as well as the regular COPY mode.
- Fix pg_restore to accept POSIX-conformant `tar` files (Brian Weaver, Tom Lane)

  The original coding of pg_dump's `tar` output mode produced files that are not fully conformant with the POSIX standard. This has been corrected for version 9.3. This patch updates previous branches so that they will accept both the incorrect and the corrected formats, in hopes of avoiding compatibility problems when 9.3 comes out.
- Fix `tar` files emitted by pg_basebackup to be POSIX conformant (Brian Weaver, Tom Lane)
- Fix pg_resetxlog to locate `postmaster.pid` correctly when given a relative path to the data directory (Tom Lane)

  This mistake could lead to pg_resetxlog not noticing that there is an active postmaster using the data directory.
- Fix libpq's `lo_import()` and `lo_export()` functions to report file I/O errors properly (Tom Lane)
- Fix ecpg's processing of nested structure pointer variables (Muhammad Usama)
- Fix ecpg's `ecpg_get_data` function to handle arrays properly (Michael Meskes)
- Make `contrib/pageinspect`'s btree page inspection functions take buffer locks while examining pages (Tom Lane)
- Ensure that `make install` for an extension creates the `extension` installation directory (Cédric Villemain)

  Previously, this step was missed if `MODULEDIR` was set in the extension's Makefile.
- Fix pgxs support for building loadable modules on AIX (Tom Lane)

  Building modules outside the original source tree didn't work on AIX.
- Update time zone data files to tzdata release 2012j for DST law changes in Cuba, Israel, Jordan, Libya, Palestine, Western Samoa, and portions of Brazil.

E.44. Release 9.1.6

Release date: 2012-09-24

This release contains a variety of fixes from 9.1.5. For information about new features in the 9.1 major release, see 第 E.50 节.

E.44.1. Migration to Version 9.1.6

A dump/restore is not required for those running 9.1.X.

However, you may need to perform `REINDEX` operations to recover from the effects of the data corruption bug described in the first changelog item below.

Also, if you are upgrading from a version earlier than 9.1.4, see 第 E.46 节.

E.44.2. Changes

- Fix persistence marking of shared buffers during WAL replay (Jeff Davis)

  This mistake can result in buffers not being written out during checkpoints, resulting in data corruption if the server later crashes without ever having written those buffers. Corruption can occur on any server following crash recovery, but it is significantly more likely to occur on standby slave servers since those perform much more WAL replay. There is a low probability of corruption of btree and GIN indexes. There is a much higher probability of corruption of table "visibility maps". Fortunately, visibility maps are non-critical data in 9.1, so the worst consequence of such corruption in 9.1 installations is transient inefficiency of vacuuming. Table data proper cannot be corrupted by this bug.

  While no index corruption due to this bug is known to have occurred in the field, as a precautionary measure it is recommended that production installations `REINDEX` all btree and GIN indexes at a convenient time after upgrading to 9.1.6.

  Also, if you intend to do an in-place upgrade to 9.2.X, before doing so it is recommended to perform a `VACUUM` of all tables while having `vacuum_freeze_table_age` set to zero. This will ensure that any lingering wrong data in the visibility maps is corrected before 9.2.X can depend on it. `vacuum_cost_delay` can be adjusted to reduce the performance impact of vacuuming, while causing it to take longer to finish.

- Fix planner's assignment of executor parameters, and fix executor's rescan logic for CTE plan nodes (Tom Lane)

  These errors could result in wrong answers from queries that scan the same `WITH` subquery multiple times.

- Fix misbehavior when `default_transaction_isolation` is set to `serializable` (Kevin Grittner, Tom Lane, Heikki Linnakangas)

  Symptoms include crashes at process start on Windows, and crashes in hot standby operation.

- Improve selectivity estimation for text search queries involving prefixes, i.e. *word*:* patterns (Tom Lane)

- Improve page-splitting decisions in GiST indexes (Alexander Korotkov, Robert Haas, Tom Lane)

  Multi-column GiST indexes might suffer unexpected bloat due to this error.

- Fix cascading privilege revoke to stop if privileges are still held (Tom Lane)

  If we revoke a grant option from some role *X*, but *X* still holds that option via a grant from someone else, we should not recursively revoke the corresponding privilege from role(s) *Y* that *X* had granted it to.

- Disallow extensions from containing the schema they are assigned to (Thom Brown)

  This situation creates circular dependencies that confuse pg_dump and probably other things. It's confusing for humans too, so disallow it.

- Improve error messages for Hot Standby misconfiguration errors (Gurjeet Singh)

- Make configure probe for `mbstowcs_l` (Tom Lane)

  This fixes build failures on some versions of AIX.

- Fix handling of `SIGFPE` when PL/Perl is in use (Andres Freund)

  Perl resets the process's `SIGFPE` handler to `SIG_IGN`, which could result in crashes later on. Restore the normal Postgres signal handler after initializing PL/Perl.

- Prevent PL/Perl from crashing if a recursive PL/Perl function is redefined while being executed (Tom Lane)

- Work around possible misoptimization in PL/Perl (Tom Lane)

  Some Linux distributions contain an incorrect version of `pthread.h` that results in incorrect compiled code in PL/Perl, leading to crashes if a PL/Perl function calls another one that throws an error.

- Fix bugs in `contrib/pg_trgm`'s `LIKE` pattern analysis code (Fujii Masao)

  `LIKE` queries using a trigram index could produce wrong results if the pattern contained `LIKE` escape characters.

- Fix pg_upgrade's handling of line endings on Windows (Andrew Dunstan)

  Previously, pg_upgrade might add or remove carriage returns in places such as function bodies.

- On Windows, make pg_upgrade use backslash path separators in the scripts it emits (Andrew Dunstan)

- Remove unnecessary dependency on pg_config from pg_upgrade (Peter Eisentraut)

- Update time zone data files to tzdata release 2012f for DST law changes in Fiji

E.45. Release 9.1.5

Release date: 2012-08-17

This release contains a variety of fixes from 9.1.4. For information about new features in the 9.1 major release, see 第 E.50 节.

E.45.1. Migration to Version 9.1.5

A dump/restore is not required for those running 9.1.X.

However, if you are upgrading from a version earlier than 9.1.4, see 第 E.46 节.

E.45.2. Changes

- Prevent access to external files/URLs via XML entity references (Noah Misch, Tom Lane)

`xml_parse()` would attempt to fetch external files or URLs as needed to resolve DTD and entity references in an XML value, thus allowing unprivileged database users to attempt to fetch data with the privileges of the database server. While the external data wouldn't get returned directly to the user, portions of it could be exposed in error messages if the data didn't parse as valid XML; and in any case the mere ability to check existence of a file might be useful to an attacker. (CVE-2012-3489)

- Prevent access to external files/URLs via `contrib/xml2`'s `xslt_process()` (Peter Eisentraut)

  libxslt offers the ability to read and write both files and URLs through stylesheet commands, thus allowing unprivileged database users to both read and write data with the privileges of the database server. Disable that through proper use of libxslt's security options. (CVE-2012-3488)

  Also, remove `xslt_process()`'s ability to fetch documents and stylesheets from external files/URLs. While this was a documented "feature", it was long regarded as a bad idea. The fix for CVE-2012-3489 broke that capability, and rather than expend effort on trying to fix it, we're just going to summarily remove it.

- Prevent too-early recycling of btree index pages (Noah Misch)

  When we allowed read-only transactions to skip assigning XIDs, we introduced the possibility that a deleted btree page could be recycled while a read-only transaction was still in flight to it. This would result in incorrect index search results. The probability of such an error occurring in the field seems very low because of the timing requirements, but nonetheless it should be fixed.

- Fix crash-safety bug with newly-created-or-reset sequences (Tom Lane)

  If `ALTER SEQUENCE` was executed on a freshly created or reset sequence, and then precisely one `nextval()` call was made on it, and then the server crashed, WAL replay would restore the sequence to a state in which it appeared that no `nextval()` had been done, thus allowing the first sequence value to be returned again by the next `nextval()` call. In particular this could manifest for `serial` columns, since creation of a serial column's sequence includes an `ALTER SEQUENCE OWNED BY` step.

- Fix race condition in `enum`-type value comparisons (Robert Haas, Tom Lane)

  Comparisons could fail when encountering an enum value added since the current query started.

- Fix `txid_current()` to report the correct epoch when not in hot standby (Heikki Linnakangas)

  This fixes a regression introduced in the previous minor release.

- Prevent selection of unsuitable replication connections as the synchronous standby (Fujii Masao)

  The master might improperly choose pseudo-servers such as pg_receivexlog or pg_basebackup as the synchronous standby, and then wait indefinitely for them.

- Fix bug in startup of Hot Standby when a master transaction has many subtransactions (Andres Freund)

  This mistake led to failures reported as "out-of-order XID insertion in KnownAssignedXids".

- Ensure the `backup_label` file is fsync'd after `pg_start_backup()` (Dave Kerr)

- Fix timeout handling in walsender processes (Tom Lane)

  WAL sender background processes neglected to establish a SIGALRM handler, meaning they would wait forever in some corner cases where a timeout ought to happen.

- Wake walsenders after each background flush by walwriter (Andres Freund, Simon Riggs)

  This greatly reduces replication delay when the workload contains only asynchronously-committed transactions.

- Fix `LISTEN`/`NOTIFY` to cope better with I/O problems, such as out of disk space (Tom Lane)

  After a write failure, all subsequent attempts to send more `NOTIFY` messages would fail with messages like “Could not read from file "pg_notify/*nnnn*" at offset *nnnnn* : Success” .

- Only allow autovacuum to be auto-canceled by a directly blocked process (Tom Lane)

  The original coding could allow inconsistent behavior in some cases; in particular, an autovacuum could get canceled after less than `deadlock_timeout` grace period.

- Improve logging of autovacuum cancels (Robert Haas)

- Fix log collector so that `log_truncate_on_rotation` works during the very first log rotation after server start (Tom Lane)

- Fix `WITH` attached to a nested set operation (`UNION`/`INTERSECT`/`EXCEPT`) (Tom Lane)

- Ensure that a whole-row reference to a subquery doesn't include any extra `GROUP BY` or `ORDER BY` columns (Tom Lane)

- Fix dependencies generated during `ALTER TABLE ... ADD CONSTRAINT USING INDEX` (Tom Lane)

  This command left behind a redundant `pg_depend` entry for the index, which could confuse later operations, notably `ALTER TABLE ... ALTER COLUMN TYPE` on one of the indexed columns.

- Fix `REASSIGN OWNED` to work on extensions (Alvaro Herrera)

- Disallow copying whole-row references in `CHECK` constraints and index definitions during `CREATE TABLE` (Tom Lane)

  This situation can arise in `CREATE TABLE` with `LIKE` or `INHERITS`. The copied whole-row variable was incorrectly labeled with the row type of the original table not the new one. Rejecting the case seems reasonable for `LIKE`, since the row types might well diverge later. For `INHERITS` we should ideally allow it, with an implicit coercion to the parent table's row type; but that will require more work than seems safe to back-patch.

- Fix memory leak in `ARRAY(SELECT ...)` subqueries (Heikki Linnakangas, Tom Lane)

- Fix planner to pass correct collation to operator selectivity estimators (Tom Lane)

  This was not previously required by any core selectivity estimation function, but third-party code might need it.

- Fix extraction of common prefixes from regular expressions (Tom Lane)

The code could get confused by quantified parenthesized subexpressions, such as `^(foo)?bar`. This would lead to incorrect index optimization of searches for such patterns.

- Fix bugs with parsing signed *hh*:*mm* and *hh*:*mm*:*ss* fields in `interval` constants (Amit Kapila, Tom Lane)
- Fix pg_dump to better handle views containing partial `GROUP BY` lists (Tom Lane)

  A view that lists only a primary key column in `GROUP BY`, but uses other table columns as if they were grouped, gets marked as depending on the primary key. Improper handling of such primary key dependencies in pg_dump resulted in poorly-ordered dumps, which at best would be inefficient to restore and at worst could result in outright failure of a parallel pg_restore run.
- In PL/Perl, avoid setting UTF8 flag when in SQL_ASCII encoding (Alex Hunsaker, Kyotaro Horiguchi, Alvaro Herrera)
- Use Postgres' encoding conversion functions, not Python's, when converting a Python Unicode string to the server encoding in PL/Python (Jan Urbanski)

  This avoids some corner-case problems, notably that Python doesn't support all the encodings Postgres does. A notable functional change is that if the server encoding is SQL_ASCII, you will get the UTF-8 representation of the string; formerly, any non-ASCII characters in the string would result in an error.
- Fix mapping of PostgreSQL encodings to Python encodings in PL/Python (Jan Urbanski)
- Report errors properly in `contrib/xml2`'s `xslt_process()` (Tom Lane)
- Update time zone data files to tzdata release 2012e for DST law changes in Morocco and Tokelau

E.46. Release 9.1.4

Release date: 2012-06-04

This release contains a variety of fixes from 9.1.3. For information about new features in the 9.1 major release, see 第 E.50 节.

E.46.1. Migration to Version 9.1.4

A dump/restore is not required for those running 9.1.X.

However, if you use the `citext` data type, and you upgraded from a previous major release by running pg_upgrade, you should run `CREATE EXTENSION citext FROM unpackaged` to avoid collation-related failures in `citext` operations. The same is necessary if you restore a dump from a pre-9.1 database that contains an instance of the `citext` data type. If you've already run the `CREATE EXTENSION` command before upgrading to 9.1.4, you will instead need to do manual catalog updates as explained in the third changelog item below.

Also, if you are upgrading from a version earlier than 9.1.2, see 第 E.48 节.

E.46.2. Changes

- Fix incorrect password transformation in `contrib/pgcrypto`'s DES `crypt()` function (Solar Designer)

If a password string contained the byte value `0x80`, the remainder of the password was ignored, causing the password to be much weaker than it appeared. With this fix, the rest of the string is properly included in the DES hash. Any stored password values that are affected by this bug will thus no longer match, so the stored values may need to be updated. (CVE-2012-2143)

- Ignore `SECURITY DEFINER` and `SET` attributes for a procedural language's call handler (Tom Lane)

  Applying such attributes to a call handler could crash the server. (CVE-2012-2655)

- Make `contrib/citext`'s upgrade script fix collations of `citext` arrays and domains over `citext` (Tom Lane)

  Release 9.1.2 provided a fix for collations of `citext` columns and indexes in databases upgraded or reloaded from pre-9.1 installations, but that fix was incomplete: it neglected to handle arrays and domains over `citext`. This release extends the module's upgrade script to handle these cases. As before, if you have already run the upgrade script, you'll need to run the collation update commands by hand instead. See the 9.1.2 release notes for more information about doing this.

- Allow numeric timezone offsets in `timestamp` input to be up to 16 hours away from UTC (Tom Lane)

  Some historical time zones have offsets larger than 15 hours, the previous limit. This could result in dumped data values being rejected during reload.

- Fix timestamp conversion to cope when the given time is exactly the last DST transition time for the current timezone (Tom Lane)

  This oversight has been there a long time, but was not noticed previously because most DST-using zones are presumed to have an indefinite sequence of future DST transitions.

- Fix `text` to `name` and `char` to `name` casts to perform string truncation correctly in multibyte encodings (Karl Schnaitter)

- Fix memory copying bug in `to_tsquery()` (Heikki Linnakangas)

- Ensure `txid_current()` reports the correct epoch when executed in hot standby (Simon Riggs)

- Fix planner's handling of outer PlaceHolderVars within subqueries (Tom Lane)

  This bug concerns sub-SELECTs that reference variables coming from the nullable side of an outer join of the surrounding query. In 9.1, queries affected by this bug would fail with "ERROR: Upper-level PlaceHolderVar found where not expected". But in 9.0 and 8.4, you'd silently get possibly-wrong answers, since the value transmitted into the subquery wouldn't go to null when it should.

- Fix planning of `UNION ALL` subqueries with output columns that are not simple variables (Tom Lane)

  Planning of such cases got noticeably worse in 9.1 as a result of a misguided fix for "MergeAppend child's targetlist doesn't match MergeAppend" errors. Revert that fix and do it another way.

- Fix slow session startup when `pg_attribute` is very large (Tom Lane)

  If `pg_attribute` exceeds one-fourth of `shared_buffers`, cache rebuilding code that is sometimes needed during session start would trigger the synchronized-

scan logic, causing it to take many times longer than normal. The problem was particularly acute if many new sessions were starting at once.

- Ensure sequential scans check for query cancel reasonably often (Merlin Moncure)

  A scan encountering many consecutive pages that contain no live tuples would not respond to interrupts meanwhile.

- Ensure the Windows implementation of `PGSemaphoreLock()` clears `ImmediateInterruptOK` before returning (Tom Lane)

  This oversight meant that a query-cancel interrupt received later in the same query could be accepted at an unsafe time, with unpredictable but not good consequences.

- Show whole-row variables safely when printing views or rules (Abbas Butt, Tom Lane)

  Corner cases involving ambiguous names (that is, the name could be either a table or column name of the query) were printed in an ambiguous way, risking that the view or rule would be interpreted differently after dump and reload. Avoid the ambiguous case by attaching a no-op cast.

- Fix `COPY FROM` to properly handle null marker strings that correspond to invalid encoding (Tom Lane)

  A null marker string such as `E'\\0'` should work, and did work in the past, but the case got broken in 8.4.

- Fix `EXPLAIN VERBOSE` for writable CTEs containing `RETURNING` clauses (Tom Lane)

- Fix `PREPARE TRANSACTION` to work correctly in the presence of advisory locks (Tom Lane)

  Historically, `PREPARE TRANSACTION` has simply ignored any session-level advisory locks the session holds, but this case was accidentally broken in 9.1.

- Fix truncation of unlogged tables (Robert Haas)

- Ignore missing schemas during non-interactive assignments of `search_path` (Tom Lane)

  This re-aligns 9.1's behavior with that of older branches. Previously 9.1 would throw an error for nonexistent schemas mentioned in `search_path` settings obtained from places such as `ALTER DATABASE SET`.

- Fix bugs with temporary or transient tables used in extension scripts (Tom Lane)

  This includes cases such as a rewriting `ALTER TABLE` within an extension update script, since that uses a transient table behind the scenes.

- Ensure autovacuum worker processes perform stack depth checking properly (Heikki Linnakangas)

  Previously, infinite recursion in a function invoked by auto-`ANALYZE` could crash worker processes.

- Fix logging collector to not lose log coherency under high load (Andrew Dunstan)

  The collector previously could fail to reassemble large messages if it got too busy.

- Fix logging collector to ensure it will restart file rotation after receiving SIGHUP (Tom Lane)
- Fix “too many LWLocks taken” failure in GiST indexes (Heikki Linnakangas)
- Fix WAL replay logic for GIN indexes to not fail if the index was subsequently dropped (Tom Lane)
- Correctly detect SSI conflicts of prepared transactions after a crash (Dan Ports)
- Avoid synchronous replication delay when committing a transaction that only modified temporary tables (Heikki Linnakangas)

  In such a case the transaction's commit record need not be flushed to standby servers, but some of the code didn't know that and waited for it to happen anyway.
- Fix error handling in pg_basebackup (Thomas Ogrisegg, Fujii Masao)
- Fix walsender to not go into a busy loop if connection is terminated (Fujii Masao)
- Fix memory leak in PL/pgSQL's `RETURN NEXT` command (Joe Conway)
- Fix PL/pgSQL's `GET DIAGNOSTICS` command when the target is the function's first variable (Tom Lane)
- Ensure that PL/Perl package-qualifies the `_TD` variable (Alex Hunsaker)

  This bug caused trigger invocations to fail when they are nested within a function invocation that changes the current package.
- Fix PL/Python functions returning composite types to accept a string for their result value (Jan Urbanski)

  This case was accidentally broken by the 9.1 additions to allow a composite result value to be supplied in other formats, such as dictionaries.
- Fix potential access off the end of memory in psql's expanded display (`\x`) mode (Peter Eisentraut)
- Fix several performance problems in pg_dump when the database contains many objects (Jeff Janes, Tom Lane)

  pg_dump could get very slow if the database contained many schemas, or if many objects are in dependency loops, or if there are many owned sequences.
- Fix memory and file descriptor leaks in pg_restore when reading a directory-format archive (Peter Eisentraut)
- Fix pg_upgrade for the case that a database stored in a non-default tablespace contains a table in the cluster's default tablespace (Bruce Momjian)
- In ecpg, fix rare memory leaks and possible overwrite of one byte after the `sqlca_t` structure (Peter Eisentraut)
- Fix `contrib/dblink`'s `dblink_exec()` to not leak temporary database connections upon error (Tom Lane)
- Fix `contrib/dblink` to report the correct connection name in error messages (Kyotaro Horiguchi)
- Fix `contrib/vacuumlo` to use multiple transactions when dropping many large objects (Tim Lewis, Robert Haas, Tom Lane)

This change avoids exceeding `max_locks_per_transaction` when many objects need to be dropped. The behavior can be adjusted with the new `-l` (limit) option.

- Update time zone data files to tzdata release 2012c for DST law changes in Antarctica, Armenia, Chile, Cuba, Falkland Islands, Gaza, Haiti, Hebron, Morocco, Syria, and Tokelau Islands; also historical corrections for Canada.

E.47. Release 9.1.3

Release date: 2012-02-27

This release contains a variety of fixes from 9.1.2. For information about new features in the 9.1 major release, see 第 E.50 节.

E.47.1. Migration to Version 9.1.3

A dump/restore is not required for those running 9.1.X.

However, if you are upgrading from a version earlier than 9.1.2, see 第 E.48 节.

E.47.2. Changes

- Require execute permission on the trigger function for `CREATE TRIGGER` (Robert Haas)

  This missing check could allow another user to execute a trigger function with forged input data, by installing it on a table he owns. This is only of significance for trigger functions marked `SECURITY DEFINER`, since otherwise trigger functions run as the table owner anyway. (CVE-2012-0866)

- Remove arbitrary limitation on length of common name in SSL certificates (Heikki Linnakangas)

  Both libpq and the server truncated the common name extracted from an SSL certificate at 32 bytes. Normally this would cause nothing worse than an unexpected verification failure, but there are some rather-implausible scenarios in which it might allow one certificate holder to impersonate another. The victim would have to have a common name exactly 32 bytes long, and the attacker would have to persuade a trusted CA to issue a certificate in which the common name has that string as a prefix. Impersonating a server would also require some additional exploit to redirect client connections. (CVE-2012-0867)

- Convert newlines to spaces in names written in pg_dump comments (Robert Haas)

  pg_dump was incautious about sanitizing object names that are emitted within SQL comments in its output script. A name containing a newline would at least render the script syntactically incorrect. Maliciously crafted object names could present a SQL injection risk when the script is reloaded. (CVE-2012-0868)

- Fix btree index corruption from insertions concurrent with vacuuming (Tom Lane)

  An index page split caused by an insertion could sometimes cause a concurrently-running `VACUUM` to miss removing index entries that it should remove. After the corresponding table rows are removed, the dangling index entries would cause errors (such as "could not read block N in file ...") or worse, silently wrong query results after unrelated rows are re-inserted at the now-free table locations. This bug has been present since release 8.2, but occurs so infrequently that it was not

diagnosed until now. If you have reason to suspect that it has happened in your database, reindexing the affected index will fix things.

- Fix transient zeroing of shared buffers during WAL replay (Tom Lane)

  The replay logic would sometimes zero and refill a shared buffer, so that the contents were transiently invalid. In hot standby mode this can result in a query that's executing in parallel seeing garbage data. Various symptoms could result from that, but the most common one seems to be "invalid memory alloc request size".

- Fix handling of data-modifying `WITH` subplans in `READ COMMITTED` rechecking (Tom Lane)

  A `WITH` clause containing `INSERT/UPDATE/DELETE` would crash if the parent `UPDATE` or `DELETE` command needed to be re-evaluated at one or more rows due to concurrent updates in `READ COMMITTED` mode.

- Fix corner case in SSI transaction cleanup (Dan Ports)

  When finishing up a read-write serializable transaction, a crash could occur if all remaining active serializable transactions are read-only.

- Fix postmaster to attempt restart after a hot-standby crash (Tom Lane)

  A logic error caused the postmaster to terminate, rather than attempt to restart the cluster, if any backend process crashed while operating in hot standby mode.

- Fix `CLUSTER/VACUUM FULL` handling of toast values owned by recently-updated rows (Tom Lane)

  This oversight could lead to "duplicate key value violates unique constraint" errors being reported against the toast table's index during one of these commands.

- Update per-column permissions, not only per-table permissions, when changing table owner (Tom Lane)

  Failure to do this meant that any previously granted column permissions were still shown as having been granted by the old owner. This meant that neither the new owner nor a superuser could revoke the now-untraceable-to-table-owner permissions.

- Support foreign data wrappers and foreign servers in `REASSIGN OWNED` (Alvaro Herrera)

  This command failed with "unexpected classid" errors if it needed to change the ownership of any such objects.

- Allow non-existent values for some settings in `ALTER USER/DATABASE SET` (Heikki Linnakangas)

  Allow `default_text_search_config`, `default_tablespace`, and `temp_tablespaces` to be set to names that are not known. This is because they might be known in another database where the setting is intended to be used, or for the tablespace cases because the tablespace might not be created yet. The same issue was previously recognized for `search_path`, and these settings now act like that one.

- Fix "unsupported node type" error caused by `COLLATE` in an `INSERT` expression (Tom Lane)

- Avoid crashing when we have problems deleting table files post-commit (Tom Lane)

Dropping a table should lead to deleting the underlying disk files only after the transaction commits. In event of failure then (for instance, because of wrong file permissions) the code is supposed to just emit a warning message and go on, since it's too late to abort the transaction. This logic got broken as of release 8.4, causing such situations to result in a PANIC and an unrestartable database.

- Recover from errors occurring during WAL replay of `DROP TABLESPACE` (Tom Lane)

  Replay will attempt to remove the tablespace's directories, but there are various reasons why this might fail (for example, incorrect ownership or permissions on those directories). Formerly the replay code would panic, rendering the database unrestartable without manual intervention. It seems better to log the problem and continue, since the only consequence of failure to remove the directories is some wasted disk space.

- Fix race condition in logging AccessExclusiveLocks for hot standby (Simon Riggs)

  Sometimes a lock would be logged as being held by “transaction zero”. This is at least known to produce assertion failures on slave servers, and might be the cause of more serious problems.

- Track the OID counter correctly during WAL replay, even when it wraps around (Tom Lane)

  Previously the OID counter would remain stuck at a high value until the system exited replay mode. The practical consequences of that are usually nil, but there are scenarios wherein a standby server that's been promoted to master might take a long time to advance the OID counter to a reasonable value once values are needed.

- Prevent emitting misleading “consistent recovery state reached” log message at the beginning of crash recovery (Heikki Linnakangas)

- Fix initial value of `pg_stat_replication.replay_location` (Fujii Masao)

  Previously, the value shown would be wrong until at least one WAL record had been replayed.

- Fix regular expression back-references with `*` attached (Tom Lane)

  Rather than enforcing an exact string match, the code would effectively accept any string that satisfies the pattern sub-expression referenced by the back-reference symbol.

  A similar problem still afflicts back-references that are embedded in a larger quantified expression, rather than being the immediate subject of the quantifier. This will be addressed in a future PostgreSQL release.

- Fix recently-introduced memory leak in processing of `inet/cidr` values (Heikki Linnakangas)

  A patch in the December 2011 releases of PostgreSQL caused memory leakage in these operations, which could be significant in scenarios such as building a btree index on such a column.

- Fix planner's ability to push down index-expression restrictions through `UNION ALL` (Tom Lane)

  This type of optimization was inadvertently disabled by a fix for another problem in 9.1.2.

- Fix planning of `WITH` clauses referenced in `UPDATE/DELETE` on an inherited table (Tom Lane)

  This bug led to "could not find plan for CTE" failures.

- Fix GIN cost estimation to handle `column IN (...)` index conditions (Marti Raudsepp)

  This oversight would usually lead to crashes if such a condition could be used with a GIN index.

- Prevent assertion failure when exiting a session with an open, failed transaction (Tom Lane)

  This bug has no impact on normal builds with asserts not enabled.

- Fix dangling pointer after `CREATE TABLE AS/SELECT INTO` in a SQL-language function (Tom Lane)

  In most cases this only led to an assertion failure in assert-enabled builds, but worse consequences seem possible.

- Avoid double close of file handle in syslogger on Windows (MauMau)

  Ordinarily this error was invisible, but it would cause an exception when running on a debug version of Windows.

- Fix I/O-conversion-related memory leaks in plpgsql (Andres Freund, Jan Urbanski, Tom Lane)

  Certain operations would leak memory until the end of the current function.

- Work around bug in perl's SvPVutf8() function (Andrew Dunstan)

  This function crashes when handed a typeglob or certain read-only objects such as `$^V`. Make plperl avoid passing those to it.

- In pg_dump, don't dump contents of an extension's configuration tables if the extension itself is not being dumped (Tom Lane)

- Improve pg_dump's handling of inherited table columns (Tom Lane)

  pg_dump mishandled situations where a child column has a different default expression than its parent column. If the default is textually identical to the parent's default, but not actually the same (for instance, because of schema search path differences) it would not be recognized as different, so that after dump and restore the child would be allowed to inherit the parent's default. Child columns that are `NOT NULL` where their parent is not could also be restored subtly incorrectly.

- Fix pg_restore's direct-to-database mode for INSERT-style table data (Tom Lane)

  Direct-to-database restores from archive files made with `--inserts` or `--column-inserts` options fail when using pg_restore from a release dated September or December 2011, as a result of an oversight in a fix for another problem. The archive file itself is not at fault, and text-mode output is okay.

- Teach pg_upgrade to handle renaming of plpython's shared library (Bruce Momjian)

  Upgrading a pre-9.1 database that included plpython would fail because of this oversight.

- Allow pg_upgrade to process tables containing `regclass` columns (Bruce Momjian)

Since pg_upgrade now takes care to preserve `pg_class` OIDs, there was no longer any reason for this restriction.

- Make libpq ignore `ENOTDIR` errors when looking for an SSL client certificate file (Magnus Hagander)

  This allows SSL connections to be established, though without a certificate, even when the user's home directory is set to something like `/dev/null`.

- Fix some more field alignment issues in ecpg's SQLDA area (Zoltan Boszormenyi)

- Allow `AT` option in ecpg `DEALLOCATE` statements (Michael Meskes)

  The infrastructure to support this has been there for awhile, but through an oversight there was still an error check rejecting the case.

- Do not use the variable name when defining a varchar structure in ecpg (Michael Meskes)

- Fix `contrib/auto_explain`'s JSON output mode to produce valid JSON (Andrew Dunstan)

  The output used brackets at the top level, when it should have used braces.

- Fix error in `contrib/intarray`'s `int[] & int[]` operator (Guillaume Lelarge)

  If the smallest integer the two input arrays have in common is 1, and there are smaller values in either array, then 1 would be incorrectly omitted from the result.

- Fix error detection in `contrib/pgcrypto`'s `encrypt_iv()` and `decrypt_iv()` (Marko Kreen)

  These functions failed to report certain types of invalid-input errors, and would instead return random garbage values for incorrect input.

- Fix one-byte buffer overrun in `contrib/test_parser` (Paul Guyot)

  The code would try to read one more byte than it should, which would crash in corner cases. Since `contrib/test_parser` is only example code, this is not a security issue in itself, but bad example code is still bad.

- Use `__sync_lock_test_and_set()` for spinlocks on ARM, if available (Martin Pitt)

  This function replaces our previous use of the `SWPB` instruction, which is deprecated and not available on ARMv6 and later. Reports suggest that the old code doesn't fail in an obvious way on recent ARM boards, but simply doesn't interlock concurrent accesses, leading to bizarre failures in multiprocess operation.

- Use `-fexcess-precision=standard` option when building with gcc versions that accept it (Andrew Dunstan)

  This prevents assorted scenarios wherein recent versions of gcc will produce creative results.

- Allow use of threaded Python on FreeBSD (Chris Rees)

  Our configure script previously believed that this combination wouldn't work; but FreeBSD fixed the problem, so remove that error check.

- Allow MinGW builds to use standardly-named OpenSSL libraries (Tomasz Ostrowski)

E.48. Release 9.1.2

Release date: 2011-12-05

This release contains a variety of fixes from 9.1.1. For information about new features in the 9.1 major release, see 第 E.50 节.

E.48.1. Migration to Version 9.1.2

A dump/restore is not required for those running 9.1.X.

However, a longstanding error was discovered in the definition of the `information_schema.referential_constraints` view. If you rely on correct results from that view, you should replace its definition as explained in the first changelog item below.

Also, if you use the `citext` data type, and you upgraded from a previous major release by running pg_upgrade, you should run `CREATE EXTENSION citext FROM unpackaged` to avoid collation-related failures in `citext` operations. The same is necessary if you restore a dump from a pre-9.1 database that contains an instance of the `citext` data type. If you've already run the `CREATE EXTENSION` command before upgrading to 9.1.2, you will instead need to do manual catalog updates as explained in the second changelog item.

E.48.2. Changes

- Fix bugs in `information_schema.referential_constraints` view (Tom Lane)

  This view was being insufficiently careful about matching the foreign-key constraint to the depended-on primary or unique key constraint. That could result in failure to show a foreign key constraint at all, or showing it multiple times, or claiming that it depends on a different constraint than the one it really does.

  Since the view definition is installed by initdb, merely upgrading will not fix the problem. If you need to fix this in an existing installation, you can (as a superuser) drop the `information_schema` schema then re-create it by sourcing *`SHAREDIR`*`/information_schema.sql`. (Run `pg_config --sharedir` if you're uncertain where *`SHAREDIR`* is.) This must be repeated in each database to be fixed.

- Make `contrib/citext`'s upgrade script fix collations of `citext` columns and indexes (Tom Lane)

  Existing `citext` columns and indexes aren't correctly marked as being of a collatable data type during pg_upgrade from a pre-9.1 server, or when a pre-9.1 dump containing the `citext` type is loaded into a 9.1 server. That leads to operations on these columns failing with errors such as "could not determine which collation to use for string comparison". This change allows them to be fixed by the same script that upgrades the `citext` module into a proper 9.1 extension during `CREATE EXTENSION citext FROM unpackaged`.

  If you have a previously-upgraded database that is suffering from this problem, and you already ran the `CREATE EXTENSION` command, you can manually run (as superuser) the `UPDATE` commands found at the end of *`SHAREDIR`*`/extension/citext--unpackaged--1.0.sql`. (Run `pg_config --sharedir` if you're uncertain where *`SHAREDIR`* is.) There is no harm in doing this again if unsure.

- Fix possible crash during `UPDATE` or `DELETE` that joins to the output of a scalar-returning function (Tom Lane)

A crash could only occur if the target row had been concurrently updated, so this problem surfaced only intermittently.

- Fix incorrect replay of WAL records for GIN index updates (Tom Lane)

  This could result in transiently failing to find index entries after a crash, or on a hot-standby server. The problem would be repaired by the next `VACUUM` of the index, however.

- Fix TOAST-related data corruption during `CREATE TABLE dest AS SELECT * FROM src` or `INSERT INTO dest SELECT * FROM src` (Tom Lane)

  If a table has been modified by `ALTER TABLE ADD COLUMN`, attempts to copy its data verbatim to another table could produce corrupt results in certain corner cases. The problem can only manifest in this precise form in 8.4 and later, but we patched earlier versions as well in case there are other code paths that could trigger the same bug.

- Fix possible failures during hot standby startup (Simon Riggs)

- Start hot standby faster when initial snapshot is incomplete (Simon Riggs)

- Fix race condition during toast table access from stale syscache entries (Tom Lane)

  The typical symptom was transient errors like “missing chunk number 0 for toast value NNNNN in pg_toast_2619”, where the cited toast table would always belong to a system catalog.

- Track dependencies of functions on items used in parameter default expressions (Tom Lane)

  Previously, a referenced object could be dropped without having dropped or modified the function, leading to misbehavior when the function was used. Note that merely installing this update will not fix the missing dependency entries; to do that, you'd need to `CREATE OR REPLACE` each such function afterwards. If you have functions whose defaults depend on non-built-in objects, doing so is recommended.

- Fix incorrect management of placeholder variables in nestloop joins (Tom Lane)

  This bug is known to lead to “variable not found in subplan target list” planner errors, and could possibly result in wrong query output when outer joins are involved.

- Fix window functions that sort by expressions involving aggregates (Tom Lane)

  Previously these could fail with “could not find pathkey item to sort” planner errors.

- Fix “MergeAppend child's targetlist doesn't match MergeAppend” planner errors (Tom Lane)

- Fix index matching for operators with both collatable and noncollatable inputs (Tom Lane)

  In 9.1.0, an indexable operator that has a non-collatable left-hand input type and a collatable right-hand input type would not be recognized as matching the left-hand column's index. An example is the `hstore ? text` operator.

- Allow inlining of set-returning SQL functions with multiple OUT parameters (Tom Lane)

- Don't trust deferred-unique indexes for join removal (Tom Lane and Marti Raudsepp)

A deferred uniqueness constraint might not hold intra-transaction, so assuming that it does could give incorrect query results.

- Make `DatumGetInetP()` unpack inet datums that have a 1-byte header, and add a new macro, `DatumGetInetPP()`, that does not (Heikki Linnakangas)

  This change affects no core code, but might prevent crashes in add-on code that expects `DatumGetInetP()` to produce an unpacked datum as per usual convention.

- Improve locale support in `money` type's input and output (Tom Lane)

  Aside from not supporting all standard `lc_monetary` formatting options, the input and output functions were inconsistent, meaning there were locales in which dumped `money` values could not be re-read.

- Don't let `transform_null_equals` affect `CASE foo WHEN NULL ...` constructs (Heikki Linnakangas)

  `transform_null_equals` is only supposed to affect `foo = NULL` expressions written directly by the user, not equality checks generated internally by this form of `CASE`.

- Change foreign-key trigger creation order to better support self-referential foreign keys (Tom Lane)

  For a cascading foreign key that references its own table, a row update will fire both the `ON UPDATE` trigger and the `CHECK` trigger as one event. The `ON UPDATE` trigger must execute first, else the `CHECK` will check a non-final state of the row and possibly throw an inappropriate error. However, the firing order of these triggers is determined by their names, which generally sort in creation order since the triggers have auto-generated names following the convention "RI_ConstraintTrigger_NNNN". A proper fix would require modifying that convention, which we will do in 9.2, but it seems risky to change it in existing releases. So this patch just changes the creation order of the triggers. Users encountering this type of error should drop and re-create the foreign key constraint to get its triggers into the right order.

- Fix `IF EXISTS` to work correctly in `DROP OPERATOR FAMILY` (Robert Haas)

- Disallow dropping of an extension from within its own script (Tom Lane)

  This prevents odd behavior in case of incorrect management of extension dependencies.

- Don't mark auto-generated types as extension members (Robert Haas)

  Relation rowtypes and automatically-generated array types do not need to have their own extension membership entries in `pg_depend`, and creating such entries complicates matters for extension upgrades.

- Cope with invalid pre-existing `search_path` settings during `CREATE EXTENSION` (Tom Lane)

- Avoid floating-point underflow while tracking buffer allocation rate (Greg Matthews)

  While harmless in itself, on certain platforms this would result in annoying kernel log messages.

- Prevent autovacuum transactions from running in serializable mode (Tom Lane)

Autovacuum formerly used the cluster-wide default transaction isolation level, but there is no need for it to use anything higher than READ COMMITTED, and using SERIALIZABLE could result in unnecessary delays for other processes.

- Ensure walsender processes respond promptly to SIGTERM (Magnus Hagander)
- Exclude `postmaster.opts` from base backups (Magnus Hagander)
- Preserve configuration file name and line number values when starting child processes under Windows (Tom Lane)

  Formerly, these would not be displayed correctly in the `pg_settings` view.
- Fix incorrect field alignment in ecpg's SQLDA area (Zoltan Boszormenyi)
- Preserve blank lines within commands in psql's command history (Robert Haas)

  The former behavior could cause problems if an empty line was removed from within a string literal, for example.
- Avoid platform-specific infinite loop in pg_dump (Steve Singer)
- Fix compression of plain-text output format in pg_dump (Adrian Klaver and Tom Lane)

  pg_dump has historically understood `-Z` with no `-F` switch to mean that it should emit a gzip-compressed version of its plain text output. Restore that behavior.
- Fix pg_dump to dump user-defined casts between auto-generated types, such as table rowtypes (Tom Lane)
- Fix missed quoting of foreign server names in pg_dump (Tom Lane)
- Assorted fixes for pg_upgrade (Bruce Momjian)

  Handle exclusion constraints correctly, avoid failures on Windows, don't complain about mismatched toast table names in 8.4 databases.
- In PL/pgSQL, allow foreign tables to define row types (Alexander Soudakov)
- Fix up conversions of PL/Perl functions' results (Alex Hunsaker and Tom Lane)

  Restore the pre-9.1 behavior that PL/Perl functions returning `void` ignore the result value of their last Perl statement; 9.1.0 would throw an error if that statement returned a reference. Also, make sure it works to return a string value for a composite type, so long as the string meets the type's input format. In addition, throw errors for attempts to return Perl arrays or hashes when the function's declared result type is not an array or composite type, respectively. (Pre-9.1 versions rather uselessly returned strings like `ARRAY(0x221a9a0)` or `HASH(0x221aa90)` in such cases.)
- Ensure PL/Perl strings are always correctly UTF8-encoded (Amit Khandekar and Alex Hunsaker)
- Use the preferred version of xsubpp to build PL/Perl, not necessarily the operating system's main copy (David Wheeler and Alex Hunsaker)
- Correctly propagate SQLSTATE in PL/Python exceptions (Mika Eloranta and Jan Urbanski)
- Do not install PL/Python extension files for Python major versions other than the one built against (Peter Eisentraut)

- Change all the `contrib` extension script files to report a useful error message if they are fed to psql (Andrew Dunstan and Tom Lane)

  This should help teach people about the new method of using `CREATE EXTENSION` to load these files. In most cases, sourcing the scripts directly would fail anyway, but with harder-to-interpret messages.

- Fix incorrect coding in `contrib/dict_int` and `contrib/dict_xsyn` (Tom Lane)

  Some functions incorrectly assumed that memory returned by `palloc()` is guaranteed zeroed.

- Remove `contrib/sepgsql` tests from the regular regression test mechanism (Tom Lane)

  Since these tests require root privileges for setup, they're impractical to run automatically. Switch over to a manual approach instead, and provide a testing script to help with that.

- Fix assorted errors in `contrib/unaccent`'s configuration file parsing (Tom Lane)

- Honor query cancel interrupts promptly in `pgstatindex()` (Robert Haas)

- Fix incorrect quoting of log file name in macOS start script (Sidar Lopez)

- Revert unintentional enabling of `WAL_DEBUG` (Robert Haas)

  Fortunately, as debugging tools go, this one is pretty cheap; but it's not intended to be enabled by default, so revert.

- Ensure VPATH builds properly install all server header files (Peter Eisentraut)

- Shorten file names reported in verbose error messages (Peter Eisentraut)

  Regular builds have always reported just the name of the C file containing the error message call, but VPATH builds formerly reported an absolute path name.

- Fix interpretation of Windows timezone names for Central America (Tom Lane)

  Map "Central America Standard Time" to `CST6`, not `CST6CDT`, because DST is generally not observed anywhere in Central America.

- Update time zone data files to tzdata release 2011n for DST law changes in Brazil, Cuba, Fiji, Palestine, Russia, and Samoa; also historical corrections for Alaska and British East Africa.

E.49. Release 9.1.1

Release date: 2011-09-26

This release contains a small number of fixes from 9.1.0. For information about new features in the 9.1 major release, see 第 E.50 节.

E.49.1. Migration to Version 9.1.1

A dump/restore is not required for those running 9.1.X.

E.49.2. Changes

- Make `pg_options_to_table` return NULL for an option with no value (Tom Lane)

Previously such cases would result in a server crash.

- Fix memory leak at end of a GiST index scan (Tom Lane)

  Commands that perform many separate GiST index scans, such as verification of a new GiST-based exclusion constraint on a table already containing many rows, could transiently require large amounts of memory due to this leak.

- Fix explicit reference to `pg_temp` schema in `CREATE TEMPORARY TABLE` (Robert Haas)

  This used to be allowed, but failed in 9.1.0.

E.50. Release 9.1

Release date: 2011-09-12

E.50.1. Overview

This release shows PostgreSQL moving beyond the traditional relational-database feature set with new, ground-breaking functionality that is unique to PostgreSQL. The streaming replication feature introduced in release 9.0 is significantly enhanced by adding a synchronous-replication option, streaming backups, and monitoring improvements. Major enhancements include:

- Allow synchronous replication
- Add support for foreign tables
- Add per-column collation support
- Add extensions which simplify packaging of additions to PostgreSQL
- Add a true serializable isolation level
- Support unlogged tables using the `UNLOGGED` option in `CREATE TABLE`
- Allow data-modification commands (`INSERT`/`UPDATE`/`DELETE`) in `WITH` clauses
- Add nearest-neighbor (order-by-operator) searching to GiST indexes
- Add a `SECURITY LABEL` command and support for SELinux permissions control
- Update the PL/Python server-side language

The above items are explained in more detail in the sections below.

E.50.2. Migration to Version 9.1

A dump/restore using pg_dump, or use of pg_upgrade, is required for those wishing to migrate data from any previous release.

Version 9.1 contains a number of changes that may affect compatibility with previous releases. Observe the following incompatibilities:

E.50.2.1. Strings

- Change the default value of `standard_conforming_strings` to on (Robert Haas)

By default, backslashes are now ordinary characters in string literals, not escape characters. This change removes a long-standing incompatibility with the SQL standard. `escape_string_warning` has produced warnings about this usage for years. `E''` strings are the proper way to embed backslash escapes in strings and are unaffected by this change.

> **警告**
>
> This change can break applications that are not expecting it and do their own string escaping according to the old rules. The consequences could be as severe as introducing SQL-injection security holes. Be sure to test applications that are exposed to untrusted input, to ensure that they correctly handle single quotes and backslashes in text strings.

E.50.2.2. Casting

- Disallow function-style and attribute-style data type casts for composite types (Tom Lane)

  For example, disallow *composite_value*`.text` and `text(`*composite_value*`)`. Unintentional uses of this syntax have frequently resulted in bug reports; although it was not a bug, it seems better to go back to rejecting such expressions. The `CAST` and `::` syntaxes are still available for use when a cast of an entire composite value is actually intended.

- Tighten casting checks for domains based on arrays (Tom Lane)

  When a domain is based on an array type, it is allowed to "look through" the domain type to access the array elements, including subscripting the domain value to fetch or assign an element. Assignment to an element of such a domain value, for instance via `UPDATE ... SET domaincol[5] = ...`, will now result in rechecking the domain type's constraints, whereas before the checks were skipped.

E.50.2.3. Arrays

- Change `string_to_array()` to return an empty array for a zero-length string (Pavel Stehule)

  Previously this returned a null value.

- Change `string_to_array()` so a `NULL` separator splits the string into characters (Pavel Stehule)

  Previously this returned a null value.

E.50.2.4. Object Modification

- Fix improper checks for before/after triggers (Tom Lane)

  Triggers can now be fired in three cases: `BEFORE`, `AFTER`, or `INSTEAD OF` some action. Trigger function authors should verify that their logic behaves sanely in all three cases.

- Require superuser or `CREATEROLE` permissions in order to set comments on roles (Tom Lane)

E.50.2.5. Server Settings

- Change `pg_last_xlog_receive_location()` so it never moves backwards (Fujii Masao)

  Previously, the value of `pg_last_xlog_receive_location()` could move backward when streaming replication is restarted.

- Have logging of replication connections honor `log_connections` (Magnus Hagander)

  Previously, replication connections were always logged.

E.50.2.6. PL/pgSQL Server-Side Language

- Change PL/pgSQL's `RAISE` command without parameters to be catchable by the attached exception block (Piyush Newe)

  Previously `RAISE` in a code block was always scoped to an attached exception block, so it was uncatchable at the same scope.

- Adjust PL/pgSQL's error line numbering code to be consistent with other PLs (Pavel Stehule)

  Previously, PL/pgSQL would ignore (not count) an empty line at the start of the function body. Since this was inconsistent with all other languages, the special case was removed.

- Make PL/pgSQL complain about conflicting IN and OUT parameter names (Tom Lane)

  Formerly, the collision was not detected, and the name would just silently refer to only the OUT parameter.

- Type modifiers of PL/pgSQL variables are now visible to the SQL parser (Tom Lane)

  A type modifier (such as a varchar length limit) attached to a PL/pgSQL variable was formerly enforced during assignments, but was ignored for all other purposes. Such variables will now behave more like table columns declared with the same modifier. This is not expected to make any visible difference in most cases, but it could result in subtle changes for some SQL commands issued by PL/pgSQL functions.

E.50.2.7. Contrib

- All contrib modules are now installed with `CREATE EXTENSION` rather than by manually invoking their SQL scripts (Dimitri Fontaine, Tom Lane)

  To update an existing database containing the 9.0 version of a contrib module, use `CREATE EXTENSION ... FROM unpackaged` to wrap the existing contrib module's objects into an extension. When updating from a pre-9.0 version, drop the contrib module's objects using its old uninstall script, then use `CREATE EXTENSION`.

E.50.2.8. Other Incompatibilities

- Make `pg_stat_reset()` reset all database-level statistics (Tomas Vondra)

  Some `pg_stat_database` counters were not being reset.

- Fix some `information_schema.triggers` column names to match the new SQL-standard names (Dean Rasheed)

- Treat ECPG cursor names as case-insensitive (Zoltan Boszormenyi)

E.50.3. Changes

Below you will find a detailed account of the changes between PostgreSQL 9.1 and the previous major release.

E.50.3.1. Server

E.50.3.1.1. Performance

- Support unlogged tables using the `UNLOGGED` option in `CREATE TABLE` (Robert Haas)

  Such tables provide better update performance than regular tables, but are not crash-safe: their contents are automatically cleared in case of a server crash. Their contents do not propagate to replication slaves, either.

- Allow `FULL OUTER JOIN` to be implemented as a hash join, and allow either side of a `LEFT OUTER JOIN` or `RIGHT OUTER JOIN` to be hashed (Tom Lane)

  Previously `FULL OUTER JOIN` could only be implemented as a merge join, and `LEFT OUTER JOIN` and `RIGHT OUTER JOIN` could hash only the nullable side of the join. These changes provide additional query optimization possibilities.

- Merge duplicate fsync requests (Robert Haas, Greg Smith)

  This greatly improves performance under heavy write loads.

- Improve performance of `commit_siblings` (Greg Smith)

  This allows the use of `commit_siblings` with less overhead.

- Reduce the memory requirement for large ispell dictionaries (Pavel Stehule, Tom Lane)

- Avoid leaving data files open after "blind writes" (Alvaro Herrera)

  This fixes scenarios in which backends might hold files open long after they were deleted, preventing the kernel from reclaiming disk space.

E.50.3.1.2. Optimizer

- Allow inheritance table scans to return meaningfully-sorted results (Greg Stark, Hans-Jurgen Schonig, Robert Haas, Tom Lane)

  This allows better optimization of queries that use `ORDER BY`, `LIMIT`, or `MIN`/`MAX` with inherited tables.

- Improve GIN index scan cost estimation (Teodor Sigaev)

- Improve cost estimation for aggregates and window functions (Tom Lane)

E.50.3.1.3. Authentication

- Support host names and host suffixes (e.g. `.example.com`) in `pg_hba.conf` (Peter Eisentraut)

  Previously only host IP addresses and CIDR values were supported.

- Support the key word `all` in the host column of `pg_hba.conf` (Peter Eisentraut)

Previously people used `0.0.0.0/0` or `::/0` for this.

- Reject `local` lines in `pg_hba.conf` on platforms that don't support Unix-socket connections (Magnus Hagander)

  Formerly, such lines were silently ignored, which could be surprising. This makes the behavior more like other unsupported cases.

- Allow GSSAPI to be used to authenticate to servers via SSPI (Christian Ullrich)

  Specifically this allows Unix-based GSSAPI clients to do SSPI authentication with Windows servers.

- `ident` authentication over local sockets is now known as `peer` (Magnus Hagander)

  The old term is still accepted for backward compatibility, but since the two methods are fundamentally different, it seemed better to adopt different names for them.

- Rewrite peer authentication to avoid use of credential control messages (Tom Lane)

  This change makes the peer authentication code simpler and better-performing. However, it requires the platform to provide the `getpeereid` function or an equivalent socket operation. So far as is known, the only platform for which peer authentication worked before and now will not is pre-5.0 NetBSD.

E.50.3.1.4. Monitoring

- Add details to the logging of restartpoints and checkpoints, which is controlled by `log_checkpoints` (Fujii Masao, Greg Smith)

  New details include WAL file and sync activity.

- Add `log_file_mode` which controls the permissions on log files created by the logging collector (Martin Pihlak)

- Reduce the default maximum line length for syslog logging to 900 bytes plus prefixes (Noah Misch)

  This avoids truncation of long log lines on syslog implementations that have a 1KB length limit, rather than the more common 2KB.

E.50.3.1.5. Statistical Views

- Add `client_hostname` column to `pg_stat_activity` (Peter Eisentraut)

  Previously only the client address was reported.

- Add `pg_stat_xact_*` statistics functions and views (Joel Jacobson)

  These are like the database-wide statistics counter views, but reflect counts for only the current transaction.

- Add time of last reset in database-level and background writer statistics views (Tomas Vondra)

- Add columns showing the number of vacuum and analyze operations in `pg_stat_*_tables` views (Magnus Hagander)

- Add `buffers_backend_fsync` column to `pg_stat_bgwriter` (Greg Smith)

  This new column counts the number of times a backend fsyncs a buffer.

E.50.3.1.6. Server Settings

- Provide auto-tuning of `wal_buffers` (Greg Smith)

  By default, the value of `wal_buffers` is now chosen automatically based on the value of `shared_buffers`.

- Increase the maximum values for `deadlock_timeout`, `log_min_duration_statement`, and `log_autovacuum_min_duration` (Peter Eisentraut)

  The maximum value for each of these parameters was previously only about 35 minutes. Much larger values are now allowed.

E.50.3.2. Replication and Recovery

E.50.3.2.1. Streaming Replication and Continuous Archiving

- Allow synchronous replication (Simon Riggs, Fujii Masao)

  This allows the primary server to wait for a standby to write a transaction's information to disk before acknowledging the commit. One standby at a time can take the role of the synchronous standby, as controlled by the `synchronous_standby_names` setting. Synchronous replication can be enabled or disabled on a per-transaction basis using the `synchronous_commit` setting.

- Add protocol support for sending file system backups to standby servers using the streaming replication network connection (Magnus Hagander, Heikki Linnakangas)

  This avoids the requirement of manually transferring a file system backup when setting up a standby server.

- Add `replication_timeout` setting (Fujii Masao, Heikki Linnakangas)

  Replication connections that are idle for more than the `replication_timeout` interval will be terminated automatically. Formerly, a failed connection was typically not detected until the TCP timeout elapsed, which is inconveniently long in many situations.

- Add command-line tool pg_basebackup for creating a new standby server or database backup (Magnus Hagander)

- Add a replication permission for roles (Magnus Hagander)

  This is a read-only permission used for streaming replication. It allows a non-superuser role to be used for replication connections. Previously only superusers could initiate replication connections; superusers still have this permission by default.

E.50.3.2.2. Replication Monitoring

- Add system view `pg_stat_replication` which displays activity of WAL sender processes (Itagaki Takahiro, Simon Riggs)

  This reports the status of all connected standby servers.

- Add monitoring function `pg_last_xact_replay_timestamp()` (Fujii Masao)

  This returns the time at which the primary generated the most recent commit or abort record applied on the standby.

E.50.3.2.3. Hot Standby

- Add configuration parameter `hot_standby_feedback` to enable standbys to postpone cleanup of old row versions on the primary (Simon Riggs)

  This helps avoid canceling long-running queries on the standby.

- Add the `pg_stat_database_conflicts` system view to show queries that have been canceled and the reason (Magnus Hagander)

  Cancellations can occur because of dropped tablespaces, lock timeouts, old snapshots, pinned buffers, and deadlocks.

- Add a `conflicts` count to `pg_stat_database` (Magnus Hagander)

  This is the number of conflicts that occurred in the database.

- Increase the maximum values for `max_standby_archive_delay` and `max_standby_streaming_delay`

  The maximum value for each of these parameters was previously only about 35 minutes. Much larger values are now allowed.

- Add `ERRCODE_T_R_DATABASE_DROPPED` error code to report recovery conflicts due to dropped databases (Tatsuo Ishii)

  This is useful for connection pooling software.

E.50.3.2.4. Recovery Control

- Add functions to control streaming replication replay (Simon Riggs)

  The new functions are `pg_xlog_replay_pause()`, `pg_xlog_replay_resume()`, and the status function `pg_is_xlog_replay_paused()`.

- Add `recovery.conf` setting `pause_at_recovery_target` to pause recovery at target (Simon Riggs)

  This allows a recovery server to be queried to check whether the recovery point is the one desired.

- Add the ability to create named restore points using `pg_create_restore_point()` (Jaime Casanova)

  These named restore points can be specified as recovery targets using the new `recovery.conf` setting `recovery_target_name`.

- Allow standby recovery to switch to a new timeline automatically (Heikki Linnakangas)

  Now standby servers scan the archive directory for new timelines periodically.

- Add `restart_after_crash` setting which disables automatic server restart after a backend crash (Robert Haas)

  This allows external cluster management software to control whether the database server restarts or not.

- Allow `recovery.conf` to use the same quoting behavior as `postgresql.conf` (Dimitri Fontaine)

Previously all values had to be quoted.

E.50.3.3. Queries

- Add a true serializable isolation level (Kevin Grittner, Dan Ports)

  Previously, asking for serializable isolation guaranteed only that a single MVCC snapshot would be used for the entire transaction, which allowed certain documented anomalies. The old snapshot isolation behavior is still available by requesting the `REPEATABLE READ` isolation level.

- Allow data-modification commands (`INSERT`/`UPDATE`/`DELETE`) in `WITH` clauses (Marko Tiikkaja, Hitoshi Harada)

  These commands can use `RETURNING` to pass data up to the containing query.

- Allow `WITH` clauses to be attached to `INSERT`, `UPDATE`, `DELETE` statements (Marko Tiikkaja, Hitoshi Harada)

- Allow non-`GROUP BY` columns in the query target list when the primary key is specified in the `GROUP BY` clause (Peter Eisentraut)

  The SQL standard allows this behavior, and because of the primary key, the result is unambiguous.

- Allow use of the key word `DISTINCT` in `UNION`/`INTERSECT`/`EXCEPT` clauses (Tom Lane)

  `DISTINCT` is the default behavior so use of this key word is redundant, but the SQL standard allows it.

- Fix ordinary queries with rules to use the same snapshot behavior as `EXPLAIN ANALYZE` (Marko Tiikkaja)

  Previously `EXPLAIN ANALYZE` used slightly different snapshot timing for queries involving rules. The `EXPLAIN ANALYZE` behavior was judged to be more logical.

E.50.3.3.1. Strings

- Add per-column collation support (Peter Eisentraut, Tom Lane)

  Previously collation (the sort ordering of text strings) could only be chosen at database creation. Collation can now be set per column, domain, index, or expression, via the SQL-standard `COLLATE` clause.

E.50.3.4. Object Manipulation

- Add extensions which simplify packaging of additions to PostgreSQL (Dimitri Fontaine, Tom Lane)

  Extensions are controlled by the new `CREATE`/`ALTER`/`DROP EXTENSION` commands. This replaces ad-hoc methods of grouping objects that are added to a PostgreSQL installation.

- Add support for foreign tables (Shigeru Hanada, Robert Haas, Jan Urbanski, Heikki Linnakangas)

  This allows data stored outside the database to be used like native PostgreSQL-stored data. Foreign tables are currently read-only, however.

- Allow new values to be added to an existing enum type via `ALTER TYPE` (Andrew Dunstan)
- Add `ALTER TYPE ... ADD/DROP/ALTER/RENAME ATTRIBUTE` (Peter Eisentraut)

  This allows modification of composite types.

E.50.3.4.1. `ALTER` Object

- Add `RESTRICT/CASCADE` to `ALTER TYPE` operations on typed tables (Peter Eisentraut)

  This controls `ADD/DROP/ALTER/RENAME ATTRIBUTE` cascading behavior.
- Support `ALTER TABLE name {OF | NOT OF} type` (Noah Misch)

  This syntax allows a standalone table to be made into a typed table, or a typed table to be made standalone.
- Add support for more object types in `ALTER ... SET SCHEMA` commands (Dimitri Fontaine)

  This command is now supported for conversions, operators, operator classes, operator families, text search configurations, text search dictionaries, text search parsers, and text search templates.

E.50.3.4.2. `CREATE/ALTER TABLE`

- Add `ALTER TABLE ... ADD UNIQUE/PRIMARY KEY USING INDEX` (Gurjeet Singh)

  This allows a primary key or unique constraint to be defined using an existing unique index, including a concurrently created unique index.
- Allow `ALTER TABLE` to add foreign keys without validation (Simon Riggs)

  The new option is called `NOT VALID`. The constraint's state can later be modified to `VALIDATED` and validation checks performed. Together these allow you to add a foreign key with minimal impact on read and write operations.
- Allow `ALTER TABLE ... SET DATA TYPE` to avoid table rewrites in appropriate cases (Noah Misch, Robert Haas)

  For example, converting a `varchar` column to `text` no longer requires a rewrite of the table. However, increasing the length constraint on a `varchar` column still requires a table rewrite.
- Add `CREATE TABLE IF NOT EXISTS` syntax (Robert Haas)

  This allows table creation without causing an error if the table already exists.
- Fix possible "tuple concurrently updated" error when two backends attempt to add an inheritance child to the same table at the same time (Robert Haas)

  `ALTER TABLE` now takes a stronger lock on the parent table, so that the sessions cannot try to update it simultaneously.

E.50.3.4.3. Object Permissions

- Add a `SECURITY LABEL` command (KaiGai Kohei)

  This allows security labels to be assigned to objects.

E.50.3.5. Utility Operations

- Add transaction-level advisory locks (Marko Tiikkaja)

  These are similar to the existing session-level advisory locks, but such locks are automatically released at transaction end.

- Make `TRUNCATE ... RESTART IDENTITY` restart sequences transactionally (Steve Singer)

  Previously the counter could have been left out of sync if a backend crashed between the on-commit truncation activity and commit completion.

E.50.3.5.1. COPY

- Add `ENCODING` option to `COPY TO/FROM` (Hitoshi Harada, Itagaki Takahiro)

  This allows the encoding of the `COPY` file to be specified separately from client encoding.

- Add bidirectional `COPY` protocol support (Fujii Masao)

  This is currently only used by streaming replication.

E.50.3.5.2. EXPLAIN

- Make `EXPLAIN VERBOSE` show the function call expression in a `FunctionScan` node (Tom Lane)

E.50.3.5.3. VACUUM

- Add additional details to the output of `VACUUM FULL VERBOSE` and `CLUSTER VERBOSE` (Itagaki Takahiro)

  New information includes the live and dead tuple count and whether `CLUSTER` is using an index to rebuild.

- Prevent autovacuum from waiting if it cannot acquire a table lock (Robert Haas)

  It will try to vacuum that table later.

E.50.3.5.4. CLUSTER

- Allow `CLUSTER` to sort the table rather than scanning the index when it seems likely to be cheaper (Leonardo Francalanci)

E.50.3.5.5. Indexes

- Add nearest-neighbor (order-by-operator) searching to GiST indexes (Teodor Sigaev, Tom Lane)

  This allows GiST indexes to quickly return the *N* closest values in a query with `LIMIT`. For example

  ```
  SELECT * FROM places ORDER BY location <-> point '(101,456)'
   LIMIT 10;
  ```

  finds the ten places closest to a given target point.

- Allow GIN indexes to index null and empty values (Tom Lane)

This allows full GIN index scans, and fixes various corner cases in which GIN scans would fail.

- Allow GIN indexes to better recognize duplicate search entries (Tom Lane)

  This reduces the cost of index scans, especially in cases where it avoids unnecessary full index scans.

- Fix GiST indexes to be fully crash-safe (Heikki Linnakangas)

  Previously there were rare cases where a `REINDEX` would be required (you would be informed).

E.50.3.6. Data Types

- Allow `numeric` to use a more compact, two-byte header in common cases (Robert Haas)

  Previously all `numeric` values had four-byte headers; this change saves on disk storage.

- Add support for dividing `money` by `money` (Andy Balholm)
- Allow binary I/O on type `void` (Radoslaw Smogura)
- Improve hypotenuse calculations for geometric operators (Paul Matthews)

  This avoids unnecessary overflows, and may also be more accurate.

- Support hashing array values (Tom Lane)

  This provides additional query optimization possibilities.

- Don't treat a composite type as sortable unless all its column types are sortable (Tom Lane)

  This avoids possible "could not identify a comparison function" failures at runtime, if it is possible to implement the query without sorting. Also, `ANALYZE` won't try to use inappropriate statistics-gathering methods for columns of such composite types.

E.50.3.6.1. Casting

- Add support for casting between `money` and `numeric` (Andy Balholm)
- Add support for casting from `int4` and `int8` to `money` (Joey Adams)
- Allow casting a table's row type to the table's supertype if it's a typed table (Peter Eisentraut)

  This is analogous to the existing facility that allows casting a row type to a supertable's row type.

E.50.3.6.2. XML

- Add XML function `XMLEXISTS` and `xpath_exists()` functions (Mike Fowler)

  These are used for XPath matching.

- Add XML functions `xml_is_well_formed()`, `xml_is_well_formed_document()`, `xml_is_well_formed_content()` (Mike Fowler)

These check whether the input is properly-formed XML. They provide functionality that was previously available only in the deprecated `contrib/xml2` module.

E.50.3.7. Functions

- Add SQL function `format(text, ...)`, which behaves analogously to C's `printf()` (Pavel Stehule, Robert Haas)

  It currently supports formats for strings, SQL literals, and SQL identifiers.

- Add string functions `concat()`, `concat_ws()`, `left()`, `right()`, and `reverse()` (Pavel Stehule)

  These improve compatibility with other database products.

- Add function `pg_read_binary_file()` to read binary files (Dimitri Fontaine, Itagaki Takahiro)

- Add a single-parameter version of function `pg_read_file()` to read an entire file (Dimitri Fontaine, Itagaki Takahiro)

- Add three-parameter forms of `array_to_string()` and `string_to_array()` for null value processing control (Pavel Stehule)

E.50.3.7.1. Object Information Functions

- Add the `pg_describe_object()` function (Alvaro Herrera)

  This function is used to obtain a human-readable string describing an object, based on the `pg_class` OID, object OID, and sub-object ID. It can be used to help interpret the contents of `pg_depend`.

- Update comments for built-in operators and their underlying functions (Tom Lane)

  Functions that are meant to be used via an associated operator are now commented as such.

- Add variable `quote_all_identifiers` to force the quoting of all identifiers in `EXPLAIN` and in system catalog functions like `pg_get_viewdef()` (Robert Haas)

  This makes exporting schemas to tools and other databases with different quoting rules easier.

- Add columns to the `information_schema.sequences` system view (Peter Eisentraut)

  Previously, though the view existed, the columns about the sequence parameters were unimplemented.

- Allow `public` as a pseudo-role name in `has_table_privilege()` and related functions (Alvaro Herrera)

  This allows checking for public permissions.

E.50.3.7.2. Function and Trigger Creation

- Support `INSTEAD OF` triggers on views (Dean Rasheed)

  This feature can be used to implement fully updatable views.

E.50.3.8. Server-Side Languages

E.50.3.8.1. PL/pgSQL Server-Side Language

- Add `FOREACH IN ARRAY` to PL/pgSQL (Pavel Stehule)

  This is more efficient and readable than previous methods of iterating through the elements of an array value.

- Allow `RAISE` without parameters to be caught in the same places that could catch a `RAISE ERROR` from the same location (Piyush Newe)

  The previous coding threw the error from the block containing the active exception handler. The new behavior is more consistent with other DBMS products.

E.50.3.8.2. PL/Perl Server-Side Language

- Allow generic record arguments to PL/Perl functions (Andrew Dunstan)

  PL/Perl functions can now be declared to accept type `record`. The behavior is the same as for any named composite type.

- Convert PL/Perl array arguments to Perl arrays (Alexey Klyukin, Alex Hunsaker)

  String representations are still available.

- Convert PL/Perl composite-type arguments to Perl hashes (Alexey Klyukin, Alex Hunsaker)

  String representations are still available.

E.50.3.8.3. PL/Python Server-Side Language

- Add table function support for PL/Python (Jan Urbanski)

  PL/Python can now return multiple `OUT` parameters and record sets.

- Add a validator to PL/Python (Jan Urbanski)

  This allows PL/Python functions to be syntax-checked at function creation time.

- Allow exceptions for SQL queries in PL/Python (Jan Urbanski)

  This allows access to SQL-generated exception error codes from PL/Python exception blocks.

- Add explicit subtransactions to PL/Python (Jan Urbanski)

- Add PL/Python functions for quoting strings (Jan Urbanski)

  These functions are `plpy.quote_ident`, `plpy.quote_literal`, and `plpy.quote_nullable`.

- Add traceback information to PL/Python errors (Jan Urbanski)

- Report PL/Python errors from iterators with `PLy_elog` (Jan Urbanski)

- Fix exception handling with Python 3 (Jan Urbanski)

  Exception classes were previously not available in `plpy` under Python 3.

E.50.3.9. Client Applications

- Mark createlang and droplang as deprecated now that they just invoke extension commands (Tom Lane)

E.50.3.9.1. psql

- Add psql command `\conninfo` to show current connection information (David Christensen)
- Add psql command `\sf` to show a function's definition (Pavel Stehule)
- Add psql command `\dL` to list languages (Fernando Ike)
- Add the `S` (“system”) option to psql's `\dn` (list schemas) command (Tom Lane)

  `\dn` without `S` now suppresses system schemas.
- Allow psql's `\e` and `\ef` commands to accept a line number to be used to position the cursor in the editor (Pavel Stehule)

  This is passed to the editor according to the `PSQL_EDITOR_LINENUMBER_ARG` environment variable.
- Have psql set the client encoding from the operating system locale by default (Heikki Linnakangas)

  This only happens if the `PGCLIENTENCODING` environment variable is not set.
- Make `\d` distinguish between unique indexes and unique constraints (Josh Kupershmidt)
- Make `\dt+` report `pg_table_size` instead of `pg_relation_size` when talking to 9.0 or later servers (Bernd Helmle)

  This is a more useful measure of table size, but note that it is not identical to what was previously reported in the same display.
- Additional tab completion support (Itagaki Takahiro, Pavel Stehule, Andrey Popp, Christoph Berg, David Fetter, Josh Kupershmidt)

E.50.3.9.2. pg_dump

- Add pg_dump and pg_dumpall option `--quote-all-identifiers` to force quoting of all identifiers (Robert Haas)
- Add `directory` format to pg_dump (Joachim Wieland, Heikki Linnakangas)

  This is internally similar to the `tar` pg_dump format.

E.50.3.9.3. pg_ctl

- Fix pg_ctl so it no longer incorrectly reports that the server is not running (Bruce Momjian)

  Previously this could happen if the server was running but pg_ctl could not authenticate.
- Improve pg_ctl start's “wait” (`-w`) option (Bruce Momjian, Tom Lane)

The wait mode is now significantly more robust. It will not get confused by non-default postmaster port numbers, non-default Unix-domain socket locations, permission problems, or stale postmaster lock files.

- Add `promote` option to pg_ctl to switch a standby server to primary (Fujii Masao)

E.50.3.10. Development Tools

E.50.3.10.1. libpq

- Add a libpq connection option `client_encoding` which behaves like the `PGCLIENTENCODING` environment variable (Heikki Linnakangas)

  The value `auto` sets the client encoding based on the operating system locale.

- Add `PQlibVersion()` function which returns the libpq library version (Magnus Hagander)

  libpq already had `PQserverVersion()` which returns the server version.

- Allow libpq-using clients to check the user name of the server process when connecting via Unix-domain sockets, with the new `requirepeer` connection option (Peter Eisentraut)

  PostgreSQL already allowed servers to check the client user name when connecting via Unix-domain sockets.

- Add `PQping()` and `PQpingParams()` to libpq (Bruce Momjian, Tom Lane)

  These functions allow detection of the server's status without trying to open a new session.

E.50.3.10.2. ECPG

- Allow ECPG to accept dynamic cursor names even in `WHERE CURRENT OF` clauses (Zoltan Boszormenyi)
- Make ecpglib write `double` values with a precision of 15 digits, not 14 as formerly (Akira Kurosawa)

E.50.3.11. Build Options

- Use `+Olibmerrno` compile flag with HP-UX C compilers that accept it (Ibrar Ahmed)

  This avoids possible misbehavior of math library calls on recent HP platforms.

E.50.3.11.1. Makefiles

- Improved parallel make support (Peter Eisentraut)

  This allows for faster compiles. Also, `make -k` now works more consistently.

- Require GNU make 3.80 or newer (Peter Eisentraut)

  This is necessary because of the parallel-make improvements.

- Add `make maintainer-check` target (Peter Eisentraut)

  This target performs various source code checks that are not appropriate for either the build or the regression tests. Currently: duplicate_oids, SGML syntax and tabs check, NLS syntax check.

- Support `make check` in `contrib` (Peter Eisentraut)

  Formerly only `make installcheck` worked, but now there is support for testing in a temporary installation. The top-level `make check-world` target now includes testing `contrib` this way.

E.50.3.11.2. Windows

- On Windows, allow pg_ctl to register the service as auto-start or start-on-demand (Quan Zongliang)
- Add support for collecting crash dumps on Windows (Craig Ringer, Magnus Hagander)

  minidumps can now be generated by non-debug Windows binaries and analyzed by standard debugging tools.
- Enable building with the MinGW64 compiler (Andrew Dunstan)

  This allows building 64-bit Windows binaries even on non-Windows platforms via cross-compiling.

E.50.3.12. Source Code

- Revise the API for GUC variable assign hooks (Tom Lane)

  The previous functions of assign hooks are now split between check hooks and assign hooks, where the former can fail but the latter shouldn't. This change will impact add-on modules that define custom GUC parameters.
- Add latches to the source code to support waiting for events (Heikki Linnakangas)
- Centralize data modification permissions-checking logic (KaiGai Kohei)
- Add missing `get_object_oid()` functions, for consistency (Robert Haas)
- Improve ability to use C++ compilers for compiling add-on modules by removing conflicting key words (Tom Lane)
- Add support for DragonFly BSD (Rumko)
- Expose `quote_literal_cstr()` for backend use (Robert Haas)
- Run regression tests in the default encoding (Peter Eisentraut)

  Regression tests were previously always run with `SQL_ASCII` encoding.
- Add src/tools/git_changelog to replace cvs2cl and pgcvslog (Robert Haas, Tom Lane)
- Add git-external-diff script to `src/tools` (Bruce Momjian)

  This is used to generate context diffs from git.
- Improve support for building with Clang (Peter Eisentraut)

E.50.3.12.1. Server Hooks

- Add source code hooks to check permissions (Robert Haas, Stephen Frost)
- Add post-object-creation function hooks for use by security frameworks (KaiGai Kohei)

- Add a client authentication hook (KaiGai Kohei)

E.50.3.13. Contrib

- Modify `contrib` modules and procedural languages to install via the new extension mechanism (Tom Lane, Dimitri Fontaine)
- Add `contrib/file_fdw` foreign-data wrapper (Shigeru Hanada)

  Foreign tables using this foreign data wrapper can read flat files in a manner very similar to `COPY`.
- Add nearest-neighbor search support to `contrib/pg_trgm` and `contrib/btree_gist` (Teodor Sigaev)
- Add `contrib/btree_gist` support for searching on not-equals (Jeff Davis)
- Fix `contrib/fuzzystrmatch`'s `levenshtein()` function to handle multibyte characters (Alexander Korotkov)
- Add `ssl_cipher()` and `ssl_version()` functions to `contrib/sslinfo` (Robert Haas)
- Fix `contrib/intarray` and `contrib/hstore` to give consistent results with indexed empty arrays (Tom Lane)

  Previously an empty-array query that used an index might return different results from one that used a sequential scan.
- Allow `contrib/intarray` to work properly on multidimensional arrays (Tom Lane)
- In `contrib/intarray`, avoid errors complaining about the presence of nulls in cases where no nulls are actually present (Tom Lane)
- In `contrib/intarray`, fix behavior of containment operators with respect to empty arrays (Tom Lane)

  Empty arrays are now correctly considered to be contained in any other array.
- Remove `contrib/xml2`'s arbitrary limit on the number of *`parameter`*=*`value`* pairs that can be handled by `xslt_process()` (Pavel Stehule)

  The previous limit was 10.
- In `contrib/pageinspect`, fix heap_page_item to return infomasks as 32-bit values (Alvaro Herrera)

  This avoids returning negative values, which was confusing. The underlying value is a 16-bit unsigned integer.

E.50.3.13.1. Security

- Add `contrib/sepgsql` to interface permission checks with SELinux (KaiGai Kohei)

  This uses the new `SECURITY LABEL` facility.
- Add contrib module `auth_delay` (KaiGai Kohei)

  This causes the server to pause before returning authentication failure; it is designed to make brute force password attacks more difficult.

- Add `dummy_seclabel` contrib module (KaiGai Kohei)

  This is used for permission regression testing.

E.50.3.13.2. Performance

- Add support for `LIKE` and `ILIKE` index searches to `contrib/pg_trgm` (Alexander Korotkov)
- Add `levenshtein_less_equal()` function to `contrib/fuzzystrmatch`, which is optimized for small distances (Alexander Korotkov)
- Improve performance of index lookups on `contrib/seg` columns (Alexander Korotkov)
- Improve performance of pg_upgrade for databases with many relations (Bruce Momjian)
- Add flag to `contrib/pgbench` to report per-statement latencies (Florian Pflug)

E.50.3.13.3. Fsync Testing

- Move `src/tools/test_fsync` to `contrib/pg_test_fsync` (Bruce Momjian, Tom Lane)
- Add `O_DIRECT` support to `contrib/pg_test_fsync` (Bruce Momjian)

  This matches the use of `O_DIRECT` by `wal_sync_method`.
- Add new tests to `contrib/pg_test_fsync` (Bruce Momjian)

E.50.3.14. Documentation

- Extensive ECPG documentation improvements (Satoshi Nagayasu)
- Extensive proofreading and documentation improvements (Thom Brown, Josh Kupershmidt, Susanne Ebrecht)
- Add documentation for `exit_on_error` (Robert Haas)

  This parameter causes sessions to exit on any error.
- Add documentation for `pg_options_to_table()` (Josh Berkus)

  This function shows table storage options in a readable form.
- Document that it is possible to access all composite type fields using `(compositeval).*` syntax (Peter Eisentraut)
- Document that `translate()` removes characters in `from` that don't have a corresponding `to` character (Josh Kupershmidt)
- Merge documentation for `CREATE CONSTRAINT TRIGGER` and `CREATE TRIGGER` (Alvaro Herrera)
- Centralize permission and upgrade documentation (Bruce Momjian)
- Add kernel tuning documentation for Solaris 10 (Josh Berkus)

  Previously only Solaris 9 kernel tuning was documented.

- Handle non-ASCII characters consistently in `HISTORY` file (Peter Eisentraut)

  While the `HISTORY` file is in English, we do have to deal with non-ASCII letters in contributor names. These are now transliterated so that they are reasonably legible without assumptions about character set.

E.51. Release 9.0.23

Release date: 2015-10-08

This release contains a variety of fixes from 9.0.22. For information about new features in the 9.0 major release, see 第 E.74 节.

This is expected to be the last PostgreSQL release in the 9.0.X series. Users are encouraged to update to a newer release branch soon.

E.51.1. Migration to Version 9.0.23

A dump/restore is not required for those running 9.0.X.

However, if you are upgrading from a version earlier than 9.0.18, see 第 E.56 节.

E.51.2. Changes

- Fix `contrib/pgcrypto` to detect and report too-short `crypt()` salts (Josh Kupershmidt)

  Certain invalid salt arguments crashed the server or disclosed a few bytes of server memory. We have not ruled out the viability of attacks that arrange for presence of confidential information in the disclosed bytes, but they seem unlikely. (CVE-2015-5288)

- Fix subtransaction cleanup after a portal (cursor) belonging to an outer subtransaction fails (Tom Lane, Michael Paquier)

  A function executed in an outer-subtransaction cursor could cause an assertion failure or crash by referencing a relation created within an inner subtransaction.

- Fix insertion of relations into the relation cache "init file" (Tom Lane)

  An oversight in a patch in the most recent minor releases caused `pg_trigger_tgrelid_tgname_index` to be omitted from the init file. Subsequent sessions detected this, then deemed the init file to be broken and silently ignored it, resulting in a significant degradation in session startup time. In addition to fixing the bug, install some guards so that any similar future mistake will be more obvious.

- Avoid O(N^2) behavior when inserting many tuples into a SPI query result (Neil Conway)

- Improve `LISTEN` startup time when there are many unread notifications (Matt Newell)

- Disable SSL renegotiation by default (Michael Paquier, Andres Freund)

  While use of SSL renegotiation is a good idea in theory, we have seen too many bugs in practice, both in the underlying OpenSSL library and in our usage of it. Renegotiation will be removed entirely in 9.5 and later. In the older branches, just change the default value of `ssl_renegotiation_limit` to zero (disabled).

- Lower the minimum values of the `*_freeze_max_age` parameters (Andres Freund)

  This is mainly to make tests of related behavior less time-consuming, but it may also be of value for installations with limited disk space.

- Limit the maximum value of `wal_buffers` to 2GB to avoid server crashes (Josh Berkus)

- Fix rare internal overflow in multiplication of `numeric` values (Dean Rasheed)

- Guard against hard-to-reach stack overflows involving record types, range types, `json`, `jsonb`, `tsquery`, `ltxtquery` and `query_int` (Noah Misch)

- Fix handling of `DOW` and `DOY` in datetime input (Greg Stark)

  These tokens aren't meant to be used in datetime values, but previously they resulted in opaque internal error messages rather than "invalid input syntax".

- Add more query-cancel checks to regular expression matching (Tom Lane)

- Add recursion depth protections to regular expression, `SIMILAR TO`, and `LIKE` matching (Tom Lane)

  Suitable search patterns and a low stack depth limit could lead to stack-overrun crashes.

- Fix potential infinite loop in regular expression execution (Tom Lane)

  A search pattern that can apparently match a zero-length string, but actually doesn't match because of a back reference, could lead to an infinite loop.

- Fix low-memory failures in regular expression compilation (Andreas Seltenreich)

- Fix low-probability memory leak during regular expression execution (Tom Lane)

- Fix rare low-memory failure in lock cleanup during transaction abort (Tom Lane)

- Fix "unexpected out-of-memory situation during sort" errors when using tuplestores with small `work_mem` settings (Tom Lane)

- Fix very-low-probability stack overrun in `qsort` (Tom Lane)

- Fix "invalid memory alloc request size" failure in hash joins with large `work_mem` settings (Tomas Vondra, Tom Lane)

- Fix assorted planner bugs (Tom Lane)

  These mistakes could lead to incorrect query plans that would give wrong answers, or to assertion failures in assert-enabled builds, or to odd planner errors such as "could not devise a query plan for the given query", "could not find pathkey item to sort", "plan should not reference subplan's variable", or "failed to assign all NestLoopParams to plan nodes". Thanks are due to Andreas Seltenreich and Piotr Stefaniak for fuzz testing that exposed these problems.

- Use fuzzy path cost tiebreaking rule in all supported branches (Tom Lane)

  This change is meant to avoid platform-specific behavior when alternative plan choices have effectively-identical estimated costs.

- During postmaster shutdown, ensure that per-socket lock files are removed and listen sockets are closed before we remove the `postmaster.pid` file (Tom Lane)

This avoids race-condition failures if an external script attempts to start a new postmaster as soon as `pg_ctl stop` returns.

- Fix postmaster's handling of a startup-process crash during crash recovery (Tom Lane)

  If, during a crash recovery cycle, the startup process crashes without having restored database consistency, we'd try to launch a new startup process, which typically would just crash again, leading to an infinite loop.

- Do not print a `WARNING` when an autovacuum worker is already gone when we attempt to signal it, and reduce log verbosity for such signals (Tom Lane)

- Prevent autovacuum launcher from sleeping unduly long if the server clock is moved backwards a large amount (Álvaro Herrera)

- Ensure that cleanup of a GIN index's pending-insertions list is interruptable by cancel requests (Jeff Janes)

- Allow all-zeroes pages in GIN indexes to be reused (Heikki Linnakangas)

  Such a page might be left behind after a crash.

- Fix off-by-one error that led to otherwise-harmless warnings about "apparent wraparound" in subtrans/multixact truncation (Thomas Munro)

- Fix misreporting of `CONTINUE` and `MOVE` statement types in PL/pgSQL's error context messages (Pavel Stehule, Tom Lane)

- Fix some places in PL/Tcl that neglected to check for failure of `malloc()` calls (Michael Paquier, Álvaro Herrera)

- Improve libpq's handling of out-of-memory conditions (Michael Paquier, Heikki Linnakangas)

- Fix memory leaks and missing out-of-memory checks in ecpg (Michael Paquier)

- Fix psql's code for locale-aware formatting of numeric output (Tom Lane)

  The formatting code invoked by `\pset numericlocale on` did the wrong thing for some uncommon cases such as numbers with an exponent but no decimal point. It could also mangle already-localized output from the `money` data type.

- Prevent crash in psql's `\c` command when there is no current connection (Noah Misch)

- Ensure that temporary files created during a pg_dump run with tar-format output are not world-readable (Michael Paquier)

- Fix pg_dump and pg_upgrade to support cases where the `postgres` or `template1` database is in a non-default tablespace (Marti Raudsepp, Bruce Momjian)

- Fix pg_dump to handle object privileges sanely when dumping from a server too old to have a particular privilege type (Tom Lane)

  When dumping functions or procedural languages from pre-7.3 servers, pg_dump would produce `GRANT/REVOKE` commands that revoked the owner's grantable privileges and instead granted all privileges to `PUBLIC`. Since the privileges involved are just `USAGE` and `EXECUTE`, this isn't a security problem, but it's certainly a

surprising representation of the older systems' behavior. Fix it to leave the default privilege state alone in these cases.

- Fix pg_dump to dump shell types (Tom Lane)

  Shell types (that is, not-yet-fully-defined types) aren't useful for much, but nonetheless pg_dump should dump them.

- Fix spinlock assembly code for PPC hardware to be compatible with AIX's native assembler (Tom Lane)

  Building with gcc didn't work if gcc had been configured to use the native assembler, which is becoming more common.

- On AIX, test the `-qlonglong` compiler option rather than just assuming it's safe to use (Noah Misch)

- On AIX, use `-Wl,-brtllib` link option to allow symbols to be resolved at runtime (Noah Misch)

  Perl relies on this ability in 5.8.0 and later.

- Avoid use of inline functions when compiling with 32-bit xlc, due to compiler bugs (Noah Misch)

- Use `librt` for `sched_yield()` when necessary, which it is on some Solaris versions (Oskari Saarenmaa)

- Fix Windows `install.bat` script to handle target directory names that contain spaces (Heikki Linnakangas)

- Make the numeric form of the PostgreSQL version number (e.g., `90405`) readily available to extension Makefiles, as a variable named `VERSION_NUM` (Michael Paquier)

- Update time zone data files to tzdata release 2015g for DST law changes in Cayman Islands, Fiji, Moldova, Morocco, Norfolk Island, North Korea, Turkey, and Uruguay. There is a new zone name `America/Fort_Nelson` for the Canadian Northern Rockies.

E.52. Release 9.0.22

Release date: 2015-06-12

This release contains a small number of fixes from 9.0.21. For information about new features in the 9.0 major release, see 第 E.74 节.

The PostgreSQL community will stop releasing updates for the 9.0.X release series in September 2015. Users are encouraged to update to a newer release branch soon.

E.52.1. Migration to Version 9.0.22

A dump/restore is not required for those running 9.0.X.

However, if you are upgrading from a version earlier than 9.0.18, see 第 E.56 节.

E.52.2. Changes

- Fix rare failure to invalidate relation cache init file (Tom Lane)

With just the wrong timing of concurrent activity, a `VACUUM FULL` on a system catalog might fail to update the "init file" that's used to avoid cache-loading work for new sessions. This would result in later sessions being unable to access that catalog at all. This is a very ancient bug, but it's so hard to trigger that no reproducible case had been seen until recently.

- Avoid deadlock between incoming sessions and `CREATE/DROP DATABASE` (Tom Lane)

  A new session starting in a database that is the target of a `DROP DATABASE` command, or is the template for a `CREATE DATABASE` command, could cause the command to wait for five seconds and then fail, even if the new session would have exited before that.

E.53. Release 9.0.21

Release date: 2015-06-04

This release contains a small number of fixes from 9.0.20. For information about new features in the 9.0 major release, see 第 E.74 节.

The PostgreSQL community will stop releasing updates for the 9.0.X release series in September 2015. Users are encouraged to update to a newer release branch soon.

E.53.1. Migration to Version 9.0.21

A dump/restore is not required for those running 9.0.X.

However, if you are upgrading from a version earlier than 9.0.18, see 第 E.56 节.

E.53.2. Changes

- Avoid failures while `fsync`'ing data directory during crash restart (Abhijit Menon-Sen, Tom Lane)

  In the previous minor releases we added a patch to `fsync` everything in the data directory after a crash. Unfortunately its response to any error condition was to fail, thereby preventing the server from starting up, even when the problem was quite harmless. An example is that an unwritable file in the data directory would prevent restart on some platforms; but it is common to make SSL certificate files unwritable by the server. Revise this behavior so that permissions failures are ignored altogether, and other types of failures are logged but do not prevent continuing.

- Remove configure's check prohibiting linking to a threaded libpython on OpenBSD (Tom Lane)

  The failure this restriction was meant to prevent seems to not be a problem anymore on current OpenBSD versions.

- Allow libpq to use TLS protocol versions beyond v1 (Noah Misch)

  For a long time, libpq was coded so that the only SSL protocol it would allow was TLS v1. Now that newer TLS versions are becoming popular, allow it to negotiate the highest commonly-supported TLS version with the server. (PostgreSQL servers were already capable of such negotiation, so no change is needed on the server side.) This is a back-patch of a change already released in 9.4.0.

E.54. Release 9.0.20

Release date: 2015-05-22

This release contains a variety of fixes from 9.0.19. For information about new features in the 9.0 major release, see 第 E.74 节.

The PostgreSQL community will stop releasing updates for the 9.0.X release series in September 2015. Users are encouraged to update to a newer release branch soon.

E.54.1. Migration to Version 9.0.20

A dump/restore is not required for those running 9.0.X.

However, if you are upgrading from a version earlier than 9.0.18, see 第 E.56 节.

E.54.2. Changes

- Avoid possible crash when client disconnects just before the authentication timeout expires (Benkocs Norbert Attila)

  If the timeout interrupt fired partway through the session shutdown sequence, SSL-related state would be freed twice, typically causing a crash and hence denial of service to other sessions. Experimentation shows that an unauthenticated remote attacker could trigger the bug somewhat consistently, hence treat as security issue. (CVE-2015-3165)

- Improve detection of system-call failures (Noah Misch)

  Our replacement implementation of `snprintf()` failed to check for errors reported by the underlying system library calls; the main case that might be missed is out-of-memory situations. In the worst case this might lead to information exposure, due to our code assuming that a buffer had been overwritten when it hadn't been. Also, there were a few places in which security-relevant calls of other system library functions did not check for failure.

  It remains possible that some calls of the `*printf()` family of functions are vulnerable to information disclosure if an out-of-memory error occurs at just the wrong time. We judge the risk to not be large, but will continue analysis in this area. (CVE-2015-3166)

- In `contrib/pgcrypto`, uniformly report decryption failures as "Wrong key or corrupt data" (Noah Misch)

  Previously, some cases of decryption with an incorrect key could report other error message texts. It has been shown that such variance in error reports can aid attackers in recovering keys from other systems. While it's unknown whether `pgcrypto`'s specific behaviors are likewise exploitable, it seems better to avoid the risk by using a one-size-fits-all message. (CVE-2015-3167)

- Fix incorrect checking of deferred exclusion constraints after a HOT update (Tom Lane)

  If a new row that potentially violates a deferred exclusion constraint is HOT-updated (that is, no indexed columns change and the row can be stored back onto the same table page) later in the same transaction, the exclusion constraint would be reported as violated when the check finally occurred, even if the row(s) the new row originally conflicted with had been deleted.

- Prevent improper reordering of antijoins (NOT EXISTS joins) versus other outer joins (Tom Lane)

  This oversight in the planner has been observed to cause "could not find RelOptInfo for given relids" errors, but it seems possible that sometimes an incorrect query plan might get past that consistency check and result in silently-wrong query output.

- Fix incorrect matching of subexpressions in outer-join plan nodes (Tom Lane)

  Previously, if textually identical non-strict subexpressions were used both above and below an outer join, the planner might try to re-use the value computed below the join, which would be incorrect because the executor would force the value to NULL in case of an unmatched outer row.

- Fix GEQO planner to cope with failure of its join order heuristic (Tom Lane)

  This oversight has been seen to lead to "failed to join all relations together" errors in queries involving `LATERAL`, and that might happen in other cases as well.

- Fix possible deadlock at startup when `max_prepared_transactions` is too small (Heikki Linnakangas)

- Don't archive useless preallocated WAL files after a timeline switch (Heikki Linnakangas)

- Avoid "cannot GetMultiXactIdMembers() during recovery" error (Álvaro Herrera)

- Recursively `fsync()` the data directory after a crash (Abhijit Menon-Sen, Robert Haas)

  This ensures consistency if another crash occurs shortly later. (The second crash would have to be a system-level crash, not just a database crash, for there to be a problem.)

- Fix autovacuum launcher's possible failure to shut down, if an error occurs after it receives SIGTERM (Álvaro Herrera)

- Cope with unexpected signals in `LockBufferForCleanup()` (Andres Freund)

  This oversight could result in spurious errors about "multiple backends attempting to wait for pincount 1".

- Avoid waiting for WAL flush or synchronous replication during commit of a transaction that was read-only so far as the user is concerned (Andres Freund)

  Previously, a delay could occur at commit in transactions that had written WAL due to HOT page pruning, leading to undesirable effects such as sessions getting stuck at startup if all synchronous replicas are down. Sessions have also been observed to get stuck in catchup interrupt processing when using synchronous replication; this will fix that problem as well.

- Fix crash when manipulating hash indexes on temporary tables (Heikki Linnakangas)

- Fix possible failure during hash index bucket split, if other processes are modifying the index concurrently (Tom Lane)

- Check for interrupts while analyzing index expressions (Jeff Janes)

  `ANALYZE` executes index expressions many times; if there are slow functions in such an expression, it's desirable to be able to cancel the `ANALYZE` before that loop finishes.

- Add the name of the target server to object description strings for foreign-server user mappings (Álvaro Herrera)
- Recommend setting `include_realm` to 1 when using Kerberos/GSSAPI/SSPI authentication (Stephen Frost)

  Without this, identically-named users from different realms cannot be distinguished. For the moment this is only a documentation change, but it will become the default setting in PostgreSQL 9.5.
- Remove code for matching IPv4 `pg_hba.conf` entries to IPv4-in-IPv6 addresses (Tom Lane)

  This hack was added in 2003 in response to a report that some Linux kernels of the time would report IPv4 connections as having IPv4-in-IPv6 addresses. However, the logic was accidentally broken in 9.0. The lack of any field complaints since then shows that it's not needed anymore. Now we have reports that the broken code causes crashes on some systems, so let's just remove it rather than fix it. (Had we chosen to fix it, that would make for a subtle and potentially security-sensitive change in the effective meaning of IPv4 `pg_hba.conf` entries, which does not seem like a good thing to do in minor releases.)
- While shutting down service on Windows, periodically send status updates to the Service Control Manager to prevent it from killing the service too soon; and ensure that pg_ctl will wait for shutdown (Krystian Bigaj)
- Reduce risk of network deadlock when using libpq's non-blocking mode (Heikki Linnakangas)

  When sending large volumes of data, it's important to drain the input buffer every so often, in case the server has sent enough response data to cause it to block on output. (A typical scenario is that the server is sending a stream of NOTICE messages during `COPY FROM STDIN`.) This worked properly in the normal blocking mode, but not so much in non-blocking mode. We've modified libpq to opportunistically drain input when it can, but a full defense against this problem requires application cooperation: the application should watch for socket read-ready as well as write-ready conditions, and be sure to call `PQconsumeInput()` upon read-ready.
- Fix array handling in ecpg (Michael Meskes)
- Fix psql to sanely handle URIs and conninfo strings as the first parameter to `\connect` (David Fetter, Andrew Dunstan, Álvaro Herrera)

  This syntax has been accepted (but undocumented) for a long time, but previously some parameters might be taken from the old connection instead of the given string, which was agreed to be undesirable.
- Suppress incorrect complaints from psql on some platforms that it failed to write `~/.psql_history` at exit (Tom Lane)

  This misbehavior was caused by a workaround for a bug in very old (pre-2006) versions of libedit. We fixed it by removing the workaround, which will cause a similar failure to appear for anyone still using such versions of libedit. Recommendation: upgrade that library, or use libreadline.
- Fix pg_dump's rule for deciding which casts are system-provided casts that should not be dumped (Tom Lane)
- Fix dumping of views that are just `VALUES(...)` but have column aliases (Tom Lane)
- In pg_upgrade, force timeline 1 in the new cluster (Bruce Momjian)

This change prevents upgrade failures caused by bogus complaints about missing WAL history files.

- In pg_upgrade, check for improperly non-connectable databases before proceeding (Bruce Momjian)
- In pg_upgrade, quote directory paths properly in the generated `delete_old_cluster` script (Bruce Momjian)
- In pg_upgrade, preserve database-level freezing info properly (Bruce Momjian)

  This oversight could cause missing-clog-file errors for tables within the `postgres` and `template1` databases.
- Run pg_upgrade and pg_resetxlog with restricted privileges on Windows, so that they don't fail when run by an administrator (Muhammad Asif Naeem)
- Fix slow sorting algorithm in `contrib/intarray` (Tom Lane)
- Fix compile failure on Sparc V8 machines (Rob Rowan)
- Update time zone data files to tzdata release 2015d for DST law changes in Egypt, Mongolia, and Palestine, plus historical changes in Canada and Chile. Also adopt revised zone abbreviations for the America/Adak zone (HST/HDT not HAST/HADT).

E.55. Release 9.0.19

Release date: 2015-02-05

This release contains a variety of fixes from 9.0.18. For information about new features in the 9.0 major release, see 第 E.74 节.

E.55.1. Migration to Version 9.0.19

A dump/restore is not required for those running 9.0.X.

However, if you are upgrading from a version earlier than 9.0.18, see 第 E.56 节.

E.55.2. Changes

- Fix buffer overruns in `to_char()` (Bruce Momjian)

  When `to_char()` processes a numeric formatting template calling for a large number of digits, PostgreSQL would read past the end of a buffer. When processing a crafted timestamp formatting template, PostgreSQL would write past the end of a buffer. Either case could crash the server. We have not ruled out the possibility of attacks that lead to privilege escalation, though they seem unlikely. (CVE-2015-0241)
- Fix buffer overrun in replacement `*printf()` functions (Tom Lane)

  PostgreSQL includes a replacement implementation of `printf` and related functions. This code will overrun a stack buffer when formatting a floating point number (conversion specifiers `e`, `E`, `f`, `F`, `g` or `G`) with requested precision greater than about 500. This will crash the server, and we have not ruled out the possibility of attacks that lead to privilege escalation. A database user can trigger such a buffer overrun through the `to_char()` SQL function. While that is the only affected core

PostgreSQL functionality, extension modules that use printf-family functions may be at risk as well.

This issue primarily affects PostgreSQL on Windows. PostgreSQL uses the system implementation of these functions where adequate, which it is on other modern platforms. (CVE-2015-0242)

- Fix buffer overruns in `contrib/pgcrypto` (Marko Tiikkaja, Noah Misch)

  Errors in memory size tracking within the `pgcrypto` module permitted stack buffer overruns and improper dependence on the contents of uninitialized memory. The buffer overrun cases can crash the server, and we have not ruled out the possibility of attacks that lead to privilege escalation. (CVE-2015-0243)

- Fix possible loss of frontend/backend protocol synchronization after an error (Heikki Linnakangas)

  If any error occurred while the server was in the middle of reading a protocol message from the client, it could lose synchronization and incorrectly try to interpret part of the message's data as a new protocol message. An attacker able to submit crafted binary data within a command parameter might succeed in injecting his own SQL commands this way. Statement timeout and query cancellation are the most likely sources of errors triggering this scenario. Particularly vulnerable are applications that use a timeout and also submit arbitrary user-crafted data as binary query parameters. Disabling statement timeout will reduce, but not eliminate, the risk of exploit. Our thanks to Emil Lenngren for reporting this issue. (CVE-2015-0244)

- Fix information leak via constraint-violation error messages (Stephen Frost)

  Some server error messages show the values of columns that violate a constraint, such as a unique constraint. If the user does not have `SELECT` privilege on all columns of the table, this could mean exposing values that the user should not be able to see. Adjust the code so that values are displayed only when they came from the SQL command or could be selected by the user. (CVE-2014-8161)

- Lock down regression testing's temporary installations on Windows (Noah Misch)

  Use SSPI authentication to allow connections only from the OS user who launched the test suite. This closes on Windows the same vulnerability previously closed on other platforms, namely that other users might be able to connect to the test postmaster. (CVE-2014-0067)

- Avoid possible data corruption if `ALTER DATABASE SET TABLESPACE` is used to move a database to a new tablespace and then shortly later move it back to its original tablespace (Tom Lane)

- Avoid corrupting tables when `ANALYZE` inside a transaction is rolled back (Andres Freund, Tom Lane, Michael Paquier)

  If the failing transaction had earlier removed the last index, rule, or trigger from the table, the table would be left in a corrupted state with the relevant `pg_class` flags not set though they should be.

- Fix use-of-already-freed-memory problem in EvalPlanQual processing (Tom Lane)

  In `READ COMMITTED` mode, queries that lock or update recently-updated rows could crash as a result of this bug.

- Fix planning of `SELECT FOR UPDATE` when using a partial index on a child table (Kyotaro Horiguchi)

In `READ COMMITTED` mode, `SELECT FOR UPDATE` must also recheck the partial index's `WHERE` condition when rechecking a recently-updated row to see if it still satisfies the query's `WHERE` condition. This requirement was missed if the index belonged to an inheritance child table, so that it was possible to incorrectly return rows that no longer satisfy the query condition.

- Fix corner case wherein `SELECT FOR UPDATE` could return a row twice, and possibly miss returning other rows (Tom Lane)

  In `READ COMMITTED` mode, a `SELECT FOR UPDATE` that is scanning an inheritance tree could incorrectly return a row from a prior child table instead of the one it should return from a later child table.

- Reject duplicate column names in the referenced-columns list of a `FOREIGN KEY` declaration (David Rowley)

  This restriction is per SQL standard. Previously we did not reject the case explicitly, but later on the code would fail with bizarre-looking errors.

- Fix bugs in raising a `numeric` value to a large integral power (Tom Lane)

  The previous code could get a wrong answer, or consume excessive amounts of time and memory before realizing that the answer must overflow.

- In `numeric_recv()`, truncate away any fractional digits that would be hidden according to the value's `dscale` field (Tom Lane)

  A `numeric` value's display scale (`dscale`) should never be less than the number of nonzero fractional digits; but apparently there's at least one broken client application that transmits binary `numeric` values in which that's true. This leads to strange behavior since the extra digits are taken into account by arithmetic operations even though they aren't printed. The least risky fix seems to be to truncate away such “hidden” digits on receipt, so that the value is indeed what it prints as.

- Reject out-of-range numeric timezone specifications (Tom Lane)

  Simple numeric timezone specifications exceeding +/- 168 hours (one week) would be accepted, but could then cause null-pointer dereference crashes in certain operations. There's no use-case for such large UTC offsets, so reject them.

- Fix bugs in `tsquery @> tsquery` operator (Heikki Linnakangas)

  Two different terms would be considered to match if they had the same CRC. Also, if the second operand had more terms than the first, it would be assumed not to be contained in the first; which is wrong since it might contain duplicate terms.

- Improve ispell dictionary's defenses against bad affix files (Tom Lane)

- Allow more than 64K phrases in a thesaurus dictionary (David Boutin)

  The previous coding could crash on an oversize dictionary, so this was deemed a back-patchable bug fix rather than a feature addition.

- Fix namespace handling in `xpath()` (Ali Akbar)

  Previously, the `xml` value resulting from an `xpath()` call would not have namespace declarations if the namespace declarations were attached to an ancestor element in the input `xml` value, rather than to the specific element being returned. Propagate the ancestral declaration so that the result is correct when considered in isolation.

- Fix planner problems with nested append relations, such as inherited tables within `UNION ALL` subqueries (Tom Lane)
- Fail cleanly when a GiST index tuple doesn't fit on a page, rather than going into infinite recursion (Andrew Gierth)
- Exempt tables that have per-table `cost_limit` and/or `cost_delay` settings from autovacuum's global cost balancing rules (Álvaro Herrera)

  The previous behavior resulted in basically ignoring these per-table settings, which was unintended. Now, a table having such settings will be vacuumed using those settings, independently of what is going on in other autovacuum workers. This may result in heavier total I/O load than before, so such settings should be re-examined for sanity.
- Avoid wholesale autovacuuming when autovacuum is nominally off (Tom Lane)

  Even when autovacuum is nominally off, we will still launch autovacuum worker processes to vacuum tables that are at risk of XID wraparound. However, such a worker process then proceeded to vacuum all tables in the target database, if they met the usual thresholds for autovacuuming. This is at best pretty unexpected; at worst it delays response to the wraparound threat. Fix it so that if autovacuum is turned off, workers only do anti-wraparound vacuums and not any other work.
- Fix race condition between hot standby queries and replaying a full-page image (Heikki Linnakangas)

  This mistake could result in transient errors in queries being executed in hot standby.
- Fix several cases where recovery logic improperly ignored WAL records for `COMMIT/ABORT PREPARED` (Heikki Linnakangas)

  The most notable oversight was that `recovery_target_xid` could not be used to stop at a two-phase commit.
- Avoid creating unnecessary `.ready` marker files for timeline history files (Fujii Masao)
- Fix possible null pointer dereference when an empty prepared statement is used and the `log_statement` setting is `mod` or `ddl` (Fujii Masao)
- Change “pgstat wait timeout” warning message to be LOG level, and rephrase it to be more understandable (Tom Lane)

  This message was originally thought to be essentially a can't-happen case, but it occurs often enough on our slower buildfarm members to be a nuisance. Reduce it to LOG level, and expend a bit more effort on the wording: it now reads “using stale statistics instead of current ones because stats collector is not responding”.
- Fix SPARC spinlock implementation to ensure correctness if the CPU is being run in a non-TSO coherency mode, as some non-Solaris kernels do (Andres Freund)
- Warn if OS X's `setlocale()` starts an unwanted extra thread inside the postmaster (Noah Misch)
- Fix processing of repeated `dbname` parameters in `PQconnectdbParams()` (Alex Shulgin)

  Unexpected behavior ensued if the first occurrence of `dbname` contained a connection string or URI to be expanded.

- Ensure that libpq reports a suitable error message on unexpected socket EOF (Marko Tiikkaja, Tom Lane)

  Depending on kernel behavior, libpq might return an empty error string rather than something useful when the server unexpectedly closed the socket.

- Clear any old error message during `PQreset()` (Heikki Linnakangas)

  If `PQreset()` is called repeatedly, and the connection cannot be re-established, error messages from the failed connection attempts kept accumulating in the `PGconn`'s error string.

- Properly handle out-of-memory conditions while parsing connection options in libpq (Alex Shulgin, Heikki Linnakangas)

- Fix array overrun in ecpg's version of `ParseDateTime()` (Michael Paquier)

- In initdb, give a clearer error message if a password file is specified but is empty (Mats Erik Andersson)

- Fix psql's `\s` command to work nicely with libedit, and add pager support (Stepan Rutz, Tom Lane)

  When using libedit rather than readline, `\s` printed the command history in a fairly unreadable encoded format, and on recent libedit versions might fail altogether. Fix that by printing the history ourselves rather than having the library do it. A pleasant side-effect is that the pager is used if appropriate.

  This patch also fixes a bug that caused newline encoding to be applied inconsistently when saving the command history with libedit. Multiline history entries written by older psql versions will be read cleanly with this patch, but perhaps not vice versa, depending on the exact libedit versions involved.

- Improve consistency of parsing of psql's special variables (Tom Lane)

  Allow variant spellings of `on` and `off` (such as `1`/`0`) for `ECHO_HIDDEN` and `ON_ERROR_ROLLBACK`. Report a warning for unrecognized values for `COMP_KEYWORD_CASE`, `ECHO`, `ECHO_HIDDEN`, `HISTCONTROL`, `ON_ERROR_ROLLBACK`, and `VERBOSITY`. Recognize all values for all these variables case-insensitively; previously there was a mishmash of case-sensitive and case-insensitive behaviors.

- Fix psql's expanded-mode display to work consistently when using `border` = 3 and `linestyle` = `ascii` or `unicode` (Stephen Frost)

- Fix possible deadlock during parallel restore of a schema-only dump (Robert Haas, Tom Lane)

- Fix core dump in `pg_dump --binary-upgrade` on zero-column composite type (Rushabh Lathia)

- Fix block number checking in `contrib/pageinspect`'s `get_raw_page()` (Tom Lane)

  The incorrect checking logic could prevent access to some pages in non-main relation forks.

- Fix `contrib/pgcrypto`'s `pgp_sym_decrypt()` to not fail on messages whose length is 6 less than a power of 2 (Marko Tiikkaja)

- Handle unexpected query results, especially NULLs, safely in `contrib/tablefunc`'s `connectby()` (Michael Paquier)

`connectby()` previously crashed if it encountered a NULL key value. It now prints that row but doesn't recurse further.

- Avoid a possible crash in `contrib/xml2`'s `xslt_process()` (Mark Simonetti)

  libxslt seems to have an undocumented dependency on the order in which resources are freed; reorder our calls to avoid a crash.

- Numerous cleanups of warnings from Coverity static code analyzer (Andres Freund, Tatsuo Ishii, Marko Kreen, Tom Lane, Michael Paquier)

  These changes are mostly cosmetic but in some cases fix corner-case bugs, for example a crash rather than a proper error report after an out-of-memory failure. None are believed to represent security issues.

- Detect incompatible OpenLDAP versions during build (Noah Misch)

  With OpenLDAP versions 2.4.24 through 2.4.31, inclusive, PostgreSQL backends can crash at exit. Raise a warning during configure based on the compile-time OpenLDAP version number, and test the crashing scenario in the `contrib/dblink` regression test.

- In non-MSVC Windows builds, ensure `libpq.dll` is installed with execute permissions (Noah Misch)

- Make pg_regress remove any temporary installation it created upon successful exit (Tom Lane)

  This results in a very substantial reduction in disk space usage during `make check-world`, since that sequence involves creation of numerous temporary installations.

- Support time zone abbreviations that change UTC offset from time to time (Tom Lane)

  Previously, PostgreSQL assumed that the UTC offset associated with a time zone abbreviation (such as `EST`) never changes in the usage of any particular locale. However this assumption fails in the real world, so introduce the ability for a zone abbreviation to represent a UTC offset that sometimes changes. Update the zone abbreviation definition files to make use of this feature in timezone locales that have changed the UTC offset of their abbreviations since 1970 (according to the IANA timezone database). In such timezones, PostgreSQL will now associate the correct UTC offset with the abbreviation depending on the given date.

- Update time zone abbreviations lists (Tom Lane)

  Add CST (China Standard Time) to our lists. Remove references to ADT as "Arabia Daylight Time", an abbreviation that's been out of use since 2007; therefore, claiming there is a conflict with "Atlantic Daylight Time" doesn't seem especially helpful. Fix entirely incorrect GMT offsets for CKT (Cook Islands), FJT, and FJST (Fiji); we didn't even have them on the proper side of the date line.

- Update time zone data files to tzdata release 2015a.

  The IANA timezone database has adopted abbreviations of the form `AxST`/`AxDT` for all Australian time zones, reflecting what they believe to be current majority practice Down Under. These names do not conflict with usage elsewhere (other than ACST for Acre Summer Time, which has been in disuse since 1994). Accordingly, adopt these names into our "Default" timezone abbreviation set. The "Australia" abbreviation set now contains only CST, EAST, EST, SAST, SAT, and WST, all of which are thought to

be mostly historical usage. Note that SAST has also been changed to be South Africa Standard Time in the "Default" abbreviation set.

Also, add zone abbreviations SRET (Asia/Srednekolymsk) and XJT (Asia/Urumqi), and use WSST/WSDT for western Samoa. Also, there were DST law changes in Chile, Mexico, the Turks & Caicos Islands (America/Grand_Turk), and Fiji. There is a new zone Pacific/Bougainville for portions of Papua New Guinea. Also, numerous corrections for historical (pre-1970) time zone data.

E.56. Release 9.0.18

Release date: 2014-07-24

This release contains a variety of fixes from 9.0.17. For information about new features in the 9.0 major release, see 第 E.74 节.

E.56.1. Migration to Version 9.0.18

A dump/restore is not required for those running 9.0.X.

However, this release corrects an index corruption problem in some GiST indexes. See the first changelog entry below to find out whether your installation has been affected and what steps you should take if so.

Also, if you are upgrading from a version earlier than 9.0.15, see 第 E.59 节.

E.56.2. Changes

- Correctly initialize padding bytes in `contrib/btree_gist` indexes on `bit` columns (Heikki Linnakangas)

  This error could result in incorrect query results due to values that should compare equal not being seen as equal. Users with GiST indexes on `bit` or `bit varying` columns should `REINDEX` those indexes after installing this update.

- Protect against torn pages when deleting GIN list pages (Heikki Linnakangas)

  This fix prevents possible index corruption if a system crash occurs while the page update is being written to disk.

- Don't clear the right-link of a GiST index page while replaying updates from WAL (Heikki Linnakangas)

  This error could lead to transiently wrong answers from GiST index scans performed in Hot Standby.

- Fix possibly-incorrect cache invalidation during nested calls to `ReceiveSharedInvalidMessages` (Andres Freund)

- Don't assume a subquery's output is unique if there's a set-returning function in its targetlist (David Rowley)

  This oversight could lead to misoptimization of constructs like `WHERE x IN (SELECT y, generate_series(1,10) FROM t GROUP BY y)`.

- Fix failure to detoast fields in composite elements of structured types (Tom Lane)

  This corrects cases where TOAST pointers could be copied into other tables without being dereferenced. If the original data is later deleted, it would lead to errors like

"missing chunk number 0 for toast value ..." when the now-dangling pointer is used.

- Fix "record type has not been registered" failures with whole-row references to the output of Append plan nodes (Tom Lane)
- Fix possible crash when invoking a user-defined function while rewinding a cursor (Tom Lane)
- Fix query-lifespan memory leak while evaluating the arguments for a function in `FROM` (Tom Lane)
- Fix session-lifespan memory leaks in regular-expression processing (Tom Lane, Arthur O'Dwyer, Greg Stark)
- Fix data encoding error in `hungarian.stop` (Tom Lane)
- Fix liveness checks for rows that were inserted in the current transaction and then deleted by a now-rolled-back subtransaction (Andres Freund)

  This could cause problems (at least spurious warnings, and at worst an infinite loop) if `CREATE INDEX` or `CLUSTER` were done later in the same transaction.
- Clear `pg_stat_activity.xact_start` during `PREPARE TRANSACTION` (Andres Freund)

  After the `PREPARE`, the originating session is no longer in a transaction, so it should not continue to display a transaction start time.
- Fix `REASSIGN OWNED` to not fail for text search objects (Álvaro Herrera)
- Block signals during postmaster startup (Tom Lane)

  This ensures that the postmaster will properly clean up after itself if, for example, it receives SIGINT while still starting up.
- Secure Unix-domain sockets of temporary postmasters started during `make check` (Noah Misch)

  Any local user able to access the socket file could connect as the server's bootstrap superuser, then proceed to execute arbitrary code as the operating-system user running the test, as we previously noted in CVE-2014-0067. This change defends against that risk by placing the server's socket in a temporary, mode 0700 subdirectory of `/tmp`. The hazard remains however on platforms where Unix sockets are not supported, notably Windows, because then the temporary postmaster must accept local TCP connections.

  A useful side effect of this change is to simplify `make check` testing in builds that override `DEFAULT_PGSOCKET_DIR`. Popular non-default values like `/var/run/postgresql` are often not writable by the build user, requiring workarounds that will no longer be necessary.
- Fix tablespace creation WAL replay to work on Windows (MauMau)
- Fix detection of socket creation failures on Windows (Bruce Momjian)
- On Windows, allow new sessions to absorb values of PGC_BACKEND parameters (such as log_connections) from the configuration file (Amit Kapila)

  Previously, if such a parameter were changed in the file post-startup, the change would have no effect.

- Properly quote executable path names on Windows (Nikhil Deshpande)

  This oversight could cause initdb and pg_upgrade to fail on Windows, if the installation path contained both spaces and `@` signs.

- Fix linking of libpython on OS X (Tom Lane)

  The method we previously used can fail with the Python library supplied by Xcode 5.0 and later.

- Avoid buffer bloat in libpq when the server consistently sends data faster than the client can absorb it (Shin-ichi Morita, Tom Lane)

  libpq could be coerced into enlarging its input buffer until it runs out of memory (which would be reported misleadingly as "lost synchronization with server"). Under ordinary circumstances it's quite far-fetched that data could be continuously transmitted more quickly than the `recv()` loop can absorb it, but this has been observed when the client is artificially slowed by scheduler constraints.

- Ensure that LDAP lookup attempts in libpq time out as intended (Laurenz Albe)

- Fix ecpg to do the right thing when an array of `char *` is the target for a FETCH statement returning more than one row, as well as some other array-handling fixes (Ashutosh Bapat)

- Fix pg_restore's processing of old-style large object comments (Tom Lane)

  A direct-to-database restore from an archive file generated by a pre-9.0 version of pg_dump would usually fail if the archive contained more than a few comments for large objects.

- In `contrib/pgcrypto` functions, ensure sensitive information is cleared from stack variables before returning (Marko Kreen)

- In `contrib/uuid-ossp`, cache the state of the OSSP UUID library across calls (Tom Lane)

  This improves the efficiency of UUID generation and reduces the amount of entropy drawn from `/dev/urandom`, on platforms that have that.

- Update time zone data files to tzdata release 2014e for DST law changes in Crimea, Egypt, and Morocco.

E.57. Release 9.0.17

Release date: 2014-03-20

This release contains a variety of fixes from 9.0.16. For information about new features in the 9.0 major release, see 第 E.74 节.

E.57.1. Migration to Version 9.0.17

A dump/restore is not required for those running 9.0.X.

However, if you are upgrading from a version earlier than 9.0.15, see 第 E.59 节.

E.57.2. Changes

- Restore GIN metapages unconditionally to avoid torn-page risk (Heikki Linnakangas)

Although this oversight could theoretically result in a corrupted index, it is unlikely to have caused any problems in practice, since the active part of a GIN metapage is smaller than a standard 512-byte disk sector.

- Avoid race condition in checking transaction commit status during receipt of a `NOTIFY` message (Marko Tiikkaja)

  This prevents a scenario wherein a sufficiently fast client might respond to a notification before database updates made by the notifier have become visible to the recipient.

- Allow regular-expression operators to be terminated early by query cancel requests (Tom Lane)

  This prevents scenarios wherein a pathological regular expression could lock up a server process uninterruptibly for a long time.

- Remove incorrect code that tried to allow `OVERLAPS` with single-element row arguments (Joshua Yanovski)

  This code never worked correctly, and since the case is neither specified by the SQL standard nor documented, it seemed better to remove it than fix it.

- Avoid getting more than `AccessShareLock` when de-parsing a rule or view (Dean Rasheed)

  This oversight resulted in pg_dump unexpectedly acquiring `RowExclusiveLock` locks on tables mentioned as the targets of `INSERT`/`UPDATE`/`DELETE` commands in rules. While usually harmless, that could interfere with concurrent transactions that tried to acquire, for example, `ShareLock` on those tables.

- Improve performance of index endpoint probes during planning (Tom Lane)

  This change fixes a significant performance problem that occurred when there were many not-yet-committed rows at the end of the index, which is a common situation for indexes on sequentially-assigned values such as timestamps or sequence-generated identifiers.

- Fix test to see if hot standby connections can be allowed immediately after a crash (Heikki Linnakangas)

- Prevent interrupts while reporting non-`ERROR` messages (Tom Lane)

  This guards against rare server-process freezeups due to recursive entry to `syslog()`, and perhaps other related problems.

- Prevent intermittent "could not reserve shared memory region" failures on recent Windows versions (MauMau)

- Update time zone data files to tzdata release 2014a for DST law changes in Fiji and Turkey, plus historical changes in Israel and Ukraine.

E.58. Release 9.0.16

Release date: 2014-02-20

This release contains a variety of fixes from 9.0.15. For information about new features in the 9.0 major release, see 第 E.74 节.

E.58.1. Migration to Version 9.0.16

A dump/restore is not required for those running 9.0.X.

However, if you are upgrading from a version earlier than 9.0.15, see 第 E.59 节.

E.58.2. Changes

- Shore up `GRANT ... WITH ADMIN OPTION` restrictions (Noah Misch)

  Granting a role without `ADMIN OPTION` is supposed to prevent the grantee from adding or removing members from the granted role, but this restriction was easily bypassed by doing `SET ROLE` first. The security impact is mostly that a role member can revoke the access of others, contrary to the wishes of his grantor. Unapproved role member additions are a lesser concern, since an uncooperative role member could provide most of his rights to others anyway by creating views or `SECURITY DEFINER` functions. (CVE-2014-0060)

- Prevent privilege escalation via manual calls to PL validator functions (Andres Freund)

  The primary role of PL validator functions is to be called implicitly during `CREATE FUNCTION`, but they are also normal SQL functions that a user can call explicitly. Calling a validator on a function actually written in some other language was not checked for and could be exploited for privilege-escalation purposes. The fix involves adding a call to a privilege-checking function in each validator function. Non-core procedural languages will also need to make this change to their own validator functions, if any. (CVE-2014-0061)

- Avoid multiple name lookups during table and index DDL (Robert Haas, Andres Freund)

  If the name lookups come to different conclusions due to concurrent activity, we might perform some parts of the DDL on a different table than other parts. At least in the case of `CREATE INDEX`, this can be used to cause the permissions checks to be performed against a different table than the index creation, allowing for a privilege escalation attack. (CVE-2014-0062)

- Prevent buffer overrun with long datetime strings (Noah Misch)

  The `MAXDATELEN` constant was too small for the longest possible value of type `interval`, allowing a buffer overrun in `interval_out()`. Although the datetime input functions were more careful about avoiding buffer overrun, the limit was short enough to cause them to reject some valid inputs, such as input containing a very long timezone name. The ecpg library contained these vulnerabilities along with some of its own. (CVE-2014-0063)

- Prevent buffer overrun due to integer overflow in size calculations (Noah Misch, Heikki Linnakangas)

  Several functions, mostly type input functions, calculated an allocation size without checking for overflow. If overflow did occur, a too-small buffer would be allocated and then written past. (CVE-2014-0064)

- Prevent overruns of fixed-size buffers (Peter Eisentraut, Jozef Mlich)

  Use `strlcpy()` and related functions to provide a clear guarantee that fixed-size buffers are not overrun. Unlike the preceding items, it is unclear whether these cases really represent live issues, since in most cases there appear to be previous

constraints on the size of the input string. Nonetheless it seems prudent to silence all Coverity warnings of this type. (CVE-2014-0065)

- Avoid crashing if `crypt()` returns NULL (Honza Horak, Bruce Momjian)

  There are relatively few scenarios in which `crypt()` could return NULL, but `contrib/chkpass` would crash if it did. One practical case in which this could be an issue is if libc is configured to refuse to execute unapproved hashing algorithms (e.g.,“FIPS mode”). (CVE-2014-0066)

- Document risks of `make check` in the regression testing instructions (Noah Misch, Tom Lane)

  Since the temporary server started by `make check` uses “trust” authentication, another user on the same machine could connect to it as database superuser, and then potentially exploit the privileges of the operating-system user who started the tests. A future release will probably incorporate changes in the testing procedure to prevent this risk, but some public discussion is needed first. So for the moment, just warn people against using `make check` when there are untrusted users on the same machine. (CVE-2014-0067)

- Fix possible mis-replay of WAL records when some segments of a relation aren't full size (Greg Stark, Tom Lane)

  The WAL update could be applied to the wrong page, potentially many pages past where it should have been. Aside from corrupting data, this error has been observed to result in significant “bloat” of standby servers compared to their masters, due to updates being applied far beyond where the end-of-file should have been. This failure mode does not appear to be a significant risk during crash recovery, only when initially synchronizing a standby created from a base backup taken from a quickly-changing master.

- Fix bug in determining when recovery has reached consistency (Tomonari Katsumata, Heikki Linnakangas)

  In some cases WAL replay would mistakenly conclude that the database was already consistent at the start of replay, thus possibly allowing hot-standby queries before the database was really consistent. Other symptoms such as “PANIC: WAL contains references to invalid pages” were also possible.

- Fix improper locking of btree index pages while replaying a `VACUUM` operation in hot-standby mode (Andres Freund, Heikki Linnakangas, Tom Lane)

  This error could result in “PANIC: WAL contains references to invalid pages” failures.

- Ensure that insertions into non-leaf GIN index pages write a full-page WAL record when appropriate (Heikki Linnakangas)

  The previous coding risked index corruption in the event of a partial-page write during a system crash.

- Fix race conditions during server process exit (Robert Haas)

  Ensure that signal handlers don't attempt to use the process's `MyProc` pointer after it's no longer valid.

- Fix unsafe references to `errno` within error reporting logic (Christian Kruse)

  This would typically lead to odd behaviors such as missing or inappropriate `HINT` fields.

- Fix possible crashes from using `ereport()` too early during server startup (Tom Lane)

  The principal case we've seen in the field is a crash if the server is started in a directory it doesn't have permission to read.

- Clear retry flags properly in OpenSSL socket write function (Alexander Kukushkin)

  This omission could result in a server lockup after unexpected loss of an SSL-encrypted connection.

- Fix length checking for Unicode identifiers (`U&"..."` syntax) containing escapes (Tom Lane)

  A spurious truncation warning would be printed for such identifiers if the escaped form of the identifier was too long, but the identifier actually didn't need truncation after de-escaping.

- Allow keywords that are type names to be used in lists of roles (Stephen Frost)

  A previous patch allowed such keywords to be used without quoting in places such as role identifiers; but it missed cases where a list of role identifiers was permitted, such as `DROP ROLE`.

- Fix possible crash due to invalid plan for nested sub-selects, such as `WHERE (... x IN (SELECT ...) ...) IN (SELECT ...)` (Tom Lane)

- Ensure that `ANALYZE` creates statistics for a table column even when all the values in it are "too wide" (Tom Lane)

  `ANALYZE` intentionally omits very wide values from its histogram and most-common-values calculations, but it neglected to do something sane in the case that all the sampled entries are too wide.

- In `ALTER TABLE ... SET TABLESPACE`, allow the database's default tablespace to be used without a permissions check (Stephen Frost)

  `CREATE TABLE` has always allowed such usage, but `ALTER TABLE` didn't get the memo.

- Fix "cannot accept a set" error when some arms of a `CASE` return a set and others don't (Tom Lane)

- Fix checks for all-zero client addresses in pgstat functions (Kevin Grittner)

- Fix possible misclassification of multibyte characters by the text search parser (Tom Lane)

  Non-ASCII characters could be misclassified when using C locale with a multibyte encoding. On Cygwin, non-C locales could fail as well.

- Fix possible misbehavior in `plainto_tsquery()` (Heikki Linnakangas)

  Use `memmove()` not `memcpy()` for copying overlapping memory regions. There have been no field reports of this actually causing trouble, but it's certainly risky.

- Accept `SHIFT_JIS` as an encoding name for locale checking purposes (Tatsuo Ishii)

- Fix misbehavior of `PQhost()` on Windows (Fujii Masao)

  It should return `localhost` if no host has been specified.

- Improve error handling in libpq and psql for failures during `COPY TO STDOUT/FROM STDIN` (Tom Lane)

  In particular this fixes an infinite loop that could occur in 9.2 and up if the server connection was lost during `COPY FROM STDIN`. Variants of that scenario might be possible in older versions, or with other client applications.

- Fix misaligned descriptors in ecpg (MauMau)

- In ecpg, handle lack of a hostname in the connection parameters properly (Michael Meskes)

- Fix performance regression in `contrib/dblink` connection startup (Joe Conway)

  Avoid an unnecessary round trip when client and server encodings match.

- In `contrib/isn`, fix incorrect calculation of the check digit for ISMN values (Fabien Coelho)

- Ensure client-code-only installation procedure works as documented (Peter Eisentraut)

- In Mingw and Cygwin builds, install the libpq DLL in the `bin` directory (Andrew Dunstan)

  This duplicates what the MSVC build has long done. It should fix problems with programs like psql failing to start because they can't find the DLL.

- Avoid using the deprecated `dllwrap` tool in Cygwin builds (Marco Atzeri)

- Don't generate plain-text `HISTORY` and `src/test/regress/README` files anymore (Tom Lane)

  These text files duplicated the main HTML and PDF documentation formats. The trouble involved in maintaining them greatly outweighs the likely audience for plain-text format. Distribution tarballs will still contain files by these names, but they'll just be stubs directing the reader to consult the main documentation. The plain-text `INSTALL` file will still be maintained, as there is arguably a use-case for that.

- Update time zone data files to tzdata release 2013i for DST law changes in Jordan and historical changes in Cuba.

  In addition, the zones `Asia/Riyadh87`, `Asia/Riyadh88`, and `Asia/Riyadh89` have been removed, as they are no longer maintained by IANA, and never represented actual civil timekeeping practice.

E.59. Release 9.0.15

Release date: 2013-12-05

This release contains a variety of fixes from 9.0.14. For information about new features in the 9.0 major release, see 第 E.74 节.

E.59.1. Migration to Version 9.0.15

A dump/restore is not required for those running 9.0.X.

However, this release corrects a number of potential data corruption issues. See the first two changelog entries below to find out whether your installation has been affected and what steps you can take if so.

Also, if you are upgrading from a version earlier than 9.0.13, see 第 E.61 节.

E.59.2. Changes

- Fix `VACUUM`'s tests to see whether it can update `relfrozenxid` (Andres Freund)

  In some cases `VACUUM` (either manual or autovacuum) could incorrectly advance a table's `relfrozenxid` value, allowing tuples to escape freezing, causing those rows to become invisible once 2^31 transactions have elapsed. The probability of data loss is fairly low since multiple incorrect advancements would need to happen before actual loss occurs, but it's not zero. Users upgrading from releases 9.0.4 or 8.4.8 or earlier are not affected, but all later versions contain the bug.

  The issue can be ameliorated by, after upgrading, vacuuming all tables in all databases while having `vacuum_freeze_table_age` set to zero. This will fix any latent corruption but will not be able to fix all pre-existing data errors. However, an installation can be presumed safe after performing this vacuuming if it has executed fewer than 2^31 update transactions in its lifetime (check this with `SELECT txid_current() < 2^31`).

- Fix initialization of `pg_clog` and `pg_subtrans` during hot standby startup (Andres Freund, Heikki Linnakangas)

  This bug can cause data loss on standby servers at the moment they start to accept hot-standby queries, by marking committed transactions as uncommitted. The likelihood of such corruption is small unless, at the time of standby startup, the primary server has executed many updating transactions since its last checkpoint. Symptoms include missing rows, rows that should have been deleted being still visible, and obsolete versions of updated rows being still visible alongside their newer versions.

  This bug was introduced in versions 9.3.0, 9.2.5, 9.1.10, and 9.0.14. Standby servers that have only been running earlier releases are not at risk. It's recommended that standby servers that have ever run any of the buggy releases be re-cloned from the primary (e.g., with a new base backup) after upgrading.

- Truncate `pg_multixact` contents during WAL replay (Andres Freund)

  This avoids ever-increasing disk space consumption in standby servers.

- Fix race condition in GIN index posting tree page deletion (Heikki Linnakangas)

  This could lead to transient wrong answers or query failures.

- Avoid flattening a subquery whose `SELECT` list contains a volatile function wrapped inside a sub-`SELECT` (Tom Lane)

  This avoids unexpected results due to extra evaluations of the volatile function.

- Fix planner's processing of non-simple-variable subquery outputs nested within outer joins (Tom Lane)

  This error could lead to incorrect plans for queries involving multiple levels of subqueries within `JOIN` syntax.

- Fix premature deletion of temporary files (Andres Freund)

- Fix possible read past end of memory in rule printing (Peter Eisentraut)

- Fix array slicing of `int2vector` and `oidvector` values (Tom Lane)

Expressions of this kind are now implicitly promoted to regular `int2` or `oid` arrays.

- Fix incorrect behaviors when using a SQL-standard, simple GMT offset timezone (Tom Lane)

  In some cases, the system would use the simple GMT offset value when it should have used the regular timezone setting that had prevailed before the simple offset was selected. This change also causes the `timeofday` function to honor the simple GMT offset zone.

- Prevent possible misbehavior when logging translations of Windows error codes (Tom Lane)

- Properly quote generated command lines in pg_ctl (Naoya Anzai and Tom Lane)

  This fix applies only to Windows.

- Fix pg_dumpall to work when a source database sets `default_transaction_read_only` via `ALTER DATABASE SET` (Kevin Grittner)

  Previously, the generated script would fail during restore.

- Fix ecpg's processing of lists of variables declared `varchar` (Zoltán Böszörményi)

- Make `contrib/lo` defend against incorrect trigger definitions (Marc Cousin)

- Update time zone data files to tzdata release 2013h for DST law changes in Argentina, Brazil, Jordan, Libya, Liechtenstein, Morocco, and Palestine. Also, new timezone abbreviations WIB, WIT, WITA for Indonesia.

E.60. Release 9.0.14

Release date: 2013-10-10

This release contains a variety of fixes from 9.0.13. For information about new features in the 9.0 major release, see 第 E.74 节.

E.60.1. Migration to Version 9.0.14

A dump/restore is not required for those running 9.0.X.

However, if you are upgrading from a version earlier than 9.0.13, see 第 E.61 节.

E.60.2. Changes

- Prevent corruption of multi-byte characters when attempting to case-fold identifiers (Andrew Dunstan)

  PostgreSQL case-folds non-ASCII characters only when using a single-byte server encoding.

- Fix checkpoint memory leak in background writer when `wal_level = hot_standby` (Naoya Anzai)

- Fix memory leak caused by `lo_open()` failure (Heikki Linnakangas)

- Fix memory overcommit bug when `work_mem` is using more than 24GB of memory (Stephen Frost)

- Fix deadlock bug in libpq when using SSL (Stephen Frost)
- Fix possible SSL state corruption in threaded libpq applications (Nick Phillips, Stephen Frost)
- Properly compute row estimates for boolean columns containing many NULL values (Andrew Gierth)

  Previously tests like `col IS NOT TRUE` and `col IS NOT FALSE` did not properly factor in NULL values when estimating plan costs.
- Prevent pushing down `WHERE` clauses into unsafe `UNION/INTERSECT` subqueries (Tom Lane)

  Subqueries of a `UNION` or `INTERSECT` that contain set-returning functions or volatile functions in their `SELECT` lists could be improperly optimized, leading to run-time errors or incorrect query results.
- Fix rare case of "failed to locate grouping columns" planner failure (Tom Lane)
- Improve view dumping code's handling of dropped columns in referenced tables (Tom Lane)
- Properly record index comments created using `UNIQUE` and `PRIMARY KEY` syntax (Andres Freund)

  This fixes a parallel pg_restore failure.
- Fix `REINDEX TABLE` and `REINDEX DATABASE` to properly revalidate constraints and mark invalidated indexes as valid (Noah Misch)

  `REINDEX INDEX` has always worked properly.
- Fix possible deadlock during concurrent `CREATE INDEX CONCURRENTLY` operations (Tom Lane)
- Fix `regexp_matches()` handling of zero-length matches (Jeevan Chalke)

  Previously, zero-length matches like '^' could return too many matches.
- Fix crash for overly-complex regular expressions (Heikki Linnakangas)
- Fix regular expression match failures for back references combined with non-greedy quantifiers (Jeevan Chalke)
- Prevent `CREATE FUNCTION` from checking `SET` variables unless function body checking is enabled (Tom Lane)
- Allow `ALTER DEFAULT PRIVILEGES` to operate on schemas without requiring CREATE permission (Tom Lane)
- Loosen restriction on keywords used in queries (Tom Lane)

  Specifically, lessen keyword restrictions for role names, language names, `EXPLAIN` and `COPY` options, and `SET` values. This allows `COPY ... (FORMAT BINARY)` to work as expected; previously `BINARY` needed to be quoted.
- Fix `pgp_pub_decrypt()` so it works for secret keys with passwords (Marko Kreen)
- Remove rare inaccurate warning during vacuum of index-less tables (Heikki Linnakangas)

- Ensure that `VACUUM ANALYZE` still runs the ANALYZE phase if its attempt to truncate the file is cancelled due to lock conflicts (Kevin Grittner)
- Avoid possible failure when performing transaction control commands (e.g ROLLBACK) in prepared queries (Tom Lane)
- Ensure that floating-point data input accepts standard spellings of "infinity" on all platforms (Tom Lane)

  The C99 standard says that allowable spellings are `inf`, `+inf`, `-inf`, `infinity`, `+infinity`, and `-infinity`. Make sure we recognize these even if the platform's `strtod` function doesn't.
- Expand ability to compare rows to records and arrays (Rafal Rzepecki, Tom Lane)
- Update time zone data files to tzdata release 2013d for DST law changes in Israel, Morocco, Palestine, and Paraguay. Also, historical zone data corrections for Macquarie Island.

E.61. Release 9.0.13

Release date: 2013-04-04

This release contains a variety of fixes from 9.0.12. For information about new features in the 9.0 major release, see 第 E.74 节.

E.61.1. Migration to Version 9.0.13

A dump/restore is not required for those running 9.0.X.

However, this release corrects several errors in management of GiST indexes. After installing this update, it is advisable to `REINDEX` any GiST indexes that meet one or more of the conditions described below.

Also, if you are upgrading from a version earlier than 9.0.6, see 第 E.68 节.

E.61.2. Changes

- Fix insecure parsing of server command-line switches (Mitsumasa Kondo, Kyotaro Horiguchi)

  A connection request containing a database name that begins with "-" could be crafted to damage or destroy files within the server's data directory, even if the request is eventually rejected. (CVE-2013-1899)
- Reset OpenSSL randomness state in each postmaster child process (Marko Kreen)

  This avoids a scenario wherein random numbers generated by `contrib/pgcrypto` functions might be relatively easy for another database user to guess. The risk is only significant when the postmaster is configured with `ssl` = `on` but most connections don't use SSL encryption. (CVE-2013-1900)
- Fix GiST indexes to not use "fuzzy" geometric comparisons when it's not appropriate to do so (Alexander Korotkov)

  The core geometric types perform comparisons using "fuzzy" equality, but `gist_box_same` must do exact comparisons, else GiST indexes using it might become

inconsistent. After installing this update, users should `REINDEX` any GiST indexes on `box`, `polygon`, `circle`, or `point` columns, since all of these use `gist_box_same`.

- Fix erroneous range-union and penalty logic in GiST indexes that use `contrib/btree_gist` for variable-width data types, that is `text`, `bytea`, `bit`, and `numeric` columns (Tom Lane)

  These errors could result in inconsistent indexes in which some keys that are present would not be found by searches, and also in useless index bloat. Users are advised to `REINDEX` such indexes after installing this update.

- Fix bugs in GiST page splitting code for multi-column indexes (Tom Lane)

  These errors could result in inconsistent indexes in which some keys that are present would not be found by searches, and also in indexes that are unnecessarily inefficient to search. Users are advised to `REINDEX` multi-column GiST indexes after installing this update.

- Fix `gist_point_consistent` to handle fuzziness consistently (Alexander Korotkov)

  Index scans on GiST indexes on `point` columns would sometimes yield results different from a sequential scan, because `gist_point_consistent` disagreed with the underlying operator code about whether to do comparisons exactly or fuzzily.

- Fix buffer leak in WAL replay (Heikki Linnakangas)

  This bug could result in "incorrect local pin count" errors during replay, making recovery impossible.

- Fix race condition in `DELETE RETURNING` (Tom Lane)

  Under the right circumstances, `DELETE RETURNING` could attempt to fetch data from a shared buffer that the current process no longer has any pin on. If some other process changed the buffer meanwhile, this would lead to garbage `RETURNING` output, or even a crash.

- Fix infinite-loop risk in regular expression compilation (Tom Lane, Don Porter)

- Fix potential null-pointer dereference in regular expression compilation (Tom Lane)

- Fix `to_char()` to use ASCII-only case-folding rules where appropriate (Tom Lane)

  This fixes misbehavior of some template patterns that should be locale-independent, but mishandled "`I`" and "`i`" in Turkish locales.

- Fix unwanted rejection of timestamp `1999-12-31 24:00:00` (Tom Lane)

- Fix logic error when a single transaction does `UNLISTEN` then `LISTEN` (Tom Lane)

  The session wound up not listening for notify events at all, though it surely should listen in this case.

- Remove useless "picksplit doesn't support secondary split" log messages (Josh Hansen, Tom Lane)

  This message seems to have been added in expectation of code that was never written, and probably never will be, since GiST's default handling of secondary splits is actually pretty good. So stop nagging end users about it.

- Fix possible failure to send a session's last few transaction commit/abort counts to the statistics collector (Tom Lane)

- Eliminate memory leaks in PL/Perl's `spi_prepare()` function (Alex Hunsaker, Tom Lane)
- Fix pg_dumpall to handle database names containing "=" correctly (Heikki Linnakangas)
- Avoid crash in pg_dump when an incorrect connection string is given (Heikki Linnakangas)
- Ignore invalid indexes in pg_dump and pg_upgrade (Michael Paquier, Bruce Momjian)

  Dumping invalid indexes can cause problems at restore time, for example if the reason the index creation failed was because it tried to enforce a uniqueness condition not satisfied by the table's data. Also, if the index creation is in fact still in progress, it seems reasonable to consider it to be an uncommitted DDL change, which pg_dump wouldn't be expected to dump anyway. pg_upgrade now also skips invalid indexes rather than failing.
- Fix `contrib/pg_trgm`'s `similarity()` function to return zero for trigram-less strings (Tom Lane)

  Previously it returned `NaN` due to internal division by zero.
- Update time zone data files to tzdata release 2013b for DST law changes in Chile, Haiti, Morocco, Paraguay, and some Russian areas. Also, historical zone data corrections for numerous places.

  Also, update the time zone abbreviation files for recent changes in Russia and elsewhere: `CHOT`, `GET`, `IRKT`, `KGT`, `KRAT`, `MAGT`, `MAWT`, `MSK`, `NOVT`, `OMST`, `TKT`, `VLAT`, `WST`, `YAKT`, `YEKT` now follow their current meanings, and `VOLT` (Europe/Volgograd) and `MIST` (Antarctica/Macquarie) are added to the default abbreviations list.

E.62. Release 9.0.12

Release date: 2013-02-07

This release contains a variety of fixes from 9.0.11. For information about new features in the 9.0 major release, see 第 E.74 节.

E.62.1. Migration to Version 9.0.12

A dump/restore is not required for those running 9.0.X.

However, if you are upgrading from a version earlier than 9.0.6, see 第 E.68 节.

E.62.2. Changes

- Prevent execution of `enum_recv` from SQL (Tom Lane)

  The function was misdeclared, allowing a simple SQL command to crash the server. In principle an attacker might be able to use it to examine the contents of server memory. Our thanks to Sumit Soni (via Secunia SVCRP) for reporting this issue. (CVE-2013-0255)
- Fix multiple problems in detection of when a consistent database state has been reached during WAL replay (Fujii Masao, Heikki Linnakangas, Simon Riggs, Andres Freund)

- Update minimum recovery point when truncating a relation file (Heikki Linnakangas)

  Once data has been discarded, it's no longer safe to stop recovery at an earlier point in the timeline.

- Fix missing cancellations in hot standby mode (Noah Misch, Simon Riggs)

  The need to cancel conflicting hot-standby queries would sometimes be missed, allowing those queries to see inconsistent data.

- Fix SQL grammar to allow subscripting or field selection from a sub-SELECT result (Tom Lane)

- Fix performance problems with autovacuum truncation in busy workloads (Jan Wieck)

  Truncation of empty pages at the end of a table requires exclusive lock, but autovacuum was coded to fail (and release the table lock) when there are conflicting lock requests. Under load, it is easily possible that truncation would never occur, resulting in table bloat. Fix by performing a partial truncation, releasing the lock, then attempting to re-acquire the lock and continue. This fix also greatly reduces the average time before autovacuum releases the lock after a conflicting request arrives.

- Protect against race conditions when scanning `pg_tablespace` (Stephen Frost, Tom Lane)

  `CREATE DATABASE` and `DROP DATABASE` could misbehave if there were concurrent updates of `pg_tablespace` entries.

- Prevent `DROP OWNED` from trying to drop whole databases or tablespaces (Álvaro Herrera)

  For safety, ownership of these objects must be reassigned, not dropped.

- Fix error in `vacuum_freeze_table_age` implementation (Andres Freund)

  In installations that have existed for more than `vacuum_freeze_min_age` transactions, this mistake prevented autovacuum from using partial-table scans, so that a full-table scan would always happen instead.

- Prevent misbehavior when a `RowExpr` or `XmlExpr` is parse-analyzed twice (Andres Freund, Tom Lane)

  This mistake could be user-visible in contexts such as `CREATE TABLE LIKE INCLUDING INDEXES`.

- Improve defenses against integer overflow in hashtable sizing calculations (Jeff Davis)

- Reject out-of-range dates in `to_date()` (Hitoshi Harada)

- Ensure that non-ASCII prompt strings are translated to the correct code page on Windows (Alexander Law, Noah Misch)

  This bug affected psql and some other client programs.

- Fix possible crash in psql's `\?` command when not connected to a database (Meng Qingzhong)

- Fix pg_upgrade to deal with invalid indexes safely (Bruce Momjian)

- Fix one-byte buffer overrun in libpq's `PQprintTuples` (Xi Wang)

This ancient function is not used anywhere by PostgreSQL itself, but it might still be used by some client code.

- Make ecpglib use translated messages properly (Chen Huajun)
- Properly install ecpg_compat and pgtypes libraries on MSVC (Jiang Guiqing)
- Include our version of `isinf()` in libecpg if it's not provided by the system (Jiang Guiqing)
- Rearrange configure's tests for supplied functions so it is not fooled by bogus exports from libedit/libreadline (Christoph Berg)
- Ensure Windows build number increases over time (Magnus Hagander)
- Make pgxs build executables with the right `.exe` suffix when cross-compiling for Windows (Zoltan Boszormenyi)
- Add new timezone abbreviation `FET` (Tom Lane)

  This is now used in some eastern-European time zones.

E.63. Release 9.0.11

Release date: 2012-12-06

This release contains a variety of fixes from 9.0.10. For information about new features in the 9.0 major release, see 第 E.74 节.

E.63.1. Migration to Version 9.0.11

A dump/restore is not required for those running 9.0.X.

However, if you are upgrading from a version earlier than 9.0.6, see 第 E.68 节.

E.63.2. Changes

- Fix multiple bugs associated with `CREATE INDEX CONCURRENTLY` (Andres Freund, Tom Lane)

  Fix `CREATE INDEX CONCURRENTLY` to use in-place updates when changing the state of an index's `pg_index` row. This prevents race conditions that could cause concurrent sessions to miss updating the target index, thus resulting in corrupt concurrently-created indexes.

  Also, fix various other operations to ensure that they ignore invalid indexes resulting from a failed `CREATE INDEX CONCURRENTLY` command. The most important of these is `VACUUM`, because an auto-vacuum could easily be launched on the table before corrective action can be taken to fix or remove the invalid index.

- Fix buffer locking during WAL replay (Tom Lane)

  The WAL replay code was insufficiently careful about locking buffers when replaying WAL records that affect more than one page. This could result in hot standby queries transiently seeing inconsistent states, resulting in wrong answers or unexpected failures.

- Fix an error in WAL generation logic for GIN indexes (Tom Lane)

This could result in index corruption, if a torn-page failure occurred.

- Properly remove startup process's virtual XID lock when promoting a hot standby server to normal running (Simon Riggs)

  This oversight could prevent subsequent execution of certain operations such as `CREATE INDEX CONCURRENTLY`.

- Avoid bogus "out-of-sequence timeline ID" errors in standby mode (Heikki Linnakangas)

- Prevent the postmaster from launching new child processes after it's received a shutdown signal (Tom Lane)

  This mistake could result in shutdown taking longer than it should, or even never completing at all without additional user action.

- Avoid corruption of internal hash tables when out of memory (Hitoshi Harada)

- Fix planning of non-strict equivalence clauses above outer joins (Tom Lane)

  The planner could derive incorrect constraints from a clause equating a non-strict construct to something else, for example `WHERE COALESCE(foo, 0) = 0` when `foo` is coming from the nullable side of an outer join.

- Improve planner's ability to prove exclusion constraints from equivalence classes (Tom Lane)

- Fix partial-row matching in hashed subplans to handle cross-type cases correctly (Tom Lane)

  This affects multicolumn `NOT IN` subplans, such as `WHERE (a, b) NOT IN (SELECT x, y FROM ...)` when for instance `b` and `y` are `int4` and `int8` respectively. This mistake led to wrong answers or crashes depending on the specific datatypes involved.

- Acquire buffer lock when re-fetching the old tuple for an `AFTER ROW UPDATE/DELETE` trigger (Andres Freund)

  In very unusual circumstances, this oversight could result in passing incorrect data to the precheck logic for a foreign-key enforcement trigger. That could result in a crash, or in an incorrect decision about whether to fire the trigger.

- Fix `ALTER COLUMN TYPE` to handle inherited check constraints properly (Pavan Deolasee)

  This worked correctly in pre-8.4 releases, and now works correctly in 8.4 and later.

- Fix `REASSIGN OWNED` to handle grants on tablespaces (Álvaro Herrera)

- Ignore incorrect `pg_attribute` entries for system columns for views (Tom Lane)

  Views do not have any system columns. However, we forgot to remove such entries when converting a table to a view. That's fixed properly for 9.3 and later, but in previous branches we need to defend against existing mis-converted views.

- Fix rule printing to dump `INSERT INTO` *`table`* `DEFAULT VALUES` correctly (Tom Lane)

- Guard against stack overflow when there are too many `UNION`/`INTERSECT`/`EXCEPT` clauses in a query (Tom Lane)

- Prevent platform-dependent failures when dividing the minimum possible integer value by -1 (Xi Wang, Tom Lane)
- Fix possible access past end of string in date parsing (Hitoshi Harada)
- Fix failure to advance XID epoch if XID wraparound happens during a checkpoint and `wal_level` is `hot_standby` (Tom Lane, Andres Freund)

  While this mistake had no particular impact on PostgreSQL itself, it was bad for applications that rely on `txid_current()` and related functions: the TXID value would appear to go backwards.
- Produce an understandable error message if the length of the path name for a Unix-domain socket exceeds the platform-specific limit (Tom Lane, Andrew Dunstan)

  Formerly, this would result in something quite unhelpful, such as “Non-recoverable failure in name resolution” .
- Fix memory leaks when sending composite column values to the client (Tom Lane)
- Make pg_ctl more robust about reading the `postmaster.pid` file (Heikki Linnakangas)

  Fix race conditions and possible file descriptor leakage.
- Fix possible crash in psql if incorrectly-encoded data is presented and the `client_encoding` setting is a client-only encoding, such as SJIS (Jiang Guiqing)
- Fix bugs in the `restore.sql` script emitted by pg_dump in `tar` output format (Tom Lane)

  The script would fail outright on tables whose names include upper-case characters. Also, make the script capable of restoring data in `--inserts` mode as well as the regular COPY mode.
- Fix pg_restore to accept POSIX-conformant `tar` files (Brian Weaver, Tom Lane)

  The original coding of pg_dump's `tar` output mode produced files that are not fully conformant with the POSIX standard. This has been corrected for version 9.3. This patch updates previous branches so that they will accept both the incorrect and the corrected formats, in hopes of avoiding compatibility problems when 9.3 comes out.
- Fix pg_resetxlog to locate `postmaster.pid` correctly when given a relative path to the data directory (Tom Lane)

  This mistake could lead to pg_resetxlog not noticing that there is an active postmaster using the data directory.
- Fix libpq's `lo_import()` and `lo_export()` functions to report file I/O errors properly (Tom Lane)
- Fix ecpg's processing of nested structure pointer variables (Muhammad Usama)
- Fix ecpg's `ecpg_get_data` function to handle arrays properly (Michael Meskes)
- Make `contrib/pageinspect`'s btree page inspection functions take buffer locks while examining pages (Tom Lane)
- Fix pgxs support for building loadable modules on AIX (Tom Lane)

  Building modules outside the original source tree didn't work on AIX.

- Update time zone data files to tzdata release 2012j for DST law changes in Cuba, Israel, Jordan, Libya, Palestine, Western Samoa, and portions of Brazil.

E.64. Release 9.0.10

Release date: 2012-09-24

This release contains a variety of fixes from 9.0.9. For information about new features in the 9.0 major release, see 第 E.74 节.

E.64.1. Migration to Version 9.0.10

A dump/restore is not required for those running 9.0.X.

However, if you are upgrading from a version earlier than 9.0.6, see 第 E.68 节.

E.64.2. Changes

- Fix planner's assignment of executor parameters, and fix executor's rescan logic for CTE plan nodes (Tom Lane)

  These errors could result in wrong answers from queries that scan the same `WITH` subquery multiple times.

- Improve page-splitting decisions in GiST indexes (Alexander Korotkov, Robert Haas, Tom Lane)

  Multi-column GiST indexes might suffer unexpected bloat due to this error.

- Fix cascading privilege revoke to stop if privileges are still held (Tom Lane)

  If we revoke a grant option from some role *X*, but *X* still holds that option via a grant from someone else, we should not recursively revoke the corresponding privilege from role(s) *Y* that *X* had granted it to.

- Improve error messages for Hot Standby misconfiguration errors (Gurjeet Singh)

- Fix handling of `SIGFPE` when PL/Perl is in use (Andres Freund)

  Perl resets the process's `SIGFPE` handler to `SIG_IGN`, which could result in crashes later on. Restore the normal Postgres signal handler after initializing PL/Perl.

- Prevent PL/Perl from crashing if a recursive PL/Perl function is redefined while being executed (Tom Lane)

- Work around possible misoptimization in PL/Perl (Tom Lane)

  Some Linux distributions contain an incorrect version of `pthread.h` that results in incorrect compiled code in PL/Perl, leading to crashes if a PL/Perl function calls another one that throws an error.

- Fix pg_upgrade's handling of line endings on Windows (Andrew Dunstan)

  Previously, pg_upgrade might add or remove carriage returns in places such as function bodies.

- On Windows, make pg_upgrade use backslash path separators in the scripts it emits (Andrew Dunstan)

- Update time zone data files to tzdata release 2012f for DST law changes in Fiji

E.65. Release 9.0.9

Release date: 2012-08-17

This release contains a variety of fixes from 9.0.8. For information about new features in the 9.0 major release, see 第 E.74 节.

E.65.1. Migration to Version 9.0.9

A dump/restore is not required for those running 9.0.X.

However, if you are upgrading from a version earlier than 9.0.6, see 第 E.68 节.

E.65.2. Changes

- Prevent access to external files/URLs via XML entity references (Noah Misch, Tom Lane)

  `xml_parse()` would attempt to fetch external files or URLs as needed to resolve DTD and entity references in an XML value, thus allowing unprivileged database users to attempt to fetch data with the privileges of the database server. While the external data wouldn't get returned directly to the user, portions of it could be exposed in error messages if the data didn't parse as valid XML; and in any case the mere ability to check existence of a file might be useful to an attacker. (CVE-2012-3489)

- Prevent access to external files/URLs via `contrib/xml2`'s `xslt_process()` (Peter Eisentraut)

  libxslt offers the ability to read and write both files and URLs through stylesheet commands, thus allowing unprivileged database users to both read and write data with the privileges of the database server. Disable that through proper use of libxslt's security options. (CVE-2012-3488)

  Also, remove `xslt_process()`'s ability to fetch documents and stylesheets from external files/URLs. While this was a documented "feature", it was long regarded as a bad idea. The fix for CVE-2012-3489 broke that capability, and rather than expend effort on trying to fix it, we're just going to summarily remove it.

- Prevent too-early recycling of btree index pages (Noah Misch)

  When we allowed read-only transactions to skip assigning XIDs, we introduced the possibility that a deleted btree page could be recycled while a read-only transaction was still in flight to it. This would result in incorrect index search results. The probability of such an error occurring in the field seems very low because of the timing requirements, but nonetheless it should be fixed.

- Fix crash-safety bug with newly-created-or-reset sequences (Tom Lane)

  If `ALTER SEQUENCE` was executed on a freshly created or reset sequence, and then precisely one `nextval()` call was made on it, and then the server crashed, WAL replay would restore the sequence to a state in which it appeared that no `nextval()` had been done, thus allowing the first sequence value to be returned again by the next `nextval()` call. In particular this could manifest for `serial` columns, since creation of a serial column's sequence includes an `ALTER SEQUENCE OWNED BY` step.

- Fix `txid_current()` to report the correct epoch when not in hot standby (Heikki Linnakangas)

  This fixes a regression introduced in the previous minor release.

- Fix bug in startup of Hot Standby when a master transaction has many subtransactions (Andres Freund)

  This mistake led to failures reported as “out-of-order XID insertion in KnownAssignedXids” .

- Ensure the `backup_label` file is fsync'd after `pg_start_backup()` (Dave Kerr)

- Fix timeout handling in walsender processes (Tom Lane)

  WAL sender background processes neglected to establish a SIGALRM handler, meaning they would wait forever in some corner cases where a timeout ought to happen.

- Back-patch 9.1 improvement to compress the fsync request queue (Robert Haas)

  This improves performance during checkpoints. The 9.1 change has now seen enough field testing to seem safe to back-patch.

- Fix `LISTEN`/`NOTIFY` to cope better with I/O problems, such as out of disk space (Tom Lane)

  After a write failure, all subsequent attempts to send more `NOTIFY` messages would fail with messages like “Could not read from file "pg_notify/*nnnn*" at offset *nnnnn* : Success” .

- Only allow autovacuum to be auto-canceled by a directly blocked process (Tom Lane)

  The original coding could allow inconsistent behavior in some cases; in particular, an autovacuum could get canceled after less than `deadlock_timeout` grace period.

- Improve logging of autovacuum cancels (Robert Haas)

- Fix log collector so that `log_truncate_on_rotation` works during the very first log rotation after server start (Tom Lane)

- Fix `WITH` attached to a nested set operation (`UNION`/`INTERSECT`/`EXCEPT`) (Tom Lane)

- Ensure that a whole-row reference to a subquery doesn't include any extra `GROUP BY` or `ORDER BY` columns (Tom Lane)

- Disallow copying whole-row references in `CHECK` constraints and index definitions during `CREATE TABLE` (Tom Lane)

  This situation can arise in `CREATE TABLE` with `LIKE` or `INHERITS`. The copied whole-row variable was incorrectly labeled with the row type of the original table not the new one. Rejecting the case seems reasonable for `LIKE`, since the row types might well diverge later. For `INHERITS` we should ideally allow it, with an implicit coercion to the parent table's row type; but that will require more work than seems safe to back-patch.

- Fix memory leak in `ARRAY(SELECT ...)` subqueries (Heikki Linnakangas, Tom Lane)

- Fix extraction of common prefixes from regular expressions (Tom Lane)

  The code could get confused by quantified parenthesized subexpressions, such as `^(foo)?bar`. This would lead to incorrect index optimization of searches for such patterns.

- Fix bugs with parsing signed *hh*:*mm* and *hh*:*mm*:*ss* fields in `interval` constants (Amit Kapila, Tom Lane)

- Use Postgres' encoding conversion functions, not Python's, when converting a Python Unicode string to the server encoding in PL/Python (Jan Urbanski)

  This avoids some corner-case problems, notably that Python doesn't support all the encodings Postgres does. A notable functional change is that if the server encoding is SQL_ASCII, you will get the UTF-8 representation of the string; formerly, any non-ASCII characters in the string would result in an error.

- Fix mapping of PostgreSQL encodings to Python encodings in PL/Python (Jan Urbanski)

- Report errors properly in `contrib/xml2`'s `xslt_process()` (Tom Lane)

- Update time zone data files to tzdata release 2012e for DST law changes in Morocco and Tokelau

E.66. Release 9.0.8

Release date: 2012-06-04

This release contains a variety of fixes from 9.0.7. For information about new features in the 9.0 major release, see 第 E.74 节.

E.66.1. Migration to Version 9.0.8

A dump/restore is not required for those running 9.0.X.

However, if you are upgrading from a version earlier than 9.0.6, see 第 E.68 节.

E.66.2. Changes

- Fix incorrect password transformation in `contrib/pgcrypto`'s DES `crypt()` function (Solar Designer)

  If a password string contained the byte value `0x80`, the remainder of the password was ignored, causing the password to be much weaker than it appeared. With this fix, the rest of the string is properly included in the DES hash. Any stored password values that are affected by this bug will thus no longer match, so the stored values may need to be updated. (CVE-2012-2143)

- Ignore `SECURITY DEFINER` and `SET` attributes for a procedural language's call handler (Tom Lane)

  Applying such attributes to a call handler could crash the server. (CVE-2012-2655)

- Allow numeric timezone offsets in `timestamp` input to be up to 16 hours away from UTC (Tom Lane)

  Some historical time zones have offsets larger than 15 hours, the previous limit. This could result in dumped data values being rejected during reload.

- Fix timestamp conversion to cope when the given time is exactly the last DST transition time for the current timezone (Tom Lane)

  This oversight has been there a long time, but was not noticed previously because most DST-using zones are presumed to have an indefinite sequence of future DST transitions.

- Fix `text` to `name` and `char` to `name` casts to perform string truncation correctly in multibyte encodings (Karl Schnaitter)

- Fix memory copying bug in `to_tsquery()` (Heikki Linnakangas)
- Ensure `txid_current()` reports the correct epoch when executed in hot standby (Simon Riggs)
- Fix planner's handling of outer PlaceHolderVars within subqueries (Tom Lane)

  This bug concerns sub-SELECTs that reference variables coming from the nullable side of an outer join of the surrounding query. In 9.1, queries affected by this bug would fail with "ERROR: Upper-level PlaceHolderVar found where not expected". But in 9.0 and 8.4, you'd silently get possibly-wrong answers, since the value transmitted into the subquery wouldn't go to null when it should.
- Fix slow session startup when `pg_attribute` is very large (Tom Lane)

  If `pg_attribute` exceeds one-fourth of `shared_buffers`, cache rebuilding code that is sometimes needed during session start would trigger the synchronized-scan logic, causing it to take many times longer than normal. The problem was particularly acute if many new sessions were starting at once.
- Ensure sequential scans check for query cancel reasonably often (Merlin Moncure)

  A scan encountering many consecutive pages that contain no live tuples would not respond to interrupts meanwhile.
- Ensure the Windows implementation of `PGSemaphoreLock()` clears `ImmediateInterruptOK` before returning (Tom Lane)

  This oversight meant that a query-cancel interrupt received later in the same query could be accepted at an unsafe time, with unpredictable but not good consequences.
- Show whole-row variables safely when printing views or rules (Abbas Butt, Tom Lane)

  Corner cases involving ambiguous names (that is, the name could be either a table or column name of the query) were printed in an ambiguous way, risking that the view or rule would be interpreted differently after dump and reload. Avoid the ambiguous case by attaching a no-op cast.
- Fix `COPY FROM` to properly handle null marker strings that correspond to invalid encoding (Tom Lane)

  A null marker string such as `E'\\0'` should work, and did work in the past, but the case got broken in 8.4.
- Ensure autovacuum worker processes perform stack depth checking properly (Heikki Linnakangas)

  Previously, infinite recursion in a function invoked by auto-`ANALYZE` could crash worker processes.
- Fix logging collector to not lose log coherency under high load (Andrew Dunstan)

  The collector previously could fail to reassemble large messages if it got too busy.
- Fix logging collector to ensure it will restart file rotation after receiving SIGHUP (Tom Lane)
- Fix WAL replay logic for GIN indexes to not fail if the index was subsequently dropped (Tom Lane)

- Fix memory leak in PL/pgSQL's `RETURN NEXT` command (Joe Conway)
- Fix PL/pgSQL's `GET DIAGNOSTICS` command when the target is the function's first variable (Tom Lane)
- Fix potential access off the end of memory in psql's expanded display (`\x`) mode (Peter Eisentraut)
- Fix several performance problems in pg_dump when the database contains many objects (Jeff Janes, Tom Lane)

  pg_dump could get very slow if the database contained many schemas, or if many objects are in dependency loops, or if there are many owned sequences.
- Fix pg_upgrade for the case that a database stored in a non-default tablespace contains a table in the cluster's default tablespace (Bruce Momjian)
- In ecpg, fix rare memory leaks and possible overwrite of one byte after the `sqlca_t` structure (Peter Eisentraut)
- Fix `contrib/dblink`'s `dblink_exec()` to not leak temporary database connections upon error (Tom Lane)
- Fix `contrib/dblink` to report the correct connection name in error messages (Kyotaro Horiguchi)
- Fix `contrib/vacuumlo` to use multiple transactions when dropping many large objects (Tim Lewis, Robert Haas, Tom Lane)

  This change avoids exceeding `max_locks_per_transaction` when many objects need to be dropped. The behavior can be adjusted with the new `-l` (limit) option.
- Update time zone data files to tzdata release 2012c for DST law changes in Antarctica, Armenia, Chile, Cuba, Falkland Islands, Gaza, Haiti, Hebron, Morocco, Syria, and Tokelau Islands; also historical corrections for Canada.

E.67. Release 9.0.7

Release date: 2012-02-27

This release contains a variety of fixes from 9.0.6. For information about new features in the 9.0 major release, see 第 E.74 节.

E.67.1. Migration to Version 9.0.7

A dump/restore is not required for those running 9.0.X.

However, if you are upgrading from a version earlier than 9.0.6, see 第 E.68 节.

E.67.2. Changes

- Require execute permission on the trigger function for `CREATE TRIGGER` (Robert Haas)

  This missing check could allow another user to execute a trigger function with forged input data, by installing it on a table he owns. This is only of significance for trigger functions marked `SECURITY DEFINER`, since otherwise trigger functions run as the table owner anyway. (CVE-2012-0866)

- Remove arbitrary limitation on length of common name in SSL certificates (Heikki Linnakangas)

  Both libpq and the server truncated the common name extracted from an SSL certificate at 32 bytes. Normally this would cause nothing worse than an unexpected verification failure, but there are some rather-implausible scenarios in which it might allow one certificate holder to impersonate another. The victim would have to have a common name exactly 32 bytes long, and the attacker would have to persuade a trusted CA to issue a certificate in which the common name has that string as a prefix. Impersonating a server would also require some additional exploit to redirect client connections. (CVE-2012-0867)

- Convert newlines to spaces in names written in pg_dump comments (Robert Haas)

  pg_dump was incautious about sanitizing object names that are emitted within SQL comments in its output script. A name containing a newline would at least render the script syntactically incorrect. Maliciously crafted object names could present a SQL injection risk when the script is reloaded. (CVE-2012-0868)

- Fix btree index corruption from insertions concurrent with vacuuming (Tom Lane)

  An index page split caused by an insertion could sometimes cause a concurrently-running `VACUUM` to miss removing index entries that it should remove. After the corresponding table rows are removed, the dangling index entries would cause errors (such as "could not read block N in file ...") or worse, silently wrong query results after unrelated rows are re-inserted at the now-free table locations. This bug has been present since release 8.2, but occurs so infrequently that it was not diagnosed until now. If you have reason to suspect that it has happened in your database, reindexing the affected index will fix things.

- Fix transient zeroing of shared buffers during WAL replay (Tom Lane)

  The replay logic would sometimes zero and refill a shared buffer, so that the contents were transiently invalid. In hot standby mode this can result in a query that's executing in parallel seeing garbage data. Various symptoms could result from that, but the most common one seems to be "invalid memory alloc request size".

- Fix postmaster to attempt restart after a hot-standby crash (Tom Lane)

  A logic error caused the postmaster to terminate, rather than attempt to restart the cluster, if any backend process crashed while operating in hot standby mode.

- Fix `CLUSTER`/`VACUUM FULL` handling of toast values owned by recently-updated rows (Tom Lane)

  This oversight could lead to "duplicate key value violates unique constraint" errors being reported against the toast table's index during one of these commands.

- Update per-column permissions, not only per-table permissions, when changing table owner (Tom Lane)

  Failure to do this meant that any previously granted column permissions were still shown as having been granted by the old owner. This meant that neither the new owner nor a superuser could revoke the now-untraceable-to-table-owner permissions.

- Support foreign data wrappers and foreign servers in `REASSIGN OWNED` (Alvaro Herrera)

  This command failed with "unexpected classid" errors if it needed to change the ownership of any such objects.

- Allow non-existent values for some settings in `ALTER USER/DATABASE SET` (Heikki Linnakangas)

  Allow `default_text_search_config`, `default_tablespace`, and `temp_tablespaces` to be set to names that are not known. This is because they might be known in another database where the setting is intended to be used, or for the tablespace cases because the tablespace might not be created yet. The same issue was previously recognized for `search_path`, and these settings now act like that one.

- Avoid crashing when we have problems deleting table files post-commit (Tom Lane)

  Dropping a table should lead to deleting the underlying disk files only after the transaction commits. In event of failure then (for instance, because of wrong file permissions) the code is supposed to just emit a warning message and go on, since it's too late to abort the transaction. This logic got broken as of release 8.4, causing such situations to result in a PANIC and an unrestartable database.

- Recover from errors occurring during WAL replay of `DROP TABLESPACE` (Tom Lane)

  Replay will attempt to remove the tablespace's directories, but there are various reasons why this might fail (for example, incorrect ownership or permissions on those directories). Formerly the replay code would panic, rendering the database unrestartable without manual intervention. It seems better to log the problem and continue, since the only consequence of failure to remove the directories is some wasted disk space.

- Fix race condition in logging AccessExclusiveLocks for hot standby (Simon Riggs)

  Sometimes a lock would be logged as being held by “transaction zero”. This is at least known to produce assertion failures on slave servers, and might be the cause of more serious problems.

- Track the OID counter correctly during WAL replay, even when it wraps around (Tom Lane)

  Previously the OID counter would remain stuck at a high value until the system exited replay mode. The practical consequences of that are usually nil, but there are scenarios wherein a standby server that's been promoted to master might take a long time to advance the OID counter to a reasonable value once values are needed.

- Prevent emitting misleading “consistent recovery state reached” log message at the beginning of crash recovery (Heikki Linnakangas)

- Fix initial value of `pg_stat_replication.replay_location` (Fujii Masao)

  Previously, the value shown would be wrong until at least one WAL record had been replayed.

- Fix regular expression back-references with `*` attached (Tom Lane)

  Rather than enforcing an exact string match, the code would effectively accept any string that satisfies the pattern sub-expression referenced by the back-reference symbol.

  A similar problem still afflicts back-references that are embedded in a larger quantified expression, rather than being the immediate subject of the quantifier. This will be addressed in a future PostgreSQL release.

- Fix recently-introduced memory leak in processing of `inet/cidr` values (Heikki Linnakangas)

A patch in the December 2011 releases of PostgreSQL caused memory leakage in these operations, which could be significant in scenarios such as building a btree index on such a column.

- Fix dangling pointer after `CREATE TABLE AS/SELECT INTO` in a SQL-language function (Tom Lane)

  In most cases this only led to an assertion failure in assert-enabled builds, but worse consequences seem possible.

- Avoid double close of file handle in syslogger on Windows (MauMau)

  Ordinarily this error was invisible, but it would cause an exception when running on a debug version of Windows.

- Fix I/O-conversion-related memory leaks in plpgsql (Andres Freund, Jan Urbanski, Tom Lane)

  Certain operations would leak memory until the end of the current function.

- Improve pg_dump's handling of inherited table columns (Tom Lane)

  pg_dump mishandled situations where a child column has a different default expression than its parent column. If the default is textually identical to the parent's default, but not actually the same (for instance, because of schema search path differences) it would not be recognized as different, so that after dump and restore the child would be allowed to inherit the parent's default. Child columns that are `NOT NULL` where their parent is not could also be restored subtly incorrectly.

- Fix pg_restore's direct-to-database mode for INSERT-style table data (Tom Lane)

  Direct-to-database restores from archive files made with `--inserts` or `--column-inserts` options fail when using pg_restore from a release dated September or December 2011, as a result of an oversight in a fix for another problem. The archive file itself is not at fault, and text-mode output is okay.

- Allow pg_upgrade to process tables containing `regclass` columns (Bruce Momjian)

  Since pg_upgrade now takes care to preserve `pg_class` OIDs, there was no longer any reason for this restriction.

- Make libpq ignore `ENOTDIR` errors when looking for an SSL client certificate file (Magnus Hagander)

  This allows SSL connections to be established, though without a certificate, even when the user's home directory is set to something like `/dev/null`.

- Fix some more field alignment issues in ecpg's SQLDA area (Zoltan Boszormenyi)

- Allow `AT` option in ecpg `DEALLOCATE` statements (Michael Meskes)

  The infrastructure to support this has been there for awhile, but through an oversight there was still an error check rejecting the case.

- Do not use the variable name when defining a varchar structure in ecpg (Michael Meskes)

- Fix `contrib/auto_explain`'s JSON output mode to produce valid JSON (Andrew Dunstan)

  The output used brackets at the top level, when it should have used braces.

- Fix error in `contrib/intarray`'s `int[] & int[]` operator (Guillaume Lelarge)

  If the smallest integer the two input arrays have in common is 1, and there are smaller values in either array, then 1 would be incorrectly omitted from the result.

- Fix error detection in `contrib/pgcrypto`'s `encrypt_iv()` and `decrypt_iv()` (Marko Kreen)

  These functions failed to report certain types of invalid-input errors, and would instead return random garbage values for incorrect input.

- Fix one-byte buffer overrun in `contrib/test_parser` (Paul Guyot)

  The code would try to read one more byte than it should, which would crash in corner cases. Since `contrib/test_parser` is only example code, this is not a security issue in itself, but bad example code is still bad.

- Use `__sync_lock_test_and_set()` for spinlocks on ARM, if available (Martin Pitt)

  This function replaces our previous use of the `SWPB` instruction, which is deprecated and not available on ARMv6 and later. Reports suggest that the old code doesn't fail in an obvious way on recent ARM boards, but simply doesn't interlock concurrent accesses, leading to bizarre failures in multiprocess operation.

- Use `-fexcess-precision=standard` option when building with gcc versions that accept it (Andrew Dunstan)

  This prevents assorted scenarios wherein recent versions of gcc will produce creative results.

- Allow use of threaded Python on FreeBSD (Chris Rees)

  Our configure script previously believed that this combination wouldn't work; but FreeBSD fixed the problem, so remove that error check.

E.68. Release 9.0.6

Release date: 2011-12-05

This release contains a variety of fixes from 9.0.5. For information about new features in the 9.0 major release, see 第 E.74 节.

E.68.1. Migration to Version 9.0.6

A dump/restore is not required for those running 9.0.X.

However, a longstanding error was discovered in the definition of the `information_schema.referential_constraints` view. If you rely on correct results from that view, you should replace its definition as explained in the first changelog item below.

Also, if you are upgrading from a version earlier than 9.0.4, see 第 E.70 节.

E.68.2. Changes

- Fix bugs in `information_schema.referential_constraints` view (Tom Lane)

  This view was being insufficiently careful about matching the foreign-key constraint to the depended-on primary or unique key constraint. That could result in failure to show a foreign key constraint at all, or showing it multiple times, or claiming that it depends on a different constraint than the one it really does.

Since the view definition is installed by initdb, merely upgrading will not fix the problem. If you need to fix this in an existing installation, you can (as a superuser) drop the `information_schema` schema then re-create it by sourcing `SHAREDIR /information_schema.sql`. (Run `pg_config --sharedir` if you're uncertain where `SHAREDIR` is.) This must be repeated in each database to be fixed.

- Fix possible crash during `UPDATE` or `DELETE` that joins to the output of a scalar-returning function (Tom Lane)

  A crash could only occur if the target row had been concurrently updated, so this problem surfaced only intermittently.

- Fix incorrect replay of WAL records for GIN index updates (Tom Lane)

  This could result in transiently failing to find index entries after a crash, or on a hot-standby server. The problem would be repaired by the next `VACUUM` of the index, however.

- Fix TOAST-related data corruption during `CREATE TABLE dest AS SELECT * FROM src` or `INSERT INTO dest SELECT * FROM src` (Tom Lane)

  If a table has been modified by `ALTER TABLE ADD COLUMN`, attempts to copy its data verbatim to another table could produce corrupt results in certain corner cases. The problem can only manifest in this precise form in 8.4 and later, but we patched earlier versions as well in case there are other code paths that could trigger the same bug.

- Fix possible failures during hot standby startup (Simon Riggs)

- Start hot standby faster when initial snapshot is incomplete (Simon Riggs)

- Fix race condition during toast table access from stale syscache entries (Tom Lane)

  The typical symptom was transient errors like “missing chunk number 0 for toast value NNNNN in pg_toast_2619”, where the cited toast table would always belong to a system catalog.

- Track dependencies of functions on items used in parameter default expressions (Tom Lane)

  Previously, a referenced object could be dropped without having dropped or modified the function, leading to misbehavior when the function was used. Note that merely installing this update will not fix the missing dependency entries; to do that, you'd need to `CREATE OR REPLACE` each such function afterwards. If you have functions whose defaults depend on non-built-in objects, doing so is recommended.

- Allow inlining of set-returning SQL functions with multiple OUT parameters (Tom Lane)

- Don't trust deferred-unique indexes for join removal (Tom Lane and Marti Raudsepp)

  A deferred uniqueness constraint might not hold intra-transaction, so assuming that it does could give incorrect query results.

- Make `DatumGetInetP()` unpack inet datums that have a 1-byte header, and add a new macro, `DatumGetInetPP()`, that does not (Heikki Linnakangas)

  This change affects no core code, but might prevent crashes in add-on code that expects `DatumGetInetP()` to produce an unpacked datum as per usual convention.

- Improve locale support in `money` type's input and output (Tom Lane)

  Aside from not supporting all standard `lc_monetary` formatting options, the input and output functions were inconsistent, meaning there were locales in which dumped `money` values could not be re-read.

- Don't let `transform_null_equals` affect `CASE foo WHEN NULL ...` constructs (Heikki Linnakangas)

  `transform_null_equals` is only supposed to affect `foo = NULL` expressions written directly by the user, not equality checks generated internally by this form of `CASE`.

- Change foreign-key trigger creation order to better support self-referential foreign keys (Tom Lane)

  For a cascading foreign key that references its own table, a row update will fire both the `ON UPDATE` trigger and the `CHECK` trigger as one event. The `ON UPDATE` trigger must execute first, else the `CHECK` will check a non-final state of the row and possibly throw an inappropriate error. However, the firing order of these triggers is determined by their names, which generally sort in creation order since the triggers have auto-generated names following the convention "RI_ConstraintTrigger_NNNN". A proper fix would require modifying that convention, which we will do in 9.2, but it seems risky to change it in existing releases. So this patch just changes the creation order of the triggers. Users encountering this type of error should drop and re-create the foreign key constraint to get its triggers into the right order.

- Avoid floating-point underflow while tracking buffer allocation rate (Greg Matthews)

  While harmless in itself, on certain platforms this would result in annoying kernel log messages.

- Preserve configuration file name and line number values when starting child processes under Windows (Tom Lane)

  Formerly, these would not be displayed correctly in the `pg_settings` view.

- Fix incorrect field alignment in ecpg's SQLDA area (Zoltan Boszormenyi)

- Preserve blank lines within commands in psql's command history (Robert Haas)

  The former behavior could cause problems if an empty line was removed from within a string literal, for example.

- Fix pg_dump to dump user-defined casts between auto-generated types, such as table rowtypes (Tom Lane)

- Assorted fixes for pg_upgrade (Bruce Momjian)

  Handle exclusion constraints correctly, avoid failures on Windows, don't complain about mismatched toast table names in 8.4 databases.

- Use the preferred version of xsubpp to build PL/Perl, not necessarily the operating system's main copy (David Wheeler and Alex Hunsaker)

- Fix incorrect coding in `contrib/dict_int` and `contrib/dict_xsyn` (Tom Lane)

  Some functions incorrectly assumed that memory returned by `palloc()` is guaranteed zeroed.

- Fix assorted errors in `contrib/unaccent`'s configuration file parsing (Tom Lane)
- Honor query cancel interrupts promptly in `pgstatindex()` (Robert Haas)
- Fix incorrect quoting of log file name in Mac OS X start script (Sidar Lopez)
- Ensure VPATH builds properly install all server header files (Peter Eisentraut)
- Shorten file names reported in verbose error messages (Peter Eisentraut)

  Regular builds have always reported just the name of the C file containing the error message call, but VPATH builds formerly reported an absolute path name.
- Fix interpretation of Windows timezone names for Central America (Tom Lane)

  Map "Central America Standard Time" to `CST6`, not `CST6CDT`, because DST is generally not observed anywhere in Central America.
- Update time zone data files to tzdata release 2011n for DST law changes in Brazil, Cuba, Fiji, Palestine, Russia, and Samoa; also historical corrections for Alaska and British East Africa.

E.69. Release 9.0.5

Release date: 2011-09-26

This release contains a variety of fixes from 9.0.4. For information about new features in the 9.0 major release, see 第 E.74 节.

E.69.1. Migration to Version 9.0.5

A dump/restore is not required for those running 9.0.X.

However, if you are upgrading from a version earlier than 9.0.4, see 第 E.70 节.

E.69.2. Changes

- Fix catalog cache invalidation after a `VACUUM FULL` or `CLUSTER` on a system catalog (Tom Lane)

  In some cases the relocation of a system catalog row to another place would not be recognized by concurrent server processes, allowing catalog corruption to occur if they then tried to update that row. The worst-case outcome could be as bad as complete loss of a table.
- Fix incorrect order of operations during sinval reset processing, and ensure that TOAST OIDs are preserved in system catalogs (Tom Lane)

  These mistakes could lead to transient failures after a `VACUUM FULL` or `CLUSTER` on a system catalog.
- Fix bugs in indexing of in-doubt HOT-updated tuples (Tom Lane)

  These bugs could result in index corruption after reindexing a system catalog. They are not believed to affect user indexes.
- Fix multiple bugs in GiST index page split processing (Heikki Linnakangas)

  The probability of occurrence was low, but these could lead to index corruption.

- Fix possible buffer overrun in `tsvector_concat()` (Tom Lane)

  The function could underestimate the amount of memory needed for its result, leading to server crashes.

- Fix crash in `xml_recv` when processing a "standalone" parameter (Tom Lane)

- Make `pg_options_to_table` return NULL for an option with no value (Tom Lane)

  Previously such cases would result in a server crash.

- Avoid possibly accessing off the end of memory in `ANALYZE` and in SJIS-2004 encoding conversion (Noah Misch)

  This fixes some very-low-probability server crash scenarios.

- Protect `pg_stat_reset_shared()` against NULL input (Magnus Hagander)

- Fix possible failure when a recovery conflict deadlock is detected within a sub-transaction (Tom Lane)

- Avoid spurious conflicts while recycling btree index pages during hot standby (Noah Misch, Simon Riggs)

- Shut down WAL receiver if it's still running at end of recovery (Heikki Linnakangas)

  The postmaster formerly panicked in this situation, but it's actually a legitimate case.

- Fix race condition in relcache init file invalidation (Tom Lane)

  There was a window wherein a new backend process could read a stale init file but miss the inval messages that would tell it the data is stale. The result would be bizarre failures in catalog accesses, typically "could not read block 0 in file ..." later during startup.

- Fix memory leak at end of a GiST index scan (Tom Lane)

  Commands that perform many separate GiST index scans, such as verification of a new GiST-based exclusion constraint on a table already containing many rows, could transiently require large amounts of memory due to this leak.

- Fix memory leak when encoding conversion has to be done on incoming command strings and `LISTEN` is active (Tom Lane)

- Fix incorrect memory accounting (leading to possible memory bloat) in tuplestores supporting holdable cursors and plpgsql's `RETURN NEXT` command (Tom Lane)

- Fix trigger `WHEN` conditions when both `BEFORE` and `AFTER` triggers exist (Tom Lane)

  Evaluation of `WHEN` conditions for `AFTER ROW UPDATE` triggers could crash if there had been a `BEFORE ROW` trigger fired for the same update.

- Fix performance problem when constructing a large, lossy bitmap (Tom Lane)

- Fix join selectivity estimation for unique columns (Tom Lane)

  This fixes an erroneous planner heuristic that could lead to poor estimates of the result size of a join.

- Fix nested PlaceHolderVar expressions that appear only in sub-select target lists (Tom Lane)

This mistake could result in outputs of an outer join incorrectly appearing as NULL.

- Allow the planner to assume that empty parent tables really are empty (Tom Lane)

  Normally an empty table is assumed to have a certain minimum size for planning purposes; but this heuristic seems to do more harm than good for the parent table of an inheritance hierarchy, which often is permanently empty.

- Allow nested `EXISTS` queries to be optimized properly (Tom Lane)

- Fix array- and path-creating functions to ensure padding bytes are zeroes (Tom Lane)

  This avoids some situations where the planner will think that semantically-equal constants are not equal, resulting in poor optimization.

- Fix `EXPLAIN` to handle gating Result nodes within inner-indexscan subplans (Tom Lane)

  The usual symptom of this oversight was "bogus varno" errors.

- Fix btree preprocessing of *`indexedcol`* `IS NULL` conditions (Dean Rasheed)

  Such a condition is unsatisfiable if combined with any other type of btree-indexable condition on the same index column. The case was handled incorrectly in 9.0.0 and later, leading to query output where there should be none.

- Work around gcc 4.6.0 bug that breaks WAL replay (Tom Lane)

  This could lead to loss of committed transactions after a server crash.

- Fix dump bug for `VALUES` in a view (Tom Lane)

- Disallow `SELECT FOR UPDATE/SHARE` on sequences (Tom Lane)

  This operation doesn't work as expected and can lead to failures.

- Fix `VACUUM` so that it always updates `pg_class.reltuples/relpages` (Tom Lane)

  This fixes some scenarios where autovacuum could make increasingly poor decisions about when to vacuum tables.

- Defend against integer overflow when computing size of a hash table (Tom Lane)

- Fix cases where `CLUSTER` might attempt to access already-removed TOAST data (Tom Lane)

- Fix premature timeout failures during initial authentication transaction (Tom Lane)

- Fix portability bugs in use of credentials control messages for "peer" authentication (Tom Lane)

- Fix SSPI login when multiple roundtrips are required (Ahmed Shinwari, Magnus Hagander)

  The typical symptom of this problem was "The function requested is not supported" errors during SSPI login.

- Fix failure when adding a new variable of a custom variable class to `postgresql.conf` (Tom Lane)

- Throw an error if `pg_hba.conf` contains `hostssl` but SSL is disabled (Tom Lane)

  This was concluded to be more user-friendly than the previous behavior of silently ignoring such lines.

- Fix failure when `DROP OWNED BY` attempts to remove default privileges on sequences (Shigeru Hanada)

- Fix typo in `pg_srand48` seed initialization (Andres Freund)

  This led to failure to use all bits of the provided seed. This function is not used on most platforms (only those without `srandom`), and the potential security exposure from a less-random-than-expected seed seems minimal in any case.

- Avoid integer overflow when the sum of `LIMIT` and `OFFSET` values exceeds 2^63 (Heikki Linnakangas)

- Add overflow checks to `int4` and `int8` versions of `generate_series()` (Robert Haas)

- Fix trailing-zero removal in `to_char()` (Marti Raudsepp)

  In a format with `FM` and no digit positions after the decimal point, zeroes to the left of the decimal point could be removed incorrectly.

- Fix `pg_size_pretty()` to avoid overflow for inputs close to 2^63 (Tom Lane)

- Weaken plpgsql's check for typmod matching in record values (Tom Lane)

  An overly enthusiastic check could lead to discarding length modifiers that should have been kept.

- Correctly handle quotes in locale names during initdb (Heikki Linnakangas)

  The case can arise with some Windows locales, such as “People's Republic of China” .

- In pg_upgrade, avoid dumping orphaned temporary tables (Bruce Momjian)

  This prevents situations wherein table OID assignments could get out of sync between old and new installations.

- Fix pg_upgrade to preserve toast tables' relfrozenxids during an upgrade from 8.3 (Bruce Momjian)

  Failure to do this could lead to `pg_clog` files being removed too soon after the upgrade.

- In pg_upgrade, fix the `-l` (log) option to work on Windows (Bruce Momjian)

- In pg_ctl, support silent mode for service registrations on Windows (MauMau)

- Fix psql's counting of script file line numbers during `COPY` from a different file (Tom Lane)

- Fix pg_restore's direct-to-database mode for `standard_conforming_strings` (Tom Lane)

  pg_restore could emit incorrect commands when restoring directly to a database server from an archive file that had been made with `standard_conforming_strings` set to `on`.

- Be more user-friendly about unsupported cases for parallel pg_restore (Tom Lane)

  This change ensures that such cases are detected and reported before any restore actions have been taken.

- Fix write-past-buffer-end and memory leak in libpq's LDAP service lookup code (Albe Laurenz)

- In libpq, avoid failures when using nonblocking I/O and an SSL connection (Martin Pihlak, Tom Lane)

- Improve libpq's handling of failures during connection startup (Tom Lane)

  In particular, the response to a server report of `fork()` failure during SSL connection startup is now saner.

- Improve libpq's error reporting for SSL failures (Tom Lane)

- Fix `PQsetvalue()` to avoid possible crash when adding a new tuple to a `PGresult` originally obtained from a server query (Andrew Chernow)

- Make ecpglib write `double` values with 15 digits precision (Akira Kurosawa)

- In ecpglib, be sure `LC_NUMERIC` setting is restored after an error (Michael Meskes)

- Apply upstream fix for blowfish signed-character bug (CVE-2011-2483) (Tom Lane)

  `contrib/pg_crypto`'s blowfish encryption code could give wrong results on platforms where char is signed (which is most), leading to encrypted passwords being weaker than they should be.

- Fix memory leak in `contrib/seg` (Heikki Linnakangas)

- Fix `pgstatindex()` to give consistent results for empty indexes (Tom Lane)

- Allow building with perl 5.14 (Alex Hunsaker)

- Fix assorted issues with build and install file paths containing spaces (Tom Lane)

- Update time zone data files to tzdata release 2011i for DST law changes in Canada, Egypt, Russia, Samoa, and South Sudan.

E.70. Release 9.0.4

Release date: 2011-04-18

This release contains a variety of fixes from 9.0.3. For information about new features in the 9.0 major release, see 第 E.74 节.

E.70.1. Migration to Version 9.0.4

A dump/restore is not required for those running 9.0.X.

However, if your installation was upgraded from a previous major release by running pg_upgrade, you should take action to prevent possible data loss due to a now-fixed bug in pg_upgrade. The recommended solution is to run `VACUUM FREEZE` on all TOAST tables. More information is available at http://wiki.postgresql.org/wiki/20110408pg_upgrade_fix[3].

[3] http://wiki.postgresql.org/wiki/20110408pg_upgrade_fix

E.70.2. Changes

- Fix pg_upgrade's handling of TOAST tables (Bruce Momjian)

  The `pg_class.relfrozenxid` value for TOAST tables was not correctly copied into the new installation during pg_upgrade. This could later result in `pg_clog` files being discarded while they were still needed to validate tuples in the TOAST tables, leading to "could not access status of transaction" failures.

  This error poses a significant risk of data loss for installations that have been upgraded with pg_upgrade. This patch corrects the problem for future uses of pg_upgrade, but does not in itself cure the issue in installations that have been processed with a buggy version of pg_upgrade.

- Suppress incorrect "PD_ALL_VISIBLE flag was incorrectly set" warning (Heikki Linnakangas)

  `VACUUM` would sometimes issue this warning in cases that are actually valid.

- Use better SQLSTATE error codes for hot standby conflict cases (Tatsuo Ishii and Simon Riggs)

  All retryable conflict errors now have an error code that indicates that a retry is possible. Also, session closure due to the database being dropped on the master is now reported as `ERRCODE_DATABASE_DROPPED`, rather than `ERRCODE_ADMIN_SHUTDOWN`, so that connection poolers can handle the situation correctly.

- Prevent intermittent hang in interactions of startup process with bgwriter process (Simon Riggs)

  This affected recovery in non-hot-standby cases.

- Disallow including a composite type in itself (Tom Lane)

  This prevents scenarios wherein the server could recurse infinitely while processing the composite type. While there are some possible uses for such a structure, they don't seem compelling enough to justify the effort required to make sure it always works safely.

- Avoid potential deadlock during catalog cache initialization (Nikhil Sontakke)

  In some cases the cache loading code would acquire share lock on a system index before locking the index's catalog. This could deadlock against processes trying to acquire exclusive locks in the other, more standard order.

- Fix dangling-pointer problem in `BEFORE ROW UPDATE` trigger handling when there was a concurrent update to the target tuple (Tom Lane)

  This bug has been observed to result in intermittent "cannot extract system attribute from virtual tuple" failures while trying to do `UPDATE RETURNING ctid`. There is a very small probability of more serious errors, such as generating incorrect index entries for the updated tuple.

- Disallow `DROP TABLE` when there are pending deferred trigger events for the table (Tom Lane)

  Formerly the `DROP` would go through, leading to "could not open relation with OID nnn" errors when the triggers were eventually fired.

- Allow "replication" as a user name in `pg_hba.conf` (Andrew Dunstan)

"replication" is special in the database name column, but it was mistakenly also treated as special in the user name column.

- Prevent crash triggered by constant-false WHERE conditions during GEQO optimization (Tom Lane)
- Improve planner's handling of semi-join and anti-join cases (Tom Lane)
- Fix handling of `SELECT FOR UPDATE` in a sub-SELECT (Tom Lane)

  This bug typically led to "cannot extract system attribute from virtual tuple" errors.
- Fix selectivity estimation for text search to account for NULLs (Jesper Krogh)
- Fix get_actual_variable_range() to support hypothetical indexes injected by an index adviser plugin (Gurjeet Singh)
- Fix PL/Python memory leak involving array slices (Daniel Popowich)
- Allow libpq's SSL initialization to succeed when user's home directory is unavailable (Tom Lane)

  If the SSL mode is such that a root certificate file is not required, there is no need to fail. This change restores the behavior to what it was in pre-9.0 releases.
- Fix libpq to return a useful error message for errors detected in `conninfo_array_parse` (Joseph Adams)

  A typo caused the library to return NULL, rather than the `PGconn` structure containing the error message, to the application.
- Fix ecpg preprocessor's handling of float constants (Heikki Linnakangas)
- Fix parallel pg_restore to handle comments on POST_DATA items correctly (Arnd Hannemann)
- Fix pg_restore to cope with long lines (over 1KB) in TOC files (Tom Lane)
- Put in more safeguards against crashing due to division-by-zero with overly enthusiastic compiler optimization (Aurelien Jarno)
- Support use of dlopen() in FreeBSD and OpenBSD on MIPS (Tom Lane)

  There was a hard-wired assumption that this system function was not available on MIPS hardware on these systems. Use a compile-time test instead, since more recent versions have it.
- Fix compilation failures on HP-UX (Heikki Linnakangas)
- Avoid crash when trying to write to the Windows console very early in process startup (Rushabh Lathia)
- Support building with MinGW 64 bit compiler for Windows (Andrew Dunstan)
- Fix version-incompatibility problem with libintl on Windows (Hiroshi Inoue)
- Fix usage of xcopy in Windows build scripts to work correctly under Windows 7 (Andrew Dunstan)

  This affects the build scripts only, not installation or usage.
- Fix path separator used by pg_regress on Cygwin (Andrew Dunstan)

- Update time zone data files to tzdata release 2011f for DST law changes in Chile, Cuba, Falkland Islands, Morocco, Samoa, and Turkey; also historical corrections for South Australia, Alaska, and Hawaii.

E.71. Release 9.0.3

Release date: 2011-01-31

This release contains a variety of fixes from 9.0.2. For information about new features in the 9.0 major release, see 第 E.74 节.

E.71.1. Migration to Version 9.0.3

A dump/restore is not required for those running 9.0.X.

E.71.2. Changes

- Before exiting walreceiver, ensure all the received WAL is fsync'd to disk (Heikki Linnakangas)

  Otherwise the standby server could replay some un-synced WAL, conceivably leading to data corruption if the system crashes just at that point.

- Avoid excess fsync activity in walreceiver (Heikki Linnakangas)

- Make `ALTER TABLE` revalidate uniqueness and exclusion constraints when needed (Noah Misch)

  This was broken in 9.0 by a change that was intended to suppress revalidation during `VACUUM FULL` and `CLUSTER`, but unintentionally affected `ALTER TABLE` as well.

- Fix EvalPlanQual for `UPDATE` of an inheritance tree in which the tables are not all alike (Tom Lane)

  Any variation in the table row types (including dropped columns present in only some child tables) would confuse the EvalPlanQual code, leading to misbehavior or even crashes. Since EvalPlanQual is only executed during concurrent updates to the same row, the problem was only seen intermittently.

- Avoid failures when `EXPLAIN` tries to display a simple-form `CASE` expression (Tom Lane)

  If the `CASE`'s test expression was a constant, the planner could simplify the `CASE` into a form that confused the expression-display code, resulting in “unexpected CASE WHEN clause” errors.

- Fix assignment to an array slice that is before the existing range of subscripts (Tom Lane)

  If there was a gap between the newly added subscripts and the first pre-existing subscript, the code miscalculated how many entries needed to be copied from the old array's null bitmap, potentially leading to data corruption or crash.

- Avoid unexpected conversion overflow in planner for very distant date values (Tom Lane)

  The `date` type supports a wider range of dates than can be represented by the `timestamp` types, but the planner assumed it could always convert a date to timestamp with impunity.

- Fix PL/Python crash when an array contains null entries (Alex Hunsaker)
- Remove ecpg's fixed length limit for constants defining an array dimension (Michael Meskes)
- Fix erroneous parsing of `tsquery` values containing `... & !(subexpression) | ...` (Tom Lane)

  Queries containing this combination of operators were not executed correctly. The same error existed in `contrib/intarray`'s `query_int` type and `contrib/ltree`'s `ltxtquery` type.
- Fix buffer overrun in `contrib/intarray`'s input function for the `query_int` type (Apple)

  This bug is a security risk since the function's return address could be overwritten. Thanks to Apple Inc's security team for reporting this issue and supplying the fix. (CVE-2010-4015)
- Fix bug in `contrib/seg`'s GiST picksplit algorithm (Alexander Korotkov)

  This could result in considerable inefficiency, though not actually incorrect answers, in a GiST index on a `seg` column. If you have such an index, consider `REINDEX`ing it after installing this update. (This is identical to the bug that was fixed in `contrib/cube` in the previous update.)

E.72. Release 9.0.2

Release date: 2010-12-16

This release contains a variety of fixes from 9.0.1. For information about new features in the 9.0 major release, see 第 E.74 节.

E.72.1. Migration to Version 9.0.2

A dump/restore is not required for those running 9.0.X.

E.72.2. Changes

- Force the default `wal_sync_method` to be `fdatasync` on Linux (Tom Lane, Marti Raudsepp)

  The default on Linux has actually been `fdatasync` for many years, but recent kernel changes caused PostgreSQL to choose `open_datasync` instead. This choice did not result in any performance improvement, and caused outright failures on certain filesystems, notably `ext4` with the `data=journal` mount option.
- Fix “too many KnownAssignedXids” error during Hot Standby replay (Heikki Linnakangas)
- Fix race condition in lock acquisition during Hot Standby (Simon Riggs)
- Avoid unnecessary conflicts during Hot Standby (Simon Riggs)

  This fixes some cases where replay was considered to conflict with standby queries (causing delay of replay or possibly cancellation of the queries), but there was no real conflict.

- Fix assorted bugs in WAL replay logic for GIN indexes (Tom Lane)

  This could result in "bad buffer id: 0" failures or corruption of index contents during replication.

- Fix recovery from base backup when the starting checkpoint WAL record is not in the same WAL segment as its redo point (Jeff Davis)

- Fix corner-case bug when streaming replication is enabled immediately after creating the master database cluster (Heikki Linnakangas)

- Fix persistent slowdown of autovacuum workers when multiple workers remain active for a long time (Tom Lane)

  The effective `vacuum_cost_limit` for an autovacuum worker could drop to nearly zero if it processed enough tables, causing it to run extremely slowly.

- Fix long-term memory leak in autovacuum launcher (Alvaro Herrera)

- Avoid failure when trying to report an impending transaction wraparound condition from outside a transaction (Tom Lane)

  This oversight prevented recovery after transaction wraparound got too close, because database startup processing would fail.

- Add support for detecting register-stack overrun on `IA64` (Tom Lane)

  The `IA64` architecture has two hardware stacks. Full prevention of stack-overrun failures requires checking both.

- Add a check for stack overflow in `copyObject()` (Tom Lane)

  Certain code paths could crash due to stack overflow given a sufficiently complex query.

- Fix detection of page splits in temporary GiST indexes (Heikki Linnakangas)

  It is possible to have a "concurrent" page split in a temporary index, if for example there is an open cursor scanning the index when an insertion is done. GiST failed to detect this case and hence could deliver wrong results when execution of the cursor continued.

- Fix error checking during early connection processing (Tom Lane)

  The check for too many child processes was skipped in some cases, possibly leading to postmaster crash when attempting to add the new child process to fixed-size arrays.

- Improve efficiency of window functions (Tom Lane)

  Certain cases where a large number of tuples needed to be read in advance, but `work_mem` was large enough to allow them all to be held in memory, were unexpectedly slow. `percent_rank()`, `cume_dist()` and `ntile()` in particular were subject to this problem.

- Avoid memory leakage while `ANALYZE`'ing complex index expressions (Tom Lane)

- Ensure an index that uses a whole-row Var still depends on its table (Tom Lane)

  An index declared like `create index i on t (foo(t.*))` would not automatically get dropped when its table was dropped.

- Add missing support in `DROP OWNED BY` for removing foreign data wrapper/server privileges belonging to a user (Heikki Linnakangas)
- Do not "inline" a SQL function with multiple `OUT` parameters (Tom Lane)

  This avoids a possible crash due to loss of information about the expected result rowtype.
- Fix crash when inline-ing a set-returning function whose argument list contains a reference to an inline-able user function (Tom Lane)
- Behave correctly if `ORDER BY`, `LIMIT`, `FOR UPDATE`, or `WITH` is attached to the `VALUES` part of `INSERT ... VALUES` (Tom Lane)
- Make the `OFF` keyword unreserved (Heikki Linnakangas)

  This prevents problems with using `off` as a variable name in PL/pgSQL. That worked before 9.0, but was now broken because PL/pgSQL now treats all core reserved words as reserved.
- Fix constant-folding of `COALESCE()` expressions (Tom Lane)

  The planner would sometimes attempt to evaluate sub-expressions that in fact could never be reached, possibly leading to unexpected errors.
- Fix "could not find pathkey item to sort" planner failure with comparison of whole-row Vars (Tom Lane)
- Fix postmaster crash when connection acceptance (`accept()` or one of the calls made immediately after it) fails, and the postmaster was compiled with GSSAPI support (Alexander Chernikov)
- Retry after receiving an invalid response packet from a RADIUS authentication server (Magnus Hagander)

  This fixes a low-risk potential denial of service condition.
- Fix missed unlink of temporary files when `log_temp_files` is active (Tom Lane)

  If an error occurred while attempting to emit the log message, the unlink was not done, resulting in accumulation of temp files.
- Add print functionality for `InhRelation` nodes (Tom Lane)

  This avoids a failure when `debug_print_parse` is enabled and certain types of query are executed.
- Fix incorrect calculation of distance from a point to a horizontal line segment (Tom Lane)

  This bug affected several different geometric distance-measurement operators.
- Fix incorrect calculation of transaction status in ecpg (Itagaki Takahiro)
- Fix errors in psql's Unicode-escape support (Tom Lane)
- Speed up parallel pg_restore when the archive contains many large objects (blobs) (Tom Lane)
- Fix PL/pgSQL's handling of "simple" expressions to not fail in recursion or error-recovery cases (Tom Lane)

- Fix PL/pgSQL's error reporting for no-such-column cases (Tom Lane)

  As of 9.0, it would sometimes report "missing FROM-clause entry for table foo" when "record foo has no field bar" would be more appropriate.

- Fix PL/Python to honor typmod (i.e., length or precision restrictions) when assigning to tuple fields (Tom Lane)

  This fixes a regression from 8.4.

- Fix PL/Python's handling of set-returning functions (Jan Urbanski)

  Attempts to call SPI functions within the iterator generating a set result would fail.

- Fix bug in `contrib/cube`'s GiST picksplit algorithm (Alexander Korotkov)

  This could result in considerable inefficiency, though not actually incorrect answers, in a GiST index on a `cube` column. If you have such an index, consider `REINDEX`ing it after installing this update.

- Don't emit "identifier will be truncated" notices in `contrib/dblink` except when creating new connections (Itagaki Takahiro)

- Fix potential coredump on missing public key in `contrib/pgcrypto` (Marti Raudsepp)

- Fix buffer overrun in `contrib/pg_upgrade` (Hernan Gonzalez)

- Fix memory leak in `contrib/xml2`'s XPath query functions (Tom Lane)

- Update time zone data files to tzdata release 2010o for DST law changes in Fiji and Samoa; also historical corrections for Hong Kong.

E.73. Release 9.0.1

Release date: 2010-10-04

This release contains a variety of fixes from 9.0.0. For information about new features in the 9.0 major release, see 第 E.74 节.

E.73.1. Migration to Version 9.0.1

A dump/restore is not required for those running 9.0.X.

E.73.2. Changes

- Use a separate interpreter for each calling SQL userid in PL/Perl and PL/Tcl (Tom Lane)

  This change prevents security problems that can be caused by subverting Perl or Tcl code that will be executed later in the same session under another SQL user identity (for example, within a `SECURITY DEFINER` function). Most scripting languages offer numerous ways that that might be done, such as redefining standard functions or operators called by the target function. Without this change, any SQL user with Perl or Tcl language usage rights can do essentially anything with the SQL privileges of the target function's owner.

  The cost of this change is that intentional communication among Perl and Tcl functions becomes more difficult. To provide an escape hatch, PL/PerlU and PL/TclU

functions continue to use only one interpreter per session. This is not considered a security issue since all such functions execute at the trust level of a database superuser already.

It is likely that third-party procedural languages that claim to offer trusted execution have similar security issues. We advise contacting the authors of any PL you are depending on for security-critical purposes.

Our thanks to Tim Bunce for pointing out this issue (CVE-2010-3433).

- Improve `pg_get_expr()` security fix so that the function can still be used on the output of a sub-select (Tom Lane)
- Fix incorrect placement of placeholder evaluation (Tom Lane)

  This bug could result in query outputs being non-null when they should be null, in cases where the inner side of an outer join is a sub-select with non-strict expressions in its output list.
- Fix join removal's handling of placeholder expressions (Tom Lane)
- Fix possible duplicate scans of `UNION ALL` member relations (Tom Lane)
- Prevent infinite loop in ProcessIncomingNotify() after unlistening (Jeff Davis)
- Prevent show_session_authorization() from crashing within autovacuum processes (Tom Lane)
- Re-allow input of Julian dates prior to 0001-01-01 AD (Tom Lane)

  Input such as `'J100000'::date` worked before 8.4, but was unintentionally broken by added error-checking.
- Make psql recognize `DISCARD ALL` as a command that should not be encased in a transaction block in autocommit-off mode (Itagaki Takahiro)
- Update build infrastructure and documentation to reflect the source code repository's move from CVS to Git (Magnus Hagander and others)

E.74. Release 9.0

Release date: 2010-09-20

E.74.1. Overview

This release of PostgreSQL adds features that have been requested for years, such as easy-to-use replication, a mass permission-changing facility, and anonymous code blocks. While past major releases have been conservative in their scope, this release shows a bold new desire to provide facilities that new and existing users of PostgreSQL will embrace. This has all been done with few incompatibilities. Major enhancements include:

- Built-in replication based on log shipping. This advance consists of two features: Streaming Replication, allowing continuous archive (WAL) files to be streamed over a network connection to a standby server, and Hot Standby, allowing continuous archive standby servers to execute read-only queries. The net effect is to support a single master with multiple read-only slave servers.
- Easier database object permissions management. `GRANT/REVOKE IN SCHEMA` supports mass permissions changes on existing objects, while `ALTER DEFAULT`

PRIVILEGES allows control of privileges for objects created in the future. Large objects (BLOBs) now support permissions management as well.

- Broadly enhanced stored procedure support. The DO statement supports ad-hoc or "anonymous" code blocks. Functions can now be called using named parameters. PL/pgSQL is now installed by default, and PL/Perl and PL/Python have been enhanced in several ways, including support for Python3.
- Full support for 64-bit Windows.
- More advanced reporting queries, including additional windowing options (PRECEDING and FOLLOWING) and the ability to control the order in which values are fed to aggregate functions.
- New trigger features, including SQL-standard-compliant per-column triggers and conditional trigger execution.
- Deferrable unique constraints. Mass updates to unique keys are now possible without trickery.
- Exclusion constraints. These provide a generalized version of unique constraints, allowing enforcement of complex conditions.
- New and enhanced security features, including RADIUS authentication, LDAP authentication improvements, and a new contrib module passwordcheck for testing password strength.
- New high-performance implementation of the LISTEN/NOTIFY feature. Pending events are now stored in a memory-based queue rather than a table. Also, a "payload" string can be sent with each event, rather than transmitting just an event name as before.
- New implementation of VACUUM FULL. This command now rewrites the entire table and indexes, rather than moving individual rows to compact space. It is substantially faster in most cases, and no longer results in index bloat.
- New contrib module pg_upgrade to support in-place upgrades from 8.3 or 8.4 to 9.0.
- Multiple performance enhancements for specific types of queries, including elimination of unnecessary joins. This helps optimize some automatically-generated queries, such as those produced by object-relational mappers (ORMs).
- EXPLAIN enhancements. The output is now available in JSON, XML, or YAML format, and includes buffer utilization and other data not previously available.
- hstore improvements, including new functions and greater data capacity.

The above items are explained in more detail in the sections below.

E.74.2. Migration to Version 9.0

A dump/restore using pg_dump, or use of pg_upgrade, is required for those wishing to migrate data from any previous release.

Version 9.0 contains a number of changes that selectively break backwards compatibility in order to support new features and code quality improvements. In particular, users who make extensive use of PL/pgSQL, Point-In-Time Recovery (PITR), or Warm Standby should test their applications because of slight user-visible changes in those areas. Observe the following incompatibilities:

E.74.2.1. Server Settings

- Remove server parameter `add_missing_from`, which was defaulted to off for many years (Tom Lane)
- Remove server parameter `regex_flavor`, which was defaulted to `advanced` for many years (Tom Lane)
- `archive_mode` now only affects `archive_command`; a new setting, `wal_level`, affects the contents of the write-ahead log (Heikki Linnakangas)
- `log_temp_files` now uses default file size units of kilobytes (Robert Haas)

E.74.2.2. Queries

- When querying a parent table, do not do any separate permission checks on child tables scanned as part of the query (Peter Eisentraut)

  The SQL standard specifies this behavior, and it is also much more convenient in practice than the former behavior of checking permissions on each child as well as the parent.

E.74.2.3. Data Types

- `bytea` output now appears in hex format by default (Peter Eisentraut)

  The server parameter `bytea_output` can be used to select the traditional output format if needed for compatibility.

- Array input now considers only plain ASCII whitespace characters to be potentially ignorable; it will never ignore non-ASCII characters, even if they are whitespace according to some locales (Tom Lane)

  This avoids some corner cases where array values could be interpreted differently depending on the server's locale settings.

- Improve standards compliance of `SIMILAR TO` patterns and SQL-style `substring( )` patterns (Tom Lane)

  This includes treating `?` and `{...}` as pattern metacharacters, while they were simple literal characters before; that corresponds to new features added in SQL:2008. Also, `^` and `$` are now treated as simple literal characters; formerly they were treated as metacharacters, as if the pattern were following POSIX rather than SQL rules. Also, in SQL-standard `substring()`, use of parentheses for nesting no longer interferes with capturing of a substring. Also, processing of bracket expressions (character classes) is now more standards-compliant.

- Reject negative length values in 3-parameter `substring()` for bit strings, per the SQL standard (Tom Lane)
- Make `date_trunc` truncate rather than round when reducing precision of fractional seconds (Tom Lane)

  The code always acted this way for integer-based dates/times. Now float-based dates/times behave similarly.

E.74.2.4. Object Renaming

- Tighten enforcement of column name consistency during `RENAME` when a child table inherits the same column from multiple unrelated parents (KaiGai Kohei)

- No longer automatically rename indexes and index columns when the underlying table columns are renamed (Tom Lane)

  Administrators can still rename such indexes and columns manually. This change will require an update of the JDBC driver, and possibly other drivers, so that unique indexes are correctly recognized after a rename.

- `CREATE OR REPLACE FUNCTION` can no longer change the declared names of function parameters (Pavel Stehule)

  In order to avoid creating ambiguity in named-parameter calls, it is no longer allowed to change the aliases for input parameters in the declaration of an existing function (although names can still be assigned to previously unnamed parameters). You now have to `DROP` and recreate the function to do that.

E.74.2.5. PL/pgSQL

- PL/pgSQL now throws an error if a variable name conflicts with a column name used in a query (Tom Lane)

  The former behavior was to bind ambiguous names to PL/pgSQL variables in preference to query columns, which often resulted in surprising misbehavior. Throwing an error allows easy detection of ambiguous situations. Although it's recommended that functions encountering this type of error be modified to remove the conflict, the old behavior can be restored if necessary via the configuration parameter `plpgsql.variable_conflict`, or via the per-function option `#variable_conflict`.

- PL/pgSQL no longer allows variable names that match certain SQL reserved words (Tom Lane)

  This is a consequence of aligning the PL/pgSQL parser to match the core SQL parser more closely. If necessary, variable names can be double-quoted to avoid this restriction.

- PL/pgSQL now requires columns of composite results to match the expected type modifier as well as base type (Pavel Stehule, Tom Lane)

  For example, if a column of the result type is declared as `NUMERIC(30,2)`, it is no longer acceptable to return a `NUMERIC` of some other precision in that column. Previous versions neglected to check the type modifier and would thus allow result rows that didn't actually conform to the declared restrictions.

- PL/pgSQL now treats selection into composite fields more consistently (Tom Lane)

  Formerly, a statement like `SELECT ... INTO rec.fld FROM ...` was treated as a scalar assignment even if the record field *`fld`* was of composite type. Now it is treated as a record assignment, the same as when the `INTO` target is a regular variable of composite type. So the values to be assigned to the field's subfields should be written as separate columns of the `SELECT` list, not as a `ROW(...)` construct as in previous versions.

  If you need to do this in a way that will work in both 9.0 and previous releases, you can write something like `rec.fld := ROW(...) FROM ...`.

- Remove PL/pgSQL's `RENAME` declaration (Tom Lane)

  Instead of `RENAME`, use `ALIAS`, which can now create an alias for any variable, not only dollar sign parameter names (such as `$1`) as before.

E.74.2.6. Other Incompatibilities

- Deprecate use of => as an operator name (Robert Haas)

  Future versions of PostgreSQL will probably reject this operator name entirely, in order to support the SQL-standard notation for named function parameters. For the moment, it is still allowed, but a warning is emitted when such an operator is defined.

- Remove support for platforms that don't have a working 64-bit integer data type (Tom Lane)

  It is believed all still-supported platforms have working 64-bit integer data types.

E.74.3. Changes

Version 9.0 has an unprecedented number of new major features, and over 200 enhancements, improvements, new commands, new functions, and other changes.

E.74.3.1. Server

E.74.3.1.1. Continuous Archiving and Streaming Replication

PostgreSQL's existing standby-server capability has been expanded both to support read-only queries on standby servers and to greatly reduce the lag between master and standby servers. For many users, this will be a useful and low-administration form of replication, either for high availability or for horizontal scalability.

- Allow a standby server to accept read-only queries (Simon Riggs, Heikki Linnakangas)

  This feature is called Hot Standby. There are new `postgresql.conf` and `recovery.conf` settings to control this feature, as well as extensive documentation.

- Allow write-ahead log (WAL) data to be streamed to a standby server (Fujii Masao, Heikki Linnakangas)

  This feature is called Streaming Replication. Previously WAL data could be sent to standby servers only in units of entire WAL files (normally 16 megabytes each). Streaming Replication eliminates this inefficiency and allows updates on the master to be propagated to standby servers with very little delay. There are new `postgresql.conf` and `recovery.conf` settings to control this feature, as well as extensive documentation.

- Add `pg_last_xlog_receive_location()` and `pg_last_xlog_replay_location()`, which can be used to monitor standby server WAL activity (Simon Riggs, Fujii Masao, Heikki Linnakangas)

E.74.3.1.2. Performance

- Allow per-tablespace values to be set for sequential and random page cost estimates (`seq_page_cost`/`random_page_cost`) via `ALTER TABLESPACE ... SET/RESET` (Robert Haas)

- Improve performance and reliability of EvalPlanQual rechecks in join queries (Tom Lane)

  `UPDATE`, `DELETE`, and `SELECT FOR UPDATE/SHARE` queries that involve joins will now behave much better when encountering freshly-updated rows.

- Improve performance of `TRUNCATE` when the table was created or truncated earlier in the same transaction (Tom Lane)
- Improve performance of finding inheritance child tables (Tom Lane)

E.74.3.1.3. Optimizer

- Remove unnecessary outer joins (Robert Haas)

  Outer joins where the inner side is unique and not referenced above the join are unnecessary and are therefore now removed. This will accelerate many automatically generated queries, such as those created by object-relational mappers (ORMs).

- Allow `IS NOT NULL` restrictions to use indexes (Tom Lane)

  This is particularly useful for finding `MAX()`/`MIN()` values in indexes that contain many null values.

- Improve the optimizer's choices about when to use materialize nodes, and when to use sorting versus hashing for `DISTINCT` (Tom Lane)
- Improve the optimizer's equivalence detection for expressions involving `boolean <>` operators (Tom Lane)

E.74.3.1.4. GEQO

- Use the same random seed every time GEQO plans a query (Andres Freund)

  While the Genetic Query Optimizer (GEQO) still selects random plans, it now always selects the same random plans for identical queries, thus giving more consistent performance. You can modify `geqo_seed` to experiment with alternative plans.

- Improve GEQO plan selection (Tom Lane)

  This avoids the rare error “failed to make a valid plan” , and should also improve planning speed.

E.74.3.1.5. Optimizer Statistics

- Improve `ANALYZE` to support inheritance-tree statistics (Tom Lane)

  This is particularly useful for partitioned tables. However, autovacuum does not yet automatically re-analyze parent tables when child tables change.

- Improve autovacuum's detection of when re-analyze is necessary (Tom Lane)
- Improve optimizer's estimation for greater/less-than comparisons (Tom Lane)

  When looking up statistics for greater/less-than comparisons, if the comparison value is in the first or last histogram bucket, use an index (if available) to fetch the current actual column minimum or maximum. This greatly improves the accuracy of estimates for comparison values near the ends of the data range, particularly if the range is constantly changing due to addition of new data.

- Allow setting of number-of-distinct-values statistics using `ALTER TABLE` (Robert Haas)

  This allows users to override the estimated number or percentage of distinct values for a column. This statistic is normally computed by `ANALYZE`, but the estimate can be poor, especially on tables with very large numbers of rows.

E.74.3.1.6. Authentication

- Add support for RADIUS (Remote Authentication Dial In User Service) authentication (Magnus Hagander)
- Allow LDAP (Lightweight Directory Access Protocol) authentication to operate in "search/bind" mode (Robert Fleming, Magnus Hagander)

  This allows the user to be looked up first, then the system uses the DN (Distinguished Name) returned for that user.
- Add `samehost` and `samenet` designations to `pg_hba.conf` (Stef Walter)

  These match the server's IP address and subnet address respectively.
- Pass trusted SSL root certificate names to the client so the client can return an appropriate client certificate (Craig Ringer)

E.74.3.1.7. Monitoring

- Add the ability for clients to set an application name, which is displayed in `pg_stat_activity` (Dave Page)

  This allows administrators to characterize database traffic and troubleshoot problems by source application.
- Add a SQLSTATE option (`%e`) to `log_line_prefix` (Guillaume Smet)

  This allows users to compile statistics on errors and messages by error code number.
- Write to the Windows event log in UTF16 encoding (Itagaki Takahiro)

  Now there is true multilingual support for PostgreSQL log messages on Windows.

E.74.3.1.8. Statistics Counters

- Add `pg_stat_reset_shared('bgwriter')` to reset the cluster-wide shared statistics for the background writer (Greg Smith)
- Add `pg_stat_reset_single_table_counters()` and `pg_stat_reset_single_function_counters()` to allow resetting the statistics counters for individual tables and functions (Magnus Hagander)

E.74.3.1.9. Server Settings

- Allow setting of configuration parameters based on database/role combinations (Alvaro Herrera)

  Previously only per-database and per-role settings were possible, not combinations. All role and database settings are now stored in the new `pg_db_role_setting` system catalog. A new psql command `\drds` shows these settings. The legacy system views `pg_roles`, `pg_shadow`, and `pg_user` do not show combination settings, and therefore no longer completely represent the configuration for a user or database.
- Add server parameter `bonjour`, which controls whether a Bonjour-enabled server advertises itself via Bonjour (Tom Lane)

  The default is off, meaning it does not advertise. This allows packagers to distribute Bonjour-enabled builds without worrying that individual users might not want the feature.

- Add server parameter `enable_material`, which controls the use of materialize nodes in the optimizer (Robert Haas)

  The default is on. When off, the optimizer will not add materialize nodes purely for performance reasons, though they will still be used when necessary for correctness.

- Change server parameter `log_temp_files` to use default file size units of kilobytes (Robert Haas)

  Previously this setting was interpreted in bytes if no units were specified.

- Log changes of parameter values when `postgresql.conf` is reloaded (Peter Eisentraut)

  This lets administrators and security staff audit changes of database settings, and is also very convenient for checking the effects of `postgresql.conf` edits.

- Properly enforce superuser permissions for custom server parameters (Tom Lane)

  Non-superusers can no longer issue `ALTER ROLE/DATABASE SET` for parameters that are not currently known to the server. This allows the server to correctly check that superuser-only parameters are only set by superusers. Previously, the `SET` would be allowed and then ignored at session start, making superuser-only custom parameters much less useful than they should be.

E.74.3.2. Queries

- Perform `SELECT FOR UPDATE/SHARE` processing after applying `LIMIT`, so the number of rows returned is always predictable (Tom Lane)

  Previously, changes made by concurrent transactions could cause a `SELECT FOR UPDATE` to unexpectedly return fewer rows than specified by its `LIMIT`. `FOR UPDATE` in combination with `ORDER BY` can still produce surprising results, but that can be corrected by placing `FOR UPDATE` in a subquery.

- Allow mixing of traditional and SQL-standard `LIMIT/OFFSET` syntax (Tom Lane)

- Extend the supported frame options in window functions (Hitoshi Harada)

  Frames can now start with `CURRENT ROW`, and the `ROWS` *`n`* `PRECEDING/FOLLOWING` options are now supported.

- Make `SELECT INTO` and `CREATE TABLE AS` return row counts to the client in their command tags (Boszormenyi Zoltan)

  This can save an entire round-trip to the client, allowing result counts and pagination to be calculated without an additional `COUNT` query.

E.74.3.2.1. Unicode Strings

- Support Unicode surrogate pairs (dual 16-bit representation) in `U&` strings and identifiers (Peter Eisentraut)

- Support Unicode escapes in `E'...'` strings (Marko Kreen)

E.74.3.3. Object Manipulation

- Speed up `CREATE DATABASE` by deferring flushes to disk (Andres Freund, Greg Stark)

- Allow comments on columns of tables, views, and composite types only, not other relation types such as indexes and TOAST tables (Tom Lane)

- Allow the creation of enumerated types containing no values (Bruce Momjian)
- Let values of columns having storage type `MAIN` remain on the main heap page unless the row cannot fit on a page (Kevin Grittner)

  Previously `MAIN` values were forced out to TOAST tables until the row size was less than one-quarter of the page size.

E.74.3.3.1. `ALTER TABLE`

- Implement `IF EXISTS` for `ALTER TABLE DROP COLUMN` and `ALTER TABLE DROP CONSTRAINT` (Andres Freund)
- Allow `ALTER TABLE` commands that rewrite tables to skip WAL logging (Itagaki Takahiro)

  Such operations either produce a new copy of the table or are rolled back, so WAL archiving can be skipped, unless running in continuous archiving mode. This reduces I/O overhead and improves performance.
- Fix failure of `ALTER TABLE` *`table`* `ADD COLUMN` *`col`* `serial` when done by non-owner of table (Tom Lane)

E.74.3.3.2. `CREATE TABLE`

- Add support for copying `COMMENTS` and `STORAGE` settings in `CREATE TABLE ... LIKE` commands (Itagaki Takahiro)
- Add a shortcut for copying all properties in `CREATE TABLE ... LIKE` commands (Itagaki Takahiro)
- Add the SQL-standard `CREATE TABLE ... OF` *`type`* command (Peter Eisentraut)

  This allows creation of a table that matches an existing composite type. Additional constraints and defaults can be specified in the command.

E.74.3.3.3. Constraints

- Add deferrable unique constraints (Dean Rasheed)

  This allows mass updates, such as `UPDATE tab SET col = col + 1`, to work reliably on columns that have unique indexes or are marked as primary keys. If the constraint is specified as `DEFERRABLE` it will be checked at the end of the statement, rather than after each row is updated. The constraint check can also be deferred until the end of the current transaction, allowing such updates to be spread over multiple SQL commands.
- Add exclusion constraints (Jeff Davis)

  Exclusion constraints generalize uniqueness constraints by allowing arbitrary comparison operators, not just equality. They are created with the `CREATE TABLE CONSTRAINT ... EXCLUDE` clause. The most common use of exclusion constraints is to specify that column entries must not overlap, rather than simply not be equal. This is useful for time periods and other ranges, as well as arrays. This feature enhances checking of data integrity for many calendaring, time-management, and scientific applications.
- Improve uniqueness-constraint violation error messages to report the values causing the failure (Itagaki Takahiro)

For example, a uniqueness constraint violation might now report `Key (x)=(2) already exists`.

E.74.3.3.4. Object Permissions

- Add the ability to make mass permission changes across a whole schema using the new `GRANT/REVOKE IN SCHEMA` clause (Petr Jelinek)

  This simplifies management of object permissions and makes it easier to utilize database roles for application data security.

- Add `ALTER DEFAULT PRIVILEGES` command to control privileges of objects created later (Petr Jelinek)

  This greatly simplifies the assignment of object privileges in a complex database application. Default privileges can be set for tables, views, sequences, and functions. Defaults may be assigned on a per-schema basis, or database-wide.

- Add the ability to control large object (BLOB) permissions with `GRANT/REVOKE` (KaiGai Kohei)

  Formerly, any database user could read or modify any large object. Read and write permissions can now be granted and revoked per large object, and the ownership of large objects is tracked.

E.74.3.4. Utility Operations

- Make `LISTEN/NOTIFY` store pending events in a memory queue, rather than in a system table (Joachim Wieland)

  This substantially improves performance, while retaining the existing features of transactional support and guaranteed delivery.

- Allow `NOTIFY` to pass an optional "payload" string to listeners (Joachim Wieland)

  This greatly improves the usefulness of `LISTEN/NOTIFY` as a general-purpose event queue system.

- Allow `CLUSTER` on all per-database system catalogs (Tom Lane)

  Shared catalogs still cannot be clustered.

E.74.3.4.1. `COPY`

- Accept `COPY ... CSV FORCE QUOTE *` (Itagaki Takahiro)

  Now `*` can be used as shorthand for "all columns" in the `FORCE QUOTE` clause.

- Add new `COPY` syntax that allows options to be specified inside parentheses (Robert Haas, Emmanuel Cecchet)

  This allows greater flexibility for future `COPY` options. The old syntax is still supported, but only for pre-existing options.

E.74.3.4.2. `EXPLAIN`

- Allow `EXPLAIN` to output in XML, JSON, or YAML format (Robert Haas, Greg Sabino Mullane)

The new output formats are easily machine-readable, supporting the development of new tools for analysis of `EXPLAIN` output.

- Add new `BUFFERS` option to report query buffer usage during `EXPLAIN ANALYZE` (Itagaki Takahiro)

  This allows better query profiling for individual queries. Buffer usage is no longer reported in the output for log_statement_stats and related settings.

- Add hash usage information to `EXPLAIN` output (Robert Haas)

- Add new `EXPLAIN` syntax that allows options to be specified inside parentheses (Robert Haas)

  This allows greater flexibility for future `EXPLAIN` options. The old syntax is still supported, but only for pre-existing options.

E.74.3.4.3. VACUUM

- Change `VACUUM FULL` to rewrite the entire table and rebuild its indexes, rather than moving individual rows around to compact space (Itagaki Takahiro, Tom Lane)

  The previous method was usually slower and caused index bloat. Note that the new method will use more disk space transiently during `VACUUM FULL`; potentially as much as twice the space normally occupied by the table and its indexes.

- Add new `VACUUM` syntax that allows options to be specified inside parentheses (Itagaki Takahiro)

  This allows greater flexibility for future `VACUUM` options. The old syntax is still supported, but only for pre-existing options.

E.74.3.4.4. Indexes

- Allow an index to be named automatically by omitting the index name in `CREATE INDEX` (Tom Lane)

- By default, multicolumn indexes are now named after all their columns; and index expression columns are now named based on their expressions (Tom Lane)

- Reindexing shared system catalogs is now fully transactional and crash-safe (Tom Lane)

  Formerly, reindexing a shared index was only allowed in standalone mode, and a crash during the operation could leave the index in worse condition than it was before.

- Add `point_ops` operator class for GiST (Teodor Sigaev)

  This feature permits GiST indexing of `point` columns. The index can be used for several types of queries such as *point* `<@` *polygon* (point is in polygon). This should make many PostGIS queries faster.

- Use red-black binary trees for GIN index creation (Teodor Sigaev)

  Red-black trees are self-balancing. This avoids slowdowns in cases where the input is in nonrandom order.

E.74.3.5. Data Types

- Allow `bytea` values to be written in hex notation (Peter Eisentraut)

The server parameter `bytea_output` controls whether hex or traditional format is used for `bytea` output. Libpq's `PQescapeByteaConn()` function automatically uses the hex format when connected to PostgreSQL 9.0 or newer servers. However, pre-9.0 libpq versions will not correctly process hex format from newer servers.

The new hex format will be directly compatible with more applications that use binary data, allowing them to store and retrieve it without extra conversion. It is also significantly faster to read and write than the traditional format.

- Allow server parameter extra_float_digits to be increased to `3` (Tom Lane)

  The previous maximum `extra_float_digits` setting was `2`. There are cases where 3 digits are needed to dump and restore `float4` values exactly. pg_dump will now use the setting of 3 when dumping from a server that allows it.

- Tighten input checking for `int2vector` values (Caleb Welton)

E.74.3.5.1. Full Text Search

- Add prefix support in `synonym` dictionaries (Teodor Sigaev)
- Add filtering dictionaries (Teodor Sigaev)

  Filtering dictionaries allow tokens to be modified then passed to subsequent dictionaries.

- Allow underscores in email-address tokens (Teodor Sigaev)
- Use more standards-compliant rules for parsing URL tokens (Tom Lane)

E.74.3.6. Functions

- Allow function calls to supply parameter names and match them to named parameters in the function definition (Pavel Stehule)

  For example, if a function is defined to take parameters `a` and `b`, it can be called with `func(a := 7, b := 12)` or `func(b := 12, a := 7)`.

- Support locale-specific regular expression processing with UTF-8 server encoding (Tom Lane)

  Locale-specific regular expression functionality includes case-insensitive matching and locale-specific character classes. Previously, these features worked correctly for non-ASCII characters only if the database used a single-byte server encoding (such as LATIN1). They will still misbehave in multi-byte encodings other than UTF-8.

- Add support for scientific notation in `to_char()` (`EEEE` specification) (Pavel Stehule, Brendan Jurd)
- Make `to_char()` honor `FM` (fill mode) in `Y`, `YY`, and `YYY` specifications (Bruce Momjian, Tom Lane)

  It was already honored by `YYYY`.

- Fix `to_char()` to output localized numeric and monetary strings in the correct encoding on Windows (Hiroshi Inoue, Itagaki Takahiro, Bruce Momjian)
- Correct calculations of “overlaps” and “contains” operations for polygons (Teodor Sigaev)

The polygon && (overlaps) operator formerly just checked to see if the two polygons' bounding boxes overlapped. It now does a more correct check. The polygon @> and <@ (contains/contained by) operators formerly checked to see if one polygon's vertexes were all contained in the other; this can wrongly report "true" for some non-convex polygons. Now they check that all line segments of one polygon are contained in the other.

E.74.3.6.1. Aggregates

- Allow aggregate functions to use ORDER BY (Andrew Gierth)

  For example, this is now supported: array_agg(a ORDER BY b). This is useful with aggregates for which the order of input values is significant, and eliminates the need to use a nonstandard subquery to determine the ordering.

- Multi-argument aggregate functions can now use DISTINCT (Andrew Gierth)

- Add the string_agg() aggregate function to combine values into a single string (Pavel Stehule)

- Aggregate functions that are called with DISTINCT are now passed NULL values if the aggregate transition function is not marked as STRICT (Andrew Gierth)

  For example, agg(DISTINCT x) might pass a NULL x value to agg(). This is more consistent with the behavior in non-DISTINCT cases.

E.74.3.6.2. Bit Strings

- Add get_bit() and set_bit() functions for bit strings, mirroring those for bytea (Leonardo F)

- Implement OVERLAY() (replace) for bit strings and bytea (Leonardo F)

E.74.3.6.3. Object Information Functions

- Add pg_table_size() and pg_indexes_size() to provide a more user-friendly interface to the pg_relation_size() function (Bernd Helmle)

- Add has_sequence_privilege() for sequence permission checking (Abhijit Menon-Sen)

- Update the information_schema views to conform to SQL:2008 (Peter Eisentraut)

- Make the information_schema views correctly display maximum octet lengths for char and varchar columns (Peter Eisentraut)

- Speed up information_schema privilege views (Joachim Wieland)

E.74.3.6.4. Function and Trigger Creation

- Support execution of anonymous code blocks using the DO statement (Petr Jelinek, Joshua Tolley, Hannu Valtonen)

  This allows execution of server-side code without the need to create and delete a temporary function definition. Code can be executed in any language for which the user has permissions to define a function.

- Implement SQL-standard-compliant per-column triggers (Itagaki Takahiro)

  Such triggers are fired only when the specified column(s) are affected by the query, e.g. appear in an UPDATE's SET list.

- Add the `WHEN` clause to `CREATE TRIGGER` to allow control over whether a trigger is fired (Itagaki Takahiro)

  While the same type of check can always be performed inside the trigger, doing it in an external `WHEN` clause can have performance benefits.

E.74.3.7. Server-Side Languages

- Add the `OR REPLACE` clause to `CREATE LANGUAGE` (Tom Lane)

  This is helpful to optionally install a language if it does not already exist, and is particularly helpful now that PL/pgSQL is installed by default.

E.74.3.7.1. PL/pgSQL Server-Side Language

- Install PL/pgSQL by default (Bruce Momjian)

  The language can still be removed from a particular database if the administrator has security or performance concerns about making it available.

- Improve handling of cases where PL/pgSQL variable names conflict with identifiers used in queries within a function (Tom Lane)

  The default behavior is now to throw an error when there is a conflict, so as to avoid surprising behaviors. This can be modified, via the configuration parameter `plpgsql.variable_conflict` or the per-function option `#variable_conflict`, to allow either the variable or the query-supplied column to be used. In any case PL/pgSQL will no longer attempt to substitute variables in places where they would not be syntactically valid.

- Make PL/pgSQL use the main lexer, rather than its own version (Tom Lane)

  This ensures accurate tracking of the main system's behavior for details such as string escaping. Some user-visible details, such as the set of keywords considered reserved in PL/pgSQL, have changed in consequence.

- Avoid throwing an unnecessary error for an invalid record reference (Tom Lane)

  An error is now thrown only if the reference is actually fetched, rather than whenever the enclosing expression is reached. For example, many people have tried to do this in triggers:

  ```
  if TG_OP = 'INSERT' and NEW.col1 = ... then
  ```

  This will now actually work as expected.

- Improve PL/pgSQL's ability to handle row types with dropped columns (Pavel Stehule)

- Allow input parameters to be assigned values within PL/pgSQL functions (Steve Prentice)

  Formerly, input parameters were treated as being declared `CONST`, so the function's code could not change their values. This restriction has been removed to simplify porting of functions from other DBMSes that do not impose the equivalent restriction. An input parameter now acts like a local variable initialized to the passed-in value.

- Improve error location reporting in PL/pgSQL (Tom Lane)

- Add *`count`* and `ALL` options to `MOVE FORWARD/BACKWARD` in PL/pgSQL (Pavel Stehule)

- Allow PL/pgSQL's `WHERE CURRENT OF` to use a cursor variable (Tom Lane)
- Allow PL/pgSQL's `OPEN cursor FOR EXECUTE` to use parameters (Pavel Stehule, Itagaki Takahiro)

  This is accomplished with a new `USING` clause.

E.74.3.7.2. PL/Perl Server-Side Language

- Add new PL/Perl functions: `quote_literal()`, `quote_nullable()`, `quote_ident()`, `encode_bytea()`, `decode_bytea()`, `looks_like_number()`, `encode_array_literal()`, `encode_array_constructor()` (Tim Bunce)
- Add server parameter `plperl.on_init` to specify a PL/Perl initialization function (Tim Bunce)

  `plperl.on_plperl_init` and `plperl.on_plperlu_init` are also available for initialization that is specific to the trusted or untrusted language respectively.
- Support `END` blocks in PL/Perl (Tim Bunce)

  `END` blocks do not currently allow database access.
- Allow `use strict` in PL/Perl (Tim Bunce)

  Perl `strict` checks can also be globally enabled with the new server parameter `plperl.use_strict`.
- Allow `require` in PL/Perl (Tim Bunce)

  This basically tests to see if the module is loaded, and if not, generates an error. It will not allow loading of modules that the administrator has not preloaded via the initialization parameters.
- Allow `use feature` in PL/Perl if Perl version 5.10 or later is used (Tim Bunce)
- Verify that PL/Perl return values are valid in the server encoding (Andrew Dunstan)

E.74.3.7.3. PL/Python Server-Side Language

- Add Unicode support in PL/Python (Peter Eisentraut)

  Strings are automatically converted from/to the server encoding as necessary.
- Improve `bytea` support in PL/Python (Caleb Welton)

  `Bytea` values passed into PL/Python are now represented as binary, rather than the PostgreSQL `bytea` text format. `Bytea` values containing null bytes are now also output properly from PL/Python. Passing of boolean, integer, and float values was also improved.
- Support arrays as parameters and return values in PL/Python (Peter Eisentraut)
- Improve mapping of SQL domains to Python types (Peter Eisentraut)
- Add Python 3 support to PL/Python (Peter Eisentraut)

  The new server-side language is called `plpython3u`. This cannot be used in the same session with the Python 2 server-side language.
- Improve error location and exception reporting in PL/Python (Peter Eisentraut)

E.74.3.8. Client Applications

- Add an `--analyze-only` option to `vacuumdb`, to analyze without vacuuming (Bruce Momjian)

E.74.3.8.1. psql

- Add support for quoting/escaping the values of psql variables as SQL strings or identifiers (Pavel Stehule, Robert Haas)

  For example, `:'var'` will produce the value of `var` quoted and properly escaped as a literal string, while `:"var"` will produce its value quoted and escaped as an identifier.

- Ignore a leading UTF-8-encoded Unicode byte-order marker in script files read by psql (Itagaki Takahiro)

  This is enabled when the client encoding is UTF-8. It improves compatibility with certain editors, mostly on Windows, that insist on inserting such markers.

- Fix `psql --file -` to properly honor `--single-transaction` (Bruce Momjian)

- Avoid overwriting of psql's command-line history when two psql sessions are run concurrently (Tom Lane)

- Improve psql's tab completion support (Itagaki Takahiro)

- Show `\timing` output when it is enabled, regardless of "quiet" mode (Peter Eisentraut)

E.74.3.8.1.1. psql Display

- Improve display of wrapped columns in psql (Roger Leigh)

  This behavior is now the default. The previous formatting is available by using `\pset linestyle old-ascii`.

- Allow psql to use fancy Unicode line-drawing characters via `\pset linestyle unicode` (Roger Leigh)

E.74.3.8.1.2. psql \d Commands

- Make `\d` show child tables that inherit from the specified parent (Damien Clochard)

  `\d` shows only the number of child tables, while `\d+` shows the names of all child tables.

- Show definitions of index columns in `\d index_name` (Khee Chin)

  The definition is useful for expression indexes.

- Show a view's defining query only in `\d+`, not in `\d` (Peter Eisentraut)

  Always including the query was deemed overly verbose.

E.74.3.8.2. pg_dump

- Make pg_dump/pg_restore `--clean` also remove large objects (Itagaki Takahiro)

- Fix pg_dump to properly dump large objects when `standard_conforming_strings` is enabled (Tom Lane)

The previous coding could fail when dumping to an archive file and then generating script output from pg_restore.

- pg_restore now emits large-object data in hex format when generating script output (Tom Lane)

  This could cause compatibility problems if the script is then loaded into a pre-9.0 server. To work around that, restore directly to the server, instead.

- Allow pg_dump to dump comments attached to columns of composite types (Taro Minowa (Higepon))

- Make pg_dump `--verbose` output the pg_dump and server versions in text output mode (Jim Cox, Tom Lane)

  These were already provided in custom output mode.

- pg_restore now complains if any command-line arguments remain after the switches and optional file name (Tom Lane)

  Previously, it silently ignored any such arguments.

E.74.3.8.3. pg_ctl

- Allow pg_ctl to be used safely to start the postmaster during a system reboot (Tom Lane)

  Previously, pg_ctl's parent process could have been mistakenly identified as a running postmaster based on a stale postmaster lock file, resulting in a transient failure to start the database.

- Give pg_ctl the ability to initialize the database (by invoking initdb) (Zdenek Kotala)

E.74.3.9. Development Tools

E.74.3.9.1. libpq

- Add new libpq functions `PQconnectdbParams()` and `PQconnectStartParams()` (Guillaume Lelarge)

  These functions are similar to `PQconnectdb()` and `PQconnectStart()` except that they accept a null-terminated array of connection options, rather than requiring all options to be provided in a single string.

- Add libpq functions `PQescapeLiteral()` and `PQescapeIdentifier()` (Robert Haas)

  These functions return appropriately quoted and escaped SQL string literals and identifiers. The caller is not required to pre-allocate the string result, as is required by `PQescapeStringConn()`.

- Add support for a per-user service file (`.pg_service.conf`), which is checked before the site-wide service file (Peter Eisentraut)

- Properly report an error if the specified libpq service cannot be found (Peter Eisentraut)

- Add TCP keepalive settings in libpq (Tollef Fog Heen, Fujii Masao, Robert Haas)

  Keepalive settings were already supported on the server end of TCP connections.

- Avoid extra system calls to block and unblock `SIGPIPE` in libpq, on platforms that offer alternative methods (Jeremy Kerr)
- When a `.pgpass`-supplied password fails, mention where the password came from in the error message (Bruce Momjian)
- Load all SSL certificates given in the client certificate file (Tom Lane)

  This improves support for indirectly-signed SSL certificates.

E.74.3.9.2. ecpg

- Add SQLDA (SQL Descriptor Area) support to ecpg (Boszormenyi Zoltan)
- Add the `DESCRIBE` [`OUTPUT`] statement to ecpg (Boszormenyi Zoltan)
- Add an ECPGtransactionStatus function to return the current transaction status (Bernd Helmle)
- Add the `string` data type in ecpg Informix-compatibility mode (Boszormenyi Zoltan)
- Allow ecpg to use `new` and `old` variable names without restriction (Michael Meskes)
- Allow ecpg to use variable names in `free()` (Michael Meskes)
- Make `ecpg_dynamic_type()` return zero for non-SQL3 data types (Michael Meskes)

  Previously it returned the negative of the data type OID. This could be confused with valid type OIDs, however.
- Support `long long` types on platforms that already have 64-bit `long` (Michael Meskes)

E.74.3.9.2.1. ecpg Cursors

- Add out-of-scope cursor support in ecpg's native mode (Boszormenyi Zoltan)

  This allows `DECLARE` to use variables that are not in scope when `OPEN` is called. This facility already existed in ecpg's Informix-compatibility mode.
- Allow dynamic cursor names in ecpg (Boszormenyi Zoltan)
- Allow ecpg to use noise words `FROM` and `IN` in `FETCH` and `MOVE` (Boszormenyi Zoltan)

E.74.3.10. Build Options

- Enable client thread safety by default (Bruce Momjian)

  The thread-safety option can be disabled with `configure --disable-thread-safety`.
- Add support for controlling the Linux out-of-memory killer (Alex Hunsaker, Tom Lane)

  Now that `/proc/self/oom_adj` allows disabling of the Linux out-of-memory (OOM) killer, it's recommendable to disable OOM kills for the postmaster. It may then be desirable to re-enable OOM kills for the postmaster's child processes. The new compile-time option `LINUX_OOM_ADJ` allows the killer to be reactivated for child processes.

E.74.3.10.1. Makefiles

- New `Makefile` targets `world`, `install-world`, and `installcheck-world` (Andrew Dunstan)

  These are similar to the existing `all`, `install`, and `installcheck` targets, but they also build the HTML documentation, build and test `contrib`, and test server-side languages and ecpg.

- Add data and documentation installation location control to PGXS Makefiles (Mark Cave-Ayland)

- Add Makefile rules to build the PostgreSQL documentation as a single HTML file or as a single plain-text file (Peter Eisentraut, Bruce Momjian)

E.74.3.10.2. Windows

- Support compiling on 64-bit Windows and running in 64-bit mode (Tsutomu Yamada, Magnus Hagander)

  This allows for large shared memory sizes on Windows.

- Support server builds using Visual Studio 2008 (Magnus Hagander)

E.74.3.11. Source Code

- Distribute prebuilt documentation in a subdirectory tree, rather than as tar archive files inside the distribution tarball (Peter Eisentraut)

  For example, the prebuilt HTML documentation is now in `doc/src/sgml/html/`; the manual pages are packaged similarly.

- Make the server's lexer reentrant (Tom Lane)

  This was needed for use of the lexer by PL/pgSQL.

- Improve speed of memory allocation (Tom Lane, Greg Stark)

- User-defined constraint triggers now have entries in `pg_constraint` as well as `pg_trigger` (Tom Lane)

  Because of this change, `pg_constraint.pgconstrname` is now redundant and has been removed.

- Add system catalog columns `pg_constraint.conindid` and `pg_trigger.tgconstrindid` to better document the use of indexes for constraint enforcement (Tom Lane)

- Allow multiple conditions to be communicated to backends using a single operating system signal (Fujii Masao)

  This allows new features to be added without a platform-specific constraint on the number of signal conditions.

- Improve source code test coverage, including `contrib`, PL/Python, and PL/Perl (Peter Eisentraut, Andrew Dunstan)

- Remove the use of flat files for system table bootstrapping (Tom Lane, Alvaro Herrera)

  This improves performance when using many roles or databases, and eliminates some possible failure conditions.

- Automatically generate the initial contents of `pg_attribute` for "bootstrapped" catalogs (John Naylor)

  This greatly simplifies changes to these catalogs.

- Split the processing of `INSERT/UPDATE/DELETE` operations out of `execMain.c` (Marko Tiikkaja)

  Updates are now executed in a separate ModifyTable node. This change is necessary infrastructure for future improvements.

- Simplify translation of psql's SQL help text (Peter Eisentraut)

- Reduce the lengths of some file names so that all file paths in the distribution tarball are less than 100 characters (Tom Lane)

  Some decompression programs have problems with longer file paths.

- Add a new `ERRCODE_INVALID_PASSWORD` `SQLSTATE` error code (Bruce Momjian)

- With authors' permissions, remove the few remaining personal source code copyright notices (Bruce Momjian)

  The personal copyright notices were insignificant but the community occasionally had to answer questions about them.

- Add new documentation section about running PostgreSQL in non-durable mode to improve performance (Bruce Momjian)

- Restructure the HTML documentation `Makefile` rules to make their dependency checks work correctly, avoiding unnecessary rebuilds (Peter Eisentraut)

- Use DocBook XSL stylesheets for man page building, rather than Docbook2X (Peter Eisentraut)

  This changes the set of tools needed to build the man pages.

- Improve PL/Perl code structure (Tim Bunce)

- Improve error context reports in PL/Perl (Alexey Klyukin)

E.74.3.11.1. New Build Requirements

Note that these requirements do not apply when building from a distribution tarball, since tarballs include the files that these programs are used to build.

- Require Autoconf 2.63 to build configure (Peter Eisentraut)

- Require Flex 2.5.31 or later to build from a CVS checkout (Tom Lane)

- Require Perl version 5.8 or later to build from a CVS checkout (John Naylor, Andrew Dunstan)

E.74.3.11.2. Portability

- Use a more modern API for Bonjour (Tom Lane)

  Bonjour support now requires OS X 10.3 or later. The older API has been deprecated by Apple.

- Add spinlock support for the SuperH architecture (Nobuhiro Iwamatsu)

- Allow non-GCC compilers to use inline functions if they support them (Kurt Harriman)

- Remove support for platforms that don't have a working 64-bit integer data type (Tom Lane)
- Restructure use of `LDFLAGS` to be more consistent across platforms (Tom Lane)

  `LDFLAGS` is now used for linking both executables and shared libraries, and we add on `LDFLAGS_EX` when linking executables, or `LDFLAGS_SL` when linking shared libraries.

E.74.3.11.3. Server Programming

- Make backend header files safe to include in C++ (Kurt Harriman, Peter Eisentraut)

  These changes remove keyword conflicts that previously made C++ usage difficult in backend code. However, there are still other complexities when using C++ for backend functions. `extern "C" { }` is still necessary in appropriate places, and memory management and error handling are still problematic.
- Add `AggCheckCallContext()` for use in detecting if a C function is being called as an aggregate (Hitoshi Harada)
- Change calling convention for `SearchSysCache()` and related functions to avoid hard-wiring the maximum number of cache keys (Robert Haas)

  Existing calls will still work for the moment, but can be expected to break in 9.1 or later if not converted to the new style.
- Require calls of `fastgetattr()` and `heap_getattr()` backend macros to provide a non-NULL fourth argument (Robert Haas)
- Custom typanalyze functions should no longer rely on `VacAttrStats.attr` to determine the type of data they will be passed (Tom Lane)

  This was changed to allow collection of statistics on index columns for which the storage type is different from the underlying column data type. There are new fields that tell the actual datatype being analyzed.

E.74.3.11.4. Server Hooks

- Add parser hooks for processing ColumnRef and ParamRef nodes (Tom Lane)
- Add a ProcessUtility hook so loadable modules can control utility commands (Itagaki Takahiro)

E.74.3.11.5. Binary Upgrade Support

- Add `contrib/pg_upgrade` to support in-place upgrades (Bruce Momjian)

  This avoids the requirement of dumping/reloading the database when upgrading to a new major release of PostgreSQL, thus reducing downtime by orders of magnitude. It supports upgrades to 9.0 from PostgreSQL 8.3 and 8.4.
- Add support for preserving relation `relfilenode` values during binary upgrades (Bruce Momjian)
- Add support for preserving `pg_type` and `pg_enum` OIDs during binary upgrades (Bruce Momjian)
- Move data files within tablespaces into PostgreSQL-version-specific subdirectories (Bruce Momjian)

This simplifies binary upgrades.

E.74.3.12. Contrib

- Add multithreading option (`-j`) to `contrib/pgbench` (Itagaki Takahiro)

  This allows multiple CPUs to be used by pgbench, reducing the risk of pgbench itself becoming the test bottleneck.

- Add `\shell` and `\setshell` meta commands to `contrib/pgbench` (Michael Paquier)

- New features for `contrib/dict_xsyn` (Sergey Karpov)

  The new options are `matchorig`, `matchsynonyms`, and `keepsynonyms`.

- Add full text dictionary `contrib/unaccent` (Teodor Sigaev)

  This filtering dictionary removes accents from letters, which makes full-text searches over multiple languages much easier.

- Add `dblink_get_notify()` to `contrib/dblink` (Marcus Kempe)

  This allows asynchronous notifications in dblink.

- Improve `contrib/dblink`'s handling of dropped columns (Tom Lane)

  This affects `dblink_build_sql_insert()` and related functions. These functions now number columns according to logical not physical column numbers.

- Greatly increase `contrib/hstore`'s data length limit, and add B-tree and hash support so `GROUP BY` and `DISTINCT` operations are possible on `hstore` columns (Andrew Gierth)

  New functions and operators were also added. These improvements make `hstore` a full-function key-value store embedded in PostgreSQL.

- Add `contrib/passwordcheck` to support site-specific password strength policies (Laurenz Albe)

  The source code of this module should be modified to implement site-specific password policies.

- Add `contrib/pg_archivecleanup` tool (Simon Riggs)

  This is designed to be used in the `archive_cleanup_command` server parameter, to remove no-longer-needed archive files.

- Add query text to `contrib/auto_explain` output (Andrew Dunstan)

- Add buffer access counters to `contrib/pg_stat_statements` (Itagaki Takahiro)

- Update `contrib/start-scripts/linux` to use `/proc/self/oom_adj` to disable the Linux out-of-memory (OOM) killer (Alex Hunsaker, Tom Lane)

E.75. Release 8.4.22

Release date: 2014-07-24

This release contains a variety of fixes from 8.4.21. For information about new features in the 8.4 major release, see 第 E.97 节.

This is expected to be the last PostgreSQL release in the 8.4.X series. Users are encouraged to update to a newer release branch soon.

E.75.1. Migration to Version 8.4.22

A dump/restore is not required for those running 8.4.X.

However, this release corrects an index corruption problem in some GiST indexes. See the first changelog entry below to find out whether your installation has been affected and what steps you should take if so.

Also, if you are upgrading from a version earlier than 8.4.19, see 第 E.78 节.

E.75.2. Changes

- Correctly initialize padding bytes in `contrib/btree_gist` indexes on `bit` columns (Heikki Linnakangas)

  This error could result in incorrect query results due to values that should compare equal not being seen as equal. Users with GiST indexes on `bit` or `bit varying` columns should `REINDEX` those indexes after installing this update.

- Protect against torn pages when deleting GIN list pages (Heikki Linnakangas)

  This fix prevents possible index corruption if a system crash occurs while the page update is being written to disk.

- Fix possibly-incorrect cache invalidation during nested calls to `ReceiveSharedInvalidMessages` (Andres Freund)

- Don't assume a subquery's output is unique if there's a set-returning function in its targetlist (David Rowley)

  This oversight could lead to misoptimization of constructs like `WHERE x IN (SELECT y, generate_series(1,10) FROM t GROUP BY y)`.

- Fix failure to detoast fields in composite elements of structured types (Tom Lane)

  This corrects cases where TOAST pointers could be copied into other tables without being dereferenced. If the original data is later deleted, it would lead to errors like "missing chunk number 0 for toast value ..." when the now-dangling pointer is used.

- Fix "record type has not been registered" failures with whole-row references to the output of Append plan nodes (Tom Lane)

- Fix possible crash when invoking a user-defined function while rewinding a cursor (Tom Lane)

- Fix query-lifespan memory leak while evaluating the arguments for a function in `FROM` (Tom Lane)

- Fix session-lifespan memory leaks in regular-expression processing (Tom Lane, Arthur O'Dwyer, Greg Stark)

- Fix data encoding error in `hungarian.stop` (Tom Lane)

- Fix liveness checks for rows that were inserted in the current transaction and then deleted by a now-rolled-back subtransaction (Andres Freund)

  This could cause problems (at least spurious warnings, and at worst an infinite loop) if `CREATE INDEX` or `CLUSTER` were done later in the same transaction.

- Clear `pg_stat_activity.xact_start` during `PREPARE TRANSACTION` (Andres Freund)

  After the `PREPARE`, the originating session is no longer in a transaction, so it should not continue to display a transaction start time.

- Fix `REASSIGN OWNED` to not fail for text search objects (Álvaro Herrera)

- Block signals during postmaster startup (Tom Lane)

  This ensures that the postmaster will properly clean up after itself if, for example, it receives SIGINT while still starting up.

- Secure Unix-domain sockets of temporary postmasters started during `make check` (Noah Misch)

  Any local user able to access the socket file could connect as the server's bootstrap superuser, then proceed to execute arbitrary code as the operating-system user running the test, as we previously noted in CVE-2014-0067. This change defends against that risk by placing the server's socket in a temporary, mode 0700 subdirectory of `/tmp`. The hazard remains however on platforms where Unix sockets are not supported, notably Windows, because then the temporary postmaster must accept local TCP connections.

  A useful side effect of this change is to simplify `make check` testing in builds that override `DEFAULT_PGSOCKET_DIR`. Popular non-default values like `/var/run/postgresql` are often not writable by the build user, requiring workarounds that will no longer be necessary.

- On Windows, allow new sessions to absorb values of PGC_BACKEND parameters (such as log_connections) from the configuration file (Amit Kapila)

  Previously, if such a parameter were changed in the file post-startup, the change would have no effect.

- Properly quote executable path names on Windows (Nikhil Deshpande)

  This oversight could cause initdb and pg_upgrade to fail on Windows, if the installation path contained both spaces and `@` signs.

- Fix linking of libpython on OS X (Tom Lane)

  The method we previously used can fail with the Python library supplied by Xcode 5.0 and later.

- Avoid buffer bloat in libpq when the server consistently sends data faster than the client can absorb it (Shin-ichi Morita, Tom Lane)

  libpq could be coerced into enlarging its input buffer until it runs out of memory (which would be reported misleadingly as "lost synchronization with server"). Under ordinary circumstances it's quite far-fetched that data could be continuously transmitted more quickly than the `recv()` loop can absorb it, but this has been observed when the client is artificially slowed by scheduler constraints.

- Ensure that LDAP lookup attempts in libpq time out as intended (Laurenz Albe)

- Fix pg_restore's processing of old-style large object comments (Tom Lane)

  A direct-to-database restore from an archive file generated by a pre-9.0 version of pg_dump would usually fail if the archive contained more than a few comments for large objects.

- In `contrib/pgcrypto` functions, ensure sensitive information is cleared from stack variables before returning (Marko Kreen)

- In `contrib/uuid-ossp`, cache the state of the OSSP UUID library across calls (Tom Lane)

  This improves the efficiency of UUID generation and reduces the amount of entropy drawn from `/dev/urandom`, on platforms that have that.

- Update time zone data files to tzdata release 2014e for DST law changes in Crimea, Egypt, and Morocco.

E.76. Release 8.4.21

Release date: 2014-03-20

This release contains a variety of fixes from 8.4.20. For information about new features in the 8.4 major release, see 第 E.97 节.

The PostgreSQL community will stop releasing updates for the 8.4.X release series in July 2014. Users are encouraged to update to a newer release branch soon.

E.76.1. Migration to Version 8.4.21

A dump/restore is not required for those running 8.4.X.

However, if you are upgrading from a version earlier than 8.4.19, see 第 E.78 节.

E.76.2. Changes

- Restore GIN metapages unconditionally to avoid torn-page risk (Heikki Linnakangas)

  Although this oversight could theoretically result in a corrupted index, it is unlikely to have caused any problems in practice, since the active part of a GIN metapage is smaller than a standard 512-byte disk sector.

- Allow regular-expression operators to be terminated early by query cancel requests (Tom Lane)

  This prevents scenarios wherein a pathological regular expression could lock up a server process uninterruptibly for a long time.

- Remove incorrect code that tried to allow `OVERLAPS` with single-element row arguments (Joshua Yanovski)

  This code never worked correctly, and since the case is neither specified by the SQL standard nor documented, it seemed better to remove it than fix it.

- Avoid getting more than `AccessShareLock` when de-parsing a rule or view (Dean Rasheed)

  This oversight resulted in pg_dump unexpectedly acquiring `RowExclusiveLock` locks on tables mentioned as the targets of `INSERT`/`UPDATE`/`DELETE` commands in

rules. While usually harmless, that could interfere with concurrent transactions that tried to acquire, for example, `ShareLock` on those tables.

- Prevent interrupts while reporting non-`ERROR` messages (Tom Lane)

  This guards against rare server-process freezeups due to recursive entry to `syslog()`, and perhaps other related problems.

- Update time zone data files to tzdata release 2014a for DST law changes in Fiji and Turkey, plus historical changes in Israel and Ukraine.

E.77. Release 8.4.20

Release date: 2014-02-20

This release contains a variety of fixes from 8.4.19. For information about new features in the 8.4 major release, see 第 E.97 节.

The PostgreSQL community will stop releasing updates for the 8.4.X release series in July 2014. Users are encouraged to update to a newer release branch soon.

E.77.1. Migration to Version 8.4.20

A dump/restore is not required for those running 8.4.X.

However, if you are upgrading from a version earlier than 8.4.19, see 第 E.78 节.

E.77.2. Changes

- Shore up `GRANT ... WITH ADMIN OPTION` restrictions (Noah Misch)

  Granting a role without `ADMIN OPTION` is supposed to prevent the grantee from adding or removing members from the granted role, but this restriction was easily bypassed by doing `SET ROLE` first. The security impact is mostly that a role member can revoke the access of others, contrary to the wishes of his grantor. Unapproved role member additions are a lesser concern, since an uncooperative role member could provide most of his rights to others anyway by creating views or `SECURITY DEFINER` functions. (CVE-2014-0060)

- Prevent privilege escalation via manual calls to PL validator functions (Andres Freund)

  The primary role of PL validator functions is to be called implicitly during `CREATE FUNCTION`, but they are also normal SQL functions that a user can call explicitly. Calling a validator on a function actually written in some other language was not checked for and could be exploited for privilege-escalation purposes. The fix involves adding a call to a privilege-checking function in each validator function. Non-core procedural languages will also need to make this change to their own validator functions, if any. (CVE-2014-0061)

- Avoid multiple name lookups during table and index DDL (Robert Haas, Andres Freund)

  If the name lookups come to different conclusions due to concurrent activity, we might perform some parts of the DDL on a different table than other parts. At least in the case of `CREATE INDEX`, this can be used to cause the permissions checks to be performed against a different table than the index creation, allowing for a privilege escalation attack. (CVE-2014-0062)

- Prevent buffer overrun with long datetime strings (Noah Misch)

  The `MAXDATELEN` constant was too small for the longest possible value of type `interval`, allowing a buffer overrun in `interval_out()`. Although the datetime input functions were more careful about avoiding buffer overrun, the limit was short enough to cause them to reject some valid inputs, such as input containing a very long timezone name. The ecpg library contained these vulnerabilities along with some of its own. (CVE-2014-0063)

- Prevent buffer overrun due to integer overflow in size calculations (Noah Misch, Heikki Linnakangas)

  Several functions, mostly type input functions, calculated an allocation size without checking for overflow. If overflow did occur, a too-small buffer would be allocated and then written past. (CVE-2014-0064)

- Prevent overruns of fixed-size buffers (Peter Eisentraut, Jozef Mlich)

  Use `strlcpy()` and related functions to provide a clear guarantee that fixed-size buffers are not overrun. Unlike the preceding items, it is unclear whether these cases really represent live issues, since in most cases there appear to be previous constraints on the size of the input string. Nonetheless it seems prudent to silence all Coverity warnings of this type. (CVE-2014-0065)

- Avoid crashing if `crypt()` returns NULL (Honza Horak, Bruce Momjian)

  There are relatively few scenarios in which `crypt()` could return NULL, but `contrib/chkpass` would crash if it did. One practical case in which this could be an issue is if libc is configured to refuse to execute unapproved hashing algorithms (e.g., “FIPS mode”). (CVE-2014-0066)

- Document risks of `make check` in the regression testing instructions (Noah Misch, Tom Lane)

  Since the temporary server started by `make check` uses “trust” authentication, another user on the same machine could connect to it as database superuser, and then potentially exploit the privileges of the operating-system user who started the tests. A future release will probably incorporate changes in the testing procedure to prevent this risk, but some public discussion is needed first. So for the moment, just warn people against using `make check` when there are untrusted users on the same machine. (CVE-2014-0067)

- Fix possible mis-replay of WAL records when some segments of a relation aren't full size (Greg Stark, Tom Lane)

  The WAL update could be applied to the wrong page, potentially many pages past where it should have been. Aside from corrupting data, this error has been observed to result in significant “bloat” of standby servers compared to their masters, due to updates being applied far beyond where the end-of-file should have been. This failure mode does not appear to be a significant risk during crash recovery, only when initially synchronizing a standby created from a base backup taken from a quickly-changing master.

- Ensure that insertions into non-leaf GIN index pages write a full-page WAL record when appropriate (Heikki Linnakangas)

  The previous coding risked index corruption in the event of a partial-page write during a system crash.

- Fix race conditions during server process exit (Robert Haas)

Ensure that signal handlers don't attempt to use the process's `MyProc` pointer after it's no longer valid.

- Fix unsafe references to `errno` within error reporting logic (Christian Kruse)

  This would typically lead to odd behaviors such as missing or inappropriate `HINT` fields.

- Fix possible crashes from using `ereport()` too early during server startup (Tom Lane)

  The principal case we've seen in the field is a crash if the server is started in a directory it doesn't have permission to read.

- Clear retry flags properly in OpenSSL socket write function (Alexander Kukushkin)

  This omission could result in a server lockup after unexpected loss of an SSL-encrypted connection.

- Fix length checking for Unicode identifiers (`U&"..."` syntax) containing escapes (Tom Lane)

  A spurious truncation warning would be printed for such identifiers if the escaped form of the identifier was too long, but the identifier actually didn't need truncation after de-escaping.

- Fix possible crash due to invalid plan for nested sub-selects, such as `WHERE (... x IN (SELECT ...) ...) IN (SELECT ...)` (Tom Lane)

- Ensure that `ANALYZE` creates statistics for a table column even when all the values in it are "too wide" (Tom Lane)

  `ANALYZE` intentionally omits very wide values from its histogram and most-common-values calculations, but it neglected to do something sane in the case that all the sampled entries are too wide.

- In `ALTER TABLE ... SET TABLESPACE`, allow the database's default tablespace to be used without a permissions check (Stephen Frost)

  `CREATE TABLE` has always allowed such usage, but `ALTER TABLE` didn't get the memo.

- Fix "cannot accept a set" error when some arms of a `CASE` return a set and others don't (Tom Lane)

- Fix checks for all-zero client addresses in pgstat functions (Kevin Grittner)

- Fix possible misclassification of multibyte characters by the text search parser (Tom Lane)

  Non-ASCII characters could be misclassified when using C locale with a multibyte encoding. On Cygwin, non-C locales could fail as well.

- Fix possible misbehavior in `plainto_tsquery()` (Heikki Linnakangas)

  Use `memmove()` not `memcpy()` for copying overlapping memory regions. There have been no field reports of this actually causing trouble, but it's certainly risky.

- Accept `SHIFT_JIS` as an encoding name for locale checking purposes (Tatsuo Ishii)

- Fix misbehavior of `PQhost()` on Windows (Fujii Masao)

It should return `localhost` if no host has been specified.

- Improve error handling in libpq and psql for failures during `COPY TO STDOUT/FROM STDIN` (Tom Lane)

  In particular this fixes an infinite loop that could occur in 9.2 and up if the server connection was lost during `COPY FROM STDIN`. Variants of that scenario might be possible in older versions, or with other client applications.

- Fix misaligned descriptors in ecpg (MauMau)

- In ecpg, handle lack of a hostname in the connection parameters properly (Michael Meskes)

- Fix performance regression in `contrib/dblink` connection startup (Joe Conway)

  Avoid an unnecessary round trip when client and server encodings match.

- In `contrib/isn`, fix incorrect calculation of the check digit for ISMN values (Fabien Coelho)

- Ensure client-code-only installation procedure works as documented (Peter Eisentraut)

- In Mingw and Cygwin builds, install the libpq DLL in the `bin` directory (Andrew Dunstan)

  This duplicates what the MSVC build has long done. It should fix problems with programs like psql failing to start because they can't find the DLL.

- Don't generate plain-text `HISTORY` and `src/test/regress/README` files anymore (Tom Lane)

  These text files duplicated the main HTML and PDF documentation formats. The trouble involved in maintaining them greatly outweighs the likely audience for plain-text format. Distribution tarballs will still contain files by these names, but they'll just be stubs directing the reader to consult the main documentation. The plain-text `INSTALL` file will still be maintained, as there is arguably a use-case for that.

- Update time zone data files to tzdata release 2013i for DST law changes in Jordan and historical changes in Cuba.

  In addition, the zones `Asia/Riyadh87`, `Asia/Riyadh88`, and `Asia/Riyadh89` have been removed, as they are no longer maintained by IANA, and never represented actual civil timekeeping practice.

E.78. Release 8.4.19

Release date: 2013-12-05

This release contains a variety of fixes from 8.4.18. For information about new features in the 8.4 major release, see 第 E.97 节.

E.78.1. Migration to Version 8.4.19

A dump/restore is not required for those running 8.4.X.

However, this release corrects a potential data corruption issue. See the first changelog entry below to find out whether your installation has been affected and what steps you can take if so.

Also, if you are upgrading from a version earlier than 8.4.17, see 第 E.80 节.

E.78.2. Changes

- Fix `VACUUM`'s tests to see whether it can update `relfrozenxid` (Andres Freund)

  In some cases `VACUUM` (either manual or autovacuum) could incorrectly advance a table's `relfrozenxid` value, allowing tuples to escape freezing, causing those rows to become invisible once 2^31 transactions have elapsed. The probability of data loss is fairly low since multiple incorrect advancements would need to happen before actual loss occurs, but it's not zero. Users upgrading from release 8.4.8 or earlier are not affected, but all later versions contain the bug.

  The issue can be ameliorated by, after upgrading, vacuuming all tables in all databases while having `vacuum_freeze_table_age` set to zero. This will fix any latent corruption but will not be able to fix all pre-existing data errors. However, an installation can be presumed safe after performing this vacuuming if it has executed fewer than 2^31 update transactions in its lifetime (check this with `SELECT txid_current() < 2^31`).

- Fix race condition in GIN index posting tree page deletion (Heikki Linnakangas)

  This could lead to transient wrong answers or query failures.

- Avoid flattening a subquery whose `SELECT` list contains a volatile function wrapped inside a sub-`SELECT` (Tom Lane)

  This avoids unexpected results due to extra evaluations of the volatile function.

- Fix planner's processing of non-simple-variable subquery outputs nested within outer joins (Tom Lane)

  This error could lead to incorrect plans for queries involving multiple levels of subqueries within `JOIN` syntax.

- Fix premature deletion of temporary files (Andres Freund)

- Fix possible read past end of memory in rule printing (Peter Eisentraut)

- Fix array slicing of `int2vector` and `oidvector` values (Tom Lane)

  Expressions of this kind are now implicitly promoted to regular `int2` or `oid` arrays.

- Fix incorrect behaviors when using a SQL-standard, simple GMT offset timezone (Tom Lane)

  In some cases, the system would use the simple GMT offset value when it should have used the regular timezone setting that had prevailed before the simple offset was selected. This change also causes the `timeofday` function to honor the simple GMT offset zone.

- Prevent possible misbehavior when logging translations of Windows error codes (Tom Lane)

- Properly quote generated command lines in pg_ctl (Naoya Anzai and Tom Lane)

  This fix applies only to Windows.

- Fix pg_dumpall to work when a source database sets `default_transaction_read_only` via `ALTER DATABASE SET` (Kevin Grittner)

Previously, the generated script would fail during restore.

- Fix ecpg's processing of lists of variables declared `varchar` (Zoltán Böszörményi)
- Make `contrib/lo` defend against incorrect trigger definitions (Marc Cousin)
- Update time zone data files to tzdata release 2013h for DST law changes in Argentina, Brazil, Jordan, Libya, Liechtenstein, Morocco, and Palestine. Also, new timezone abbreviations WIB, WIT, WITA for Indonesia.

E.79. Release 8.4.18

Release date: 2013-10-10

This release contains a variety of fixes from 8.4.17. For information about new features in the 8.4 major release, see 第 E.97 节.

E.79.1. Migration to Version 8.4.18

A dump/restore is not required for those running 8.4.X.

However, if you are upgrading from a version earlier than 8.4.17, see 第 E.80 节.

E.79.2. Changes

- Prevent corruption of multi-byte characters when attempting to case-fold identifiers (Andrew Dunstan)

  PostgreSQL case-folds non-ASCII characters only when using a single-byte server encoding.

- Fix memory leak caused by `lo_open()` failure (Heikki Linnakangas)
- Fix memory overcommit bug when `work_mem` is using more than 24GB of memory (Stephen Frost)
- Fix deadlock bug in libpq when using SSL (Stephen Frost)
- Properly compute row estimates for boolean columns containing many NULL values (Andrew Gierth)

  Previously tests like `col IS NOT TRUE` and `col IS NOT FALSE` did not properly factor in NULL values when estimating plan costs.

- Prevent pushing down `WHERE` clauses into unsafe `UNION/INTERSECT` subqueries (Tom Lane)

  Subqueries of a `UNION` or `INTERSECT` that contain set-returning functions or volatile functions in their `SELECT` lists could be improperly optimized, leading to run-time errors or incorrect query results.

- Fix rare case of "failed to locate grouping columns" planner failure (Tom Lane)
- Improve view dumping code's handling of dropped columns in referenced tables (Tom Lane)
- Fix possible deadlock during concurrent `CREATE INDEX CONCURRENTLY` operations (Tom Lane)
- Fix `regexp_matches()` handling of zero-length matches (Jeevan Chalke)

Previously, zero-length matches like '^' could return too many matches.

- Fix crash for overly-complex regular expressions (Heikki Linnakangas)
- Fix regular expression match failures for back references combined with non-greedy quantifiers (Jeevan Chalke)
- Prevent `CREATE FUNCTION` from checking `SET` variables unless function body checking is enabled (Tom Lane)
- Fix `pgp_pub_decrypt()` so it works for secret keys with passwords (Marko Kreen)
- Remove rare inaccurate warning during vacuum of index-less tables (Heikki Linnakangas)
- Avoid possible failure when performing transaction control commands (e.g `ROLLBACK`) in prepared queries (Tom Lane)
- Ensure that floating-point data input accepts standard spellings of "infinity" on all platforms (Tom Lane)

  The C99 standard says that allowable spellings are `inf`, `+inf`, `-inf`, `infinity`, `+infinity`, and `-infinity`. Make sure we recognize these even if the platform's `strtod` function doesn't.
- Expand ability to compare rows to records and arrays (Rafal Rzepecki, Tom Lane)
- Update time zone data files to tzdata release 2013d for DST law changes in Israel, Morocco, Palestine, and Paraguay. Also, historical zone data corrections for Macquarie Island.

E.80. Release 8.4.17

Release date: 2013-04-04

This release contains a variety of fixes from 8.4.16. For information about new features in the 8.4 major release, see 第 E.97 节.

E.80.1. Migration to Version 8.4.17

A dump/restore is not required for those running 8.4.X.

However, this release corrects several errors in management of GiST indexes. After installing this update, it is advisable to `REINDEX` any GiST indexes that meet one or more of the conditions described below.

Also, if you are upgrading from a version earlier than 8.4.10, see 第 E.87 节.

E.80.2. Changes

- Reset OpenSSL randomness state in each postmaster child process (Marko Kreen)

  This avoids a scenario wherein random numbers generated by `contrib/pgcrypto` functions might be relatively easy for another database user to guess. The risk is only significant when the postmaster is configured with `ssl` = `on` but most connections don't use SSL encryption. (CVE-2013-1900)
- Fix GiST indexes to not use "fuzzy" geometric comparisons when it's not appropriate to do so (Alexander Korotkov)

The core geometric types perform comparisons using "fuzzy" equality, but `gist_box_same` must do exact comparisons, else GiST indexes using it might become inconsistent. After installing this update, users should `REINDEX` any GiST indexes on `box`, `polygon`, `circle`, or `point` columns, since all of these use `gist_box_same`.

- Fix erroneous range-union and penalty logic in GiST indexes that use `contrib/btree_gist` for variable-width data types, that is `text`, `bytea`, `bit`, and `numeric` columns (Tom Lane)

  These errors could result in inconsistent indexes in which some keys that are present would not be found by searches, and also in useless index bloat. Users are advised to `REINDEX` such indexes after installing this update.

- Fix bugs in GiST page splitting code for multi-column indexes (Tom Lane)

  These errors could result in inconsistent indexes in which some keys that are present would not be found by searches, and also in indexes that are unnecessarily inefficient to search. Users are advised to `REINDEX` multi-column GiST indexes after installing this update.

- Fix infinite-loop risk in regular expression compilation (Tom Lane, Don Porter)

- Fix potential null-pointer dereference in regular expression compilation (Tom Lane)

- Fix `to_char()` to use ASCII-only case-folding rules where appropriate (Tom Lane)

  This fixes misbehavior of some template patterns that should be locale-independent, but mishandled "`I`" and "`i`" in Turkish locales.

- Fix unwanted rejection of timestamp `1999-12-31 24:00:00` (Tom Lane)

- Remove useless "picksplit doesn't support secondary split" log messages (Josh Hansen, Tom Lane)

  This message seems to have been added in expectation of code that was never written, and probably never will be, since GiST's default handling of secondary splits is actually pretty good. So stop nagging end users about it.

- Fix possible failure to send a session's last few transaction commit/abort counts to the statistics collector (Tom Lane)

- Eliminate memory leaks in PL/Perl's `spi_prepare()` function (Alex Hunsaker, Tom Lane)

- Fix pg_dumpall to handle database names containing "`=`" correctly (Heikki Linnakangas)

- Avoid crash in pg_dump when an incorrect connection string is given (Heikki Linnakangas)

- Ignore invalid indexes in pg_dump (Michael Paquier)

  Dumping invalid indexes can cause problems at restore time, for example if the reason the index creation failed was because it tried to enforce a uniqueness condition not satisfied by the table's data. Also, if the index creation is in fact still in progress, it seems reasonable to consider it to be an uncommitted DDL change, which pg_dump wouldn't be expected to dump anyway.

- Fix `contrib/pg_trgm`'s `similarity()` function to return zero for trigram-less strings (Tom Lane)

Previously it returned `NaN` due to internal division by zero.

- Update time zone data files to tzdata release 2013b for DST law changes in Chile, Haiti, Morocco, Paraguay, and some Russian areas. Also, historical zone data corrections for numerous places.

  Also, update the time zone abbreviation files for recent changes in Russia and elsewhere: `CHOT`, `GET`, `IRKT`, `KGT`, `KRAT`, `MAGT`, `MAWT`, `MSK`, `NOVT`, `OMST`, `TKT`, `VLAT`, `WST`, `YAKT`, `YEKT` now follow their current meanings, and `VOLT` (Europe/Volgograd) and `MIST` (Antarctica/Macquarie) are added to the default abbreviations list.

E.81. Release 8.4.16

Release date: 2013-02-07

This release contains a variety of fixes from 8.4.15. For information about new features in the 8.4 major release, see 第 E.97 节.

E.81.1. Migration to Version 8.4.16

A dump/restore is not required for those running 8.4.X.

However, if you are upgrading from a version earlier than 8.4.10, see 第 E.87 节.

E.81.2. Changes

- Prevent execution of `enum_recv` from SQL (Tom Lane)

  The function was misdeclared, allowing a simple SQL command to crash the server. In principle an attacker might be able to use it to examine the contents of server memory. Our thanks to Sumit Soni (via Secunia SVCRP) for reporting this issue. (CVE-2013-0255)

- Update minimum recovery point when truncating a relation file (Heikki Linnakangas)

  Once data has been discarded, it's no longer safe to stop recovery at an earlier point in the timeline.

- Fix SQL grammar to allow subscripting or field selection from a sub-SELECT result (Tom Lane)

- Protect against race conditions when scanning `pg_tablespace` (Stephen Frost, Tom Lane)

  `CREATE DATABASE` and `DROP DATABASE` could misbehave if there were concurrent updates of `pg_tablespace` entries.

- Prevent `DROP OWNED` from trying to drop whole databases or tablespaces (Álvaro Herrera)

  For safety, ownership of these objects must be reassigned, not dropped.

- Fix error in `vacuum_freeze_table_age` implementation (Andres Freund)

  In installations that have existed for more than `vacuum_freeze_min_age` transactions, this mistake prevented autovacuum from using partial-table scans, so that a full-table scan would always happen instead.

- Prevent misbehavior when a `RowExpr` or `XmlExpr` is parse-analyzed twice (Andres Freund, Tom Lane)

  This mistake could be user-visible in contexts such as `CREATE TABLE LIKE INCLUDING INDEXES`.

- Improve defenses against integer overflow in hashtable sizing calculations (Jeff Davis)

- Reject out-of-range dates in `to_date()` (Hitoshi Harada)

- Ensure that non-ASCII prompt strings are translated to the correct code page on Windows (Alexander Law, Noah Misch)

  This bug affected psql and some other client programs.

- Fix possible crash in psql's `\?` command when not connected to a database (Meng Qingzhong)

- Fix one-byte buffer overrun in libpq's `PQprintTuples` (Xi Wang)

  This ancient function is not used anywhere by PostgreSQL itself, but it might still be used by some client code.

- Make ecpglib use translated messages properly (Chen Huajun)

- Properly install ecpg_compat and pgtypes libraries on MSVC (Jiang Guiqing)

- Rearrange configure's tests for supplied functions so it is not fooled by bogus exports from libedit/libreadline (Christoph Berg)

- Ensure Windows build number increases over time (Magnus Hagander)

- Make pgxs build executables with the right `.exe` suffix when cross-compiling for Windows (Zoltan Boszormenyi)

- Add new timezone abbreviation `FET` (Tom Lane)

  This is now used in some eastern-European time zones.

E.82. Release 8.4.15

Release date: 2012-12-06

This release contains a variety of fixes from 8.4.14. For information about new features in the 8.4 major release, see 第 E.97 节.

E.82.1. Migration to Version 8.4.15

A dump/restore is not required for those running 8.4.X.

However, if you are upgrading from a version earlier than 8.4.10, see 第 E.87 节.

E.82.2. Changes

- Fix multiple bugs associated with `CREATE INDEX CONCURRENTLY` (Andres Freund, Tom Lane)

  Fix `CREATE INDEX CONCURRENTLY` to use in-place updates when changing the state of an index's `pg_index` row. This prevents race conditions that could cause

concurrent sessions to miss updating the target index, thus resulting in corrupt concurrently-created indexes.

Also, fix various other operations to ensure that they ignore invalid indexes resulting from a failed `CREATE INDEX CONCURRENTLY` command. The most important of these is `VACUUM`, because an auto-vacuum could easily be launched on the table before corrective action can be taken to fix or remove the invalid index.

- Avoid corruption of internal hash tables when out of memory (Hitoshi Harada)
- Fix planning of non-strict equivalence clauses above outer joins (Tom Lane)

  The planner could derive incorrect constraints from a clause equating a non-strict construct to something else, for example `WHERE COALESCE(foo, 0) = 0` when `foo` is coming from the nullable side of an outer join.
- Improve planner's ability to prove exclusion constraints from equivalence classes (Tom Lane)
- Fix partial-row matching in hashed subplans to handle cross-type cases correctly (Tom Lane)

  This affects multicolumn `NOT IN` subplans, such as `WHERE (a, b) NOT IN (SELECT x, y FROM ...)` when for instance `b` and `y` are `int4` and `int8` respectively. This mistake led to wrong answers or crashes depending on the specific datatypes involved.
- Acquire buffer lock when re-fetching the old tuple for an `AFTER ROW UPDATE/DELETE` trigger (Andres Freund)

  In very unusual circumstances, this oversight could result in passing incorrect data to the precheck logic for a foreign-key enforcement trigger. That could result in a crash, or in an incorrect decision about whether to fire the trigger.
- Fix `ALTER COLUMN TYPE` to handle inherited check constraints properly (Pavan Deolasee)

  This worked correctly in pre-8.4 releases, and now works correctly in 8.4 and later.
- Fix `REASSIGN OWNED` to handle grants on tablespaces (Álvaro Herrera)
- Ignore incorrect `pg_attribute` entries for system columns for views (Tom Lane)

  Views do not have any system columns. However, we forgot to remove such entries when converting a table to a view. That's fixed properly for 9.3 and later, but in previous branches we need to defend against existing mis-converted views.
- Fix rule printing to dump `INSERT INTO` *`table`* `DEFAULT VALUES` correctly (Tom Lane)
- Guard against stack overflow when there are too many `UNION`/`INTERSECT`/`EXCEPT` clauses in a query (Tom Lane)
- Prevent platform-dependent failures when dividing the minimum possible integer value by -1 (Xi Wang, Tom Lane)
- Fix possible access past end of string in date parsing (Hitoshi Harada)
- Produce an understandable error message if the length of the path name for a Unix-domain socket exceeds the platform-specific limit (Tom Lane, Andrew Dunstan)

Formerly, this would result in something quite unhelpful, such as “Non-recoverable failure in name resolution” .

- Fix memory leaks when sending composite column values to the client (Tom Lane)
- Make pg_ctl more robust about reading the `postmaster.pid` file (Heikki Linnakangas)

  Fix race conditions and possible file descriptor leakage.
- Fix possible crash in psql if incorrectly-encoded data is presented and the `client_encoding` setting is a client-only encoding, such as SJIS (Jiang Guiqing)
- Fix bugs in the `restore.sql` script emitted by pg_dump in `tar` output format (Tom Lane)

  The script would fail outright on tables whose names include upper-case characters. Also, make the script capable of restoring data in `--inserts` mode as well as the regular COPY mode.
- Fix pg_restore to accept POSIX-conformant `tar` files (Brian Weaver, Tom Lane)

  The original coding of pg_dump's `tar` output mode produced files that are not fully conformant with the POSIX standard. This has been corrected for version 9.3. This patch updates previous branches so that they will accept both the incorrect and the corrected formats, in hopes of avoiding compatibility problems when 9.3 comes out.
- Fix pg_resetxlog to locate `postmaster.pid` correctly when given a relative path to the data directory (Tom Lane)

  This mistake could lead to pg_resetxlog not noticing that there is an active postmaster using the data directory.
- Fix libpq's `lo_import()` and `lo_export()` functions to report file I/O errors properly (Tom Lane)
- Fix ecpg's processing of nested structure pointer variables (Muhammad Usama)
- Make `contrib/pageinspect`'s btree page inspection functions take buffer locks while examining pages (Tom Lane)
- Fix pgxs support for building loadable modules on AIX (Tom Lane)

  Building modules outside the original source tree didn't work on AIX.
- Update time zone data files to tzdata release 2012j for DST law changes in Cuba, Israel, Jordan, Libya, Palestine, Western Samoa, and portions of Brazil.

E.83. Release 8.4.14

Release date: 2012-09-24

This release contains a variety of fixes from 8.4.13. For information about new features in the 8.4 major release, see 第 E.97 节.

E.83.1. Migration to Version 8.4.14

A dump/restore is not required for those running 8.4.X.

However, if you are upgrading from a version earlier than 8.4.10, see 第 E.87 节.

E.83.2. Changes

- Fix planner's assignment of executor parameters, and fix executor's rescan logic for CTE plan nodes (Tom Lane)

  These errors could result in wrong answers from queries that scan the same `WITH` subquery multiple times.

- Improve page-splitting decisions in GiST indexes (Alexander Korotkov, Robert Haas, Tom Lane)

  Multi-column GiST indexes might suffer unexpected bloat due to this error.

- Fix cascading privilege revoke to stop if privileges are still held (Tom Lane)

  If we revoke a grant option from some role *X*, but *X* still holds that option via a grant from someone else, we should not recursively revoke the corresponding privilege from role(s) *Y* that *X* had granted it to.

- Fix handling of `SIGFPE` when PL/Perl is in use (Andres Freund)

  Perl resets the process's `SIGFPE` handler to `SIG_IGN`, which could result in crashes later on. Restore the normal Postgres signal handler after initializing PL/Perl.

- Prevent PL/Perl from crashing if a recursive PL/Perl function is redefined while being executed (Tom Lane)

- Work around possible misoptimization in PL/Perl (Tom Lane)

  Some Linux distributions contain an incorrect version of `pthread.h` that results in incorrect compiled code in PL/Perl, leading to crashes if a PL/Perl function calls another one that throws an error.

- Update time zone data files to tzdata release 2012f for DST law changes in Fiji

E.84. Release 8.4.13

Release date: 2012-08-17

This release contains a variety of fixes from 8.4.12. For information about new features in the 8.4 major release, see 第 E.97 节.

E.84.1. Migration to Version 8.4.13

A dump/restore is not required for those running 8.4.X.

However, if you are upgrading from a version earlier than 8.4.10, see 第 E.87 节.

E.84.2. Changes

- Prevent access to external files/URLs via XML entity references (Noah Misch, Tom Lane)

  `xml_parse()` would attempt to fetch external files or URLs as needed to resolve DTD and entity references in an XML value, thus allowing unprivileged database users to attempt to fetch data with the privileges of the database server. While the external data wouldn't get returned directly to the user, portions of it could be exposed in error messages if the data didn't parse as valid XML; and in any case the mere ability to check existence of a file might be useful to an attacker. (CVE-2012-3489)

- Prevent access to external files/URLs via `contrib/xml2`'s `xslt_process()` (Peter Eisentraut)

  libxslt offers the ability to read and write both files and URLs through stylesheet commands, thus allowing unprivileged database users to both read and write data with the privileges of the database server. Disable that through proper use of libxslt's security options. (CVE-2012-3488)

  Also, remove `xslt_process()`'s ability to fetch documents and stylesheets from external files/URLs. While this was a documented "feature", it was long regarded as a bad idea. The fix for CVE-2012-3489 broke that capability, and rather than expend effort on trying to fix it, we're just going to summarily remove it.

- Prevent too-early recycling of btree index pages (Noah Misch)

  When we allowed read-only transactions to skip assigning XIDs, we introduced the possibility that a deleted btree page could be recycled while a read-only transaction was still in flight to it. This would result in incorrect index search results. The probability of such an error occurring in the field seems very low because of the timing requirements, but nonetheless it should be fixed.

- Fix crash-safety bug with newly-created-or-reset sequences (Tom Lane)

  If `ALTER SEQUENCE` was executed on a freshly created or reset sequence, and then precisely one `nextval()` call was made on it, and then the server crashed, WAL replay would restore the sequence to a state in which it appeared that no `nextval()` had been done, thus allowing the first sequence value to be returned again by the next `nextval()` call. In particular this could manifest for `serial` columns, since creation of a serial column's sequence includes an `ALTER SEQUENCE OWNED BY` step.

- Ensure the `backup_label` file is fsync'd after `pg_start_backup()` (Dave Kerr)

- Back-patch 9.1 improvement to compress the fsync request queue (Robert Haas)

  This improves performance during checkpoints. The 9.1 change has now seen enough field testing to seem safe to back-patch.

- Only allow autovacuum to be auto-canceled by a directly blocked process (Tom Lane)

  The original coding could allow inconsistent behavior in some cases; in particular, an autovacuum could get canceled after less than `deadlock_timeout` grace period.

- Improve logging of autovacuum cancels (Robert Haas)

- Fix log collector so that `log_truncate_on_rotation` works during the very first log rotation after server start (Tom Lane)

- Fix `WITH` attached to a nested set operation (`UNION`/`INTERSECT`/`EXCEPT`) (Tom Lane)

- Ensure that a whole-row reference to a subquery doesn't include any extra `GROUP BY` or `ORDER BY` columns (Tom Lane)

- Disallow copying whole-row references in `CHECK` constraints and index definitions during `CREATE TABLE` (Tom Lane)

  This situation can arise in `CREATE TABLE` with `LIKE` or `INHERITS`. The copied whole-row variable was incorrectly labeled with the row type of the original table not the new one. Rejecting the case seems reasonable for `LIKE`, since the row types might well diverge later. For `INHERITS` we should ideally allow it, with an implicit coercion

to the parent table's row type; but that will require more work than seems safe to back-patch.

- Fix memory leak in `ARRAY(SELECT ...)` subqueries (Heikki Linnakangas, Tom Lane)
- Fix extraction of common prefixes from regular expressions (Tom Lane)

  The code could get confused by quantified parenthesized subexpressions, such as `^(foo)?bar`. This would lead to incorrect index optimization of searches for such patterns.
- Fix bugs with parsing signed *hh*:*mm* and *hh*:*mm*:*ss* fields in `interval` constants (Amit Kapila, Tom Lane)
- Report errors properly in `contrib/xml2`'s `xslt_process()` (Tom Lane)
- Update time zone data files to tzdata release 2012e for DST law changes in Morocco and Tokelau

E.85. Release 8.4.12

Release date: 2012-06-04

This release contains a variety of fixes from 8.4.11. For information about new features in the 8.4 major release, see 第 E.97 节.

E.85.1. Migration to Version 8.4.12

A dump/restore is not required for those running 8.4.X.

However, if you are upgrading from a version earlier than 8.4.10, see 第 E.87 节.

E.85.2. Changes

- Fix incorrect password transformation in `contrib/pgcrypto`'s DES `crypt()` function (Solar Designer)

  If a password string contained the byte value `0x80`, the remainder of the password was ignored, causing the password to be much weaker than it appeared. With this fix, the rest of the string is properly included in the DES hash. Any stored password values that are affected by this bug will thus no longer match, so the stored values may need to be updated. (CVE-2012-2143)
- Ignore `SECURITY DEFINER` and `SET` attributes for a procedural language's call handler (Tom Lane)

  Applying such attributes to a call handler could crash the server. (CVE-2012-2655)
- Allow numeric timezone offsets in `timestamp` input to be up to 16 hours away from UTC (Tom Lane)

  Some historical time zones have offsets larger than 15 hours, the previous limit. This could result in dumped data values being rejected during reload.
- Fix timestamp conversion to cope when the given time is exactly the last DST transition time for the current timezone (Tom Lane)

  This oversight has been there a long time, but was not noticed previously because most DST-using zones are presumed to have an indefinite sequence of future DST transitions.

- Fix `text` to `name` and `char` to `name` casts to perform string truncation correctly in multibyte encodings (Karl Schnaitter)
- Fix memory copying bug in `to_tsquery()` (Heikki Linnakangas)
- Fix planner's handling of outer PlaceHolderVars within subqueries (Tom Lane)

  This bug concerns sub-SELECTs that reference variables coming from the nullable side of an outer join of the surrounding query. In 9.1, queries affected by this bug would fail with "ERROR: Upper-level PlaceHolderVar found where not expected". But in 9.0 and 8.4, you'd silently get possibly-wrong answers, since the value transmitted into the subquery wouldn't go to null when it should.
- Fix slow session startup when `pg_attribute` is very large (Tom Lane)

  If `pg_attribute` exceeds one-fourth of `shared_buffers`, cache rebuilding code that is sometimes needed during session start would trigger the synchronized-scan logic, causing it to take many times longer than normal. The problem was particularly acute if many new sessions were starting at once.
- Ensure sequential scans check for query cancel reasonably often (Merlin Moncure)

  A scan encountering many consecutive pages that contain no live tuples would not respond to interrupts meanwhile.
- Ensure the Windows implementation of `PGSemaphoreLock()` clears `ImmediateInterruptOK` before returning (Tom Lane)

  This oversight meant that a query-cancel interrupt received later in the same query could be accepted at an unsafe time, with unpredictable but not good consequences.
- Show whole-row variables safely when printing views or rules (Abbas Butt, Tom Lane)

  Corner cases involving ambiguous names (that is, the name could be either a table or column name of the query) were printed in an ambiguous way, risking that the view or rule would be interpreted differently after dump and reload. Avoid the ambiguous case by attaching a no-op cast.
- Fix `COPY FROM` to properly handle null marker strings that correspond to invalid encoding (Tom Lane)

  A null marker string such as `E'\\0'` should work, and did work in the past, but the case got broken in 8.4.
- Ensure autovacuum worker processes perform stack depth checking properly (Heikki Linnakangas)

  Previously, infinite recursion in a function invoked by auto-`ANALYZE` could crash worker processes.
- Fix logging collector to not lose log coherency under high load (Andrew Dunstan)

  The collector previously could fail to reassemble large messages if it got too busy.
- Fix logging collector to ensure it will restart file rotation after receiving SIGHUP (Tom Lane)
- Fix WAL replay logic for GIN indexes to not fail if the index was subsequently dropped (Tom Lane)

- Fix memory leak in PL/pgSQL's `RETURN NEXT` command (Joe Conway)
- Fix PL/pgSQL's `GET DIAGNOSTICS` command when the target is the function's first variable (Tom Lane)
- Fix potential access off the end of memory in psql's expanded display (`\x`) mode (Peter Eisentraut)
- Fix several performance problems in pg_dump when the database contains many objects (Jeff Janes, Tom Lane)

  pg_dump could get very slow if the database contained many schemas, or if many objects are in dependency loops, or if there are many owned sequences.
- Fix `contrib/dblink`'s `dblink_exec()` to not leak temporary database connections upon error (Tom Lane)
- Fix `contrib/dblink` to report the correct connection name in error messages (Kyotaro Horiguchi)
- Update time zone data files to tzdata release 2012c for DST law changes in Antarctica, Armenia, Chile, Cuba, Falkland Islands, Gaza, Haiti, Hebron, Morocco, Syria, and Tokelau Islands; also historical corrections for Canada.

E.86. Release 8.4.11

Release date: 2012-02-27

This release contains a variety of fixes from 8.4.10. For information about new features in the 8.4 major release, see 第 E.97 节.

E.86.1. Migration to Version 8.4.11

A dump/restore is not required for those running 8.4.X.

However, if you are upgrading from a version earlier than 8.4.10, see 第 E.87 节.

E.86.2. Changes

- Require execute permission on the trigger function for `CREATE TRIGGER` (Robert Haas)

  This missing check could allow another user to execute a trigger function with forged input data, by installing it on a table he owns. This is only of significance for trigger functions marked `SECURITY DEFINER`, since otherwise trigger functions run as the table owner anyway. (CVE-2012-0866)
- Remove arbitrary limitation on length of common name in SSL certificates (Heikki Linnakangas)

  Both libpq and the server truncated the common name extracted from an SSL certificate at 32 bytes. Normally this would cause nothing worse than an unexpected verification failure, but there are some rather-implausible scenarios in which it might allow one certificate holder to impersonate another. The victim would have to have a common name exactly 32 bytes long, and the attacker would have to persuade a trusted CA to issue a certificate in which the common name has that string as a prefix. Impersonating a server would also require some additional exploit to redirect client connections. (CVE-2012-0867)
- Convert newlines to spaces in names written in pg_dump comments (Robert Haas)

pg_dump was incautious about sanitizing object names that are emitted within SQL comments in its output script. A name containing a newline would at least render the script syntactically incorrect. Maliciously crafted object names could present a SQL injection risk when the script is reloaded. (CVE-2012-0868)

- Fix btree index corruption from insertions concurrent with vacuuming (Tom Lane)

  An index page split caused by an insertion could sometimes cause a concurrently-running `VACUUM` to miss removing index entries that it should remove. After the corresponding table rows are removed, the dangling index entries would cause errors (such as "could not read block N in file ...") or worse, silently wrong query results after unrelated rows are re-inserted at the now-free table locations. This bug has been present since release 8.2, but occurs so infrequently that it was not diagnosed until now. If you have reason to suspect that it has happened in your database, reindexing the affected index will fix things.

- Update per-column permissions, not only per-table permissions, when changing table owner (Tom Lane)

  Failure to do this meant that any previously granted column permissions were still shown as having been granted by the old owner. This meant that neither the new owner nor a superuser could revoke the now-untraceable-to-table-owner permissions.

- Allow non-existent values for some settings in `ALTER USER/DATABASE SET` (Heikki Linnakangas)

  Allow `default_text_search_config`, `default_tablespace`, and `temp_tablespaces` to be set to names that are not known. This is because they might be known in another database where the setting is intended to be used, or for the tablespace cases because the tablespace might not be created yet. The same issue was previously recognized for `search_path`, and these settings now act like that one.

- Avoid crashing when we have problems deleting table files post-commit (Tom Lane)

  Dropping a table should lead to deleting the underlying disk files only after the transaction commits. In event of failure then (for instance, because of wrong file permissions) the code is supposed to just emit a warning message and go on, since it's too late to abort the transaction. This logic got broken as of release 8.4, causing such situations to result in a PANIC and an unrestartable database.

- Track the OID counter correctly during WAL replay, even when it wraps around (Tom Lane)

  Previously the OID counter would remain stuck at a high value until the system exited replay mode. The practical consequences of that are usually nil, but there are scenarios wherein a standby server that's been promoted to master might take a long time to advance the OID counter to a reasonable value once values are needed.

- Fix regular expression back-references with `*` attached (Tom Lane)

  Rather than enforcing an exact string match, the code would effectively accept any string that satisfies the pattern sub-expression referenced by the back-reference symbol.

  A similar problem still afflicts back-references that are embedded in a larger quantified expression, rather than being the immediate subject of the quantifier. This will be addressed in a future PostgreSQL release.

- Fix recently-introduced memory leak in processing of `inet/cidr` values (Heikki Linnakangas)

  A patch in the December 2011 releases of PostgreSQL caused memory leakage in these operations, which could be significant in scenarios such as building a btree index on such a column.

- Fix dangling pointer after `CREATE TABLE AS/SELECT INTO` in a SQL-language function (Tom Lane)

  In most cases this only led to an assertion failure in assert-enabled builds, but worse consequences seem possible.

- Avoid double close of file handle in syslogger on Windows (MauMau)

  Ordinarily this error was invisible, but it would cause an exception when running on a debug version of Windows.

- Fix I/O-conversion-related memory leaks in plpgsql (Andres Freund, Jan Urbanski, Tom Lane)

  Certain operations would leak memory until the end of the current function.

- Improve pg_dump's handling of inherited table columns (Tom Lane)

  pg_dump mishandled situations where a child column has a different default expression than its parent column. If the default is textually identical to the parent's default, but not actually the same (for instance, because of schema search path differences) it would not be recognized as different, so that after dump and restore the child would be allowed to inherit the parent's default. Child columns that are `NOT NULL` where their parent is not could also be restored subtly incorrectly.

- Fix pg_restore's direct-to-database mode for INSERT-style table data (Tom Lane)

  Direct-to-database restores from archive files made with `--inserts` or `--column-inserts` options fail when using pg_restore from a release dated September or December 2011, as a result of an oversight in a fix for another problem. The archive file itself is not at fault, and text-mode output is okay.

- Allow `AT` option in ecpg `DEALLOCATE` statements (Michael Meskes)

  The infrastructure to support this has been there for awhile, but through an oversight there was still an error check rejecting the case.

- Fix error in `contrib/intarray`'s `int[] & int[]` operator (Guillaume Lelarge)

  If the smallest integer the two input arrays have in common is 1, and there are smaller values in either array, then 1 would be incorrectly omitted from the result.

- Fix error detection in `contrib/pgcrypto`'s `encrypt_iv()` and `decrypt_iv()` (Marko Kreen)

  These functions failed to report certain types of invalid-input errors, and would instead return random garbage values for incorrect input.

- Fix one-byte buffer overrun in `contrib/test_parser` (Paul Guyot)

  The code would try to read one more byte than it should, which would crash in corner cases. Since `contrib/test_parser` is only example code, this is not a security issue in itself, but bad example code is still bad.

- Use `__sync_lock_test_and_set()` for spinlocks on ARM, if available (Martin Pitt)

  This function replaces our previous use of the `SWPB` instruction, which is deprecated and not available on ARMv6 and later. Reports suggest that the old code doesn't fail in an obvious way on recent ARM boards, but simply doesn't interlock concurrent accesses, leading to bizarre failures in multiprocess operation.

- Use `-fexcess-precision=standard` option when building with gcc versions that accept it (Andrew Dunstan)

  This prevents assorted scenarios wherein recent versions of gcc will produce creative results.

- Allow use of threaded Python on FreeBSD (Chris Rees)

  Our configure script previously believed that this combination wouldn't work; but FreeBSD fixed the problem, so remove that error check.

E.87. Release 8.4.10

Release date: 2011-12-05

This release contains a variety of fixes from 8.4.9. For information about new features in the 8.4 major release, see 第 E.97 节.

E.87.1. Migration to Version 8.4.10

A dump/restore is not required for those running 8.4.X.

However, a longstanding error was discovered in the definition of the `information_schema.referential_constraints` view. If you rely on correct results from that view, you should replace its definition as explained in the first changelog item below.

Also, if you are upgrading from a version earlier than 8.4.8, see 第 E.89 节.

E.87.2. Changes

- Fix bugs in `information_schema.referential_constraints` view (Tom Lane)

  This view was being insufficiently careful about matching the foreign-key constraint to the depended-on primary or unique key constraint. That could result in failure to show a foreign key constraint at all, or showing it multiple times, or claiming that it depends on a different constraint than the one it really does.

  Since the view definition is installed by initdb, merely upgrading will not fix the problem. If you need to fix this in an existing installation, you can (as a superuser) drop the `information_schema` schema then re-create it by sourcing *`SHAREDIR`*`/information_schema.sql`. (Run `pg_config --sharedir` if you're uncertain where *`SHAREDIR`* is.) This must be repeated in each database to be fixed.

- Fix incorrect replay of WAL records for GIN index updates (Tom Lane)

  This could result in transiently failing to find index entries after a crash, or on a hot-standby server. The problem would be repaired by the next `VACUUM` of the index, however.

- Fix TOAST-related data corruption during `CREATE TABLE dest AS SELECT * FROM src` or `INSERT INTO dest SELECT * FROM src` (Tom Lane)

If a table has been modified by `ALTER TABLE ADD COLUMN`, attempts to copy its data verbatim to another table could produce corrupt results in certain corner cases. The problem can only manifest in this precise form in 8.4 and later, but we patched earlier versions as well in case there are other code paths that could trigger the same bug.

- Fix race condition during toast table access from stale syscache entries (Tom Lane)

  The typical symptom was transient errors like "missing chunk number 0 for toast value NNNNN in pg_toast_2619", where the cited toast table would always belong to a system catalog.

- Track dependencies of functions on items used in parameter default expressions (Tom Lane)

  Previously, a referenced object could be dropped without having dropped or modified the function, leading to misbehavior when the function was used. Note that merely installing this update will not fix the missing dependency entries; to do that, you'd need to `CREATE OR REPLACE` each such function afterwards. If you have functions whose defaults depend on non-built-in objects, doing so is recommended.

- Allow inlining of set-returning SQL functions with multiple OUT parameters (Tom Lane)

- Make `DatumGetInetP()` unpack inet datums that have a 1-byte header, and add a new macro, `DatumGetInetPP()`, that does not (Heikki Linnakangas)

  This change affects no core code, but might prevent crashes in add-on code that expects `DatumGetInetP()` to produce an unpacked datum as per usual convention.

- Improve locale support in `money` type's input and output (Tom Lane)

  Aside from not supporting all standard `lc_monetary` formatting options, the input and output functions were inconsistent, meaning there were locales in which dumped `money` values could not be re-read.

- Don't let `transform_null_equals` affect `CASE foo WHEN NULL ...` constructs (Heikki Linnakangas)

  `transform_null_equals` is only supposed to affect `foo = NULL` expressions written directly by the user, not equality checks generated internally by this form of `CASE`.

- Change foreign-key trigger creation order to better support self-referential foreign keys (Tom Lane)

  For a cascading foreign key that references its own table, a row update will fire both the `ON UPDATE` trigger and the `CHECK` trigger as one event. The `ON UPDATE` trigger must execute first, else the `CHECK` will check a non-final state of the row and possibly throw an inappropriate error. However, the firing order of these triggers is determined by their names, which generally sort in creation order since the triggers have auto-generated names following the convention "RI_ConstraintTrigger_NNNN". A proper fix would require modifying that convention, which we will do in 9.2, but it seems risky to change it in existing releases. So this patch just changes the creation order of the triggers. Users encountering this type of error should drop and re-create the foreign key constraint to get its triggers into the right order.

- Avoid floating-point underflow while tracking buffer allocation rate (Greg Matthews)

While harmless in itself, on certain platforms this would result in annoying kernel log messages.

- Preserve configuration file name and line number values when starting child processes under Windows (Tom Lane)

  Formerly, these would not be displayed correctly in the `pg_settings` view.

- Preserve blank lines within commands in psql's command history (Robert Haas)

  The former behavior could cause problems if an empty line was removed from within a string literal, for example.

- Fix pg_dump to dump user-defined casts between auto-generated types, such as table rowtypes (Tom Lane)

- Use the preferred version of xsubpp to build PL/Perl, not necessarily the operating system's main copy (David Wheeler and Alex Hunsaker)

- Fix incorrect coding in `contrib/dict_int` and `contrib/dict_xsyn` (Tom Lane)

  Some functions incorrectly assumed that memory returned by `palloc()` is guaranteed zeroed.

- Honor query cancel interrupts promptly in `pgstatindex()` (Robert Haas)

- Ensure VPATH builds properly install all server header files (Peter Eisentraut)

- Shorten file names reported in verbose error messages (Peter Eisentraut)

  Regular builds have always reported just the name of the C file containing the error message call, but VPATH builds formerly reported an absolute path name.

- Fix interpretation of Windows timezone names for Central America (Tom Lane)

  Map "Central America Standard Time" to `CST6`, not `CST6CDT`, because DST is generally not observed anywhere in Central America.

- Update time zone data files to tzdata release 2011n for DST law changes in Brazil, Cuba, Fiji, Palestine, Russia, and Samoa; also historical corrections for Alaska and British East Africa.

E.88. Release 8.4.9

Release date: 2011-09-26

This release contains a variety of fixes from 8.4.8. For information about new features in the 8.4 major release, see 第 E.97 节.

E.88.1. Migration to Version 8.4.9

A dump/restore is not required for those running 8.4.X.

However, if you are upgrading from a version earlier than 8.4.8, see 第 E.89 节.

E.88.2. Changes

- Fix bugs in indexing of in-doubt HOT-updated tuples (Tom Lane)

These bugs could result in index corruption after reindexing a system catalog. They are not believed to affect user indexes.

- Fix multiple bugs in GiST index page split processing (Heikki Linnakangas)

  The probability of occurrence was low, but these could lead to index corruption.

- Fix possible buffer overrun in `tsvector_concat()` (Tom Lane)

  The function could underestimate the amount of memory needed for its result, leading to server crashes.

- Fix crash in `xml_recv` when processing a "standalone" parameter (Tom Lane)

- Make `pg_options_to_table` return NULL for an option with no value (Tom Lane)

  Previously such cases would result in a server crash.

- Avoid possibly accessing off the end of memory in `ANALYZE` and in SJIS-2004 encoding conversion (Noah Misch)

  This fixes some very-low-probability server crash scenarios.

- Prevent intermittent hang in interactions of startup process with bgwriter process (Simon Riggs)

  This affected recovery in non-hot-standby cases.

- Fix race condition in relcache init file invalidation (Tom Lane)

  There was a window wherein a new backend process could read a stale init file but miss the inval messages that would tell it the data is stale. The result would be bizarre failures in catalog accesses, typically "could not read block 0 in file ..." later during startup.

- Fix memory leak at end of a GiST index scan (Tom Lane)

  Commands that perform many separate GiST index scans, such as verification of a new GiST-based exclusion constraint on a table already containing many rows, could transiently require large amounts of memory due to this leak.

- Fix incorrect memory accounting (leading to possible memory bloat) in tuplestores supporting holdable cursors and plpgsql's `RETURN NEXT` command (Tom Lane)

- Fix performance problem when constructing a large, lossy bitmap (Tom Lane)

- Fix join selectivity estimation for unique columns (Tom Lane)

  This fixes an erroneous planner heuristic that could lead to poor estimates of the result size of a join.

- Fix nested PlaceHolderVar expressions that appear only in sub-select target lists (Tom Lane)

  This mistake could result in outputs of an outer join incorrectly appearing as NULL.

- Allow nested `EXISTS` queries to be optimized properly (Tom Lane)

- Fix array- and path-creating functions to ensure padding bytes are zeroes (Tom Lane)

This avoids some situations where the planner will think that semantically-equal constants are not equal, resulting in poor optimization.

- Fix `EXPLAIN` to handle gating Result nodes within inner-indexscan subplans (Tom Lane)

  The usual symptom of this oversight was "bogus varno" errors.

- Work around gcc 4.6.0 bug that breaks WAL replay (Tom Lane)

  This could lead to loss of committed transactions after a server crash.

- Fix dump bug for `VALUES` in a view (Tom Lane)

- Disallow `SELECT FOR UPDATE/SHARE` on sequences (Tom Lane)

  This operation doesn't work as expected and can lead to failures.

- Fix `VACUUM` so that it always updates `pg_class.reltuples/relpages` (Tom Lane)

  This fixes some scenarios where autovacuum could make increasingly poor decisions about when to vacuum tables.

- Defend against integer overflow when computing size of a hash table (Tom Lane)

- Fix cases where `CLUSTER` might attempt to access already-removed TOAST data (Tom Lane)

- Fix portability bugs in use of credentials control messages for "peer" authentication (Tom Lane)

- Fix SSPI login when multiple roundtrips are required (Ahmed Shinwari, Magnus Hagander)

  The typical symptom of this problem was "The function requested is not supported" errors during SSPI login.

- Throw an error if `pg_hba.conf` contains `hostssl` but SSL is disabled (Tom Lane)

  This was concluded to be more user-friendly than the previous behavior of silently ignoring such lines.

- Fix typo in `pg_srand48` seed initialization (Andres Freund)

  This led to failure to use all bits of the provided seed. This function is not used on most platforms (only those without `srandom`), and the potential security exposure from a less-random-than-expected seed seems minimal in any case.

- Avoid integer overflow when the sum of `LIMIT` and `OFFSET` values exceeds 2^63 (Heikki Linnakangas)

- Add overflow checks to `int4` and `int8` versions of `generate_series()` (Robert Haas)

- Fix trailing-zero removal in `to_char()` (Marti Raudsepp)

  In a format with `FM` and no digit positions after the decimal point, zeroes to the left of the decimal point could be removed incorrectly.

- Fix `pg_size_pretty()` to avoid overflow for inputs close to 2^63 (Tom Lane)

- Weaken plpgsql's check for typmod matching in record values (Tom Lane)

An overly enthusiastic check could lead to discarding length modifiers that should have been kept.

- Correctly handle quotes in locale names during initdb (Heikki Linnakangas)

  The case can arise with some Windows locales, such as “People's Republic of China”.

- Fix pg_upgrade to preserve toast tables' relfrozenxids during an upgrade from 8.3 (Bruce Momjian)

  Failure to do this could lead to `pg_clog` files being removed too soon after the upgrade.

- In pg_ctl, support silent mode for service registrations on Windows (MauMau)
- Fix psql's counting of script file line numbers during `COPY` from a different file (Tom Lane)
- Fix pg_restore's direct-to-database mode for `standard_conforming_strings` (Tom Lane)

  pg_restore could emit incorrect commands when restoring directly to a database server from an archive file that had been made with `standard_conforming_strings` set to `on`.

- Be more user-friendly about unsupported cases for parallel pg_restore (Tom Lane)

  This change ensures that such cases are detected and reported before any restore actions have been taken.

- Fix write-past-buffer-end and memory leak in libpq's LDAP service lookup code (Albe Laurenz)
- In libpq, avoid failures when using nonblocking I/O and an SSL connection (Martin Pihlak, Tom Lane)
- Improve libpq's handling of failures during connection startup (Tom Lane)

  In particular, the response to a server report of `fork()` failure during SSL connection startup is now saner.

- Improve libpq's error reporting for SSL failures (Tom Lane)
- Fix `PQsetvalue()` to avoid possible crash when adding a new tuple to a `PGresult` originally obtained from a server query (Andrew Chernow)
- Make ecpglib write `double` values with 15 digits precision (Akira Kurosawa)
- In ecpglib, be sure `LC_NUMERIC` setting is restored after an error (Michael Meskes)
- Apply upstream fix for blowfish signed-character bug (CVE-2011-2483) (Tom Lane)

  `contrib/pg_crypto`'s blowfish encryption code could give wrong results on platforms where char is signed (which is most), leading to encrypted passwords being weaker than they should be.

- Fix memory leak in `contrib/seg` (Heikki Linnakangas)
- Fix `pgstatindex()` to give consistent results for empty indexes (Tom Lane)
- Allow building with perl 5.14 (Alex Hunsaker)

- Update configure script's method for probing existence of system functions (Tom Lane)

  The version of autoconf we used in 8.3 and 8.2 could be fooled by compilers that perform link-time optimization.

- Fix assorted issues with build and install file paths containing spaces (Tom Lane)

- Update time zone data files to tzdata release 2011i for DST law changes in Canada, Egypt, Russia, Samoa, and South Sudan.

E.89. Release 8.4.8

Release date: 2011-04-18

This release contains a variety of fixes from 8.4.7. For information about new features in the 8.4 major release, see 第 E.97 节.

E.89.1. Migration to Version 8.4.8

A dump/restore is not required for those running 8.4.X.

However, if your installation was upgraded from a previous major release by running pg_upgrade, you should take action to prevent possible data loss due to a now-fixed bug in pg_upgrade. The recommended solution is to run `VACUUM FREEZE` on all TOAST tables. More information is available at http://wiki.postgresql.org/wiki/20110408pg_upgrade_fix[4].

Also, if you are upgrading from a version earlier than 8.4.2, see 第 E.95 节.

E.89.2. Changes

- Fix pg_upgrade's handling of TOAST tables (Bruce Momjian)

  The `pg_class.relfrozenxid` value for TOAST tables was not correctly copied into the new installation during pg_upgrade. This could later result in `pg_clog` files being discarded while they were still needed to validate tuples in the TOAST tables, leading to "could not access status of transaction" failures.

  This error poses a significant risk of data loss for installations that have been upgraded with pg_upgrade. This patch corrects the problem for future uses of pg_upgrade, but does not in itself cure the issue in installations that have been processed with a buggy version of pg_upgrade.

- Suppress incorrect "PD_ALL_VISIBLE flag was incorrectly set" warning (Heikki Linnakangas)

  `VACUUM` would sometimes issue this warning in cases that are actually valid.

- Disallow including a composite type in itself (Tom Lane)

  This prevents scenarios wherein the server could recurse infinitely while processing the composite type. While there are some possible uses for such a structure, they don't seem compelling enough to justify the effort required to make sure it always works safely.

- Avoid potential deadlock during catalog cache initialization (Nikhil Sontakke)

[4] http://wiki.postgresql.org/wiki/20110408pg_upgrade_fix

In some cases the cache loading code would acquire share lock on a system index before locking the index's catalog. This could deadlock against processes trying to acquire exclusive locks in the other, more standard order.

- Fix dangling-pointer problem in `BEFORE ROW UPDATE` trigger handling when there was a concurrent update to the target tuple (Tom Lane)

  This bug has been observed to result in intermittent "cannot extract system attribute from virtual tuple" failures while trying to do `UPDATE RETURNING ctid`. There is a very small probability of more serious errors, such as generating incorrect index entries for the updated tuple.

- Disallow `DROP TABLE` when there are pending deferred trigger events for the table (Tom Lane)

  Formerly the `DROP` would go through, leading to "could not open relation with OID nnn" errors when the triggers were eventually fired.

- Prevent crash triggered by constant-false WHERE conditions during GEQO optimization (Tom Lane)

- Improve planner's handling of semi-join and anti-join cases (Tom Lane)

- Fix selectivity estimation for text search to account for NULLs (Jesper Krogh)

- Improve PL/pgSQL's ability to handle row types with dropped columns (Pavel Stehule)

  This is a back-patch of fixes previously made in 9.0.

- Fix PL/Python memory leak involving array slices (Daniel Popowich)

- Fix pg_restore to cope with long lines (over 1KB) in TOC files (Tom Lane)

- Put in more safeguards against crashing due to division-by-zero with overly enthusiastic compiler optimization (Aurelien Jarno)

- Support use of dlopen() in FreeBSD and OpenBSD on MIPS (Tom Lane)

  There was a hard-wired assumption that this system function was not available on MIPS hardware on these systems. Use a compile-time test instead, since more recent versions have it.

- Fix compilation failures on HP-UX (Heikki Linnakangas)

- Fix version-incompatibility problem with libintl on Windows (Hiroshi Inoue)

- Fix usage of xcopy in Windows build scripts to work correctly under Windows 7 (Andrew Dunstan)

  This affects the build scripts only, not installation or usage.

- Fix path separator used by pg_regress on Cygwin (Andrew Dunstan)

- Update time zone data files to tzdata release 2011f for DST law changes in Chile, Cuba, Falkland Islands, Morocco, Samoa, and Turkey; also historical corrections for South Australia, Alaska, and Hawaii.

E.90. Release 8.4.7

Release date: 2011-01-31

This release contains a variety of fixes from 8.4.6. For information about new features in the 8.4 major release, see 第 E.97 节.

E.90.1. Migration to Version 8.4.7

A dump/restore is not required for those running 8.4.X. However, if you are upgrading from a version earlier than 8.4.2, see 第 E.95 节.

E.90.2. Changes

- Avoid failures when `EXPLAIN` tries to display a simple-form `CASE` expression (Tom Lane)

  If the `CASE`'s test expression was a constant, the planner could simplify the `CASE` into a form that confused the expression-display code, resulting in "unexpected CASE WHEN clause" errors.

- Fix assignment to an array slice that is before the existing range of subscripts (Tom Lane)

  If there was a gap between the newly added subscripts and the first pre-existing subscript, the code miscalculated how many entries needed to be copied from the old array's null bitmap, potentially leading to data corruption or crash.

- Avoid unexpected conversion overflow in planner for very distant date values (Tom Lane)

  The `date` type supports a wider range of dates than can be represented by the `timestamp` types, but the planner assumed it could always convert a date to timestamp with impunity.

- Fix pg_restore's text output for large objects (BLOBs) when `standard_conforming_strings` is on (Tom Lane)

  Although restoring directly to a database worked correctly, string escaping was incorrect if pg_restore was asked for SQL text output and `standard_conforming_strings` had been enabled in the source database.

- Fix erroneous parsing of `tsquery` values containing `... & !(subexpression) | ...` (Tom Lane)

  Queries containing this combination of operators were not executed correctly. The same error existed in `contrib/intarray`'s `query_int` type and `contrib/ltree`'s `ltxtquery` type.

- Fix buffer overrun in `contrib/intarray`'s input function for the `query_int` type (Apple)

  This bug is a security risk since the function's return address could be overwritten. Thanks to Apple Inc's security team for reporting this issue and supplying the fix. (CVE-2010-4015)

- Fix bug in `contrib/seg`'s GiST picksplit algorithm (Alexander Korotkov)

  This could result in considerable inefficiency, though not actually incorrect answers, in a GiST index on a `seg` column. If you have such an index, consider `REINDEX`ing it after installing this update. (This is identical to the bug that was fixed in `contrib/cube` in the previous update.)

E.91. Release 8.4.6

Release date: 2010-12-16

This release contains a variety of fixes from 8.4.5. For information about new features in the 8.4 major release, see 第 E.97 节.

E.91.1. Migration to Version 8.4.6

A dump/restore is not required for those running 8.4.X. However, if you are upgrading from a version earlier than 8.4.2, see 第 E.95 节.

E.91.2. Changes

- Force the default `wal_sync_method` to be `fdatasync` on Linux (Tom Lane, Marti Raudsepp)

  The default on Linux has actually been `fdatasync` for many years, but recent kernel changes caused PostgreSQL to choose `open_datasync` instead. This choice did not result in any performance improvement, and caused outright failures on certain filesystems, notably `ext4` with the `data=journal` mount option.

- Fix assorted bugs in WAL replay logic for GIN indexes (Tom Lane)

  This could result in "bad buffer id: 0" failures or corruption of index contents during replication.

- Fix recovery from base backup when the starting checkpoint WAL record is not in the same WAL segment as its redo point (Jeff Davis)

- Fix persistent slowdown of autovacuum workers when multiple workers remain active for a long time (Tom Lane)

  The effective `vacuum_cost_limit` for an autovacuum worker could drop to nearly zero if it processed enough tables, causing it to run extremely slowly.

- Add support for detecting register-stack overrun on `IA64` (Tom Lane)

  The `IA64` architecture has two hardware stacks. Full prevention of stack-overrun failures requires checking both.

- Add a check for stack overflow in `copyObject()` (Tom Lane)

  Certain code paths could crash due to stack overflow given a sufficiently complex query.

- Fix detection of page splits in temporary GiST indexes (Heikki Linnakangas)

  It is possible to have a "concurrent" page split in a temporary index, if for example there is an open cursor scanning the index when an insertion is done. GiST failed to detect this case and hence could deliver wrong results when execution of the cursor continued.

- Fix error checking during early connection processing (Tom Lane)

  The check for too many child processes was skipped in some cases, possibly leading to postmaster crash when attempting to add the new child process to fixed-size arrays.

- Improve efficiency of window functions (Tom Lane)

  Certain cases where a large number of tuples needed to be read in advance, but `work_mem` was large enough to allow them all to be held in memory, were unexpectedly slow. `percent_rank()`, `cume_dist()` and `ntile()` in particular were subject to this problem.

- Avoid memory leakage while `ANALYZE`'ing complex index expressions (Tom Lane)

- Ensure an index that uses a whole-row Var still depends on its table (Tom Lane)

  An index declared like `create index i on t (foo(t.*))` would not automatically get dropped when its table was dropped.

- Do not "inline" a SQL function with multiple `OUT` parameters (Tom Lane)

  This avoids a possible crash due to loss of information about the expected result rowtype.

- Behave correctly if `ORDER BY`, `LIMIT`, `FOR UPDATE`, or `WITH` is attached to the `VALUES` part of `INSERT ... VALUES` (Tom Lane)

- Fix constant-folding of `COALESCE()` expressions (Tom Lane)

  The planner would sometimes attempt to evaluate sub-expressions that in fact could never be reached, possibly leading to unexpected errors.

- Fix postmaster crash when connection acceptance (`accept()` or one of the calls made immediately after it) fails, and the postmaster was compiled with GSSAPI support (Alexander Chernikov)

- Fix missed unlink of temporary files when `log_temp_files` is active (Tom Lane)

  If an error occurred while attempting to emit the log message, the unlink was not done, resulting in accumulation of temp files.

- Add print functionality for `InhRelation` nodes (Tom Lane)

  This avoids a failure when `debug_print_parse` is enabled and certain types of query are executed.

- Fix incorrect calculation of distance from a point to a horizontal line segment (Tom Lane)

  This bug affected several different geometric distance-measurement operators.

- Fix incorrect calculation of transaction status in ecpg (Itagaki Takahiro)

- Fix PL/pgSQL's handling of "simple" expressions to not fail in recursion or error-recovery cases (Tom Lane)

- Fix PL/Python's handling of set-returning functions (Jan Urbanski)

  Attempts to call SPI functions within the iterator generating a set result would fail.

- Fix bug in `contrib/cube`'s GiST picksplit algorithm (Alexander Korotkov)

  This could result in considerable inefficiency, though not actually incorrect answers, in a GiST index on a `cube` column. If you have such an index, consider `REINDEX`ing it after installing this update.

- Don't emit “identifier will be truncated” notices in `contrib/dblink` except when creating new connections (Itagaki Takahiro)
- Fix potential coredump on missing public key in `contrib/pgcrypto` (Marti Raudsepp)
- Fix memory leak in `contrib/xml2`'s XPath query functions (Tom Lane)
- Update time zone data files to tzdata release 2010o for DST law changes in Fiji and Samoa; also historical corrections for Hong Kong.

E.92. Release 8.4.5

Release date: 2010-10-04

This release contains a variety of fixes from 8.4.4. For information about new features in the 8.4 major release, see 第 E.97 节.

E.92.1. Migration to Version 8.4.5

A dump/restore is not required for those running 8.4.X. However, if you are upgrading from a version earlier than 8.4.2, see 第 E.95 节.

E.92.2. Changes

- Use a separate interpreter for each calling SQL userid in PL/Perl and PL/Tcl (Tom Lane)

  This change prevents security problems that can be caused by subverting Perl or Tcl code that will be executed later in the same session under another SQL user identity (for example, within a `SECURITY DEFINER` function). Most scripting languages offer numerous ways that that might be done, such as redefining standard functions or operators called by the target function. Without this change, any SQL user with Perl or Tcl language usage rights can do essentially anything with the SQL privileges of the target function's owner.

  The cost of this change is that intentional communication among Perl and Tcl functions becomes more difficult. To provide an escape hatch, PL/PerlU and PL/TclU functions continue to use only one interpreter per session. This is not considered a security issue since all such functions execute at the trust level of a database superuser already.

  It is likely that third-party procedural languages that claim to offer trusted execution have similar security issues. We advise contacting the authors of any PL you are depending on for security-critical purposes.

  Our thanks to Tim Bunce for pointing out this issue (CVE-2010-3433).

- Prevent possible crashes in `pg_get_expr()` by disallowing it from being called with an argument that is not one of the system catalog columns it's intended to be used with (Heikki Linnakangas, Tom Lane)
- Treat exit code 128 (`ERROR_WAIT_NO_CHILDREN`) as non-fatal on Windows (Magnus Hagander)

  Under high load, Windows processes will sometimes fail at startup with this error code. Formerly the postmaster treated this as a panic condition and restarted the whole database, but that seems to be an overreaction.

- Fix incorrect placement of placeholder evaluation (Tom Lane)

  This bug could result in query outputs being non-null when they should be null, in cases where the inner side of an outer join is a sub-select with non-strict expressions in its output list.

- Fix possible duplicate scans of `UNION ALL` member relations (Tom Lane)

- Fix "cannot handle unplanned sub-select" error (Tom Lane)

  This occurred when a sub-select contains a join alias reference that expands into an expression containing another sub-select.

- Fix mishandling of whole-row Vars that reference a view or sub-select and appear within a nested sub-select (Tom Lane)

- Fix mishandling of cross-type `IN` comparisons (Tom Lane)

  This could result in failures if the planner tried to implement an `IN` join with a sort-then-unique-then-plain-join plan.

- Fix computation of `ANALYZE` statistics for `tsvector` columns (Jan Urbanski)

  The original coding could produce incorrect statistics, leading to poor plan choices later.

- Improve planner's estimate of memory used by `array_agg()`, `string_agg()`, and similar aggregate functions (Hitoshi Harada)

  The previous drastic underestimate could lead to out-of-memory failures due to inappropriate choice of a hash-aggregation plan.

- Fix failure to mark cached plans as transient (Tom Lane)

  If a plan is prepared while `CREATE INDEX CONCURRENTLY` is in progress for one of the referenced tables, it is supposed to be re-planned once the index is ready for use. This was not happening reliably.

- Reduce PANIC to ERROR in some occasionally-reported btree failure cases, and provide additional detail in the resulting error messages (Tom Lane)

  This should improve the system's robustness with corrupted indexes.

- Fix incorrect search logic for partial-match queries with GIN indexes (Tom Lane)

  Cases involving AND/OR combination of several GIN index conditions didn't always give the right answer, and were sometimes much slower than necessary.

- Prevent show_session_authorization() from crashing within autovacuum processes (Tom Lane)

- Defend against functions returning setof record where not all the returned rows are actually of the same rowtype (Tom Lane)

- Fix possible corruption of pending trigger event lists during subtransaction rollback (Tom Lane)

  This could lead to a crash or incorrect firing of triggers.

- Fix possible failure when hashing a pass-by-reference function result (Tao Ma, Tom Lane)

- Improve merge join's handling of NULLs in the join columns (Tom Lane)

A merge join can now stop entirely upon reaching the first NULL, if the sort order is such that NULLs sort high.

- Take care to fsync the contents of lockfiles (both `postmaster.pid` and the socket lockfile) while writing them (Tom Lane)

  This omission could result in corrupted lockfile contents if the machine crashes shortly after postmaster start. That could in turn prevent subsequent attempts to start the postmaster from succeeding, until the lockfile is manually removed.

- Avoid recursion while assigning XIDs to heavily-nested subtransactions (Andres Freund, Robert Haas)

  The original coding could result in a crash if there was limited stack space.

- Avoid holding open old WAL segments in the walwriter process (Magnus Hagander, Heikki Linnakangas)

  The previous coding would prevent removal of no-longer-needed segments.

- Fix `log_line_prefix`'s `%i` escape, which could produce junk early in backend startup (Tom Lane)

- Prevent misinterpretation of partially-specified relation options for TOAST tables (Itagaki Takahiro)

  In particular, `fillfactor` would be read as zero if any other reloption had been set for the table, leading to serious bloat.

- Fix inheritance count tracking in `ALTER TABLE ... ADD CONSTRAINT` (Robert Haas)

- Fix possible data corruption in `ALTER TABLE ... SET TABLESPACE` when archiving is enabled (Jeff Davis)

- Allow `CREATE DATABASE` and `ALTER DATABASE ... SET TABLESPACE` to be interrupted by query-cancel (Guillaume Lelarge)

- Improve `CREATE INDEX`'s checking of whether proposed index expressions are immutable (Tom Lane)

- Fix `REASSIGN OWNED` to handle operator classes and families (Asko Tiidumaa)

- Fix possible core dump when comparing two empty `tsquery` values (Tom Lane)

- Fix `LIKE`'s handling of patterns containing `%` followed by _ (Tom Lane)

  We've fixed this before, but there were still some incorrectly-handled cases.

- Re-allow input of Julian dates prior to 0001-01-01 AD (Tom Lane)

  Input such as `'J100000'::date` worked before 8.4, but was unintentionally broken by added error-checking.

- Fix PL/pgSQL to throw an error, not crash, if a cursor is closed within a `FOR` loop that is iterating over that cursor (Heikki Linnakangas)

- In PL/Python, defend against null pointer results from `PyCObject_AsVoidPtr` and `PyCObject_FromVoidPtr` (Peter Eisentraut)

- In libpq, fix full SSL certificate verification for the case where both `host` and `hostaddr` are specified (Tom Lane)

- Make psql recognize `DISCARD ALL` as a command that should not be encased in a transaction block in autocommit-off mode (Itagaki Takahiro)
- Fix some issues in pg_dump's handling of SQL/MED objects (Tom Lane)

  Notably, pg_dump would always fail if run by a non-superuser, which was not intended.
- Improve pg_dump and pg_restore's handling of non-seekable archive files (Tom Lane, Robert Haas)

  This is important for proper functioning of parallel restore.
- Improve parallel pg_restore's ability to cope with selective restore (`-L` option) (Tom Lane)

  The original code tended to fail if the `-L` file commanded a non-default restore ordering.
- Fix ecpg to process data from `RETURNING` clauses correctly (Michael Meskes)
- Fix some memory leaks in ecpg (Zoltan Boszormenyi)
- Improve `contrib/dblink`'s handling of tables containing dropped columns (Tom Lane)
- Fix connection leak after "duplicate connection name" errors in `contrib/dblink` (Itagaki Takahiro)
- Fix `contrib/dblink` to handle connection names longer than 62 bytes correctly (Itagaki Takahiro)
- Add `hstore(text, text)` function to `contrib/hstore` (Robert Haas)

  This function is the recommended substitute for the now-deprecated `=>` operator. It was back-patched so that future-proofed code can be used with older server versions. Note that the patch will be effective only after `contrib/hstore` is installed or reinstalled in a particular database. Users might prefer to execute the `CREATE FUNCTION` command by hand, instead.
- Update build infrastructure and documentation to reflect the source code repository's move from CVS to Git (Magnus Hagander and others)
- Update time zone data files to tzdata release 2010l for DST law changes in Egypt and Palestine; also historical corrections for Finland.

  This change also adds new names for two Micronesian timezones: Pacific/Chuuk is now preferred over Pacific/Truk (and the preferred abbreviation is CHUT not TRUT) and Pacific/Pohnpei is preferred over Pacific/Ponape.
- Make Windows' "N. Central Asia Standard Time" timezone map to Asia/Novosibirsk, not Asia/Almaty (Magnus Hagander)

  Microsoft changed the DST behavior of this zone in the timezone update from KB976098. Asia/Novosibirsk is a better match to its new behavior.

E.93. Release 8.4.4

Release date: 2010-05-17

This release contains a variety of fixes from 8.4.3. For information about new features in the 8.4 major release, see 第 E.97 节.

E.93.1. Migration to Version 8.4.4

A dump/restore is not required for those running 8.4.X. However, if you are upgrading from a version earlier than 8.4.2, see 第 E.95 节.

E.93.2. Changes

- Enforce restrictions in `plperl` using an opmask applied to the whole interpreter, instead of using `Safe.pm` (Tim Bunce, Andrew Dunstan)

  Recent developments have convinced us that `Safe.pm` is too insecure to rely on for making `plperl` trustable. This change removes use of `Safe.pm` altogether, in favor of using a separate interpreter with an opcode mask that is always applied. Pleasant side effects of the change include that it is now possible to use Perl's `strict` pragma in a natural way in `plperl`, and that Perl's `$a` and `$b` variables work as expected in sort routines, and that function compilation is significantly faster. (CVE-2010-1169)

- Prevent PL/Tcl from executing untrustworthy code from `pltcl_modules` (Tom)

  PL/Tcl's feature for autoloading Tcl code from a database table could be exploited for trojan-horse attacks, because there was no restriction on who could create or insert into that table. This change disables the feature unless `pltcl_modules` is owned by a superuser. (However, the permissions on the table are not checked, so installations that really need a less-than-secure modules table can still grant suitable privileges to trusted non-superusers.) Also, prevent loading code into the unrestricted "normal" Tcl interpreter unless we are really going to execute a `pltclu` function. (CVE-2010-1170)

- Fix data corruption during WAL replay of `ALTER ... SET TABLESPACE` (Tom)

  When `archive_mode` is on, `ALTER ... SET TABLESPACE` generates a WAL record whose replay logic was incorrect. It could write the data to the wrong place, leading to possibly-unrecoverable data corruption. Data corruption would be observed on standby slaves, and could occur on the master as well if a database crash and recovery occurred after committing the `ALTER` and before the next checkpoint.

- Fix possible crash if a cache reset message is received during rebuild of a relcache entry (Heikki)

  This error was introduced in 8.4.3 while fixing a related failure.

- Apply per-function GUC settings while running the language validator for the function (Itagaki Takahiro)

  This avoids failures if the function's code is invalid without the setting; an example is that SQL functions may not parse if the `search_path` is not correct.

- Do constraint exclusion for inherited `UPDATE` and `DELETE` target tables when `constraint_exclusion` = `partition` (Tom)

  Due to an oversight, this setting previously only caused constraint exclusion to be checked in `SELECT` commands.

- Do not allow an unprivileged user to reset superuser-only parameter settings (Alvaro)

Previously, if an unprivileged user ran `ALTER USER ... RESET ALL` for himself, or `ALTER DATABASE ... RESET ALL` for a database he owns, this would remove all special parameter settings for the user or database, even ones that are only supposed to be changeable by a superuser. Now, the `ALTER` will only remove the parameters that the user has permission to change.

- Avoid possible crash during backend shutdown if shutdown occurs when a `CONTEXT` addition would be made to log entries (Tom)

  In some cases the context-printing function would fail because the current transaction had already been rolled back when it came time to print a log message.

- Fix erroneous handling of `%r` parameter in `recovery_end_command` (Heikki)

  The value always came out zero.

- Ensure the archiver process responds to changes in `archive_command` as soon as possible (Tom)
- Fix pl/pgsql's `CASE` statement to not fail when the case expression is a query that returns no rows (Tom)
- Update pl/perl's `ppport.h` for modern Perl versions (Andrew)
- Fix assorted memory leaks in pl/python (Andreas Freund, Tom)
- Handle empty-string connect parameters properly in ecpg (Michael)
- Prevent infinite recursion in psql when expanding a variable that refers to itself (Tom)
- Fix psql's `\copy` to not add spaces around a dot within `\copy (select ...)` (Tom)

  Addition of spaces around the decimal point in a numeric literal would result in a syntax error.

- Avoid formatting failure in psql when running in a locale context that doesn't match the `client_encoding` (Tom)
- Fix unnecessary "GIN indexes do not support whole-index scans" errors for unsatisfiable queries using `contrib/intarray` operators (Tom)
- Ensure that `contrib/pgstattuple` functions respond to cancel interrupts promptly (Tatsuhito Kasahara)
- Make server startup deal properly with the case that `shmget()` returns `EINVAL` for an existing shared memory segment (Tom)

  This behavior has been observed on BSD-derived kernels including OS X. It resulted in an entirely-misleading startup failure complaining that the shared memory request size was too large.

- Avoid possible crashes in syslogger process on Windows (Heikki)
- Deal more robustly with incomplete time zone information in the Windows registry (Magnus)
- Update the set of known Windows time zone names (Magnus)
- Update time zone data files to tzdata release 2010j for DST law changes in Argentina, Australian Antarctic, Bangladesh, Mexico, Morocco, Pakistan, Palestine, Russia, Syria, Tunisia; also historical corrections for Taiwan.

Also, add `PKST` (Pakistan Summer Time) to the default set of timezone abbreviations.

E.94. Release 8.4.3

Release date: 2010-03-15

This release contains a variety of fixes from 8.4.2. For information about new features in the 8.4 major release, see 第 E.97 节.

E.94.1. Migration to Version 8.4.3

A dump/restore is not required for those running 8.4.X. However, if you are upgrading from a version earlier than 8.4.2, see 第 E.95 节.

E.94.2. Changes

- Add new configuration parameter `ssl_renegotiation_limit` to control how often we do session key renegotiation for an SSL connection (Magnus)

  This can be set to zero to disable renegotiation completely, which may be required if a broken SSL library is used. In particular, some vendors are shipping stopgap patches for CVE-2009-3555 that cause renegotiation attempts to fail.

- Fix possible deadlock during backend startup (Tom)
- Fix possible crashes due to not handling errors during relcache reload cleanly (Tom)
- Fix possible crash due to use of dangling pointer to a cached plan (Tatsuo)
- Fix possible crash due to overenthusiastic invalidation of cached plan for `ROLLBACK` (Tom)
- Fix possible crashes when trying to recover from a failure in subtransaction start (Tom)
- Fix server memory leak associated with use of savepoints and a client encoding different from server's encoding (Tom)
- Fix incorrect WAL data emitted during end-of-recovery cleanup of a GIST index page split (Yoichi Hirai)

  This would result in index corruption, or even more likely an error during WAL replay, if we were unlucky enough to crash during end-of-recovery cleanup after having completed an incomplete GIST insertion.

- Fix bug in WAL redo cleanup method for GIN indexes (Heikki)
- Fix incorrect comparison of scan key in GIN index search (Teodor)
- Make `substring()` for `bit` types treat any negative length as meaning "all the rest of the string" (Tom)

  The previous coding treated only -1 that way, and would produce an invalid result value for other negative values, possibly leading to a crash (CVE-2010-0442).

- Fix integer-to-bit-string conversions to handle the first fractional byte correctly when the output bit width is wider than the given integer by something other than a multiple of 8 bits (Tom)
- Fix some cases of pathologically slow regular expression matching (Tom)

- Fix bug occurring when trying to inline a SQL function that returns a set of a composite type that contains dropped columns (Tom)
- Fix bug with trying to update a field of an element of a composite-type array column (Tom)
- Avoid failure when `EXPLAIN` has to print a FieldStore or assignment ArrayRef expression (Tom)

  These cases can arise now that `EXPLAIN VERBOSE` tries to print plan node target lists.
- Avoid an unnecessary coercion failure in some cases where an undecorated literal string appears in a subquery within `UNION`/`INTERSECT`/`EXCEPT` (Tom)

  This fixes a regression for some cases that worked before 8.4.
- Avoid undesirable rowtype compatibility check failures in some cases where a whole-row Var has a rowtype that contains dropped columns (Tom)
- Fix the `STOP WAL LOCATION` entry in backup history files to report the next WAL segment's name when the end location is exactly at a segment boundary (Itagaki Takahiro)
- Always pass the catalog ID to an option validator function specified in `CREATE FOREIGN DATA WRAPPER` (Martin Pihlak)
- Fix some more cases of temporary-file leakage (Heikki)

  This corrects a problem introduced in the previous minor release. One case that failed is when a plpgsql function returning set is called within another function's exception handler.
- Add support for doing `FULL JOIN ON FALSE` (Tom)

  This prevents a regression from pre-8.4 releases for some queries that can now be simplified to a constant-false join condition.
- Improve constraint exclusion processing of boolean-variable cases, in particular make it possible to exclude a partition that has a “bool_column = false” constraint (Tom)
- Prevent treating an `INOUT` cast as representing binary compatibility (Heikki)
- Include column name in the message when warning about inability to grant or revoke column-level privileges (Stephen Frost)

  This is more useful than before and helps to prevent confusion when a `REVOKE` generates multiple messages, which formerly appeared to be duplicates.
- When reading `pg_hba.conf` and related files, do not treat `@something` as a file inclusion request if the `@` appears inside quote marks; also, never treat `@` by itself as a file inclusion request (Tom)

  This prevents erratic behavior if a role or database name starts with `@`. If you need to include a file whose path name contains spaces, you can still do so, but you must write `@"/path to/file"` rather than putting the quotes around the whole construct.
- Prevent infinite loop on some platforms if a directory is named as an inclusion target in `pg_hba.conf` and related files (Tom)

- Fix possible infinite loop if `SSL_read` or `SSL_write` fails without setting `errno` (Tom)

  This is reportedly possible with some Windows versions of openssl.

- Disallow GSSAPI authentication on local connections, since it requires a hostname to function correctly (Magnus)

- Protect ecpg against applications freeing strings unexpectedly (Michael)

- Make ecpg report the proper SQLSTATE if the connection disappears (Michael)

- Fix translation of cell contents in psql `\d` output (Heikki)

- Fix psql's `numericlocale` option to not format strings it shouldn't in latex and troff output formats (Heikki)

- Fix a small per-query memory leak in psql (Tom)

- Make psql return the correct exit status (3) when `ON_ERROR_STOP` and `--single-transaction` are both specified and an error occurs during the implied `COMMIT` (Bruce)

- Fix pg_dump's output of permissions for foreign servers (Heikki)

- Fix possible crash in parallel pg_restore due to out-of-range dependency IDs (Tom)

- Fix plpgsql failure in one case where a composite column is set to NULL (Tom)

- Fix possible failure when calling PL/Perl functions from PL/PerlU or vice versa (Tim Bunce)

- Add `volatile` markings in PL/Python to avoid possible compiler-specific misbehavior (Zdenek Kotala)

- Ensure PL/Tcl initializes the Tcl interpreter fully (Tom)

  The only known symptom of this oversight is that the Tcl `clock` command misbehaves if using Tcl 8.5 or later.

- Prevent `ExecutorEnd` from being run on portals created within a failed transaction or subtransaction (Tom)

  This is known to cause issues when using `contrib/auto_explain`.

- Prevent crash in `contrib/dblink` when too many key columns are specified to a `dblink_build_sql_*` function (Rushabh Lathia, Joe Conway)

- Allow zero-dimensional arrays in `contrib/ltree` operations (Tom)

  This case was formerly rejected as an error, but it's more convenient to treat it the same as a zero-element array. In particular this avoids unnecessary failures when an `ltree` operation is applied to the result of `ARRAY(SELECT ...)` and the sub-select returns no rows.

- Fix assorted crashes in `contrib/xml2` caused by sloppy memory management (Tom)

- Make building of `contrib/xml2` more robust on Windows (Andrew)

- Fix race condition in Windows signal handling (Radu Ilie)

  One known symptom of this bug is that rows in `pg_listener` could be dropped under heavy load.

- Make the configure script report failure if the C compiler does not provide a working 64-bit integer datatype (Tom)

  This case has been broken for some time, and no longer seems worth supporting, so just reject it at configure time instead.

- Update time zone data files to tzdata release 2010e for DST law changes in Bangladesh, Chile, Fiji, Mexico, Paraguay, Samoa.

E.95. Release 8.4.2

Release date: 2009-12-14

This release contains a variety of fixes from 8.4.1. For information about new features in the 8.4 major release, see 第 E.97 节.

E.95.1. Migration to Version 8.4.2

A dump/restore is not required for those running 8.4.X. However, if you have any hash indexes, you should `REINDEX` them after updating to 8.4.2, to repair possible damage.

E.95.2. Changes

- Protect against indirect security threats caused by index functions changing session-local state (Gurjeet Singh, Tom)

  This change prevents allegedly-immutable index functions from possibly subverting a superuser's session (CVE-2009-4136).

- Reject SSL certificates containing an embedded null byte in the common name (CN) field (Magnus)

  This prevents unintended matching of a certificate to a server or client name during SSL validation (CVE-2009-4034).

- Fix hash index corruption (Tom)

  The 8.4 change that made hash indexes keep entries sorted by hash value failed to update the bucket splitting and compaction routines to preserve the ordering. So application of either of those operations could lead to permanent corruption of an index, in the sense that searches might fail to find entries that are present. To deal with this, it is recommended to `REINDEX` any hash indexes you may have after installing this update.

- Fix possible crash during backend-startup-time cache initialization (Tom)

- Avoid crash on empty thesaurus dictionary (Tom)

- Prevent signals from interrupting `VACUUM` at unsafe times (Alvaro)

  This fix prevents a PANIC if a `VACUUM FULL` is canceled after it's already committed its tuple movements, as well as transient errors if a plain `VACUUM` is interrupted after having truncated the table.

- Fix possible crash due to integer overflow in hash table size calculation (Tom)

  This could occur with extremely large planner estimates for the size of a hashjoin's result.

- Fix crash if a `DROP` is attempted on an internally-dependent object (Tom)
- Fix very rare crash in `inet/cidr` comparisons (Chris Mikkelson)
- Ensure that shared tuple-level locks held by prepared transactions are not ignored (Heikki)
- Fix premature drop of temporary files used for a cursor that is accessed within a subtransaction (Heikki)
- Fix memory leak in syslogger process when rotating to a new CSV logfile (Tom)
- Fix memory leak in postmaster when re-parsing `pg_hba.conf` (Tom)
- Fix Windows permission-downgrade logic (Jesse Morris)

  This fixes some cases where the database failed to start on Windows, often with misleading error messages such as "could not locate matching postgres executable" .
- Make `FOR UPDATE/SHARE` in the primary query not propagate into `WITH` queries (Tom)

  For example, in

  ```
  WITH w AS (SELECT * FROM foo) SELECT * FROM w, bar ... FOR UPDATE
  ```

  the `FOR UPDATE` will now affect `bar` but not `foo`. This is more useful and consistent than the original 8.4 behavior, which tried to propagate `FOR UPDATE` into the `WITH` query but always failed due to assorted implementation restrictions. It also follows the design rule that `WITH` queries are executed as if independent of the main query.
- Fix bug with a `WITH RECURSIVE` query immediately inside another one (Tom)
- Fix concurrency bug in hash indexes (Tom)

  Concurrent insertions could cause index scans to transiently report wrong results.
- Fix incorrect logic for GiST index page splits, when the split depends on a non-first column of the index (Paul Ramsey)
- Fix wrong search results for a multi-column GIN index with `fastupdate` enabled (Teodor)
- Fix bugs in WAL entry creation for GIN indexes (Tom)

  These bugs were masked when `full_page_writes` was on, but with it off a WAL replay failure was certain if a crash occurred before the next checkpoint.
- Don't error out if recycling or removing an old WAL file fails at the end of checkpoint (Heikki)

  It's better to treat the problem as non-fatal and allow the checkpoint to complete. Future checkpoints will retry the removal. Such problems are not expected in normal operation, but have been seen to be caused by misdesigned Windows anti-virus and backup software.
- Ensure WAL files aren't repeatedly archived on Windows (Heikki)

  This is another symptom that could happen if some other process interfered with deletion of a no-longer-needed file.

- Fix PAM password processing to be more robust (Tom)

  The previous code is known to fail with the combination of the Linux `pam_krb5` PAM module with Microsoft Active Directory as the domain controller. It might have problems elsewhere too, since it was making unjustified assumptions about what arguments the PAM stack would pass to it.

- Raise the maximum authentication token (Kerberos ticket) size in GSSAPI and SSPI authentication methods (Ian Turner)

  While the old 2000-byte limit was more than enough for Unix Kerberos implementations, tickets issued by Windows Domain Controllers can be much larger.

- Ensure that domain constraints are enforced in constructs like `ARRAY[...]::domain`, where the domain is over an array type (Heikki)

- Fix foreign-key logic for some cases involving composite-type columns as foreign keys (Tom)

- Ensure that a cursor's snapshot is not modified after it is created (Alvaro)

  This could lead to a cursor delivering wrong results if later operations in the same transaction modify the data the cursor is supposed to return.

- Fix `CREATE TABLE` to properly merge default expressions coming from different inheritance parent tables (Tom)

  This used to work but was broken in 8.4.

- Re-enable collection of access statistics for sequences (Akira Kurosawa)

  This used to work but was broken in 8.3.

- Fix processing of ownership dependencies during `CREATE OR REPLACE FUNCTION` (Tom)

- Fix incorrect handling of `WHERE` *x*=*x* conditions (Tom)

  In some cases these could get ignored as redundant, but they aren't – they're equivalent to *x* `IS NOT NULL`.

- Fix incorrect plan construction when using hash aggregation to implement `DISTINCT` for textually identical volatile expressions (Tom)

- Fix Assert failure for a volatile `SELECT DISTINCT ON` expression (Tom)

- Fix `ts_stat()` to not fail on an empty `tsvector` value (Tom)

- Make text search parser accept underscores in XML attributes (Peter)

- Fix encoding handling in `xml` binary input (Heikki)

  If the XML header doesn't specify an encoding, we now assume UTF-8 by default; the previous handling was inconsistent.

- Fix bug with calling `plperl` from `plperlu` or vice versa (Tom)

  An error exit from the inner function could result in crashes due to failure to re-select the correct Perl interpreter for the outer function.

- Fix session-lifespan memory leak when a PL/Perl function is redefined (Tom)

- Ensure that Perl arrays are properly converted to PostgreSQL arrays when returned by a set-returning PL/Perl function (Andrew Dunstan, Abhijit Menon-Sen)

  This worked correctly already for non-set-returning functions.

- Fix rare crash in exception processing in PL/Python (Peter)

- Fix ecpg problem with comments in `DECLARE CURSOR` statements (Michael)

- Fix ecpg to not treat recently-added keywords as reserved words (Tom)

  This affected the keywords `CALLED`, `CATALOG`, `DEFINER`, `ENUM`, `FOLLOWING`, `INVOKER`, `OPTIONS`, `PARTITION`, `PRECEDING`, `RANGE`, `SECURITY`, `SERVER`, `UNBOUNDED`, and `WRAPPER`.

- Re-allow regular expression special characters in psql's `\df` function name parameter (Tom)

- In `contrib/fuzzystrmatch`, correct the calculation of `levenshtein` distances with non-default costs (Marcin Mank)

- In `contrib/pg_standby`, disable triggering failover with a signal on Windows (Fujii Masao)

  This never did anything useful, because Windows doesn't have Unix-style signals, but recent changes made it actually crash.

- Put `FREEZE` and `VERBOSE` options in the right order in the `VACUUM` command that `contrib/vacuumdb` produces (Heikki)

- Fix possible leak of connections when `contrib/dblink` encounters an error (Tatsuhito Kasahara)

- Ensure psql's flex module is compiled with the correct system header definitions (Tom)

  This fixes build failures on platforms where `--enable-largefile` causes incompatible changes in the generated code.

- Make the postmaster ignore any `application_name` parameter in connection request packets, to improve compatibility with future libpq versions (Tom)

- Update the timezone abbreviation files to match current reality (Joachim Wieland)

  This includes adding `IDT` to the default timezone abbreviation set.

- Update time zone data files to tzdata release 2009s for DST law changes in Antarctica, Argentina, Bangladesh, Fiji, Novokuznetsk, Pakistan, Palestine, Samoa, Syria; also historical corrections for Hong Kong.

E.96. Release 8.4.1

Release date: 2009-09-09

This release contains a variety of fixes from 8.4. For information about new features in the 8.4 major release, see 第 E.97 节.

E.96.1. Migration to Version 8.4.1

A dump/restore is not required for those running 8.4.X.

E.96.2. Changes

- Fix WAL page header initialization at the end of archive recovery (Heikki)

  This could lead to failure to process the WAL in a subsequent archive recovery.

- Fix "cannot make new WAL entries during recovery" error (Tom)

- Fix problem that could make expired rows visible after a crash (Tom)

  This bug involved a page status bit potentially not being set correctly after a server crash.

- Disallow `RESET ROLE` and `RESET SESSION AUTHORIZATION` inside security-definer functions (Tom, Heikki)

  This covers a case that was missed in the previous patch that disallowed `SET ROLE` and `SET SESSION AUTHORIZATION` inside security-definer functions. (See CVE-2007-6600)

- Make `LOAD` of an already-loaded loadable module into a no-op (Tom)

  Formerly, `LOAD` would attempt to unload and re-load the module, but this is unsafe and not all that useful.

- Make window function `PARTITION BY` and `ORDER BY` items always be interpreted as simple expressions (Tom)

  In 8.4.0 these lists were parsed following the rules used for top-level `GROUP BY` and `ORDER BY` lists. But this was not correct per the SQL standard, and it led to possible circularity.

- Fix several errors in planning of semi-joins (Tom)

  These led to wrong query results in some cases where `IN` or `EXISTS` was used together with another join.

- Fix handling of whole-row references to subqueries that are within an outer join (Tom)

  An example is `SELECT COUNT(ss.*) FROM ... LEFT JOIN (SELECT ...) ss ON ...`. Here, `ss.*` would be treated as `ROW(NULL,NULL,...)` for null-extended join rows, which is not the same as a simple NULL. Now it is treated as a simple NULL.

- Fix Windows shared-memory allocation code (Tsutomu Yamada, Magnus)

  This bug led to the often-reported "could not reattach to shared memory" error message.

- Fix locale handling with plperl (Heikki)

  This bug could cause the server's locale setting to change when a plperl function is called, leading to data corruption.

- Fix handling of reloptions to ensure setting one option doesn't force default values for others (Itagaki Takahiro)

- Ensure that a "fast shutdown" request will forcibly terminate open sessions, even if a "smart shutdown" was already in progress (Fujii Masao)

- Avoid memory leak for `array_agg()` in `GROUP BY` queries (Tom)

- Treat `to_char(..., 'TH')` as an uppercase ordinal suffix with `'HH'/'HH12'` (Heikki)

  It was previously handled as `'th'` (lowercase).

- Include the fractional part in the result of `EXTRACT(second)` and `EXTRACT(milliseconds)` for `time` and `time with time zone` inputs (Tom)

  This has always worked for floating-point datetime configurations, but was broken in the integer datetime code.

- Fix overflow for `INTERVAL 'x ms'` when *x* is more than 2 million and integer datetimes are in use (Alex Hunsaker)

- Improve performance when processing toasted values in index scans (Tom)

  This is particularly useful for PostGIS[5].

- Fix a typo that disabled `commit_delay` (Jeff Janes)

- Output early-startup messages to `postmaster.log` if the server is started in silent mode (Tom)

  Previously such error messages were discarded, leading to difficulty in debugging.

- Remove translated FAQs (Peter)

  They are now on the wiki[6]. The main FAQ was moved to the wiki some time ago.

- Fix pg_ctl to not go into an infinite loop if `postgresql.conf` is empty (Jeff Davis)

- Fix several errors in pg_dump's `--binary-upgrade` mode (Bruce, Tom)

  `pg_dump --binary-upgrade` is used by pg_migrator.

- Fix `contrib/xml2`'s `xslt_process()` to properly handle the maximum number of parameters (twenty) (Tom)

- Improve robustness of libpq's code to recover from errors during `COPY FROM STDIN` (Tom)

- Avoid including conflicting readline and editline header files when both libraries are installed (Zdenek Kotala)

- Work around gcc bug that causes "floating-point exception" instead of "division by zero" on some platforms (Tom)

- Update time zone data files to tzdata release 2009l for DST law changes in Bangladesh, Egypt, Mauritius.

E.97. Release 8.4

Release date: 2009-07-01

E.97.1. Overview

After many years of development, PostgreSQL has become feature-complete in many areas. This release shows a targeted approach to adding features (e.g.,

[5] http://postgis.net/
[6] http://wiki.postgresql.org/wiki/FAQ

authentication, monitoring, space reuse), and adds capabilities defined in the later SQL standards. The major areas of enhancement are:

- Windowing Functions
- Common Table Expressions and Recursive Queries
- Default and variadic parameters for functions
- Parallel Restore
- Column Permissions
- Per-database locale settings
- Improved hash indexes
- Improved join performance for `EXISTS` and `NOT EXISTS` queries
- Easier-to-use Warm Standby
- Automatic sizing of the Free Space Map
- Visibility Map (greatly reduces vacuum overhead for slowly-changing tables)
- Version-aware psql (backslash commands work against older servers)
- Support SSL certificates for user authentication
- Per-function runtime statistics
- Easy editing of functions in psql
- New contrib modules: pg_stat_statements, auto_explain, citext, btree_gin

The above items are explained in more detail in the sections below.

E.97.2. Migration to Version 8.4

A dump/restore using pg_dump is required for those wishing to migrate data from any previous release.

Observe the following incompatibilities:

E.97.2.1. General

- Use 64-bit integer datetimes by default (Neil Conway)

  Previously this was selected by configure's `--enable-integer-datetimes` option. To retain the old behavior, build with `--disable-integer-datetimes`.

- Remove ipcclean utility command (Bruce)

  The utility only worked on a few platforms. Users should use their operating system tools instead.

E.97.2.2. Server Settings

- Change default setting for `log_min_messages` to `warning` (previously it was `notice`) to reduce log file volume (Tom)
- Change default setting for `max_prepared_transactions` to zero (previously it was 5) (Tom)

- Make `debug_print_parse`, `debug_print_rewritten`, and `debug_print_plan` output appear at `LOG` message level, not `DEBUG1` as formerly (Tom)
- Make `debug_pretty_print` default to `on` (Tom)
- Remove `explain_pretty_print` parameter (no longer needed) (Tom)
- Make `log_temp_files` settable by superusers only, like other logging options (Simon Riggs)
- Remove automatic appending of the epoch timestamp when no `%` escapes are present in `log_filename` (Robert Haas)

  This change was made because some users wanted a fixed log filename, for use with an external log rotation tool.
- Remove `log_restartpoints` from `recovery.conf`; instead use `log_checkpoints` (Simon)
- Remove `krb_realm` and `krb_server_hostname`; these are now set in `pg_hba.conf` instead (Magnus)
- There are also significant changes in `pg_hba.conf`, as described below.

E.97.2.3. Queries

- Change `TRUNCATE` and `LOCK` to apply to child tables of the specified table(s) (Peter)

  These commands now accept an `ONLY` option that prevents processing child tables; this option must be used if the old behavior is needed.
- `SELECT DISTINCT` and `UNION/INTERSECT/EXCEPT` no longer always produce sorted output (Tom)

  Previously, these types of queries always removed duplicate rows by means of Sort/Unique processing (i.e., sort then remove adjacent duplicates). Now they can be implemented by hashing, which will not produce sorted output. If an application relied on the output being in sorted order, the recommended fix is to add an `ORDER BY` clause. As a short-term workaround, the previous behavior can be restored by disabling `enable_hashagg`, but that is a very performance-expensive fix. `SELECT DISTINCT ON` never uses hashing, however, so its behavior is unchanged.
- Force child tables to inherit `CHECK` constraints from parents (Alex Hunsaker, Nikhil Sontakke, Tom)

  Formerly it was possible to drop such a constraint from a child table, allowing rows that violate the constraint to be visible when scanning the parent table. This was deemed inconsistent, as well as contrary to SQL standard.
- Disallow negative `LIMIT` or `OFFSET` values, rather than treating them as zero (Simon)
- Disallow `LOCK TABLE` outside a transaction block (Tom)

  Such an operation is useless because the lock would be released immediately.
- Sequences now contain an additional `start_value` column (Zoltan Boszormenyi)

  This supports `ALTER SEQUENCE ... RESTART`.

E.97.2.4. Functions and Operators

- Make `numeric` zero raised to a fractional power return `0`, rather than throwing an error, and make `numeric` zero raised to the zero power return `1`, rather than error (Bruce)

  This matches the longstanding `float8` behavior.

- Allow unary minus of floating-point values to produce minus zero (Tom)

  The changed behavior is more IEEE-standard compliant.

- Throw an error if an escape character is the last character in a `LIKE` pattern (i.e., it has nothing to escape) (Tom)

  Previously, such an escape character was silently ignored, thus possibly masking application logic errors.

- Remove `~=~` and `~<>~` operators formerly used for `LIKE` index comparisons (Tom)

  Pattern indexes now use the regular equality operator.

- `xpath()` now passes its arguments to libxml without any changes (Andrew)

  This means that the XML argument must be a well-formed XML document. The previous coding attempted to allow XML fragments, but it did not work well.

- Make `xmlelement()` format attribute values just like content values (Peter)

  Previously, attribute values were formatted according to the normal SQL output behavior, which is sometimes at odds with XML rules.

- Rewrite memory management for libxml-using functions (Tom)

  This change should avoid some compatibility problems with use of libxml in PL/Perl and other add-on code.

- Adopt a faster algorithm for hash functions (Kenneth Marshall, based on work of Bob Jenkins)

  Many of the built-in hash functions now deliver different results on little-endian and big-endian platforms.

E.97.2.4.1. Temporal Functions and Operators

- `DateStyle` no longer controls `interval` output formatting; instead there is a new variable `IntervalStyle` (Ron Mayer)

- Improve consistency of handling of fractional seconds in `timestamp` and `interval` output (Ron Mayer)

  This may result in displaying a different number of fractional digits than before, or rounding instead of truncating.

- Make `to_char()`'s localized month/day names depend on `LC_TIME`, not `LC_MESSAGES` (Euler Taveira de Oliveira)

- Cause `to_date()` and `to_timestamp()` to more consistently report errors for invalid input (Brendan Jurd)

  Previous versions would often ignore or silently misread input that did not match the format string. Such cases will now result in an error.

- Fix `to_timestamp()` to not require upper/lower case matching for meridian (`AM`/`PM`) and era (`BC`/`AD`) format designations (Brendan Jurd)

  For example, input value `ad` now matches the format string `AD`.

E.97.3. Changes

Below you will find a detailed account of the changes between PostgreSQL 8.4 and the previous major release.

E.97.3.1. Performance

- Improve optimizer statistics calculations (Jan Urbanski, Tom)

  In particular, estimates for full-text-search operators are greatly improved.

- Allow `SELECT DISTINCT` and `UNION`/`INTERSECT`/`EXCEPT` to use hashing (Tom)

  This means that these types of queries no longer automatically produce sorted output.

- Create explicit concepts of semi-joins and anti-joins (Tom)

  This work formalizes our previous ad-hoc treatment of `IN (SELECT ...)` clauses, and extends it to `EXISTS` and `NOT EXISTS` clauses. It should result in significantly better planning of `EXISTS` and `NOT EXISTS` queries. In general, logically equivalent `IN` and `EXISTS` clauses should now have similar performance, whereas previously `IN` often won.

- Improve optimization of sub-selects beneath outer joins (Tom)

  Formerly, a sub-select or view could not be optimized very well if it appeared within the nullable side of an outer join and contained non-strict expressions (for instance, constants) in its result list.

- Improve the performance of `text_position()` and related functions by using Boyer-Moore-Horspool searching (David Rowley)

  This is particularly helpful for long search patterns.

- Reduce I/O load of writing the statistics collection file by writing the file only when requested (Martin Pihlak)

- Improve performance for bulk inserts (Robert Haas, Simon)

- Increase the default value of `default_statistics_target` from `10` to `100` (Greg Sabino Mullane, Tom)

  The maximum value was also increased from `1000` to `10000`.

- Perform `constraint_exclusion` checking by default in queries involving inheritance or `UNION ALL` (Tom)

  A new `constraint_exclusion` setting, `partition`, was added to specify this behavior.

- Allow I/O read-ahead for bitmap index scans (Greg Stark)

  The amount of read-ahead is controlled by `effective_io_concurrency`. This feature is available only if the kernel has `posix_fadvise()` support.

- Inline simple set-returning SQL functions in `FROM` clauses (Richard Rowell)
- Improve performance of multi-batch hash joins by providing a special case for join key values that are especially common in the outer relation (Bryce Cutt, Ramon Lawrence)
- Reduce volume of temporary data in multi-batch hash joins by suppressing "physical tlist" optimization (Michael Henderson, Ramon Lawrence)
- Avoid waiting for idle-in-transaction sessions during `CREATE INDEX CONCURRENTLY` (Simon)
- Improve performance of shared cache invalidation (Tom)

E.97.3.2. Server

E.97.3.2.1. Settings

- Convert many `postgresql.conf` settings to enumerated values so that `pg_settings` can display the valid values (Magnus)
- Add `cursor_tuple_fraction` parameter to control the fraction of a cursor's rows that the planner assumes will be fetched (Robert Hell)
- Allow underscores in the names of custom variable classes in `postgresql.conf` (Tom)

E.97.3.2.2. Authentication and security

- Remove support for the (insecure) `crypt` authentication method (Magnus)

  This effectively obsoletes pre-PostgreSQL 7.2 client libraries, as there is no longer any non-plaintext password method that they can use.
- Support regular expressions in `pg_ident.conf` (Magnus)
- Allow Kerberos/GSSAPI parameters to be changed without restarting the postmaster (Magnus)
- Support SSL certificate chains in server certificate file (Andrew Gierth)

  Including the full certificate chain makes the client able to verify the certificate without having all intermediate CA certificates present in the local store, which is often the case for commercial CAs.
- Report appropriate error message for combination of `MD5` authentication and `db_user_namespace` enabled (Bruce)

E.97.3.2.3. `pg_hba.conf`

- Change all authentication options to use `name=value` syntax (Magnus)

  This makes incompatible changes to the `ldap`, `pam` and `ident` authentication methods. All `pg_hba.conf` entries with these methods need to be rewritten using the new format.
- Remove the `ident sameuser` option, instead making that behavior the default if no usermap is specified (Magnus)
- Allow a usermap parameter for all external authentication methods (Magnus)

  Previously a usermap was only supported for `ident` authentication.

- Add `clientcert` option to control requesting of a client certificate (Magnus)

  Previously this was controlled by the presence of a root certificate file in the server's data directory.

- Add `cert` authentication method to allow user authentication via SSL certificates (Magnus)

  Previously SSL certificates could only verify that the client had access to a certificate, not authenticate a user.

- Allow `krb5`, `gssapi` and `sspi` realm and `krb5` host settings to be specified in `pg_hba.conf` (Magnus)

  These override the settings in `postgresql.conf`.

- Add `include_realm` parameter for `krb5`, `gssapi`, and `sspi` methods (Magnus)

  This allows identical usernames from different realms to be authenticated as different database users using usermaps.

- Parse `pg_hba.conf` fully when it is loaded, so that errors are reported immediately (Magnus)

  Previously, most errors in the file wouldn't be detected until clients tried to connect, so an erroneous file could render the system unusable. With the new behavior, if an error is detected during reload then the bad file is rejected and the postmaster continues to use its old copy.

- Show all parsing errors in `pg_hba.conf` instead of aborting after the first one (Selena Deckelmann)

- Support `ident` authentication over Unix-domain sockets on Solaris (Garick Hamlin)

E.97.3.2.4. Continuous Archiving

- Provide an option to `pg_start_backup()` to force its implied checkpoint to finish as quickly as possible (Tom)

  The default behavior avoids excess I/O consumption, but that is pointless if no concurrent query activity is going on.

- Make `pg_stop_backup()` wait for modified WAL files to be archived (Simon)

  This guarantees that the backup is valid at the time `pg_stop_backup()` completes.

- When archiving is enabled, rotate the last WAL segment at shutdown so that all transactions can be archived immediately (Guillaume Smet, Heikki)

- Delay “smart” shutdown while a continuous archiving base backup is in progress (Laurenz Albe)

- Cancel a continuous archiving base backup if “fast” shutdown is requested (Laurenz Albe)

- Allow `recovery.conf` boolean variables to take the same range of string values as `postgresql.conf` boolean variables (Bruce)

E.97.3.2.5. Monitoring

- Add `pg_conf_load_time()` to report when the PostgreSQL configuration files were last loaded (George Gensure)

- Add `pg_terminate_backend()` to safely terminate a backend (the `SIGTERM` signal works also) (Tom, Bruce)

  While it's always been possible to `SIGTERM` a single backend, this was previously considered unsupported; and testing of the case found some bugs that are now fixed.

- Add ability to track user-defined functions' call counts and runtimes (Martin Pihlak)

  Function statistics appear in a new system view, `pg_stat_user_functions`. Tracking is controlled by the new parameter `track_functions`.

- Allow specification of the maximum query string size in `pg_stat_activity` via new `track_activity_query_size` parameter (Thomas Lee)

- Increase the maximum line length sent to syslog, in hopes of improving performance (Tom)

- Add read-only configuration variables `segment_size`, `wal_block_size`, and `wal_segment_size` (Bernd Helmle)

- When reporting a deadlock, report the text of all queries involved in the deadlock to the server log (Itagaki Takahiro)

- Add `pg_stat_get_activity(pid)` function to return information about a specific process id (Magnus)

- Allow the location of the server's statistics file to be specified via `stats_temp_directory` (Magnus)

  This allows the statistics file to be placed in a RAM-resident directory to reduce I/O requirements. On startup/shutdown, the file is copied to its traditional location (`$PGDATA/global/`) so it is preserved across restarts.

E.97.3.3. Queries

- Add support for `WINDOW` functions (Hitoshi Harada)

- Add support for `WITH` clauses (CTEs), including `WITH RECURSIVE` (Yoshiyuki Asaba, Tatsuo Ishii, Tom)

- Add `TABLE` command (Peter)

  `TABLE tablename` is a SQL standard short-hand for `SELECT * FROM tablename`.

- Allow `AS` to be optional when specifying a `SELECT` (or `RETURNING`) column output label (Hiroshi Saito)

  This works so long as the column label is not any PostgreSQL keyword; otherwise `AS` is still needed.

- Support set-returning functions in `SELECT` result lists even for functions that return their result via a tuplestore (Tom)

  In particular, this means that functions written in PL/pgSQL and other PL languages can now be called this way.

- Support set-returning functions in the output of aggregation and grouping queries (Tom)

- Allow `SELECT FOR UPDATE/SHARE` to work on inheritance trees (Tom)

- Add infrastructure for SQL/MED (Martin Pihlak, Peter)

  There are no remote or external SQL/MED capabilities yet, but this change provides a standardized and future-proof system for managing connection information for modules like `dblink` and `plproxy`.

- Invalidate cached plans when referenced schemas, functions, operators, or operator classes are modified (Martin Pihlak, Tom)

  This improves the system's ability to respond to on-the-fly DDL changes.

- Allow comparison of composite types and allow arrays of anonymous composite types (Tom)

  This allows constructs such as `row(1, 1.1) = any (array[row(7, 7.7), row(1, 1.0)])`. This is particularly useful in recursive queries.

- Add support for Unicode string literal and identifier specifications using code points, e.g. `U&'d\0061t\+000061'` (Peter)

- Reject `\000` in string literals and `COPY` data (Tom)

  Previously, this was accepted but had the effect of terminating the string contents.

- Improve the parser's ability to report error locations (Tom)

  An error location is now reported for many semantic errors, such as mismatched datatypes, that previously could not be localized.

E.97.3.3.1. TRUNCATE

- Support statement-level `ON TRUNCATE` triggers (Simon)

- Add `RESTART/CONTINUE IDENTITY` options for `TRUNCATE TABLE` (Zoltan Boszormenyi)

  The start value of a sequence can be changed by `ALTER SEQUENCE START WITH`.

- Allow `TRUNCATE tab1, tab1` to succeed (Bruce)

- Add a separate `TRUNCATE` permission (Robert Haas)

E.97.3.3.2. EXPLAIN

- Make `EXPLAIN VERBOSE` show the output columns of each plan node (Tom)

  Previously `EXPLAIN VERBOSE` output an internal representation of the query plan. (That behavior is now available via `debug_print_plan`.)

- Make `EXPLAIN` identify subplans and initplans with individual labels (Tom)

- Make `EXPLAIN` honor `debug_print_plan` (Tom)

- Allow `EXPLAIN` on `CREATE TABLE AS` (Peter)

E.97.3.3.3. LIMIT/OFFSET

- Allow sub-selects in `LIMIT` and `OFFSET` (Tom)

- Add SQL-standard syntax for `LIMIT/OFFSET` capabilities (Peter)

  To wit, `OFFSET num {ROW|ROWS} FETCH {FIRST|NEXT} [num] {ROW|ROWS} ONLY`.

E.97.3.4. Object Manipulation

- Add support for column-level privileges (Stephen Frost, KaiGai Kohei)
- Refactor multi-object `DROP` operations to reduce the need for `CASCADE` (Alex Hunsaker)

  For example, if table `B` has a dependency on table `A`, the command `DROP TABLE A, B` no longer requires the `CASCADE` option.
- Fix various problems with concurrent `DROP` commands by ensuring that locks are taken before we begin to drop dependencies of an object (Tom)
- Improve reporting of dependencies during `DROP` commands (Tom)
- Add `WITH [NO] DATA` clause to `CREATE TABLE AS`, per the SQL standard (Peter, Tom)
- Add support for user-defined I/O conversion casts (Heikki)
- Allow `CREATE AGGREGATE` to use an `internal` transition datatype (Tom)
- Add `LIKE` clause to `CREATE TYPE` (Tom)

  This simplifies creation of data types that use the same internal representation as an existing type.
- Allow specification of the type category and "preferred" status for user-defined base types (Tom)

  This allows more control over the coercion behavior of user-defined types.
- Allow `CREATE OR REPLACE VIEW` to add columns to the end of a view (Robert Haas)

E.97.3.4.1. `ALTER`

- Add `ALTER TYPE RENAME` (Petr Jelinek)
- Add `ALTER SEQUENCE ... RESTART` (with no parameter) to reset a sequence to its initial value (Zoltan Boszormenyi)
- Modify the `ALTER TABLE` syntax to allow all reasonable combinations for tables, indexes, sequences, and views (Tom)

  This change allows the following new syntaxes:

  - `ALTER SEQUENCE OWNER TO`
  - `ALTER VIEW ALTER COLUMN SET/DROP DEFAULT`
  - `ALTER VIEW OWNER TO`
  - `ALTER VIEW SET SCHEMA`

  There is no actual new functionality here, but formerly you had to say `ALTER TABLE` to do these things, which was confusing.
- Add support for the syntax `ALTER TABLE ... ALTER COLUMN ... SET DATA TYPE` (Peter)

  This is SQL-standard syntax for functionality that was already supported.
- Make `ALTER TABLE SET WITHOUT OIDS` rewrite the table to physically remove `OID` values (Tom)

Also, add `ALTER TABLE SET WITH OIDS` to rewrite the table to add `OID`s.

E.97.3.4.2. Database Manipulation

- Improve reporting of `CREATE/DROP/RENAME DATABASE` failure when uncommitted prepared transactions are the cause (Tom)
- Make `LC_COLLATE` and `LC_CTYPE` into per-database settings (Radek Strnad, Heikki)

  This makes collation similar to encoding, which was always configurable per database.
- Improve checks that the database encoding, collation (`LC_COLLATE`), and character classes (`LC_CTYPE`) match (Heikki, Tom)

  Note in particular that a new database's encoding and locale settings can be changed only when copying from `template0`. This prevents possibly copying data that doesn't match the settings.
- Add `ALTER DATABASE SET TABLESPACE` to move a database to a new tablespace (Guillaume Lelarge, Bernd Helmle)

E.97.3.5. Utility Operations

- Add a `VERBOSE` option to the `CLUSTER` command and clusterdb (Jim Cox)
- Decrease memory requirements for recording pending trigger events (Tom)

E.97.3.5.1. Indexes

- Dramatically improve the speed of building and accessing hash indexes (Tom Raney, Shreya Bhargava)

  This allows hash indexes to be sometimes faster than btree indexes. However, hash indexes are still not crash-safe.
- Make hash indexes store only the hash code, not the full value of the indexed column (Xiao Meng)

  This greatly reduces the size of hash indexes for long indexed values, improving performance.
- Implement fast update option for GIN indexes (Teodor, Oleg)

  This option greatly improves update speed at a small penalty in search speed.
- `xxx_pattern_ops` indexes can now be used for simple equality comparisons, not only for `LIKE` (Tom)

E.97.3.5.2. Full Text Indexes

- Remove the requirement to use `@@@` when doing GIN weighted lookups on full text indexes (Tom, Teodor)

  The normal `@@` text search operator can be used instead.
- Add an optimizer selectivity function for `@@` text search operations (Jan Urbanski)
- Allow prefix matching in full text searches (Teodor Sigaev, Oleg Bartunov)
- Support multi-column GIN indexes (Teodor Sigaev)

- Improve support for Nepali language and Devanagari alphabet (Teodor)

E.97.3.5.3. VACUUM

- Track free space in separate per-relation "fork" files (Heikki)

  Free space discovered by `VACUUM` is now recorded in `*_fsm` files, rather than in a fixed-sized shared memory area. The `max_fsm_pages` and `max_fsm_relations` settings have been removed, greatly simplifying administration of free space management.

- Add a visibility map to track pages that do not require vacuuming (Heikki)

  This allows `VACUUM` to avoid scanning all of a table when only a portion of the table needs vacuuming. The visibility map is stored in per-relation "fork" files.

- Add `vacuum_freeze_table_age` parameter to control when `VACUUM` should ignore the visibility map and do a full table scan to freeze tuples (Heikki)

- Track transaction snapshots more carefully (Alvaro)

  This improves `VACUUM`'s ability to reclaim space in the presence of long-running transactions.

- Add ability to specify per-relation autovacuum and TOAST parameters in `CREATE TABLE` (Alvaro, Euler Taveira de Oliveira)

  Autovacuum options used to be stored in a system table.

- Add `--freeze` option to vacuumdb (Bruce)

E.97.3.6. Data Types

- Add a `CaseSensitive` option for text search synonym dictionaries (Simon)

- Improve the precision of `NUMERIC` division (Tom)

- Add basic arithmetic operators for `int2` with `int8` (Tom)

  This eliminates the need for explicit casting in some situations.

- Allow `UUID` input to accept an optional hyphen after every fourth digit (Robert Haas)

- Allow `on`/`off` as input for the boolean data type (Itagaki Takahiro)

- Allow spaces around `NaN` in the input string for type `numeric` (Sam Mason)

E.97.3.6.1. Temporal Data Types

- Reject year `0 BC` and years `000` and `0000` (Tom)

  Previously these were interpreted as `1 BC`. (Note: years `0` and `00` are still assumed to be the year 2000.)

- Include `SGT` (Singapore time) in the default list of known time zone abbreviations (Tom)

- Support `infinity` and `-infinity` as values of type `date` (Tom)

- Make parsing of `interval` literals more standard-compliant (Tom, Ron Mayer)

  For example, `INTERVAL '1' YEAR` now does what it's supposed to.

- Allow `interval` fractional-seconds precision to be specified after the `second` keyword, for SQL standard compliance (Tom)

  Formerly the precision had to be specified after the keyword `interval`. (For backwards compatibility, this syntax is still supported, though deprecated.) Data type definitions will now be output using the standard format.

- Support the IS0 8601 `interval` syntax (Ron Mayer, Kevin Grittner)

  For example, `INTERVAL 'P1Y2M3DT4H5M6.7S'` is now supported.

- Add `IntervalStyle` parameter which controls how `interval` values are output (Ron Mayer)

  Valid values are: `postgres`, `postgres_verbose`, `sql_standard`, `iso_8601`. This setting also controls the handling of negative `interval` input when only some fields have positive/negative designations.

- Improve consistency of handling of fractional seconds in `timestamp` and `interval` output (Ron Mayer)

E.97.3.6.2. Arrays

- Improve the handling of casts applied to `ARRAY[]` constructs, such as `ARRAY[...]::integer[]` (Brendan Jurd)

  Formerly PostgreSQL attempted to determine a data type for the `ARRAY[]` construct without reference to the ensuing cast. This could fail unnecessarily in many cases, in particular when the `ARRAY[]` construct was empty or contained only ambiguous entries such as `NULL`. Now the cast is consulted to determine the type that the array elements must be.

- Make SQL-syntax `ARRAY` dimensions optional to match the SQL standard (Peter)

- Add `array_ndims()` to return the number of dimensions of an array (Robert Haas)

- Add `array_length()` to return the length of an array for a specified dimension (Jim Nasby, Robert Haas, Peter Eisentraut)

- Add aggregate function `array_agg()`, which returns all aggregated values as a single array (Robert Haas, Jeff Davis, Peter)

- Add `unnest()`, which converts an array to individual row values (Tom)

  This is the opposite of `array_agg()`.

- Add `array_fill()` to create arrays initialized with a value (Pavel Stehule)

- Add `generate_subscripts()` to simplify generating the range of an array's subscripts (Pavel Stehule)

E.97.3.6.3. Wide-Value Storage (TOAST)

- Consider TOAST compression on values as short as 32 bytes (previously 256 bytes) (Greg Stark)

- Require 25% minimum space savings before using TOAST compression (previously 20% for small values and any-savings-at-all for large values) (Greg)

- Improve TOAST heuristics for rows that have a mix of large and small toastable fields, so that we prefer to push large values out of line and don't compress small values unnecessarily (Greg, Tom)

E.97.3.7. Functions

- Document that `setseed()` allows values from `-1` to `1` (not just `0` to `1`), and enforce the valid range (Kris Jurka)
- Add server-side function `lo_import(filename, oid)` (Tatsuo)
- Add `quote_nullable()`, which behaves like `quote_literal()` but returns the string `NULL` for a null argument (Brendan Jurd)
- Improve full text search `headline()` function to allow extracting several fragments of text (Sushant Sinha)
- Add `suppress_redundant_updates_trigger()` trigger function to avoid overhead for non-data-changing updates (Andrew)
- Add `div(numeric, numeric)` to perform `numeric` division without rounding (Tom)
- Add `timestamp` and `timestamptz` versions of `generate_series()` (Hitoshi Harada)

E.97.3.7.1. Object Information Functions

- Implement `current_query()` for use by functions that need to know the currently running query (Tomas Doran)
- Add `pg_get_keywords()` to return a list of the parser keywords (Dave Page)
- Add `pg_get_functiondef()` to see a function's definition (Abhijit Menon-Sen)
- Allow the second argument of `pg_get_expr()` to be zero when deparsing an expression that does not contain variables (Tom)
- Modify `pg_relation_size()` to use `regclass` (Heikki)

  `pg_relation_size(data_type_name)` no longer works.
- Add `boot_val` and `reset_val` columns to `pg_settings` output (Greg Smith)
- Add source file name and line number columns to `pg_settings` output for variables set in a configuration file (Magnus, Alvaro)

  For security reasons, these columns are only visible to superusers.
- Add support for `CURRENT_CATALOG`, `CURRENT_SCHEMA`, `SET CATALOG`, `SET SCHEMA` (Peter)

  These provide SQL-standard syntax for existing features.
- Add `pg_typeof()` which returns the data type of any value (Brendan Jurd)
- Make `version()` return information about whether the server is a 32- or 64-bit binary (Bruce)
- Fix the behavior of information schema columns `is_insertable_into` and `is_updatable` to be consistent (Peter)
- Improve the behavior of information schema `datetime_precision` columns (Peter)

  These columns now show zero for `date` columns, and 6 (the default precision) for `time`, `timestamp`, and `interval` without a declared precision, rather than showing null as formerly.

- Convert remaining builtin set-returning functions to use `OUT` parameters (Jaime Casanova)

  This makes it possible to call these functions without specifying a column list: `pg_show_all_settings()`, `pg_lock_status()`, `pg_prepared_xact()`, `pg_prepared_statement()`, `pg_cursor()`

- Make `pg_*_is_visible()` and `has_*_privilege()` functions return `NULL` for invalid OIDs, rather than reporting an error (Tom)

- Extend `has_*_privilege()` functions to allow inquiring about the OR of multiple privileges in one call (Stephen Frost, Tom)

- Add `has_column_privilege()` and `has_any_column_privilege()` functions (Stephen Frost, Tom)

E.97.3.7.2. Function Creation

- Support variadic functions (functions with a variable number of arguments) (Pavel Stehule)

  Only trailing arguments can be optional, and they all must be of the same data type.

- Support default values for function arguments (Pavel Stehule)

- Add `CREATE FUNCTION ... RETURNS TABLE` clause (Pavel Stehule)

- Allow SQL-language functions to return the output of an `INSERT/UPDATE/DELETE RETURNING` clause (Tom)

E.97.3.7.3. PL/pgSQL Server-Side Language

- Support `EXECUTE USING` for easier insertion of data values into a dynamic query string (Pavel Stehule)

- Allow looping over the results of a cursor using a `FOR` loop (Pavel Stehule)

- Support `RETURN QUERY EXECUTE` (Pavel Stehule)

- Improve the `RAISE` command (Pavel Stehule)

  - Support `DETAIL` and `HINT` fields

  - Support specification of the `SQLSTATE` error code

  - Support an exception name parameter

  - Allow `RAISE` without parameters in an exception block to re-throw the current error

- Allow specification of `SQLSTATE` codes in `EXCEPTION` lists (Pavel Stehule)

  This is useful for handling custom `SQLSTATE` codes.

- Support the `CASE` statement (Pavel Stehule)

- Make `RETURN QUERY` set the special `FOUND` and `GET DIAGNOSTICS ROW_COUNT` variables (Pavel Stehule)

- Make `FETCH` and `MOVE` set the `GET DIAGNOSTICS ROW_COUNT` variable (Andrew Gierth)

- Make `EXIT` without a label always exit the innermost loop (Tom)

  Formerly, if there were a `BEGIN` block more closely nested than any loop, it would exit that block instead. The new behavior matches Oracle(TM) and is also what was previously stated by our own documentation.

- Make processing of string literals and nested block comments match the main SQL parser's processing (Tom)

  In particular, the format string in `RAISE` now works the same as any other string literal, including being subject to `standard_conforming_strings`. This change also fixes other cases in which valid commands would fail when `standard_conforming_strings` is on.

- Avoid memory leakage when the same function is called at varying exception-block nesting depths (Tom)

E.97.3.8. Client Applications

- Fix `pg_ctl restart` to preserve command-line arguments (Bruce)
- Add `-w`/`--no-password` option that prevents password prompting in all utilities that have a `-W`/`--password` option (Peter)
- Remove `-q` (quiet) option of createdb, createuser, dropdb, dropuser (Peter)

  These options have had no effect since PostgreSQL 8.3.

E.97.3.8.1. psql

- Remove verbose startup banner; now just suggest `help` (Joshua Drake)
- Make `help` show common backslash commands (Greg Sabino Mullane)
- Add `\pset format wrapped` mode to wrap output to the screen width, or file/pipe output too if `\pset columns` is set (Bryce Nesbitt)
- Allow all supported spellings of boolean values in `\pset`, rather than just `on` and `off` (Bruce)

  Formerly, any string other than "off" was silently taken to mean `true`. psql will now complain about unrecognized spellings (but still take them as `true`).

- Use the pager for wide output (Bruce)
- Require a space between a one-letter backslash command and its first argument (Bernd Helmle)

  This removes a historical source of ambiguity.

- Improve tab completion support for schema-qualified and quoted identifiers (Greg Sabino Mullane)
- Add optional `on`/`off` argument for `\timing` (David Fetter)
- Display access control rights on multiple lines (Brendan Jurd, Andreas Scherbaum)
- Make `\l` show database access privileges (Andrew Gilligan)
- Make `\l+` show database sizes, if permissions allow (Andrew Gilligan)
- Add the `\ef` command to edit function definitions (Abhijit Menon-Sen)

E.97.3.8.2. psql \d* commands

- Make `\d*` commands that do not have a pattern argument show system objects only if the `S` modifier is specified (Greg Sabino Mullane, Bruce)

  The former behavior was inconsistent across different variants of `\d`, and in most cases it provided no easy way to see just user objects.

- Improve `\d*` commands to work with older PostgreSQL server versions (back to 7.4), not only the current server version (Guillaume Lelarge)

- Make `\d` show foreign-key constraints that reference the selected table (Kenneth D'Souza)

- Make `\d` on a sequence show its column values (Euler Taveira de Oliveira)

- Add column storage type and other relation options to the `\d+` display (Gregory Stark, Euler Taveira de Oliveira)

- Show relation size in `\dt+` output (Dickson S. Guedes)

- Show the possible values of `enum` types in `\dT+` (David Fetter)

- Allow `\dC` to accept a wildcard pattern, which matches either datatype involved in the cast (Tom)

- Add a function type column to `\df`'s output, and add options to list only selected types of functions (David Fetter)

- Make `\df` not hide functions that take or return type `cstring` (Tom)

  Previously, such functions were hidden because most of them are datatype I/O functions, which were deemed uninteresting. The new policy about hiding system functions by default makes this wart unnecessary.

E.97.3.8.3. pg_dump

- Add a `--no-tablespaces` option to pg_dump/pg_dumpall/pg_restore so that dumps can be restored to clusters that have non-matching tablespace layouts (Gavin Roy)

- Remove `-d` and `-D` options from pg_dump and pg_dumpall (Tom)

  These options were too frequently confused with the option to select a database name in other PostgreSQL client applications. The functionality is still available, but you must now spell out the long option name `--inserts` or `--column-inserts`.

- Remove `-i`/`--ignore-version` option from pg_dump and pg_dumpall (Tom)

  Use of this option does not throw an error, but it has no effect. This option was removed because the version checks are necessary for safety.

- Disable `statement_timeout` during dump and restore (Joshua Drake)

- Add pg_dump/pg_dumpall option `--lock-wait-timeout` (David Gould)

  This allows dumps to fail if unable to acquire a shared lock within the specified amount of time.

- Reorder pg_dump `--data-only` output to dump tables referenced by foreign keys before the referencing tables (Tom)

This allows data loads when foreign keys are already present. If circular references make a safe ordering impossible, a `NOTICE` is issued.

- Allow pg_dump, pg_dumpall, and pg_restore to use a specified role (Benedek László)
- Allow pg_restore to use multiple concurrent connections to do the restore (Andrew)

  The number of concurrent connections is controlled by the option `--jobs`. This is supported only for custom-format archives.

E.97.3.9. Programming Tools

E.97.3.9.1. libpq

- Allow the `OID` to be specified when importing a large object, via new function `lo_import_with_oid()` (Tatsuo)
- Add "events" support (Andrew Chernow, Merlin Moncure)

  This adds the ability to register callbacks to manage private data associated with `PGconn` and `PGresult` objects.
- Improve error handling to allow the return of multiple error messages as multi-line error reports (Magnus)
- Make `PQexecParams()` and related functions return `PGRES_EMPTY_QUERY` for an empty query (Tom)

  They previously returned `PGRES_COMMAND_OK`.
- Document how to avoid the overhead of `WSACleanup()` on Windows (Andrew Chernow)
- Do not rely on Kerberos tickets to determine the default database username (Magnus)

  Previously, a Kerberos-capable build of libpq would use the principal name from any available Kerberos ticket as default database username, even if the connection wasn't using Kerberos authentication. This was deemed inconsistent and confusing. The default username is now determined the same way with or without Kerberos. Note however that the database username must still match the ticket when Kerberos authentication is used.

E.97.3.9.2. libpq SSL (Secure Sockets Layer) support

- Fix certificate validation for SSL connections (Magnus)

  libpq now supports verifying both the certificate and the name of the server when making SSL connections. If a root certificate is not available to use for verification, SSL connections will fail. The `sslmode` parameter is used to enable certificate verification and set the level of checking. The default is still not to do any verification, allowing connections to SSL-enabled servers without requiring a root certificate on the client.
- Support wildcard server certificates (Magnus)

  If a certificate CN starts with `*`, it will be treated as a wildcard when matching the hostname, allowing the use of the same certificate for multiple servers.

- Allow the file locations for client certificates to be specified (Mark Woodward, Alvaro, Magnus)
- Add a `PQinitOpenSSL` function to allow greater control over OpenSSL/libcrypto initialization (Andrew Chernow)
- Make libpq unregister its OpenSSL callbacks when no database connections remain open (Bruce, Magnus, Russell Smith)

  This is required for applications that unload the libpq library, otherwise invalid OpenSSL callbacks will remain.

E.97.3.9.3. ecpg

- Add localization support for messages (Euler Taveira de Oliveira)
- ecpg parser is now automatically generated from the server parser (Michael)

  Previously the ecpg parser was hand-maintained.

E.97.3.9.4. Server Programming Interface (SPI)

- Add support for single-use plans with out-of-line parameters (Tom)
- Add new `SPI_OK_REWRITTEN` return code for `SPI_execute()` (Heikki)

  This is used when a command is rewritten to another type of command.
- Remove unnecessary inclusions from `executor/spi.h` (Tom)

  SPI-using modules might need to add some `#include` lines if they were depending on `spi.h` to include things for them.

E.97.3.10. Build Options

- Update build system to use Autoconf 2.61 (Peter)
- Require GNU bison for source code builds (Peter)

  This has effectively been required for several years, but now there is no infrastructure claiming to support other parser tools.
- Add pg_config `--htmldir` option (Peter)
- Pass `float4` by value inside the server (Zoltan Boszormenyi)

  Add configure option `--disable-float4-byval` to use the old behavior. External C functions that use old-style (version 0) call convention and pass or return `float4` values will be broken by this change, so you may need the configure option if you have such functions and don't want to update them.
- Pass `float8`, `int8`, and related datatypes by value inside the server on 64-bit platforms (Zoltan Boszormenyi)

  Add configure option `--disable-float8-byval` to use the old behavior. As above, this change might break old-style external C functions.
- Add configure options `--with-segsize`, `--with-blocksize`, `--with-wal-blocksize`, `--with-wal-segsize` (Zdenek Kotala, Tom)

  This simplifies build-time control over several constants that previously could only be changed by editing `pg_config_manual.h`.

- Allow threaded builds on Solaris 2.5 (Bruce)
- Use the system's `getopt_long()` on Solaris (Zdenek Kotala, Tom)

  This makes option processing more consistent with what Solaris users expect.
- Add support for the Sun Studio compiler on Linux (Julius Stroffek)
- Append the major version number to the backend gettext domain, and the `soname` major version number to libraries' gettext domain (Peter)

  This simplifies parallel installations of multiple versions.
- Add support for code coverage testing with gcov (Michelle Caisse)
- Allow out-of-tree builds on Mingw and Cygwin (Richard Evans)
- Fix the use of Mingw as a cross-compiling source platform (Peter)

E.97.3.11. Source Code

- Support 64-bit time zone data files (Heikki)

  This adds support for daylight saving time (DST) calculations beyond the year 2038.
- Deprecate use of platform's `time_t` data type (Tom)

  Some platforms have migrated to 64-bit `time_t`, some have not, and Windows can't make up its mind what it's doing. Define `pg_time_t` to have the same meaning as `time_t`, but always be 64 bits (unless the platform has no 64-bit integer type), and use that type in all module APIs and on-disk data formats.
- Fix bug in handling of the time zone database when cross-compiling (Richard Evans)
- Link backend object files in one step, rather than in stages (Peter)
- Improve gettext support to allow better translation of plurals (Peter)
- Add message translation support to the PL languages (Alvaro, Peter)
- Add more DTrace probes (Robert Lor)
- Enable DTrace support on Mac OS X Leopard and other non-Solaris platforms (Robert Lor)
- Simplify and standardize conversions between C strings and `text` datums, by providing common functions for the purpose (Brendan Jurd, Tom)
- Clean up the `include/catalog/` header files so that frontend programs can include them without including `postgres.h` (Zdenek Kotala)
- Make `name` char-aligned, and suppress zero-padding of `name` entries in indexes (Tom)
- Recover better if dynamically-loaded code executes `exit()` (Tom)
- Add a hook to let plug-ins monitor the executor (Itagaki Takahiro)
- Add a hook to allow the planner's statistics lookup behavior to be overridden (Simon Riggs)
- Add `shmem_startup_hook()` for custom shared memory requirements (Tom)

- Replace the index access method `amgetmulti` entry point with `amgetbitmap`, and extend the API for `amgettuple` to support run-time determination of operator lossiness (Heikki, Tom, Teodor)

  The API for GIN and GiST opclass `consistent` functions has been extended as well.

- Add support for partial-match searches in GIN indexes (Teodor Sigaev, Oleg Bartunov)

- Replace `pg_class` column `reltriggers` with boolean `relhastriggers` (Simon)

  Also remove unused `pg_class` columns `relukeys`, `relfkeys`, and `relrefs`.

- Add a `relistemp` column to `pg_class` to ease identification of temporary tables (Tom)

- Move platform FAQs into the main documentation (Peter)

- Prevent parser input files from being built with any conflicts (Peter)

- Add support for the `KOI8U` (Ukrainian) encoding (Peter)

- Add Japanese message translations (Japan PostgreSQL Users Group)

  This used to be maintained as a separate project.

- Fix problem when setting `LC_MESSAGES` on MSVC-built systems (Hiroshi Inoue, Hiroshi Saito, Magnus)

E.97.3.12. Contrib

- Add `contrib/auto_explain` to automatically run `EXPLAIN` on queries exceeding a specified duration (Itagaki Takahiro, Tom)

- Add `contrib/btree_gin` to allow GIN indexes to handle more datatypes (Oleg, Teodor)

- Add `contrib/citext` to provide a case-insensitive, multibyte-aware text data type (David Wheeler)

- Add `contrib/pg_stat_statements` for server-wide tracking of statement execution statistics (Itagaki Takahiro)

- Add duration and query mode options to `contrib/pgbench` (Itagaki Takahiro)

- Make `contrib/pgbench` use table names `pgbench_accounts`, `pgbench_branches`, `pgbench_history`, and `pgbench_tellers`, rather than just `accounts`, `branches`, `history`, and `tellers` (Tom)

  This is to reduce the risk of accidentally destroying real data by running pgbench.

- Fix `contrib/pgstattuple` to handle tables and indexes with over 2 billion pages (Tatsuhito Kasahara)

- In `contrib/fuzzystrmatch`, add a version of the Levenshtein string-distance function that allows the user to specify the costs of insertion, deletion, and substitution (Volkan Yazici)

- Make `contrib/ltree` support multibyte encodings (laser)

- Enable `contrib/dblink` to use connection information stored in the SQL/MED catalogs (Joe Conway)

- Improve `contrib/dblink`'s reporting of errors from the remote server (Joe Conway)
- Make `contrib/dblink` set `client_encoding` to match the local database's encoding (Joe Conway)

  This prevents encoding problems when communicating with a remote database that uses a different encoding.
- Make sure `contrib/dblink` uses a password supplied by the user, and not accidentally taken from the server's `.pgpass` file (Joe Conway)

  This is a minor security enhancement.
- Add `fsm_page_contents()` to `contrib/pageinspect` (Heikki)
- Modify `get_raw_page()` to support free space map (`*_fsm`) files. Also update `contrib/pg_freespacemap`.
- Add support for multibyte encodings to `contrib/pg_trgm` (Teodor)
- Rewrite `contrib/intagg` to use new functions `array_agg()` and `unnest()` (Tom)
- Make `contrib/pg_standby` recover all available WAL before failover (Fujii Masao, Simon, Heikki)

  To make this work safely, you now need to set the new `recovery_end_command` option in `recovery.conf` to clean up the trigger file after failover. pg_standby will no longer remove the trigger file itself.
- `contrib/pg_standby`'s `-l` option is now a no-op, because it is unsafe to use a symlink (Simon)

E.98. Release 8.3.23

Release date: 2013-02-07

This release contains a variety of fixes from 8.3.22. For information about new features in the 8.3 major release, see 第 E.121 节.

This is expected to be the last PostgreSQL release in the 8.3.X series. Users are encouraged to update to a newer release branch soon.

E.98.1. Migration to Version 8.3.23

A dump/restore is not required for those running 8.3.X.

However, if you are upgrading from a version earlier than 8.3.17, see 第 E.104 节.

E.98.2. Changes

- Prevent execution of `enum_recv` from SQL (Tom Lane)

  The function was misdeclared, allowing a simple SQL command to crash the server. In principle an attacker might be able to use it to examine the contents of server memory. Our thanks to Sumit Soni (via Secunia SVCRP) for reporting this issue. (CVE-2013-0255)
- Fix SQL grammar to allow subscripting or field selection from a sub-SELECT result (Tom Lane)

- Protect against race conditions when scanning `pg_tablespace` (Stephen Frost, Tom Lane)

  `CREATE DATABASE` and `DROP DATABASE` could misbehave if there were concurrent updates of `pg_tablespace` entries.

- Prevent `DROP OWNED` from trying to drop whole databases or tablespaces (Álvaro Herrera)

  For safety, ownership of these objects must be reassigned, not dropped.

- Prevent misbehavior when a `RowExpr` or `XmlExpr` is parse-analyzed twice (Andres Freund, Tom Lane)

  This mistake could be user-visible in contexts such as `CREATE TABLE LIKE INCLUDING INDEXES`.

- Improve defenses against integer overflow in hashtable sizing calculations (Jeff Davis)

- Ensure that non-ASCII prompt strings are translated to the correct code page on Windows (Alexander Law, Noah Misch)

  This bug affected psql and some other client programs.

- Fix possible crash in psql's `\?` command when not connected to a database (Meng Qingzhong)

- Fix one-byte buffer overrun in libpq's `PQprintTuples` (Xi Wang)

  This ancient function is not used anywhere by PostgreSQL itself, but it might still be used by some client code.

- Rearrange configure's tests for supplied functions so it is not fooled by bogus exports from libedit/libreadline (Christoph Berg)

- Ensure Windows build number increases over time (Magnus Hagander)

- Make pgxs build executables with the right `.exe` suffix when cross-compiling for Windows (Zoltan Boszormenyi)

- Add new timezone abbreviation `FET` (Tom Lane)

  This is now used in some eastern-European time zones.

E.99. Release 8.3.22

Release date: 2012-12-06

This release contains a variety of fixes from 8.3.21. For information about new features in the 8.3 major release, see 第 E.121 节.

The PostgreSQL community will stop releasing updates for the 8.3.X release series in February 2013. Users are encouraged to update to a newer release branch soon.

E.99.1. Migration to Version 8.3.22

A dump/restore is not required for those running 8.3.X.

However, if you are upgrading from a version earlier than 8.3.17, see 第 E.104 节.

E.99.2. Changes

- Fix multiple bugs associated with `CREATE INDEX CONCURRENTLY` (Andres Freund, Tom Lane)

  Fix `CREATE INDEX CONCURRENTLY` to use in-place updates when changing the state of an index's `pg_index` row. This prevents race conditions that could cause concurrent sessions to miss updating the target index, thus resulting in corrupt concurrently-created indexes.

  Also, fix various other operations to ensure that they ignore invalid indexes resulting from a failed `CREATE INDEX CONCURRENTLY` command. The most important of these is `VACUUM`, because an auto-vacuum could easily be launched on the table before corrective action can be taken to fix or remove the invalid index.

- Avoid corruption of internal hash tables when out of memory (Hitoshi Harada)

- Fix planning of non-strict equivalence clauses above outer joins (Tom Lane)

  The planner could derive incorrect constraints from a clause equating a non-strict construct to something else, for example `WHERE COALESCE(foo, 0) = 0` when `foo` is coming from the nullable side of an outer join.

- Improve planner's ability to prove exclusion constraints from equivalence classes (Tom Lane)

- Fix partial-row matching in hashed subplans to handle cross-type cases correctly (Tom Lane)

  This affects multicolumn `NOT IN` subplans, such as `WHERE (a, b) NOT IN (SELECT x, y FROM ...)` when for instance `b` and `y` are `int4` and `int8` respectively. This mistake led to wrong answers or crashes depending on the specific datatypes involved.

- Acquire buffer lock when re-fetching the old tuple for an `AFTER ROW UPDATE/DELETE` trigger (Andres Freund)

  In very unusual circumstances, this oversight could result in passing incorrect data to the precheck logic for a foreign-key enforcement trigger. That could result in a crash, or in an incorrect decision about whether to fire the trigger.

- Fix `REASSIGN OWNED` to handle grants on tablespaces (Álvaro Herrera)

- Ignore incorrect `pg_attribute` entries for system columns for views (Tom Lane)

  Views do not have any system columns. However, we forgot to remove such entries when converting a table to a view. That's fixed properly for 9.3 and later, but in previous branches we need to defend against existing mis-converted views.

- Fix rule printing to dump `INSERT INTO table DEFAULT VALUES` correctly (Tom Lane)

- Guard against stack overflow when there are too many `UNION/INTERSECT/EXCEPT` clauses in a query (Tom Lane)

- Prevent platform-dependent failures when dividing the minimum possible integer value by -1 (Xi Wang, Tom Lane)

- Fix possible access past end of string in date parsing (Hitoshi Harada)

- Produce an understandable error message if the length of the path name for a Unix-domain socket exceeds the platform-specific limit (Tom Lane, Andrew Dunstan)

  Formerly, this would result in something quite unhelpful, such as “Non-recoverable failure in name resolution”.

- Fix memory leaks when sending composite column values to the client (Tom Lane)

- Make pg_ctl more robust about reading the `postmaster.pid` file (Heikki Linnakangas)

  Fix race conditions and possible file descriptor leakage.

- Fix possible crash in psql if incorrectly-encoded data is presented and the `client_encoding` setting is a client-only encoding, such as SJIS (Jiang Guiqing)

- Fix bugs in the `restore.sql` script emitted by pg_dump in `tar` output format (Tom Lane)

  The script would fail outright on tables whose names include upper-case characters. Also, make the script capable of restoring data in `--inserts` mode as well as the regular COPY mode.

- Fix pg_restore to accept POSIX-conformant `tar` files (Brian Weaver, Tom Lane)

  The original coding of pg_dump's `tar` output mode produced files that are not fully conformant with the POSIX standard. This has been corrected for version 9.3. This patch updates previous branches so that they will accept both the incorrect and the corrected formats, in hopes of avoiding compatibility problems when 9.3 comes out.

- Fix pg_resetxlog to locate `postmaster.pid` correctly when given a relative path to the data directory (Tom Lane)

  This mistake could lead to pg_resetxlog not noticing that there is an active postmaster using the data directory.

- Fix libpq's `lo_import()` and `lo_export()` functions to report file I/O errors properly (Tom Lane)

- Fix ecpg's processing of nested structure pointer variables (Muhammad Usama)

- Make `contrib/pageinspect`'s btree page inspection functions take buffer locks while examining pages (Tom Lane)

- Fix pgxs support for building loadable modules on AIX (Tom Lane)

  Building modules outside the original source tree didn't work on AIX.

- Update time zone data files to tzdata release 2012j for DST law changes in Cuba, Israel, Jordan, Libya, Palestine, Western Samoa, and portions of Brazil.

E.100. Release 8.3.21

Release date: 2012-09-24

This release contains a variety of fixes from 8.3.20. For information about new features in the 8.3 major release, see 第 E.121 节.

The PostgreSQL community will stop releasing updates for the 8.3.X release series in February 2013. Users are encouraged to update to a newer release branch soon.

E.100.1. Migration to Version 8.3.21

A dump/restore is not required for those running 8.3.X.

However, if you are upgrading from a version earlier than 8.3.17, see 第 E.104 节.

E.100.2. Changes

- Improve page-splitting decisions in GiST indexes (Alexander Korotkov, Robert Haas, Tom Lane)

  Multi-column GiST indexes might suffer unexpected bloat due to this error.

- Fix cascading privilege revoke to stop if privileges are still held (Tom Lane)

  If we revoke a grant option from some role *X*, but *X* still holds that option via a grant from someone else, we should not recursively revoke the corresponding privilege from role(s) *Y* that *X* had granted it to.

- Fix handling of `SIGFPE` when PL/Perl is in use (Andres Freund)

  Perl resets the process's `SIGFPE` handler to `SIG_IGN`, which could result in crashes later on. Restore the normal Postgres signal handler after initializing PL/Perl.

- Prevent PL/Perl from crashing if a recursive PL/Perl function is redefined while being executed (Tom Lane)

- Work around possible misoptimization in PL/Perl (Tom Lane)

  Some Linux distributions contain an incorrect version of `pthread.h` that results in incorrect compiled code in PL/Perl, leading to crashes if a PL/Perl function calls another one that throws an error.

- Update time zone data files to tzdata release 2012f for DST law changes in Fiji

E.101. Release 8.3.20

Release date: 2012-08-17

This release contains a variety of fixes from 8.3.19. For information about new features in the 8.3 major release, see 第 E.121 节.

The PostgreSQL community will stop releasing updates for the 8.3.X release series in February 2013. Users are encouraged to update to a newer release branch soon.

E.101.1. Migration to Version 8.3.20

A dump/restore is not required for those running 8.3.X.

However, if you are upgrading from a version earlier than 8.3.17, see 第 E.104 节.

E.101.2. Changes

- Prevent access to external files/URLs via XML entity references (Noah Misch, Tom Lane)

  `xml_parse()` would attempt to fetch external files or URLs as needed to resolve DTD and entity references in an XML value, thus allowing unprivileged database users to

attempt to fetch data with the privileges of the database server. While the external data wouldn't get returned directly to the user, portions of it could be exposed in error messages if the data didn't parse as valid XML; and in any case the mere ability to check existence of a file might be useful to an attacker. (CVE-2012-3489)

- Prevent access to external files/URLs via `contrib/xml2`'s `xslt_process()` (Peter Eisentraut)

  libxslt offers the ability to read and write both files and URLs through stylesheet commands, thus allowing unprivileged database users to both read and write data with the privileges of the database server. Disable that through proper use of libxslt's security options. (CVE-2012-3488)

  Also, remove `xslt_process()`'s ability to fetch documents and stylesheets from external files/URLs. While this was a documented "feature", it was long regarded as a bad idea. The fix for CVE-2012-3489 broke that capability, and rather than expend effort on trying to fix it, we're just going to summarily remove it.

- Prevent too-early recycling of btree index pages (Noah Misch)

  When we allowed read-only transactions to skip assigning XIDs, we introduced the possibility that a deleted btree page could be recycled while a read-only transaction was still in flight to it. This would result in incorrect index search results. The probability of such an error occurring in the field seems very low because of the timing requirements, but nonetheless it should be fixed.

- Fix crash-safety bug with newly-created-or-reset sequences (Tom Lane)

  If `ALTER SEQUENCE` was executed on a freshly created or reset sequence, and then precisely one `nextval()` call was made on it, and then the server crashed, WAL replay would restore the sequence to a state in which it appeared that no `nextval()` had been done, thus allowing the first sequence value to be returned again by the next `nextval()` call. In particular this could manifest for `serial` columns, since creation of a serial column's sequence includes an `ALTER SEQUENCE OWNED BY` step.

- Ensure the `backup_label` file is fsync'd after `pg_start_backup()` (Dave Kerr)

- Back-patch 9.1 improvement to compress the fsync request queue (Robert Haas)

  This improves performance during checkpoints. The 9.1 change has now seen enough field testing to seem safe to back-patch.

- Only allow autovacuum to be auto-canceled by a directly blocked process (Tom Lane)

  The original coding could allow inconsistent behavior in some cases; in particular, an autovacuum could get canceled after less than `deadlock_timeout` grace period.

- Improve logging of autovacuum cancels (Robert Haas)

- Fix log collector so that `log_truncate_on_rotation` works during the very first log rotation after server start (Tom Lane)

- Ensure that a whole-row reference to a subquery doesn't include any extra `GROUP BY` or `ORDER BY` columns (Tom Lane)

- Disallow copying whole-row references in `CHECK` constraints and index definitions during `CREATE TABLE` (Tom Lane)

  This situation can arise in `CREATE TABLE` with `LIKE` or `INHERITS`. The copied whole-row variable was incorrectly labeled with the row type of the original table not the

new one. Rejecting the case seems reasonable for `LIKE`, since the row types might well diverge later. For `INHERITS` we should ideally allow it, with an implicit coercion to the parent table's row type; but that will require more work than seems safe to back-patch.

- Fix memory leak in `ARRAY(SELECT ...)` subqueries (Heikki Linnakangas, Tom Lane)
- Fix extraction of common prefixes from regular expressions (Tom Lane)

  The code could get confused by quantified parenthesized subexpressions, such as `^(foo)?bar`. This would lead to incorrect index optimization of searches for such patterns.
- Report errors properly in `contrib/xml2`'s `xslt_process()` (Tom Lane)
- Update time zone data files to tzdata release 2012e for DST law changes in Morocco and Tokelau

E.102. Release 8.3.19

Release date: 2012-06-04

This release contains a variety of fixes from 8.3.18. For information about new features in the 8.3 major release, see 第 E.121 节.

E.102.1. Migration to Version 8.3.19

A dump/restore is not required for those running 8.3.X.

However, if you are upgrading from a version earlier than 8.3.17, see 第 E.104 节.

E.102.2. Changes

- Fix incorrect password transformation in `contrib/pgcrypto`'s DES `crypt()` function (Solar Designer)

  If a password string contained the byte value `0x80`, the remainder of the password was ignored, causing the password to be much weaker than it appeared. With this fix, the rest of the string is properly included in the DES hash. Any stored password values that are affected by this bug will thus no longer match, so the stored values may need to be updated. (CVE-2012-2143)
- Ignore `SECURITY DEFINER` and `SET` attributes for a procedural language's call handler (Tom Lane)

  Applying such attributes to a call handler could crash the server. (CVE-2012-2655)
- Allow numeric timezone offsets in `timestamp` input to be up to 16 hours away from UTC (Tom Lane)

  Some historical time zones have offsets larger than 15 hours, the previous limit. This could result in dumped data values being rejected during reload.
- Fix timestamp conversion to cope when the given time is exactly the last DST transition time for the current timezone (Tom Lane)

  This oversight has been there a long time, but was not noticed previously because most DST-using zones are presumed to have an indefinite sequence of future DST transitions.

- Fix `text` to `name` and `char` to `name` casts to perform string truncation correctly in multibyte encodings (Karl Schnaitter)

- Fix memory copying bug in `to_tsquery()` (Heikki Linnakangas)

- Fix slow session startup when `pg_attribute` is very large (Tom Lane)

  If `pg_attribute` exceeds one-fourth of `shared_buffers`, cache rebuilding code that is sometimes needed during session start would trigger the synchronized-scan logic, causing it to take many times longer than normal. The problem was particularly acute if many new sessions were starting at once.

- Ensure sequential scans check for query cancel reasonably often (Merlin Moncure)

  A scan encountering many consecutive pages that contain no live tuples would not respond to interrupts meanwhile.

- Ensure the Windows implementation of `PGSemaphoreLock()` clears `Immediate InterruptOK` before returning (Tom Lane)

  This oversight meant that a query-cancel interrupt received later in the same query could be accepted at an unsafe time, with unpredictable but not good consequences.

- Show whole-row variables safely when printing views or rules (Abbas Butt, Tom Lane)

  Corner cases involving ambiguous names (that is, the name could be either a table or column name of the query) were printed in an ambiguous way, risking that the view or rule would be interpreted differently after dump and reload. Avoid the ambiguous case by attaching a no-op cast.

- Ensure autovacuum worker processes perform stack depth checking properly (Heikki Linnakangas)

  Previously, infinite recursion in a function invoked by auto-`ANALYZE` could crash worker processes.

- Fix logging collector to not lose log coherency under high load (Andrew Dunstan)

  The collector previously could fail to reassemble large messages if it got too busy.

- Fix logging collector to ensure it will restart file rotation after receiving SIGHUP (Tom Lane)

- Fix PL/pgSQL's `GET DIAGNOSTICS` command when the target is the function's first variable (Tom Lane)

- Fix several performance problems in pg_dump when the database contains many objects (Jeff Janes, Tom Lane)

  pg_dump could get very slow if the database contained many schemas, or if many objects are in dependency loops, or if there are many owned sequences.

- Fix `contrib/dblink`'s `dblink_exec()` to not leak temporary database connections upon error (Tom Lane)

- Update time zone data files to tzdata release 2012c for DST law changes in Antarctica, Armenia, Chile, Cuba, Falkland Islands, Gaza, Haiti, Hebron, Morocco, Syria, and Tokelau Islands; also historical corrections for Canada.

E.103. Release 8.3.18

Release date: 2012-02-27

This release contains a variety of fixes from 8.3.17. For information about new features in the 8.3 major release, see 第 E.121 节.

E.103.1. Migration to Version 8.3.18

A dump/restore is not required for those running 8.3.X.

However, if you are upgrading from a version earlier than 8.3.17, see 第 E.104 节.

E.103.2. Changes

- Require execute permission on the trigger function for `CREATE TRIGGER` (Robert Haas)

  This missing check could allow another user to execute a trigger function with forged input data, by installing it on a table he owns. This is only of significance for trigger functions marked `SECURITY DEFINER`, since otherwise trigger functions run as the table owner anyway. (CVE-2012-0866)

- Convert newlines to spaces in names written in pg_dump comments (Robert Haas)

  pg_dump was incautious about sanitizing object names that are emitted within SQL comments in its output script. A name containing a newline would at least render the script syntactically incorrect. Maliciously crafted object names could present a SQL injection risk when the script is reloaded. (CVE-2012-0868)

- Fix btree index corruption from insertions concurrent with vacuuming (Tom Lane)

  An index page split caused by an insertion could sometimes cause a concurrently-running `VACUUM` to miss removing index entries that it should remove. After the corresponding table rows are removed, the dangling index entries would cause errors (such as "could not read block N in file ...") or worse, silently wrong query results after unrelated rows are re-inserted at the now-free table locations. This bug has been present since release 8.2, but occurs so infrequently that it was not diagnosed until now. If you have reason to suspect that it has happened in your database, reindexing the affected index will fix things.

- Allow non-existent values for some settings in `ALTER USER/DATABASE SET` (Heikki Linnakangas)

  Allow `default_text_search_config`, `default_tablespace`, and `temp_tablespaces` to be set to names that are not known. This is because they might be known in another database where the setting is intended to be used, or for the tablespace cases because the tablespace might not be created yet. The same issue was previously recognized for `search_path`, and these settings now act like that one.

- Track the OID counter correctly during WAL replay, even when it wraps around (Tom Lane)

  Previously the OID counter would remain stuck at a high value until the system exited replay mode. The practical consequences of that are usually nil, but there are scenarios wherein a standby server that's been promoted to master might take a long time to advance the OID counter to a reasonable value once values are needed.

- Fix regular expression back-references with `*` attached (Tom Lane)

  Rather than enforcing an exact string match, the code would effectively accept any string that satisfies the pattern sub-expression referenced by the back-reference symbol.

  A similar problem still afflicts back-references that are embedded in a larger quantified expression, rather than being the immediate subject of the quantifier. This will be addressed in a future PostgreSQL release.

- Fix recently-introduced memory leak in processing of `inet/cidr` values (Heikki Linnakangas)

  A patch in the December 2011 releases of PostgreSQL caused memory leakage in these operations, which could be significant in scenarios such as building a btree index on such a column.

- Avoid double close of file handle in syslogger on Windows (MauMau)

  Ordinarily this error was invisible, but it would cause an exception when running on a debug version of Windows.

- Fix I/O-conversion-related memory leaks in plpgsql (Andres Freund, Jan Urbanski, Tom Lane)

  Certain operations would leak memory until the end of the current function.

- Improve pg_dump's handling of inherited table columns (Tom Lane)

  pg_dump mishandled situations where a child column has a different default expression than its parent column. If the default is textually identical to the parent's default, but not actually the same (for instance, because of schema search path differences) it would not be recognized as different, so that after dump and restore the child would be allowed to inherit the parent's default. Child columns that are `NOT NULL` where their parent is not could also be restored subtly incorrectly.

- Fix pg_restore's direct-to-database mode for INSERT-style table data (Tom Lane)

  Direct-to-database restores from archive files made with `--inserts` or `--column-inserts` options fail when using pg_restore from a release dated September or December 2011, as a result of an oversight in a fix for another problem. The archive file itself is not at fault, and text-mode output is okay.

- Fix error in `contrib/intarray`'s `int[] & int[]` operator (Guillaume Lelarge)

  If the smallest integer the two input arrays have in common is 1, and there are smaller values in either array, then 1 would be incorrectly omitted from the result.

- Fix error detection in `contrib/pgcrypto`'s `encrypt_iv()` and `decrypt_iv()` (Marko Kreen)

  These functions failed to report certain types of invalid-input errors, and would instead return random garbage values for incorrect input.

- Fix one-byte buffer overrun in `contrib/test_parser` (Paul Guyot)

  The code would try to read one more byte than it should, which would crash in corner cases. Since `contrib/test_parser` is only example code, this is not a security issue in itself, but bad example code is still bad.

- Use `__sync_lock_test_and_set()` for spinlocks on ARM, if available (Martin Pitt)

This function replaces our previous use of the `SWPB` instruction, which is deprecated and not available on ARMv6 and later. Reports suggest that the old code doesn't fail in an obvious way on recent ARM boards, but simply doesn't interlock concurrent accesses, leading to bizarre failures in multiprocess operation.

- Use `-fexcess-precision=standard` option when building with gcc versions that accept it (Andrew Dunstan)

  This prevents assorted scenarios wherein recent versions of gcc will produce creative results.

- Allow use of threaded Python on FreeBSD (Chris Rees)

  Our configure script previously believed that this combination wouldn't work; but FreeBSD fixed the problem, so remove that error check.

E.104. Release 8.3.17

Release date: 2011-12-05

This release contains a variety of fixes from 8.3.16. For information about new features in the 8.3 major release, see 第 E.121 节.

E.104.1. Migration to Version 8.3.17

A dump/restore is not required for those running 8.3.X.

However, a longstanding error was discovered in the definition of the `information_schema.referential_constraints` view. If you rely on correct results from that view, you should replace its definition as explained in the first changelog item below.

Also, if you are upgrading from a version earlier than 8.3.8, see 第 E.113 节.

E.104.2. Changes

- Fix bugs in `information_schema.referential_constraints` view (Tom Lane)

  This view was being insufficiently careful about matching the foreign-key constraint to the depended-on primary or unique key constraint. That could result in failure to show a foreign key constraint at all, or showing it multiple times, or claiming that it depends on a different constraint than the one it really does.

  Since the view definition is installed by initdb, merely upgrading will not fix the problem. If you need to fix this in an existing installation, you can (as a superuser) drop the `information_schema` schema then re-create it by sourcing *`SHAREDIR`*`/information_schema.sql`. (Run `pg_config --sharedir` if you're uncertain where *`SHAREDIR`* is.) This must be repeated in each database to be fixed.

- Fix TOAST-related data corruption during `CREATE TABLE dest AS SELECT * FROM src` or `INSERT INTO dest SELECT * FROM src` (Tom Lane)

  If a table has been modified by `ALTER TABLE ADD COLUMN`, attempts to copy its data verbatim to another table could produce corrupt results in certain corner cases. The problem can only manifest in this precise form in 8.4 and later, but we patched earlier versions as well in case there are other code paths that could trigger the same bug.

- Fix race condition during toast table access from stale syscache entries (Tom Lane)

The typical symptom was transient errors like "missing chunk number 0 for toast value NNNNN in pg_toast_2619", where the cited toast table would always belong to a system catalog.

- Make `DatumGetInetP()` unpack inet datums that have a 1-byte header, and add a new macro, `DatumGetInetPP()`, that does not (Heikki Linnakangas)

  This change affects no core code, but might prevent crashes in add-on code that expects `DatumGetInetP()` to produce an unpacked datum as per usual convention.

- Improve locale support in `money` type's input and output (Tom Lane)

  Aside from not supporting all standard `lc_monetary` formatting options, the input and output functions were inconsistent, meaning there were locales in which dumped `money` values could not be re-read.

- Don't let `transform_null_equals` affect `CASE foo WHEN NULL ...` constructs (Heikki Linnakangas)

  `transform_null_equals` is only supposed to affect `foo = NULL` expressions written directly by the user, not equality checks generated internally by this form of `CASE`.

- Change foreign-key trigger creation order to better support self-referential foreign keys (Tom Lane)

  For a cascading foreign key that references its own table, a row update will fire both the `ON UPDATE` trigger and the `CHECK` trigger as one event. The `ON UPDATE` trigger must execute first, else the `CHECK` will check a non-final state of the row and possibly throw an inappropriate error. However, the firing order of these triggers is determined by their names, which generally sort in creation order since the triggers have auto-generated names following the convention "RI_ConstraintTrigger_NNNN". A proper fix would require modifying that convention, which we will do in 9.2, but it seems risky to change it in existing releases. So this patch just changes the creation order of the triggers. Users encountering this type of error should drop and re-create the foreign key constraint to get its triggers into the right order.

- Avoid floating-point underflow while tracking buffer allocation rate (Greg Matthews)

  While harmless in itself, on certain platforms this would result in annoying kernel log messages.

- Preserve blank lines within commands in psql's command history (Robert Haas)

  The former behavior could cause problems if an empty line was removed from within a string literal, for example.

- Fix pg_dump to dump user-defined casts between auto-generated types, such as table rowtypes (Tom Lane)

- Use the preferred version of xsubpp to build PL/Perl, not necessarily the operating system's main copy (David Wheeler and Alex Hunsaker)

- Fix incorrect coding in `contrib/dict_int` and `contrib/dict_xsyn` (Tom Lane)

  Some functions incorrectly assumed that memory returned by `palloc()` is guaranteed zeroed.

- Honor query cancel interrupts promptly in `pgstatindex()` (Robert Haas)

- Ensure VPATH builds properly install all server header files (Peter Eisentraut)
- Shorten file names reported in verbose error messages (Peter Eisentraut)

  Regular builds have always reported just the name of the C file containing the error message call, but VPATH builds formerly reported an absolute path name.
- Fix interpretation of Windows timezone names for Central America (Tom Lane)

  Map "Central America Standard Time" to `CST6`, not `CST6CDT`, because DST is generally not observed anywhere in Central America.
- Update time zone data files to tzdata release 2011n for DST law changes in Brazil, Cuba, Fiji, Palestine, Russia, and Samoa; also historical corrections for Alaska and British East Africa.

E.105. Release 8.3.16

Release date: 2011-09-26

This release contains a variety of fixes from 8.3.15. For information about new features in the 8.3 major release, see 第 E.121 节.

E.105.1. Migration to Version 8.3.16

A dump/restore is not required for those running 8.3.X. However, if you are upgrading from a version earlier than 8.3.8, see 第 E.113 节.

E.105.2. Changes

- Fix bugs in indexing of in-doubt HOT-updated tuples (Tom Lane)

  These bugs could result in index corruption after reindexing a system catalog. They are not believed to affect user indexes.
- Fix multiple bugs in GiST index page split processing (Heikki Linnakangas)

  The probability of occurrence was low, but these could lead to index corruption.
- Fix possible buffer overrun in `tsvector_concat()` (Tom Lane)

  The function could underestimate the amount of memory needed for its result, leading to server crashes.
- Fix crash in `xml_recv` when processing a "standalone" parameter (Tom Lane)
- Avoid possibly accessing off the end of memory in `ANALYZE` and in SJIS-2004 encoding conversion (Noah Misch)

  This fixes some very-low-probability server crash scenarios.
- Fix race condition in relcache init file invalidation (Tom Lane)

  There was a window wherein a new backend process could read a stale init file but miss the inval messages that would tell it the data is stale. The result would be bizarre failures in catalog accesses, typically "could not read block 0 in file ..." later during startup.
- Fix memory leak at end of a GiST index scan (Tom Lane)

Commands that perform many separate GiST index scans, such as verification of a new GiST-based exclusion constraint on a table already containing many rows, could transiently require large amounts of memory due to this leak.

- Fix performance problem when constructing a large, lossy bitmap (Tom Lane)
- Fix array- and path-creating functions to ensure padding bytes are zeroes (Tom Lane)

  This avoids some situations where the planner will think that semantically-equal constants are not equal, resulting in poor optimization.

- Work around gcc 4.6.0 bug that breaks WAL replay (Tom Lane)

  This could lead to loss of committed transactions after a server crash.

- Fix dump bug for `VALUES` in a view (Tom Lane)
- Disallow `SELECT FOR UPDATE/SHARE` on sequences (Tom Lane)

  This operation doesn't work as expected and can lead to failures.

- Defend against integer overflow when computing size of a hash table (Tom Lane)
- Fix cases where `CLUSTER` might attempt to access already-removed TOAST data (Tom Lane)
- Fix portability bugs in use of credentials control messages for "peer" authentication (Tom Lane)
- Fix SSPI login when multiple roundtrips are required (Ahmed Shinwari, Magnus Hagander)

  The typical symptom of this problem was "The function requested is not supported" errors during SSPI login.

- Fix typo in `pg_srand48` seed initialization (Andres Freund)

  This led to failure to use all bits of the provided seed. This function is not used on most platforms (only those without `srandom`), and the potential security exposure from a less-random-than-expected seed seems minimal in any case.

- Avoid integer overflow when the sum of `LIMIT` and `OFFSET` values exceeds 2^63 (Heikki Linnakangas)
- Add overflow checks to `int4` and `int8` versions of `generate_series()` (Robert Haas)
- Fix trailing-zero removal in `to_char()` (Marti Raudsepp)

  In a format with `FM` and no digit positions after the decimal point, zeroes to the left of the decimal point could be removed incorrectly.

- Fix `pg_size_pretty()` to avoid overflow for inputs close to 2^63 (Tom Lane)
- In pg_ctl, support silent mode for service registrations on Windows (MauMau)
- Fix psql's counting of script file line numbers during `COPY` from a different file (Tom Lane)

- Fix pg_restore's direct-to-database mode for `standard_conforming_strings` (Tom Lane)

  pg_restore could emit incorrect commands when restoring directly to a database server from an archive file that had been made with `standard_conforming_strings` set to `on`.

- Fix write-past-buffer-end and memory leak in libpq's LDAP service lookup code (Albe Laurenz)

- In libpq, avoid failures when using nonblocking I/O and an SSL connection (Martin Pihlak, Tom Lane)

- Improve libpq's handling of failures during connection startup (Tom Lane)

  In particular, the response to a server report of `fork()` failure during SSL connection startup is now saner.

- Improve libpq's error reporting for SSL failures (Tom Lane)

- Make ecpglib write `double` values with 15 digits precision (Akira Kurosawa)

- In ecpglib, be sure `LC_NUMERIC` setting is restored after an error (Michael Meskes)

- Apply upstream fix for blowfish signed-character bug (CVE-2011-2483) (Tom Lane)

  `contrib/pg_crypto`'s blowfish encryption code could give wrong results on platforms where char is signed (which is most), leading to encrypted passwords being weaker than they should be.

- Fix memory leak in `contrib/seg` (Heikki Linnakangas)

- Fix `pgstatindex()` to give consistent results for empty indexes (Tom Lane)

- Allow building with perl 5.14 (Alex Hunsaker)

- Update configure script's method for probing existence of system functions (Tom Lane)

  The version of autoconf we used in 8.3 and 8.2 could be fooled by compilers that perform link-time optimization.

- Fix assorted issues with build and install file paths containing spaces (Tom Lane)

- Update time zone data files to tzdata release 2011i for DST law changes in Canada, Egypt, Russia, Samoa, and South Sudan.

E.106. Release 8.3.15

Release date: 2011-04-18

This release contains a variety of fixes from 8.3.14. For information about new features in the 8.3 major release, see 第 E.121 节.

E.106.1. Migration to Version 8.3.15

A dump/restore is not required for those running 8.3.X. However, if you are upgrading from a version earlier than 8.3.8, see 第 E.113 节.

E.106.2. Changes

- Disallow including a composite type in itself (Tom Lane)

  This prevents scenarios wherein the server could recurse infinitely while processing the composite type. While there are some possible uses for such a structure, they don't seem compelling enough to justify the effort required to make sure it always works safely.

- Avoid potential deadlock during catalog cache initialization (Nikhil Sontakke)

  In some cases the cache loading code would acquire share lock on a system index before locking the index's catalog. This could deadlock against processes trying to acquire exclusive locks in the other, more standard order.

- Fix dangling-pointer problem in `BEFORE ROW UPDATE` trigger handling when there was a concurrent update to the target tuple (Tom Lane)

  This bug has been observed to result in intermittent "cannot extract system attribute from virtual tuple" failures while trying to do `UPDATE RETURNING ctid`. There is a very small probability of more serious errors, such as generating incorrect index entries for the updated tuple.

- Disallow `DROP TABLE` when there are pending deferred trigger events for the table (Tom Lane)

  Formerly the `DROP` would go through, leading to "could not open relation with OID nnn" errors when the triggers were eventually fired.

- Fix PL/Python memory leak involving array slices (Daniel Popowich)

- Fix pg_restore to cope with long lines (over 1KB) in TOC files (Tom Lane)

- Put in more safeguards against crashing due to division-by-zero with overly enthusiastic compiler optimization (Aurelien Jarno)

- Support use of dlopen() in FreeBSD and OpenBSD on MIPS (Tom Lane)

  There was a hard-wired assumption that this system function was not available on MIPS hardware on these systems. Use a compile-time test instead, since more recent versions have it.

- Fix compilation failures on HP-UX (Heikki Linnakangas)

- Fix version-incompatibility problem with libintl on Windows (Hiroshi Inoue)

- Fix usage of xcopy in Windows build scripts to work correctly under Windows 7 (Andrew Dunstan)

  This affects the build scripts only, not installation or usage.

- Fix path separator used by pg_regress on Cygwin (Andrew Dunstan)

- Update time zone data files to tzdata release 2011f for DST law changes in Chile, Cuba, Falkland Islands, Morocco, Samoa, and Turkey; also historical corrections for South Australia, Alaska, and Hawaii.

E.107. Release 8.3.14

Release date: 2011-01-31

This release contains a variety of fixes from 8.3.13. For information about new features in the 8.3 major release, see 第 E.121 节.

E.107.1. Migration to Version 8.3.14

A dump/restore is not required for those running 8.3.X. However, if you are upgrading from a version earlier than 8.3.8, see 第 E.113 节.

E.107.2. Changes

- Avoid failures when `EXPLAIN` tries to display a simple-form `CASE` expression (Tom Lane)

  If the `CASE`'s test expression was a constant, the planner could simplify the `CASE` into a form that confused the expression-display code, resulting in "unexpected CASE WHEN clause" errors.

- Fix assignment to an array slice that is before the existing range of subscripts (Tom Lane)

  If there was a gap between the newly added subscripts and the first pre-existing subscript, the code miscalculated how many entries needed to be copied from the old array's null bitmap, potentially leading to data corruption or crash.

- Avoid unexpected conversion overflow in planner for very distant date values (Tom Lane)

  The `date` type supports a wider range of dates than can be represented by the `timestamp` types, but the planner assumed it could always convert a date to timestamp with impunity.

- Fix pg_restore's text output for large objects (BLOBs) when `standard_conforming_strings` is on (Tom Lane)

  Although restoring directly to a database worked correctly, string escaping was incorrect if pg_restore was asked for SQL text output and `standard_conforming_strings` had been enabled in the source database.

- Fix erroneous parsing of `tsquery` values containing `... & !(subexpression) | ...` (Tom Lane)

  Queries containing this combination of operators were not executed correctly. The same error existed in `contrib/intarray`'s `query_int` type and `contrib/ltree`'s `ltxtquery` type.

- Fix buffer overrun in `contrib/intarray`'s input function for the `query_int` type (Apple)

  This bug is a security risk since the function's return address could be overwritten. Thanks to Apple Inc's security team for reporting this issue and supplying the fix. (CVE-2010-4015)

- Fix bug in `contrib/seg`'s GiST picksplit algorithm (Alexander Korotkov)

  This could result in considerable inefficiency, though not actually incorrect answers, in a GiST index on a `seg` column. If you have such an index, consider `REINDEX`ing it after installing this update. (This is identical to the bug that was fixed in `contrib/cube` in the previous update.)

E.108. Release 8.3.13

Release date: 2010-12-16

This release contains a variety of fixes from 8.3.12. For information about new features in the 8.3 major release, see 第 E.121 节.

E.108.1. Migration to Version 8.3.13

A dump/restore is not required for those running 8.3.X. However, if you are upgrading from a version earlier than 8.3.8, see 第 E.113 节.

E.108.2. Changes

- Force the default `wal_sync_method` to be `fdatasync` on Linux (Tom Lane, Marti Raudsepp)

  The default on Linux has actually been `fdatasync` for many years, but recent kernel changes caused PostgreSQL to choose `open_datasync` instead. This choice did not result in any performance improvement, and caused outright failures on certain filesystems, notably `ext4` with the `data=journal` mount option.

- Fix assorted bugs in WAL replay logic for GIN indexes (Tom Lane)

  This could result in "bad buffer id: 0" failures or corruption of index contents during replication.

- Fix recovery from base backup when the starting checkpoint WAL record is not in the same WAL segment as its redo point (Jeff Davis)

- Fix persistent slowdown of autovacuum workers when multiple workers remain active for a long time (Tom Lane)

  The effective `vacuum_cost_limit` for an autovacuum worker could drop to nearly zero if it processed enough tables, causing it to run extremely slowly.

- Add support for detecting register-stack overrun on `IA64` (Tom Lane)

  The `IA64` architecture has two hardware stacks. Full prevention of stack-overrun failures requires checking both.

- Add a check for stack overflow in `copyObject()` (Tom Lane)

  Certain code paths could crash due to stack overflow given a sufficiently complex query.

- Fix detection of page splits in temporary GiST indexes (Heikki Linnakangas)

  It is possible to have a "concurrent" page split in a temporary index, if for example there is an open cursor scanning the index when an insertion is done. GiST failed to detect this case and hence could deliver wrong results when execution of the cursor continued.

- Avoid memory leakage while `ANALYZE`'ing complex index expressions (Tom Lane)

- Ensure an index that uses a whole-row Var still depends on its table (Tom Lane)

  An index declared like `create index i on t (foo(t.*))` would not automatically get dropped when its table was dropped.

- Do not “inline” a SQL function with multiple `OUT` parameters (Tom Lane)

  This avoids a possible crash due to loss of information about the expected result rowtype.

- Behave correctly if `ORDER BY`, `LIMIT`, `FOR UPDATE`, or `WITH` is attached to the `VALUES` part of `INSERT ... VALUES` (Tom Lane)

- Fix constant-folding of `COALESCE()` expressions (Tom Lane)

  The planner would sometimes attempt to evaluate sub-expressions that in fact could never be reached, possibly leading to unexpected errors.

- Fix postmaster crash when connection acceptance (`accept()` or one of the calls made immediately after it) fails, and the postmaster was compiled with GSSAPI support (Alexander Chernikov)

- Fix missed unlink of temporary files when `log_temp_files` is active (Tom Lane)

  If an error occurred while attempting to emit the log message, the unlink was not done, resulting in accumulation of temp files.

- Add print functionality for `InhRelation` nodes (Tom Lane)

  This avoids a failure when `debug_print_parse` is enabled and certain types of query are executed.

- Fix incorrect calculation of distance from a point to a horizontal line segment (Tom Lane)

  This bug affected several different geometric distance-measurement operators.

- Fix PL/pgSQL's handling of “simple” expressions to not fail in recursion or error-recovery cases (Tom Lane)

- Fix PL/Python's handling of set-returning functions (Jan Urbanski)

  Attempts to call SPI functions within the iterator generating a set result would fail.

- Fix bug in `contrib/cube`'s GiST picksplit algorithm (Alexander Korotkov)

  This could result in considerable inefficiency, though not actually incorrect answers, in a GiST index on a `cube` column. If you have such an index, consider `REINDEX`ing it after installing this update.

- Don't emit “identifier will be truncated” notices in `contrib/dblink` except when creating new connections (Itagaki Takahiro)

- Fix potential coredump on missing public key in `contrib/pgcrypto` (Marti Raudsepp)

- Fix memory leak in `contrib/xml2`'s XPath query functions (Tom Lane)

- Update time zone data files to tzdata release 2010o for DST law changes in Fiji and Samoa; also historical corrections for Hong Kong.

E.109. Release 8.3.12

Release date: 2010-10-04

This release contains a variety of fixes from 8.3.11. For information about new features in the 8.3 major release, see 第 E.121 节.

E.109.1. Migration to Version 8.3.12

A dump/restore is not required for those running 8.3.X. However, if you are upgrading from a version earlier than 8.3.8, see 第 E.113 节.

E.109.2. Changes

- Use a separate interpreter for each calling SQL userid in PL/Perl and PL/Tcl (Tom Lane)

  This change prevents security problems that can be caused by subverting Perl or Tcl code that will be executed later in the same session under another SQL user identity (for example, within a `SECURITY DEFINER` function). Most scripting languages offer numerous ways that that might be done, such as redefining standard functions or operators called by the target function. Without this change, any SQL user with Perl or Tcl language usage rights can do essentially anything with the SQL privileges of the target function's owner.

  The cost of this change is that intentional communication among Perl and Tcl functions becomes more difficult. To provide an escape hatch, PL/PerlU and PL/TclU functions continue to use only one interpreter per session. This is not considered a security issue since all such functions execute at the trust level of a database superuser already.

  It is likely that third-party procedural languages that claim to offer trusted execution have similar security issues. We advise contacting the authors of any PL you are depending on for security-critical purposes.

  Our thanks to Tim Bunce for pointing out this issue (CVE-2010-3433).

- Prevent possible crashes in `pg_get_expr()` by disallowing it from being called with an argument that is not one of the system catalog columns it's intended to be used with (Heikki Linnakangas, Tom Lane)

- Treat exit code 128 (`ERROR_WAIT_NO_CHILDREN`) as non-fatal on Windows (Magnus Hagander)

  Under high load, Windows processes will sometimes fail at startup with this error code. Formerly the postmaster treated this as a panic condition and restarted the whole database, but that seems to be an overreaction.

- Fix incorrect usage of non-strict OR joinclauses in Append indexscans (Tom Lane)

  This is a back-patch of an 8.4 fix that was missed in the 8.3 branch. This corrects an error introduced in 8.3.8 that could cause incorrect results for outer joins when the inner relation is an inheritance tree or `UNION ALL` subquery.

- Fix possible duplicate scans of `UNION ALL` member relations (Tom Lane)

- Fix “cannot handle unplanned sub-select” error (Tom Lane)

  This occurred when a sub-select contains a join alias reference that expands into an expression containing another sub-select.

- Fix failure to mark cached plans as transient (Tom Lane)

  If a plan is prepared while `CREATE INDEX CONCURRENTLY` is in progress for one of the referenced tables, it is supposed to be re-planned once the index is ready for use. This was not happening reliably.

- Reduce PANIC to ERROR in some occasionally-reported btree failure cases, and provide additional detail in the resulting error messages (Tom Lane)

  This should improve the system's robustness with corrupted indexes.

- Prevent show_session_authorization() from crashing within autovacuum processes (Tom Lane)

- Defend against functions returning setof record where not all the returned rows are actually of the same rowtype (Tom Lane)

- Fix possible failure when hashing a pass-by-reference function result (Tao Ma, Tom Lane)

- Improve merge join's handling of NULLs in the join columns (Tom Lane)

  A merge join can now stop entirely upon reaching the first NULL, if the sort order is such that NULLs sort high.

- Take care to fsync the contents of lockfiles (both `postmaster.pid` and the socket lockfile) while writing them (Tom Lane)

  This omission could result in corrupted lockfile contents if the machine crashes shortly after postmaster start. That could in turn prevent subsequent attempts to start the postmaster from succeeding, until the lockfile is manually removed.

- Avoid recursion while assigning XIDs to heavily-nested subtransactions (Andres Freund, Robert Haas)

  The original coding could result in a crash if there was limited stack space.

- Avoid holding open old WAL segments in the walwriter process (Magnus Hagander, Heikki Linnakangas)

  The previous coding would prevent removal of no-longer-needed segments.

- Fix `log_line_prefix`'s `%i` escape, which could produce junk early in backend startup (Tom Lane)

- Fix possible data corruption in `ALTER TABLE ... SET TABLESPACE` when archiving is enabled (Jeff Davis)

- Allow `CREATE DATABASE` and `ALTER DATABASE ... SET TABLESPACE` to be interrupted by query-cancel (Guillaume Lelarge)

- Fix `REASSIGN OWNED` to handle operator classes and families (Asko Tiidumaa)

- Fix possible core dump when comparing two empty `tsquery` values (Tom Lane)

- Fix `LIKE`'s handling of patterns containing `%` followed by _ (Tom Lane)

  We've fixed this before, but there were still some incorrectly-handled cases.

- In PL/Python, defend against null pointer results from `PyCObject_AsVoidPtr` and `PyCObject_FromVoidPtr` (Peter Eisentraut)

- Make psql recognize `DISCARD ALL` as a command that should not be encased in a transaction block in autocommit-off mode (Itagaki Takahiro)

- Fix ecpg to process data from `RETURNING` clauses correctly (Michael Meskes)

- Improve `contrib/dblink`'s handling of tables containing dropped columns (Tom Lane)

- Fix connection leak after "duplicate connection name" errors in `contrib/dblink` (Itagaki Takahiro)
- Fix `contrib/dblink` to handle connection names longer than 62 bytes correctly (Itagaki Takahiro)
- Add `hstore(text, text)` function to `contrib/hstore` (Robert Haas)

  This function is the recommended substitute for the now-deprecated `=>` operator. It was back-patched so that future-proofed code can be used with older server versions. Note that the patch will be effective only after `contrib/hstore` is installed or reinstalled in a particular database. Users might prefer to execute the `CREATE FUNCTION` command by hand, instead.
- Update build infrastructure and documentation to reflect the source code repository's move from CVS to Git (Magnus Hagander and others)
- Update time zone data files to tzdata release 2010l for DST law changes in Egypt and Palestine; also historical corrections for Finland.

  This change also adds new names for two Micronesian timezones: Pacific/Chuuk is now preferred over Pacific/Truk (and the preferred abbreviation is CHUT not TRUT) and Pacific/Pohnpei is preferred over Pacific/Ponape.
- Make Windows' "N. Central Asia Standard Time" timezone map to Asia/Novosibirsk, not Asia/Almaty (Magnus Hagander)

  Microsoft changed the DST behavior of this zone in the timezone update from KB976098. Asia/Novosibirsk is a better match to its new behavior.

E.110. Release 8.3.11

Release date: 2010-05-17

This release contains a variety of fixes from 8.3.10. For information about new features in the 8.3 major release, see 第 E.121 节.

E.110.1. Migration to Version 8.3.11

A dump/restore is not required for those running 8.3.X. However, if you are upgrading from a version earlier than 8.3.8, see 第 E.113 节.

E.110.2. Changes

- Enforce restrictions in `plperl` using an opmask applied to the whole interpreter, instead of using `Safe.pm` (Tim Bunce, Andrew Dunstan)

  Recent developments have convinced us that `Safe.pm` is too insecure to rely on for making `plperl` trustable. This change removes use of `Safe.pm` altogether, in favor of using a separate interpreter with an opcode mask that is always applied. Pleasant side effects of the change include that it is now possible to use Perl's `strict` pragma in a natural way in `plperl`, and that Perl's `$a` and `$b` variables work as expected in sort routines, and that function compilation is significantly faster. (CVE-2010-1169)
- Prevent PL/Tcl from executing untrustworthy code from `pltcl_modules` (Tom)

  PL/Tcl's feature for autoloading Tcl code from a database table could be exploited for trojan-horse attacks, because there was no restriction on who could create or

insert into that table. This change disables the feature unless `pltcl_modules` is owned by a superuser. (However, the permissions on the table are not checked, so installations that really need a less-than-secure modules table can still grant suitable privileges to trusted non-superusers.) Also, prevent loading code into the unrestricted "normal" Tcl interpreter unless we are really going to execute a `pltclu` function. (CVE-2010-1170)

- Fix possible crash if a cache reset message is received during rebuild of a relcache entry (Heikki)

  This error was introduced in 8.3.10 while fixing a related failure.

- Apply per-function GUC settings while running the language validator for the function (Itagaki Takahiro)

  This avoids failures if the function's code is invalid without the setting; an example is that SQL functions may not parse if the `search_path` is not correct.

- Do not allow an unprivileged user to reset superuser-only parameter settings (Alvaro)

  Previously, if an unprivileged user ran `ALTER USER ... RESET ALL` for himself, or `ALTER DATABASE ... RESET ALL` for a database he owns, this would remove all special parameter settings for the user or database, even ones that are only supposed to be changeable by a superuser. Now, the `ALTER` will only remove the parameters that the user has permission to change.

- Avoid possible crash during backend shutdown if shutdown occurs when a `CONTEXT` addition would be made to log entries (Tom)

  In some cases the context-printing function would fail because the current transaction had already been rolled back when it came time to print a log message.

- Ensure the archiver process responds to changes in `archive_command` as soon as possible (Tom)

- Update pl/perl's `ppport.h` for modern Perl versions (Andrew)

- Fix assorted memory leaks in pl/python (Andreas Freund, Tom)

- Prevent infinite recursion in psql when expanding a variable that refers to itself (Tom)

- Fix psql's `\copy` to not add spaces around a dot within `\copy (select ...)` (Tom)

  Addition of spaces around the decimal point in a numeric literal would result in a syntax error.

- Fix unnecessary "GIN indexes do not support whole-index scans" errors for unsatisfiable queries using `contrib/intarray` operators (Tom)

- Ensure that `contrib/pgstattuple` functions respond to cancel interrupts promptly (Tatsuhito Kasahara)

- Make server startup deal properly with the case that `shmget()` returns `EINVAL` for an existing shared memory segment (Tom)

  This behavior has been observed on BSD-derived kernels including OS X. It resulted in an entirely-misleading startup failure complaining that the shared memory request size was too large.

- Avoid possible crashes in syslogger process on Windows (Heikki)

- Deal more robustly with incomplete time zone information in the Windows registry (Magnus)
- Update the set of known Windows time zone names (Magnus)
- Update time zone data files to tzdata release 2010j for DST law changes in Argentina, Australian Antarctic, Bangladesh, Mexico, Morocco, Pakistan, Palestine, Russia, Syria, Tunisia; also historical corrections for Taiwan.

  Also, add `PKST` (Pakistan Summer Time) to the default set of timezone abbreviations.

E.111. Release 8.3.10

Release date: 2010-03-15

This release contains a variety of fixes from 8.3.9. For information about new features in the 8.3 major release, see 第 E.121 节.

E.111.1. Migration to Version 8.3.10

A dump/restore is not required for those running 8.3.X. However, if you are upgrading from a version earlier than 8.3.8, see 第 E.113 节.

E.111.2. Changes

- Add new configuration parameter `ssl_renegotiation_limit` to control how often we do session key renegotiation for an SSL connection (Magnus)

  This can be set to zero to disable renegotiation completely, which may be required if a broken SSL library is used. In particular, some vendors are shipping stopgap patches for CVE-2009-3555 that cause renegotiation attempts to fail.
- Fix possible deadlock during backend startup (Tom)
- Fix possible crashes due to not handling errors during relcache reload cleanly (Tom)
- Fix possible crash due to use of dangling pointer to a cached plan (Tatsuo)
- Fix possible crashes when trying to recover from a failure in subtransaction start (Tom)
- Fix server memory leak associated with use of savepoints and a client encoding different from server's encoding (Tom)
- Fix incorrect WAL data emitted during end-of-recovery cleanup of a GIST index page split (Yoichi Hirai)

  This would result in index corruption, or even more likely an error during WAL replay, if we were unlucky enough to crash during end-of-recovery cleanup after having completed an incomplete GIST insertion.
- Make `substring()` for `bit` types treat any negative length as meaning “all the rest of the string” (Tom)

  The previous coding treated only -1 that way, and would produce an invalid result value for other negative values, possibly leading to a crash (CVE-2010-0442).
- Fix integer-to-bit-string conversions to handle the first fractional byte correctly when the output bit width is wider than the given integer by something other than a multiple of 8 bits (Tom)

- Fix some cases of pathologically slow regular expression matching (Tom)
- Fix assorted crashes in `xml` processing caused by sloppy memory management (Tom)

  This is a back-patch of changes first applied in 8.4. The 8.3 code was known buggy, but the new code was sufficiently different to not want to back-patch it until it had gotten some field testing.
- Fix bug with trying to update a field of an element of a composite-type array column (Tom)
- Fix the `STOP WAL LOCATION` entry in backup history files to report the next WAL segment's name when the end location is exactly at a segment boundary (Itagaki Takahiro)
- Fix some more cases of temporary-file leakage (Heikki)

  This corrects a problem introduced in the previous minor release. One case that failed is when a plpgsql function returning set is called within another function's exception handler.
- Improve constraint exclusion processing of boolean-variable cases, in particular make it possible to exclude a partition that has a “bool_column = false” constraint (Tom)
- When reading `pg_hba.conf` and related files, do not treat `@something` as a file inclusion request if the `@` appears inside quote marks; also, never treat `@` by itself as a file inclusion request (Tom)

  This prevents erratic behavior if a role or database name starts with `@`. If you need to include a file whose path name contains spaces, you can still do so, but you must write `@"/path to/file"` rather than putting the quotes around the whole construct.
- Prevent infinite loop on some platforms if a directory is named as an inclusion target in `pg_hba.conf` and related files (Tom)
- Fix possible infinite loop if `SSL_read` or `SSL_write` fails without setting `errno` (Tom)

  This is reportedly possible with some Windows versions of openssl.
- Disallow GSSAPI authentication on local connections, since it requires a hostname to function correctly (Magnus)
- Make ecpg report the proper SQLSTATE if the connection disappears (Michael)
- Fix psql's `numericlocale` option to not format strings it shouldn't in latex and troff output formats (Heikki)
- Make psql return the correct exit status (3) when `ON_ERROR_STOP` and `--single-transaction` are both specified and an error occurs during the implied `COMMIT` (Bruce)
- Fix plpgsql failure in one case where a composite column is set to NULL (Tom)
- Fix possible failure when calling PL/Perl functions from PL/PerlU or vice versa (Tim Bunce)
- Add `volatile` markings in PL/Python to avoid possible compiler-specific misbehavior (Zdenek Kotala)

- Ensure PL/Tcl initializes the Tcl interpreter fully (Tom)

  The only known symptom of this oversight is that the Tcl `clock` command misbehaves if using Tcl 8.5 or later.

- Prevent crash in `contrib/dblink` when too many key columns are specified to a `dblink_build_sql_*` function (Rushabh Lathia, Joe Conway)

- Allow zero-dimensional arrays in `contrib/ltree` operations (Tom)

  This case was formerly rejected as an error, but it's more convenient to treat it the same as a zero-element array. In particular this avoids unnecessary failures when an `ltree` operation is applied to the result of `ARRAY(SELECT ...)` and the sub-select returns no rows.

- Fix assorted crashes in `contrib/xml2` caused by sloppy memory management (Tom)

- Make building of `contrib/xml2` more robust on Windows (Andrew)

- Fix race condition in Windows signal handling (Radu Ilie)

  One known symptom of this bug is that rows in `pg_listener` could be dropped under heavy load.

- Update time zone data files to tzdata release 2010e for DST law changes in Bangladesh, Chile, Fiji, Mexico, Paraguay, Samoa.

E.112. Release 8.3.9

Release date: 2009-12-14

This release contains a variety of fixes from 8.3.8. For information about new features in the 8.3 major release, see 第 E.121 节.

E.112.1. Migration to Version 8.3.9

A dump/restore is not required for those running 8.3.X. However, if you are upgrading from a version earlier than 8.3.8, see 第 E.113 节.

E.112.2. Changes

- Protect against indirect security threats caused by index functions changing session-local state (Gurjeet Singh, Tom)

  This change prevents allegedly-immutable index functions from possibly subverting a superuser's session (CVE-2009-4136).

- Reject SSL certificates containing an embedded null byte in the common name (CN) field (Magnus)

  This prevents unintended matching of a certificate to a server or client name during SSL validation (CVE-2009-4034).

- Fix possible crash during backend-startup-time cache initialization (Tom)

- Avoid crash on empty thesaurus dictionary (Tom)

- Prevent signals from interrupting `VACUUM` at unsafe times (Alvaro)

This fix prevents a PANIC if a `VACUUM FULL` is canceled after it's already committed its tuple movements, as well as transient errors if a plain `VACUUM` is interrupted after having truncated the table.

- Fix possible crash due to integer overflow in hash table size calculation (Tom)

  This could occur with extremely large planner estimates for the size of a hashjoin's result.

- Fix very rare crash in `inet/cidr` comparisons (Chris Mikkelson)

- Ensure that shared tuple-level locks held by prepared transactions are not ignored (Heikki)

- Fix premature drop of temporary files used for a cursor that is accessed within a subtransaction (Heikki)

- Fix memory leak in syslogger process when rotating to a new CSV logfile (Tom)

- Fix Windows permission-downgrade logic (Jesse Morris)

  This fixes some cases where the database failed to start on Windows, often with misleading error messages such as “could not locate matching postgres executable” .

- Fix incorrect logic for GiST index page splits, when the split depends on a non-first column of the index (Paul Ramsey)

- Don't error out if recycling or removing an old WAL file fails at the end of checkpoint (Heikki)

  It's better to treat the problem as non-fatal and allow the checkpoint to complete. Future checkpoints will retry the removal. Such problems are not expected in normal operation, but have been seen to be caused by misdesigned Windows anti-virus and backup software.

- Ensure WAL files aren't repeatedly archived on Windows (Heikki)

  This is another symptom that could happen if some other process interfered with deletion of a no-longer-needed file.

- Fix PAM password processing to be more robust (Tom)

  The previous code is known to fail with the combination of the Linux `pam_krb5` PAM module with Microsoft Active Directory as the domain controller. It might have problems elsewhere too, since it was making unjustified assumptions about what arguments the PAM stack would pass to it.

- Raise the maximum authentication token (Kerberos ticket) size in GSSAPI and SSPI authentication methods (Ian Turner)

  While the old 2000-byte limit was more than enough for Unix Kerberos implementations, tickets issued by Windows Domain Controllers can be much larger.

- Re-enable collection of access statistics for sequences (Akira Kurosawa)

  This used to work but was broken in 8.3.

- Fix processing of ownership dependencies during `CREATE OR REPLACE FUNCTION` (Tom)

- Fix incorrect handling of `WHERE x=x` conditions (Tom)

  In some cases these could get ignored as redundant, but they aren't – they're equivalent to `x IS NOT NULL`.

- Make text search parser accept underscores in XML attributes (Peter)

- Fix encoding handling in `xml` binary input (Heikki)

  If the XML header doesn't specify an encoding, we now assume UTF-8 by default; the previous handling was inconsistent.

- Fix bug with calling `plperl` from `plperlu` or vice versa (Tom)

  An error exit from the inner function could result in crashes due to failure to re-select the correct Perl interpreter for the outer function.

- Fix session-lifespan memory leak when a PL/Perl function is redefined (Tom)

- Ensure that Perl arrays are properly converted to PostgreSQL arrays when returned by a set-returning PL/Perl function (Andrew Dunstan, Abhijit Menon-Sen)

  This worked correctly already for non-set-returning functions.

- Fix rare crash in exception processing in PL/Python (Peter)

- In `contrib/pg_standby`, disable triggering failover with a signal on Windows (Fujii Masao)

  This never did anything useful, because Windows doesn't have Unix-style signals, but recent changes made it actually crash.

- Ensure psql's flex module is compiled with the correct system header definitions (Tom)

  This fixes build failures on platforms where `--enable-largefile` causes incompatible changes in the generated code.

- Make the postmaster ignore any `application_name` parameter in connection request packets, to improve compatibility with future libpq versions (Tom)

- Update the timezone abbreviation files to match current reality (Joachim Wieland)

  This includes adding `IDT` and `SGT` to the default timezone abbreviation set.

- Update time zone data files to tzdata release 2009s for DST law changes in Antarctica, Argentina, Bangladesh, Fiji, Novokuznetsk, Pakistan, Palestine, Samoa, Syria; also historical corrections for Hong Kong.

E.113. Release 8.3.8

Release date: 2009-09-09

This release contains a variety of fixes from 8.3.7. For information about new features in the 8.3 major release, see 第 E.121 节.

E.113.1. Migration to Version 8.3.8

A dump/restore is not required for those running 8.3.X. However, if you have any hash indexes on `interval` columns, you must `REINDEX` them after updating to 8.3.8. Also, if you are upgrading from a version earlier than 8.3.5, see 第 E.116 节.

E.113.2. Changes

- Fix Windows shared-memory allocation code (Tsutomu Yamada, Magnus)

  This bug led to the often-reported "could not reattach to shared memory" error message.

- Force WAL segment switch during `pg_start_backup()` (Heikki)

  This avoids corner cases that could render a base backup unusable.

- Disallow `RESET ROLE` and `RESET SESSION AUTHORIZATION` inside security-definer functions (Tom, Heikki)

  This covers a case that was missed in the previous patch that disallowed `SET ROLE` and `SET SESSION AUTHORIZATION` inside security-definer functions. (See CVE-2007-6600)

- Make `LOAD` of an already-loaded loadable module into a no-op (Tom)

  Formerly, `LOAD` would attempt to unload and re-load the module, but this is unsafe and not all that useful.

- Disallow empty passwords during LDAP authentication (Magnus)

- Fix handling of sub-SELECTs appearing in the arguments of an outer-level aggregate function (Tom)

- Fix bugs associated with fetching a whole-row value from the output of a Sort or Materialize plan node (Tom)

- Prevent `synchronize_seqscans` from changing the results of scrollable and `WITH HOLD` cursors (Tom)

- Revert planner change that disabled partial-index and constraint exclusion optimizations when there were more than 100 clauses in an AND or OR list (Tom)

- Fix hash calculation for data type `interval` (Tom)

  This corrects wrong results for hash joins on interval values. It also changes the contents of hash indexes on interval columns. If you have any such indexes, you must `REINDEX` them after updating.

- Treat `to_char(..., 'TH')` as an uppercase ordinal suffix with `'HH'/'HH12'` (Heikki)

  It was previously handled as `'th'` (lowercase).

- Fix overflow for `INTERVAL 'x ms'` when *x* is more than 2 million and integer datetimes are in use (Alex Hunsaker)

- Fix calculation of distance between a point and a line segment (Tom)

  This led to incorrect results from a number of geometric operators.

- Fix `money` data type to work in locales where currency amounts have no fractional digits, e.g. Japan (Itagaki Takahiro)

- Fix `LIKE` for case where pattern contains `%_` (Tom)

- Properly round datetime input like `00:12:57.999999999999999999999999999` (Tom)

- Fix memory leaks in XML operations (Tom)
- Fix poor choice of page split point in GiST R-tree operator classes (Teodor)
- Ensure that a “fast shutdown” request will forcibly terminate open sessions, even if a “smart shutdown” was already in progress (Fujii Masao)
- Avoid performance degradation in bulk inserts into GIN indexes when the input values are (nearly) in sorted order (Tom)
- Correctly enforce NOT NULL domain constraints in some contexts in PL/pgSQL (Tom)
- Fix portability issues in plperl initialization (Andrew Dunstan)
- Fix pg_ctl to not go into an infinite loop if `postgresql.conf` is empty (Jeff Davis)
- Improve pg_dump's efficiency when there are many large objects (Tamas Vincze)
- Use `SIGUSR1`, not `SIGQUIT`, as the failover signal for pg_standby (Heikki)
- Make pg_standby's `maxretries` option behave as documented (Fujii Masao)
- Make `contrib/hstore` throw an error when a key or value is too long to fit in its data structure, rather than silently truncating it (Andrew Gierth)
- Fix `contrib/xml2`'s `xslt_process()` to properly handle the maximum number of parameters (twenty) (Tom)
- Improve robustness of libpq's code to recover from errors during `COPY FROM STDIN` (Tom)
- Avoid including conflicting readline and editline header files when both libraries are installed (Zdenek Kotala)
- Update time zone data files to tzdata release 2009l for DST law changes in Bangladesh, Egypt, Jordan, Pakistan, Argentina/San_Luis, Cuba, Jordan (historical correction only), Mauritius, Morocco, Palestine, Syria, Tunisia.

E.114. Release 8.3.7

Release date: 2009-03-16

This release contains a variety of fixes from 8.3.6. For information about new features in the 8.3 major release, see 第 E.121 节.

E.114.1. Migration to Version 8.3.7

A dump/restore is not required for those running 8.3.X. However, if you are upgrading from a version earlier than 8.3.5, see 第 E.116 节.

E.114.2. Changes

- Prevent error recursion crashes when encoding conversion fails (Tom)

  This change extends fixes made in the last two minor releases for related failure scenarios. The previous fixes were narrowly tailored for the original problem reports, but we have now recognized that any error thrown by an encoding conversion function could potentially lead to infinite recursion while trying to report the error. The solution therefore is to disable translation and encoding conversion and report the plain-ASCII form of any error message, if we find we have gotten into a recursive error reporting situation. (CVE-2009-0922)

- Disallow `CREATE CONVERSION` with the wrong encodings for the specified conversion function (Heikki)

  This prevents one possible scenario for encoding conversion failure. The previous change is a backstop to guard against other kinds of failures in the same area.

- Fix `xpath()` to not modify the path expression unless necessary, and to make a saner attempt at it when necessary (Andrew)

  The SQL standard suggests that `xpath` should work on data that is a document fragment, but libxml doesn't support that, and indeed it's not clear that this is sensible according to the XPath standard. `xpath` attempted to work around this mismatch by modifying both the data and the path expression, but the modification was buggy and could cause valid searches to fail. Now, `xpath` checks whether the data is in fact a well-formed document, and if so invokes libxml with no change to the data or path expression. Otherwise, a different modification method that is somewhat less likely to fail is used.

  > **注意**
  >
  > The new modification method is still not 100% satisfactory, and it seems likely that no real solution is possible. This patch should therefore be viewed as a band-aid to keep from breaking existing applications unnecessarily. It is likely that PostgreSQL 8.4 will simply reject use of `xpath` on data that is not a well-formed document.

- Fix core dump when `to_char()` is given format codes that are inappropriate for the type of the data argument (Tom)

- Fix possible failure in text search when C locale is used with a multi-byte encoding (Teodor)

  Crashes were possible on platforms where `wchar_t` is narrower than `int`; Windows in particular.

- Fix extreme inefficiency in text search parser's handling of an email-like string containing multiple `@` characters (Heikki)

- Fix planner problem with sub-`SELECT` in the output list of a larger subquery (Tom)

  The known symptom of this bug is a “failed to locate grouping columns” error that is dependent on the datatype involved; but there could be other issues as well.

- Fix decompilation of `CASE WHEN` with an implicit coercion (Tom)

  This mistake could lead to Assert failures in an Assert-enabled build, or an “unexpected CASE WHEN clause” error message in other cases, when trying to examine or dump a view.

- Fix possible misassignment of the owner of a TOAST table's rowtype (Tom)

  If `CLUSTER` or a rewriting variant of `ALTER TABLE` were executed by someone other than the table owner, the `pg_type` entry for the table's TOAST table would end up marked as owned by that someone. This caused no immediate problems, since the permissions on the TOAST rowtype aren't examined by any ordinary database operation. However, it could lead to unexpected failures if one later tried to drop the role that issued the command (in 8.1 or 8.2), or “owner of data type appears to be invalid” warnings from pg_dump after having done so (in 8.3).

- Change `UNLISTEN` to exit quickly if the current session has never executed any `LISTEN` command (Tom)

  Most of the time this is not a particularly useful optimization, but since `DISCARD ALL` invokes `UNLISTEN`, the previous coding caused a substantial performance problem for applications that made heavy use of `DISCARD ALL`.

- Fix PL/pgSQL to not treat `INTO` after `INSERT` as an INTO-variables clause anywhere in the string, not only at the start; in particular, don't fail for `INSERT INTO` within `CREATE RULE` (Tom)

- Clean up PL/pgSQL error status variables fully at block exit (Ashesh Vashi and Dave Page)

  This is not a problem for PL/pgSQL itself, but the omission could cause the PL/pgSQL Debugger to crash while examining the state of a function.

- Retry failed calls to `CallNamedPipe()` on Windows (Steve Marshall, Magnus)

  It appears that this function can sometimes fail transiently; we previously treated any failure as a hard error, which could confuse `LISTEN`/`NOTIFY` as well as other operations.

- Add `MUST` (Mauritius Island Summer Time) to the default list of known timezone abbreviations (Xavier Bugaud)

E.115. Release 8.3.6

Release date: 2009-02-02

This release contains a variety of fixes from 8.3.5. For information about new features in the 8.3 major release, see 第 E.121 节.

E.115.1. Migration to Version 8.3.6

A dump/restore is not required for those running 8.3.X. However, if you are upgrading from a version earlier than 8.3.5, see 第 E.116 节.

E.115.2. Changes

- Make `DISCARD ALL` release advisory locks, in addition to everything it already did (Tom)

  This was decided to be the most appropriate behavior. This could affect existing applications, however.

- Fix whole-index GiST scans to work correctly (Teodor)

  This error could cause rows to be lost if a table is clustered on a GiST index.

- Fix crash of `xmlconcat(NULL)` (Peter)

- Fix possible crash in `ispell` dictionary if high-bit-set characters are used as flags (Teodor)

  This is known to be done by one widely available Norwegian dictionary, and the same condition may exist in others.

- Fix misordering of pg_dump output for composite types (Tom)

The most likely problem was for user-defined operator classes to be dumped after indexes or views that needed them.

- Improve handling of URLs in `headline()` function (Teodor)
- Improve handling of overlength headlines in `headline()` function (Teodor)
- Prevent possible Assert failure or misconversion if an encoding conversion is created with the wrong conversion function for the specified pair of encodings (Tom, Heikki)
- Fix possible Assert failure if a statement executed in PL/pgSQL is rewritten into another kind of statement, for example if an `INSERT` is rewritten into an `UPDATE` (Heikki)
- Ensure that a snapshot is available to datatype input functions (Tom)

  This primarily affects domains that are declared with `CHECK` constraints involving user-defined stable or immutable functions. Such functions typically fail if no snapshot has been set.
- Make it safer for SPI-using functions to be used within datatype I/O; in particular, to be used in domain check constraints (Tom)
- Avoid unnecessary locking of small tables in `VACUUM` (Heikki)
- Fix a problem that sometimes kept `ALTER TABLE ENABLE/DISABLE RULE` from being recognized by active sessions (Tom)
- Fix a problem that made `UPDATE RETURNING tableoid` return zero instead of the correct OID (Tom)
- Allow functions declared as taking `ANYARRAY` to work on the `pg_statistic` columns of that type (Tom)

  This used to work, but was unintentionally broken in 8.3.
- Fix planner misestimation of selectivity when transitive equality is applied to an outer-join clause (Tom)

  This could result in bad plans for queries like `... from a left join b on a.a1 = b.b1 where a.a1 = 42 ...`
- Improve optimizer's handling of long `IN` lists (Tom)

  This change avoids wasting large amounts of time on such lists when constraint exclusion is enabled.
- Prevent synchronous scan during GIN index build (Tom)

  Because GIN is optimized for inserting tuples in increasing TID order, choosing to use a synchronous scan could slow the build by a factor of three or more.
- Ensure that the contents of a holdable cursor don't depend on the contents of TOAST tables (Tom)

  Previously, large field values in a cursor result might be represented as TOAST pointers, which would fail if the referenced table got dropped before the cursor is read, or if the large value is deleted and then vacuumed away. This cannot happen with an ordinary cursor, but it could with a cursor that is held past its creating transaction.

- Fix memory leak when a set-returning function is terminated without reading its whole result (Tom)
- Fix encoding conversion problems in XML functions when the database encoding isn't UTF-8 (Tom)
- Fix `contrib/dblink`'s `dblink_get_result(text,bool)` function (Joe)
- Fix possible garbage output from `contrib/sslinfo` functions (Tom)
- Fix incorrect behavior of `contrib/tsearch2` compatibility trigger when it's fired more than once in a command (Teodor)
- Fix possible mis-signaling in autovacuum (Heikki)
- Support running as a service on Windows 7 beta (Dave and Magnus)
- Fix ecpg's handling of varchar structs (Michael)
- Fix configure script to properly report failure when unable to obtain linkage information for PL/Perl (Andrew)
- Make all documentation reference `pgsql-bugs` and/or `pgsql-hackers` as appropriate, instead of the now-decommissioned `pgsql-ports` and `pgsql-patches` mailing lists (Tom)
- Update time zone data files to tzdata release 2009a (for Kathmandu and historical DST corrections in Switzerland, Cuba)

E.116. Release 8.3.5

Release date: 2008-11-03

This release contains a variety of fixes from 8.3.4. For information about new features in the 8.3 major release, see 第 E.121 节.

E.116.1. Migration to Version 8.3.5

A dump/restore is not required for those running 8.3.X. However, if you are upgrading from a version earlier than 8.3.1, see 第 E.120 节. Also, if you were running a previous 8.3.X release, it is recommended to `REINDEX` all GiST indexes after the upgrade.

E.116.2. Changes

- Fix GiST index corruption due to marking the wrong index entry "dead" after a deletion (Teodor)

  This would result in index searches failing to find rows they should have found. Corrupted indexes can be fixed with `REINDEX`.

- Fix backend crash when the client encoding cannot represent a localized error message (Tom)

  We have addressed similar issues before, but it would still fail if the "character has no equivalent" message itself couldn't be converted. The fix is to disable localization and send the plain ASCII error message when we detect such a situation.

- Fix possible crash in `bytea`-to-XML mapping (Michael McMaster)
- Fix possible crash when deeply nested functions are invoked from a trigger (Tom)

- Improve optimization of *expression* IN (*expression-list*) queries (Tom, per an idea from Robert Haas)

  Cases in which there are query variables on the right-hand side had been handled less efficiently in 8.2.x and 8.3.x than in prior versions. The fix restores 8.1 behavior for such cases.

- Fix mis-expansion of rule queries when a sub-SELECT appears in a function call in FROM, a multi-row VALUES list, or a RETURNING list (Tom)

  The usual symptom of this problem is an "unrecognized node type" error.

- Fix Assert failure during rescan of an IS NULL search of a GiST index (Teodor)

- Fix memory leak during rescan of a hashed aggregation plan (Neil)

- Ensure an error is reported when a newly-defined PL/pgSQL trigger function is invoked as a normal function (Tom)

- Force a checkpoint before CREATE DATABASE starts to copy files (Heikki)

  This prevents a possible failure if files had recently been deleted in the source database.

- Prevent possible collision of relfilenode numbers when moving a table to another tablespace with ALTER SET TABLESPACE (Heikki)

  The command tried to re-use the existing filename, instead of picking one that is known unused in the destination directory.

- Fix incorrect text search headline generation when single query item matches first word of text (Sushant Sinha)

- Fix improper display of fractional seconds in interval values when using a non-ISO datestyle in an --enable-integer-datetimes build (Ron Mayer)

- Make ILIKE compare characters case-insensitively even when they're escaped (Andrew)

- Ensure DISCARD is handled properly by statement logging (Tom)

- Fix incorrect logging of last-completed-transaction time during PITR recovery (Tom)

- Ensure SPI_getvalue and SPI_getbinval behave correctly when the passed tuple and tuple descriptor have different numbers of columns (Tom)

  This situation is normal when a table has had columns added or removed, but these two functions didn't handle it properly. The only likely consequence is an incorrect error indication.

- Mark SessionReplicationRole as PGDLLIMPORT so it can be used by Slony on Windows (Magnus)

- Fix small memory leak when using libpq's gsslib parameter (Magnus)

  The space used by the parameter string was not freed at connection close.

- Ensure libgssapi is linked into libpq if needed (Markus Schaaf)

- Fix ecpg's parsing of CREATE ROLE (Michael)

- Fix recent breakage of pg_ctl restart (Tom)

- Ensure `pg_control` is opened in binary mode (Itagaki Takahiro)

  pg_controldata and pg_resetxlog did this incorrectly, and so could fail on Windows.

- Update time zone data files to tzdata release 2008i (for DST law changes in Argentina, Brazil, Mauritius, Syria)

E.117. Release 8.3.4

Release date: 2008-09-22

This release contains a variety of fixes from 8.3.3. For information about new features in the 8.3 major release, see 第 E.121 节.

E.117.1. Migration to Version 8.3.4

A dump/restore is not required for those running 8.3.X. However, if you are upgrading from a version earlier than 8.3.1, see 第 E.120 节.

E.117.2. Changes

- Fix bug in btree WAL recovery code (Heikki)

  Recovery failed if the WAL ended partway through a page split operation.

- Fix potential use of wrong cutoff XID for HOT page pruning (Alvaro)

  This error created a risk of corruption in system catalogs that are consulted by `VACUUM`: dead tuple versions might be removed too soon. The impact of this on actual database operations would be minimal, since the system doesn't follow MVCC rules while examining catalogs, but it might result in transiently wrong output from pg_dump or other client programs.

- Fix potential miscalculation of `datfrozenxid` (Alvaro)

  This error may explain some recent reports of failure to remove old `pg_clog` data.

- Fix incorrect HOT updates after `pg_class` is reindexed (Tom)

  Corruption of `pg_class` could occur if `REINDEX TABLE pg_class` was followed in the same session by an `ALTER TABLE RENAME` or `ALTER TABLE SET SCHEMA` command.

- Fix missed "combo cid" case (Karl Schnaitter)

  This error made rows incorrectly invisible to a transaction in which they had been deleted by multiple subtransactions that all aborted.

- Prevent autovacuum from crashing if the table it's currently checking is deleted at just the wrong time (Alvaro)

- Widen local lock counters from 32 to 64 bits (Tom)

  This responds to reports that the counters could overflow in sufficiently long transactions, leading to unexpected "lock is already held" errors.

- Fix possible duplicate output of tuples during a GiST index scan (Teodor)

- Regenerate foreign key checking queries from scratch when either table is modified (Tom)

Previously, 8.3 would attempt to replan the query, but would work from previously generated query text. This led to failures if a table or column was renamed.

- Fix missed permissions checks when a view contains a simple `UNION ALL` construct (Heikki)

  Permissions for the referenced tables were checked properly, but not permissions for the view itself.

- Add checks in executor startup to ensure that the tuples produced by an `INSERT` or `UPDATE` will match the target table's current rowtype (Tom)

  This situation is believed to be impossible in 8.3, but it can happen in prior releases, so a check seems prudent.

- Fix possible repeated drops during `DROP OWNED` (Tom)

  This would typically result in strange errors such as "cache lookup failed for relation NNN".

- Fix several memory leaks in XML operations (Kris Jurka, Tom)

- Fix `xmlserialize()` to raise error properly for unacceptable target data type (Tom)

- Fix a couple of places that mis-handled multibyte characters in text search configuration file parsing (Tom)

  Certain characters occurring in configuration files would always cause "invalid byte sequence for encoding" failures.

- Provide file name and line number location for all errors reported in text search configuration files (Tom)

- Fix `AT TIME ZONE` to first try to interpret its timezone argument as a timezone abbreviation, and only try it as a full timezone name if that fails, rather than the other way around as formerly (Tom)

  The timestamp input functions have always resolved ambiguous zone names in this order. Making `AT TIME ZONE` do so as well improves consistency, and fixes a compatibility bug introduced in 8.1: in ambiguous cases we now behave the same as 8.0 and before did, since in the older versions `AT TIME ZONE` accepted only abbreviations.

- Fix datetime input functions to correctly detect integer overflow when running on a 64-bit platform (Tom)

- Prevent integer overflows during units conversion when displaying a configuration parameter that has units (Tom)

- Improve performance of writing very long log messages to syslog (Tom)

- Allow spaces in the suffix part of an LDAP URL in `pg_hba.conf` (Tom)

- Fix bug in backwards scanning of a cursor on a `SELECT DISTINCT ON` query (Tom)

- Fix planner bug that could improperly push down `IS NULL` tests below an outer join (Tom)

  This was triggered by occurrence of `IS NULL` tests for the same relation in all arms of an upper `OR` clause.

- Fix planner bug with nested sub-select expressions (Tom)

  If the outer sub-select has no direct dependency on the parent query, but the inner one does, the outer value might not get recalculated for new parent query rows.

- Fix planner to estimate that `GROUP BY` expressions yielding boolean results always result in two groups, regardless of the expressions' contents (Tom)

  This is very substantially more accurate than the regular `GROUP BY` estimate for certain boolean tests like *col* `IS NULL`.

- Fix PL/pgSQL to not fail when a `FOR` loop's target variable is a record containing composite-type fields (Tom)

- Fix PL/Tcl to behave correctly with Tcl 8.5, and to be more careful about the encoding of data sent to or from Tcl (Tom)

- Improve performance of `PQescapeBytea()` (Rudolf Leitgeb)

- On Windows, work around a Microsoft bug by preventing libpq from trying to send more than 64kB per system call (Magnus)

- Fix ecpg to handle variables properly in `SET` commands (Michael)

- Improve pg_dump and pg_restore's error reporting after failure to send a SQL command (Tom)

- Fix pg_ctl to properly preserve postmaster command-line arguments across a `restart` (Bruce)

- Fix erroneous WAL file cutoff point calculation in pg_standby (Simon)

- Update time zone data files to tzdata release 2008f (for DST law changes in Argentina, Bahamas, Brazil, Mauritius, Morocco, Pakistan, Palestine, and Paraguay)

E.118. Release 8.3.3

Release date: 2008-06-12

This release contains one serious and one minor bug fix over 8.3.2. For information about new features in the 8.3 major release, see 第 E.121 节.

E.118.1. Migration to Version 8.3.3

A dump/restore is not required for those running 8.3.X. However, if you are upgrading from a version earlier than 8.3.1, see 第 E.120 节.

E.118.2. Changes

- Make `pg_get_ruledef()` parenthesize negative constants (Tom)

  Before this fix, a negative constant in a view or rule might be dumped as, say, `-42::integer`, which is subtly incorrect: it should be `(-42)::integer` due to operator precedence rules. Usually this would make little difference, but it could interact with another recent patch to cause PostgreSQL to reject what had been a valid `SELECT DISTINCT` view query. Since this could result in pg_dump output failing to reload, it is being treated as a high-priority fix. The only released versions in which dump output is actually incorrect are 8.3.1 and 8.2.7.

- Make `ALTER AGGREGATE ... OWNER TO` update `pg_shdepend` (Tom)

This oversight could lead to problems if the aggregate was later involved in a `DROP OWNED` or `REASSIGN OWNED` operation.

E.119. Release 8.3.2

Release date: never released

This release contains a variety of fixes from 8.3.1. For information about new features in the 8.3 major release, see 第 E.121 节.

E.119.1. Migration to Version 8.3.2

A dump/restore is not required for those running 8.3.X. However, if you are upgrading from a version earlier than 8.3.1, see 第 E.120 节.

E.119.2. Changes

- Fix `ERRORDATA_STACK_SIZE exceeded` crash that occurred on Windows when using UTF-8 database encoding and a different client encoding (Tom)
- Fix incorrect archive truncation point calculation for the `%r` macro in `recovery_command` parameters (Simon)

  This could lead to data loss if a warm-standby script relied on `%r` to decide when to throw away WAL segment files.
- Fix `ALTER TABLE ADD COLUMN ... PRIMARY KEY` so that the new column is correctly checked to see if it's been initialized to all non-nulls (Brendan Jurd)

  Previous versions neglected to check this requirement at all.
- Fix `REASSIGN OWNED` so that it works on procedural languages too (Alvaro)
- Fix problems with `SELECT FOR UPDATE/SHARE` occurring as a subquery in a query with a non-`SELECT` top-level operation (Tom)
- Fix possible `CREATE TABLE` failure when inheriting the "same" constraint from multiple parent relations that inherited that constraint from a common ancestor (Tom)
- Fix `pg_get_ruledef()` to show the alias, if any, attached to the target table of an `UPDATE` or `DELETE` (Tom)
- Restore the pre-8.3 behavior that an out-of-range block number in a TID being used in a TidScan plan results in silently not matching any rows (Tom)

  8.3.0 and 8.3.1 threw an error instead.
- Fix GIN bug that could result in a `too many LWLocks taken` failure (Teodor)
- Fix broken GiST comparison function for `tsquery` (Teodor)
- Fix `tsvector_update_trigger()` and `ts_stat()` to accept domains over the types they expect to work with (Tom)
- Fix failure to support enum data types as foreign keys (Tom)
- Avoid possible crash when decompressing corrupted data (Zdenek Kotala)
- Fix race conditions between delayed unlinks and `DROP DATABASE` (Heikki)

In the worst case this could result in deleting a newly created table in a new database that happened to get the same OID as the recently-dropped one; but of course that is an extremely low-probability scenario.

- Repair two places where SIGTERM exit of a backend could leave corrupted state in shared memory (Tom)

  Neither case is very important if SIGTERM is used to shut down the whole database cluster together, but there was a problem if someone tried to SIGTERM individual backends.

- Fix possible crash due to incorrect plan generated for an `x IN (SELECT y FROM ...)` clause when `x` and `y` have different data types; and make sure the behavior is semantically correct when the conversion from `y`'s type to `x`'s type is lossy (Tom)

- Fix oversight that prevented the planner from substituting known Param values as if they were constants (Tom)

  This mistake partially disabled optimization of unnamed extended-Query statements in 8.3.0 and 8.3.1: in particular the LIKE-to-indexscan optimization would never be applied if the LIKE pattern was passed as a parameter, and constraint exclusion depending on a parameter value didn't work either.

- Fix planner failure when an indexable `MIN` or `MAX` aggregate is used with `DISTINCT` or `ORDER BY` (Tom)

- Fix planner to ensure it never uses a "physical tlist" for a plan node that is feeding a Sort node (Tom)

  This led to the sort having to push around more data than it really needed to, since unused column values were included in the sorted data.

- Avoid unnecessary copying of query strings (Tom)

  This fixes a performance problem introduced in 8.3.0 when a very large number of commands are submitted as a single query string.

- Make `TransactionIdIsCurrentTransactionId()` use binary search instead of linear search when checking child-transaction XIDs (Heikki)

  This fixes some cases in which 8.3.0 was significantly slower than earlier releases.

- Fix conversions between ISO-8859-5 and other encodings to handle Cyrillic "Yo" characters (`e` and `E` with two dots) (Sergey Burladyan)

- Fix several datatype input functions, notably `array_in()`, that were allowing unused bytes in their results to contain uninitialized, unpredictable values (Tom)

  This could lead to failures in which two apparently identical literal values were not seen as equal, resulting in the parser complaining about unmatched `ORDER BY` and `DISTINCT` expressions.

- Fix a corner case in regular-expression substring matching (`substring(string from pattern)`) (Tom)

  The problem occurs when there is a match to the pattern overall but the user has specified a parenthesized subexpression and that subexpression hasn't got a match. An example is `substring('foo' from 'foo(bar)?')`. This should return NULL, since `(bar)` isn't matched, but it was mistakenly returning the whole-pattern match instead (ie, `foo`).

- Prevent cancellation of an auto-vacuum that was launched to prevent XID wraparound (Alvaro)
- Improve ANALYZE's handling of in-doubt tuples (those inserted or deleted by a not-yet-committed transaction) so that the counts it reports to the stats collector are more likely to be correct (Pavan Deolasee)
- Fix initdb to reject a relative path for its --xlogdir (-X) option (Tom)
- Make psql print tab characters as an appropriate number of spaces, rather than \x09 as was done in 8.3.0 and 8.3.1 (Bruce)
- Update time zone data files to tzdata release 2008c (for DST law changes in Morocco, Iraq, Choibalsan, Pakistan, Syria, Cuba, and Argentina/San_Luis)
- Add ECPGget_PGconn() function to ecpglib (Michael)
- Fix incorrect result from ecpg's PGTYPEStimestamp_sub() function (Michael)
- Fix handling of continuation line markers in ecpg (Michael)
- Fix possible crashes in contrib/cube functions (Tom)
- Fix core dump in contrib/xml2's xpath_table() function when the input query returns a NULL value (Tom)
- Fix contrib/xml2's makefile to not override CFLAGS, and make it auto-configure properly for libxslt present or not (Tom)

E.120. Release 8.3.1

Release date: 2008-03-17

This release contains a variety of fixes from 8.3.0. For information about new features in the 8.3 major release, see 第 E.121 节.

E.120.1. Migration to Version 8.3.1

A dump/restore is not required for those running 8.3.X. However, you might need to REINDEX indexes on textual columns after updating, if you are affected by the Windows locale issue described below.

E.120.2. Changes

- Fix character string comparison for Windows locales that consider different character combinations as equal (Tom)

  This fix applies only on Windows and only when using UTF-8 database encoding. The same fix was made for all other cases over two years ago, but Windows with UTF-8 uses a separate code path that was not updated. If you are using a locale that considers some non-identical strings as equal, you may need to REINDEX to fix existing indexes on textual columns.

- Repair corner-case bugs in VACUUM FULL (Tom)

  A potential deadlock between concurrent VACUUM FULL operations on different system catalogs was introduced in 8.2. This has now been corrected. 8.3 made this worse because the deadlock could occur within a critical code section, making it a PANIC rather than just ERROR condition.

Also, a VACUUM FULL that failed partway through vacuuming a system catalog could result in cache corruption in concurrent database sessions.

Another VACUUM FULL bug introduced in 8.3 could result in a crash or out-of-memory report when dealing with pages containing no live tuples.

- Fix misbehavior of foreign key checks involving character or bit columns (Tom)

  If the referencing column were of a different but compatible type (for instance varchar), the constraint was enforced incorrectly.

- Avoid needless deadlock failures in no-op foreign-key checks (Stephan Szabo, Tom)

- Fix possible core dump when re-planning a prepared query (Tom)

  This bug affected only protocol-level prepare operations, not SQL PREPARE, and so tended to be seen only with JDBC, DBI, and other client-side drivers that use prepared statements heavily.

- Fix possible failure when re-planning a query that calls an SPI-using function (Tom)

- Fix failure in row-wise comparisons involving columns of different datatypes (Tom)

- Fix longstanding LISTEN/NOTIFY race condition (Tom)

  In rare cases a session that had just executed a LISTEN might not get a notification, even though one would be expected because the concurrent transaction executing NOTIFY was observed to commit later.

  A side effect of the fix is that a transaction that has executed a not-yet-committed LISTEN command will not see any row in pg_listener for the LISTEN, should it choose to look; formerly it would have. This behavior was never documented one way or the other, but it is possible that some applications depend on the old behavior.

- Disallow LISTEN and UNLISTEN within a prepared transaction (Tom)

  This was formerly allowed but trying to do it had various unpleasant consequences, notably that the originating backend could not exit as long as an UNLISTEN remained uncommitted.

- Disallow dropping a temporary table within a prepared transaction (Heikki)

  This was correctly disallowed by 8.1, but the check was inadvertently broken in 8.2 and 8.3.

- Fix rare crash when an error occurs during a query using a hash index (Heikki)

- Fix incorrect comparison of tsquery values (Teodor)

- Fix incorrect behavior of LIKE with non-ASCII characters in single-byte encodings (Rolf Jentsch)

- Disable xmlvalidate (Tom)

  This function should have been removed before 8.3 release, but was inadvertently left in the source code. It poses a small security risk since unprivileged users could use it to read the first few characters of any file accessible to the server.

- Fix memory leaks in certain usages of set-returning functions (Neil)

- Make `encode(bytea, 'escape')` convert all high-bit-set byte values into `\nnn` octal escape sequences (Tom)

  This is necessary to avoid encoding problems when the database encoding is multi-byte. This change could pose compatibility issues for applications that are expecting specific results from `encode`.

- Fix input of datetime values for February 29 in years BC (Tom)

  The former coding was mistaken about which years were leap years.

- Fix "unrecognized node type" error in some variants of `ALTER OWNER` (Tom)

- Avoid tablespace permissions errors in `CREATE TABLE LIKE INCLUDING INDEXES` (Tom)

- Ensure `pg_stat_activity.waiting` flag is cleared when a lock wait is aborted (Tom)

- Fix handling of process permissions on Windows Vista (Dave, Magnus)

  In particular, this fix allows starting the server as the Administrator user.

- Update time zone data files to tzdata release 2008a (in particular, recent Chile changes); adjust timezone abbreviation `VET` (Venezuela) to mean UTC-4:30, not UTC-4:00 (Tom)

- Fix ecpg problems with arrays (Michael)

- Fix pg_ctl to correctly extract the postmaster's port number from command-line options (Itagaki Takahiro, Tom)

  Previously, `pg_ctl start -w` could try to contact the postmaster on the wrong port, leading to bogus reports of startup failure.

- Use `-fwrapv` to defend against possible misoptimization in recent gcc versions (Tom)

  This is known to be necessary when building PostgreSQL with gcc 4.3 or later.

- Enable building `contrib/uuid-ossp` with MSVC (Hiroshi Saito)

E.121. Release 8.3

Release date: 2008-02-04

E.121.1. Overview

With significant new functionality and performance enhancements, this release represents a major leap forward for PostgreSQL. This was made possible by a growing community that has dramatically accelerated the pace of development. This release adds the following major features:

- Full text search is integrated into the core database system

- Support for the SQL/XML standard, including new operators and an `XML` data type

- Enumerated data types (`ENUM`)

- Arrays of composite types

- Universally Unique Identifier (`UUID`) data type

- Add control over whether `NULL`s sort first or last
- Updatable cursors
- Server configuration parameters can now be set on a per-function basis
- User-defined types can now have type modifiers
- Automatically re-plan cached queries when table definitions change or statistics are updated
- Numerous improvements in logging and statistics collection
- Support Security Service Provider Interface (SSPI) for authentication on Windows
- Support multiple concurrent autovacuum processes, and other autovacuum improvements
- Allow the whole PostgreSQL distribution to be compiled with Microsoft Visual C++

Major performance improvements are listed below. Most of these enhancements are automatic and do not require user changes or tuning:

- Asynchronous commit delays writes to WAL during transaction commit
- Checkpoint writes can be spread over a longer time period to smooth the I/O spike during each checkpoint
- Heap-Only Tuples (HOT) accelerate space reuse for most `UPDATE`s and `DELETE`s
- Just-in-time background writer strategy improves disk write efficiency
- Using non-persistent transaction IDs for read-only transactions reduces overhead and `VACUUM` requirements
- Per-field and per-row storage overhead has been reduced
- Large sequential scans no longer force out frequently used cached pages
- Concurrent large sequential scans can now share disk reads
- `ORDER BY ... LIMIT` can be done without sorting

The above items are explained in more detail in the sections below.

E.121.2. Migration to Version 8.3

A dump/restore using pg_dump is required for those wishing to migrate data from any previous release.

Observe the following incompatibilities:

E.121.2.1. General

- Non-character data types are no longer automatically cast to `TEXT` (Peter, Tom)

  Previously, if a non-character value was supplied to an operator or function that requires `text` input, it was automatically cast to `text`, for most (though not all) built-in data types. This no longer happens: an explicit cast to `text` is now required for all non-character-string types. For example, these expressions formerly worked:

  ```
  substr(current_date, 1, 4)
  ```

```
23 LIKE '2%'
```

but will now draw "function does not exist" and "operator does not exist" errors respectively. Use an explicit cast instead:

```
substr(current_date::text, 1, 4)
23::text LIKE '2%'
```

(Of course, you can use the more verbose `CAST()` syntax too.) The reason for the change is that these automatic casts too often caused surprising behavior. An example is that in previous releases, this expression was accepted but did not do what was expected:

```
current_date < 2017-11-17
```

This is actually comparing a date to an integer, which should be (and now is) rejected – but in the presence of automatic casts both sides were cast to `text` and a textual comparison was done, because the `text < text` operator was able to match the expression when no other `<` operator could.

Types `char(n)` and `varchar(n)` still cast to `text` automatically. Also, automatic casting to `text` still works for inputs to the concatenation (`||`) operator, so long as least one input is a character-string type.

- Full text search features from `contrib/tsearch2` have been moved into the core server, with some minor syntax changes

  `contrib/tsearch2` now contains a compatibility interface.

- `ARRAY(SELECT ...)`, where the `SELECT` returns no rows, now returns an empty array, rather than NULL (Tom)

- The array type name for a base data type is no longer always the base type's name with an underscore prefix

  The old naming convention is still honored when possible, but application code should no longer depend on it. Instead use the new `pg_type.typarray` column to identify the array data type associated with a given type.

- `ORDER BY ... USING` *`operator`* must now use a less-than or greater-than *`operator`* that is defined in a btree operator class

  This restriction was added to prevent inconsistent results.

- `SET LOCAL` changes now persist until the end of the outermost transaction, unless rolled back (Tom)

  Previously `SET LOCAL`'s effects were lost after subtransaction commit (`RELEASE SAVEPOINT` or exit from a PL/pgSQL exception block).

- Commands rejected in transaction blocks are now also rejected in multiple-statement query strings (Tom)

  For example, `"BEGIN; DROP DATABASE; COMMIT"` will now be rejected even if submitted as a single query message.

- `ROLLBACK` outside a transaction block now issues `NOTICE` instead of `WARNING` (Bruce)

- Prevent `NOTIFY/LISTEN/UNLISTEN` from accepting schema-qualified names (Bruce)

Formerly, these commands accepted `schema.relation` but ignored the schema part, which was confusing.

- `ALTER SEQUENCE` no longer affects the sequence's `currval()` state (Tom)
- Foreign keys now must match indexable conditions for cross-data-type references (Tom)

  This improves semantic consistency and helps avoid performance problems.
- Restrict object size functions to users who have reasonable permissions to view such information (Tom)

  For example, `pg_database_size()` now requires `CONNECT` permission, which is granted to everyone by default. `pg_tablespace_size()` requires `CREATE` permission in the tablespace, or is allowed if the tablespace is the default tablespace for the database.
- Remove the undocumented `!!=` (not in) operator (Tom)

  `NOT IN (SELECT ...)` is the proper way to perform this operation.
- Internal hashing functions are now more uniformly-distributed (Tom)

  If application code was computing and storing hash values using internal PostgreSQL hashing functions, the hash values must be regenerated.
- C-code conventions for handling variable-length data values have changed (Greg Stark, Tom)

  The new `SET_VARSIZE()` macro must be used to set the length of generated `varlena` values. Also, it might be necessary to expand (“de-TOAST”) input values in more cases.
- Continuous archiving no longer reports each successful archive operation to the server logs unless `DEBUG` level is used (Simon)

E.121.2.2. Configuration Parameters

- Numerous changes in administrative server parameters

  `bgwriter_lru_percent`, `bgwriter_all_percent`, `bgwriter_all_maxpages`, `stats_start_collector`, and `stats_reset_on_server_start` are removed. `redirect_stderr` is renamed to `logging_collector`. `stats_command_string` is renamed to `track_activities`. `stats_block_level` and `stats_row_level` are merged into `track_counts`. A new boolean configuration parameter, `archive_mode`, controls archiving. Autovacuum's default settings have changed.
- Remove `stats_start_collector` parameter (Tom)

  We now always start the collector process, unless UDP socket creation fails.
- Remove `stats_reset_on_server_start` parameter (Tom)

  This was removed because `pg_stat_reset()` can be used for this purpose.
- Commenting out a parameter in `postgresql.conf` now causes it to revert to its default value (Joachim Wieland)

  Previously, commenting out an entry left the parameter's value unchanged until the next server restart.

E.121.2.3. Character Encodings

- Add more checks for invalidly-encoded data (Andrew)

  This change plugs some holes that existed in literal backslash escape string processing and `COPY` escape processing. Now the de-escaped string is rechecked to see if the result created an invalid multi-byte character.

- Disallow database encodings that are inconsistent with the server's locale setting (Tom)

  On most platforms, `C` locale is the only locale that will work with any database encoding. Other locale settings imply a specific encoding and will misbehave if the database encoding is something different. (Typical symptoms include bogus textual sort order and wrong results from `upper()` or `lower()`.) The server now rejects attempts to create databases that have an incompatible encoding.

- Ensure that `chr()` cannot create invalidly-encoded values (Andrew)

  In UTF8-encoded databases the argument of `chr()` is now treated as a Unicode code point. In other multi-byte encodings `chr()`'s argument must designate a 7-bit ASCII character. Zero is no longer accepted. `ascii()` has been adjusted to match.

- Adjust `convert()` behavior to ensure encoding validity (Andrew)

  The two argument form of `convert()` has been removed. The three argument form now takes a `bytea` first argument and returns a `bytea`. To cover the loss of functionality, three new functions have been added:

  - `convert_from(bytea, name)` returns `text` – converts the first argument from the named encoding to the database encoding
  - `convert_to(text, name)` returns `bytea` – converts the first argument from the database encoding to the named encoding
  - `length(bytea, name)` returns `integer` – gives the length of the first argument in characters in the named encoding

- Remove `convert(argument USING conversion_name)` (Andrew)

  Its behavior did not match the SQL standard.

- Make JOHAB encoding client-only (Tatsuo)

  JOHAB is not safe as a server-side encoding.

E.121.3. Changes

Below you will find a detailed account of the changes between PostgreSQL 8.3 and the previous major release.

E.121.3.1. Performance

- Asynchronous commit delays writes to WAL during transaction commit (Simon)

  This feature dramatically increases performance for short data-modifying transactions. The disadvantage is that because disk writes are delayed, if the database or operating system crashes before data is written to the disk, committed data will be lost. This feature is useful for applications that can accept some data loss. Unlike turning off `fsync`, using asynchronous commit does not put database

consistency at risk; the worst case is that after a crash the last few reportedly-committed transactions might not be committed after all. This feature is enabled by turning off `synchronous_commit` (which can be done per-session or per-transaction, if some transactions are critical and others are not). `wal_writer_delay` can be adjusted to control the maximum delay before transactions actually reach disk.

- Checkpoint writes can be spread over a longer time period to smooth the I/O spike during each checkpoint (Itagaki Takahiro and Heikki Linnakangas)

  Previously all modified buffers were forced to disk as quickly as possible during a checkpoint, causing an I/O spike that decreased server performance. This new approach spreads out disk writes during checkpoints, reducing peak I/O usage. (User-requested and shutdown checkpoints are still written as quickly as possible.)

- Heap-Only Tuples (HOT) accelerate space reuse for most `UPDATE`s and `DELETE`s (Pavan Deolasee, with ideas from many others)

  `UPDATE`s and `DELETE`s leave dead tuples behind, as do failed `INSERT`s. Previously only `VACUUM` could reclaim space taken by dead tuples. With HOT dead tuple space can be automatically reclaimed at the time of `INSERT` or `UPDATE` if no changes are made to indexed columns. This allows for more consistent performance. Also, HOT avoids adding duplicate index entries.

- Just-in-time background writer strategy improves disk write efficiency (Greg Smith, Itagaki Takahiro)

  This greatly reduces the need for manual tuning of the background writer.

- Per-field and per-row storage overhead have been reduced (Greg Stark, Heikki Linnakangas)

  Variable-length data types with data values less than 128 bytes long will see a storage decrease of 3 to 6 bytes. For example, two adjacent `char(1)` fields now use 4 bytes instead of 16. Row headers are also 4 bytes shorter than before.

- Using non-persistent transaction IDs for read-only transactions reduces overhead and `VACUUM` requirements (Florian Pflug)

  Non-persistent transaction IDs do not increment the global transaction counter. Therefore, they reduce the load on `pg_clog` and increase the time between forced vacuums to prevent transaction ID wraparound. Other performance improvements were also made that should improve concurrency.

- Avoid incrementing the command counter after a read-only command (Tom)

  There was formerly a hard limit of 2^{32} (4 billion) commands per transaction. Now only commands that actually changed the database count, so while this limit still exists, it should be significantly less annoying.

- Create a dedicated WAL writer process to off-load work from backends (Simon)

- Skip unnecessary WAL writes for `CLUSTER` and `COPY` (Simon)

  Unless WAL archiving is enabled, the system now avoids WAL writes for `CLUSTER` and just `fsync()`s the table at the end of the command. It also does the same for `COPY` if the table was created in the same transaction.

- Large sequential scans no longer force out frequently used cached pages (Simon, Heikki, Tom)

- Concurrent large sequential scans can now share disk reads (Jeff Davis)

  This is accomplished by starting the new sequential scan in the middle of the table (where another sequential scan is already in-progress) and wrapping around to the beginning to finish. This can affect the order of returned rows in a query that does not specify `ORDER BY`. The `synchronize_seqscans` configuration parameter can be used to disable this if necessary.

- `ORDER BY ... LIMIT` can be done without sorting (Greg Stark)

  This is done by sequentially scanning the table and tracking just the "top N" candidate rows, rather than performing a full sort of the entire table. This is useful when there is no matching index and the `LIMIT` is not large.

- Put a rate limit on messages sent to the statistics collector by backends (Tom)

  This reduces overhead for short transactions, but might sometimes increase the delay before statistics are tallied.

- Improve hash join performance for cases with many NULLs (Tom)

- Speed up operator lookup for cases with non-exact datatype matches (Tom)

E.121.3.2. Server

- Autovacuum is now enabled by default (Alvaro)

  Several changes were made to eliminate disadvantages of having autovacuum enabled, thereby justifying the change in default. Several other autovacuum parameter defaults were also modified.

- Support multiple concurrent autovacuum processes (Alvaro, Itagaki Takahiro)

  This allows multiple vacuums to run concurrently. This prevents vacuuming of a large table from delaying vacuuming of smaller tables.

- Automatically re-plan cached queries when table definitions change or statistics are updated (Tom)

  Previously PL/pgSQL functions that referenced temporary tables would fail if the temporary table was dropped and recreated between function invocations, unless `EXECUTE` was used. This improvement fixes that problem and many related issues.

- Add a `temp_tablespaces` parameter to control the tablespaces for temporary tables and files (Jaime Casanova, Albert Cervera, Bernd Helmle)

  This parameter defines a list of tablespaces to be used. This enables spreading the I/O load across multiple tablespaces. A random tablespace is chosen each time a temporary object is created. Temporary files are no longer stored in per-database `pgsql_tmp/` directories but in per-tablespace directories.

- Place temporary tables' TOAST tables in special schemas named `pg_toast_temp_`*`nnn`* (Tom)

  This allows low-level code to recognize these tables as temporary, which enables various optimizations such as not WAL-logging changes and using local rather than shared buffers for access. This also fixes a bug wherein backends unexpectedly held open file references to temporary TOAST tables.

- Fix problem that a constant flow of new connection requests could indefinitely delay the postmaster from completing a shutdown or a crash restart (Tom)

- Guard against a very-low-probability data loss scenario by preventing re-use of a deleted table's relfilenode until after the next checkpoint (Heikki)
- Fix `CREATE CONSTRAINT TRIGGER` to convert old-style foreign key trigger definitions into regular foreign key constraints (Tom)

  This will ease porting of foreign key constraints carried forward from pre-7.3 databases, if they were never converted using `contrib/adddepend`.
- Fix `DEFAULT NULL` to override inherited defaults (Tom)

  `DEFAULT NULL` was formerly considered a noise phrase, but it should (and now does) override non-null defaults that would otherwise be inherited from a parent table or domain.
- Add new encodings EUC_JIS_2004 and SHIFT_JIS_2004 (Tatsuo)

  These new encodings can be converted to and from UTF-8.
- Change server startup log message from "database system is ready" to "database system is ready to accept connections", and adjust its timing

  The message now appears only when the postmaster is really ready to accept connections.

E.121.3.3. Monitoring

- Add `log_autovacuum_min_duration` parameter to support configurable logging of autovacuum activity (Simon, Alvaro)
- Add `log_lock_waits` parameter to log lock waiting (Simon)
- Add `log_temp_files` parameter to log temporary file usage (Bill Moran)
- Add `log_checkpoints` parameter to improve logging of checkpoints (Greg Smith, Heikki)
- `log_line_prefix` now supports `%s` and `%c` escapes in all processes (Andrew)

  Previously these escapes worked only for user sessions, not for background database processes.
- Add `log_restartpoints` to control logging of point-in-time recovery restart points (Simon)
- Last transaction end time is now logged at end of recovery and at each logged restart point (Simon)
- Autovacuum now reports its activity start time in `pg_stat_activity` (Tom)
- Allow server log output in comma-separated value (CSV) format (Arul Shaji, Greg Smith, Andrew Dunstan)

  CSV-format log files can easily be loaded into a database table for subsequent analysis.
- Use PostgreSQL-supplied timezone support for formatting timestamps displayed in the server log (Tom)

  This avoids Windows-specific problems with localized time zone names that are in the wrong encoding. There is a new `log_timezone` parameter that controls

the timezone used in log messages, independently of the client-visible `timezone` parameter.

- New system view `pg_stat_bgwriter` displays statistics about background writer activity (Magnus)
- Add new columns for database-wide tuple statistics to `pg_stat_database` (Magnus)
- Add an `xact_start` (transaction start time) column to `pg_stat_activity` (Neil)

  This makes it easier to identify long-running transactions.
- Add `n_live_tuples` and `n_dead_tuples` columns to `pg_stat_all_tables` and related views (Glen Parker)
- Merge `stats_block_level` and `stats_row_level` parameters into a single parameter `track_counts`, which controls all messages sent to the statistics collector process (Tom)
- Rename `stats_command_string` parameter to `track_activities` (Tom)
- Fix statistical counting of live and dead tuples to recognize that committed and aborted transactions have different effects (Tom)

E.121.3.4. Authentication

- Support Security Service Provider Interface (SSPI) for authentication on Windows (Magnus)
- Support GSSAPI authentication (Henry Hotz, Magnus)

  This should be preferred to native Kerberos authentication because GSSAPI is an industry standard.
- Support a global SSL configuration file (Victor Wagner)
- Add `ssl_ciphers` parameter to control accepted SSL ciphers (Victor Wagner)
- Add a Kerberos realm parameter, `krb_realm` (Magnus)

E.121.3.5. Write-Ahead Log (WAL) and Continuous Archiving

- Change the timestamps recorded in transaction WAL records from time_t to TimestampTz representation (Tom)

  This provides sub-second resolution in WAL, which can be useful for point-in-time recovery.
- Reduce WAL disk space needed by warm standby servers (Simon)

  This change allows a warm standby server to pass the name of the earliest still-needed WAL file to the recovery script, allowing automatic removal of no-longer-needed WAL files. This is done using `%r` in the `restore_command` parameter of `recovery.conf`.
- New boolean configuration parameter, `archive_mode`, controls archiving (Simon)

  Previously setting `archive_command` to an empty string turned off archiving. Now `archive_mode` turns archiving on and off, independently of `archive_command`. This is useful for stopping archiving temporarily.

E.121.3.6. Queries

- Full text search is integrated into the core database system (Teodor, Oleg)

  Text search has been improved, moved into the core code, and is now installed by default. `contrib/tsearch2` now contains a compatibility interface.

- Add control over whether `NULL`s sort first or last (Teodor, Tom)

  The syntax is `ORDER BY ... NULLS FIRST/LAST`.

- Allow per-column ascending/descending (`ASC`/`DESC`) ordering options for indexes (Teodor, Tom)

  Previously a query using `ORDER BY` with mixed `ASC`/`DESC` specifiers could not fully use an index. Now an index can be fully used in such cases if the index was created with matching `ASC`/`DESC` specifications. `NULL` sort order within an index can be controlled, too.

- Allow `col IS NULL` to use an index (Teodor)

- Updatable cursors (Arul Shaji, Tom)

  This eliminates the need to reference a primary key to `UPDATE` or `DELETE` rows returned by a cursor. The syntax is `UPDATE/DELETE WHERE CURRENT OF`.

- Allow `FOR UPDATE` in cursors (Arul Shaji, Tom)

- Create a general mechanism that supports casts to and from the standard string types (`TEXT`, `VARCHAR`, `CHAR`) for every datatype, by invoking the datatype's I/O functions (Tom)

  Previously, such casts were available only for types that had specialized function(s) for the purpose. These new casts are assignment-only in the to-string direction, explicit-only in the other direction, and therefore should create no surprising behavior.

- Allow `UNION` and related constructs to return a domain type, when all inputs are of that domain type (Tom)

  Formerly, the output would be considered to be of the domain's base type.

- Allow limited hashing when using two different data types (Tom)

  This allows hash joins, hash indexes, hashed subplans, and hash aggregation to be used in situations involving cross-data-type comparisons, if the data types have compatible hash functions. Currently, cross-data-type hashing support exists for `smallint`/`integer`/`bigint`, and for `float4`/`float8`.

- Improve optimizer logic for detecting when variables are equal in a `WHERE` clause (Tom)

  This allows mergejoins to work with descending sort orders, and improves recognition of redundant sort columns.

- Improve performance when planning large inheritance trees in cases where most tables are excluded by constraints (Tom)

E.121.3.7. Object Manipulation

- Arrays of composite types (David Fetter, Andrew, Tom)

In addition to arrays of explicitly-declared composite types, arrays of the rowtypes of regular tables and views are now supported, except for rowtypes of system catalogs, sequences, and TOAST tables.

- Server configuration parameters can now be set on a per-function basis (Tom)

  For example, functions can now set their own `search_path` to prevent unexpected behavior if a different `search_path` exists at run-time. Security definer functions should set `search_path` to avoid security loopholes.

- `CREATE/ALTER FUNCTION` now supports `COST` and `ROWS` options (Tom)

  `COST` allows specification of the cost of a function call. `ROWS` allows specification of the average number or rows returned by a set-returning function. These values are used by the optimizer in choosing the best plan.

- Implement `CREATE TABLE LIKE ... INCLUDING INDEXES` (Trevor Hardcastle, Nikhil Sontakke, Neil)

- Allow `CREATE INDEX CONCURRENTLY` to ignore transactions in other databases (Simon)

- Add `ALTER VIEW ... RENAME TO` and `ALTER SEQUENCE ... RENAME TO` (David Fetter, Neil)

  Previously this could only be done via `ALTER TABLE ... RENAME TO`.

- Make `CREATE/DROP/RENAME DATABASE` wait briefly for conflicting backends to exit before failing (Tom)

  This increases the likelihood that these commands will succeed.

- Allow triggers and rules to be deactivated in groups using a configuration parameter, for replication purposes (Jan)

  This allows replication systems to disable triggers and rewrite rules as a group without modifying the system catalogs directly. The behavior is controlled by `ALTER TABLE` and a new parameter `session_replication_role`.

- User-defined types can now have type modifiers (Teodor, Tom)

  This allows a user-defined type to take a modifier, like `ssnum(7)`. Previously only built-in data types could have modifiers.

E.121.3.8. Utility Commands

- Non-superuser database owners now are able to add trusted procedural languages to their databases by default (Jeremy Drake)

  While this is reasonably safe, some administrators might wish to revoke the privilege. It is controlled by `pg_pltemplate.tmpldbacreate`.

- Allow a session's current parameter setting to be used as the default for future sessions (Tom)

  This is done with `SET ... FROM CURRENT` in `CREATE/ALTER FUNCTION`, `ALTER DATABASE`, or `ALTER ROLE`.

- Implement new commands `DISCARD ALL`, `DISCARD PLANS`, `DISCARD TEMPORARY`, `CLOSE ALL`, and `DEALLOCATE ALL` (Marko Kreen, Neil)

These commands simplify resetting a database session to its initial state, and are particularly useful for connection-pooling software.

- Make `CLUSTER` MVCC-safe (Heikki Linnakangas)

  Formerly, `CLUSTER` would discard all tuples that were committed dead, even if there were still transactions that should be able to see them under MVCC visibility rules.

- Add new `CLUSTER` syntax: `CLUSTER table USING index` (Holger Schurig)

  The old `CLUSTER` syntax is still supported, but the new form is considered more logical.

- Fix `EXPLAIN` so it can show complex plans more accurately (Tom)

  References to subplan outputs are now always shown correctly, instead of using `?column*N*?` for complicated cases.

- Limit the amount of information reported when a user is dropped (Alvaro)

  Previously, dropping (or attempting to drop) a user who owned many objects could result in large `NOTICE` or `ERROR` messages listing all these objects; this caused problems for some client applications. The length of the message is now limited, although a full list is still sent to the server log.

E.121.3.9. Data Types

- Support for the SQL/XML standard, including new operators and an `XML` data type (Nikolay Samokhvalov, Pavel Stehule, Peter)

- Enumerated data types (`ENUM`) (Tom Dunstan)

  This feature provides convenient support for fields that have a small, fixed set of allowed values. An example of creating an `ENUM` type is `CREATE TYPE mood AS ENUM ('sad', 'ok', 'happy')`.

- Universally Unique Identifier (`UUID`) data type (Gevik Babakhani, Neil)

  This closely matches RFC 4122.

- Widen the `MONEY` data type to 64 bits (D'Arcy Cain)

  This greatly increases the range of supported `MONEY` values.

- Fix `float4/float8` to handle `Infinity` and `NAN` (Not A Number) consistently (Bruce)

  The code formerly was not consistent about distinguishing `Infinity` from overflow conditions.

- Allow leading and trailing whitespace during input of `boolean` values (Neil)

- Prevent `COPY` from using digits and lowercase letters as delimiters (Tom)

E.121.3.10. Functions

- Add new regular expression functions `regexp_matches()`, `regexp_split_to_array()`, and `regexp_split_to_table()` (Jeremy Drake, Neil)

These functions provide extraction of regular expression subexpressions and allow splitting a string using a POSIX regular expression.

- Add `lo_truncate()` for large object truncation (Kris Jurka)
- Implement `width_bucket()` for the `float8` data type (Neil)
- Add `pg_stat_clear_snapshot()` to discard statistics snapshots collected during the current transaction (Tom)

  The first request for statistics in a transaction takes a statistics snapshot that does not change during the transaction. This function allows the snapshot to be discarded and a new snapshot loaded during the next statistics query. This is particularly useful for PL/pgSQL functions, which are confined to a single transaction.

- Add `isodow` option to `EXTRACT()` and `date_part()` (Bruce)

  This returns the day of the week, with Sunday as seven. (`dow` returns Sunday as zero.)

- Add `ID` (ISO day of week) and `IDDD` (ISO day of year) format codes for `to_char()`, `to_date()`, and `to_timestamp()` (Brendan Jurd)
- Make `to_timestamp()` and `to_date()` assume `TM` (trim) option for potentially variable-width fields (Bruce)

  This matches Oracle's behavior.

- Fix off-by-one conversion error in `to_date()`/`to_timestamp() D` (non-ISO day of week) fields (Bruce)
- Make `setseed()` return void, rather than a useless integer value (Neil)
- Add a hash function for `NUMERIC` (Neil)

  This allows hash indexes and hash-based plans to be used with `NUMERIC` columns.

- Improve efficiency of `LIKE`/`ILIKE`, especially for multi-byte character sets like UTF-8 (Andrew, Itagaki Takahiro)
- Make `currtid()` functions require `SELECT` privileges on the target table (Tom)
- Add several `txid_*()` functions to query active transaction IDs (Jan)

  This is useful for various replication solutions.

E.121.3.11. PL/pgSQL Server-Side Language

- Add scrollable cursor support, including directional control in `FETCH` (Pavel Stehule)
- Allow `IN` as an alternative to `FROM` in PL/pgSQL's `FETCH` statement, for consistency with the backend's `FETCH` command (Pavel Stehule)
- Add `MOVE` to PL/pgSQL (Magnus, Pavel Stehule, Neil)
- Implement `RETURN QUERY` (Pavel Stehule, Neil)

  This adds convenient syntax for PL/pgSQL set-returning functions that want to return the result of a query. `RETURN QUERY` is easier and more efficient than a loop around `RETURN NEXT`.

- Allow function parameter names to be qualified with the function's name (Tom)

For example, `myfunc.myvar`. This is particularly useful for specifying variables in a query where the variable name might match a column name.

- Make qualification of variables with block labels work properly (Tom)

  Formerly, outer-level block labels could unexpectedly interfere with recognition of inner-level record or row references.

- Tighten requirements for `FOR` loop `STEP` values (Tom)

  Prevent non-positive `STEP` values, and handle loop overflows.

- Improve accuracy when reporting syntax error locations (Tom)

E.121.3.12. Other Server-Side Languages

- Allow type-name arguments to PL/Perl `spi_prepare()` to be data type aliases in addition to names found in `pg_type` (Andrew)
- Allow type-name arguments to PL/Python `plpy.prepare()` to be data type aliases in addition to names found in `pg_type` (Andrew)
- Allow type-name arguments to PL/Tcl `spi_prepare` to be data type aliases in addition to names found in `pg_type` (Andrew)
- Enable PL/PythonU to compile on Python 2.5 (Marko Kreen)
- Support a true PL/Python boolean type in compatible Python versions (Python 2.3 and later) (Marko Kreen)
- Fix PL/Tcl problems with thread-enabled `libtcl` spawning multiple threads within the backend (Steve Marshall, Paul Bayer, Doug Knight)

  This caused all sorts of unpleasantness.

E.121.3.13. psql

- List disabled triggers separately in `\d` output (Brendan Jurd)
- In `\d` patterns, always match `$` literally (Tom)
- Show aggregate return types in `\da` output (Greg Sabino Mullane)
- Add the function's volatility status to the output of `\df+` (Neil)
- Add `\prompt` capability (Chad Wagner)
- Allow `\pset`, `\t`, and `\x` to specify `on` or `off`, rather than just toggling (Chad Wagner)
- Add `\sleep` capability (Jan)
- Enable `\timing` output for `\copy` (Andrew)
- Improve `\timing` resolution on Windows (Itagaki Takahiro)
- Flush `\o` output after each backslash command (Tom)
- Correctly detect and report errors while reading a `-f` input file (Peter)

- Remove `-u` option (this option has long been deprecated) (Tom)

E.121.3.14. pg_dump

- Add `--tablespaces-only` and `--roles-only` options to pg_dumpall (Dave Page)
- Add an output file option to pg_dumpall (Dave Page)

  This is primarily useful on Windows, where output redirection of child pg_dump processes does not work.
- Allow pg_dumpall to accept an initial-connection database name rather than the default `template1` (Dave Page)
- In `-n` and `-t` switches, always match `$` literally (Tom)
- Improve performance when a database has thousands of objects (Tom)
- Remove `-u` option (this option has long been deprecated) (Tom)

E.121.3.15. Other Client Applications

- In initdb, allow the location of the `pg_xlog` directory to be specified (Euler Taveira de Oliveira)
- Enable server core dump generation in pg_regress on supported operating systems (Andrew)
- Add a `-t` (timeout) parameter to pg_ctl (Bruce)

  This controls how long pg_ctl will wait when waiting for server startup or shutdown. Formerly the timeout was hard-wired as 60 seconds.
- Add a pg_ctl option to control generation of server core dumps (Andrew)
- Allow Control-C to cancel clusterdb, reindexdb, and vacuumdb (Itagaki Takahiro, Magnus)
- Suppress command tag output for createdb, createuser, dropdb, and dropuser (Peter)

  The `--quiet` option is ignored and will be removed in 8.4. Progress messages when acting on all databases now go to stdout instead of stderr because they are not actually errors.

E.121.3.16. libpq

- Interpret the `dbName` parameter of `PQsetdbLogin()` as a `conninfo` string if it contains an equals sign (Andrew)

  This allows use of `conninfo` strings in client programs that still use `PQsetdbLogin()`.
- Support a global SSL configuration file (Victor Wagner)
- Add environment variable `PGSSLKEY` to control SSL hardware keys (Victor Wagner)
- Add `lo_truncate()` for large object truncation (Kris Jurka)
- Add `PQconnectionNeedsPassword()` that returns true if the server required a password but none was supplied (Joe Conway, Tom)

If this returns true after a failed connection attempt, a client application should prompt the user for a password. In the past applications have had to check for a specific error message string to decide whether a password is needed; that approach is now deprecated.

- Add `PQconnectionUsedPassword()` that returns true if the supplied password was actually used (Joe Conway, Tom)

  This is useful in some security contexts where it is important to know whether a user-supplied password is actually valid.

E.121.3.17. ecpg

- Use V3 frontend/backend protocol (Michael)

  This adds support for server-side prepared statements.

- Use native threads, instead of pthreads, on Windows (Magnus)
- Improve thread-safety of ecpglib (Itagaki Takahiro)
- Make the ecpg libraries export only necessary API symbols (Michael)

E.121.3.18. Windows Port

- Allow the whole PostgreSQL distribution to be compiled with Microsoft Visual C++ (Magnus and others)

  This allows Windows-based developers to use familiar development and debugging tools. Windows executables made with Visual C++ might also have better stability and performance than those made with other tool sets. The client-only Visual C++ build scripts have been removed.

- Drastically reduce postmaster's memory usage when it has many child processes (Magnus)
- Allow regression tests to be started by an administrative user (Magnus)
- Add native shared memory implementation (Magnus)

E.121.3.19. Server Programming Interface (SPI)

- Add cursor-related functionality in SPI (Pavel Stehule)

  Allow access to the cursor-related planning options, and add `FETCH`/`MOVE` routines.

- Allow execution of cursor commands through `SPI_execute` (Tom)

  The macro `SPI_ERROR_CURSOR` still exists but will never be returned.

- SPI plan pointers are now declared as `SPIPlanPtr` instead of `void *` (Tom)

  This does not break application code, but switching is recommended to help catch simple programming mistakes.

E.121.3.20. Build Options

- Add configure option `--enable-profiling` to enable code profiling (works only with gcc) (Korry Douglas and Nikhil Sontakke)

- Add configure option `--with-system-tzdata` to use the operating system's time zone database (Peter)
- Fix PGXS so extensions can be built against PostgreSQL installations whose pg_config program does not appear first in the `PATH` (Tom)
- Support `gmake draft` when building the SGML documentation (Bruce)

  Unless `draft` is used, the documentation build will now be repeated if necessary to ensure the index is up-to-date.

E.121.3.21. Source Code

- Rename macro `DLLIMPORT` to `PGDLLIMPORT` to avoid conflicting with third party includes (like Tcl) that define `DLLIMPORT` (Magnus)
- Create "operator families" to improve planning of queries involving cross-data-type comparisons (Tom)
- Update GIN `extractQuery()` API to allow signalling that nothing can satisfy the query (Teodor)
- Move `NAMEDATALEN` definition from `postgres_ext.h` to `pg_config_manual.h` (Peter)
- Provide `strlcpy()` and `strlcat()` on all platforms, and replace error-prone uses of `strncpy()`, `strncat()`, etc (Peter)
- Create hooks to let an external plugin monitor (or even replace) the planner and create plans for hypothetical situations (Gurjeet Singh, Tom)
- Create a function variable `join_search_hook` to let plugins override the join search order portion of the planner (Julius Stroffek)
- Add `tas()` support for Renesas' M32R processor (Kazuhiro Inaoka)
- `quote_identifier()` and pg_dump no longer quote keywords that are unreserved according to the grammar (Tom)
- Change the on-disk representation of the `NUMERIC` data type so that the `sign_dscale` word comes before the weight (Tom)
- Use SYSV semaphores rather than POSIX on Darwin >= 6.0, i.e., OS X 10.2 and up (Chris Marcellino)
- Add acronym and NFS documentation sections (Bruce)
- "Postgres" is now documented as an accepted alias for "PostgreSQL" (Peter)
- Add documentation about preventing database server spoofing when the server is down (Bruce)

E.121.3.22. Contrib

- Move `contrib README` content into the main PostgreSQL documentation (Albert Cervera i Areny)
- Add `contrib/pageinspect` module for low-level page inspection (Simon, Heikki)
- Add `contrib/pg_standby` module for controlling warm standby operation (Simon)

- Add `contrib/uuid-ossp` module for generating `UUID` values using the OSSP UUID library (Peter)

  Use configure `--with-ossp-uuid` to activate. This takes advantage of the new `UUID` builtin type.

- Add `contrib/dict_int`, `contrib/dict_xsyn`, and `contrib/test_parser` modules to provide sample add-on text search dictionary templates and parsers (Sergey Karpov)

- Allow contrib/pgbench to set the fillfactor (Pavan Deolasee)

- Add timestamps to contrib/pgbench `-l` (Greg Smith)

- Add usage count statistics to `contrib/pgbuffercache` (Greg Smith)

- Add GIN support for `contrib/hstore` (Teodor)

- Add GIN support for `contrib/pg_trgm` (Guillaume Smet, Teodor)

- Update OS/X startup scripts in `contrib/start-scripts` (Mark Cotner, David Fetter)

- Restrict `pgrowlocks()` and `dblink_get_pkey()` to users who have `SELECT` privilege on the target table (Tom)

- Restrict `contrib/pgstattuple` functions to superusers (Tom)

- `contrib/xml2` is deprecated and planned for removal in 8.4 (Peter)

  The new XML support in core PostgreSQL supersedes this module.

E.122. Release 8.2.23

Release date: 2011-12-05

This release contains a variety of fixes from 8.2.22. For information about new features in the 8.2 major release, see 第 E.145 节.

This is expected to be the last PostgreSQL release in the 8.2.X series. Users are encouraged to update to a newer release branch soon.

E.122.1. Migration to Version 8.2.23

A dump/restore is not required for those running 8.2.X.

However, a longstanding error was discovered in the definition of the `information_schema.referential_constraints` view. If you rely on correct results from that view, you should replace its definition as explained in the first changelog item below.

Also, if you are upgrading from a version earlier than 8.2.14, see 第 E.131 节.

E.122.2. Changes

- Fix bugs in `information_schema.referential_constraints` view (Tom Lane)

  This view was being insufficiently careful about matching the foreign-key constraint to the depended-on primary or unique key constraint. That could result in failure to

show a foreign key constraint at all, or showing it multiple times, or claiming that it depends on a different constraint than the one it really does.

Since the view definition is installed by initdb, merely upgrading will not fix the problem. If you need to fix this in an existing installation, you can (as a superuser) drop the `information_schema` schema then re-create it by sourcing *`SHAREDIR`* `/information_schema.sql`. (Run `pg_config --sharedir` if you're uncertain where *`SHAREDIR`* is.) This must be repeated in each database to be fixed.

- Fix TOAST-related data corruption during `CREATE TABLE dest AS SELECT * FROM src` or `INSERT INTO dest SELECT * FROM src` (Tom Lane)

  If a table has been modified by `ALTER TABLE ADD COLUMN`, attempts to copy its data verbatim to another table could produce corrupt results in certain corner cases. The problem can only manifest in this precise form in 8.4 and later, but we patched earlier versions as well in case there are other code paths that could trigger the same bug.

- Fix race condition during toast table access from stale syscache entries (Tom Lane)

  The typical symptom was transient errors like "missing chunk number 0 for toast value NNNNN in pg_toast_2619", where the cited toast table would always belong to a system catalog.

- Improve locale support in `money` type's input and output (Tom Lane)

  Aside from not supporting all standard `lc_monetary` formatting options, the input and output functions were inconsistent, meaning there were locales in which dumped `money` values could not be re-read.

- Don't let `transform_null_equals` affect `CASE foo WHEN NULL ...` constructs (Heikki Linnakangas)

  `transform_null_equals` is only supposed to affect `foo = NULL` expressions written directly by the user, not equality checks generated internally by this form of `CASE`.

- Change foreign-key trigger creation order to better support self-referential foreign keys (Tom Lane)

  For a cascading foreign key that references its own table, a row update will fire both the `ON UPDATE` trigger and the `CHECK` trigger as one event. The `ON UPDATE` trigger must execute first, else the `CHECK` will check a non-final state of the row and possibly throw an inappropriate error. However, the firing order of these triggers is determined by their names, which generally sort in creation order since the triggers have auto-generated names following the convention "RI_ConstraintTrigger_NNNN". A proper fix would require modifying that convention, which we will do in 9.2, but it seems risky to change it in existing releases. So this patch just changes the creation order of the triggers. Users encountering this type of error should drop and re-create the foreign key constraint to get its triggers into the right order.

- Preserve blank lines within commands in psql's command history (Robert Haas)

  The former behavior could cause problems if an empty line was removed from within a string literal, for example.

- Use the preferred version of xsubpp to build PL/Perl, not necessarily the operating system's main copy (David Wheeler and Alex Hunsaker)

- Honor query cancel interrupts promptly in `pgstatindex()` (Robert Haas)

- Ensure VPATH builds properly install all server header files (Peter Eisentraut)
- Shorten file names reported in verbose error messages (Peter Eisentraut)

  Regular builds have always reported just the name of the C file containing the error message call, but VPATH builds formerly reported an absolute path name.
- Fix interpretation of Windows timezone names for Central America (Tom Lane)

  Map "Central America Standard Time" to `CST6`, not `CST6CDT`, because DST is generally not observed anywhere in Central America.
- Update time zone data files to tzdata release 2011n for DST law changes in Brazil, Cuba, Fiji, Palestine, Russia, and Samoa; also historical corrections for Alaska and British East Africa.

E.123. Release 8.2.22

Release date: 2011-09-26

This release contains a variety of fixes from 8.2.21. For information about new features in the 8.2 major release, see 第 E.145 节.

The PostgreSQL community will stop releasing updates for the 8.2.X release series in December 2011. Users are encouraged to update to a newer release branch soon.

E.123.1. Migration to Version 8.2.22

A dump/restore is not required for those running 8.2.X. However, if you are upgrading from a version earlier than 8.2.14, see 第 E.131 节.

E.123.2. Changes

- Fix multiple bugs in GiST index page split processing (Heikki Linnakangas)

  The probability of occurrence was low, but these could lead to index corruption.
- Avoid possibly accessing off the end of memory in `ANALYZE` (Noah Misch)

  This fixes a very-low-probability server crash scenario.
- Fix race condition in relcache init file invalidation (Tom Lane)

  There was a window wherein a new backend process could read a stale init file but miss the inval messages that would tell it the data is stale. The result would be bizarre failures in catalog accesses, typically "could not read block 0 in file ..." later during startup.
- Fix memory leak at end of a GiST index scan (Tom Lane)

  Commands that perform many separate GiST index scans, such as verification of a new GiST-based exclusion constraint on a table already containing many rows, could transiently require large amounts of memory due to this leak.
- Fix performance problem when constructing a large, lossy bitmap (Tom Lane)
- Fix array- and path-creating functions to ensure padding bytes are zeroes (Tom Lane)

This avoids some situations where the planner will think that semantically-equal constants are not equal, resulting in poor optimization.

- Work around gcc 4.6.0 bug that breaks WAL replay (Tom Lane)

  This could lead to loss of committed transactions after a server crash.

- Fix dump bug for `VALUES` in a view (Tom Lane)

- Disallow `SELECT FOR UPDATE/SHARE` on sequences (Tom Lane)

  This operation doesn't work as expected and can lead to failures.

- Defend against integer overflow when computing size of a hash table (Tom Lane)

- Fix portability bugs in use of credentials control messages for "peer" authentication (Tom Lane)

- Fix typo in `pg_srand48` seed initialization (Andres Freund)

  This led to failure to use all bits of the provided seed. This function is not used on most platforms (only those without `srandom`), and the potential security exposure from a less-random-than-expected seed seems minimal in any case.

- Avoid integer overflow when the sum of `LIMIT` and `OFFSET` values exceeds 2^63 (Heikki Linnakangas)

- Add overflow checks to `int4` and `int8` versions of `generate_series()` (Robert Haas)

- Fix trailing-zero removal in `to_char()` (Marti Raudsepp)

  In a format with `FM` and no digit positions after the decimal point, zeroes to the left of the decimal point could be removed incorrectly.

- Fix `pg_size_pretty()` to avoid overflow for inputs close to 2^63 (Tom Lane)

- Fix psql's counting of script file line numbers during `COPY` from a different file (Tom Lane)

- Fix pg_restore's direct-to-database mode for `standard_conforming_strings` (Tom Lane)

  pg_restore could emit incorrect commands when restoring directly to a database server from an archive file that had been made with `standard_conforming_strings` set to `on`.

- Fix write-past-buffer-end and memory leak in libpq's LDAP service lookup code (Albe Laurenz)

- In libpq, avoid failures when using nonblocking I/O and an SSL connection (Martin Pihlak, Tom Lane)

- Improve libpq's handling of failures during connection startup (Tom Lane)

  In particular, the response to a server report of `fork()` failure during SSL connection startup is now saner.

- Make ecpglib write `double` values with 15 digits precision (Akira Kurosawa)

- Apply upstream fix for blowfish signed-character bug (CVE-2011-2483) (Tom Lane)

contrib/pg_crypto's blowfish encryption code could give wrong results on platforms where char is signed (which is most), leading to encrypted passwords being weaker than they should be.

- Fix memory leak in contrib/seg (Heikki Linnakangas)
- Fix pgstatindex() to give consistent results for empty indexes (Tom Lane)
- Allow building with perl 5.14 (Alex Hunsaker)
- Update configure script's method for probing existence of system functions (Tom Lane)

  The version of autoconf we used in 8.3 and 8.2 could be fooled by compilers that perform link-time optimization.
- Fix assorted issues with build and install file paths containing spaces (Tom Lane)
- Update time zone data files to tzdata release 2011i for DST law changes in Canada, Egypt, Russia, Samoa, and South Sudan.

E.124. Release 8.2.21

Release date: 2011-04-18

This release contains a variety of fixes from 8.2.20. For information about new features in the 8.2 major release, see 第 E.145 节.

E.124.1. Migration to Version 8.2.21

A dump/restore is not required for those running 8.2.X. However, if you are upgrading from a version earlier than 8.2.14, see 第 E.131 节.

E.124.2. Changes

- Avoid potential deadlock during catalog cache initialization (Nikhil Sontakke)

  In some cases the cache loading code would acquire share lock on a system index before locking the index's catalog. This could deadlock against processes trying to acquire exclusive locks in the other, more standard order.
- Fix dangling-pointer problem in BEFORE ROW UPDATE trigger handling when there was a concurrent update to the target tuple (Tom Lane)

  This bug has been observed to result in intermittent "cannot extract system attribute from virtual tuple" failures while trying to do UPDATE RETURNING ctid. There is a very small probability of more serious errors, such as generating incorrect index entries for the updated tuple.
- Disallow DROP TABLE when there are pending deferred trigger events for the table (Tom Lane)

  Formerly the DROP would go through, leading to "could not open relation with OID nnn" errors when the triggers were eventually fired.
- Fix PL/Python memory leak involving array slices (Daniel Popowich)
- Fix pg_restore to cope with long lines (over 1KB) in TOC files (Tom Lane)

- Put in more safeguards against crashing due to division-by-zero with overly enthusiastic compiler optimization (Aurelien Jarno)
- Support use of dlopen() in FreeBSD and OpenBSD on MIPS (Tom Lane)

  There was a hard-wired assumption that this system function was not available on MIPS hardware on these systems. Use a compile-time test instead, since more recent versions have it.
- Fix compilation failures on HP-UX (Heikki Linnakangas)
- Fix path separator used by pg_regress on Cygwin (Andrew Dunstan)
- Update time zone data files to tzdata release 2011f for DST law changes in Chile, Cuba, Falkland Islands, Morocco, Samoa, and Turkey; also historical corrections for South Australia, Alaska, and Hawaii.

E.125. Release 8.2.20

Release date: 2011-01-31

This release contains a variety of fixes from 8.2.19. For information about new features in the 8.2 major release, see 第 E.145 节.

E.125.1. Migration to Version 8.2.20

A dump/restore is not required for those running 8.2.X. However, if you are upgrading from a version earlier than 8.2.14, see 第 E.131 节.

E.125.2. Changes

- Avoid failures when `EXPLAIN` tries to display a simple-form `CASE` expression (Tom Lane)

  If the `CASE`'s test expression was a constant, the planner could simplify the `CASE` into a form that confused the expression-display code, resulting in “unexpected CASE WHEN clause” errors.
- Fix assignment to an array slice that is before the existing range of subscripts (Tom Lane)

  If there was a gap between the newly added subscripts and the first pre-existing subscript, the code miscalculated how many entries needed to be copied from the old array's null bitmap, potentially leading to data corruption or crash.
- Avoid unexpected conversion overflow in planner for very distant date values (Tom Lane)

  The `date` type supports a wider range of dates than can be represented by the `timestamp` types, but the planner assumed it could always convert a date to timestamp with impunity.
- Fix pg_restore's text output for large objects (BLOBs) when `standard_conforming_strings` is on (Tom Lane)

  Although restoring directly to a database worked correctly, string escaping was incorrect if pg_restore was asked for SQL text output and `standard_conforming_strings` had been enabled in the source database.

- Fix erroneous parsing of `tsquery` values containing `... & !(subexpression) | ...` (Tom Lane)

  Queries containing this combination of operators were not executed correctly. The same error existed in `contrib/intarray`'s `query_int` type and `contrib/ltree`'s `ltxtquery` type.

- Fix buffer overrun in `contrib/intarray`'s input function for the `query_int` type (Apple)

  This bug is a security risk since the function's return address could be overwritten. Thanks to Apple Inc's security team for reporting this issue and supplying the fix. (CVE-2010-4015)

- Fix bug in `contrib/seg`'s GiST picksplit algorithm (Alexander Korotkov)

  This could result in considerable inefficiency, though not actually incorrect answers, in a GiST index on a `seg` column. If you have such an index, consider `REINDEX`ing it after installing this update. (This is identical to the bug that was fixed in `contrib/cube` in the previous update.)

E.126. Release 8.2.19

Release date: 2010-12-16

This release contains a variety of fixes from 8.2.18. For information about new features in the 8.2 major release, see 第 E.145 节.

E.126.1. Migration to Version 8.2.19

A dump/restore is not required for those running 8.2.X. However, if you are upgrading from a version earlier than 8.2.14, see 第 E.131 节.

E.126.2. Changes

- Force the default `wal_sync_method` to be `fdatasync` on Linux (Tom Lane, Marti Raudsepp)

  The default on Linux has actually been `fdatasync` for many years, but recent kernel changes caused PostgreSQL to choose `open_datasync` instead. This choice did not result in any performance improvement, and caused outright failures on certain filesystems, notably `ext4` with the `data=journal` mount option.

- Fix assorted bugs in WAL replay logic for GIN indexes (Tom Lane)

  This could result in “bad buffer id: 0” failures or corruption of index contents during replication.

- Fix recovery from base backup when the starting checkpoint WAL record is not in the same WAL segment as its redo point (Jeff Davis)

- Add support for detecting register-stack overrun on `IA64` (Tom Lane)

  The `IA64` architecture has two hardware stacks. Full prevention of stack-overrun failures requires checking both.

- Add a check for stack overflow in `copyObject()` (Tom Lane)

Certain code paths could crash due to stack overflow given a sufficiently complex query.

- Fix detection of page splits in temporary GiST indexes (Heikki Linnakangas)

  It is possible to have a "concurrent" page split in a temporary index, if for example there is an open cursor scanning the index when an insertion is done. GiST failed to detect this case and hence could deliver wrong results when execution of the cursor continued.

- Avoid memory leakage while `ANALYZE`'ing complex index expressions (Tom Lane)

- Ensure an index that uses a whole-row Var still depends on its table (Tom Lane)

  An index declared like `create index i on t (foo(t.*))` would not automatically get dropped when its table was dropped.

- Do not "inline" a SQL function with multiple `OUT` parameters (Tom Lane)

  This avoids a possible crash due to loss of information about the expected result rowtype.

- Behave correctly if `ORDER BY`, `LIMIT`, `FOR UPDATE`, or `WITH` is attached to the `VALUES` part of `INSERT ... VALUES` (Tom Lane)

- Fix constant-folding of `COALESCE()` expressions (Tom Lane)

  The planner would sometimes attempt to evaluate sub-expressions that in fact could never be reached, possibly leading to unexpected errors.

- Add print functionality for `InhRelation` nodes (Tom Lane)

  This avoids a failure when `debug_print_parse` is enabled and certain types of query are executed.

- Fix incorrect calculation of distance from a point to a horizontal line segment (Tom Lane)

  This bug affected several different geometric distance-measurement operators.

- Fix PL/pgSQL's handling of "simple" expressions to not fail in recursion or error-recovery cases (Tom Lane)

- Fix PL/Python's handling of set-returning functions (Jan Urbanski)

  Attempts to call SPI functions within the iterator generating a set result would fail.

- Fix bug in `contrib/cube`'s GiST picksplit algorithm (Alexander Korotkov)

  This could result in considerable inefficiency, though not actually incorrect answers, in a GiST index on a `cube` column. If you have such an index, consider `REINDEX`ing it after installing this update.

- Don't emit "identifier will be truncated" notices in `contrib/dblink` except when creating new connections (Itagaki Takahiro)

- Fix potential coredump on missing public key in `contrib/pgcrypto` (Marti Raudsepp)

- Fix memory leak in `contrib/xml2`'s XPath query functions (Tom Lane)

- Update time zone data files to tzdata release 2010o for DST law changes in Fiji and Samoa; also historical corrections for Hong Kong.

E.127. Release 8.2.18

Release date: 2010-10-04

This release contains a variety of fixes from 8.2.17. For information about new features in the 8.2 major release, see 第 E.145 节.

E.127.1. Migration to Version 8.2.18

A dump/restore is not required for those running 8.2.X. However, if you are upgrading from a version earlier than 8.2.14, see 第 E.131 节.

E.127.2. Changes

- Use a separate interpreter for each calling SQL userid in PL/Perl and PL/Tcl (Tom Lane)

  This change prevents security problems that can be caused by subverting Perl or Tcl code that will be executed later in the same session under another SQL user identity (for example, within a `SECURITY DEFINER` function). Most scripting languages offer numerous ways that that might be done, such as redefining standard functions or operators called by the target function. Without this change, any SQL user with Perl or Tcl language usage rights can do essentially anything with the SQL privileges of the target function's owner.

  The cost of this change is that intentional communication among Perl and Tcl functions becomes more difficult. To provide an escape hatch, PL/PerlU and PL/TclU functions continue to use only one interpreter per session. This is not considered a security issue since all such functions execute at the trust level of a database superuser already.

  It is likely that third-party procedural languages that claim to offer trusted execution have similar security issues. We advise contacting the authors of any PL you are depending on for security-critical purposes.

  Our thanks to Tim Bunce for pointing out this issue (CVE-2010-3433).

- Prevent possible crashes in `pg_get_expr()` by disallowing it from being called with an argument that is not one of the system catalog columns it's intended to be used with (Heikki Linnakangas, Tom Lane)

- Fix Windows shared-memory allocation code (Tsutomu Yamada, Magnus Hagander)

  This bug led to the often-reported "could not reattach to shared memory" error message. This is a back-patch of a fix that was applied to newer branches some time ago.

- Treat exit code 128 (`ERROR_WAIT_NO_CHILDREN`) as non-fatal on Windows (Magnus Hagander)

  Under high load, Windows processes will sometimes fail at startup with this error code. Formerly the postmaster treated this as a panic condition and restarted the whole database, but that seems to be an overreaction.

- Fix possible duplicate scans of `UNION ALL` member relations (Tom Lane)

- Fix “cannot handle unplanned sub-select” error (Tom Lane)

  This occurred when a sub-select contains a join alias reference that expands into an expression containing another sub-select.

- Reduce PANIC to ERROR in some occasionally-reported btree failure cases, and provide additional detail in the resulting error messages (Tom Lane)

  This should improve the system's robustness with corrupted indexes.

- Prevent show_session_authorization() from crashing within autovacuum processes (Tom Lane)

- Defend against functions returning setof record where not all the returned rows are actually of the same rowtype (Tom Lane)

- Fix possible failure when hashing a pass-by-reference function result (Tao Ma, Tom Lane)

- Take care to fsync the contents of lockfiles (both `postmaster.pid` and the socket lockfile) while writing them (Tom Lane)

  This omission could result in corrupted lockfile contents if the machine crashes shortly after postmaster start. That could in turn prevent subsequent attempts to start the postmaster from succeeding, until the lockfile is manually removed.

- Avoid recursion while assigning XIDs to heavily-nested subtransactions (Andres Freund, Robert Haas)

  The original coding could result in a crash if there was limited stack space.

- Fix `log_line_prefix`'s `%i` escape, which could produce junk early in backend startup (Tom Lane)

- Fix possible data corruption in `ALTER TABLE ... SET TABLESPACE` when archiving is enabled (Jeff Davis)

- Allow `CREATE DATABASE` and `ALTER DATABASE ... SET TABLESPACE` to be interrupted by query-cancel (Guillaume Lelarge)

- In PL/Python, defend against null pointer results from `PyCObject_AsVoidPtr` and `PyCObject_FromVoidPtr` (Peter Eisentraut)

- Improve `contrib/dblink`'s handling of tables containing dropped columns (Tom Lane)

- Fix connection leak after “duplicate connection name” errors in `contrib/dblink` (Itagaki Takahiro)

- Fix `contrib/dblink` to handle connection names longer than 62 bytes correctly (Itagaki Takahiro)

- Add `hstore(text, text)` function to `contrib/hstore` (Robert Haas)

  This function is the recommended substitute for the now-deprecated `=>` operator. It was back-patched so that future-proofed code can be used with older server versions. Note that the patch will be effective only after `contrib/hstore` is installed or reinstalled in a particular database. Users might prefer to execute the `CREATE FUNCTION` command by hand, instead.

- Update build infrastructure and documentation to reflect the source code repository's move from CVS to Git (Magnus Hagander and others)

- Update time zone data files to tzdata release 2010l for DST law changes in Egypt and Palestine; also historical corrections for Finland.

  This change also adds new names for two Micronesian timezones: Pacific/Chuuk is now preferred over Pacific/Truk (and the preferred abbreviation is CHUT not TRUT) and Pacific/Pohnpei is preferred over Pacific/Ponape.

- Make Windows' "N. Central Asia Standard Time" timezone map to Asia/Novosibirsk, not Asia/Almaty (Magnus Hagander)

  Microsoft changed the DST behavior of this zone in the timezone update from KB976098. Asia/Novosibirsk is a better match to its new behavior.

E.128. Release 8.2.17

Release date: 2010-05-17

This release contains a variety of fixes from 8.2.16. For information about new features in the 8.2 major release, see 第 E.145 节.

E.128.1. Migration to Version 8.2.17

A dump/restore is not required for those running 8.2.X. However, if you are upgrading from a version earlier than 8.2.14, see 第 E.131 节.

E.128.2. Changes

- Enforce restrictions in `plperl` using an opmask applied to the whole interpreter, instead of using `Safe.pm` (Tim Bunce, Andrew Dunstan)

  Recent developments have convinced us that `Safe.pm` is too insecure to rely on for making `plperl` trustable. This change removes use of `Safe.pm` altogether, in favor of using a separate interpreter with an opcode mask that is always applied. Pleasant side effects of the change include that it is now possible to use Perl's `strict` pragma in a natural way in `plperl`, and that Perl's `$a` and `$b` variables work as expected in sort routines, and that function compilation is significantly faster. (CVE-2010-1169)

- Prevent PL/Tcl from executing untrustworthy code from `pltcl_modules` (Tom)

  PL/Tcl's feature for autoloading Tcl code from a database table could be exploited for trojan-horse attacks, because there was no restriction on who could create or insert into that table. This change disables the feature unless `pltcl_modules` is owned by a superuser. (However, the permissions on the table are not checked, so installations that really need a less-than-secure modules table can still grant suitable privileges to trusted non-superusers.) Also, prevent loading code into the unrestricted "normal" Tcl interpreter unless we are really going to execute a `pltclu` function. (CVE-2010-1170)

- Fix possible crash if a cache reset message is received during rebuild of a relcache entry (Heikki)

  This error was introduced in 8.2.16 while fixing a related failure.

- Do not allow an unprivileged user to reset superuser-only parameter settings (Alvaro)

  Previously, if an unprivileged user ran `ALTER USER ... RESET ALL` for himself, or `ALTER DATABASE ... RESET ALL` for a database he owns, this would remove

all special parameter settings for the user or database, even ones that are only supposed to be changeable by a superuser. Now, the `ALTER` will only remove the parameters that the user has permission to change.

- Avoid possible crash during backend shutdown if shutdown occurs when a `CONTEXT` addition would be made to log entries (Tom)

  In some cases the context-printing function would fail because the current transaction had already been rolled back when it came time to print a log message.

- Update pl/perl's `ppport.h` for modern Perl versions (Andrew)
- Fix assorted memory leaks in pl/python (Andreas Freund, Tom)
- Prevent infinite recursion in psql when expanding a variable that refers to itself (Tom)
- Fix psql's `\copy` to not add spaces around a dot within `\copy (select ...)` (Tom)

  Addition of spaces around the decimal point in a numeric literal would result in a syntax error.

- Ensure that `contrib/pgstattuple` functions respond to cancel interrupts promptly (Tatsuhito Kasahara)
- Make server startup deal properly with the case that `shmget()` returns `EINVAL` for an existing shared memory segment (Tom)

  This behavior has been observed on BSD-derived kernels including OS X. It resulted in an entirely-misleading startup failure complaining that the shared memory request size was too large.

- Avoid possible crashes in syslogger process on Windows (Heikki)
- Deal more robustly with incomplete time zone information in the Windows registry (Magnus)
- Update the set of known Windows time zone names (Magnus)
- Update time zone data files to tzdata release 2010j for DST law changes in Argentina, Australian Antarctic, Bangladesh, Mexico, Morocco, Pakistan, Palestine, Russia, Syria, Tunisia; also historical corrections for Taiwan.

  Also, add `PKST` (Pakistan Summer Time) to the default set of timezone abbreviations.

E.129. Release 8.2.16

Release date: 2010-03-15

This release contains a variety of fixes from 8.2.15. For information about new features in the 8.2 major release, see 第 E.145 节.

E.129.1. Migration to Version 8.2.16

A dump/restore is not required for those running 8.2.X. However, if you are upgrading from a version earlier than 8.2.14, see 第 E.131 节.

E.129.2. Changes

- Add new configuration parameter `ssl_renegotiation_limit` to control how often we do session key renegotiation for an SSL connection (Magnus)

This can be set to zero to disable renegotiation completely, which may be required if a broken SSL library is used. In particular, some vendors are shipping stopgap patches for CVE-2009-3555 that cause renegotiation attempts to fail.

- Fix possible deadlock during backend startup (Tom)
- Fix possible crashes due to not handling errors during relcache reload cleanly (Tom)
- Fix possible crashes when trying to recover from a failure in subtransaction start (Tom)
- Fix server memory leak associated with use of savepoints and a client encoding different from server's encoding (Tom)
- Fix incorrect WAL data emitted during end-of-recovery cleanup of a GIST index page split (Yoichi Hirai)

  This would result in index corruption, or even more likely an error during WAL replay, if we were unlucky enough to crash during end-of-recovery cleanup after having completed an incomplete GIST insertion.

- Make `substring()` for `bit` types treat any negative length as meaning "all the rest of the string" (Tom)

  The previous coding treated only -1 that way, and would produce an invalid result value for other negative values, possibly leading to a crash (CVE-2010-0442).

- Fix integer-to-bit-string conversions to handle the first fractional byte correctly when the output bit width is wider than the given integer by something other than a multiple of 8 bits (Tom)
- Fix some cases of pathologically slow regular expression matching (Tom)
- Fix the `STOP WAL LOCATION` entry in backup history files to report the next WAL segment's name when the end location is exactly at a segment boundary (Itagaki Takahiro)
- Fix some more cases of temporary-file leakage (Heikki)

  This corrects a problem introduced in the previous minor release. One case that failed is when a plpgsql function returning set is called within another function's exception handler.

- Improve constraint exclusion processing of boolean-variable cases, in particular make it possible to exclude a partition that has a "bool_column = false" constraint (Tom)
- When reading `pg_hba.conf` and related files, do not treat `@something` as a file inclusion request if the `@` appears inside quote marks; also, never treat `@` by itself as a file inclusion request (Tom)

  This prevents erratic behavior if a role or database name starts with `@`. If you need to include a file whose path name contains spaces, you can still do so, but you must write `@"/path to/file"` rather than putting the quotes around the whole construct.

- Prevent infinite loop on some platforms if a directory is named as an inclusion target in `pg_hba.conf` and related files (Tom)
- Fix possible infinite loop if `SSL_read` or `SSL_write` fails without setting `errno` (Tom)

This is reportedly possible with some Windows versions of openssl.

- Fix psql's `numericlocale` option to not format strings it shouldn't in latex and troff output formats (Heikki)
- Make psql return the correct exit status (3) when `ON_ERROR_STOP` and `--single-transaction` are both specified and an error occurs during the implied `COMMIT` (Bruce)
- Fix plpgsql failure in one case where a composite column is set to NULL (Tom)
- Fix possible failure when calling PL/Perl functions from PL/PerlU or vice versa (Tim Bunce)
- Add `volatile` markings in PL/Python to avoid possible compiler-specific misbehavior (Zdenek Kotala)
- Ensure PL/Tcl initializes the Tcl interpreter fully (Tom)

  The only known symptom of this oversight is that the Tcl `clock` command misbehaves if using Tcl 8.5 or later.
- Prevent crash in `contrib/dblink` when too many key columns are specified to a `dblink_build_sql_*` function (Rushabh Lathia, Joe Conway)
- Fix assorted crashes in `contrib/xml2` caused by sloppy memory management (Tom)
- Make building of `contrib/xml2` more robust on Windows (Andrew)
- Fix race condition in Windows signal handling (Radu Ilie)

  One known symptom of this bug is that rows in `pg_listener` could be dropped under heavy load.
- Update time zone data files to tzdata release 2010e for DST law changes in Bangladesh, Chile, Fiji, Mexico, Paraguay, Samoa.

E.130. Release 8.2.15

Release date: 2009-12-14

This release contains a variety of fixes from 8.2.14. For information about new features in the 8.2 major release, see 第 E.145 节.

E.130.1. Migration to Version 8.2.15

A dump/restore is not required for those running 8.2.X. However, if you are upgrading from a version earlier than 8.2.14, see 第 E.131 节.

E.130.2. Changes

- Protect against indirect security threats caused by index functions changing session-local state (Gurjeet Singh, Tom)

  This change prevents allegedly-immutable index functions from possibly subverting a superuser's session (CVE-2009-4136).

- Reject SSL certificates containing an embedded null byte in the common name (CN) field (Magnus)

  This prevents unintended matching of a certificate to a server or client name during SSL validation (CVE-2009-4034).

- Fix possible crash during backend-startup-time cache initialization (Tom)

- Prevent signals from interrupting `VACUUM` at unsafe times (Alvaro)

  This fix prevents a PANIC if a `VACUUM FULL` is canceled after it's already committed its tuple movements, as well as transient errors if a plain `VACUUM` is interrupted after having truncated the table.

- Fix possible crash due to integer overflow in hash table size calculation (Tom)

  This could occur with extremely large planner estimates for the size of a hashjoin's result.

- Fix very rare crash in `inet/cidr` comparisons (Chris Mikkelson)

- Ensure that shared tuple-level locks held by prepared transactions are not ignored (Heikki)

- Fix premature drop of temporary files used for a cursor that is accessed within a subtransaction (Heikki)

- Fix incorrect logic for GiST index page splits, when the split depends on a non-first column of the index (Paul Ramsey)

- Don't error out if recycling or removing an old WAL file fails at the end of checkpoint (Heikki)

  It's better to treat the problem as non-fatal and allow the checkpoint to complete. Future checkpoints will retry the removal. Such problems are not expected in normal operation, but have been seen to be caused by misdesigned Windows anti-virus and backup software.

- Ensure WAL files aren't repeatedly archived on Windows (Heikki)

  This is another symptom that could happen if some other process interfered with deletion of a no-longer-needed file.

- Fix PAM password processing to be more robust (Tom)

  The previous code is known to fail with the combination of the Linux `pam_krb5` PAM module with Microsoft Active Directory as the domain controller. It might have problems elsewhere too, since it was making unjustified assumptions about what arguments the PAM stack would pass to it.

- Fix processing of ownership dependencies during `CREATE OR REPLACE FUNCTION` (Tom)

- Fix bug with calling `plperl` from `plperlu` or vice versa (Tom)

  An error exit from the inner function could result in crashes due to failure to re-select the correct Perl interpreter for the outer function.

- Fix session-lifespan memory leak when a PL/Perl function is redefined (Tom)

- Ensure that Perl arrays are properly converted to PostgreSQL arrays when returned by a set-returning PL/Perl function (Andrew Dunstan, Abhijit Menon-Sen)

This worked correctly already for non-set-returning functions.

- Fix rare crash in exception processing in PL/Python (Peter)
- Ensure psql's flex module is compiled with the correct system header definitions (Tom)

  This fixes build failures on platforms where `--enable-largefile` causes incompatible changes in the generated code.
- Make the postmaster ignore any `application_name` parameter in connection request packets, to improve compatibility with future libpq versions (Tom)
- Update the timezone abbreviation files to match current reality (Joachim Wieland)

  This includes adding `IDT` and `SGT` to the default timezone abbreviation set.
- Update time zone data files to tzdata release 2009s for DST law changes in Antarctica, Argentina, Bangladesh, Fiji, Novokuznetsk, Pakistan, Palestine, Samoa, Syria; also historical corrections for Hong Kong.

E.131. Release 8.2.14

Release date: 2009-09-09

This release contains a variety of fixes from 8.2.13. For information about new features in the 8.2 major release, see 第 E.145 节.

E.131.1. Migration to Version 8.2.14

A dump/restore is not required for those running 8.2.X. However, if you have any hash indexes on `interval` columns, you must `REINDEX` them after updating to 8.2.14. Also, if you are upgrading from a version earlier than 8.2.11, see 第 E.134 节.

E.131.2. Changes

- Force WAL segment switch during `pg_start_backup()` (Heikki)

  This avoids corner cases that could render a base backup unusable.
- Disallow `RESET ROLE` and `RESET SESSION AUTHORIZATION` inside security-definer functions (Tom, Heikki)

  This covers a case that was missed in the previous patch that disallowed `SET ROLE` and `SET SESSION AUTHORIZATION` inside security-definer functions. (See CVE-2007-6600)
- Make `LOAD` of an already-loaded loadable module into a no-op (Tom)

  Formerly, `LOAD` would attempt to unload and re-load the module, but this is unsafe and not all that useful.
- Disallow empty passwords during LDAP authentication (Magnus)
- Fix handling of sub-SELECTs appearing in the arguments of an outer-level aggregate function (Tom)
- Fix bugs associated with fetching a whole-row value from the output of a Sort or Materialize plan node (Tom)

- Revert planner change that disabled partial-index and constraint exclusion optimizations when there were more than 100 clauses in an AND or OR list (Tom)
- Fix hash calculation for data type `interval` (Tom)

  This corrects wrong results for hash joins on interval values. It also changes the contents of hash indexes on interval columns. If you have any such indexes, you must `REINDEX` them after updating.
- Treat `to_char(..., 'TH')` as an uppercase ordinal suffix with `'HH'/'HH12'` (Heikki)

  It was previously handled as `'th'` (lowercase).
- Fix overflow for `INTERVAL 'x ms'` when *x* is more than 2 million and integer datetimes are in use (Alex Hunsaker)
- Fix calculation of distance between a point and a line segment (Tom)

  This led to incorrect results from a number of geometric operators.
- Fix `money` data type to work in locales where currency amounts have no fractional digits, e.g. Japan (Itagaki Takahiro)
- Properly round datetime input like `00:12:57.999999999999999999999999999` (Tom)
- Fix poor choice of page split point in GiST R-tree operator classes (Teodor)
- Avoid performance degradation in bulk inserts into GIN indexes when the input values are (nearly) in sorted order (Tom)
- Correctly enforce NOT NULL domain constraints in some contexts in PL/pgSQL (Tom)
- Fix portability issues in plperl initialization (Andrew Dunstan)
- Fix pg_ctl to not go into an infinite loop if `postgresql.conf` is empty (Jeff Davis)
- Make `contrib/hstore` throw an error when a key or value is too long to fit in its data structure, rather than silently truncating it (Andrew Gierth)
- Fix `contrib/xml2`'s `xslt_process()` to properly handle the maximum number of parameters (twenty) (Tom)
- Improve robustness of libpq's code to recover from errors during `COPY FROM STDIN` (Tom)
- Avoid including conflicting readline and editline header files when both libraries are installed (Zdenek Kotala)
- Update time zone data files to tzdata release 2009l for DST law changes in Bangladesh, Egypt, Jordan, Pakistan, Argentina/San_Luis, Cuba, Jordan (historical correction only), Mauritius, Morocco, Palestine, Syria, Tunisia.

E.132. Release 8.2.13

Release date: 2009-03-16

This release contains a variety of fixes from 8.2.12. For information about new features in the 8.2 major release, see 第 E.145 节.

E.132.1. Migration to Version 8.2.13

A dump/restore is not required for those running 8.2.X. However, if you are upgrading from a version earlier than 8.2.11, see 第 E.134 节.

E.132.2. Changes

- Prevent error recursion crashes when encoding conversion fails (Tom)

  This change extends fixes made in the last two minor releases for related failure scenarios. The previous fixes were narrowly tailored for the original problem reports, but we have now recognized that any error thrown by an encoding conversion function could potentially lead to infinite recursion while trying to report the error. The solution therefore is to disable translation and encoding conversion and report the plain-ASCII form of any error message, if we find we have gotten into a recursive error reporting situation. (CVE-2009-0922)

- Disallow `CREATE CONVERSION` with the wrong encodings for the specified conversion function (Heikki)

  This prevents one possible scenario for encoding conversion failure. The previous change is a backstop to guard against other kinds of failures in the same area.

- Fix core dump when `to_char()` is given format codes that are inappropriate for the type of the data argument (Tom)

- Fix possible failure in `contrib/tsearch2` when C locale is used with a multi-byte encoding (Teodor)

  Crashes were possible on platforms where `wchar_t` is narrower than `int`; Windows in particular.

- Fix extreme inefficiency in `contrib/tsearch2` parser's handling of an email-like string containing multiple `@` characters (Heikki)

- Fix decompilation of `CASE WHEN` with an implicit coercion (Tom)

  This mistake could lead to Assert failures in an Assert-enabled build, or an "unexpected CASE WHEN clause" error message in other cases, when trying to examine or dump a view.

- Fix possible misassignment of the owner of a TOAST table's rowtype (Tom)

  If `CLUSTER` or a rewriting variant of `ALTER TABLE` were executed by someone other than the table owner, the `pg_type` entry for the table's TOAST table would end up marked as owned by that someone. This caused no immediate problems, since the permissions on the TOAST rowtype aren't examined by any ordinary database operation. However, it could lead to unexpected failures if one later tried to drop the role that issued the command (in 8.1 or 8.2), or "owner of data type appears to be invalid" warnings from pg_dump after having done so (in 8.3).

- Fix PL/pgSQL to not treat `INTO` after `INSERT` as an INTO-variables clause anywhere in the string, not only at the start; in particular, don't fail for `INSERT INTO` within `CREATE RULE` (Tom)

- Clean up PL/pgSQL error status variables fully at block exit (Ashesh Vashi and Dave Page)

  This is not a problem for PL/pgSQL itself, but the omission could cause the PL/pgSQL Debugger to crash while examining the state of a function.

- Retry failed calls to `CallNamedPipe()` on Windows (Steve Marshall, Magnus)

  It appears that this function can sometimes fail transiently; we previously treated any failure as a hard error, which could confuse `LISTEN`/`NOTIFY` as well as other operations.

- Add `MUST` (Mauritius Island Summer Time) to the default list of known timezone abbreviations (Xavier Bugaud)

E.133. Release 8.2.12

Release date: 2009-02-02

This release contains a variety of fixes from 8.2.11. For information about new features in the 8.2 major release, see 第 E.145 节.

E.133.1. Migration to Version 8.2.12

A dump/restore is not required for those running 8.2.X. However, if you are upgrading from a version earlier than 8.2.11, see 第 E.134 节.

E.133.2. Changes

- Improve handling of URLs in `headline()` function (Teodor)

- Improve handling of overlength headlines in `headline()` function (Teodor)

- Prevent possible Assert failure or misconversion if an encoding conversion is created with the wrong conversion function for the specified pair of encodings (Tom, Heikki)

- Fix possible Assert failure if a statement executed in PL/pgSQL is rewritten into another kind of statement, for example if an `INSERT` is rewritten into an `UPDATE` (Heikki)

- Ensure that a snapshot is available to datatype input functions (Tom)

  This primarily affects domains that are declared with `CHECK` constraints involving user-defined stable or immutable functions. Such functions typically fail if no snapshot has been set.

- Make it safer for SPI-using functions to be used within datatype I/O; in particular, to be used in domain check constraints (Tom)

- Avoid unnecessary locking of small tables in `VACUUM` (Heikki)

- Fix a problem that made `UPDATE RETURNING tableoid` return zero instead of the correct OID (Tom)

- Fix planner misestimation of selectivity when transitive equality is applied to an outer-join clause (Tom)

  This could result in bad plans for queries like `... from a left join b on a.a1 = b.b1 where a.a1 = 42 ...`

- Improve optimizer's handling of long `IN` lists (Tom)

  This change avoids wasting large amounts of time on such lists when constraint exclusion is enabled.

- Ensure that the contents of a holdable cursor don't depend on the contents of TOAST tables (Tom)

  Previously, large field values in a cursor result might be represented as TOAST pointers, which would fail if the referenced table got dropped before the cursor is read, or if the large value is deleted and then vacuumed away. This cannot happen with an ordinary cursor, but it could with a cursor that is held past its creating transaction.

- Fix memory leak when a set-returning function is terminated without reading its whole result (Tom)

- Fix `contrib/dblink`'s `dblink_get_result(text,bool)` function (Joe)

- Fix possible garbage output from `contrib/sslinfo` functions (Tom)

- Fix configure script to properly report failure when unable to obtain linkage information for PL/Perl (Andrew)

- Make all documentation reference `pgsql-bugs` and/or `pgsql-hackers` as appropriate, instead of the now-decommissioned `pgsql-ports` and `pgsql-patches` mailing lists (Tom)

- Update time zone data files to tzdata release 2009a (for Kathmandu and historical DST corrections in Switzerland, Cuba)

E.134. Release 8.2.11

Release date: 2008-11-03

This release contains a variety of fixes from 8.2.10. For information about new features in the 8.2 major release, see 第 E.145 节.

E.134.1. Migration to Version 8.2.11

A dump/restore is not required for those running 8.2.X. However, if you are upgrading from a version earlier than 8.2.7, see 第 E.138 节. Also, if you were running a previous 8.2.X release, it is recommended to `REINDEX` all GiST indexes after the upgrade.

E.134.2. Changes

- Fix GiST index corruption due to marking the wrong index entry "dead" after a deletion (Teodor)

  This would result in index searches failing to find rows they should have found. Corrupted indexes can be fixed with `REINDEX`.

- Fix backend crash when the client encoding cannot represent a localized error message (Tom)

  We have addressed similar issues before, but it would still fail if the "character has no equivalent" message itself couldn't be converted. The fix is to disable localization and send the plain ASCII error message when we detect such a situation.

- Fix possible crash when deeply nested functions are invoked from a trigger (Tom)

- Improve optimization of *`expression`* `IN` (*`expression-list`*) queries (Tom, per an idea from Robert Haas)

Cases in which there are query variables on the right-hand side had been handled less efficiently in 8.2.x and 8.3.x than in prior versions. The fix restores 8.1 behavior for such cases.

- Fix mis-expansion of rule queries when a sub-`SELECT` appears in a function call in `FROM`, a multi-row `VALUES` list, or a `RETURNING` list (Tom)

  The usual symptom of this problem is an “unrecognized node type” error.

- Fix memory leak during rescan of a hashed aggregation plan (Neil)
- Ensure an error is reported when a newly-defined PL/pgSQL trigger function is invoked as a normal function (Tom)
- Prevent possible collision of `relfilenode` numbers when moving a table to another tablespace with `ALTER SET TABLESPACE` (Heikki)

  The command tried to re-use the existing filename, instead of picking one that is known unused in the destination directory.

- Fix incorrect tsearch2 headline generation when single query item matches first word of text (Sushant Sinha)
- Fix improper display of fractional seconds in interval values when using a non-ISO datestyle in an `--enable-integer-datetimes` build (Ron Mayer)
- Ensure `SPI_getvalue` and `SPI_getbinval` behave correctly when the passed tuple and tuple descriptor have different numbers of columns (Tom)

  This situation is normal when a table has had columns added or removed, but these two functions didn't handle it properly. The only likely consequence is an incorrect error indication.

- Fix ecpg's parsing of `CREATE ROLE` (Michael)
- Fix recent breakage of `pg_ctl restart` (Tom)
- Ensure `pg_control` is opened in binary mode (Itagaki Takahiro)

  pg_controldata and pg_resetxlog did this incorrectly, and so could fail on Windows.

- Update time zone data files to tzdata release 2008i (for DST law changes in Argentina, Brazil, Mauritius, Syria)

E.135. Release 8.2.10

Release date: 2008-09-22

This release contains a variety of fixes from 8.2.9. For information about new features in the 8.2 major release, see 第 E.145 节.

E.135.1. Migration to Version 8.2.10

A dump/restore is not required for those running 8.2.X. However, if you are upgrading from a version earlier than 8.2.7, see 第 E.138 节.

E.135.2. Changes

- Fix bug in btree WAL recovery code (Heikki)

Recovery failed if the WAL ended partway through a page split operation.

- Fix potential miscalculation of `datfrozenxid` (Alvaro)

  This error may explain some recent reports of failure to remove old `pg_clog` data.

- Widen local lock counters from 32 to 64 bits (Tom)

  This responds to reports that the counters could overflow in sufficiently long transactions, leading to unexpected "lock is already held" errors.

- Fix possible duplicate output of tuples during a GiST index scan (Teodor)

- Fix missed permissions checks when a view contains a simple `UNION ALL` construct (Heikki)

  Permissions for the referenced tables were checked properly, but not permissions for the view itself.

- Add checks in executor startup to ensure that the tuples produced by an `INSERT` or `UPDATE` will match the target table's current rowtype (Tom)

  `ALTER COLUMN TYPE`, followed by re-use of a previously cached plan, could produce this type of situation. The check protects against data corruption and/or crashes that could ensue.

- Fix possible repeated drops during `DROP OWNED` (Tom)

  This would typically result in strange errors such as "cache lookup failed for relation NNN".

- Fix `AT TIME ZONE` to first try to interpret its timezone argument as a timezone abbreviation, and only try it as a full timezone name if that fails, rather than the other way around as formerly (Tom)

  The timestamp input functions have always resolved ambiguous zone names in this order. Making `AT TIME ZONE` do so as well improves consistency, and fixes a compatibility bug introduced in 8.1: in ambiguous cases we now behave the same as 8.0 and before did, since in the older versions `AT TIME ZONE` accepted only abbreviations.

- Fix datetime input functions to correctly detect integer overflow when running on a 64-bit platform (Tom)

- Prevent integer overflows during units conversion when displaying a configuration parameter that has units (Tom)

- Improve performance of writing very long log messages to syslog (Tom)

- Allow spaces in the suffix part of an LDAP URL in `pg_hba.conf` (Tom)

- Fix bug in backwards scanning of a cursor on a `SELECT DISTINCT ON` query (Tom)

- Fix planner bug with nested sub-select expressions (Tom)

  If the outer sub-select has no direct dependency on the parent query, but the inner one does, the outer value might not get recalculated for new parent query rows.

- Fix planner to estimate that `GROUP BY` expressions yielding boolean results always result in two groups, regardless of the expressions' contents (Tom)

This is very substantially more accurate than the regular `GROUP BY` estimate for certain boolean tests like *col* `IS NULL`.

- Fix PL/pgSQL to not fail when a `FOR` loop's target variable is a record containing composite-type fields (Tom)
- Fix PL/Tcl to behave correctly with Tcl 8.5, and to be more careful about the encoding of data sent to or from Tcl (Tom)
- On Windows, work around a Microsoft bug by preventing libpq from trying to send more than 64kB per system call (Magnus)
- Improve pg_dump and pg_restore's error reporting after failure to send a SQL command (Tom)
- Fix pg_ctl to properly preserve postmaster command-line arguments across a `restart` (Bruce)
- Update time zone data files to tzdata release 2008f (for DST law changes in Argentina, Bahamas, Brazil, Mauritius, Morocco, Pakistan, Palestine, and Paraguay)

E.136. Release 8.2.9

Release date: 2008-06-12

This release contains one serious and one minor bug fix over 8.2.8. For information about new features in the 8.2 major release, see 第 E.145 节.

E.136.1. Migration to Version 8.2.9

A dump/restore is not required for those running 8.2.X. However, if you are upgrading from a version earlier than 8.2.7, see 第 E.138 节.

E.136.2. Changes

- Make `pg_get_ruledef()` parenthesize negative constants (Tom)

  Before this fix, a negative constant in a view or rule might be dumped as, say, `-42::integer`, which is subtly incorrect: it should be `(-42)::integer` due to operator precedence rules. Usually this would make little difference, but it could interact with another recent patch to cause PostgreSQL to reject what had been a valid `SELECT DISTINCT` view query. Since this could result in pg_dump output failing to reload, it is being treated as a high-priority fix. The only released versions in which dump output is actually incorrect are 8.3.1 and 8.2.7.

- Make `ALTER AGGREGATE ... OWNER TO` update `pg_shdepend` (Tom)

  This oversight could lead to problems if the aggregate was later involved in a `DROP OWNED` or `REASSIGN OWNED` operation.

E.137. Release 8.2.8

Release date: never released

This release contains a variety of fixes from 8.2.7. For information about new features in the 8.2 major release, see 第 E.145 节.

E.137.1. Migration to Version 8.2.8

A dump/restore is not required for those running 8.2.X. However, if you are upgrading from a version earlier than 8.2.7, see 第 E.138 节.

E.137.2. Changes

- Fix `ERRORDATA_STACK_SIZE exceeded` crash that occurred on Windows when using UTF-8 database encoding and a different client encoding (Tom)
- Fix `ALTER TABLE ADD COLUMN ... PRIMARY KEY` so that the new column is correctly checked to see if it's been initialized to all non-nulls (Brendan Jurd)

  Previous versions neglected to check this requirement at all.
- Fix possible `CREATE TABLE` failure when inheriting the "same" constraint from multiple parent relations that inherited that constraint from a common ancestor (Tom)
- Fix `pg_get_ruledef()` to show the alias, if any, attached to the target table of an `UPDATE` or `DELETE` (Tom)
- Fix GIN bug that could result in a `too many LWLocks taken` failure (Teodor)
- Avoid possible crash when decompressing corrupted data (Zdenek Kotala)
- Repair two places where SIGTERM exit of a backend could leave corrupted state in shared memory (Tom)

  Neither case is very important if SIGTERM is used to shut down the whole database cluster together, but there was a problem if someone tried to SIGTERM individual backends.
- Fix conversions between ISO-8859-5 and other encodings to handle Cyrillic "Yo" characters (`e` and `E` with two dots) (Sergey Burladyan)
- Fix several datatype input functions, notably `array_in()`, that were allowing unused bytes in their results to contain uninitialized, unpredictable values (Tom)

  This could lead to failures in which two apparently identical literal values were not seen as equal, resulting in the parser complaining about unmatched `ORDER BY` and `DISTINCT` expressions.
- Fix a corner case in regular-expression substring matching (`substring(`*`string`* `from` *`pattern`*`)`) (Tom)

  The problem occurs when there is a match to the pattern overall but the user has specified a parenthesized subexpression and that subexpression hasn't got a match. An example is `substring('foo' from 'foo(bar)?')`. This should return NULL, since `(bar)` isn't matched, but it was mistakenly returning the whole-pattern match instead (ie, `foo`).
- Update time zone data files to tzdata release 2008c (for DST law changes in Morocco, Iraq, Choibalsan, Pakistan, Syria, Cuba, and Argentina/San_Luis)
- Fix incorrect result from ecpg's `PGTYPEStimestamp_sub()` function (Michael)
- Fix broken GiST comparison function for `contrib/tsearch2`'s `tsquery` type (Teodor)

- Fix possible crashes in `contrib/cube` functions (Tom)
- Fix core dump in `contrib/xml2`'s `xpath_table()` function when the input query returns a NULL value (Tom)
- Fix `contrib/xml2`'s makefile to not override `CFLAGS` (Tom)
- Fix `DatumGetBool` macro to not fail with gcc 4.3 (Tom)

  This problem affects "old style" (V0) C functions that return boolean. The fix is already in 8.3, but the need to back-patch it was not realized at the time.

E.138. Release 8.2.7

Release date: 2008-03-17

This release contains a variety of fixes from 8.2.6. For information about new features in the 8.2 major release, see 第 E.145 节.

E.138.1. Migration to Version 8.2.7

A dump/restore is not required for those running 8.2.X. However, you might need to `REINDEX` indexes on textual columns after updating, if you are affected by the Windows locale issue described below.

E.138.2. Changes

- Fix character string comparison for Windows locales that consider different character combinations as equal (Tom)

  This fix applies only on Windows and only when using UTF-8 database encoding. The same fix was made for all other cases over two years ago, but Windows with UTF-8 uses a separate code path that was not updated. If you are using a locale that considers some non-identical strings as equal, you may need to `REINDEX` to fix existing indexes on textual columns.

- Repair potential deadlock between concurrent `VACUUM FULL` operations on different system catalogs (Tom)
- Fix longstanding `LISTEN`/`NOTIFY` race condition (Tom)

  In rare cases a session that had just executed a `LISTEN` might not get a notification, even though one would be expected because the concurrent transaction executing `NOTIFY` was observed to commit later.

  A side effect of the fix is that a transaction that has executed a not-yet-committed `LISTEN` command will not see any row in `pg_listener` for the `LISTEN`, should it choose to look; formerly it would have. This behavior was never documented one way or the other, but it is possible that some applications depend on the old behavior.

- Disallow `LISTEN` and `UNLISTEN` within a prepared transaction (Tom)

  This was formerly allowed but trying to do it had various unpleasant consequences, notably that the originating backend could not exit as long as an `UNLISTEN` remained uncommitted.

- Disallow dropping a temporary table within a prepared transaction (Heikki)

This was correctly disallowed by 8.1, but the check was inadvertently broken in 8.2.

- Fix rare crash when an error occurs during a query using a hash index (Heikki)
- Fix memory leaks in certain usages of set-returning functions (Neil)
- Fix input of datetime values for February 29 in years BC (Tom)

  The former coding was mistaken about which years were leap years.
- Fix "unrecognized node type" error in some variants of `ALTER OWNER` (Tom)
- Ensure `pg_stat_activity.waiting` flag is cleared when a lock wait is aborted (Tom)
- Fix handling of process permissions on Windows Vista (Dave, Magnus)

  In particular, this fix allows starting the server as the Administrator user.
- Update time zone data files to tzdata release 2008a (in particular, recent Chile changes); adjust timezone abbreviation `VET` (Venezuela) to mean UTC-4:30, not UTC-4:00 (Tom)
- Fix pg_ctl to correctly extract the postmaster's port number from command-line options (Itagaki Takahiro, Tom)

  Previously, `pg_ctl start -w` could try to contact the postmaster on the wrong port, leading to bogus reports of startup failure.
- Use `-fwrapv` to defend against possible misoptimization in recent gcc versions (Tom)

  This is known to be necessary when building PostgreSQL with gcc 4.3 or later.
- Correctly enforce `statement_timeout` values longer than `INT_MAX` microseconds (about 35 minutes) (Tom)

  This bug affects only builds with `--enable-integer-datetimes`.
- Fix "unexpected PARAM_SUBLINK ID" planner error when constant-folding simplifies a sub-select (Tom)
- Fix logical errors in constraint-exclusion handling of `IS NULL` and `NOT` expressions (Tom)

  The planner would sometimes exclude partitions that should not have been excluded because of the possibility of NULL results.
- Fix another cause of "failed to build any N-way joins" planner errors (Tom)

  This could happen in cases where a clauseless join needed to be forced before a join clause could be exploited.
- Fix incorrect constant propagation in outer-join planning (Tom)

  The planner could sometimes incorrectly conclude that a variable could be constrained to be equal to a constant, leading to wrong query results.
- Fix display of constant expressions in `ORDER BY` and `GROUP BY` (Tom)

  An explicitly casted constant would be shown incorrectly. This could for example lead to corruption of a view definition during dump and reload.

- Fix libpq to handle NOTICE messages correctly during COPY OUT (Tom)

  This failure has only been observed to occur when a user-defined datatype's output routine issues a NOTICE, but there is no guarantee it couldn't happen due to other causes.

E.139. Release 8.2.6

Release date: 2008-01-07

This release contains a variety of fixes from 8.2.5, including fixes for significant security issues. For information about new features in the 8.2 major release, see 第 E.145 节.

E.139.1. Migration to Version 8.2.6

A dump/restore is not required for those running 8.2.X.

E.139.2. Changes

- Prevent functions in indexes from executing with the privileges of the user running `VACUUM`, `ANALYZE`, etc (Tom)

  Functions used in index expressions and partial-index predicates are evaluated whenever a new table entry is made. It has long been understood that this poses a risk of trojan-horse code execution if one modifies a table owned by an untrustworthy user. (Note that triggers, defaults, check constraints, etc. pose the same type of risk.) But functions in indexes pose extra danger because they will be executed by routine maintenance operations such as `VACUUM FULL`, which are commonly performed automatically under a superuser account. For example, a nefarious user can execute code with superuser privileges by setting up a trojan-horse index definition and waiting for the next routine vacuum. The fix arranges for standard maintenance operations (including `VACUUM`, `ANALYZE`, `REINDEX`, and `CLUSTER`) to execute as the table owner rather than the calling user, using the same privilege-switching mechanism already used for `SECURITY DEFINER` functions. To prevent bypassing this security measure, execution of `SET SESSION AUTHORIZATION` and `SET ROLE` is now forbidden within a `SECURITY DEFINER` context. (CVE-2007-6600)

- Repair assorted bugs in the regular-expression package (Tom, Will Drewry)

  Suitably crafted regular-expression patterns could cause crashes, infinite or near-infinite looping, and/or massive memory consumption, all of which pose denial-of-service hazards for applications that accept regex search patterns from untrustworthy sources. (CVE-2007-4769, CVE-2007-4772, CVE-2007-6067)

- Require non-superusers who use `/contrib/dblink` to use only password authentication, as a security measure (Joe)

  The fix that appeared for this in 8.2.5 was incomplete, as it plugged the hole for only some `dblink` functions. (CVE-2007-6601, CVE-2007-3278)

- Fix bugs in WAL replay for GIN indexes (Teodor)

- Fix GIN index build to work properly when `maintenance_work_mem` is 4GB or more (Tom)

- Update time zone data files to tzdata release 2007k (in particular, recent Argentina changes) (Tom)

- Improve planner's handling of LIKE/regex estimation in non-C locales (Tom)
- Fix planning-speed problem for deep outer-join nests, as well as possible poor choice of join order (Tom)
- Fix planner failure in some cases of `WHERE false AND var IN (SELECT ...)` (Tom)
- Make `CREATE TABLE ... SERIAL` and `ALTER SEQUENCE ... OWNED BY` not change the `currval()` state of the sequence (Tom)
- Preserve the tablespace and storage parameters of indexes that are rebuilt by `ALTER TABLE ... ALTER COLUMN TYPE` (Tom)
- Make archive recovery always start a new WAL timeline, rather than only when a recovery stop time was used (Simon)

  This avoids a corner-case risk of trying to overwrite an existing archived copy of the last WAL segment, and seems simpler and cleaner than the original definition.
- Make `VACUUM` not use all of `maintenance_work_mem` when the table is too small for it to be useful (Alvaro)
- Fix potential crash in `translate()` when using a multibyte database encoding (Tom)
- Make `corr()` return the correct result for negative correlation values (Neil)
- Fix overflow in `extract(epoch from interval)` for intervals exceeding 68 years (Tom)
- Fix PL/Perl to not fail when a UTF-8 regular expression is used in a trusted function (Andrew)
- Fix PL/Perl to cope when platform's Perl defines type `bool` as `int` rather than `char` (Tom)

  While this could theoretically happen anywhere, no standard build of Perl did things this way ... until Mac OS X 10.5.
- Fix PL/Python to work correctly with Python 2.5 on 64-bit machines (Marko Kreen)
- Fix PL/Python to not crash on long exception messages (Alvaro)
- Fix pg_dump to correctly handle inheritance child tables that have default expressions different from their parent's (Tom)
- Fix libpq crash when `PGPASSFILE` refers to a file that is not a plain file (Martin Pitt)
- ecpg parser fixes (Michael)
- Make `contrib/pgcrypto` defend against OpenSSL libraries that fail on keys longer than 128 bits; which is the case at least on some Solaris versions (Marko Kreen)
- Make `contrib/tablefunc`'s `crosstab()` handle NULL rowid as a category in its own right, rather than crashing (Joe)
- Fix `tsvector` and `tsquery` output routines to escape backslashes correctly (Teodor, Bruce)
- Fix crash of `to_tsvector()` on huge input strings (Teodor)

- Require a specific version of Autoconf to be used when re-generating the `configure` script (Peter)

  This affects developers and packagers only. The change was made to prevent accidental use of untested combinations of Autoconf and PostgreSQL versions. You can remove the version check if you really want to use a different Autoconf version, but it's your responsibility whether the result works or not.

- Update `gettimeofday` configuration check so that PostgreSQL can be built on newer versions of MinGW (Magnus)

E.140. Release 8.2.5

Release date: 2007-09-17

This release contains a variety of fixes from 8.2.4. For information about new features in the 8.2 major release, see 第 E.145 节.

E.140.1. Migration to Version 8.2.5

A dump/restore is not required for those running 8.2.X.

E.140.2. Changes

- Prevent index corruption when a transaction inserts rows and then aborts close to the end of a concurrent `VACUUM` on the same table (Tom)
- Fix `ALTER DOMAIN ADD CONSTRAINT` for cases involving domains over domains (Tom)
- Make `CREATE DOMAIN ... DEFAULT NULL` work properly (Tom)
- Fix some planner problems with outer joins, notably poor size estimation for `t1 LEFT JOIN t2 WHERE t2.col IS NULL` (Tom)
- Allow the `interval` data type to accept input consisting only of milliseconds or microseconds (Neil)
- Allow timezone name to appear before the year in `timestamp` input (Tom)
- Fixes for GIN indexes used by `/contrib/tsearch2` (Teodor)
- Speed up rtree index insertion (Teodor)
- Fix excessive logging of SSL error messages (Tom)
- Fix logging so that log messages are never interleaved when using the syslogger process (Andrew)
- Fix crash when `log_min_error_statement` logging runs out of memory (Tom)
- Fix incorrect handling of some foreign-key corner cases (Tom)
- Fix `stddev_pop(numeric)` and `var_pop(numeric)` (Tom)
- Prevent `REINDEX` and `CLUSTER` from failing due to attempting to process temporary tables of other sessions (Alvaro)
- Update the time zone database rules, particularly New Zealand's upcoming changes (Tom)

- Windows socket and semaphore improvements (Magnus)
- Make `pg_ctl -w` work properly in Windows service mode (Dave Page)
- Fix memory allocation bug when using MIT Kerberos on Windows (Magnus)
- Suppress timezone name (`%Z`) in log timestamps on Windows because of possible encoding mismatches (Tom)
- Require non-superusers who use `/contrib/dblink` to use only password authentication, as a security measure (Joe)
- Restrict `/contrib/pgstattuple` functions to superusers, for security reasons (Tom)
- Do not let `/contrib/intarray` try to make its GIN opclass the default (this caused problems at dump/restore) (Tom)

E.141. Release 8.2.4

Release date: 2007-04-23

This release contains a variety of fixes from 8.2.3, including a security fix. For information about new features in the 8.2 major release, see 第 E.145 节.

E.141.1. Migration to Version 8.2.4

A dump/restore is not required for those running 8.2.X.

E.141.2. Changes

- Support explicit placement of the temporary-table schema within `search_path`, and disable searching it for functions and operators (Tom)

  This is needed to allow a security-definer function to set a truly secure value of `search_path`. Without it, an unprivileged SQL user can use temporary objects to execute code with the privileges of the security-definer function (CVE-2007-2138). See `CREATE FUNCTION` for more information.
- Fix `shared_preload_libraries` for Windows by forcing reload in each backend (Korry Douglas)
- Fix `to_char()` so it properly upper/lower cases localized day or month names (Pavel Stehule)
- `/contrib/tsearch2` crash fixes (Teodor)
- Require `COMMIT PREPARED` to be executed in the same database as the transaction was prepared in (Heikki)
- Allow `pg_dump` to do binary backups larger than two gigabytes on Windows (Magnus)
- New traditional (Taiwan) Chinese FAQ (Zhou Daojing)
- Prevent the statistics collector from writing to disk too frequently (Tom)
- Fix potential-data-corruption bug in how `VACUUM FULL` handles `UPDATE` chains (Tom, Pavan Deolasee)
- Fix bug in domains that use array types (Tom)

- Fix `pg_dump` so it can dump a serial column's sequence using `-t` when not also dumping the owning table (Tom)
- Planner fixes, including improving outer join and bitmap scan selection logic (Tom)
- Fix possible wrong answers or crash when a PL/pgSQL function tries to `RETURN` from within an `EXCEPTION` block (Tom)
- Fix PANIC during enlargement of a hash index (Tom)
- Fix POSIX-style timezone specs to follow new USA DST rules (Tom)

E.142. Release 8.2.3

Release date: 2007-02-07

This release contains two fixes from 8.2.2. For information about new features in the 8.2 major release, see 第 E.145 节.

E.142.1. Migration to Version 8.2.3

A dump/restore is not required for those running 8.2.X.

E.142.2. Changes

- Remove overly-restrictive check for type length in constraints and functional indexes(Tom)
- Fix optimization so MIN/MAX in subqueries can again use indexes (Tom)

E.143. Release 8.2.2

Release date: 2007-02-05

This release contains a variety of fixes from 8.2.1, including a security fix. For information about new features in the 8.2 major release, see 第 E.145 节.

E.143.1. Migration to Version 8.2.2

A dump/restore is not required for those running 8.2.X.

E.143.2. Changes

- Remove security vulnerabilities that allowed connected users to read backend memory (Tom)

  The vulnerabilities involve suppressing the normal check that a SQL function returns the data type it's declared to, and changing the data type of a table column (CVE-2007-0555, CVE-2007-0556). These errors can easily be exploited to cause a backend crash, and in principle might be used to read database content that the user should not be able to access.
- Fix not-so-rare-anymore bug wherein btree index page splits could fail due to choosing an infeasible split point (Heikki Linnakangas)
- Fix Borland C compile scripts (L Bayuk)
- Properly handle `to_char('CC')` for years ending in `00` (Tom)

Year 2000 is in the twentieth century, not the twenty-first.

- `/contrib/tsearch2` localization improvements (Tatsuo, Teodor)
- Fix incorrect permission check in `information_schema.key_column_usage` view (Tom)

  The symptom is "relation with OID nnnnn does not exist" errors. To get this fix without using `initdb`, use `CREATE OR REPLACE VIEW` to install the corrected definition found in `share/information_schema.sql`. Note you will need to do this in each database.
- Improve `VACUUM` performance for databases with many tables (Tom)
- Fix for rare Assert() crash triggered by `UNION` (Tom)
- Fix potentially incorrect results from index searches using `ROW` inequality conditions (Tom)
- Tighten security of multi-byte character processing for UTF8 sequences over three bytes long (Tom)
- Fix bogus "permission denied" failures occurring on Windows due to attempts to fsync already-deleted files (Magnus, Tom)
- Fix bug that could cause the statistics collector to hang on Windows (Magnus)

  This would in turn lead to autovacuum not working.
- Fix possible crashes when an already-in-use PL/pgSQL function is updated (Tom)
- Improve PL/pgSQL handling of domain types (Sergiy Vyshnevetskiy, Tom)
- Fix possible errors in processing PL/pgSQL exception blocks (Tom)

E.144. Release 8.2.1

Release date: 2007-01-08

This release contains a variety of fixes from 8.2. For information about new features in the 8.2 major release, see 第 E.145 节.

E.144.1. Migration to Version 8.2.1

A dump/restore is not required for those running 8.2.

E.144.2. Changes

- Fix crash with `SELECT ... LIMIT ALL` (also `LIMIT NULL`) (Tom)
- `Several /contrib/tsearch2` fixes (Teodor)
- On Windows, make log messages coming from the operating system use ASCII encoding (Hiroshi Saito)

  This fixes a conversion problem when there is a mismatch between the encoding of the operating system and database server.
- Fix Windows linking of pg_dump using `win32.mak` (Hiroshi Saito)

- Fix planner mistakes for outer join queries (Tom)
- Fix several problems in queries involving sub-SELECTs (Tom)
- Fix potential crash in SPI during subtransaction abort (Tom)

  This affects all PL functions since they all use SPI.
- Improve build speed of PDF documentation (Peter)
- Re-add JST (Japan) timezone abbreviation (Tom)
- Improve optimization decisions related to index scans (Tom)
- Have psql print multi-byte combining characters as before, rather than output as `\u` (Tom)
- Improve index usage of regular expressions that use parentheses (Tom)

  This improves psql `\d` performance also.
- Make pg_dumpall assume that databases have public `CONNECT` privilege, when dumping from a pre-8.2 server (Tom)

  This preserves the previous behavior that anyone can connect to a database if allowed by `pg_hba.conf`.

E.145. Release 8.2

Release date: 2006-12-05

E.145.1. Overview

This release adds many functionality and performance improvements that were requested by users, including:

- Query language enhancements including `INSERT/UPDATE/DELETE RETURNING`, multirow `VALUES` lists, and optional target-table alias in `UPDATE/DELETE`
- Index creation without blocking concurrent `INSERT/UPDATE/DELETE` operations
- Many query optimization improvements, including support for reordering outer joins
- Improved sorting performance with lower memory usage
- More efficient locking with better concurrency
- More efficient vacuuming
- Easier administration of warm standby servers
- New `FILLFACTOR` support for tables and indexes
- Monitoring, logging, and performance tuning additions
- More control over creating and dropping objects
- Table inheritance relationships can be defined for and removed from pre-existing tables
- `COPY TO` can copy the output of an arbitrary `SELECT` statement

- Array improvements, including nulls in arrays
- Aggregate-function improvements, including multiple-input aggregates and SQL:2003 statistical functions
- Many `contrib/` improvements

E.145.2. Migration to Version 8.2

A dump/restore using pg_dump is required for those wishing to migrate data from any previous release.

Observe the following incompatibilities:

- Set `escape_string_warning` to `on` by default (Bruce)

  This issues a warning if backslash escapes are used in non-escape (non-`E''`) strings.

- Change the row constructor syntax (`ROW(...)`) so that list elements `foo.*` will be expanded to a list of their member fields, rather than creating a nested row type field as formerly (Tom)

  The new behavior is substantially more useful since it allows, for example, triggers to check for data changes with `IF row(new.*) IS DISTINCT FROM row(old.*)`. The old behavior is still available by omitting `.*`.

- Make row comparisons follow SQL standard semantics and allow them to be used in index scans (Tom)

  Previously, row = and <> comparisons followed the standard but < <= > >= did not. A row comparison can now be used as an index constraint for a multicolumn index matching the row value.

- Make row `IS [NOT] NULL` tests follow SQL standard semantics (Tom)

  The former behavior conformed to the standard for simple cases with `IS NULL`, but `IS NOT NULL` would return true if any row field was non-null, whereas the standard says it should return true only when all fields are non-null.

- Make `SET CONSTRAINT` affect only one constraint (Kris Jurka)

  In previous releases, `SET CONSTRAINT` modified all constraints with a matching name. In this release, the schema search path is used to modify only the first matching constraint. A schema specification is also supported. This more nearly conforms to the SQL standard.

- Remove `RULE` permission for tables, for security reasons (Tom)

  As of this release, only a table's owner can create or modify rules for the table. For backwards compatibility, `GRANT/REVOKE RULE` is still accepted, but it does nothing.

- Array comparison improvements (Tom)

  Now array dimensions are also compared.

- Change array concatenation to match documented behavior (Tom)

  This changes the previous behavior where concatenation would modify the array lower bound.

- Make command-line options of postmaster and postgres identical (Peter)

This allows the postmaster to pass arguments to each backend without using `-o`. Note that some options are now only available as long-form options, because there were conflicting single-letter options.

- Deprecate use of postmaster symbolic link (Peter)

  postmaster and postgres commands now act identically, with the behavior determined by command-line options. The postmaster symbolic link is kept for compatibility, but is not really needed.

- Change `log_duration` to output even if the query is not output (Tom)

  In prior releases, `log_duration` only printed if the query appeared earlier in the log.

- Make `to_char(time)` and `to_char(interval)` treat `HH` and `HH12` as 12-hour intervals

  Most applications should use `HH24` unless they want a 12-hour display.

- Zero unmasked bits in conversion from `INET` to `CIDR` (Tom)

  This ensures that the converted value is actually valid for `CIDR`.

- Remove `australian_timezones` configuration variable (Joachim Wieland)

  This variable has been superseded by a more general facility for configuring timezone abbreviations.

- Improve cost estimation for nested-loop index scans (Tom)

  This might eliminate the need to set unrealistically small values of `random_page_cost`. If you have been using a very small `random_page_cost`, please recheck your test cases.

- Change behavior of `pg_dump -n` and `-t` options. (Greg Sabino Mullane)

  See the `pg_dump` manual page for details.

- Change libpq `PQdsplen()` to return a useful value (Martijn van Oosterhout)

- Declare libpq `PQgetssl()` as returning `void *`, rather than `SSL *` (Martijn van Oosterhout)

  This allows applications to use the function without including the OpenSSL headers.

- C-language loadable modules must now include a `PG_MODULE_MAGIC` macro call for version compatibility checking (Martijn van Oosterhout)

- For security's sake, modules used by a PL/PerlU function are no longer available to PL/Perl functions (Andrew)

> **注意**
>
> This also implies that data can no longer be shared between a PL/Perl function and a PL/PerlU function. Some Perl installations have not been compiled with the correct flags to allow multiple interpreters to exist within a single process. In this situation PL/Perl and PL/PerlU cannot both be used in a single backend. The solution is to get a Perl installation which supports multiple interpreters.

- In `contrib/xml2/`, rename `xml_valid()` to `xml_is_well_formed()` (Tom)

  `xml_valid()` will remain for backward compatibility, but its behavior will change to do schema checking in a future release.

- Remove `contrib/ora2pg/`, now at http://www.samse.fr/GPL/ora2pg

- Remove contrib modules that have been migrated to PgFoundry: `adddepend`, `dbase`, `dbmirror`, `fulltextindex`, `mac`, `userlock`

- Remove abandoned contrib modules: `mSQL-interface`, `tips`

- Remove QNX and BEOS ports (Bruce)

  These ports no longer had active maintainers.

E.145.3. Changes

Below you will find a detailed account of the changes between PostgreSQL 8.2 and the previous major release.

E.145.3.1. Performance Improvements

- Allow the planner to reorder outer joins in some circumstances (Tom)

  In previous releases, outer joins would always be evaluated in the order written in the query. This change allows the query optimizer to consider reordering outer joins, in cases where it can determine that the join order can be changed without altering the meaning of the query. This can make a considerable performance difference for queries involving multiple outer joins or mixed inner and outer joins.

- Improve efficiency of `IN` (list-of-expressions) clauses (Tom)

- Improve sorting speed and reduce memory usage (Simon, Tom)

- Improve subtransaction performance (Alvaro, Itagaki Takahiro, Tom)

- Add `FILLFACTOR` to table and index creation (ITAGAKI Takahiro)

  This leaves extra free space in each table or index page, allowing improved performance as the database grows. This is particularly valuable to maintain clustering.

- Increase default values for `shared_buffers` and `max_fsm_pages` (Andrew)

- Improve locking performance by breaking the lock manager tables into sections (Tom)

  This allows locking to be more fine-grained, reducing contention.

- Reduce locking requirements of sequential scans (Qingqing Zhou)

- Reduce locking required for database creation and destruction (Tom)

- Improve the optimizer's selectivity estimates for `LIKE`, `ILIKE`, and regular expression operations (Tom)

- Improve planning of joins to inherited tables and `UNION ALL` views (Tom)

- Allow constraint exclusion to be applied to inherited `UPDATE` and `DELETE` queries (Tom)

SELECT already honored constraint exclusion.

- Improve planning of constant WHERE clauses, such as a condition that depends only on variables inherited from an outer query level (Tom)
- Protocol-level unnamed prepared statements are re-planned for each set of BIND values (Tom)

  This improves performance because the exact parameter values can be used in the plan.
- Speed up vacuuming of B-Tree indexes (Heikki Linnakangas, Tom)
- Avoid extra scan of tables without indexes during VACUUM (Greg Stark)
- Improve multicolumn GiST indexing (Oleg, Teodor)
- Remove dead index entries before B-Tree page split (Junji Teramoto)

E.145.3.2. Server Changes

- Allow a forced switch to a new transaction log file (Simon, Tom)

  This is valuable for keeping warm standby slave servers in sync with the master. Transaction log file switching now also happens automatically during pg_stop_backup(). This ensures that all transaction log files needed for recovery can be archived immediately.
- Add WAL informational functions (Simon)

  Add functions for interrogating the current transaction log insertion point and determining WAL filenames from the hex WAL locations displayed by pg_stop_backup() and related functions.
- Improve recovery from a crash during WAL replay (Simon)

  The server now does periodic checkpoints during WAL recovery, so if there is a crash, future WAL recovery is shortened. This also eliminates the need for warm standby servers to replay the entire log since the base backup if they crash.
- Improve reliability of long-term WAL replay (Heikki, Simon, Tom)

  Formerly, trying to roll forward through more than 2 billion transactions would not work due to XID wraparound. This meant warm standby servers had to be reloaded from fresh base backups periodically.
- Add archive_timeout to force transaction log file switches at a given interval (Simon)

  This enforces a maximum replication delay for warm standby servers.
- Add native LDAP authentication (Magnus Hagander)

  This is particularly useful for platforms that do not support PAM, such as Windows.
- Add GRANT CONNECT ON DATABASE (Gevik Babakhani)

  This gives SQL-level control over database access. It works as an additional filter on top of the existing pg_hba.conf controls.
- Add support for SSL Certificate Revocation List (CRL) files (Libor Hohoš)

The server and libpq both recognize CRL files now.

- GiST indexes are now clusterable (Teodor)
- Remove routine autovacuum server log entries (Bruce)

  `pg_stat_activity` now shows autovacuum activity.
- Track maximum XID age within individual tables, instead of whole databases (Alvaro)

  This reduces the overhead involved in preventing transaction ID wraparound, by avoiding unnecessary VACUUMs.
- Add last vacuum and analyze timestamp columns to the stats collector (Larry Rosenman)

  These values now appear in the `pg_stat_*_tables` system views.
- Improve performance of statistics monitoring, especially `stats_command_string` (Tom, Bruce)

  This release enables `stats_command_string` by default, now that its overhead is minimal. This means `pg_stat_activity` will now show all active queries by default.
- Add a `waiting` column to `pg_stat_activity` (Tom)

  This allows `pg_stat_activity` to show all the information included in the ps display.
- Add configuration parameter `update_process_title` to control whether the ps display is updated for every command (Bruce)

  On platforms where it is expensive to update the ps display, it might be worthwhile to turn this off and rely solely on `pg_stat_activity` for status information.
- Allow units to be specified in configuration settings (Peter)

  For example, you can now set `shared_buffers` to `32MB` rather than mentally converting sizes.
- Add support for include directives in `postgresql.conf` (Joachim Wieland)
- Improve logging of protocol-level prepare/bind/execute messages (Bruce, Tom)

  Such logging now shows statement names, bind parameter values, and the text of the query being executed. Also, the query text is properly included in logged error messages when enabled by `log_min_error_statement`.
- Prevent `max_stack_depth` from being set to unsafe values

  On platforms where we can determine the actual kernel stack depth limit (which is most), make sure that the initial default value of `max_stack_depth` is safe, and reject attempts to set it to unsafely large values.
- Enable highlighting of error location in query in more cases (Tom)

  The server is now able to report a specific error location for some semantic errors (such as unrecognized column name), rather than just for basic syntax errors as before.

- Fix "failed to re-find parent key" errors in `VACUUM` (Tom)
- Clean out `pg_internal.init` cache files during server restart (Simon)

  This avoids a hazard that the cache files might contain stale data after PITR recovery.
- Fix race condition for truncation of a large relation across a gigabyte boundary by `VACUUM` (Tom)
- Fix bug causing needless deadlock errors on row-level locks (Tom)
- Fix bugs affecting multi-gigabyte hash indexes (Tom)
- Each backend process is now its own process group leader (Tom)

  This allows query cancel to abort subprocesses invoked from a backend or archive/recovery process.

E.145.3.3. Query Changes

- Add `INSERT`/`UPDATE`/`DELETE RETURNING` (Jonah Harris, Tom)

  This allows these commands to return values, such as the computed serial key for a new row. In the `UPDATE` case, values from the updated version of the row are returned.
- Add support for multiple-row `VALUES` clauses, per SQL standard (Joe, Tom)

  This allows `INSERT` to insert multiple rows of constants, or queries to generate result sets using constants. For example, `INSERT ... VALUES (...), (...), ....`, and `SELECT * FROM (VALUES (...), (...), ....) AS alias(f1, ...)`.
- Allow `UPDATE` and `DELETE` to use an alias for the target table (Atsushi Ogawa)

  The SQL standard does not permit an alias in these commands, but many database systems allow one anyway for notational convenience.
- Allow `UPDATE` to set multiple columns with a list of values (Susanne Ebrecht)

  This is basically a short-hand for assigning the columns and values in pairs. The syntax is `UPDATE tab SET (column, ...) = (val, ...)`.
- Make row comparisons work per standard (Tom)

  The forms <, <=, >, >= now compare rows lexicographically, that is, compare the first elements, if equal compare the second elements, and so on. Formerly they expanded to an AND condition across all the elements, which was neither standard nor very useful.
- Add `CASCADE` option to `TRUNCATE` (Joachim Wieland)

  This causes `TRUNCATE` to automatically include all tables that reference the specified table(s) via foreign keys. While convenient, this is a dangerous tool – use with caution!
- Support `FOR UPDATE` and `FOR SHARE` in the same `SELECT` command (Tom)
- Add `IS NOT DISTINCT FROM` (Pavel Stehule)

  This operator is similar to equality (=), but evaluates to true when both left and right operands are `NULL`, and to false when just one is, rather than yielding `NULL` in these cases.

- Improve the length output used by `UNION/INTERSECT/EXCEPT` (Tom)

  When all corresponding columns are of the same defined length, that length is used for the result, rather than a generic length.

- Allow `ILIKE` to work for multi-byte encodings (Tom)

  Internally, `ILIKE` now calls `lower()` and then uses `LIKE`. Locale-specific regular expression patterns still do not work in these encodings.

- Enable `standard_conforming_strings` to be turned `on` (Kevin Grittner)

  This allows backslash escaping in strings to be disabled, making PostgreSQL more standards-compliant. The default is `off` for backwards compatibility, but future releases will default this to `on`.

- Do not flatten subqueries that contain `volatile` functions in their target lists (Jaime Casanova)

  This prevents surprising behavior due to multiple evaluation of a `volatile` function (such as `random()` or `nextval()`). It might cause performance degradation in the presence of functions that are unnecessarily marked as `volatile`.

- Add system views `pg_prepared_statements` and `pg_cursors` to show prepared statements and open cursors (Joachim Wieland, Neil)

  These are very useful in pooled connection setups.

- Support portal parameters in `EXPLAIN` and `EXECUTE` (Tom)

  This allows, for example, JDBC `?` parameters to work in these commands.

- If SQL-level `PREPARE` parameters are unspecified, infer their types from the content of the query (Neil)

  Protocol-level `PREPARE` already did this.

- Allow `LIMIT` and `OFFSET` to exceed two billion (Dhanaraj M)

E.145.3.4. Object Manipulation Changes

- Add `TABLESPACE` clause to `CREATE TABLE AS` (Neil)

  This allows a tablespace to be specified for the new table.

- Add `ON COMMIT` clause to `CREATE TABLE AS` (Neil)

  This allows temporary tables to be truncated or dropped on transaction commit. The default behavior is for the table to remain until the session ends.

- Add `INCLUDING CONSTRAINTS` to `CREATE TABLE LIKE` (Greg Stark)

  This allows easy copying of `CHECK` constraints to a new table.

- Allow the creation of placeholder (shell) types (Martijn van Oosterhout)

  A shell type declaration creates a type name, without specifying any of the details of the type. Making a shell type is useful because it allows cleaner declaration of the type's input/output functions, which must exist before the type can be defined “for real” . The syntax is `CREATE TYPE` *`typename`*.

- Aggregate functions now support multiple input parameters (Sergey Koposov, Tom)

- Add new aggregate creation syntax (Tom)

  The new syntax is `CREATE AGGREGATE aggname (input_type) (parameter_list)`. This more naturally supports the new multi-parameter aggregate functionality. The previous syntax is still supported.

- Add `ALTER ROLE PASSWORD NULL` to remove a previously set role password (Peter)

- Add `DROP` object `IF EXISTS` for many object types (Andrew)

  This allows `DROP` operations on non-existent objects without generating an error.

- Add `DROP OWNED` to drop all objects owned by a role (Alvaro)

- Add `REASSIGN OWNED` to reassign ownership of all objects owned by a role (Alvaro)

  This, and `DROP OWNED` above, facilitate dropping roles.

- Add `GRANT ON SEQUENCE` syntax (Bruce)

  This was added for setting sequence-specific permissions. `GRANT ON TABLE` for sequences is still supported for backward compatibility.

- Add `USAGE` permission for sequences that allows only `currval()` and `nextval()`, not `setval()` (Bruce)

  `USAGE` permission allows more fine-grained control over sequence access. Granting `USAGE` allows users to increment a sequence, but prevents them from setting the sequence to an arbitrary value using `setval()`.

- Add `ALTER TABLE [ NO ] INHERIT` (Greg Stark)

  This allows inheritance to be adjusted dynamically, rather than just at table creation and destruction. This is very valuable when using inheritance to implement table partitioning.

- Allow comments on global objects to be stored globally (Kris Jurka)

  Previously, comments attached to databases were stored in individual databases, making them ineffective, and there was no provision at all for comments on roles or tablespaces. This change adds a new shared catalog `pg_shdescription` and stores comments on databases, roles, and tablespaces therein.

E.145.3.5. Utility Command Changes

- Add option to allow indexes to be created without blocking concurrent writes to the table (Greg Stark, Tom)

  The new syntax is `CREATE INDEX CONCURRENTLY`. The default behavior is still to block table modification while an index is being created.

- Provide advisory locking functionality (Abhijit Menon-Sen, Tom)

  This is a new locking API designed to replace what used to be in /contrib/userlock. The userlock code is now on pgfoundry.

- Allow `COPY` to dump a `SELECT` query (Zoltan Boszormenyi, Karel Zak)

  This allows `COPY` to dump arbitrary SQL queries. The syntax is `COPY (SELECT ...) TO`.

- Make the `COPY` command return a command tag that includes the number of rows copied (Volkan YAZICI)
- Allow `VACUUM` to expire rows without being affected by other concurrent `VACUUM` operations (Hannu Krossing, Alvaro, Tom)
- Make initdb detect the operating system locale and set the default `DateStyle` accordingly (Peter)

  This makes it more likely that the installed `postgresql.conf` `DateStyle` value will be as desired.
- Reduce number of progress messages displayed by initdb (Tom)

E.145.3.6. Date/Time Changes

- Allow full timezone names in `timestamp` input values (Joachim Wieland)

  For example, `'2006-05-24 21:11 America/New_York'::timestamptz`.
- Support configurable timezone abbreviations (Joachim Wieland)

  A desired set of timezone abbreviations can be chosen via the configuration parameter `timezone_abbreviations`.
- Add `pg_timezone_abbrevs` and `pg_timezone_names` views to show supported timezones (Magnus Hagander)
- Add `clock_timestamp()`, `statement_timestamp()`, and `transaction_timestamp()` (Bruce)

  `clock_timestamp()` is the current wall-clock time, `statement_timestamp()` is the time the current statement arrived at the server, and `transaction_timestamp()` is an alias for `now()`.
- Allow `to_char()` to print localized month and day names (Euler Taveira de Oliveira)
- Allow `to_char(time)` and `to_char(interval)` to output AM/PM specifications (Bruce)

  Intervals and times are treated as 24-hour periods, e.g. `25 hours` is considered AM.
- Add new function `justify_interval()` to adjust interval units (Mark Dilger)
- Allow timezone offsets up to 14:59 away from GMT

  Kiribati uses GMT+14, so we'd better accept that.
- Interval computation improvements (Michael Glaesemann, Bruce)

E.145.3.7. Other Data Type and Function Changes

- Allow arrays to contain `NULL` elements (Tom)
- Allow assignment to array elements not contiguous with the existing entries (Tom)

  The intervening array positions will be filled with nulls. This is per SQL standard.
- New built-in operators for array-subset comparisons (`@>`, `<@`, `&&`) (Teodor, Tom)

  These operators can be indexed for many data types using GiST or GIN indexes.

- Add convenient arithmetic operations on `INET`/`CIDR` values (Stephen R. van den Berg)

  The new operators are `&` (and), `|` (or), `~` (not), `inet + int8`, `inet - int8`, and `inet - inet`.

- Add new aggregate functions from SQL:2003 (Neil)

  The new functions are `var_pop()`, `var_samp()`, `stddev_pop()`, and `stddev_samp()`. `var_samp()` and `stddev_samp()` are merely renamings of the existing aggregates `variance()` and `stddev()`. The latter names remain available for backward compatibility.

- Add SQL:2003 statistical aggregates (Sergey Koposov)

  New functions: `regr_intercept()`, `regr_slope()`, `regr_r2()`, `corr()`, `covar_samp()`, `covar_pop()`, `regr_avgx()`, `regr_avgy()`, `regr_sxy()`, `regr_sxx()`, `regr_syy()`, `regr_count()`.

- Allow domains to be based on other domains (Tom)

- Properly enforce domain `CHECK` constraints everywhere (Neil, Tom)

  For example, the result of a user-defined function that is declared to return a domain type is now checked against the domain's constraints. This closes a significant hole in the domain implementation.

- Fix problems with dumping renamed `SERIAL` columns (Tom)

  The fix is to dump a `SERIAL` column by explicitly specifying its `DEFAULT` and sequence elements, and reconstructing the `SERIAL` column on reload using a new `ALTER SEQUENCE OWNED BY` command. This also allows dropping a `SERIAL` column specification.

- Add a server-side sleep function `pg_sleep()` (Joachim Wieland)

- Add all comparison operators for the `tid` (tuple id) data type (Mark Kirkwood, Greg Stark, Tom)

E.145.3.8. PL/pgSQL Server-Side Language Changes

- Add `TG_table_name` and `TG_table_schema` to trigger parameters (Andrew)

  `TG_relname` is now deprecated. Comparable changes have been made in the trigger parameters for the other PLs as well.

- Allow `FOR` statements to return values to scalars as well as records and row types (Pavel Stehule)

- Add a `BY` clause to the `FOR` loop, to control the iteration increment (Jaime Casanova)

- Add `STRICT` to `SELECT INTO` (Matt Miller)

  `STRICT` mode throws an exception if more or less than one row is returned by the `SELECT`, for Oracle PL/SQL compatibility.

E.145.3.9. PL/Perl Server-Side Language Changes

- Add `table_name` and `table_schema` to trigger parameters (Adam Sjøgren)

- Add prepared queries (Dmitry Karasik)

- Make `$_TD` trigger data a global variable (Andrew)

  Previously, it was lexical, which caused unexpected sharing violations.

- Run PL/Perl and PL/PerlU in separate interpreters, for security reasons (Andrew)

  In consequence, they can no longer share data nor loaded modules. Also, if Perl has not been compiled with the requisite flags to allow multiple interpreters, only one of these languages can be used in any given backend process.

E.145.3.10. PL/Python Server-Side Language Changes

- Named parameters are passed as ordinary variables, as well as in the `args[]` array (Sven Suursoho)
- Add `table_name` and `table_schema` to trigger parameters (Andrew)
- Allow returning of composite types and result sets (Sven Suursoho)
- Return result-set as `list`, `iterator`, or `generator` (Sven Suursoho)
- Allow functions to return `void` (Neil)
- Python 2.5 is now supported (Tom)

E.145.3.11. psql Changes

- Add new command `\password` for changing role password with client-side password encryption (Peter)
- Allow `\c` to connect to a new host and port number (David, Volkan YAZICI)
- Add tablespace display to `\l+` (Philip Yarra)
- Improve `\df` slash command to include the argument names and modes (`OUT` or `INOUT`) of the function (David Fetter)
- Support binary `COPY` (Andreas Pflug)
- Add option to run the entire session in a single transaction (Simon)

  Use option `-1` or `--single-transaction`.

- Support for automatically retrieving `SELECT` results in batches using a cursor (Chris Mair)

  This is enabled using `\set FETCH_COUNT` *n*. This feature allows large result sets to be retrieved in psql without attempting to buffer the entire result set in memory.

- Make multi-line values align in the proper column (Martijn van Oosterhout)

  Field values containing newlines are now displayed in a more readable fashion.

- Save multi-line statements as a single entry, rather than one line at a time (Sergey E. Koposov)

  This makes up-arrow recall of queries easier. (This is not available on Windows, because that platform uses the native command-line editing present in the operating system.)

- Make the line counter 64-bit so it can handle files with more than two billion lines (David Fetter)

- Report both the returned data and the command status tag for `INSERT/UPDATE/DELETE RETURNING` (Tom)

E.145.3.12. pg_dump Changes

- Allow complex selection of objects to be included or excluded by pg_dump (Greg Sabino Mullane)

  pg_dump now supports multiple `-n` (schema) and `-t` (table) options, and adds `-N` and `-T` options to exclude objects. Also, the arguments of these switches can now be wild-card expressions rather than single object names, for example `-t 'foo*'`, and a schema can be part of a `-t` or `-T` switch, for example `-t schema1.table1`.

- Add pg_restore `--no-data-for-failed-tables` option to suppress loading data if table creation failed (i.e., the table already exists) (Martin Pitt)

- Add pg_restore option to run the entire session in a single transaction (Simon)

  Use option `-1` or `--single-transaction`.

E.145.3.13. libpq Changes

- Add `PQencryptPassword()` to encrypt passwords (Tom)

  This allows passwords to be sent pre-encrypted for commands like `ALTER ROLE ... PASSWORD`.

- Add function `PQisthreadsafe()` (Bruce)

  This allows applications to query the thread-safety status of the library.

- Add `PQdescribePrepared()`, `PQdescribePortal()`, and related functions to return information about previously prepared statements and open cursors (Volkan YAZICI)

- Allow LDAP lookups from `pg_service.conf` (Laurenz Albe)

- Allow a hostname in `~/.pgpass` to match the default socket directory (Bruce)

  A blank hostname continues to match any Unix-socket connection, but this addition allows entries that are specific to one of several postmasters on the machine.

E.145.3.14. ecpg Changes

- Allow `SHOW` to put its result into a variable (Joachim Wieland)

- Add `COPY TO STDOUT` (Joachim Wieland)

- Add regression tests (Joachim Wieland, Michael)

- Major source code cleanups (Joachim Wieland, Michael)

E.145.3.15. Windows Port

- Allow MSVC to compile the PostgreSQL server (Magnus, Hiroshi Saito)

- Add MSVC support for utility commands and pg_dump (Hiroshi Saito)

- Add support for Windows code pages `1253`, `1254`, `1255`, and `1257` (Kris Jurka)

- Drop privileges on startup, so that the server can be started from an administrative account (Magnus)
- Stability fixes (Qingqing Zhou, Magnus)
- Add native semaphore implementation (Qingqing Zhou)

  The previous code mimicked SysV semaphores.

E.145.3.16. Source Code Changes

- Add GIN (Generalized Inverted iNdex) index access method (Teodor, Oleg)
- Remove R-tree indexing (Tom)

  Rtree has been re-implemented using GiST. Among other differences, this means that rtree indexes now have support for crash recovery via write-ahead logging (WAL).
- Reduce libraries needlessly linked into the backend (Martijn van Oosterhout, Tom)
- Add a configure flag to allow libedit to be preferred over GNU readline (Bruce)

  Use configure `--with-libedit-preferred`.
- Allow installation into directories containing spaces (Peter)
- Improve ability to relocate installation directories (Tom)
- Add support for Solaris x86_64 using the Solaris compiler (Pierre Girard, Theo Schlossnagle, Bruce)
- Add DTrace support (Robert Lor)
- Add `PG_VERSION_NUM` for use by third-party applications wanting to test the backend version in C using > and < comparisons (Bruce)
- Add `XLOG_BLCKSZ` as independent from `BLCKSZ` (Mark Wong)
- Add `LWLOCK_STATS` define to report locking activity (Tom)
- Emit warnings for unknown configure options (Martijn van Oosterhout)
- Add server support for “plugin” libraries that can be used for add-on tasks such as debugging and performance measurement (Korry Douglas)

  This consists of two features: a table of “rendezvous variables” that allows separately-loaded shared libraries to communicate, and a new configuration parameter `local_preload_libraries` that allows libraries to be loaded into specific sessions without explicit cooperation from the client application. This allows external add-ons to implement features such as a PL/pgSQL debugger.
- Rename existing configuration parameter `preload_libraries` to `shared_preload_libraries` (Tom)

  This was done for clarity in comparison to `local_preload_libraries`.
- Add new configuration parameter `server_version_num` (Greg Sabino Mullane)

  This is like `server_version`, but is an integer, e.g. `80200`. This allows applications to make version checks more easily.

- Add a configuration parameter `seq_page_cost` (Tom)
- Re-implement the regression test script as a C program (Magnus, Tom)
- Allow loadable modules to allocate shared memory and lightweight locks (Marc Munro)
- Add automatic initialization and finalization of dynamically loaded libraries (Ralf Engelschall, Tom)

  New functions `_PG_init()` and `_PG_fini()` are called if the library defines such symbols. Hence we no longer need to specify an initialization function in `shared_preload_libraries`; we can assume that the library used the `_PG_init()` convention instead.
- Add `PG_MODULE_MAGIC` header block to all shared object files (Martijn van Oosterhout)

  The magic block prevents version mismatches between loadable object files and servers.
- Add shared library support for AIX (Laurenz Albe)
- New XML documentation section (Bruce)

E.145.3.17. Contrib Changes

- Major tsearch2 improvements (Oleg, Teodor)
  - multibyte encoding support, including UTF8
  - query rewriting support
  - improved ranking functions
  - thesaurus dictionary support
  - Ispell dictionaries now recognize MySpell format, used by OpenOffice
  - GIN support
- Add adminpack module containing Pgadmin administration functions (Dave)

  These functions provide additional file system access routines not present in the default PostgreSQL server.
- Add sslinfo module (Victor Wagner)

  Reports information about the current connection's SSL certificate.
- Add pgrowlocks module (Tatsuo)

  This shows row locking information for a specified table.
- Add hstore module (Oleg, Teodor)
- Add isn module, replacing isbn_issn (Jeremy Kronuz)

  This new implementation supports EAN13, UPC, ISBN (books), ISMN (music), and ISSN (serials).
- Add index information functions to pgstattuple (ITAGAKI Takahiro, Satoshi Nagayasu)

- Add pg_freespacemap module to display free space map information (Mark Kirkwood)
- pgcrypto now has all planned functionality (Marko Kreen)
  - Include iMath library in pgcrypto to have the public-key encryption functions always available.
  - Add SHA224 algorithm that was missing in OpenBSD code.
  - Activate builtin code for SHA224/256/384/512 hashes on older OpenSSL to have those algorithms always available.
  - New function gen_random_bytes() that returns cryptographically strong randomness. Useful for generating encryption keys.
  - Remove digest_exists(), hmac_exists() and cipher_exists() functions.
- Improvements to cube module (Joshua Reich)

  New functions are `cube(float[])`, `cube(float[], float[])`, and `cube_subset(cube, int4[])`.
- Add async query capability to dblink (Kai Londenberg, Joe Conway)
- New operators for array-subset comparisons (`@>`, `<@`, `&&`) (Tom)

  Various contrib packages already had these operators for their datatypes, but the naming wasn't consistent. We have now added consistently named array-subset comparison operators to the core code and all the contrib packages that have such functionality. (The old names remain available, but are deprecated.)
- Add uninstall scripts for all contrib packages that have install scripts (David, Josh Drake)

E.146. Release 8.1.23

Release date: 2010-12-16

This release contains a variety of fixes from 8.1.22. For information about new features in the 8.1 major release, see 第 E.169 节.

This is expected to be the last PostgreSQL release in the 8.1.X series. Users are encouraged to update to a newer release branch soon.

E.146.1. Migration to Version 8.1.23

A dump/restore is not required for those running 8.1.X. However, if you are upgrading from a version earlier than 8.1.18, see 第 E.151 节.

E.146.2. Changes

- Force the default `wal_sync_method` to be `fdatasync` on Linux (Tom Lane, Marti Raudsepp)

  The default on Linux has actually been `fdatasync` for many years, but recent kernel changes caused PostgreSQL to choose `open_datasync` instead. This choice did not result in any performance improvement, and caused outright failures on certain filesystems, notably `ext4` with the `data=journal` mount option.

- Fix recovery from base backup when the starting checkpoint WAL record is not in the same WAL segment as its redo point (Jeff Davis)
- Add support for detecting register-stack overrun on `IA64` (Tom Lane)

  The `IA64` architecture has two hardware stacks. Full prevention of stack-overrun failures requires checking both.
- Add a check for stack overflow in `copyObject()` (Tom Lane)

  Certain code paths could crash due to stack overflow given a sufficiently complex query.
- Fix detection of page splits in temporary GiST indexes (Heikki Linnakangas)

  It is possible to have a "concurrent" page split in a temporary index, if for example there is an open cursor scanning the index when an insertion is done. GiST failed to detect this case and hence could deliver wrong results when execution of the cursor continued.
- Avoid memory leakage while `ANALYZE`'ing complex index expressions (Tom Lane)
- Ensure an index that uses a whole-row Var still depends on its table (Tom Lane)

  An index declared like `create index i on t (foo(t.*))` would not automatically get dropped when its table was dropped.
- Do not "inline" a SQL function with multiple `OUT` parameters (Tom Lane)

  This avoids a possible crash due to loss of information about the expected result rowtype.
- Fix constant-folding of `COALESCE()` expressions (Tom Lane)

  The planner would sometimes attempt to evaluate sub-expressions that in fact could never be reached, possibly leading to unexpected errors.
- Add print functionality for `InhRelation` nodes (Tom Lane)

  This avoids a failure when `debug_print_parse` is enabled and certain types of query are executed.
- Fix incorrect calculation of distance from a point to a horizontal line segment (Tom Lane)

  This bug affected several different geometric distance-measurement operators.
- Fix PL/pgSQL's handling of "simple" expressions to not fail in recursion or error-recovery cases (Tom Lane)
- Fix bug in `contrib/cube`'s GiST picksplit algorithm (Alexander Korotkov)

  This could result in considerable inefficiency, though not actually incorrect answers, in a GiST index on a `cube` column. If you have such an index, consider `REINDEX`ing it after installing this update.
- Don't emit "identifier will be truncated" notices in `contrib/dblink` except when creating new connections (Itagaki Takahiro)
- Fix potential coredump on missing public key in `contrib/pgcrypto` (Marti Raudsepp)
- Fix memory leak in `contrib/xml2`'s XPath query functions (Tom Lane)

- Update time zone data files to tzdata release 2010o for DST law changes in Fiji and Samoa; also historical corrections for Hong Kong.

E.147. Release 8.1.22

Release date: 2010-10-04

This release contains a variety of fixes from 8.1.21. For information about new features in the 8.1 major release, see 第 E.169 节.

The PostgreSQL community will stop releasing updates for the 8.1.X release series in November 2010. Users are encouraged to update to a newer release branch soon.

E.147.1. Migration to Version 8.1.22

A dump/restore is not required for those running 8.1.X. However, if you are upgrading from a version earlier than 8.1.18, see 第 E.151 节.

E.147.2. Changes

- Use a separate interpreter for each calling SQL userid in PL/Perl and PL/Tcl (Tom Lane)

  This change prevents security problems that can be caused by subverting Perl or Tcl code that will be executed later in the same session under another SQL user identity (for example, within a `SECURITY DEFINER` function). Most scripting languages offer numerous ways that that might be done, such as redefining standard functions or operators called by the target function. Without this change, any SQL user with Perl or Tcl language usage rights can do essentially anything with the SQL privileges of the target function's owner.

  The cost of this change is that intentional communication among Perl and Tcl functions becomes more difficult. To provide an escape hatch, PL/PerlU and PL/TclU functions continue to use only one interpreter per session. This is not considered a security issue since all such functions execute at the trust level of a database superuser already.

  It is likely that third-party procedural languages that claim to offer trusted execution have similar security issues. We advise contacting the authors of any PL you are depending on for security-critical purposes.

  Our thanks to Tim Bunce for pointing out this issue (CVE-2010-3433).

- Prevent possible crashes in `pg_get_expr()` by disallowing it from being called with an argument that is not one of the system catalog columns it's intended to be used with (Heikki Linnakangas, Tom Lane)

- Fix “cannot handle unplanned sub-select” error (Tom Lane)

  This occurred when a sub-select contains a join alias reference that expands into an expression containing another sub-select.

- Prevent show_session_authorization() from crashing within autovacuum processes (Tom Lane)

- Defend against functions returning setof record where not all the returned rows are actually of the same rowtype (Tom Lane)

- Fix possible failure when hashing a pass-by-reference function result (Tao Ma, Tom Lane)

- Take care to fsync the contents of lockfiles (both `postmaster.pid` and the socket lockfile) while writing them (Tom Lane)

  This omission could result in corrupted lockfile contents if the machine crashes shortly after postmaster start. That could in turn prevent subsequent attempts to start the postmaster from succeeding, until the lockfile is manually removed.

- Avoid recursion while assigning XIDs to heavily-nested subtransactions (Andres Freund, Robert Haas)

  The original coding could result in a crash if there was limited stack space.

- Fix `log_line_prefix`'s `%i` escape, which could produce junk early in backend startup (Tom Lane)

- Fix possible data corruption in `ALTER TABLE ... SET TABLESPACE` when archiving is enabled (Jeff Davis)

- Allow `CREATE DATABASE` and `ALTER DATABASE ... SET TABLESPACE` to be interrupted by query-cancel (Guillaume Lelarge)

- In PL/Python, defend against null pointer results from `PyCObject_AsVoidPtr` and `PyCObject_FromVoidPtr` (Peter Eisentraut)

- Improve `contrib/dblink`'s handling of tables containing dropped columns (Tom Lane)

- Fix connection leak after "duplicate connection name" errors in `contrib/dblink` (Itagaki Takahiro)

- Fix `contrib/dblink` to handle connection names longer than 62 bytes correctly (Itagaki Takahiro)

- Update build infrastructure and documentation to reflect the source code repository's move from CVS to Git (Magnus Hagander and others)

- Update time zone data files to tzdata release 2010l for DST law changes in Egypt and Palestine; also historical corrections for Finland.

  This change also adds new names for two Micronesian timezones: Pacific/Chuuk is now preferred over Pacific/Truk (and the preferred abbreviation is CHUT not TRUT) and Pacific/Pohnpei is preferred over Pacific/Ponape.

E.148. Release 8.1.21

Release date: 2010-05-17

This release contains a variety of fixes from 8.1.20. For information about new features in the 8.1 major release, see 第 E.169 节.

E.148.1. Migration to Version 8.1.21

A dump/restore is not required for those running 8.1.X. However, if you are upgrading from a version earlier than 8.1.18, see 第 E.151 节.

E.148.2. Changes

- Enforce restrictions in `plperl` using an opmask applied to the whole interpreter, instead of using `Safe.pm` (Tim Bunce, Andrew Dunstan)

Recent developments have convinced us that `Safe.pm` is too insecure to rely on for making `plperl` trustable. This change removes use of `Safe.pm` altogether, in favor of using a separate interpreter with an opcode mask that is always applied. Pleasant side effects of the change include that it is now possible to use Perl's `strict` pragma in a natural way in `plperl`, and that Perl's `$a` and `$b` variables work as expected in sort routines, and that function compilation is significantly faster. (CVE-2010-1169)

- Prevent PL/Tcl from executing untrustworthy code from `pltcl_modules` (Tom)

  PL/Tcl's feature for autoloading Tcl code from a database table could be exploited for trojan-horse attacks, because there was no restriction on who could create or insert into that table. This change disables the feature unless `pltcl_modules` is owned by a superuser. (However, the permissions on the table are not checked, so installations that really need a less-than-secure modules table can still grant suitable privileges to trusted non-superusers.) Also, prevent loading code into the unrestricted "normal" Tcl interpreter unless we are really going to execute a `pltclu` function. (CVE-2010-1170)

- Do not allow an unprivileged user to reset superuser-only parameter settings (Alvaro)

  Previously, if an unprivileged user ran `ALTER USER ... RESET ALL` for himself, or `ALTER DATABASE ... RESET ALL` for a database he owns, this would remove all special parameter settings for the user or database, even ones that are only supposed to be changeable by a superuser. Now, the `ALTER` will only remove the parameters that the user has permission to change.

- Avoid possible crash during backend shutdown if shutdown occurs when a `CONTEXT` addition would be made to log entries (Tom)

  In some cases the context-printing function would fail because the current transaction had already been rolled back when it came time to print a log message.

- Update pl/perl's `ppport.h` for modern Perl versions (Andrew)

- Fix assorted memory leaks in pl/python (Andreas Freund, Tom)

- Prevent infinite recursion in psql when expanding a variable that refers to itself (Tom)

- Ensure that `contrib/pgstattuple` functions respond to cancel interrupts promptly (Tatsuhito Kasahara)

- Make server startup deal properly with the case that `shmget()` returns `EINVAL` for an existing shared memory segment (Tom)

  This behavior has been observed on BSD-derived kernels including OS X. It resulted in an entirely-misleading startup failure complaining that the shared memory request size was too large.

- Update time zone data files to tzdata release 2010j for DST law changes in Argentina, Australian Antarctic, Bangladesh, Mexico, Morocco, Pakistan, Palestine, Russia, Syria, Tunisia; also historical corrections for Taiwan.

E.149. Release 8.1.20

Release date: 2010-03-15

This release contains a variety of fixes from 8.1.19. For information about new features in the 8.1 major release, see 第 E.169 节.

E.149.1. Migration to Version 8.1.20

A dump/restore is not required for those running 8.1.X. However, if you are upgrading from a version earlier than 8.1.18, see 第 E.151 节.

E.149.2. Changes

- Add new configuration parameter `ssl_renegotiation_limit` to control how often we do session key renegotiation for an SSL connection (Magnus)

  This can be set to zero to disable renegotiation completely, which may be required if a broken SSL library is used. In particular, some vendors are shipping stopgap patches for CVE-2009-3555 that cause renegotiation attempts to fail.

- Fix possible crashes when trying to recover from a failure in subtransaction start (Tom)

- Fix server memory leak associated with use of savepoints and a client encoding different from server's encoding (Tom)

- Make `substring()` for `bit` types treat any negative length as meaning "all the rest of the string" (Tom)

  The previous coding treated only -1 that way, and would produce an invalid result value for other negative values, possibly leading to a crash (CVE-2010-0442).

- Fix integer-to-bit-string conversions to handle the first fractional byte correctly when the output bit width is wider than the given integer by something other than a multiple of 8 bits (Tom)

- Fix some cases of pathologically slow regular expression matching (Tom)

- Fix the `STOP WAL LOCATION` entry in backup history files to report the next WAL segment's name when the end location is exactly at a segment boundary (Itagaki Takahiro)

- Fix some more cases of temporary-file leakage (Heikki)

  This corrects a problem introduced in the previous minor release. One case that failed is when a plpgsql function returning set is called within another function's exception handler.

- When reading `pg_hba.conf` and related files, do not treat `@something` as a file inclusion request if the `@` appears inside quote marks; also, never treat `@` by itself as a file inclusion request (Tom)

  This prevents erratic behavior if a role or database name starts with `@`. If you need to include a file whose path name contains spaces, you can still do so, but you must write `@"/path to/file"` rather than putting the quotes around the whole construct.

- Prevent infinite loop on some platforms if a directory is named as an inclusion target in `pg_hba.conf` and related files (Tom)

- Fix psql's `numericlocale` option to not format strings it shouldn't in latex and troff output formats (Heikki)

- Fix plpgsql failure in one case where a composite column is set to NULL (Tom)

- Add `volatile` markings in PL/Python to avoid possible compiler-specific misbehavior (Zdenek Kotala)
- Ensure PL/Tcl initializes the Tcl interpreter fully (Tom)

  The only known symptom of this oversight is that the Tcl `clock` command misbehaves if using Tcl 8.5 or later.
- Prevent crash in `contrib/dblink` when too many key columns are specified to a `dblink_build_sql_*` function (Rushabh Lathia, Joe Conway)
- Fix assorted crashes in `contrib/xml2` caused by sloppy memory management (Tom)
- Update time zone data files to tzdata release 2010e for DST law changes in Bangladesh, Chile, Fiji, Mexico, Paraguay, Samoa.

E.150. Release 8.1.19

Release date: 2009-12-14

This release contains a variety of fixes from 8.1.18. For information about new features in the 8.1 major release, see 第 E.169 节.

E.150.1. Migration to Version 8.1.19

A dump/restore is not required for those running 8.1.X. However, if you are upgrading from a version earlier than 8.1.18, see 第 E.151 节.

E.150.2. Changes

- Protect against indirect security threats caused by index functions changing session-local state (Gurjeet Singh, Tom)

  This change prevents allegedly-immutable index functions from possibly subverting a superuser's session (CVE-2009-4136).
- Reject SSL certificates containing an embedded null byte in the common name (CN) field (Magnus)

  This prevents unintended matching of a certificate to a server or client name during SSL validation (CVE-2009-4034).
- Fix possible crash during backend-startup-time cache initialization (Tom)
- Prevent signals from interrupting `VACUUM` at unsafe times (Alvaro)

  This fix prevents a PANIC if a `VACUUM FULL` is canceled after it's already committed its tuple movements, as well as transient errors if a plain `VACUUM` is interrupted after having truncated the table.
- Fix possible crash due to integer overflow in hash table size calculation (Tom)

  This could occur with extremely large planner estimates for the size of a hashjoin's result.
- Fix very rare crash in `inet/cidr` comparisons (Chris Mikkelson)
- Ensure that shared tuple-level locks held by prepared transactions are not ignored (Heikki)

- Fix premature drop of temporary files used for a cursor that is accessed within a subtransaction (Heikki)
- Fix PAM password processing to be more robust (Tom)

  The previous code is known to fail with the combination of the Linux `pam_krb5` PAM module with Microsoft Active Directory as the domain controller. It might have problems elsewhere too, since it was making unjustified assumptions about what arguments the PAM stack would pass to it.
- Fix processing of ownership dependencies during `CREATE OR REPLACE FUNCTION` (Tom)
- Ensure that Perl arrays are properly converted to PostgreSQL arrays when returned by a set-returning PL/Perl function (Andrew Dunstan, Abhijit Menon-Sen)

  This worked correctly already for non-set-returning functions.
- Fix rare crash in exception processing in PL/Python (Peter)
- Ensure psql's flex module is compiled with the correct system header definitions (Tom)

  This fixes build failures on platforms where `--enable-largefile` causes incompatible changes in the generated code.
- Make the postmaster ignore any `application_name` parameter in connection request packets, to improve compatibility with future libpq versions (Tom)
- Update time zone data files to tzdata release 2009s for DST law changes in Antarctica, Argentina, Bangladesh, Fiji, Novokuznetsk, Pakistan, Palestine, Samoa, Syria; also historical corrections for Hong Kong.

E.151. Release 8.1.18

Release date: 2009-09-09

This release contains a variety of fixes from 8.1.17. For information about new features in the 8.1 major release, see 第 E.169 节.

E.151.1. Migration to Version 8.1.18

A dump/restore is not required for those running 8.1.X. However, if you have any hash indexes on `interval` columns, you must `REINDEX` them after updating to 8.1.18. Also, if you are upgrading from a version earlier than 8.1.15, see 第 E.154 节.

E.151.2. Changes

- Disallow `RESET ROLE` and `RESET SESSION AUTHORIZATION` inside security-definer functions (Tom, Heikki)

  This covers a case that was missed in the previous patch that disallowed `SET ROLE` and `SET SESSION AUTHORIZATION` inside security-definer functions. (See CVE-2007-6600)
- Fix handling of sub-SELECTs appearing in the arguments of an outer-level aggregate function (Tom)

- Fix hash calculation for data type `interval` (Tom)

  This corrects wrong results for hash joins on interval values. It also changes the contents of hash indexes on interval columns. If you have any such indexes, you must `REINDEX` them after updating.

- Treat `to_char(..., 'TH')` as an uppercase ordinal suffix with `'HH'/'HH12'` (Heikki)

  It was previously handled as `'th'` (lowercase).

- Fix overflow for `INTERVAL 'x ms'` when *x* is more than 2 million and integer datetimes are in use (Alex Hunsaker)

- Fix calculation of distance between a point and a line segment (Tom)

  This led to incorrect results from a number of geometric operators.

- Fix `money` data type to work in locales where currency amounts have no fractional digits, e.g. Japan (Itagaki Takahiro)

- Properly round datetime input like `00:12:57.999999999999999999999999999` (Tom)

- Fix poor choice of page split point in GiST R-tree operator classes (Teodor)

- Fix portability issues in plperl initialization (Andrew Dunstan)

- Fix pg_ctl to not go into an infinite loop if `postgresql.conf` is empty (Jeff Davis)

- Fix `contrib/xml2`'s `xslt_process()` to properly handle the maximum number of parameters (twenty) (Tom)

- Improve robustness of libpq's code to recover from errors during `COPY FROM STDIN` (Tom)

- Avoid including conflicting readline and editline header files when both libraries are installed (Zdenek Kotala)

- Update time zone data files to tzdata release 2009l for DST law changes in Bangladesh, Egypt, Jordan, Pakistan, Argentina/San_Luis, Cuba, Jordan (historical correction only), Mauritius, Morocco, Palestine, Syria, Tunisia.

E.152. Release 8.1.17

Release date: 2009-03-16

This release contains a variety of fixes from 8.1.16. For information about new features in the 8.1 major release, see 第 E.169 节.

E.152.1. Migration to Version 8.1.17

A dump/restore is not required for those running 8.1.X. However, if you are upgrading from a version earlier than 8.1.15, see 第 E.154 节.

E.152.2. Changes

- Prevent error recursion crashes when encoding conversion fails (Tom)

This change extends fixes made in the last two minor releases for related failure scenarios. The previous fixes were narrowly tailored for the original problem reports, but we have now recognized that any error thrown by an encoding conversion function could potentially lead to infinite recursion while trying to report the error. The solution therefore is to disable translation and encoding conversion and report the plain-ASCII form of any error message, if we find we have gotten into a recursive error reporting situation. (CVE-2009-0922)

- Disallow `CREATE CONVERSION` with the wrong encodings for the specified conversion function (Heikki)

  This prevents one possible scenario for encoding conversion failure. The previous change is a backstop to guard against other kinds of failures in the same area.

- Fix core dump when `to_char()` is given format codes that are inappropriate for the type of the data argument (Tom)

- Fix decompilation of `CASE WHEN` with an implicit coercion (Tom)

  This mistake could lead to Assert failures in an Assert-enabled build, or an "unexpected CASE WHEN clause" error message in other cases, when trying to examine or dump a view.

- Fix possible misassignment of the owner of a TOAST table's rowtype (Tom)

  If `CLUSTER` or a rewriting variant of `ALTER TABLE` were executed by someone other than the table owner, the `pg_type` entry for the table's TOAST table would end up marked as owned by that someone. This caused no immediate problems, since the permissions on the TOAST rowtype aren't examined by any ordinary database operation. However, it could lead to unexpected failures if one later tried to drop the role that issued the command (in 8.1 or 8.2), or "owner of data type appears to be invalid" warnings from pg_dump after having done so (in 8.3).

- Clean up PL/pgSQL error status variables fully at block exit (Ashesh Vashi and Dave Page)

  This is not a problem for PL/pgSQL itself, but the omission could cause the PL/pgSQL Debugger to crash while examining the state of a function.

- Add `MUST` (Mauritius Island Summer Time) to the default list of known timezone abbreviations (Xavier Bugaud)

E.153. Release 8.1.16

Release date: 2009-02-02

This release contains a variety of fixes from 8.1.15. For information about new features in the 8.1 major release, see 第 E.169 节.

E.153.1. Migration to Version 8.1.16

A dump/restore is not required for those running 8.1.X. However, if you are upgrading from a version earlier than 8.1.15, see 第 E.154 节.

E.153.2. Changes

- Fix crash in autovacuum (Alvaro)

The crash occurs only after vacuuming a whole database for anti-transaction-wraparound purposes, which means that it occurs infrequently and is hard to track down.

- Improve handling of URLs in `headline()` function (Teodor)
- Improve handling of overlength headlines in `headline()` function (Teodor)
- Prevent possible Assert failure or misconversion if an encoding conversion is created with the wrong conversion function for the specified pair of encodings (Tom, Heikki)
- Avoid unnecessary locking of small tables in `VACUUM` (Heikki)
- Ensure that the contents of a holdable cursor don't depend on the contents of TOAST tables (Tom)

  Previously, large field values in a cursor result might be represented as TOAST pointers, which would fail if the referenced table got dropped before the cursor is read, or if the large value is deleted and then vacuumed away. This cannot happen with an ordinary cursor, but it could with a cursor that is held past its creating transaction.
- Fix uninitialized variables in `contrib/tsearch2`'s `get_covers()` function (Teodor)
- Fix configure script to properly report failure when unable to obtain linkage information for PL/Perl (Andrew)
- Make all documentation reference `pgsql-bugs` and/or `pgsql-hackers` as appropriate, instead of the now-decommissioned `pgsql-ports` and `pgsql-patches` mailing lists (Tom)
- Update time zone data files to tzdata release 2009a (for Kathmandu and historical DST corrections in Switzerland, Cuba)

E.154. Release 8.1.15

Release date: 2008-11-03

This release contains a variety of fixes from 8.1.14. For information about new features in the 8.1 major release, see 第 E.169 节.

E.154.1. Migration to Version 8.1.15

A dump/restore is not required for those running 8.1.X. However, if you are upgrading from a version earlier than 8.1.2, see 第 E.167 节. Also, if you were running a previous 8.1.X release, it is recommended to `REINDEX` all GiST indexes after the upgrade.

E.154.2. Changes

- Fix GiST index corruption due to marking the wrong index entry "dead" after a deletion (Teodor)

  This would result in index searches failing to find rows they should have found. Corrupted indexes can be fixed with `REINDEX`.
- Fix backend crash when the client encoding cannot represent a localized error message (Tom)

We have addressed similar issues before, but it would still fail if the "character has no equivalent" message itself couldn't be converted. The fix is to disable localization and send the plain ASCII error message when we detect such a situation.

- Fix possible crash when deeply nested functions are invoked from a trigger (Tom)
- Fix mis-expansion of rule queries when a sub-`SELECT` appears in a function call in `FROM`, a multi-row `VALUES` list, or a `RETURNING` list (Tom)

  The usual symptom of this problem is an "unrecognized node type" error.
- Ensure an error is reported when a newly-defined PL/pgSQL trigger function is invoked as a normal function (Tom)
- Prevent possible collision of `relfilenode` numbers when moving a table to another tablespace with `ALTER SET TABLESPACE` (Heikki)

  The command tried to re-use the existing filename, instead of picking one that is known unused in the destination directory.
- Fix incorrect tsearch2 headline generation when single query item matches first word of text (Sushant Sinha)
- Fix improper display of fractional seconds in interval values when using a non-ISO datestyle in an `--enable-integer-datetimes` build (Ron Mayer)
- Ensure `SPI_getvalue` and `SPI_getbinval` behave correctly when the passed tuple and tuple descriptor have different numbers of columns (Tom)

  This situation is normal when a table has had columns added or removed, but these two functions didn't handle it properly. The only likely consequence is an incorrect error indication.
- Fix ecpg's parsing of `CREATE ROLE` (Michael)
- Fix recent breakage of `pg_ctl restart` (Tom)
- Update time zone data files to tzdata release 2008i (for DST law changes in Argentina, Brazil, Mauritius, Syria)

E.155. Release 8.1.14

Release date: 2008-09-22

This release contains a variety of fixes from 8.1.13. For information about new features in the 8.1 major release, see 第 E.169 节.

E.155.1. Migration to Version 8.1.14

A dump/restore is not required for those running 8.1.X. However, if you are upgrading from a version earlier than 8.1.2, see 第 E.167 节.

E.155.2. Changes

- Widen local lock counters from 32 to 64 bits (Tom)

  This responds to reports that the counters could overflow in sufficiently long transactions, leading to unexpected "lock is already held" errors.

- Fix possible duplicate output of tuples during a GiST index scan (Teodor)
- Add checks in executor startup to ensure that the tuples produced by an `INSERT` or `UPDATE` will match the target table's current rowtype (Tom)

  `ALTER COLUMN TYPE`, followed by re-use of a previously cached plan, could produce this type of situation. The check protects against data corruption and/or crashes that could ensue.
- Fix `AT TIME ZONE` to first try to interpret its timezone argument as a timezone abbreviation, and only try it as a full timezone name if that fails, rather than the other way around as formerly (Tom)

  The timestamp input functions have always resolved ambiguous zone names in this order. Making `AT TIME ZONE` do so as well improves consistency, and fixes a compatibility bug introduced in 8.1: in ambiguous cases we now behave the same as 8.0 and before did, since in the older versions `AT TIME ZONE` accepted only abbreviations.
- Fix datetime input functions to correctly detect integer overflow when running on a 64-bit platform (Tom)
- Improve performance of writing very long log messages to syslog (Tom)
- Fix bug in backwards scanning of a cursor on a `SELECT DISTINCT ON` query (Tom)
- Fix planner bug with nested sub-select expressions (Tom)

  If the outer sub-select has no direct dependency on the parent query, but the inner one does, the outer value might not get recalculated for new parent query rows.
- Fix planner to estimate that `GROUP BY` expressions yielding boolean results always result in two groups, regardless of the expressions' contents (Tom)

  This is very substantially more accurate than the regular `GROUP BY` estimate for certain boolean tests like *`col`* `IS NULL`.
- Fix PL/pgSQL to not fail when a `FOR` loop's target variable is a record containing composite-type fields (Tom)
- Fix PL/Tcl to behave correctly with Tcl 8.5, and to be more careful about the encoding of data sent to or from Tcl (Tom)
- Fix PL/Python to work with Python 2.5

  This is a back-port of fixes made during the 8.2 development cycle.
- Improve pg_dump and pg_restore's error reporting after failure to send a SQL command (Tom)
- Fix pg_ctl to properly preserve postmaster command-line arguments across a `restart` (Bruce)
- Update time zone data files to tzdata release 2008f (for DST law changes in Argentina, Bahamas, Brazil, Mauritius, Morocco, Pakistan, Palestine, and Paraguay)

E.156. Release 8.1.13

Release date: 2008-06-12

This release contains one serious and one minor bug fix over 8.1.12. For information about new features in the 8.1 major release, see 第 E.169 节.

E.156.1. Migration to Version 8.1.13

A dump/restore is not required for those running 8.1.X. However, if you are upgrading from a version earlier than 8.1.2, see 第 E.167 节.

E.156.2. Changes

- Make `pg_get_ruledef()` parenthesize negative constants (Tom)

  Before this fix, a negative constant in a view or rule might be dumped as, say, `-42::integer`, which is subtly incorrect: it should be `(-42)::integer` due to operator precedence rules. Usually this would make little difference, but it could interact with another recent patch to cause PostgreSQL to reject what had been a valid `SELECT DISTINCT` view query. Since this could result in pg_dump output failing to reload, it is being treated as a high-priority fix. The only released versions in which dump output is actually incorrect are 8.3.1 and 8.2.7.

- Make `ALTER AGGREGATE ... OWNER TO` update `pg_shdepend` (Tom)

  This oversight could lead to problems if the aggregate was later involved in a `DROP OWNED` or `REASSIGN OWNED` operation.

E.157. Release 8.1.12

Release date: never released

This release contains a variety of fixes from 8.1.11. For information about new features in the 8.1 major release, see 第 E.169 节.

E.157.1. Migration to Version 8.1.12

A dump/restore is not required for those running 8.1.X. However, if you are upgrading from a version earlier than 8.1.2, see 第 E.167 节.

E.157.2. Changes

- Fix `ALTER TABLE ADD COLUMN ... PRIMARY KEY` so that the new column is correctly checked to see if it's been initialized to all non-nulls (Brendan Jurd)

  Previous versions neglected to check this requirement at all.

- Fix possible `CREATE TABLE` failure when inheriting the "same" constraint from multiple parent relations that inherited that constraint from a common ancestor (Tom)

- Fix conversions between ISO-8859-5 and other encodings to handle Cyrillic "Yo" characters (`e` and `E` with two dots) (Sergey Burladyan)

- Fix a few datatype input functions that were allowing unused bytes in their results to contain uninitialized, unpredictable values (Tom)

  This could lead to failures in which two apparently identical literal values were not seen as equal, resulting in the parser complaining about unmatched `ORDER BY` and `DISTINCT` expressions.

- Fix a corner case in regular-expression substring matching (`substring(string from pattern)`) (Tom)

  The problem occurs when there is a match to the pattern overall but the user has specified a parenthesized subexpression and that subexpression hasn't got a match. An example is `substring('foo' from 'foo(bar)?')`. This should return NULL, since `(bar)` isn't matched, but it was mistakenly returning the whole-pattern match instead (ie, `foo`).

- Update time zone data files to tzdata release 2008c (for DST law changes in Morocco, Iraq, Choibalsan, Pakistan, Syria, Cuba, Argentina/San_Luis, and Chile)

- Fix incorrect result from ecpg's `PGTYPEStimestamp_sub()` function (Michael)

- Fix core dump in `contrib/xml2`'s `xpath_table()` function when the input query returns a NULL value (Tom)

- Fix `contrib/xml2`'s makefile to not override `CFLAGS` (Tom)

- Fix `DatumGetBool` macro to not fail with gcc 4.3 (Tom)

  This problem affects "old style" (V0) C functions that return boolean. The fix is already in 8.3, but the need to back-patch it was not realized at the time.

- Fix longstanding `LISTEN/NOTIFY` race condition (Tom)

  In rare cases a session that had just executed a `LISTEN` might not get a notification, even though one would be expected because the concurrent transaction executing `NOTIFY` was observed to commit later.

  A side effect of the fix is that a transaction that has executed a not-yet-committed `LISTEN` command will not see any row in `pg_listener` for the `LISTEN`, should it choose to look; formerly it would have. This behavior was never documented one way or the other, but it is possible that some applications depend on the old behavior.

- Disallow `LISTEN` and `UNLISTEN` within a prepared transaction (Tom)

  This was formerly allowed but trying to do it had various unpleasant consequences, notably that the originating backend could not exit as long as an `UNLISTEN` remained uncommitted.

- Fix rare crash when an error occurs during a query using a hash index (Heikki)

- Fix input of datetime values for February 29 in years BC (Tom)

  The former coding was mistaken about which years were leap years.

- Fix "unrecognized node type" error in some variants of `ALTER OWNER` (Tom)

- Fix pg_ctl to correctly extract the postmaster's port number from command-line options (Itagaki Takahiro, Tom)

  Previously, `pg_ctl start -w` could try to contact the postmaster on the wrong port, leading to bogus reports of startup failure.

- Use `-fwrapv` to defend against possible misoptimization in recent gcc versions (Tom)

  This is known to be necessary when building PostgreSQL with gcc 4.3 or later.

- Fix display of constant expressions in `ORDER BY` and `GROUP BY` (Tom)

An explicitly casted constant would be shown incorrectly. This could for example lead to corruption of a view definition during dump and reload.

- Fix libpq to handle NOTICE messages correctly during COPY OUT (Tom)

  This failure has only been observed to occur when a user-defined datatype's output routine issues a NOTICE, but there is no guarantee it couldn't happen due to other causes.

E.158. Release 8.1.11

Release date: 2008-01-07

This release contains a variety of fixes from 8.1.10, including fixes for significant security issues. For information about new features in the 8.1 major release, see 第 E.169 节.

This is the last 8.1.X release for which the PostgreSQL community will produce binary packages for Windows. Windows users are encouraged to move to 8.2.X or later, since there are Windows-specific fixes in 8.2.X that are impractical to back-port. 8.1.X will continue to be supported on other platforms.

E.158.1. Migration to Version 8.1.11

A dump/restore is not required for those running 8.1.X. However, if you are upgrading from a version earlier than 8.1.2, see 第 E.167 节.

E.158.2. Changes

- Prevent functions in indexes from executing with the privileges of the user running `VACUUM`, `ANALYZE`, etc (Tom)

  Functions used in index expressions and partial-index predicates are evaluated whenever a new table entry is made. It has long been understood that this poses a risk of trojan-horse code execution if one modifies a table owned by an untrustworthy user. (Note that triggers, defaults, check constraints, etc. pose the same type of risk.) But functions in indexes pose extra danger because they will be executed by routine maintenance operations such as `VACUUM FULL`, which are commonly performed automatically under a superuser account. For example, a nefarious user can execute code with superuser privileges by setting up a trojan-horse index definition and waiting for the next routine vacuum. The fix arranges for standard maintenance operations (including `VACUUM`, `ANALYZE`, `REINDEX`, and `CLUSTER`) to execute as the table owner rather than the calling user, using the same privilege-switching mechanism already used for `SECURITY DEFINER` functions. To prevent bypassing this security measure, execution of `SET SESSION AUTHORIZATION` and `SET ROLE` is now forbidden within a `SECURITY DEFINER` context. (CVE-2007-6600)

- Repair assorted bugs in the regular-expression package (Tom, Will Drewry)

  Suitably crafted regular-expression patterns could cause crashes, infinite or near-infinite looping, and/or massive memory consumption, all of which pose denial-of-service hazards for applications that accept regex search patterns from untrustworthy sources. (CVE-2007-4769, CVE-2007-4772, CVE-2007-6067)

- Require non-superusers who use `/contrib/dblink` to use only password authentication, as a security measure (Joe)

The fix that appeared for this in 8.1.10 was incomplete, as it plugged the hole for only some `dblink` functions. (CVE-2007-6601, CVE-2007-3278)

- Update time zone data files to tzdata release 2007k (in particular, recent Argentina changes) (Tom)
- Improve planner's handling of LIKE/regex estimation in non-C locales (Tom)
- Fix planner failure in some cases of `WHERE false AND var IN (SELECT ...)` (Tom)
- Preserve the tablespace of indexes that are rebuilt by `ALTER TABLE ... ALTER COLUMN TYPE` (Tom)
- Make archive recovery always start a new WAL timeline, rather than only when a recovery stop time was used (Simon)

  This avoids a corner-case risk of trying to overwrite an existing archived copy of the last WAL segment, and seems simpler and cleaner than the original definition.
- Make `VACUUM` not use all of `maintenance_work_mem` when the table is too small for it to be useful (Alvaro)
- Fix potential crash in `translate()` when using a multibyte database encoding (Tom)
- Fix overflow in `extract(epoch from interval)` for intervals exceeding 68 years (Tom)
- Fix PL/Perl to not fail when a UTF-8 regular expression is used in a trusted function (Andrew)
- Fix PL/Perl to cope when platform's Perl defines type `bool` as `int` rather than `char` (Tom)

  While this could theoretically happen anywhere, no standard build of Perl did things this way ... until Mac OS X 10.5.
- Fix PL/Python to not crash on long exception messages (Alvaro)
- Fix pg_dump to correctly handle inheritance child tables that have default expressions different from their parent's (Tom)
- Fix libpq crash when `PGPASSFILE` refers to a file that is not a plain file (Martin Pitt)
- ecpg parser fixes (Michael)
- Make `contrib/pgcrypto` defend against OpenSSL libraries that fail on keys longer than 128 bits; which is the case at least on some Solaris versions (Marko Kreen)
- Make `contrib/tablefunc`'s `crosstab()` handle NULL rowid as a category in its own right, rather than crashing (Joe)
- Fix `tsvector` and `tsquery` output routines to escape backslashes correctly (Teodor, Bruce)
- Fix crash of `to_tsvector()` on huge input strings (Teodor)
- Require a specific version of Autoconf to be used when re-generating the `configure` script (Peter)

This affects developers and packagers only. The change was made to prevent accidental use of untested combinations of Autoconf and PostgreSQL versions. You can remove the version check if you really want to use a different Autoconf version, but it's your responsibility whether the result works or not.

E.159. Release 8.1.10

Release date: 2007-09-17

This release contains a variety of fixes from 8.1.9. For information about new features in the 8.1 major release, see 第 E.169 节.

E.159.1. Migration to Version 8.1.10

A dump/restore is not required for those running 8.1.X. However, if you are upgrading from a version earlier than 8.1.2, see 第 E.167 节.

E.159.2. Changes

- Prevent index corruption when a transaction inserts rows and then aborts close to the end of a concurrent `VACUUM` on the same table (Tom)
- Make `CREATE DOMAIN ... DEFAULT NULL` work properly (Tom)
- Allow the `interval` data type to accept input consisting only of milliseconds or microseconds (Neil)
- Speed up rtree index insertion (Teodor)
- Fix excessive logging of SSL error messages (Tom)
- Fix logging so that log messages are never interleaved when using the syslogger process (Andrew)
- Fix crash when `log_min_error_statement` logging runs out of memory (Tom)
- Fix incorrect handling of some foreign-key corner cases (Tom)
- Prevent `REINDEX` and `CLUSTER` from failing due to attempting to process temporary tables of other sessions (Alvaro)
- Update the time zone database rules, particularly New Zealand's upcoming changes (Tom)
- Windows socket improvements (Magnus)
- Suppress timezone name (`%Z`) in log timestamps on Windows because of possible encoding mismatches (Tom)
- Require non-superusers who use `/contrib/dblink` to use only password authentication, as a security measure (Joe)

E.160. Release 8.1.9

Release date: 2007-04-23

This release contains a variety of fixes from 8.1.8, including a security fix. For information about new features in the 8.1 major release, see 第 E.169 节.

E.160.1. Migration to Version 8.1.9

A dump/restore is not required for those running 8.1.X. However, if you are upgrading from a version earlier than 8.1.2, see 第 E.167 节.

E.160.2. Changes

- Support explicit placement of the temporary-table schema within `search_path`, and disable searching it for functions and operators (Tom)

  This is needed to allow a security-definer function to set a truly secure value of `search_path`. Without it, an unprivileged SQL user can use temporary objects to execute code with the privileges of the security-definer function (CVE-2007-2138). See `CREATE FUNCTION` for more information.

- `/contrib/tsearch2` crash fixes (Teodor)
- Require `COMMIT PREPARED` to be executed in the same database as the transaction was prepared in (Heikki)
- Fix potential-data-corruption bug in how `VACUUM FULL` handles `UPDATE` chains (Tom, Pavan Deolasee)
- Planner fixes, including improving outer join and bitmap scan selection logic (Tom)
- Fix PANIC during enlargement of a hash index (bug introduced in 8.1.6) (Tom)
- Fix POSIX-style timezone specs to follow new USA DST rules (Tom)

E.161. Release 8.1.8

Release date: 2007-02-07

This release contains one fix from 8.1.7. For information about new features in the 8.1 major release, see 第 E.169 节.

E.161.1. Migration to Version 8.1.8

A dump/restore is not required for those running 8.1.X. However, if you are upgrading from a version earlier than 8.1.2, see 第 E.167 节.

E.161.2. Changes

- Remove overly-restrictive check for type length in constraints and functional indexes(Tom)

E.162. Release 8.1.7

Release date: 2007-02-05

This release contains a variety of fixes from 8.1.6, including a security fix. For information about new features in the 8.1 major release, see 第 E.169 节.

E.162.1. Migration to Version 8.1.7

A dump/restore is not required for those running 8.1.X. However, if you are upgrading from a version earlier than 8.1.2, see 第 E.167 节.

E.162.2. Changes

- Remove security vulnerabilities that allowed connected users to read backend memory (Tom)

  The vulnerabilities involve suppressing the normal check that a SQL function returns the data type it's declared to, and changing the data type of a table column (CVE-2007-0555, CVE-2007-0556). These errors can easily be exploited to cause a backend crash, and in principle might be used to read database content that the user should not be able to access.

- Fix rare bug wherein btree index page splits could fail due to choosing an infeasible split point (Heikki Linnakangas)

- Improve `VACUUM` performance for databases with many tables (Tom)

- Fix autovacuum to avoid leaving non-permanent transaction IDs in non-connectable databases (Alvaro)

  This bug affects the 8.1 branch only.

- Fix for rare Assert() crash triggered by `UNION` (Tom)

- Tighten security of multi-byte character processing for UTF8 sequences over three bytes long (Tom)

- Fix bogus "permission denied" failures occurring on Windows due to attempts to fsync already-deleted files (Magnus, Tom)

- Fix possible crashes when an already-in-use PL/pgSQL function is updated (Tom)

E.163. Release 8.1.6

Release date: 2007-01-08

This release contains a variety of fixes from 8.1.5. For information about new features in the 8.1 major release, see 第 E.169 节.

E.163.1. Migration to Version 8.1.6

A dump/restore is not required for those running 8.1.X. However, if you are upgrading from a version earlier than 8.1.2, see 第 E.167 节.

E.163.2. Changes

- Improve handling of `getaddrinfo()` on AIX (Tom)

  This fixes a problem with starting the statistics collector, among other things.

- Fix pg_restore to handle a tar-format backup that contains large objects (blobs) with comments (Tom)

- Fix "failed to re-find parent key" errors in `VACUUM` (Tom)

- Clean out `pg_internal.init` cache files during server restart (Simon)

  This avoids a hazard that the cache files might contain stale data after PITR recovery.

- Fix race condition for truncation of a large relation across a gigabyte boundary by `VACUUM` (Tom)
- Fix bug causing needless deadlock errors on row-level locks (Tom)
- Fix bugs affecting multi-gigabyte hash indexes (Tom)
- Fix possible deadlock in Windows signal handling (Teodor)
- Fix error when constructing an `ARRAY[]` made up of multiple empty elements (Tom)
- Fix ecpg memory leak during connection (Michael)
- Fix for Darwin (OS X) compilation (Tom)
- `to_number()` and `to_char(numeric)` are now `STABLE`, not `IMMUTABLE`, for new initdb installs (Tom)

  This is because `lc_numeric` can potentially change the output of these functions.
- Improve index usage of regular expressions that use parentheses (Tom)

  This improves psql `\d` performance also.
- Update timezone database

  This affects Australian and Canadian daylight-savings rules in particular.

E.164. Release 8.1.5

Release date: 2006-10-16

This release contains a variety of fixes from 8.1.4. For information about new features in the 8.1 major release, see 第 E.169 节.

E.164.1. Migration to Version 8.1.5

A dump/restore is not required for those running 8.1.X. However, if you are upgrading from a version earlier than 8.1.2, see 第 E.167 节.

E.164.2. Changes

- Disallow aggregate functions in `UPDATE` commands, except within sub-SELECTs (Tom)

  The behavior of such an aggregate was unpredictable, and in 8.1.X could cause a crash, so it has been disabled. The SQL standard does not allow this either.
- Fix core dump when an untyped literal is taken as ANYARRAY
- Fix core dump in duration logging for extended query protocol when a `COMMIT` or `ROLLBACK` is executed
- Fix mishandling of AFTER triggers when query contains a SQL function returning multiple rows (Tom)
- Fix `ALTER TABLE ... TYPE` to recheck `NOT NULL` for `USING` clause (Tom)
- Fix `string_to_array()` to handle overlapping matches for the separator string

For example, `string_to_array('123xx456xxx789', 'xx')`.

- Fix `to_timestamp()` for `AM`/`PM` formats (Bruce)
- Fix autovacuum's calculation that decides whether `ANALYZE` is needed (Alvaro)
- Fix corner cases in pattern matching for psql's `\d` commands
- Fix index-corrupting bugs in /contrib/ltree (Teodor)
- Numerous robustness fixes in ecpg (Joachim Wieland)
- Fix backslash escaping in /contrib/dbmirror
- Minor fixes in /contrib/dblink and /contrib/tsearch2
- Efficiency improvements in hash tables and bitmap index scans (Tom)
- Fix instability of statistics collection on Windows (Tom, Andrew)
- Fix `statement_timeout` to use the proper units on Win32 (Bruce)

  In previous Win32 8.1.X versions, the delay was off by a factor of 100.
- Fixes for MSVC and Borland C++ compilers (Hiroshi Saito)
- Fixes for AIX and Intel compilers (Tom)
- Fix rare bug in continuous archiving (Tom)

E.165. Release 8.1.4

Release date: 2006-05-23

This release contains a variety of fixes from 8.1.3, including patches for extremely serious security issues. For information about new features in the 8.1 major release, see 第 E.169 节.

E.165.1. Migration to Version 8.1.4

A dump/restore is not required for those running 8.1.X. However, if you are upgrading from a version earlier than 8.1.2, see 第 E.167 节.

Full security against the SQL-injection attacks described in CVE-2006-2313 and CVE-2006-2314 might require changes in application code. If you have applications that embed untrustworthy strings into SQL commands, you should examine them as soon as possible to ensure that they are using recommended escaping techniques. In most cases, applications should be using subroutines provided by libraries or drivers (such as libpq's `PQescapeStringConn()`) to perform string escaping, rather than relying on ad hoc code to do it.

E.165.2. Changes

- Change the server to reject invalidly-encoded multibyte characters in all cases (Tatsuo, Tom)

  While PostgreSQL has been moving in this direction for some time, the checks are now applied uniformly to all encodings and all textual input, and are now always

errors not merely warnings. This change defends against SQL-injection attacks of the type described in CVE-2006-2313.

- Reject unsafe uses of `\'` in string literals

  As a server-side defense against SQL-injection attacks of the type described in CVE-2006-2314, the server now only accepts `''` and not `\'` as a representation of ASCII single quote in SQL string literals. By default, `\'` is rejected only when `client_encoding` is set to a client-only encoding (SJIS, BIG5, GBK, GB18030, or UHC), which is the scenario in which SQL injection is possible. A new configuration parameter `backslash_quote` is available to adjust this behavior when needed. Note that full security against CVE-2006-2314 might require client-side changes; the purpose of `backslash_quote` is in part to make it obvious that insecure clients are insecure.

- Modify libpq's string-escaping routines to be aware of encoding considerations and `standard_conforming_strings`

  This fixes libpq-using applications for the security issues described in CVE-2006-2313 and CVE-2006-2314, and also future-proofs them against the planned changeover to SQL-standard string literal syntax. Applications that use multiple PostgreSQL connections concurrently should migrate to `PQescapeStringConn()` and `PQescapeByteaConn()` to ensure that escaping is done correctly for the settings in use in each database connection. Applications that do string escaping "by hand" should be modified to rely on library routines instead.

- Fix weak key selection in pgcrypto (Marko Kreen)

  Errors in fortuna PRNG reseeding logic could cause a predictable session key to be selected by `pgp_sym_encrypt()` in some cases. This only affects non-OpenSSL-using builds.

- Fix some incorrect encoding conversion functions

  `win1251_to_iso`, `win866_to_iso`, `euc_tw_to_big5`, `euc_tw_to_mic`, `mic_to_euc_tw` were all broken to varying extents.

- Clean up stray remaining uses of `\'` in strings (Bruce, Jan)

- Make autovacuum visible in `pg_stat_activity` (Alvaro)

- Disable `full_page_writes` (Tom)

  In certain cases, having `full_page_writes` off would cause crash recovery to fail. A proper fix will appear in 8.2; for now it's just disabled.

- Various planner fixes, particularly for bitmap index scans and MIN/MAX optimization (Tom)

- Fix incorrect optimization in merge join (Tom)

  Outer joins could sometimes emit multiple copies of unmatched rows.

- Fix crash from using and modifying a plpgsql function in the same transaction

- Fix WAL replay for case where a B-Tree index has been truncated

- Fix `SIMILAR TO` for patterns involving `|` (Tom)

- Fix `SELECT INTO` and `CREATE TABLE AS` to create tables in the default tablespace, not the base directory (Kris Jurka)

- Fix server to use custom DH SSL parameters correctly (Michael Fuhr)
- Improve qsort performance (Dann Corbit)

  Currently this code is only used on Solaris.
- Fix for OS/X Bonjour on x86 systems (Ashley Clark)
- Fix various minor memory leaks
- Fix problem with password prompting on some Win32 systems (Robert Kinberg)
- Improve pg_dump's handling of default values for domains
- Fix pg_dumpall to handle identically-named users and groups reasonably (only possible when dumping from a pre-8.1 server) (Tom)

  The user and group will be merged into a single role with `LOGIN` permission. Formerly the merged role wouldn't have `LOGIN` permission, making it unusable as a user.
- Fix pg_restore `-n` to work as documented (Tom)

E.166. Release 8.1.3

Release date: 2006-02-14

This release contains a variety of fixes from 8.1.2, including one very serious security issue. For information about new features in the 8.1 major release, see 第 E.169 节.

E.166.1. Migration to Version 8.1.3

A dump/restore is not required for those running 8.1.X. However, if you are upgrading from a version earlier than 8.1.2, see 第 E.167 节.

E.166.2. Changes

- Fix bug that allowed any logged-in user to `SET ROLE` to any other database user id (CVE-2006-0553)

  Due to inadequate validity checking, a user could exploit the special case that `SET ROLE` normally uses to restore the previous role setting after an error. This allowed ordinary users to acquire superuser status, for example. The escalation-of-privilege risk exists only in 8.1.0-8.1.2. However, in all releases back to 7.3 there is a related bug in `SET SESSION AUTHORIZATION` that allows unprivileged users to crash the server, if it has been compiled with Asserts enabled (which is not the default). Thanks to Akio Ishida for reporting this problem.
- Fix bug with row visibility logic in self-inserted rows (Tom)

  Under rare circumstances a row inserted by the current command could be seen as already valid, when it should not be. Repairs bug created in 8.0.4, 7.4.9, and 7.3.11 releases.
- Fix race condition that could lead to “file already exists” errors during pg_clog and pg_subtrans file creation (Tom)
- Fix cases that could lead to crashes if a cache-invalidation message arrives at just the wrong time (Tom)

- Properly check `DOMAIN` constraints for `UNKNOWN` parameters in prepared statements (Neil)
- Ensure `ALTER COLUMN TYPE` will process `FOREIGN KEY`, `UNIQUE`, and `PRIMARY KEY` constraints in the proper order (Nakano Yoshihisa)
- Fixes to allow restoring dumps that have cross-schema references to custom operators or operator classes (Tom)
- Allow pg_restore to continue properly after a `COPY` failure; formerly it tried to treat the remaining `COPY` data as SQL commands (Stephen Frost)
- Fix pg_ctl `unregister` crash when the data directory is not specified (Magnus)
- Fix libpq `PQprint` HTML tags (Christoph Zwerschke)
- Fix ecpg crash on AMD64 and PPC (Neil)
- Allow `SETOF` and `%TYPE` to be used together in function result type declarations
- Recover properly if error occurs during argument passing in PL/python (Neil)
- Fix memory leak in `plperl_return_next` (Neil)
- Fix PL/perl's handling of locales on Win32 to match the backend (Andrew)
- Various optimizer fixes (Tom)
- Fix crash when `log_min_messages` is set to `DEBUG3` or above in `postgresql.conf` on Win32 (Bruce)
- Fix pgxs `-L` library path specification for Win32, Cygwin, OS X, AIX (Bruce)
- Check that SID is enabled while checking for Win32 admin privileges (Magnus)
- Properly reject out-of-range date inputs (Kris Jurka)
- Portability fix for testing presence of `finite` and `isinf` during configure (Tom)
- Improve speed of `COPY IN` via libpq, by avoiding a kernel call per data line (Alon Goldshuv)
- Improve speed of `/contrib/tsearch2` index creation (Tom)

E.167. Release 8.1.2

Release date: 2006-01-09

This release contains a variety of fixes from 8.1.1. For information about new features in the 8.1 major release, see 第 E.169 节.

E.167.1. Migration to Version 8.1.2

A dump/restore is not required for those running 8.1.X. However, you might need to `REINDEX` indexes on textual columns after updating, if you are affected by the locale or plperl issues described below.

E.167.2. Changes

- Fix Windows code so that postmaster will continue rather than exit if there is no more room in ShmemBackendArray (Magnus)

The previous behavior could lead to a denial-of-service situation if too many connection requests arrive close together. This applies only to the Windows port.

- Fix bug introduced in 8.0 that could allow ReadBuffer to return an already-used page as new, potentially causing loss of recently-committed data (Tom)
- Fix for protocol-level Describe messages issued outside a transaction or in a failed transaction (Tom)
- Fix character string comparison for locales that consider different character combinations as equal, such as Hungarian (Tom)

  This might require `REINDEX` to fix existing indexes on textual columns.
- Set locale environment variables during postmaster startup to ensure that plperl won't change the locale later

  This fixes a problem that occurred if the postmaster was started with environment variables specifying a different locale than what initdb had been told. Under these conditions, any use of plperl was likely to lead to corrupt indexes. You might need `REINDEX` to fix existing indexes on textual columns if this has happened to you.
- Allow more flexible relocation of installation directories (Tom)

  Previous releases supported relocation only if all installation directory paths were the same except for the last component.
- Prevent crashes caused by the use of `ISO-8859-5` and `ISO-8859-9` encodings (Tatsuo)
- Fix longstanding bug in strpos() and regular expression handling in certain rarely used Asian multi-byte character sets (Tatsuo)
- Fix bug where COPY CSV mode considered any `\.` to terminate the copy data

  The new code requires `\.` to appear alone on a line, as per documentation.
- Make COPY CSV mode quote a literal data value of `\.` to ensure it cannot be interpreted as the end-of-data marker (Bruce)
- Various fixes for functions returning `RECORD`s (Tom)
- Fix processing of `postgresql.conf` so a final line with no newline is processed properly (Tom)
- Fix bug in `/contrib/pgcrypto` gen_salt, which caused it not to use all available salt space for MD5 and XDES algorithms (Marko Kreen, Solar Designer)

  Salts for Blowfish and standard DES are unaffected.
- Fix autovacuum crash when processing expression indexes
- Fix `/contrib/dblink` to throw an error, rather than crashing, when the number of columns specified is different from what's actually returned by the query (Joe)

E.168. Release 8.1.1

Release date: 2005-12-12

This release contains a variety of fixes from 8.1.0. For information about new features in the 8.1 major release, see 第 E.169 节.

E.168.1. Migration to Version 8.1.1

A dump/restore is not required for those running 8.1.X.

E.168.2. Changes

- Fix incorrect optimizations of outer-join conditions (Tom)
- Fix problems with wrong reported column names in cases involving sub-selects flattened by the optimizer (Tom)
- Fix update failures in scenarios involving CHECK constraints, toasted columns, and indexes (Tom)
- Fix bgwriter problems after recovering from errors (Tom)

  The background writer was found to leak buffer pins after write errors. While not fatal in itself, this might lead to mysterious blockages of later VACUUM commands.
- Prevent failure if client sends Bind protocol message when current transaction is already aborted
- `/contrib/tsearch2` and `/contrib/ltree` fixes (Teodor)
- Fix problems with translated error messages in languages that require word reordering, such as Turkish; also problems with unexpected truncation of output strings and wrong display of the smallest possible bigint value (Andrew, Tom)

  These problems only appeared on platforms that were using our `port/snprintf.c` code, which includes BSD variants if `--enable-nls` was given, and perhaps others. In addition, a different form of the translated-error-message problem could appear on Windows depending on which version of `libintl` was used.
- Re-allow `AM`/`PM`, `HH`, `HH12`, and `D` format specifiers for `to_char(time)` and `to_char(interval)`. (`to_char(interval)` should probably use `HH24`.) (Bruce)
- AIX, HPUX, and MSVC compile fixes (Tom, Hiroshi Saito)
- Optimizer improvements (Tom)
- Retry file reads and writes after Windows NO_SYSTEM_RESOURCES error (Qingqing Zhou)
- Prevent autovacuum from crashing during ANALYZE of expression index (Alvaro)
- Fix problems with ON COMMIT DELETE ROWS temp tables
- Fix problems when a trigger alters the output of a SELECT DISTINCT query
- Add 8.1.0 release note item on how to migrate invalid `UTF-8` byte sequences (Paul Lindner)

E.169. Release 8.1

Release date: 2005-11-08

E.169.1. Overview

Major changes in this release:

Improve concurrent access to the shared buffer cache (Tom)

Access to the shared buffer cache was identified as a significant scalability problem, particularly on multi-CPU systems. In this release, the way that locking is done in the buffer manager has been overhauled to reduce lock contention and improve scalability. The buffer manager has also been changed to use a “clock sweep” replacement policy.

Allow index scans to use an intermediate in-memory bitmap (Tom)

In previous releases, only a single index could be used to do lookups on a table. With this feature, if a query has `WHERE tab.col1 = 4 and tab.col2 = 9`, and there is no multicolumn index on `col1` and `col2`, but there is an index on `col1` and another on `col2`, it is possible to search both indexes and combine the results in memory, then do heap fetches for only the rows matching both the `col1` and `col2` restrictions. This is very useful in environments that have a lot of unstructured queries where it is impossible to create indexes that match all possible access conditions. Bitmap scans are useful even with a single index, as they reduce the amount of random access needed; a bitmap index scan is efficient for retrieving fairly large fractions of the complete table, whereas plain index scans are not.

Add two-phase commit (Heikki Linnakangas, Alvaro, Tom)

Two-phase commit allows transactions to be "prepared" on several computers, and once all computers have successfully prepared their transactions (none failed), all transactions can be committed. Even if a machine crashes after a prepare, the prepared transaction can be committed after the machine is restarted. New syntax includes `PREPARE TRANSACTION` and `COMMIT/ROLLBACK PREPARED`. A new system view `pg_prepared_xacts` has also been added.

Create a new role system that replaces users and groups (Stephen Frost)

Roles are a combination of users and groups. Like users, they can have login capability, and like groups, a role can have other roles as members. Roles basically remove the distinction between users and groups. For example, a role can:

- Have login capability (optionally)
- Own objects
- Hold access permissions for database objects
- Inherit permissions from other roles it is a member of

Once a user logs into a role, she obtains capabilities of the login role plus any inherited roles, and can use `SET ROLE` to switch to other roles she is a member of. This feature is a generalization of the SQL standard's concept of roles. This change also replaces `pg_shadow` and `pg_group` by new role-capable catalogs `pg_authid` and `pg_auth_members`. The old tables are redefined as read-only views on the new role tables.

Automatically use indexes for `MIN()` and `MAX()` (Tom)

In previous releases, the only way to use an index for `MIN()` or `MAX()` was to rewrite the query as `SELECT col FROM tab ORDER BY col LIMIT 1`. Index usage now happens automatically.

Move `/contrib/pg_autovacuum` into the main server (Alvaro)

Integrating autovacuum into the server allows it to be automatically started and stopped in sync with the database server, and allows autovacuum to be configured from `postgresql.conf`.

Add shared row level locks using `SELECT ... FOR SHARE` (Alvaro)

While PostgreSQL's MVCC locking allows `SELECT` to never be blocked by writers and therefore does not need shared row locks for typical operations, shared locks are useful for applications that require shared row locking. In particular this reduces the locking requirements imposed by referential integrity checks.

Add dependencies on shared objects, specifically roles (Alvaro)

This extension of the dependency mechanism prevents roles from being dropped while there are still database objects they own. Formerly it was possible to accidentally "orphan" objects by deleting their owner. While this could be recovered from, it was messy and unpleasant.

Improve performance for partitioned tables (Simon)

The new `constraint_exclusion` configuration parameter avoids lookups on child tables where constraints indicate that no matching rows exist in the child table.

This allows for a basic type of table partitioning. If child tables store separate key ranges and this is enforced using appropriate `CHECK` constraints, the optimizer will skip child table accesses when the constraint guarantees no matching rows exist in the child table.

E.169.2. Migration to Version 8.1

A dump/restore using pg_dump is required for those wishing to migrate data from any previous release.

The 8.0 release announced that the `to_char()` function for intervals would be removed in 8.1. However, since no better API has been suggested, `to_char(interval)` has been enhanced in 8.1 and will remain in the server.

Observe the following incompatibilities:

- `add_missing_from` is now false by default (Neil)

  By default, we now generate an error if a table is used in a query without a `FROM` reference. The old behavior is still available, but the parameter must be set to 'true' to obtain it.

  It might be necessary to set `add_missing_from` to true in order to load an existing dump file, if the dump contains any views or rules created using the implicit-`FROM` syntax. This should be a one-time annoyance, because PostgreSQL 8.1 will convert such views and rules to standard explicit-`FROM` syntax. Subsequent dumps will therefore not have the problem.

- Cause input of a zero-length string (`''`) for `float4`/`float8`/`oid` to throw an error, rather than treating it as a zero (Neil)

  This change is consistent with the current handling of zero-length strings for integers. The schedule for this change was announced in 8.0.

- `default_with_oids` is now false by default (Neil)

With this option set to false, user-created tables no longer have an OID column unless `WITH OIDS` is specified in `CREATE TABLE`. Though OIDs have existed in all releases of PostgreSQL, their use is limited because they are only four bytes long and the counter is shared across all installed databases. The preferred way of uniquely identifying rows is via sequences and the `SERIAL` type, which have been supported since PostgreSQL 6.4.

- Add `E''` syntax so eventually ordinary strings can treat backslashes literally (Bruce)

  Currently PostgreSQL processes a backslash in a string literal as introducing a special escape sequence, e.g. `\n` or `\010`. While this allows easy entry of special values, it is nonstandard and makes porting of applications from other databases more difficult. For this reason, the PostgreSQL project is planning to remove the special meaning of backslashes in strings. For backward compatibility and for users who want special backslash processing, a new string syntax has been created. This new string syntax is formed by writing an `E` immediately preceding the single quote that starts the string, e.g. `E'hi\n'`. While this release does not change the handling of backslashes in strings, it does add new configuration parameters to help users migrate applications for future releases:

  - `standard_conforming_strings` – does this release treat backslashes literally in ordinary strings?
  - `escape_string_warning` – warn about backslashes in ordinary (non-E) strings

  The `standard_conforming_strings` value is read-only. Applications can retrieve the value to know how backslashes are processed. (Presence of the parameter can also be taken as an indication that `E''` string syntax is supported.) In a future release, `standard_conforming_strings` will be true, meaning backslashes will be treated literally in non-E strings. To prepare for this change, use `E''` strings in places that need special backslash processing, and turn on `escape_string_warning` to find additional strings that need to be converted to use `E''`. Also, use two single-quotes (`''`) to embed a literal single-quote in a string, rather than the PostgreSQL-supported syntax of backslash single-quote (`\'`). The former is standards-conforming and does not require the use of the `E''` string syntax. You can also use the `$$` string syntax, which does not treat backslashes specially.

- Make `REINDEX DATABASE` reindex all indexes in the database (Tom)

  Formerly, `REINDEX DATABASE` reindexed only system tables. This new behavior seems more intuitive. A new command `REINDEX SYSTEM` provides the old functionality of reindexing just the system tables.

- Read-only large object descriptors now obey MVCC snapshot semantics

  When a large object is opened with `INV_READ` (and not `INV_WRITE`), the data read from the descriptor will now reflect a "snapshot" of the large object's state at the time of the transaction snapshot in use by the query that called `lo_open()`. To obtain the old behavior of always returning the latest committed data, include `INV_WRITE` in the mode flags for `lo_open()`.

- Add proper dependencies for arguments of sequence functions (Tom)

  In previous releases, sequence names passed to `nextval()`, `currval()`, and `setval()` were stored as simple text strings, meaning that renaming or dropping a sequence used in a `DEFAULT` clause made the clause invalid. This release stores all newly-created sequence function arguments as internal OIDs, allowing them to track sequence renaming, and adding dependency information that prevents

improper sequence removal. It also makes such `DEFAULT` clauses immune to schema renaming and search path changes.

Some applications might rely on the old behavior of run-time lookup for sequence names. This can still be done by explicitly casting the argument to `text`, for example `nextval('myseq'::text)`.

Pre-8.1 database dumps loaded into 8.1 will use the old text-based representation and therefore will not have the features of OID-stored arguments. However, it is possible to update a database containing text-based `DEFAULT` clauses. First, save this query into a file, such as `fixseq.sql`:

```
SELECT  'ALTER TABLE ' ||
   pg_catalog.quote_ident(n.nspname) || '.' ||
   pg_catalog.quote_ident(c.relname) ||
   ' ALTER COLUMN ' || pg_catalog.quote_ident(a.attname) ||
   ' SET DEFAULT ' ||
   regexp_replace(d.adsrc,
                  $$val\(\(('[^']*')::text\)::regclass$$,
                  $$val(\1$$,
                  'g') ||
   ';'
FROM    pg_namespace n, pg_class c, pg_attribute a, pg_attrdef d
WHERE   n.oid = c.relnamespace AND
   c.oid = a.attrelid AND
   a.attrelid = d.adrelid AND
   a.attnum = d.adnum AND
   d.adsrc ~ $$val\(\('[^']*'::text\)::regclass$$;
```

Next, run the query against a database to find what adjustments are required, like this for database `db1`:

```
psql -t -f fixseq.sql db1
```

This will show the `ALTER TABLE` commands needed to convert the database to the newer OID-based representation. If the commands look reasonable, run this to update the database:

```
psql -t -f fixseq.sql db1 | psql -e db1
```

This process must be repeated in each database to be updated.

- In psql, treat unquoted `\{digit}+` sequences as octal (Bruce)

  In previous releases, `\{digit}+` sequences were treated as decimal, and only `\0{digit}+` were treated as octal. This change was made for consistency.

- Remove grammar productions for prefix and postfix `%` and `^` operators (Tom)

  These have never been documented and complicated the use of the modulus operator (`%`) with negative numbers.

- Make `&<` and `&>` for polygons consistent with the box "over" operators (Tom)

- `CREATE LANGUAGE` can ignore the provided arguments in favor of information from `pg_pltemplate` (Tom)

  A new system catalog `pg_pltemplate` has been defined to carry information about the preferred definitions of procedural languages (such as whether they have validator functions). When an entry exists in this catalog for the language being

created, `CREATE LANGUAGE` will ignore all its parameters except the language name and instead use the catalog information. This measure was taken because of increasing problems with obsolete language definitions being loaded by old dump files. As of 8.1, pg_dump will dump procedural language definitions as just `CREATE LANGUAGE name`, relying on a template entry to exist at load time. We expect this will be a more future-proof representation.

- Make `pg_cancel_backend(int)` return a `boolean` rather than an `integer` (Neil)
- Some users are having problems loading UTF-8 data into 8.1.X. This is because previous versions allowed invalid UTF-8 byte sequences to be entered into the database, and this release properly accepts only valid UTF-8 sequences. One way to correct a dumpfile is to run the command `iconv -c -f UTF-8 -t UTF-8 -o cleanfile.sql dumpfile.sql`. The `-c` option removes invalid character sequences. A diff of the two files will show the sequences that are invalid. `iconv` reads the entire input file into memory so it might be necessary to use split to break up the dump into multiple smaller files for processing.

E.169.3. Additional Changes

Below you will find a detailed account of the additional changes between PostgreSQL 8.1 and the previous major release.

E.169.3.1. Performance Improvements

- Improve GiST and R-tree index performance (Neil)
- Improve the optimizer, including auto-resizing of hash joins (Tom)
- Overhaul internal API in several areas
- Change WAL record CRCs from 64-bit to 32-bit (Tom)

  We determined that the extra cost of computing 64-bit CRCs was significant, and the gain in reliability too marginal to justify it.
- Prevent writing large empty gaps in WAL pages (Tom)
- Improve spinlock behavior on SMP machines, particularly Opterons (Tom)
- Allow nonconsecutive index columns to be used in a multicolumn index (Tom)

  For example, this allows an index on columns a,b,c to be used in a query with `WHERE a = 4 and c = 10`.
- Skip WAL logging for `CREATE TABLE AS` / `SELECT INTO` (Simon)

  Since a crash during `CREATE TABLE AS` would cause the table to be dropped during recovery, there is no reason to WAL log as the table is loaded. (Logging still happens if WAL archiving is enabled, however.)
- Allow concurrent GiST index access (Teodor, Oleg)
- Add configuration parameter `full_page_writes` to control writing full pages to WAL (Bruce)

  To prevent partial disk writes from corrupting the database, PostgreSQL writes a complete copy of each database disk page to WAL the first time it is modified after a checkpoint. This option turns off that functionality for more speed. This is safe to use with battery-backed disk caches where partial page writes cannot happen.

- Use `O_DIRECT` if available when using `O_SYNC` for `wal_sync_method` (Itagaki Takahiro)

  `O_DIRECT` causes disk writes to bypass the kernel cache, and for WAL writes, this improves performance.

- Improve `COPY FROM` performance (Alon Goldshuv)

  This was accomplished by reading `COPY` input in larger chunks, rather than character by character.

- Improve the performance of `COUNT()`, `SUM`, `AVG()`, `STDDEV()`, and `VARIANCE()` (Neil, Tom)

E.169.3.2. Server Changes

- Prevent problems due to transaction ID (XID) wraparound (Tom)

  The server will now warn when the transaction counter approaches the wraparound point. If the counter becomes too close to wraparound, the server will stop accepting queries. This ensures that data is not lost before needed vacuuming is performed.

- Fix problems with object IDs (OIDs) conflicting with existing system objects after the OID counter has wrapped around (Tom)

- Add warning about the need to increase `max_fsm_relations` and `max_fsm_pages` during `VACUUM` (Ron Mayer)

- Add `temp_buffers` configuration parameter to allow users to determine the size of the local buffer area for temporary table access (Tom)

- Add session start time and client IP address to `pg_stat_activity` (Magnus)

- Adjust `pg_stat` views for bitmap scans (Tom)

  The meanings of some of the fields have changed slightly.

- Enhance `pg_locks` view (Tom)

- Log queries for client-side `PREPARE` and `EXECUTE` (Simon)

- Allow Kerberos name and user name case sensitivity to be specified in `postgresql.conf` (Magnus)

- Add configuration parameter `krb_server_hostname` so that the server host name can be specified as part of service principal (Todd Kover)

  If not set, any service principal matching an entry in the keytab can be used. This is new Kerberos matching behavior in this release.

- Add `log_line_prefix` options for millisecond timestamps (`%m`) and remote host (`%h`) (Ed L.)

- Add WAL logging for GiST indexes (Teodor, Oleg)

  GiST indexes are now safe for crash and point-in-time recovery.

- Remove old `*.backup` files when we do `pg_stop_backup()` (Bruce)

  This prevents a large number of `*.backup` files from existing in `pg_xlog/`.

- Add configuration parameters to control TCP/IP keep-alive times for idle, interval, and count (Oliver Jowett)

  These values can be changed to allow more rapid detection of lost client connections.

- Add per-user and per-database connection limits (Petr Jelinek)

  Using `ALTER USER` and `ALTER DATABASE`, limits can now be enforced on the maximum number of sessions that can concurrently connect as a specific user or to a specific database. Setting the limit to zero disables user or database connections.

- Allow more than two gigabytes of shared memory and per-backend work memory on 64-bit machines (Koichi Suzuki)

- New system catalog `pg_pltemplate` allows overriding obsolete procedural-language definitions in dump files (Tom)

E.169.3.3. Query Changes

- Add temporary views (Koju Iijima, Neil)

- Fix `HAVING` without any aggregate functions or `GROUP BY` so that the query returns a single group (Tom)

  Previously, such a case would treat the `HAVING` clause the same as a `WHERE` clause. This was not per spec.

- Add `USING` clause to allow additional tables to be specified to `DELETE` (Euler Taveira de Oliveira, Neil)

  In prior releases, there was no clear method for specifying additional tables to be used for joins in a `DELETE` statement. `UPDATE` already has a `FROM` clause for this purpose.

- Add support for `\x` hex escapes in backend and ecpg strings (Bruce)

  This is just like the standard C `\x` escape syntax. Octal escapes were already supported.

- Add `BETWEEN SYMMETRIC` query syntax (Pavel Stehule)

  This feature allows `BETWEEN` comparisons without requiring the first value to be less than the second. For example, `2 BETWEEN [ASYMMETRIC] 3 AND 1` returns false, while `2 BETWEEN SYMMETRIC 3 AND 1` returns true. `BETWEEN ASYMMETRIC` was already supported.

- Add `NOWAIT` option to `SELECT ... FOR UPDATE/SHARE` (Hans-Juergen Schoenig)

  While the `statement_timeout` configuration parameter allows a query taking more than a certain amount of time to be canceled, the `NOWAIT` option allows a query to be canceled as soon as a `SELECT ... FOR UPDATE/SHARE` command cannot immediately acquire a row lock.

E.169.3.4. Object Manipulation Changes

- Track dependencies of shared objects (Alvaro)

  PostgreSQL allows global tables (users, databases, tablespaces) to reference information in multiple databases. This addition adds dependency information for

global tables, so, for example, user ownership can be tracked across databases, so a user who owns something in any database can no longer be removed. Dependency tracking already existed for database-local objects.

- Allow limited `ALTER OWNER` commands to be performed by the object owner (Stephen Frost)

  Prior releases allowed only superusers to change object owners. Now, ownership can be transferred if the user executing the command owns the object and would be able to create it as the new owner (that is, the user is a member of the new owning role and that role has the CREATE permission that would be needed to create the object afresh).

- Add `ALTER` object `SET SCHEMA` capability for some object types (tables, functions, types) (Bernd Helmle)

  This allows objects to be moved to different schemas.

- Add `ALTER TABLE ENABLE/DISABLE TRIGGER` to disable triggers (Satoshi Nagayasu)

E.169.3.5. Utility Command Changes

- Allow `TRUNCATE` to truncate multiple tables in a single command (Alvaro)

  Because of referential integrity checks, it is not allowed to truncate a table that is part of a referential integrity constraint. Using this new functionality, `TRUNCATE` can be used to truncate such tables, if both tables involved in a referential integrity constraint are truncated in a single `TRUNCATE` command.

- Properly process carriage returns and line feeds in `COPY CSV` mode (Andrew)

  In release 8.0, carriage returns and line feeds in `CSV COPY TO` were processed in an inconsistent manner. (This was documented on the TODO list.)

- Add `COPY WITH CSV HEADER` to allow a header line as the first line in `COPY` (Andrew)

  This allows handling of the common `CSV` usage of placing the column names on the first line of the data file. For `COPY TO`, the first line contains the column names, and for `COPY FROM`, the first line is ignored.

- On Windows, display better sub-second precision in `EXPLAIN ANALYZE` (Magnus)

- Add trigger duration display to `EXPLAIN ANALYZE` (Tom)

  Prior releases included trigger execution time as part of the total execution time, but did not show it separately. It is now possible to see how much time is spent in each trigger.

- Add support for `\x` hex escapes in `COPY` (Sergey Ten)

  Previous releases only supported octal escapes.

- Make `SHOW ALL` include variable descriptions (Matthias Schmidt)

  `SHOW` varname still only displays the variable's value and does not include the description.

- Make initdb create a new standard database called `postgres`, and convert utilities to use `postgres` rather than `template1` for standard lookups (Dave)

In prior releases, `template1` was used both as a default connection for utilities like createuser, and as a template for new databases. This caused `CREATE DATABASE` to sometimes fail, because a new database cannot be created if anyone else is in the template database. With this change, the default connection database is now `postgres`, meaning it is much less likely someone will be using `template1` during `CREATE DATABASE`.

- Create new reindexdb command-line utility by moving `/contrib/reindexdb` into the server (Euler Taveira de Oliveira)

E.169.3.6. Data Type and Function Changes

- Add `MAX()` and `MIN()` aggregates for array types (Koju Iijima)
- Fix `to_date()` and `to_timestamp()` to behave reasonably when `CC` and `YY` fields are both used (Karel Zak)

  If the format specification contains `CC` and a year specification is `YYY` or longer, ignore the `CC`. If the year specification is `YY` or shorter, interpret `CC` as the previous century.

- Add `md5(bytea)` (Abhijit Menon-Sen)

  `md5(text)` already existed.

- Add support for `numeric ^ numeric` based on `power(numeric, numeric)`

  The function already existed, but there was no operator assigned to it.

- Fix `NUMERIC` modulus by properly truncating the quotient during computation (Bruce)

  In previous releases, modulus for large values sometimes returned negative results due to rounding of the quotient.

- Add a function `lastval()` (Dennis Björklund)

  `lastval()` is a simplified version of `currval()`. It automatically determines the proper sequence name based on the most recent `nextval()` or `setval()` call performed by the current session.

- Add `to_timestamp(DOUBLE PRECISION) (Michael Glaesemann)`

  Converts Unix seconds since 1970 to a `TIMESTAMP WITH TIMEZONE`.

- Add `pg_postmaster_start_time()` function (Euler Taveira de Oliveira, Matthias Schmidt)
- Allow the full use of time zone names in `AT TIME ZONE`, not just the short list previously available (Magnus)

  Previously, only a predefined list of time zone names were supported by `AT TIME ZONE`. Now any supported time zone name can be used, e.g.:

  ```
  SELECT CURRENT_TIMESTAMP AT TIME ZONE 'Europe/London';
  ```

  In the above query, the time zone used is adjusted based on the daylight saving time rules that were in effect on the supplied date.

- Add `GREATEST()` and `LEAST()` variadic functions (Pavel Stehule)

These functions take a variable number of arguments and return the greatest or least value among the arguments.

- Add `pg_column_size()` (Mark Kirkwood)

  This returns storage size of a column, which might be compressed.

- Add `regexp_replace()` (Atsushi Ogawa)

  This allows regular expression replacement, like sed. An optional flag argument allows selection of global (replace all) and case-insensitive modes.

- Fix interval division and multiplication (Bruce)

  Previous versions sometimes returned unjustified results, like `'4 months'::interval / 5` returning `'1 mon -6 days'`.

- Fix roundoff behavior in timestamp, time, and interval output (Tom)

  This fixes some cases in which the seconds field would be shown as `60` instead of incrementing the higher-order fields.

- Add a separate day field to type `interval` so a one day interval can be distinguished from a 24 hour interval (Michael Glaesemann)

  Days that contain a daylight saving time adjustment are not 24 hours long, but typically 23 or 25 hours. This change creates a conceptual distinction between intervals of "so many days" and intervals of "so many hours". Adding `1 day` to a timestamp now gives the same local time on the next day even if a daylight saving time adjustment occurs between, whereas adding `24 hours` will give a different local time when this happens. For example, under US DST rules:

  ```
  '2005-04-03 00:00:00-05' + '1 day' = '2005-04-04 00:00:00-04'
  '2005-04-03 00:00:00-05' + '24 hours' = '2005-04-04 01:00:00-04'
  ```

- Add `justify_days()` and `justify_hours()` (Michael Glaesemann)

  These functions, respectively, adjust days to an appropriate number of full months and days, and adjust hours to an appropriate number of full days and hours.

- Move `/contrib/dbsize` into the backend, and rename some of the functions (Dave Page, Andreas Pflug)

  - `pg_tablespace_size()`
  - `pg_database_size()`
  - `pg_relation_size()`
  - `pg_total_relation_size()`
  - `pg_size_pretty()`

  `pg_total_relation_size()` includes indexes and TOAST tables.

- Add functions for read-only file access to the cluster directory (Dave Page, Andreas Pflug)

  - `pg_stat_file()`
  - `pg_read_file()`

  - pg_ls_dir()

- Add pg_reload_conf() to force reloading of the configuration files (Dave Page, Andreas Pflug)
- Add pg_rotate_logfile() to force rotation of the server log file (Dave Page, Andreas Pflug)
- Change pg_stat_* views to include TOAST tables (Tom)

E.169.3.7. Encoding and Locale Changes

- Rename some encodings to be more consistent and to follow international standards (Bruce)
  - UNICODE is now UTF8
  - ALT is now WIN866
  - WIN is now WIN1251
  - TCVN is now WIN1258

  The original names still work.
- Add support for WIN1252 encoding (Roland Volkmann)
- Add support for four-byte UTF8 characters (John Hansen)

  Previously only one, two, and three-byte UTF8 characters were supported. This is particularly important for support for some Chinese character sets.
- Allow direct conversion between EUC_JP and SJIS to improve performance (Atsushi Ogawa)
- Allow the UTF8 encoding to work on Windows (Magnus)

  This is done by mapping UTF8 to the Windows-native UTF16 implementation.

E.169.3.8. General Server-Side Language Changes

- Fix ALTER LANGUAGE RENAME (Sergey Yatskevich)
- Allow function characteristics, like strictness and volatility, to be modified via ALTER FUNCTION (Neil)
- Increase the maximum number of function arguments to 100 (Tom)
- Allow SQL and PL/pgSQL functions to use OUT and INOUT parameters (Tom)

  OUT is an alternate way for a function to return values. Instead of using RETURN, values can be returned by assigning to parameters declared as OUT or INOUT. This is notationally simpler in some cases, particularly so when multiple values need to be returned. While returning multiple values from a function was possible in previous releases, this greatly simplifies the process. (The feature will be extended to other server-side languages in future releases.)
- Move language handler functions into the pg_catalog schema

  This makes it easier to drop the public schema if desired.

- Add `SPI_getnspname()` to SPI (Neil)

E.169.3.9. PL/pgSQL Server-Side Language Changes

- Overhaul the memory management of PL/pgSQL functions (Neil)

  The parsetree of each function is now stored in a separate memory context. This allows this memory to be easily reclaimed when it is no longer needed.

- Check function syntax at `CREATE FUNCTION` time, rather than at runtime (Neil)

  Previously, most syntax errors were reported only when the function was executed.

- Allow `OPEN` to open non-`SELECT` queries like `EXPLAIN` and `SHOW` (Tom)

- No longer require functions to issue a `RETURN` statement (Tom)

  This is a byproduct of the newly added `OUT` and `INOUT` functionality. `RETURN` can be omitted when it is not needed to provide the function's return value.

- Add support for an optional `INTO` clause to PL/pgSQL's `EXECUTE` statement (Pavel Stehule, Neil)

- Make `CREATE TABLE AS` set `ROW_COUNT` (Tom)

- Define `SQLSTATE` and `SQLERRM` to return the `SQLSTATE` and error message of the current exception (Pavel Stehule, Neil)

  These variables are only defined inside exception blocks.

- Allow the parameters to the `RAISE` statement to be expressions (Pavel Stehule, Neil)

- Add a loop `CONTINUE` statement (Pavel Stehule, Neil)

- Allow block and loop labels (Pavel Stehule)

E.169.3.10. PL/Perl Server-Side Language Changes

- Allow large result sets to be returned efficiently (Abhijit Menon-Sen)

  This allows functions to use `return_next()` to avoid building the entire result set in memory.

- Allow one-row-at-a-time retrieval of query results (Abhijit Menon-Sen)

  This allows functions to use `spi_query()` and `spi_fetchrow()` to avoid accumulating the entire result set in memory.

- Force PL/Perl to handle strings as `UTF8` if the server encoding is `UTF8` (David Kamholz)

- Add a validator function for PL/Perl (Andrew)

  This allows syntax errors to be reported at definition time, rather than execution time.

- Allow PL/Perl to return a Perl array when the function returns an array type (Andrew)

  This basically maps PostgreSQL arrays to Perl arrays.

- Allow Perl nonfatal warnings to generate `NOTICE` messages (Andrew)

- Allow Perl's `strict` mode to be enabled (Andrew)

E.169.3.11. psql Changes

- Add `\set ON_ERROR_ROLLBACK` to allow statements in a transaction to error without affecting the rest of the transaction (Greg Sabino Mullane)

  This is basically implemented by wrapping every statement in a sub-transaction.

- Add support for `\x` hex strings in psql variables (Bruce)

  Octal escapes were already supported.

- Add support for `troff -ms` output format (Roger Leigh)

- Allow the history file location to be controlled by `HISTFILE` (Andreas Seltenreich)

  This allows configuration of per-database history storage.

- Prevent `\x` (expanded mode) from affecting the output of `\d tablename` (Neil)

- Add `-L` option to psql to log sessions (Lorne Sunley)

  This option was added because some operating systems do not have simple command-line activity logging functionality.

- Make `\d` show the tablespaces of indexes (Qingqing Zhou)

- Allow psql help (`\h`) to make a best guess on the proper help information (Greg Sabino Mullane)

  This allows the user to just add `\h` to the front of the syntax error query and get help on the supported syntax. Previously any additional query text beyond the command name had to be removed to use `\h`.

- Add `\pset numericlocale` to allow numbers to be output in a locale-aware format (Eugen Nedelcu)

  For example, using `C` locale `100000` would be output as `100,000.0` while a European locale might output this value as `100.000,0`.

- Make startup banner show both server version number and psql's version number, when they are different (Bruce)

  Also, a warning will be shown if the server and psql are from different major releases.

E.169.3.12. pg_dump Changes

- Add `-n` / `--schema` switch to pg_restore (Richard van den Berg)

  This allows just the objects in a specified schema to be restored.

- Allow pg_dump to dump large objects even in text mode (Tom)

  With this change, large objects are now always dumped; the former `-b` switch is a no-op.

- Allow pg_dump to dump a consistent snapshot of large objects (Tom)

- Dump comments for large objects (Tom)

- Add `--encoding` to pg_dump (Magnus Hagander)

This allows a database to be dumped in an encoding that is different from the server's encoding. This is valuable when transferring the dump to a machine with a different encoding.

- Rely on `pg_pltemplate` for procedural languages (Tom)

  If the call handler for a procedural language is in the `pg_catalog` schema, pg_dump does not dump the handler. Instead, it dumps the language using just `CREATE LANGUAGE name`, relying on the `pg_pltemplate` catalog to provide the language's creation parameters at load time.

E.169.3.13. libpq Changes

- Add a `PGPASSFILE` environment variable to specify the password file's filename (Andrew)
- Add `lo_create()`, that is similar to `lo_creat()` but allows the OID of the large object to be specified (Tom)
- Make libpq consistently return an error to the client application on `malloc()` failure (Neil)

E.169.3.14. Source Code Changes

- Fix pgxs to support building against a relocated installation
- Add spinlock support for the Itanium processor using Intel compiler (Vikram Kalsi)
- Add Kerberos 5 support for Windows (Magnus)
- Add Chinese FAQ (laser@pgsqldb.com)
- Rename Rendezvous to Bonjour to match OS/X feature renaming (Bruce)
- Add support for `fsync_writethrough` on Darwin (Chris Campbell)
- Streamline the passing of information within the server, the optimizer, and the lock system (Tom)
- Allow pg_config to be compiled using MSVC (Andrew)

  This is required to build DBD::Pg using MSVC.

- Remove support for Kerberos V4 (Magnus)

  Kerberos 4 had security vulnerabilities and is no longer maintained.

- Code cleanups (Coverity static analysis performed by EnterpriseDB)
- Modify `postgresql.conf` to use documentation defaults `on`/`off` rather than `true`/`false` (Bruce)
- Enhance pg_config to be able to report more build-time values (Tom)
- Allow libpq to be built thread-safe on Windows (Dave Page)
- Allow IPv6 connections to be used on Windows (Andrew)
- Add Server Administration documentation about I/O subsystem reliability (Bruce)
- Move private declarations from `gist.h` to `gist_private.h` (Neil)

In previous releases, `gist.h` contained both the public GiST API (intended for use by authors of GiST index implementations) as well as some private declarations used by the implementation of GiST itself. The latter have been moved to a separate file, `gist_private.h`. Most GiST index implementations should be unaffected.

- Overhaul GiST memory management (Neil)

  GiST methods are now always invoked in a short-lived memory context. Therefore, memory allocated via `palloc()` will be reclaimed automatically, so GiST index implementations do not need to manually release allocated memory via `pfree()`.

E.169.3.15. Contrib Changes

- Add `/contrib/pg_buffercache` contrib module (Mark Kirkwood)

  This displays the contents of the buffer cache, for debugging and performance tuning purposes.

- Remove `/contrib/array` because it is obsolete (Tom)
- Clean up the `/contrib/lo` module (Tom)
- Move `/contrib/findoidjoins` to `/src/tools` (Tom)
- Remove the `<<`, `>>`, `&<`, and `&>` operators from `/contrib/cube`

  These operators were not useful.

- Improve `/contrib/btree_gist` (Janko Richter)
- Improve `/contrib/pgbench` (Tomoaki Sato, Tatsuo)

  There is now a facility for testing with SQL command scripts given by the user, instead of only a hard-wired command sequence.

- Improve `/contrib/pgcrypto` (Marko Kreen)
  - Implementation of OpenPGP symmetric-key and public-key encryption

    Both RSA and Elgamal public-key algorithms are supported.

  - Stand alone build: include SHA256/384/512 hashes, Fortuna PRNG
  - OpenSSL build: support 3DES, use internal AES with OpenSSL < 0.9.7
  - Take build parameters (OpenSSL, zlib) from `configure` result

    There is no need to edit the `Makefile` anymore.

  - Remove support for `libmhash` and `libmcrypt`

E.170. Release 8.0.26

Release date: 2010-10-04

This release contains a variety of fixes from 8.0.25. For information about new features in the 8.0 major release, see 第 E.196 节.

This is expected to be the last PostgreSQL release in the 8.0.X series. Users are encouraged to update to a newer release branch soon.

E.170.1. Migration to Version 8.0.26

A dump/restore is not required for those running 8.0.X. However, if you are upgrading from a version earlier than 8.0.22, see 第 E.174 节.

E.170.2. Changes

- Use a separate interpreter for each calling SQL userid in PL/Perl and PL/Tcl (Tom Lane)

  This change prevents security problems that can be caused by subverting Perl or Tcl code that will be executed later in the same session under another SQL user identity (for example, within a `SECURITY DEFINER` function). Most scripting languages offer numerous ways that that might be done, such as redefining standard functions or operators called by the target function. Without this change, any SQL user with Perl or Tcl language usage rights can do essentially anything with the SQL privileges of the target function's owner.

  The cost of this change is that intentional communication among Perl and Tcl functions becomes more difficult. To provide an escape hatch, PL/PerlU and PL/TclU functions continue to use only one interpreter per session. This is not considered a security issue since all such functions execute at the trust level of a database superuser already.

  It is likely that third-party procedural languages that claim to offer trusted execution have similar security issues. We advise contacting the authors of any PL you are depending on for security-critical purposes.

  Our thanks to Tim Bunce for pointing out this issue (CVE-2010-3433).

- Prevent possible crashes in `pg_get_expr()` by disallowing it from being called with an argument that is not one of the system catalog columns it's intended to be used with (Heikki Linnakangas, Tom Lane)

- Fix “cannot handle unplanned sub-select” error (Tom Lane)

  This occurred when a sub-select contains a join alias reference that expands into an expression containing another sub-select.

- Defend against functions returning setof record where not all the returned rows are actually of the same rowtype (Tom Lane)

- Take care to fsync the contents of lockfiles (both `postmaster.pid` and the socket lockfile) while writing them (Tom Lane)

  This omission could result in corrupted lockfile contents if the machine crashes shortly after postmaster start. That could in turn prevent subsequent attempts to start the postmaster from succeeding, until the lockfile is manually removed.

- Avoid recursion while assigning XIDs to heavily-nested subtransactions (Andres Freund, Robert Haas)

  The original coding could result in a crash if there was limited stack space.

- Fix `log_line_prefix`'s `%i` escape, which could produce junk early in backend startup (Tom Lane)

- Fix possible data corruption in `ALTER TABLE ... SET TABLESPACE` when archiving is enabled (Jeff Davis)

- Allow `CREATE DATABASE` and `ALTER DATABASE ... SET TABLESPACE` to be interrupted by query-cancel (Guillaume Lelarge)
- In PL/Python, defend against null pointer results from `PyCObject_AsVoidPtr` and `PyCObject_FromVoidPtr` (Peter Eisentraut)
- Improve `contrib/dblink`'s handling of tables containing dropped columns (Tom Lane)
- Fix connection leak after "duplicate connection name" errors in `contrib/dblink` (Itagaki Takahiro)
- Fix `contrib/dblink` to handle connection names longer than 62 bytes correctly (Itagaki Takahiro)
- Update build infrastructure and documentation to reflect the source code repository's move from CVS to Git (Magnus Hagander and others)
- Update time zone data files to tzdata release 2010l for DST law changes in Egypt and Palestine; also historical corrections for Finland.

  This change also adds new names for two Micronesian timezones: Pacific/Chuuk is now preferred over Pacific/Truk (and the preferred abbreviation is CHUT not TRUT) and Pacific/Pohnpei is preferred over Pacific/Ponape.

E.171. Release 8.0.25

Release date: 2010-05-17

This release contains a variety of fixes from 8.0.24. For information about new features in the 8.0 major release, see 第 E.196 节.

The PostgreSQL community will stop releasing updates for the 8.0.X release series in July 2010. Users are encouraged to update to a newer release branch soon.

E.171.1. Migration to Version 8.0.25

A dump/restore is not required for those running 8.0.X. However, if you are upgrading from a version earlier than 8.0.22, see 第 E.174 节.

E.171.2. Changes

- Enforce restrictions in `plperl` using an opmask applied to the whole interpreter, instead of using `Safe.pm` (Tim Bunce, Andrew Dunstan)

  Recent developments have convinced us that `Safe.pm` is too insecure to rely on for making `plperl` trustable. This change removes use of `Safe.pm` altogether, in favor of using a separate interpreter with an opcode mask that is always applied. Pleasant side effects of the change include that it is now possible to use Perl's `strict` pragma in a natural way in `plperl`, and that Perl's `$a` and `$b` variables work as expected in sort routines, and that function compilation is significantly faster. (CVE-2010-1169)

- Prevent PL/Tcl from executing untrustworthy code from `pltcl_modules` (Tom)

  PL/Tcl's feature for autoloading Tcl code from a database table could be exploited for trojan-horse attacks, because there was no restriction on who could create or insert into that table. This change disables the feature unless `pltcl_modules` is owned by a superuser. (However, the permissions on the table are not checked,

so installations that really need a less-than-secure modules table can still grant suitable privileges to trusted non-superusers.) Also, prevent loading code into the unrestricted "normal" Tcl interpreter unless we are really going to execute a `pltclu` function. (CVE-2010-1170)

- Do not allow an unprivileged user to reset superuser-only parameter settings (Alvaro)

  Previously, if an unprivileged user ran `ALTER USER ... RESET ALL` for himself, or `ALTER DATABASE ... RESET ALL` for a database he owns, this would remove all special parameter settings for the user or database, even ones that are only supposed to be changeable by a superuser. Now, the `ALTER` will only remove the parameters that the user has permission to change.

- Avoid possible crash during backend shutdown if shutdown occurs when a `CONTEXT` addition would be made to log entries (Tom)

  In some cases the context-printing function would fail because the current transaction had already been rolled back when it came time to print a log message.

- Update pl/perl's `ppport.h` for modern Perl versions (Andrew)

- Fix assorted memory leaks in pl/python (Andreas Freund, Tom)

- Prevent infinite recursion in psql when expanding a variable that refers to itself (Tom)

- Ensure that `contrib/pgstattuple` functions respond to cancel interrupts promptly (Tatsuhito Kasahara)

- Make server startup deal properly with the case that `shmget()` returns `EINVAL` for an existing shared memory segment (Tom)

  This behavior has been observed on BSD-derived kernels including OS X. It resulted in an entirely-misleading startup failure complaining that the shared memory request size was too large.

- Update time zone data files to tzdata release 2010j for DST law changes in Argentina, Australian Antarctic, Bangladesh, Mexico, Morocco, Pakistan, Palestine, Russia, Syria, Tunisia; also historical corrections for Taiwan.

E.172. Release 8.0.24

Release date: 2010-03-15

This release contains a variety of fixes from 8.0.23. For information about new features in the 8.0 major release, see 第 E.196 节.

The PostgreSQL community will stop releasing updates for the 8.0.X release series in July 2010. Users are encouraged to update to a newer release branch soon.

E.172.1. Migration to Version 8.0.24

A dump/restore is not required for those running 8.0.X. However, if you are upgrading from a version earlier than 8.0.22, see 第 E.174 节.

E.172.2. Changes

- Add new configuration parameter `ssl_renegotiation_limit` to control how often we do session key renegotiation for an SSL connection (Magnus)

This can be set to zero to disable renegotiation completely, which may be required if a broken SSL library is used. In particular, some vendors are shipping stopgap patches for CVE-2009-3555 that cause renegotiation attempts to fail.

- Fix possible crashes when trying to recover from a failure in subtransaction start (Tom)
- Fix server memory leak associated with use of savepoints and a client encoding different from server's encoding (Tom)
- Make `substring()` for `bit` types treat any negative length as meaning "all the rest of the string" (Tom)

  The previous coding treated only -1 that way, and would produce an invalid result value for other negative values, possibly leading to a crash (CVE-2010-0442).
- Fix integer-to-bit-string conversions to handle the first fractional byte correctly when the output bit width is wider than the given integer by something other than a multiple of 8 bits (Tom)
- Fix some cases of pathologically slow regular expression matching (Tom)
- Fix the `STOP WAL LOCATION` entry in backup history files to report the next WAL segment's name when the end location is exactly at a segment boundary (Itagaki Takahiro)
- When reading `pg_hba.conf` and related files, do not treat `@something` as a file inclusion request if the `@` appears inside quote marks; also, never treat `@` by itself as a file inclusion request (Tom)

  This prevents erratic behavior if a role or database name starts with `@`. If you need to include a file whose path name contains spaces, you can still do so, but you must write `@"/path to/file"` rather than putting the quotes around the whole construct.
- Prevent infinite loop on some platforms if a directory is named as an inclusion target in `pg_hba.conf` and related files (Tom)
- Fix plpgsql failure in one case where a composite column is set to NULL (Tom)
- Add `volatile` markings in PL/Python to avoid possible compiler-specific misbehavior (Zdenek Kotala)
- Ensure PL/Tcl initializes the Tcl interpreter fully (Tom)

  The only known symptom of this oversight is that the Tcl `clock` command misbehaves if using Tcl 8.5 or later.
- Prevent crash in `contrib/dblink` when too many key columns are specified to a `dblink_build_sql_*` function (Rushabh Lathia, Joe Conway)
- Fix assorted crashes in `contrib/xml2` caused by sloppy memory management (Tom)
- Update time zone data files to tzdata release 2010e for DST law changes in Bangladesh, Chile, Fiji, Mexico, Paraguay, Samoa.

E.173. Release 8.0.23

Release date: 2009-12-14

This release contains a variety of fixes from 8.0.22. For information about new features in the 8.0 major release, see 第 E.196 节.

E.173.1. Migration to Version 8.0.23

A dump/restore is not required for those running 8.0.X. However, if you are upgrading from a version earlier than 8.0.22, see 第 E.174 节.

E.173.2. Changes

- Protect against indirect security threats caused by index functions changing session-local state (Gurjeet Singh, Tom)

  This change prevents allegedly-immutable index functions from possibly subverting a superuser's session (CVE-2009-4136).

- Reject SSL certificates containing an embedded null byte in the common name (CN) field (Magnus)

  This prevents unintended matching of a certificate to a server or client name during SSL validation (CVE-2009-4034).

- Fix possible crash during backend-startup-time cache initialization (Tom)

- Prevent signals from interrupting `VACUUM` at unsafe times (Alvaro)

  This fix prevents a PANIC if a `VACUUM FULL` is canceled after it's already committed its tuple movements, as well as transient errors if a plain `VACUUM` is interrupted after having truncated the table.

- Fix possible crash due to integer overflow in hash table size calculation (Tom)

  This could occur with extremely large planner estimates for the size of a hashjoin's result.

- Fix very rare crash in `inet/cidr` comparisons (Chris Mikkelson)

- Fix premature drop of temporary files used for a cursor that is accessed within a subtransaction (Heikki)

- Fix PAM password processing to be more robust (Tom)

  The previous code is known to fail with the combination of the Linux `pam_krb5` PAM module with Microsoft Active Directory as the domain controller. It might have problems elsewhere too, since it was making unjustified assumptions about what arguments the PAM stack would pass to it.

- Fix rare crash in exception processing in PL/Python (Peter)

- Ensure psql's flex module is compiled with the correct system header definitions (Tom)

  This fixes build failures on platforms where `--enable-largefile` causes incompatible changes in the generated code.

- Make the postmaster ignore any `application_name` parameter in connection request packets, to improve compatibility with future libpq versions (Tom)

- Update time zone data files to tzdata release 2009s for DST law changes in Antarctica, Argentina, Bangladesh, Fiji, Novokuznetsk, Pakistan, Palestine, Samoa, Syria; also historical corrections for Hong Kong.

E.174. Release 8.0.22

Release date: 2009-09-09

This release contains a variety of fixes from 8.0.21. For information about new features in the 8.0 major release, see 第 E.196 节.

E.174.1. Migration to Version 8.0.22

A dump/restore is not required for those running 8.0.X. However, if you have any hash indexes on `interval` columns, you must `REINDEX` them after updating to 8.0.22. Also, if you are upgrading from a version earlier than 8.0.6, see 第 E.190 节.

E.174.2. Changes

- Disallow `RESET ROLE` and `RESET SESSION AUTHORIZATION` inside security-definer functions (Tom, Heikki)

  This covers a case that was missed in the previous patch that disallowed `SET ROLE` and `SET SESSION AUTHORIZATION` inside security-definer functions. (See CVE-2007-6600)

- Fix handling of sub-SELECTs appearing in the arguments of an outer-level aggregate function (Tom)

- Fix hash calculation for data type `interval` (Tom)

  This corrects wrong results for hash joins on interval values. It also changes the contents of hash indexes on interval columns. If you have any such indexes, you must `REINDEX` them after updating.

- Treat `to_char(..., 'TH')` as an uppercase ordinal suffix with `'HH'/'HH12'` (Heikki)

  It was previously handled as `'th'` (lowercase).

- Fix overflow for `INTERVAL 'x ms'` when *x* is more than 2 million and integer datetimes are in use (Alex Hunsaker)

- Fix calculation of distance between a point and a line segment (Tom)

  This led to incorrect results from a number of geometric operators.

- Fix `money` data type to work in locales where currency amounts have no fractional digits, e.g. Japan (Itagaki Takahiro)

- Properly round datetime input like `00:12:57.999999999999999999999999999` (Tom)

- Fix poor choice of page split point in GiST R-tree operator classes (Teodor)

- Fix portability issues in plperl initialization (Andrew Dunstan)

- Fix pg_ctl to not go into an infinite loop if `postgresql.conf` is empty (Jeff Davis)

- Fix `contrib/xml2`'s `xslt_process()` to properly handle the maximum number of parameters (twenty) (Tom)

- Improve robustness of libpq's code to recover from errors during `COPY FROM STDIN` (Tom)
- Avoid including conflicting readline and editline header files when both libraries are installed (Zdenek Kotala)
- Update time zone data files to tzdata release 2009l for DST law changes in Bangladesh, Egypt, Jordan, Pakistan, Argentina/San_Luis, Cuba, Jordan (historical correction only), Mauritius, Morocco, Palestine, Syria, Tunisia.

E.175. Release 8.0.21

Release date: 2009-03-16

This release contains a variety of fixes from 8.0.20. For information about new features in the 8.0 major release, see 第 E.196 节.

E.175.1. Migration to Version 8.0.21

A dump/restore is not required for those running 8.0.X. However, if you are upgrading from a version earlier than 8.0.6, see 第 E.190 节.

E.175.2. Changes

- Prevent error recursion crashes when encoding conversion fails (Tom)

  This change extends fixes made in the last two minor releases for related failure scenarios. The previous fixes were narrowly tailored for the original problem reports, but we have now recognized that any error thrown by an encoding conversion function could potentially lead to infinite recursion while trying to report the error. The solution therefore is to disable translation and encoding conversion and report the plain-ASCII form of any error message, if we find we have gotten into a recursive error reporting situation. (CVE-2009-0922)

- Disallow `CREATE CONVERSION` with the wrong encodings for the specified conversion function (Heikki)

  This prevents one possible scenario for encoding conversion failure. The previous change is a backstop to guard against other kinds of failures in the same area.

- Fix core dump when `to_char()` is given format codes that are inappropriate for the type of the data argument (Tom)
- Add `MUST` (Mauritius Island Summer Time) to the default list of known timezone abbreviations (Xavier Bugaud)

E.176. Release 8.0.20

Release date: 2009-02-02

This release contains a variety of fixes from 8.0.19. For information about new features in the 8.0 major release, see 第 E.196 节.

E.176.1. Migration to Version 8.0.20

A dump/restore is not required for those running 8.0.X. However, if you are upgrading from a version earlier than 8.0.6, see 第 E.190 节.

E.176.2. Changes

- Improve handling of URLs in `headline()` function (Teodor)
- Improve handling of overlength headlines in `headline()` function (Teodor)
- Prevent possible Assert failure or misconversion if an encoding conversion is created with the wrong conversion function for the specified pair of encodings (Tom, Heikki)
- Avoid unnecessary locking of small tables in `VACUUM` (Heikki)
- Fix uninitialized variables in `contrib/tsearch2`'s `get_covers()` function (Teodor)
- Make all documentation reference `pgsql-bugs` and/or `pgsql-hackers` as appropriate, instead of the now-decommissioned `pgsql-ports` and `pgsql-patches` mailing lists (Tom)
- Update time zone data files to tzdata release 2009a (for Kathmandu and historical DST corrections in Switzerland, Cuba)

E.177. Release 8.0.19

Release date: 2008-11-03

This release contains a variety of fixes from 8.0.18. For information about new features in the 8.0 major release, see 第 E.196 节.

E.177.1. Migration to Version 8.0.19

A dump/restore is not required for those running 8.0.X. However, if you are upgrading from a version earlier than 8.0.6, see 第 E.190 节.

E.177.2. Changes

- Fix backend crash when the client encoding cannot represent a localized error message (Tom)

  We have addressed similar issues before, but it would still fail if the "character has no equivalent" message itself couldn't be converted. The fix is to disable localization and send the plain ASCII error message when we detect such a situation.
- Fix possible crash when deeply nested functions are invoked from a trigger (Tom)
- Ensure an error is reported when a newly-defined PL/pgSQL trigger function is invoked as a normal function (Tom)
- Fix incorrect tsearch2 headline generation when single query item matches first word of text (Sushant Sinha)
- Fix improper display of fractional seconds in interval values when using a non-ISO datestyle in an `--enable-integer-datetimes` build (Ron Mayer)
- Ensure `SPI_getvalue` and `SPI_getbinval` behave correctly when the passed tuple and tuple descriptor have different numbers of columns (Tom)

  This situation is normal when a table has had columns added or removed, but these two functions didn't handle it properly. The only likely consequence is an incorrect error indication.

- Fix ecpg's parsing of `CREATE USER` (Michael)
- Fix recent breakage of `pg_ctl restart` (Tom)
- Update time zone data files to tzdata release 2008i (for DST law changes in Argentina, Brazil, Mauritius, Syria)

E.178. Release 8.0.18

Release date: 2008-09-22

This release contains a variety of fixes from 8.0.17. For information about new features in the 8.0 major release, see 第 E.196 节.

E.178.1. Migration to Version 8.0.18

A dump/restore is not required for those running 8.0.X. However, if you are upgrading from a version earlier than 8.0.6, see 第 E.190 节.

E.178.2. Changes

- Widen local lock counters from 32 to 64 bits (Tom)

  This responds to reports that the counters could overflow in sufficiently long transactions, leading to unexpected "lock is already held" errors.

- Add checks in executor startup to ensure that the tuples produced by an `INSERT` or `UPDATE` will match the target table's current rowtype (Tom)

  `ALTER COLUMN TYPE`, followed by re-use of a previously cached plan, could produce this type of situation. The check protects against data corruption and/or crashes that could ensue.

- Fix datetime input functions to correctly detect integer overflow when running on a 64-bit platform (Tom)
- Improve performance of writing very long log messages to syslog (Tom)
- Fix bug in backwards scanning of a cursor on a `SELECT DISTINCT ON` query (Tom)
- Fix planner to estimate that `GROUP BY` expressions yielding boolean results always result in two groups, regardless of the expressions' contents (Tom)

  This is very substantially more accurate than the regular `GROUP BY` estimate for certain boolean tests like *col* `IS NULL`.

- Fix PL/Tcl to behave correctly with Tcl 8.5, and to be more careful about the encoding of data sent to or from Tcl (Tom)
- Fix PL/Python to work with Python 2.5

  This is a back-port of fixes made during the 8.2 development cycle.

- Improve pg_dump and pg_restore's error reporting after failure to send a SQL command (Tom)
- Fix pg_ctl to properly preserve postmaster command-line arguments across a `restart` (Bruce)

- Update time zone data files to tzdata release 2008f (for DST law changes in Argentina, Bahamas, Brazil, Mauritius, Morocco, Pakistan, Palestine, and Paraguay)

E.179. Release 8.0.17

Release date: 2008-06-12

This release contains one serious bug fix over 8.0.16. For information about new features in the 8.0 major release, see 第 E.196 节.

E.179.1. Migration to Version 8.0.17

A dump/restore is not required for those running 8.0.X. However, if you are upgrading from a version earlier than 8.0.6, see 第 E.190 节.

E.179.2. Changes

- Make `pg_get_ruledef()` parenthesize negative constants (Tom)

  Before this fix, a negative constant in a view or rule might be dumped as, say, `-42::integer`, which is subtly incorrect: it should be `(-42)::integer` due to operator precedence rules. Usually this would make little difference, but it could interact with another recent patch to cause PostgreSQL to reject what had been a valid `SELECT DISTINCT` view query. Since this could result in pg_dump output failing to reload, it is being treated as a high-priority fix. The only released versions in which dump output is actually incorrect are 8.3.1 and 8.2.7.

E.180. Release 8.0.16

Release date: never released

This release contains a variety of fixes from 8.0.15. For information about new features in the 8.0 major release, see 第 E.196 节.

E.180.1. Migration to Version 8.0.16

A dump/restore is not required for those running 8.0.X. However, if you are upgrading from a version earlier than 8.0.6, see 第 E.190 节.

E.180.2. Changes

- Fix `ALTER TABLE ADD COLUMN ... PRIMARY KEY` so that the new column is correctly checked to see if it's been initialized to all non-nulls (Brendan Jurd)

  Previous versions neglected to check this requirement at all.

- Fix possible `CREATE TABLE` failure when inheriting the "same" constraint from multiple parent relations that inherited that constraint from a common ancestor (Tom)

- Fix conversions between ISO-8859-5 and other encodings to handle Cyrillic "Yo" characters (`e` and `E` with two dots) (Sergey Burladyan)

- Fix a few datatype input functions that were allowing unused bytes in their results to contain uninitialized, unpredictable values (Tom)

This could lead to failures in which two apparently identical literal values were not seen as equal, resulting in the parser complaining about unmatched `ORDER BY` and `DISTINCT` expressions.

- Fix a corner case in regular-expression substring matching (`substring(string from pattern)`) (Tom)

  The problem occurs when there is a match to the pattern overall but the user has specified a parenthesized subexpression and that subexpression hasn't got a match. An example is `substring('foo' from 'foo(bar)?')`. This should return NULL, since `(bar)` isn't matched, but it was mistakenly returning the whole-pattern match instead (ie, `foo`).

- Update time zone data files to tzdata release 2008c (for DST law changes in Morocco, Iraq, Choibalsan, Pakistan, Syria, Cuba, Argentina/San_Luis, and Chile)

- Fix incorrect result from ecpg's `PGTYPEStimestamp_sub()` function (Michael)

- Fix core dump in `contrib/xml2`'s `xpath_table()` function when the input query returns a NULL value (Tom)

- Fix `contrib/xml2`'s makefile to not override `CFLAGS` (Tom)

- Fix `DatumGetBool` macro to not fail with gcc 4.3 (Tom)

  This problem affects "old style" (V0) C functions that return boolean. The fix is already in 8.3, but the need to back-patch it was not realized at the time.

- Fix longstanding `LISTEN/NOTIFY` race condition (Tom)

  In rare cases a session that had just executed a `LISTEN` might not get a notification, even though one would be expected because the concurrent transaction executing `NOTIFY` was observed to commit later.

  A side effect of the fix is that a transaction that has executed a not-yet-committed `LISTEN` command will not see any row in `pg_listener` for the `LISTEN`, should it choose to look; formerly it would have. This behavior was never documented one way or the other, but it is possible that some applications depend on the old behavior.

- Fix rare crash when an error occurs during a query using a hash index (Heikki)

- Fix input of datetime values for February 29 in years BC (Tom)

  The former coding was mistaken about which years were leap years.

- Fix "unrecognized node type" error in some variants of `ALTER OWNER` (Tom)

- Fix pg_ctl to correctly extract the postmaster's port number from command-line options (Itagaki Takahiro, Tom)

  Previously, `pg_ctl start -w` could try to contact the postmaster on the wrong port, leading to bogus reports of startup failure.

- Use `-fwrapv` to defend against possible misoptimization in recent gcc versions (Tom)

  This is known to be necessary when building PostgreSQL with gcc 4.3 or later.

- Fix display of constant expressions in `ORDER BY` and `GROUP BY` (Tom)

An explicitly casted constant would be shown incorrectly. This could for example lead to corruption of a view definition during dump and reload.

- Fix libpq to handle NOTICE messages correctly during COPY OUT (Tom)

  This failure has only been observed to occur when a user-defined datatype's output routine issues a NOTICE, but there is no guarantee it couldn't happen due to other causes.

E.181. Release 8.0.15

Release date: 2008-01-07

This release contains a variety of fixes from 8.0.14, including fixes for significant security issues. For information about new features in the 8.0 major release, see 第 E.196 节.

This is the last 8.0.X release for which the PostgreSQL community will produce binary packages for Windows. Windows users are encouraged to move to 8.2.X or later, since there are Windows-specific fixes in 8.2.X that are impractical to back-port. 8.0.X will continue to be supported on other platforms.

E.181.1. Migration to Version 8.0.15

A dump/restore is not required for those running 8.0.X. However, if you are upgrading from a version earlier than 8.0.6, see 第 E.190 节.

E.181.2. Changes

- Prevent functions in indexes from executing with the privileges of the user running `VACUUM`, `ANALYZE`, etc (Tom)

  Functions used in index expressions and partial-index predicates are evaluated whenever a new table entry is made. It has long been understood that this poses a risk of trojan-horse code execution if one modifies a table owned by an untrustworthy user. (Note that triggers, defaults, check constraints, etc. pose the same type of risk.) But functions in indexes pose extra danger because they will be executed by routine maintenance operations such as `VACUUM FULL`, which are commonly performed automatically under a superuser account. For example, a nefarious user can execute code with superuser privileges by setting up a trojan-horse index definition and waiting for the next routine vacuum. The fix arranges for standard maintenance operations (including `VACUUM`, `ANALYZE`, `REINDEX`, and `CLUSTER`) to execute as the table owner rather than the calling user, using the same privilege-switching mechanism already used for `SECURITY DEFINER` functions. To prevent bypassing this security measure, execution of `SET SESSION AUTHORIZATION` and `SET ROLE` is now forbidden within a `SECURITY DEFINER` context. (CVE-2007-6600)

- Repair assorted bugs in the regular-expression package (Tom, Will Drewry)

  Suitably crafted regular-expression patterns could cause crashes, infinite or near-infinite looping, and/or massive memory consumption, all of which pose denial-of-service hazards for applications that accept regex search patterns from untrustworthy sources. (CVE-2007-4769, CVE-2007-4772, CVE-2007-6067)

- Require non-superusers who use `/contrib/dblink` to use only password authentication, as a security measure (Joe)

The fix that appeared for this in 8.0.14 was incomplete, as it plugged the hole for only some `dblink` functions. (CVE-2007-6601, CVE-2007-3278)

- Update time zone data files to tzdata release 2007k (in particular, recent Argentina changes) (Tom)
- Fix planner failure in some cases of `WHERE false AND var IN (SELECT ...)` (Tom)
- Preserve the tablespace of indexes that are rebuilt by `ALTER TABLE ... ALTER COLUMN TYPE` (Tom)
- Make archive recovery always start a new WAL timeline, rather than only when a recovery stop time was used (Simon)

  This avoids a corner-case risk of trying to overwrite an existing archived copy of the last WAL segment, and seems simpler and cleaner than the original definition.
- Make `VACUUM` not use all of `maintenance_work_mem` when the table is too small for it to be useful (Alvaro)
- Fix potential crash in `translate()` when using a multibyte database encoding (Tom)
- Fix PL/Perl to cope when platform's Perl defines type `bool` as `int` rather than `char` (Tom)

  While this could theoretically happen anywhere, no standard build of Perl did things this way ... until Mac OS X 10.5.
- Fix PL/Python to not crash on long exception messages (Alvaro)
- Fix pg_dump to correctly handle inheritance child tables that have default expressions different from their parent's (Tom)
- ecpg parser fixes (Michael)
- Make `contrib/tablefunc`'s `crosstab()` handle NULL rowid as a category in its own right, rather than crashing (Joe)
- Fix `tsvector` and `tsquery` output routines to escape backslashes correctly (Teodor, Bruce)
- Fix crash of `to_tsvector()` on huge input strings (Teodor)
- Require a specific version of Autoconf to be used when re-generating the `configure` script (Peter)

  This affects developers and packagers only. The change was made to prevent accidental use of untested combinations of Autoconf and PostgreSQL versions. You can remove the version check if you really want to use a different Autoconf version, but it's your responsibility whether the result works or not.

E.182. Release 8.0.14

Release date: 2007-09-17

This release contains a variety of fixes from 8.0.13. For information about new features in the 8.0 major release, see 第 E.196 节.

E.182.1. Migration to Version 8.0.14

A dump/restore is not required for those running 8.0.X. However, if you are upgrading from a version earlier than 8.0.6, see 第 E.190 节.

E.182.2. Changes

- Prevent index corruption when a transaction inserts rows and then aborts close to the end of a concurrent `VACUUM` on the same table (Tom)
- Make `CREATE DOMAIN ... DEFAULT NULL` work properly (Tom)
- Fix excessive logging of SSL error messages (Tom)
- Fix logging so that log messages are never interleaved when using the syslogger process (Andrew)
- Fix crash when `log_min_error_statement` logging runs out of memory (Tom)
- Fix incorrect handling of some foreign-key corner cases (Tom)
- Prevent `CLUSTER` from failing due to attempting to process temporary tables of other sessions (Alvaro)
- Update the time zone database rules, particularly New Zealand's upcoming changes (Tom)
- Windows socket improvements (Magnus)
- Suppress timezone name (`%Z`) in log timestamps on Windows because of possible encoding mismatches (Tom)
- Require non-superusers who use `/contrib/dblink` to use only password authentication, as a security measure (Joe)

E.183. Release 8.0.13

Release date: 2007-04-23

This release contains a variety of fixes from 8.0.12, including a security fix. For information about new features in the 8.0 major release, see 第 E.196 节.

E.183.1. Migration to Version 8.0.13

A dump/restore is not required for those running 8.0.X. However, if you are upgrading from a version earlier than 8.0.6, see 第 E.190 节.

E.183.2. Changes

- Support explicit placement of the temporary-table schema within `search_path`, and disable searching it for functions and operators (Tom)

  This is needed to allow a security-definer function to set a truly secure value of `search_path`. Without it, an unprivileged SQL user can use temporary objects to execute code with the privileges of the security-definer function (CVE-2007-2138). See `CREATE FUNCTION` for more information.

- `/contrib/tsearch2` crash fixes (Teodor)
- Fix potential-data-corruption bug in how `VACUUM FULL` handles `UPDATE` chains (Tom, Pavan Deolasee)
- Fix PANIC during enlargement of a hash index (bug introduced in 8.0.10) (Tom)
- Fix POSIX-style timezone specs to follow new USA DST rules (Tom)

E.184. Release 8.0.12

Release date: 2007-02-07

This release contains one fix from 8.0.11. For information about new features in the 8.0 major release, see 第 E.196 节.

E.184.1. Migration to Version 8.0.12

A dump/restore is not required for those running 8.0.X. However, if you are upgrading from a version earlier than 8.0.6, see 第 E.190 节.

E.184.2. Changes

- Remove overly-restrictive check for type length in constraints and functional indexes(Tom)

E.185. Release 8.0.11

Release date: 2007-02-05

This release contains a variety of fixes from 8.0.10, including a security fix. For information about new features in the 8.0 major release, see 第 E.196 节.

E.185.1. Migration to Version 8.0.11

A dump/restore is not required for those running 8.0.X. However, if you are upgrading from a version earlier than 8.0.6, see 第 E.190 节.

E.185.2. Changes

- Remove security vulnerabilities that allowed connected users to read backend memory (Tom)

  The vulnerabilities involve suppressing the normal check that a SQL function returns the data type it's declared to, and changing the data type of a table column (CVE-2007-0555, CVE-2007-0556). These errors can easily be exploited to cause a backend crash, and in principle might be used to read database content that the user should not be able to access.

- Fix rare bug wherein btree index page splits could fail due to choosing an infeasible split point (Heikki Linnakangas)
- Fix for rare Assert() crash triggered by `UNION` (Tom)
- Tighten security of multi-byte character processing for UTF8 sequences over three bytes long (Tom)

E.186. Release 8.0.10

Release date: 2007-01-08

This release contains a variety of fixes from 8.0.9. For information about new features in the 8.0 major release, see 第 E.196 节.

E.186.1. Migration to Version 8.0.10

A dump/restore is not required for those running 8.0.X. However, if you are upgrading from a version earlier than 8.0.6, see 第 E.190 节.

E.186.2. Changes

- Improve handling of `getaddrinfo()` on AIX (Tom)

  This fixes a problem with starting the statistics collector, among other things.

- Fix "failed to re-find parent key" errors in `VACUUM` (Tom)
- Fix race condition for truncation of a large relation across a gigabyte boundary by `VACUUM` (Tom)
- Fix bugs affecting multi-gigabyte hash indexes (Tom)
- Fix possible deadlock in Windows signal handling (Teodor)
- Fix error when constructing an `ARRAY[]` made up of multiple empty elements (Tom)
- Fix ecpg memory leak during connection (Michael)
- `to_number()` and `to_char(numeric)` are now `STABLE`, not `IMMUTABLE`, for new initdb installs (Tom)

  This is because `lc_numeric` can potentially change the output of these functions.

- Improve index usage of regular expressions that use parentheses (Tom)

  This improves psql `\d` performance also.

- Update timezone database

  This affects Australian and Canadian daylight-savings rules in particular.

E.187. Release 8.0.9

Release date: 2006-10-16

This release contains a variety of fixes from 8.0.8. For information about new features in the 8.0 major release, see 第 E.196 节.

E.187.1. Migration to Version 8.0.9

A dump/restore is not required for those running 8.0.X. However, if you are upgrading from a version earlier than 8.0.6, see 第 E.190 节.

E.187.2. Changes

- Fix crash when referencing `NEW` row values in rule WHERE expressions (Tom)

- Fix core dump when an untyped literal is taken as ANYARRAY
- Fix mishandling of AFTER triggers when query contains a SQL function returning multiple rows (Tom)
- Fix `ALTER TABLE ... TYPE` to recheck `NOT NULL` for `USING` clause (Tom)
- Fix `string_to_array()` to handle overlapping matches for the separator string

  For example, `string_to_array('123xx456xxx789', 'xx')`.
- Fix corner cases in pattern matching for psql's `\d` commands
- Fix index-corrupting bugs in /contrib/ltree (Teodor)
- Numerous robustness fixes in ecpg (Joachim Wieland)
- Fix backslash escaping in /contrib/dbmirror
- Fix instability of statistics collection on Win32 (Tom, Andrew)
- Fixes for AIX and Intel compilers (Tom)

E.188. Release 8.0.8

Release date: 2006-05-23

This release contains a variety of fixes from 8.0.7, including patches for extremely serious security issues. For information about new features in the 8.0 major release, see 第 E.196 节.

E.188.1. Migration to Version 8.0.8

A dump/restore is not required for those running 8.0.X. However, if you are upgrading from a version earlier than 8.0.6, see 第 E.190 节.

Full security against the SQL-injection attacks described in CVE-2006-2313 and CVE-2006-2314 might require changes in application code. If you have applications that embed untrustworthy strings into SQL commands, you should examine them as soon as possible to ensure that they are using recommended escaping techniques. In most cases, applications should be using subroutines provided by libraries or drivers (such as libpq's `PQescapeStringConn()`) to perform string escaping, rather than relying on ad hoc code to do it.

E.188.2. Changes

- Change the server to reject invalidly-encoded multibyte characters in all cases (Tatsuo, Tom)

  While PostgreSQL has been moving in this direction for some time, the checks are now applied uniformly to all encodings and all textual input, and are now always errors not merely warnings. This change defends against SQL-injection attacks of the type described in CVE-2006-2313.
- Reject unsafe uses of `\'` in string literals

  As a server-side defense against SQL-injection attacks of the type described in CVE-2006-2314, the server now only accepts `''` and not `\'` as a representation of

ASCII single quote in SQL string literals. By default, `\'` is rejected only when `client_encoding` is set to a client-only encoding (SJIS, BIG5, GBK, GB18030, or UHC), which is the scenario in which SQL injection is possible. A new configuration parameter `backslash_quote` is available to adjust this behavior when needed. Note that full security against CVE-2006-2314 might require client-side changes; the purpose of `backslash_quote` is in part to make it obvious that insecure clients are insecure.

- Modify libpq's string-escaping routines to be aware of encoding considerations and `standard_conforming_strings`

  This fixes libpq-using applications for the security issues described in CVE-2006-2313 and CVE-2006-2314, and also future-proofs them against the planned changeover to SQL-standard string literal syntax. Applications that use multiple PostgreSQL connections concurrently should migrate to `PQescapeStringConn()` and `PQescapeByteaConn()` to ensure that escaping is done correctly for the settings in use in each database connection. Applications that do string escaping "by hand" should be modified to rely on library routines instead.

- Fix some incorrect encoding conversion functions

  `win1251_to_iso`, `alt_to_iso`, `euc_tw_to_big5`, `euc_tw_to_mic`, `mic_to_euc_tw` were all broken to varying extents.

- Clean up stray remaining uses of `\'` in strings (Bruce, Jan)

- Fix bug that sometimes caused OR'd index scans to miss rows they should have returned

- Fix WAL replay for case where a btree index has been truncated

- Fix `SIMILAR TO` for patterns involving `|` (Tom)

- Fix `SELECT INTO` and `CREATE TABLE AS` to create tables in the default tablespace, not the base directory (Kris Jurka)

- Fix server to use custom DH SSL parameters correctly (Michael Fuhr)

- Fix for Bonjour on Intel Macs (Ashley Clark)

- Fix various minor memory leaks

- Fix problem with password prompting on some Win32 systems (Robert Kinberg)

E.189. Release 8.0.7

Release date: 2006-02-14

This release contains a variety of fixes from 8.0.6. For information about new features in the 8.0 major release, see 第 E.196 节.

E.189.1. Migration to Version 8.0.7

A dump/restore is not required for those running 8.0.X. However, if you are upgrading from a version earlier than 8.0.6, see 第 E.190 节.

E.189.2. Changes

- Fix potential crash in `SET SESSION AUTHORIZATION` (CVE-2006-0553)

An unprivileged user could crash the server process, resulting in momentary denial of service to other users, if the server has been compiled with Asserts enabled (which is not the default). Thanks to Akio Ishida for reporting this problem.

- Fix bug with row visibility logic in self-inserted rows (Tom)

  Under rare circumstances a row inserted by the current command could be seen as already valid, when it should not be. Repairs bug created in 8.0.4, 7.4.9, and 7.3.11 releases.

- Fix race condition that could lead to "file already exists" errors during pg_clog and pg_subtrans file creation (Tom)
- Fix cases that could lead to crashes if a cache-invalidation message arrives at just the wrong time (Tom)
- Properly check `DOMAIN` constraints for `UNKNOWN` parameters in prepared statements (Neil)
- Ensure `ALTER COLUMN TYPE` will process `FOREIGN KEY`, `UNIQUE`, and `PRIMARY KEY` constraints in the proper order (Nakano Yoshihisa)
- Fixes to allow restoring dumps that have cross-schema references to custom operators or operator classes (Tom)
- Allow pg_restore to continue properly after a `COPY` failure; formerly it tried to treat the remaining `COPY` data as SQL commands (Stephen Frost)
- Fix pg_ctl `unregister` crash when the data directory is not specified (Magnus)
- Fix ecpg crash on AMD64 and PPC (Neil)
- Recover properly if error occurs during argument passing in PL/python (Neil)
- Fix PL/perl's handling of locales on Win32 to match the backend (Andrew)
- Fix crash when `log_min_messages` is set to `DEBUG3` or above in `postgresql.conf` on Win32 (Bruce)
- Fix pgxs `-L` library path specification for Win32, Cygwin, OS X, AIX (Bruce)
- Check that SID is enabled while checking for Win32 admin privileges (Magnus)
- Properly reject out-of-range date inputs (Kris Jurka)
- Portability fix for testing presence of `finite` and `isinf` during configure (Tom)

E.190. Release 8.0.6

Release date: 2006-01-09

This release contains a variety of fixes from 8.0.5. For information about new features in the 8.0 major release, see 第 E.196 节.

E.190.1. Migration to Version 8.0.6

A dump/restore is not required for those running 8.0.X. However, if you are upgrading from a version earlier than 8.0.3, see 第 E.193 节. Also, you might need to `REINDEX`

indexes on textual columns after updating, if you are affected by the locale or plperl issues described below.

E.190.2. Changes

- Fix Windows code so that postmaster will continue rather than exit if there is no more room in ShmemBackendArray (Magnus)

  The previous behavior could lead to a denial-of-service situation if too many connection requests arrive close together. This applies only to the Windows port.

- Fix bug introduced in 8.0 that could allow ReadBuffer to return an already-used page as new, potentially causing loss of recently-committed data (Tom)

- Fix for protocol-level Describe messages issued outside a transaction or in a failed transaction (Tom)

- Fix character string comparison for locales that consider different character combinations as equal, such as Hungarian (Tom)

  This might require `REINDEX` to fix existing indexes on textual columns.

- Set locale environment variables during postmaster startup to ensure that plperl won't change the locale later

  This fixes a problem that occurred if the postmaster was started with environment variables specifying a different locale than what initdb had been told. Under these conditions, any use of plperl was likely to lead to corrupt indexes. You might need `REINDEX` to fix existing indexes on textual columns if this has happened to you.

- Allow more flexible relocation of installation directories (Tom)

  Previous releases supported relocation only if all installation directory paths were the same except for the last component.

- Fix longstanding bug in strpos() and regular expression handling in certain rarely used Asian multi-byte character sets (Tatsuo)

- Various fixes for functions returning `RECORD`s (Tom)

- Fix bug in `/contrib/pgcrypto` gen_salt, which caused it not to use all available salt space for MD5 and XDES algorithms (Marko Kreen, Solar Designer)

  Salts for Blowfish and standard DES are unaffected.

- Fix `/contrib/dblink` to throw an error, rather than crashing, when the number of columns specified is different from what's actually returned by the query (Joe)

E.191. Release 8.0.5

Release date: 2005-12-12

This release contains a variety of fixes from 8.0.4. For information about new features in the 8.0 major release, see 第 E.196 节.

E.191.1. Migration to Version 8.0.5

A dump/restore is not required for those running 8.0.X. However, if you are upgrading from a version earlier than 8.0.3, see 第 E.193 节.

E.191.2. Changes

- Fix race condition in transaction log management

  There was a narrow window in which an I/O operation could be initiated for the wrong page, leading to an Assert failure or data corruption.

- Fix bgwriter problems after recovering from errors (Tom)

  The background writer was found to leak buffer pins after write errors. While not fatal in itself, this might lead to mysterious blockages of later VACUUM commands.

- Prevent failure if client sends Bind protocol message when current transaction is already aborted

- `/contrib/ltree` fixes (Teodor)

- AIX and HPUX compile fixes (Tom)

- Retry file reads and writes after Windows NO_SYSTEM_RESOURCES error (Qingqing Zhou)

- Fix intermittent failure when `log_line_prefix` includes `%i`

- Fix psql performance issue with long scripts on Windows (Merlin Moncure)

- Fix missing updates of `pg_group` flat file

- Fix longstanding planning error for outer joins

  This bug sometimes caused a bogus error "RIGHT JOIN is only supported with merge-joinable join conditions".

- Postpone timezone initialization until after `postmaster.pid` is created

  This avoids confusing startup scripts that expect the pid file to appear quickly.

- Prevent core dump in pg_autovacuum when a table has been dropped

- Fix problems with whole-row references (`foo.*`) to subquery results

E.192. Release 8.0.4

Release date: 2005-10-04

This release contains a variety of fixes from 8.0.3. For information about new features in the 8.0 major release, see 第 E.196 节.

E.192.1. Migration to Version 8.0.4

A dump/restore is not required for those running 8.0.X. However, if you are upgrading from a version earlier than 8.0.3, see 第 E.193 节.

E.192.2. Changes

- Fix error that allowed `VACUUM` to remove `ctid` chains too soon, and add more checking in code that follows `ctid` links

  This fixes a long-standing problem that could cause crashes in very rare circumstances.

- Fix `CHAR()` to properly pad spaces to the specified length when using a multiple-byte character set (Yoshiyuki Asaba)

  In prior releases, the padding of `CHAR()` was incorrect because it only padded to the specified number of bytes without considering how many characters were stored.

- Force a checkpoint before committing `CREATE DATABASE`

  This should fix recent reports of "index is not a btree" failures when a crash occurs shortly after `CREATE DATABASE`.

- Fix the sense of the test for read-only transaction in `COPY`

  The code formerly prohibited `COPY TO`, where it should prohibit `COPY FROM`.

- Handle consecutive embedded newlines in `COPY` CSV-mode input

- Fix `date_trunc(week)` for dates near year end

- Fix planning problem with outer-join ON clauses that reference only the inner-side relation

- Further fixes for `x FULL JOIN y ON true` corner cases

- Fix overenthusiastic optimization of `x IN (SELECT DISTINCT ...)` and related cases

- Fix mis-planning of queries with small `LIMIT` values due to poorly thought out "fuzzy" cost comparison

- Make `array_in` and `array_recv` more paranoid about validating their OID parameter

- Fix missing rows in queries like `UPDATE a=... WHERE a...` with GiST index on column `a`

- Improve robustness of datetime parsing

- Improve checking for partially-written WAL pages

- Improve robustness of signal handling when SSL is enabled

- Improve MIPS and M68K spinlock code

- Don't try to open more than `max_files_per_process` files during postmaster startup

- Various memory leakage fixes

- Various portability improvements

- Update timezone data files

- Improve handling of DLL load failures on Windows

- Improve random-number generation on Windows

- Make `psql -f filename` return a nonzero exit code when opening the file fails

- Change pg_dump to handle inherited check constraints more reliably

- Fix password prompting in pg_restore on Windows
- Fix PL/pgSQL to handle `var := var` correctly when the variable is of pass-by-reference type
- Fix PL/Perl `%_SHARED` so it's actually shared
- Fix `contrib/pg_autovacuum` to allow sleep intervals over 2000 sec
- Update `contrib/tsearch2` to use current Snowball code

E.193. Release 8.0.3

Release date: 2005-05-09

This release contains a variety of fixes from 8.0.2, including several security-related issues. For information about new features in the 8.0 major release, see 第 E.196 节.

E.193.1. Migration to Version 8.0.3

A dump/restore is not required for those running 8.0.X. However, it is one possible way of handling two significant security problems that have been found in the initial contents of 8.0.X system catalogs. A dump/initdb/reload sequence using 8.0.3's initdb will automatically correct these problems.

The larger security problem is that the built-in character set encoding conversion functions can be invoked from SQL commands by unprivileged users, but the functions were not designed for such use and are not secure against malicious choices of arguments. The fix involves changing the declared parameter list of these functions so that they can no longer be invoked from SQL commands. (This does not affect their normal use by the encoding conversion machinery.)

The lesser problem is that the `contrib/tsearch2` module creates several functions that are improperly declared to return `internal` when they do not accept `internal` arguments. This breaks type safety for all functions using `internal` arguments.

It is strongly recommended that all installations repair these errors, either by initdb or by following the manual repair procedure given below. The errors at least allow unprivileged database users to crash their server process, and might allow unprivileged users to gain the privileges of a database superuser.

If you wish not to do an initdb, perform the same manual repair procedures shown in the 7.4.8 release notes.

E.193.2. Changes

- Change encoding function signature to prevent misuse
- Change `contrib/tsearch2` to avoid unsafe use of `INTERNAL` function results
- Guard against incorrect second parameter to `record_out`
- Repair ancient race condition that allowed a transaction to be seen as committed for some purposes (eg SELECT FOR UPDATE) slightly sooner than for other purposes

  This is an extremely serious bug since it could lead to apparent data inconsistencies being briefly visible to applications.
- Repair race condition between relation extension and VACUUM

This could theoretically have caused loss of a page's worth of freshly-inserted data, although the scenario seems of very low probability. There are no known cases of it having caused more than an Assert failure.

- Fix comparisons of `TIME WITH TIME ZONE` values

  The comparison code was wrong in the case where the `--enable-integer-datetimes` configuration switch had been used. NOTE: if you have an index on a `TIME WITH TIME ZONE` column, it will need to be `REINDEX`ed after installing this update, because the fix corrects the sort order of column values.

- Fix `EXTRACT(EPOCH)` for `TIME WITH TIME ZONE` values

- Fix mis-display of negative fractional seconds in `INTERVAL` values

  This error only occurred when the `--enable-integer-datetimes` configuration switch had been used.

- Fix pg_dump to dump trigger names containing `%` correctly (Neil)

- Still more 64-bit fixes for `contrib/intagg`

- Prevent incorrect optimization of functions returning `RECORD`

- Prevent crash on `COALESCE(NULL,NULL)`

- Fix Borland makefile for libpq

- Fix `contrib/btree_gist` for `timetz` type (Teodor)

- Make `pg_ctl` check the PID found in `postmaster.pid` to see if it is still a live process

- Fix `pg_dump`/`pg_restore` problems caused by addition of dump timestamps

- Fix interaction between materializing holdable cursors and firing deferred triggers during transaction commit

- Fix memory leak in SQL functions returning pass-by-reference data types

E.194. Release 8.0.2

Release date: 2005-04-07

This release contains a variety of fixes from 8.0.1. For information about new features in the 8.0 major release, see 第 E.196 节.

E.194.1. Migration to Version 8.0.2

A dump/restore is not required for those running 8.0.*. This release updates the major version number of the PostgreSQL libraries, so it might be necessary to re-link some user applications if they cannot find the properly-numbered shared library.

E.194.2. Changes

- Increment the major version number of all interface libraries (Bruce)

  This should have been done in 8.0.0. It is required so 7.4.X versions of PostgreSQL client applications, like psql, can be used on the same machine as 8.0.X applications. This might require re-linking user applications that use these libraries.

- Add Windows-only `wal_sync_method` setting of `fsync_writethrough` (Magnus, Bruce)

  This setting causes PostgreSQL to write through any disk-drive write cache when writing to WAL. This behavior was formerly called `fsync`, but was renamed because it acts quite differently from `fsync` on other platforms.

- Enable the `wal_sync_method` setting of `open_datasync` on Windows, and make it the default for that platform (Magnus, Bruce)

  Because the default is no longer `fsync_writethrough`, data loss is possible during a power failure if the disk drive has write caching enabled. To turn off the write cache on Windows, from the Device Manager, choose the drive properties, then `Policies`.

- New cache management algorithm 2Q replaces ARC (Tom)

  This was done to avoid a pending US patent on ARC. The 2Q code might be a few percentage points slower than ARC for some work loads. A better cache management algorithm will appear in 8.1.

- Planner adjustments to improve behavior on freshly-created tables (Tom)

- Allow plpgsql to assign to an element of an array that is initially `NULL` (Tom)

  Formerly the array would remain `NULL`, but now it becomes a single-element array. The main SQL engine was changed to handle `UPDATE` of a null array value this way in 8.0, but the similar case in plpgsql was overlooked.

- Convert `\r\n` and `\r` to `\n` in plpython function bodies (Michael Fuhr)

  This prevents syntax errors when plpython code is written on a Windows or Mac client.

- Allow SPI cursors to handle utility commands that return rows, such as `EXPLAIN` (Tom)

- Fix `CLUSTER` failure after `ALTER TABLE SET WITHOUT OIDS` (Tom)

- Reduce memory usage of `ALTER TABLE ADD COLUMN` (Neil)

- Fix `ALTER LANGUAGE RENAME` (Tom)

- Document the Windows-only `register` and `unregister` options of pg_ctl (Magnus)

- Ensure operations done during backend shutdown are counted by statistics collector

  This is expected to resolve reports of pg_autovacuum not vacuuming the system catalogs often enough – it was not being told about catalog deletions caused by temporary table removal during backend exit.

- Change the Windows default for configuration parameter `log_destination` to `eventlog` (Magnus)

  By default, a server running on Windows will now send log output to the Windows event logger rather than standard error.

- Make Kerberos authentication work on Windows (Magnus)

- Allow `ALTER DATABASE RENAME` by superusers who aren't flagged as having CREATEDB privilege (Tom)

- Modify WAL log entries for `CREATE` and `DROP DATABASE` to not specify absolute paths (Tom)

  This allows point-in-time recovery on a different machine with possibly different database location. Note that `CREATE TABLESPACE` still poses a hazard in such situations.

- Fix crash from a backend exiting with an open transaction that created a table and opened a cursor on it (Tom)
- Fix `array_map()` so it can call PL functions (Tom)
- Several `contrib/tsearch2` and `contrib/btree_gist` fixes (Teodor)
- Fix crash of some `contrib/pgcrypto` functions on some platforms (Marko Kreen)
- Fix `contrib/intagg` for 64-bit platforms (Tom)
- Fix ecpg bugs in parsing of `CREATE` statement (Michael)
- Work around gcc bug on powerpc and amd64 causing problems in ecpg (Christof Petig)
- Do not use locale-aware versions of `upper()`, `lower()`, and `initcap()` when the locale is `C` (Bruce)

  This allows these functions to work on platforms that generate errors for non-7-bit data when the locale is `C`.

- Fix `quote_ident()` to quote names that match keywords (Tom)
- Fix `to_date()` to behave reasonably when `CC` and `YY` fields are both used (Karel)
- Prevent `to_char(interval)` from failing when given a zero-month interval (Tom)
- Fix wrong week returned by `date_trunc('week')` (Bruce)

  `date_trunc('week')` returned the wrong year for the first few days of January in some years.

- Use the correct default mask length for class `D` addresses in `INET` data types (Tom)

E.195. Release 8.0.1

Release date: 2005-01-31

This release contains a variety of fixes from 8.0.0, including several security-related issues. For information about new features in the 8.0 major release, see 第 E.196 节.

E.195.1. Migration to Version 8.0.1

A dump/restore is not required for those running 8.0.0.

E.195.2. Changes

- Disallow `LOAD` to non-superusers

  On platforms that will automatically execute initialization functions of a shared library (this includes at least Windows and ELF-based Unixen), `LOAD` can be used to make the server execute arbitrary code. Thanks to NGS Software for reporting this.

- Check that creator of an aggregate function has the right to execute the specified transition functions

  This oversight made it possible to bypass denial of EXECUTE permission on a function.

- Fix security and 64-bit issues in contrib/intagg
- Add needed STRICT marking to some contrib functions (Kris Jurka)
- Avoid buffer overrun when plpgsql cursor declaration has too many parameters (Neil)
- Make `ALTER TABLE ADD COLUMN` enforce domain constraints in all cases
- Fix planning error for FULL and RIGHT outer joins

  The result of the join was mistakenly supposed to be sorted the same as the left input. This could not only deliver mis-sorted output to the user, but in case of nested merge joins could give outright wrong answers.

- Improve planning of grouped aggregate queries
- `ROLLBACK TO` *`savepoint`* closes cursors created since the savepoint
- Fix inadequate backend stack size on Windows
- Avoid SHGetSpecialFolderPath() on Windows (Magnus)
- Fix some problems in running pg_autovacuum as a Windows service (Dave Page)
- Multiple minor bug fixes in pg_dump/pg_restore
- Fix ecpg segfault with named structs used in typedefs (Michael)

E.196. Release 8.0

Release date: 2005-01-19

E.196.1. Overview

Major changes in this release:

Microsoft Windows Native Server

This is the first PostgreSQL release to run natively on Microsoft Windows® as a server. It can run as a Windows service. This release supports NT-based Windows releases like Windows 2000 SP4, Windows XP, and Windows 2003. Older releases like Windows 95, Windows 98, and Windows ME are not supported because these operating systems do not have the infrastructure to support PostgreSQL. A separate installer project has been created to ease installation on Windows – see http://www.postgresql.org/ftp/win32/.

Although tested throughout our release cycle, the Windows port does not have the benefit of years of use in production environments that PostgreSQL has on Unix platforms. Therefore it should be treated with the same level of caution as you would a new product.

Previous releases required the Unix emulation toolkit Cygwin in order to run the server on Windows operating systems. PostgreSQL has supported native clients on Windows for many years.

Savepoints

Savepoints allow specific parts of a transaction to be aborted without affecting the remainder of the transaction. Prior releases had no such capability; there was no way to recover from a statement failure within a transaction except by aborting the whole transaction. This feature is valuable for application writers who require error recovery within a complex transaction.

Point-In-Time Recovery

In previous releases there was no way to recover from disk drive failure except to restore from a previous backup or use a standby replication server. Point-in-time recovery allows continuous backup of the server. You can recover either to the point of failure or to some transaction in the past.

Tablespaces

Tablespaces allow administrators to select different file systems for storage of individual tables, indexes, and databases. This improves performance and control over disk space usage. Prior releases used initlocation and manual symlink management for such tasks.

Improved Buffer Management, `CHECKPOINT`, `VACUUM`

This release has a more intelligent buffer replacement strategy, which will make better use of available shared buffers and improve performance. The performance impact of vacuum and checkpoints is also lessened.

Change Column Types

A column's data type can now be changed with `ALTER TABLE`.

New Perl Server-Side Language

A new version of the plperl server-side language now supports a persistent shared storage area, triggers, returning records and arrays of records, and SPI calls to access the database.

Comma-separated-value (CSV) support in `COPY`

`COPY` can now read and write comma-separated-value files. It has the flexibility to interpret nonstandard quoting and separation characters too.

E.196.2. Migration to Version 8.0

A dump/restore using pg_dump is required for those wishing to migrate data from any previous release.

Observe the following incompatibilities:

- In `READ COMMITTED` serialization mode, volatile functions now see the results of concurrent transactions committed up to the beginning of each statement within the function, rather than up to the beginning of the interactive command that called the function.
- Functions declared `STABLE` or `IMMUTABLE` always use the snapshot of the calling query, and therefore do not see the effects of actions taken after the calling query starts, whether in their own transaction or other transactions. Such a function must be read-only, too, meaning that it cannot use any SQL commands other than `SELECT` .

- Nondeferred `AFTER` triggers are now fired immediately after completion of the triggering query, rather than upon finishing the current interactive command. This makes a difference when the triggering query occurred within a function: the trigger is invoked before the function proceeds to its next operation.
- Server configuration parameters `virtual_host` and `tcpip_socket` have been replaced with a more general parameter `listen_addresses`. Also, the server now listens on `localhost` by default, which eliminates the need for the `-i` postmaster switch in many scenarios.
- Server configuration parameters `SortMem` and `VacuumMem` have been renamed to `work_mem` and `maintenance_work_mem` to better reflect their use. The original names are still supported in `SET` and `SHOW`.
- Server configuration parameters `log_pid`, `log_timestamp`, and `log_source_port` have been replaced with a more general parameter `log_line_prefix`.
- Server configuration parameter `syslog` has been replaced with a more logical `log_destination` variable to control the log output destination.
- Server configuration parameter `log_statement` has been changed so it can selectively log just database modification or data definition statements. Server configuration parameter `log_duration` now prints only when `log_statement` prints the query.
- Server configuration parameter `max_expr_depth` parameter has been replaced with `max_stack_depth` which measures the physical stack size rather than the expression nesting depth. This helps prevent session termination due to stack overflow caused by recursive functions.
- The `length()` function no longer counts trailing spaces in `CHAR(n)` values.
- Casting an integer to `BIT(N)` selects the rightmost N bits of the integer, not the leftmost N bits as before.
- Updating an element or slice of a NULL array value now produces a nonnull array result, namely an array containing just the assigned-to positions.
- Syntax checking of array input values has been tightened up considerably. Junk that was previously allowed in odd places with odd results now causes an error. Empty-string element values must now be written as `""`, rather than writing nothing. Also changed behavior with respect to whitespace surrounding array elements: trailing whitespace is now ignored, for symmetry with leading whitespace (which has always been ignored).
- Overflow in integer arithmetic operations is now detected and reported as an error.
- The arithmetic operators associated with the single-byte `"char"` data type have been removed.
- The `extract()` function (also called `date_part`) now returns the proper year for BC dates. It previously returned one less than the correct year. The function now also returns the proper values for millennium and century.
- `CIDR` values now must have their nonmasked bits be zero. For example, we no longer allow `204.248.199.1/31` as a `CIDR` value. Such values should never have been accepted by PostgreSQL and will now be rejected.
- `EXECUTE` now returns a completion tag that matches the executed statement.

- psql's `\copy` command now reads or writes to the query's `stdin/stdout`, rather than psql's `stdin/stdout`. The previous behavior can be accessed via new `pstdin/pstdout` parameters.
- The JDBC client interface has been removed from the core distribution, and is now hosted at http://jdbc.postgresql.org.
- The Tcl client interface has also been removed. There are several Tcl interfaces now hosted at http://gborg.postgresql.org.
- The server now uses its own time zone database, rather than the one supplied by the operating system. This will provide consistent behavior across all platforms. In most cases, there should be little noticeable difference in time zone behavior, except that the time zone names used by `SET/SHOW TimeZone` might be different from what your platform provides.
- Configure's threading option no longer requires users to run tests or edit configuration files; threading options are now detected automatically.
- Now that tablespaces have been implemented, initlocation has been removed.
- The API for user-defined GiST indexes has been changed. The Union and PickSplit methods are now passed a pointer to a special `GistEntryVector` structure, rather than a `bytea`.

E.196.3. Deprecated Features

Some aspects of PostgreSQL's behavior have been determined to be suboptimal. For the sake of backward compatibility these have not been removed in 8.0, but they are considered deprecated and will be removed in the next major release.

- The 8.1 release will remove the `to_char()` function for intervals.
- The server now warns of empty strings passed to `oid/float4/float8` data types, but continues to interpret them as zeroes as before. In the next major release, empty strings will be considered invalid input for these data types.
- By default, tables in PostgreSQL 8.0 and earlier are created with `OID`s. In the next release, this will not be the case: to create a table that contains `OID`s, the `WITH OIDS` clause must be specified or the `default_with_oids` configuration parameter must be set. Users are encouraged to explicitly specify `WITH OIDS` if their tables require OIDs for compatibility with future releases of PostgreSQL.

E.196.4. Changes

Below you will find a detailed account of the changes between release 8.0 and the previous major release.

E.196.4.1. Performance Improvements

- Support cross-data-type index usage (Tom)

  Before this change, many queries would not use an index if the data types did not match exactly. This improvement makes index usage more intuitive and consistent.

- New buffer replacement strategy that improves caching (Jan)

  Prior releases used a least-recently-used (LRU) cache to keep recently referenced pages in memory. The LRU algorithm did not consider the number of times a specific

cache entry was accessed, so large table scans could force out useful cache pages. The new cache algorithm uses four separate lists to track most recently used and most frequently used cache pages and dynamically optimize their replacement based on the work load. This should lead to much more efficient use of the shared buffer cache. Administrators who have tested shared buffer sizes in the past should retest with this new cache replacement policy.

- Add subprocess to write dirty buffers periodically to reduce checkpoint writes (Jan)

  In previous releases, the checkpoint process, which runs every few minutes, would write all dirty buffers to the operating system's buffer cache then flush all dirty operating system buffers to disk. This resulted in a periodic spike in disk usage that often hurt performance. The new code uses a background writer to trickle disk writes at a steady pace so checkpoints have far fewer dirty pages to write to disk. Also, the new code does not issue a global `sync()` call, but instead `fsync()`s just the files written since the last checkpoint. This should improve performance and minimize degradation during checkpoints.

- Add ability to prolong vacuum to reduce performance impact (Jan)

  On busy systems, `VACUUM` performs many I/O requests which can hurt performance for other users. This release allows you to slow down `VACUUM` to reduce its impact on other users, though this increases the total duration of `VACUUM`.

- Improve B-tree index performance for duplicate keys (Dmitry Tkach, Tom)

  This improves the way indexes are scanned when many duplicate values exist in the index.

- Use dynamically-generated table size estimates while planning (Tom)

  Formerly the planner estimated table sizes using the values seen by the last `VACUUM` or `ANALYZE`, both as to physical table size (number of pages) and number of rows. Now, the current physical table size is obtained from the kernel, and the number of rows is estimated by multiplying the table size by the row density (rows per page) seen by the last `VACUUM` or `ANALYZE`. This should produce more reliable estimates in cases where the table size has changed significantly since the last housekeeping command.

- Improved index usage with `OR` clauses (Tom)

  This allows the optimizer to use indexes in statements with many OR clauses that would not have been indexed in the past. It can also use multi-column indexes where the first column is specified and the second column is part of an `OR` clause.

- Improve matching of partial index clauses (Tom)

  The server is now smarter about using partial indexes in queries involving complex `WHERE` clauses.

- Improve performance of the GEQO optimizer (Tom)

  The GEQO optimizer is used to plan queries involving many tables (by default, twelve or more). This release speeds up the way queries are analyzed to decrease time spent in optimization.

- Miscellaneous optimizer improvements

  There is not room here to list all the minor improvements made, but numerous special cases work better than in prior releases.

- Improve lookup speed for C functions (Tom)

  This release uses a hash table to lookup information for dynamically loaded C functions. This improves their speed so they perform nearly as quickly as functions that are built into the server executable.

- Add type-specific `ANALYZE` statistics capability (Mark Cave-Ayland)

  This feature allows more flexibility in generating statistics for nonstandard data types.

- `ANALYZE` now collects statistics for expression indexes (Tom)

  Expression indexes (also called functional indexes) allow users to index not just columns but the results of expressions and function calls. With this release, the optimizer can gather and use statistics about the contents of expression indexes. This will greatly improve the quality of planning for queries in which an expression index is relevant.

- New two-stage sampling method for `ANALYZE` (Manfred Koizar)

  This gives better statistics when the density of valid rows is very different in different regions of a table.

- Speed up `TRUNCATE` (Tom)

  This buys back some of the performance loss observed in 7.4, while still keeping `TRUNCATE` transaction-safe.

E.196.4.2. Server Changes

- Add WAL file archiving and point-in-time recovery (Simon Riggs)
- Add tablespaces so admins can control disk layout (Gavin)
- Add a built-in log rotation program (Andreas Pflug)

  It is now possible to log server messages conveniently without relying on either syslog or an external log rotation program.

- Add new read-only server configuration parameters to show server compile-time settings: `block_size`, `integer_datetimes`, `max_function_args`, `max_identifier_length`, `max_index_keys` (Joe)
- Make quoting of `sameuser`, `samegroup`, and `all` remove special meaning of these terms in `pg_hba.conf` (Andrew)
- Use clearer IPv6 name `::1/128` for `localhost` in default `pg_hba.conf` (Andrew)
- Use CIDR format in `pg_hba.conf` examples (Andrew)
- Rename server configuration parameters `SortMem` and `VacuumMem` to `work_mem` and `maintenance_work_mem` (Old names still supported) (Tom)

  This change was made to clarify that bulk operations such as index and foreign key creation use `maintenance_work_mem`, while `work_mem` is for workspaces used during query execution.

- Allow logging of session disconnections using server configuration `log_disconnections` (Andrew)

- Add new server configuration parameter `log_line_prefix` to allow control of information emitted in each log line (Andrew)

  Available information includes user name, database name, remote IP address, and session start time.

- Remove server configuration parameters `log_pid`, `log_timestamp`, `log_source_port`; functionality superseded by `log_line_prefix` (Andrew)

- Replace the `virtual_host` and `tcpip_socket` parameters with a unified `listen_addresses` parameter (Andrew, Tom)

  `virtual_host` could only specify a single IP address to listen on. `listen_addresses` allows multiple addresses to be specified.

- Listen on localhost by default, which eliminates the need for the `-i` postmaster switch in many scenarios (Andrew)

  Listening on localhost (`127.0.0.1`) opens no new security holes but allows configurations like Windows and JDBC, which do not support local sockets, to work without special adjustments.

- Remove `syslog` server configuration parameter, and add more logical `log_destination` variable to control log output location (Magnus)

- Change server configuration parameter `log_statement` to take values `all`, `mod`, `ddl`, or `none` to select which queries are logged (Bruce)

  This allows administrators to log only data definition changes or only data modification statements.

- Some logging-related configuration parameters could formerly be adjusted by ordinary users, but only in the "more verbose" direction. They are now treated more strictly: only superusers can set them. However, a superuser can use `ALTER USER` to provide per-user settings of these values for non-superusers. Also, it is now possible for superusers to set values of superuser-only configuration parameters via `PGOPTIONS`.

- Allow configuration files to be placed outside the data directory (mlw)

  By default, configuration files are kept in the cluster's top directory. With this addition, configuration files can be placed outside the data directory, easing administration.

- Plan prepared queries only when first executed so constants can be used for statistics (Oliver Jowett)

  Prepared statements plan queries once and execute them many times. While prepared queries avoid the overhead of re-planning on each use, the quality of the plan suffers from not knowing the exact parameters to be used in the query. In this release, planning of unnamed prepared statements is delayed until the first execution, and the actual parameter values of that execution are used as optimization hints. This allows use of out-of-line parameter passing without incurring a performance penalty.

- Allow `DECLARE CURSOR` to take parameters (Oliver Jowett)

  It is now useful to issue `DECLARE CURSOR` in a `Parse` message with parameters. The parameter values sent at `Bind` time will be substituted into the execution of the cursor's query.

- Fix hash joins and aggregates of `inet` and `cidr` data types (Tom)

  Release 7.4 handled hashing of mixed `inet` and `cidr` values incorrectly. (This bug did not exist in prior releases because they wouldn't try to hash either data type.)

- Make `log_duration` print only when `log_statement` prints the query (Ed L.)

E.196.4.3. Query Changes

- Add savepoints (nested transactions) (Alvaro)
- Unsupported isolation levels are now accepted and promoted to the nearest supported level (Peter)

  The SQL specification states that if a database doesn't support a specific isolation level, it should use the next more restrictive level. This change complies with that recommendation.

- Allow `BEGIN WORK` to specify transaction isolation levels like `START TRANSACTION` does (Bruce)
- Fix table permission checking for cases in which rules generate a query type different from the originally submitted query (Tom)
- Implement dollar quoting to simplify single-quote usage (Andrew, Tom, David Fetter)

  In previous releases, because single quotes had to be used to quote a function's body, the use of single quotes inside the function text required use of two single quotes or other error-prone notations. With this release we add the ability to use "dollar quoting" to quote a block of text. The ability to use different quoting delimiters at different nesting levels greatly simplifies the task of quoting correctly, especially in complex functions. Dollar quoting can be used anywhere quoted text is needed.

- Make `CASE val WHEN compval1 THEN ...` evaluate `val` only once (Tom)

  `CASE` no longer evaluates the tested expression multiple times. This has benefits when the expression is complex or is volatile.

- Test `HAVING` before computing target list of an aggregate query (Tom)

  Fixes improper failure of cases such as `SELECT SUM(win)/SUM(lose) ... GROUP BY ... HAVING SUM(lose) > 0`. This should work but formerly could fail with divide-by-zero.

- Replace `max_expr_depth` parameter with `max_stack_depth` parameter, measured in kilobytes of stack size (Tom)

  This gives us a fairly bulletproof defense against crashing due to runaway recursive functions. Instead of measuring the depth of expression nesting, we now directly measure the size of the execution stack.

- Allow arbitrary row expressions (Tom)

  This release allows SQL expressions to contain arbitrary composite types, that is, row values. It also allows functions to more easily take rows as arguments and return row values.

- Allow `LIKE`/`ILIKE` to be used as the operator in row and subselect comparisons (Fabien Coelho)

- Avoid locale-specific case conversion of basic ASCII letters in identifiers and keywords (Tom)

  This solves the "Turkish problem" with mangling of words containing `I` and `i`. Folding of characters outside the 7-bit-ASCII set is still locale-aware.

- Improve syntax error reporting (Fabien, Tom)

  Syntax error reports are more useful than before.

- Change `EXECUTE` to return a completion tag matching the executed statement (Kris Jurka)

  Previous releases return an `EXECUTE` tag for any `EXECUTE` call. In this release, the tag returned will reflect the command executed.

- Avoid emitting `NATURAL CROSS JOIN` in rule listings (Tom)

  Such a clause makes no logical sense, but in some cases the rule decompiler formerly produced this syntax.

E.196.4.4. Object Manipulation Changes

- Add `COMMENT ON` for casts, conversions, languages, operator classes, and large objects (Christopher)

- Add new server configuration parameter `default_with_oids` to control whether tables are created with `OID`s by default (Neil)

  This allows administrators to control whether `CREATE TABLE` commands create tables with or without `OID` columns by default. (Note: the current factory default setting for `default_with_oids` is `TRUE`, but the default will become `FALSE` in future releases.)

- Add `WITH` / `WITHOUT OIDS` clause to `CREATE TABLE AS` (Neil)

- Allow `ALTER TABLE DROP COLUMN` to drop an `OID` column (`ALTER TABLE SET WITHOUT OIDS` still works) (Tom)

- Allow composite types as table columns (Tom)

- Allow `ALTER ... ADD COLUMN` with defaults and `NOT NULL` constraints; works per SQL spec (Rod)

  It is now possible for `ADD COLUMN` to create a column that is not initially filled with NULLs, but with a specified default value.

- Add `ALTER COLUMN TYPE` to change column's type (Rod)

  It is now possible to alter a column's data type without dropping and re-adding the column.

- Allow multiple `ALTER` actions in a single `ALTER TABLE` command (Rod)

  This is particularly useful for `ALTER` commands that rewrite the table (which include `ALTER COLUMN TYPE` and `ADD COLUMN` with a default). By grouping `ALTER` commands together, the table need be rewritten only once.

- Allow `ALTER TABLE` to add `SERIAL` columns (Tom)

  This falls out from the new capability of specifying defaults for new columns.

- Allow changing the owners of aggregates, conversions, databases, functions, operators, operator classes, schemas, types, and tablespaces (Christopher, Euler Taveira de Oliveira)

  Previously this required modifying the system tables directly.

- Allow temporary object creation to be limited to `SECURITY DEFINER` functions (Sean Chittenden)

- Add `ALTER TABLE ... SET WITHOUT CLUSTER` (Christopher)

  Prior to this release, there was no way to clear an auto-cluster specification except to modify the system tables.

- Constraint/Index/`SERIAL` names are now *`table_column_type`* with numbers appended to guarantee uniqueness within the schema (Tom)

  The SQL specification states that such names should be unique within a schema.

- Add `pg_get_serial_sequence()` to return a `SERIAL` column's sequence name (Christopher)

  This allows automated scripts to reliably find the `SERIAL` sequence name.

- Warn when primary/foreign key data type mismatch requires costly lookup

- New `ALTER INDEX` command to allow moving of indexes between tablespaces (Gavin)

- Make `ALTER TABLE OWNER` change dependent sequence ownership too (Alvaro)

E.196.4.5. Utility Command Changes

- Allow `CREATE SCHEMA` to create triggers, indexes, and sequences (Neil)

- Add `ALSO` keyword to `CREATE RULE` (Fabien Coelho)

  This allows `ALSO` to be added to rule creation to contrast it with `INSTEAD` rules.

- Add `NOWAIT` option to `LOCK` (Tatsuo)

  This allows the `LOCK` command to fail if it would have to wait for the requested lock.

- Allow `COPY` to read and write comma-separated-value (CSV) files (Andrew, Bruce)

- Generate error if the `COPY` delimiter and NULL string conflict (Bruce)

- `GRANT`/`REVOKE` behavior follows the SQL spec more closely

- Avoid locking conflict between `CREATE INDEX` and `CHECKPOINT` (Tom)

  In 7.3 and 7.4, a long-running B-tree index build could block concurrent `CHECKPOINT` s from completing, thereby causing WAL bloat because the WAL log could not be recycled.

- Database-wide `ANALYZE` does not hold locks across tables (Tom)

  This reduces the potential for deadlocks against other backends that want exclusive locks on tables. To get the benefit of this change, do not execute database-wide `ANALYZE` inside a transaction block (`BEGIN` block); it must be able to commit and start a new transaction for each table.

- `REINDEX` does not exclusively lock the index's parent table anymore

  The index itself is still exclusively locked, but readers of the table can continue if they are not using the particular index being rebuilt.

- Erase MD5 user passwords when a user is renamed (Bruce)

  PostgreSQL uses the user name as salt when encrypting passwords via MD5. When a user's name is changed, the salt will no longer match the stored MD5 password, so the stored password becomes useless. In this release a notice is generated and the password is cleared. A new password must then be assigned if the user is to be able to log in with a password.

- New pg_ctl `kill` option for Windows (Andrew)

  Windows does not have a `kill` command to send signals to backends so this capability was added to pg_ctl.

- Information schema improvements

- Add `--pwfile` option to initdb so the initial password can be set by GUI tools (Magnus)

- Detect locale/encoding mismatch in initdb (Peter)

- Add `register` command to pg_ctl to register Windows operating system service (Dave Page)

E.196.4.6. Data Type and Function Changes

- More complete support for composite types (row types) (Tom)

  Composite values can be used in many places where only scalar values worked before.

- Reject nonrectangular array values as erroneous (Joe)

  Formerly, `array_in` would silently build a surprising result.

- Overflow in integer arithmetic operations is now detected (Tom)

- The arithmetic operators associated with the single-byte `"char"` data type have been removed.

  Formerly, the parser would select these operators in many situations where an “unable to select an operator” error would be more appropriate, such as `null * null`. If you actually want to do arithmetic on a `"char"` column, you can cast it to integer explicitly.

- Syntax checking of array input values considerably tightened up (Joe)

  Junk that was previously allowed in odd places with odd results now causes an `ERROR`, for example, non-whitespace after the closing right brace.

- Empty-string array element values must now be written as `""`, rather than writing nothing (Joe)

  Formerly, both ways of writing an empty-string element value were allowed, but now a quoted empty string is required. The case where nothing at all appears will probably be considered to be a NULL element value in some future release.

- Array element trailing whitespace is now ignored (Joe)

  Formerly leading whitespace was ignored, but trailing whitespace between an element value and the delimiter or right brace was significant. Now trailing whitespace is also ignored.

- Emit array values with explicit array bounds when lower bound is not one (Joe)
- Accept `YYYY-monthname-DD` as a date string (Tom)
- Make `netmask` and `hostmask` functions return maximum-length mask length (Tom)
- Change factorial function to return `numeric` (Gavin)

  Returning `numeric` allows the factorial function to work for a wider range of input values.

- `to_char`/`to_date()` date conversion improvements (Kurt Roeckx, Fabien Coelho)
- Make `length()` disregard trailing spaces in `CHAR(n)` (Gavin)

  This change was made to improve consistency: trailing spaces are semantically insignificant in `CHAR(n)` data, so they should not be counted by `length()`.

- Warn about empty string being passed to `OID`/`float4`/`float8` data types (Neil)

  8.1 will throw an error instead.

- Allow leading or trailing whitespace in `int2`/`int4`/`int8`/`float4`/`float8` input routines (Neil)
- Better support for IEEE `Infinity` and `NaN` values in `float4`/`float8` (Neil)

  These should now work on all platforms that support IEEE-compliant floating point arithmetic.

- Add `week` option to `date_trunc()` (Robert Creager)
- Fix `to_char` for `1 BC` (previously it returned `1 AD`) (Bruce)
- Fix `date_part(year)` for BC dates (previously it returned one less than the correct year) (Bruce)
- Fix `date_part()` to return the proper millennium and century (Fabien Coelho)

  In previous versions, the century and millennium results had a wrong number and started in the wrong year, as compared to standard reckoning of such things.

- Add `ceiling()` as an alias for `ceil()`, and `power()` as an alias for `pow()` for standards compliance (Neil)
- Change `ln()`, `log()`, `power()`, and `sqrt()` to emit the correct `SQLSTATE` error codes for certain error conditions, as specified by SQL:2003 (Neil)
- Add `width_bucket()` function as defined by SQL:2003 (Neil)
- Add `generate_series()` functions to simplify working with numeric sets (Joe)
- Fix `upper/lower/initcap()` functions to work with multibyte encodings (Tom)
- Add boolean and bitwise integer `AND`/`OR` aggregates (Fabien Coelho)

- New session information functions to return network addresses for client and server (Sean Chittenden)
- Add function to determine the area of a closed path (Sean Chittenden)
- Add function to send cancel request to other backends (Magnus)
- Add `interval` plus `datetime` operators (Tom)

  The reverse ordering, `datetime` plus `interval`, was already supported, but both are required by the SQL standard.
- Casting an integer to `BIT(N)` selects the rightmost N bits of the integer (Tom)

  In prior releases, the leftmost N bits were selected, but this was deemed unhelpful, not to mention inconsistent with casting from bit to int.
- Require `CIDR` values to have all nonmasked bits be zero (Kevin Brintnall)

E.196.4.7. Server-Side Language Changes

- In `READ COMMITTED` serialization mode, volatile functions now see the results of concurrent transactions committed up to the beginning of each statement within the function, rather than up to the beginning of the interactive command that called the function.
- Functions declared `STABLE` or `IMMUTABLE` always use the snapshot of the calling query, and therefore do not see the effects of actions taken after the calling query starts, whether in their own transaction or other transactions. Such a function must be read-only, too, meaning that it cannot use any SQL commands other than `SELECT` . There is a considerable performance gain from declaring a function `STABLE` or `IMMUTABLE` rather than `VOLATILE`.
- Nondeferred `AFTER` triggers are now fired immediately after completion of the triggering query, rather than upon finishing the current interactive command. This makes a difference when the triggering query occurred within a function: the trigger is invoked before the function proceeds to its next operation. For example, if a function inserts a new row into a table, any nondeferred foreign key checks occur before proceeding with the function.
- Allow function parameters to be declared with names (Dennis Björklund)

  This allows better documentation of functions. Whether the names actually do anything depends on the specific function language being used.
- Allow PL/pgSQL parameter names to be referenced in the function (Dennis Björklund)

  This basically creates an automatic alias for each named parameter.
- Do minimal syntax checking of PL/pgSQL functions at creation time (Tom)

  This allows us to catch simple syntax errors sooner.
- More support for composite types (row and record variables) in PL/pgSQL

  For example, it now works to pass a rowtype variable to another function as a single variable.
- Default values for PL/pgSQL variables can now reference previously declared variables

- Improve parsing of PL/pgSQL FOR loops (Tom)

  Parsing is now driven by presence of `..` rather than data type of `FOR` variable. This makes no difference for correct functions, but should result in more understandable error messages when a mistake is made.

- Major overhaul of PL/Perl server-side language (Command Prompt, Andrew Dunstan)

- In PL/Tcl, SPI commands are now run in subtransactions. If an error occurs, the subtransaction is cleaned up and the error is reported as an ordinary Tcl error, which can be trapped with `catch`. Formerly, it was not possible to catch such errors.

- Accept `ELSEIF` in PL/pgSQL (Neil)

  Previously PL/pgSQL only allowed `ELSIF`, but many people are accustomed to spelling this keyword `ELSEIF`.

E.196.4.8. psql Changes

- Improve psql information display about database objects (Christopher)

- Allow psql to display group membership in `\du` and `\dg` (Markus Bertheau)

- Prevent psql `\dn` from showing temporary schemas (Bruce)

- Allow psql to handle tilde user expansion for file names (Zach Irmen)

- Allow psql to display fancy prompts, including color, via readline (Reece Hart, Chet Ramey)

- Make psql `\copy` match `COPY` command syntax fully (Tom)

- Show the location of syntax errors (Fabien Coelho, Tom)

- Add `CLUSTER` information to psql `\d` display (Bruce)

- Change psql `\copy stdin/stdout` to read from command input/output (Bruce)

- Add `pstdin/pstdout` to read from psql's `stdin/stdout` (Mark Feit)

- Add global psql configuration file, `psqlrc.sample` (Bruce)

  This allows a central file where global psql startup commands can be stored.

- Have psql `\d+` indicate if the table has an `OID` column (Neil)

- On Windows, use binary mode in psql when reading files so control-Z is not seen as end-of-file

- Have `\dn+` show permissions and description for schemas (Dennis Björklund)

- Improve tab completion support (Stefan Kaltenbrunn, Greg Sabino Mullane)

- Allow boolean settings to be set using upper or lower case (Michael Paesold)

E.196.4.9. pg_dump Changes

- Use dependency information to improve the reliability of pg_dump (Tom)

  This should solve the longstanding problems with related objects sometimes being dumped in the wrong order.

- Have pg_dump output objects in alphabetical order if possible (Tom)

  This should make it easier to identify changes between dump files.

- Allow pg_restore to ignore some SQL errors (Fabien Coelho)

  This makes pg_restore's behavior similar to the results of feeding a pg_dump output script to psql. In most cases, ignoring errors and plowing ahead is the most useful thing to do. Also added was a pg_restore option to give the old behavior of exiting on an error.

- pg_restore `-l` display now includes objects' schema names
- New begin/end markers in pg_dump text output (Bruce)
- Add start/stop times for pg_dump/pg_dumpall in verbose mode (Bruce)
- Allow most pg_dump options in pg_dumpall (Christopher)
- Have pg_dump use `ALTER OWNER` rather than `SET SESSION AUTHORIZATION` by default (Christopher)

E.196.4.10. libpq Changes

- Make libpq's `SIGPIPE` handling thread-safe (Bruce)
- Add `PQmbdsplen()` which returns the display length of a character (Tatsuo)
- Add thread locking to SSL and Kerberos connections (Manfred Spraul)
- Allow `PQoidValue()`, `PQcmdTuples()`, and `PQoidStatus()` to work on `EXECUTE` commands (Neil)
- Add `PQserverVersion()` to provide more convenient access to the server version number (Greg Sabino Mullane)
- Add `PQprepare/PQsendPrepared()` functions to support preparing statements without necessarily specifying the data types of their parameters (Abhijit Menon-Sen)
- Many ECPG improvements, including `SET DESCRIPTOR` (Michael)

E.196.4.11. Source Code Changes

- Allow the database server to run natively on Windows (Claudio, Magnus, Andrew)
- Shell script commands converted to C versions for Windows support (Andrew)
- Create an extension makefile framework (Fabien Coelho, Peter)

  This simplifies the task of building extensions outside the original source tree.

- Support relocatable installations (Bruce)

  Directory paths for installed files (such as the `/share` directory) are now computed relative to the actual location of the executables, so that an installation tree can be moved to another place without reconfiguring and rebuilding.

- Use `--with-docdir` to choose installation location of documentation; also allow `--infodir` (Peter)
- Add `--without-docdir` to prevent installation of documentation (Peter)

- Upgrade to DocBook V4.2 SGML (Peter)
- New `PostgreSQL` CVS tag (Marc)

  This was done to make it easier for organizations to manage their own copies of the PostgreSQL CVS repository. File version stamps from the master repository will not get munged by checking into or out of a copied repository.
- Clarify locking code (Manfred Koizar)
- Buffer manager cleanup (Neil)
- Decouple platform tests from CPU spinlock code (Bruce, Tom)
- Add inlined test-and-set code on PA-RISC for gcc (ViSolve, Tom)
- Improve i386 spinlock code (Manfred Spraul)
- Clean up spinlock assembly code to avoid warnings from newer gcc releases (Tom)
- Remove JDBC from source tree; now a separate project
- Remove the libpgtcl client interface; now a separate project
- More accurately estimate memory and file descriptor usage (Tom)
- Improvements to the Mac OS X startup scripts (Ray A.)
- New `fsync()` test program (Bruce)
- Major documentation improvements (Neil, Peter)
- Remove pg_encoding; not needed anymore
- Remove pg_id; not needed anymore
- Remove initlocation; not needed anymore
- Auto-detect thread flags (no more manual testing) (Bruce)
- Use Olson's public domain timezone library (Magnus)
- With threading enabled, use thread flags on Unixware for backend executables too (Bruce)

  Unixware cannot mix threaded and nonthreaded object files in the same executable, so everything must be compiled as threaded.
- psql now uses a flex-generated lexical analyzer to process command strings
- Reimplement the linked list data structure used throughout the backend (Neil)

  This improves performance by allowing list append and length operations to be more efficient.
- Allow dynamically loaded modules to create their own server configuration parameters (Thomas Hallgren)
- New Brazilian version of FAQ (Euler Taveira de Oliveira)
- Add French FAQ (Guillaume Lelarge)
- New pgevent for Windows logging

- Make libpq and ECPG build as proper shared libraries on OS X (Tom)

E.196.4.12. Contrib Changes

- Overhaul of `contrib/dblink` (Joe)
- `contrib/dbmirror` improvements (Steven Singer)
- New `contrib/xml2` (John Gray, Torchbox)
- Updated `contrib/mysql`
- New version of `contrib/btree_gist` (Teodor)
- New `contrib/trgm`, trigram matching for PostgreSQL (Teodor)
- Many `contrib/tsearch2` improvements (Teodor)
- Add double metaphone to `contrib/fuzzystrmatch` (Andrew)
- Allow `contrib/pg_autovacuum` to run as a Windows service (Dave Page)
- Add functions to `contrib/dbsize` (Andreas Pflug)
- Removed `contrib/pg_logger`: obsoleted by integrated logging subprocess
- Removed `contrib/rserv`: obsoleted by various separate projects

E.197. Release 7.4.30

Release date: 2010-10-04

This release contains a variety of fixes from 7.4.29. For information about new features in the 7.4 major release, see 第 E.227 节.

This is expected to be the last PostgreSQL release in the 7.4.X series. Users are encouraged to update to a newer release branch soon.

E.197.1. Migration to Version 7.4.30

A dump/restore is not required for those running 7.4.X. However, if you are upgrading from a version earlier than 7.4.26, see 第 E.201 节.

E.197.2. Changes

- Use a separate interpreter for each calling SQL userid in PL/Perl and PL/Tcl (Tom Lane)

  This change prevents security problems that can be caused by subverting Perl or Tcl code that will be executed later in the same session under another SQL user identity (for example, within a `SECURITY DEFINER` function). Most scripting languages offer numerous ways that that might be done, such as redefining standard functions or operators called by the target function. Without this change, any SQL user with Perl or Tcl language usage rights can do essentially anything with the SQL privileges of the target function's owner.

  The cost of this change is that intentional communication among Perl and Tcl functions becomes more difficult. To provide an escape hatch, PL/PerlU and PL/TclU

functions continue to use only one interpreter per session. This is not considered a security issue since all such functions execute at the trust level of a database superuser already.

It is likely that third-party procedural languages that claim to offer trusted execution have similar security issues. We advise contacting the authors of any PL you are depending on for security-critical purposes.

Our thanks to Tim Bunce for pointing out this issue (CVE-2010-3433).

- Prevent possible crashes in `pg_get_expr()` by disallowing it from being called with an argument that is not one of the system catalog columns it's intended to be used with (Heikki Linnakangas, Tom Lane)
- Fix "cannot handle unplanned sub-select" error (Tom Lane)

  This occurred when a sub-select contains a join alias reference that expands into an expression containing another sub-select.
- Take care to fsync the contents of lockfiles (both `postmaster.pid` and the socket lockfile) while writing them (Tom Lane)

  This omission could result in corrupted lockfile contents if the machine crashes shortly after postmaster start. That could in turn prevent subsequent attempts to start the postmaster from succeeding, until the lockfile is manually removed.
- Improve `contrib/dblink`'s handling of tables containing dropped columns (Tom Lane)
- Fix connection leak after "duplicate connection name" errors in `contrib/dblink` (Itagaki Takahiro)
- Update build infrastructure and documentation to reflect the source code repository's move from CVS to Git (Magnus Hagander and others)

E.198. Release 7.4.29

Release date: 2010-05-17

This release contains a variety of fixes from 7.4.28. For information about new features in the 7.4 major release, see 第 E.227 节.

The PostgreSQL community will stop releasing updates for the 7.4.X release series in July 2010. Users are encouraged to update to a newer release branch soon.

E.198.1. Migration to Version 7.4.29

A dump/restore is not required for those running 7.4.X. However, if you are upgrading from a version earlier than 7.4.26, see 第 E.201 节.

E.198.2. Changes

- Enforce restrictions in `plperl` using an opmask applied to the whole interpreter, instead of using `Safe.pm` (Tim Bunce, Andrew Dunstan)

  Recent developments have convinced us that `Safe.pm` is too insecure to rely on for making `plperl` trustable. This change removes use of `Safe.pm` altogether, in favor of using a separate interpreter with an opcode mask that is always applied. Pleasant

side effects of the change include that it is now possible to use Perl's `strict` pragma in a natural way in `plperl`, and that Perl's `$a` and `$b` variables work as expected in sort routines, and that function compilation is significantly faster. (CVE-2010-1169)

- Prevent PL/Tcl from executing untrustworthy code from `pltcl_modules` (Tom)

  PL/Tcl's feature for autoloading Tcl code from a database table could be exploited for trojan-horse attacks, because there was no restriction on who could create or insert into that table. This change disables the feature unless `pltcl_modules` is owned by a superuser. (However, the permissions on the table are not checked, so installations that really need a less-than-secure modules table can still grant suitable privileges to trusted non-superusers.) Also, prevent loading code into the unrestricted "normal" Tcl interpreter unless we are really going to execute a `pltclu` function. (CVE-2010-1170)

- Do not allow an unprivileged user to reset superuser-only parameter settings (Alvaro)

  Previously, if an unprivileged user ran `ALTER USER ... RESET ALL` for himself, or `ALTER DATABASE ... RESET ALL` for a database he owns, this would remove all special parameter settings for the user or database, even ones that are only supposed to be changeable by a superuser. Now, the `ALTER` will only remove the parameters that the user has permission to change.

- Avoid possible crash during backend shutdown if shutdown occurs when a `CONTEXT` addition would be made to log entries (Tom)

  In some cases the context-printing function would fail because the current transaction had already been rolled back when it came time to print a log message.

- Update pl/perl's `ppport.h` for modern Perl versions (Andrew)

- Fix assorted memory leaks in pl/python (Andreas Freund, Tom)

- Ensure that `contrib/pgstattuple` functions respond to cancel interrupts promptly (Tatsuhito Kasahara)

- Make server startup deal properly with the case that `shmget()` returns `EINVAL` for an existing shared memory segment (Tom)

  This behavior has been observed on BSD-derived kernels including OS X. It resulted in an entirely-misleading startup failure complaining that the shared memory request size was too large.

E.199. Release 7.4.28

Release date: 2010-03-15

This release contains a variety of fixes from 7.4.27. For information about new features in the 7.4 major release, see 第 E.227 节.

The PostgreSQL community will stop releasing updates for the 7.4.X release series in July 2010. Users are encouraged to update to a newer release branch soon.

E.199.1. Migration to Version 7.4.28

A dump/restore is not required for those running 7.4.X. However, if you are upgrading from a version earlier than 7.4.26, see 第 E.201 节.

E.199.2. Changes

- Add new configuration parameter `ssl_renegotiation_limit` to control how often we do session key renegotiation for an SSL connection (Magnus)

  This can be set to zero to disable renegotiation completely, which may be required if a broken SSL library is used. In particular, some vendors are shipping stopgap patches for CVE-2009-3555 that cause renegotiation attempts to fail.

- Make `substring()` for `bit` types treat any negative length as meaning "all the rest of the string" (Tom)

  The previous coding treated only -1 that way, and would produce an invalid result value for other negative values, possibly leading to a crash (CVE-2010-0442).

- Fix some cases of pathologically slow regular expression matching (Tom)

- When reading `pg_hba.conf` and related files, do not treat `@something` as a file inclusion request if the `@` appears inside quote marks; also, never treat `@` by itself as a file inclusion request (Tom)

  This prevents erratic behavior if a role or database name starts with `@`. If you need to include a file whose path name contains spaces, you can still do so, but you must write `@"/path to/file"` rather than putting the quotes around the whole construct.

- Prevent infinite loop on some platforms if a directory is named as an inclusion target in `pg_hba.conf` and related files (Tom)

- Ensure PL/Tcl initializes the Tcl interpreter fully (Tom)

  The only known symptom of this oversight is that the Tcl `clock` command misbehaves if using Tcl 8.5 or later.

- Prevent crash in `contrib/dblink` when too many key columns are specified to a `dblink_build_sql_*` function (Rushabh Lathia, Joe Conway)

E.200. Release 7.4.27

Release date: 2009-12-14

This release contains a variety of fixes from 7.4.26. For information about new features in the 7.4 major release, see 第 E.227 节.

E.200.1. Migration to Version 7.4.27

A dump/restore is not required for those running 7.4.X. However, if you are upgrading from a version earlier than 7.4.26, see 第 E.201 节.

E.200.2. Changes

- Protect against indirect security threats caused by index functions changing session-local state (Gurjeet Singh, Tom)

  This change prevents allegedly-immutable index functions from possibly subverting a superuser's session (CVE-2009-4136).

- Reject SSL certificates containing an embedded null byte in the common name (CN) field (Magnus)

This prevents unintended matching of a certificate to a server or client name during SSL validation (CVE-2009-4034).

- Fix possible crash during backend-startup-time cache initialization (Tom)
- Prevent signals from interrupting `VACUUM` at unsafe times (Alvaro)

  This fix prevents a PANIC if a `VACUUM FULL` is canceled after it's already committed its tuple movements, as well as transient errors if a plain `VACUUM` is interrupted after having truncated the table.

- Fix possible crash due to integer overflow in hash table size calculation (Tom)

  This could occur with extremely large planner estimates for the size of a hashjoin's result.

- Fix very rare crash in `inet/cidr` comparisons (Chris Mikkelson)
- Fix PAM password processing to be more robust (Tom)

  The previous code is known to fail with the combination of the Linux `pam_krb5` PAM module with Microsoft Active Directory as the domain controller. It might have problems elsewhere too, since it was making unjustified assumptions about what arguments the PAM stack would pass to it.

- Make the postmaster ignore any `application_name` parameter in connection request packets, to improve compatibility with future libpq versions (Tom)

E.201. Release 7.4.26

Release date: 2009-09-09

This release contains a variety of fixes from 7.4.25. For information about new features in the 7.4 major release, see 第 E.227 节.

E.201.1. Migration to Version 7.4.26

A dump/restore is not required for those running 7.4.X. However, if you have any hash indexes on `interval` columns, you must `REINDEX` them after updating to 7.4.26. Also, if you are upgrading from a version earlier than 7.4.11, see 第 E.216 节.

E.201.2. Changes

- Disallow `RESET ROLE` and `RESET SESSION AUTHORIZATION` inside security-definer functions (Tom, Heikki)

  This covers a case that was missed in the previous patch that disallowed `SET ROLE` and `SET SESSION AUTHORIZATION` inside security-definer functions. (See CVE-2007-6600)

- Fix handling of sub-SELECTs appearing in the arguments of an outer-level aggregate function (Tom)
- Fix hash calculation for data type `interval` (Tom)

  This corrects wrong results for hash joins on interval values. It also changes the contents of hash indexes on interval columns. If you have any such indexes, you must `REINDEX` them after updating.

- Fix overflow for `INTERVAL 'x ms'` when *x* is more than 2 million and integer datetimes are in use (Alex Hunsaker)
- Fix calculation of distance between a point and a line segment (Tom)

  This led to incorrect results from a number of geometric operators.
- Fix `money` data type to work in locales where currency amounts have no fractional digits, e.g. Japan (Itagaki Takahiro)
- Properly round datetime input like `00:12:57.999999999999999999999999999` (Tom)
- Fix poor choice of page split point in GiST R-tree operator classes (Teodor)
- Fix portability issues in plperl initialization (Andrew Dunstan)
- Improve robustness of libpq's code to recover from errors during `COPY FROM STDIN` (Tom)
- Avoid including conflicting readline and editline header files when both libraries are installed (Zdenek Kotala)

E.202. Release 7.4.25

Release date: 2009-03-16

This release contains a variety of fixes from 7.4.24. For information about new features in the 7.4 major release, see 第 E.227 节.

E.202.1. Migration to Version 7.4.25

A dump/restore is not required for those running 7.4.X. However, if you are upgrading from a version earlier than 7.4.11, see 第 E.216 节.

E.202.2. Changes

- Prevent error recursion crashes when encoding conversion fails (Tom)

  This change extends fixes made in the last two minor releases for related failure scenarios. The previous fixes were narrowly tailored for the original problem reports, but we have now recognized that any error thrown by an encoding conversion function could potentially lead to infinite recursion while trying to report the error. The solution therefore is to disable translation and encoding conversion and report the plain-ASCII form of any error message, if we find we have gotten into a recursive error reporting situation. (CVE-2009-0922)
- Disallow `CREATE CONVERSION` with the wrong encodings for the specified conversion function (Heikki)

  This prevents one possible scenario for encoding conversion failure. The previous change is a backstop to guard against other kinds of failures in the same area.
- Fix core dump when `to_char()` is given format codes that are inappropriate for the type of the data argument (Tom)
- Add `MUST` (Mauritius Island Summer Time) to the default list of known timezone abbreviations (Xavier Bugaud)

E.203. Release 7.4.24

Release date: 2009-02-02

This release contains a variety of fixes from 7.4.23. For information about new features in the 7.4 major release, see 第 E.227 节.

E.203.1. Migration to Version 7.4.24

A dump/restore is not required for those running 7.4.X. However, if you are upgrading from a version earlier than 7.4.11, see 第 E.216 节.

E.203.2. Changes

- Improve handling of URLs in `headline()` function (Teodor)
- Improve handling of overlength headlines in `headline()` function (Teodor)
- Prevent possible Assert failure or misconversion if an encoding conversion is created with the wrong conversion function for the specified pair of encodings (Tom, Heikki)
- Avoid unnecessary locking of small tables in `VACUUM` (Heikki)
- Fix uninitialized variables in `contrib/tsearch2`'s `get_covers()` function (Teodor)
- Fix bug in `to_char()`'s handling of `TH` format codes (Andreas Scherbaum)
- Make all documentation reference `pgsql-bugs` and/or `pgsql-hackers` as appropriate, instead of the now-decommissioned `pgsql-ports` and `pgsql-patches` mailing lists (Tom)

E.204. Release 7.4.23

Release date: 2008-11-03

This release contains a variety of fixes from 7.4.22. For information about new features in the 7.4 major release, see 第 E.227 节.

E.204.1. Migration to Version 7.4.23

A dump/restore is not required for those running 7.4.X. However, if you are upgrading from a version earlier than 7.4.11, see 第 E.216 节.

E.204.2. Changes

- Fix backend crash when the client encoding cannot represent a localized error message (Tom)

  We have addressed similar issues before, but it would still fail if the "character has no equivalent" message itself couldn't be converted. The fix is to disable localization and send the plain ASCII error message when we detect such a situation.

- Fix incorrect tsearch2 headline generation when single query item matches first word of text (Sushant Sinha)
- Fix improper display of fractional seconds in interval values when using a non-ISO datestyle in an `--enable-integer-datetimes` build (Ron Mayer)

- Ensure `SPI_getvalue` and `SPI_getbinval` behave correctly when the passed tuple and tuple descriptor have different numbers of columns (Tom)

  This situation is normal when a table has had columns added or removed, but these two functions didn't handle it properly. The only likely consequence is an incorrect error indication.

- Fix ecpg's parsing of `CREATE USER` (Michael)

E.205. Release 7.4.22

Release date: 2008-09-22

This release contains a variety of fixes from 7.4.21. For information about new features in the 7.4 major release, see 第 E.227 节.

E.205.1. Migration to Version 7.4.22

A dump/restore is not required for those running 7.4.X. However, if you are upgrading from a version earlier than 7.4.11, see 第 E.216 节.

E.205.2. Changes

- Fix datetime input functions to correctly detect integer overflow when running on a 64-bit platform (Tom)
- Improve performance of writing very long log messages to syslog (Tom)
- Fix bug in backwards scanning of a cursor on a `SELECT DISTINCT ON` query (Tom)
- Fix planner to estimate that `GROUP BY` expressions yielding boolean results always result in two groups, regardless of the expressions' contents (Tom)

  This is very substantially more accurate than the regular `GROUP BY` estimate for certain boolean tests like *`col`* `IS NULL`.

- Improve pg_dump and pg_restore's error reporting after failure to send a SQL command (Tom)

E.206. Release 7.4.21

Release date: 2008-06-12

This release contains one serious bug fix over 7.4.20. For information about new features in the 7.4 major release, see 第 E.227 节.

E.206.1. Migration to Version 7.4.21

A dump/restore is not required for those running 7.4.X. However, if you are upgrading from a version earlier than 7.4.11, see 第 E.216 节.

E.206.2. Changes

- Make `pg_get_ruledef()` parenthesize negative constants (Tom)

  Before this fix, a negative constant in a view or rule might be dumped as, say, `-42::integer`, which is subtly incorrect: it should be `(-42)::integer` due to operator

precedence rules. Usually this would make little difference, but it could interact with another recent patch to cause PostgreSQL to reject what had been a valid `SELECT DISTINCT` view query. Since this could result in pg_dump output failing to reload, it is being treated as a high-priority fix. The only released versions in which dump output is actually incorrect are 8.3.1 and 8.2.7.

E.207. Release 7.4.20

Release date: never released

This release contains a variety of fixes from 7.4.19. For information about new features in the 7.4 major release, see 第 E.227 节.

E.207.1. Migration to Version 7.4.20

A dump/restore is not required for those running 7.4.X. However, if you are upgrading from a version earlier than 7.4.11, see 第 E.216 节.

E.207.2. Changes

- Fix conversions between ISO-8859-5 and other encodings to handle Cyrillic "Yo" characters (`e` and `E` with two dots) (Sergey Burladyan)
- Fix a few datatype input functions that were allowing unused bytes in their results to contain uninitialized, unpredictable values (Tom)

  This could lead to failures in which two apparently identical literal values were not seen as equal, resulting in the parser complaining about unmatched `ORDER BY` and `DISTINCT` expressions.
- Fix a corner case in regular-expression substring matching (`substring(`*`string`* `from` *`pattern`*`)`) (Tom)

  The problem occurs when there is a match to the pattern overall but the user has specified a parenthesized subexpression and that subexpression hasn't got a match. An example is `substring('foo' from 'foo(bar)?')`. This should return NULL, since `(bar)` isn't matched, but it was mistakenly returning the whole-pattern match instead (ie, `foo`).
- Fix incorrect result from ecpg's `PGTYPEStimestamp_sub()` function (Michael)
- Fix `DatumGetBool` macro to not fail with gcc 4.3 (Tom)

  This problem affects "old style" (V0) C functions that return boolean. The fix is already in 8.3, but the need to back-patch it was not realized at the time.
- Fix longstanding `LISTEN`/`NOTIFY` race condition (Tom)

  In rare cases a session that had just executed a `LISTEN` might not get a notification, even though one would be expected because the concurrent transaction executing `NOTIFY` was observed to commit later.

  A side effect of the fix is that a transaction that has executed a not-yet-committed `LISTEN` command will not see any row in `pg_listener` for the `LISTEN`, should it choose to look; formerly it would have. This behavior was never documented one way or the other, but it is possible that some applications depend on the old behavior.
- Fix display of constant expressions in `ORDER BY` and `GROUP BY` (Tom)

An explicitly casted constant would be shown incorrectly. This could for example lead to corruption of a view definition during dump and reload.

- Fix libpq to handle NOTICE messages correctly during COPY OUT (Tom)

  This failure has only been observed to occur when a user-defined datatype's output routine issues a NOTICE, but there is no guarantee it couldn't happen due to other causes.

E.208. Release 7.4.19

Release date: 2008-01-07

This release contains a variety of fixes from 7.4.18, including fixes for significant security issues. For information about new features in the 7.4 major release, see 第 E.227 节.

E.208.1. Migration to Version 7.4.19

A dump/restore is not required for those running 7.4.X. However, if you are upgrading from a version earlier than 7.4.11, see 第 E.216 节.

E.208.2. Changes

- Prevent functions in indexes from executing with the privileges of the user running `VACUUM`, `ANALYZE`, etc (Tom)

  Functions used in index expressions and partial-index predicates are evaluated whenever a new table entry is made. It has long been understood that this poses a risk of trojan-horse code execution if one modifies a table owned by an untrustworthy user. (Note that triggers, defaults, check constraints, etc. pose the same type of risk.) But functions in indexes pose extra danger because they will be executed by routine maintenance operations such as `VACUUM FULL`, which are commonly performed automatically under a superuser account. For example, a nefarious user can execute code with superuser privileges by setting up a trojan-horse index definition and waiting for the next routine vacuum. The fix arranges for standard maintenance operations (including `VACUUM`, `ANALYZE`, `REINDEX`, and `CLUSTER`) to execute as the table owner rather than the calling user, using the same privilege-switching mechanism already used for `SECURITY DEFINER` functions. To prevent bypassing this security measure, execution of `SET SESSION AUTHORIZATION` and `SET ROLE` is now forbidden within a `SECURITY DEFINER` context. (CVE-2007-6600)

- Repair assorted bugs in the regular-expression package (Tom, Will Drewry)

  Suitably crafted regular-expression patterns could cause crashes, infinite or near-infinite looping, and/or massive memory consumption, all of which pose denial-of-service hazards for applications that accept regex search patterns from untrustworthy sources. (CVE-2007-4769, CVE-2007-4772, CVE-2007-6067)

- Require non-superusers who use `/contrib/dblink` to use only password authentication, as a security measure (Joe)

  The fix that appeared for this in 7.4.18 was incomplete, as it plugged the hole for only some `dblink` functions. (CVE-2007-6601, CVE-2007-3278)

- Fix planner failure in some cases of `WHERE false AND var IN (SELECT ...)` (Tom)

- Fix potential crash in `translate()` when using a multibyte database encoding (Tom)
- Fix PL/Python to not crash on long exception messages (Alvaro)
- ecpg parser fixes (Michael)
- Make `contrib/tablefunc`'s `crosstab()` handle NULL rowid as a category in its own right, rather than crashing (Joe)
- Fix `tsvector` and `tsquery` output routines to escape backslashes correctly (Teodor, Bruce)
- Fix crash of `to_tsvector()` on huge input strings (Teodor)
- Require a specific version of Autoconf to be used when re-generating the `configure` script (Peter)

  This affects developers and packagers only. The change was made to prevent accidental use of untested combinations of Autoconf and PostgreSQL versions. You can remove the version check if you really want to use a different Autoconf version, but it's your responsibility whether the result works or not.

E.209. Release 7.4.18

Release date: 2007-09-17

This release contains fixes from 7.4.17. For information about new features in the 7.4 major release, see 第 E.227 节.

E.209.1. Migration to Version 7.4.18

A dump/restore is not required for those running 7.4.X. However, if you are upgrading from a version earlier than 7.4.11, see 第 E.216 节.

E.209.2. Changes

- Prevent index corruption when a transaction inserts rows and then aborts close to the end of a concurrent `VACUUM` on the same table (Tom)
- Make `CREATE DOMAIN ... DEFAULT NULL` work properly (Tom)
- Fix excessive logging of SSL error messages (Tom)
- Fix crash when `log_min_error_statement` logging runs out of memory (Tom)
- Prevent `CLUSTER` from failing due to attempting to process temporary tables of other sessions (Alvaro)
- Require non-superusers who use `/contrib/dblink` to use only password authentication, as a security measure (Joe)

E.210. Release 7.4.17

Release date: 2007-04-23

This release contains fixes from 7.4.16, including a security fix. For information about new features in the 7.4 major release, see 第 E.227 节.

E.210.1. Migration to Version 7.4.17

A dump/restore is not required for those running 7.4.X. However, if you are upgrading from a version earlier than 7.4.11, see 第 E.216 节.

E.210.2. Changes

- Support explicit placement of the temporary-table schema within `search_path`, and disable searching it for functions and operators (Tom)

  This is needed to allow a security-definer function to set a truly secure value of `search_path`. Without it, an unprivileged SQL user can use temporary objects to execute code with the privileges of the security-definer function (CVE-2007-2138). See `CREATE FUNCTION` for more information.

- `/contrib/tsearch2` crash fixes (Teodor)

- Fix potential-data-corruption bug in how `VACUUM FULL` handles `UPDATE` chains (Tom, Pavan Deolasee)

- Fix PANIC during enlargement of a hash index (bug introduced in 7.4.15) (Tom)

E.211. Release 7.4.16

Release date: 2007-02-05

This release contains a variety of fixes from 7.4.15, including a security fix. For information about new features in the 7.4 major release, see 第 E.227 节.

E.211.1. Migration to Version 7.4.16

A dump/restore is not required for those running 7.4.X. However, if you are upgrading from a version earlier than 7.4.11, see 第 E.216 节.

E.211.2. Changes

- Remove security vulnerability that allowed connected users to read backend memory (Tom)

  The vulnerability involves suppressing the normal check that a SQL function returns the data type it's declared to, or changing the data type of a table column used in a SQL function (CVE-2007-0555). This error can easily be exploited to cause a backend crash, and in principle might be used to read database content that the user should not be able to access.

- Fix rare bug wherein btree index page splits could fail due to choosing an infeasible split point (Heikki Linnakangas)

- Fix for rare Assert() crash triggered by `UNION` (Tom)

- Tighten security of multi-byte character processing for UTF8 sequences over three bytes long (Tom)

E.212. Release 7.4.15

Release date: 2007-01-08

This release contains a variety of fixes from 7.4.14. For information about new features in the 7.4 major release, see 第 E.227 节.

E.212.1. Migration to Version 7.4.15

A dump/restore is not required for those running 7.4.X. However, if you are upgrading from a version earlier than 7.4.11, see 第 E.216 节.

E.212.2. Changes

- Improve handling of `getaddrinfo()` on AIX (Tom)

  This fixes a problem with starting the statistics collector, among other things.

- Fix "failed to re-find parent key" errors in `VACUUM` (Tom)

- Fix bugs affecting multi-gigabyte hash indexes (Tom)

- Fix error when constructing an `ARRAY[]` made up of multiple empty elements (Tom)

- `to_number()` and `to_char(numeric)` are now `STABLE`, not `IMMUTABLE`, for new initdb installs (Tom)

  This is because `lc_numeric` can potentially change the output of these functions.

- Improve index usage of regular expressions that use parentheses (Tom)

  This improves psql `\d` performance also.

E.213. Release 7.4.14

Release date: 2006-10-16

This release contains a variety of fixes from 7.4.13. For information about new features in the 7.4 major release, see 第 E.227 节.

E.213.1. Migration to Version 7.4.14

A dump/restore is not required for those running 7.4.X. However, if you are upgrading from a version earlier than 7.4.11, see 第 E.216 节.

E.213.2. Changes

- Fix core dump when an untyped literal is taken as ANYARRAY

- Fix `string_to_array()` to handle overlapping matches for the separator string

  For example, `string_to_array('123xx456xxx789', 'xx')`.

- Fix corner cases in pattern matching for psql's `\d` commands

- Fix index-corrupting bugs in /contrib/ltree (Teodor)

- Fix backslash escaping in /contrib/dbmirror

- Adjust regression tests for recent changes in US DST laws

E.214. Release 7.4.13

Release date: 2006-05-23

This release contains a variety of fixes from 7.4.12, including patches for extremely serious security issues. For information about new features in the 7.4 major release, see 第 E.227 节.

E.214.1. Migration to Version 7.4.13

A dump/restore is not required for those running 7.4.X. However, if you are upgrading from a version earlier than 7.4.11, see 第 E.216 节.

Full security against the SQL-injection attacks described in CVE-2006-2313 and CVE-2006-2314 might require changes in application code. If you have applications that embed untrustworthy strings into SQL commands, you should examine them as soon as possible to ensure that they are using recommended escaping techniques. In most cases, applications should be using subroutines provided by libraries or drivers (such as libpq's `PQescapeStringConn()`) to perform string escaping, rather than relying on ad hoc code to do it.

E.214.2. Changes

- Change the server to reject invalidly-encoded multibyte characters in all cases (Tatsuo, Tom)

  While PostgreSQL has been moving in this direction for some time, the checks are now applied uniformly to all encodings and all textual input, and are now always errors not merely warnings. This change defends against SQL-injection attacks of the type described in CVE-2006-2313.

- Reject unsafe uses of `\'` in string literals

  As a server-side defense against SQL-injection attacks of the type described in CVE-2006-2314, the server now only accepts `''` and not `\'` as a representation of ASCII single quote in SQL string literals. By default, `\'` is rejected only when `client_encoding` is set to a client-only encoding (SJIS, BIG5, GBK, GB18030, or UHC), which is the scenario in which SQL injection is possible. A new configuration parameter `backslash_quote` is available to adjust this behavior when needed. Note that full security against CVE-2006-2314 might require client-side changes; the purpose of `backslash_quote` is in part to make it obvious that insecure clients are insecure.

- Modify libpq's string-escaping routines to be aware of encoding considerations and `standard_conforming_strings`

  This fixes libpq-using applications for the security issues described in CVE-2006-2313 and CVE-2006-2314, and also future-proofs them against the planned changeover to SQL-standard string literal syntax. Applications that use multiple PostgreSQL connections concurrently should migrate to `PQescapeStringConn()` and `PQescapeByteaConn()` to ensure that escaping is done correctly for the settings in use in each database connection. Applications that do string escaping "by hand" should be modified to rely on library routines instead.

- Fix some incorrect encoding conversion functions

  `win1251_to_iso`, `alt_to_iso`, `euc_tw_to_big5`, `euc_tw_to_mic`, `mic_to_euc_tw` were all broken to varying extents.

- Clean up stray remaining uses of `\'` in strings (Bruce, Jan)
- Fix bug that sometimes caused OR'd index scans to miss rows they should have returned
- Fix WAL replay for case where a btree index has been truncated
- Fix `SIMILAR TO` for patterns involving `|` (Tom)
- Fix server to use custom DH SSL parameters correctly (Michael Fuhr)
- Fix for Bonjour on Intel Macs (Ashley Clark)
- Fix various minor memory leaks

E.215. Release 7.4.12

Release date: 2006-02-14

This release contains a variety of fixes from 7.4.11. For information about new features in the 7.4 major release, see 第 E.227 节.

E.215.1. Migration to Version 7.4.12

A dump/restore is not required for those running 7.4.X. However, if you are upgrading from a version earlier than 7.4.11, see 第 E.216 节.

E.215.2. Changes

- Fix potential crash in `SET SESSION AUTHORIZATION` (CVE-2006-0553)

  An unprivileged user could crash the server process, resulting in momentary denial of service to other users, if the server has been compiled with Asserts enabled (which is not the default). Thanks to Akio Ishida for reporting this problem.

- Fix bug with row visibility logic in self-inserted rows (Tom)

  Under rare circumstances a row inserted by the current command could be seen as already valid, when it should not be. Repairs bug created in 7.4.9 and 7.3.11 releases.

- Fix race condition that could lead to “file already exists” errors during pg_clog file creation (Tom)
- Properly check `DOMAIN` constraints for `UNKNOWN` parameters in prepared statements (Neil)
- Fix to allow restoring dumps that have cross-schema references to custom operators (Tom)
- Portability fix for testing presence of `finite` and `isinf` during configure (Tom)

E.216. Release 7.4.11

Release date: 2006-01-09

This release contains a variety of fixes from 7.4.10. For information about new features in the 7.4 major release, see 第 E.227 节.

E.216.1. Migration to Version 7.4.11

A dump/restore is not required for those running 7.4.X. However, if you are upgrading from a version earlier than 7.4.8, see 第 E.219 节. Also, you might need to `REINDEX` indexes on textual columns after updating, if you are affected by the locale or plperl issues described below.

E.216.2. Changes

- Fix for protocol-level Describe messages issued outside a transaction or in a failed transaction (Tom)
- Fix character string comparison for locales that consider different character combinations as equal, such as Hungarian (Tom)

  This might require `REINDEX` to fix existing indexes on textual columns.
- Set locale environment variables during postmaster startup to ensure that plperl won't change the locale later

  This fixes a problem that occurred if the postmaster was started with environment variables specifying a different locale than what initdb had been told. Under these conditions, any use of plperl was likely to lead to corrupt indexes. You might need `REINDEX` to fix existing indexes on textual columns if this has happened to you.
- Fix longstanding bug in strpos() and regular expression handling in certain rarely used Asian multi-byte character sets (Tatsuo)
- Fix bug in `/contrib/pgcrypto` gen_salt, which caused it not to use all available salt space for MD5 and XDES algorithms (Marko Kreen, Solar Designer)

  Salts for Blowfish and standard DES are unaffected.
- Fix `/contrib/dblink` to throw an error, rather than crashing, when the number of columns specified is different from what's actually returned by the query (Joe)

E.217. Release 7.4.10

Release date: 2005-12-12

This release contains a variety of fixes from 7.4.9. For information about new features in the 7.4 major release, see 第 E.227 节.

E.217.1. Migration to Version 7.4.10

A dump/restore is not required for those running 7.4.X. However, if you are upgrading from a version earlier than 7.4.8, see 第 E.219 节.

E.217.2. Changes

- Fix race condition in transaction log management

  There was a narrow window in which an I/O operation could be initiated for the wrong page, leading to an Assert failure or data corruption.
- Prevent failure if client sends Bind protocol message when current transaction is already aborted

- `/contrib/ltree` fixes (Teodor)
- AIX and HPUX compile fixes (Tom)
- Fix longstanding planning error for outer joins

  This bug sometimes caused a bogus error "RIGHT JOIN is only supported with merge-joinable join conditions".
- Prevent core dump in pg_autovacuum when a table has been dropped

E.218. Release 7.4.9

Release date: 2005-10-04

This release contains a variety of fixes from 7.4.8. For information about new features in the 7.4 major release, see 第 E.227 节.

E.218.1. Migration to Version 7.4.9

A dump/restore is not required for those running 7.4.X. However, if you are upgrading from a version earlier than 7.4.8, see 第 E.219 节.

E.218.2. Changes

- Fix error that allowed `VACUUM` to remove `ctid` chains too soon, and add more checking in code that follows `ctid` links

  This fixes a long-standing problem that could cause crashes in very rare circumstances.
- Fix `CHAR()` to properly pad spaces to the specified length when using a multiple-byte character set (Yoshiyuki Asaba)

  In prior releases, the padding of `CHAR()` was incorrect because it only padded to the specified number of bytes without considering how many characters were stored.
- Fix the sense of the test for read-only transaction in `COPY`

  The code formerly prohibited `COPY TO`, where it should prohibit `COPY FROM`.
- Fix planning problem with outer-join ON clauses that reference only the inner-side relation
- Further fixes for `x FULL JOIN y ON true` corner cases
- Make `array_in` and `array_recv` more paranoid about validating their OID parameter
- Fix missing rows in queries like `UPDATE a=... WHERE a...` with GiST index on column `a`
- Improve robustness of datetime parsing
- Improve checking for partially-written WAL pages
- Improve robustness of signal handling when SSL is enabled
- Don't try to open more than `max_files_per_process` files during postmaster startup

- Various memory leakage fixes
- Various portability improvements
- Fix PL/pgSQL to handle `var := var` correctly when the variable is of pass-by-reference type
- Update `contrib/tsearch2` to use current Snowball code

E.219. Release 7.4.8

Release date: 2005-05-09

This release contains a variety of fixes from 7.4.7, including several security-related issues. For information about new features in the 7.4 major release, see 第 E.227 节.

E.219.1. Migration to Version 7.4.8

A dump/restore is not required for those running 7.4.X. However, it is one possible way of handling two significant security problems that have been found in the initial contents of 7.4.X system catalogs. A dump/initdb/reload sequence using 7.4.8's initdb will automatically correct these problems.

The larger security problem is that the built-in character set encoding conversion functions can be invoked from SQL commands by unprivileged users, but the functions were not designed for such use and are not secure against malicious choices of arguments. The fix involves changing the declared parameter list of these functions so that they can no longer be invoked from SQL commands. (This does not affect their normal use by the encoding conversion machinery.)

The lesser problem is that the `contrib/tsearch2` module creates several functions that are misdeclared to return `internal` when they do not accept `internal` arguments. This breaks type safety for all functions using `internal` arguments.

It is strongly recommended that all installations repair these errors, either by initdb or by following the manual repair procedures given below. The errors at least allow unprivileged database users to crash their server process, and might allow unprivileged users to gain the privileges of a database superuser.

If you wish not to do an initdb, perform the following procedures instead. As the database superuser, do:

```
BEGIN;
UPDATE pg_proc SET proargtypes[3] = 'internal'::regtype
WHERE pronamespace = 11 AND pronargs = 5
     AND proargtypes[2] = 'cstring'::regtype;
-- The command should report having updated 90 rows;
-- if not, rollback and investigate instead of committing!
COMMIT;
```

Next, if you have installed `contrib/tsearch2`, do:

```
BEGIN;
UPDATE pg_proc SET proargtypes[0] = 'internal'::regtype
WHERE oid IN (
   'dex_init(text)'::regprocedure,
   'snb_en_init(text)'::regprocedure,
   'snb_ru_init(text)'::regprocedure,
```

```
   'spell_init(text)'::regprocedure,
   'syn_init(text)'::regprocedure
);
-- The command should report having updated 5 rows;
-- if not, rollback and investigate instead of committing!
COMMIT;
```

If this command fails with a message like “function "dex_init(text)" does not exist” , then either `tsearch2` is not installed in this database, or you already did the update.

The above procedures must be carried out in each database of an installation, including `template1`, and ideally including `template0` as well. If you do not fix the template databases then any subsequently created databases will contain the same errors. `template1` can be fixed in the same way as any other database, but fixing `template0` requires additional steps. First, from any database issue:

```
UPDATE pg_database SET datallowconn = true WHERE datname =
 'template0';
```

Next connect to `template0` and perform the above repair procedures. Finally, do:

```
-- re-freeze template0:
VACUUM FREEZE;
-- and protect it against future alterations:
UPDATE pg_database SET datallowconn = false WHERE datname =
 'template0';
```

E.219.2. Changes

- Change encoding function signature to prevent misuse
- Change `contrib/tsearch2` to avoid unsafe use of `INTERNAL` function results
- Repair ancient race condition that allowed a transaction to be seen as committed for some purposes (eg SELECT FOR UPDATE) slightly sooner than for other purposes

  This is an extremely serious bug since it could lead to apparent data inconsistencies being briefly visible to applications.
- Repair race condition between relation extension and VACUUM

  This could theoretically have caused loss of a page's worth of freshly-inserted data, although the scenario seems of very low probability. There are no known cases of it having caused more than an Assert failure.
- Fix comparisons of `TIME WITH TIME ZONE` values

  The comparison code was wrong in the case where the `--enable-integer-datetimes` configuration switch had been used. NOTE: if you have an index on a `TIME WITH TIME ZONE` column, it will need to be `REINDEX`ed after installing this update, because the fix corrects the sort order of column values.
- Fix `EXTRACT(EPOCH)` for `TIME WITH TIME ZONE` values
- Fix mis-display of negative fractional seconds in `INTERVAL` values

  This error only occurred when the `--enable-integer-datetimes` configuration switch had been used.
- Ensure operations done during backend shutdown are counted by statistics collector

This is expected to resolve reports of pg_autovacuum not vacuuming the system catalogs often enough – it was not being told about catalog deletions caused by temporary table removal during backend exit.

- Additional buffer overrun checks in plpgsql (Neil)
- Fix pg_dump to dump trigger names containing `%` correctly (Neil)
- Fix `contrib/pgcrypto` for newer OpenSSL builds (Marko Kreen)
- Still more 64-bit fixes for `contrib/intagg`
- Prevent incorrect optimization of functions returning `RECORD`
- Prevent `to_char(interval)` from dumping core for month-related formats
- Prevent crash on `COALESCE(NULL,NULL)`
- Fix `array_map` to call PL functions correctly
- Fix permission checking in `ALTER DATABASE RENAME`
- Fix `ALTER LANGUAGE RENAME`
- Make `RemoveFromWaitQueue` clean up after itself

  This fixes a lock management error that would only be visible if a transaction was kicked out of a wait for a lock (typically by query cancel) and then the holder of the lock released it within a very narrow window.
- Fix problem with untyped parameter appearing in `INSERT ... SELECT`
- Fix `CLUSTER` failure after `ALTER TABLE SET WITHOUT OIDS`

E.220. Release 7.4.7

Release date: 2005-01-31

This release contains a variety of fixes from 7.4.6, including several security-related issues. For information about new features in the 7.4 major release, see 第 E.227 节.

E.220.1. Migration to Version 7.4.7

A dump/restore is not required for those running 7.4.X.

E.220.2. Changes

- Disallow `LOAD` to non-superusers

  On platforms that will automatically execute initialization functions of a shared library (this includes at least Windows and ELF-based Unixen), `LOAD` can be used to make the server execute arbitrary code. Thanks to NGS Software for reporting this.
- Check that creator of an aggregate function has the right to execute the specified transition functions

  This oversight made it possible to bypass denial of EXECUTE permission on a function.

- Fix security and 64-bit issues in contrib/intagg
- Add needed STRICT marking to some contrib functions (Kris Jurka)
- Avoid buffer overrun when plpgsql cursor declaration has too many parameters (Neil)
- Fix planning error for FULL and RIGHT outer joins

  The result of the join was mistakenly supposed to be sorted the same as the left input. This could not only deliver mis-sorted output to the user, but in case of nested merge joins could give outright wrong answers.
- Fix plperl for quote marks in tuple fields
- Fix display of negative intervals in SQL and GERMAN datestyles
- Make age(timestamptz) do calculation in local timezone not GMT

E.221. Release 7.4.6

Release date: 2004-10-22

This release contains a variety of fixes from 7.4.5. For information about new features in the 7.4 major release, see 第 E.227 节.

E.221.1. Migration to Version 7.4.6

A dump/restore is not required for those running 7.4.X.

E.221.2. Changes

- Repair possible failure to update hint bits on disk

  Under rare circumstances this oversight could lead to "could not access transaction status" failures, which qualifies it as a potential-data-loss bug.
- Ensure that hashed outer join does not miss tuples

  Very large left joins using a hash join plan could fail to output unmatched left-side rows given just the right data distribution.
- Disallow running pg_ctl as root

  This is to guard against any possible security issues.
- Avoid using temp files in `/tmp` in `make_oidjoins_check`

  This has been reported as a security issue, though it's hardly worthy of concern since there is no reason for non-developers to use this script anyway.
- Prevent forced backend shutdown from re-emitting prior command result

  In rare cases, a client might think that its last command had succeeded when it really had been aborted by forced database shutdown.
- Repair bug in `pg_stat_get_backend_idset`

  This could lead to misbehavior in some of the system-statistics views.

- Fix small memory leak in postmaster
- Fix “expected both swapped tables to have TOAST tables” bug

  This could arise in cases such as CLUSTER after ALTER TABLE DROP COLUMN.
- Prevent `pg_ctl restart` from adding `-D` multiple times
- Fix problem with NULL values in GiST indexes
- `::` is no longer interpreted as a variable in an ECPG prepare statement

E.222. Release 7.4.5

Release date: 2004-08-18

This release contains one serious bug fix over 7.4.4. For information about new features in the 7.4 major release, see 第 E.227 节.

E.222.1. Migration to Version 7.4.5

A dump/restore is not required for those running 7.4.X.

E.222.2. Changes

- Repair possible crash during concurrent B-tree index insertions

  This patch fixes a rare case in which concurrent insertions into a B-tree index could result in a server panic. No permanent damage would result, but it's still worth a re-release. The bug does not exist in pre-7.4 releases.

E.223. Release 7.4.4

Release date: 2004-08-16

This release contains a variety of fixes from 7.4.3. For information about new features in the 7.4 major release, see 第 E.227 节.

E.223.1. Migration to Version 7.4.4

A dump/restore is not required for those running 7.4.X.

E.223.2. Changes

- Prevent possible loss of committed transactions during crash

  Due to insufficient interlocking between transaction commit and checkpointing, it was possible for transactions committed just before the most recent checkpoint to be lost, in whole or in part, following a database crash and restart. This is a serious bug that has existed since PostgreSQL 7.1.
- Check HAVING restriction before evaluating result list of an aggregate plan
- Avoid crash when session's current user ID is deleted
- Fix hashed crosstab for zero-rows case (Joe)
- Force cache update after renaming a column in a foreign key

- Pretty-print UNION queries correctly
- Make psql handle `\r\n` newlines properly in COPY IN
- pg_dump handled ACLs with grant options incorrectly
- Fix thread support for OS X and Solaris
- Updated JDBC driver (build 215) with various fixes
- ECPG fixes
- Translation updates (various contributors)

E.224. Release 7.4.3

Release date: 2004-06-14

This release contains a variety of fixes from 7.4.2. For information about new features in the 7.4 major release, see 第 E.227 节.

E.224.1. Migration to Version 7.4.3

A dump/restore is not required for those running 7.4.X.

E.224.2. Changes

- Fix temporary memory leak when using non-hashed aggregates (Tom)
- ECPG fixes, including some for Informix compatibility (Michael)
- Fixes for compiling with thread-safety, particularly Solaris (Bruce)
- Fix error in COPY IN termination when using the old network protocol (ljb)
- Several important fixes in pg_autovacuum, including fixes for large tables, unsigned oids, stability, temp tables, and debug mode (Matthew T. O'Connor)
- Fix problem with reading tar-format dumps on NetBSD and BSD/OS (Bruce)
- Several JDBC fixes
- Fix ALTER SEQUENCE RESTART where last_value equals the restart value (Tom)
- Repair failure to recalculate nested sub-selects (Tom)
- Fix problems with non-constant expressions in LIMIT/OFFSET
- Support FULL JOIN with no join clause, such as X FULL JOIN Y ON TRUE (Tom)
- Fix another zero-column table bug (Tom)
- Improve handling of non-qualified identifiers in GROUP BY clauses in sub-selects (Tom)

  Select-list aliases within the sub-select will now take precedence over names from outer query levels.
- Do not generate “NATURAL CROSS JOIN” when decompiling rules (Tom)
- Add checks for invalid field length in binary COPY (Tom)

This fixes a difficult-to-exploit security hole.

- Avoid locking conflict between `ANALYZE` and `LISTEN/NOTIFY`
- Numerous translation updates (various contributors)

E.225. Release 7.4.2

Release date: 2004-03-08

This release contains a variety of fixes from 7.4.1. For information about new features in the 7.4 major release, see 第 E.227 节.

E.225.1. Migration to Version 7.4.2

A dump/restore is not required for those running 7.4.X. However, it might be advisable as the easiest method of incorporating fixes for two errors that have been found in the initial contents of 7.4.X system catalogs. A dump/initdb/reload sequence using 7.4.2's initdb will automatically correct these problems.

The more severe of the two errors is that data type `anyarray` has the wrong alignment label; this is a problem because the `pg_statistic` system catalog uses `anyarray` columns. The mislabeling can cause planner misestimations and even crashes when planning queries that involve `WHERE` clauses on double-aligned columns (such as `float8` and `timestamp`). It is strongly recommended that all installations repair this error, either by initdb or by following the manual repair procedure given below.

The lesser error is that the system view `pg_settings` ought to be marked as having public update access, to allow `UPDATE pg_settings` to be used as a substitute for `SET`. This can also be fixed either by initdb or manually, but it is not necessary to fix unless you want to use `UPDATE pg_settings`.

If you wish not to do an initdb, the following procedure will work for fixing `pg_statistic`. As the database superuser, do:

```
-- clear out old data in pg_statistic:
DELETE FROM pg_statistic;
VACUUM pg_statistic;
-- this should update 1 row:
UPDATE pg_type SET typalign = 'd' WHERE oid = 2277;
-- this should update 6 rows:
UPDATE pg_attribute SET attalign = 'd' WHERE atttypid = 2277;
--
-- At this point you MUST start a fresh backend to avoid a crash!
--
-- repopulate pg_statistic:
ANALYZE;
```

This can be done in a live database, but beware that all backends running in the altered database must be restarted before it is safe to repopulate `pg_statistic`.

To repair the `pg_settings` error, simply do:

```
GRANT SELECT, UPDATE ON pg_settings TO PUBLIC;
```

The above procedures must be carried out in each database of an installation, including `template1`, and ideally including `template0` as well. If you do not fix the template databases then any subsequently created databases will contain the same

errors. `template1` can be fixed in the same way as any other database, but fixing `template0` requires additional steps. First, from any database issue:

```
UPDATE pg_database SET datallowconn = true WHERE datname =
 'template0';
```

Next connect to `template0` and perform the above repair procedures. Finally, do:

```
-- re-freeze template0:
VACUUM FREEZE;
-- and protect it against future alterations:
UPDATE pg_database SET datallowconn = false WHERE datname =
 'template0';
```

E.225.2. Changes

Release 7.4.2 incorporates all the fixes included in release 7.3.6, plus the following fixes:

- Fix `pg_statistics` alignment bug that could crash optimizer

  See above for details about this problem.

- Allow non-super users to update `pg_settings`
- Fix several optimizer bugs, most of which led to "variable not found in subplan target lists" errors
- Avoid out-of-memory failure during startup of large multiple index scan
- Fix multibyte problem that could lead to "out of memory" error during `COPY IN`
- Fix problems with `SELECT INTO` / `CREATE TABLE AS` from tables without OIDs
- Fix problems with `alter_table` regression test during parallel testing
- Fix problems with hitting open file limit, especially on OS X (Tom)
- Partial fix for Turkish-locale issues

  initdb will succeed now in Turkish locale, but there are still some inconveniences associated with the `i/I` problem.

- Make pg_dump set client encoding on restore
- Other minor pg_dump fixes
- Allow ecpg to again use C keywords as column names (Michael)
- Added ecpg `WHENEVER NOT_FOUND` to `SELECT/INSERT/UPDATE/DELETE` (Michael)
- Fix ecpg crash for queries calling set-returning functions (Michael)
- Various other ecpg fixes (Michael)
- Fixes for Borland compiler
- Thread build improvements (Bruce)
- Various other build fixes
- Various JDBC fixes

E.226. Release 7.4.1

Release date: 2003-12-22

This release contains a variety of fixes from 7.4. For information about new features in the 7.4 major release, see 第 E.227 节.

E.226.1. Migration to Version 7.4.1

A dump/restore is not required for those running 7.4.

If you want to install the fixes in the information schema you need to reload it into the database. This is either accomplished by initializing a new cluster by running `initdb`, or by running the following sequence of SQL commands in each database (ideally including `template1`) as a superuser in psql, after installing the new release:

```
DROP SCHEMA information_schema CASCADE;
\i /usr/local/pgsql/share/information_schema.sql
```

Substitute your installation path in the second command.

E.226.2. Changes

- Fixed bug in `CREATE SCHEMA` parsing in ECPG (Michael)
- Fix compile error when `--enable-thread-safety` and `--with-perl` are used together (Peter)
- Fix for subqueries that used hash joins (Tom)

  Certain subqueries that used hash joins would crash because of improperly shared structures.
- Fix free space map compaction bug (Tom)

  This fixes a bug where compaction of the free space map could lead to a database server shutdown.
- Fix for Borland compiler build of libpq (Bruce)
- Fix `netmask()` and `hostmask()` to return the maximum-length masklen (Tom)

  Fix these functions to return values consistent with pre-7.4 releases.
- Several `contrib/pg_autovacuum` fixes

  Fixes include improper variable initialization, missing vacuum after `TRUNCATE`, and duration computation overflow for long vacuums.
- Allow compile of `contrib/cube` under Cygwin (Jason Tishler)
- Fix Solaris use of password file when no passwords are defined (Tom)

  Fix crash on Solaris caused by use of any type of password authentication when no passwords were defined.
- JDBC fix for thread problems, other fixes
- Fix for `bytea` index lookups (Joe)
- Fix information schema for bit data types (Peter)

- Force zero_damaged_pages to be on during recovery from WAL
- Prevent some obscure cases of "variable not in subplan target lists"
- Make `PQescapeBytea` and `byteaout` consistent with each other (Joe)
- Escape `bytea` output for bytes > 0x7e(Joe)

  If different client encodings are used for `bytea` output and input, it is possible for `bytea` values to be corrupted by the differing encodings. This fix escapes all bytes that might be affected.
- Added missing `SPI_finish()` calls to dblink's `get_tuple_of_interest()` (Joe)
- New Czech FAQ
- Fix information schema view `constraint_column_usage` for foreign keys (Peter)
- ECPG fixes (Michael)
- Fix bug with multiple `IN` subqueries and joins in the subqueries (Tom)
- Allow `COUNT('x')` to work (Tom)
- Install ECPG include files for Informix compatibility into separate directory (Peter)

  Some names of ECPG include files for Informix compatibility conflicted with operating system include files. By installing them in their own directory, name conflicts have been reduced.
- Fix SSL memory leak (Neil)

  This release fixes a bug in 7.4 where SSL didn't free all memory it allocated.
- Prevent `pg_service.conf` from using service name as default dbname (Bruce)
- Fix local ident authentication on FreeBSD (Tom)

E.227. Release 7.4

Release date: 2003-11-17

E.227.1. Overview

Major changes in this release:

`IN` / `NOT IN` subqueries are now much more efficient

In previous releases, `IN`/`NOT IN` subqueries were joined to the upper query by sequentially scanning the subquery looking for a match. The 7.4 code uses the same sophisticated techniques used by ordinary joins and so is much faster. An `IN` will now usually be as fast as or faster than an equivalent `EXISTS` subquery; this reverses the conventional wisdom that applied to previous releases.

Improved `GROUP BY` processing by using hash buckets

In previous releases, rows to be grouped had to be sorted first. The 7.4 code can do `GROUP BY` without sorting, by accumulating results into a hash table with one entry per group. It will still use the sort technique, however, if the hash table is estimated to be too large to fit in `sort_mem`.

New multikey hash join capability

In previous releases, hash joins could only occur on single keys. This release allows multicolumn hash joins.

Queries using the explicit `JOIN` syntax are now better optimized

Prior releases evaluated queries using the explicit `JOIN` syntax only in the order implied by the syntax. 7.4 allows full optimization of these queries, meaning the optimizer considers all possible join orderings and chooses the most efficient. Outer joins, however, must still follow the declared ordering.

Faster and more powerful regular expression code

The entire regular expression module has been replaced with a new version by Henry Spencer, originally written for Tcl. The code greatly improves performance and supports several flavors of regular expressions.

Function-inlining for simple SQL functions

Simple SQL functions can now be inlined by including their SQL in the main query. This improves performance by eliminating per-call overhead. That means simple SQL functions now behave like macros.

Full support for IPv6 connections and IPv6 address data types

Previous releases allowed only IPv4 connections, and the IP data types only supported IPv4 addresses. This release adds full IPv6 support in both of these areas.

Major improvements in SSL performance and reliability

Several people very familiar with the SSL API have overhauled our SSL code to improve SSL key negotiation and error recovery.

Make free space map efficiently reuse empty index pages, and other free space management improvements

In previous releases, B-tree index pages that were left empty because of deleted rows could only be reused by rows with index values similar to the rows originally indexed on that page. In 7.4, `VACUUM` records empty index pages and allows them to be reused for any future index rows.

SQL-standard information schema

The information schema provides a standardized and stable way to access information about the schema objects defined in a database.

Cursors conform more closely to the SQL standard

The commands `FETCH` and `MOVE` have been overhauled to conform more closely to the SQL standard.

Cursors can exist outside transactions

These cursors are also called holdable cursors.

New client-to-server protocol

The new protocol adds error codes, more status information, faster startup, better support for binary data transmission, parameter values separated from

SQL commands, prepared statements available at the protocol level, and cleaner recovery from `COPY` failures. The older protocol is still supported by both server and clients.

libpq and ECPG applications are now fully thread-safe

While previous libpq releases already supported threads, this release improves thread safety by fixing some non-thread-safe code that was used during database connection startup. The `configure` option `--enable-thread-safety` must be used to enable this feature.

New version of full-text indexing

A new full-text indexing suite is available in `contrib/tsearch2`.

New autovacuum tool

The new autovacuum tool in `contrib/autovacuum` monitors the database statistics tables for `INSERT`/`UPDATE`/`DELETE` activity and automatically vacuums tables when needed.

Array handling has been improved and moved into the server core

Many array limitations have been removed, and arrays behave more like fully-supported data types.

E.227.2. Migration to Version 7.4

A dump/restore using pg_dump is required for those wishing to migrate data from any previous release.

Observe the following incompatibilities:

- The server-side autocommit setting was removed and reimplemented in client applications and languages. Server-side autocommit was causing too many problems with languages and applications that wanted to control their own autocommit behavior, so autocommit was removed from the server and added to individual client APIs as appropriate.
- Error message wording has changed substantially in this release. Significant effort was invested to make the messages more consistent and user-oriented. If your applications try to detect different error conditions by parsing the error message, you are strongly encouraged to use the new error code facility instead.
- Inner joins using the explicit `JOIN` syntax might behave differently because they are now better optimized.
- A number of server configuration parameters have been renamed for clarity, primarily those related to logging.
- `FETCH 0` or `MOVE 0` now does nothing. In prior releases, `FETCH 0` would fetch all remaining rows, and `MOVE 0` would move to the end of the cursor.
- `FETCH` and `MOVE` now return the actual number of rows fetched/moved, or zero if at the beginning/end of the cursor. Prior releases would return the row count passed to the command, not the number of rows actually fetched or moved.
- `COPY` now can process files that use carriage-return or carriage-return/line-feed end-of-line sequences. Literal carriage-returns and line-feeds are no longer accepted in data values; use `\r` and `\n` instead.

- Trailing spaces are now trimmed when converting from type `char(n)` to `varchar(n)` or `text`. This is what most people always expected to happen anyway.
- The data type `float(p)` now measures *p* in binary digits, not decimal digits. The new behavior follows the SQL standard.
- Ambiguous date values now must match the ordering specified by the `datestyle` setting. In prior releases, a date specification of `10/20/03` was interpreted as a date in October even if `datestyle` specified that the day should be first. 7.4 will throw an error if a date specification is invalid for the current setting of `datestyle`.
- The functions `oidrand`, `oidsrand`, and `userfntest` have been removed. These functions were determined to be no longer useful.
- String literals specifying time-varying date/time values, such as `'now'` or `'today'` will no longer work as expected in column default expressions; they now cause the time of the table creation to be the default, not the time of the insertion. Functions such as `now()`, `current_timestamp`, or `current_date` should be used instead.

  In previous releases, there was special code so that strings such as `'now'` were interpreted at `INSERT` time and not at table creation time, but this work around didn't cover all cases. Release 7.4 now requires that defaults be defined properly using functions such as `now()` or `current_timestamp`. These will work in all situations.
- The dollar sign (`$`) is no longer allowed in operator names. It can instead be a non-first character in identifiers. This was done to improve compatibility with other database systems, and to avoid syntax problems when parameter placeholders (`$n`) are written adjacent to operators.

E.227.3. Changes

Below you will find a detailed account of the changes between release 7.4 and the previous major release.

E.227.3.1. Server Operation Changes

- Allow IPv6 server connections (Nigel Kukard, Johan Jordaan, Bruce, Tom, Kurt Roeckx, Andrew Dunstan)
- Fix SSL to handle errors cleanly (Nathan Mueller)

  In prior releases, certain SSL API error reports were not handled correctly. This release fixes those problems.
- SSL protocol security and performance improvements (Sean Chittenden)

  SSL key renegotiation was happening too frequently, causing poor SSL performance. Also, initial key handling was improved.
- Print lock information when a deadlock is detected (Tom)

  This allows easier debugging of deadlock situations.
- Update `/tmp` socket modification times regularly to avoid their removal (Tom)

  This should help prevent `/tmp` directory cleaner administration scripts from removing server socket files.
- Enable PAM for Mac OS X (Aaron Hillegass)
- Make B-tree indexes fully WAL-safe (Tom)

In prior releases, under certain rare cases, a server crash could cause B-tree indexes to become corrupt. This release removes those last few rare cases.

- Allow B-tree index compaction and empty page reuse (Tom)
- Fix inconsistent index lookups during split of first root page (Tom)

  In prior releases, when a single-page index split into two pages, there was a brief period when another database session could miss seeing an index entry. This release fixes that rare failure case.

- Improve free space map allocation logic (Tom)
- Preserve free space information between server restarts (Tom)

  In prior releases, the free space map was not saved when the postmaster was stopped, so newly started servers had no free space information. This release saves the free space map, and reloads it when the server is restarted.

- Add start time to `pg_stat_activity` (Neil)
- New code to detect corrupt disk pages; erase with `zero_damaged_pages` (Tom)
- New client/server protocol: faster, no username length limit, allow clean exit from `COPY` (Tom)
- Add transaction status, table ID, column ID to client/server protocol (Tom)
- Add binary I/O to client/server protocol (Tom)
- Remove autocommit server setting; move to client applications (Tom)
- New error message wording, error codes, and three levels of error detail (Tom, Joe, Peter)

E.227.3.2. Performance Improvements

- Add hashing for `GROUP BY` aggregates (Tom)
- Make nested-loop joins be smarter about multicolumn indexes (Tom)
- Allow multikey hash joins (Tom)
- Improve constant folding (Tom)
- Add ability to inline simple SQL functions (Tom)
- Reduce memory usage for queries using complex functions (Tom)

  In prior releases, functions returning allocated memory would not free it until the query completed. This release allows the freeing of function-allocated memory when the function call completes, reducing the total memory used by functions.

- Improve GEQO optimizer performance (Tom)

  This release fixes several inefficiencies in the way the GEQO optimizer manages potential query paths.

- Allow `IN`/`NOT IN` to be handled via hash tables (Tom)
- Improve `NOT IN (subquery)` performance (Tom)
- Allow most `IN` subqueries to be processed as joins (Tom)

- Pattern matching operations can use indexes regardless of locale (Peter)

  There is no way for non-ASCII locales to use the standard indexes for `LIKE` comparisons. This release adds a way to create a special index for `LIKE`.

- Allow the postmaster to preload libraries using `preload_libraries` (Joe)

  For shared libraries that require a long time to load, this option is available so the library can be preloaded in the postmaster and inherited by all database sessions.

- Improve optimizer cost computations, particularly for subqueries (Tom)
- Avoid sort when subquery `ORDER BY` matches upper query (Tom)
- Deduce that `WHERE a.x = b.y AND b.y = 42` also means `a.x = 42` (Tom)
- Allow hash/merge joins on complex joins (Tom)
- Allow hash joins for more data types (Tom)
- Allow join optimization of explicit inner joins, disable with `join_collapse_limit` (Tom)
- Add parameter `from_collapse_limit` to control conversion of subqueries to joins (Tom)
- Use faster and more powerful regular expression code from Tcl (Henry Spencer, Tom)
- Use bit-mapped relation sets in the optimizer (Tom)
- Improve connection startup time (Tom)

  The new client/server protocol requires fewer network packets to start a database session.

- Improve trigger/constraint performance (Stephan)
- Improve speed of `col IN (const, const, const, ...)` (Tom)
- Fix hash indexes which were broken in rare cases (Tom)
- Improve hash index concurrency and speed (Tom)

  Prior releases suffered from poor hash index performance, particularly for high concurrency situations. This release fixes that, and the development group is interested in reports comparing B-tree and hash index performance.

- Align shared buffers on 32-byte boundary for copy speed improvement (Manfred Spraul)

  Certain CPU's perform faster data copies when addresses are 32-byte aligned.

- Data type `numeric` reimplemented for better performance (Tom)

  `numeric` used to be stored in base 100. The new code uses base 10000, for significantly better performance.

E.227.3.3. Server Configuration Changes

- Rename server parameter `server_min_messages` to `log_min_messages` (Bruce)

  This was done so most parameters that control the server logs begin with `log_`.

- Rename `show_*_stats` to `log_*_stats` (Bruce)
- Rename `show_source_port` to `log_source_port` (Bruce)
- Rename `hostname_lookup` to `log_hostname` (Bruce)
- Add `checkpoint_warning` to warn of excessive checkpointing (Bruce)

  In prior releases, it was difficult to determine if checkpoint was happening too frequently. This feature adds a warning to the server logs when excessive checkpointing happens.
- New read-only server parameters for localization (Tom)
- Change debug server log messages to output as `DEBUG` rather than `LOG` (Bruce)
- Prevent server log variables from being turned off by non-superusers (Bruce)

  This is a security feature so non-superusers cannot disable logging that was enabled by the administrator.
- `log_min_messages`/`client_min_messages` now controls `debug_*` output (Bruce)

  This centralizes client debug information so all debug output can be sent to either the client or server logs.
- Add Mac OS X Rendezvous server support (Chris Campbell)

  This allows Mac OS X hosts to query the network for available PostgreSQL servers.
- Add ability to print only slow statements using `log_min_duration_statement` (Christopher)

  This is an often requested debugging feature that allows administrators to see only slow queries in their server logs.
- Allow `pg_hba.conf` to accept netmasks in CIDR format (Andrew Dunstan)

  This allows administrators to merge the host IP address and netmask fields into a single CIDR field in `pg_hba.conf`.
- New read-only parameter `is_superuser` (Tom)
- New parameter `log_error_verbosity` to control error detail (Tom)

  This works with the new error reporting feature to supply additional error information like hints, file names and line numbers.
- `postgres --describe-config` now dumps server config variables (Aizaz Ahmed, Peter)

  This option is useful for administration tools that need to know the configuration variable names and their minimums, maximums, defaults, and descriptions.
- Add new columns in `pg_settings`: `context`, `type`, `source`, `min_val`, `max_val` (Joe)
- Make default `shared_buffers` 1000 and `max_connections` 100, if possible (Tom)

  Prior versions defaulted to 64 shared buffers so PostgreSQL would start on even very old systems. This release tests the amount of shared memory allowed by the platform and selects more reasonable default values if possible. Of course, users

are still encouraged to evaluate their resource load and size `shared_buffers` accordingly.

- New `pg_hba.conf` record type `hostnossl` to prevent SSL connections (Jon Jensen)

  In prior releases, there was no way to prevent SSL connections if both the client and server supported SSL. This option allows that capability.

- Remove parameter `geqo_random_seed` (Tom)
- Add server parameter `regex_flavor` to control regular expression processing (Tom)
- Make `pg_ctl` better handle nonstandard ports (Greg)

E.227.3.4. Query Changes

- New SQL-standard information schema (Peter)
- Add read-only transactions (Peter)
- Print key name and value in foreign-key violation messages (Dmitry Tkach)
- Allow users to see their own queries in `pg_stat_activity` (Kevin Brown)

  In prior releases, only the superuser could see query strings using `pg_stat_activity`. Now ordinary users can see their own query strings.

- Fix aggregates in subqueries to match SQL standard (Tom)

  The SQL standard says that an aggregate function appearing within a nested subquery belongs to the outer query if its argument contains only outer-query variables. Prior PostgreSQL releases did not handle this fine point correctly.

- Add option to prevent auto-addition of tables referenced in query (Nigel J. Andrews)

  By default, tables mentioned in the query are automatically added to the `FROM` clause if they are not already there. This is compatible with historic POSTGRES behavior but is contrary to the SQL standard. This option allows selecting standard-compatible behavior.

- Allow `UPDATE ... SET col = DEFAULT` (Rod)

  This allows `UPDATE` to set a column to its declared default value.

- Allow expressions to be used in `LIMIT/OFFSET` (Tom)

  In prior releases, `LIMIT/OFFSET` could only use constants, not expressions.

- Implement `CREATE TABLE AS EXECUTE` (Neil, Peter)

E.227.3.5. Object Manipulation Changes

- Make `CREATE SEQUENCE` grammar more conforming to SQL:2003 (Neil)
- Add statement-level triggers (Neil)

  While this allows a trigger to fire at the end of a statement, it does not allow the trigger to access all rows modified by the statement. This capability is planned for a future release.

- Add check constraints for domains (Rod)

This greatly increases the usefulness of domains by allowing them to use check constraints.

- Add `ALTER DOMAIN` (Rod)

  This allows manipulation of existing domains.

- Fix several zero-column table bugs (Tom)

  PostgreSQL supports zero-column tables. This fixes various bugs that occur when using such tables.

- Have `ALTER TABLE ... ADD PRIMARY KEY` add not-null constraint (Rod)

  In prior releases, `ALTER TABLE ... ADD PRIMARY` would add a unique index, but not a not-null constraint. That is fixed in this release.

- Add `ALTER TABLE ... WITHOUT OIDS` (Rod)

  This allows control over whether new and updated rows will have an OID column. This is most useful for saving storage space.

- Add `ALTER SEQUENCE` to modify minimum, maximum, increment, cache, cycle values (Rod)

- Add `ALTER TABLE ... CLUSTER ON` (Alvaro Herrera)

  This command is used by `pg_dump` to record the cluster column for each table previously clustered. This information is used by database-wide cluster to cluster all previously clustered tables.

- Improve automatic type casting for domains (Rod, Tom)

- Allow dollar signs in identifiers, except as first character (Tom)

- Disallow dollar signs in operator names, so `x=$1` works (Tom)

- Allow copying table schema using `LIKE` *`subtable`*, also SQL:2003 feature `INCLUDING DEFAULTS` (Rod)

- Add `WITH GRANT OPTION` clause to `GRANT` (Peter)

  This enabled `GRANT` to give other users the ability to grant privileges on an object.

E.227.3.6. Utility Command Changes

- Add `ON COMMIT` clause to `CREATE TABLE` for temporary tables (Gavin)

  This adds the ability for a table to be dropped or all rows deleted on transaction commit.

- Allow cursors outside transactions using `WITH HOLD` (Neil)

  In previous releases, cursors were removed at the end of the transaction that created them. Cursors can now be created with the `WITH HOLD` option, which allows them to continue to be accessed after the creating transaction has committed.

- `FETCH 0` and `MOVE 0` now do nothing (Bruce)

  In previous releases, `FETCH 0` fetched all remaining rows, and `MOVE 0` moved to the end of the cursor.

- Cause `FETCH` and `MOVE` to return the number of rows fetched/moved, or zero if at the beginning/end of cursor, per SQL standard (Bruce)

  In prior releases, the row count returned by `FETCH` and `MOVE` did not accurately reflect the number of rows processed.

- Properly handle `SCROLL` with cursors, or report an error (Neil)

  Allowing random access (both forward and backward scrolling) to some kinds of queries cannot be done without some additional work. If `SCROLL` is specified when the cursor is created, this additional work will be performed. Furthermore, if the cursor has been created with `NO SCROLL`, no random access is allowed.

- Implement SQL-compatible options `FIRST`, `LAST`, `ABSOLUTE` *n*, `RELATIVE` *n* for `FETCH` and `MOVE` (Tom)

- Allow `EXPLAIN` on `DECLARE CURSOR` (Tom)

- Allow `CLUSTER` to use index marked as pre-clustered by default (Alvaro Herrera)

- Allow `CLUSTER` to cluster all tables (Alvaro Herrera)

  This allows all previously clustered tables in a database to be reclustered with a single command.

- Prevent `CLUSTER` on partial indexes (Tom)

- Allow DOS and Mac line-endings in `COPY` files (Bruce)

- Disallow literal carriage return as a data value, backslash-carriage-return and `\r` are still allowed (Bruce)

- `COPY` changes (binary, `\.`) (Tom)

- Recover from `COPY` failure cleanly (Tom)

- Prevent possible memory leaks in `COPY` (Tom)

- Make `TRUNCATE` transaction-safe (Rod)

  `TRUNCATE` can now be used inside a transaction. If the transaction aborts, the changes made by the `TRUNCATE` are automatically rolled back.

- Allow prepare/bind of utility commands like `FETCH` and `EXPLAIN` (Tom)

- Add `EXPLAIN EXECUTE` (Neil)

- Improve `VACUUM` performance on indexes by reducing WAL traffic (Tom)

- Functional indexes have been generalized into indexes on expressions (Tom)

  In prior releases, functional indexes only supported a simple function applied to one or more column names. This release allows any type of scalar expression.

- Have `SHOW TRANSACTION ISOLATION` match input to `SET TRANSACTION ISOLATION` (Tom)

- Have `COMMENT ON DATABASE` on nonlocal database generate a warning, rather than an error (Rod)

  Database comments are stored in database-local tables so comments on a database have to be stored in each database.

- Improve reliability of `LISTEN/NOTIFY` (Tom)
- Allow `REINDEX` to reliably reindex nonshared system catalog indexes (Tom)

  This allows system tables to be reindexed without the requirement of a standalone session, which was necessary in previous releases. The only tables that now require a standalone session for reindexing are the global system tables `pg_database`, `pg_shadow`, and `pg_group`.

E.227.3.7. Data Type and Function Changes

- New server parameter `extra_float_digits` to control precision display of floating-point numbers (Pedro Ferreira, Tom)

  This controls output precision which was causing regression testing problems.
- Allow `+1300` as a numeric time-zone specifier, for FJST (Tom)
- Remove rarely used functions `oidrand`, `oidsrand`, and `userfntest` functions (Neil)
- Add `md5()` function to main server, already in `contrib/pgcrypto` (Joe)

  An MD5 function was frequently requested. For more complex encryption capabilities, use `contrib/pgcrypto`.
- Increase date range of `timestamp` (John Cochran)
- Change `EXTRACT(EPOCH FROM timestamp)` so `timestamp without time zone` is assumed to be in local time, not GMT (Tom)
- Trap division by zero in case the operating system doesn't prevent it (Tom)
- Change the `numeric` data type internally to base 10000 (Tom)
- New `hostmask()` function (Greg Wickham)
- Fixes for `to_char()` and `to_timestamp()` (Karel)
- Allow functions that can take any argument data type and return any data type, using `anyelement` and `anyarray` (Joe)

  This allows the creation of functions that can work with any data type.
- Arrays can now be specified as `ARRAY[1,2,3]`, `ARRAY[['a','b'],['c','d']]`, or `ARRAY[ARRAY[ARRAY[2]]]` (Joe)
- Allow proper comparisons for arrays, including `ORDER BY` and `DISTINCT` support (Joe)
- Allow indexes on array columns (Joe)
- Allow array concatenation with `||` (Joe)
- Allow `WHERE` qualification *`expr op`* `ANY/SOME/ALL (`*`array_expr`*`)` (Joe)

  This allows arrays to behave like a list of values, for purposes like `SELECT * FROM tab WHERE col IN (array_val)`.
- New array functions `array_append`, `array_cat`, `array_lower`, `array_prepend`, `array_to_string`, `array_upper`, `string_to_array` (Joe)

- Allow user defined aggregates to use polymorphic functions (Joe)
- Allow assignments to empty arrays (Joe)
- Allow 60 in seconds fields of `time`, `timestamp`, and `interval` input values (Tom)

  Sixty-second values are needed for leap seconds.
- Allow `cidr` data type to be cast to `text` (Tom)
- Disallow invalid time zone names in SET TIMEZONE
- Trim trailing spaces when `char` is cast to `varchar` or `text` (Tom)
- Make `float(p)` measure the precision *p* in binary digits, not decimal digits (Tom)
- Add IPv6 support to the `inet` and `cidr` data types (Michael Graff)
- Add `family()` function to report whether address is IPv4 or IPv6 (Michael Graff)
- Have `SHOW datestyle` generate output similar to that used by `SET datestyle` (Tom)
- Make `EXTRACT(TIMEZONE)` and `SET/SHOW TIME ZONE` follow the SQL convention for the sign of time zone offsets, i.e., positive is east from UTC (Tom)
- Fix `date_trunc('quarter', ...)` (Böjthe Zoltán)

  Prior releases returned an incorrect value for this function call.
- Make `initcap()` more compatible with Oracle (Mike Nolan)

  `initcap()` now uppercases a letter appearing after any non-alphanumeric character, rather than only after whitespace.
- Allow only `datestyle` field order for date values not in ISO-8601 format (Greg)
- Add new `datestyle` values `MDY`, `DMY`, and `YMD` to set input field order; honor `US` and `European` for backward compatibility (Tom)
- String literals like `'now'` or `'today'` will no longer work as a column default. Use functions such as `now()`, `current_timestamp` instead. (change required for prepared statements) (Tom)
- Treat NaN as larger than any other value in `min()`/`max()` (Tom)

  NaN was already sorted after ordinary numeric values for most purposes, but `min()` and `max()` didn't get this right.
- Prevent interval from suppressing `:00` seconds display
- New functions `pg_get_triggerdef(prettyprint)` and `pg_conversion_is_visible()` (Christopher)
- Allow time to be specified as `040506` or `0405` (Tom)
- Input date order must now be `YYYY-MM-DD` (with 4-digit year) or match `datestyle`
- Make `pg_get_constraintdef` support unique, primary-key, and check constraints (Christopher)

E.227.3.8. Server-Side Language Changes

- Prevent PL/pgSQL crash when `RETURN NEXT` is used on a zero-row record variable (Tom)
- Make PL/Python's `spi_execute` interface handle null values properly (Andrew Bosma)
- Allow PL/pgSQL to declare variables of composite types without `%ROWTYPE` (Tom)
- Fix PL/Python's `_quote()` function to handle big integers
- Make PL/Python an untrusted language, now called `plpythonu` (Kevin Jacobs, Tom)

  The Python language no longer supports a restricted execution environment, so the trusted version of PL/Python was removed. If this situation changes, a version of PL/Python that can be used by non-superusers will be readded.
- Allow polymorphic PL/pgSQL functions (Joe, Tom)
- Allow polymorphic SQL functions (Joe)
- Improved compiled function caching mechanism in PL/pgSQL with full support for polymorphism (Joe)
- Add new parameter `$0` in PL/pgSQL representing the function's actual return type (Joe)
- Allow PL/Tcl and PL/Python to use the same trigger on multiple tables (Tom)
- Fixed PL/Tcl's `spi_prepare` to accept fully qualified type names in the parameter type list (Jan)

E.227.3.9. psql Changes

- Add `\pset pager always` to always use pager (Greg)

  This forces the pager to be used even if the number of rows is less than the screen height. This is valuable for rows that wrap across several screen rows.
- Improve tab completion (Rod, Ross Reedstrom, Ian Barwick)
- Reorder `\?` help into groupings (Harald Armin Massa, Bruce)
- Add backslash commands for listing schemas, casts, and conversions (Christopher)
- `\encoding` now changes based on the server parameter `client_encoding` (Tom)

  In previous versions, `\encoding` was not aware of encoding changes made using `SET client_encoding`.
- Save editor buffer into readline history (Ross)

  When `\e` is used to edit a query, the result is saved in the readline history for retrieval using the up arrow.
- Improve `\d` display (Christopher)
- Enhance HTML mode to be more standards-conforming (Greg)
- New `\set AUTOCOMMIT off` capability (Tom)

This takes the place of the removed server parameter `autocommit`.

- New `\set VERBOSITY` to control error detail (Tom)

  This controls the new error reporting details.

- New prompt escape sequence `%x` to show transaction status (Tom)
- Long options for psql are now available on all platforms

E.227.3.10. pg_dump Changes

- Multiple pg_dump fixes, including tar format and large objects
- Allow pg_dump to dump specific schemas (Neil)
- Make pg_dump preserve column storage characteristics (Christopher)

  This preserves `ALTER TABLE ... SET STORAGE` information.

- Make pg_dump preserve `CLUSTER` characteristics (Christopher)
- Have pg_dumpall use `GRANT`/`REVOKE` to dump database-level privileges (Tom)
- Allow pg_dumpall to support the options `-a`, `-s`, `-x` of pg_dump (Tom)
- Prevent pg_dump from lowercasing identifiers specified on the command line (Tom)
- pg_dump options `--use-set-session-authorization` and `--no-reconnect` now do nothing, all dumps use `SET SESSION AUTHORIZATION`

  pg_dump no longer reconnects to switch users, but instead always uses `SET SESSION AUTHORIZATION`. This will reduce password prompting during restores.

- Long options for pg_dump are now available on all platforms

  PostgreSQL now includes its own long-option processing routines.

E.227.3.11. libpq Changes

- Add function `PQfreemem` for freeing memory on Windows, suggested for `NOTIFY` (Bruce)

  Windows requires that memory allocated in a library be freed by a function in the same library, hence `free()` doesn't work for freeing memory allocated by libpq. `PQfreemem` is the proper way to free libpq memory, especially on Windows, and is recommended for other platforms as well.

- Document service capability, and add sample file (Bruce)

  This allows clients to look up connection information in a central file on the client machine.

- Make `PQsetdbLogin` have the same defaults as `PQconnectdb` (Tom)
- Allow libpq to cleanly fail when result sets are too large (Tom)
- Improve performance of function `PQunescapeBytea` (Ben Lamb)
- Allow thread-safe libpq with `configure` option `--enable-thread-safety` (Lee Kindness, Philip Yarra)

- Allow function `pqInternalNotice` to accept a format string and arguments instead of just a preformatted message (Tom, Sean Chittenden)
- Control SSL negotiation with `sslmode` values `disable`, `allow`, `prefer`, and `require` (Jon Jensen)
- Allow new error codes and levels of text (Tom)
- Allow access to the underlying table and column of a query result (Tom)

  This is helpful for query-builder applications that want to know the underlying table and column names associated with a specific result set.
- Allow access to the current transaction status (Tom)
- Add ability to pass binary data directly to the server (Tom)
- Add function `PQexecPrepared` and `PQsendQueryPrepared` functions which perform bind/execute of previously prepared statements (Tom)

E.227.3.12. JDBC Changes

- Allow `setNull` on updateable result sets
- Allow `executeBatch` on a prepared statement (Barry)
- Support SSL connections (Barry)
- Handle schema names in result sets (Paul Sorenson)
- Add refcursor support (Nic Ferrier)

E.227.3.13. Miscellaneous Interface Changes

- Prevent possible memory leak or core dump during libpgtcl shutdown (Tom)
- Add Informix compatibility to ECPG (Michael)

  This allows ECPG to process embedded C programs that were written using certain Informix extensions.
- Add type `decimal` to ECPG that is fixed length, for Informix (Michael)
- Allow thread-safe embedded SQL programs with `configure` option `--enable-thread-safety` (Lee Kindness, Bruce)

  This allows multiple threads to access the database at the same time.
- Moved Python client PyGreSQL to http://www.pygresql.org (Marc)

E.227.3.14. Source Code Changes

- Prevent need for separate platform geometry regression result files (Tom)
- Improved PPC locking primitive (Reinhard Max)
- New function `palloc0` to allocate and clear memory (Bruce)
- Fix locking code for s390x CPU (64-bit) (Tom)
- Allow OpenBSD to use local ident credentials (William Ahern)

- Make query plan trees read-only to executor (Tom)
- Add Darwin startup scripts (David Wheeler)
- Allow libpq to compile with Borland C++ compiler (Lester Godwin, Karl Waclawek)
- Use our own version of `getopt_long()` if needed (Peter)
- Convert administration scripts to C (Peter)
- Bison >= 1.85 is now required to build the PostgreSQL grammar, if building from CVS
- Merge documentation into one book (Peter)
- Add Windows compatibility functions (Bruce)
- Allow client interfaces to compile under MinGW (Bruce)
- New `ereport()` function for error reporting (Tom)
- Support Intel compiler on Linux (Peter)
- Improve Linux startup scripts (Slawomir Sudnik, Darko Prenosil)
- Add support for AMD Opteron and Itanium (Jeffrey W. Baker, Bruce)
- Remove `--enable-recode` option from `configure`

  This was no longer needed now that we have `CREATE CONVERSION`.
- Generate a compile error if spinlock code is not found (Bruce)

  Platforms without spinlock code will now fail to compile, rather than silently using semaphores. This failure can be disabled with a new `configure` option.

E.227.3.15. Contrib Changes

- Change dbmirror license to BSD
- Improve earthdistance (Bruno Wolff III)
- Portability improvements to pgcrypto (Marko Kreen)
- Prevent crash in xml (John Gray, Michael Richards)
- Update oracle
- Update mysql
- Update cube (Bruno Wolff III)
- Update earthdistance to use cube (Bruno Wolff III)
- Update btree_gist (Oleg)
- New tsearch2 full-text search module (Oleg, Teodor)
- Add hash-based crosstab function to tablefuncs (Joe)
- Add serial column to order `connectby()` siblings in tablefuncs (Nabil Sayegh,Joe)
- Add named persistent connections to dblink (Shridhar Daithanka)
- New pg_autovacuum allows automatic `VACUUM` (Matthew T. O'Connor)

- Make pgbench honor environment variables `PGHOST`, `PGPORT`, `PGUSER` (Tatsuo)
- Improve intarray (Teodor Sigaev)
- Improve pgstattuple (Rod)
- Fix bug in `metaphone()` in fuzzystrmatch
- Improve adddepend (Rod)
- Update spi/timetravel (Böjthe Zoltán)
- Fix dbase `-s` option and improve non-ASCII handling (Thomas Behr, Márcio Smiderle)
- Remove array module because features now included by default (Joe)

E.228. Release 7.3.21

Release date: 2008-01-07

This release contains a variety of fixes from 7.3.20, including fixes for significant security issues.

This is expected to be the last PostgreSQL release in the 7.3.X series. Users are encouraged to update to a newer release branch soon.

E.228.1. Migration to Version 7.3.21

A dump/restore is not required for those running 7.3.X. However, if you are upgrading from a version earlier than 7.3.13, see 第 E.236 节.

E.228.2. Changes

- Prevent functions in indexes from executing with the privileges of the user running `VACUUM`, `ANALYZE`, etc (Tom)

  Functions used in index expressions and partial-index predicates are evaluated whenever a new table entry is made. It has long been understood that this poses a risk of trojan-horse code execution if one modifies a table owned by an untrustworthy user. (Note that triggers, defaults, check constraints, etc. pose the same type of risk.) But functions in indexes pose extra danger because they will be executed by routine maintenance operations such as `VACUUM FULL`, which are commonly performed automatically under a superuser account. For example, a nefarious user can execute code with superuser privileges by setting up a trojan-horse index definition and waiting for the next routine vacuum. The fix arranges for standard maintenance operations (including `VACUUM`, `ANALYZE`, `REINDEX`, and `CLUSTER`) to execute as the table owner rather than the calling user, using the same privilege-switching mechanism already used for `SECURITY DEFINER` functions. To prevent bypassing this security measure, execution of `SET SESSION AUTHORIZATION` and `SET ROLE` is now forbidden within a `SECURITY DEFINER` context. (CVE-2007-6600)

- Require non-superusers who use `/contrib/dblink` to use only password authentication, as a security measure (Joe)

  The fix that appeared for this in 7.3.20 was incomplete, as it plugged the hole for only some `dblink` functions. (CVE-2007-6601, CVE-2007-3278)

- Fix potential crash in `translate()` when using a multibyte database encoding (Tom)
- Make `contrib/tablefunc`'s `crosstab()` handle NULL rowid as a category in its own right, rather than crashing (Joe)
- Require a specific version of Autoconf to be used when re-generating the `configure` script (Peter)

  This affects developers and packagers only. The change was made to prevent accidental use of untested combinations of Autoconf and PostgreSQL versions. You can remove the version check if you really want to use a different Autoconf version, but it's your responsibility whether the result works or not.

E.229. Release 7.3.20

Release date: 2007-09-17

This release contains fixes from 7.3.19.

E.229.1. Migration to Version 7.3.20

A dump/restore is not required for those running 7.3.X. However, if you are upgrading from a version earlier than 7.3.13, see 第 E.236 节.

E.229.2. Changes

- Prevent index corruption when a transaction inserts rows and then aborts close to the end of a concurrent `VACUUM` on the same table (Tom)
- Make `CREATE DOMAIN ... DEFAULT NULL` work properly (Tom)
- Fix crash when `log_min_error_statement` logging runs out of memory (Tom)
- Require non-superusers who use `/contrib/dblink` to use only password authentication, as a security measure (Joe)

E.230. Release 7.3.19

Release date: 2007-04-23

This release contains fixes from 7.3.18, including a security fix.

E.230.1. Migration to Version 7.3.19

A dump/restore is not required for those running 7.3.X. However, if you are upgrading from a version earlier than 7.3.13, see 第 E.236 节.

E.230.2. Changes

- Support explicit placement of the temporary-table schema within `search_path`, and disable searching it for functions and operators (Tom)

  This is needed to allow a security-definer function to set a truly secure value of `search_path`. Without it, an unprivileged SQL user can use temporary objects to execute code with the privileges of the security-definer function (CVE-2007-2138). See `CREATE FUNCTION` for more information.

- Fix potential-data-corruption bug in how `VACUUM FULL` handles `UPDATE` chains (Tom, Pavan Deolasee)

E.231. Release 7.3.18

Release date: 2007-02-05

This release contains a variety of fixes from 7.3.17, including a security fix.

E.231.1. Migration to Version 7.3.18

A dump/restore is not required for those running 7.3.X. However, if you are upgrading from a version earlier than 7.3.13, see 第 E.236 节.

E.231.2. Changes

- Remove security vulnerability that allowed connected users to read backend memory (Tom)

  The vulnerability involves changing the data type of a table column used in a SQL function (CVE-2007-0555). This error can easily be exploited to cause a backend crash, and in principle might be used to read database content that the user should not be able to access.

- Fix rare bug wherein btree index page splits could fail due to choosing an infeasible split point (Heikki Linnakangas)

- Tighten security of multi-byte character processing for UTF8 sequences over three bytes long (Tom)

E.232. Release 7.3.17

Release date: 2007-01-08

This release contains a variety of fixes from 7.3.16.

E.232.1. Migration to Version 7.3.17

A dump/restore is not required for those running 7.3.X. However, if you are upgrading from a version earlier than 7.3.13, see 第 E.236 节.

E.232.2. Changes

- `to_number()` and `to_char(numeric)` are now `STABLE`, not `IMMUTABLE`, for new initdb installs (Tom)

  This is because `lc_numeric` can potentially change the output of these functions.

- Improve index usage of regular expressions that use parentheses (Tom)

  This improves psql `\d` performance also.

E.233. Release 7.3.16

Release date: 2006-10-16

This release contains a variety of fixes from 7.3.15.

E.233.1. Migration to Version 7.3.16

A dump/restore is not required for those running 7.3.X. However, if you are upgrading from a version earlier than 7.3.13, see 第 E.236 节.

E.233.2. Changes

- Fix corner cases in pattern matching for psql's `\d` commands
- Fix index-corrupting bugs in /contrib/ltree (Teodor)
- Back-port 7.4 spinlock code to improve performance and support 64-bit architectures better
- Fix SSL-related memory leak in libpq
- Fix backslash escaping in /contrib/dbmirror
- Adjust regression tests for recent changes in US DST laws

E.234. Release 7.3.15

Release date: 2006-05-23

This release contains a variety of fixes from 7.3.14, including patches for extremely serious security issues.

E.234.1. Migration to Version 7.3.15

A dump/restore is not required for those running 7.3.X. However, if you are upgrading from a version earlier than 7.3.13, see 第 E.236 节.

Full security against the SQL-injection attacks described in CVE-2006-2313 and CVE-2006-2314 might require changes in application code. If you have applications that embed untrustworthy strings into SQL commands, you should examine them as soon as possible to ensure that they are using recommended escaping techniques. In most cases, applications should be using subroutines provided by libraries or drivers (such as libpq's `PQescapeStringConn()`) to perform string escaping, rather than relying on ad hoc code to do it.

E.234.2. Changes

- Change the server to reject invalidly-encoded multibyte characters in all cases (Tatsuo, Tom)

  While PostgreSQL has been moving in this direction for some time, the checks are now applied uniformly to all encodings and all textual input, and are now always errors not merely warnings. This change defends against SQL-injection attacks of the type described in CVE-2006-2313.

- Reject unsafe uses of `\'` in string literals

  As a server-side defense against SQL-injection attacks of the type described in CVE-2006-2314, the server now only accepts `''` and not `\'` as a representation of ASCII single quote in SQL string literals. By default, `\'` is rejected only when `client_encoding` is set to a client-only encoding (SJIS, BIG5, GBK, GB18030, or UHC), which is the scenario in which SQL injection is possible. A new configuration parameter `backslash_quote` is available to adjust this behavior when needed. Note that full

security against CVE-2006-2314 might require client-side changes; the purpose of `backslash_quote` is in part to make it obvious that insecure clients are insecure.

- Modify libpq's string-escaping routines to be aware of encoding considerations

  This fixes libpq-using applications for the security issues described in CVE-2006-2313 and CVE-2006-2314. Applications that use multiple PostgreSQL connections concurrently should migrate to `PQescapeStringConn()` and `PQescapeByteaConn()` to ensure that escaping is done correctly for the settings in use in each database connection. Applications that do string escaping "by hand" should be modified to rely on library routines instead.

- Fix some incorrect encoding conversion functions

  `win1251_to_iso`, `alt_to_iso`, `euc_tw_to_big5`, `euc_tw_to_mic`, `mic_to_euc_tw` were all broken to varying extents.

- Clean up stray remaining uses of `\'` in strings (Bruce, Jan)

- Fix server to use custom DH SSL parameters correctly (Michael Fuhr)

- Fix various minor memory leaks

E.235. Release 7.3.14

Release date: 2006-02-14

This release contains a variety of fixes from 7.3.13.

E.235.1. Migration to Version 7.3.14

A dump/restore is not required for those running 7.3.X. However, if you are upgrading from a version earlier than 7.3.13, see 第 E.236 节.

E.235.2. Changes

- Fix potential crash in `SET SESSION AUTHORIZATION` (CVE-2006-0553)

  An unprivileged user could crash the server process, resulting in momentary denial of service to other users, if the server has been compiled with Asserts enabled (which is not the default). Thanks to Akio Ishida for reporting this problem.

- Fix bug with row visibility logic in self-inserted rows (Tom)

  Under rare circumstances a row inserted by the current command could be seen as already valid, when it should not be. Repairs bug created in 7.3.11 release.

- Fix race condition that could lead to "file already exists" errors during pg_clog file creation (Tom)

- Fix to allow restoring dumps that have cross-schema references to custom operators (Tom)

- Portability fix for testing presence of `finite` and `isinf` during configure (Tom)

E.236. Release 7.3.13

Release date: 2006-01-09

This release contains a variety of fixes from 7.3.12.

E.236.1. Migration to Version 7.3.13

A dump/restore is not required for those running 7.3.X. However, if you are upgrading from a version earlier than 7.3.10, see 第 E.239 节. Also, you might need to `REINDEX` indexes on textual columns after updating, if you are affected by the locale or plperl issues described below.

E.236.2. Changes

- Fix character string comparison for locales that consider different character combinations as equal, such as Hungarian (Tom)

  This might require `REINDEX` to fix existing indexes on textual columns.

- Set locale environment variables during postmaster startup to ensure that plperl won't change the locale later

  This fixes a problem that occurred if the postmaster was started with environment variables specifying a different locale than what initdb had been told. Under these conditions, any use of plperl was likely to lead to corrupt indexes. You might need `REINDEX` to fix existing indexes on textual columns if this has happened to you.

- Fix longstanding bug in strpos() and regular expression handling in certain rarely used Asian multi-byte character sets (Tatsuo)

- Fix bug in `/contrib/pgcrypto` gen_salt, which caused it not to use all available salt space for MD5 and XDES algorithms (Marko Kreen, Solar Designer)

  Salts for Blowfish and standard DES are unaffected.

- Fix `/contrib/dblink` to throw an error, rather than crashing, when the number of columns specified is different from what's actually returned by the query (Joe)

E.237. Release 7.3.12

Release date: 2005-12-12

This release contains a variety of fixes from 7.3.11.

E.237.1. Migration to Version 7.3.12

A dump/restore is not required for those running 7.3.X. However, if you are upgrading from a version earlier than 7.3.10, see 第 E.239 节.

E.237.2. Changes

- Fix race condition in transaction log management

  There was a narrow window in which an I/O operation could be initiated for the wrong page, leading to an Assert failure or data corruption.

- `/contrib/ltree` fixes (Teodor)

- Fix longstanding planning error for outer joins

  This bug sometimes caused a bogus error “RIGHT JOIN is only supported with merge-joinable join conditions” .

- Prevent core dump in pg_autovacuum when a table has been dropped

E.238. Release 7.3.11

Release date: 2005-10-04

This release contains a variety of fixes from 7.3.10.

E.238.1. Migration to Version 7.3.11

A dump/restore is not required for those running 7.3.X. However, if you are upgrading from a version earlier than 7.3.10, see 第 E.239 节.

E.238.2. Changes

- Fix error that allowed `VACUUM` to remove `ctid` chains too soon, and add more checking in code that follows `ctid` links

  This fixes a long-standing problem that could cause crashes in very rare circumstances.

- Fix `CHAR()` to properly pad spaces to the specified length when using a multiple-byte character set (Yoshiyuki Asaba)

  In prior releases, the padding of `CHAR()` was incorrect because it only padded to the specified number of bytes without considering how many characters were stored.

- Fix missing rows in queries like `UPDATE a=... WHERE a...` with GiST index on column `a`

- Improve checking for partially-written WAL pages

- Improve robustness of signal handling when SSL is enabled

- Various memory leakage fixes

- Various portability improvements

- Fix PL/pgSQL to handle `var := var` correctly when the variable is of pass-by-reference type

E.239. Release 7.3.10

Release date: 2005-05-09

This release contains a variety of fixes from 7.3.9, including several security-related issues.

E.239.1. Migration to Version 7.3.10

A dump/restore is not required for those running 7.3.X. However, it is one possible way of handling a significant security problem that has been found in the initial contents of 7.3.X system catalogs. A dump/initdb/reload sequence using 7.3.10's initdb will automatically correct this problem.

The security problem is that the built-in character set encoding conversion functions can be invoked from SQL commands by unprivileged users, but the functions were not

designed for such use and are not secure against malicious choices of arguments. The fix involves changing the declared parameter list of these functions so that they can no longer be invoked from SQL commands. (This does not affect their normal use by the encoding conversion machinery.) It is strongly recommended that all installations repair this error, either by initdb or by following the manual repair procedure given below. The error at least allows unprivileged database users to crash their server process, and might allow unprivileged users to gain the privileges of a database superuser.

If you wish not to do an initdb, perform the following procedure instead. As the database superuser, do:

```
BEGIN;
UPDATE pg_proc SET proargtypes[3] = 'internal'::regtype
WHERE pronamespace = 11 AND pronargs = 5
     AND proargtypes[2] = 'cstring'::regtype;
-- The command should report having updated 90 rows;
-- if not, rollback and investigate instead of committing!
COMMIT;
```

The above procedure must be carried out in each database of an installation, including `template1`, and ideally including `template0` as well. If you do not fix the template databases then any subsequently created databases will contain the same error. `template1` can be fixed in the same way as any other database, but fixing `template0` requires additional steps. First, from any database issue:

```
UPDATE pg_database SET datallowconn = true WHERE datname =
 'template0';
```

Next connect to `template0` and perform the above repair procedure. Finally, do:

```
-- re-freeze template0:
VACUUM FREEZE;
-- and protect it against future alterations:
UPDATE pg_database SET datallowconn = false WHERE datname =
 'template0';
```

E.239.2. Changes

- Change encoding function signature to prevent misuse
- Repair ancient race condition that allowed a transaction to be seen as committed for some purposes (eg SELECT FOR UPDATE) slightly sooner than for other purposes

  This is an extremely serious bug since it could lead to apparent data inconsistencies being briefly visible to applications.
- Repair race condition between relation extension and VACUUM

  This could theoretically have caused loss of a page's worth of freshly-inserted data, although the scenario seems of very low probability. There are no known cases of it having caused more than an Assert failure.
- Fix comparisons of `TIME WITH TIME ZONE` values

  The comparison code was wrong in the case where the `--enable-integer-datetimes` configuration switch had been used. NOTE: if you have an index on a `TIME WITH TIME ZONE` column, it will need to be `REINDEX`ed after installing this update, because the fix corrects the sort order of column values.

- Fix `EXTRACT(EPOCH)` for `TIME WITH TIME ZONE` values
- Fix mis-display of negative fractional seconds in `INTERVAL` values

  This error only occurred when the `--enable-integer-datetimes` configuration switch had been used.
- Additional buffer overrun checks in plpgsql (Neil)
- Fix pg_dump to dump trigger names containing `%` correctly (Neil)
- Prevent `to_char(interval)` from dumping core for month-related formats
- Fix `contrib/pgcrypto` for newer OpenSSL builds (Marko Kreen)
- Still more 64-bit fixes for `contrib/intagg`
- Prevent incorrect optimization of functions returning `RECORD`

E.240. Release 7.3.9

Release date: 2005-01-31

This release contains a variety of fixes from 7.3.8, including several security-related issues.

E.240.1. Migration to Version 7.3.9

A dump/restore is not required for those running 7.3.X.

E.240.2. Changes

- Disallow `LOAD` to non-superusers

  On platforms that will automatically execute initialization functions of a shared library (this includes at least Windows and ELF-based Unixen), `LOAD` can be used to make the server execute arbitrary code. Thanks to NGS Software for reporting this.
- Check that creator of an aggregate function has the right to execute the specified transition functions

  This oversight made it possible to bypass denial of EXECUTE permission on a function.
- Fix security and 64-bit issues in contrib/intagg
- Add needed STRICT marking to some contrib functions (Kris Jurka)
- Avoid buffer overrun when plpgsql cursor declaration has too many parameters (Neil)
- Fix planning error for FULL and RIGHT outer joins

  The result of the join was mistakenly supposed to be sorted the same as the left input. This could not only deliver mis-sorted output to the user, but in case of nested merge joins could give outright wrong answers.
- Fix plperl for quote marks in tuple fields
- Fix display of negative intervals in SQL and GERMAN datestyles

E.241. Release 7.3.8

Release date: 2004-10-22

This release contains a variety of fixes from 7.3.7.

E.241.1. Migration to Version 7.3.8

A dump/restore is not required for those running 7.3.X.

E.241.2. Changes

- Repair possible failure to update hint bits on disk

  Under rare circumstances this oversight could lead to "could not access transaction status" failures, which qualifies it as a potential-data-loss bug.

- Ensure that hashed outer join does not miss tuples

  Very large left joins using a hash join plan could fail to output unmatched left-side rows given just the right data distribution.

- Disallow running pg_ctl as root

  This is to guard against any possible security issues.

- Avoid using temp files in /tmp in make_oidjoins_check

  This has been reported as a security issue, though it's hardly worthy of concern since there is no reason for non-developers to use this script anyway.

E.242. Release 7.3.7

Release date: 2004-08-16

This release contains one critical fix over 7.3.6, and some minor items.

E.242.1. Migration to Version 7.3.7

A dump/restore is not required for those running 7.3.X.

E.242.2. Changes

- Prevent possible loss of committed transactions during crash

  Due to insufficient interlocking between transaction commit and checkpointing, it was possible for transactions committed just before the most recent checkpoint to be lost, in whole or in part, following a database crash and restart. This is a serious bug that has existed since PostgreSQL 7.1.

- Remove asymmetrical word processing in tsearch (Teodor)

- Properly schema-qualify function names when pg_dump'ing a CAST

E.243. Release 7.3.6

Release date: 2004-03-02

This release contains a variety of fixes from 7.3.5.

E.243.1. Migration to Version 7.3.6

A dump/restore is not required for those running 7.3.*.

E.243.2. Changes

- Revert erroneous changes in rule permissions checking

  A patch applied in 7.3.3 to fix a corner case in rule permissions checks turns out to have disabled rule-related permissions checks in many not-so-corner cases. This would for example allow users to insert into views they weren't supposed to have permission to insert into. We have therefore reverted the 7.3.3 patch. The original bug will be fixed in 8.0.

- Repair incorrect order of operations in GetNewTransactionId()

  This bug could result in failure under out-of-disk-space conditions, including inability to restart even after disk space is freed.

- Ensure configure selects -fno-strict-aliasing even when an external value for CFLAGS is supplied

  On some platforms, building with -fstrict-aliasing causes bugs.

- Make pg_restore handle 64-bit off_t correctly

  This bug prevented proper restoration from archive files exceeding 4 GB.

- Make contrib/dblink not assume that local and remote type OIDs match (Joe)
- Quote connectby()'s start_with argument properly (Joe)
- Don't crash when a rowtype argument to a plpgsql function is NULL
- Avoid generating invalid character encoding sequences in corner cases when planning LIKE operations
- Ensure text_position() cannot scan past end of source string in multibyte cases (Korea PostgreSQL Users' Group)
- Fix index optimization and selectivity estimates for LIKE operations on bytea columns (Joe)

E.244. Release 7.3.5

Release date: 2003-12-03

This has a variety of fixes from 7.3.4.

E.244.1. Migration to Version 7.3.5

A dump/restore is not required for those running 7.3.*.

E.244.2. Changes

- Force zero_damaged_pages to be on during recovery from WAL

- Prevent some obscure cases of “variable not in subplan target lists”
- Force stats processes to detach from shared memory, ensuring cleaner shutdown
- Make PQescapeBytea and byteaout consistent with each other (Joe)
- Added missing SPI_finish() calls to dblink's get_tuple_of_interest() (Joe)
- Fix for possible foreign key violation when rule rewrites INSERT (Jan)
- Support qualified type names in PL/Tcl's spi_prepare command (Jan)
- Make pg_dump handle a procedural language handler located in pg_catalog
- Make pg_dump handle cases where a custom opclass is in another schema
- Make pg_dump dump binary-compatible casts correctly (Jan)
- Fix insertion of expressions containing subqueries into rule bodies
- Fix incorrect argument processing in clusterdb script (Anand Ranganathan)
- Fix problems with dropped columns in plpython triggers
- Repair problems with to_char() reading past end of its input string (Karel)
- Fix GB18030 mapping errors (Tatsuo)
- Fix several problems with SSL error handling and asynchronous SSL I/O
- Remove ability to bind a list of values to a single parameter in JDBC (prevents possible SQL-injection attacks)
- Fix some errors in HAVE_INT64_TIMESTAMP code paths
- Fix corner case for btree search in parallel with first root page split

E.245. Release 7.3.4

Release date: 2003-07-24

This has a variety of fixes from 7.3.3.

E.245.1. Migration to Version 7.3.4

A dump/restore is not required for those running 7.3.*.

E.245.2. Changes

- Repair breakage in timestamp-to-date conversion for dates before 2000
- Prevent rare possibility of server startup failure (Tom)
- Fix bugs in interval-to-time conversion (Tom)
- Add constraint names in a few places in pg_dump (Rod)
- Improve performance of functions with many parameters (Tom)
- Fix to_ascii() buffer overruns (Tom)

- Prevent restore of database comments from throwing an error (Tom)
- Work around buggy strxfrm() present in some Solaris releases (Tom)
- Properly escape jdbc setObject() strings to improve security (Barry)

E.246. Release 7.3.3

Release date: 2003-05-22

This release contains a variety of fixes for version 7.3.2.

E.246.1. Migration to Version 7.3.3

A dump/restore is not required for those running version 7.3.*.

E.246.2. Changes

- Repair sometimes-incorrect computation of StartUpID after a crash
- Avoid slowness with lots of deferred triggers in one transaction (Stephan)
- Don't lock referenced row when `UPDATE` doesn't change foreign key's value (Jan)
- Use `-fPIC` not `-fpic` on Sparc (Tom Callaway)
- Repair lack of schema-awareness in contrib/reindexdb
- Fix contrib/intarray error for zero-element result array (Teodor)
- Ensure createuser script will exit on control-C (Oliver)
- Fix errors when the type of a dropped column has itself been dropped
- `CHECKPOINT` does not cause database panic on failure in noncritical steps
- Accept 60 in seconds fields of timestamp, time, interval input values
- Issue notice, not error, if `TIMESTAMP`, `TIME`, or `INTERVAL` precision too large
- Fix `abstime-to-time` cast function (fix is not applied unless you initdb)
- Fix pg_proc entry for `timestampt_izone` (fix is not applied unless you initdb)
- Make `EXTRACT(EPOCH FROM timestamp without time zone)` treat input as local time
- `'now'::timestamptz` gave wrong answer if timezone changed earlier in transaction
- `HAVE_INT64_TIMESTAMP` code for time with timezone overwrote its input
- Accept `GLOBAL TEMP/TEMPORARY` as a synonym for `TEMPORARY`
- Avoid improper schema-privilege-check failure in foreign-key triggers
- Fix bugs in foreign-key triggers for `SET DEFAULT` action
- Fix incorrect time-qual check in row fetch for `UPDATE` and `DELETE` triggers

- Foreign-key clauses were parsed but ignored in `ALTER TABLE ADD COLUMN`
- Fix createlang script breakage for case where handler function already exists
- Fix misbehavior on zero-column tables in pg_dump, COPY, ANALYZE, other places
- Fix misbehavior of `func_error()` on type names containing '%'
- Fix misbehavior of `replace()` on strings containing '%'
- Regular-expression patterns containing certain multibyte characters failed
- Account correctly for `NULL`s in more cases in join size estimation
- Avoid conflict with system definition of `isblank()` function or macro
- Fix failure to convert large code point values in EUC_TW conversions (Tatsuo)
- Fix error recovery for `SSL_read`/`SSL_write` calls
- Don't do early constant-folding of type coercion expressions
- Validate page header fields immediately after reading in any page
- Repair incorrect check for ungrouped variables in unnamed joins
- Fix buffer overrun in `to_ascii` (Guido Notari)
- contrib/ltree fixes (Teodor)
- Fix core dump in deadlock detection on machines where char is unsigned
- Avoid running out of buffers in many-way indexscan (bug introduced in 7.3)
- Fix planner's selectivity estimation functions to handle domains properly
- Fix dbmirror memory-allocation bug (Steven Singer)
- Prevent infinite loop in `ln(numeric)` due to roundoff error
- `GROUP BY` got confused if there were multiple equal GROUP BY items
- Fix bad plan when inherited `UPDATE`/`DELETE` references another inherited table
- Prevent clustering on incomplete (partial or non-NULL-storing) indexes
- Service shutdown request at proper time if it arrives while still starting up
- Fix left-links in temporary indexes (could make backwards scans miss entries)
- Fix incorrect handling of client_encoding setting in postgresql.conf (Tatsuo)
- Fix failure to respond to `pg_ctl stop -m fast` after Async_NotifyHandler runs
- Fix SPI for case where rule contains multiple statements of the same type
- Fix problem with checking for wrong type of access privilege in rule query
- Fix problem with `EXCEPT` in `CREATE RULE`
- Prevent problem with dropping temp tables having serial columns
- Fix replace_vars_with_subplan_refs failure in complex views

- Fix regexp slowness in single-byte encodings (Tatsuo)
- Allow qualified type names in `CREATE CAST` and `DROP CAST`
- Accept `SETOF type[]`, which formerly had to be written `SETOF _type`
- Fix pg_dump core dump in some cases with procedural languages
- Force ISO datestyle in pg_dump output, for portability (Oliver)
- pg_dump failed to handle error return from `lo_read` (Oleg Drokin)
- pg_dumpall failed with groups having no members (Nick Eskelinen)
- pg_dumpall failed to recognize --globals-only switch
- pg_restore failed to restore blobs if -X disable-triggers is specified
- Repair intrafunction memory leak in plpgsql
- pltcl's `elog` command dumped core if given wrong parameters (Ian Harding)
- plpython used wrong value of `atttypmod` (Brad McLean)
- Fix improper quoting of boolean values in Python interface (D'Arcy)
- Added `addDataType()` method to PGConnection interface for JDBC
- Fixed various problems with updateable ResultSets for JDBC (Shawn Green)
- Fixed various problems with DatabaseMetaData for JDBC (Kris Jurka, Peter Royal)
- Fixed problem with parsing table ACLs in JDBC
- Better error message for character set conversion problems in JDBC

E.247. Release 7.3.2

Release date: 2003-02-04

This release contains a variety of fixes for version 7.3.1.

E.247.1. Migration to Version 7.3.2

A dump/restore is not required for those running version 7.3.*.

E.247.2. Changes

- Restore creation of OID column in CREATE TABLE AS / SELECT INTO
- Fix pg_dump core dump when dumping views having comments
- Dump DEFERRABLE/INITIALLY DEFERRED constraints properly
- Fix UPDATE when child table's column numbering differs from parent
- Increase default value of max_fsm_relations
- Fix problem when fetching backwards in a cursor for a single-row query

- Make backward fetch work properly with cursor on SELECT DISTINCT query
- Fix problems with loading pg_dump files containing contrib/lo usage
- Fix problem with all-numeric user names
- Fix possible memory leak and core dump during disconnect in libpgtcl
- Make plpython's spi_execute command handle nulls properly (Andrew Bosma)
- Adjust plpython error reporting so that its regression test passes again
- Work with bison 1.875
- Handle mixed-case names properly in plpgsql's %type (Neil)
- Fix core dump in pltcl when executing a query rewritten by a rule
- Repair array subscript overruns (per report from Yichen Xie)
- Reduce MAX_TIME_PRECISION from 13 to 10 in floating-point case
- Correctly case-fold variable names in per-database and per-user settings
- Fix coredump in plpgsql's RETURN NEXT when SELECT into record returns no rows
- Fix outdated use of pg_type.typprtlen in python client interface
- Correctly handle fractional seconds in timestamps in JDBC driver
- Improve performance of getImportedKeys() in JDBC
- Make shared-library symlinks work standardly on HPUX (Giles)
- Repair inconsistent rounding behavior for timestamp, time, interval
- SSL negotiation fixes (Nathan Mueller)
- Make libpq's ~/.pgpass feature work when connecting with PQconnectDB
- Update my2pg, ora2pg
- Translation updates
- Add casts between types lo and oid in contrib/lo
- fastpath code now checks for privilege to call function

E.248. Release 7.3.1

Release date: 2002-12-18

This release contains a variety of fixes for version 7.3.

E.248.1. Migration to Version 7.3.1

A dump/restore is not required for those running version 7.3. However, it should be noted that the main PostgreSQL interface library, libpq, has a new major version number for this release, which might require recompilation of client code in certain cases.

E.248.2. Changes

- Fix a core dump of COPY TO when client/server encodings don't match (Tom)
- Allow pg_dump to work with pre-7.2 servers (Philip)
- contrib/adddepend fixes (Tom)
- Fix problem with deletion of per-user/per-database config settings (Tom)
- contrib/vacuumlo fix (Tom)
- Allow 'password' encryption even when pg_shadow contains MD5 passwords (Bruce)
- contrib/dbmirror fix (Steven Singer)
- Optimizer fixes (Tom)
- contrib/tsearch fixes (Teodor Sigaev, Magnus)
- Allow locale names to be mixed case (Nicolai Tufar)
- Increment libpq library's major version number (Bruce)
- pg_hba.conf error reporting fixes (Bruce, Neil)
- Add SCO Openserver 5.0.4 as a supported platform (Bruce)
- Prevent EXPLAIN from crashing server (Tom)
- SSL fixes (Nathan Mueller)
- Prevent composite column creation via ALTER TABLE (Tom)

E.249. Release 7.3

Release date: 2002-11-27

E.249.1. Overview

Major changes in this release:

Schemas

Schemas allow users to create objects in separate namespaces, so two people or applications can have tables with the same name. There is also a public schema for shared tables. Table/index creation can be restricted by removing privileges on the public schema.

Drop Column

PostgreSQL now supports the `ALTER TABLE ... DROP COLUMN` functionality.

Table Functions

Functions returning multiple rows and/or multiple columns are now much easier to use than before. You can call such a “table function” in the `SELECT FROM` clause, treating its output like a table. Also, PL/pgSQL functions can now return sets.

Prepared Queries

PostgreSQL now supports prepared queries, for improved performance.

Dependency Tracking

PostgreSQL now records object dependencies, which allows improvements in many areas. `DROP` statements now take either `CASCADE` or `RESTRICT` to control whether dependent objects are also dropped.

Privileges

Functions and procedural languages now have privileges, and functions can be defined to run with the privileges of their creator.

Internationalization

Both multibyte and locale support are now always enabled.

Logging

A variety of logging options have been enhanced.

Interfaces

A large number of interfaces have been moved to http://gborg.postgresql.org where they can be developed and released independently.

Functions/Identifiers

By default, functions can now take up to 32 parameters, and identifiers can be up to 63 bytes long. Also, `OPAQUE` is now deprecated: there are specific "pseudo-datatypes" to represent each of the former meanings of `OPAQUE` in function argument and result types.

E.249.2. Migration to Version 7.3

A dump/restore using pg_dump is required for those wishing to migrate data from any previous release. If your application examines the system catalogs, additional changes will be required due to the introduction of schemas in 7.3; for more information, see: http://developer.postgresql.org/~momjian/upgrade_tips_7.3.

Observe the following incompatibilities:

- Pre-6.3 clients are no longer supported.
- `pg_hba.conf` now has a column for the user name and additional features. Existing files need to be adjusted.
- Several `postgresql.conf` logging parameters have been renamed.
- `LIMIT #,#` has been disabled; use `LIMIT # OFFSET #`.
- `INSERT` statements with column lists must specify a value for each specified column. For example, `INSERT INTO tab (col1, col2) VALUES ('val1')` is now invalid. It's still allowed to supply fewer columns than expected if the `INSERT` does not have a column list.
- `serial` columns are no longer automatically `UNIQUE`; thus, an index will not automatically be created.

- A `SET` command inside an aborted transaction is now rolled back.
- `COPY` no longer considers missing trailing columns to be null. All columns need to be specified. (However, one can achieve a similar effect by specifying a column list in the `COPY` command.)
- The data type `timestamp` is now equivalent to `timestamp without time zone`, instead of `timestamp with time zone`.
- Pre-7.3 databases loaded into 7.3 will not have the new object dependencies for `serial` columns, unique constraints, and foreign keys. See the directory `contrib/adddepend/` for a detailed description and a script that will add such dependencies.
- An empty string (`''`) is no longer allowed as the input into an integer field. Formerly, it was silently interpreted as 0.

E.249.3. Changes

E.249.3.1. Server Operation

- Add pg_locks view to show locks (Neil)
- Security fixes for password negotiation memory allocation (Neil)
- Remove support for version 0 FE/BE protocol (PostgreSQL 6.2 and earlier) (Tom)
- Reserve the last few backend slots for superusers, add parameter superuser_reserved_connections to control this (Nigel J. Andrews)

E.249.3.2. Performance

- Improve startup by calling localtime() only once (Tom)
- Cache system catalog information in flat files for faster startup (Tom)
- Improve caching of index information (Tom)
- Optimizer improvements (Tom, Fernando Nasser)
- Catalog caches now store failed lookups (Tom)
- Hash function improvements (Neil)
- Improve performance of query tokenization and network handling (Peter)
- Speed improvement for large object restore (Mario Weilguni)
- Mark expired index entries on first lookup, saving later heap fetches (Tom)
- Avoid excessive NULL bitmap padding (Manfred Koizar)
- Add BSD-licensed qsort() for Solaris, for performance (Bruce)
- Reduce per-row overhead by four bytes (Manfred Koizar)
- Fix GEQO optimizer bug (Neil Conway)
- Make WITHOUT OID actually save four bytes per row (Manfred Koizar)
- Add default_statistics_target variable to specify ANALYZE buckets (Neil)

- Use local buffer cache for temporary tables so no WAL overhead (Tom)
- Improve free space map performance on large tables (Stephen Marshall, Tom)
- Improved WAL write concurrency (Tom)

E.249.3.3. Privileges

- Add privileges on functions and procedural languages (Peter)
- Add OWNER to CREATE DATABASE so superusers can create databases on behalf of unprivileged users (Gavin Sherry, Tom)
- Add new object privilege bits EXECUTE and USAGE (Tom)
- Add SET SESSION AUTHORIZATION DEFAULT and RESET SESSION AUTHORIZATION (Tom)
- Allow functions to be executed with the privilege of the function owner (Peter)

E.249.3.4. Server Configuration

- Server log messages now tagged with LOG, not DEBUG (Bruce)
- Add user column to pg_hba.conf (Bruce)
- Have log_connections output two lines in log file (Tom)
- Remove debug_level from postgresql.conf, now server_min_messages (Bruce)
- New ALTER DATABASE/USER ... SET command for per-user/database initialization (Peter)
- New parameters server_min_messages and client_min_messages to control which messages are sent to the server logs or client applications (Bruce)
- Allow pg_hba.conf to specify lists of users/databases separated by commas, group names prepended with +, and file names prepended with @ (Bruce)
- Remove secondary password file capability and pg_password utility (Bruce)
- Add variable db_user_namespace for database-local user names (Bruce)
- SSL improvements (Bear Giles)
- Make encryption of stored passwords the default (Bruce)
- Allow pg_statistics to be reset by calling pg_stat_reset() (Christopher)
- Add log_duration parameter (Bruce)
- Rename debug_print_query to log_statement (Bruce)
- Rename show_query_stats to show_statement_stats (Bruce)
- Add param log_min_error_statement to print commands to logs on error (Gavin)

E.249.3.5. Queries

- Make cursors insensitive, meaning their contents do not change (Tom)
- Disable LIMIT #,# syntax; now only LIMIT # OFFSET # supported (Bruce)

- Increase identifier length to 63 (Neil, Bruce)
- UNION fixes for merging >= 3 columns of different lengths (Tom)
- Add DEFAULT key word to INSERT, e.g., INSERT ... (..., DEFAULT, ...) (Rod)
- Allow views to have default values using ALTER COLUMN ... SET DEFAULT (Neil)
- Fail on INSERTs with column lists that don't supply all column values, e.g., INSERT INTO tab (col1, col2) VALUES ('val1'); (Rod)
- Fix for join aliases (Tom)
- Fix for FULL OUTER JOINs (Tom)
- Improve reporting of invalid identifier and location (Tom, Gavin)
- Fix OPEN cursor(args) (Tom)
- Allow 'ctid' to be used in a view and currtid(viewname) (Hiroshi)
- Fix for CREATE TABLE AS with UNION (Tom)
- SQL99 syntax improvements (Thomas)
- Add statement_timeout variable to cancel queries (Bruce)
- Allow prepared queries with PREPARE/EXECUTE (Neil)
- Allow FOR UPDATE to appear after LIMIT/OFFSET (Bruce)
- Add variable autocommit (Tom, David Van Wie)

E.249.3.6. Object Manipulation

- Make equals signs optional in CREATE DATABASE (Gavin Sherry)
- Make ALTER TABLE OWNER change index ownership too (Neil)
- New ALTER TABLE tabname ALTER COLUMN colname SET STORAGE controls TOAST storage, compression (John Gray)
- Add schema support, CREATE/DROP SCHEMA (Tom)
- Create schema for temporary tables (Tom)
- Add variable search_path for schema search (Tom)
- Add ALTER TABLE SET/DROP NOT NULL (Christopher)
- New CREATE FUNCTION volatility levels (Tom)
- Make rule names unique only per table (Tom)
- Add 'ON tablename' clause to DROP RULE and COMMENT ON RULE (Tom)
- Add ALTER TRIGGER RENAME (Joe)
- New current_schema() and current_schemas() inquiry functions (Tom)
- Allow functions to return multiple rows (table functions) (Joe)
- Make WITH optional in CREATE DATABASE, for consistency (Bruce)

- Add object dependency tracking (Rod, Tom)
- Add RESTRICT/CASCADE to DROP commands (Rod)
- Add ALTER TABLE DROP for non-CHECK CONSTRAINT (Rod)
- Autodestroy sequence on DROP of table with SERIAL (Rod)
- Prevent column dropping if column is used by foreign key (Rod)
- Automatically drop constraints/functions when object is dropped (Rod)
- Add CREATE/DROP OPERATOR CLASS (Bill Studenmund, Tom)
- Add ALTER TABLE DROP COLUMN (Christopher, Tom, Hiroshi)
- Prevent inherited columns from being removed or renamed (Alvaro Herrera)
- Fix foreign key constraints to not error on intermediate database states (Stephan)
- Propagate column or table renaming to foreign key constraints
- Add CREATE OR REPLACE VIEW (Gavin, Neil, Tom)
- Add CREATE OR REPLACE RULE (Gavin, Neil, Tom)
- Have rules execute alphabetically, returning more predictable values (Tom)
- Triggers are now fired in alphabetical order (Tom)
- Add /contrib/adddepend to handle pre-7.3 object dependencies (Rod)
- Allow better casting when inserting/updating values (Tom)

E.249.3.7. Utility Commands

- Have COPY TO output embedded carriage returns and newlines as \r and \n (Tom)
- Allow DELIMITER in COPY FROM to be 8-bit clean (Tatsuo)
- Make pg_dump use ALTER TABLE ADD PRIMARY KEY, for performance (Neil)
- Disable brackets in multistatement rules (Bruce)
- Disable VACUUM from being called inside a function (Bruce)
- Allow dropdb and other scripts to use identifiers with spaces (Bruce)
- Restrict database comment changes to the current database
- Allow comments on operators, independent of the underlying function (Rod)
- Rollback SET commands in aborted transactions (Tom)
- EXPLAIN now outputs as a query (Tom)
- Display condition expressions and sort keys in EXPLAIN (Tom)
- Add 'SET LOCAL var = value' to set configuration variables for a single transaction (Tom)
- Allow ANALYZE to run in a transaction (Bruce)

- Improve COPY syntax using new WITH clauses, keep backward compatibility (Bruce)
- Fix pg_dump to consistently output tags in non-ASCII dumps (Bruce)
- Make foreign key constraints clearer in dump file (Rod)
- Add COMMENT ON CONSTRAINT (Rod)
- Allow COPY TO/FROM to specify column names (Brent Verner)
- Dump UNIQUE and PRIMARY KEY constraints as ALTER TABLE (Rod)
- Have SHOW output a query result (Joe)
- Generate failure on short COPY lines rather than pad NULLs (Neil)
- Fix CLUSTER to preserve all table attributes (Alvaro Herrera)
- New pg_settings table to view/modify GUC settings (Joe)
- Add smart quoting, portability improvements to pg_dump output (Peter)
- Dump serial columns out as SERIAL (Tom)
- Enable large file support, >2G for pg_dump (Peter, Philip Warner, Bruce)
- Disallow TRUNCATE on tables that are involved in referential constraints (Rod)
- Have TRUNCATE also auto-truncate the toast table of the relation (Tom)
- Add clusterdb utility that will auto-cluster an entire database based on previous CLUSTER operations (Alvaro Herrera)
- Overhaul pg_dumpall (Peter)
- Allow REINDEX of TOAST tables (Tom)
- Implemented START TRANSACTION, per SQL99 (Neil)
- Fix rare index corruption when a page split affects bulk delete (Tom)
- Fix ALTER TABLE ... ADD COLUMN for inheritance (Alvaro Herrera)

E.249.3.8. Data Types and Functions

- Fix factorial(0) to return 1 (Bruce)
- Date/time/timezone improvements (Thomas)
- Fix for array slice extraction (Tom)
- Fix extract/date_part to report proper microseconds for timestamp (Tatsuo)
- Allow text_substr() and bytea_substr() to read TOAST values more efficiently (John Gray)
- Add domain support (Rod)
- Make WITHOUT TIME ZONE the default for TIMESTAMP and TIME data types (Thomas)
- Allow alternate storage scheme of 64-bit integers for date/time types using --enable-integer-datetimes in configure (Thomas)

- Make timezone(timestamptz) return timestamp rather than a string (Thomas)
- Allow fractional seconds in date/time types for dates prior to 1BC (Thomas)
- Limit timestamp data types to 6 decimal places of precision (Thomas)
- Change timezone conversion functions from timetz() to timezone() (Thomas)
- Add configuration variables datestyle and timezone (Tom)
- Add OVERLAY(), which allows substitution of a substring in a string (Thomas)
- Add SIMILAR TO (Thomas, Tom)
- Add regular expression SUBSTRING(string FROM pat FOR escape) (Thomas)
- Add LOCALTIME and LOCALTIMESTAMP functions (Thomas)
- Add named composite types using CREATE TYPE typename AS (column) (Joe)
- Allow composite type definition in the table alias clause (Joe)
- Add new API to simplify creation of C language table functions (Joe)
- Remove ODBC-compatible empty parentheses from calls to SQL99 functions for which these parentheses do not match the standard (Thomas)
- Allow macaddr data type to accept 12 hex digits with no separators (Mike Wyer)
- Add CREATE/DROP CAST (Peter)
- Add IS DISTINCT FROM operator (Thomas)
- Add SQL99 TREAT() function, synonym for CAST() (Thomas)
- Add pg_backend_pid() to output backend pid (Bruce)
- Add IS OF / IS NOT OF type predicate (Thomas)
- Allow bit string constants without fully-specified length (Thomas)
- Allow conversion between 8-byte integers and bit strings (Thomas)
- Implement hex literal conversion to bit string literal (Thomas)
- Allow table functions to appear in the FROM clause (Joe)
- Increase maximum number of function parameters to 32 (Bruce)
- No longer automatically create index for SERIAL column (Tom)
- Add current_database() (Rod)
- Fix cash_words() to not overflow buffer (Tom)
- Add functions replace(), split_part(), to_hex() (Joe)
- Fix LIKE for bytea as a right-hand argument (Joe)
- Prevent crashes caused by SELECT cash_out(2) (Tom)
- Fix to_char(1,'FM999.99') to return a period (Karel)

- Fix trigger/type/language functions returning OPAQUE to return proper type (Tom)

E.249.3.9. Internationalization

- Add additional encodings: Korean (JOHAB), Thai (WIN874), Vietnamese (TCVN), Arabic (WIN1256), Simplified Chinese (GBK), Korean (UHC) (Eiji Tokuya)
- Enable locale support by default (Peter)
- Add locale variables (Peter)
- Escape byes >= 0x7f for multibyte in PQescapeBytea/PQunescapeBytea (Tatsuo)
- Add locale awareness to regular expression character classes
- Enable multibyte support by default (Tatsuo)
- Add GB18030 multibyte support (Bill Huang)
- Add CREATE/DROP CONVERSION, allowing loadable encodings (Tatsuo, Kaori)
- Add pg_conversion table (Tatsuo)
- Add SQL99 CONVERT() function (Tatsuo)
- pg_dumpall, pg_controldata, and pg_resetxlog now national-language aware (Peter)
- New and updated translations

E.249.3.10. Server-side Languages

- Allow recursive SQL function (Peter)
- Change PL/Tcl build to use configured compiler and Makefile.shlib (Peter)
- Overhaul the PL/pgSQL FOUND variable to be more Oracle-compatible (Neil, Tom)
- Allow PL/pgSQL to handle quoted identifiers (Tom)
- Allow set-returning PL/pgSQL functions (Neil)
- Make PL/pgSQL schema-aware (Joe)
- Remove some memory leaks (Nigel J. Andrews, Tom)

E.249.3.11. psql

- Don't lowercase psql \connect database name for 7.2.0 compatibility (Tom)
- Add psql \timing to time user queries (Greg Sabino Mullane)
- Have psql \d show index information (Greg Sabino Mullane)
- New psql \dD shows domains (Jonathan Eisler)
- Allow psql to show rules on views (Paul ?)
- Fix for psql variable substitution (Tom)
- Allow psql \d to show temporary table structure (Tom)

- Allow psql \d to show foreign keys (Rod)
- Fix \? to honor \pset pager (Bruce)
- Have psql reports its version number on startup (Tom)
- Allow \copy to specify column names (Tom)

E.249.3.12. libpq

- Add ~/.pgpass to store host/user password combinations (Alvaro Herrera)
- Add PQunescapeBytea() function to libpq (Patrick Welche)
- Fix for sending large queries over non-blocking connections (Bernhard Herzog)
- Fix for libpq using timers on Win9X (David Ford)
- Allow libpq notify to handle servers with different-length identifiers (Tom)
- Add libpq PQescapeString() and PQescapeBytea() to Windows (Bruce)
- Fix for SSL with non-blocking connections (Jack Bates)
- Add libpq connection timeout parameter (Denis A Ustimenko)

E.249.3.13. JDBC

- Allow JDBC to compile with JDK 1.4 (Dave)
- Add JDBC 3 support (Barry)
- Allows JDBC to set loglevel by adding ?loglevel=X to the connection URL (Barry)
- Add Driver.info() message that prints out the version number (Barry)
- Add updateable result sets (Raghu Nidagal, Dave)
- Add support for callable statements (Paul Bethe)
- Add query cancel capability
- Add refresh row (Dave)
- Fix MD5 encryption handling for multibyte servers (Jun Kawai)
- Add support for prepared statements (Barry)

E.249.3.14. Miscellaneous Interfaces

- Fixed ECPG bug concerning octal numbers in single quotes (Michael)
- Move src/interfaces/libpgeasy to http://gborg.postgresql.org (Marc, Bruce)
- Improve Python interface (Elliot Lee, Andrew Johnson, Greg Copeland)
- Add libpgtcl connection close event (Gerhard Hintermayer)
- Move src/interfaces/libpq++ to http://gborg.postgresql.org (Marc, Bruce)
- Move src/interfaces/odbc to http://gborg.postgresql.org (Marc)

- Move src/interfaces/libpgeasy to http://gborg.postgresql.org (Marc, Bruce)
- Move src/interfaces/perl5 to http://gborg.postgresql.org (Marc, Bruce)
- Remove src/bin/pgaccess from main tree, now at http://www.pgaccess.org (Bruce)
- Add pg_on_connection_loss command to libpgtcl (Gerhard Hintermayer, Tom)

E.249.3.15. Source Code

- Fix for parallel make (Peter)
- AIX fixes for linking Tcl (Andreas Zeugswetter)
- Allow PL/Perl to build under Cygwin (Jason Tishler)
- Improve MIPS compiles (Peter, Oliver Elphick)
- Require Autoconf version 2.53 (Peter)
- Require readline and zlib by default in configure (Peter)
- Allow Solaris to use Intimate Shared Memory (ISM), for performance (Scott Brunza, P.J. Josh Rovero)
- Always enable syslog in compile, remove --enable-syslog option (Tatsuo)
- Always enable multibyte in compile, remove --enable-multibyte option (Tatsuo)
- Always enable locale in compile, remove --enable-locale option (Peter)
- Fix for Win9x DLL creation (Magnus Naeslund)
- Fix for link() usage by WAL code on Windows, BeOS (Jason Tishler)
- Add sys/types.h to c.h, remove from main files (Peter, Bruce)
- Fix AIX hang on SMP machines (Tomoyuki Niijima)
- AIX SMP hang fix (Tomoyuki Niijima)
- Fix pre-1970 date handling on newer glibc libraries (Tom)
- Fix PowerPC SMP locking (Tom)
- Prevent gcc -ffast-math from being used (Peter, Tom)
- Bison >= 1.50 now required for developer builds
- Kerberos 5 support now builds with Heimdal (Peter)
- Add appendix in the User's Guide which lists SQL features (Thomas)
- Improve loadable module linking to use RTLD_NOW (Tom)
- New error levels WARNING, INFO, LOG, DEBUG[1-5] (Bruce)
- New src/port directory holds replaced libc functions (Peter, Bruce)
- New pg_namespace system catalog for schemas (Tom)
- Add pg_class.relnamespace for schemas (Tom)

- Add pg_type.typnamespace for schemas (Tom)
- Add pg_proc.pronamespace for schemas (Tom)
- Restructure aggregates to have pg_proc entries (Tom)
- System relations now have their own namespace, pg_* test not required (Fernando Nasser)
- Rename TOAST index names to be *_index rather than *_idx (Neil)
- Add namespaces for operators, opclasses (Tom)
- Add additional checks to server control file (Thomas)
- New Polish FAQ (Marcin Mazurek)
- Add Posix semaphore support (Tom)
- Document need for reindex (Bruce)
- Rename some internal identifiers to simplify Windows compile (Jan, Katherine Ward)
- Add documentation on computing disk space (Bruce)
- Remove KSQO from GUC (Bruce)
- Fix memory leak in rtree (Kenneth Been)
- Modify a few error messages for consistency (Bruce)
- Remove unused system table columns (Peter)
- Make system columns NOT NULL where appropriate (Tom)
- Clean up use of sprintf in favor of snprintf() (Neil, Jukka Holappa)
- Remove OPAQUE and create specific subtypes (Tom)
- Cleanups in array internal handling (Joe, Tom)
- Disallow pg_atoi('') (Bruce)
- Remove parameter wal_files because WAL files are now recycled (Bruce)
- Add version numbers to heap pages (Tom)

E.249.3.16. Contrib

- Allow inet arrays in /contrib/array (Neil)
- GiST fixes (Teodor Sigaev, Neil)
- Upgrade /contrib/mysql
- Add /contrib/dbsize which shows table sizes without vacuum (Peter)
- Add /contrib/intagg, integer aggregator routines (mlw)
- Improve /contrib/oid2name (Neil, Bruce)
- Improve /contrib/tsearch (Oleg, Teodor Sigaev)

- Cleanups of /contrib/rserver (Alexey V. Borzov)
- Update /contrib/oracle conversion utility (Gilles Darold)
- Update /contrib/dblink (Joe)
- Improve options supported by /contrib/vacuumlo (Mario Weilguni)
- Improvements to /contrib/intarray (Oleg, Teodor Sigaev, Andrey Oktyabrski)
- Add /contrib/reindexdb utility (Shaun Thomas)
- Add indexing to /contrib/isbn_issn (Dan Weston)
- Add /contrib/dbmirror (Steven Singer)
- Improve /contrib/pgbench (Neil)
- Add /contrib/tablefunc table function examples (Joe)
- Add /contrib/ltree data type for tree structures (Teodor Sigaev, Oleg Bartunov)
- Move /contrib/pg_controldata, pg_resetxlog into main tree (Bruce)
- Fixes to /contrib/cube (Bruno Wolff)
- Improve /contrib/fulltextindex (Christopher)

E.250. Release 7.2.8

Release date: 2005-05-09

This release contains a variety of fixes from 7.2.7, including one security-related issue.

E.250.1. Migration to Version 7.2.8

A dump/restore is not required for those running 7.2.X.

E.250.2. Changes

- Repair ancient race condition that allowed a transaction to be seen as committed for some purposes (eg SELECT FOR UPDATE) slightly sooner than for other purposes

  This is an extremely serious bug since it could lead to apparent data inconsistencies being briefly visible to applications.

- Repair race condition between relation extension and VACUUM

  This could theoretically have caused loss of a page's worth of freshly-inserted data, although the scenario seems of very low probability. There are no known cases of it having caused more than an Assert failure.

- Fix `EXTRACT(EPOCH)` for `TIME WITH TIME ZONE` values
- Additional buffer overrun checks in plpgsql (Neil)
- Fix pg_dump to dump index names and trigger names containing `%` correctly (Neil)
- Prevent `to_char(interval)` from dumping core for month-related formats

- Fix `contrib/pgcrypto` for newer OpenSSL builds (Marko Kreen)

E.251. Release 7.2.7

Release date: 2005-01-31

This release contains a variety of fixes from 7.2.6, including several security-related issues.

E.251.1. Migration to Version 7.2.7

A dump/restore is not required for those running 7.2.X.

E.251.2. Changes

- Disallow `LOAD` to non-superusers

  On platforms that will automatically execute initialization functions of a shared library (this includes at least Windows and ELF-based Unixen), `LOAD` can be used to make the server execute arbitrary code. Thanks to NGS Software for reporting this.

- Add needed STRICT marking to some contrib functions (Kris Jurka)

- Avoid buffer overrun when plpgsql cursor declaration has too many parameters (Neil)

- Fix planning error for FULL and RIGHT outer joins

  The result of the join was mistakenly supposed to be sorted the same as the left input. This could not only deliver mis-sorted output to the user, but in case of nested merge joins could give outright wrong answers.

- Fix display of negative intervals in SQL and GERMAN datestyles

E.252. Release 7.2.6

Release date: 2004-10-22

This release contains a variety of fixes from 7.2.5.

E.252.1. Migration to Version 7.2.6

A dump/restore is not required for those running 7.2.X.

E.252.2. Changes

- Repair possible failure to update hint bits on disk

  Under rare circumstances this oversight could lead to “could not access transaction status” failures, which qualifies it as a potential-data-loss bug.

- Ensure that hashed outer join does not miss tuples

  Very large left joins using a hash join plan could fail to output unmatched left-side rows given just the right data distribution.

- Disallow running pg_ctl as root

This is to guard against any possible security issues.

- Avoid using temp files in /tmp in make_oidjoins_check

  This has been reported as a security issue, though it's hardly worthy of concern since there is no reason for non-developers to use this script anyway.

- Update to newer versions of Bison

E.253. Release 7.2.5

Release date: 2004-08-16

This release contains a variety of fixes from 7.2.4.

E.253.1. Migration to Version 7.2.5

A dump/restore is not required for those running 7.2.X.

E.253.2. Changes

- Prevent possible loss of committed transactions during crash

  Due to insufficient interlocking between transaction commit and checkpointing, it was possible for transactions committed just before the most recent checkpoint to be lost, in whole or in part, following a database crash and restart. This is a serious bug that has existed since PostgreSQL 7.1.

- Fix corner case for btree search in parallel with first root page split
- Fix buffer overrun in `to_ascii` (Guido Notari)
- Fix core dump in deadlock detection on machines where char is unsigned
- Fix failure to respond to `pg_ctl stop -m fast` after Async_NotifyHandler runs
- Repair memory leaks in pg_dump
- Avoid conflict with system definition of `isblank()` function or macro

E.254. Release 7.2.4

Release date: 2003-01-30

This release contains a variety of fixes for version 7.2.3, including fixes to prevent possible data loss.

E.254.1. Migration to Version 7.2.4

A dump/restore is not required for those running version 7.2.*.

E.254.2. Changes

- Fix some additional cases of VACUUM "No one parent tuple was found" error
- Prevent VACUUM from being called inside a function (Bruce)

- Ensure pg_clog updates are sync'd to disk before marking checkpoint complete
- Avoid integer overflow during large hash joins
- Make GROUP commands work when pg_group.grolist is large enough to be toasted
- Fix errors in datetime tables; some timezone names weren't being recognized
- Fix integer overflows in circle_poly(), path_encode(), path_add() (Neil)
- Repair long-standing logic errors in lseg_eq(), lseg_ne(), lseg_center()

E.255. Release 7.2.3

Release date: 2002-10-01

This release contains a variety of fixes for version 7.2.2, including fixes to prevent possible data loss.

E.255.1. Migration to Version 7.2.3

A dump/restore is not required for those running version 7.2.*.

E.255.2. Changes

- Prevent possible compressed transaction log loss (Tom)
- Prevent non-superuser from increasing most recent vacuum info (Tom)
- Handle pre-1970 date values in newer versions of glibc (Tom)
- Fix possible hang during server shutdown
- Prevent spinlock hangs on SMP PPC machines (Tomoyuki Niijima)
- Fix pg_dump to properly dump FULL JOIN USING (Tom)

E.256. Release 7.2.2

Release date: 2002-08-23

This release contains a variety of fixes for version 7.2.1.

E.256.1. Migration to Version 7.2.2

A dump/restore is not required for those running version 7.2.*.

E.256.2. Changes

- Allow EXECUTE of "CREATE TABLE AS ... SELECT" in PL/pgSQL (Tom)
- Fix for compressed transaction log id wraparound (Tom)
- Fix PQescapeBytea/PQunescapeBytea so that they handle bytes > 0x7f (Tatsuo)
- Fix for psql and pg_dump crashing when invoked with non-existent long options (Tatsuo)
- Fix crash when invoking geometric operators (Tom)

- Allow OPEN cursor(args) (Tom)
- Fix for rtree_gist index build (Teodor)
- Fix for dumping user-defined aggregates (Tom)
- contrib/intarray fixes (Oleg)
- Fix for complex UNION/EXCEPT/INTERSECT queries using parens (Tom)
- Fix to pg_convert (Tatsuo)
- Fix for crash with long DATA strings (Thomas, Neil)
- Fix for repeat(), lpad(), rpad() and long strings (Neil)

E.257. Release 7.2.1

Release date: 2002-03-21

This release contains a variety of fixes for version 7.2.

E.257.1. Migration to Version 7.2.1

A dump/restore is not required for those running version 7.2.

E.257.2. Changes

- Ensure that sequence counters do not go backwards after a crash (Tom)
- Fix pgaccess kanji-conversion key binding (Tatsuo)
- Optimizer improvements (Tom)
- Cash I/O improvements (Tom)
- New Russian FAQ
- Compile fix for missing AuthBlockSig (Heiko)
- Additional time zones and time zone fixes (Thomas)
- Allow psql \connect to handle mixed case database and user names (Tom)
- Return proper OID on command completion even with ON INSERT rules (Tom)
- Allow COPY FROM to use 8-bit DELIMITERS (Tatsuo)
- Fix bug in extract/date_part for milliseconds/microseconds (Tatsuo)
- Improve handling of multiple UNIONs with different lengths (Tom)
- contrib/btree_gist improvements (Teodor Sigaev)
- contrib/tsearch dictionary improvements, see README.tsearch for an additional installation step (Thomas T. Thai, Teodor Sigaev)
- Fix for array subscripts handling (Tom)
- Allow EXECUTE of "CREATE TABLE AS ... SELECT" in PL/pgSQL (Tom)

E.258. Release 7.2

Release date: 2002-02-04

E.258.1. Overview

This release improves PostgreSQL for use in high-volume applications.

Major changes in this release:

VACUUM

Vacuuming no longer locks tables, thus allowing normal user access during the vacuum. A new `VACUUM FULL` command does old-style vacuum by locking the table and shrinking the on-disk copy of the table.

Transactions

There is no longer a problem with installations that exceed four billion transactions.

OIDs

OIDs are now optional. Users can now create tables without OIDs for cases where OID usage is excessive.

Optimizer

The system now computes histogram column statistics during `ANALYZE`, allowing much better optimizer choices.

Security

A new MD5 encryption option allows more secure storage and transfer of passwords. A new Unix-domain socket authentication option is available on Linux and BSD systems.

Statistics

Administrators can use the new table access statistics module to get fine-grained information about table and index usage.

Internationalization

Program and library messages can now be displayed in several languages.

E.258.2. Migration to Version 7.2

A dump/restore using `pg_dump` is required for those wishing to migrate data from any previous release.

Observe the following incompatibilities:

- The semantics of the `VACUUM` command have changed in this release. You might wish to update your maintenance procedures accordingly.
- In this release, comparisons using `= NULL` will always return false (or NULL, more precisely). Previous releases automatically transformed this syntax to `IS NULL`. The old behavior can be re-enabled using a `postgresql.conf` parameter.

- The `pg_hba.conf` and `pg_ident.conf` configuration is now only reloaded after receiving a SIGHUP signal, not with each connection.
- The function `octet_length()` now returns the uncompressed data length.
- The date/time value `'current'` is no longer available. You will need to rewrite your applications.
- The `timestamp()`, `time()`, and `interval()` functions are no longer available. Instead of `timestamp()`, use `timestamp 'string'` or `CAST`.

The `SELECT ... LIMIT #,#` syntax will be removed in the next release. You should change your queries to use separate LIMIT and OFFSET clauses, e.g. `LIMIT 10 OFFSET 20`.

E.258.3. Changes

E.258.3.1. Server Operation

- Create temporary files in a separate directory (Bruce)
- Delete orphaned temporary files on postmaster startup (Bruce)
- Added unique indexes to some system tables (Tom)
- System table operator reorganization (Oleg Bartunov, Teodor Sigaev, Tom)
- Renamed pg_log to pg_clog (Tom)
- Enable SIGTERM, SIGQUIT to kill backends (Jan)
- Removed compile-time limit on number of backends (Tom)
- Better cleanup for semaphore resource failure (Tatsuo, Tom)
- Allow safe transaction ID wraparound (Tom)
- Removed OIDs from some system tables (Tom)
- Removed "triggered data change violation" error check (Tom)
- SPI portal creation of prepared/saved plans (Jan)
- Allow SPI column functions to work for system columns (Tom)
- Long value compression improvement (Tom)
- Statistics collector for table, index access (Jan)
- Truncate extra-long sequence names to a reasonable value (Tom)
- Measure transaction times in milliseconds (Thomas)
- Fix TID sequential scans (Hiroshi)
- Superuser ID now fixed at 1 (Peter E)
- New pg_ctl "reload" option (Tom)

E.258.3.2. Performance

- Optimizer improvements (Tom)

- New histogram column statistics for optimizer (Tom)
- Reuse write-ahead log files rather than discarding them (Tom)
- Cache improvements (Tom)
- IS NULL, IS NOT NULL optimizer improvement (Tom)
- Improve lock manager to reduce lock contention (Tom)
- Keep relcache entries for index access support functions (Tom)
- Allow better selectivity with NaN and infinities in NUMERIC (Tom)
- R-tree performance improvements (Kenneth Been)
- B-tree splits more efficient (Tom)

E.258.3.3. Privileges

- Change UPDATE, DELETE privileges to be distinct (Peter E)
- New REFERENCES, TRIGGER privileges (Peter E)
- Allow GRANT/REVOKE to/from more than one user at a time (Peter E)
- New has_table_privilege() function (Joe Conway)
- Allow non-superuser to vacuum database (Tom)
- New SET SESSION AUTHORIZATION command (Peter E)
- Fix bug in privilege modifications on newly created tables (Tom)
- Disallow access to pg_statistic for non-superuser, add user-accessible views (Tom)

E.258.3.4. Client Authentication

- Fork postmaster before doing authentication to prevent hangs (Peter E)
- Add ident authentication over Unix domain sockets on Linux, *BSD (Helge Bahmann, Oliver Elphick, Teodor Sigaev, Bruce)
- Add a password authentication method that uses MD5 encryption (Bruce)
- Allow encryption of stored passwords using MD5 (Bruce)
- PAM authentication (Dominic J. Eidson)
- Load pg_hba.conf and pg_ident.conf only on startup and SIGHUP (Bruce)

E.258.3.5. Server Configuration

- Interpretation of some time zone abbreviations as Australian rather than North American now settable at run time (Bruce)
- New parameter to set default transaction isolation level (Peter E)
- New parameter to enable conversion of "expr = NULL" into "expr IS NULL", off by default (Peter E)

- New parameter to control memory usage by VACUUM (Tom)
- New parameter to set client authentication timeout (Tom)
- New parameter to set maximum number of open files (Tom)

E.258.3.6. Queries

- Statements added by INSERT rules now execute after the INSERT (Jan)
- Prevent unadorned relation names in target list (Bruce)
- NULLs now sort after all normal values in ORDER BY (Tom)
- New IS UNKNOWN, IS NOT UNKNOWN Boolean tests (Tom)
- New SHARE UPDATE EXCLUSIVE lock mode (Tom)
- New EXPLAIN ANALYZE command that shows run times and row counts (Martijn van Oosterhout)
- Fix problem with LIMIT and subqueries (Tom)
- Fix for LIMIT, DISTINCT ON pushed into subqueries (Tom)
- Fix nested EXCEPT/INTERSECT (Tom)

E.258.3.7. Schema Manipulation

- Fix SERIAL in temporary tables (Bruce)
- Allow temporary sequences (Bruce)
- Sequences now use int8 internally (Tom)
- New SERIAL8 creates int8 columns with sequences, default still SERIAL4 (Tom)
- Make OIDs optional using WITHOUT OIDS (Tom)
- Add %TYPE syntax to CREATE TYPE (Ian Lance Taylor)
- Add ALTER TABLE / DROP CONSTRAINT for CHECK constraints (Christopher Kings-Lynne)
- New CREATE OR REPLACE FUNCTION to alter existing function (preserving the function OID) (Gavin Sherry)
- Add ALTER TABLE / ADD [UNIQUE | PRIMARY] (Christopher Kings-Lynne)
- Allow column renaming in views
- Make ALTER TABLE / RENAME COLUMN update column names of indexes (Brent Verner)
- Fix for ALTER TABLE / ADD CONSTRAINT ... CHECK with inherited tables (Stephan Szabo)
- ALTER TABLE RENAME update foreign-key trigger arguments correctly (Brent Verner)
- DROP AGGREGATE and COMMENT ON AGGREGATE now accept an aggtype (Tom)
- Add automatic return type data casting for SQL functions (Tom)

- Allow GiST indexes to handle NULLs and multikey indexes (Oleg Bartunov, Teodor Sigaev, Tom)
- Enable partial indexes (Martijn van Oosterhout)

E.258.3.8. Utility Commands

- Add RESET ALL, SHOW ALL (Marko Kreen)
- CREATE/ALTER USER/GROUP now allow options in any order (Vince)
- Add LOCK A, B, C functionality (Neil Padgett)
- New ENCRYPTED/UNENCRYPTED option to CREATE/ALTER USER (Bruce)
- New light-weight VACUUM does not lock table; old semantics are available as VACUUM FULL (Tom)
- Disable COPY TO/FROM on views (Bruce)
- COPY DELIMITERS string must be exactly one character (Tom)
- VACUUM warning about index tuples fewer than heap now only appears when appropriate (Martijn van Oosterhout)
- Fix privilege checks for CREATE INDEX (Tom)
- Disallow inappropriate use of CREATE/DROP INDEX/TRIGGER/VIEW (Tom)

E.258.3.9. Data Types and Functions

- SUM(), AVG(), COUNT() now uses int8 internally for speed (Tom)
- Add convert(), convert2() (Tatsuo)
- New function bit_length() (Peter E)
- Make the "n" in CHAR(n)/VARCHAR(n) represents letters, not bytes (Tatsuo)
- CHAR(), VARCHAR() now reject strings that are too long (Peter E)
- BIT VARYING now rejects bit strings that are too long (Peter E)
- BIT now rejects bit strings that do not match declared size (Peter E)
- INET, CIDR text conversion functions (Alex Pilosov)
- INET, CIDR operators << and <<= indexable (Alex Pilosov)
- Bytea \### now requires valid three digit octal number
- Bytea comparison improvements, now supports =, <>, >, >=, <, and <=
- Bytea now supports B-tree indexes
- Bytea now supports LIKE, LIKE...ESCAPE, NOT LIKE, NOT LIKE...ESCAPE
- Bytea now supports concatenation
- New bytea functions: position, substring, trim, btrim, and length
- New encode() function mode, "escaped", converts minimally escaped bytea to/from text

- Add pg_database_encoding_max_length() (Tatsuo)
- Add pg_client_encoding() function (Tatsuo)
- now() returns time with millisecond precision (Thomas)
- New TIMESTAMP WITHOUT TIMEZONE data type (Thomas)
- Add ISO date/time specification with "T", yyyy-mm-ddThh:mm:ss (Thomas)
- New xid/int comparison functions (Hiroshi)
- Add precision to TIME, TIMESTAMP, and INTERVAL data types (Thomas)
- Modify type coercion logic to attempt binary-compatible functions first (Tom)
- New encode() function installed by default (Marko Kreen)
- Improved to_*() conversion functions (Karel Zak)
- Optimize LIKE/ILIKE when using single-byte encodings (Tatsuo)
- New functions in contrib/pgcrypto: crypt(), hmac(), encrypt(), gen_salt() (Marko Kreen)
- Correct description of translate() function (Bruce)
- Add INTERVAL argument for SET TIME ZONE (Thomas)
- Add INTERVAL YEAR TO MONTH (etc.) syntax (Thomas)
- Optimize length functions when using single-byte encodings (Tatsuo)
- Fix path_inter, path_distance, path_length, dist_ppath to handle closed paths (Curtis Barrett, Tom)
- octet_length(text) now returns non-compressed length (Tatsuo, Bruce)
- Handle "July" full name in date/time literals (Greg Sabino Mullane)
- Some datatype() function calls now evaluated differently
- Add support for Julian and ISO time specifications (Thomas)

E.258.3.10. Internationalization

- National language support in psql, pg_dump, libpq, and server (Peter E)
- Message translations in Chinese (simplified, traditional), Czech, French, German, Hungarian, Russian, Swedish (Peter E, Serguei A. Mokhov, Karel Zak, Weiping He, Zhenbang Wei, Kovacs Zoltan)
- Make trim, ltrim, rtrim, btrim, lpad, rpad, translate multibyte aware (Tatsuo)
- Add LATIN5,6,7,8,9,10 support (Tatsuo)
- Add ISO 8859-5,6,7,8 support (Tatsuo)
- Correct LATIN5 to mean ISO-8859-9, not ISO-8859-5 (Tatsuo)
- Make mic2ascii() non-ASCII aware (Tatsuo)

- Reject invalid multibyte character sequences (Tatsuo)

E.258.3.11. PL/pgSQL

- Now uses portals for SELECT loops, allowing huge result sets (Jan)
- CURSOR and REFCURSOR support (Jan)
- Can now return open cursors (Jan)
- Add ELSEIF (Klaus Reger)
- Improve PL/pgSQL error reporting, including location of error (Tom)
- Allow IS or FOR key words in cursor declaration, for compatibility (Bruce)
- Fix for SELECT ... FOR UPDATE (Tom)
- Fix for PERFORM returning multiple rows (Tom)
- Make PL/pgSQL use the server's type coercion code (Tom)
- Memory leak fix (Jan, Tom)
- Make trailing semicolon optional (Tom)

E.258.3.12. PL/Perl

- New untrusted PL/Perl (Alex Pilosov)
- PL/Perl is now built on some platforms even if libperl is not shared (Peter E)

E.258.3.13. PL/Tcl

- Now reports errorInfo (Vsevolod Lobko)
- Add spi_lastoid function (bob@redivi.com)

E.258.3.14. PL/Python

- ...is new (Andrew Bosma)

E.258.3.15. psql

- \d displays indexes in unique, primary groupings (Christopher Kings-Lynne)
- Allow trailing semicolons in backslash commands (Greg Sabino Mullane)
- Read password from /dev/tty if possible
- Force new password prompt when changing user and database (Tatsuo, Tom)
- Format the correct number of columns for Unicode (Patrice)

E.258.3.16. libpq

- New function PQescapeString() to escape quotes in command strings (Florian Weimer)
- New function PQescapeBytea() escapes binary strings for use as SQL string literals

E.258.3.17. JDBC

- Return OID of INSERT (Ken K)
- Handle more data types (Ken K)
- Handle single quotes and newlines in strings (Ken K)
- Handle NULL variables (Ken K)
- Fix for time zone handling (Barry Lind)
- Improved Druid support
- Allow eight-bit characters with non-multibyte server (Barry Lind)
- Support BIT, BINARY types (Ned Wolpert)
- Reduce memory usage (Michael Stephens, Dave Cramer)
- Update DatabaseMetaData (Peter E)
- Add DatabaseMetaData.getCatalogs() (Peter E)
- Encoding fixes (Anders Bengtsson)
- Get/setCatalog methods (Jason Davies)
- DatabaseMetaData.getColumns() now returns column defaults (Jason Davies)
- DatabaseMetaData.getColumns() performance improvement (Jeroen van Vianen)
- Some JDBC1 and JDBC2 merging (Anders Bengtsson)
- Transaction performance improvements (Barry Lind)
- Array fixes (Greg Zoller)
- Serialize addition
- Fix batch processing (Rene Pijlman)
- ExecSQL method reorganization (Anders Bengtsson)
- GetColumn() fixes (Jeroen van Vianen)
- Fix isWriteable() function (Rene Pijlman)
- Improved passage of JDBC2 conformance tests (Rene Pijlman)
- Add bytea type capability (Barry Lind)
- Add isNullable() (Rene Pijlman)
- JDBC date/time test suite fixes (Liam Stewart)
- Fix for SELECT 'id' AS xxx FROM table (Dave Cramer)
- Fix DatabaseMetaData to show precision properly (Mark Lillywhite)
- New getImported/getExported keys (Jason Davies)
- MD5 password encryption support (Jeremy Wohl)

- Fix to actually use type cache (Ned Wolpert)

E.258.3.18. ODBC

- Remove query size limit (Hiroshi)
- Remove text field size limit (Hiroshi)
- Fix for SQLPrimaryKeys in multibyte mode (Hiroshi)
- Allow ODBC procedure calls (Hiroshi)
- Improve boolean handing (Aidan Mountford)
- Most configuration options now settable via DSN (Hiroshi)
- Multibyte, performance fixes (Hiroshi)
- Allow driver to be used with iODBC or unixODBC (Peter E)
- MD5 password encryption support (Bruce)
- Add more compatibility functions to odbc.sql (Peter E)

E.258.3.19. ECPG

- EXECUTE ... INTO implemented (Christof Petig)
- Multiple row descriptor support (e.g. CARDINALITY) (Christof Petig)
- Fix for GRANT parameters (Lee Kindness)
- Fix INITIALLY DEFERRED bug
- Various bug fixes (Michael, Christof Petig)
- Auto allocation for indicator variable arrays (int *ind_p=NULL)
- Auto allocation for string arrays (char **foo_pp=NULL)
- ECPGfree_auto_mem fixed
- All function names with external linkage are now prefixed by ECPG
- Fixes for arrays of structures (Michael)

E.258.3.20. Misc. Interfaces

- Python fix fetchone() (Gerhard Haring)
- Use UTF, Unicode in Tcl where appropriate (Vsevolod Lobko, Reinhard Max)
- Add Tcl COPY TO/FROM (ljb)
- Prevent output of default index op class in pg_dump (Tom)
- Fix libpgeasy memory leak (Bruce)

E.258.3.21. Build and Install

- Configure, dynamic loader, and shared library fixes (Peter E)
- Fixes in QNX 4 port (Bernd Tegge)

- Fixes in Cygwin and Windows ports (Jason Tishler, Gerhard Haring, Dmitry Yurtaev, Darko Prenosil, Mikhail Terekhov)
- Fix for Windows socket communication failures (Magnus, Mikhail Terekhov)
- Hurd compile fix (Oliver Elphick)
- BeOS fixes (Cyril Velter)
- Remove configure --enable-unicode-conversion, now enabled by multibyte (Tatsuo)
- AIX fixes (Tatsuo, Andreas)
- Fix parallel make (Peter E)
- Install SQL language manual pages into OS-specific directories (Peter E)
- Rename config.h to pg_config.h (Peter E)
- Reorganize installation layout of header files (Peter E)

E.258.3.22. Source Code

- Remove SEP_CHAR (Bruce)
- New GUC hooks (Tom)
- Merge GUC and command line handling (Marko Kreen)
- Remove EXTEND INDEX (Martijn van Oosterhout, Tom)
- New pgjindent utility to indent java code (Bruce)
- Remove define of true/false when compiling under C++ (Leandro Fanzone, Tom)
- pgindent fixes (Bruce, Tom)
- Replace strcasecmp() with strcmp() where appropriate (Peter E)
- Dynahash portability improvements (Tom)
- Add 'volatile' usage in spinlock structures
- Improve signal handling logic (Tom)

E.258.3.23. Contrib

- New contrib/rtree_gist (Oleg Bartunov, Teodor Sigaev)
- New contrib/tsearch full-text indexing (Oleg, Teodor Sigaev)
- Add contrib/dblink for remote database access (Joe Conway)
- contrib/ora2pg Oracle conversion utility (Gilles Darold)
- contrib/xml XML conversion utility (John Gray)
- contrib/fulltextindex fixes (Christopher Kings-Lynne)
- New contrib/fuzzystrmatch with levenshtein and metaphone, soundex merged (Joe Conway)

- Add contrib/intarray boolean queries, binary search, fixes (Oleg Bartunov)
- New pg_upgrade utility (Bruce)
- Add new pg_resetxlog options (Bruce, Tom)

E.259. Release 7.1.3

Release date: 2001-08-15

E.259.1. Migration to Version 7.1.3

A dump/restore is not required for those running 7.1.X.

E.259.2. Changes

```
Remove unused WAL segments of large transactions (Tom)
Multiaction rule fix (Tom)
PL/pgSQL memory allocation fix (Jan)
VACUUM buffer fix (Tom)
Regression test fixes (Tom)
pg_dump fixes for GRANT/REVOKE/comments on views, user-defined
 types (Tom)
Fix subselects with DISTINCT ON or LIMIT (Tom)
BeOS fix
Disable COPY TO/FROM a view (Tom)
Cygwin build (Jason Tishler)
```

E.260. Release 7.1.2

Release date: 2001-05-11

This has one fix from 7.1.1.

E.260.1. Migration to Version 7.1.2

A dump/restore is not required for those running 7.1.X.

E.260.2. Changes

```
Fix PL/pgSQL SELECTs when returning no rows
Fix for psql backslash core dump
Referential integrity privilege fix
Optimizer fixes
pg_dump cleanups
```

E.261. Release 7.1.1

Release date: 2001-05-05

This has a variety of fixes from 7.1.

E.261.1. Migration to Version 7.1.1

A dump/restore is not required for those running 7.1.

E.261.2. Changes

```
Fix for numeric MODULO operator (Tom)
pg_dump fixes (Philip)
pg_dump can dump 7.0 databases (Philip)
readline 4.2 fixes (Peter E)
JOIN fixes (Tom)
AIX, MSWIN, VAX, N32K fixes (Tom)
Multibytes fixes (Tom)
Unicode fixes (Tatsuo)
Optimizer improvements (Tom)
Fix for whole rows in functions (Tom)
Fix for pg_ctl and option strings with spaces (Peter E)
ODBC fixes (Hiroshi)
EXTRACT can now take string argument (Thomas)
Python fixes (Darcy)
```

E.262. Release 7.1

Release date: 2001-04-13

This release focuses on removing limitations that have existed in the PostgreSQL code for many years.

Major changes in this release:

Write-ahead Log (WAL)

To maintain database consistency in case of an operating system crash, previous releases of PostgreSQL have forced all data modifications to disk before each transaction commit. With WAL, only one log file must be flushed to disk, greatly improving performance. If you have been using -F in previous releases to disable disk flushes, you might want to consider discontinuing its use.

TOAST

TOAST - Previous releases had a compiled-in row length limit, typically 8k - 32k. This limit made storage of long text fields difficult. With TOAST, long rows of any length can be stored with good performance.

Outer Joins

We now support outer joins. The UNION/NOT IN workaround for outer joins is no longer required. We use the SQL92 outer join syntax.

Function Manager

The previous C function manager did not handle null values properly, nor did it support 64-bit CPU's (Alpha). The new function manager does. You can continue using your old custom functions, but you might want to rewrite them in the future to use the new function manager call interface.

Complex Queries

A large number of complex queries that were unsupported in previous releases now work. Many combinations of views, aggregates, UNION, LIMIT, cursors, subqueries, and inherited tables now work properly. Inherited tables are now accessed by default. Subqueries in FROM are now supported.

E.262.1. Migration to Version 7.1

A dump/restore using pg_dump is required for those wishing to migrate data from any previous release.

E.262.2. Changes

```
Bug Fixes
---------
Many multibyte/Unicode/locale fixes (Tatsuo and others)
More reliable ALTER TABLE RENAME (Tom)
Kerberos V fixes (David Wragg)
Fix for INSERT INTO...SELECT where targetlist has subqueries (Tom)
Prompt username/password on standard error (Bruce)
Large objects inv_read/inv_write fixes (Tom)
Fixes for to_char(), to_date(), to_ascii(), and to_timestamp() (
Karel,
   Daniel Baldoni)
Prevent query expressions from leaking memory (Tom)
Allow UPDATE of arrays elements (Tom)
Wake up lock waiters during cancel (Hiroshi)
Fix rare cursor crash when using hash join (Tom)
Fix for DROP TABLE/INDEX in rolled-back transaction (Hiroshi)
Fix psql crash from \l+ if MULTIBYTE enabled (Peter E)
Fix truncation of rule names during CREATE VIEW (Ross Reedstrom)
Fix PL/perl (Alex Kapranoff)
Disallow LOCK on views (Mark Hollomon)
Disallow INSERT/UPDATE/DELETE on views (Mark Hollomon)
Disallow DROP RULE, CREATE INDEX, TRUNCATE on views (Mark Hollomon)
Allow PL/pgSQL accept non-ASCII identifiers (Tatsuo)
Allow views to proper handle GROUP BY, aggregates, DISTINCT (Tom)
Fix rare failure with TRUNCATE command (Tom)
Allow UNION/INTERSECT/EXCEPT to be used with ALL, subqueries,
 views,
   DISTINCT, ORDER BY, SELECT...INTO (Tom)
Fix parser failures during aborted transactions (Tom)
Allow temporary relations to properly clean up indexes (Bruce)
Fix VACUUM problem with moving rows in same page (Tom)
Modify pg_dump to better handle user-defined items in template1 (
Philip)
Allow LIMIT in VIEW (Tom)
Require cursor FETCH to honor LIMIT (Tom)
Allow PRIMARY/FOREIGN Key definitions on inherited columns (
Stephan)
Allow ORDER BY, LIMIT in subqueries (Tom)
Allow UNION in CREATE RULE (Tom)
Make ALTER/DROP TABLE rollback-able (Vadim, Tom)
Store initdb collation in pg_control so collation cannot be changed
 (Tom)
Fix INSERT...SELECT with rules (Tom)
Fix FOR UPDATE inside views and subselects (Tom)
Fix OVERLAPS operators conform to SQL92 spec regarding NULLs (Tom)
Fix lpad() and rpad() to handle length less than input string (Tom)
Fix use of NOTIFY in some rules (Tom)
Overhaul btree code (Tom)
Fix NOT NULL use in Pl/pgSQL variables (Tom)
Overhaul GIST code (Oleg)
```

```
Fix CLUSTER to preserve constraints and column default (Tom)
Improved deadlock detection handling (Tom)
Allow multiple SERIAL columns in a table (Tom)
Prevent occasional index corruption (Vadim)

Enhancements
------------
Add OUTER JOINs (Tom)
Function manager overhaul (Tom)
Allow ALTER TABLE RENAME on indexes (Tom)
Improve CLUSTER (Tom)
Improve ps status display for more platforms (Peter E, Marc)
Improve CREATE FUNCTION failure message (Ross)
JDBC improvements (Peter, Travis Bauer, Christopher Cain, William
 Webber,
   Gunnar)
Grand Unified Configuration scheme/GUC.  Many options can now be
 set in
   data/postgresql.conf, postmaster/postgres flags, or SET commands
 (Peter E)
Improved handling of file descriptor cache (Tom)
New warning code about auto-created table alias entries (Bruce)
Overhaul initdb process (Tom, Peter E)
Overhaul of inherited tables; inherited tables now accessed by
 default;
  new ONLY key word prevents it (Chris Bitmead, Tom)
ODBC cleanups/improvements (Nick Gorham, Stephan Szabo, Zoltan
 Kovacs,
   Michael Fork)
Allow renaming of temp tables (Tom)
Overhaul memory manager contexts (Tom)
pg_dumpall uses CREATE USER or CREATE GROUP rather using COPY (
Peter E)
Overhaul pg_dump (Philip Warner)
Allow pg_hba.conf secondary password file to specify only username
 (Peter E)
Allow TEMPORARY or TEMP key word when creating temporary tables (
Bruce)
New memory leak checker (Karel)
New SET SESSION CHARACTERISTICS (Thomas)
Allow nested block comments (Thomas)
Add WITHOUT TIME ZONE type qualifier (Thomas)
New ALTER TABLE ADD CONSTRAINT (Stephan)
Use NUMERIC accumulators for INTEGER aggregates (Tom)
Overhaul aggregate code (Tom)
New VARIANCE and STDDEV() aggregates
Improve dependency ordering of pg_dump (Philip)
New pg_restore command (Philip)
New pg_dump tar output option (Philip)
New pg_dump of large objects  (Philip)
New ESCAPE option to LIKE (Thomas)
New case-insensitive LIKE - ILIKE (Thomas)
Allow functional indexes to use binary-compatible type (Tom)
Allow SQL functions to be used in more contexts (Tom)
New pg_config utility (Peter E)
New PL/pgSQL EXECUTE command which allows dynamic SQL and utility
 statements
   (Jan)
```

```
New PL/pgSQL GET DIAGNOSTICS statement for SPI value access (Jan)
New quote_identifiers() and quote_literal() functions (Jan)
New ALTER TABLE table OWNER TO user command (Mark Hollomon)
Allow subselects in FROM, i.e. FROM (SELECT ...) [AS] alias (Tom)
Update PyGreSQL to version 3.1 (D'Arcy)
Store tables as files named by OID (Vadim)
New SQL function setval(seq,val,bool) for use in pg_dump (Philip)
Require DROP VIEW to remove views, no DROP TABLE (Mark)
Allow DROP VIEW view1, view2 (Mark)
Allow multiple objects in DROP INDEX, DROP RULE, and DROP TYPE (
Tom)
Allow automatic conversion to/from Unicode (Tatsuo, Eiji)
New /contrib/pgcrypto hashing functions (Marko Kreen)
New pg_dumpall --globals-only option (Peter E)
New CHECKPOINT command for WAL which creates new WAL log file (
Vadim)
New AT TIME ZONE syntax (Thomas)
Allow location of Unix domain socket to be configurable (David J.
 MacKenzie)
Allow postmaster to listen on a specific IP address (David J. Mac
Kenzie)
Allow socket path name to be specified in hostname by using leading
 slash
   (David J. MacKenzie)
Allow CREATE DATABASE to specify template database (Tom)
New utility to convert MySQL schema dumps to SQL92 and PostgreSQL (
Thomas)
New /contrib/rserv replication toolkit (Vadim)
New file format for COPY BINARY (Tom)
New /contrib/oid2name to map numeric files to table names (B
 Palmer)
New "idle in transaction" ps status message (Marc)
Update to pgaccess 0.98.7 (Constantin Teodorescu)
pg_ctl now defaults to -w (wait) on shutdown, new -l (log) option
Add rudimentary dependency checking to pg_dump (Philip)

Types
-----
Fix INET/CIDR type ordering and add new functions (Tom)
Make OID behave as an unsigned type (Tom)
Allow BIGINT as synonym for INT8 (Peter E)
New int2 and int8 comparison operators (Tom)
New BIT and BIT VARYING types (Adriaan Joubert, Tom, Peter E)
CHAR() no longer faster than VARCHAR() because of TOAST (Tom)
New GIST seg/cube examples (Gene Selkov)
Improved round(numeric) handling (Tom)
Fix CIDR output formatting (Tom)
New CIDR abbrev() function (Tom)

Performance
-----------
Write-Ahead Log (WAL) to provide crash recovery with less
 performance
   overhead (Vadim)
ANALYZE stage of VACUUM no longer exclusively locks table (Bruce)
Reduced file seeks (Denis Perchine)
Improve BTREE code for duplicate keys (Tom)
Store all large objects in a single table (Denis Perchine, Tom)
```

```
Improve memory allocation performance (Karel, Tom)

Source Code
-----------
New function manager call conventions (Tom)
SGI portability fixes (David Kaelbling)
New configure --enable-syslog option (Peter E)
New BSDI README (Bruce)
configure script moved to top level, not /src (Peter E)
Makefile/configuration/compilation overhaul (Peter E)
New configure --with-python option (Peter E)
Solaris cleanups (Peter E)
Overhaul /contrib Makefiles (Karel)
New OpenSSL configuration option (Magnus, Peter E)
AIX fixes (Andreas)
QNX fixes (Maurizio)
New heap_open(), heap_openr() API (Tom)
Remove colon and semi-colon operators (Thomas)
New pg_class.relkind value for views (Mark Hollomon)
Rename ichar() to chr() (Karel)
New documentation for btrim(), ascii(), chr(), repeat() (Karel)
Fixes for NT/Cygwin (Pete Forman)
AIX port fixes (Andreas)
New BeOS port (David Reid, Cyril Velter)
Add proofreader's changes to docs (Addison-Wesley, Bruce)
New Alpha spinlock code (Adriaan Joubert, Compaq)
UnixWare port overhaul (Peter E)
New Darwin/Mac OS X port (Peter Bierman, Bruce Hartzler)
New FreeBSD Alpha port (Alfred)
Overhaul shared memory segments (Tom)
Add IBM S/390 support (Neale Ferguson)
Moved macmanuf to /contrib (Larry Rosenman)
Syslog improvements (Larry Rosenman)
New template0 database that contains no user additions (Tom)
New /contrib/cube and /contrib/seg GIST sample code (Gene Selkov)
Allow NetBSD's libedit instead of readline (Peter)
Improved assembly language source code format (Bruce)
New contrib/pg_logger
New --template option to createdb
New contrib/pg_control utility (Oliver)
New FreeBSD tools ipc_check, start-scripts/freebsd
```

E.263. Release 7.0.3

Release date: 2000-11-11

This has a variety of fixes from 7.0.2.

E.263.1. Migration to Version 7.0.3

A dump/restore is not required for those running 7.0.*.

E.263.2. Changes

```
Jdbc fixes (Peter)
Large object fix (Tom)
Fix lean in COPY WITH OIDS leak (Tom)
```

```
Fix backwards-index-scan (Tom)
Fix SELECT ... FOR UPDATE so it checks for duplicate keys (Hiroshi)
Add --enable-syslog to configure (Marc)
Fix abort transaction at backend exit in rare cases (Tom)
Fix for psql \l+ when multibyte enabled (Tatsuo)
Allow PL/pgSQL to accept non ascii identifiers (Tatsuo)
Make vacuum always flush buffers (Tom)
Fix to allow cancel while waiting for a lock (Hiroshi)
Fix for memory allocation problem in user authentication code (Tom)
Remove bogus use of int4out() (Tom)
Fixes for multiple subqueries in COALESCE or BETWEEN (Tom)
Fix for failure of triggers on heap open in certain cases (Jeroen
 van
   Vianen)
Fix for erroneous selectivity of not-equals (Tom)
Fix for erroneous use of strcmp() (Tom)
Fix for bug where storage manager accesses items beyond end of file
   (Tom)
Fix to include kernel errno message in all smgr elog messages (Tom)
Fix for '.' not in PATH at build time (SL Baur)
Fix for out-of-file-descriptors error (Tom)
Fix to make pg_dump dump 'iscachable' flag for functions (Tom)
Fix for subselect in targetlist of Append node (Tom)
Fix for mergejoin plans (Tom)
Fix TRUNCATE failure on relations with indexes (Tom)
Avoid database-wide restart on write error (Hiroshi)
Fix nodeMaterial to honor chgParam by recomputing its output (Tom)
Fix VACUUM problem with moving chain of update row versions when
 source
   and destination of a row version lie on the same page (Tom)
Fix user.c CommandCounterIncrement (Tom)
Fix for AM/PM boundary problem in to_char() (Karel Zak)
Fix TIME aggregate handling (Tom)
Fix to_char() to avoid coredump on NULL input (Tom)
Buffer fix (Tom)
Fix for inserting/copying longer multibyte strings into char() data
   types (Tatsuo)
Fix for crash of backend, on abort (Tom)
```

E.264. Release 7.0.2

Release date: 2000-06-05

This is a repackaging of 7.0.1 with added documentation.

E.264.1. Migration to Version 7.0.2

A dump/restore is not required for those running 7.*.

E.264.2. Changes

```
Added documentation to tarball.
```

E.265. Release 7.0.1

Release date: 2000-06-01

This is a cleanup release for 7.0.

E.265.1. Migration to Version 7.0.1

A dump/restore is not required for those running 7.0.

E.265.2. Changes

```
Fix many CLUSTER failures (Tom)
Allow ALTER TABLE RENAME works on indexes (Tom)
Fix plpgsql to handle datetime->timestamp and timespan->interval (
Bruce)
New configure --with-setproctitle switch to use setproctitle() (
Marc, Bruce)
Fix the off by one errors in ResultSet from 6.5.3, and more.
jdbc ResultSet fixes (Joseph Shraibman)
optimizer tunings (Tom)
Fix create user for pgaccess
Fix for UNLISTEN failure
IRIX fixes (David Kaelbling)
QNX fixes (Andreas Kardos)
Reduce COPY IN lock level (Tom)
Change libpqeasy to use PQconnectdb() style parameters (Bruce)
Fix pg_dump to handle OID indexes (Tom)
Fix small memory leak (Tom)
Solaris fix for createdb/dropdb (Tatsuo)
Fix for non-blocking connections (Alfred Perlstein)
Fix improper recovery after RENAME TABLE failures (Tom)
Copy pg_ident.conf.sample into /lib directory in install (Bruce)
Add SJIS UDC (NEC selection IBM kanji) support (Eiji Tokuya)
Fix too long syslog message (Tatsuo)
Fix problem with quoted indexes that are too long (Tom)
JDBC ResultSet.getTimestamp() fix (Gregory Krasnow & Floyd
 Marinescu)
ecpg changes (Michael)
```

E.266. Release 7.0

Release date: 2000-05-08

This release contains improvements in many areas, demonstrating the continued growth of PostgreSQL. There are more improvements and fixes in 7.0 than in any previous release. The developers have confidence that this is the best release yet; we do our best to put out only solid releases, and this one is no exception.

Major changes in this release:

Foreign Keys

Foreign keys are now implemented, with the exception of PARTIAL MATCH foreign keys. Many users have been asking for this feature, and we are pleased to offer it.

Optimizer Overhaul

Continuing on work started a year ago, the optimizer has been improved, allowing better query plan selection and faster performance with less memory usage.

Updated psql

psql, our interactive terminal monitor, has been updated with a variety of new features. See the psql manual page for details.

Join Syntax

SQL92 join syntax is now supported, though only as `INNER JOIN` for this release. `JOIN`, `NATURAL JOIN`, `JOIN/USING`, and `JOIN/ON` are available, as are column correlation names.

E.266.1. Migration to Version 7.0

A dump/restore using pg_dump is required for those wishing to migrate data from any previous release of PostgreSQL. For those upgrading from 6.5.*, you can instead use pg_upgrade to upgrade to this release; however, a full dump/reload installation is always the most robust method for upgrades.

Interface and compatibility issues to consider for the new release include:

- The date/time types `datetime` and `timespan` have been superseded by the SQL92-defined types `timestamp` and `interval`. Although there has been some effort to ease the transition by allowing PostgreSQL to recognize the deprecated type names and translate them to the new type names, this mechanism cannot be completely transparent to your existing application.
- The optimizer has been substantially improved in the area of query cost estimation. In some cases, this will result in decreased query times as the optimizer makes a better choice for the preferred plan. However, in a small number of cases, usually involving pathological distributions of data, your query times might go up. If you are dealing with large amounts of data, you might want to check your queries to verify performance.
- The JDBC and ODBC interfaces have been upgraded and extended.
- The string function `CHAR_LENGTH` is now a native function. Previous versions translated this into a call to `LENGTH`, which could result in ambiguity with other types implementing `LENGTH` such as the geometric types.

E.266.2. Changes

```
Bug Fixes
---------
Prevent function calls exceeding maximum number of arguments (Tom)
Improve CASE construct (Tom)
Fix SELECT coalesce(f1,0) FROM int4_tbl GROUP BY f1 (Tom)
Fix SELECT sentence.words[0] FROM sentence GROUP BY sentence.words[
0] (Tom)
Fix GROUP BY scan bug (Tom)
Improvements in SQL grammar processing (Tom)
Fix for views involved in INSERT ... SELECT ... (Tom)
Fix for SELECT a/2, a/2 FROM test_missing_target GROUP BY a/2 (Tom)
Fix for subselects in INSERT ... SELECT (Tom)
Prevent INSERT ... SELECT ... ORDER BY (Tom)
Fixes for relations greater than 2GB, including vacuum
Improve propagating system table changes to other backends (Tom)
Improve propagating user table changes to other backends (Tom)
Fix handling of temp tables in complex situations (Bruce, Tom)
Allow table locking at table open, improving concurrent reliability
 (Tom)
```

```
Properly quote sequence names in pg_dump (Ross J. Reedstrom)
Prevent DROP DATABASE while others accessing
Prevent any rows from being returned by GROUP BY if no rows
 processed (Tom)
Fix SELECT COUNT(1) FROM table WHERE ...' if no rows matching WHERE
 (Tom)
Fix pg_upgrade so it works for MVCC (Tom)
Fix for SELECT ... WHERE x IN (SELECT ... HAVING SUM(x) > 1) (Tom)
Fix for "f1 datetime DEFAULT 'now'"  (Tom)
Fix problems with CURRENT_DATE used in DEFAULT (Tom)
Allow comment-only lines, and ;;; lines too. (Tom)
Improve recovery after failed disk writes, disk full (Hiroshi)
Fix cases where table is mentioned in FROM but not joined (Tom)
Allow HAVING clause without aggregate functions (Tom)
Fix for "--" comment and no trailing newline, as seen in perl
 interface
Improve pg_dump failure error reports (Bruce)
Allow sorts and hashes to exceed 2GB file sizes (Tom)
Fix for pg_dump dumping of inherited rules (Tom)
Fix for NULL handling comparisons (Tom)
Fix inconsistent state caused by failed CREATE/DROP commands (
Hiroshi)
Fix for dbname with dash
Prevent DROP INDEX from interfering with other backends (Tom)
Fix file descriptor leak in verify_password()
Fix for "Unable to identify an operator =$" problem
Fix ODBC so no segfault if CommLog and Debug enabled (Dirk
 Niggemann)
Fix for recursive exit call (Massimo)
Fix for extra-long timezones (Jeroen van Vianen)
Make pg_dump preserve primary key information (Peter E)
Prevent databases with single quotes (Peter E)
Prevent DROP DATABASE inside  transaction (Peter E)
ecpg memory leak fixes (Stephen Birch)
Fix for SELECT null::text, SELECT int4fac(null) and SELECT 2 + (
null) (Tom)
Y2K timestamp fix (Massimo)
Fix for VACUUM 'HEAP_MOVED_IN was not expected' errors (Tom)
Fix for views with tables/columns containing spaces  (Tom)
Prevent privileges on indexes (Peter E)
Fix for spinlock stuck problem when error is generated (Hiroshi)
Fix ipcclean on Linux
Fix handling of NULL constraint conditions (Tom)
Fix memory leak in odbc driver (Nick Gorham)
Fix for privilege check on UNION tables (Tom)
Fix to allow SELECT 'a' LIKE 'a' (Tom)
Fix for SELECT 1 + NULL (Tom)
Fixes to CHAR
Fix log() on numeric type (Tom)
Deprecate ':' and ';' operators
Allow vacuum of temporary tables
Disallow inherited columns with the same name as new columns
Recover or force failure when disk space is exhausted (Hiroshi)
Fix INSERT INTO ... SELECT with AS columns matching result columns
Fix INSERT ... SELECT ... GROUP BY groups by target columns not
 source columns (Tom)
Fix CREATE TABLE test (a char(5) DEFAULT text '', b int4) with
 INSERT (Tom)
```

```
Fix UNION with LIMIT
Fix CREATE TABLE x AS SELECT 1 UNION SELECT 2
Fix CREATE TABLE test(col char(2) DEFAULT user)
Fix mismatched types in CREATE TABLE ... DEFAULT
Fix SELECT * FROM pg_class where oid in (0,-1)
Fix SELECT COUNT('asdf') FROM pg_class WHERE oid=12
Prevent user who can create databases can modifying pg_database
 table (Peter E)
Fix btree to give a useful elog when key > 1/2 (page - overhead) (
Tom)
Fix INSERT of 0.0 into DECIMAL(4,4) field (Tom)

Enhancements
------------
New CLI interface include file sqlcli.h, based on SQL3/SQL98
Remove all limits on query length, row length limit still exists (
Tom)
Update jdbc protocol to 2.0 (Jens Glaser <jens@jens.de>)
Add TRUNCATE command to quickly truncate relation (Mike Mascari)
Fix to give super user and createdb user proper update catalog
 rights (Peter E)
Allow ecpg bool variables to have NULL values (Christof)
Issue ecpg error if NULL value for variable with no NULL indicator
 (Christof)
Allow ^C to cancel COPY command (Massimo)
Add SET FSYNC and SHOW PG_OPTIONS commands(Massimo)
Function name overloading for dynamically-loaded C functions (
Frankpitt)
Add CmdTuples() to libpq++(Vince)
New CREATE CONSTRAINT TRIGGER and SET CONSTRAINTS commands(Jan)
Allow CREATE FUNCTION/WITH clause to be used for all language types
configure --enable-debug adds -g (Peter E)
configure --disable-debug removes -g (Peter E)
Allow more complex default expressions (Tom)
First real FOREIGN KEY constraint trigger functionality (Jan)
Add FOREIGN KEY ... MATCH FULL ... ON DELETE CASCADE (Jan)
Add FOREIGN KEY ... MATCH <unspecified> referential actions (Don
 Baccus)
Allow WHERE restriction on ctid (physical heap location) (Hiroshi)
Move pginterface from contrib to interface directory, rename to
 pgeasy (Bruce)
Change pgeasy connectdb() parameter ordering (Bruce)
Require SELECT DISTINCT target list to have all ORDER BY columns (
Tom)
Add Oracle's COMMENT ON command (Mike Mascari <mascarim@yahoo.com>)
libpq's PQsetNoticeProcessor function now returns previous hook(
Peter E)
Prevent PQsetNoticeProcessor from being set to NULL (Peter E)
Make USING in COPY optional (Bruce)
Allow subselects in the target list (Tom)
Allow subselects on the left side of comparison operators (Tom)
New parallel regression test (Jan)
Change backend-side COPY to write files with permissions 644 not
 666 (Tom)
Force permissions on PGDATA directory to be secure, even if it
 exists (Tom)
Added psql LASTOID variable to return last inserted oid (Peter E)
Allow concurrent vacuum and remove pg_vlock vacuum lock file (Tom)
```

```
Add privilege check for vacuum (Peter E)
New libpq functions to allow asynchronous connections: PQconnect
Start(),
  PQconnectPoll(), PQresetStart(), PQresetPoll(), PQsetenvStart(),
  PQsetenvPoll(), PQsetenvAbort (Ewan Mellor)
New libpq PQsetenv() function (Ewan Mellor)
create/alter user extension (Peter E)
New postmaster.pid and postmaster.opts under $PGDATA (Tatsuo)
New scripts for create/drop user/db (Peter E)
Major psql overhaul (Peter E)
Add const to libpq interface (Peter E)
New libpq function PQoidValue (Peter E)
Show specific non-aggregate causing problem with GROUP BY (Tom)
Make changes to pg_shadow recreate pg_pwd file (Peter E)
Add aggregate(DISTINCT ...) (Tom)
Allow flag to control COPY input/output of NULLs (Peter E)
Make postgres user have a password by default (Peter E)
Add CREATE/ALTER/DROP GROUP (Peter E)
All administration scripts now support --long options (Peter E,
 Karel)
Vacuumdb script now supports --all option (Peter E)
ecpg new portable FETCH syntax
Add ecpg EXEC SQL IFDEF, EXEC SQL IFNDEF, EXEC SQL ELSE, EXEC SQL
 ELIF
       and EXEC SQL ENDIF directives
Add pg_ctl script to control backend start-up (Tatsuo)
Add postmaster.opts.default file to store start-up flags (Tatsuo)
Allow --with-mb=SQL_ASCII
Increase maximum number of index keys to 16 (Bruce)
Increase maximum number of function arguments to 16 (Bruce)
Allow configuration of maximum number of index keys and arguments (
Bruce)
Allow unprivileged users to change their passwords (Peter E)
Password authentication enabled; required for new users (Peter E)
Disallow dropping a user who owns a database (Peter E)
Change initdb option --with-mb to --enable-multibyte
Add option for initdb to prompts for superuser password (Peter E)
Allow complex type casts like col::numeric(9,2) and col::int2::
float8 (Tom)
Updated user interfaces on initdb, initlocation, pg_dump, ipcclean
 (Peter E)
New pg_char_to_encoding() and pg_encoding_to_char() functions (
Tatsuo)
libpq non-blocking mode (Alfred Perlstein)
Improve conversion of types in casts that don't specify a length
New plperl internal programming language (Mark Hollomon)
Allow COPY IN to read file that do not end with a newline (Tom)
Indicate when long identifiers are truncated (Tom)
Allow aggregates to use type equivalency (Peter E)
Add Oracle's to_char(), to_date(), to_datetime(), to_timestamp(),
 to_number()
       conversion functions (Karel Zak <zakkr@zf.jcu.cz>)
Add SELECT DISTINCT ON (expr [, expr ...]) targetlist ... (Tom)
Check to be sure ORDER BY is compatible with the DISTINCT operation
 (Tom)
Add NUMERIC and int8 types to ODBC
Improve EXPLAIN results for Append, Group, Agg, Unique (Tom)
Add ALTER TABLE ... ADD FOREIGN KEY (Stephan Szabo)
```

```
Allow SELECT .. FOR UPDATE in PL/pgSQL (Hiroshi)
Enable backward sequential scan even after reaching EOF (Hiroshi)
Add btree indexing of boolean values, >= and <= (Don Baccus)
Print current line number when COPY FROM fails (Massimo)
Recognize POSIX time zone e.g. "PST+8" and "GMT-8" (Thomas)
Add DEC as synonym for DECIMAL (Thomas)
Add SESSION_USER as SQL92 key word, same as CURRENT_USER (Thomas)
Implement SQL92 column aliases (aka correlation names) (Thomas)
Implement SQL92 join syntax (Thomas)
Make INTERVAL reserved word allowed as a column identifier (Thomas)
Implement REINDEX command (Hiroshi)
Accept ALL in aggregate function SUM(ALL col) (Tom)
Prevent GROUP BY from using column aliases (Tom)
New psql \encoding option (Tatsuo)
Allow PQrequestCancel() to terminate when in waiting-for-lock state
 (Hiroshi)
Allow negation of a negative number in all cases
Add ecpg descriptors (Christof, Michael)
Allow CREATE VIEW v AS SELECT f1::char(8) FROM tbl
Allow casts with length, like foo::char(8)
New libpq functions PQsetClientEncoding(), PQclientEncoding() (
Tatsuo)
Add support for SJIS user defined characters (Tatsuo)
Larger views/rules supported
Make libpq's PQconndefaults() thread-safe (Tom)
Disable // as comment to be ANSI conforming, should use -- (Tom)
Allow column aliases on views CREATE VIEW name (collist)
Fixes for views with subqueries (Tom)
Allow UPDATE table SET fld = (SELECT ...) (Tom)
SET command options no longer require quotes
Update pgaccess to 0.98.6
New SET SEED command
New pg_options.sample file
New SET FSYNC command (Massimo)
Allow pg_descriptions when creating tables
Allow pg_descriptions when creating types, columns, and functions
Allow psql \copy to allow delimiters (Peter E)
Allow psql to print nulls as distinct from "" [null] (Peter E)

Types
-----
Many array fixes (Tom)
Allow bare column names to be subscripted as arrays (Tom)
Improve type casting of int and float constants (Tom)
Cleanups for int8 inputs, range checking, and type conversion (Tom)
Fix for SELECT timespan('21:11:26'::time) (Tom)
netmask('x.x.x.x/0') is 255.255.255.255 instead of 0.0.0.0 (Oleg
 Sharoiko)
Add btree index on NUMERIC (Jan)
Perl fix for large objects containing NUL characters (Douglas
 Thomson)
ODBC fix for large objects (free)
Fix indexing of cidr data type
Fix for Ethernet MAC addresses (macaddr type) comparisons
Fix for date/time types when overflows happened in computations (
Tom)
Allow array on int8 (Peter E)
Fix for rounding/overflow of NUMERIC type, like NUMERIC(4,4) (Tom)
```

```
Allow NUMERIC arrays
Fix bugs in NUMERIC ceil() and floor() functions (Tom)
Make char_length()/octet_length including trailing blanks (Tom)
Made abstime/reltime use int4 instead of time_t (Peter E)
New lztext data type for compressed text fields
Revise code to handle coercion of int and float constants (Tom)
Start at new code to implement a BIT and BIT VARYING type (Adriaan
 Joubert)
NUMERIC now accepts scientific notation (Tom)
NUMERIC to int4 rounds (Tom)
Convert float4/8 to NUMERIC properly (Tom)
Allow type conversion with NUMERIC (Thomas)
Make ISO date style (2000-02-16 09:33) the default (Thomas)
Add NATIONAL CHAR [ VARYING ] (Thomas)
Allow NUMERIC round and trunc to accept negative scales (Tom)
New TIME WITH TIME ZONE type (Thomas)
Add MAX()/MIN() on time type (Thomas)
Add abs(), mod(), fac() for int8 (Thomas)
Rename functions to round(), sqrt(), cbrt(), pow() for float8 (
Thomas)
Add transcendental math functions (e.g. sin(), acos()) for float8 (
Thomas)
Add exp() and ln() for NUMERIC type
Rename NUMERIC power() to pow() (Thomas)
Improved TRANSLATE() function (Edwin Ramirez, Tom)
Allow X=-Y operators  (Tom)
Allow SELECT float8(COUNT(*))/(SELECT COUNT(*) FROM t) FROM t GROUP
 BY f1; (Tom)
Allow LOCALE to use indexes in regular expression searches (Tom)
Allow creation of functional indexes to use default types

Performance
-----------
Prevent exponential space consumption with many AND's and OR's (
Tom)
Collect attribute selectivity values for system columns (Tom)
Reduce memory usage of aggregates (Tom)
Fix for LIKE optimization to use indexes with multibyte encodings (
Tom)
Fix r-tree index optimizer selectivity (Thomas)
Improve optimizer selectivity computations and functions (Tom)
Optimize btree searching for cases where many equal keys exist (
Tom)
Enable fast LIKE index processing only if index present (Tom)
Re-use free space on index pages with duplicates (Tom)
Improve hash join processing (Tom)
Prevent descending sort if result is already sorted(Hiroshi)
Allow commuting of index scan query qualifications (Tom)
Prefer index scans in cases where ORDER BY/GROUP BY is required (
Tom)
Allocate large memory requests in fix-sized chunks for performance
 (Tom)
Fix vacuum's performance by reducing memory allocation requests (
Tom)
Implement constant-expression simplification (Bernard Frankpitt,
 Tom)
Use secondary columns to be used to determine start of index scan (
Hiroshi)
```

```
Prevent quadruple use of disk space when doing internal sorting (
Tom)
Faster sorting by calling fewer functions (Tom)
Create system indexes to match all system caches (Bruce, Hiroshi)
Make system caches use system indexes (Bruce)
Make all system indexes unique (Bruce)
Improve pg_statistics management for VACUUM speed improvement (Tom)
Flush backend cache less frequently (Tom, Hiroshi)
COPY now reuses previous memory allocation, improving performance (
Tom)
Improve optimization cost estimation (Tom)
Improve optimizer estimate of range queries x > lowbound AND x <
 highbound (Tom)
Use DNF instead of CNF where appropriate (Tom, Taral)
Further cleanup for OR-of-AND WHERE-clauses (Tom)
Make use of index in OR clauses (x = 1 AND y = 2) OR (x = 2 AND y =
 4) (Tom)
Smarter optimizer computations for random index page access (Tom)
New SET variable to control optimizer costs (Tom)
Optimizer queries based on LIMIT, OFFSET, and EXISTS qualifications
 (Tom)
Reduce optimizer internal housekeeping of join paths for speedup (
Tom)
Major subquery speedup (Tom)
Fewer fsync writes when fsync is not disabled (Tom)
Improved LIKE optimizer estimates (Tom)
Prevent fsync in SELECT-only queries (Vadim)
Make index creation use psort code, because it is now faster (Tom)
Allow creation of sort temp tables > 1 Gig

Source Tree Changes
-------------------
Fix for linux PPC compile
New generic expression-tree-walker subroutine (Tom)
Change form() to varargform() to prevent portability problems
Improved range checking for large integers on Alphas
Clean up #include in /include directory (Bruce)
Add scripts for checking includes (Bruce)
Remove un-needed #include's from *.c files (Bruce)
Change #include's to use <> and "" as appropriate (Bruce)
Enable Windows compilation of libpq
Alpha spinlock fix from Uncle George <gatgul@voicenet.com>
Overhaul of optimizer data structures (Tom)
Fix to cygipc library (Yutaka Tanida)
Allow pgsql to work on newer Cygwin snapshots (Dan)
New catalog version number (Tom)
Add Linux ARM
Rename heap_replace to heap_update
Update for QNX (Dr. Andreas Kardos)
New platform-specific regression handling (Tom)
Rename oid8 -> oidvector and int28 -> int2vector (Bruce)
Included all yacc and lex files into the distribution (Peter E.)
Remove lextest, no longer needed (Peter E)
Fix for libpq and psql on Windows (Magnus)
Internally change datetime and timespan into timestamp and interval
 (Thomas)
Fix for plpgsql on BSD/OS
Add SQL_ASCII test case to the regression test (Tatsuo)
```

```
configure --with-mb now deprecated (Tatsuo)
NT fixes
NetBSD fixes (Johnny C. Lam <lamj@stat.cmu.edu>)
Fixes for Alpha compiles
New multibyte encodings
```

E.267. Release 6.5.3

Release date: 1999-10-13

This is basically a cleanup release for 6.5.2. We have added a new PgAccess that was missing in 6.5.2, and installed an NT-specific fix.

E.267.1. Migration to Version 6.5.3

A dump/restore is not required for those running 6.5.*.

E.267.2. Changes

```
Updated version of pgaccess 0.98
NT-specific patch
Fix dumping rules on inherited tables
```

E.268. Release 6.5.2

Release date: 1999-09-15

This is basically a cleanup release for 6.5.1. We have fixed a variety of problems reported by 6.5.1 users.

E.268.1. Migration to Version 6.5.2

A dump/restore is not required for those running 6.5.*.

E.268.2. Changes

```
subselect+CASE fixes(Tom)
Add SHLIB_LINK setting for solaris_i386 and solaris_sparc ports(
Daren Sefcik)
Fixes for CASE in WHERE join clauses(Tom)
Fix BTScan abort(Tom)
Repair the check for redundant UNIQUE and PRIMARY KEY indexes(
Thomas)
Improve it so that it checks for multicolumn constraints(Thomas)
Fix for Windows making problem with MB enabled(Hiroki Kataoka)
Allow BSD yacc and bison to compile pl code(Bruce)
Fix SET NAMES working
int8 fixes(Thomas)
Fix vacuum's memory consumption(Hiroshi,Tatsuo)
Reduce the total memory consumption of vacuum(Tom)
Fix for timestamp(datetime)
Rule deparsing bugfixes(Tom)
Fix quoting problems in mkMakefile.tcldefs.sh.in and mkMakefile.
tkdefs.sh.in(Tom)
This is to re-use space on index pages freed by vacuum(Vadim)
document -x for pg_dump(Bruce)
```

```
Fix for unary operators in rule deparser(Tom)
Comment out FileUnlink of excess segments during mdtruncate()(Tom)
IRIX linking fix from Yu Cao >yucao@falcon.kla-tencor.com<
Repair logic error in LIKE: should not return LIKE_ABORT
  when reach end of pattern before end of text(Tom)
Repair incorrect cleanup of heap memory allocation during
 transaction abort(Tom)
Updated version of pgaccess 0.98
```

E.269. Release 6.5.1

Release date: 1999-07-15

This is basically a cleanup release for 6.5. We have fixed a variety of problems reported by 6.5 users.

E.269.1. Migration to Version 6.5.1

A dump/restore is not required for those running 6.5.

E.269.2. Changes

```
Add NT README file
Portability fixes for linux_ppc, IRIX, linux_alpha, OpenBSD, alpha
Remove QUERY_LIMIT, use SELECT...LIMIT
Fix for EXPLAIN on inheritance(Tom)
Patch to allow vacuum on multisegment tables(Hiroshi)
R-Tree optimizer selectivity fix(Tom)
ACL file descriptor leak fix(Atsushi Ogawa)
New expression subtree code(Tom)
Avoid disk writes for read-only transactions(Vadim)
Fix for removal of temp tables if last transaction was aborted(
Bruce)
Fix to prevent too large row from being created(Bruce)
plpgsql fixes
Allow port numbers 32k - 64k(Bruce)
Add ^ precedence(Bruce)
Rename sort files called pg_temp to pg_sorttemp(Bruce)
Fix for microseconds in time values(Tom)
Tutorial source cleanup
New linux_m68k port
Fix for sorting of NULL's in some cases(Tom)
Shared library dependencies fixed (Tom)
Fixed glitches affecting GROUP BY in subselects(Tom)
Fix some compiler warnings (Tomoaki Nishiyama)
Add Win1250 (Czech) support (Pavel Behal)
```

E.270. Release 6.5

Release date: 1999-06-09

This release marks a major step in the development team's mastery of the source code we inherited from Berkeley. You will see we are now easily adding major features, thanks to the increasing size and experience of our world-wide development team.

Here is a brief summary of the more notable changes:

Multiversion concurrency control(MVCC)

This removes our old table-level locking, and replaces it with a locking system that is superior to most commercial database systems. In a traditional system, each row that is modified is locked until committed, preventing reads by other users. MVCC uses the natural multiversion nature of PostgreSQL to allow readers to continue reading consistent data during writer activity. Writers continue to use the compact pg_log transaction system. This is all performed without having to allocate a lock for every row like traditional database systems. So, basically, we no longer are restricted by simple table-level locking; we have something better than row-level locking.

Hot backups from pg_dump

pg_dump takes advantage of the new MVCC features to give a consistent database dump/backup while the database stays online and available for queries.

Numeric data type

We now have a true numeric data type, with user-specified precision.

Temporary tables

Temporary tables are guaranteed to have unique names within a database session, and are destroyed on session exit.

New SQL features

We now have CASE, INTERSECT, and EXCEPT statement support. We have new LIMIT/OFFSET, SET TRANSACTION ISOLATION LEVEL, SELECT ... FOR UPDATE, and an improved LOCK TABLE command.

Speedups

We continue to speed up PostgreSQL, thanks to the variety of talents within our team. We have sped up memory allocation, optimization, table joins, and row transfer routines.

Ports

We continue to expand our port list, this time including Windows NT/ix86 and NetBSD/arm32.

Interfaces

Most interfaces have new versions, and existing functionality has been improved.

Documentation

New and updated material is present throughout the documentation. New FAQs have been contributed for SGI and AIX platforms. The Tutorial has introductory information on SQL from Stefan Simkovics. For the User's Guide, there are reference pages covering the postmaster and more utility programs, and a new appendix contains details on date/time behavior. The Administrator's Guide has a new chapter on troubleshooting from Tom Lane. And the Programmer's Guide has a description of query processing, also from Stefan, and details on obtaining the PostgreSQL source tree via anonymous CVS and CVSup.

E.270.1. Migration to Version 6.5

A dump/restore using pg_dump is required for those wishing to migrate data from any previous release of PostgreSQL. pg_upgrade can not be used to upgrade to this

release because the on-disk structure of the tables has changed compared to previous releases.

The new Multiversion Concurrency Control (MVCC) features can give somewhat different behaviors in multiuser environments. Read and understand the following section to ensure that your existing applications will give you the behavior you need.

E.270.1.1. Multiversion Concurrency Control

Because readers in 6.5 don't lock data, regardless of transaction isolation level, data read by one transaction can be overwritten by another. In other words, if a row is returned by `SELECT` it doesn't mean that this row really exists at the time it is returned (i.e. sometime after the statement or transaction began) nor that the row is protected from being deleted or updated by concurrent transactions before the current transaction does a commit or rollback.

To ensure the actual existence of a row and protect it against concurrent updates one must use `SELECT FOR UPDATE` or an appropriate `LOCK TABLE` statement. This should be taken into account when porting applications from previous releases of PostgreSQL and other environments.

Keep the above in mind if you are using `contrib/refint.*` triggers for referential integrity. Additional techniques are required now. One way is to use `LOCK parent_table IN SHARE ROW EXCLUSIVE MODE` command if a transaction is going to update/delete a primary key and use `LOCK parent_table IN SHARE MODE` command if a transaction is going to update/insert a foreign key.

> **注意**
>
> Note that if you run a transaction in SERIALIZABLE mode then you must execute the `LOCK` commands above before execution of any DML statement (`SELECT/INSERT/DELETE/UPDATE/FETCH/COPY_TO`) in the transaction.

These inconveniences will disappear in the future when the ability to read dirty (uncommitted) data (regardless of isolation level) and true referential integrity will be implemented.

E.270.2. Changes

```
Bug Fixes
---------
Fix text<->float8 and text<->float4 conversion functions(Thomas)
Fix for creating tables with mixed-case constraints(Billy)
Change exp()/pow() behavior to generate error on underflow/
overflow(Jan)
Fix bug in pg_dump -z
Memory overrun cleanups(Tatsuo)
Fix for lo_import crash(Tatsuo)
Adjust handling of data type names to suppress double quotes(
Thomas)
Use type coercion for matching columns and DEFAULT(Thomas)
Fix deadlock so it only checks once after one second of sleep(
Bruce)
Fixes for aggregates and PL/pgsql(Hiroshi)
Fix for subquery crash(Vadim)
Fix for libpq function PQfnumber and case-insensitive names(Bahman
 Rafatjoo)
```

```
Fix for large object write-in-middle, no extra block, memory
 consumption(Tatsuo)
Fix for pg_dump -d or -D and  quote special characters in INSERT
Repair serious problems with dynahash(Tom)
Fix INET/CIDR portability problems
Fix problem with selectivity error in ALTER TABLE ADD COLUMN(Bruce)
Fix executor so mergejoin of different column types works(Tom)
Fix for Alpha OR selectivity bug
Fix OR index selectivity problem(Bruce)
Fix so \d shows proper length for char()/varchar()(Ryan)
Fix tutorial code(Clark)
Improve destroyuser checking(Oliver)
Fix for Kerberos(Rodney McDuff)
Fix for dropping database while dirty buffers(Bruce)
Fix so sequence nextval() can be case-sensitive(Bruce)
Fix !!= operator
Drop buffers before destroying database files(Bruce)
Fix case where executor evaluates functions twice(Tatsuo)
Allow sequence nextval actions to be case-sensitive(Bruce)
Fix optimizer indexing not working for negative numbers(Bruce)
Fix for memory leak in executor with fjIsNull
Fix for aggregate memory leaks(Erik Riedel)
Allow user name containing a dash to grant privileges
Cleanup of NULL in inet types
Clean up system table bugs(Tom)
Fix problems of PAGER and \? command(Masaaki Sakaida)
Reduce default multisegment file size limit to 1GB(Peter)
Fix for dumping of CREATE OPERATOR(Tom)
Fix for backward scanning of cursors(Hiroshi Inoue)
Fix for COPY FROM STDIN when using \i(Tom)
Fix for subselect is compared inside an expression(Jan)
Fix handling of error reporting while returning rows(Tom)
Fix problems with reference to array types(Tom,Jan)
Prevent UPDATE SET oid(Jan)
Fix pg_dump so -t option can handle case-sensitive tablenames
Fixes for GROUP BY in special cases(Tom, Jan)
Fix for memory leak in failed queries(Tom)
DEFAULT now supports mixed-case identifiers(Tom)
Fix for multisegment uses of DROP/RENAME table, indexes(Ole Gjerde)
Disable use of pg_dump with both -o and -d options(Bruce)
Allow pg_dump to properly dump group privileges(Bruce)
Fix GROUP BY in INSERT INTO table SELECT * FROM table2(Jan)
Fix for computations in views(Jan)
Fix for aggregates on array indexes(Tom)
Fix for DEFAULT handles single quotes in value requiring too many
 quotes
Fix security problem with non-super users importing/exporting large
 objects(Tom)
Rollback of transaction that creates table cleaned up properly(Tom)
Fix to allow long table and column names to generate proper serial
 names(Tom)

Enhancements
------------
Add "vacuumdb" utility
Speed up libpq by allocating memory better(Tom)
EXPLAIN all indexes used(Tom)
Implement CASE, COALESCE, NULLIF  expression(Thomas)
```

```
New pg_dump table output format(Constantin)
Add string min()/max() functions(Thomas)
Extend new type coercion techniques to aggregates(Thomas)
New moddatetime contrib(Terry)
Update to pgaccess 0.96(Constantin)
Add routines for single-byte "char" type(Thomas)
Improved substr() function(Thomas)
Improved multibyte handling(Tatsuo)
Multiversion concurrency control/MVCC(Vadim)
New Serialized mode(Vadim)
Fix for tables over 2gigs(Peter)
New SET TRANSACTION ISOLATION LEVEL(Vadim)
New LOCK TABLE IN ... MODE(Vadim)
Update ODBC driver(Byron)
New NUMERIC data type(Jan)
New SELECT FOR UPDATE(Vadim)
Handle "NaN" and "Infinity" for input values(Jan)
Improved date/year handling(Thomas)
Improved handling of backend connections(Magnus)
New options ELOG_TIMESTAMPS and USE_SYSLOG options for log files(
Massimo)
New TCL_ARRAYS option(Massimo)
New INTERSECT and EXCEPT(Stefan)
New pg_index.indisprimary for primary key tracking(D'Arcy)
New pg_dump option to allow dropping of tables before creation(
Brook)
Speedup of row output routines(Tom)
New READ COMMITTED isolation level(Vadim)
New TEMP tables/indexes(Bruce)
Prevent sorting if result is already sorted(Jan)
New memory allocation optimization(Jan)
Allow psql to do \p\g(Bruce)
Allow multiple rule actions(Jan)
Added LIMIT/OFFSET functionality(Jan)
Improve optimizer when joining a large number of tables(Bruce)
New intro to SQL from S. Simkovics' Master's Thesis (Stefan,
 Thomas)
New intro to backend processing from S. Simkovics' Master's Thesis
 (Stefan)
Improved int8 support(Ryan Bradetich, Thomas, Tom)
New routines to convert between int8 and text/varchar types(Thomas)
New bushy plans, where meta-tables are joined(Bruce)
Enable right-hand queries by default(Bruce)
Allow reliable maximum number of backends to be set at configure
 time
     (--with-maxbackends and postmaster switch (-N backends))(Tom)
GEQO default now 10 tables because of optimizer speedups(Tom)
Allow NULL=Var for MS-SQL portability(Michael, Bruce)
Modify contrib check_primary_key() so either "automatic" or
 "dependent"(Anand)
Allow psql \d on a view show query(Ryan)
Speedup for LIKE(Bruce)
Ecpg fixes/features, see src/interfaces/ecpg/ChangeLog file(
Michael)
JDBC fixes/features, see src/interfaces/jdbc/CHANGELOG(Peter)
Make % operator have precedence like /(Bruce)
Add new postgres -O option to allow system table structure changes(
Bruce)
```

```
Update contrib/pginterface/findoidjoins script(Tom)
Major speedup in vacuum of deleted rows with indexes(Vadim)
Allow non-SQL functions to run different versions based on
 arguments(Tom)
Add -E option that shows actual queries sent by \dt and friends(
Masaaki Sakaida)
Add version number in start-up banners for psql(Masaaki Sakaida)
New contrib/vacuumlo removes large objects not referenced(Peter)
New initialization for table sizes so non-vacuumed tables perform
 better(Tom)
Improve error messages when a connection is rejected(Tom)
Support for arrays of char() and varchar() fields(Massimo)
Overhaul of hash code to increase reliability and performance(Tom)
Update to PyGreSQL 2.4(D'Arcy)
Changed debug options so -d4 and -d5 produce different node
 displays(Jan)
New pg_options: pretty_plan, pretty_parse, pretty_rewritten(Jan)
Better optimization statistics for system table access(Tom)
Better handling of non-default block sizes(Massimo)
Improve GEQO optimizer memory consumption(Tom)
UNION now supports ORDER BY of columns not in target list(Jan)
Major libpq++ improvements(Vince Vielhaber)
pg_dump now uses -z(ACL's) as default(Bruce)
backend cache, memory speedups(Tom)
have pg_dump do everything in one snapshot transaction(Vadim)
fix for large object memory leakage, fix for pg_dumping(Tom)
INET type now respects netmask for comparisons
Make VACUUM ANALYZE only use a readlock(Vadim)
Allow VIEWs on UNIONS(Jan)
pg_dump now can generate consistent snapshots on active databases(
Vadim)

Source Tree Changes
-------------------
Improve port matching(Tom)
Portability fixes for SunOS
Add Windows NT backend port and enable dynamic loading(Magnus and
 Daniel Horak)
New port to Cobalt Qube(Mips) running Linux(Tatsuo)
Port to NetBSD/m68k(Mr. Mutsuki Nakajima)
Port to NetBSD/sun3(Mr. Mutsuki Nakajima)
Port to NetBSD/macppc(Toshimi Aoki)
Fix for tcl/tk configuration(Vince)
Removed CURRENT key word for rule queries(Jan)
NT dynamic loading now works(Daniel Horak)
Add ARM32 support(Andrew McMurry)
Better support for HP-UX 11 and UnixWare
Improve file handling to be more uniform, prevent file descriptor
 leak(Tom)
New install commands for plpgsql(Jan)
```

E.271. Release 6.4.2

Release date: 1998-12-20

The 6.4.1 release was improperly packaged. This also has one additional bug fix.

E.271.1. Migration to Version 6.4.2

A dump/restore is not required for those running 6.4.*.

E.271.2. Changes

```
Fix for datetime constant problem on some platforms(Thomas)
```

E.272. Release 6.4.1

Release date: 1998-12-18

This is basically a cleanup release for 6.4. We have fixed a variety of problems reported by 6.4 users.

E.272.1. Migration to Version 6.4.1

A dump/restore is not required for those running 6.4.

E.272.2. Changes

```
Add pg_dump -N flag to force double quotes around identifiers.
 This is
       the default(Thomas)
Fix for NOT in where clause causing crash(Bruce)
EXPLAIN VERBOSE coredump fix(Vadim)
Fix shared-library problems on Linux
Fix test for table existence to allow mixed-case and whitespace in
       the table name(Thomas)
Fix a couple of pg_dump bugs
Configure matches template/.similar entries better(Tom)
Change builtin function names from SPI_* to spi_*
OR WHERE clause fix(Vadim)
Fixes for mixed-case table names(Billy)
contrib/linux/postgres.init.csh/sh fix(Thomas)
libpq memory overrun fix
SunOS fixes(Tom)
Change exp() behavior to generate error on underflow(Thomas)
pg_dump fixes for memory leak, inheritance constraints, layout
 change
update pgaccess to 0.93
Fix prototype for 64-bit platforms
Multibyte fixes(Tatsuo)
New ecpg man page
Fix memory overruns(Tatsuo)
Fix for lo_import() crash(Bruce)
Better search for install program(Tom)
Timezone fixes(Tom)
HP-UX fixes(Tom)
Use implicit type coercion for matching DEFAULT values(Thomas)
Add routines to help with single-byte (internal) character type(
Thomas)
Compilation of libpq for Windows fixes(Magnus)
Upgrade to PyGreSQL 2.2(D'Arcy)
```

E.273. Release 6.4

Release date: 1998-10-30

There are many new features and improvements in this release. Thanks to our developers and maintainers, nearly every aspect of the system has received some attention since the previous release. Here is a brief, incomplete summary:

- Views and rules are now functional thanks to extensive new code in the rewrite rules system from Jan Wieck. He also wrote a chapter on it for the Programmer's Guide.
- Jan also contributed a second procedural language, PL/pgSQL, to go with the original PL/pgTCL procedural language he contributed last release.
- We have optional multiple-byte character set support from Tatsuo Ishii to complement our existing locale support.
- Client/server communications has been cleaned up, with better support for asynchronous messages and interrupts thanks to Tom Lane.
- The parser will now perform automatic type coercion to match arguments to available operators and functions, and to match columns and expressions with target columns. This uses a generic mechanism which supports the type extensibility features of PostgreSQL. There is a new chapter in the User's Guide which covers this topic.
- Three new data types have been added. Two types, `inet` and `cidr`, support various forms of IP network, subnet, and machine addressing. There is now an 8-byte integer type available on some platforms. See the chapter on data types in the User's Guide for details. A fourth type, `serial`, is now supported by the parser as an amalgam of the `int4` type, a sequence, and a unique index.
- Several more SQL92-compatible syntax features have been added, including `INSERT DEFAULT VALUES`
- The automatic configuration and installation system has received some attention, and should be more robust for more platforms than it has ever been.

E.273.1. Migration to Version 6.4

A dump/restore using pg_dump or pg_dumpall is required for those wishing to migrate data from any previous release of PostgreSQL.

E.273.2. Changes

```
Bug Fixes
---------
Fix for a tiny memory leak in PQsetdb/PQfinish(Bryan)
Remove char2-16 data types, use char/varchar(Darren)
Pqfn not handles a NOTICE message(Anders)
Reduced busywaiting overhead for spinlocks with many backends (dg)
Stuck spinlock detection (dg)
Fix up "ISO-style" timespan decoding and encoding(Thomas)
Fix problem with table drop after rollback of transaction(Vadim)
Change error message and remove non-functional update message(
Vadim)
Fix for COPY array checking
Fix for SELECT 1 UNION SELECT NULL
Fix for buffer leaks in large object calls(Pascal)
Change owner from oid to int4 type(Bruce)
```

```
Fix a bug in the oracle compatibility functions btrim() ltrim() and
 rtrim()
Fix for shared invalidation cache overflow(Massimo)
Prevent file descriptor leaks in failed COPY's(Bruce)
Fix memory leak in libpgtcl's pg_select(Constantin)
Fix problems with username/passwords over 8 characters(Tom)
Fix problems with handling of asynchronous NOTIFY in backend(Tom)
Fix of many bad system table entries(Tom)

Enhancements
------------
Upgrade ecpg and ecpglib,see src/interfaces/ecpc/ChangeLog(Michael)
Show the index used in an EXPLAIN(Zeugswetter)
EXPLAIN  invokes  rule system and shows plan(s) for rewritten
 queries(Jan)
Multibyte awareness of many data types and functions, via
 configure(Tatsuo)
New configure --with-mb option(Tatsuo)
New initdb --pgencoding option(Tatsuo)
New createdb -E multibyte option(Tatsuo)
Select version(); now returns PostgreSQL version(Jeroen)
libpq now allows asynchronous clients(Tom)
Allow cancel from client of backend query(Tom)
psql now cancels query with Control-C(Tom)
libpq users need not issue dummy queries to get NOTIFY messages(
Tom)
NOTIFY now sends sender's PID, so you can tell whether it was your
 own(Tom)
PGresult struct now includes associated error message, if any(Tom)
Define "tz_hour" and "tz_minute" arguments to date_part()(Thomas)
Add routines to convert between varchar and bpchar(Thomas)
Add routines to allow sizing of varchar and bpchar into target
 columns(Thomas)
Add bit flags to support timezonehour and minute in data retrieval(
Thomas)
Allow more variations on valid floating point numbers (e.g. ".1",
 "1e6")(Thomas)
Fixes for unary minus parsing with leading spaces(Thomas)
Implement TIMEZONE_HOUR, TIMEZONE_MINUTE per SQL92 specs(Thomas)
Check for and properly ignore FOREIGN KEY column constraints(
Thomas)
Define USER as synonym for CURRENT_USER per SQL92 specs(Thomas)
Enable HAVING clause but no fixes elsewhere yet.
Make "char" type a synonym for "char(1)" (actually implemented as
 bpchar)(Thomas)
Save string type if specified for DEFAULT clause handling(Thomas)
Coerce operations involving different data types(Thomas)
Allow some index use for columns of different types(Thomas)
Add capabilities for automatic type conversion(Thomas)
Cleanups for large objects, so file is truncated on open(Peter)
Readline cleanups(Tom)
Allow psql  \f \ to make spaces as delimiter(Bruce)
Pass pg_attribute.atttypmod to the frontend for column field
 lengths(Tom,Bruce)
Msql compatibility library in /contrib(Aldrin)
Remove the requirement that ORDER/GROUP BY clause identifiers be
included in the target list(David)
Convert columns to match columns in UNION clauses(Thomas)
```

```
Remove fork()/exec() and only do fork()(Bruce)
Jdbc cleanups(Peter)
Show backend status on ps command line(only works on some
 platforms)(Bruce)
Pg_hba.conf now has a sameuser option in the database field
Make lo_unlink take oid param, not int4
New DISABLE_COMPLEX_MACRO for compilers that cannot handle our
 macros(Bruce)
Libpgtcl now handles NOTIFY as a Tcl event, need not send dummy
 queries(Tom)
libpgtcl cleanups(Tom)
Add -error option to libpgtcl's pg_result command(Tom)
New locale patch, see docs/README/locale(Oleg)
Fix for pg_dump so CONSTRAINT and CHECK syntax is correct(ccb)
New contrib/lo code for large object orphan removal(Peter)
New psql command "SET CLIENT_ENCODING TO 'encoding'" for multibytes
feature, see /doc/README.mb(Tatsuo)
contrib/noupdate code to revoke update permission on a column
libpq can now be compiled on Windows(Magnus)
Add PQsetdbLogin() in libpq
New 8-byte integer type, checked by configure for OS support(
Thomas)
Better support for quoted table/column names(Thomas)
Surround table and column names with double-quotes in pg_dump(
Thomas)
PQreset() now works with passwords(Tom)
Handle case of GROUP BY target list column number out of range(
David)
Allow UNION in subselects
Add auto-size to screen to \d? commands(Bruce)
Use UNION to show all \d? results in one query(Bruce)
Add \d? field search feature(Bruce)
Pg_dump issues fewer \connect requests(Tom)
Make pg_dump -z flag work better, document it in manual page(Tom)
Add HAVING clause with full support for subselects and unions(
Stephan)
Full text indexing routines in contrib/fulltextindex(Maarten)
Transaction ids now stored in shared memory(Vadim)
New PGCLIENTENCODING when issuing COPY command(Tatsuo)
Support for SQL92 syntax "SET NAMES"(Tatsuo)
Support for LATIN2-5(Tatsuo)
Add UNICODE regression test case(Tatsuo)
Lock manager cleanup, new locking modes for LLL(Vadim)
Allow index use with OR clauses(Bruce)
Allows "SELECT NULL ORDER BY 1;"
Explain VERBOSE prints the plan, and now pretty-prints the plan to
the postmaster log file(Bruce)
Add indexes display to \d command(Bruce)
Allow GROUP BY on functions(David)
New pg_class.relkind for large objects(Bruce)
New way to send libpq NOTICE messages to a different location(Tom)
New \w write command to psql(Bruce)
New /contrib/findoidjoins scans oid columns to find join
 relationships(Bruce)
Allow binary-compatible indexes to be considered when checking for
 valid
Indexes for restriction clauses containing a constant(Thomas)
New ISBN/ISSN code in /contrib/isbn_issn
```

```
Allow NOT LIKE, IN, NOT IN, BETWEEN, and NOT BETWEEN constraint(
Thomas)
New rewrite system fixes many problems with rules and views(Jan)
       * Rules on relations work
       * Event qualifications on insert/update/delete work
       * New OLD variable to reference CURRENT, CURRENT will be
 remove in future
       * Update rules can reference NEW and OLD in rule
 qualifications/actions
       * Insert/update/delete rules on views work
       * Multiple rule actions are now supported, surrounded by
 parentheses
       * Regular users can create views/rules on tables they have
 RULE permits
       * Rules and views inherit the privileges of the creator
       * No rules at the column level
       * No UPDATE NEW/OLD rules
       * New pg_tables, pg_indexes, pg_rules and pg_views system
 views
       * Only a single action on SELECT rules
       * Total rewrite overhaul, perhaps for 6.5
       * handle subselects
       * handle aggregates on views
       * handle insert into select from view works
System indexes are now multikey(Bruce)
Oidint2, oidint4, and oidname types are removed(Bruce)
Use system cache for more system table lookups(Bruce)
New backend programming language PL/pgSQL in backend/pl(Jan)
New SERIAL data type, auto-creates sequence/index(Thomas)
Enable assert checking without a recompile(Massimo)
User lock enhancements(Massimo)
New setval() command to set sequence value(Massimo)
Auto-remove unix socket file on start-up if no postmaster running(
Massimo)
Conditional trace package(Massimo)
New UNLISTEN command(Massimo)
psql and libpq now compile under Windows using win32.mak(Magnus)
Lo_read no longer stores trailing NULL(Bruce)
Identifiers are now truncated to 31 characters internally(Bruce)
Createuser options now available on the command line
Code for 64-bit integer supported added, configure tested, int8
 type(Thomas)
Prevent file descriptor leaf from failed COPY(Bruce)
New pg_upgrade command(Bruce)
Updated /contrib directories(Massimo)
New CREATE TABLE DEFAULT VALUES statement available(Thomas)
New INSERT INTO TABLE DEFAULT VALUES statement available(Thomas)
New DECLARE and FETCH feature(Thomas)
libpq's internal structures now not exported(Tom)
Allow up to 8 key indexes(Bruce)
Remove ARCHIVE key word, that is no longer used(Thomas)
pg_dump -n flag to suppress quotes around identifiers
disable system columns for views(Jan)
new INET and CIDR types for network addresses(TomH, Paul)
no more double quotes in psql output
pg_dump now dumps views(Terry)
new SET QUERY_LIMIT(Tatsuo,Jan)
```

```
Source Tree Changes
-------------------
/contrib cleanup(Jun)
Inline some small functions called for every row(Bruce)
Alpha/linux fixes
HP-UX cleanups(Tom)
Multibyte regression tests(Soonmyung.)
Remove --disabled options from configure
Define PGDOC to use POSTGRESDIR by default
Make regression optional
Remove extra braces code to pgindent(Bruce)
Add bsdi shared library support(Bruce)
New --without-CXX support configure option(Brook)
New FAQ_CVS
Update backend flowchart in tools/backend(Bruce)
Change atttypmod from int16 to int32(Bruce, Tom)
Getrusage() fix for platforms that do not have it(Tom)
Add PQconnectdb, PGUSER, PGPASSWORD to libpq man page
NS32K platform fixes(Phil Nelson, John Buller)
SCO 7/UnixWare 2.x fixes(Billy,others)
Sparc/Solaris 2.5 fixes(Ryan)
Pgbuiltin.3 is obsolete, move to doc files(Thomas)
Even more documentation(Thomas)
Nextstep support(Jacek)
Aix support(David)
pginterface manual page(Bruce)
shared libraries all have version numbers
merged all OS-specific shared library defines into one file
smarter TCL/TK configuration checking(Billy)
smarter perl configuration(Brook)
configure uses supplied install-sh if no install script found(Tom)
new Makefile.shlib for shared library configuration(Tom)
```

E.274. Release 6.3.2

Release date: 1998-04-07

This is a bug-fix release for 6.3.x. Refer to the release notes for version 6.3 for a more complete summary of new features.

Summary:

- Repairs automatic configuration support for some platforms, including Linux, from breakage inadvertently introduced in version 6.3.1.
- Correctly handles function calls on the left side of BETWEEN and LIKE clauses.

A dump/restore is NOT required for those running 6.3 or 6.3.1. A `make distclean`, `make`, and `make install` is all that is required. This last step should be performed while the postmaster is not running. You should re-link any custom applications that use PostgreSQL libraries.

For upgrades from pre-6.3 installations, refer to the installation and migration instructions for version 6.3.

E.274.1. Changes

```
Configure detection improvements for tcl/tk(Brook Milligan, Alvin)
Manual page improvements(Bruce)
```

```
BETWEEN and LIKE fix(Thomas)
fix for psql \connect used by pg_dump(Oliver Elphick)
New odbc driver
pgaccess, version 0.86
qsort removed, now uses libc version, cleanups(Jeroen)
fix for buffer over-runs detected(Maurice Gittens)
fix for buffer overrun in libpgtcl(Randy Kunkee)
fix for UNION with DISTINCT or ORDER BY(Bruce)
gettimeofday configure check(Doug Winterburn)
Fix "indexes not used" bug(Vadim)
docs additions(Thomas)
Fix for backend memory leak(Bruce)
libreadline cleanup(Erwan MAS)
Remove DISTDIR(Bruce)
Makefile dependency cleanup(Jeroen van Vianen)
ASSERT fixes(Bruce)
```

E.275. Release 6.3.1

Release date: 1998-03-23

Summary:

- Additional support for multibyte character sets.
- Repair byte ordering for mixed-endian clients and servers.
- Minor updates to allowed SQL syntax.
- Improvements to the configuration autodetection for installation.

A dump/restore is NOT required for those running 6.3. A `make distclean`, `make`, and `make install` is all that is required. This last step should be performed while the postmaster is not running. You should re-link any custom applications that use PostgreSQL libraries.

For upgrades from pre-6.3 installations, refer to the installation and migration instructions for version 6.3.

E.275.1. Changes

```
ecpg cleanup/fixes, now version 1.1(Michael Meskes)
pg_user cleanup(Bruce)
large object fix for pg_dump and tclsh (alvin)
LIKE fix for multiple adjacent underscores
fix for redefining builtin functions(Thomas)
ultrix4 cleanup
upgrade to pg_access 0.83
updated CLUSTER manual page
multibyte character set support, see doc/README.mb(Tatsuo)
configure --with-pgport fix
pg_ident fix
big-endian fix for backend communications(Kataoka)
SUBSTR() and substring() fix(Jan)
several jdbc fixes(Peter)
libpgtcl improvements, see libptcl/README(Randy Kunkee)
Fix for "Datasize = 0" error(Vadim)
Prevent \do from wrapping(Bruce)
Remove duplicate Russian character set entries
```

```
Sunos4 cleanup
Allow optional TABLE key word in LOCK and SELECT INTO(Thomas)
CREATE SEQUENCE options to allow a negative integer(Thomas)
Add "PASSWORD" as an allowed column identifier(Thomas)
Add checks for UNION target fields(Bruce)
Fix Alpha port(Dwayne Bailey)
Fix for text arrays containing quotes(Doug Gibson)
Solaris compile fix(Albert Chin-A-Young)
Better identify tcl and tk libs and includes(Bruce)
```

E.276. Release 6.3

Release date: 1998-03-01

There are many new features and improvements in this release. Here is a brief, incomplete summary:

- Many new SQL features, including full SQL92 subselect capability (everything is here but target-list subselects).
- Support for client-side environment variables to specify time zone and date style.
- Socket interface for client/server connection. This is the default now so you might need to start postmaster with the `-i` flag.
- Better password authorization mechanisms. Default table privileges have changed.
- Old-style time travel has been removed. Performance has been improved.

> **注意**
>
> Bruce Momjian wrote the following notes to introduce the new release.

There are some general 6.3 issues that I want to mention. These are only the big items that cannot be described in one sentence. A review of the detailed changes list is still needed.

First, we now have subselects. Now that we have them, I would like to mention that without subselects, SQL is a very limited language. Subselects are a major feature, and you should review your code for places where subselects provide a better solution for your queries. I think you will find that there are more uses for subselects than you might think. Vadim has put us on the big SQL map with subselects, and fully functional ones too. The only thing you cannot do with subselects is to use them in the target list.

Second, 6.3 uses Unix domain sockets rather than TCP/IP by default. To enable connections from other machines, you have to use the new postmaster -i option, and of course edit `pg_hba.conf`. Also, for this reason, the format of `pg_hba.conf` has changed.

Third, `char()` fields will now allow faster access than `varchar()` or `text`. Specifically, the `text` and `varchar()` have a penalty for access to any columns after the first column of this type. `char()` used to also have this access penalty, but it no longer does. This might suggest that you redesign some of your tables, especially if you have short character columns that you have defined as `varchar()` or `text`. This and other changes make 6.3 even faster than earlier releases.

We now have passwords definable independent of any Unix file. There are new SQL USER commands. See the Administrator's Guide for more information. There is a new

table, pg_shadow, which is used to store user information and user passwords, and it by default only SELECT-able by the postgres super-user. pg_user is now a view of pg_shadow, and is SELECT-able by PUBLIC. You should keep using pg_user in your application without changes.

User-created tables now no longer have SELECT privilege to PUBLIC by default. This was done because the ANSI standard requires it. You can of course GRANT any privileges you want after the table is created. System tables continue to be SELECT-able by PUBLIC.

We also have real deadlock detection code. No more sixty-second timeouts. And the new locking code implements a FIFO better, so there should be less resource starvation during heavy use.

Many complaints have been made about inadequate documentation in previous releases. Thomas has put much effort into many new manuals for this release. Check out the doc/ directory.

For performance reasons, time travel is gone, but can be implemented using triggers (see `pgsql/contrib/spi/README`). Please check out the new \d command for types, operators, etc. Also, views have their own privileges now, not based on the underlying tables, so privileges on them have to be set separately. Check `/pgsql/interfaces` for some new ways to talk to PostgreSQL.

This is the first release that really required an explanation for existing users. In many ways, this was necessary because the new release removes many limitations, and the work-arounds people were using are no longer needed.

E.276.1. Migration to Version 6.3

A dump/restore using pg_dump or pg_dumpall is required for those wishing to migrate data from any previous release of PostgreSQL.

E.276.2. Changes

```
Bug Fixes
---------
Fix binary cursors broken by MOVE implementation(Vadim)
Fix for tcl library crash(Jan)
Fix for array handling, from Gerhard Hintermayer
Fix acl error, and remove duplicate pqtrace(Bruce)
Fix psql \e for empty file(Bruce)
Fix for textcat on varchar() fields(Bruce)
Fix for DBT Sendproc (Zeugswetter Andres)
Fix vacuum analyze syntax problem(Bruce)
Fix for international identifiers(Tatsuo)
Fix aggregates on inherited tables(Bruce)
Fix substr() for out-of-bounds data
Fix for select 1=1 or 2=2, select 1=1 and 2=2, and select sum(2+2)(
Bruce)
Fix notty output to show status result.  -q option still turns it
 off(Bruce)
Fix for count(*), aggs with views and multiple tables and sum(3)(
Bruce)
Fix cluster(Bruce)
Fix for PQtrace start/stop several times(Bruce)
Fix a variety of locking problems like newer lock waiters getting
       lock before older waiters, and having readlock people not
 share
```

```
        locks if a writer is waiting for a lock, and waiting writers
 not
        getting priority over waiting readers(Bruce)
Fix crashes in psql when executing queries from external files(
James)
Fix problem with multiple order by columns, with the first one
 having
        NULL values(Jeroen)
Use correct hash table support functions for float8 and int4(
Thomas)
Re-enable JOIN= option in CREATE OPERATOR statement (Thomas)
Change precedence for boolean operators to match expected behavior(
Thomas)
Generate elog(ERROR) on over-large integer(Bruce)
Allow multiple-argument functions in constraint clauses(Thomas)
Check boolean input literals for 'true','false','yes','no','1','0'
        and throw elog(ERROR) if unrecognized(Thomas)
Major large objects fix
Fix for GROUP BY showing duplicates(Vadim)
Fix for index scans in MergeJoin(Vadim)

Enhancements
------------
Subselects with EXISTS, IN, ALL, ANY key words (Vadim, Bruce,
 Thomas)
New User Manual(Thomas, others)
Speedup by inlining some frequently-called functions
Real deadlock detection, no more timeouts(Bruce)
Add SQL92 "constants" CURRENT_DATE, CURRENT_TIME, CURRENT_
TIMESTAMP,
        CURRENT_USER(Thomas)
Modify constraint syntax to be SQL92-compliant(Thomas)
Implement SQL92 PRIMARY KEY and UNIQUE clauses using indexes(
Thomas)
Recognize SQL92 syntax for FOREIGN KEY. Throw elog notice(Thomas)
Allow NOT NULL UNIQUE constraint clause (each allowed separately
 before)(Thomas)
Allow PostgreSQL-style casting ("::") of non-constants(Thomas)
Add support for SQL3 TRUE and FALSE boolean constants(Thomas)
Support SQL92 syntax for IS TRUE/IS FALSE/IS NOT TRUE/IS NOT FALSE(
Thomas)
Allow shorter strings for boolean literals (e.g. "t", "tr", "tru")(
Thomas)
Allow SQL92 delimited identifiers(Thomas)
Implement SQL92 binary and hexadecimal string decoding (b'10' and
 x'1F')(Thomas)
Support SQL92 syntax for type coercion of literal strings
        (e.g. "DATETIME 'now'")(Thomas)
Add conversions for int2, int4, and OID types to and from text(
Thomas)
Use shared lock when building indexes(Vadim)
Free memory allocated for a user query inside transaction block
 after
        this query is done, was turned off in <= 6.2.1(Vadim)
New SQL statement CREATE PROCEDURAL LANGUAGE(Jan)
New PostgreSQL Procedural Language (PL) backend interface(Jan)
Rename pg_dump -H option to -h(Bruce)
Add Java support for passwords, European dates(Peter)
```

```
Use indexes for LIKE and ~, !~ operations(Bruce)
Add hash functions for datetime and timespan(Thomas)
Time Travel removed(Vadim, Bruce)
Add paging for \d and \z, and fix \i(Bruce)
Add Unix domain socket support to backend and to frontend library(
Goran)
Implement CREATE DATABASE/WITH LOCATION and initlocation utility(
Thomas)
Allow more SQL92 and/or PostgreSQL reserved words as column
 identifiers(Thomas)
Augment support for SQL92 SET TIME ZONE...(Thomas)
SET/SHOW/RESET TIME ZONE uses TZ backend environment variable(
Thomas)
Implement SET keyword = DEFAULT and SET TIME ZONE DEFAULT(Thomas)
Enable SET TIME ZONE using TZ environment variable(Thomas)
Add PGDATESTYLE environment variable to frontend and backend
 initialization(Thomas)
Add PGTZ, PGCOSTHEAP, PGCOSTINDEX, PGRPLANS, PGGEQO
       frontend library initialization environment variables(
Thomas)
Regression tests time zone automatically set with "setenv PGTZ
 PST8PDT"(Thomas)
Add pg_description table for info on tables, columns, operators,
 types, and
       aggregates(Bruce)
Increase 16 char limit on system table/index names to 32
 characters(Bruce)
Rename system indexes(Bruce)
Add 'GERMAN' option to SET DATESTYLE(Thomas)
Define an "ISO-style" timespan output format with "hh:mm:ss"
 fields(Thomas)
Allow fractional values for delta times (e.g. '2.5 days')(Thomas)
Validate numeric input more carefully for delta times(Thomas)
Implement day of year as possible input to date_part()(Thomas)
Define timespan_finite() and text_timespan() functions(Thomas)
Remove archive stuff(Bruce)
Allow for a pg_password authentication database that is separate
 from
       the system password file(Todd)
Dump ACLs, GRANT, REVOKE privileges(Matt)
Define text, varchar, and bpchar string length functions(Thomas)
Fix Query handling for inheritance, and cost computations(Bruce)
Implement CREATE TABLE/AS SELECT (alternative to SELECT/INTO)(
Thomas)
Allow NOT, IS NULL, IS NOT NULL in constraints(Thomas)
Implement UNIONs for SELECT(Bruce)
Add UNION, GROUP, DISTINCT to INSERT(Bruce)
varchar() stores only necessary bytes on disk(Bruce)
Fix for BLOBs(Peter)
Mega-Patch for JDBC...see README_6.3 for list of changes(Peter)
Remove unused "option" from PQconnectdb()
New LOCK command and lock manual page describing deadlocks(Bruce)
Add new psql \da, \dd, \df, \do, \dS, and \dT commands(Bruce)
Enhance psql \z to show sequences(Bruce)
Show NOT NULL and DEFAULT in psql \d table(Bruce)
New psql .psqlrc file start-up(Andrew)
Modify sample start-up script in contrib/linux to show syslog(
Thomas)
```

```
New types for IP and MAC addresses in contrib/ip_and_mac(TomH)
Unix system time conversions with date/time types in contrib/
unixdate(Thomas)
Update of contrib stuff(Massimo)
Add Unix socket support to DBD::Pg(Goran)
New python interface (PyGreSQL 2.0)(D'Arcy)
New frontend/backend protocol has a version number, network byte
 order(Phil)
Security features in pg_hba.conf enhanced and documented, many
 cleanups(Phil)
CHAR() now faster access than VARCHAR() or TEXT
ecpg embedded SQL preprocessor
Reduce system column overhead(Vadmin)
Remove pg_time table(Vadim)
Add pg_type attribute to identify types that need length (bpchar,
 varchar)
Add report of offending line when COPY command fails
Allow VIEW privileges to be set separately from the underlying
 tables.
       For security, use GRANT/REVOKE on views as appropriate(Jan)
Tables now have no default GRANT SELECT TO PUBLIC.  You must
       explicitly grant such privileges.
Clean up tutorial examples(Darren)

Source Tree Changes
-------------------
Add new html development tools, and flow chart in /tools/backend
Fix for SCO compiles
Stratus computer port Robert Gillies
Added support for shlib for BSD44_derived & i386_solaris
Make configure more automated(Brook)
Add script to check regression test results
Break parser functions into smaller files, group together(Bruce)
Rename heap_create to heap_create_and_catalog, rename heap_creatr
       to heap_create()(Bruce)
Sparc/Linux patch for locking(TomS)
Remove PORTNAME and reorganize port-specific stuff(Marc)
Add optimizer README file(Bruce)
Remove some recursion in optimizer and clean up some code there(
Bruce)
Fix for NetBSD locking(Henry)
Fix for libptcl make(Tatsuo)
AIX patch(Darren)
Change IS TRUE, IS FALSE, ... to expressions using "=" rather than
       function calls to istrue() or isfalse() to allow
 optimization(Thomas)
Various fixes NetBSD/Sparc related(TomH)
Alpha linux locking(Travis,Ryan)
Change elog(WARN) to elog(ERROR)(Bruce)
FAQ for FreeBSD(Marc)
Bring in the PostODBC source tree as part of our standard
 distribution(Marc)
A minor patch for HP/UX 10 vs 9(Stan)
New pg_attribute.atttypmod for type-specific info like varchar
 length(Bruce)
UnixWare patches(Billy)
New i386 'lock' for spinlock asm(Billy)
Support for multiplexed backends is removed
```

```
Start an OpenBSD port
Start an AUX port
Start a Cygnus port
Add string functions to regression suite(Thomas)
Expand a few function names formerly truncated to 16 characters(
Thomas)
Remove un-needed malloc() calls and replace with palloc()(Bruce)
```

E.277. Release 6.2.1

Release date: 1997-10-17

6.2.1 is a bug-fix and usability release on 6.2.

Summary:

- Allow strings to span lines, per SQL92.
- Include example trigger function for inserting user names on table updates.

This is a minor bug-fix release on 6.2. For upgrades from pre-6.2 systems, a full dump/reload is required. Refer to the 6.2 release notes for instructions.

E.277.1. Migration from version 6.2 to version 6.2.1

This is a minor bug-fix release. A dump/reload is not required from version 6.2, but is required from any release prior to 6.2.

In upgrading from version 6.2, if you choose to dump/reload you will find that avg(money) is now calculated correctly. All other bug fixes take effect upon updating the executables.

Another way to avoid dump/reload is to use the following SQL command from `psql` to update the existing system table:

```
update pg_aggregate set aggfinalfn = 'cash_div_flt8'
 where aggname = 'avg' and aggbasetype = 790;
```

This will need to be done to every existing database, including template1.

E.277.2. Changes

```
Allow TIME and TYPE column names(Thomas)
Allow larger range of true/false as boolean values(Thomas)
Support output of "now" and "current"(Thomas)
Handle DEFAULT with INSERT of NULL properly(Vadim)
Fix for relation reference counts problem in buffer manager(Vadim)
Allow strings to span lines, like ANSI(Thomas)
Fix for backward cursor with ORDER BY(Vadim)
Fix avg(cash) computation(Thomas)
Fix for specifying a column twice in ORDER/GROUP BY(Vadim)
Documented new libpq function to return affected rows, PQcmdTuples(
Bruce)
Trigger function for inserting user names for INSERT/UPDATE(Brook
 Milligan)
```

E.278. Release 6.2

Release date: 1997-10-02

A dump/restore is required for those wishing to migrate data from previous releases of PostgreSQL.

E.278.1. Migration from version 6.1 to version 6.2

This migration requires a complete dump of the 6.1 database and a restore of the database in 6.2.

Note that the `pg_dump` and `pg_dumpall` utility from 6.2 should be used to dump the 6.1 database.

E.278.2. Migration from version 1.*x* to version 6.2

Those migrating from earlier 1.* releases should first upgrade to 1.09 because the COPY output format was improved from the 1.02 release.

E.278.3. Changes

```
Bug Fixes
---------
Fix problems with pg_dump for inheritance, sequences, archive
 tables(Bruce)
Fix compile errors on overflow due to shifts, unsigned, and bad
 prototypes
        from Solaris(Diab Jerius)
Fix bugs in geometric line arithmetic (bad intersection
 calculations)(Thomas)
Check for geometric intersections at endpoints to avoid rounding
 ugliness(Thomas)
Catch non-functional delete attempts(Vadim)
Change time function names to be more consistent(Michael
 Reifenberg)
Check for zero divides(Michael Reifenberg)
Fix very old bug which made rows changed/inserted by a command
       visible to the command itself (so we had multiple update of
       updated rows, etc.)(Vadim)
Fix for SELECT null, 'fail' FROM pg_am (Patrick)
SELECT NULL as EMPTY_FIELD now allowed(Patrick)
Remove un-needed signal stuff from contrib/pginterface
Fix OR (where x != 1 or x isnull didn't return rows with x NULL) (
Vadim)
Fix time_cmp function (Vadim)
Fix handling of functions with non-attribute first argument in
       WHERE clauses (Vadim)
Fix GROUP BY when order of entries is different from order
       in target list (Vadim)
Fix pg_dump for aggregates without sfunc1 (Vadim)

Enhancements
------------
Default genetic optimizer GEQO parameter is now 8(Bruce)
Allow use parameters in target list having aggregates in functions(
Vadim)
Added JDBC driver as an interface(Adrian & Peter)
pg_password utility
Return number of rows inserted/affected by INSERT/UPDATE/DELETE
 etc.(Vadim)
Triggers implemented with CREATE TRIGGER (SQL3)(Vadim)
```

```
SPI (Server Programming Interface) allows execution of queries
 inside
       C-functions (Vadim)
NOT NULL implemented (SQL92)(Robson Paniago de Miranda)
Include reserved words for string handling, outer joins, and
 unions(Thomas)
Implement extended comments ("/* ... */") using exclusive states(
Thomas)
Add "//" single-line comments(Bruce)
Remove some restrictions on characters in operator names(Thomas)
DEFAULT and CONSTRAINT for tables implemented (SQL92)(Vadim &
 Thomas)
Add text concatenation operator and function (SQL92)(Thomas)
Support WITH TIME ZONE syntax (SQL92)(Thomas)
Support INTERVAL unit TO unit syntax (SQL92)(Thomas)
Define types DOUBLE PRECISION, INTERVAL, CHARACTER,
       and CHARACTER VARYING (SQL92)(Thomas)
Define type FLOAT(p) and rudimentary DECIMAL(p,s), NUMERIC(p,s) (
SQL92)(Thomas)
Define EXTRACT(), POSITION(), SUBSTRING(), and TRIM() (SQL92)(
Thomas)
Define CURRENT_DATE, CURRENT_TIME, CURRENT_TIMESTAMP (SQL92)(
Thomas)
Add syntax and warnings for UNION, HAVING, INNER and OUTER JOIN (
SQL92)(Thomas)
Add more reserved words, mostly for SQL92 compliance(Thomas)
Allow hh:mm:ss time entry for timespan/reltime types(Thomas)
Add center() routines for lseg, path, polygon(Thomas)
Add distance() routines for circle-polygon, polygon-polygon(Thomas)
Check explicitly for points and polygons contained within polygons
       using an axis-crossing algorithm(Thomas)
Add routine to convert circle-box(Thomas)
Merge conflicting operators for different geometric data types(
Thomas)
Replace distance operator "<===>" with "<->"(Thomas)
Replace "above" operator "!^" with ">^" and "below" operator "!|"
 with "<^"(Thomas)
Add routines for text trimming on both ends, substring, and string
 position(Thomas)
Added conversion routines circle(box) and poly(circle)(Thomas)
Allow internal sorts to be stored in memory rather than in files(
Bruce & Vadim)
Allow functions and operators on internally-identical types to
 succeed(Bruce)
Speed up backend start-up after profiling analysis(Bruce)
Inline frequently called functions for performance(Bruce)
Reduce open() calls(Bruce)
psql:  Add PAGER for \h and \?,\C fix
Fix for psql pager when no tty(Bruce)
New entab utility(Bruce)
General trigger functions for referential integrity (Vadim)
General trigger functions for time travel (Vadim)
General trigger functions for AUTOINCREMENT/IDENTITY feature (
Vadim)
MOVE implementation (Vadim)

Source Tree Changes
-------------------
```

```
HP-UX 10 patches (Vladimir Turin)
Added SCO support, (Daniel Harris)
MkLinux patches (Tatsuo Ishii)
Change geometric box terminology from "length" to "width"(Thomas)
Deprecate temporary unstored slope fields in geometric code(Thomas)
Remove restart instructions from INSTALL(Bruce)
Look in /usr/ucb first for install(Bruce)
Fix c++ copy example code(Thomas)
Add -o to psql manual page(Bruce)
Prevent relname unallocated string length from being copied into
 database(Bruce)
Cleanup for NAMEDATALEN use(Bruce)
Fix pg_proc names over 15 chars in output(Bruce)
Add strNcpy() function(Bruce)
remove some (void) casts that are unnecessary(Bruce)
new interfaces directory(Marc)
Replace fopen() calls with calls to fd.c functions(Bruce)
Make functions static where possible(Bruce)
enclose unused functions in #ifdef NOT_USED(Bruce)
Remove call to difftime() in timestamp support to fix SunOS(Bruce &
 Thomas)
Changes for Digital Unix
Portability fix for pg_dumpall(Bruce)
Rename pg_attribute.attnvals to attdispersion(Bruce)
"intro/unix" manual page now "pgintro"(Bruce)
"built-in" manual page now "pgbuiltin"(Bruce)
"drop" manual page now "drop_table"(Bruce)
Add "create_trigger", "drop_trigger" manual pages(Thomas)
Add constraints regression test(Vadim & Thomas)
Add comments syntax regression test(Thomas)
Add PGINDENT and support program(Bruce)
Massive commit to run PGINDENT on all *.c and *.h files(Bruce)
Files moved to /src/tools directory(Bruce)
SPI and Trigger programming guides (Vadim & D'Arcy)
```

E.279. Release 6.1.1

Release date: 1997-07-22

E.279.1. Migration from version 6.1 to version 6.1.1

This is a minor bug-fix release. A dump/reload is not required from version 6.1, but is required from any release prior to 6.1. Refer to the release notes for 6.1 for more details.

E.279.2. Changes

```
fix for SET with options (Thomas)
allow pg_dump/pg_dumpall to preserve ownership of all tables/
objects(Bruce)
new psql \connect option allows changing usernames without changing
 databases
fix for initdb --debug option(Yoshihiko Ichikawa))
lextest cleanup(Bruce)
hash fixes(Vadim)
fix date/time month boundary arithmetic(Thomas)
```

```
fix timezone daylight handling for some ports(Thomas, Bruce,
 Tatsuo)
timestamp overhauled to use standard functions(Thomas)
other code cleanup in date/time routines(Thomas)
psql's \d now case-insensitive(Bruce)
psql's backslash commands can now have trailing semicolon(Bruce)
fix memory leak in psql when using \g(Bruce)
major fix for endian handling of communication to server(Thomas,
 Tatsuo)
Fix for Solaris assembler and include files(Yoshihiko Ichikawa)
allow underscores in usernames(Bruce)
pg_dumpall now returns proper status, portability fix(Bruce)
```

E.280. Release 6.1

Release date: 1997-06-08

The regression tests have been adapted and extensively modified for the 6.1 release of PostgreSQL.

Three new data types (`datetime`, `timespan`, and `circle`) have been added to the native set of PostgreSQL types. Points, boxes, paths, and polygons have had their output formats made consistent across the data types. The polygon output in misc.out has only been spot-checked for correctness relative to the original regression output.

PostgreSQL 6.1 introduces a new, alternate optimizer which uses genetic algorithms. These algorithms introduce a random behavior in the ordering of query results when the query contains multiple qualifiers or multiple tables (giving the optimizer a choice on order of evaluation). Several regression tests have been modified to explicitly order the results, and hence are insensitive to optimizer choices. A few regression tests are for data types which are inherently unordered (e.g. points and time intervals) and tests involving those types are explicitly bracketed with `set geqo to 'off'` and `reset geqo`.

The interpretation of array specifiers (the curly braces around atomic values) appears to have changed sometime after the original regression tests were generated. The current `./expected/*.out` files reflect this new interpretation, which might not be correct!

The float8 regression test fails on at least some platforms. This is due to differences in implementations of `pow()` and `exp()` and the signaling mechanisms used for overflow and underflow conditions.

The “random” results in the random test should cause the “random” test to be “failed”, since the regression tests are evaluated using a simple diff. However, “random” does not seem to produce random results on my test machine (Linux/gcc/i686).

E.280.1. Migration to Version 6.1

This migration requires a complete dump of the 6.0 database and a restore of the database in 6.1.

Those migrating from earlier 1.* releases should first upgrade to 1.09 because the COPY output format was improved from the 1.02 release.

E.280.2. Changes

```
Bug Fixes
```

```
---------
packet length checking in library routines
lock manager priority patch
check for under/over flow of float8(Bruce)
multitable join fix(Vadim)
SIGPIPE crash fix(Darren)
large object fixes(Sven)
allow btree indexes to handle NULLs(Vadim)
timezone fixes(D'Arcy)
select SUM(x) can return NULL on no rows(Thomas)
internal optimizer, executor bug fixes(Vadim)
fix problem where inner loop in < or <= has no rows(Vadim)
prevent re-commuting join index clauses(Vadim)
fix join clauses for multiple tables(Vadim)
fix hash, hashjoin for arrays(Vadim)
fix btree for abstime type(Vadim)
large object fixes(Raymond)
fix buffer leak in hash indexes (Vadim)
fix rtree for use in inner scan (Vadim)
fix gist for use in inner scan, cleanups (Vadim, Andrea)
avoid unnecessary local buffers allocation (Vadim, Massimo)
fix local buffers leak in transaction aborts (Vadim)
fix file manager memory leaks, cleanups (Vadim, Massimo)
fix storage manager memory leaks (Vadim)
fix btree duplicates handling (Vadim)
fix deleted rows reincarnation caused by vacuum (Vadim)
fix SELECT varchar()/char() INTO TABLE made zero-length fields(
Bruce)
many psql, pg_dump, and libpq memory leaks fixed using Purify (
Igor)

Enhancements
------------
attribute optimization statistics(Bruce)
much faster new btree bulk load code(Paul)
BTREE UNIQUE added to bulk load code(Vadim)
new lock debug code(Massimo)
massive changes to libpg++(Leo)
new GEQO optimizer speeds table multitable optimization(Martin)
new WARN message for non-unique insert into unique key(Marc)
update x=-3, no spaces, now valid(Bruce)
remove case-sensitive identifier handling(Bruce,Thomas,Dan)
debug backend now pretty-prints tree(Darren)
new Oracle character functions(Edmund)
new plaintext password functions(Dan)
no such class or insufficient privilege changed to distinct
 messages(Dan)
new ANSI timestamp function(Dan)
new ANSI Time and Date types (Thomas)
move large chunks of data in backend(Martin)
multicolumn btree indexes(Vadim)
new SET var TO value command(Martin)
update transaction status on reads(Dan)
new locale settings for character types(Oleg)
new SEQUENCE serial number generator(Vadim)
GROUP BY function now possible(Vadim)
re-organize regression test(Thomas,Marc)
new optimizer operation weights(Vadim)
```

```
new psql \z grant/permit option(Marc)
new MONEY data type(D'Arcy,Thomas)
tcp socket communication speed improved(Vadim)
new VACUUM option for attribute statistics, and for certain columns
 (Vadim)
many geometric type improvements(Thomas,Keith)
additional regression tests(Thomas)
new datestyle variable(Thomas,Vadim,Martin)
more comparison operators for sorting types(Thomas)
new conversion functions(Thomas)
new more compact btree format(Vadim)
allow pg_dumpall to preserve database ownership(Bruce)
new SET GEQO=# and R_PLANS variable(Vadim)
old (!GEQO) optimizer can use right-sided plans (Vadim)
typechecking improvement in SQL parser(Bruce)
new SET, SHOW, RESET commands(Thomas,Vadim)
new \connect database USER option
new destroydb -i option (Igor)
new \dt and \di psql commands (Darren)
SELECT "\n" now escapes newline (A. Duursma)
new geometry conversion functions from old format (Thomas)

Source tree changes
-------------------
new configuration script(Marc)
readline configuration option added(Marc)
OS-specific configuration options removed(Marc)
new OS-specific template files(Marc)
no more need to edit Makefile.global(Marc)
re-arrange include files(Marc)
nextstep patches (Gregor Hoffleit)
removed Windows-specific code(Bruce)
removed postmaster -e option, now only postgres -e option (Bruce)
merge duplicate library code in front/backends(Martin)
now works with eBones, international Kerberos(Jun)
more shared library support
c++ include file cleanup(Bruce)
warn about buggy flex(Bruce)
DG/UX, Ultrix, IRIX, AIX portability fixes
```

E.281. Release 6.0

Release date: 1997-01-29

A dump/restore is required for those wishing to migrate data from previous releases of PostgreSQL.

E.281.1. Migration from version 1.09 to version 6.0

This migration requires a complete dump of the 1.09 database and a restore of the database in 6.0.

E.281.2. Migration from pre-1.09 to version 6.0

Those migrating from earlier 1.* releases should first upgrade to 1.09 because the COPY output format was improved from the 1.02 release.

E.281.3. Changes

```
Bug Fixes
---------
ALTER TABLE bug - running postgres process needs to re-read table
 definition
Allow vacuum to be run on one table or entire database(Bruce)
Array fixes
Fix array over-runs of memory writes(Kurt)
Fix elusive btree range/non-range bug(Dan)
Fix for hash indexes on some types like time and date
Fix for pg_log size explosion
Fix permissions on lo_export()(Bruce)
Fix uninitialized reads of memory(Kurt)
Fixed ALTER TABLE ... char(3) bug(Bruce)
Fixed a few small memory leaks
Fixed EXPLAIN handling of options and changed full_path option name
Fixed output of group acl privileges
Memory leaks (hunt and destroy with tools like Purify(Kurt)
Minor improvements to rules system
NOTIFY fixes
New asserts for run-checking
Overhauled parser/analyze code to properly report errors and
 increase speed
Pg_dump -d now handles NULL's properly(Bruce)
Prevent SELECT NULL from crashing server (Bruce)
Properly report errors when INSERT ... SELECT columns did not match
Properly report errors when insert column names were not correct
psql \g filename now works(Bruce)
psql fixed problem with multiple statements on one line with
 multiple outputs
Removed duplicate system OIDs
SELECT * INTO TABLE . GROUP/ORDER BY gives unlink error if table
 exists(Bruce)
Several fixes for queries that crashed the backend
Starting quote in insert string errors(Bruce)
Submitting an empty query now returns empty status, not just " "
 query(Bruce)

Enhancements
------------
Add EXPLAIN manual page(Bruce)
Add UNIQUE index capability(Dan)
Add hostname/user level access control rather than just hostname
 and user
Add synonym of != for <>(Bruce)
Allow "select oid,* from table"
Allow BY,ORDER BY to specify columns by number, or by non-alias
 table.column(Bruce)
Allow COPY from the frontend(Bryan)
Allow GROUP BY to use alias column name(Bruce)
Allow actual compression, not just reuse on the same page(Vadim)
Allow installation-configuration option to auto-add all local
 users(Bryan)
Allow libpq to distinguish between text value '' and null(Bruce)
Allow non-postgres users with createdb privs to destroydb's
Allow restriction on who can create C functions(Bryan)
```

```
Allow restriction on who can do backend COPY(Bryan)
Can shrink tables, pg_time and pg_log(Vadim & Erich)
Change debug level 2 to print queries only, changed debug heading
 layout(Bruce)
Change default decimal constant representation from float4 to
 float8(Bruce)
European date format now set when postmaster is started
Execute lowercase function names if not found with exact case
Fixes for aggregate/GROUP processing, allow 'select sum(func(x),
sum(x+y) from z'
Gist now included in the distribution(Marc)
Ident authentication of local users(Bryan)
Implement BETWEEN qualifier(Bruce)
Implement IN qualifier(Bruce)
libpq has PQgetisnull()(Bruce)
libpq++ improvements
New options to initdb(Bryan)
Pg_dump allow dump of OIDs(Bruce)
Pg_dump create indexes after tables are loaded for speed(Bruce)
Pg_dumpall dumps all databases, and the user table
Pginterface additions for NULL values(Bruce)
Prevent postmaster from being run as root
psql \h and \? is now readable(Bruce)
psql allow backslashed, semicolons anywhere on the line(Bruce)
psql changed command prompt for lines in query or in quotes(Bruce)
psql char(3) now displays as (bp)char in \d output(Bruce)
psql return code now more accurate(Bryan?)
psql updated help syntax(Bruce)
Re-visit and fix vacuum(Vadim)
Reduce size of regression diffs, remove timezone name difference(
Bruce)
Remove compile-time parameters to enable binary distributions(
Bryan)
Reverse meaning of HBA masks(Bryan)
Secure Authentication of local users(Bryan)
Speed up vacuum(Vadim)
Vacuum now had VERBOSE option(Bruce)

Source tree changes
-------------------
All functions now have prototypes that are compared against the
 calls
Allow asserts to be disabled easily from Makefile.global(Bruce)
Change oid constants used in code to #define names
Decoupled sparc and solaris defines(Kurt)
Gcc -Wall compiles cleanly with warnings only from unfixable
 constructs
Major include file reorganization/reduction(Marc)
Make now stops on compile failure(Bryan)
Makefile restructuring(Bryan, Marc)
Merge bsdi_2_1 to bsdi(Bruce)
Monitor program removed
Name change from Postgres95 to PostgreSQL
New config.h file(Marc, Bryan)
PG_VERSION now set to 6.0 and used by postmaster
Portability additions, including Ultrix, DG/UX, AIX, and Solaris
Reduced the number of #define's, centralized #define's
Remove duplicate OIDS in system tables(Dan)
```

```
Remove duplicate system catalog info or report mismatches(Dan)
Removed many os-specific #define's
Restructured object file generation/location(Bryan, Marc)
Restructured port-specific file locations(Bryan, Marc)
Unused/uninitialized variables corrected
```

E.282. Release 1.09

Release date: 1996-11-04

Sorry, we didn't keep track of changes from 1.02 to 1.09. Some of the changes listed in 6.0 were actually included in the 1.02.1 to 1.09 releases.

E.283. Release 1.02

Release date: 1996-08-01

E.283.1. Migration from version 1.02 to version 1.02.1

Here is a new migration file for 1.02.1. It includes the 'copy' change and a script to convert old ASCII files.

> **注意**
>
> The following notes are for the benefit of users who want to migrate databases from Postgres95 1.01 and 1.02 to Postgres95 1.02.1.
>
> If you are starting afresh with Postgres95 1.02.1 and do not need to migrate old databases, you do not need to read any further.

In order to upgrade older Postgres95 version 1.01 or 1.02 databases to version 1.02.1, the following steps are required:

1. Start up a new 1.02.1 postmaster

2. Add the new built-in functions and operators of 1.02.1 to 1.01 or 1.02 databases. This is done by running the new 1.02.1 server against your own 1.01 or 1.02 database and applying the queries attached at the end of the file. This can be done easily through `psql`. If your 1.01 or 1.02 database is named `testdb` and you have cut the commands from the end of this file and saved them in `addfunc.sql`:

   ```
   % psql testdb -f addfunc.sql
   ```

   Those upgrading 1.02 databases will get a warning when executing the last two statements in the file because they are already present in 1.02. This is not a cause for concern.

E.283.2. Dump/Reload Procedure

If you are trying to reload a pg_dump or text-mode, `copy tablename to stdout` generated with a previous version, you will need to run the attached `sed` script on the ASCII file before loading it into the database. The old format used '.' as end-of-data, while '\.' is now the end-of-data marker. Also, empty strings are now loaded in as '' rather than NULL. See the copy manual page for full details.

```
sed 's/^\.$/\\./g' <in_file >out_file
```

If you are loading an older binary copy or non-stdout copy, there is no end-of-data character, and hence no conversion necessary.

```
-- following lines added by agc to reflect the case-insensitive
-- regexp searching for varchar (in 1.02), and bpchar (in 1.02.1)
create operator ~* (leftarg = bpchar, rightarg = text, procedure =
 texticregexeq);
create operator !~* (leftarg = bpchar, rightarg = text, procedure =
 texticregexne);
create operator ~* (leftarg = varchar, rightarg = text, procedure =
 texticregexeq);
create operator !~* (leftarg = varchar, rightarg = text, procedure
 = texticregexne);
```

E.283.3. Changes

```
Source code maintenance and development
 * worldwide team of volunteers
 * the source tree now in CVS at ftp.ki.net

Enhancements
 * psql (and underlying libpq library) now has many more options
 for
   formatting output, including HTML
 * pg_dump now output the schema and/or the data, with many fixes
 to
   enhance completeness.
 * psql used in place of monitor in administration shell scripts.
   monitor to be deprecated in next release.
 * date/time functions enhanced
 * NULL insert/update/comparison fixed/enhanced
 * TCL/TK lib and shell fixed to work with both tck7.4/tk4.0 and
 tcl7.5/tk4.1

Bug Fixes (almost too numerous to mention)
 * indexes
 * storage management
 * check for NULL pointer before dereferencing
 * Makefile fixes

New Ports
 * added SolarisX86 port
 * added BSD/OS 2.1 port
 * added DG/UX port
```

E.284. Release 1.01

Release date: 1996-02-23

E.284.1. Migration from version 1.0 to version 1.01

The following notes are for the benefit of users who want to migrate databases from Postgres95 1.0 to Postgres95 1.01.

If you are starting afresh with Postgres95 1.01 and do not need to migrate old databases, you do not need to read any further.

In order to Postgres95 version 1.01 with databases created with Postgres95 version 1.0, the following steps are required:

1. Set the definition of `NAMEDATALEN` in `src/Makefile.global` to 16 and `OIDNAMELEN` to 20.
2. Decide whether you want to use Host based authentication.
   a. If you do, you must create a file name `pg_hba` in your top-level data directory (typically the value of your `$PGDATA`). `src/libpq/pg_hba` shows an example syntax.
   b. If you do not want host-based authentication, you can comment out the line:

      ```
      HBA = 1
      ```

      in `src/Makefile.global`

      Note that host-based authentication is turned on by default, and if you do not take steps A or B above, the out-of-the-box 1.01 will not allow you to connect to 1.0 databases.
3. Compile and install 1.01, but DO NOT do the `initdb` step.
4. Before doing anything else, terminate your 1.0 postmaster, and backup your existing `$PGDATA` directory.
5. Set your `PGDATA` environment variable to your 1.0 databases, but set up path up so that 1.01 binaries are being used.
6. Modify the file `$PGDATA/PG_VERSION` from 5.0 to 5.1
7. Start up a new 1.01 postmaster
8. Add the new built-in functions and operators of 1.01 to 1.0 databases. This is done by running the new 1.01 server against your own 1.0 database and applying the queries attached and saving in the file 1.0_to_1.01.sql. This can be done easily through `psql`. If your 1.0 database is name `testdb`:

   ```
   % psql testdb -f 1.0_to_1.01.sql
   ```

   and then execute the following commands (cut and paste from here):

   ```
   -- add builtin functions that are new to 1.01

   create function int4eqoid (int4, oid) returns bool as 'foo'
   language 'internal';
   create function oideqint4 (oid, int4) returns bool as 'foo'
   language 'internal';
   create function char2icregexeq (char2, text) returns bool as
    'foo'
   language 'internal';
   create function char2icregexne (char2, text) returns bool as
    'foo'
   language 'internal';
   create function char4icregexeq (char4, text) returns bool as
    'foo'
   language 'internal';
   create function char4icregexne (char4, text) returns bool as
    'foo'
   language 'internal';
   ```

```
create function char8icregexeq (char8, text) returns bool as
 'foo'
language 'internal';
create function char8icregexne (char8, text) returns bool as
 'foo'
language 'internal';
create function char16icregexeq (char16, text) returns bool as
 'foo'
language 'internal';
create function char16icregexne (char16, text) returns bool as
 'foo'
language 'internal';
create function texticregexeq (text, text) returns bool as
 'foo'
language 'internal';
create function texticregexne (text, text) returns bool as
 'foo'
language 'internal';

-- add builtin functions that are new to 1.01

create operator = (leftarg = int4, rightarg = oid, procedure =
 int4eqoid);
create operator = (leftarg = oid, rightarg = int4, procedure =
 oideqint4);
create operator ~* (leftarg = char2, rightarg = text, procedure
 = char2icregexeq);
create operator !~* (leftarg = char2, rightarg = text,
 procedure = char2icregexne);
create operator ~* (leftarg = char4, rightarg = text, procedure
 = char4icregexeq);
create operator !~* (leftarg = char4, rightarg = text,
 procedure = char4icregexne);
create operator ~* (leftarg = char8, rightarg = text, procedure
 = char8icregexeq);
create operator !~* (leftarg = char8, rightarg = text,
 procedure = char8icregexne);
create operator ~* (leftarg = char16, rightarg = text,
 procedure = char16icregexeq);
create operator !~* (leftarg = char16, rightarg = text,
 procedure = char16icregexne);
create operator ~* (leftarg = text, rightarg = text, procedure
 = texticregexeq);
create operator !~* (leftarg = text, rightarg = text, procedure
 = texticregexne);
```

E.284.2. Changes

```
Incompatibilities:
 * 1.01 is backwards compatible with 1.0 database provided the user
   follow the steps outlined in the MIGRATION_from_1.0_to_1.01
 file.
   If those steps are not taken, 1.01 is not compatible with 1.0
 database.

Enhancements:
 * added PQdisplayTuples() to libpq and changed monitor and psql to
 use it
```

```
 * added NeXT port (requires SysVIPC implementation)
 * added CAST .. AS ... syntax
 * added ASC and DESC key words
 * added 'internal' as a possible language for CREATE FUNCTION
   internal functions are C functions which have been statically
 linked
   into the postgres backend.
 * a new type "name" has been added for system identifiers (table
 names,
   attribute names, etc.)  This replaces the old char16 type.   The
   of name is set by the NAMEDATALEN #define in src/Makefile.global
 * a readable reference manual that describes the query language.
 * added host-based access control.  A configuration file ($PGDATA/
pg_hba)
   is used to hold the configuration data.  If host-based access
 control
   is not desired, comment out HBA=1 in src/Makefile.global.
 * changed regex handling to be uniform use of Henry Spencer's
 regex code
   regardless of platform.  The regex code is included in the
 distribution
 * added functions and operators for case-insensitive regular
 expressions.
   The operators are ~* and !~*.
 * pg_dump uses COPY instead of SELECT loop for better performance

Bug fixes:
 * fixed an optimizer bug that was causing core dumps when
   functions calls were used in comparisons in the WHERE clause
 * changed all uses of getuid to geteuid so that effective uids are
 used
 * psql now returns non-zero status on errors when using -c
 * applied public patches 1-14
```

E.285. Release 1.0

Release date: 1995-09-05

E.285.1. Changes

```
Copyright change:
 * The copyright of Postgres 1.0 has been loosened to be freely
 modifiable
   and modifiable for any purpose.  Please read the COPYRIGHT file.
   Thanks to Professor Michael Stonebraker for making this
 possible.

Incompatibilities:
 *  date formats have to be MM-DD-YYYY (or DD-MM-YYYY if you're
 using
   EUROPEAN STYLE).  This follows SQL-92 specs.
 *  "delimiters" is now a key word

Enhancements:
 *  sql LIKE syntax has been added
 *  copy command now takes an optional USING DELIMITER
 specification.
```

```
    delimiters can be any single-character string.
  *  IRIX 5.3 port has been added.
    Thanks to Paul Walmsley and others.
  *  updated pg_dump to work with new libpq
  *  \d has been added psql
    Thanks to Keith Parks
  *  regexp performance for architectures that use POSIX regex has
  been
    improved due to caching of precompiled patterns.
    Thanks to Alistair Crooks
  *  a new version of libpq++
    Thanks to William Wanders

Bug fixes:
  *  arbitrary userids can be specified in the createuser script
  *  \c to connect to other databases in psql now works.
  *  bad pg_proc entry for float4inc() is fixed
  *  users with usecreatedb field set can now create databases
  without
    having to be usesuper
  *  remove access control entries when the entry no longer has any
    privileges
  *  fixed non-portable datetimes implementation
  *  added kerberos flags to the src/backend/Makefile
  *  libpq now works with kerberos
  *  typographic errors in the user manual have been corrected.
  *  btrees with multiple index never worked, now we tell you they
  don't
    work when you try to use them
```

E.286. Postgres95 Release 0.03

Release date: 1995-07-21

E.286.1. Changes

```
Incompatible changes:
  * BETA-0.3 IS INCOMPATIBLE WITH DATABASES CREATED WITH PREVIOUS
  VERSIONS
    (due to system catalog changes and indexing structure changes).
  * double-quote (") is deprecated as a quoting character for string
  literals;
    you need to convert them to single quotes (').
  * name of aggregates (eg. int4sum) are renamed in accordance with
  the
    SQL standard (eg. sum).
  * CHANGE ACL syntax is replaced by GRANT/REVOKE syntax.
  * float literals (eg. 3.14) are now of type float4 (instead of
  float8 in
    previous releases); you might have to do typecasting if you
  depend on it
    being of type float8.  If you neglect to do the typecasting and
  you assign
    a float literal to a field of type float8, you might get
  incorrect values
    stored!
  * LIBPQ has been totally revamped so that frontend applications
```

```
    can connect to multiple backends
  * the usesysid field in pg_user has been changed from int2 to int4
  to
    allow wider range of Unix user ids.
  * the netbsd/freebsd/bsd o/s ports have been consolidated into a
    single BSD44_derived port.  (thanks to Alistair Crooks)

SQL standard-compliance (the following details changes that makes
 postgres95
more compliant to the SQL-92 standard):
  * the following SQL types are now built-in: smallint, int(eger),
  float, real,
    char(N), varchar(N), date and time.

    The following are aliases to existing postgres types:
                 smallint -> int2
                 integer, int -> int4
                 float, real  -> float4
    char(N) and varchar(N) are implemented as truncated text types.
  In
    addition, char(N) does blank-padding.
  * single-quote (') is used for quoting string literals; '' (in
  addition to
    \') is supported as means of inserting a single quote in a
  string
  * SQL standard aggregate names (MAX, MIN, AVG, SUM, COUNT) are
  used
    (Also, aggregates can now be overloaded, i.e. you can define
  your
    own MAX aggregate to take in a user-defined type.)
  * CHANGE ACL removed. GRANT/REVOKE syntax added.
    - Privileges can be given to a group using the "GROUP" key word.
          For example:
                 GRANT SELECT ON foobar TO GROUP my_group;
          The key word 'PUBLIC' is also supported to mean all users.

          Privileges can only be granted or revoked to one user or
  group
          at a time.

          "WITH GRANT OPTION" is not supported.  Only class owners
  can change
          access control
    - The default access control is to grant users readonly access.
      You must explicitly grant insert/update access to users.  To
  change
      this, modify the line in
                 src/backend/utils/acl.h
      that defines ACL_WORLD_DEFAULT

Bug fixes:
  * the bug where aggregates of empty tables were not run has been
  fixed. Now,
    aggregates run on empty tables will return the initial
  conditions of the
    aggregates. Thus, COUNT of an empty  table will now properly
  return 0.
    MAX/MIN of an empty table will return a row of value NULL.
```

```
 * allow the use of \; inside the monitor
 * the LISTEN/NOTIFY asynchronous notification mechanism now work
 * NOTIFY in rule action bodies now work
 * hash indexes work, and access methods in general should perform
 better.
   creation of large btree indexes should be much faster.  (thanks
 to Paul
   Aoki)

Other changes and enhancements:
 * addition of an EXPLAIN statement used for explaining the query
 execution
   plan (eg. "EXPLAIN SELECT * FROM EMP" prints out the execution
 plan for
   the query).
 * WARN and NOTICE messages no longer have timestamps on them. To
 turn on
   timestamps of error messages, uncomment the line in
   src/backend/utils/elog.h:
        /* define ELOG_TIMESTAMPS */
 * On an access control violation, the message
        "Either no such class or insufficient privilege"
   will be given.  This is the same message that is returned when
   a class is not found.  This dissuades non-privileged users from
   guessing the existence of privileged classes.
 * some additional system catalog changes have been made that are
 not
   visible to the user.

libpgtcl changes:
 * The -oid option has been added to the "pg_result" tcl command.
   pg_result -oid returns oid of the last row inserted.   If the
   last command was not an INSERT, then pg_result -oid returns "".
 * the large object interface is available as pg_lo* tcl commands:
   pg_lo_open, pg_lo_close, pg_lo_creat, etc.

Portability enhancements and New Ports:
 * flex/lex problems have been cleared up.  Now, you should be able
 to use
   flex instead of lex on any platforms.  We no longer make
 assumptions of
   what lexer you use based on the platform you use.
 * The Linux-ELF port is now supported.  Various configuration have
 been
   tested:  The following configuration is known to work:
        kernel 1.2.10, gcc 2.6.3, libc 4.7.2, flex 2.5.2, bison 1.
24
   with everything in ELF format,

New utilities:
 * ipcclean added to the distribution
   ipcclean usually does not need to be run, but if your backend
 crashes
   and leaves shared memory segments hanging around, ipcclean will
   clean them up for you.

New documentation:
 * the user manual has been revised and libpq documentation added.
```

E.287. Postgres95 Release 0.02

Release date: 1995-05-25

E.287.1. Changes

```
Incompatible changes:
 * The SQL statement for creating a database is 'CREATE DATABASE'
 instead
   of 'CREATEDB'. Similarly, dropping a database is 'DROP DATABASE'
 instead
   of 'DESTROYDB'. However, the names of the executables 'createdb'
 and
   'destroydb' remain the same.

New tools:
 * pgperl - a Perl (4.036) interface to Postgres95
 * pg_dump - a utility for dumping out a postgres database into a
        script file containing query commands. The script files are
 in an ASCII
        format and can be used to reconstruct the database, even on
 other
        machines and other architectures. (Also good for converting
        a Postgres 4.2 database to Postgres95 database.)

The following ports have been incorporated into postgres95-beta-0.
02:
 * the NetBSD port by Alistair Crooks
 * the AIX port by Mike Tung
 * the Windows NT port by Jon Forrest (more stuff but not done yet)
 * the Linux ELF port by Brian Gallew

The following bugs have been fixed in postgres95-beta-0.02:
 * new lines not escaped in COPY OUT and problem with COPY OUT when
 first
   attribute is a '.'
 * cannot type return to use the default user id in createuser
 * SELECT DISTINCT on big tables crashes
 * Linux installation problems
 * monitor doesn't allow use of 'localhost' as PGHOST
 * psql core dumps when doing \c or \l
 * the "pgtclsh" target missing from src/bin/pgtclsh/Makefile
 * libpgtcl has a hard-wired default port number
 * SELECT DISTINCT INTO TABLE hangs
 * CREATE TYPE doesn't accept 'variable' as the internallength
 * wrong result using more than 1 aggregate in a SELECT
```

E.288. Postgres95 Release 0.01

Release date: 1995-05-01

Initial release.

附录 F. 额外提供的模块与扩展

本附录与下一附录包含有关 PostgreSQL 发行版中 `contrib` 目录中可选组件的信息。这些组件包括移植工具、分析实用程序，以及不属于 PostgreSQL 核心系统的插件功能。之所以将它们单独提供，主要是因为它们面向的受众有限，或者实验性太强，
不适合作为主源码树的一部分。但这并不影响它们的实用性。

本附录介绍位于 `contrib` 中的扩展以及其他服务器插件模块库。附录 G介绍实用程序。

从源码发行版构建时，除非构建“world”目标（见步骤 2），否则这些组件不会自动构建。可以运行以下命令来构建并安装所有组件：

```
gmake
gmake install
```

这些命令应在已配置好的源码树的`contrib`目录中运行；
若只想构建并安装某个选定的模块，则可在该模块的子目录中执行同样的命令。
许多模块都带有回归测试，可以运行以下命令来执行：

```
gmake check
```

这是在安装前执行的命令；也可以运行

```
gmake installcheck
```

，此时需要已有一个PostgreSQL服务器正在运行。

如果你使用的是预打包版本的 PostgreSQL，这些组件通常会作为单独的子包提供，例如 `postgresql-contrib`。

许多模块提供新的用户定义函数、操作符或类型。安装代码后，要使用其中某个模块，
就需要在数据库系统中注册新的 SQL 对象。在PostgreSQL9.1 及后续版本中，这可通过执行CREATE EXTENSION命令完成。在一个新建的数据库中，你可以直接执行：

```
CREATE EXTENSION module_name;
```

该命令必须由数据库超级用户执行。它只会在当前数据库中注册新的 SQL 对象，因此需要在每个希望使用该模块功能的数据库中运行它。另一种做法是在数据库`template1`中运行它，这样该扩展默认会被复制到随后创建的数据库中。

许多模块允许你将其对象安装到所选的模式中。要这样做，请在`CREATE EXTENSION`命令中加入`SCHEMA schema_name`。默认情况下，这些对象会被放入当前的创建目标模式，而该模式默认是`public`。

如果你的数据库是从PostgreSQL9.1 之前的版本通过转储和重新载入升级而来的，并且其中一直在使用该模块的 9.1 之前版本，则应改为执行：

```
CREATE EXTENSION module_name FROM unpackaged;
```

这会将该模块在 9.1 之前版本中的对象更新为一个正规的扩展对象。今后该模块的更新将由ALTER EXTENSION管理。有关扩展更新的更多信息，请参见第 35.15 节。

但请注意，其中有些模块并不是这种意义上的“扩展”，而是通过其他方式加载到服务器中，例如借助shared_preload_libraries。详情见各模块文档。

F.1. adminpack

adminpack提供了一些支持函数，pgAdmin和其他管理工具可以使用它们来提供额外功能，例如远程管理服务器日志文件。所有这些函数都只允许超级用户使用。

表 F.1所示的函数提供对服务器所在机器上文件的写入访问。（另请参阅表 9.66中的函数，它们提供只读访问。）只能访问数据库集簇目录内的文件，
但相对路径和绝对路径都可以使用。

表 F.1. adminpack 函数

名称	返回类型	描述
pg_catalog.pg_file_write(filename text, data text, append boolean)	bigint	写入文本文件或向其追加内容
pg_catalog.pg_file_rename(oldname text, newname text [, archivename text])	boolean	重命名文件
pg_catalog.pg_file_unlink(filename text)	boolean	删除文件
pg_catalog.pg_logdir_ls()	setof record	列出log_directory目录中的日志文件

pg_file_write将指定的*data*写入由*filename*指定的文件中。如果*append*为 false，则该文件必须尚不存在。如果*append*为 true，则该文件可以已经存在，并且若已存在则会向其追加内容。返回写入的字节数。

pg_file_rename重命名文件。如果省略*archivename*或其值为空值，则它只是将*oldname*重命名为*newname*（后者必须尚不存在）。如果提供了*archivename*，它会先将*newname*重命名为*archivename*（后者必须尚不存在），然后再将*oldname*
重命名为*newname*。如果第二个重命名步骤失败，它会在报告错误之前尝试将*archivename*再改回*newname*。成功时返回 true；如果源文件不存在或不可写，则返回 false；其他情况会抛出错误。

pg_file_unlink删除指定的文件。成功时返回 true；如果指定的文件不存在，或者unlink()调用失败，则返回 false；其他情况会抛出错误。

pg_logdir_ls返回log_directory目录中所有日志文件的起始时间戳和路径名。
要使用此函数，log_filename参数必须保持默认设置（postgresql-%Y-%m-%d_%H%M%S.log）。

表 F.2所示的函数已经弃用，不应在新应用中使用；应改用表 9.59和表 9.66所示的函数。adminpack提供这些函数只是为了兼容旧版本的pgAdmin。

表 F.2. 已弃用的adminpack函数

名称	返回类型	描述
pg_catalog.pg_file_read(filename text, offset bigint, nbytes bigint)	text	pg_read_file()的别名
pg_catalog.pg_file_length(filename text)	bigint	与pg_stat_file()返回的size列相同

名称	返回类型	描述
`pg_catalog.pg_logfile_rotate()`	`integer`	`pg_rotate_logfile()`的别名，但请注意它返回整数 0 或 1，而不是 boolean

F.2. auth_delay

`auth_delay`会使服务器在报告认证失败之前短暂暂停，
从而增加对数据库密码进行暴力破解攻击的难度。注意，它并不能阻止拒绝服务攻击，
甚至可能加剧此类攻击，因为在报告认证失败前处于等待状态的进程仍会占用连接槽位。

为了使该模块正常工作，必须在`postgresql.conf`中通过shared_preload_
libraries加载它。

F.2.1. 配置参数

`auth_delay.milliseconds` (`int`)

在报告认证失败之前等待的毫秒数。默认值为 0。

这些参数必须在`postgresql.conf`中设置。典型的用法如下：

```
# postgresql.conf
shared_preload_libraries = 'auth_delay'

auth_delay.milliseconds = '500'
```

F.2.2. 作者

KaiGai Kohei `<kaigai@ak.jp.nec.com>`

F.3. auto_explain

The `auto_explain` module provides a means for logging execution plans of slow statements automatically, without having to run EXPLAIN by hand. This is especially helpful for tracking down un-optimized queries in large applications.

The module provides no SQL-accessible functions. To use it, simply load it into the server. You can load it into an individual session:

```
LOAD 'auto_explain';
```

(You must be superuser to do that.) More typical usage is to preload it into all sessions by including `auto_explain` in shared_preload_libraries in `postgresql.conf`. Then you can track unexpectedly slow queries no matter when they happen. Of course there is a price in overhead for that.

F.3.1. Configuration Parameters

There are several configuration parameters that control the behavior of `auto_explain`. Note that the default behavior is to do nothing, so you must set at least `auto_explain.log_min_duration` if you want any results.

`auto_explain.log_min_duration` (`integer`)

`auto_explain.log_min_duration` is the minimum statement execution time, in milliseconds, that will cause the statement's plan to be logged. Setting this to zero logs all plans. Minus-one (the default) disables logging of plans. For example, if you set it to `250ms` then all statements that run 250ms or longer will be logged. Only superusers can change this setting.

`auto_explain.log_analyze` (`boolean`)

`auto_explain.log_analyze` causes `EXPLAIN ANALYZE` output, rather than just `EXPLAIN` output, to be printed when an execution plan is logged. This parameter is off by default. Only superusers can change this setting.

> **注意**
>
> When this parameter is on, per-plan-node timing occurs for all statements executed, whether or not they run long enough to actually get logged. This can have an extremely negative impact on performance. Turning off `auto_explain.log_timing` ameliorates the performance cost, at the price of obtaining less information.

`auto_explain.log_buffers` (`boolean`)

`auto_explain.log_buffers` controls whether buffer usage statistics are printed when an execution plan is logged; it's equivalent to the `BUFFERS` option of `EXPLAIN`. This parameter has no effect unless `auto_explain.log_analyze` is enabled. This parameter is off by default. Only superusers can change this setting.

`auto_explain.log_timing` (`boolean`)

`auto_explain.log_timing` controls whether per-node timing information is printed when an execution plan is logged; it's equivalent to the `TIMING` option

of EXPLAIN. The overhead of repeatedly reading the system clock can slow down queries significantly on some systems, so it may be useful to set this parameter to off when only actual row counts, and not exact times, are needed. This parameter has no effect unless auto_explain.log_analyze is enabled. This parameter is on by default. Only superusers can change this setting.

auto_explain.log_verbose (boolean)

auto_explain.log_verbose controls whether verbose details are printed when an execution plan is logged; it's equivalent to the VERBOSE option of EXPLAIN. This parameter is off by default. Only superusers can change this setting.

auto_explain.log_format (enum)

auto_explain.log_format selects the EXPLAIN output format to be used. The allowed values are text, xml, json, and yaml. The default is text. Only superusers can change this setting.

auto_explain.log_nested_statements (boolean)

auto_explain.log_nested_statements causes nested statements (statements executed inside a function) to be considered for logging. When it is off, only top-level query plans are logged. This parameter is off by default. Only superusers can change this setting.

In ordinary usage, these parameters are set in postgresql.conf, although superusers can alter them on-the-fly within their own sessions. Typical usage might be:

```
# postgresql.conf
shared_preload_libraries = 'auto_explain'

auto_explain.log_min_duration = '3s'
```

F.3.2. Example

```
postgres=# LOAD 'auto_explain';
postgres=# SET auto_explain.log_min_duration = 0;
postgres=# SET auto_explain.log_analyze = true;
postgres=# SELECT count(*)
           FROM pg_class, pg_index
           WHERE oid = indrelid AND indisunique;
```

This might produce log output such as:

```
LOG:  duration: 3.651 ms  plan:
  Query Text: SELECT count(*)
              FROM pg_class, pg_index
              WHERE oid = indrelid AND indisunique;
  Aggregate  (cost=16.79..16.80 rows=1 width=0) (actual time=3.626.
.3.627 rows=1 loops=1)
    ->  Hash Join  (cost=4.17..16.55 rows=92 width=0) (actual time=
3.349..3.594 rows=92 loops=1)
          Hash Cond: (pg_class.oid = pg_index.indrelid)
          ->  Seq Scan on pg_class  (cost=0.00..9.55 rows=255
 width=4) (actual time=0.016..0.140 rows=255 loops=1)
          ->  Hash  (cost=3.02..3.02 rows=92 width=4) (actual time=
3.238..3.238 rows=92 loops=1)
                Buckets: 1024  Batches: 1  Memory Usage: 4kB
```

```
                ->  Seq Scan on pg_index  (cost=0.00..3.02 rows=92
 width=4) (actual time=0.008..3.187 rows=92 loops=1)
                      Filter: indisunique
```

F.3.3. Author

Takahiro Itagaki <itagaki.takahiro@oss.ntt.co.jp>

F.4. btree_gin

btree_gin提供了 GIN 操作符类，为以下数据类型实现 B-树等价行为：int2、int4、int8、float4、float8、timestamp with time zone、timestamp without time zone、time with time zone、time without time zone、date、interval、oid、money、"char"、varchar、text、bytea、bit、varbit、macaddr、inet和cidr。

一般来说，这些操作符类的性能不会优于等价的标准 B-树索引方法，而且它们缺少标准 B-树代码的一项主要特性：强制唯一性的能力。不过，它们对于 GIN 测试以及作为开发其他 GIN 操作符类的基础很有用。此外，对于同时测试一个可由 GIN 建索引的列和一个可由 B-树建索引的列的查询，创建一个使用这些操作符类之一的多列 GIN 索引，可能比创建两个必须通过位图 AND 运算组合的独立索引更高效。

F.4.1. 用法示例

```
CREATE TABLE test (a int4);
-- create index
CREATE INDEX testidx ON test USING gin (a);
-- query
SELECT * FROM test WHERE a < 10;
```

F.4.2. 作者

Teodor Sigaev (<teodor@stack.net>) 和 Oleg Bartunov (<oleg@sai.msu.su>) 。参阅http://www.sai.msu.su/~megera/oddmuse/index.cgi/Gin。

F.5. btree_gist

btree_gist提供了 GiST 索引操作符类，可为以下数据类型实现与 B-树等价的行为：int2, int4, int8, float4, float8, numeric, timestamp with time zone, timestamp without time zone, time with time zone, time without time zone, date, interval, oid, money, char, varchar, text, bytea, bit, varbit, macaddr, inet, 和 cidr。

一般来说，这些操作符类的性能不会优于对应的标准 B-树索引方法，而且它们缺少标准 B-树实现的一项主要特性：强制唯一性的能力。不过，正如下文所述，它们提供了一些 B-树索引所不具备的其他特性。另外，当需要多列 GiST 索引，而其中某些列的数据类型只能用 GiST 建立索引、其他列只是简单数据类型时，这些操作符类就很有用。最后，这些操作符类对于 GiST 测试以及作为开发其他 GiST 操作符类的基础也很有用。

除典型的 B-树搜索操作符之外，btree_gist还为<>（“不等于”）提供索引支持。这在与下文描述的排他约束结合使用时可能很有用。

此外，对于那些具有自然距离度量的数据类型，btree_gist定义了距离操作符<->，并为使用该操作符的最近邻搜索提供 GiST 索引支持。为以下类型提供了距离操作符：int2、int4、int8、float4、float8、timestamp with time zone、timestamp without time zone、time without time zone、date、interval、oid和money。

F.5.1. 用法示例

一个使用btree_gist代替btree的简单示例：

```
CREATE TABLE test (a int4);
-- create index
CREATE INDEX testidx ON test USING gist (a);
-- query
SELECT * FROM test WHERE a < 10;
-- nearest-neighbor search: find the ten entries closest to "42"
SELECT *, a <-> 42 AS dist FROM test ORDER BY a <-> 42 LIMIT 10;
```

使用排他约束来强制执行这样一条规则：动物园中的一个笼子只能容纳一种动物：

```
=> CREATE TABLE zoo (
  cage   INTEGER,
  animal TEXT,
  EXCLUDE USING gist (cage WITH =, animal WITH <>)
);

=> INSERT INTO zoo VALUES(123, 'zebra');
INSERT 0 1
=> INSERT INTO zoo VALUES(123, 'zebra');
INSERT 0 1
=> INSERT INTO zoo VALUES(123, 'lion');
ERROR:  conflicting key value violates exclusion constraint "zoo_
cage_animal_excl"
DETAIL:  Key (cage, animal)=(123, lion) conflicts with existing key
 (cage, animal)=(123, zebra).
=> INSERT INTO zoo VALUES(124, 'lion');
INSERT 0 1
```

F.5.2. 作者

Teodor Sigaev（<teodor@stack.net>）、Oleg Bartunov（<oleg@sai.msu.su>）和 Janko Richter（<jankorichter@yahoo.de>）。更多信息见http://www.sai.msu.su/~megera/postgres/gist/。

F.6. chkpass

此模块实现了用于存储加密密码的数据类型chkpass。
每个密码在输入时都会自动转换为加密形式，并且始终以加密形式存储。比较时，
只需与明文密码比较，比较函数会在比较之前将其加密。

代码中预留了在判定密码容易被破解时报告错误的机制。不过，
目前这只是一个不做任何事情的占位实现。

如果在输入字符串前加上冒号，它就会被视为已加密的密码，直接存储而不再加密。
这样就可以输入以前加密过的密码。

输出时会在前面加上冒号。这样就可以转储并重新载入密码，而不必重新加密。
如果需要不带冒号的加密密码，请使用raw()函数。这样就可以将此类型与 Apache 的Auth_PostgreSQL模块等工具一起使用。

加密使用标准 Unix 函数crypt()，因此也受该函数所有常见限制的影响，尤其是只考虑密码的前八个字符。

请注意，chkpass数据类型不能建立索引。

使用示例：

```
test=# create table test (p chkpass);
CREATE TABLE
test=# insert into test values ('hello');
INSERT 0 1
test=# select * from test;
       p
----------------
 :dVGkpXdOrE3ko
(1 row)

test=# select raw(p) from test;
      raw
---------------
 dVGkpXdOrE3ko
(1 row)

test=# select p = 'hello' from test;
 ?column?
----------
 t
(1 row)

test=# select p = 'goodbye' from test;
 ?column?
----------
 f
(1 row)
```

F.6.1. 作者

D'Arcy J.M. Cain（<darcy@druid.net>）

F.7. citext – 大小写不敏感的字符串类型

citext模块提供一种大小写不敏感的字符串类型 citext。本质上，它在比较值时会在内部调用 lower。除此之外，它的行为几乎与 text完全相同。

F.7.1. 原理

在PostgreSQL中进行大小写不敏感匹配的标准方法，一直是在比较值时使用lower函数，例如：

```
SELECT * FROM tab WHERE lower(col) = LOWER(?);
```

这种做法效果尚可，但有几个缺点：

- 它会让 SQL 语句变得冗长，而且你必须时刻记得同时对列和查询值调用 lower。
- 除非你创建一个使用lower的函数索引，否则它不会使用索引。
- 如果把列声明为UNIQUE或PRIMARY KEY，隐式生成的索引仍然是大小写敏感的。因此，它既无法用于大小写不敏感搜索，也不能以大小写不敏感的方式强制唯一性。

citext数据类型允许你在 SQL 查询中省去对 lower的调用，并允许主键不区分大小写。与text一样，citext也与区域设置相关，
这意味着大写字符和小写字符如何匹配取决于数据库 LC_CTYPE设置的规则。同样，这种行为与在查询中使用 lower完全一致。但由于这是由数据类型透明地完成的，因此你不必在查询中额外记住任何特殊处理。

F.7.2. 如何使用

下面是一个简单的使用示例：

```
CREATE TABLE users (
    nick CITEXT PRIMARY KEY,
    pass TEXT   NOT NULL
);

INSERT INTO users VALUES ( 'larry',  md5(random()::text) );
INSERT INTO users VALUES ( 'Tom',    md5(random()::text) );
INSERT INTO users VALUES ( 'Damian', md5(random()::text) );
INSERT INTO users VALUES ( 'NEAL',   md5(random()::text) );
INSERT INTO users VALUES ( 'Bjørn',  md5(random()::text) );

SELECT * FROM users WHERE nick = 'Larry';
```

这个SELECT语句仍会返回一个元组，尽管nick列中存的是larry，而查询条件写的是Larry。

F.7.3. 字符串比较行为

citext在比较时会先把每个字符串转换为小写（就像调用了lower一样），
然后再按常规方式比较结果。因此，举例来说，如果两个字符串经过lower
后得到的结果相同，它们就会被视为相等。

为了尽可能贴近大小写不敏感排序规则的行为，一些字符串处理操作符和函数都提供了citext专用版本。例如，当应用于citext时，正则表达式操作符~和~*
表现相同：它们都会以大小写不敏感的方式匹配。!~和 !~*也是如此，LIKE操作符 ~~和~~*、以及 !~~和!~~*也一样。如果你希望进行大小写敏感匹配，可以把这些操作符的参数转换为text。

同样地，如果这些函数的参数是citext，它们也会以大小写不敏感方式进行匹配：

- regexp_matches()
- regexp_replace()
- regexp_split_to_array()
- regexp_split_to_table()
- replace()
- split_part()
- strpos()
- translate()

对于正则表达式函数，如果你想按大小写敏感方式匹配，可以指定“c”标志来强制大小写敏感匹配。否则，若要获得大小写敏感行为，就必须在调用这些函数之前先把值转换为text。

F.7.4. 限制

- citext的大小写折叠行为依赖于数据库的 LC_CTYPE设置，因此它如何比较值是在创建数据库时确定的。按照 Unicode 标准中的定义，它并不是真正意义上的大小写不敏感。实际上，这意味着只要你对当前排序规则满意，通常也会对 citext的比较结果满意。但是，如果数据库中存有多种语言的数据，而排序规则只适用于其中某一种语言，那么其他语言的用户可能会发现查询结果并不符合预期。
- 从PostgreSQL 9.1 起，你可以为 citext列或数据值附加COLLATE说明。当前，citext操作符在比较已完成大小写折叠的字符串时，会遵从非默认的 COLLATE说明；但最初转换为小写这一步，始终仍按数据库的 LC_CTYPE设置执行（也就是说，等同于给出了 COLLATE "default"）。未来的发行版中，这一点可能会改变，从而使这两步都遵循输入的COLLATE说明。
- citext不如text高效，因为操作符函数和 B-树比较函数必须复制数据，并将数据转换为小写后才能进行比较。不过，在需要大小写不敏感匹配时，它仍然比使用lower略高效一些。
- 如果你在某些场景下需要大小写敏感比较，而在另一些场景下又需要大小写不敏感比较，那么citext并不会帮上太多忙。标准做法是使用 text类型，并在需要大小写不敏感比较时手工调用 lower；如果只是偶尔需要这类比较，这种做法是完全可行的。如果你大多数时候都需要大小写不敏感行为，而只是在少数场景下需要大小写敏感比较，那么可以考虑把数据存储为citext，并在需要大小写敏感比较时显式地把列转换为text。无论哪种情况，如果你希望这两类搜索都足够快，就都需要建立两个索引。
- 包含citext操作符的模式必须位于当前 search_path中（通常是public）；如果不在，调用的将是普通的、大小写敏感的text操作符。

F.7.5. 作者

David E. Wheeler <david@kineticode.com>

灵感来自 Donald Fraser 最初编写的citext模块。

F.8. cube – 多维立方体数据类型

该模块实现了用于表示多维立方体的cube数据类型。

F.8.1. 语法

表 F.3展示了cube类型合法的外部表示。*x*、*y*等表示浮点数。

表 F.3. cube 外部表示

x	一个一维点（或者长度为零的一维区间）
(*x*)	同上
x1,*x2*,...,*xn*	n 维空间中的一个点，在内部表示为零体积立方体
(*x1*,*x2*,...,*xn*)	同上
(*x*),(*y*)	一个从*x*开始、到*y*结束的一维区间，反过来也可以；顺序无关紧要
[(*x*),(*y*)]	同上
(*x1*,...,*xn*),(*y1*,...,*yn*)	一个 n 维立方体，由一对对角相对的角点表示
[(*x1*,...,*xn*),(*y1*,...,*yn*)]	同上

立方体的两个对角点以何种顺序输入都无关紧要。必要时，cube函数会自动交换这些值，以创建统一的“左下角 – 右上角”内部表示。

空白会被忽略，因此 [(*x*),(*y*)]与 [(*x*), (*y*)] 相同。

F.8.2. 精度

值在内部以 64 位浮点数存储。这意味着有效数字超过大约 16 位的数值会被截断。

F.8.3. 用法

cube模块为cube值提供了一个 GiST 索引操作符类。GiST 操作符类支持的操作符如表 F.4所示。

表 F.4. cube GiST 操作符

操作符	描述
a = b	立方体 a 和 b 相同。
a && b	立方体 a 和 b 重叠。
a @> b	立方体 a 包含立方体 b。
a <@ b	立方体 a 包含在立方体 b 中。

（在 PostgreSQL 8.2 之前，包含操作符 @> 和 <@ 分别称为 @ 和 ~。这些名称仍然可用，但已弃用，最终将被删除。请注意，旧名称与核心几何数据类型以前采用的约定正好相反！）

也提供了标准的 B-树操作符，例如

操作符	描述
[a, b] < [c, d]	小于
[a, b] > [c, d]	大于

这些操作符除了排序之外，在实际用途上没有太大意义。它们首先比较 (a) 与 (c)，如果相等，再比较 (b) 与 (d)。这在大多数情况下能得到相当好的排序结果，如果你希望对此类型使用 ORDER BY，这会很有用。

表 F.5展示了可用函数。

表 F.5. cube 函数

cube(float8) returns cube	创建一个一维立方体，其两个坐标相同。cube(1) == '(1)'
cube(float8, float8) returns cube	创建一个一维立方体。cube(1,2) == '(1),(2)'
cube(float8[]) returns cube	根据数组定义的坐标创建一个零体积立方体。cube(ARRAY[1,2]) == '(1,2)'
cube(float8[], float8[]) returns cube	创建一个立方体，其右上角和左下角坐标由这两个数组定义，这两个数组必须等长。cube('{1,2}'::float[], '{3,4}'::float[]) == '(1,2),(3,4)'
cube(cube, float8) returns cube	通过向现有立方体增加一个维度来创建新的立方体，新坐标的两部分取相同的值。这可用于根据计算得到的值逐步构建立方体。cube('(1)',2) == '(1,2),(1,2)'
cube(cube, float8, float8) returns cube	通过向现有立方体增加一个维度来创建新的立方体。这可用于根据计算得到的值逐步构建立方体。cube('(1,2)',3,4) == '(1,3),(2,4)'
cube_dim(cube) returns int	返回立方体的维数
cube_ll_coord(cube, int) returns double	返回立方体左下角的第 n 个坐标值
cube_ur_coord(cube, int) returns double	返回立方体右上角的第 n 个坐标值
cube_is_point(cube) returns bool	如果立方体是一个点，也就是定义它的两个角相同，则返回真。
cube_distance(cube, cube) returns double	返回两个立方体之间的距离。如果两个立方体都是点，这就是普通的距离函数。
cube_subset(cube, int[]) returns cube	根据数组中给出的维度索引列表，从现有立方体创建一个新立方体。可用于找出单个维度的左下角（LL）和右上角（UR）坐标，例如，cube_subset(cube('(1,3,5),(6,7,8)'), ARRAY[2]) = '(3),(7)'。也可用于删除维度，或者按需要重新排列维度，例如，cube_subset(cube('(1,3,5),(6,7,8)'), ARRAY[3,2,1,1]) = '(5, 3, 1, 1),(8, 7, 6, 6)'。
cube_union(cube, cube) returns cube	生成两个立方体的并集
cube_inter(cube, cube) returns cube	生成两个立方体的交集
cube_enlarge(cube c, double r, int n) returns cube	将立方体至少 n 个维度的大小增加指定半径。如果半径为负，则改为缩小立方体。此函数对于创建点周围的边界框、以搜索附近点很有用。所有已定义的维度都会按半径 r 改变。左下角（LL）坐标减小 r，右上角（UR）坐标增大 r。如果某个左下角坐标增大到超过对应的右上角坐标（这只可能在 r < 0 时发生），则将两个坐标都设为它们的平均值。如果 n 大于已定义维度数且立方体正在扩大（r >= 0），则额外坐标以 0 作为基准。

F.8.4. 默认规则

下面这个并集：

```
select cube_union('(0,5,2),(2,3,1)', '0');
```

```
cube_union
-------------------
(0, 0, 0),(2, 5, 2)
(1 row)
```

并不违背常识，下面这个交集也是如此：

```
select cube_inter('(0,-1),(1,1)', '(-2),(2)');
cube_inter
-------------
(0, 0),(1, 0)
(1 row)
```

在所有对不同维度立方体执行的二元操作中，都假定维度较低的那个是一个笛卡尔投影，也就是说，在字符串表示中省略的坐标位置上补 0。上面的例子等价于：

```
cube_union('(0,5,2),(2,3,1)','(0,0,0),(0,0,0)');
cube_inter('(0,-1),(1,1)','(-2,0),(2,0)');
```

下面的包含谓词使用的是点语法，但实际上第二个参数在内部表示为一个 box。这种语法使我们无须单独定义点类型以及用于 (box,point) 谓词的函数。

```
select cube_contains('(0,0),(1,1)', '0.5,0.5');
cube_contains
--------------
t
(1 row)
```

F.8.5. 注意

有关用法示例，请参见回归测试`sql/cube.sql`。

为了避免用户轻易把事情弄坏，立方体的维数上限被设为 100。如果需要更大的值，可在`cubedata.h`中修改该限制。

F.8.6. 致谢

原作者：Gene Selkov, Jr. `<selkovjr@mcs.anl.gov>`，阿贡国家实验室数学与计算机科学部。

我首先要感谢 Joe Hellerstein 教授 (http://db.cs.berkeley.edu/jmh/)，他为我阐明了 GiST (http://gist.cs.berkeley.edu/) 的要旨；也感谢他曾经的学生 Andy Dong (http://best.me.berkeley.edu/~adong/) 为 Illustra 编写了示例，http://best.berkeley.edu/~adong/rtree/index.html。我同样感谢过去和现在所有的 Postgres 开发者，他们使我得以创造自己的世界并在其中不受打扰地生活。我还要感谢阿贡实验室以及美国能源部，多年来始终如一地支持我的数据库研究。

Bruno Wolff III `<bruno@wolff.to>` 在 2002 年 8 月和 9 月对该软件包做了一些小更新，包括将精度从单精度改为双精度，并增加了一些新函数。

Joshua Reich `<josh@root.net>` 在 2006 年 7 月做了进一步更新。这些更新包括加入`cube(float8[], float8[])`，并清理代码，使其使用 V1 调用协议而不是已废弃的 V0 协议。

F.9. dblink

`dblink` is a module which supports connections to other PostgreSQL databases from within a database session.

dblink_connect

dblink_connect — opens a persistent connection to a remote database

大纲

```
dblink_connect(text connstr) returns text
dblink_connect(text connname, text connstr) returns text
```

Description

`dblink_connect()` establishes a connection to a remote PostgreSQL database. The server and database to be contacted are identified through a standard libpq connection string. Optionally, a name can be assigned to the connection. Multiple named connections can be open at once, but only one unnamed connection is permitted at a time. The connection will persist until closed or until the database session is ended.

The connection string may also be the name of an existing foreign server. It is recommended to use the `postgresql_fdw_validator` when defining the corresponding foreign-data wrapper. See the example below, as well as the following: CREATE FOREIGN DATA WRAPPER, CREATE SERVER, CREATE USER MAPPING

Arguments

connname

The name to use for this connection; if omitted, an unnamed connection is opened, replacing any existing unnamed connection.

connstr

libpq-style connection info string, for example `hostaddr=127.0.0.1 port=5432 dbname=mydb user=postgres password=mypasswd`. For details see `PQconnectdb` in 第 31.1 节.

Return Value

Returns status, which is always `OK` (since any error causes the function to throw an error instead of returning).

Notes

Only superusers may use `dblink_connect` to create non-password-authenticated connections. If non-superusers need this capability, use `dblink_connect_u` instead.

It is unwise to choose connection names that contain equal signs, as this opens a risk of confusion with connection info strings in other `dblink` functions.

Examples

```
SELECT dblink_connect('dbname=postgres');
 dblink_connect
----------------
 OK
(1 row)

SELECT dblink_connect('myconn', 'dbname=postgres');
 dblink_connect
```

```
----------------
 OK
(1 row)

-- FOREIGN DATA WRAPPER functionality
-- Note: local connection must require password authentication for
 this to work properly
--       Otherwise, you will receive the following error from
 dblink_connect():
--       ------------------------------------------------------------
------------
--       ERROR:  password is required
--       DETAIL:  Non-superuser cannot connect if the server does
 not request a password.
--       HINT:  Target server's authentication method must be
 changed.
CREATE USER dblink_regression_test WITH PASSWORD 'secret';
CREATE FOREIGN DATA WRAPPER postgresql VALIDATOR postgresql_fdw_
validator;
CREATE SERVER fdtest FOREIGN DATA WRAPPER postgresql OPTIONS (
hostaddr '127.0.0.1', dbname 'contrib_regression');

CREATE USER MAPPING FOR dblink_regression_test SERVER fdtest
 OPTIONS (user 'dblink_regression_test', password 'secret');
GRANT USAGE ON FOREIGN SERVER fdtest TO dblink_regression_test;
GRANT SELECT ON TABLE foo TO dblink_regression_test;

\set ORIGINAL_USER :USER
\c - dblink_regression_test
SELECT dblink_connect('myconn', 'fdtest');
 dblink_connect
----------------
 OK
(1 row)

SELECT * FROM dblink('myconn','SELECT * FROM foo') AS t(a int, b
 text, c text[]);
 a  | b |       c
----+---+---------------
  0 | a | {a0,b0,c0}
  1 | b | {a1,b1,c1}
  2 | c | {a2,b2,c2}
  3 | d | {a3,b3,c3}
  4 | e | {a4,b4,c4}
  5 | f | {a5,b5,c5}
  6 | g | {a6,b6,c6}
  7 | h | {a7,b7,c7}
  8 | i | {a8,b8,c8}
  9 | j | {a9,b9,c9}
 10 | k | {a10,b10,c10}
(11 rows)

\c - :ORIGINAL_USER
REVOKE USAGE ON FOREIGN SERVER fdtest FROM dblink_regression_test;
REVOKE SELECT ON TABLE foo FROM  dblink_regression_test;
DROP USER MAPPING FOR dblink_regression_test SERVER fdtest;
DROP USER dblink_regression_test;
DROP SERVER fdtest;
```

```
DROP FOREIGN DATA WRAPPER postgresql;
```

dblink_connect_u

dblink_connect_u — opens a persistent connection to a remote database, insecurely

大纲

```
dblink_connect_u(text connstr) returns text
dblink_connect_u(text connname, text connstr) returns text
```

Description

`dblink_connect_u()` is identical to `dblink_connect()`, except that it will allow non-superusers to connect using any authentication method.

If the remote server selects an authentication method that does not involve a password, then impersonation and subsequent escalation of privileges can occur, because the session will appear to have originated from the user as which the local PostgreSQL server runs. Also, even if the remote server does demand a password, it is possible for the password to be supplied from the server environment, such as a `~/.pgpass` file belonging to the server's user. This opens not only a risk of impersonation, but the possibility of exposing a password to an untrustworthy remote server. Therefore, `dblink_connect_u()` is initially installed with all privileges revoked from `PUBLIC`, making it un-callable except by superusers. In some situations it may be appropriate to grant `EXECUTE` permission for `dblink_connect_u()` to specific users who are considered trustworthy, but this should be done with care. It is also recommended that any `~/.pgpass` file belonging to the server's user not contain any records specifying a wildcard host name.

For further details see `dblink_connect()`.

dblink_disconnect

dblink_disconnect — closes a persistent connection to a remote database

大纲

```
dblink_disconnect() returns text
dblink_disconnect(text connname) returns text
```

Description

`dblink_disconnect()` closes a connection previously opened by `dblink_connect()`. The form with no arguments closes an unnamed connection.

Arguments

`connname`

The name of a named connection to be closed.

Return Value

Returns status, which is always `OK` (since any error causes the function to throw an error instead of returning).

Examples

```
SELECT dblink_disconnect();
 dblink_disconnect
-------------------
 OK
(1 row)

SELECT dblink_disconnect('myconn');
 dblink_disconnect
-------------------
 OK
(1 row)
```

dblink

dblink — executes a query in a remote database

大纲

```
dblink(text connname, text sql [, bool fail_on_error]) returns
 setof record
dblink(text connstr, text sql [, bool fail_on_error]) returns setof
 record
dblink(text sql [, bool fail_on_error]) returns setof record
```

Description

`dblink` executes a query (usually a `SELECT`, but it can be any SQL statement that returns rows) in a remote database.

When two `text` arguments are given, the first one is first looked up as a persistent connection's name; if found, the command is executed on that connection. If not found, the first argument is treated as a connection info string as for `dblink_connect`, and the indicated connection is made just for the duration of this command.

Arguments

connname

Name of the connection to use; omit this parameter to use the unnamed connection.

connstr

A connection info string, as previously described for `dblink_connect`.

sql

The SQL query that you wish to execute in the remote database, for example `select * from foo`.

fail_on_error

If true (the default when omitted) then an error thrown on the remote side of the connection causes an error to also be thrown locally. If false, the remote error is locally reported as a NOTICE, and the function returns no rows.

Return Value

The function returns the row(s) produced by the query. Since `dblink` can be used with any query, it is declared to return `record`, rather than specifying any particular set of columns. This means that you must specify the expected set of columns in the calling query – otherwise PostgreSQL would not know what to expect. Here is an example:

```
SELECT *
    FROM dblink('dbname=mydb', 'select proname, prosrc from pg_
proc')
      AS t1(proname name, prosrc text)
    WHERE proname LIKE 'bytea%';
```

The “alias” part of the `FROM` clause must specify the column names and types that the function will return. (Specifying column names in an alias is actually standard SQL

syntax, but specifying column types is a PostgreSQL extension.) This allows the system to understand what `*` should expand to, and what `proname` in the `WHERE` clause refers to, in advance of trying to execute the function. At run time, an error will be thrown if the actual query result from the remote database does not have the same number of columns shown in the `FROM` clause. The column names need not match, however, and `dblink` does not insist on exact type matches either. It will succeed so long as the returned data strings are valid input for the column type declared in the `FROM` clause.

Notes

A convenient way to use `dblink` with predetermined queries is to create a view. This allows the column type information to be buried in the view, instead of having to spell it out in every query. For example,

```
CREATE VIEW myremote_pg_proc AS
  SELECT *
    FROM dblink('dbname=postgres', 'select proname, prosrc from pg_
proc')
    AS t1(proname name, prosrc text);

SELECT * FROM myremote_pg_proc WHERE proname LIKE 'bytea%';
```

Examples

```
SELECT * FROM dblink('dbname=postgres', 'select proname, prosrc
 from pg_proc')
  AS t1(proname name, prosrc text) WHERE proname LIKE 'bytea%';
  proname   |   prosrc
------------+------------
 byteacat   | byteacat
 byteaeq    | byteaeq
 bytealt    | bytealt
 byteale    | byteale
 byteagt    | byteagt
 byteage    | byteage
 byteane    | byteane
 byteacmp   | byteacmp
 bytealike  | bytealike
 byteanlike | byteanlike
 byteain    | byteain
 byteaout   | byteaout
(12 rows)

SELECT dblink_connect('dbname=postgres');
 dblink_connect
----------------
 OK
(1 row)

SELECT * FROM dblink('select proname, prosrc from pg_proc')
  AS t1(proname name, prosrc text) WHERE proname LIKE 'bytea%';
  proname   |   prosrc
------------+------------
 byteacat   | byteacat
 byteaeq    | byteaeq
 bytealt    | bytealt
 byteale    | byteale
 byteagt    | byteagt
```

```
 byteage    | byteage
 byteane    | byteane
 byteacmp   | byteacmp
 bytealike  | bytealike
 byteanlike | byteanlike
 byteain    | byteain
 byteaout   | byteaout
(12 rows)

SELECT dblink_connect('myconn', 'dbname=regression');
 dblink_connect
----------------
 OK
(1 row)

SELECT * FROM dblink('myconn', 'select proname, prosrc from pg_
proc')
  AS t1(proname name, prosrc text) WHERE proname LIKE 'bytea%';
  proname   |   prosrc
------------+------------
 bytearecv  | bytearecv
 byteasend  | byteasend
 byteale    | byteale
 byteagt    | byteagt
 byteage    | byteage
 byteane    | byteane
 byteacmp   | byteacmp
 bytealike  | bytealike
 byteanlike | byteanlike
 byteacat   | byteacat
 byteaeq    | byteaeq
 bytealt    | bytealt
 byteain    | byteain
 byteaout   | byteaout
(14 rows)
```

dblink_exec

dblink_exec — executes a command in a remote database

大纲

```
dblink_exec(text connname, text sql [, bool fail_on_error]) returns
 text
dblink_exec(text connstr, text sql [, bool fail_on_error]) returns
 text
dblink_exec(text sql [, bool fail_on_error]) returns text
```

Description

`dblink_exec` executes a command (that is, any SQL statement that doesn't return rows) in a remote database.

When two `text` arguments are given, the first one is first looked up as a persistent connection's name; if found, the command is executed on that connection. If not found, the first argument is treated as a connection info string as for `dblink_connect`, and the indicated connection is made just for the duration of this command.

Arguments

connname

Name of the connection to use; omit this parameter to use the unnamed connection.

connstr

A connection info string, as previously described for `dblink_connect`.

sql

The SQL command that you wish to execute in the remote database, for example `insert into foo values(0,'a','{"a0","b0","c0"}')`.

fail_on_error

If true (the default when omitted) then an error thrown on the remote side of the connection causes an error to also be thrown locally. If false, the remote error is locally reported as a NOTICE, and the function's return value is set to `ERROR`.

Return Value

Returns status, either the command's status string or `ERROR`.

Examples

```
SELECT dblink_connect('dbname=dblink_test_standby');
 dblink_connect
----------------
 OK
(1 row)

SELECT dblink_exec('insert into foo values(21,''z'',''{"a0","b0",
"c0"}'');');
```

```
   dblink_exec
-----------------
 INSERT 943366 1
(1 row)

SELECT dblink_connect('myconn', 'dbname=regression');
 dblink_connect
----------------
 OK
(1 row)

SELECT dblink_exec('myconn', 'insert into foo values(21,''z'',''{
"a0","b0","c0"}'');');
   dblink_exec
------------------
 INSERT 6432584 1
(1 row)

SELECT dblink_exec('myconn', 'insert into pg_class values (''foo'')
',false);
NOTICE:  sql error
DETAIL:  ERROR:  null value in column "relnamespace" violates not-
null constraint

 dblink_exec
-------------
 ERROR
(1 row)
```

dblink_open

dblink_open — opens a cursor in a remote database

大纲

```
dblink_open(text cursorname, text sql [, bool fail_on_error])
 returns text
dblink_open(text connname, text cursorname, text sql [, bool fail_
on_error]) returns text
```

Description

`dblink_open()` opens a cursor in a remote database. The cursor can subsequently be manipulated with `dblink_fetch()` and `dblink_close()`.

Arguments

connname

Name of the connection to use; omit this parameter to use the unnamed connection.

cursorname

The name to assign to this cursor.

sql

The `SELECT` statement that you wish to execute in the remote database, for example `select * from pg_class`.

fail_on_error

If true (the default when omitted) then an error thrown on the remote side of the connection causes an error to also be thrown locally. If false, the remote error is locally reported as a NOTICE, and the function's return value is set to `ERROR`.

Return Value

Returns status, either `OK` or `ERROR`.

Notes

Since a cursor can only persist within a transaction, `dblink_open` starts an explicit transaction block (`BEGIN`) on the remote side, if the remote side was not already within a transaction. This transaction will be closed again when the matching `dblink_close` is executed. Note that if you use `dblink_exec` to change data between `dblink_open` and `dblink_close`, and then an error occurs or you use `dblink_disconnect` before `dblink_close`, your change will be lost because the transaction will be aborted.

Examples

```
SELECT dblink_connect('dbname=postgres');
 dblink_connect
----------------
 OK
```

```
(1 row)

SELECT dblink_open('foo', 'select proname, prosrc from pg_proc');
 dblink_open
-------------
 OK
(1 row)
```

dblink_fetch

dblink_fetch — returns rows from an open cursor in a remote database

大纲

```
dblink_fetch(text cursorname, int howmany [, bool fail_on_error])
 returns setof record
dblink_fetch(text connname, text cursorname, int howmany [, bool
 fail_on_error]) returns setof record
```

Description

`dblink_fetch` fetches rows from a cursor previously established by `dblink_open`.

Arguments

connname

Name of the connection to use; omit this parameter to use the unnamed connection.

cursorname

The name of the cursor to fetch from.

howmany

The maximum number of rows to retrieve. The next *howmany* rows are fetched, starting at the current cursor position, moving forward. Once the cursor has reached its end, no more rows are produced.

fail_on_error

If true (the default when omitted) then an error thrown on the remote side of the connection causes an error to also be thrown locally. If false, the remote error is locally reported as a NOTICE, and the function returns no rows.

Return Value

The function returns the row(s) fetched from the cursor. To use this function, you will need to specify the expected set of columns, as previously discussed for `dblink`.

Notes

On a mismatch between the number of return columns specified in the `FROM` clause, and the actual number of columns returned by the remote cursor, an error will be thrown. In this event, the remote cursor is still advanced by as many rows as it would have been if the error had not occurred. The same is true for any other error occurring in the local query after the remote `FETCH` has been done.

Examples

```
SELECT dblink_connect('dbname=postgres');
 dblink_connect
----------------
 OK
(1 row)
```

```
SELECT dblink_open('foo', 'select proname, prosrc from pg_proc
 where proname like ''bytea%''');
 dblink_open
-------------
 OK
(1 row)

SELECT * FROM dblink_fetch('foo', 5) AS (funcname name, source
 text);
 funcname |  source
----------+----------
 byteacat | byteacat
 byteacmp | byteacmp
 byteaeq  | byteaeq
 byteage  | byteage
 byteagt  | byteagt
(5 rows)

SELECT * FROM dblink_fetch('foo', 5) AS (funcname name, source
 text);
 funcname  |  source
-----------+-----------
 byteain   | byteain
 byteale   | byteale
 bytealike | bytealike
 bytealt   | bytealt
 byteane   | byteane
(5 rows)

SELECT * FROM dblink_fetch('foo', 5) AS (funcname name, source
 text);
  funcname  |   source
------------+------------
 byteanlike | byteanlike
 byteaout   | byteaout
(2 rows)

SELECT * FROM dblink_fetch('foo', 5) AS (funcname name, source
 text);
 funcname | source
----------+--------
(0 rows)
```

dblink_close

dblink_close — closes a cursor in a remote database

大纲

```
dblink_close(text cursorname [, bool fail_on_error]) returns text
dblink_close(text connname, text cursorname [, bool fail_on_error])
 returns text
```

Description

`dblink_close` closes a cursor previously opened with `dblink_open`.

Arguments

connname

Name of the connection to use; omit this parameter to use the unnamed connection.

cursorname

The name of the cursor to close.

fail_on_error

If true (the default when omitted) then an error thrown on the remote side of the connection causes an error to also be thrown locally. If false, the remote error is locally reported as a NOTICE, and the function's return value is set to `ERROR`.

Return Value

Returns status, either `OK` or `ERROR`.

Notes

If `dblink_open` started an explicit transaction block, and this is the last remaining open cursor in this connection, `dblink_close` will issue the matching `COMMIT`.

Examples

```
SELECT dblink_connect('dbname=postgres');
 dblink_connect
----------------
 OK
(1 row)

SELECT dblink_open('foo', 'select proname, prosrc from pg_proc');
 dblink_open
-------------
 OK
(1 row)

SELECT dblink_close('foo');
 dblink_close
--------------
```

```
 OK
(1 row)
```

dblink_get_connections

dblink_get_connections — returns the names of all open named dblink connections

大纲

```
dblink_get_connections() returns text[]
```

Description

`dblink_get_connections` returns an array of the names of all open named `dblink` connections.

Return Value

Returns a text array of connection names, or NULL if none.

Examples

```
SELECT dblink_get_connections();
```

dblink_error_message

dblink_error_message — gets last error message on the named connection

大纲

```
dblink_error_message(text connname) returns text
```

Description

`dblink_error_message` fetches the most recent remote error message for a given connection.

Arguments

connname

Name of the connection to use.

Return Value

Returns last error message, or an empty string if there has been no error in this connection.

Examples

```
SELECT dblink_error_message('dtest1');
```

dblink_send_query

dblink_send_query — sends an async query to a remote database

大纲

```
dblink_send_query(text connname, text sql) returns int
```

Description

`dblink_send_query` sends a query to be executed asynchronously, that is, without immediately waiting for the result. There must not be an async query already in progress on the connection.

After successfully dispatching an async query, completion status can be checked with `dblink_is_busy`, and the results are ultimately collected with `dblink_get_result`. It is also possible to attempt to cancel an active async query using `dblink_cancel_query`.

Arguments

connname

Name of the connection to use.

sql

The SQL statement that you wish to execute in the remote database, for example `select * from pg_class`.

Return Value

Returns 1 if the query was successfully dispatched, 0 otherwise.

Examples

```
SELECT dblink_send_query('dtest1', 'SELECT * FROM foo WHERE f1 <
 3');
```

dblink_is_busy

dblink_is_busy — checks if connection is busy with an async query

大纲

```
dblink_is_busy(text connname) returns int
```

Description

`dblink_is_busy` tests whether an async query is in progress.

Arguments

connname

Name of the connection to check.

Return Value

Returns 1 if connection is busy, 0 if it is not busy. If this function returns 0, it is guaranteed that `dblink_get_result` will not block.

Examples

```
SELECT dblink_is_busy('dtest1');
```

dblink_get_notify

dblink_get_notify — retrieve async notifications on a connection

大纲

```
dblink_get_notify() returns setof (notify_name text, be_pid int,
 extra text)
dblink_get_notify(text connname) returns setof (notify_name text,
 be_pid int, extra text)
```

Description

`dblink_get_notify` retrieves notifications on either the unnamed connection, or on a named connection if specified. To receive notifications via dblink, `LISTEN` must first be issued, using `dblink_exec`. For details see LISTEN and NOTIFY.

Arguments

connname

The name of a named connection to get notifications on.

Return Value

Returns `setof (notify_name text, be_pid int, extra text)`, or an empty set if none.

Examples

```
SELECT dblink_exec('LISTEN virtual');
 dblink_exec
-------------
 LISTEN
(1 row)

SELECT * FROM dblink_get_notify();
 notify_name | be_pid | extra
-------------+--------+-------
(0 rows)

NOTIFY virtual;
NOTIFY

SELECT * FROM dblink_get_notify();
 notify_name | be_pid | extra
-------------+--------+-------
 virtual     |   1229 |
(1 row)
```

dblink_get_result

dblink_get_result — gets an async query result

大纲

```
dblink_get_result(text connname [, bool fail_on_error]) returns
 setof record
```

Description

`dblink_get_result` collects the results of an asynchronous query previously sent with `dblink_send_query`. If the query is not already completed, `dblink_get_result` will wait until it is.

Arguments

connname

Name of the connection to use.

fail_on_error

If true (the default when omitted) then an error thrown on the remote side of the connection causes an error to also be thrown locally. If false, the remote error is locally reported as a NOTICE, and the function returns no rows.

Return Value

For an async query (that is, a SQL statement returning rows), the function returns the row(s) produced by the query. To use this function, you will need to specify the expected set of columns, as previously discussed for `dblink`.

For an async command (that is, a SQL statement not returning rows), the function returns a single row with a single text column containing the command's status string. It is still necessary to specify that the result will have a single text column in the calling `FROM` clause.

Notes

This function must be called if `dblink_send_query` returned 1. It must be called once for each query sent, and one additional time to obtain an empty set result, before the connection can be used again.

When using `dblink_send_query` and `dblink_get_result`, dblink fetches the entire remote query result before returning any of it to the local query processor. If the query returns a large number of rows, this can result in transient memory bloat in the local session. It may be better to open such a query as a cursor with `dblink_open` and then fetch a manageable number of rows at a time. Alternatively, use plain `dblink()`, which avoids memory bloat by spooling large result sets to disk.

Examples

```
contrib_regression=# SELECT dblink_connect('dtest1', 'dbname=
contrib_regression');
 dblink_connect
----------------
```

```
 OK
(1 row)

contrib_regression=# SELECT * FROM
contrib_regression-# dblink_send_query('dtest1', 'select * from foo
 where f1 < 3') AS t1;
 t1
----
  1
(1 row)

contrib_regression=# SELECT * FROM dblink_get_result('dtest1') AS
 t1(f1 int, f2 text, f3 text[]);
 f1 | f2 |     f3
----+----+------------
  0 | a  | {a0,b0,c0}
  1 | b  | {a1,b1,c1}
  2 | c  | {a2,b2,c2}
(3 rows)

contrib_regression=# SELECT * FROM dblink_get_result('dtest1') AS
 t1(f1 int, f2 text, f3 text[]);
 f1 | f2 | f3
----+----+----
(0 rows)

contrib_regression=# SELECT * FROM
contrib_regression-# dblink_send_query('dtest1', 'select * from foo
 where f1 < 3; select * from foo where f1 > 6') AS t1;
 t1
----
  1
(1 row)

contrib_regression=# SELECT * FROM dblink_get_result('dtest1') AS
 t1(f1 int, f2 text, f3 text[]);
 f1 | f2 |     f3
----+----+------------
  0 | a  | {a0,b0,c0}
  1 | b  | {a1,b1,c1}
  2 | c  | {a2,b2,c2}
(3 rows)

contrib_regression=# SELECT * FROM dblink_get_result('dtest1') AS
 t1(f1 int, f2 text, f3 text[]);
 f1 | f2 |      f3
----+----+---------------
  7 | h  | {a7,b7,c7}
  8 | i  | {a8,b8,c8}
  9 | j  | {a9,b9,c9}
 10 | k  | {a10,b10,c10}
(4 rows)

contrib_regression=# SELECT * FROM dblink_get_result('dtest1') AS
 t1(f1 int, f2 text, f3 text[]);
 f1 | f2 | f3
----+----+----
(0 rows)
```

dblink_cancel_query

dblink_cancel_query — cancels any active query on the named connection

大纲

```
dblink_cancel_query(text connname) returns text
```

Description

`dblink_cancel_query` attempts to cancel any query that is in progress on the named connection. Note that this is not certain to succeed (since, for example, the remote query might already have finished). A cancel request simply improves the odds that the query will fail soon. You must still complete the normal query protocol, for example by calling `dblink_get_result`.

Arguments

connname

Name of the connection to use.

Return Value

Returns `OK` if the cancel request has been sent, or the text of an error message on failure.

Examples

```
SELECT dblink_cancel_query('dtest1');
```

dblink_get_pkey

dblink_get_pkey — returns the positions and field names of a relation's primary key fields

大纲

```
dblink_get_pkey(text relname) returns setof dblink_pkey_results
```

Description

`dblink_get_pkey` provides information about the primary key of a relation in the local database. This is sometimes useful in generating queries to be sent to remote databases.

Arguments

relname

Name of a local relation, for example `foo` or `myschema.mytab`. Include double quotes if the name is mixed-case or contains special characters, for example `"Foo Bar"`; without quotes, the string will be folded to lower case.

Return Value

Returns one row for each primary key field, or no rows if the relation has no primary key. The result row type is defined as

```
CREATE TYPE dblink_pkey_results AS (position int, colname text);
```

The `position` column simply runs from 1 to *N*; it is the number of the field within the primary key, not the number within the table's columns.

Examples

```
CREATE TABLE foobar (
    f1 int,
    f2 int,
    f3 int,
    PRIMARY KEY (f1, f2, f3)
);
CREATE TABLE

SELECT * FROM dblink_get_pkey('foobar');
 position | colname
----------+---------
        1 | f1
        2 | f2
        3 | f3
(3 rows)
```

dblink_build_sql_insert

dblink_build_sql_insert — builds an INSERT statement using a local tuple, replacing the primary key field values with alternative supplied values

大纲

```
dblink_build_sql_insert(text relname,
                        int2vector primary_key_attnums,
                        integer num_primary_key_atts,
                        text[] src_pk_att_vals_array,
                        text[] tgt_pk_att_vals_array) returns text
```

Description

`dblink_build_sql_insert` can be useful in doing selective replication of a local table to a remote database. It selects a row from the local table based on primary key, and then builds a SQL `INSERT` command that will duplicate that row, but with the primary key values replaced by the values in the last argument. (To make an exact copy of the row, just specify the same values for the last two arguments.)

Arguments

relname

Name of a local relation, for example `foo` or `myschema.mytab`. Include double quotes if the name is mixed-case or contains special characters, for example `"Foo Bar"`; without quotes, the string will be folded to lower case.

primary_key_attnums

Attribute numbers (1-based) of the primary key fields, for example `1 2`.

num_primary_key_atts

The number of primary key fields.

src_pk_att_vals_array

Values of the primary key fields to be used to look up the local tuple. Each field is represented in text form. An error is thrown if there is no local row with these primary key values.

tgt_pk_att_vals_array

Values of the primary key fields to be placed in the resulting `INSERT` command. Each field is represented in text form.

Return Value

Returns the requested SQL statement as text.

Notes

As of PostgreSQL 9.0, the attribute numbers in *primary_key_attnums* are interpreted as logical column numbers, corresponding to the column's position in `SELECT * FROM relname`. Previous versions interpreted the numbers as physical column positions.

There is a difference if any column(s) to the left of the indicated column have been dropped during the lifetime of the table.

Examples

```
SELECT dblink_build_sql_insert('foo', '1 2', 2, '{"1", "a"}', '{
"1", "b''a"}');
             dblink_build_sql_insert
--------------------------------------------------
 INSERT INTO foo(f1,f2,f3) VALUES('1','b''a','1')
(1 row)
```

dblink_build_sql_delete

dblink_build_sql_delete — builds a DELETE statement using supplied values for primary key field values

大纲

```
dblink_build_sql_delete(text relname,
                        int2vector primary_key_attnums,
                        integer num_primary_key_atts,
                        text[] tgt_pk_att_vals_array) returns text
```

Description

`dblink_build_sql_delete` can be useful in doing selective replication of a local table to a remote database. It builds a SQL `DELETE` command that will delete the row with the given primary key values.

Arguments

relname

Name of a local relation, for example `foo` or `myschema.mytab`. Include double quotes if the name is mixed-case or contains special characters, for example `"Foo Bar"`; without quotes, the string will be folded to lower case.

primary_key_attnums

Attribute numbers (1-based) of the primary key fields, for example `1 2`.

num_primary_key_atts

The number of primary key fields.

tgt_pk_att_vals_array

Values of the primary key fields to be used in the resulting `DELETE` command. Each field is represented in text form.

Return Value

Returns the requested SQL statement as text.

Notes

As of PostgreSQL 9.0, the attribute numbers in *primary_key_attnums* are interpreted as logical column numbers, corresponding to the column's position in `SELECT * FROM relname`. Previous versions interpreted the numbers as physical column positions. There is a difference if any column(s) to the left of the indicated column have been dropped during the lifetime of the table.

Examples

```
SELECT dblink_build_sql_delete('"MyFoo"', '1 2', 2, '{"1", "b"}');
           dblink_build_sql_delete
----------------------------------------------
 DELETE FROM "MyFoo" WHERE f1='1' AND f2='b'
```

```
(1 row)
```

dblink_build_sql_update

dblink_build_sql_update — builds an UPDATE statement using a local tuple, replacing the primary key field values with alternative supplied values

大纲

```
dblink_build_sql_update(text relname,
                        int2vector primary_key_attnums,
                        integer num_primary_key_atts,
                        text[] src_pk_att_vals_array,
                        text[] tgt_pk_att_vals_array) returns text
```

Description

`dblink_build_sql_update` can be useful in doing selective replication of a local table to a remote database. It selects a row from the local table based on primary key, and then builds a SQL `UPDATE` command that will duplicate that row, but with the primary key values replaced by the values in the last argument. (To make an exact copy of the row, just specify the same values for the last two arguments.) The `UPDATE` command always assigns all fields of the row – the main difference between this and `dblink_build_sql_insert` is that it's assumed that the target row already exists in the remote table.

Arguments

`relname`

Name of a local relation, for example `foo` or `myschema.mytab`. Include double quotes if the name is mixed-case or contains special characters, for example `"Foo Bar"`; without quotes, the string will be folded to lower case.

`primary_key_attnums`

Attribute numbers (1-based) of the primary key fields, for example `1 2`.

`num_primary_key_atts`

The number of primary key fields.

`src_pk_att_vals_array`

Values of the primary key fields to be used to look up the local tuple. Each field is represented in text form. An error is thrown if there is no local row with these primary key values.

`tgt_pk_att_vals_array`

Values of the primary key fields to be placed in the resulting `UPDATE` command. Each field is represented in text form.

Return Value

Returns the requested SQL statement as text.

Notes

As of PostgreSQL 9.0, the attribute numbers in *`primary_key_attnums`* are interpreted as logical column numbers, corresponding to the column's position in `SELECT * FROM`

`relname`. Previous versions interpreted the numbers as physical column positions. There is a difference if any column(s) to the left of the indicated column have been dropped during the lifetime of the table.

Examples

```
SELECT dblink_build_sql_update('foo', '1 2', 2, '{"1", "a"}', '{
"1", "b"}');
                   dblink_build_sql_update
-------------------------------------------------------------
 UPDATE foo SET f1='1',f2='b',f3='1' WHERE f1='1' AND f2='b'
(1 row)
```

F.10. dict_int – 用于整数的示例全文搜索词典

dict_int是一个全文搜索附加词典模板的示例。
引入这个示例词典是为了控制整数（有符号和无符号）的索引，使这类数字能够被索引，
同时又避免唯一词的数量过度增长，因为那会严重影响搜索性能。

F.10.1. 配置

该词典接受两个选项：

- maxlen参数指定整数词中允许的最大数字位数。默认值为 6。
- rejectlong参数指定超长整数应被截断还是忽略。如果rejectlong为false（默认值），词典将返回该整数的前maxlen位数字。如果rejectlong为true，词典会将超长整数视为停用词，因此不会为其建立索引。注意，这也意味着无法搜索这类整数。

F.10.2. 用法

安装dict_int扩展后，会创建一个文本搜索模板 intdict_template以及一个基于该模板、使用默认参数的词典 intdict。你可以修改这些参数，例如：

```
mydb# ALTER TEXT SEARCH DICTIONARY intdict (MAXLEN = 4, REJECTLONG
 = true);
ALTER TEXT SEARCH DICTIONARY
```

或者基于该模板创建新的词典。

要测试该词典，可以尝试：

```
mydb# select ts_lexize('intdict', '12345678');
 ts_lexize
-----------
 {123456}
```

但在实际使用中，通常需要像第 12 章所述那样，将它包含到某个文本搜索配置中。可能类似如下：

```
ALTER TEXT SEARCH CONFIGURATION english
    ALTER MAPPING FOR int, uint WITH intdict;
```

F.11. dict_xsyn – 示例同义词全文检索词典

dict_xsyn（扩展同义词词典）是全文检索的一个附加词典模板示例。
这种词典类型会将词替换为其同义词组，因此可以使用某个词的任一同义词来搜索该词。

F.11.1. 配置

该dict_xsyn词典接受以下选项：

- matchorig控制词典是否接受原词。默认为true。
- matchsynonyms控制词典是否接受同义词。默认为false。
- keeporig控制词典输出中是否包含原词。默认为true。
- keepsynonyms控制词典输出中是否包含同义词。默认为true。
- rules是包含同义词列表的文件的基本名。该文件必须存放在$SHAREDIR/tsearch_data/（其中$SHAREDIR表示PostgreSQL安装的共享数据目录）中。文件名必须以.rules结尾（这一后缀不应写入rules参数）。

规则文件具有下列格式：

- 每一行表示某个单词的一组同义词，且该单词位于行首。同义词之间用空白分隔，例如：

  ```
  word syn1 syn2 syn3
  ```

- 井号（#）是注释分隔符。它可以出现在一行中的任意位置，该行余下的内容都会被跳过。

示例可参见安装在$SHAREDIR/tsearch_data/中的 xsyn_sample.rules。

F.11.2. 用法

安装dict_xsyn扩展会使用默认参数创建一个全文检索模板 xsyn_template和一个基于它的词典xsyn。你可以修改这些参数，例如：

```
mydb# ALTER TEXT SEARCH DICTIONARY xsyn (RULES='my_rules',
 KEEPORIG=false);
ALTER TEXT SEARCH DICTIONARY
```

或者基于该模板创建新的词典。

要测试该词典，可以尝试：

```
mydb=# SELECT ts_lexize('xsyn', 'word');
      ts_lexize
-----------------------
 {syn1,syn2,syn3}

mydb# ALTER TEXT SEARCH DICTIONARY xsyn (RULES='my_rules',
 KEEPORIG=true);
ALTER TEXT SEARCH DICTIONARY

mydb=# SELECT ts_lexize('xsyn', 'word');
      ts_lexize
-----------------------
 {word,syn1,syn2,syn3}
```

```
mydb# ALTER TEXT SEARCH DICTIONARY xsyn (RULES='my_rules',
 KEEPORIG=false, MATCHSYNONYMS=true);
ALTER TEXT SEARCH DICTIONARY

mydb=# SELECT ts_lexize('xsyn', 'syn1');
      ts_lexize
-----------------------
 {syn1,syn2,syn3}

mydb# ALTER TEXT SEARCH DICTIONARY xsyn (RULES='my_rules',
 KEEPORIG=true, MATCHORIG=false, KEEPSYNONYMS=false);
ALTER TEXT SEARCH DICTIONARY

mydb=# SELECT ts_lexize('xsyn', 'syn1');
      ts_lexize
-----------------------
 {word}
```

但实际使用通常需要把它纳入第 12 章中描述的全文检索配置。可能会像这样：

```
ALTER TEXT SEARCH CONFIGURATION english
    ALTER MAPPING FOR word, asciiword WITH xsyn, english_stem;
```

F.12. dummy_seclabel

dummy_seclabel模块的存在只是为了支持对SECURITY LABEL语句的回归测试。它不适合在生产环境中使用。

F.12.1. 缘由

SECURITY LABEL语句允许用户为数据库对象分配安全标签；但是，只有当可装载模块明确允许时才能分配安全标签，因此提供这个模块是为了能够进行正常的回归测试。

打算在生产环境中使用的安全标签提供程序通常依赖于特定于平台的特性，例如SE-Linux。这个模块是平台无关的，因此更适合用于回归测试。

F.12.2. 用法

下面是一个简单的用法示例：

```
# postgresql.conf
shared_preload_libraries = 'dummy_seclabel'

postgres=# CREATE TABLE t (a int, b text);
CREATE TABLE
postgres=# SECURITY LABEL ON TABLE t IS 'classified';
SECURITY LABEL
```

dummy_seclabel模块只提供四个硬编码的标签：unclassified、classified、secret和top secret。它不允许将任何其他字符串用作安全标签。

这些标签并不用于实施访问控制。它们只用于检查SECURITY LABEL语句是否按预期工作。

F.12.3. 作者

KaiGai Kohei <kaigai@ak.jp.nec.com>

F.13. earthdistance – 计算大圆距离

earthdistance 模块提供了两种不同的方法，用于计算地球表面上的大圆距离。首先介绍的一种依赖于 cube 模块（在安装 earthdistance 之前必须先安装该模块）。第二种方法基于内置的 point 数据类型，使用经度和纬度作为坐标。

在这个模块中，地球被假定为完全球形的。（如果这种假设对你而言精度不足，可以参见 PostGIS[1] 项目。）

F.13.1. 基于立方体的地球距离

数据以立方体形式存储，而这些立方体实际上是点（两个角相同），使用 3 个坐标分别表示距地心的 x、y、z 距离。这里还提供了一个基于 cube 的域 earth，其中包含约束检查，用以确认该值满足这些限制，并且与地球实际表面足够接近。

地球半径取自 earth() 函数，其单位为米。不过，只要修改这一个函数，你就可以让该模块改用其他单位，或者使用你认为更合适的半径值。

这个模块在天文数据库中也有用途。天文学家大概会希望修改 earth()，使其返回半径 180/pi()，这样距离单位就是度。

这里提供了一些函数，用于按经纬度（以度为单位）输入、输出经纬度、计算两点之间的大圆距离，以及方便地指定一个可用于索引搜索的边界框。

所提供的函数见表 F.6。

表 F.6. 基于立方体的地球距离函数

函数	返回值	描述
earth()	float8	返回假定的地球半径。
sec_to_gc(float8)	float8	将地球表面两点间的普通直线（割线）距离转换为它们之间的大圆距离。
gc_to_sec(float8)	float8	将地球表面两点间的大圆距离转换为它们之间的普通直线（割线）距离。
ll_to_earth(float8, float8)	earth	给定某点以度为单位的纬度（参数 1）和经度（参数 2），返回该点在地球表面上的位置。
latitude(earth)	float8	返回地球表面上一个点的纬度（以度数形式）。
longitude(earth)	float8	返回地球表面上一个点的经度（以度数形式）。
earth_distance(earth, earth)	float8	返回地球表面上两个点之间的大圆距离。
earth_box(earth, float8)	cube	返回一个适合使用 cube @> 操作符进行索引搜索的框，用于查找距离某个位置在给定大圆距离以内的点。框中的某些点到该位置的距离会超过指定的大圆距离，因此查询中应包含使用earth_distance 的二次检查。

[1] http://postgis.net/

F.13.2. 基于点的地球距离

该模块的第二部分依赖于将地球上的位置表示为 `point` 类型的值，其中第一个分量被视为以度为单位的经度，第二个分量被视为以度为单位的纬度。之所以采用 (longitude, latitude) 而不是相反的顺序，是因为经度更接近 x 轴的直观概念，而纬度更接近 y 轴。

这一部分只提供了一个操作符，见表 F.7。

表 F.7. 基于点的地球距离操作符

操作符	返回值	描述
`point <@> point`	`float8`	给出地球表面两点之间的法定英里距离。

注意，与该模块基于 `cube` 的部分不同，这里的单位是固定的：修改 `earth()` 函数不会影响该操作符的结果。

经度/纬度表示的一个缺点是，你需要小心靠近两极以及接近经度 +/- 180 度时的边界情况。基于 `cube` 的表示可以避免这些不连续性。

F.14. file_fdw

file_fdw 模块提供外部数据包装器 file_fdw，可用于访问服务器文件系统中的数据文件。数据文件必须采用 COPY FROM 可读取的格式；详见COPY。

使用该包装器创建的外部表可以具有以下选项：

filename

指定要读取的文件。必需。必须是绝对路径名。

format

指定文件的格式，与COPY的FORMAT选项相同。

header

指定文件是否包含标题行，与COPY的HEADER选项相同。

delimiter

指定文件的定界符字符，与COPY的DELIMITER选项相同。

quote

指定文件的引用字符，与COPY的QUOTE选项相同。

escape

指定文件的转义字符，与COPY的ESCAPE选项相同。

null

指定文件的空值串，与COPY的NULL选项相同。

encoding

指定文件的编码，与COPY的ENCODING选项相同。

使用此包装器创建的外部表的列可以具有下列选项：

force_not_null

这是一个布尔选项。如果为真，它指定该列的值不应与空值串（即文件级的 null选项）匹配。其效果与在COPY的 FORCE_NOT_NULL选项中列出该列相同。

file_fdw当前尚不支持COPY的OIDS和FORCE_QUOTE选项。

这些选项只能在外部表或其列上指定，不能在file_fdw外部数据包装器的选项中指定，也不能在使用该包装器的服务器或用户映射的选项中指定。

出于安全原因，更改表级选项需要超级用户权限：只有超级用户才能决定读取哪个文件。原则上可以允许非超级用户更改其他选项，但目前尚不支持。

对于使用file_fdw的外部表，EXPLAIN会显示待读取的文件名。除非指定COSTS OFF，否则还会显示文件大小（以字节为单位）。

例 F.1. 为 PostgreSQL CSV 日志创建外部表

file_fdw的一个明显用途，是将 PostgreSQL 活动日志作为表提供以供查询。要做到这一点，首先必须将日志记录到 CSV 文件中，这里将该文件称为pglog.csv。首先，将file_fdw 安装为扩展：

```
CREATE EXTENSION file_fdw;
```

然后创建一个外部服务器：

```
CREATE SERVER pglog FOREIGN DATA WRAPPER file_fdw;
```

现在可以创建外部数据表了。使用`CREATE FOREIGN TABLE`命令时，需要定义表的列、CSV文件名及其格式：

```
CREATE FOREIGN TABLE pglog (
  log_time timestamp(3) with time zone,
  user_name text,
  database_name text,
  process_id integer,
  connection_from text,
  session_id text,
  session_line_num bigint,
  command_tag text,
  session_start_time timestamp with time zone,
  virtual_transaction_id text,
  transaction_id bigint,
  error_severity text,
  sql_state_code text,
  message text,
  detail text,
  hint text,
  internal_query text,
  internal_query_pos integer,
  context text,
  query text,
  query_pos integer,
  location text,
  application_name text
) SERVER pglog
OPTIONS ( filename '/home/josh/9.1/data/pg_log/pglog.csv', format
 'csv' );
```

到这里就完成了 – 现在可以直接查询日志了。当然，在生产环境中，还需要定义某种方法来处理日志轮转。

F.15. fuzzystrmatch – 确定字符串的相似性和距离

fuzzystrmatch模块提供多个函数，用于确定字符串的相似性和距离。

> **小心**
>
> 目前，soundex、metaphone、dmetaphone 和 dmetaphone_alt 函数不能很好地处理多字节编码（例如 UTF-8）。

F.15.1. Soundex

Soundex 系统是一种通过将发音相近的名字转换为相同代码来进行匹配的方法。它最初在 1880 年、1900 年和 1910 年的美国人口普查中使用。注意，Soundex 对非英语名字并不是很有用。

fuzzystrmatch模块提供两个用于处理 Soundex 代码的函数：

```
soundex(text) returns text
difference(text, text) returns int
```

soundex函数将字符串转换为其 Soundex 代码。difference函数将两个字符串转换为各自的 Soundex 代码，然后返回代码中相同位置上匹配的个数。由于 Soundex 代码有四个字符，因此结果范围为 0 到 4，其中 0 表示没有匹配，4 表示完全匹配。（因此，这个函数的命名并不贴切，similarity 会是更好的名字。）

下面是一些用法示例：

```
SELECT soundex('hello world!');

SELECT soundex('Anne'), soundex('Ann'), difference('Anne', 'Ann');
SELECT soundex('Anne'), soundex('Andrew'), difference('Anne',
 'Andrew');
SELECT soundex('Anne'), soundex('Margaret'), difference('Anne',
 'Margaret');

CREATE TABLE s (nm text);

INSERT INTO s VALUES ('john');
INSERT INTO s VALUES ('joan');
INSERT INTO s VALUES ('wobbly');
INSERT INTO s VALUES ('jack');

SELECT * FROM s WHERE soundex(nm) = soundex('john');

SELECT * FROM s WHERE difference(s.nm, 'john') > 2;
```

F.15.2. Levenshtein

该函数计算两个字符串之间的 Levenshtein 距离：

```
levenshtein(text source, text target, int ins_cost, int del_cost,
 int sub_cost) returns int
levenshtein(text source, text target) returns int
levenshtein_less_equal(text source, text target, int ins_cost, int
 del_cost, int sub_cost, int max_d) returns int
```

```
levenshtein_less_equal(text source, text target, int max_d) returns
 int
```

source 和 target 都可以是任意非空字符串，最大长度为 255 字节。代价参数分别指定字符插入、删除或替换的代价。可以像该函数的第二种形式那样省略代价参数；
此时它们都默认为 1。levenshtein_less_equal 是 Levenshtein 函数在距离值较小时的加速版本。如果实际距离小于等于 max_d，levenshtein_less_equal 会返回准确的距离值；否则该函数返回某个大于 max_d 的值。

示例：

```
test=# SELECT levenshtein('GUMBO', 'GAMBOL');
 levenshtein
-------------
           2
(1 row)

test=# SELECT levenshtein('GUMBO', 'GAMBOL', 2,1,1);
 levenshtein
-------------
           3
(1 row)

test=# SELECT levenshtein_less_equal('extensive', 'exhaustive',2);
 levenshtein_less_equal
------------------------
                      3
(1 row)

test=# SELECT levenshtein_less_equal('extensive', 'exhaustive',4);
 levenshtein_less_equal
------------------------
                      4
(1 row)
```

F.15.3. Metaphone

Metaphone 与 Soundex 一样，基于为输入字符串构造一个代表性代码的思想。如果两个字符串具有相同的代码，则认为它们相似。

该函数计算输入字符串的 Metaphone 代码：

```
metaphone(text source, int max_output_length) returns text
```

source 必须是非空字符串，最大长度为 255 字节。max_output_length 设置输出 Metaphone 代码的最大长度；如果超过该长度，输出会被截断为该长度。

示例：

```
test=# SELECT metaphone('GUMBO', 4);
 metaphone
-----------
 KM
(1 row)
```

F.15.4. Double Metaphone

Double Metaphone 系统会为给定输入字符串计算两个“近音”代码，即一个“主”代码和一个“备选”代码。在大多数情况下，它们相同，但尤其对于非英语名字，
这两个代码可能会因发音不同而略有差异。这些函数分别计算主代码和备选代码：

```
dmetaphone(text source) returns text
dmetaphone_alt(text source) returns text
```

输入字符串没有长度限制。

示例：

```
test=# select dmetaphone('gumbo');
 dmetaphone
------------
 KMP
(1 row)
```

F.16. hstore

This module implements the `hstore` data type for storing sets of key/value pairs within a single PostgreSQL value. This can be useful in various scenarios, such as rows with many attributes that are rarely examined, or semi-structured data. Keys and values are simply text strings.

F.16.1. `hstore` External Representation

The text representation of an `hstore`, used for input and output, includes zero or more *`key`* `=>` *`value`* pairs separated by commas. Some examples:

```
k => v
foo => bar, baz => whatever
"1-a" => "anything at all"
```

The order of the pairs is not significant (and may not be reproduced on output). Whitespace between pairs or around the `=>` sign is ignored. Double-quote keys and values that include whitespace, commas, `=`s or `>`s. To include a double quote or a backslash in a key or value, escape it with a backslash.

Each key in an `hstore` is unique. If you declare an `hstore` with duplicate keys, only one will be stored in the `hstore` and there is no guarantee as to which will be kept:

```
SELECT 'a=>1,a=>2'::hstore;
  hstore
----------
 "a"=>"1"
```

A value (but not a key) can be an SQL `NULL`. For example:

```
key => NULL
```

The `NULL` keyword is case-insensitive. Double-quote the `NULL` to treat it as the ordinary string “NULL”.

> **注意**
>
> Keep in mind that the `hstore` text format, when used for input, applies before any required quoting or escaping. If you are passing an `hstore` literal via a parameter, then no additional processing is needed. But if you're passing it as a quoted literal constant, then any single-quote characters and (depending on the setting of the `standard_conforming_strings` configuration parameter) backslash characters need to be escaped correctly. See 第 4.1.2.1 节 for more on the handling of string constants.

On output, double quotes always surround keys and values, even when it's not strictly necessary.

F.16.2. `hstore` Operators and Functions

The operators provided by the `hstore` module are shown in 表 F.8, the functions in 表 F.9.

表 F.8. hstore Operators

Operator	Description	Example	Result
hstore -> text	get value for key (NULL if not present)	'a=>x, b=>y'::hstore -> 'a'	x
hstore -> text[]	get values for keys (NULL if not present)	'a=>x, b=>y, c=>z'::hstore -> ARRAY['c','a']	{"z","x"}
hstore || hstore	concatenate hstores	'a=>b, c=>d'::hstore || 'c=>x, d=>q'::hstore	"a"=>"b", "c"=>"x", "d"=>"q"
hstore ? text	does hstore contain key?	'a=>1'::hstore ? 'a'	t
hstore ?& text[]	does hstore contain all specified keys?	'a=>1,b=>2'::hstore ?& ARRAY['a','b']	t
hstore ?| text[]	does hstore contain any of the specified keys?	'a=>1,b=>2'::hstore ?| ARRAY['b','c']	t
hstore @> hstore	does left operand contain right?	'a=>b, b=>1, c=>NULL'::hstore @> 'b=>1'	t
hstore <@ hstore	is left operand contained in right?	'a=>c'::hstore <@ 'a=>b, b=>1, c=>NULL'	f
hstore - text	delete key from left operand	'a=>1, b=>2, c=>3'::hstore - 'b'::text	"a"=>"1", "c"=>"3"
hstore - text[]	delete keys from left operand	'a=>1, b=>2, c=>3'::hstore - ARRAY['a','b']	"c"=>"3"
hstore - hstore	delete matching pairs from left operand	'a=>1, b=>2, c=>3'::hstore - 'a=>4, b=>2'::hstore	"a"=>"1", "c"=>"3"
record #= hstore	replace fields in record with matching values from hstore	see Examples section	
%% hstore	convert hstore to array of alternating keys and values	%% 'a=>foo, b=>bar'::hstore	{a,foo,b,bar}
%# hstore	convert hstore to two-dimensional key/value array	%# 'a=>foo, b=>bar'::hstore	{{a,foo},{b,bar}}

注意

Prior to PostgreSQL 8.2, the containment operators @> and <@ were called @ and ~, respectively. These names are still available, but are deprecated and will eventually be removed. Notice that the old names are reversed from the convention formerly followed by the core geometric data types!

表 F.9. hstore Functions

Function	Return Type	Description	Example	Result
hstore(record)	hstore	construct an hstore from a record or row	hstore(ROW(1, 2))	f1=>1,f2=>2
hstore(text[])	hstore	construct an hstore from an array, which may be either a key/value array, or a two-dimensional array	hstore(ARRAY['a','1','b','2']) || hstore(ARRAY[['c','3'],['d','4']])	a=>1, b=>2, c=>3, d=>4
hstore(text[], text[])	hstore	construct an hstore from separate key and value arrays	hstore(ARRAY['a','b'], ARRAY['1','2'])	"a"=>"1","b"=>"2"
hstore(text, text)	hstore	make single-item hstore	hstore('a', 'b')	"a"=>"b"
akeys(hstore)	text[]	get hstore's keys as an array	akeys('a=>1,b=>2')	{a,b}
skeys(hstore)	setof text	get hstore's keys as a set	skeys('a=>1,b=>2')	a b
avals(hstore)	text[]	get hstore's values as an array	avals('a=>1,b=>2')	{1,2}
svals(hstore)	setof text	get hstore's values as a set	svals('a=>1,b=>2')	1 2
hstore_to_array(hstore)	text[]	get hstore's keys and values as an array of alternating keys and values	hstore_to_array('a=>1,b=>2')	{a,1,b,2}
hstore_to_matrix(hstore)	text[]	get hstore's keys and values as a two-dimensional array	hstore_to_matrix('a=>1,b=>2')	{{a,1},{b,2}}
slice(hstore, text[])	hstore	extract a subset of an hstore	slice('a=>1,b=>2,c=>3'::hstore, ARRAY['b','c','x'])	"b"=>"2", "c"=>"3"
each(hstore)	setof(key text, value text)	get hstore's keys and values as a set	select * from each('a=>1,b=>2')	key | value -----+------- a | 1 b | 2
exist(hstore, text)	boolean	does hstore contain key?	exist('a=>1','a')	t
defined(hstore,text)	boolean	does hstore contain non-NULL value for key?	defined('a=>NULL','a')	f
delete(hstore, text)	hstore	delete pair with matching key	delete('a=>1,b=>2','b')	"a"=>"1"
delete(hstore, text[])	hstore	delete pairs with matching keys	delete('a=>1,b=>2,c=>3',	"c"=>"3"

Function	Return Type	Description	Example	Result
			`ARRAY['a','b']`)	
`delete(hstore, hstore)`	`hstore`	delete pairs matching those in the second argument	`delete('a=>1, b=>2','a=>4,b=>2'::hstore)`	`"a"=>"1"`
`populate_record(record, hstore)`	`record`	replace fields in `record` with matching values from `hstore`	see Examples section	

注意

The function `populate_record` is actually declared with `anyelement`, not `record`, as its first argument, but it will reject non-record types with a run-time error.

F.16.3. Indexes

`hstore` has GiST and GIN index support for the `@>`, `?`, `?&` and `?|` operators. For example:

```
CREATE INDEX hidx ON testhstore USING GIST (h);

CREATE INDEX hidx ON testhstore USING GIN (h);
```

`hstore` also supports `btree` or `hash` indexes for the `=` operator. This allows `hstore` columns to be declared `UNIQUE`, or to be used in `GROUP BY`, `ORDER BY` or `DISTINCT` expressions. The sort ordering for `hstore` values is not particularly useful, but these indexes may be useful for equivalence lookups. Create indexes for `=` comparisons as follows:

```
CREATE INDEX hidx ON testhstore USING BTREE (h);

CREATE INDEX hidx ON testhstore USING HASH (h);
```

F.16.4. Examples

Add a key, or update an existing key with a new value:

```
UPDATE tab SET h = h || hstore('c', '3');
```

Delete a key:

```
UPDATE tab SET h = delete(h, 'k1');
```

Convert a `record` to an `hstore`:

```
CREATE TABLE test (col1 integer, col2 text, col3 text);
INSERT INTO test VALUES (123, 'foo', 'bar');

SELECT hstore(t) FROM test AS t;
                   hstore
--------------------------------------------
 "col1"=>"123", "col2"=>"foo", "col3"=>"bar"
(1 row)
```

Convert an `hstore` to a predefined `record` type:

```
CREATE TABLE test (col1 integer, col2 text, col3 text);

SELECT * FROM populate_record(null::test,
                              '"col1"=>"456", "col2"=>"zzz"');
 col1 | col2 | col3
------+------+------
  456 | zzz  |
(1 row)
```

Modify an existing record using the values from an `hstore`:

```
CREATE TABLE test (col1 integer, col2 text, col3 text);
INSERT INTO test VALUES (123, 'foo', 'bar');

SELECT (r).* FROM (SELECT t #= '"col3"=>"baz"' AS r FROM test t) s;
 col1 | col2 | col3
------+------+------
  123 | foo  | baz
(1 row)
```

F.16.5. Statistics

The `hstore` type, because of its intrinsic liberality, could contain a lot of different keys. Checking for valid keys is the task of the application. The following examples demonstrate several techniques for checking keys and obtaining statistics.

Simple example:

```
SELECT * FROM each('aaa=>bq, b=>NULL, ""=>1');
```

Using a table:

```
SELECT (each(h)).key, (each(h)).value INTO stat FROM testhstore;
```

Online statistics:

```
SELECT key, count(*) FROM
  (SELECT (each(h)).key FROM testhstore) AS stat
  GROUP BY key
  ORDER BY count DESC, key;
    key    | count
-----------+-------
 line      |   883
 query     |   207
 pos       |   203
 node      |   202
 space     |   197
 status    |   195
 public    |   194
 title     |   190
 org       |   189
...................
```

F.16.6. Compatibility

As of PostgreSQL 9.0, `hstore` uses a different internal representation than previous versions. This presents no obstacle for dump/restore upgrades since the text representation (used in the dump) is unchanged.

In the event of a binary upgrade, upward compatibility is maintained by having the new code recognize old-format data. This will entail a slight performance penalty when processing data that has not yet been modified by the new code. It is possible to force an upgrade of all values in a table column by doing an `UPDATE` statement as follows:

```
UPDATE tablename SET hstorecol = hstorecol || '';
```

Another way to do it is:

```
ALTER TABLE tablename ALTER hstorecol TYPE hstore USING hstorecol |
| '';
```

The `ALTER TABLE` method requires an exclusive lock on the table, but does not result in bloating the table with old row versions.

F.16.7. Authors

Oleg Bartunov `<oleg@sai.msu.su>`, Moscow, Moscow University, Russia

Teodor Sigaev `<teodor@sigaev.ru>`, Moscow, Delta-Soft Ltd., Russia

Additional enhancements by Andrew Gierth `<andrew@tao11.riddles.org.uk>`, United Kingdom

F.17. intagg – 整数聚合器和枚举器

intagg 模块提供一个整数聚合器和一个枚举器。由于已经有内置函数提供了其能力的超集，所以 intagg 现已过时。不过，该模块仍作为这些内置函数的兼容性包装器提供。

F.17.1. 函数

聚合器是聚合函数int_array_aggregate(integer)，它会生成一个整数数组，其中恰好包含输入给它的那些整数。这是对array_agg的包装器，后者对任意数组类型都能做同样的事情。

枚举器是函数int_array_enum(integer[])，它返回setof integer。本质上，它是聚合器的逆操作：给定一个整数数组，将其展开为一组行。这是对unnest的包装器，后者对任意数组类型都能做同样的事情。

F.17.2. 使用示例

许多数据库系统有一对多表的概念。这种表通常位于两个带索引的表之间，例如：

```
CREATE TABLE left (id INT PRIMARY KEY, ...);
CREATE TABLE right (id INT PRIMARY KEY, ...);
CREATE TABLE one_to_many(left INT REFERENCES left, right INT
 REFERENCES right);
```

它通常按如下方式使用：

```
SELECT right.* from right JOIN one_to_many ON (right.id = one_to_
many.right)
  WHERE one_to_many.left = item;
```

这将返回左侧表中某个条目在右侧表中对应的所有项。这是 SQL 中非常常见的一种构造。

不过，如果one_to_many表中的条目非常多，这种方法就会变得相当繁琐。通常，对于左侧表中的某个特定条目，右侧表中的每个对应项都要执行一次索引扫描并取回一行。如果你的系统非常动态，那就没有太多办法。不过，如果有一部分数据相当静态，可以借助聚合器创建一个汇总表。

```
CREATE TABLE summary AS
  SELECT left, int_array_aggregate(right) AS right
  FROM one_to_many
  GROUP BY left;
```

这样会创建一个表，其中左侧每个条目对应一行，并附带一个由右侧各项组成的数组。不过，如果没有某种使用该数组的方法，它就几乎没有用处；这正是数组枚举器存在的原因。你可以这样做：

```
SELECT left, int_array_enum(right) FROM summary WHERE left = item;
```

上面这个使用int_array_enum的查询，产生的结果与下面的查询相同：

```
SELECT left, right FROM one_to_many WHERE left = item;
```

区别在于，针对汇总表的查询只需要从表中取出一行，而直接查询one_to_many则必须对每个对应项都进行索引扫描并取回一行。

在某个系统上，EXPLAIN显示，某个查询的代价从 8488 降到了 329。原始查询是一个涉及one_to_many表的连接，后来被替换为：

```
SELECT right, count(right) FROM
```

```
 ( SELECT left, int_array_enum(right) AS right
   FROM summary JOIN (SELECT left FROM left_table WHERE left =
item) AS lefts
        ON (summary.left = lefts.left)
 ) AS list
 GROUP BY right
 ORDER BY count DESC;
```

F.18. intarray

intarray模块提供了一些有用的函数和操作符，用于操作不含 NULL 的整数数组。它还支持使用其中某些操作符进行索引搜索。

如果所提供的数组中包含任何 NULL 元素，所有这些操作都会抛出错误。

这些操作中有许多只对一维数组才有意义。尽管它们也接受更高维度的输入数组，但数据会被视为按存储顺序排列的线性数组。

F.18.1. intarray 函数和操作符

intarray模块提供的函数列在表 F.10中，操作符列在表 F.11中。

表 F.10. intarray 函数

函数	返回类型	描述	示例	结果
icount(int[])	int	数组中的元素个数	icount('{1,2,3}'::int[])	3
sort(int[], text dir)	int[]	对数组排序 – *dir* 必须为asc或desc	sort('{1,2,3}'::int[], 'desc')	{3,2,1}
sort(int[])	int[]	按升序排序	sort(array[11,77,44])	{11,44,77}
sort_asc(int[])	int[]	按升序排序		
sort_desc(int[])	int[]	按降序排序		
uniq(int[])	int[]	删除相邻的重复元素	uniq(sort('{1,2,3,2,1}'::int[]))	{1,2,3}
idx(int[], int item)	int	与*item*匹配的第一个元素的索引（如果没有则为 0）	idx(array[11,22,33,22,11], 22)	2
subarray(int[], int start, int len)	int[]	从位置*start*开始的数组部分，共*len*个元素	subarray('{1,2,3,2,1}'::int[], 2, 3)	{2,3,2}
subarray(int[], int start)	int[]	从位置*start*开始的数组部分	subarray('{1,2,3,2,1}'::int[], 2)	{2,3,2,1}
intset(int)	int[]	创建一个只含一个元素的数组	intset(42)	{42}

表 F.11. intarray 操作符

操作符	返回值	描述
int[] && int[]	boolean	重叠 – 如果数组至少有一个共同元素，则为true
int[] @> int[]	boolean	包含 – 如果左数组包含右数组，则为true
int[] <@ int[]	boolean	包含于 – 如果左数组包含在右数组中，则为true
# int[]	int	数组中的元素个数
int[] # int	int	索引（与idx函数相同）

操作符	返回值	描述
int[] + int	int[]	将元素压入数组（添加到数组末尾）
int[] + int[]	int[]	数组连接（右数组添加到左数组末尾）
int[] - int	int[]	从数组中移除与右参数匹配的项
int[] - int[]	int[]	从左数组中移除右数组中的元素
int[] | int	int[]	参数的并集
int[] | int[]	int[]	数组的并集
int[] & int[]	int[]	数组的交集
int[] @@ query_int	boolean	如果数组满足查询（见下文），则为true
query_int ~~ int[]	boolean	如果数组满足查询（@@的交换操作符），则为true

（在 PostgreSQL 8.2 之前，包含操作符 @> 和 <@ 分别称为 @ 和 ~。这些名称仍然可用，但已弃用，最终将被删除。请注意，旧名称与核心几何数据类型以前采用的约定正好相反！）

操作符&&、@>和 <@等价于PostgreSQL内置的同名操作符，不同之处在于它们只适用于不包含 NULL 的整数数组，而内置操作符适用于任何数组类型。这一限制使它们在很多情况下比内置操作符更快。

@@和~~操作符用于测试数组是否满足某个查询，该查询表示为专用数据类型query_int 的一个值。一个查询由若干整数值组成，这些值会与数组元素进行检查，并且可通过操作符&（AND）、|（OR）和!（NOT）组合起来。必要时可以使用括号。例如，查询1&(2|3)可匹配包含 1 且还包含 2 或 3 之一的数组。

F.18.2. 索引支持

intarray为&&、@>、<@和@@操作符以及常规数组相等运算提供索引支持。

提供了两个 GiST 索引操作符类：gist__int_ops（默认使用）适用于小型到中型数据集，而gist__intbig_ops使用更大的签名，
更适合为大型数据集建立索引（即包含大量不同数组值的列）。
实现使用带有内置有损压缩的 RD 树数据结构。

另有一个非默认的 GIN 操作符类gin__int_ops，支持相同的操作符。

作为经验法则，GIN 索引的搜索速度快于 GiST 索引，但构建或更新较慢；因此 GIN 更适合静态数据，而 GiST 更适合经常更新的数据。

F.18.3. 示例

```
-- a message can be in one or more “sections”
CREATE TABLE message (mid INT PRIMARY KEY, sections INT[], ...);

-- create specialized index
CREATE INDEX message_rdtree_idx ON message USING GIST (sections
 gist__int_ops);

-- select messages in section 1 OR 2 - OVERLAP operator
SELECT message.mid FROM message WHERE message.sections && '{1,2}';

-- select messages in sections 1 AND 2 - CONTAINS operator
SELECT message.mid FROM message WHERE message.sections @> '{1,2}';
```

```
-- the same, using QUERY operator
SELECT message.mid FROM message WHERE message.sections @@ '1&2'::
query_int;
```

F.18.4. 基准测试

源代码目录contrib/intarray/bench包含一个基准测试套件。运行方式如下：

```
cd .../bench
createdb TEST
psql TEST < ../_int.sql
./create_test.pl | psql TEST
./bench.pl
```

bench.pl脚本有很多选项，在不带任何参数运行时会显示这些选项。

F.18.5. 作者

所有工作都由 Teodor Sigaev（<teodor@sigaev.ru>）和 Oleg Bartunov（<oleg@sai.msu.su>）完成。更多信息请见http://www.sai.msu.su/~megera/postgres/gist/。Andrey Oktyabrski 在添加新函数和新操作方面也做出了很大贡献。

F.19. isn

isn模块为以下国际产品编号标准提供数据类型：EAN13、UPC、ISBN（图书）、ISMN（音乐）和 ISSN（连续出版物）。输入这些编号时，会依据一份硬编码的前缀列表进行校验；输出时，这份前缀列表也会用于给编号加上连字符。由于新的前缀会不时被分配，这份前缀列表可能已经过时。希望该模块未来版本能从一个或多个表中获取前缀列表，以便用户按需更新；但目前，这份列表只能通过修改源代码并重新编译来更新。或者，该模块未来版本也可能取消前缀校验和连字符支持。

F.19.1. 数据类型

表 F.12显示了isn模块提供的数据类型。

表 F.12. isn 数据类型

数据类型	描述
EAN13	欧洲商品编号，始终以 EAN13 显示格式显示
ISBN13	以新的 EAN13 显示格式显示的国际标准图书编号
ISMN13	以新的 EAN13 显示格式显示的国际标准音乐编号
ISSN13	以新的 EAN13 显示格式显示的国际标准连续出版物编号
ISBN	以旧的短显示格式显示的国际标准图书编号
ISMN	以旧的短显示格式显示的国际标准音乐编号
ISSN	以旧的短显示格式显示的国际标准连续出版物编号
UPC	通用产品代码

几点说明：

1. ISBN13、ISMN13 和 ISSN13 编号都是 EAN13 编号。
2. EAN13 编号并不总是 ISBN13、ISMN13 或 ISSN13（不过其中有些确实是）。
3. 某些 ISBN13 编号可以显示为 ISBN。
4. 某些 ISMN13 编号可以显示为 ISMN。
5. 某些 ISSN13 编号可以显示为 ISSN。
6. UPC 编号是 EAN13 编号的一个子集（它们基本上就是去掉首位 0 的 EAN13 编号）。
7. 所有 UPC、ISBN、ISMN 和 ISSN 编号都可以表示为 EAN13 编号。

在内部，所有这些类型都使用同一种表示形式（一个 64 位整数），并且彼此可以互换。之所以提供多种类型，是为了控制显示格式，
并对原本应表示某一特定类型编号的输入执行更严格的有效性检查。

只要可能，ISBN、ISMN 和 ISSN 类型都会显示编号的短版本（ISxN 10）；对于不能用短版本表示的编号，则显示为 ISxN 13 格式。EAN13、ISBN13、ISMN13 和 ISSN13 类型则始终显示 ISxN 的长版本（EAN13）。

F.19.2. 类型转换

isn模块提供以下几对类型转换：

- ISBN13 <=> EAN13

- ISMN13 <=> EAN13
- ISSN13 <=> EAN13
- ISBN <=> EAN13
- ISMN <=> EAN13
- ISSN <=> EAN13
- UPC <=> EAN13
- ISBN <=> ISBN13
- ISMN <=> ISMN13
- ISSN <=> ISSN13

当从EAN13转换到其他类型时，会在运行时检查该值是否落在另一类型的取值范围内；如果不在，就会抛出错误。其他类型转换都只是简单地重新标记，因此总会成功。

F.19.3. 函数和操作符

isn模块提供标准比较操作符，以及对所有这些数据类型的 B-树和哈希索引支持。此外，它还提供了一些专用函数，如表 F.13所示。在该表中，isn表示该模块提供的任意一种数据类型。

表 F.13. isn 函数

函数	返回值	描述
isn_weak(boolean)	boolean	设置弱输入模式（返回新的设置值）
isn_weak()	boolean	获取弱模式的当前状态
make_valid(isn)	isn	使无效编号变为有效（清除无效标记）
is_valid(isn)	boolean	检查是否存在无效标记

弱模式用于允许向表中插入无效数据。这里的“无效”指的是校验位错误，而不是缺少数字。

为什么会需要使用弱模式呢？例如，手头可能有一大批 ISBN 编号，数量多到难免会有一些编号因为某些奇怪的原因而带有错误的校验位（也许这些编号是从印刷清单扫描得到的，而 OCR 把数字识别错了；也许这些编号是人工录入的.....谁知道呢）。总之，可能想把这些混乱情况清理干净，但同时仍希望先把所有编号都装入数据库，并借助外部工具在数据库中定位无效编号，以便核对信息并更容易完成校验；例如，可能会想把表中所有无效编号都查询出来。

当在弱模式下向表中插入无效编号时，实际插入的是校验位已更正的编号，但显示时会在末尾附加一个感叹号（!），例如0-11-000322-5!。可以用is_valid函数检查这个无效标记，并用make_valid函数清除它。

即使未启用弱模式，也可以通过在编号末尾附加!字符来强制插入无效编号。

另一个特殊功能是，在输入时可以用?代替校验位，系统会自动插入正确的校验位。

F.19.4. 示例

```
--Using the types directly:
SELECT isbn('978-0-393-04002-9');
```

```
SELECT isbn13('0901690546');
SELECT issn('1436-4522');

--Casting types:
-- note that you can only cast from ean13 to another type when the
-- number would be valid in the realm of the target type;
-- thus, the following will NOT work: select isbn(ean13(
'0220356483481'));
-- but these will:
SELECT upc(ean13('0220356483481'));
SELECT ean13(upc('220356483481'));

--Create a table with a single column to hold ISBN numbers:
CREATE TABLE test (id isbn);
INSERT INTO test VALUES('9780393040029');

--Automatically calculate check digits (observe the '?'):
INSERT INTO test VALUES('220500896?');
INSERT INTO test VALUES('978055215372?');

SELECT issn('3251231?');
SELECT ismn('979047213542?');

--Using the weak mode:
SELECT isn_weak(true);
INSERT INTO test VALUES('978-0-11-000533-4');
INSERT INTO test VALUES('9780141219307');
INSERT INTO test VALUES('2-205-00876-X');
SELECT isn_weak(false);

SELECT id FROM test WHERE NOT is_valid(id);
UPDATE test SET id = make_valid(id) WHERE id = '2-205-00876-X!';

SELECT * FROM test;

SELECT isbn13(id) FROM test;
```

F.19.5. 参考文献

实现该模块所需的信息收集自若干网站，包括：

- http://www.isbn-international.org/
- http://www.issn.org/
- http://www.ismn-international.org/
- http://www.wikipedia.org/

用于进行连字符分隔的前缀还整理自：

- http://www.gs1.org/productssolutions/idkeys/support/prefix_list.html
- http://www.isbn-international.org/en/identifiers.html
- http://www.ismn-international.org/ranges.html

在创建这些算法时已十分谨慎，并依据官方 ISBN、ISMN、ISSN 用户手册中建议的算法进行了细致核验。

F.19.6. 作者

Germán Méndez Bravo (Kronuz), 2004 - 2006

该模块受到了 Garrett A. Wollman 的`isbn_issn`代码的启发。

F.20. lo – 管理大对象

`lo`模块为管理大对象（也称为 LO 或 BLOB）提供支持，其中包括数据类型`lo`和触发器`lo_manage`。

F.20.1. 原理

JDBC 驱动的一个问题是（ODBC 驱动也受此影响），其规范假定对 BLOB（二进制大对象）的引用存储在表内，并且如果该表项被更改，相关的 BLOB 就会从数据库中删除。

但在目前的PostgreSQL中并非如此。大对象被视为独立对象；表中的一个项可以通过 OID 引用某个大对象，但也可能有多个表项引用同一个大对象 OID，因此系统不会仅因你更改或删除了其中一个表项就删除该大对象。

这对于PostgreSQL专用应用没有问题，但使用 JDBC 或 ODBC 的标准代码不会删除这些对象，从而产生孤立对象，也就是不再被任何内容引用、只是占用磁盘空间的对象。

`lo`模块允许通过把触发器附加到包含 LO 引用列的表上来解决这个问题。该触发器本质上就是在你删除或修改引用大对象的值时调用`lo_unlink`。使用这个触发器时，
你实际上是假定：凡是出现在受该触发器控制的列中的大对象，
在数据库中都只有一个引用！

该模块还提供了数据类型`lo`，它实际上只是`oid`类型的一个域。
这有助于区分保存大对象引用的数据库列与保存其他对象 OID 的列。使用该触发器并不要求必须使用`lo`类型，但用它来标识数据库中哪些列表示由该触发器管理的大对象，
可能会更方便。还有传言说，如果 BLOB 列不用`lo`，ODBC 驱动会感到困惑。

F.20.2. 如何使用

下面是一个简单的用法示例：

```
CREATE TABLE image (title TEXT, raster lo);

CREATE TRIGGER t_raster BEFORE UPDATE OR DELETE ON image
    FOR EACH ROW EXECUTE PROCEDURE lo_manage(raster);
```

对于每个将保存指向大对象的唯一引用的列，创建一个`BEFORE UPDATE OR DELETE`触发器，并将该列名作为唯一的触发器参数。如果同一张表中需要多个`lo`列，
就为每一列分别创建一个触发器，并记得为同一张表上的每个触发器指定不同的名称。

F.20.3. 限制

- 删除表时，其中包含的任何对象仍会变成孤立对象，因为这种情况下不会执行触发器。
  为避免这种情况，可以先执行`DELETE FROM *table*`，再执行`DROP TABLE`。

  `TRUNCATE`也有同样的风险。

  如果你已经有，或者怀疑有，孤立的大对象，请参见vacuumlo模块来帮助清理它们。
  偶尔运行vacuumlo，作为对`lo_manage`触发器的补充保障，是个不错的主意。

- 某些前端可能会创建自己的表，但不会创建相应的触发器。此外，
  用户也可能不记得（或根本不知道）要创建这些触发器。

F.20.4. 作者

Peter Mount `<peter@retep.org.uk>`

F.21. ltree

该模块实现了数据类型 `ltree`，用于表示存储在层次化树状结构中的数据标签。它还提供了丰富的标签树搜索能力。

F.21.1. 定义

标签是由字母数字字符和下划线组成的序列（例如，在 C 区域设置下，允许的字符为 `A-Za-z0-9_`）。标签长度必须少于 256 字节。

示例：`42`、`Personal_Services`

标签路径是由点号分隔的零个或多个标签组成的序列，例如 `L1.L2.L3`，表示从层次树根节点到某个特定节点的一条路径。标签路径的长度必须小于 65Kb，但最好保持在 2Kb 以下。在实践中这并不是一个很大的限制；例如，DMOZ 目录（http://www.dmoz.org）中最长的标签路径约为 240 字节。

示例：`Top.Countries.Europe.Russia`

`ltree`模块提供了几种数据类型：

- `ltree`存储一个标签路径。
- `lquery`表示一种用于匹配`ltree`值的、类似正则表达式的模式。
  一个简单单词会匹配路径中的相应标签。星号（`*`）匹配零个或多个标签。例如：

`foo`	精确匹配标签路径*foo*
`*.foo.*`	匹配任何包含标签*foo*的标签路径
`*.foo`	匹配最后一个标签为*foo*的任意标签路径

  星号还可以带量词，以限制它们能够匹配的标签数量：

`*{n}`	精确匹配 *n* 个标签
`*{n,}`	至少匹配 *n* 个标签
`*{n,m}`	至少匹配 *n* 个、但不超过 *m* 个标签
`*{,m}`	至多匹配 *m* 个标签 — 与下式相同： `*{0,m}`

  在`lquery`中，有几个修饰符可以放在非星号标签的末尾，使其不只匹配完全相同的标签：

`@`	不区分大小写地匹配，例如 *a@* 可匹配 *A*
`*`	匹配以此前缀开头的任意标签，例如 *foo** 可匹配 *foobar*
`%`	匹配标签起始处由下划线分隔的单词

  修饰符`%`的行为稍微复杂一些。它尝试匹配单词，而不是整个标签。例如，`foo_bar%`可匹配`foo_bar_baz`，但不能匹配`foo_barbaz`。如果与`*`组合使用，则前缀匹配会分别作用于每个单词，例如`foo_bar%*`可匹配`foo1_bar2_baz`，但不能匹配`foo1_br2_baz`。

  此外，还可以写出多个可能带修饰符的非星号项，并用`|`（OR）分隔，以匹配其中任意一项；也可以在非星号组前加上`!`（NOT），以匹配不符合这些备选项中任意一项的标签。

  下面是一个带注释的示例，使用`lquery`：

  ```
  Top.*{0,2}.sport*@.!football|tennis.Russ*|Spain
  a.  b.     c.      d.                e.
  ```

  此查询将匹配满足以下条件的任意标签路径：

a. 以标签Top开头

b. 接下来，在下一个条件之前有零到两个标签

c. 然后是一个以前缀sport开头的标签，且匹配时不区分大小写

d. 接着有一个不匹配football或tennis的标签

e. 最后以一个以Russ开头的标签，或精确匹配Spain的标签结束。

- ltxtquery表示一种用于匹配ltree值的、类似全文检索的模式。一个ltxtquery值包含单词，末尾还可以带有修饰符@、*、%；这些修饰符与它们在lquery中的含义相同。单词可以通过&（AND）、|（OR）、!（NOT）以及圆括号组合。它与lquery的关键区别在于，ltxtquery匹配单词时不考虑它们在标签路径中的位置。

  下面是一个ltxtquery示例：

  ```
  Europe & Russia*@ & !Transportation
  ```

  它将匹配包含标签Europe以及任意以Russia开头（不区分大小写）的标签的路径，但不匹配包含标签Transportation的路径。这些单词在路径中的位置并不重要。另外，当使用%时，该单词可以匹配标签中任意由下划线分隔的单词，而不考虑其位置。

注意：ltxtquery允许在符号之间出现空白，而ltree和lquery不允许。

F.21.2. 操作符和函数

类型ltree具有常见的比较操作符 =、<>、<、>、<=、>=。比较时采用树遍历顺序，其中节点的子节点按标签文本排序。此外，还提供了表 F.14中所示的专用操作符。

表 F.14. ltree 操作符

操作符	返回值	描述
ltree @> ltree	boolean	左参数是否为右参数的祖先（或与之相等）？
ltree <@ ltree	boolean	左参数是否为右参数的后代（或与之相等）？
ltree ~ lquery	boolean	ltree是否匹配lquery?
lquery ~ ltree	boolean	ltree是否匹配lquery?
ltree ? lquery[]	boolean	ltree是否匹配数组中的任意lquery?
lquery[] ? ltree	boolean	ltree是否匹配数组中的任意lquery?
ltree @ ltxtquery	boolean	ltree是否匹配ltxtquery?
ltxtquery @ ltree	boolean	ltree是否匹配ltxtquery?
ltree || ltree	ltree	连接ltree路径
ltree || text	ltree	将文本转换为ltree后再连接
text || ltree	ltree	将文本转换为ltree后再连接
ltree[] @> ltree	boolean	数组是否包含ltree的某个祖先?
ltree <@ ltree[]	boolean	数组是否包含ltree的某个祖先?
ltree[] <@ ltree	boolean	数组是否包含ltree的某个后代?

操作符	返回值	描述
ltree @> ltree[]	boolean	数组是否包含ltree的某个后代？
ltree[] ~ lquery	boolean	数组是否包含匹配lquery的任意路径？
lquery ~ ltree[]	boolean	数组是否包含匹配lquery的任意路径？
ltree[] ? lquery[]	boolean	ltree数组是否包含匹配任意lquery的路径？
lquery[] ? ltree[]	boolean	ltree数组是否包含匹配任意lquery的路径？
ltree[] @ ltxtquery	boolean	数组是否包含匹配ltxtquery的任意路径？
ltxtquery @ ltree[]	boolean	数组是否包含匹配ltxtquery的任意路径？
ltree[] ?@> ltree	ltree	返回数组中第一个是ltree祖先的项，如果没有则返回 NULL
ltree[] ?<@ ltree	ltree	返回数组中第一个是ltree后代的项，如果没有则返回 NULL
ltree[] ?~ lquery	ltree	返回数组中第一个匹配lquery的项，如果没有则返回 NULL
ltree[] ?@ ltxtquery	ltree	返回数组中第一个匹配ltxtquery的项，如果没有则返回 NULL

操作符<@、@>、@和~都有对应的 ^<@、^@>、^@、^~变体，它们除了不使用索引之外完全相同。这些变体仅对测试有用。

可用函数见表 F.15。

表 F.15. ltree 函数

函数	返回类型	描述	示例	结果
subltree(ltree, int start, int end)	ltree	从位置*start*到位置*end*-1 的ltree子路径（从 0 开始计数）	subltree('Top.Child1.Child2',1,2)	Child1
subpath(ltree, int offset, int len)	ltree	从位置*offset*开始、长度为*len*的ltree子路径。如果*offset*为负，则子路径从距路径末尾 -offset 个标签处开始。如果*len*为负，则从路径末尾省去那么多个标签。	subpath('Top.Child1.Child2',0,2)	Top.Child1
subpath(ltree, int offset)	ltree	从位置*offset*开始、一直延伸到路径末尾的ltree子路径。如果*offset*为负，则子路径从距路径末尾 -offset 个标签处开始。	subpath('Top.Child1.Child2',1)	Child1.Child2
nlevel(ltree)	integer	路径中的标签数	nlevel('Top.Child1.Child2')	3

函数	返回类型	描述	示例	结果
index(ltree a, ltree b)	integer	*a* 中首次出现 *b* 的位置；如果未找到则为 -1	index('0.1.2.3.5.4.5.6.8.5.6.8','5.6')	6
index(ltree a, ltree b, int offset)	integer	从*offset*开始搜索时，*a* 中首次出现 *b* 的位置；负的*offset*表示从路径末尾向前-*offset*个标签处开始	index('0.1.2.3.5.4.5.6.8.5.6.8','5.6',-4)	9
text2ltree(text)	ltree	将text转换为ltree		
ltree2text(ltree)	text	将ltree转换为text		
lca(ltree, ltree, ...)	ltree	最低公共祖先，即路径的最长公共前缀（最多支持 8 个参数）	lca('1.2.2.3','1.2.3.4.5.6')	1.2
lca(ltree[])	ltree	最低公共祖先，即路径的最长公共前缀	lca(array['1.2.2.3'::ltree,'1.2.3'])	1.2

F.21.3. 索引

ltree支持几种能够加速所示操作符的索引类型：

- ltree上的 B-树索引：<、<=、=、>=、>
- ltree上的 GiST 索引：<、<=、=、>=、>、@>、<@、@、~、?

  创建此类索引的示例：

  ```
  CREATE INDEX path_gist_idx ON test USING GIST (path);
  ```

- ltree[]上的 GiST 索引：ltree[] <@ ltree、ltree @> ltree[]、@、~、?

  创建此类索引的示例：

  ```
  CREATE INDEX path_gist_idx ON test USING GIST (array_path);
  ```

  注意：这种索引类型是有损的。

F.21.4. 示例

本示例使用下列数据（在源代码发行包中的 contrib/ltree/ltreetest.sql文件里也能找到）：

```
CREATE TABLE test (path ltree);
INSERT INTO test VALUES ('Top');
INSERT INTO test VALUES ('Top.Science');
INSERT INTO test VALUES ('Top.Science.Astronomy');
INSERT INTO test VALUES ('Top.Science.Astronomy.Astrophysics');
INSERT INTO test VALUES ('Top.Science.Astronomy.Cosmology');
INSERT INTO test VALUES ('Top.Hobbies');
INSERT INTO test VALUES ('Top.Hobbies.Amateurs_Astronomy');
INSERT INTO test VALUES ('Top.Collections');
INSERT INTO test VALUES ('Top.Collections.Pictures');
```

```
INSERT INTO test VALUES ('Top.Collections.Pictures.Astronomy');
INSERT INTO test VALUES ('Top.Collections.Pictures.Astronomy.
Stars');
INSERT INTO test VALUES ('Top.Collections.Pictures.Astronomy.
Galaxies');
INSERT INTO test VALUES ('Top.Collections.Pictures.Astronomy.
Astronauts');
CREATE INDEX path_gist_idx ON test USING gist(path);
CREATE INDEX path_idx ON test USING btree(path);
```

现在，我们有一个表test，其中的数据描述了下图所示的层次结构：

```
                        Top
                     /   |  \
             Science Hobbies Collections
                 /       |              \
        Astronomy   Amateurs_Astronomy Pictures
           /  \                            |
Astrophysics  Cosmology                Astronomy
                                        /  |    \
                                 Galaxies Stars Astronauts
```

我们可以做继承查询：

```
ltreetest=> SELECT path FROM test WHERE path <@ 'Top.Science';
                path
-------------------------------------
 Top.Science
 Top.Science.Astronomy
 Top.Science.Astronomy.Astrophysics
 Top.Science.Astronomy.Cosmology
(4 rows)
```

下面是一些路径匹配的示例：

```
ltreetest=> SELECT path FROM test WHERE path ~ '*.Astronomy.*';
                     path
-----------------------------------------------
 Top.Science.Astronomy
 Top.Science.Astronomy.Astrophysics
 Top.Science.Astronomy.Cosmology
 Top.Collections.Pictures.Astronomy
 Top.Collections.Pictures.Astronomy.Stars
 Top.Collections.Pictures.Astronomy.Galaxies
 Top.Collections.Pictures.Astronomy.Astronauts
(7 rows)

ltreetest=> SELECT path FROM test WHERE path ~ '*.!pictures@.*.
Astronomy.*';
                path
-------------------------------------
 Top.Science.Astronomy
 Top.Science.Astronomy.Astrophysics
 Top.Science.Astronomy.Cosmology
(3 rows)
```

下面是一些全文检索示例：

```
ltreetest=> SELECT path FROM test WHERE path @ 'Astro*% & !
pictures@';
```

```
                path
-----------------------------------
 Top.Science.Astronomy
 Top.Science.Astronomy.Astrophysics
 Top.Science.Astronomy.Cosmology
 Top.Hobbies.Amateurs_Astronomy
(4 rows)

ltreetest=> SELECT path FROM test WHERE path @ 'Astro* & !
pictures@';
                path
-----------------------------------
 Top.Science.Astronomy
 Top.Science.Astronomy.Astrophysics
 Top.Science.Astronomy.Cosmology
(3 rows)
```

使用函数构造路径：

```
ltreetest=> SELECT subpath(path,0,2)||'Space'||subpath(path,2) FROM
 test WHERE path <@ 'Top.Science.Astronomy';
                 ?column?
-----------------------------------------
 Top.Science.Space.Astronomy
 Top.Science.Space.Astronomy.Astrophysics
 Top.Science.Space.Astronomy.Cosmology
(3 rows)
```

可以通过创建一个 SQL 函数，在路径的指定位置插入标签来简化这一操作：

```
CREATE FUNCTION ins_label(ltree, int, text) RETURNS ltree
    AS 'select subpath($1,0,$2) || $3 || subpath($1,$2);'
    LANGUAGE SQL IMMUTABLE;

ltreetest=> SELECT ins_label(path,2,'Space') FROM test WHERE path
 <@ 'Top.Science.Astronomy';
                ins_label
-----------------------------------------
 Top.Science.Space.Astronomy
 Top.Science.Space.Astronomy.Astrophysics
 Top.Science.Space.Astronomy.Cosmology
(3 rows)
```

F.21.5. 作者

全部工作均由 Teodor Sigaev（<teodor@stack.net>）和 Oleg Bartunov（<oleg@sai.msu.su>）完成。更多信息见http://www.sai.msu.su/~megera/postgres/gist/。作者谨感谢 Eugeny Rodichev 的有益讨论。欢迎提出意见和缺陷报告。

F.22. pageinspect

pageinspect模块提供了一组函数，允许你在底层检查数据库页的内容，这对调试很有帮助。所有这些函数都只能由超级用户使用。

F.22.1. 函数

get_raw_page(relname text, fork text, blkno int) returns bytea

get_raw_page读取指定关系的指定块，并以bytea值的形式返回其副本。这样就能获得该块在单一时刻的一致副本。*fork*应当为'main'（主数据分支）、'fsm'（空闲空间映射）、'vm'（可见性映射）或'init'（初始化分支）。

get_raw_page(relname text, blkno int) returns bytea

这是get_raw_page的简写形式，用于从主分支读取。等价于 get_raw_page(relname, 'main', blkno)

page_header(page bytea) returns record

page_header显示PostgreSQL所有堆页和索引页共有的字段。

应将页面映像（通过get_raw_page获得）作为参数传入。例如：

```
test=# SELECT * FROM page_header(get_raw_page('pg_class', 0));
    lsn    | tli | flags | lower | upper | special | pagesize |
 version | prune_xid
-----------+-----+-------+-------+-------+---------+----------+-
--------+-----------
 0/24A1B50 |   1 |     1 |   232 |   368 |    8192 |     8192 |
      4 |         0
```

返回的列对应于PageHeaderData结构体中的字段。详细信息见src/include/storage/bufpage.h。

heap_page_items(page bytea) returns setof record

heap_page_items显示堆页上的所有行指针。对于正在使用的行指针，还会显示元组头。无论这些元组在复制原始页时是否对某个 MVCC 快照可见，都会显示出来。

应将堆页面映像（通过get_raw_page获得）作为参数传入。例如：

```
test=# SELECT * FROM heap_page_items(get_raw_page('pg_class', 0)
);
```

有关返回字段的说明，参见src/include/storage/itemid.h和src/include/access/htup.h。

bt_metap(relname text) returns record

bt_metap返回 B-树索引元页的信息。例如：

```
test=# SELECT * FROM bt_metap('pg_cast_oid_index');
-[ RECORD 1 ]-----
magic     | 340322
version   | 2
root      | 1
level     | 0
```

```
fastroot  | 1
fastlevel | 0
```

`bt_page_stats(relname text, blkno int) returns record`

`bt_page_stats`返回 B-树索引单个页面的概要信息。例如：

```
test=# SELECT * FROM bt_page_stats('pg_cast_oid_index', 1);
-[ RECORD 1 ]-+-----
blkno         | 1
type          | l
live_items    | 256
dead_items    | 0
avg_item_size | 12
page_size     | 8192
free_size     | 4056
btpo_prev     | 0
btpo_next     | 0
btpo          | 0
btpo_flags    | 3
```

`bt_page_items(relname text, blkno int) returns setof record`

`bt_page_items`返回 B-树索引页中所有项的详细信息。例如：

```
test=# SELECT * FROM bt_page_items('pg_cast_oid_index', 1);
 itemoffset |  ctid   | itemlen | nulls | vars |    data
------------+---------+---------+-------+------+-------------
          1 | (0,1)   |      12 | f     | f    | 23 27 00 00
          2 | (0,2)   |      12 | f     | f    | 24 27 00 00
          3 | (0,3)   |      12 | f     | f    | 25 27 00 00
          4 | (0,4)   |      12 | f     | f    | 26 27 00 00
          5 | (0,5)   |      12 | f     | f    | 27 27 00 00
          6 | (0,6)   |      12 | f     | f    | 28 27 00 00
          7 | (0,7)   |      12 | f     | f    | 29 27 00 00
          8 | (0,8)   |      12 | f     | f    | 2a 27 00 00
```

`fsm_page_contents(page bytea) returns text`

`fsm_page_contents`显示 FSM 页的内部节点结构。输出是一个多行字符串，页内二叉树中的每个节点对应一行。只打印非零节点。还会打印所谓的 “next” 指针，它指向下一个将从该页返回的槽位。

参见`src/backend/storage/freespace/README`，了解 FSM 页面结构的更多信息。

F.23. passwordcheck

每当通过CREATE ROLE或ALTER ROLE设置用户密码时，`passwordcheck` 模块都会检查密码强度。如果密码被认为过弱，就会被拒绝，该命令也会因错误而终止。

要启用该模块，请将 `'$libdir/passwordcheck'` 加入 `postgresql.conf` 中的shared_preload_libraries，然后重启服务器。

可以通过修改源代码使该模块适应你的需求。例如，可以使用 CrackLib[2] 来检查密码，这只需要在 `Makefile` 中取消两行注释并重新构建该模块。（由于许可原因，我们不能默认包含 CrackLib。）如果不使用 CrackLib，该模块会对密码强度施加一些简单规则，而这些规则也可以按需修改或扩展。

小心

为了防止未加密的密码通过网络传输、被写入服务器日志，
或以其他方式被数据库管理员窃取，PostgreSQL允许用户提供预加密密码。
很多客户端程序都会利用这一能力，在将密码发送给服务器之前先行加密。

这就限制了 `passwordcheck` 模块的作用，因为在这种情况下，它只能尝试猜测密码。因此，如果安全要求很高，并不建议使用 `passwordcheck`。与其依赖数据库内部的密码，不如使用 Kerberos 之类的外部认证方法（见第 19 章），那样更安全。

另外，也可以修改 `passwordcheck` 以拒绝预加密密码，但强迫用户以明文设置密码本身也会带来安全风险。

[2] http://sourceforge.net/projects/cracklib/

F.24. pg_buffercache

pg_buffercache 模块提供了一种手段，可实时检查共享缓冲区缓存中正在发生的情况。它还提供了一种底层方式，可出于测试目的从中逐出数据。

该模块提供一个 C 函数pg_buffercache_pages，它返回一组记录；另外还提供一个视图pg_buffercache，该视图封装此函数以便使用。

默认情况下，这两个对象的公共访问权限都已被撤销，以防存在潜在的安全问题。

F.24.1. pg_buffercache 视图

该视图公开的列定义见表 F.16。

表 F.16. pg_buffercache 列

名称	类型	引用	描述
bufferid	integer		ID，范围为 1..shared_buffers
relfilenode	oid	pg_class.relfilenode	关系的 filenode 编号
reltablespace	oid	pg_tablespace.oid	关系的表空间 OID
reldatabase	oid	pg_database.oid	关系的数据库 OID
relblocknumber	bigint		关系中的页号
relforknumber	smallint		关系中的分支编号；参见include/storage/relfilenode.h
isdirty	boolean		该页是否为脏页？
usagecount	smallint		时钟扫描访问计数

共享缓存中的每个缓冲区都有一行记录。未使用的缓冲区除了 bufferid 之外，其余字段均显示为 null。共享系统目录显示为属于数据库 0。

由于缓存由所有数据库共享，其中通常会有不属于当前数据库的关系页。这意味着某些行在 pg_class 中可能没有可匹配的行，甚至还可能出现错误的连接结果。如果要与 pg_class 进行连接，最好将连接限制在 reldatabase 等于当前数据库 OID 或 0 的那些行上。

访问 pg_buffercache 视图时，会在足够长的时间内持有内部缓冲区管理器锁，以复制该视图将要显示的全部缓冲区状态数据。这确保了该视图产生一致的结果集，
同时又不会对正常缓冲区活动造成超出必要时长的阻塞。尽管如此，如果频繁读取该视图，
仍可能对数据库性能产生一定影响。

F.24.2. 示例输出

```
regression=# SELECT c.relname, count(*) AS buffers
             FROM pg_buffercache b INNER JOIN pg_class c
             ON b.relfilenode = pg_relation_filenode(c.oid) AND
                b.reldatabase IN (0, (SELECT oid FROM pg_database
                                      WHERE datname = current_
database()))
             GROUP BY c.relname
             ORDER BY 2 DESC
             LIMIT 10;

             relname              | buffers
```

```
---------------------------------+---------
 tenk2                           |     345
 tenk1                           |     141
 pg_proc                         |      46
 pg_class                        |      45
 pg_attribute                    |      43
 pg_class_relname_nsp_index      |      30
 pg_proc_proname_args_nsp_index  |      28
 pg_attribute_relid_attnam_index |      26
 pg_depend                       |      22
 pg_depend_reference_index       |      20
(10 rows)
```

F.24.3. 作者

Mark Kirkwood <markir@paradise.net.nz>

设计建议：Neil Conway <neilc@samurai.com>

调试建议：Tom Lane <tgl@sss.pgh.pa.us>

F.25. pgcrypto

The `pgcrypto` module provides cryptographic functions for PostgreSQL.

F.25.1. General Hashing Functions

F.25.1.1. `digest()`

```
digest(data text, type text) returns bytea
digest(data bytea, type text) returns bytea
```

Computes a binary hash of the given *`data`*. *`type`* is the algorithm to use. Standard algorithms are `md5`, `sha1`, `sha224`, `sha256`, `sha384` and `sha512`. If `pgcrypto` was built with OpenSSL, more algorithms are available, as detailed in 表 F.20.

If you want the digest as a hexadecimal string, use `encode()` on the result. For example:

```
CREATE OR REPLACE FUNCTION sha1(bytea) returns text AS $$
    SELECT encode(digest($1, 'sha1'), 'hex')
$$ LANGUAGE SQL STRICT IMMUTABLE;
```

F.25.1.2. `hmac()`

```
hmac(data text, key text, type text) returns bytea
hmac(data bytea, key text, type text) returns bytea
```

Calculates hashed MAC for *`data`* with key *`key`*. *`type`* is the same as in `digest()`.

This is similar to `digest()` but the hash can only be recalculated knowing the key. This prevents the scenario of someone altering data and also changing the hash to match.

If the key is larger than the hash block size it will first be hashed and the result will be used as key.

F.25.2. Password Hashing Functions

The functions `crypt()` and `gen_salt()` are specifically designed for hashing passwords. `crypt()` does the hashing and `gen_salt()` prepares algorithm parameters for it.

The algorithms in `crypt()` differ from usual hashing algorithms like MD5 or SHA1 in the following respects:

1. They are slow. As the amount of data is so small, this is the only way to make brute-forcing passwords hard.
2. They use a random value, called the salt, so that users having the same password will have different encrypted passwords. This is also an additional defense against reversing the algorithm.
3. They include the algorithm type in the result, so passwords hashed with different algorithms can co-exist.
4. Some of them are adaptive – that means when computers get faster, you can tune the algorithm to be slower, without introducing incompatibility with existing passwords.

表 F.17 lists the algorithms supported by the `crypt()` function.

表 F.17. Supported Algorithms for `crypt()`

Algorithm	Max Password Length	Adaptive?	Salt Bits	Description
`bf`	72	yes	128	Blowfish-based, variant 2a
`md5`	unlimited	no	48	MD5-based crypt
`xdes`	8	yes	24	Extended DES
`des`	8	no	12	Original UNIX crypt

F.25.2.1. `crypt()`

```
crypt(password text, salt text) returns text
```

Calculates a crypt(3)-style hash of *`password`*. When storing a new password, you need to use `gen_salt()` to generate a new *`salt`* value. To check a password, pass the stored hash value as *`salt`*, and test whether the result matches the stored value.

Example of setting a new password:

```
UPDATE ... SET pswhash = crypt('new password', gen_salt('md5'));
```

Example of authentication:

```
SELECT pswhash = crypt('entered password', pswhash) FROM ... ;
```

This returns `true` if the entered password is correct.

F.25.2.2. `gen_salt()`

```
gen_salt(type text [, iter_count integer ]) returns text
```

Generates a new random salt string for use in `crypt()`. The salt string also tells `crypt()` which algorithm to use.

The *`type`* parameter specifies the hashing algorithm. The accepted types are: `des`, `xdes`, `md5` and `bf`.

The *`iter_count`* parameter lets the user specify the iteration count, for algorithms that have one. The higher the count, the more time it takes to hash the password and therefore the more time to break it. Although with too high a count the time to calculate a hash may be several years – which is somewhat impractical. If the *`iter_count`* parameter is omitted, the default iteration count is used. Allowed values for *`iter_count`* depend on the algorithm and are shown in 表 F.18.

表 F.18. Iteration Counts for `crypt()`

Algorithm	Default	Min	Max
`xdes`	725	1	16777215
`bf`	6	4	31

For `xdes` there is an additional limitation that the iteration count must be an odd number.

To pick an appropriate iteration count, consider that the original DES crypt was designed to have the speed of 4 hashes per second on the hardware of that time.

Slower than 4 hashes per second would probably dampen usability. Faster than 100 hashes per second is probably too fast.

表 F.19 gives an overview of the relative slowness of different hashing algorithms. The table shows how much time it would take to try all combinations of characters in an 8-character password, assuming that the password contains either only lower case letters, or upper- and lower-case letters and numbers. In the `crypt-bf` entries, the number after a slash is the *`iter_count`* parameter of `gen_salt`.

表 F.19. Hash Algorithm Speeds

Algorithm	Hashes/sec	For `[a-z]`	For `[A-Za-z0-9]`
`crypt-bf/8`	28	246 years	251322 years
`crypt-bf/7`	57	121 years	123457 years
`crypt-bf/6`	112	62 years	62831 years
`crypt-bf/5`	211	33 years	33351 years
`crypt-md5`	2681	2.6 years	2625 years
`crypt-des`	362837	7 days	19 years
`sha1`	590223	4 days	12 years
`md5`	2345086	1 day	3 years

Notes:

- The machine used is a 1.5GHz Pentium 4.
- `crypt-des` and `crypt-md5` algorithm numbers are taken from John the Ripper v1.6.38 `-test` output.
- `md5` numbers are from mdcrack 1.2.
- `sha1` numbers are from lcrack-20031130-beta.
- `crypt-bf` numbers are taken using a simple program that loops over 1000 8-character passwords. That way I can show the speed with different numbers of iterations. For reference: `john -test` shows 213 loops/sec for `crypt-bf/5`. (The very small difference in results is in accordance with the fact that the `crypt-bf` implementation in `pgcrypto` is the same one used in John the Ripper.)

Note that "try all combinations" is not a realistic exercise. Usually password cracking is done with the help of dictionaries, which contain both regular words and various mutations of them. So, even somewhat word-like passwords could be cracked much faster than the above numbers suggest, while a 6-character non-word-like password may escape cracking. Or not.

F.25.3. PGP Encryption Functions

The functions here implement the encryption part of the OpenPGP (RFC 4880) standard. Supported are both symmetric-key and public-key encryption.

An encrypted PGP message consists of 2 parts, or packets:

- Packet containing a session key – either symmetric-key or public-key encrypted.
- Packet containing data encrypted with the session key.

When encrypting with a symmetric key (i.e., a password):

1. The given password is hashed using a String2Key (S2K) algorithm. This is rather similar to `crypt()` algorithms – purposefully slow and with random salt – but it produces a full-length binary key.
2. If a separate session key is requested, a new random key will be generated. Otherwise the S2K key will be used directly as the session key.
3. If the S2K key is to be used directly, then only S2K settings will be put into the session key packet. Otherwise the session key will be encrypted with the S2K key and put into the session key packet.

When encrypting with a public key:

1. A new random session key is generated.
2. It is encrypted using the public key and put into the session key packet.

In either case the data to be encrypted is processed as follows:

1. Optional data-manipulation: compression, conversion to UTF-8, and/or conversion of line-endings.
2. The data is prefixed with a block of random bytes. This is equivalent to using a random IV.
3. An SHA1 hash of the random prefix and data is appended.
4. All this is encrypted with the session key and placed in the data packet.

F.25.3.1. pgp_sym_encrypt()

```
pgp_sym_encrypt(data text, psw text [, options text ]) returns
 bytea
pgp_sym_encrypt_bytea(data bytea, psw text [, options text ])
 returns bytea
```

Encrypt *data* with a symmetric PGP key *psw*. The *options* parameter can contain option settings, as described below.

F.25.3.2. pgp_sym_decrypt()

```
pgp_sym_decrypt(msg bytea, psw text [, options text ]) returns text
pgp_sym_decrypt_bytea(msg bytea, psw text [, options text ])
 returns bytea
```

Decrypt a symmetric-key-encrypted PGP message.

Decrypting `bytea` data with `pgp_sym_decrypt` is disallowed. This is to avoid outputting invalid character data. Decrypting originally textual data with `pgp_sym_decrypt_bytea` is fine.

The *options* parameter can contain option settings, as described below.

F.25.3.3. pgp_pub_encrypt()

```
pgp_pub_encrypt(data text, key bytea [, options text ]) returns
 bytea
pgp_pub_encrypt_bytea(data bytea, key bytea [, options text ])
 returns bytea
```

Encrypt *data* with a public PGP key *key*. Giving this function a secret key will produce an error.

The *options* parameter can contain option settings, as described below.

F.25.3.4. pgp_pub_decrypt()

```
pgp_pub_decrypt(msg bytea, key bytea [, psw text [, options text ]]
) returns text
pgp_pub_decrypt_bytea(msg bytea, key bytea [, psw text [, options
 text ]]) returns bytea
```

Decrypt a public-key-encrypted message. *key* must be the secret key corresponding to the public key that was used to encrypt. If the secret key is password-protected, you must give the password in *psw*. If there is no password, but you want to specify options, you need to give an empty password.

Decrypting `bytea` data with `pgp_pub_decrypt` is disallowed. This is to avoid outputting invalid character data. Decrypting originally textual data with `pgp_pub_decrypt_bytea` is fine.

The *options* parameter can contain option settings, as described below.

F.25.3.5. pgp_key_id()

```
pgp_key_id(bytea) returns text
```

`pgp_key_id` extracts the key ID of a PGP public or secret key. Or it gives the key ID that was used for encrypting the data, if given an encrypted message.

It can return 2 special key IDs:

- `SYMKEY`

  The message is encrypted with a symmetric key.

- `ANYKEY`

  The message is public-key encrypted, but the key ID has been removed. That means you will need to try all your secret keys on it to see which one decrypts it. `pgcrypto` itself does not produce such messages.

Note that different keys may have the same ID. This is rare but a normal event. The client application should then try to decrypt with each one, to see which fits – like handling `ANYKEY`.

F.25.3.6. armor(), dearmor()

```
armor(data bytea) returns text
dearmor(data text) returns bytea
```

These functions wrap/unwrap binary data into PGP ASCII-armor format, which is basically Base64 with CRC and additional formatting.

F.25.3.7. Options for PGP Functions

Options are named to be similar to GnuPG. An option's value should be given after an equal sign; separate options from each other with commas. For example:

```
pgp_sym_encrypt(data, psw, 'compress-algo=1, cipher-algo=aes256')
```

All of the options except `convert-crlf` apply only to encrypt functions. Decrypt functions get the parameters from the PGP data.

The most interesting options are probably `compress-algo` and `unicode-mode`. The rest should have reasonable defaults.

F.25.3.7.1. cipher-algo

Which cipher algorithm to use.

Values: bf, aes128, aes192, aes256 (OpenSSL-only: `3des`, `cast5`)
Default: aes128
Applies to: pgp_sym_encrypt, pgp_pub_encrypt

F.25.3.7.2. compress-algo

Which compression algorithm to use. Only available if PostgreSQL was built with zlib.

Values:
  0 - no compression
  1 - ZIP compression
  2 - ZLIB compression (= ZIP plus meta-data and block CRCs)
Default: 0
Applies to: pgp_sym_encrypt, pgp_pub_encrypt

F.25.3.7.3. compress-level

How much to compress. Higher levels compress smaller but are slower. 0 disables compression.

Values: 0, 1-9
Default: 6
Applies to: pgp_sym_encrypt, pgp_pub_encrypt

F.25.3.7.4. convert-crlf

Whether to convert `\n` into `\r\n` when encrypting and `\r\n` to `\n` when decrypting. RFC 4880 specifies that text data should be stored using `\r\n` line-feeds. Use this to get fully RFC-compliant behavior.

Values: 0, 1
Default: 0
Applies to: pgp_sym_encrypt, pgp_pub_encrypt, pgp_sym_decrypt, pgp_pub_decrypt

F.25.3.7.5. disable-mdc

Do not protect data with SHA-1. The only good reason to use this option is to achieve compatibility with ancient PGP products, predating the addition of SHA-1 protected packets to RFC 4880. Recent gnupg.org and pgp.com software supports it fine.

Values: 0, 1
Default: 0
Applies to: pgp_sym_encrypt, pgp_pub_encrypt

F.25.3.7.6. sess-key

Use separate session key. Public-key encryption always uses a separate session key; this option is for symmetric-key encryption, which by default uses the S2K key directly.

Values: 0, 1
Default: 0
Applies to: pgp_sym_encrypt

F.25.3.7.7. s2k-mode

Which S2K algorithm to use.

Values:
 0 - Without salt. Dangerous!
 1 - With salt but with fixed iteration count.
 3 - Variable iteration count.
Default: 3
Applies to: pgp_sym_encrypt

F.25.3.7.8. s2k-digest-algo

Which digest algorithm to use in S2K calculation.

Values: md5, sha1
Default: sha1
Applies to: pgp_sym_encrypt

F.25.3.7.9. s2k-cipher-algo

Which cipher to use for encrypting separate session key.

Values: bf, aes, aes128, aes192, aes256
Default: use cipher-algo
Applies to: pgp_sym_encrypt

F.25.3.7.10. unicode-mode

Whether to convert textual data from database internal encoding to UTF-8 and back. If your database already is UTF-8, no conversion will be done, but the message will be tagged as UTF-8. Without this option it will not be.

Values: 0, 1
Default: 0
Applies to: pgp_sym_encrypt, pgp_pub_encrypt

F.25.3.8. Generating PGP Keys with GnuPG

To generate a new key:

```
gpg --gen-key
```

The preferred key type is “DSA and Elgamal”.

For RSA encryption you must create either DSA or RSA sign-only key as master and then add an RSA encryption subkey with `gpg --edit-key`.

To list keys:

```
gpg --list-secret-keys
```

To export a public key in ASCII-armor format:

```
gpg -a --export KEYID > public.key
```

To export a secret key in ASCII-armor format:

```
gpg -a --export-secret-keys KEYID > secret.key
```

You need to use `dearmor()` on these keys before giving them to the PGP functions. Or if you can handle binary data, you can drop `-a` from the command.

For more details see `man gpg`, The GNU Privacy Handbook[3] and other documentation on http://www.gnupg.org.

F.25.3.9. Limitations of PGP Code

- No support for signing. That also means that it is not checked whether the encryption subkey belongs to the master key.
- No support for encryption key as master key. As such practice is generally discouraged, this should not be a problem.
- No support for several subkeys. This may seem like a problem, as this is common practice. On the other hand, you should not use your regular GPG/PGP keys with `pgcrypto`, but create new ones, as the usage scenario is rather different.

F.25.4. Raw Encryption Functions

These functions only run a cipher over data; they don't have any advanced features of PGP encryption. Therefore they have some major problems:

1. They use user key directly as cipher key.
2. They don't provide any integrity checking, to see if the encrypted data was modified.
3. They expect that users manage all encryption parameters themselves, even IV.
4. They don't handle text.

So, with the introduction of PGP encryption, usage of raw encryption functions is discouraged.

```
encrypt(data bytea, key bytea, type text) returns bytea
decrypt(data bytea, key bytea, type text) returns bytea

encrypt_iv(data bytea, key bytea, iv bytea, type text) returns
 bytea
decrypt_iv(data bytea, key bytea, iv bytea, type text) returns
 bytea
```

Encrypt/decrypt data using the cipher method specified by *`type`*. The syntax of the *`type`* string is:

`algorithm` `[ -` *`mode`* `] [ /pad:` *`padding`* `]`

where *`algorithm`* is one of:

- `bf` – Blowfish
- `aes` – AES (Rijndael-128)

and *`mode`* is one of:

- `cbc` – next block depends on previous (default)

[3] http://www.gnupg.org/gph/en/manual.html

- `ecb` – each block is encrypted separately (for testing only)

and *`padding`* is one of:

- `pkcs` – data may be any length (default)
- `none` – data must be multiple of cipher block size

So, for example, these are equivalent:

```
encrypt(data, 'fooz', 'bf')
encrypt(data, 'fooz', 'bf-cbc/pad:pkcs')
```

In `encrypt_iv` and `decrypt_iv`, the *`iv`* parameter is the initial value for the CBC mode; it is ignored for ECB. It is clipped or padded with zeroes if not exactly block size. It defaults to all zeroes in the functions without this parameter.

F.25.5. Random-Data Functions

```
gen_random_bytes(count integer) returns bytea
```

Returns *`count`* cryptographically strong random bytes. At most 1024 bytes can be extracted at a time. This is to avoid draining the randomness generator pool.

F.25.6. Notes

F.25.6.1. Configuration

`pgcrypto` configures itself according to the findings of the main PostgreSQL `configure` script. The options that affect it are `--with-zlib` and `--with-openssl`.

When compiled with zlib, PGP encryption functions are able to compress data before encrypting.

When compiled with OpenSSL, there will be more algorithms available. Also public-key encryption functions will be faster as OpenSSL has more optimized BIGNUM functions.

表 F.20. Summary of Functionality with and without OpenSSL

Functionality	Built-in	With OpenSSL
MD5	yes	yes
SHA1	yes	yes
SHA224/256/384/512	yes	yes (Note 1)
Other digest algorithms	no	yes (Note 2)
Blowfish	yes	yes
AES	yes	yes (Note 3)
DES/3DES/CAST5	no	yes
Raw encryption	yes	yes
PGP Symmetric encryption	yes	yes
PGP Public-Key encryption	yes	yes

Notes:

1. SHA2 algorithms were added to OpenSSL in version 0.9.8. For older versions, `pgcrypto` will use built-in code.

2. Any digest algorithm OpenSSL supports is automatically picked up. This is not possible with ciphers, which need to be supported explicitly.

3. AES is included in OpenSSL since version 0.9.7. For older versions, `pgcrypto` will use built-in code.

F.25.6.2. NULL Handling

As is standard in SQL, all functions return NULL, if any of the arguments are NULL. This may create security risks on careless usage.

F.25.6.3. Security Limitations

All `pgcrypto` functions run inside the database server. That means that all the data and passwords move between `pgcrypto` and client applications in clear text. Thus you must:

1. Connect locally or use SSL connections.

2. Trust both system and database administrator.

If you cannot, then better do crypto inside client application.

The implementation does not resist side-channel attacks[4]. For example, the time required for a `pgcrypto` decryption function to complete varies among ciphertexts of a given size.

F.25.6.4. Useful Reading

- http://www.gnupg.org/gph/en/manual.html

  The GNU Privacy Handbook.

- http://www.openwall.com/crypt/

  Describes the crypt-blowfish algorithm.

- http://www.stack.nl/~galactus/remailers/passphrase-faq.html

  How to choose a good password.

- http://world.std.com/~reinhold/diceware.html

  Interesting idea for picking passwords.

- http://www.interhack.net/people/cmcurtin/snake-oil-faq.html

  Describes good and bad cryptography.

F.25.6.5. Technical References

- http://www.ietf.org/rfc/rfc4880.txt

  OpenPGP message format.

- http://www.ietf.org/rfc/rfc1321.txt

  The MD5 Message-Digest Algorithm.

[4] http://en.wikipedia.org/wiki/Side-channel_attack

- http://www.ietf.org/rfc/rfc2104.txt

  HMAC: Keyed-Hashing for Message Authentication.

- http://www.usenix.org/events/usenix99/provos.html

  Comparison of crypt-des, crypt-md5 and bcrypt algorithms.

- http://csrc.nist.gov/cryptval/des.htm

  Standards for DES, 3DES and AES.

- http://en.wikipedia.org/wiki/Fortuna_(PRNG)

  Description of Fortuna CSPRNG.

- http://jlcooke.ca/random/

  Jean-Luc Cooke Fortuna-based `/dev/random` driver for Linux.

- http://research.cyber.ee/~lipmaa/crypto/

  Collection of cryptology pointers.

F.25.7. Author

Marko Kreen `<markokr@gmail.com>`

`pgcrypto` uses code from the following sources:

Algorithm	Author	Source origin
DES crypt	David Burren and others	FreeBSD libcrypt
MD5 crypt	Poul-Henning Kamp	FreeBSD libcrypt
Blowfish crypt	Solar Designer	www.openwall.com
Blowfish cipher	Simon Tatham	PuTTY
Rijndael cipher	Brian Gladman	OpenBSD sys/crypto
MD5 and SHA1	WIDE Project	KAME kame/sys/crypto
SHA256/384/512	Aaron D. Gifford	OpenBSD sys/crypto
BIGNUM math	Michael J. Fromberger	dartmouth.edu/~sting/sw/imath

F.26. pg_freespacemap

pg_freespacemap模块提供了一种检查空闲空间映射（FSM）的方法。
它提供了一个名为pg_freespace的函数，更准确地说，是两个重载函数。
这些函数显示FSM中为给定页面或关系中所有页面记录的值。

默认情况下，这些函数的公共访问权限已被撤销，以防存在潜在的安全问题。

F.26.1. 函数

pg_freespace(rel regclass IN, blkno bigint IN) returns int2

返回该关系中由blkno指定页面上按 FSM 记录的空闲空间量。

pg_freespace(rel regclass IN, blkno OUT bigint, avail OUT int2)

显示该关系中每个页面上按 FSM 记录的空闲空间量。返回一组(blkno bigint, avail int2)元组，其中关系中的每个页面对应一个元组。

存储在空闲空间映射中的值并不精确。它们会被舍入到BLCKSZ的 1/256 精度（在默认BLCKSZ 下为 32 字节），并且不会随着元组的插入和更新而完全保持最新。

对于索引，被跟踪的是完全未使用的页面，而不是页面内部的空闲空间。因此，
这些值本身并无实际意义，只能表明某个页面是满的还是空的。

注：该接口在 8.4 版中进行了更改，以反映同一版本引入的新 FSM 实现。

F.26.2. 示例输出

```
postgres=# SELECT * FROM pg_freespace('foo');
 blkno | avail
-------+-------
     0 |     0
     1 |     0
     2 |     0
     3 |    32
     4 |   704
     5 |   704
     6 |   704
     7 |  1216
     8 |   704
     9 |   704
    10 |   704
    11 |   704
    12 |   704
    13 |   704
    14 |   704
    15 |   704
    16 |   704
    17 |   704
    18 |   704
    19 |  3648
(20 rows)

postgres=# SELECT * FROM pg_freespace('foo', 7);
 pg_freespace
--------------
```

```
          1216
(1 row)
```

F.26.3. 作者

最初版本由 Mark Kirkwood <markir@paradise.net.nz> 编写。在 8.4 版中，Heikki Linnakangas <heikki@enterprisedb.com> 为适应新的 FSM 实现对其进行了重写。

F.27. pgrowlocks – 显示表的行锁信息

pgrowlocks 模块提供了一个函数，用于显示指定表的行锁信息。

F.27.1. 概述

```
pgrowlocks(text) returns setof record
```

参数为表名。结果是一个记录集合，其中表中每个被锁定的行对应结果中的一行。
输出列见表 F.21。

表 F.21. pgrowlocks 输出列

名称	类型	描述
locked_row	tid	被锁定行的元组 ID（TID）
lock_type	text	共享锁为 Shared，排他锁为 Exclusive
locker	xid	持锁者的事务 ID；若为多事务，则为多事务 ID
multi	boolean	如果持锁者为多事务，则为真
xids	xid[]	持锁者的事务 ID（若为多事务，则会有多个）
pids	integer[]	加锁后端的进程 ID（若为多事务，则会有多个）

pgrowlocks 会对目标表获取 AccessShareLock，并逐行读取以收集行锁信息。对于大表，这样做速度并不快。请注意：

1. 如果整张表被其他人独占锁定，pgrowlocks 将被阻塞。
2. pgrowlocks 并不保证给出一个自一致快照。在其执行期间，可能会有新的行锁被获取，也可能有旧的锁被释放。

pgrowlocks 不显示被锁定行的内容。如果你想同时查看这些行的内容，可以这样做：

```
SELECT * FROM accounts AS a, pgrowlocks('accounts') AS p
  WHERE p.locked_row = a.ctid;
```

不过要注意，这样的查询（截至 PostgreSQL 8.3）效率会很低。

F.27.2. 示例输出

```
test=# SELECT * FROM pgrowlocks('t1');
 locked_row | lock_type | locker | multi |   xids    |     pids
-------------+-----------+--------+-------+-----------+------------
---
      (0,1) | Shared    |     19 | t     | {804,805} | {29066,
29068}
      (0,2) | Shared    |     19 | t     | {804,805} | {29066,
29068}
      (0,3) | Exclusive |    804 | f     | {804}     | {29066}
      (0,4) | Exclusive |    804 | f     | {804}     | {29066}
(4 rows)
```

F.27.3. 作者

Tatsuo Ishii

F.28. pg_stat_statements

The `pg_stat_statements` module provides a means for tracking execution statistics of all SQL statements executed by a server.

The module must be loaded by adding `pg_stat_statements` to shared_preload_libraries in `postgresql.conf`, because it requires additional shared memory. This means that a server restart is needed to add or remove the module.

F.28.1. The `pg_stat_statements` View

The statistics gathered by the module are made available via a system view named `pg_stat_statements`. This view contains one row for each distinct query, database ID, and user ID (up to the maximum number of distinct statements that the module can track). The columns of the view are shown in 表 F.22.

表 F.22. `pg_stat_statements` Columns

Name	Type	References	Description
`userid`	`oid`	`pg_authid.oid`	OID of user who executed the statem ent
`dbid`	`oid`	`pg_database.oid`	OID of database in which the statement was executed
`query`	`text`		Text of a representati ve statement (up to track_activity_query_ size bytes)
`calls`	`bigint`		Number of times executed
`total_time`	`double precision`		Total time spent in the statement, in millis econds
`rows`	`bigint`		Total number of rows retrieved or affected by the statement
`shared_blks_hit`	`bigint`		Total number of shared block cache hits by the statement
`shared_blks_read`	`bigint`		Total number of shared blocks read by the statement
`shared_blks_dirtied`	`bigint`		Total number of shared blocks dirtied by the statement
`shared_blks_written`	`bigint`		Total number of shared blocks written by the statement
`local_blks_hit`	`bigint`		Total number of local block cache hits by the statement
`local_blks_read`	`bigint`		Total number of local blocks read by the statement

Name	Type	References	Description
local_blks_dirtied	bigint		Total number of local blocks dirtied by the statement
local_blks_written	bigint		Total number of local blocks written by the statement
temp_blks_read	bigint		Total number of temp blocks read by the statement
temp_blks_written	bigint		Total number of temp blocks written by the statement
blk_read_time	double precision		Total time the statement spent reading blocks, in milliseconds (if track_io_timing is enabled, otherwise zero)
blk_write_time	double precision		Total time the statement spent writing blocks, in milliseconds (if track_io_timing is enabled, otherwise zero)

This view, and the function `pg_stat_statements_reset`, are available only in databases they have been specifically installed into by installing the `pg_stat_statements` extension. However, statistics are tracked across all databases of the server whenever the `pg_stat_statements` module is loaded into the server, regardless of presence of the view.

For security reasons, non-superusers are not allowed to see the text of queries executed by other users. They can see the statistics, however, if the view has been installed in their database.

Plannable queries (that is, `SELECT`, `INSERT`, `UPDATE`, and `DELETE`) are combined into a single `pg_stat_statements` entry whenever they have identical query structures according to an internal hash calculation. Typically, two queries will be considered the same for this purpose if they are semantically equivalent except for the values of literal constants appearing in the query. Utility commands (that is, all other commands) are compared strictly on the basis of their textual query strings, however.

When a constant's value has been ignored for purposes of matching the query to other queries, the constant is replaced by `?` in the `pg_stat_statements` display. The rest of the query text is that of the first query that had the particular hash value associated with the `pg_stat_statements` entry.

In some cases, queries with visibly different texts might get merged into a single `pg_stat_statements` entry. Normally this will happen only for semantically equivalent queries, but there is a small chance of hash collisions causing unrelated queries to be merged into one entry. (This cannot happen for queries belonging to different users or databases, however.)

Since the hash value is computed on the post-parse-analysis representation of the queries, the opposite is also possible: queries with identical texts might appear as separate entries, if they have different meanings as a result of factors such as different `search_path` settings.

F.28.2. Functions

`pg_stat_statements_reset() returns void`

`pg_stat_statements_reset` discards all statistics gathered so far by `pg_stat_statements`. By default, this function can only be executed by superusers.

F.28.3. Configuration Parameters

`pg_stat_statements.max` (`integer`)

`pg_stat_statements.max` is the maximum number of statements tracked by the module (i.e., the maximum number of rows in the `pg_stat_statements` view). If more distinct statements than that are observed, information about the least-executed statements is discarded. The default value is 1000. This parameter can only be set at server start.

`pg_stat_statements.track` (`enum`)

`pg_stat_statements.track` controls which statements are counted by the module. Specify `top` to track top-level statements (those issued directly by clients), `all` to also track nested statements (such as statements invoked within functions), or `none` to disable statement statistics collection. The default value is `top`. Only superusers can change this setting.

`pg_stat_statements.track_utility` (`boolean`)

`pg_stat_statements.track_utility` controls whether utility commands are tracked by the module. Utility commands are all those other than `SELECT`, `INSERT`, `UPDATE` and `DELETE`. The default value is `on`. Only superusers can change this setting.

`pg_stat_statements.save` (`boolean`)

`pg_stat_statements.save` specifies whether to save statement statistics across server shutdowns. If it is `off` then statistics are not saved at shutdown nor reloaded at server start. The default value is `on`. This parameter can only be set in the `postgresql.conf` file or on the server command line.

The module requires additional shared memory amounting to about `pg_stat_statements.max * track_activity_query_size` bytes. Note that this memory is consumed whenever the module is loaded, even if `pg_stat_statements.track` is set to `none`.

These parameters must be set in `postgresql.conf`. Typical usage might be:

```
# postgresql.conf
shared_preload_libraries = 'pg_stat_statements'

pg_stat_statements.max = 10000
pg_stat_statements.track = all
```

F.28.4. Sample Output

```
bench=# SELECT pg_stat_statements_reset();

$ pgbench -i bench
$ pgbench -c10 -t300 bench
```

```
bench=# \x
bench=# SELECT query, calls, total_time, rows, 100.0 * shared_blks_
hit /
               nullif(shared_blks_hit + shared_blks_read, 0) AS
 hit_percent
          FROM pg_stat_statements ORDER BY total_time DESC LIMIT 5;
-[ RECORD 1 ]-------------------------------------------------------
---------------
query       | UPDATE pgbench_branches SET bbalance = bbalance + ?
 WHERE bid = ?;
calls       | 3000
total_time  | 9609.00100000002
rows        | 2836
hit_percent | 99.9778970000200936
-[ RECORD 2 ]-------------------------------------------------------
---------------
query       | UPDATE pgbench_tellers SET tbalance = tbalance + ?
 WHERE tid = ?;
calls       | 3000
total_time  | 8015.156
rows        | 2990
hit_percent | 99.9731126579631345
-[ RECORD 3 ]-------------------------------------------------------
---------------
query       | copy pgbench_accounts from stdin
calls       | 1
total_time  | 310.624
rows        | 100000
hit_percent | 0.30395136778115501520
-[ RECORD 4 ]-------------------------------------------------------
---------------
query       | UPDATE pgbench_accounts SET abalance = abalance + ?
 WHERE aid = ?;
calls       | 3000
total_time  | 271.741999999997
rows        | 3000
hit_percent | 93.7968855088209426
-[ RECORD 5 ]-------------------------------------------------------
---------------
query       | alter table pgbench_accounts add primary key (aid)
calls       | 1
total_time  | 81.42
rows        | 0
hit_percent | 34.4947735191637631
```

F.28.5. Authors

Takahiro Itagaki <itagaki.takahiro@oss.ntt.co.jp>. Query normalization added by Peter Geoghegan <peter@2ndquadrant.com>.

F.29. pgstattuple – 获取元组级统计信息

pgstattuple 模块提供了多种用于获取元组级统计信息的函数。

F.29.1. 函数

pgstattuple(text) returns record

pgstattuple 返回一个关系的物理长度、“死”元组所占百分比以及其他信息。这可以帮助用户判断是否需要执行清理。参数是目标关系的名称（可选带模式限定）。例如：

```
test=> SELECT * FROM pgstattuple('pg_catalog.pg_proc');
-[ RECORD 1 ]------+-------
table_len          | 458752
tuple_count        | 1470
tuple_len          | 438896
tuple_percent      | 95.67
dead_tuple_count   | 11
dead_tuple_len     | 3157
dead_tuple_percent | 0.69
free_space         | 8932
free_percent       | 1.95
```

输出列的说明见表 F.23。

表 F.23. pgstattuple 输出列

列	类型	描述
table_len	bigint	关系的物理长度（字节）
tuple_count	bigint	活元组数量
tuple_len	bigint	活元组总长度（字节）
tuple_percent	float8	活元组百分比
dead_tuple_count	bigint	死元组数量
dead_tuple_len	bigint	死元组总长度（字节）
dead_tuple_percent	float8	死元组百分比
free_space	bigint	总空闲空间（字节）
free_percent	float8	空闲空间百分比

注意

table_len 总会大于 tuple_len、dead_tuple_len 和 free_space 之和。差值来自固定的页开销、每页的元组指针表，以及为保证元组正确对齐而产生的填充。

pgstattuple 只会在该关系上获取读锁。因此，结果并不代表一个瞬时快照；并发更新会影响结果。

如果 HeapTupleSatisfiesNow 返回 false，pgstattuple 就会将该元组判定为“死”。

pgstattuple(oid) returns record

这与 pgstattuple(text) 相同，只是目标关系是以 OID 指定的。

`pgstatindex(text) returns record`

`pgstatindex` 返回一条记录，显示有关 B-树索引的信息。例如：

```
test=> SELECT * FROM pgstatindex('pg_cast_oid_index');
-[ RECORD 1 ]------+------
version            | 2
tree_level         | 0
index_size         | 16384
root_block_no      | 1
internal_pages     | 0
leaf_pages         | 1
empty_pages        | 0
deleted_pages      | 0
avg_leaf_density   | 54.27
leaf_fragmentation | 0
```

输出列如下：

列	类型	描述
`version`	`integer`	B-树版本号
`tree_level`	`integer`	根页所在的树层级
`index_size`	`bigint`	索引总大小（字节）
`root_block_no`	`bigint`	根页的位置（若无则为零）
`internal_pages`	`bigint`	“内部”（上层）页的数量
`leaf_pages`	`bigint`	叶子页数量
`empty_pages`	`bigint`	空页数量
`deleted_pages`	`bigint`	已删除页数量
`avg_leaf_density`	`float8`	叶子页平均密度
`leaf_fragmentation`	`float8`	叶子页碎片化程度

报告的 `index_size` 通常会比 `internal_pages + leaf_pages + empty_pages + deleted_pages` 所对应的页数多 1 页，因为它还包含索引的元页。

与 `pgstattuple` 一样，结果是逐页累积得到的，因此不应期望它表示整个索引的瞬时快照。

`pg_relpages(text) returns bigint`

`pg_relpages` 返回该关系中的页数。

F.29.2. 作者

Tatsuo Ishii 和 Satoshi Nagayasu

F.30. pg_trgm

pg_trgm模块提供函数和操作符，用于基于三字符组匹配确定 ASCII 文本的相似度，同时还提供支持快速搜索相似字符串的索引操作符类。

F.30.1. 三字符组（Trigram 或 Trigraph）概念

三字符组是一组从字符串中取出的三个连续字符。
我们可以通过统计两个字符串共享的三字符组数量来度量它们的相似度。
这个简单的思想在度量许多自然语言中词的相似度时都非常有效。

> **注意**
>
> 在确定字符串所包含的三字符组集合时，认为字符串前面有两个空格，后面有一个空格。例如，字符串“cat”的三字符组集合是“ c”、“ ca”、“cat”和“at ”。

F.30.2. 函数和操作符

pg_trgm模块提供的函数列在表 F.24中，操作符列在表 F.25中。

表 F.24. pg_trgm函数

函数	返回值	描述
similarity(text, text)	real	返回一个表示两个参数有多相似的数值。结果范围从 0（表示两个字符串完全不同）到 1（表示两个字符串完全相同）。
show_trgm(text)	text[]	返回由给定字符串中所有三字符组构成的数组。（实际应用中，除了调试之外很少有用。）
show_limit()	real	返回%操作符当前使用的相似度阈值。它设置两个单词被视为足够相似、例如可以看作彼此的拼写错误所需的最小相似度。
set_limit(real)	real	设置%操作符当前使用的相似度阈值。阈值必须介于 0 和 1 之间（默认值为 0.3）。返回传入的相同值。

表 F.25. pg_trgm操作符

操作符	返回值	描述
text % text	boolean	如果参数之间的相似度大于set_limit()设置的当前相似度阈值，则返回true。
text <-> text	real	返回参数之间的“距离”，即 1 减去similarity()的值。

F.30.3. 索引支持

pg_trgm模块提供 GiST 和 GIN 索引操作符类，允许你为文本列创建索引，以实现非常快速的相似度搜索。这些索引类型支持上述相似度操作符，还支持对LIKE和ILIKE

查询执行基于三字符组的索引搜索。（这些索引不支持等值操作符或简单比较操作符，因此你可能还需要一个常规 B-树索引。）

示例：

```
CREATE TABLE test_trgm (t text);
CREATE INDEX trgm_idx ON test_trgm USING gist (t gist_trgm_ops);
```

或者

```
CREATE INDEX trgm_idx ON test_trgm USING gin (t gin_trgm_ops);
```

此时，你已经在`t`列上有了一个可用于相似度搜索的索引。典型查询如下：

```
SELECT t, similarity(t, 'word') AS sml
  FROM test_trgm
  WHERE t % 'word'
  ORDER BY sml DESC, t;
```

这将返回文本列中所有与以下词足够相似的值：*word*，按从最佳匹配到最差匹配的顺序排序。即使在非常大的数据集上，索引也会让这一操作保持高效。

上述查询的一种变体是：

```
SELECT t, t <-> 'word' AS dist
  FROM test_trgm
  ORDER BY dist LIMIT 10;
```

GiST 索引可以相当高效地实现这一点，但 GIN 索引不能。当只需要少量最接近的匹配项时，它通常会优于第一种写法。

从PostgreSQL9.1 起，这些索引类型还支持以下操作的索引搜索：`LIKE`以及`ILIKE`，例如：

```
SELECT * FROM test_trgm WHERE t LIKE '%foo%bar';
```

索引搜索的工作方式是从搜索字符串中提取三字符组，然后在索引中查找这些三字符组。搜索字符串中包含的三字符组越多，索引搜索就越有效。与基于 B-树的搜索不同，搜索字符串不需要在左端锚定。

GiST 和 GIN 索引之间如何取舍，取决于二者各自的相对性能特征；相关讨论见其他章节。作为经验法则，GIN 索引的搜索速度快于 GiST 索引，但构建或更新较慢；因此 GIN 更适合静态数据，而 GiST 更适合经常更新的数据。

F.30.4. 文本检索集成

与全文索引结合使用时，三字符组匹配是非常有用的工具。尤其是，它有助于识别那些因拼写错误而无法被全文检索机制直接匹配的输入词。

第一步是生成一个辅助表，其中包含文档中的全部唯一词：

```
CREATE TABLE words AS SELECT word FROM
        ts_stat('SELECT to_tsvector(''simple'', bodytext) FROM
 documents');
```

其中`documents`是一个表，包含我们希望搜索的文本字段`bodytext`。之所以对`to_tsvector`函数使用`simple`配置，而不是使用特定语言的配置，是因为我们需要原始的（未经词干提取的）词列表。

接下来，在词列上创建一个三字符组索引：

```
CREATE INDEX words_idx ON words USING gin (word gin_trgm_ops);
```

现在，可以使用与前面示例类似的SELECT查询，为用户搜索词中拼错的单词提供拼写建议。一个有用的附加测试是要求选出的词长度也与该拼错单词相近。

> **注意**
>
> 由于words表是作为一张独立的静态表生成的，因此需要定期重新生成，以便与文档集合保持大致同步。通常没有必要让它始终保持精确同步。

F.30.5. 参考

GiST 开发站点 http://www.sai.msu.su/~megera/postgres/gist/

Tsearch2 开发站点 http://www.sai.msu.su/~megera/postgres/gist/tsearch/V2/

F.30.6. 作者

Oleg Bartunov <oleg@sai.msu.su>，俄罗斯莫斯科，莫斯科大学

Teodor Sigaev <teodor@sigaev.ru>，俄罗斯莫斯科，Delta-Soft Ltd.

文档：Christopher Kings-Lynne

本模块由俄罗斯莫斯科的 Delta-Soft Ltd. 赞助。

F.31. seg – 用于线段或浮点区间的数据类型

该模块实现了一个用于表示线段或浮点区间的`seg`数据类型。`seg`
可以表示区间端点中的不确定性，因此特别适合表示实验室测量结果。

F.31.1. 原理

测量值的几何形态通常比数值连续体中的一个点更复杂。
一次测量通常是该连续体上的一段，其边界多少有些模糊。测量结果之所以会呈现为区间，
一方面是由于不确定性和随机性，
另一方面也是因为被测值本身可能天然就是表示某种状态的区间，
例如蛋白质的稳定温度范围。

凭常识也能看出，把这类数据存储为区间比存储为一对数字更方便。实际上，
在大多数应用中，这样做甚至更高效。

再顺着这一常识往下想，边界的模糊性说明，
使用传统数值数据类型会造成一定的信息损失。设想一下：你的仪器读数是 6.50，而你把这个读数输入到数据库。取出来时会得到什么？请看：

```
test=> select 6.50 :: float8 as "pH";
 pH
---
6.5
(1 row)
```

在测量领域，6.50 与 6.5 并不相同。有时这种差别至关重要。实验人员通常会记下（并发表）他们认为可靠的那些数位。6.50 实际上是一个模糊区间，它包含在更大、也更模糊的区间 6.5 之中；它们共享的特征（大概）只有中心点。我们当然不希望这类不同的数据项看起来却一样。

结论是什么？最好能有一种专门的数据类型，能够以任意可变的精度记录区间边界。
这里所谓“可变”，是指每个数据元素都记录其自身的精度。

看看这个：

```
test=> select '6.25 .. 6.50'::seg as "pH";
          pH
------------
6.25 .. 6.50
(1 row)
```

F.31.2. 语法

区间的外部表示由一个或两个浮点数通过范围操作符（`..` 或 `...`）连接而成。另一种写法是指定中心点再加减一个偏差值。还可以存储可选的确定性指示符（`<`、`>`或 `~`）。（不过，所有内置操作符都会忽略这些确定性指示符。）表 F.26概述了允许的表示形式；表 F.27给出了一些示例。

在表 F.26中，*x*、*y*和*delta*表示浮点数。*x*和*y*前面可以带确定性指示符，而*delta*不可以。

表 F.26. seg 外部表示

x	单个值（零长度区间）
x `..` *y*	从*x*到*y*的区间
x `(+-)` *delta*	从*x* - *delta*到 *x* + *delta*的区间
x `..`	下界为*x*、上界开放的区间

`.. x`	上界为*x*、下界开放的区间

表 F.27. 合法 seg 输入示例

`5.0`	创建一个零长度线段（如果你愿意，也可以把它看成一个点）
`~5.0`	创建一个零长度线段，并在数据中记录`~`。`~`会被`seg`操作忽略，但会作为注释保留下来。
`<5.0`	在 5.0 处创建一个点。`<`会被忽略，但会作为注释保留下来。
`>5.0`	在 5.0 处创建一个点。`>`会被忽略，但会作为注释保留下来。
`5(+-)0.3`	创建区间`4.7 .. 5.3`。注意，`(+-)`记法不会被保留。
`50 ..`	所有大于或等于 50 的值
`.. 0`	所有小于或等于 0 的值
`1.5e-2 .. 2E-2`	创建区间`0.015 .. 0.02`
`1 ... 2`	与`1...2`、`1 .. 2`或`1..2`相同（范围操作符两侧的空格会被忽略）

由于`...`操作符在数据源中被广泛使用，因此允许把它作为 `..`操作符的另一种写法。不幸的是，这会带来解析歧义：无法确定`0...23`中的上界是`23`还是 `0.23`。解决办法是要求`seg`输入中的所有数字在小数点前至少有一位数字。

作为合理性检查，`seg`会拒绝下界大于上界的区间，例如 `5 .. 2`。

F.31.3. 精度

`seg`值在内部存储为一对 32 位浮点数。这意味着有效位数超过 7 位的数字会被截断。

有效位数不超过 7 位的数字会保留其原始精度。也就是说，如果你的查询返回 0.00，你可以确信末尾的零不是格式化造成的假象，而是反映了原始数据的精度。
前导零的个数不影响精度：值 0.0067 被认为只有 2 位有效数字。

F.31.4. 用法

`seg`模块包含适用于`seg`值的一个 GiST 索引操作符类。该 GiST 操作符类支持的操作符见表 F.28。

表 F.28. seg GiST 操作符

操作符	描述
`[a, b] << [c, d]`	[a, b] 完全位于 [c, d] 的左侧。也就是说，当 b < c 时，[a, b] << [c, d] 为真，否则为假。
`[a, b] >> [c, d]`	[a, b] 完全位于 [c, d] 的右侧。也就是说，当 a > d 时，[a, b] >> [c, d] 为真，否则为假。
`[a, b] &< [c, d]`	重叠或位于左侧 – 也可以更好地理解为“不会延伸到右侧”。当 b <= d 时为真。
`[a, b] &> [c, d]`	重叠或位于右侧 – 也可以更好地理解为“不会延伸到左侧”。当 a >= c 时为真。
`[a, b] = [c, d]`	相同 – 线段 [a, b] 和 [c, d] 相同，即 a = c 且 b = d。
`[a, b] && [c, d]`	线段 [a, b] 和 [c, d] 重叠。
`[a, b] @> [c, d]`	线段 [a, b] 包含线段 [c, d]，即 a <= c 且 b >= d。

操作符	描述
`[a, b] <@ [c, d]`	线段 [a, b] 包含于 [c, d] 中，即 a >= c 且 b <= d。

（在 PostgreSQL 8.2 之前，包含操作符 `@>` 和 `<@` 分别称为 `@` 和 `~`。这些名称仍然可用，但已弃用，最终将被删除。请注意，旧名称与核心几何数据类型以前采用的约定正好相反！）

还提供标准的 B-树操作符，例如

操作符	描述
`[a, b] < [c, d]`	小于
`[a, b] > [c, d]`	大于

这些操作符除了排序之外，在实际用途上没有太大意义。它们首先比较 (a) 与 (c)，如果相等，再比较 (b) 与 (d)。这在大多数情况下会产生相当好的排序效果；如果希望对这种类型使用 ORDER BY，这会很有用。

F.31.5. 注意

有关用法示例，请参见回归测试`sql/seg.sql`。

把 `(+-)` 转换为常规范围的机制，在确定边界的有效位数时并不完全准确。例如，
如果结果区间包含 10 的幂，它会在下边界上多加一位：

```
postgres=> select '10(+-)1'::seg as seg;
      seg
---------
9.0 .. 11             -- should be: 9 .. 11
```

R 树索引的性能很大程度上取决于输入值的初始顺序。按`seg`列对输入表排序，可能会很有帮助；示例见脚本 `sort-segments.pl`。

F.31.6. 致谢

原作者：Gene Selkov, Jr. `<selkovjr@mcs.anl.gov>`，阿贡国家实验室数学与计算机科学部。

我首先要感谢 Joe Hellerstein 教授 (http://db.cs.berkeley.edu/jmh/)，他为我阐明了 GiST (http://gist.cs.berkeley.edu/) 的要旨。我同样感谢过去和现在所有的 Postgres 开发者，他们使我得以创造自己的世界并在其中不受打扰地生活。我还要感谢阿贡实验室以及美国能源部，多年来始终如一地支持我的数据库研究。

F.32. sepgsql

`sepgsql` is a loadable module that supports label-based mandatory access control (MAC) based on SELinux security policy.

> **警告**
>
> The current implementation has significant limitations, and does not enforce mandatory access control for all actions. See 第 F.32.7 节.

F.32.1. Overview

This module integrates with SELinux to provide an additional layer of security checking above and beyond what is normally provided by PostgreSQL. From the perspective of SELinux, this module allows PostgreSQL to function as a user-space object manager. Each table or function access initiated by a DML query will be checked against the system security policy. This check is in addition to the usual SQL permissions checking performed by PostgreSQL.

SELinux access control decisions are made using security labels, which are represented by strings such as `system_u:object_r:sepgsql_table_t:s0`. Each access control decision involves two labels: the label of the subject attempting to perform the action, and the label of the object on which the operation is to be performed. Since these labels can be applied to any sort of object, access control decisions for objects stored within the database can be (and, with this module, are) subjected to the same general criteria used for objects of any other type, such as files. This design is intended to allow a centralized security policy to protect information assets independent of the particulars of how those assets are stored.

The SECURITY LABEL statement allows assignment of a security label to a database object.

F.32.2. Installation

`sepgsql` can only be used on Linux 2.6.28 or higher with SELinux enabled. It is not available on any other platform. You will also need libselinux 2.0.99 or higher and selinux-policy 3.9.13 or higher (although some distributions may backport the necessary rules into older policy versions).

The `sestatus` command allows you to check the status of SELinux. A typical display is:

```
$ sestatus
SELinux status:                 enabled
SELinuxfs mount:                /selinux
Current mode:                   enforcing
Mode from config file:          enforcing
Policy version:                 24
Policy from config file:        targeted
```

If SELinux is disabled or not installed, you must set that product up first before installing this module.

To build this module, include the option `--with-selinux` in your PostgreSQL `configure` command. Be sure that the `libselinux-devel` RPM is installed at build time.

To use this module, you must include `sepgsql` in the shared_preload_libraries parameter in `postgresql.conf`. The module will not function correctly if loaded in any other manner. Once the module is loaded, you should execute `sepgsql.sql` in each database. This will install functions needed for security label management, and assign initial security labels.

Here is an example showing how to initialize a fresh database cluster with `sepgsql` functions and security labels installed. Adjust the paths shown as appropriate for your installation:

```
$ export PGDATA=/path/to/data/directory
$ initdb
$ vi $PGDATA/postgresql.conf
  change
    #shared_preload_libraries = ''                # (change
 requires restart)
  to
    shared_preload_libraries = 'sepgsql'          # (change
 requires restart)
$ for DBNAME in template0 template1 postgres; do
    postgres --single -F -c exit_on_error=true $DBNAME \
      </usr/local/pgsql/share/contrib/sepgsql.sql >/dev/null
  done
```

Please note that you may see some or all of the following notifications depending on the particular versions you have of libselinux and selinux-policy:

```
/etc/selinux/targeted/contexts/sepgsql_contexts:  line 33 has
 invalid object type db_blobs
/etc/selinux/targeted/contexts/sepgsql_contexts:  line 36 has
 invalid object type db_language
/etc/selinux/targeted/contexts/sepgsql_contexts:  line 37 has
 invalid object type db_language
/etc/selinux/targeted/contexts/sepgsql_contexts:  line 38 has
 invalid object type db_language
/etc/selinux/targeted/contexts/sepgsql_contexts:  line 39 has
 invalid object type db_language
/etc/selinux/targeted/contexts/sepgsql_contexts:  line 40 has
 invalid object type db_language
```

These messages are harmless and should be ignored.

If the installation process completes without error, you can now start the server normally.

F.32.3. Regression Tests

Due to the nature of SELinux, running the regression tests for `sepgsql` requires several extra configuration steps, some of which must be done as root. The regression tests will not be run by an ordinary `make check` or `make installcheck` command; you must set up the configuration and then invoke the test script manually. The tests must be run in the `contrib/sepgsql` directory of a configured PostgreSQL build tree. Although they require a build tree, the tests are designed to be executed against an installed server, that is they are comparable to `make installcheck` not `make check`.

First, set up `sepgsql` in a working database according to the instructions in 第 F.32.2 节. Note that the current operating system user must be able to connect to the database as superuser without password authentication.

Second, build and install the policy package for the regression test. The `sepgsql-regtest` policy is a special purpose policy package which provides a set of rules to be allowed during the regression tests. It should be built from the policy source file `sepgsql-regtest.te`, which is done using `make` with a Makefile supplied by SELinux. You will need to locate the appropriate Makefile on your system; the path shown below is only an example. Once built, install this policy package using the `semodule` command, which loads supplied policy packages into the kernel. If the package is correctly installed, `semodule -l` should list `sepgsql-regtest` as an available policy package:

```
$ cd .../contrib/sepgsql
$ make -f /usr/share/selinux/devel/Makefile
$ sudo semodule -u sepgsql-regtest.pp
$ sudo semodule -l | grep sepgsql
sepgsql-regtest 1.04
```

Third, turn on `sepgsql_regression_test_mode`. For security reasons, the rules in `sepgsql-regtest` are not enabled by default; the `sepgsql_regression_test_mode` parameter enables the rules needed to launch the regression tests. It can be turned on using the `setsebool` command:

```
$ sudo setsebool sepgsql_regression_test_mode on
$ getsebool sepgsql_regression_test_mode
sepgsql_regression_test_mode --> on
```

Fourth, verify your shell is operating in the `unconfined_t` domain:

```
$ id -Z
unconfined_u:unconfined_r:unconfined_t:s0-s0:c0.c1023
```

See 第 F.32.8 节 for details on adjusting your working domain, if necessary.

Finally, run the regression test script:

```
$ ./test_sepgsql
```

This script will attempt to verify that you have done all the configuration steps correctly, and then it will run the regression tests for the `sepgsql` module.

After completing the tests, it's recommended you disable the `sepgsql_regression_test_mode` parameter:

```
$ sudo setsebool sepgsql_regression_test_mode off
```

You might prefer to remove the `sepgsql-regtest` policy entirely:

```
$ sudo semodule -r sepgsql-regtest
```

F.32.4. GUC Parameters

`sepgsql.permissive` (`boolean`)

This parameter enables `sepgsql` to function in permissive mode, regardless of the system setting. The default is off. This parameter can only be set in the `postgresql.conf` file or on the server command line.

When this parameter is on, `sepgsql` functions in permissive mode, even if SELinux in general is working in enforcing mode. This parameter is primarily useful for testing purposes.

`sepgsql.debug_audit` (`boolean`)

This parameter enables the printing of audit messages regardless of the system policy settings. The default is off, which means that messages will be printed according to the system settings.

The security policy of SELinux also has rules to control whether or not particular accesses are logged. By default, access violations are logged, but allowed accesses are not.

This parameter forces all possible logging to be turned on, regardless of the system policy.

F.32.5. Features

F.32.5.1. Controlled Object Classes

The security model of SELinux describes all the access control rules as relationships between a subject entity (typically, a client of the database) and an object entity (such as a database object), each of which is identified by a security label. If access to an unlabelled object is attempted, the object is treated as if it were assigned the label `unlabeled_t`.

Currently, `sepgsql` allows security labels to be assigned to schemas, tables, columns, sequences, views, and functions. When `sepgsql` is in use, security labels are automatically assigned to supported database objects at creation time. This label is called a default security label, and is decided according to the system security policy, which takes as input the creator's label and the label assigned to the new object's parent object.

A new database object basically inherits the security label of the parent object, except when the security policy has special rules known as type-transition rules, in which case a different label may be applied. For schemas, the parent object is the current database; for tables, sequences, views, and functions, it is the containing schema; for columns, it is the containing table.

F.32.5.2. DML Permissions

For tables, `db_table:select`, `db_table:insert`, `db_table:update` or `db_table:delete` are checked for all the referenced target tables depending on the kind of statement; in addition, `db_table:select` is also checked for all the tables that contain columns referenced in the `WHERE` or `RETURNING` clause, as a data source for `UPDATE`, and so on.

Column-level permissions will also be checked for each referenced column. `db_column:select` is checked on not only the columns being read using `SELECT`, but those being referenced in other DML statements; `db_column:update` or `db_column:insert` will also be checked for columns being modified by `UPDATE` or `INSERT`.

For example, consider:

```
UPDATE t1 SET x = 2, y = md5sum(y) WHERE z = 100;
```

Here, `db_column:update` will be checked for `t1.x`, since it is being updated, `db_column:{select  update}` will be checked for `t1.y`, since it is both updated and referenced, and `db_column:select` will be checked for `t1.z`, since it is only referenced. `db_table:{select update}` will also be checked at the table level.

For sequences, `db_sequence:get_value` is checked when we reference a sequence object using `SELECT`; however, note that we do not currently check permissions on execution of corresponding functions such as `lastval()`.

For views, `db_view:expand` will be checked, then any other required permissions will be checked on the objects being expanded from the view, individually.

For functions, `db_procedure:{execute}` is defined, but is not checked in this version.

The client must be allowed to access all referenced tables and columns, even if they originated from views which were then expanded, so that we apply consistent access control rules independent of the manner in which the table contents are referenced.

The default database privilege system allows database superusers to modify system catalogs using DML commands, and reference or modify toast tables. These operations are prohibited when `sepgsql` is enabled.

F.32.5.3. DDL Permissions

SELinux defines several permissions to control common operations for each object type; such as creation, alter, drop and relabel of security label. In addition, several object types have special permissions to control their characteristic operations; such as addition or deletion of name entries within a particular schema.

When a `CREATE` command is executed, `create` will be checked on the object being constructed for each object types. A default security label will be assigned to the new database object, and the `create` permission will be checked on the pair of security label of the client and the new object itself. We consider CREATE TABLE to construct a table and underlying columns at the same time, so it requires the users to have permission to create both the table and its columns.

A few additional checks are applied depending on object types. On CREATE DATABASE, `getattr` permission will be checked on the source or template database of the new database, not only `create` on the new database. On creation of objects within a particular schema (tables, views, sequences and procedures), `add_name` will be also checked on the schema, not only `create` on the new object itself.

When `DROP` command is executed, `drop` will be checked on the object being removed for each object types. Permissions will be also checked for objects dropped indirectly via `CASCADE`. Deletion of objects contained within a particular schema (tables, views, sequences and procedures) additionally requires `remove_name` on the schema.

When SECURITY LABEL is executed, `setattr` and `relabelfrom` will be checked on the object being relabeled with its old security label, then `relabelto` with the supplied new security label.

In the case where multiple label providers are installed and the user tries to set a security label, but it is not managed by SELinux, only `setattr` should be checked here. This is currently not done due to implementation restrictions.

F.32.5.4. Trusted Procedures

Trusted procedures are similar to security definer functions or setuid commands. SELinux provides a feature to allow trusted code to run using a security label different from that of the client, generally for the purpose of providing highly controlled access to sensitive data (e.g. rows might be omitted, or the precision of stored values might be reduced). Whether or not a function acts as a trusted procedure is controlled by its security label and the operating system security policy. For example:

```
postgres=# CREATE TABLE customer (
               cid     int primary key,
               cname   text,
               credit  text
           );
CREATE TABLE
postgres=# SECURITY LABEL ON COLUMN customer.credit
               IS 'system_u:object_r:sepgsql_secret_table_t:s0';
SECURITY LABEL
postgres=# CREATE FUNCTION show_credit(int) RETURNS text
             AS 'SELECT regexp_replace(credit, ''-[0-9]+$'', ''-
xxxx'', ''g'')
                        FROM customer WHERE cid = $1'
           LANGUAGE sql;
CREATE FUNCTION
postgres=# SECURITY LABEL ON FUNCTION show_credit(int)
               IS 'system_u:object_r:sepgsql_trusted_proc_exec_t:
s0';
SECURITY LABEL
```

The above operations should be performed by an administrative user.

```
postgres=# SELECT * FROM customer;
ERROR:  SELinux: security policy violation
postgres=# SELECT cid, cname, show_credit(cid) FROM customer;
 cid | cname  |     show_credit
-----+--------+---------------------
   1 | taro   | 1111-2222-3333-xxxx
   2 | hanako | 5555-6666-7777-xxxx
(2 rows)
```

In this case, a regular user cannot reference `customer.credit` directly, but a trusted procedure `show_credit` allows him to print the credit card numbers of customers with some of the digits masked out.

F.32.5.5. Dynamic Domain Transitions

It is possible to use SELinux's dynamic domain transition feature to switch the security label of the client process, the client domain, to a new context, if that is allowed by the security policy. The client domain needs the `setcurrent` permission and also `dyntransition` from the old to the new domain.

Dynamic domain transitions should be considered carefully, because they allow users to switch their label, and therefore their privileges, at their option, rather than (as in the case of a trusted procedure) as mandated by the system. Thus, the `dyntransition` permission is only considered safe when used to switch to a domain with a smaller set of privileges than the original one. For example:

```
regression=# select sepgsql_getcon();
                    sepgsql_getcon
-------------------------------------------------------
 unconfined_u:unconfined_r:unconfined_t:s0-s0:c0.c1023
(1 row)

regression=# SELECT sepgsql_setcon('unconfined_u:unconfined_r:
unconfined_t:s0-s0:c1.c4');
 sepgsql_setcon
----------------
 t
```

```
(1 row)

regression=# SELECT sepgsql_setcon('unconfined_u:unconfined_r:
unconfined_t:s0-s0:c1.c1023');
ERROR:  SELinux: security policy violation
```

In this example above we were allowed to switch from the larger MCS range `c1.c1023` to the smaller range `c1.c4`, but switching back was denied.

A combination of dynamic domain transition and trusted procedure enables an interesting use case that fits the typical process life-cycle of connection pooling software. Even if your connection pooling software is not allowed to run most of SQL commands, you can allow it to switch the security label of the client using the `sepgsql_setcon()` function from within a trusted procedure; that should take some credential to authorize the request to switch the client label. After that, this session will have the privileges of the target user, rather than the connection pooler. The connection pooler can later revert the security label change by again using `sepgsql_setcon()` with `NULL` argument, again invoked from within a trusted procedure with appropriate permissions checks. The point here is that only the trusted procedure actually has permission to change the effective security label, and only does so when given proper credentials. Of course, for secure operation, the credential store (table, procedure definition, or whatever) must be protected from unauthorized access.

F.32.5.6. Miscellaneous

We reject the LOAD command across the board, because any module loaded could easily circumvent security policy enforcement.

F.32.6. Sepgsql Functions

表 F.29 shows the available functions.

表 F.29. Sepgsql Functions

`sepgsql_getcon() returns text`	Returns the client domain, the current security label of the client.
`sepgsql_setcon(text) returns bool`	Switches the client domain of the current session to the new domain, if allowed by the security policy. It also accepts `NULL` input as a request to transition to the client's original domain.
`sepgsql_mcstrans_in(text) returns text`	Translates the given qualified MLS/MCS range into raw format if the mcstrans daemon is running.
`sepgsql_mcstrans_out(text) returns text`	Translates the given raw MLS/MCS range into qualified format if the mcstrans daemon is running.
`sepgsql_restorecon(text) returns bool`	Sets up initial security labels for all objects within the current database. The argument may be NULL, or the name of a specfile to be used as alternative of the system default.

F.32.7. Limitations

Data Definition Language (DDL) Permissions

Due to implementation restrictions, some DDL operations do not check permissions.

Data Control Language (DCL) Permissions

Due to implementation restrictions, DCL operations do not check permissions.

Row-level access control

PostgreSQL does not support row-level access; therefore, `sepgsql` does not support it either.

Covert channels

`sepgsql` does not try to hide the existence of a certain object, even if the user is not allowed to reference it. For example, we can infer the existence of an invisible object as a result of primary key conflicts, foreign key violations, and so on, even if we cannot obtain the contents of the object. The existence of a top secret table cannot be hidden; we only hope to conceal its contents.

F.32.8. External Resources

SE-PostgreSQL Introduction[5]

This wiki page provides a brief overview, security design, architecture, administration and upcoming features.

Fedora SELinux User Guide[6]

This document provides a wide spectrum of knowledge to administer SELinux on your systems. It focuses primarily on Fedora, but is not limited to Fedora.

Fedora SELinux FAQ[7]

This document answers frequently asked questions about SELinux. It focuses primarily on Fedora, but is not limited to Fedora.

F.32.9. Author

KaiGai Kohei `<kaigai@ak.jp.nec.com>`

[5] https://wiki.postgresql.org/wiki/SEPostgreSQL
[6] http://docs.fedoraproject.org/selinux-user-guide/
[7] http://docs.fedoraproject.org/selinux-faq

F.33. spi

spi 模块提供了若干使用 SPI 和触发器的实用示例。虽然这些函数本身也有一定的独立用途，但把它们作为可按自身需求修改的示例来看更有价值。这些函数足够通用，可以用于任何表，但在创建触发器时必须指定表名和字段名（如下所述）。

下文描述的每组函数都作为一个可单独安装的扩展提供。

F.33.1. refint – 用于实现引用完整性的函数

`check_primary_key()`和`check_foreign_key()`用于检查外键约束。（当然，这项功能早已被内置的外键机制取代，不过该模块作为示例仍然有用。）

`check_primary_key()` 检查引用表。使用时，在引用其他表的表上创建一个使用该函数的 `BEFORE INSERT OR UPDATE` 触发器。触发器参数依次为：引用表中构成外键的列名、被引用表名，以及被引用表中构成主键/唯一键的列名。若要处理多个外键，请为每个引用分别创建一个触发器。

`check_foreign_key()` 检查被引用表。使用时，在被其他表引用的表上创建一个使用该函数的 `BEFORE DELETE OR UPDATE` 触发器。触发器参数依次为：该函数需要检查的引用表数量、发现引用键时采取的动作（`cascade` 表示删除引用行，`restrict` 表示若存在引用键则中止事务，`setnull` 表示将引用键字段设为空）、被引用表中构成主键/唯一键的列名，然后是引用表名和列名（按第一个参数指定的引用表数量重复提供）。请注意，主键/唯一键列应标记为 NOT NULL，并具有唯一索引。

`refint.example` 中有示例。

F.33.2. timetravel – 实现时间旅行的函数

很久以前，PostgreSQL曾内置时间旅行功能，为每个元组保存插入和删除时间。可以使用这些函数模拟这一功能。要使用这些函数，必须为表添加两个`abstime`类型的列，以存储元组插入的时间（start_date）和更改/删除的时间（stop_date）：

```
CREATE TABLE mytab (
        ...             ...
        start_date      abstime,
        stop_date       abstime
        ...             ...
);
```

这些列可以任意命名，不过在这里我们将它们称为 start_date 和 stop_date。

插入新行时，start_date 通常应设置为当前时间，stop_date 应设置为`infinity`。如果插入的数据在这些列中包含空值，触发器会自动将其替换为上述值。一般来说，只有在重新载入转储数据时，才应向这些列中插入显式的非空数据。

stop_date 等于`infinity`的元组“当前有效”，可以修改。stop_date 为有限值的元组则不能再修改，触发器会阻止这种操作。（如果需要这样做，可以按下文所示关闭时间旅行。）

对于可修改的行，更新时只会将被更新元组的 stop_date 改为当前时间，并插入一个包含修改后数据的新元组。新元组的 start_date 将设置为当前时间，stop_date 设置为 `infinity`。

删除操作实际上不会删除元组，而只会将其 stop_date 设置为当前时间。

要查询“当前有效”的元组，请在查询的 WHERE 条件中加入`stop_date = 'infinity'`。（你可能希望将其放入视图中。）类似地，通过为 start_date 和 stop_date 设置合适的条件，也可以查询在过去任何时间有效的元组。

timetravel()是支持此行为的通用触发器函数。在每个使用时间旅行的表上，使用此函数创建一个BEFORE INSERT OR UPDATE OR DELETE触发器。指定两个触发器参数：start_date 和 stop_date 列的实际名称。还可以选择指定一到三个额外参数，它们必须引用text类型的列。触发器会在 INSERT 时将当前用户名存入其中的第一列，在 UPDATE 时存入第二列，在 DELETE 时存入第三列。

set_timetravel()允许为表打开或关闭时间旅行。set_timetravel('mytab', 1)会为表mytab打开 TT。set_timetravel('mytab', 0)会为表mytab关闭 TT。两种情况都会报告原来的状态。在 TT 关闭时，可以自由修改 start_date 和 stop_date 列。请注意，开关状态仅作用于当前数据库会话；新会话总是从所有表的 TT 都打开的状态开始。

get_timetravel()返回表的 TT 状态，而不改变它。

timetravel.example中有一个示例。

F.33.3. autoinc – 用于字段自动递增的函数

autoinc()是一个将序列的下一个值存入整数字段的触发器。它与内置的“序列列”功能有一些重叠，但并不相同：autoinc()会覆盖插入时试图为该字段指定其他值的操作，而且还可以选择在更新时也递增该字段。

使用时，创建一个使用该函数的 BEFORE INSERT（或者可选的 BEFORE INSERT OR UPDATE）触发器。指定两个触发器参数：要修改的整型列名，以及提供这些值的序列对象名。（实际上，如果你希望更新多个自动递增列，也可以指定任意多组这样的名称对。）

autoinc.example 中有一个示例。

F.33.4. insert_username – 用于跟踪谁修改了表的函数

insert_username() 是一个将当前用户名存入文本字段的触发器。它有助于跟踪表中某一特定行最后一次是由谁修改的。

使用时，创建一个使用该函数的 BEFORE INSERT 和/或 UPDATE 触发器。指定一个触发器参数：要修改的文本列名。

insert_username.example 中有一个示例。

F.33.5. moddatetime – 用于跟踪最后修改时间的函数

moddatetime() 是一个将当前时间存入 timestamp 字段的触发器。它有助于跟踪表中某一特定行最后一次被修改的时间。

使用时，创建一个使用该函数的 BEFORE UPDATE 触发器。指定一个触发器参数：要修改的列名。该列必须是 timestamp 或 timestamp with time zone 类型。

moddatetime.example 中有一个示例。

F.34. sslinfo

`sslinfo` 模块提供关于当前客户端在连接到 PostgreSQL 时所提供 SSL 证书的信息。如果当前连接没有使用 SSL，则该模块没有用处（大多数函数将返回 NULL）。

除非安装时使用`--with-openssl`配置，否则此扩展根本无法构建。

F.34.1. 提供的函数

`ssl_is_used() returns boolean`

如果当前到服务器的连接使用 SSL，则返回 true，否则返回 false。

`ssl_version() returns text`

返回 SSL 连接使用的协议名称（例如 SSLv2、SSLv3 或 TLSv1）。

`ssl_cipher() returns text`

返回 SSL 连接所用密码的名称（例如 DHE-RSA-AES256-SHA）。

`ssl_client_cert_present() returns boolean`

如果当前客户端已向服务器提供了有效的 SSL 客户端证书，则返回 true，否则返回 false。（服务器可能配置为要求客户端证书，也可能没有这样配置。）

`ssl_client_serial() returns numeric`

返回当前客户端证书的序列号。
证书序列号与证书颁发者的组合可保证唯一标识一个证书（但不能保证唯一标识其所有者 – 所有者应当定期更换其密钥，并从颁发者处获取新证书）。

因此，如果你运行自己的 CA，并且只允许服务器接受由该 CA 签发的证书，那么序列号就是识别用户的最可靠手段（尽管并不便于记忆）。

`ssl_client_dn() returns text`

返回当前客户端证书的完整主题，并将字符数据转换为当前数据库编码。
假定如果你在证书名称中使用了非 ASCII 字符，你的数据库也能表示这些字符。如果你的数据库使用 SQL_ASCII 编码，名称中的非 ASCII 字符将表示为 UTF-8 序列。

结果类似于 `/CN=Somebody /C=Some country/O=Some organization`。

`ssl_issuer_dn() returns text`

返回当前客户端证书的完整颁发者名称，并将字符数据转换为当前数据库编码。
编码转换的处理方式与 `ssl_client_dn` 相同。

该函数的返回值与证书序列号的组合可唯一标识该证书。

只有在服务器的`root.crt`文件中包含多个受信任的 CA 证书，或者此 CA 颁发了一些中间证书颁发机构证书时，此函数才真正有用。

`ssl_client_dn_field(fieldname text) returns text`

该函数返回证书主题中指定字段的值；如果该字段不存在，则返回 NULL。字段名是字符串常量，它们使用 OpenSSL 对象数据库转换为 ASN1 对象标识符。可接受的值如下：

```
commonName (alias CN)
surname (alias SN)
```

```
name
givenName (alias GN)
countryName (alias C)
localityName (alias L)
stateOrProvinceName (alias ST)
organizationName (alias O)
organizationalUnitName (alias OU)
title
description
initials
postalCode
streetAddress
generationQualifier
description
dnQualifier
x500UniqueIdentifier
pseudonym
role
emailAddress
```

除 `commonName` 外，所有这些字段都是可选的。究竟包含哪些字段、哪些字段不包含，完全取决于你的 CA 策略。不过，这些字段的含义由 X.500 和 X.509 标准严格定义，因此你不能随意赋予它们任意含义。

`ssl_issuer_field(fieldname text) returns text`

与 `ssl_client_dn_field` 相同，只是针对证书颁发者而不是证书主题。

F.34.2. 作者

Victor Wagner `<vitus@cryptocom.ru>`, Cryptocom LTD

Cryptocom OpenSSL 开发组的电子邮件：`<openssl@cryptocom.ru>`

F.35. tablefunc

tablefunc 模块包含多种返回表（即多行结果）的函数。这些函数本身很有用，也可作为如何编写返回多行的 C 函数的示例。

F.35.1. 提供的函数

表 F.30总结了 tablefunc 模块提供的函数。

表 F.30. tablefunc函数

函数	返回值	描述
normal_rand(int numvals, float8 mean, float8 stddev)	setof float8	生成一组正态分布的随机值。
crosstab(text sql)	setof record	生成一个“透视表”，其中包含行名以及 *N* 个值列，其中 *N* 由调用查询中指定的行类型决定。
crosstab*N*(text sql)	setof table_crosstab_*N*	生成一个“透视表”，其中包含行名以及 *N* 个值列。crosstab2、crosstab3 和 crosstab4 是预定义的，但也可以按下文所述创建额外的 crosstab*N* 函数。
crosstab(text source_sql, text category_sql)	setof record	生成一个“透视表”，其值列由第二个查询指定。
crosstab(text sql, int N)	setof record	这是 crosstab(text) 的过时版本。参数 *N* 现已被忽略，因为值列的数量始终由调用查询决定。
connectby(text relname, text keyid_fld, text parent_keyid_fld [, text orderby_fld], text start_with, int max_depth [, text branch_delim])	setof record	生成层次树结构的表示形式。

F.35.1.1. normal_rand

```
normal_rand(int numvals, float8 mean, float8 stddev) returns setof
 float8
```

normal_rand 生成一组正态分布（高斯分布）的随机值。

numvals 是该函数要返回的值的个数。*mean* 是这些值所服从的正态分布的均值，*stddev* 是这些值所服从的正态分布的标准偏差。

例如，下面这个调用请求生成 1000 个值，均值为 5，标准偏差为 3：

```
test=# SELECT * FROM normal_rand(1000, 5, 3);
     normal_rand
----------------------
     1.56556322244898
     9.10040991424657
     5.36957140345079
   -0.369151492880995
    0.283600703686639
```

```
        .
        .
        .
     4.82992125404908
     9.71308014517282
     2.49639286969028
(1000 rows)
```

F.35.1.2. crosstab(text)

```
crosstab(text sql)
crosstab(text sql, int N)
```

该crosstab函数用于产生“透视”显示形式，其中数据沿页面横向排列，而不是纵向排列。例如，我们可能有这样的数据

```
row1    val11
row1    val12
row1    val13
...
row2    val21
row2    val22
row2    val23
...
```

并希望将它们显示成这样

```
row1    val11   val12   val13   ...
row2    val21   val22   val23   ...
...
```

该crosstab函数接受一个文本参数，该参数是一个 SQL 查询，用于产生按第一种方式格式化的原始数据，并生成一个按第二种方式格式化的表。

sql 参数是一个产生源数据集的 SQL 语句。该语句必须返回一个 row_name 列、一个 category 列和一个 value 列。*N* 是一个过时参数，即使提供也会被忽略（以前它必须与输出值列的个数匹配，但现在这一点由调用查询决定）。

例如，给出的查询可能会产生如下结果：

```
 row_name    cat    value
----------+-------+-------
  row1      cat1    val1
  row1      cat2    val2
  row1      cat3    val3
  row1      cat4    val4
  row2      cat1    val5
  row2      cat2    val6
  row2      cat3    val7
  row2      cat4    val8
```

该crosstab 函数声明的返回类型为 setof record，因此必须在 FROM 子句中定义输出列的实际名称和类型，该子句属于调用它的 SELECT 语句。例如：

```
SELECT * FROM crosstab('...') AS ct(row_name text, category_1 text,
 category_2 text);
```

此示例生成的集合类似于：

```
           <== value  columns  ==>
 row_name   category_1   category_2
```

```
----------+------------+------------
  row1        val1         val2
  row2        val5         val6
```

FROM 子句必须把输出定义为一个 row_name 列（其数据类型与 SQL 查询的第一列结果相同），其后跟着 N 个 value 列（它们的数据类型都与 SQL 查询的第三列结果相同）。可以按需设置任意数量的输出值列，输出列的名称也可自行决定。

crosstab 函数会为输入行中具有相同 row_name 值的每个连续分组生成一行输出。它会用这些行中的 value 字段从左到右填充输出的 value 列。如果某个分组中的行数少于输出 value 列的个数，多余的输出列会填充为空值；如果行数更多，多出来的输入行会被跳过。

实际上，SQL 查询应始终指定 ORDER BY 1,2，以确保输入行按正确顺序排列，也就是使具有相同 row_name 的值聚在一起，并在行内正确排序。注意，crosstab 本身并不会关注查询结果的第二列；该列只是为了排序而存在，用来控制第三列的值在页面上横向排列的顺序。

下面是一个完整示例：

```
CREATE TABLE ct(id SERIAL, rowid TEXT, attribute TEXT, value TEXT);
INSERT INTO ct(rowid, attribute, value) VALUES('test1','att1',
'val1');
INSERT INTO ct(rowid, attribute, value) VALUES('test1','att2',
'val2');
INSERT INTO ct(rowid, attribute, value) VALUES('test1','att3',
'val3');
INSERT INTO ct(rowid, attribute, value) VALUES('test1','att4',
'val4');
INSERT INTO ct(rowid, attribute, value) VALUES('test2','att1',
'val5');
INSERT INTO ct(rowid, attribute, value) VALUES('test2','att2',
'val6');
INSERT INTO ct(rowid, attribute, value) VALUES('test2','att3',
'val7');
INSERT INTO ct(rowid, attribute, value) VALUES('test2','att4',
'val8');

SELECT *
FROM crosstab(
  'select rowid, attribute, value
   from ct
   where attribute = ''att2'' or attribute = ''att3''
   order by 1,2')
AS ct(row_name text, category_1 text, category_2 text, category_3
 text);

 row_name | category_1 | category_2 | category_3
----------+------------+------------+------------
 test1    | val2       | val3       |
 test2    | val6       | val7       |
(2 rows)
```

可以定义一个在其定义中固定了所需输出行类型的自定义 crosstab 函数，以避免每次都必须写出用于定义输出列的 FROM 子句。下一节会介绍这种做法。另一种可能性是在视图定义中嵌入所需的 FROM 子句。

F.35.1.3. crosstab*N*(text)

crosstab*N*(text sql)

crosstab*N* 函数展示了如何为通用的 crosstab 函数设置自定义包装器，这样就不必在调用的 SELECT 查询中写出列名和类型。tablefunc 模块包含 crosstab2、crosstab3 和 crosstab4，其输出行类型定义为：

```
CREATE TYPE tablefunc_crosstab_N AS (
    row_name TEXT,
    category_1 TEXT,
    category_2 TEXT,
        .
        .
        .
    category_N TEXT
);
```

因此，当输入查询产生类型为 text 的 row_name 列和 value 列，并且需要 2、3 或 4 个输出值列时，这些函数可以直接使用。除此之外，它们在其他方面的行为与上文描述的通用 crosstab 函数完全相同。

例如，前一节中的示例也可以这样写：

```
SELECT *
FROM crosstab3(
  'select rowid, attribute, value
   from ct
   where attribute = ''att2'' or attribute = ''att3''
   order by 1,2');
```

这些函数主要用于说明。也可以基于底层的 crosstab() 函数创建自己的返回类型和函数。做法有两种：

- 创建一个描述所需输出列的复合类型，类似于 contrib/tablefunc/tablefunc--1.0.sql 中的示例。然后定义一个具有唯一名称的函数，该函数接受单个 text 参数并返回 setof your_type_name，但底层链接到同一个 crosstab C 函数。例如，如果源数据生成的行名是 text，值是 float8，而希望有 5 个值列：

  ```
  CREATE TYPE my_crosstab_float8_5_cols AS (
      my_row_name text,
      my_category_1 float8,
      my_category_2 float8,
      my_category_3 float8,
      my_category_4 float8,
      my_category_5 float8
  );

  CREATE OR REPLACE FUNCTION crosstab_float8_5_cols(text)
      RETURNS setof my_crosstab_float8_5_cols
      AS '$libdir/tablefunc','crosstab' LANGUAGE C STABLE STRICT;
  ```

- 使用 OUT 参数隐式定义返回类型。同一个示例也可以这样实现：

  ```
  CREATE OR REPLACE FUNCTION crosstab_float8_5_cols(
      IN text,
      OUT my_row_name text,
      OUT my_category_1 float8,
      OUT my_category_2 float8,
      OUT my_category_3 float8,
      OUT my_category_4 float8,
      OUT my_category_5 float8)
    RETURNS setof record
    AS '$libdir/tablefunc','crosstab' LANGUAGE C STABLE STRICT;
  ```

F.35.1.4. crosstab(text, text)

```
crosstab(text source_sql, text category_sql)
```

crosstab 单参数形式的主要限制在于，它会把同一组中的所有值一视同仁，并把每个值插入第一个可用列中。如果希望值列对应于特定的数据类别，
而某些分组可能没有某些类别的数据，这种形式就不太适用了。crosstab 的双参数形式通过提供一个与输出列对应的显式类别列表来处理这种情况。

source_sql 是一个产生源数据集的 SQL 语句。该语句必须返回一个 row_name 列、一个 category 列和一个 value 列。它也可以有一个或多个 “extra” 列。row_name 列必须位于第一列。category 列和 value 列必须是最后两列，并且顺序必须如此。位于 row_name 和 category 之间的任何列都被视为 “extra” 。对于具有相同 row_name 值的所有行，这些 “extra” 列预期应当相同。

例如，*source_sql*可能会产生如下结果：

```
SELECT row_name, extra_col, cat, value FROM foo ORDER BY 1;

 row_name    extra_col   cat    value
----------+------------+-----+---------
  row1         extra1    cat1    val1
  row1         extra1    cat2    val2
  row1         extra1    cat4    val4
  row2         extra2    cat1    val5
  row2         extra2    cat2    val6
  row2         extra2    cat3    val7
  row2         extra2    cat4    val8
```

*category_sql*是一个生成类别集合的 SQL 语句。该语句必须只返回一列。它必须至少生成一行，否则会产生错误。此外，它不得生成重复值，否则会产生错误。*category_sql*
可以是这样的：

```
SELECT DISTINCT cat FROM foo ORDER BY 1;
    cat
  -------
    cat1
    cat2
    cat3
    cat4
```

该crosstab 函数声明的返回类型为 setof record，因此必须在 FROM 子句中定义输出列的实际名称和类型，该子句属于调用它的 SELECT 语句。例如：

```
SELECT * FROM crosstab('...', '...')
    AS ct(row_name text, extra text, cat1 text, cat2 text, cat3
 text, cat4 text);
```

这将产生如下结果：

```
                  <==  value  columns   ==>
row_name   extra   cat1   cat2   cat3   cat4
---------+-------+------+------+------+------
  row1     extra1  val1   val2          val4
  row2     extra2  val5   val6   val7   val8
```

FROM 子句必须定义数量和数据类型都正确的输出列。如果 *source_sql* 查询结果有 *N* 列，那么其中前 *N*-2 列必须与前 *N*-2 个输出列相匹配。其余输出列必须具有 *source_sql* 查询结果最后一列的数据类型，并且这些输出列的数量必须与 *category_sql* 查询结果的行数完全相同。

crosstab 函数会为输入行中具有相同 row_name 值的每个连续分组生成一行输出。输出的 row_name 列以及任何 “extra” 列都从该分组的第一行复制而来。输出的 value 列则使用那些具有匹配 category 值的行中的 value 字段来填充。如果某一行的 category 与 *category_sql* 查询的任何输出都不匹配，那么它的 value 会被忽略。那些匹配类别未出现在该分组任何输入行中的输出列会填充为空值。

实际上，*source_sql* 查询应始终指定 ORDER BY 1，以确保具有相同 row_name 的值被聚在一起。不过，分组内类别的顺序并不重要。另外，必须确保 *category_sql* 查询输出的顺序与指定的输出列顺序一致。

下面给出两个完整示例：

```
create table sales(year int, month int, qty int);
insert into sales values(2007, 1, 1000);
insert into sales values(2007, 2, 1500);
insert into sales values(2007, 7, 500);
insert into sales values(2007, 11, 1500);
insert into sales values(2007, 12, 2000);
insert into sales values(2008, 1, 1000);

select * from crosstab(
  'select year, month, qty from sales order by 1',
  'select m from generate_series(1,12) m'
) as (
  year int,
  "Jan" int,
  "Feb" int,
  "Mar" int,
  "Apr" int,
  "May" int,
  "Jun" int,
  "Jul" int,
  "Aug" int,
  "Sep" int,
  "Oct" int,
  "Nov" int,
  "Dec" int
);
 year | Jan  | Feb  | Mar | Apr | May | Jun | Jul | Aug | Sep | Oct
 | Nov  | Dec
------+------+------+-----+-----+-----+-----+-----+-----+-----+----
-+------+------
 2007 | 1000 | 1500 |     |     |     |     | 500 |     |     |
 | 1500 | 2000
 2008 | 1000 |      |     |     |     |     |     |     |     |
 |      |
(2 rows)

CREATE TABLE cth(rowid text, rowdt timestamp, attribute text, val
 text);
INSERT INTO cth VALUES('test1','01 March 2003','temperature','42');
INSERT INTO cth VALUES('test1','01 March 2003','test_result',
'PASS');
INSERT INTO cth VALUES('test1','01 March 2003','volts','2.6987');
INSERT INTO cth VALUES('test2','02 March 2003','temperature','53');
INSERT INTO cth VALUES('test2','02 March 2003','test_result',
'FAIL');
INSERT INTO cth VALUES('test2','02 March 2003','test_startdate','01
 March 2003');
```

```
INSERT INTO cth VALUES('test2','02 March 2003','volts','3.1234');

SELECT * FROM crosstab
(
  'SELECT rowid, rowdt, attribute, val FROM cth ORDER BY 1',
  'SELECT DISTINCT attribute FROM cth ORDER BY 1'
)
AS
(
       rowid text,
       rowdt timestamp,
       temperature int4,
       test_result text,
       test_startdate timestamp,
       volts float8
);
 rowid |          rowdt           | temperature | test_result |
  test_startdate      | volts
-------+--------------------------+-------------+-------------+----
----------------------+--------
 test1 | Sat Mar 01 00:00:00 2003 |          42 | PASS        |
                      | 2.6987
 test2 | Sun Mar 02 00:00:00 2003 |          53 | FAIL        | Sat
 Mar 01 00:00:00 2003 | 3.1234
(2 rows)
```

可以创建预定义函数，以避免在每个查询中都写出结果列的名称和类型。
请参见前一节中的示例。这种形式的 `crosstab` 所对应的底层 C 函数名为 `crosstab_hash`。

F.35.1.5. connectby

```
connectby(text relname, text keyid_fld, text parent_keyid_fld
          [, text orderby_fld ], text start_with, int max_depth
          [, text branch_delim ])
```

`connectby` 函数会显示存储在表中的层次数据。该表必须有一个唯一标识各行的键字段，以及一个引用每一行的父行（如果有）的父键字段。`connectby` 可以显示从任意一行开始向下展开的子树。

表 F.31解释了这些参数。

表 F.31. connectby参数

参数	描述
relname	源关系名称
keyid_fld	键字段名称
parent_keyid_fld	父键字段名称
orderby_fld	用于对同级节点排序的字段名称（可选）
start_with	起始行的键值
max_depth	向下遍历的最大深度，零表示深度不受限制
branch_delim	在分支输出中分隔各键值的字符串（可选）

键字段和父键字段可以是任意数据类型，但它们必须是同一类型。注意，
无论键字段的类型是什么，*start_with* 值都必须作为文本字符串输入。

connectby 函数被声明为返回 setof record，因此输出列的实际名称和类型必须在调用 SELECT 语句的 FROM 子句中定义，例如：

```
SELECT * FROM connectby('connectby_tree', 'keyid', 'parent_keyid',
 'pos', 'row2', 0, '~')
    AS t(keyid text, parent_keyid text, level int, branch text, pos
 int);
```

前两个输出列用于当前行的键和其父行的键，它们必须与该表键字段的类型匹配。
第三个输出列表示树中的深度，必须是 integer 类型。如果给出了 *branch_delim* 参数，下一个输出列就是分支路径显示，必须是 text 类型。最后，如果给出了 *orderby_fld* 参数，最后一个输出列就是一个序列号，必须是 integer 类型。

“branch” 输出列显示了到达当前行所经过的键路径。各键之间用指定的 *branch_delim* 字符串分隔。如果不需要分支显示，则在输出列列表中同时省略 *branch_delim* 参数和 branch 列即可。

如果同一父节点下各同级节点的顺序很重要，可以包含 *orderby_fld* 参数来指定按哪个字段对同级节点排序。该字段可以是任何可排序的数据类型。当且仅当指定了 *orderby_fld* 时，输出列列表才必须包含最后那个整数类型的序列号列。

表名和字段名参数会原样复制到 connectby 在内部生成的 SQL 查询中。因此，如果名称是大小写混合的，或者包含特殊字符，就应包含双引号。还可能需要对表名进行模式限定。

在大表中，除非父键字段上建有索引，否则性能会很差。

重要的是，*branch_delim* 字符串不要出现在任何键值中，否则 connectby 可能会错误地报告无限递归错误。注意，如果没有提供 *branch_delim*，为了递归检测会使用默认值 ~。

下面是一个示例：

```
CREATE TABLE connectby_tree(keyid text, parent_keyid text, pos int)
;

INSERT INTO connectby_tree VALUES('row1',NULL, 0);
INSERT INTO connectby_tree VALUES('row2','row1', 0);
INSERT INTO connectby_tree VALUES('row3','row1', 0);
INSERT INTO connectby_tree VALUES('row4','row2', 1);
INSERT INTO connectby_tree VALUES('row5','row2', 0);
INSERT INTO connectby_tree VALUES('row6','row4', 0);
INSERT INTO connectby_tree VALUES('row7','row3', 0);
INSERT INTO connectby_tree VALUES('row8','row6', 0);
INSERT INTO connectby_tree VALUES('row9','row5', 0);

-- with branch, without orderby_fld (order of results is not
 guaranteed)
SELECT * FROM connectby('connectby_tree', 'keyid', 'parent_keyid',
 'row2', 0, '~')
 AS t(keyid text, parent_keyid text, level int, branch text);
 keyid | parent_keyid | level |       branch
-------+--------------+-------+----------------------
 row2  |              |     0 | row2
 row4  | row2         |     1 | row2~row4
 row6  | row4         |     2 | row2~row4~row6
 row8  | row6         |     3 | row2~row4~row6~row8
 row5  | row2         |     1 | row2~row5
 row9  | row5         |     2 | row2~row5~row9
(6 rows)
```

```
-- without branch, without orderby_fld (order of results is not
 guaranteed)
SELECT * FROM connectby('connectby_tree', 'keyid', 'parent_keyid',
 'row2', 0)
 AS t(keyid text, parent_keyid text, level int);
 keyid | parent_keyid | level
-------+--------------+-------
 row2  |              |     0
 row4  | row2         |     1
 row6  | row4         |     2
 row8  | row6         |     3
 row5  | row2         |     1
 row9  | row5         |     2
(6 rows)

-- with branch, with orderby_fld (notice that row5 comes before
 row4)
SELECT * FROM connectby('connectby_tree', 'keyid', 'parent_keyid',
 'pos', 'row2', 0, '~')
 AS t(keyid text, parent_keyid text, level int, branch text, pos
 int);
 keyid | parent_keyid | level |       branch        | pos
-------+--------------+-------+---------------------+-----
 row2  |              |     0 | row2                |   1
 row5  | row2         |     1 | row2~row5           |   2
 row9  | row5         |     2 | row2~row5~row9      |   3
 row4  | row2         |     1 | row2~row4           |   4
 row6  | row4         |     2 | row2~row4~row6      |   5
 row8  | row6         |     3 | row2~row4~row6~row8 |   6
(6 rows)

-- without branch, with orderby_fld (notice that row5 comes before
 row4)
SELECT * FROM connectby('connectby_tree', 'keyid', 'parent_keyid',
 'pos', 'row2', 0)
 AS t(keyid text, parent_keyid text, level int, pos int);
 keyid | parent_keyid | level | pos
-------+--------------+-------+-----
 row2  |              |     0 |   1
 row5  | row2         |     1 |   2
 row9  | row5         |     2 |   3
 row4  | row2         |     1 |   4
 row6  | row4         |     2 |   5
 row8  | row6         |     3 |   6
(6 rows)
```

F.35.2. 作者

Joe Conway

F.36. tcn

tcn模块提供一个触发器函数，用于将其所附着任意表的内容变更通知给监听者。它必须作为AFTER触发器，并以FOR EACH ROW方式使用。

在CREATE TRIGGER语句中，至多可以为该函数提供一个参数，而且该参数是可选的。如果提供该参数，它将用作通知的通道名；如果省略，则使用tcn作为通道名。

通知的载荷由表名、一个用于指示执行了哪种操作的字母，以及主键列的列名/值对组成。各部分之间都以逗号分隔。为了便于用正则表达式解析，表名和列名始终用双引号括起，数据值始终用单引号括起。嵌入的引号会被双写。

下面是一个使用该扩展的简短示例。

```
test=# create table tcndata
test-#   (
test(#     a int not null,
test(#     b date not null,
test(#     c text,
test(#     primary key (a, b)
test(#   );
NOTICE:  CREATE TABLE / PRIMARY KEY will create implicit index
 "tcndata_pkey" for table "tcndata"
CREATE TABLE
test=# create trigger tcndata_tcn_trigger
test-#   after insert or update or delete on tcndata
test-#   for each row execute procedure triggered_change_
notification();
CREATE TRIGGER
test=# listen tcn;
LISTEN
test=# insert into tcndata values (1, date '2012-12-22', 'one'),
test-#                            (1, date '2012-12-23', 'another')
,
test-#                            (2, date '2012-12-23', 'two');
INSERT 0 3
Asynchronous notification "tcn" with payload ""tcndata",I,"a"='1',
"b"='2012-12-22'" received from server process with PID 22770.
Asynchronous notification "tcn" with payload ""tcndata",I,"a"='1',
"b"='2012-12-23'" received from server process with PID 22770.
Asynchronous notification "tcn" with payload ""tcndata",I,"a"='2',
"b"='2012-12-23'" received from server process with PID 22770.
test=# update tcndata set c = 'uno' where a = 1;
UPDATE 2
Asynchronous notification "tcn" with payload ""tcndata",U,"a"='1',
"b"='2012-12-22'" received from server process with PID 22770.
Asynchronous notification "tcn" with payload ""tcndata",U,"a"='1',
"b"='2012-12-23'" received from server process with PID 22770.
test=# delete from tcndata where a = 1 and b = date '2012-12-22';
DELETE 1
Asynchronous notification "tcn" with payload ""tcndata",D,"a"='1',
"b"='2012-12-22'" received from server process with PID 22770.
```

F.37. test_parser

test_parser是一个全文检索自定义解析器的示例。它并不做任何特别有用的事情，但可以作为开发你自己的解析器的起点。

test_parser识别由空白分隔的单词，并且只返回两种词元类型：

```
mydb=# SELECT * FROM ts_token_type('testparser');
 tokid | alias |  description
-------+-------+---------------
     3 | word  | Word
    12 | blank | Space symbols
(2 rows)
```

这些词元编号是特意选择为与默认解析器的编号兼容的。这使我们可以使用它的headline()函数，从而让示例保持简单。

F.37.1. 用法

安装test_parser扩展会创建一个文本搜索解析器testparser。它没有可供用户配置的参数。

你可以用如下方式测试这个解析器：

```
mydb=# SELECT * FROM ts_parse('testparser', 'That''s my first own
 parser');
 tokid | token
-------+--------
     3 | That's
    12 |
     3 | my
    12 |
     3 | first
    12 |
     3 | own
    12 |
     3 | parser
```

实际使用时需要建立一个使用该解析器的文本搜索配置。例如：

```
mydb=# CREATE TEXT SEARCH CONFIGURATION testcfg ( PARSER =
 testparser );
CREATE TEXT SEARCH CONFIGURATION

mydb=# ALTER TEXT SEARCH CONFIGURATION testcfg
mydb-#   ADD MAPPING FOR word WITH english_stem;
ALTER TEXT SEARCH CONFIGURATION

mydb=#  SELECT to_tsvector('testcfg', 'That''s my first own
 parser');
          to_tsvector
-------------------------------
 'that':1 'first':3 'parser':5
(1 row)

mydb=# SELECT ts_headline('testcfg', 'Supernovae stars are the
 brightest phenomena in galaxies',
mydb(#                    to_tsquery('testcfg', 'star'));
```

```
                          ts_headline
-------------------------------------------------------------------
 Supernovae <b>stars</b> are the brightest phenomena in galaxies
(1 row)
```

F.38. tsearch2

tsearch2 模块为那些在文本搜索于 8.3 版集成进 PostgreSQL 核心之前使用过 tsearch2 的应用提供向后兼容的文本搜索功能。

F.38.1. 可移植性问题

尽管内置的文本搜索特性以 tsearch2 为基础，并且与它大体相似，但仍存在许多细微差异，会给现有应用带来可移植性问题：

- 一些函数的名称发生了变化，例如 `rank` 改为 `ts_rank`。替代的 `tsearch2` 模块提供了使用旧名称的别名。
- 内置文本搜索的数据类型和函数都存在于系统模式 `pg_catalog` 中。在使用 tsearch2 的安装中，这些对象通常位于 `public` 模式，不过也有些用户选择把它们放在自己单独的模式里。因此，无论哪种情况，对这些对象的显式模式限定引用都会失败。替代的 `tsearch2` 模块提供了存储在 `public`（必要时也可放在其他模式）中的别名对象，使这类引用仍能工作。
- 内置文本搜索特性中没有“当前解析器”或“当前词典”的概念，只有当前搜索配置（由 `default_text_search_config` 参数设置）的概念。虽然当前解析器和当前词典只被用于调试目的的函数使用，但在某些情况下这仍可能构成移植障碍。替代的 `tsearch2` 模块模拟了这些额外的状态变量，并提供用于设置和获取它们的向后兼容函数。

还有一些问题是替代的 `tsearch2` 模块没有解决的，因而无论如何都需要修改应用代码：

- 旧的 `tsearch2` 触发器函数允许其参数列表中的项是函数名，这些函数会在文本数据被转换为 `tsvector` 格式之前对其调用。这项功能因为属于安全漏洞而被移除，因为无法保证被调用的函数就是原本想要调用的那个。如果数据在建立索引之前必须经过处理，推荐的做法是编写一个自定义触发器自行完成这项工作。
- 文本搜索配置信息已被移入核心系统目录中，这些目录与 tsearch2 使用的表明显不同。任何检查或修改那些表的应用都需要调整。
- 如果应用使用了任何自定义文本搜索配置，就需要使用新的文本搜索配置 SQL 命令在核心目录中重建它们。替代的 `tsearch2` 模块对此提供了一点支持：它使旧的一组 tsearch2 配置表可以装载到 PostgreSQL 8.3 中。（如果没有该模块，配置数据将无法装载，因为 `regprocedure` 列中的值无法解析为函数。）虽然这些配置表实际上不会做任何事情，但至少在 8.3 中设置等效的自定义配置时可以查阅它们的内容。
- 不支持旧的 `reset_tsearch()` 和 `get_covers()` 函数。
- 替代的 `tsearch2` 模块没有定义任何别名操作符，而是完全依赖内置操作符。只有当应用使用了显式模式限定的操作符名称时这才会成为问题，而这种情况非常少见。

F.38.2. 转换 8.3 之前的安装

更新一个使用 tsearch2 的 8.3 之前安装的推荐方法是：

1. 按通常方式从旧安装中做一个转储，但务必不要使用 pg_dump 或 pg_dumpall 的 `-c`（`--clean`）选项。
2. 在新安装中创建空数据库，并将替代的 `tsearch2` 模块安装到每个要使用文本搜索的数据库中。这必须在装载转储数据之前完成！如果你的旧安装把 tsearch2 对象放在了 `public` 以外的模式中，请务必调整 `CREATE EXTENSION` 命令，使替代对象创建在同一模式中。

3. 装载转储数据。由于无法重新创建原始的 tsearch2 对象，会报告相当多的错误。这些错误可以忽略，但这意味着你不能在单个事务中恢复转储（例如，不能使用 pg_restore 的 `-1` 开关）。

4. 检查恢复出的 tsearch2 配置表（`pg_ts_cfg` 等）的内容，并按需创建等效的内置文本搜索配置。从中提取出所有有用信息之后，就可以删除旧的配置表。

5. 测试你的应用。

之后你也许希望把应用中对别名文本搜索对象的引用改名，以便最终卸载替代的 `tsearch2` 模块。

F.38.3. 参考文献

Tsearch2 开发站点 http://www.sai.msu.su/~megera/postgres/gist/tsearch/V2/

F.39. unaccent

unaccent是一个文本检索词典，它会从词位中移除重音符号（变音符号）。
它是一个过滤词典，也就是说，它的输出总会被传递给下一个词典（如果有），
这不同于词典的通常行为。这使得全文检索能够以不区分重音的方式处理文本。

unaccent当前的实现还不能作为thesaurus词典的正规化词典使用。

F.39.1. 配置

unaccent词典接受下列选项：

- RULES是包含转换规则列表的文件的基名。该文件必须存放在$SHAREDIR/tsearch_data/中（其中$SHAREDIR表示PostgreSQL安装的共享数据目录）。其名称必须以.rules结尾，但在RULES参数中不应包含这个后缀。

规则文件的格式如下：

- 每一行表示一对字符，由一个带重音的字符和一个不带重音的字符组成。
  前者会被转换成后者。例如：

```
À        A
Á        A
Â        A
Ã        A
Ä        A
Å        A
Æ        A
```

一个更完整并且对大多数欧洲语言都可直接使用的示例可见于unaccent.rules。
安装unaccent模块时，该文件会被安装到$SHAREDIR/tsearch_data/中。

F.39.2. 用法

安装unaccent扩展会创建一个文本检索模板unaccent
以及一个基于该模板的词典unaccent。unaccent词典的默认参数设置是RULES='unaccent'，因此它可立即配合标准的unaccent.rules文件使用。如果愿意，
也可以修改这个参数，例如

```
mydb=# ALTER TEXT SEARCH DICTIONARY unaccent (RULES='my_rules');
```

或者基于该模板创建新的词典。

要测试该词典，可以尝试：

```
mydb=# select ts_lexize('unaccent','Hôtel');
 ts_lexize
-----------
 {Hotel}
(1 row)
```

下面的示例展示了如何将unaccent词典插入到文本检索配置中：

```
mydb=# CREATE TEXT SEARCH CONFIGURATION fr ( COPY = french );
mydb=# ALTER TEXT SEARCH CONFIGURATION fr
        ALTER MAPPING FOR hword, hword_part, word
        WITH unaccent, french_stem;
mydb=# select to_tsvector('fr','Hôtels de la Mer');
```

```
    to_tsvector
-------------------
 'hotel':1 'mer':4
(1 row)

mydb=# select to_tsvector('fr','Hôtel de la Mer') @@ to_tsquery(
'fr','Hotels');
 ?column?
----------
 t
(1 row)

mydb=# select ts_headline('fr','Hôtel de la Mer',to_tsquery('fr',
'Hotels'));
      ts_headline
------------------------
 <b>Hôtel</b> de la Mer
(1 row)
```

F.39.3. 函数

`unaccent()`函数会从给定字符串中移除重音符号（变音符号）。从本质上说，它是对`unaccent`这一类型词典的一个包装器，但也可以在常规文本检索环境之外使用。

```
unaccent([dictionary, ] string) returns text
```

例如：

```
SELECT unaccent('unaccent', 'Hôtel');
SELECT unaccent('Hôtel');
```

F.40. uuid-ossp

uuid-ossp模块提供一些函数，可使用几种标准算法之一生成通用唯一标识符（UUID）。它还提供一些函数，用于生成某些特殊的 UUID 常量。

此模块依赖 OSSP UUID 库，该库位于http://www.ossp.org/pkg/lib/uuid/。

F.40.1. uuid-ossp 函数

表 F.32显示了可用于生成 UUID 的函数。相关标准 ITU-T Rec. X.667、ISO/IEC 9834-8:2005 和 RFC 4122 规定了生成 UUID 的四种算法，由版本号 1、3、4 和 5 标识。（不存在版本 2 算法。）每种算法可能适用于不同的一组应用。

表 F.32. 用于生成 UUID 的函数

函数	描述
uuid_generate_v1()	此函数生成第 1 版 UUID。这需要使用计算机的 MAC 地址和时间戳。请注意，这类 UUID 会泄露创建该标识符的计算机身份及创建时间，因此可能不适合某些对安全敏感的应用。
uuid_generate_v1mc()	此函数生成第 1 版 UUID，但使用随机多播 MAC 地址，而不是计算机的真实 MAC 地址。
uuid_generate_v3(namespace uuid, name text)	此函数在给定命名空间中使用指定输入名称生成第 3 版 UUID。命名空间应是表 F.33中所示uuid_ns_*()函数生成的特殊常量之一。（理论上它可以是任意 UUID。）名称是所选命名空间中的一个标识符。 例如： SELECT uuid_generate_v3(uuid_ns_url(), 'http://www.postgresql.org'); 名称参数会先做 MD5 哈希，因此无法从生成的 UUID 中反推出明文内容。通过这种方法生成 UUID 不包含随机因素，也不依赖于环境，因此结果可重现。
uuid_generate_v4()	此函数生成第 4 版 UUID，它完全由随机数派生。
uuid_generate_v5(namespace uuid, name text)	此函数生成第 5 版 UUID，其工作方式类似于第 3 版 UUID，但使用 SHA-1 作为哈希方法。由于 SHA-1 被认为比 MD5 更安全，应优先使用第 5 版而不是第 3 版。

表 F.33. 返回 UUID 常量的函数

uuid_nil()	一个 “nil” UUID 常量，不会作为真实 UUID 出现。
uuid_ns_dns()	指定 UUID 的 DNS 命名空间的常量。
uuid_ns_url()	指定 UUID 的 URL 命名空间的常量。
uuid_ns_oid()	指定 UUID 的 ISO 对象标识符（OID）命名空间的常量。（这涉及 ASN.1 OID，与PostgreSQL使用的 OID 无关。）
uuid_ns_x500()	指定 UUID 的 X.500 可辨识名称（DN）命名空间的常量。

F.40.2. 作者

Peter Eisentraut <peter_e@gmx.net>

F.41. xml2

`xml2` 模块提供 XPath 查询与 XSLT 功能。

F.41.1. 弃用说明

自 PostgreSQL 8.3 起，核心服务器就提供了基于 SQL/XML 标准的 XML 相关功能。该功能覆盖了 XML 语法检查和 XPath 查询，也就是本模块所做的事情，而且能力更多；不过，两者的 API 完全不兼容。计划在未来的 PostgreSQL 版本中移除此模块，转而采用较新的标准 API，因此建议你尝试迁移应用程序。如果你发现本模块的某些功能在较新的 API 中还没有以足够完善的形式提供，请向 `<pgsql-hackers@postgresql.org>` 说明你的问题，以便补齐这一不足。

F.41.2. 函数说明

表 F.34显示了该模块提供的函数。这些函数提供简单直接的 XML 解析和 XPath 查询功能。所有参数的类型都是 `text`，为简洁起见，此处不再显示。

表 F.34. 函数

函数	返回值	描述
`xml_is_well_formed(document)`	`bool`	此函数解析其参数中的文档文本，如果文档是良构 XML，则返回 true。（注意：在 PostgreSQL 8.2 之前，此函数名为 `xml_valid()`。由于 XML 中有效性和良构性具有不同含义，那是个错误的名称。旧名称仍然可用，但已被弃用。）
`xpath_string(document, query)`	`text`	这些函数计算所提供文档上的 XPath 查询，并将结果转换为指定类型。
`xpath_number(document, query)`	`float4`	
`xpath_bool(document, query)`	`bool`	

`xpath_nodeset(document, query, toptag, itemtag)`	`text`	该函数计算文档上的查询，并将结果封装在 XML 标签中。如果结果有多个值，输出将如下所示：
		
		`<toptag>`
		`<itemtag>Value 1`（可以是 XML 片段）`</itemtag>`
		`<itemtag>Value 2....</itemtag>`
		`</toptag>`
		
		如果`toptag`或`itemtag`为空字符串，就会省略相应的标签。

`xpath_nodeset(document, query)`	`text`	与 `xpath_nodeset(document, query, toptag, itemtag)` 相同，但结果省略这两个标签。
`xpath_nodeset(document, query, itemtag)`	`text`	与`xpath_nodeset(document, query, toptag, itemtag)`类似，但结果省略toptag。
`xpath_list(document, query, separator)`	`text`	此函数返回由指定分隔符分隔的多个值，例如，如果分隔符为`,`

函数	返回值	描述
		则返回 `Value 1,Value 2, Value 3`。
`xpath_list(document, query)`	`text`	这是上一个函数的包装器，使用 , 作为分隔符。

F.41.3. xpath_table

```
xpath_table(text key, text document, text relation, text xpaths,
 text criteria) returns setof record
```

`xpath_table` 是一个表函数，它对一组文档中的每个文档执行一组 XPath 查询，并将结果作为表返回。原始文档表中的主键字段会作为结果的第一列返回，
因此结果集可以方便地用于连接。参数说明见表 F.35。

表 F.35. xpath_table 参数

参数	描述
key	“键”字段的名称 – 这只是一个作为输出表第一列使用的字段，也就是说，它用于标识每个输出行来自哪条记录（关于多个值，见下文注释）
document	包含 XML 文档的字段名
relation	包含这些文档的表或视图名
xpaths	一个或多个 XPath 表达式，用 `|` 分隔
criteria	WHERE 子句的内容。这一项不能省略，因此如果你想处理该表或视图中的所有行，请使用 `true` 或 `1=1`

这些参数（XPath 字符串除外）只是直接替换进一个普通的 SQL SELECT 语句中，因此你有一定的灵活性 – 该语句为

```
SELECT <key>, <document> FROM <relation> WHERE <criteria>
```

因此，在这些特定位置上，任何合法内容都可以使用。该 SELECT 的结果必须恰好返回两列（除非你试图为 key 或 document 列出多个字段，否则都会如此）。请注意，这种简化做法要求你验证任何用户提供的值，以避免 SQL 注入攻击。

该函数必须用在 `FROM` 表达式中，并带有一个 `AS` 子句来指定输出列，例如

```
SELECT * FROM
xpath_table('article_id',
            'article_xml',
            'articles',
            '/article/author|/article/pages|/article/title',
            'date_entered > ''2003-01-01'' ')
AS t(article_id integer, author text, page_count integer, title
 text);
```

`AS` 子句定义了输出表中各列的名称和类型。第一列是“键”字段，其余各列对应 XPath 查询。如果 XPath 查询多于结果列，多出的查询将被忽略。如果结果列多于 XPath 查询，多出的列将为 NULL。

注意，这个示例把 `page_count` 结果列定义为整数。该函数内部只处理字符串表示，因此当你在输出中声明整数时，它会取 XPath 结果的字符串表示，并使用 PostgreSQL 输入函数将其转换为整数（或者 `AS` 子句要求的任何其他类型）。如果做不到这一点 – 例如结果为空 – 就会报错，因此如果你认为数据可能有问题，最好还是把列类型设为 `text`。

调用该函数的 SELECT 语句不一定非得只是 SELECT * – 它可以按名称引用输出列，或者将它们与其他表连接。该函数会生成一个虚拟表，你可以对它执行任何想要的操作（例如聚合、连接、排序等）。因此，更复杂一点的例子还可以是：

```
SELECT t.title, p.fullname, p.email
FROM xpath_table('article_id', 'article_xml', 'articles',
                 '/article/title|/article/author/@id',
                 'xpath_string(article_xml,''/article/@date'') >
 ''2003-03-20'' ')
       AS t(article_id integer, title text, author_id integer),
     tblPeopleInfo AS p
WHERE t.author_id = p.person_id;
```

当然，出于方便，你也可以把这一切封装到一个视图中。

F.41.3.1. 多值结果

xpath_table 函数假定每个 XPath 查询的结果都可能有多个值，因此该函数返回的行数可能不同于输入文档的数量。返回的第一行包含每个查询的第一个结果，
第二行包含每个查询的第二个结果。如果某个查询的值少于其他查询，
则会在相应位置返回空值。

在某些情况下，用户会知道某个 XPath 查询只会返回单个结果（例如唯一文档标识符）– 如果它与返回多个结果的 XPath 查询一起使用，这个单值结果只会出现在结果的第一行。解决办法是把键字段作为与一个更简单的 XPath 查询结果进行连接的条件之一。例如：

```
CREATE TABLE test (
    id int PRIMARY KEY,
    xml text
);

INSERT INTO test VALUES (1, '<doc num="C1">
<line num="L1"><a>1</a><b>2</b><c>3</c></line>
<line num="L2"><a>11</a><b>22</b><c>33</c></line>
</doc>');

INSERT INTO test VALUES (2, '<doc num="C2">
<line num="L1"><a>111</a><b>222</b><c>333</c></line>
<line num="L2"><a>111</a><b>222</b><c>333</c></line>
</doc>');

SELECT * FROM
  xpath_table('id','xml','test',
              '/doc/@num|/doc/line/@num|/doc/line/a|/doc/line/b|/
doc/line/c',
              'true')
  AS t(id int, doc_num varchar(10), line_num varchar(10), val1 int,
 val2 int, val3 int)
WHERE id = 1 ORDER BY doc_num, line_num

 id | doc_num | line_num | val1 | val2 | val3
----+---------+----------+------+------+------
  1 | C1      | L1       |    1 |    2 |    3
  1 |         | L2       |   11 |   22 |   33
```

要让 doc_num 在每一行中都出现，解决办法是调用两次 xpath_table 并连接结果：

```
SELECT t.*,i.doc_num FROM
  xpath_table('id', 'xml', 'test',
```

```
                '/doc/line/@num|/doc/line/a|/doc/line/b|/doc/line/c',
                'true')
    AS t(id int, line_num varchar(10), val1 int, val2 int, val3
 int),
  xpath_table('id', 'xml', 'test', '/doc/@num', 'true')
    AS i(id int, doc_num varchar(10))
WHERE i.id=t.id AND i.id=1
ORDER BY doc_num, line_num;

 id | line_num | val1 | val2 | val3 | doc_num
----+----------+------+------+------+---------
  1 | L1       |    1 |    2 |    3 | C1
  1 | L2       |   11 |   22 |   33 | C1
(2 rows)
```

F.41.4. XSLT 函数

如果安装了 libxslt，则可使用下列函数：

F.41.4.1. xslt_process

```
xslt_process(text document, text stylesheet, text paramlist)
 returns text
```

这个函数将 XSL 样式表应用到文档上，并返回转换后的结果。`paramlist` 是转换时要使用的参数赋值列表，以 `a=1,b=2` 的形式指定。注意，参数解析非常简单：参数值中不能包含逗号！

此外还有 `xslt_process` 的双参数版本，它不会向转换传递任何参数。

F.41.5. 作者

John Gray `<jgray@azuli.co.uk>`

本模块由 Torchbox Ltd. (www.torchbox.com) 赞助开发。它采用与 PostgreSQL 相同的 BSD 许可证。

附录 G. 额外提供的程序

本附录与前一个附录包含有关 PostgreSQL 发行版中 `contrib` 目录内模块的信息。关于 `contrib` 部分的一般性信息，以及 `contrib` 中服务器扩展和插件的具体信息，请参见附录 F。

本附录介绍位于 `contrib` 中的实用程序。无论是通过源码还是打包系统安装，它们安装后都会出现在 PostgreSQL 安装的 `bin` 目录中，并可像其他任何程序一样使用。

G.1. 客户端应用

本节介绍 `contrib` 中的 PostgreSQL 客户端应用程序。无论数据库服务器位于何处，都可以从任何位置运行它们。另见PostgreSQL Client Applications，其中介绍了属于核心 PostgreSQL 发行版一部分的客户端应用程序。

oid2name

oid2name — 解析 PostgreSQL 数据目录中的 OID 和 filenode

大纲

`oid2name` [*option*...]

描述

oid2name 是一个帮助管理员检查 PostgreSQL 所用文件结构的实用程序。要使用它，你需要熟悉数据库文件结构，这在第 56 章中有说明。

> **注意**
>
> “oid2name” 这个名称是历史遗留，实际上颇具误导性，
> 因为多数时候你真正关心的是表的 filenode 编号（也就是数据库目录中可见的文件名）。务必理解表 OID 与表 filenode 之间的区别。

oid2name 连接到目标数据库，并提取 OID、filenode 和/或表名信息。你也可以让它显示数据库 OID 或表空间 OID。

选项

oid2name接受以下命令行参数：

`-f` *filenode*

显示文件节点为*filenode*的表的信息

`-i`

在列表中包含索引和序列

`-o` *oid*

显示 OID 为*oid*的表的信息

`-q`

省略标题（适用于脚本）

`-s`

显示表空间 OID

`-S`

包含系统对象（位于`information_schema`、`pg_toast`和`pg_catalog`模式中的对象）

`-t` *tablename_pattern*

显示与*tablename_pattern*匹配的表的信息

`-V`
`--version`

输出oid2name版本并退出。

`-x`

显示每个对象的更多信息：表空间名、模式名和 OID

`-?`
`--help`

显示oid2name命令行参数的帮助并退出。

oid2name还接受以下用于连接参数的命令行参数：

`-d` *`database`*

要连接的数据库

`-H` *`host`*

数据库服务器的主机

`-p` *`port`*

数据库服务器的端口

`-U` *`username`*

用于连接的用户名

`-P` *`password`*

密码（已弃用 – 将其放在命令行上会带来安全风险）

若要显示特定的表，可使用`-o`、`-f`和/或`-t`来选择要显示哪些表。`-o`接受一个 OID，`-f`接受一个 filenode，而`-t`接受一个表名（实际上是一个`LIKE`模式，因此你可以使用诸如`foo%`之类的形式）。这些选项可以按需重复使用，
最终的列表将包含匹配任意一个选项的所有对象。但请注意，这些选项只能显示由`-d`指定的数据库中的对象。

如果没有给出`-o`、`-f`或`-t`，但给出了`-d`，那么它将列出`-d`指定数据库中的所有表。
在这种模式下，`-S`和`-i`选项控制要列出哪些对象。

如果连`-d`也没有给出，它将显示数据库 OID 列表。另外，你也可以给出`-s`来获取表空间列表。

注解

oid2name要求数据库服务器正在运行，且系统目录未损坏。因此，
它对于从数据库灾难性损坏中恢复的帮助非常有限。

示例

```
$ # what's in this database server, anyway?
$ oid2name
All databases:
    Oid  Database Name  Tablespace
----------------------------------
  17228       alvherre  pg_default
  17255     regression  pg_default
  17227      template0  pg_default
      1      template1  pg_default

$ oid2name -s
All tablespaces:
     Oid  Tablespace Name
```

```
-------------------------
    1663       pg_default
    1664        pg_global
  155151         fastdisk
  155152          bigdisk

$ # OK, let's look into database alvherre
$ cd $PGDATA/base/17228

$ # get top 10 db objects in the default tablespace, ordered by
 size
$ ls -lS * | head -10
-rw-------  1 alvherre alvherre 136536064 sep 14 09:51 155173
-rw-------  1 alvherre alvherre  17965056 sep 14 09:51 1155291
-rw-------  1 alvherre alvherre   1204224 sep 14 09:51 16717
-rw-------  1 alvherre alvherre    581632 sep  6 17:51 1255
-rw-------  1 alvherre alvherre    237568 sep 14 09:50 16674
-rw-------  1 alvherre alvherre    212992 sep 14 09:51 1249
-rw-------  1 alvherre alvherre    204800 sep 14 09:51 16684
-rw-------  1 alvherre alvherre    196608 sep 14 09:50 16700
-rw-------  1 alvherre alvherre    163840 sep 14 09:50 16699
-rw-------  1 alvherre alvherre    122880 sep  6 17:51 16751

$ # I wonder what file 155173 is ...
$ oid2name -d alvherre -f 155173
From database "alvherre":
  Filenode  Table Name
----------------------
    155173    accounts

$ # you can ask for more than one object
$ oid2name -d alvherre -f 155173 -f 1155291
From database "alvherre":
  Filenode     Table Name
-------------------------
    155173       accounts
   1155291  accounts_pkey

$ # you can mix the options, and get more details with -x
$ oid2name -d alvherre -t accounts -f 1155291 -x
From database "alvherre":
  Filenode     Table Name      Oid  Schema  Tablespace
-------------------------------------------------------
    155173       accounts   155173  public  pg_default
   1155291  accounts_pkey  1155291  public  pg_default

$ # show disk space for every db object
$ du [0-9]* |
> while read SIZE FILENODE
> do
>   echo "$SIZE       `oid2name -q -d alvherre -i -f $FILENODE`"
> done
16            1155287  branches_pkey
16            1155289  tellers_pkey
17561            1155291  accounts_pkey
...

$ # same, but sort by size
```

```
$ du [0-9]* | sort -rn | while read SIZE FN
> do
>   echo "$SIZE   `oid2name -q -d alvherre -f $FN`"
> done
133466             155173    accounts
17561            1155291  accounts_pkey
1177              16717  pg_proc_proname_args_nsp_index
...

$ # If you want to see what's in tablespaces, use the pg_tblspc
 directory
$ cd $PGDATA/pg_tblspc
$ oid2name -s
All tablespaces:
     Oid  Tablespace Name
-------------------------
    1663       pg_default
    1664        pg_global
  155151         fastdisk
  155152          bigdisk

$ # what databases have objects in tablespace "fastdisk"?
$ ls -d 155151/*
155151/17228/  155151/PG_VERSION

$ # Oh, what was database 17228 again?
$ oid2name
All databases:
    Oid  Database Name  Tablespace
----------------------------------
  17228       alvherre  pg_default
  17255     regression  pg_default
  17227      template0  pg_default
      1      template1  pg_default

$ # Let's see what objects does this database have in the
 tablespace.
$ cd 155151/17228
$ ls -l
total 0
-rw-------  1 postgres postgres 0 sep 13 23:20 155156

$ # OK, this is a pretty small table ... but which one is it?
$ oid2name -d alvherre -f 155156
From database "alvherre":
  Filenode  Table Name
----------------------
    155156         foo
```

作者

B. Palmer <bpalmer@crimelabs.net>

pgbench

pgbench — run a benchmark test on PostgreSQL

大纲

`pgbench -i` [*option*...] [*dbname*]

`pgbench` [*option*...] [*dbname*]

Description

pgbench is a simple program for running benchmark tests on PostgreSQL. It runs the same sequence of SQL commands over and over, possibly in multiple concurrent database sessions, and then calculates the average transaction rate (transactions per second). By default, pgbench tests a scenario that is loosely based on TPC-B, involving five `SELECT`, `UPDATE`, and `INSERT` commands per transaction. However, it is easy to test other cases by writing your own transaction script files.

Typical output from pgbench looks like:

```
transaction type: TPC-B (sort of)
scaling factor: 10
query mode: simple
number of clients: 10
number of threads: 1
number of transactions per client: 1000
number of transactions actually processed: 10000/10000
tps = 85.184871 (including connections establishing)
tps = 85.296346 (excluding connections establishing)
```

The first six lines report some of the most important parameter settings. The next line reports the number of transactions completed and intended (the latter being just the product of number of clients and number of transactions per client); these will be equal unless the run failed before completion. (In `-T` mode, only the actual number of transactions is printed.) The last two lines report the number of transactions per second, figured with and without counting the time to start database sessions.

The default TPC-B-like transaction test requires specific tables to be set up beforehand. pgbench should be invoked with the `-i` (initialize) option to create and populate these tables. (When you are testing a custom script, you don't need this step, but will instead need to do whatever setup your test needs.) Initialization looks like:

```
pgbench -i [ other-options ] dbname
```

where *dbname* is the name of the already-created database to test in. (You may also need `-h`, `-p`, and/or `-U` options to specify how to connect to the database server.)

> **小心**
>
> `pgbench -i` creates four tables `pgbench_accounts`, `pgbench_branches`, `pgbench_history`, and `pgbench_tellers`, destroying any existing tables of these names. Be very careful to use another database if you have tables having these names!

At the default "scale factor" of 1, the tables initially contain this many rows:

```
table                    # of rows
---------------------------------
pgbench_branches         1
pgbench_tellers          10
pgbench_accounts         100000
pgbench_history          0
```

You can (and, for most purposes, probably should) increase the number of rows by using the `-s` (scale factor) option. The `-F` (fillfactor) option might also be used at this point.

Once you have done the necessary setup, you can run your benchmark with a command that doesn't include `-i`, that is

```
pgbench [ options ] dbname
```

In nearly all cases, you'll need some options to make a useful test. The most important options are `-c` (number of clients), `-t` (number of transactions), `-T` (time limit), and `-f` (specify a custom script file). See below for a full list.

Options

The following is divided into three subsections: Different options are used during database initialization and while running benchmarks, some options are useful in both cases.

Initialization Options

pgbench accepts the following command-line initialization arguments:

`-i`

Required to invoke initialization mode.

`-F` *`fillfactor`*

Create the `pgbench_accounts`, `pgbench_tellers` and `pgbench_branches` tables with the given fillfactor. Default is 100.

`-s` *`scale_factor`*

Multiply the number of rows generated by the scale factor. For example, `-s 100` will create 10,000,000 rows in the `pgbench_accounts` table. Default is 1.

`--index-tablespace=`*`index_tablespace`*

Create indexes in the specified tablespace, rather than the default tablespace.

`--tablespace=`*`tablespace`*

Create tables in the specified tablespace, rather than the default tablespace.

`--unlogged-tables`

Create all tables as unlogged tables, rather than permanent tables.

Benchmarking Options

pgbench accepts the following command-line benchmarking arguments:

`-c` *`clients`*

Number of clients simulated, that is, number of concurrent database sessions. Default is 1.

`-C`

Establish a new connection for each transaction, rather than doing it just once per client session. This is useful to measure the connection overhead.

`-d`

Print debugging output.

`-D` *`varname=value`*

Define a variable for use by a custom script (see below). Multiple `-D` options are allowed.

`-f` *`filename`*

Read transaction script from *`filename`*. See below for details. `-N`, `-S`, and `-f` are mutually exclusive.

`-j` *`threads`*

Number of worker threads within pgbench. Using more than one thread can be helpful on multi-CPU machines. The number of clients must be a multiple of the number of threads, since each thread is given the same number of client sessions to manage. Default is 1.

`-l`

Write the time taken by each transaction to a log file. See below for details.

`-M` *`querymode`*

Protocol to use for submitting queries to the server:

- `simple`: use simple query protocol.
- `extended`: use extended query protocol.
- `prepared`: use extended query protocol with prepared statements.

The default is simple query protocol. (See 第 46 章 for more information.)

`-n`

Perform no vacuuming before running the test. This option is necessary if you are running a custom test scenario that does not include the standard tables `pgbench_accounts`, `pgbench_branches`, `pgbench_history`, and `pgbench_tellers`.

`-N`

Do not update `pgbench_tellers` and `pgbench_branches`. This will avoid update contention on these tables, but it makes the test case even less like TPC-B.

`-r`

Report the average per-statement latency (execution time from the perspective of the client) of each command after the benchmark finishes. See below for details.

`-s` *`scale_factor`*

Report the specified scale factor in pgbench's output. With the built-in tests, this is not necessary; the correct scale factor will be detected by counting the number of rows in the `pgbench_branches` table. However, when testing custom benchmarks (`-f` option), the scale factor will be reported as 1 unless this option is used.

`-S`

Perform select-only transactions instead of TPC-B-like test.

`-t` *`transactions`*

Number of transactions each client runs. Default is 10.

`-T` *`seconds`*

Run the test for this many seconds, rather than a fixed number of transactions per client. `-t` and `-T` are mutually exclusive.

`-v`

Vacuum all four standard tables before running the test. With neither `-n` nor `-v`, pgbench will vacuum the `pgbench_tellers` and `pgbench_branches` tables, and will truncate `pgbench_history`.

Common Options

pgbench accepts the following command-line common arguments:

`-h` *`hostname`*

The database server's host name

`-p` *`port`*

The database server's port number

`-U` *`login`*

The user name to connect as

`-V`
`--version`

Print the pgbench version and exit.

`-?`
`--help`

Show help about pgbench command line arguments, and exit.

Notes

What is the “Transaction” Actually Performed in pgbench?

The default transaction script issues seven commands per transaction:

1. `BEGIN;`

2. `UPDATE pgbench_accounts SET abalance = abalance + :delta WHERE aid = :aid;`
3. `SELECT abalance FROM pgbench_accounts WHERE aid = :aid;`
4. `UPDATE pgbench_tellers SET tbalance = tbalance + :delta WHERE tid = :tid;`
5. `UPDATE pgbench_branches SET bbalance = bbalance + :delta WHERE bid = :bid;`
6. `INSERT INTO pgbench_history (tid, bid, aid, delta, mtime) VALUES (:tid, :bid, :aid, :delta, CURRENT_TIMESTAMP);`
7. `END;`

If you specify `-N`, steps 4 and 5 aren't included in the transaction. If you specify `-S`, only the `SELECT` is issued.

Custom Scripts

pgbench has support for running custom benchmark scenarios by replacing the default transaction script (described above) with a transaction script read from a file (`-f` option). In this case a "transaction" counts as one execution of a script file. You can even specify multiple scripts (multiple `-f` options), in which case a random one of the scripts is chosen each time a client session starts a new transaction.

The format of a script file is one SQL command per line; multiline SQL commands are not supported. Empty lines and lines beginning with `--` are ignored. Script file lines can also be "meta commands", which are interpreted by pgbench itself, as described below.

There is a simple variable-substitution facility for script files. Variables can be set by the command-line `-D` option, explained above, or by the meta commands explained below. In addition to any variables preset by `-D` command-line options, the variable `scale` is preset to the current scale factor. Once set, a variable's value can be inserted into a SQL command by writing `:variablename`. When running more than one client session, each session has its own set of variables.

Script file meta commands begin with a backslash (`\`). Arguments to a meta command are separated by white space. These meta commands are supported:

`\set varname operand1 [ operator operand2 ]`

Sets variable *varname* to a calculated integer value. Each *operand* is either an integer constant or a `:variablename` reference to a variable having an integer value. The *operator* can be `+`, `-`, `*`, or `/`.

Example:

```
\set ntellers 10 * :scale
```

`\setrandom varname min max`

Sets variable *varname* to a random integer value between the limits *min* and *max* inclusive. Each limit can be either an integer constant or a `:variablename` reference to a variable having an integer value.

Example:

```
\setrandom aid 1 :naccounts
```

`\sleep number [ us | ms | s ]`

Causes script execution to sleep for the specified duration in microseconds (`us`), milliseconds (`ms`) or seconds (`s`). If the unit is omitted then seconds are the default. *`number`* can be either an integer constant or a `:`*`variablename`* reference to a variable having an integer value.

Example:

```
\sleep 10 ms
```

`\setshell varname command [ argument ... ]`

Sets variable *`varname`* to the result of the shell command *`command`*. The command must return an integer value through its standard output.

`argument` can be either a text constant or a `:`*`variablename`* reference to a variable of any types. If you want to use *`argument`* starting with colons, you need to add an additional colon at the beginning of *`argument`*.

Example:

```
\setshell variable_to_be_assigned command literal_argument :
variable ::literal_starting_with_colon
```

`\shell command [ argument ... ]`

Same as `\setshell`, but the result is ignored.

Example:

```
\shell command literal_argument :variable ::literal_starting_
with_colon
```

As an example, the full definition of the built-in TPC-B-like transaction is:

```
\set nbranches :scale
\set ntellers 10 * :scale
\set naccounts 100000 * :scale
\setrandom aid 1 :naccounts
\setrandom bid 1 :nbranches
\setrandom tid 1 :ntellers
\setrandom delta -5000 5000
BEGIN;
UPDATE pgbench_accounts SET abalance = abalance + :delta WHERE aid
 = :aid;
SELECT abalance FROM pgbench_accounts WHERE aid = :aid;
UPDATE pgbench_tellers SET tbalance = tbalance + :delta WHERE tid =
 :tid;
UPDATE pgbench_branches SET bbalance = bbalance + :delta WHERE bid
 = :bid;
INSERT INTO pgbench_history (tid, bid, aid, delta, mtime) VALUES (:
tid, :bid, :aid, :delta, CURRENT_TIMESTAMP);
END;
```

This script allows each iteration of the transaction to reference different, randomly-chosen rows. (This example also shows why it's important for each client session to have its own variables – otherwise they'd not be independently touching different rows.)

Per-Transaction Logging

With the `-l` option, pgbench writes the time taken by each transaction to a log file. The log file will be named `pgbench_log.nnn`, where *nnn* is the PID of the pgbench process. If the `-j` option is 2 or higher, creating multiple worker threads, each will have its own log file. The first worker will use the same name for its log file as in the standard single worker case. The additional log files for the other workers will be named `pgbench_log.nnn.mmm`, where *mmm* is a sequential number for each worker starting with 1.

The format of the log is:

client_id transaction_no time file_no time_epoch time_us

where *time* is the total elapsed transaction time in microseconds, *file_no* identifies which script file was used (useful when multiple scripts were specified with `-f`), and *time_epoch*/*time_us* are a UNIX epoch format timestamp and an offset in microseconds (suitable for creating an ISO 8601 timestamp with fractional seconds) showing when the transaction completed.

Here are example outputs:

```
 0 199 2241 0 1175850568 995598
 0 200 2465 0 1175850568 998079
 0 201 2513 0 1175850569 608
 0 202 2038 0 1175850569 2663
```

Per-Statement Latencies

With the `-r` option, pgbench collects the elapsed transaction time of each statement executed by every client. It then reports an average of those values, referred to as the latency for each statement, after the benchmark has finished.

For the default script, the output will look similar to this:

```
starting vacuum...end.
transaction type: TPC-B (sort of)
scaling factor: 1
query mode: simple
number of clients: 10
number of threads: 1
number of transactions per client: 1000
number of transactions actually processed: 10000/10000
tps = 618.764555 (including connections establishing)
tps = 622.977698 (excluding connections establishing)
statement latencies in milliseconds:
        0.004386        \set nbranches 1 * :scale
        0.001343        \set ntellers 10 * :scale
        0.001212        \set naccounts 100000 * :scale
        0.001310        \setrandom aid 1 :naccounts
        0.001073        \setrandom bid 1 :nbranches
        0.001005        \setrandom tid 1 :ntellers
        0.001078        \setrandom delta -5000 5000
        0.326152        BEGIN;
        0.603376        UPDATE pgbench_accounts SET abalance =
 abalance + :delta WHERE aid = :aid;
        0.454643        SELECT abalance FROM pgbench_accounts WHERE
 aid = :aid;
        5.528491        UPDATE pgbench_tellers SET tbalance =
 tbalance + :delta WHERE tid = :tid;
```

```
        7.335435        UPDATE pgbench_branches SET bbalance =
 bbalance + :delta WHERE bid = :bid;
        0.371851        INSERT INTO pgbench_history (tid, bid, aid,
 delta, mtime) VALUES (:tid, :bid, :aid, :delta, CURRENT_TIMESTAMP)
;
        1.212976        END;
```

If multiple script files are specified, the averages are reported separately for each script file.

Note that collecting the additional timing information needed for per-statement latency computation adds some overhead. This will slow average execution speed and lower the computed TPS. The amount of slowdown varies significantly depending on platform and hardware. Comparing average TPS values with and without latency reporting enabled is a good way to measure if the timing overhead is significant.

Good Practices

It is very easy to use pgbench to produce completely meaningless numbers. Here are some guidelines to help you get useful results.

In the first place, never believe any test that runs for only a few seconds. Use the `-t` or `-T` option to make the run last at least a few minutes, so as to average out noise. In some cases you could need hours to get numbers that are reproducible. It's a good idea to try the test run a few times, to find out if your numbers are reproducible or not.

For the default TPC-B-like test scenario, the initialization scale factor (`-s`) should be at least as large as the largest number of clients you intend to test (`-c`); else you'll mostly be measuring update contention. There are only `-s` rows in the `pgbench_branches` table, and every transaction wants to update one of them, so `-c` values in excess of `-s` will undoubtedly result in lots of transactions blocked waiting for other transactions.

The default test scenario is also quite sensitive to how long it's been since the tables were initialized: accumulation of dead rows and dead space in the tables changes the results. To understand the results you must keep track of the total number of updates and when vacuuming happens. If autovacuum is enabled it can result in unpredictable changes in measured performance.

A limitation of pgbench is that it can itself become the bottleneck when trying to test a large number of client sessions. This can be alleviated by running pgbench on a different machine from the database server, although low network latency will be essential. It might even be useful to run several pgbench instances concurrently, on several client machines, against the same database server.

vacuumlo

vacuumlo — 从 PostgreSQL 数据库中移除孤立的大对象

大纲

`vacuumlo` [*option*...] *dbname*...

描述

vacuumlo是一个简单的实用程序，用于从 PostgreSQL数据库中移除“孤立的”大对象。孤立的大对象（LO）是指其 OID 没有出现在该数据库任何 `oid`或`lo`数据列中的 LO。

如果你使用这个程序，也可能会对lo模块中的 `lo_manage`触发器感兴趣。`lo_manage`有助于尽量避免一开始就创建出孤立 LO。

命令行中给出的所有数据库都会被处理。

选项

vacuumlo接受以下命令行参数：

`-l` *limit*

每个事务最多移除*limit*个大对象（默认值为 1000）。由于服务器会为每个被移除的 LO 获取一个锁，在单个事务中移除过多 LO 有可能超出max_locks_per_transaction。如果你希望在单个事务中完成全部移除，请把该限制设为 0。

`-n`

不实际移除任何内容，只显示将要执行的操作。

`-v`

输出大量进度消息。

`-V`
`--version`

打印vacuumlo版本并退出。

`-?`
`--help`

显示vacuumlo命令行参数的帮助并退出。

vacuumlo还接受以下用于连接参数的命令行参数：

`-h` *hostname*

数据库服务器主机。

`-p` *port*

数据库服务器端口。

`-U` *username*

用于连接的用户名。

`-w`
`--no-password`

绝不发出密码提示。如果服务器要求密码认证，而又无法通过其他方式（例如`.pgpass`文件）获得密码，则连接尝试将失败。这个选项在批处理作业和脚本中很有用，因为这些场景下没有用户可以输入密码。

`-W`

强制vacuumlo在连接数据库之前提示输入密码。

这个选项并非必不可少，因为如果服务器要求密码认证，vacuumlo会自动提示输入密码。不过，vacuumlo会浪费一次连接尝试，才知道服务器需要密码。在某些情况下，为了避免这次额外的连接尝试，提前指定`-W`是值得的。

注解

vacuumlo按以下方法工作：首先，vacuumlo构建一个临时表，其中包含所选数据库中所有大对象的 OID。然后，它会扫描数据库中所有类型为 `oid`或`lo`的列，并从临时表中移除匹配的项。（注意：只会考虑名称正好为这两种的类型；特别是，基于这些类型定义的域不会被考虑。）临时表中剩余的项标识出孤立 LO，并会被移除。

作者

Peter Mount `<peter@retep.org.uk>`

G.2. 服务器应用

本节介绍`contrib`中与PostgreSQL服务器相关的应用程序。它们通常运行在数据库服务器所在的主机上。另见PostgreSQL Server Applications，其中介绍了作为核心PostgreSQL发行版一部分的服务器应用程序。

pg_archivecleanup

pg_archivecleanup — 清理PostgreSQL WAL 归档文件

大纲

`pg_archivecleanup` [*option*...] *archivelocation oldestkeptwalfile*

描述

pg_archivecleanup设计为在服务器作为备库运行时用作 `archive_cleanup_command`，以清理 WAL 文件归档（见第 25.2 节）。pg_archivecleanup也可作为独立程序使用，以清理 WAL 文件归档。

要配置备库使用 pg_archivecleanup，请将以下内容放入其 `recovery.conf` 配置文件中：

```
archive_cleanup_command = 'pg_archivecleanup archivelocation %r'
```

其中，*archivelocation* 是应从中移除 WAL 段文件的目录。

当在archive_cleanup_command中使用时，逻辑上早于`%r`参数值的所有 WAL 文件都会从*archivelocation*中移除。这样既能将需要保留的文件数量降到最低，又能保留崩溃后重启能力。如果*archivelocation*只是该特定备库的临时暂存区，那么使用此参数是合适的；但以下情况下则不适合使用：*archivelocation*打算作为长期 WAL 归档区域，或者多个备库正在从同一归档位置恢复。

作为独立程序使用时，逻辑上早于 *oldestkeptwalfile* 的所有 WAL 文件都会从 *archivelocation* 中移除。在此模式下，如果你指定的是 `.backup`文件名，则只会使用文件名前缀作为 *oldestkeptwalfile*。这允许你无误地删除某个特定基础备份之前归档的所有 WAL 文件。例如，下例会移除所有早于 WAL 文件名 `000000010000003700000010` 的文件：

```
pg_archivecleanup -d archive 000000010000003700000010.00000020.
backup

pg_archivecleanup:  keep WAL file "archive/
000000010000003700000010" and later
pg_archivecleanup:  removing file "archive/
00000001000000370000000F"
pg_archivecleanup:  removing file "archive/
00000001000000370000000E"
```

pg_archivecleanup 假定 *archivelocation* 是服务器属主用户可读写的目录。

选项

pg_archivecleanup接受以下命令行参数：

`-d`

在 `stderr` 上输出大量调试日志。

`-n`

在 `stdout` 上打印原本会被移除的文件名（执行试运行）。

`-V`
`--version`

输出 pg_archivecleanup 的版本号并退出。

`-x` *`extension`*

在将该程序作为独立实用程序使用时，指定一个扩展名；在判断文件是否应删除之前，会先从所有文件名中去掉该扩展名。这通常适用于清理由于存储时经过压缩、因而被压缩程序附加了扩展名的归档。例如：`-x .gz`。

`-?`
`--help`

显示有关 pg_archivecleanup 命令行参数的帮助并退出。

注解

pg_archivecleanup 设计为在作为独立工具使用时与 PostgreSQL 8.0 及更高版本配合工作，或在作为归档清理命令使用时与 PostgreSQL 9.0 及更高版本配合工作。

pg_archivecleanup 采用 C 编写，源代码易于修改，并且专门划出了若干区域供你按自身需要修改。

示例

在 Linux 或 Unix 系统上，你可以这样使用：

```
archive_cleanup_command = 'pg_archivecleanup -d /mnt/standby/
archive %r 2>>cleanup.log'
```

其中归档目录实际上位于备库上，因此 `archive_command` 通过 NFS 访问它，但这些文件对备库来说是本地的。这将会：

- 在 `cleanup.log` 中产生调试输出
- 从归档目录中移除不再需要的文件

作者

Simon Riggs `<simon@2ndquadrant.com>`

另见

pg_standby

pg_standby

pg_standby — 支持创建 PostgreSQL 温备服务器

大纲

pg_standby [*option*...] *archivelocation* *nextwalfile* *xlogfilepath* [*restartwalfile*]

描述

pg_standby 支持创建“温备”数据库服务器。它既是可用于生产环境的程序，也是可定制的模板，便于在有需要时进行特定修改。

pg_standby 设计为一个会等待的 `restore_command`，将标准归档恢复转变为温备运行需要这样的命令。此外还需要其他配置，服务器的主手册中对此均有介绍（参见第 25.2 节）。

要配置备库使用 pg_standby，请将以下内容放入其 `recovery.conf` 配置文件：

```
restore_command = 'pg_standby archiveDir %f %p %r'
```

其中，*archiveDir* 是恢复 WAL 段文件时所读取的目录。

如果指定了 *restartwalfile*（通常使用 `%r` 宏），就会从 *archivelocation* 中删除逻辑上位于该文件之前的所有 WAL 文件。这样既能尽量减少需要保留的文件数量，又能保留崩溃后重启的能力。如果 *archivelocation* 是专供此备库使用的临时暂存区，就适合使用此参数；但如果 *archivelocation* 用作长期 WAL 归档区，就不应使用。

pg_standby 假定 *archivelocation* 是服务器所属用户可读的目录。如果指定了 *restartwalfile*（或 `-k`），*archivelocation* 目录还必须可写。

主库发生故障时，切换到“温备”数据库服务器有以下两种方式：

智能故障切换

在智能故障切换中，服务器会先应用归档中所有可用的 WAL 文件，再正式启动。即使备库已经落后，这也能做到零数据丢失；但如果尚未应用的 WAL 很多，备库可能需要很长时间才能就绪。要触发智能故障切换，请创建包含单词 `smart` 的触发文件，或者直接创建一个空的触发文件。

快速故障切换

在快速故障切换中，服务器会立即正式启动。归档中所有尚未应用的 WAL 文件都会被忽略，这些文件中的全部事务都会丢失。要触发快速故障切换，请创建触发文件，并在其中写入单词 `fast`。也可以配置 pg_standby，使其在指定时间间隔内没有出现新 WAL 文件时，自动执行快速故障切换。

选项

pg_standby 接受以下命令行参数：

`-c`

使用 `cp` 或 `copy` 命令从归档恢复 WAL 文件。由于这是唯一受支持的行为，因此该选项没有实际作用。

`-d`

在 `stderr` 上输出大量调试日志。

`-k`

从 *archivelocation* 中删除文件，使归档中保留的、位于当前文件之前的 WAL 文件不超过此数量。零（默认值）表示不从 *archivelocation* 中删除任何文件。如果指定了 *restartwalfile*，就会静默忽略此参数，因为前者能更准确地确定归档的正确截断点。从 PostgreSQL 8.3 起，已弃用此参数；指定 *restartwalfile* 参数更安全、更高效。设置过小可能删除备库重启仍需使用的文件，设置过大则会浪费归档空间。

`-r` *maxretries*

设置复制命令失败后的最大重试次数（默认为 3）。每次失败后，都会等待 *sleeptime* * *num_retries*，因此等待时间会逐次增加。默认情况下，会依次等待 5 秒、10 秒、15 秒，然后才向备库报告失败。这会被解释为恢复结束，从而使备库完全启动。

`-s` *sleeptime*

设置两次检查之间的等待秒数（最多 60 秒，默认 5 秒），检查的内容是待恢复的 WAL 文件是否已在归档中可用。默认设置不一定就是推荐设置；相关讨论参见第 25.2 节。

`-t` *triggerfile*

指定触发文件；一旦该文件出现，就会引发故障切换。建议使用带有明确结构的文件名，例如 `/tmp/pgsql.trigger.5432`，以免同一系统上存在多个服务器时，混淆被触发的是哪个服务器。

`-V`
`--version`

打印 pg_standby 的版本并退出。

`-w` *maxwaittime*

设置等待下一个 WAL 文件的最长秒数，超过此时间后执行快速故障切换。零（默认值）表示永远等待。默认设置不一定就是推荐设置；相关讨论参见第 25.2 节。

`-?`
`--help`

显示有关 pg_standby 命令行参数的帮助信息并退出。

说明

pg_standby 设计为与 PostgreSQL 8.2 及更高版本配合使用。

PostgreSQL 8.3 提供了 `%r` 宏，用于让 pg_standby 知道需要保留的最后一个文件。使用 PostgreSQL 8.2 时，如果需要清理归档，就必须使用 `-k` 选项。该选项在 8.3 中仍然可用，但已弃用。

PostgreSQL 8.4 提供了 `recovery_end_command` 选项。如果没有此选项，遗留的触发文件可能带来危险。

pg_standby 用 C 编写，源码易于修改，并专门标出了可根据自身需要进行修改的部分。

示例

在 Linux 或 Unix 系统上，可以使用：

```
archive_command = 'cp %p .../archive/%f'

restore_command = 'pg_standby -d -s 2 -t /tmp/pgsql.trigger.5442 ..
./archive %f %p %r 2>>standby.log'
```

```
recovery_end_command = 'rm -f /tmp/pgsql.trigger.5442'
```

这里，归档目录实际位于备库上，因此 `archive_command` 通过 NFS 访问该目录，但这些文件对备库而言是本地文件（因此可以使用 `ln`）。此配置会：

- 将调试输出写入 `standby.log`
- 每隔 2 秒检查一次下一个 WAL 文件是否可用
- 仅在名为 `/tmp/pgsql.trigger.5442` 的触发文件出现时停止等待，并根据其内容执行故障切换
- 恢复结束时删除触发文件
- 从归档目录中移除不再需要的文件

在 Windows 上，可以使用：

```
archive_command = 'copy %p ...\\archive\\%f'

restore_command = 'pg_standby -d -s 5 -t C:\pgsql.trigger.5442 ...
\archive %f %p %r 2>>standby.log'

recovery_end_command = 'del C:\pgsql.trigger.5442'
```

注意，需要双写反斜杠的设置是 `archive_command`，而无需双写的设置是 `restore_command` 或 `recovery_end_command`。此配置会：

- 使用 `copy` 命令从归档恢复 WAL 文件
- 将调试输出写入 `standby.log`
- 每隔 5 秒检查一次下一个 WAL 文件是否可用
- 仅在名为 `C:\pgsql.trigger.5442` 的触发文件出现时停止等待，并根据其内容执行故障切换
- 恢复结束时删除触发文件
- 从归档目录中移除不再需要的文件

Windows 上的 `copy` 命令会在文件复制完成之前设置最终文件大小，这通常会让 pg_standby 产生误判。因此，pg_standby 一旦看到正确的文件大小，还会等待 `sleeptime` 秒。GNUWin32 的 `cp` 只有在文件复制完成后才设置文件大小。

由于 Windows 示例在两端都使用 `copy`，任一服务器或两个服务器都可能通过网络访问归档目录。

作者

Simon Riggs <simon@2ndquadrant.com>

参见

pg_archivecleanup

pg_test_fsync

pg_test_fsync — 为PostgreSQL确定最快的wal_sync_method

大纲

`pg_test_fsync` [*option*...]

描述

pg_test_fsync旨在让你较为合理地了解，在你的具体系统上哪一种wal_sync_method最快，并且在识别出 I/O 问题时提供诊断信息。不过，pg_test_fsync显示出的差异未必会对实际数据库吞吐量产生任何影响，
特别是因为许多数据库服务器的速度瓶颈并不在其预写式日志上。

选项

pg_test_fsync 接受以下命令行选项：

`-f`
`--filename`

指定用于写入测试数据的文件名。该文件应位于 `pg_xlog`目录所在或将要放置的同一文件系统中。（`pg_xlog`包含WAL文件。）默认值是当前目录中的`pg_test_fsync.out`。

`-s`
`--secs-per-test`

指定每项测试的秒数。每项测试用时越长，测试结果就越精确，
但运行完成所需的时间也越长。默认值是 2 秒，这使程序能够在约 30 秒内完成。

`-V`
`--version`

打印pg_test_fsync版本并退出。

`-?`
`--help`

显示有关pg_test_fsync命令行参数的帮助信息并退出。

作者

Bruce Momjian `<bruce@momjian.us>`

另见

postgres

pg_test_timing

pg_test_timing — 测量计时开销

大纲

`pg_test_timing` [*option*...]

描述

pg_test_timing是一个用于测量系统计时开销的工具，并确认系统时间不会倒退。
收集计时数据较慢的系统，可能会给出不够精确的 `EXPLAIN ANALYZE`结果。

选项

pg_test_timing 接受以下命令行选项：

`-d` *duration*
`--duration=`*duration*

指定测试持续时间，以秒为单位。持续时间越长，精度会略有提高，
也更可能发现系统时钟倒退的问题。默认测试持续时间为 3 秒。

`-V`
`--version`

打印pg_test_timing版本并退出。

`-?`
`--help`

显示有关pg_test_timing命令行参数的帮助信息并退出。

用法

结果解读

良好的结果会显示，大多数（>90%）单次计时调用耗时不到 1 微秒。每轮循环的平均开销还会更低，在 100 纳秒以下。下面这个示例来自使用 TSC 时钟源的 Intel i7-860 系统，表现非常出色：

```
Testing timing overhead for 3 seconds.
Per loop time including overhead: 35.96 nsec
Histogram of timing durations:
   < usec:      count   percent
       16:          2  0.00000%
        8:         13  0.00002%
        4:        126  0.00015%
        2:    2999652  3.59518%
        1:   80435604 96.40465%
```

注意，每轮循环时间与直方图使用的单位不同。循环的解析度可以达到几纳秒（nsec），
而单次计时调用的解析度只能达到 1 微秒（usec）。

测量执行器计时开销

当查询执行器使用`EXPLAIN ANALYZE`运行语句时，除了显示汇总信息之外，还会对各个操作分别计时。可以使用psql程序统计行数，以检查你的系统上的这类开销：

```
CREATE TABLE t AS SELECT * FROM generate_series(1,100000);
\timing
SELECT COUNT(*) FROM t;
EXPLAIN ANALYZE SELECT COUNT(*) FROM t;
```

在所测的 i7-860 系统上，普通计数查询耗时 9.8 ms，而 `EXPLAIN ANALYZE`版本耗时 16.6 ms，两者都处理了略多于 100,000 行。这 6.8 ms 的差异意味着每行的计时开销为 68 ns，大约是 pg_test_timing 估计值的两倍。即便是这样相对较小的开销，也会让完整计时的计数语句耗时增加近 70%。对于更复杂的查询，计时开销带来的问题就不会这么明显。

更改时钟源

在一些较新的 Linux 系统上，可以随时更改用于收集计时数据的时钟源。第二个示例在上述表现很快的同一系统上，展示了切换到较慢的 acpi_pm 时钟源后可能出现的减速：

```
# cat /sys/devices/system/clocksource/clocksource0/available_
clocksource
tsc hpet acpi_pm
# echo acpi_pm > /sys/devices/system/clocksource/clocksource0/
current_clocksource
# pg_test_timing
Per loop time including overhead: 722.92 nsec
Histogram of timing durations:
   < usec:      count   percent
       16:          3  0.00007%
        8:        563  0.01357%
        4:       3241  0.07810%
        2:    2990371 72.05956%
        1:    1155682 27.84870%
```

在这种配置下，上面的示例`EXPLAIN ANALYZE`需要 115.9 ms。这意味着计时开销达到 1061 nsec，同样只是本工具直接测得数值的一个小倍数。如此之大的计时开销说明，实际查询本身只占已统计时间的一小部分，其余大多耗费在开销上。在这种配置下，
任何包含大量计时操作的`EXPLAIN ANALYZE`总计值，都会因计时开销而被显著抬高。

FreeBSD 也允许动态更改时钟源，并会记录启动期间所选计时器的信息：

```
# dmesg | grep "Timecounter"
Timecounter "ACPI-fast" frequency 3579545 Hz quality 900
Timecounter "i8254" frequency 1193182 Hz quality 0
Timecounters tick every 10.000 msec
Timecounter "TSC" frequency 2531787134 Hz quality 800
# sysctl kern.timecounter.hardware=TSC
kern.timecounter.hardware: ACPI-fast -> TSC
```

其他系统可能只允许在启动时设置时钟源。在较旧的 Linux 系统上，“clock” 内核设置是进行此类更改的唯一方式。甚至在某些较新的系统上，能看到的时钟源选项也只有 “jiffies” 。Jiffies 是较旧的 Linux 软件时钟实现；如果底层计时硬件足够快，它也能提供良好的解析度，如下例所示：

```
$ cat /sys/devices/system/clocksource/clocksource0/available_
clocksource
jiffies
$ dmesg | grep time.c
time.c: Using 3.579545 MHz WALL PM GTOD PIT/TSC timer.
time.c: Detected 2400.153 MHz processor.
$ pg_test_timing
Testing timing overhead for 3 seconds.
Per timing duration including loop overhead: 97.75 ns
```

```
Histogram of timing durations:
   < usec:      count   percent
       32:          1  0.00000%
       16:          1  0.00000%
        8:         22  0.00007%
        4:       3010  0.00981%
        2:    2993204  9.75277%
        1:   27694571 90.23734%
```

时钟硬件和计时准确性

在计算机上收集精确的计时信息，通常依赖精度各异的硬件时钟。对于某些硬件，操作系统几乎可以将系统时钟时间直接传递给程序。系统时钟也可能来自一种只提供计时中断、按已知时间间隔周期性地产生滴答信号的芯片。无论哪种情况，操作系统内核都会提供一个屏蔽这些细节的时钟源。不过，这种时钟源的精度以及返回结果的速度，都取决于底层硬件。

不准确的计时可能导致系统不稳定。对时钟源的任何更改都应十分谨慎地测试。操作系统的默认设置有时会偏向可靠性，而不是追求最佳精度。如果你使用的是虚拟机，还应了解与其兼容的推荐时钟源。虚拟硬件在模拟计时器时会面临额外困难，厂商通常还会针对不同操作系统给出特定的设置建议。

时间戳计数器（TSC）时钟源是当前一代 CPU 上可用的最精确时钟源。当操作系统支持它且 TSC 时钟本身可靠时，它是跟踪系统时间的首选方式。TSC 也可能因多种原因无法提供精确的计时源，从而变得不可靠。较旧的系统中，TSC 时钟可能会随 CPU 温度变化，因此不适合用于计时。在某些较旧的多核 CPU 上使用 TSC，还可能导致不同核心报告的时间彼此不一致。这会引起时间倒退，而本程序正是用来检查这类问题的。即便在最新的系统上，若节能配置过于激进，也可能无法提供精确的 TSC 计时。

较新的操作系统可能会检测已知的 TSC 问题，并在发现时切换到更慢但更稳定的时钟源。如果你的系统支持 TSC 计时却没有默认使用它，那么禁用它很可能是有充分理由的。某些操作系统也可能无法正确检测所有潜在问题，或者即使已知 TSC 不准确，仍允许继续使用它。

如果系统上有高精度事件计时器（HPET），并且 TSC 不准确，那么 HPET 就是系统更倾向使用的计时器。计时器芯片本身可编程，最高可提供 100 纳秒的解析度，但在你的系统时钟中未必能看到这么高的准确性。

高级配置与电源接口（ACPI）提供了一种电源管理（PM）计时器，Linux 将其称为 acpi_pm。由 acpi_pm 派生的时钟在最佳情况下可提供 300 纳秒的解析度。

较旧 PC 硬件使用的计时器包括 8254 可编程间隔计时器（PIT）、实时时钟（RTC）、高级可编程中断控制器（APIC）计时器以及 Cyclone 计时器。这些计时器的目标解析度是毫秒级。

作者

Ants Aasma <ants.aasma@eesti.ee>

参见

EXPLAIN

pg_upgrade

pg_upgrade — upgrade a PostgreSQL server instance

大纲

`pg_upgrade -b oldbindir -B newbindir -d olddatadir -D newdatadir [option...]`

Description

pg_upgrade (formerly called pg_migrator) allows data stored in PostgreSQL data files to be upgraded to a later PostgreSQL major version without the data dump/reload typically required for major version upgrades, e.g. from 8.4.7 to the current major release of PostgreSQL. It is not required for minor version upgrades, e.g. from 9.0.1 to 9.0.4.

Major PostgreSQL releases regularly add new features that often change the layout of the system tables, but the internal data storage format rarely changes. pg_upgrade uses this fact to perform rapid upgrades by creating new system tables and simply reusing the old user data files. If a future major release ever changes the data storage format in a way that makes the old data format unreadable, pg_upgrade will not be usable for such upgrades. (The community will attempt to avoid such situations.)

pg_upgrade does its best to make sure the old and new clusters are binary-compatible, e.g. by checking for compatible compile-time settings, including 32/64-bit binaries. It is important that any external modules are also binary compatible, though this cannot be checked by pg_upgrade.

pg_upgrade supports upgrades from 8.3.X and later to the current major release of PostgreSQL, including snapshot and alpha releases.

Options

pg_upgrade accepts the following command-line arguments:

`-b old_bindir`
`--old-bindir=old_bindir`

the old cluster executable directory; environment variable `PGBINOLD`

`-B new_bindir`
`--new-bindir=new_bindir`

the new cluster executable directory; environment variable `PGBINNEW`

`-c`
`--check`

check clusters only, don't change any data

`-d old_datadir`
`--old-datadir=old_datadir`

the old cluster data directory; environment variable `PGDATAOLD`

`-D new_datadir`
`--new-datadir=new_datadir`

the new cluster data directory; environment variable `PGDATANEW`

`-k`
`--link`

use hard links instead of copying files to the new cluster

`-o` *`options`*
`--old-options` *`options`*

options to be passed directly to the old `postgres` command

`-O` *`options`*
`--new-options` *`options`*

options to be passed directly to the new `postgres` command

`-p` *`old_port_number`*
`--old-port=`*`old_portnum`*

the old cluster port number; environment variable `PGPORTOLD`

`-P` *`new_port_number`*
`--new-port=`*`new_portnum`*

the new cluster port number; environment variable `PGPORTNEW`

`-r`
`--retain`

retain SQL and log files even after successful completion

`-u` *`user_name`*
`--user=`*`user_name`*

cluster's super user name; environment variable `PGUSER`

`-v`
`--verbose`

enable verbose internal logging

`-V`
`--version`

display version information, then exit

`-?`
`-h`
`--help`

show help, then exit

Usage

These are the steps to perform an upgrade with pg_upgrade:

1. **Optionally move the old cluster**

   If you are using a version-specific installation directory, e.g. `/opt/PostgreSQL/9.1`, you do not need to move the old cluster. The graphical installers all use version-specific installation directories.

   If your installation directory is not version-specific, e.g. `/usr/local/pgsql`, it is necessary to move the current PostgreSQL install directory so it does not interfere with the new PostgreSQL installation. Once the current PostgreSQL server is shut

down, it is safe to rename the PostgreSQL installation directory; assuming the old directory is `/usr/local/pgsql`, you can do:

```
mv /usr/local/pgsql /usr/local/pgsql.old
```

to rename the directory.

2. **For source installs, build the new version**

   Build the new PostgreSQL source with `configure` flags that are compatible with the old cluster. pg_upgrade will check `pg_controldata` to make sure all settings are compatible before starting the upgrade.

3. **Install the new PostgreSQL binaries**

   Install the new server's binaries and support files.

   For source installs, if you wish to install the new server in a custom location, use the `prefix` variable:

   ```
   gmake prefix=/usr/local/pgsql.new install
   ```

4. **Install pg_upgrade and pg_upgrade_support**

   Install the pg_upgrade binary and pg_upgrade_support library in the new PostgreSQL cluster.

5. **Initialize the new PostgreSQL cluster**

   Initialize the new cluster using `initdb`. Again, use compatible `initdb` flags that match the old cluster. Many prebuilt installers do this step automatically. There is no need to start the new cluster.

6. **Install custom shared object files**

   Install any custom shared object files (or DLLs) used by the old cluster into the new cluster, e.g. `pgcrypto.so`, whether they are from `contrib` or some other source. Do not install the schema definitions, e.g. `pgcrypto.sql`, because these will be upgraded from the old cluster.

7. **Adjust authentication**

   `pg_upgrade` will connect to the old and new servers several times, so you might want to set authentication to `peer` in `pg_hba.conf` or use a `~/.pgpass` file (see 第 31.15 节).

8. **Stop both servers**

   Make sure both database servers are stopped using, on Unix, e.g.:

   ```
   pg_ctl -D /opt/PostgreSQL/8.4 stop
   pg_ctl -D /opt/PostgreSQL/9.0 stop
   ```

   or on Windows, using the proper service names:

   ```
   NET STOP postgresql-8.4
   NET STOP postgresql-9.0
   ```

   or

   ```
   NET STOP pgsql-8.3  (PostgreSQL 8.3 and older used a different
    service name)
   ```

9. **Run pg_upgrade**

   Always run the pg_upgrade binary of the new server, not the old one. pg_upgrade requires the specification of the old and new cluster's data and executable (`bin`) directories. You can also specify user and port values, and whether you want the data linked instead of copied (the default).

   If you use link mode, the upgrade will be much faster (no file copying), but you will not be able to access your old cluster once you start the new cluster after the upgrade. Link mode also requires that the old and new cluster data directories be in the same file system. See `pg_upgrade --help` for a full list of options.

   For Windows users, you must be logged into an administrative account, and then start a shell as the `postgres` user and set the proper path:

   ```
   RUNAS /USER:postgres "CMD.EXE"
   SET PATH=%PATH%;C:\Program Files\PostgreSQL\9.0\bin;
   ```

   and then run pg_upgrade with quoted directories, e.g.:

   ```
   pg_upgrade.exe
           --old-datadir "C:/Program Files/PostgreSQL/8.4/data"
           --new-datadir "C:/Program Files/PostgreSQL/9.0/data"
           --old-bindir "C:/Program Files/PostgreSQL/8.4/bin"
           --new-bindir "C:/Program Files/PostgreSQL/9.0/bin"
   ```

   Once started, `pg_upgrade` will verify the two clusters are compatible and then do the upgrade. You can use `pg_upgrade --check` to perform only the checks, even if the old server is still running. `pg_upgrade --check` will also outline any manual adjustments you will need to make after the upgrade. If you are going to be using link mode, you should use the `--link` option with `--check` to enable link-mode-specific checks. `pg_upgrade` requires write permission in the current directory.

   Obviously, no one should be accessing the clusters during the upgrade. pg_upgrade defaults to running servers on port 50432 to avoid unintended client connections. You can use the same port number for both clusters when doing an upgrade because the old and new clusters will not be running at the same time. However, when checking an old running server, the old and new port numbers must be different.

   If an error occurs while restoring the database schema, `pg_upgrade` will exit and you will have to revert to the old cluster as outlined in 步骤 14 below. To try `pg_upgrade` again, you will need to modify the old cluster so the pg_upgrade schema restore succeeds. If the problem is a contrib module, you might need to uninstall the contrib module from the old cluster and install it in the new cluster after the upgrade, assuming the module is not being used to store user data.

10. **Restore `pg_hba.conf`**

    If you modified `pg_hba.conf`, restore its original settings. It might also be necessary to adjust other configuration files in the new cluster to match the old cluster, e.g. `postgresql.conf`.

11. **Post-Upgrade processing**

    If any post-upgrade processing is required, pg_upgrade will issue warnings as it completes. It will also generate script files that must be run by the administrator. The script files will connect to each database that needs post-upgrade processing. Each script should be run using:

```
psql --username postgres --file script.sql postgres
```

The scripts can be run in any order and can be deleted once they have been run.

> **小心**
>
> In general it is unsafe to access tables referenced in rebuild scripts until the rebuild scripts have run to completion; doing so could yield incorrect results or poor performance. Tables not referenced in rebuild scripts can be accessed immediately.

12. **Statistics**

    Because optimizer statistics are not transferred by `pg_upgrade`, you will be instructed to run a command to regenerate that information at the end of the upgrade.

13. **Delete old cluster**

    Once you are satisfied with the upgrade, you can delete the old cluster's data directories by running the script mentioned when `pg_upgrade` completes. You can also delete the old installation directories (e.g. `bin`, `share`).

14. **Reverting to old cluster**

    If, after running `pg_upgrade`, you wish to revert to the old cluster, there are several options:

    - If you ran `pg_upgrade` with `--check`, no modifications were made to the old cluster and you can re-use it anytime.
    - If you ran `pg_upgrade` with `--link`, the data files are shared between the old and new cluster. If you started the new cluster, the new server has written to those shared files and it is unsafe to use the old cluster.
    - If you ran `pg_upgrade` without `--link` or did not start the new server, the old cluster was not modified except that, if linking started, a `.old` suffix was appended to `$PGDATA/global/pg_control`. To reuse the old cluster, possibly remove the `.old` suffix from `$PGDATA/global/pg_control`; you can then restart the old cluster.

Notes

pg_upgrade does not support upgrading of databases containing these `reg*` OID-referencing system data types: `regproc`, `regprocedure`, `regoper`, `regoperator`, `regconfig`, and `regdictionary`. (`regtype` can be upgraded.)

All failure, rebuild, and reindex cases will be reported by pg_upgrade if they affect your installation; post-upgrade scripts to rebuild tables and indexes will be generated automatically.

For deployment testing, create a schema-only copy of the old cluster, insert dummy data, and upgrade that.

If you are upgrading a pre-PostgreSQL 9.2 cluster that uses a configuration-file-only directory, you must pass the real data directory location to pg_upgrade, and pass the

configuration directory location to the server, e.g. `-d /real-data-directory -o '-D /configuration-directory'`.

If using a pre-9.1 old server that is using a non-default Unix-domain socket directory or a default that differs from the default of the new cluster, set `PGHOST` to point to the old server's socket location. (This is not relevant on Windows.)

A Log-Shipping Standby Server (第 25.2 节) cannot be upgraded because the server must allow writes. The simplest way is to upgrade the primary and use `rsync` to rebuild the standbys. You can run `rsync` while the primary is down, or as part of a base backup (第 24.3.2 节) which overwrites the old standby cluster.

If you want to use link mode and you do not want your old cluster to be modified when the new cluster is started, make a copy of the old cluster and upgrade that in link mode. To make a valid copy of the old cluster, use `rsync` to create a dirty copy of the old cluster while the server is running, then shut down the old server and run `rsync` again to update the copy with any changes to make it consistent. You might want to exclude some files, e.g. `postmaster.pid`, as documented in 第 24.3.3 节.

Limitations in Upgrading from PostgreSQL 8.3

Upgrading from PostgreSQL 8.3 has additional restrictions not present when upgrading from later PostgreSQL releases. For example, pg_upgrade will not work for upgrading from 8.3 if a user column is defined as:

- a `tsquery` data type
- data type `name` and is not the first column

You must drop any such columns and upgrade them manually.

pg_upgrade will not work if the `ltree` contrib module is installed in a database.

pg_upgrade will require a table rebuild if:

- a user column is of data type `tsvector`

pg_upgrade will require a reindex if:

- an index is of type hash or GIN
- an index uses `bpchar_pattern_ops`

Also, the default datetime storage format changed to integer after PostgreSQL 8.3. pg_upgrade will check that the datetime storage format used by the old and new clusters match. Make sure your new cluster is built with the configure flag `--disable-integer-datetimes`.

For Windows users, note that due to different integer datetimes settings used by the graphical installer and the MSI installer, it is only possible to upgrade from version 8.3 of the installer distribution to version 8.4 or later of the installer distribution. It is not possible to upgrade from the MSI installer to the new graphical installer.

See Also

initdb, pg_ctl, pg_dump, postgres

附录 H. 外部项目

PostgreSQL 是一个复杂的软件项目，对其进行管理并非易事。我们发现，许多对 PostgreSQL 的增强若脱离核心项目单独开发，往往能更高效地完成。

H.1. 客户端接口

基础PostgreSQL发行版中只包含两个客户端接口：

- 之所以包含 libpq，是因为它是主要的 C 语言接口，而且许多其他客户端接口都构建在它之上。
- 之所以包含 ECPG，是因为它依赖服务器端 SQL 语法，因此对 PostgreSQL 自身的变更非常敏感。

其他所有语言接口都属于外部项目，并单独发布。表 H.1列出了其中一些项目。请注意，其中有些软件包可能并非按与PostgreSQL相同的许可发布。有关各语言接口的更多信息，包括许可条款，请参阅其网站和文档。

表 H.1. 由外部维护的客户端接口

名称	语言	注释	网站
DBD::Pg	Perl	Perl DBI 驱动	http://search.cpan.org/dist/DBD-Pg/
JDBC	Java	第 4 类 JDBC 驱动	https://jdbc.postgresql.org/
libpqxx	C++	新式 C++ 接口	http://pqxx.org/
node-postgres	JavaScript	Node.js 驱动	https://node-postgres.com/
Npgsql	.NET	.NET 数据提供程序	http://www.npgsql.org/
pgtcl	Tcl		https://github.com/flightaware/Pgtcl
pgtclng	Tcl		http://sourceforge.net/projects/pgtclng/
pq	Go	用于 Go 的 database/sql 的纯 Go 驱动	https://github.com/lib/pq
psqlODBC	ODBC	ODBC 驱动	https://odbc.postgresql.org/
psycopg	Python	符合 DB API 2.0	http://initd.org/psycopg/

H.2. 管理工具

有若干管理工具可用于 PostgreSQL。其中最流行的是 pgAdmin III[1]，另外也有若干商业可用的工具。

H.3. 过程语言

基础 PostgreSQL 发行版中包含若干过程语言：PL/pgSQL、PL/Tcl、PL/Perl 以及 PL/Python。

[1] http://www.pgadmin.org/

此外，还有一些过程语言是在核心PostgreSQL发行版之外开发和维护的。表 H.2列出了其中一些软件包。请注意，其中有些项目可能并非按与PostgreSQL相同的许可发布。
有关各过程语言的更多信息，包括许可信息，请参阅其网站和文档。

表 H.2. 由外部维护的过程语言

名称	语言	网站
PL/Java	Java	https://github.com/tada/pljava
PL/PHP	PHP	https://public.commandprompt.com/projects/plphp
PL/Py	Python	http://python.projects.postgresql.org/backend/
PL/R	R	http://www.joeconway.com/plr.html
PL/Ruby	Ruby	http://raa.ruby-lang.org/project/pl-ruby/
PL/Scheme	Scheme	http://plscheme.projects.postgresql.org/
PL/sh	Unix shell	https://github.com/petere/plsh

H.4. 扩展

PostgreSQL 的设计目标之一，就是使其易于扩展。因此，载入数据库的扩展可以像内置特性一样工作。随源代码一同提供的 `contrib/` 目录中包含若干扩展，它们在附录 F中有所描述。其他扩展则是独立开发的，例如 PostGIS[2]。甚至 PostgreSQL 的复制解决方案也可以在外部开发。例如，Slony-I[3] 就是一个流行的主库/备库复制解决方案，它是独立于核心项目开发的。

[2] http://postgis.net/
[3] http://www.slony.info

附录 I. 源代码仓库

PostgreSQL的源代码使用Git版本控制系统进行存储和管理。官方提供了主仓库的公开镜像；主仓库发生任何变更后，该镜像都会在一分钟内更新。

我们的 wiki（https://wiki.postgresql.org/wiki/Working_with_Git）上有一些关于使用 Git 的讨论。

请注意，从源代码仓库构建PostgreSQL需要版本相当新的 bison、flex和Perl。从发行版 tarball 构建则不需要这些工具，因为这些工具生成的文件已经包含在 tarball 中。其他工具要求与第 15.2 节中的说明相同。

I.1. 通过 Git 获取源代码

使用Git时，你将在本地机器上创建整个代码仓库的一个副本，
因此即使离线也能访问全部历史和分支。这是开发或测试补丁最快且最灵活的方式。

Git

1. 你需要安装Git，可从http://git-scm.com获取。许多系统默认已安装了较新版本的Git，或者可以通过其软件包分发系统获得。

2. 要开始使用 Git 仓库，请克隆官方镜像：

   ```
   git clone git://git.postgresql.org/git/postgresql.git
   ```

   这会将完整的仓库复制到本地机器，因此可能需要一些时间才能完成，
   尤其是网络连接较慢时。文件将被放置在当前目录下新建的`postgresql`子目录中。

   如果例如防火墙阻止了对 Git 协议的访问，也可以通过 HTTP 协议访问 Git 镜像。只需将 URL 前缀改为`https`，如下所示：

   ```
   git clone https://git.postgresql.org/git/postgresql.git
   ```

   HTTP 协议的效率低于 Git 协议，因此使用起来会更慢。

3. 每当想要获取系统的最新更新时，`cd`进入该仓库目录并运行：

   ```
   git fetch
   ```

Git能做的远不止获取源代码。要了解更多信息，请参阅Git的手册页，或者访问http://git-scm.com网站。

附录 J. Documentation

PostgreSQL has four primary documentation formats:

- Plain text, for pre-installation information
- HTML, for on-line browsing and reference
- PDF or PostScript, for printing
- man pages, for quick reference.

Additionally, a number of plain-text `README` files can be found throughout the PostgreSQL source tree, documenting various implementation issues.

HTML documentation and man pages are part of a standard distribution and are installed by default. PDF and PostScript format documentation is available separately for download.

J.1. DocBook

The documentation sources are written in DocBook, which is a markup language superficially similar to HTML. Both of these languages are applications of the Standard Generalized Markup Language, SGML, which is essentially a language for describing other languages. In what follows, the terms DocBook and SGML are both used, but technically they are not interchangeable.

DocBook allows an author to specify the structure and content of a technical document without worrying about presentation details. A document style defines how that content is rendered into one of several final forms. DocBook is maintained by the OASIS group[1]. The official DocBook site[2] has good introductory and reference documentation and a complete O'Reilly book for your online reading pleasure. The NewbieDoc Docbook Guide[3] is very helpful for beginners. The FreeBSD Documentation Project[4] also uses DocBook and has some good information, including a number of style guidelines that might be worth considering.

J.2. Tool Sets

The following tools are used to process the documentation. Some might be optional, as noted.

DocBook DTD[5]

This is the definition of DocBook itself. We currently use version 4.2; you cannot use later or earlier versions. You need the SGML variant of the DocBook DTD, but to build man pages you also need the XML variant of the same version.

ISO 8879 character entities[6]

These are required by DocBook but are distributed separately because they are maintained by ISO.

[1] http://www.oasis-open.org
[2] http://www.oasis-open.org/docbook/
[3] http://newbiedoc.sourceforge.net/metadoc/docbook-guide.html
[4] http://www.freebsd.org/docproj/docproj.html
[5] http://www.oasis-open.org/docbook/
[6] http://www.oasis-open.org/cover/ISOEnts.zip

DocBook DSSSL Stylesheets[7]

These contain the processing instructions for converting the DocBook sources to other formats, such as HTML.

DocBook XSL Stylesheets[8]

This is another stylesheet for converting DocBook to other formats. We currently use this to produce man pages and optionally HTMLHelp. You can also use this toolchain to produce HTML or PDF output, but official PostgreSQL releases use the DSSSL stylesheets for that.

The minimum required version is currently 1.74.0.

OpenJade[9]

This is the base package of SGML processing. It contains an SGML parser, a DSSSL processor (that is, a program to convert SGML to other formats using DSSSL stylesheets), as well as a number of related tools. Jade is now being maintained by the OpenJade group, no longer by James Clark.

Libxslt[10] for `xsltproc`

This is the processing tool to use with the XSLT stylesheets (like `jade` is the processing tool for DSSSL stylesheets).

JadeTeX[11]

If you want to, you can also install JadeTeX to use TeX as a formatting backend for Jade. JadeTeX can create PostScript or PDF files (the latter with bookmarks).

However, the output from JadeTeX is inferior to what you get from the RTF backend. Particular problem areas are tables and various artifacts of vertical and horizontal spacing. Also, there is no opportunity to manually polish the results.

We have documented experience with several installation methods for the various tools that are needed to process the documentation. These will be described below. There might be some other packaged distributions for these tools. Please report package status to the documentation mailing list, and we will include that information here.

J.2.1. Linux RPM Installation

Most vendors provide a complete RPM set for DocBook processing in their distribution. Look for an "SGML" option while installing, or the following packages: `sgml-common`, `docbook`, `stylesheets`, `openjade` (or `jade`). Possibly `sgml-tools` will be needed as well. If your distributor does not provide these then you should be able to make use of the packages from some other, reasonably compatible vendor.

J.2.2. FreeBSD Installation

The FreeBSD Documentation Project is itself a heavy user of DocBook, so it comes as no surprise that there is a full set of "ports" of the documentation tools available on FreeBSD. The following ports need to be installed to build the documentation on FreeBSD.

[7] http://wiki.docbook.org/DocBookDssslStylesheetDocs
[8] http://wiki.docbook.org/DocBookXslStylesheets
[9] http://openjade.sourceforge.net
[10] http://xmlsoft.org/XSLT/
[11] http://jadetex.sourceforge.net

- `textproc/sp`
- `textproc/openjade`
- `textproc/iso8879`
- `textproc/dsssl-docbook-modular`
- `textproc/docbook-420`

A number of things from `/usr/ports/print` (`tex`, `jadetex`) might also be of interest.

It's possible that the ports do not update the main catalog file in `/usr/local/share/sgml/catalog.ports` or that the order isn't proper. Be sure to have the following lines in the beginning of the file:

```
CATALOG "openjade/catalog"
CATALOG "iso8879/catalog"
CATALOG "docbook/dsssl/modular/catalog"
CATALOG "docbook/4.2/catalog"
```

If you do not want to edit the file you can also set the environment variable `SGML_CATALOG_FILES` to a colon-separated list of catalog files (such as the one above).

More information about the FreeBSD documentation tools can be found in the FreeBSD Documentation Project's instructions[12].

J.2.3. Debian Packages

There is a full set of packages of the documentation tools available for Debian GNU/Linux. To install, simply use:

```
apt-get install docbook docbook-dsssl docbook-xsl openjade1.3
 opensp xsltproc
```

J.2.4. Manual Installation from Source

The manual installation process of the DocBook tools is somewhat complex, so if you have pre-built packages available, use them. We describe here only a standard setup, with reasonably standard installation paths, and no "fancy" features. For details, you should study the documentation of the respective package, and read SGML introductory material.

J.2.4.1. Installing OpenJade

1. The installation of OpenJade offers a GNU-style `./configure; make; make install` build process. Details can be found in the OpenJade source distribution. In a nutshell:

   ```
   ./configure --enable-default-catalog=/usr/local/share/sgml/
   catalog
   make
   make install
   ```

   Be sure to remember where you put the "default catalog"; you will need it below. You can also leave it off, but then you will have to set the environment variable

[12] http://www.freebsd.org/doc/en_US.ISO8859-1/books/fdp-primer/tools.html

SGML_CATALOG_FILES to point to the file whenever you use jade later on. (This method is also an option if OpenJade is already installed and you want to install the rest of the toolchain locally.)

> **注意**
>
> Some users have reported encountering a segmentation fault using OpenJade 1.4devel to build the PDFs, with a message like:
>
> ```
> openjade:./stylesheet.dsl:664:2:E: flow object not
> accepted by port; only display flow objects accepted
> make: *** [postgres-A4.tex-pdf] Segmentation fault
> ```
>
> Downgrading to OpenJade 1.3 should get rid of this error.

2. Additionally, you should install the files dsssl.dtd, fot.dtd, style-sheet.dtd, and catalog from the dsssl directory somewhere, perhaps into /usr/local/share/sgml/dsssl. It's probably easiest to copy the entire directory:

   ```
   cp -R dsssl /usr/local/share/sgml
   ```

3. Finally, create the file /usr/local/share/sgml/catalog and add this line to it:

   ```
   CATALOG "dsssl/catalog"
   ```

   (This is a relative path reference to the file installed in 步骤 2. Be sure to adjust it if you chose your installation layout differently.)

J.2.4.2. Installing the DocBook DTD Kit

1. Obtain the DocBook V4.2 distribution[13].

2. Create the directory /usr/local/share/sgml/docbook-4.2 and change to it. (The exact location is irrelevant, but this one is reasonable within the layout we are following here.)

   ```
   $ mkdir /usr/local/share/sgml/docbook-4.2
   $ cd /usr/local/share/sgml/docbook-4.2
   ```

3. Unpack the archive:

   ```
   $ unzip -a ...../docbook-4.2.zip
   ```

   (The archive will unpack its files into the current directory.)

4. Edit the file /usr/local/share/sgml/catalog (or whatever you told jade during installation) and put a line like this into it:

   ```
   CATALOG "docbook-4.2/docbook.cat"
   ```

5. Download the ISO 8879 character entities archive[14], unpack it, and put the files in the same directory you put the DocBook files in:

   ```
   $ cd /usr/local/share/sgml/docbook-4.2
   $ unzip ...../ISOEnts.zip
   ```

[13] http://www.docbook.org/sgml/4.2/docbook-4.2.zip
[14] http://www.oasis-open.org/cover/ISOEnts.zip

6. Run the following command in the directory with the DocBook and ISO files:

   ```
   perl -pi -e 's/iso-(.*).gml/ISO\1/g' docbook.cat
   ```

   (This fixes a mixup between the names used in the DocBook catalog file and the actual names of the ISO character entity files.)

J.2.4.3. Installing the DocBook DSSSL Style Sheets

To install the style sheets, unzip and untar the distribution and move it to a suitable place, for example `/usr/local/share/sgml`. (The archive will automatically create a subdirectory.)

```
$ gunzip docbook-dsssl-1.xx.tar.gz
$ tar -C /usr/local/share/sgml -xf docbook-dsssl-1.xx.tar
```

The usual catalog entry in `/usr/local/share/sgml/catalog` can also be made:

```
CATALOG "docbook-dsssl-1.xx/catalog"
```

Because stylesheets change rather often, and it's sometimes beneficial to try out alternative versions, PostgreSQL doesn't use this catalog entry. See 第 J.2.5 节 for information about how to select the stylesheets instead.

J.2.4.4. Installing JadeTeX

To install and use JadeTeX, you will need a working installation of TeX and LaTeX2e, including the supported tools and graphics packages, Babel, AMS fonts and AMS-LaTeX, the PSNFSS extension and companion kit of “the 35 fonts” , the dvips program for generating PostScript, the macro packages fancyhdr, hyperref, minitoc, url and ot2enc. All of these can be found on your friendly neighborhood CTAN site[15]. The installation of the TeX base system is far beyond the scope of this introduction. Binary packages should be available for any system that can run TeX.

Before you can use JadeTeX with the PostgreSQL documentation sources, you will need to increase the size of TeX's internal data structures. Details on this can be found in the JadeTeX installation instructions.

Once that is finished you can install JadeTeX:

```
$ gunzip jadetex-xxx.tar.gz
$ tar xf jadetex-xxx.tar
$ cd jadetex
$ make install
$ mktexlsr
```

The last two need to be done as root.

J.2.5. Detection by `configure`

Before you can build the documentation you need to run the `configure` script as you would when building the PostgreSQL programs themselves. Check the output near the end of the run, it should look something like this:

```
checking for onsgmls... onsgmls
checking for openjade... openjade
```

[15] http://www.ctan.org

```
checking for DocBook V4.2... yes
checking for DocBook stylesheets... /usr/share/sgml/docbook/
stylesheet/dsssl/modular
checking for collateindex.pl... /usr/bin/collateindex.pl
checking for xsltproc... xsltproc
checking for osx... osx
```

If neither `onsgmls` nor `nsgmls` were found then some of the following tests will be skipped. `nsgmls` is part of the Jade package. You can pass the environment variables `JADE` and `NSGMLS` to configure to point to the programs if they are not found automatically. If "DocBook V4.2" was not found then you did not install the DocBook DTD kit in a place where Jade can find it, or you have not set up the catalog files correctly. See the installation hints above. The DocBook stylesheets are looked for in a number of relatively standard places, but if you have them some other place then you should set the environment variable `DOCBOOKSTYLE` to the location and rerun `configure` afterwards.

J.3. Building The Documentation

Once you have everything set up, change to the directory `doc/src/sgml` and run one of the commands described in the following subsections to build the documentation. (Remember to use GNU make.)

J.3.1. HTML

To build the HTML version of the documentation:

```
doc/src/sgml$ gmake html
```

This is also the default target. The output appears in the subdirectory `html`.

To create a proper index, the build might process several identical stages. If you do not care about the index, and just want to proof-read the output, use `draft`:

```
doc/src/sgml$ gmake draft
```

To build the documentation as a single HTML page, use:

```
doc/src/sgml$ gmake postgres.html
```

J.3.2. Manpages

We use the DocBook XSL stylesheets to convert DocBook `refentry` pages to *roff output suitable for man pages. The man pages are also distributed as a tar archive, similar to the HTML version. To create the man pages, use the commands:

```
cd doc/src/sgml
gmake man
```

J.3.3. Print Output via JadeTeX

If you want to use JadeTex to produce a printable rendition of the documentation, you can use one of the following commands:

- To generate PostScript via DVI in A4 format:

  ```
  doc/src/sgml$ gmake postgres-A4.ps
  ```

In U.S. letter format:

```
doc/src/sgml$ gmake postgres-US.ps
```

- To make a PDF:

  ```
  doc/src/sgml$ gmake postgres-A4.pdf
  ```

  or:

  ```
  doc/src/sgml$ gmake postgres-US.pdf
  ```

  (Of course you can also make a PDF version from the PostScript, but if you generate PDF directly, it will have hyperlinks and other enhanced features.)

When using JadeTeX to build the PostgreSQL documentation, you will probably need to increase some of TeX's internal parameters. These can be set in the file `texmf.cnf` . The following settings worked at the time of this writing:

```
hash_extra.jadetex  = 200000
hash_extra.pdfjadetex  = 200000
pool_size.jadetex = 2000000
pool_size.pdfjadetex = 2000000
string_vacancies.jadetex = 150000
string_vacancies.pdfjadetex = 150000
max_strings.jadetex = 300000
max_strings.pdfjadetex = 300000
save_size.jadetex = 15000
save_size.pdfjadetex = 15000
```

J.3.4. Overflow Text

Occasionally text is too wide for the printed margins, and in extreme cases, too wide for the printed page, e.g. non-wrapped text, wide tables. Overly wide text generates "Overfull hbox" messages in the TeX log output file, e.g. `postgres-US.log` or `postgres-A4.log`. There are 72 points in an inch so anything reported as over 72 points too wide will probably not fit on the printed page (assuming one inch margins). To find the SGML text causing the overflow, find the first page number mentioned above the overflow message, e.g. `[50 ###]` (page 50), and look at the page after that (e.g. page 51) in the PDF file to see the overflow text and adjust the SGML accordingly.

J.3.5. Print Output via RTF

You can also create a printable version of the PostgreSQL documentation by converting it to RTF and applying minor formatting corrections using an office suite. Depending on the capabilities of the particular office suite, you can then convert the documentation to PostScript of PDF. The procedure below illustrates this process using Applixware.

> **注意**
>
> It appears that current versions of the PostgreSQL documentation trigger some bug in or exceed the size limit of OpenJade. If the build process of the RTF version hangs for a long time and the output file still has size 0, then you might have hit that problem. (But keep in mind that a normal build takes 5 to 10 minutes, so don't abort too soon.)

Applixware RTF Cleanup

OpenJade omits specifying a default style for body text. In the past, this undiagnosed problem led to a long process of table of contents generation. However, with great help from the Applixware folks the symptom was diagnosed and a workaround is available.

1. Generate the RTF version by typing:

   ```
   doc/src/sgml$ gmake postgres.rtf
   ```

2. Repair the RTF file to correctly specify all styles, in particular the default style. If the document contains `refentry` sections, one must also replace formatting hints which tie a preceding paragraph to the current paragraph, and instead tie the current paragraph to the following one. A utility, `fixrtf`, is available in `doc/src/sgml` to accomplish these repairs:

   ```
   doc/src/sgml$ ./fixrtf --refentry postgres.rtf
   ```

   The script adds `{\s0 Normal;}` as the zeroth style in the document. According to Applixware, the RTF standard would prohibit adding an implicit zeroth style, though Microsoft Word happens to handle this case. For repairing `refentry` sections, the script replaces `\keepn` tags with `\keep`.

3. Open a new document in Applixware Words and then import the RTF file.

4. Generate a new table of contents (ToC) using Applixware.

   a. Select the existing ToC lines, from the beginning of the first character on the first line to the last character of the last line.

   b. Build a new ToC using Tools → Book Building → Create Table of Contents. Select the first three levels of headers for inclusion in the ToC. This will replace the existing lines imported in the RTF with a native Applixware ToC.

   c. Adjust the ToC formatting by using Format → Style, selecting each of the three ToC styles, and adjusting the indents for `First` and `Left`. Use the following values:

Style	First Indent (inches)	Left Indent (inches)
TOC-Heading 1	0.4	0.4
TOC-Heading 2	0.8	0.8
TOC-Heading 3	1.2	1.2

5. Work through the document to:

   - Adjust page breaks.
   - Adjust table column widths.

6. Replace the right-justified page numbers in the Examples and Figures portions of the ToC with correct values. This only takes a few minutes.

7. (可选的) Delete the index section from the document if it is empty.

8. Regenerate and adjust the table of contents.

   a. Select the ToC field.

   b. Select Tools → Book Building → Create Table of Contents.

c. Unbind the ToC by selecting Tools → Field Editing → Unprotect.

d. Delete the first line in the ToC, which is an entry for the ToC itself.

9. Save the document as native Applixware Words format to allow easier last minute editing later.

10. "Print" the document to a file in PostScript format.

J.3.6. Plain Text Files

The installation instructions are also distributed as plain text, in case they are needed in a situation where better reading tools are not available. The `INSTALL` file corresponds to 第 15 章, with some minor changes to account for the different context. To recreate the file, change to the directory `doc/src/sgml` and enter **`gmake INSTALL`**.

In the past, the release notes and regression testing instructions were also distributed as plain text, but this practice has been discontinued.

J.3.7. Syntax Check

Building the documentation can take very long. But there is a method to just check the correct syntax of the documentation files, which only takes a few seconds:

```
doc/src/sgml$ gmake check
```

J.4. Documentation Authoring

SGML and DocBook do not suffer from an oversupply of open-source authoring tools. The most common tool set is the Emacs/XEmacs editor with appropriate editing mode. On some systems these tools are provided in a typical full installation.

J.4.1. Emacs/PSGML

PSGML is the most common and most powerful mode for editing SGML documents. When properly configured, it will allow you to use Emacs to insert tags and check markup consistency. You could use it for HTML as well. Check the PSGML web site[16] for downloads, installation instructions, and detailed documentation.

There is one important thing to note with PSGML: its author assumed that your main SGML DTD directory would be `/usr/local/lib/sgml`. If, as in the examples in this chapter, you use `/usr/local/share/sgml`, you have to compensate for this, either by setting `SGML_CATALOG_FILES` environment variable, or you can customize your PSGML installation (its manual tells you how).

Put the following in your `~/.emacs` environment file (adjusting the path names to be appropriate for your system):

```
; ********** for SGML mode (psgml)

(setq sgml-omittag t)
(setq sgml-shorttag t)
(setq sgml-minimize-attributes nil)
(setq sgml-always-quote-attributes t)
(setq sgml-indent-step 1)
```

[16] http://www.lysator.liu.se/projects/about_psgml.html

```
(setq sgml-indent-data t)
(setq sgml-parent-document nil)
(setq sgml-exposed-tags nil)
(setq sgml-catalog-files '("/usr/local/share/sgml/catalog"))

(autoload 'sgml-mode "psgml" "Major mode to edit SGML files." t )
```

and in the same file add an entry for SGML into the (existing) definition for `auto-mode-alist`:

```
(setq
  auto-mode-alist
  '(("\\.sgml$" . sgml-mode)
   ))
```

You might find that when using PSGML, a comfortable way of working with these separate files of book parts is to insert a proper `DOCTYPE` declaration while you're editing them. If you are working on this source, for instance, it is an appendix chapter, so you would specify the document as an "appendix" instance of a DocBook document by making the first line look like this:

```
<!DOCTYPE appendix PUBLIC "-//OASIS//DTD DocBook V4.2//EN">
```

This means that anything and everything that reads SGML will get it right, and I can verify the document with `nsgmls -s docguide.sgml`. (But you need to take out that line before building the entire documentation set.)

J.4.2. Other Emacs Modes

GNU Emacs ships with a different SGML mode, which is not quite as powerful as PSGML, but it's less confusing and lighter weight. Also, it offers syntax highlighting (font lock), which can be very helpful. `src/tools/editors/emacs.samples` contains sample settings for this mode.

Norm Walsh offers a major mode[17] specifically for DocBook which also has font-lock and a number of features to reduce typing.

J.5. Style Guide

J.5.1. Reference Pages

Reference pages should follow a standard layout. This allows users to find the desired information more quickly, and it also encourages writers to document all relevant aspects of a command. Consistency is not only desired among PostgreSQL reference pages, but also with reference pages provided by the operating system and other packages. Hence the following guidelines have been developed. They are for the most part consistent with similar guidelines established by various operating systems.

Reference pages that describe executable commands should contain the following sections, in this order. Sections that do not apply can be omitted. Additional top-level sections should only be used in special circumstances; often that information belongs in the "Usage" section.

Name

This section is generated automatically. It contains the command name and a half-sentence summary of its functionality.

[17] http://nwalsh.com/emacs/docbookide/index.html

Synopsis

This section contains the syntax diagram of the command. The synopsis should normally not list each command-line option; that is done below. Instead, list the major components of the command line, such as where input and output files go.

Description

Several paragraphs explaining what the command does.

Options

A list describing each command-line option. If there are a lot of options, subsections can be used.

Exit Status

If the program uses 0 for success and non-zero for failure, then you do not need to document it. If there is a meaning behind the different non-zero exit codes, list them here.

Usage

Describe any sublanguage or run-time interface of the program. If the program is not interactive, this section can usually be omitted. Otherwise, this section is a catch-all for describing run-time features. Use subsections if appropriate.

Environment

List all environment variables that the program might use. Try to be complete; even seemingly trivial variables like `SHELL` might be of interest to the user.

Files

List any files that the program might access implicitly. That is, do not list input and output files that were specified on the command line, but list configuration files, etc.

Diagnostics

Explain any unusual output that the program might create. Refrain from listing every possible error message. This is a lot of work and has little use in practice. But if, say, the error messages have a standard format that the user can parse, this would be the place to explain it.

Notes

Anything that doesn't fit elsewhere, but in particular bugs, implementation flaws, security considerations, compatibility issues.

Examples

Examples

History

If there were some major milestones in the history of the program, they might be listed here. Usually, this section can be omitted.

Author

Author (only used in the contrib section)

See Also

Cross-references, listed in the following order: other PostgreSQL command reference pages, PostgreSQL SQL command reference pages, citation of PostgreSQL manuals, other reference pages (e.g., operating system, other packages), other documentation. Items in the same group are listed alphabetically.

Reference pages describing SQL commands should contain the following sections: Name, Synopsis, Description, Parameters, Outputs, Notes, Examples, Compatibility, History, See Also. The Parameters section is like the Options section, but there is more freedom about which clauses of the command can be listed. The Outputs section is only needed if the command returns something other than a default command-completion tag. The Compatibility section should explain to what extent this command conforms to the SQL standard(s), or to which other database system it is compatible. The See Also section of SQL commands should list SQL commands before cross-references to programs.

附录 K. 缩略词

这是在PostgreSQL文档和有关PostgreSQL的讨论中常用的缩略词列表。

ANSI

美国国家标准协会[1]

API

应用程序编程接口[2]

ASCII

美国信息交换标准代码[3]

BKI

后端接口

CA

证书颁发机构[4]

CIDR

无类域间路由[5]

CPAN

Perl综合档案网络[6]

CRL

证书吊销列表[7]

CSV

逗号分隔值[8]

CTE

公共表表达式

CVE

常见漏洞与暴露[9]

DBA

数据库管理员[10]

[1] http://en.wikipedia.org/wiki/American_National_Standards_Institute
[2] http://en.wikipedia.org/wiki/API
[3] http://en.wikipedia.org/wiki/Ascii
[4] http://en.wikipedia.org/wiki/Certificate_authority
[5] http://en.wikipedia.org/wiki/Classless_Inter-Domain_Routing
[6] http://www.cpan.org/
[7] http://en.wikipedia.org/wiki/Certificate_revocation_list
[8] http://en.wikipedia.org/wiki/Comma-separated_values
[9] http://cve.mitre.org/
[10] http://en.wikipedia.org/wiki/Database_administrator

DBI

数据库接口（Perl）[11]

DBMS

数据库管理系统[12]

DDL

数据定义语言[13]，例如SQL命令`CREATE TABLE`、`ALTER USER`

DML

数据操纵语言[14]，例如SQL命令`INSERT`、`UPDATE`、`DELETE`

DST

夏令时[15]

ECPG

用于 PostgreSQL 的嵌入式 C

ESQL

嵌入式SQL[16]

FAQ

常见问题[17]

FSM

空闲空间映射

GEQO

遗传查询优化器

GIN

通用倒排索引

GiST

通用搜索树

Git

Git[18]

GMT

格林尼治标准时间[19]

[11] http://dbi.perl.org/
[12] http://en.wikipedia.org/wiki/Dbms
[13] http://en.wikipedia.org/wiki/Data_Definition_Language
[14] http://en.wikipedia.org/wiki/Data_Manipulation_Language
[15] http://en.wikipedia.org/wiki/Daylight_saving_time
[16] http://en.wikipedia.org/wiki/Embedded_SQL
[17] http://en.wikipedia.org/wiki/FAQ
[18] http://en.wikipedia.org/wiki/Git_(software)
[19] http://en.wikipedia.org/wiki/GMT

GSSAPI

通用安全服务应用程序接口[20]

GUC

Grand Unified Configuration，即负责处理服务器配置的PostgreSQL子系统

HBA

基于主机的认证

HOT

堆内元组[21]

IEC

国际电工委员会[22]

IEEE

电气和电子工程师学会[23]

IPC

进程间通信[24]

ISO

国际标准化组织[25]

ISSN

国际标准连续出版物编号[26]

JDBC

Java数据库连接[27]

LDAP

轻量级目录访问协议[28]

MSVC

Microsoft Visual C[29]

MVCC

多版本并发控制

NLS

本地语言支持[30]

[20] http://en.wikipedia.org/wiki/Generic_Security_Services_Application_Program_Interface
[21] https://git.postgresql.org/gitweb?p=postgresql.git;a=blob;f=src/backend/access/heap/README.HOT;hb=HEAD
[22] http://en.wikipedia.org/wiki/International_Electrotechnical_Commission
[23] http://standards.ieee.org/
[24] http://en.wikipedia.org/wiki/Inter-process_communication
[25] http://www.iso.org/iso/home.htm
[26] http://en.wikipedia.org/wiki/Issn
[27] http://en.wikipedia.org/wiki/Java_Database_Connectivity
[28] http://en.wikipedia.org/wiki/Lightweight_Directory_Access_Protocol
[29] http://en.wikipedia.org/wiki/Visual_C++
[30] http://en.wikipedia.org/wiki/Internationalization_and_localization

ODBC

开放数据库连接[31]

OID

对象标识符

OLAP

联机分析处理[32]

OLTP

联机事务处理[33]

ORDBMS

对象关系数据库管理系统[34]

PAM

可插拔认证模块[35]

PGSQL

PostgreSQL

PGXS

PostgreSQL扩展系统

PID

进程标识符[36]

PITR

时间点恢复（持续归档）

PL

过程语言（服务器端）

POSIX

可移植操作系统接口[37]

RDBMS

关系数据库管理系统[38]

RFC

征求意见稿[39]

[31] http://en.wikipedia.org/wiki/Open_Database_Connectivity
[32] http://en.wikipedia.org/wiki/Olap
[33] http://en.wikipedia.org/wiki/OLTP
[34] http://en.wikipedia.org/wiki/ORDBMS
[35] http://en.wikipedia.org/wiki/Pluggable_Authentication_Modules
[36] http://en.wikipedia.org/wiki/Process_identifier
[37] http://en.wikipedia.org/wiki/POSIX
[38] http://en.wikipedia.org/wiki/Relational_database_management_system
[39] http://en.wikipedia.org/wiki/Request_for_Comments

SGML

标准通用标记语言[40]

SPI

服务器编程接口

SP-GiST

空间分区通用搜索树

SQL

结构化查询语言[41]

SRF

集合返回函数

SSH

安全外壳协议[42]

SSL

安全套接字层[43]

SSPI

安全支持提供程序接口[44]

SYSV

Unix System V[45]

TCP/IP

传输控制协议（TCP）/ 互联网协议（IP）[46]

TID

元组标识符

TOAST

超长属性存储技术

TPC

事务处理性能委员会[47]

URL

统一资源定位符[48]

[40] http://en.wikipedia.org/wiki/SGML
[41] http://en.wikipedia.org/wiki/SQL
[42] http://en.wikipedia.org/wiki/Secure_Shell
[43] http://en.wikipedia.org/wiki/Secure_Sockets_Layer
[44] http://msdn.microsoft.com/en-us/library/aa380493%28VS.85%29.aspx
[45] http://en.wikipedia.org/wiki/System_V
[46] http://en.wikipedia.org/wiki/Transmission_Control_Protocol
[47] http://www.tpc.org/
[48] http://en.wikipedia.org/wiki/URL

UTC

协调世界时[49]

UTF

Unicode 转换格式[50]

UTF8

8 位 Unicode 转换格式[51]

UUID

通用唯一标识符

WAL

预写式日志

XID

事务 ID

XML

可扩展标记语言[52]

[49] http://en.wikipedia.org/wiki/Coordinated_Universal_Time
[50] http://www.unicode.org/
[51] http://en.wikipedia.org/wiki/Utf8
[52] http://en.wikipedia.org/wiki/XML

参考书目

这里列出了一些关于 SQL 和 PostgreSQL 的参考文献与读物。

来自最初的 POSTGRES 开发团队的一些白皮书和技术报告，可在加州大学伯克利分校计算机系的网站[1]上获取。

SQL参考书

Reference texts for SQL features.

[bowman01] The Practical SQL Handbook. Bowman et al, 2001. Using SQL Variants. Fourth Edition. Judith Bowman、Sandra Emerson和Marcy Darnovsky. ISBN 0-201-70309-2. 2001. Addison-Wesley Professional. 版权 © 2001.

[date97] A Guide to the SQL Standard. Date and Darwen, 1997. A user's guide to the standard database language SQL. Fourth Edition. C. J. Date和Hugh Darwen. ISBN 0-201-96426-0. 1997. Addison-Wesley. 版权 © 1997 Addison-Wesley Longman, Inc..

[date04] An Introduction to Database Systems. Date, 2004. Eighth Edition. C. J. Date. ISBN 0-321-19784-4. 2003. Addison-Wesley. 版权 © 2004 Pearson Education, Inc..

[elma04] Fundamentals of Database Systems. Fourth Edition. Ramez Elmasri和Shamkant Navathe. ISBN 0-321-12226-7. 2003. Addison-Wesley. 版权 © 2004.

[melt93] Understanding the New SQL. Melton and Simon, 1993. A complete guide. Jim Melton 和Alan R. Simon. ISBN 1-55860-245-3. 1993. Morgan Kaufmann. 版权 © 1993 Morgan Kaufmann Publishers, Inc..

[ull88] Principles of Database and Knowledge. Base Systems. Ullman, 1988. Jeffrey D. Ullman. Volume 1. Computer Science Press. 1988.

PostgreSQL相关文档

本节收录相关文档。

[sim98] Enhancement of the ANSI SQL Implementation of PostgreSQL. Simkovics, 1998. Stefan Simkovics.

该文讨论了 SQL 的历史和语法，并描述了将 `INTERSECT` 和 `EXCEPT` 构造加入 PostgreSQL 的过程。它是作者在维也纳技术大学、在 O. Univ. Prof. Dr. Georg Gottlob 和 Univ. Ass. Mag. Katrin Seyr 支持下完成的硕士学位论文。

November 29, 1998. Department of Information Systems, Vienna University of Technology. Vienna, Austria.

[yu95] The Postgres95. User Manual. Yu and Chen, 1995. A. Yu和J. Chen. . Sept. 5, 1995. University of California. Berkeley, California.

[fong] The design and implementation of the POSTGRES query optimizer [2]. Zelaine Fong. University of California, Berkeley, Computer Science Department.

会议论文和期刊文章

本节收录文章和简报。

[1] http://db.cs.berkeley.edu/papers/
[2] http://db.cs.berkeley.edu/papers/UCB-MS-zfong.pdf

[olson93] Partial indexing in POSTGRES: research project. Olson, 1993. Nels Olson. 1993. UCB Engin T7.49.1993 O676. University of California. Berkeley, California.

[ong90] “A Unified Framework for Version Modeling Using Production Rules in a Database System”. Ong and Goh, 1990. L. Ong和J. Goh. ERL Technical Memorandum M90/33. April, 1990. University of California. Berkeley, California.

[rowe87] “ The POSTGRES data model [3]”. Rowe and Stonebraker, 1987. L. Rowe和M. Stonebraker. VLDB Conference, Sept. 1987.

[seshadri95] “Generalized Partial Indexes (cached version) [4] ”. Seshardri, 1995. P. Seshadri和A. Swami. Eleventh International Conference on Data Engineering, 6-10 March 1995. 1995. Cat. No.95CH35724. IEEE Computer Society Press. Los Alamitos, California. 420-7.

[ston86] “ The design of POSTGRES [5]”. Stonebraker and Rowe, 1986. M. Stonebraker和L. Rowe. ACM-SIGMOD Conference on Management of Data, May 1986.

[ston87a] “The design of the POSTGRES. rules system”. Stonebraker, Hanson, Hong, 1987. M. Stonebraker、E. Hanson和C. H. Hong. IEEE Conference on Data Engineering, Feb. 1987.

[ston87b] “ The design of the POSTGRES storage system [6]”. Stonebraker, 1987. M. Stonebraker. VLDB Conference, Sept. 1987.

[ston89] “ A commentary on the POSTGRES rules system [7]”. Stonebraker et al, 1989. M. Stonebraker、M. Hearst和S. Potamianos. SIGMOD Record 18(3). Sept. 1989.

[ston89b] “ The case for partial indexes [8]”. Stonebraker, M, 1989b. M. Stonebraker. SIGMOD Record 18(4). 4-11. Dec. 1989.

[ston90a] “ The implementation of POSTGRES [9]”. Stonebraker, Rowe, Hirohama, 1990. M. Stonebraker、L. A. Rowe和M. Hirohama. Transactions on Knowledge and Data Engineering 2(1). IEEE. March 1990.

[ston90b] “ On Rules, Procedures, Caching and Views in Database Systems [10]”. Stonebraker et al, ACM, 1990. M. Stonebraker、A. Jhingran、J. Goh和S. Potamianos. ACM-SIGMOD Conference on Management of Data, June 1990.

[3] http://db.cs.berkeley.edu/papers/ERL-M87-13.pdf
[4] http://citeseer.ist.psu.edu/seshadri95generalized.html
[5] http://db.cs.berkeley.edu/papers/ERL-M85-95.pdf
[6] http://db.cs.berkeley.edu/papers/ERL-M87-06.pdf
[7] http://db.cs.berkeley.edu/papers/ERL-M89-82.pdf
[8] http://db.cs.berkeley.edu/papers/ERL-M89-17.pdf
[9] http://db.cs.berkeley.edu/papers/ERL-M90-34.pdf
[10] http://db.cs.berkeley.edu/papers/ERL-M90-36.pdf

索引

符号

$，37
$libdir，781
$libdir/plugins，428，1275
*，90
.pgpass，598
.pg_service.conf，598
::，41
_PG_fini，781
_PG_init，781
下标，38
不可重复读，314
不带时区的时间，108
不带时区的时间戳，108
串行化异常，317
主机名，556
事件日志
  事件日志，392
事务 ID
  回卷，474
事务日志 (见 WAL)
事务隔离，314
事务隔离级别，314
  可串行化，317
  可重复读，316
  设置，1332
  设置默认值，426
  读已提交，315
二进制串
  串接，166
  长度，168
二进制数据，106
  函数，166
交叉编译，354
交叉连接，
仅索引扫描，410
任意精度数字，101
休眠，199
位串
  函数，168
  常量，33
  数据类型，122
位图扫描，273，409
保存点
  回滚，1297
  定义，1299
  释放，1289
信号
  后端进程，250
信号量，378
信息模式，717
值表达式，36
儒略日，1681
全局数据
  在 PL/Python 中，942
  在 PL/Tcl 中，917
全文搜索，280
  数据类型，123
全文检索
  函数和操作符，123
公共表表达式 (见 WITH)
共享内存，378
共享库，358，790
关键词
  语法，29
关闭，385
内存上下文
  在 SPI 中，990
准备一个查询
  在 PL/pgSQL 中，903
准备查询
  在 PL/Python 中，944
  在 PL/Tcl 中，918
函数
  位置记法，47
  命名记法，48
  在FROM子句中，86
  多态，764
  混合记法，48
  调用，39
  调用中的类型解析，263
函数依赖，89
分组，87
列引用，37
删除，78
别名
  在FROM子句中，84
  在选择列表中，90
  查询中的表名，12
加密，388
动态装载，429
区域设置，375，462
匿名代码块，1216
升级，385
协方差
  总体，231
  样本，231
历史
  PostgreSQL 的，xxxi
参数
  语法，37
双精度，102
反斜线转义，31
取消
  SQL 命令，580
只读事务
  设置，1332
  设置默认值，426
可串行化，317
可串行化快照隔离，314
可延迟事务
  设置，1332

设置默认值，426
可重复读，316
右连接，
合取，148
同步复制，491
同步提交，534
名称
语法，29
否定，148
咨询锁，322
回归截距，231
回归斜率，231
回归测试，357
圆括号，37
基础类型，763
备份，250，479
备库，491
复制，491
复合类型，135，763
常量，136
构造器，44
外连接，
多态函数，764
多态类型，764
多版本并发控制，314
大小写敏感性
SQL 命令，29
子查询，43，85，233
字段选择，38
字符，104
字符串
串接，155
反斜线引号，430
常量，30
数据类型，104
符合标准，431
转义警告，431
长度，155
字符集，428，433，466
安装，347
在 Windows 上，369
客户端认证
期间超时，397
密码，453
超级用户的，375
密码文件，598
对象标识符
数据类型，145
左连接，
布尔
操作符 (见操作符，逻辑)
带时区的时间，108，110
带时区的时间戳，108，111
常量，30
平均值，229
并发，314
幻读，314
序列，220
延迟，199
异步提交，534
引号
与标识符，30
转义，30
引导
期间启动服务器，376
扩展，823
外部维护，2549
扩展 SQL，763
持续归档，479
排序，92
排序操作符，821
排序规则，464
接口
外部维护，2548
控制文件，823
插入，77
操作符
优先级，35
用户定义的，809
语法，34
调用，38
调用中的类型解析，260
逻辑，148
操作符族，276，819
操作符类，276，813
故障切换，491
数字
常量，33
数字（数据类型），101
数据分区，491
数据区域 (见数据库集簇)
数据库
创建权限，453
数据库活动
监控，511
数据库集簇，374
数据类型，99
enumerated (enum)，117
分类，260
基础，763
复合，763
常量，34
数字，100
用户定义的，806
转换，259
数组，127
I/O，134
修改，131
声明，127
常量，128
搜索，133
构造器，43
用户定义类型的，808
访问，129
整数，101
文件系统挂载点，375

文本搜索，280
  函数和操作符，123
  数据类型，123
  索引，308
文档
  全文搜索，280
方差
  总体，231
  样本，231
无时区的时间，110
无时区的时间戳，111
日常维护，472
日志传送，491
日志写到哪里，414
日期
  当前，197
时区，427
  转换，197
  输入缩写，1680
时区名称，427
时区数据，354
时间
  当前，197
时间点恢复，479
时间线，479
时间跨度，108
更新，78
有效数字，427
服务器日志，413
  日志文件维护，478
服务器欺骗，388
权限
  与规则，856
  与视图，856
条件表达式，222
析取，148
查询，80
标准差，231
  总体，231
  样本，231
标签 (见别名)
标识符
  语法，29
  长度，29
标量 (见表达式)
格式化，182
格里高利历，1681
检查点，535
模式，457
模式匹配，169
正则表达式，171，171
  (参见模式匹配)
  与区域设置，463
死锁，321
  期间超时，429
比较
  子查询结果行，233
  按行，235
  操作符，148
注释
  在 SQL 中，35
流复制，491
浮点，102
浮点数
  显示，427
清理，472
温备，491
游标
  显示查询计划，1255
热备，491
版本
  兼容性，385
环境变量，596
用户，452
登录权限，453
监控
  数据库活动，511
目标列表，840
相关性，231
磁盘使用情况，530
磁盘空间，473
磁盘驱动器，536
空值
  在 DISTINCT 中，91
  在 PL/Perl 中，924
  在 PL/Python 中，939
  比较，149
窗口函数
  invocation，40
  内置，231
  执行顺序，89
端口，556
管理工具
  外部维护，2548
类型 (见数据类型)
  多态，764
类型转换，34
索引，269
  B-树，270
  基于表达式，274
  GIN，271
    文本搜索，308
  GiST，270，1611
    文本搜索，308
  hash，270
  SP-GiST，271，1619
  与 ORDER BY，272
  唯一，273
  多列，271
  并发构建，1136
  检查使用情况，278
  用于用户定义的数据类型，813
  组合多个索引，273
  部分，274
  锁，324
索引扫描，410

约束排除，413
级联复制，491
线性回归，230
线段，119
线程
　用于 libpq，602
统计，230
统计信息，512
　规划器的，473
继承，431
维护，472
编译
　libpq 应用，603
网络文件系统，375
网络附加存储（NAS）(见网络文件系统)
美元引用，32
聚合函数
　内置，229
　调用，39
脏读，314
自动清理
　一般信息，476
　配置参数，423
自然连接，82
范围表，839
行估计
　规划器，1644
行比较，235
行类型，135
　构造器，44
表函数，86
表空间，460
　临时，425
　默认，425
表表达式，80
表达式
　计算的顺序，46
　语法，36
见证服务器，491
规则
　与触发器比较，858
角色，452
　创建权限，453
　发起复制的权限，453
　成员资格，454
触发器，831
　与规则比较，858
　在 C 中，833
　在 PL/Python 中，943
　在 PL/Tcl 中，920
　用于更新一个派生的 tsvector 列，292
　触发器函数的参数，832
词元，29
读已提交，315
谓词锁，317
负载均衡，491
超时
　客户端认证，397
　死锁，429
超级用户，453
路径
　用于模式，424
转义字符串
　in libpq，573
转义字符串语法，31
输入函数，806
输出函数，806
过程语言，861
　外部维护，2548
　调用处理器，1589
连接，10，81
　交叉，
　右，
　外，
　外连接，11
　左，
　自然，82
　自连接，12
连接服务文件，598
通知处理器，589
通知接收器，589
遗传查询优化，412
配置
　服务器
　　函数，249
　恢复
　　备库，508
　服务器端，393
重启点，536
重复，91
重建索引，477
重载
　操作符，809
锁，318
　咨询，322
　监控，522
错误消息，562
长度
　二进制串 (见二进制串，长度)
　字符串 (见字符串，长度)
间隔，108
集合交，91
集合差，91
集合并，91
集合操作，91
集合返回函数
　函数，238
集簇，491
非数字
　double precision，102
　数字（数据类型），102
非空约束，52
非阻塞连接，551，576
顺序扫描，410
预备语句
　显示查询计划，1255

高可用，491

A

abbrev，204
ABORT，1011
abs，151
acos，154
adminpack，2374
age，189
aggregate function，12
  user-defined，803
AIX
  IPC 配置，
  在其上安装，360
ALL，233，235
allow_system_table_mods配置参数，434
ALTER AGGREGATE，1012
ALTER COLLATION，1014
ALTER CONVERSION，1015
ALTER DATABASE，1016
ALTER DEFAULT PRIVILEGES，1018
ALTER DOMAIN，1021
ALTER EXTENSION，1024
ALTER FOREIGN DATA WRAPPER，1027
ALTER FOREIGN TABLE，1029
ALTER FUNCTION，1032
ALTER GROUP，1035
ALTER INDEX，1036
ALTER LANGUAGE，1038
ALTER LARGE OBJECT，1039
ALTER OPERATOR，1040
ALTER OPERATOR CLASS，1041
ALTER OPERATOR FAMILY，1042
ALTER ROLE，453，1046
ALTER SCHEMA，1049
ALTER SEQUENCE，1050
ALTER SERVER，1053
ALTER TABLE，1055
ALTER TABLESPACE，1066
ALTER TEXT SEARCH CONFIGURATION，1067
ALTER TEXT SEARCH DICTIONARY，1069
ALTER TEXT SEARCH PARSER，1071
ALTER TEXT SEARCH TEMPLATE，1072
ALTER TRIGGER，1073
ALTER TYPE，1074
ALTER USER，1077
ALTER USER MAPPING，1078
ALTER VIEW，1079
ANALYZE，473，1081
AND（操作符），148
any，147
ANY，230，233，235
anyarray，147
anyelement，147
anyenum，147
anynonarray，147
anyrange，147
applicable role，718
application_name配置参数，417
archive_cleanup_command 恢复参数，508
archive_command配置参数，406
archive_mode配置参数，406
archive_timeout配置参数，407
area，201
ARRAY，43
  结果类型的确定，267
array_agg，229
array_append，225
array_cat，225
array_dims，225
array_fill，225
array_length，225
array_lower，225
array_ndims，225
array_nulls配置参数，430
array_prepend，225
array_to_json，220
array_to_string，225
array_upper，225
ascii，157
asin，154
AT TIME ZONE，197
atan，154
atan2，154
authentication_timeout配置参数，397
auth_delay，2376
auth_delay.milliseconds 配置参数，
auto-increment (见 serial)
autocommit
  bulk-loading data，338
  psql，
autovacuum_analyze_scale_factor 配置参数，424
autovacuum_analyze_threshold 配置参数，423
autovacuum_freeze_max_age 配置参数，424
autovacuum_max_workers配置参数，423
autovacuum_naptime配置参数，423
autovacuum_vacuum_cost_delay 配置参数，424
autovacuum_vacuum_cost_limit 配置参数，424
autovacuum_vacuum_scale_factor 配置参数，424
autovacuum_vacuum_threshold 配置参数，423
autovacuum配置参数，423
auto_explain，2377
auto_explain.log_analyze configuration parameter，
auto_explain.log_buffers configuration parameter，
auto_explain.log_format configuration parameter，
auto_explain.log_min_duration configuration parameter，

auto_explain.log_nested_statements configuration parameter，
auto_explain.log_timing configuration parameter，
auto_explain.log_verbose configuration parameter，
avg，229

B

B-树 (见索引)
backslash_quote配置参数，430
BEGIN，1083
BETWEEN，149
BETWEEN SYMMETRIC，149
bgwriter_delay配置参数，402
bgwriter_lru_maxpages配置参数，402
bgwriter_lru_multiplier配置参数，403
bigint，34，101
bigserial，103
bison，348
bit_and，229
bit_length，155
bit_or，229
BLOB (见 large object)
block_size配置参数，432
bonjour_name配置参数，396
bonjour配置参数，396
Boolean
  数据类型，116
bool_and，229
bool_or，229
box，202
box (data type)，119
broadcast，204
btree_gin，2380
btree_gist，2381
btrim，157，167
bytea，106
bytea_output配置参数，426

C

C，549，623
C++，803
CASCADE
  with DROP，75
  外键动作，56
CASE，222
  结果类型的确定，267
cast
  I/O conversion，1107
cbrt，151
ceil，151
ceiling，151
center，201
Certificate，450
char，104
character varying，104
char_length，155
check constraint，51
CHECKPOINT，1085
checkpoint_completion_target配置参数，406
checkpoint_segments配置参数，406
checkpoint_timeout配置参数，406
checkpoint_warning配置参数，406
check_function_bodies配置参数，425
chkpass，2383
chr，157
cid，145
cidr，121
circle，120，202
citext，2384
client authentication，437
client_encoding配置参数，428
client_min_messages配置参数，416
clock_timestamp，189
CLOSE，1086
cluster
  of databases (见 database cluster)
CLUSTER，1087
clusterdb，1350
cmax，57
cmin，57
COALESCE，223
COLLATE，42
collation
  in PL/pgSQL，869
  in SQL functions，777
collation for，245
column，7，49
  adding，58
  removing，59
  renaming，60
  system column，57
column data type
  changing，60
col_description，248
comment
  关于数据库对象，248
COMMENT，1089
COMMIT，1093
COMMIT PREPARED，1094
commit_delay配置参数，
commit_siblings配置参数，406
computed field，139
concat，157
concat_ws，157
configure，349
config_file配置参数，395
conninfo，555
constraint，51
  adding，59
  check，51
  exclusion，57
  foreign key，55
  name，51

NOT NULL，52
primary key，54
removing，59
unique，53
constraint exclusion，72
constraint_exclusion配置参数，413
CONTINUE
in PL/pgSQL，882
convert，157
convert_from，158
convert_to，158
COPY，8，1095
with libpq，582
corr，231
cos，154
cot，154
count，229
covar_pop，231
covar_samp，231
cpu_index_tuple_cost配置参数，411
cpu_operator_cost配置参数，411
cpu_tuple_cost配置参数，411
CREATE AGGREGATE，1104
CREATE CAST，1107
CREATE COLLATION，1111
CREATE CONVERSION，1113
CREATE DATABASE，457，1115
CREATE DOMAIN，1118
CREATE EXTENSION，1120
CREATE FOREIGN DATA WRAPPER，1122
CREATE FOREIGN TABLE，1124
CREATE FUNCTION，1126
CREATE GROUP，1133
CREATE INDEX，1134
CREATE LANGUAGE，1140
CREATE OPERATOR，1143
CREATE OPERATOR CLASS，1146
CREATE OPERATOR FAMILY，1149
CREATE ROLE，452，1150
CREATE RULE，1154
CREATE SCHEMA，1157
CREATE SEQUENCE，1159
CREATE SERVER，1162
CREATE TABLE，7，1164
CREATE TABLE AS，1179
CREATE TABLESPACE，460，1182
CREATE TEXT SEARCH CONFIGURATION，1184
CREATE TEXT SEARCH DICTIONARY，1185
CREATE TEXT SEARCH PARSER，1187
CREATE TEXT SEARCH TEMPLATE，1189
CREATE TRIGGER，1190
CREATE TYPE，1195
CREATE USER，1203
CREATE USER MAPPING，1204
CREATE VIEW，1205
createdb，3，458，1353
createlang，1356
createuser，452，1359
cstring，147
ctid，57
CTID，846
cube（扩展），2386
cume_dist，232
current_catalog，241
current_database，241
current_date，189
current_query，241
current_role，241
current_schema，241
current_schemas，241
current_setting，249
current_time，190
current_timestamp，190
current_user，241
currval，220
cursor
CLOSE，1086
DECLARE，1209
FETCH，1260
in PL/pgSQL，888
MOVE，1278
cursor_tuple_fraction配置参数，413
Cygwin
在其上安装，363

D

data type
internal organization，782
type cast，41
database，457
creating，3
database cluster，7
data_directory配置参数，394
date，108，109
constants，112
输出格式，112
(参见 formatting)
DateStyle配置参数，427
date_part，190，193
date_trunc，190，196
dblink，2389
db_user_namespace 配置参数，399
deadlock_timeout配置参数，429
DEALLOCATE，1208
debug_assertions 配置参数，436
debug_deadlocks配置参数，435
debug_pretty_print配置参数，418
debug_print_parse配置参数，418
debug_print_plan配置参数，418
debug_print_rewritten配置参数，418
decimal (见 numeric)
DECLARE，1209
decode，158，167
decode_bytea
in PL/Perl，
default value，50

changing，60
default_statistics_target配置参数，412
default_tablespace配置参数，425
default_text_search_config配置参数，428
default_transaction_deferrable配置参数，426
default_transaction_isolation配置参数，426
default_transaction_read_only配置参数，426
default_with_oids 配置参数，430
degrees，151
DELETE，14，78，1212
RETURNING，79
dense_rank，232
diameter，201
dict_int，2421
dict_xsyn，2422
Digital UNIX (见 Tru64 UNIX)
DISCARD，1215
DISTINCT，10，91
div，151
DO，1216
DROP AGGREGATE，1217
DROP CAST，1218
DROP COLLATION，1219
DROP CONVERSION，1220
DROP DATABASE，459，1221
DROP DOMAIN，1222
DROP EXTENSION，1223
DROP FOREIGN DATA WRAPPER，1224
DROP FOREIGN TABLE，1225
DROP FUNCTION，1226
DROP GROUP，1228
DROP INDEX，1229
DROP LANGUAGE，1230
DROP OPERATOR，1231
DROP OPERATOR CLASS，1233
DROP OPERATOR FAMILY，1235
DROP OWNED，1236
DROP ROLE，452，1237
DROP RULE，1238
DROP SCHEMA，1239
DROP SEQUENCE，1240
DROP SERVER，1241
DROP TABLE，8，1242
DROP TABLESPACE，1243
DROP TEXT SEARCH CONFIGURATION，1244
DROP TEXT SEARCH DICTIONARY，1245
DROP TEXT SEARCH PARSER，1246
DROP TEXT SEARCH TEMPLATE，1247
DROP TRIGGER，1248
DROP TYPE，1249
DROP USER，1250
DROP USER MAPPING，1251
DROP VIEW，1252
dropdb，459，1363
droplang，1366
dropuser，452，1369
DTD，126
DTrace，355，522
dummy_seclabel，2424
duplicate，10
dynamic loading，781
dynamic_library_path，781
dynamic_library_path配置参数，429

E

earthdistance，2425
ECPG，623
ecpg，1371
effective_cache_size配置参数，411
effective_io_concurrency配置参数，403
elog，1577
in PL/Perl，
在 PL/Python 中，948
在 PL/Tcl 中，920
embedded SQL
in C，623
enabled role，733
enable_bitmapscan 配置参数，409
enable_hashagg 配置参数，410
enable_hashjoin 配置参数，410
enable_indexonlyscan 配置参数，410
enable_indexscan 配置参数，410
enable_material 配置参数，410
enable_mergejoin 配置参数，410
enable_nestloop 配置参数，410
enable_seqscan配置参数，410
enable_sort配置参数，410
enable_tidscan配置参数，410
encode，158，167
encode_array_constructor
in PL/Perl，
encode_array_literal
in PL/Perl，
encode_bytea
in PL/Perl，
encode_typed_literal
in PL/Perl，
encryption
for specific columns，2459
END，1253
enumerated types，117
enum_first，199
enum_last，199
enum_range，199
ereport，1577
error codes
libpq，568
list of，1670
escape_string_warning配置参数，431
event_source配置参数，416
every，230
EXCEPT，91
exceptions
in PL/pgSQL，886
exclusion constraint，57
EXECUTE，1254

EXISTS，233
EXIT
  in PL/pgSQL，881
exit_on_error配置参数，432
exp，151
EXPLAIN，325，1255
external_pid_file配置参数，395
extract，190，193
extra_float_digits配置参数，427

F

false，116
family，204
fast path，581
fdw_handler，147
FETCH，1260
field
  computed，139
file_fdw，2427
first_value，232
flex，348
float4 (见 real)
float8 (见双精度)
floor，152
foreign data，74
foreign data wrapper
  handler for，1592
foreign key，15，54
foreign table，74
format，158
  use in PL/pgSQL，874
format_type，245
Free Space Map，1637
FreeBSD
  IPC 配置，
  共享库，
  启动脚本，376
from_collapse_limit配置参数，413
FSM (见 Free Space Map)
fsync配置参数，404
full_page_writes配置参数，405
function，148
  default values for arguments，772
  internal，780
  named argument，766
  output parameter，770
  RETURNS TABLE，775
  user-defined，764
    in C，780
    in SQL，765
  variadic，771
  with SETOF，773
fuzzystrmatch，2429

G

generate_series，238
generate_subscripts，239
GEQO (见遗传查询优化)
geqo_effort配置参数，412
geqo_generations配置参数，412
geqo_pool_size配置参数，412
geqo_seed配置参数，412
geqo_selection_bias配置参数，412
geqo_threshold配置参数，412
geqo配置参数，412
get_bit，168
get_byte，168
get_current_ts_config，207
GIN (见索引)
gin_fuzzy_search_limit配置参数，429
GiST (见索引)
GRANT，60，1264
GREATEST，224
  结果类型的确定，267
GROUP BY，13，87
GSSAPI，445
GUID，125

H

hash (见索引)
has_any_column_privilege，243
has_column_privilege，243
has_database_privilege，243
has_foreign_data_wrapper_privilege，243
has_function_privilege，243
has_language_privilege，243
has_schema_privilege，243
has_sequence_privilege，243
has_server_privilege，243
has_tablespace_privilege，243
has_table_privilege，243
has_type_privilege，243
HAVING，13，89
hba_file配置参数，395
height，201
hierarchical database，7
host，204
hostmask，205
hot_standby_feedback配置参数，409
hot_standby配置参数，408
HP-UX
  IPC 配置，
  共享库，
  在其上安装，364
hstore，2432

I

ident，447
ident_file配置参数，395
IFNULL，223
ignore_system_indexes配置参数，434
IMMUTABLE，778
IN，233，235
include

配置文件中的，393
include_if_exists
配置文件中的，393
index
GIN，1627
inet_client_addr，241
inet_client_port，241
inet_server_addr，241
inet_server_port，241
inet（数据类型），121
inheritance，20，65
initcap，158
initdb，374，1448
Initialization Fork，1637
INSERT，8，77，1271
RETURNING，79
instr，909
int2 (见 smallint)
int4 (见整数)
int8 (见 bigint)
intagg，2438
intarray，2440
integer，34
integer_datetimes配置参数，432
internal，147
INTERSECT，91
interval，114
输出格式，115
(参见 formatting)
IntervalStyle配置参数，427
IRIX
共享库，
在其上安装，364
IS DISTINCT FROM，149，235
IS DOCUMENT，214
IS FALSE，150
IS NOT DISTINCT FROM，149，235
IS NOT FALSE，150
IS NOT NULL，149
IS NOT TRUE，150
IS NOT UNKNOWN，150
IS NULL，149，431
IS TRUE，150
IS UNKNOWN，150
isclosed，201
isempty，228
isfinite，190
isn，2443
ISNULL，149
isopen，201
is_array_ref
in PL/Perl，

J

join
controlling the order，336
join_collapse_limit配置参数，413
JSON，145
函数和操作符，219
justify_days，191
justify_hours，191
justify_interval，191

K

Kerberos，446
key word
list of，1683
krb_caseins_users配置参数，398
krb_server_keyfile配置参数，398
krb_srvname配置参数，

L

lag，232
language_handler，147
large object，614
lastval，220
last_value，232
lc_collate 配置参数，432
lc_ctype 配置参数，432
lc_messages配置参数，428
lc_monetary配置参数，428
lc_numeric配置参数，428
lc_time配置参数，428
LDAP，352，448
LDAP 连接参数查找，598
ldconfig，359
lead，232
LEAST，224
结果类型的确定，267
left，158
length，159，168，201，207
length(tsvector)，
lex，348
libedit，347
libperl，347
libpq，549
单行模式，579
libpq-fe.h，549，559
libpq-int.h，559
libpython，348
library finalization function，781
library initialization function，781
LIKE，170
与区域设置，463
LIMIT，93
Linux
IPC 配置，381
共享库，
启动脚本，376
LISTEN，1274
listen_addresses配置参数，395
ln，152
lo，2447
LOAD，1275
localtime，191

localtimestamp，191
local_preload_libraries配置参数，428
LOCK，318，1276
log，152
logging_collector配置参数，414
log_autovacuum_min_duration 配置参数，423
log_btree_build_stats配置参数，435
log_checkpoints配置参数，418
log_connections 配置参数，418
log_destination配置参数，414
log_directory配置参数，415
log_disconnections配置参数，418
log_duration配置参数，418
log_error_verbosity配置参数，419
log_executor_stats配置参数，423
log_filename配置参数，415
log_file_mode配置参数，415
log_hostname配置参数，419
log_line_prefix配置参数，419
log_lock_waits配置参数，420
log_min_duration_statement配置参数，417
log_min_error_statement配置参数，
log_min_messages配置参数，
log_parser_stats配置参数，423
log_planner_stats配置参数，423
log_rotation_age配置参数，415
log_rotation_size配置参数，415
log_statement_stats配置参数，423
log_statement配置参数，420
log_temp_files配置参数，420
log_timezone配置参数，420
log_truncate_on_rotation配置参数，415
looks_like_number
  in PL/Perl，
loop
  in PL/pgSQL，881
lower，155，228
  与区域设置，463
lower_inc，228
lower_inf，228
lo_close，617
lo_compat_privileges配置参数，431
lo_creat，614，617
lo_create，615，617
lo_export，615，617
lo_import，615，617
lo_import_with_oid，615
lo_lseek，616
lo_open，615
lo_read，616
lo_tell，616
lo_truncate，616
lo_unlink，617，617
lo_write，616
lpad，159
lseg，119，202
ltree，2448
ltrim，159

M

Mac OS X
  IPC 配置，
macaddr（数据类型），122
MAC地址 (见 macaddr)
magic block，781
maintenance_work_mem配置参数，400
make，347
MANPATH，359
masklen，205
max，230
max_connections配置参数，395
max_files_per_process配置参数，401
max_function_args配置参数，432
max_identifier_length配置参数，433
max_index_keys配置参数，433
max_locks_per_transaction配置参数，429
max_pred_locks_per_transaction配置参数，
430
max_prepared_transactions配置参数，400
max_stack_depth配置参数，400
max_standby_archive_delay配置参数，408
max_standby_streaming_delay配置参数，409
max_wal_senders配置参数，407
md5，159，168
MD5，445
min，230
MinGW
  在其上安装，365
mod，152
MOVE，1278
MVCC，314

N

name
  qualified，62
  unqualified，63
NaN (见非数字)
NetBSD
  IPC 配置，
  共享库，
  启动脚本，376
netmask，205
network，205
  数据类型，120
nextval，220
NFS (见网络文件系统)
non-durable，340
NOT IN，233，235
notation
  functions，47
notice processing
  in libpq，589
NOTIFY，1280
  in libpq，582
NOTNULL，149
NOT（操作符），148

now，192
npoints，201
nth_value，232
ntile，232
null value
  with check constraints，52
  default value，50
  in libpq，571
  with unique constraints，54
NULLIF，223
numeric，34
numnode，207，
NVL，223

O

object-oriented database，7
obj_description，248
octet_length，155，166
OFFSET，93
OID
  libpq 中的，573
  列，57
oid，145
oid2name，2519
ONLY，81
opaque，147
OpenBSD
  IPC 配置，
  共享库，
  启动脚本，376
OpenSSL，352
  (参见 SSL)
operator，148
Oracle
  porting from PL/SQL to PL/pgSQL，906
ORDER BY，9，92
  与区域设置，463
OR（操作符），148
OS X
  共享库，
OVER clause，40
overlay，156，166
overloading
  functions，777
owner，60

P

pageinspect，2454
palloc，790
PAM，352，450
partitioning，68
password
  authentication，445
passwordcheck，2456
password_encryption配置参数，398
path，202
PATH，359
path (data type)，120
patterns
  in psql and pg_dump，1428
pause_at_recovery_target 恢复参数，509
pclose，201
peer，448
percent_rank，232
performance，325
perl，348
Perl，923
permission (见 privilege)
pfree，790
PGAPPNAME，597
pgbench，2523
PGcancel，580
PGCLIENTENCODING，597
PGconn，549
PGCONNECT_TIMEOUT，597
pgcrypto，2459
PGDATA，374
PGDATABASE，597
PGDATESTYLE，597
PGEventProc，592
PGGEQO，597
PGGSSLIB，597
PGHOST，596
PGHOSTADDR，596
PGKRBSRVNAME，597
PGLOCALEDIR，597
PGOPTIONS，597
PGPASSFILE，597
PGPASSWORD，597
PGPORT，597
PGREQUIREPEER，597
PGREQUIRESSL，597
PGresult，566
pgrowlocks，2472
PGSERVICE，597
PGSERVICEFILE，597
PGSSLCERT，597
PGSSLCOMPRESSION，597
PGSSLCRL，597
PGSSLKEY，597
PGSSLMODE，597
PGSSLROOTCERT，597
pgstattuple，2477
PGSYSCONFDIR，597
PGTZ，597
PGUSER，597
pgxs，828
pg_advisory_lock，257
pg_advisory_lock_shared，257
pg_advisory_unlock，257
pg_advisory_unlock_all，257
pg_advisory_unlock_shared，257
pg_advisory_xact_lock，257
pg_advisory_xact_lock_shared，257
pg_aggregate，1478

pg_am，1479
pg_amop，1481
pg_amproc，1482
pg_archivecleanup，2533
pg_attrdef，1482
pg_attribute，1482
pg_authid，1485
pg_auth_members，1486
pg_available_extensions，1526
pg_available_extension_versions，1527
pg_backend_pid，241
pg_basebackup，1373
pg_buffercache，2457
pg_cancel_backend，250
pg_cast，1486
pg_class，1488
pg_client_encoding，159
pg_collation，1493
pg_collation_is_visible，245
pg_column_size，253
pg_config，1378
  用于 libpq，603
  with user-defined C functions，789
pg_conf_load_time，241
pg_constraint，1490
pg_controldata，1451
pg_conversion，1493
pg_conversion_is_visible，245
pg_create_restore_point，250
pg_ctl，374，376，1452
pg_current_xlog_insert_location，250
pg_current_xlog_location，250
pg_cursors，1527
pg_database，459，1494
pg_database_size，253
pg_db_role_setting，1495
pg_default_acl，1496
pg_depend，1496
pg_describe_object，245
pg_description，1497
pg_dump，1381
pg_dumpall，1392
  在升级期间使用，386
pg_enum，1498
pg_export_snapshot，253
pg_extension，1498
pg_extension_config_dump，826
pg_file_rename，2374
pg_file_unlink，2374
pg_file_write，2374
pg_foreign_data_wrapper，1499
pg_foreign_server，1500
pg_foreign_table，1500
pg_freespacemap，2470
pg_function_is_visible，245
pg_get_constraintdef，245
pg_get_expr，245
pg_get_functiondef，245
pg_get_function_arguments，245
pg_get_function_identity_arguments，245
pg_get_function_result，245
pg_get_indexdef，245
pg_get_keywords，245
pg_get_ruledef，245
pg_get_serial_sequence，245
pg_get_triggerdef，245
pg_get_userbyid，245
pg_get_viewdef，245
pg_group，1528
pg_has_role，243
pg_hba.conf，437
pg_ident.conf，443
pg_index，1500
pg_indexes，1528
pg_indexes_size，253
pg_inherits，1502
pg_is_in_recovery，252
pg_is_other_temp_schema，241
pg_is_xlog_replay_paused，252
pg_language，1503
pg_largeobject，1504
pg_largeobject_metadata，1504
pg_last_xact_replay_timestamp，252
pg_last_xlog_receive_location，252
pg_last_xlog_replay_location，252
pg_listening_channels，241
pg_locks，1529
pg_logdir_ls，2374
pg_ls_dir，255
pg_my_temp_schema，241
pg_namespace，1505
pg_notify，1281
pg_opclass，1505
pg_opclass_is_visible，245
pg_operator，1506
pg_operator_is_visible，245
pg_opfamily，1506
pg_opfamily_is_visible，245
pg_options_to_table，245
pg_pltemplate，1507
pg_postmaster_start_time，241
pg_prepared_statements，1531
pg_prepared_xacts，1532
pg_proc，1508
pg_range，1511
pg_read_binary_file，255
pg_read_file，255
pg_receivexlog，1397
pg_relation_filenode，255
pg_relation_filepath，255
pg_relation_size，253
pg_reload_conf，250
pg_resetxlog，1457
pg_restore，1400
pg_rewrite，1511
pg_roles，1532

pg_rotate_logfile，250
pg_rules，1533
pg_seclabel，1512
pg_seclabels，1534
pg_service.conf，598
pg_settings，1534
pg_shadow，1536
pg_shdepend，1513
pg_shdescription，1514
pg_shseclabel，1514
pg_size_pretty，253
pg_sleep，199
pg_standby，2535
pg_start_backup，250
pg_statio_all_indexes，513
pg_statio_all_sequences，514
pg_statio_all_tables，513
pg_statio_sys_indexes，513
pg_statio_sys_sequences，514
pg_statio_sys_tables，513
pg_statio_user_indexes，513
pg_statio_user_sequences，514
pg_statio_user_tables，513
pg_statistic，335，1515
pg_stats，335，1537
pg_stat_activity，513
pg_stat_all_indexes，513
pg_stat_all_tables，513
pg_stat_bgwriter，513
pg_stat_database，513
pg_stat_database_conflicts，514
pg_stat_file，255
pg_stat_replication，514
pg_stat_statements，2473
pg_stat_sys_indexes，513
pg_stat_sys_tables，513
pg_stat_user_functions，514
pg_stat_user_indexes，513
pg_stat_user_tables，513
pg_stat_xact_all_tables，513
pg_stat_xact_sys_tables，513
pg_stat_xact_user_functions，514
pg_stat_xact_user_tables，513
pg_stop_backup，250
pg_switch_xlog，250
pg_tables，1538
pg_tablespace，1516
pg_tablespace_databases，245
pg_tablespace_location，245
pg_tablespace_size，253
pg_table_is_visible，245
pg_table_size，253
pg_terminate_backend，250
pg_test_fsync，2538
pg_test_timing，2539
pg_timezone_abbrevs，1539
pg_timezone_names，1539
pg_total_relation_size，253
pg_trgm，2479
pg_trigger，1517
pg_try_advisory_lock，257
pg_try_advisory_lock_shared，257
pg_try_advisory_xact_lock，257
pg_try_advisory_xact_lock_shared，257
pg_ts_config，1518
pg_ts_config_is_visible，245
pg_ts_config_map，1519
pg_ts_dict，1519
pg_ts_dict_is_visible，245
pg_ts_parser，1520
pg_ts_parser_is_visible，245
pg_ts_template，1520
pg_ts_template_is_visible，245
pg_type，1521
pg_typeof，245
pg_type_is_visible，245
pg_upgrade，2542
pg_user，1540
pg_user_mapping，1525
pg_user_mappings，1540
pg_views，1541
pg_xlogfile_name，250
pg_xlogfile_name_offset，250
pg_xlog_location_diff，250
pg_xlog_replay_pause，252
pg_xlog_replay_resume，252
pi，152
PIC，790
PID
  确定服务器进程的 PID
    在 libpq 中，562
PITR，479
PITR 备库，491
PL/Perl，923
PL/PerlU，932
PL/pgSQL，863
PL/Python，936
PL/SQL (Oracle)
  porting to PL/pgSQL，906
PL/Tcl，916
plainto_tsquery，207，286
plperl.on_init configuration parameter，
plperl.on_plperlu_init configuration parameter，
plperl.on_plperl_init configuration parameter，
plperl.use_strict configuration parameter，
plpgsql.variable_conflict配置参数，902
point，119，202
polygon，120，203
popen，201
port配置参数，395
position，156，167
POSTGRES，xxxii
postgres，3，375，457，1459

postgres 用户，374
Postgres95，xxxii
postgresql.conf，393
postmaster，1466
post_auth_delay配置参数，434
power，152
PQbackendPID，562
PQbinaryTuples，571
  with COPY，583
PQcancel，580
PQclear，569
PQclientEncoding，586
PQcmdStatus，572
PQcmdTuples，573
PQconndefaults，552
PQconnectdb，550
PQconnectdbParams，549
PQconnectionNeedsPassword，562
PQconnectionUsedPassword，562
PQconnectPoll，551
PQconnectStart，551
PQconnectStartParams，551
PQconninfoFree，587
PQconninfoParse，553
PQconsumeInput，578
PQcopyResult，588
PQdb，559
PQdescribePortal，566
PQdescribePrepared，565
PQencryptPassword，587
PQendcopy，585
PQerrorMessage，562
PQescapeBytea，575
PQescapeByteaConn，575
PQescapeIdentifier，574
PQescapeLiteral，573
PQescapeString，574
PQescapeStringConn，574
PQexec，563
PQexecParams，563
PQexecPrepared，565
PQfformat，570
  with COPY，583
PQfinish，553
PQfireResultCreateEvents，588
PQflush，579
PQfmod，570
PQfn，581
PQfname，569
PQfnumber，569
PQfreeCancel，580
PQfreemem，587
PQfsize，571
PQftable，570
PQftablecol，570
PQftype，570
PQgetCancel，580
PQgetCopyData，584
PQgetisnull，571
PQgetlength，571
PQgetline，584
PQgetlineAsync，585
PQgetResult，577
PQgetssl，562
PQgetvalue，571
PQhost，560
PQinitOpenSSL，602
PQinitSSL，602
PQinstanceData，593
PQisBusy，578
PQisnonblocking，579
PQisthreadsafe，602
PQlibVersion，589
  (参见 PQserverVersion)
PQmakeEmptyPGresult，588
PQnfields，569
  用于 COPY，583
PQnotifies，582
PQnparams，572
PQntuples，569
PQoidStatus，573
PQoidValue，573
PQoptions，560
PQparameterStatus，561
PQparamtype，572
PQpass，560
PQping，554
PQpingParams，554
PQport，560
PQprepare，564
PQprint，572
PQprotocolVersion，561
PQputCopyData，583
PQputCopyEnd，583
PQputline，585
PQputnbytes，585
PQregisterEventProc，593
PQrequestCancel，580
PQreset，553
PQresetPoll，554
PQresetStart，554
PQresStatus，567
PQresultAlloc，589
PQresultErrorField，567
PQresultErrorMessage，567
PQresultInstanceData，593
PQresultSetInstanceData，593
PQresultStatus，566
PQsendDescribePortal，577
PQsendDescribePrepared，577
PQsendPrepare，577
PQsendQuery，576
PQsendQueryParams，576
PQsendQueryPrepared，577
PQserverVersion，561
PQsetClientEncoding，586

PQsetdb，550
PQsetdbLogin，550
PQsetErrorVerbosity，586
PQsetInstanceData，593
PQsetnonblocking，579
PQsetNoticeProcessor，589
PQsetNoticeReceiver，589
PQsetResultAttrs，588
PQsetSingleRowMode，579
PQsetvalue，589
PQsocket，562
PQstatus，560
PQtrace，587
PQtransactionStatus，560
PQtty，560
PQunescapeBytea，575
PQuntrace，587
PQuser，559
PREPARE，1282
PREPARE TRANSACTION，1284
prepared statements
  creating，1282
  executing，1254
  removing，1208
pre_auth_delay配置参数，434
primary key，54
primary_conninfo 恢复参数，510
privilege，60
  querying，242
  for schemas，64
protocol
  frontend-backend，1542
ps
  监控活动，511
psql，5，1408
Python，936

Q

qualified name，62
query，9
query plan，325
query tree，839
querytree，207，
quote_all_identifiers配置参数，431
quote_ident，159
  in PL/Perl，
  use in PL/pgSQL，874
quote_literal，160
  in PL/Perl，
  use in PL/pgSQL，874
quote_nullable，160
  in PL/Perl，
  use in PL/pgSQL，874

R

radians，153
radius，201
RADIUS，449
RAISE，893
random，154
random_page_cost配置参数，411
range type，140
  exclude，144
  indexes on，144
rank，232
readline，347
real，102
REASSIGN OWNED，1286
record，147
recovery.conf，508
recovery_end_command 恢复参数，508
recovery_target_inclusive 恢复参数，509
recovery_target_name 恢复参数，509
recovery_target_time 恢复参数，509
recovery_target_timeline 恢复参数，509
recovery_target_xid 恢复参数，509
rectangle，119
referential integrity，15，55
regclass，145
regconfig，145
regdictionary，145
regexp_matches，160，171
regexp_replace，160，171
regexp_split_to_array，161，171
regexp_split_to_table，161，171
regoper，145
regoperator，145
regproc，145
regprocedure，145
regression tests，538
regr_avgx，231
regr_avgy，231
regr_count，231
regr_intercept，231
regr_r2，231
regr_slope，231
regr_sxx，231
regr_sxy，231
regr_syy，231
regtype，145
REINDEX，1287
reindexdb，1441
relation，7
relational database，7
RELEASE SAVEPOINT，1289
repeat，161
replace，161
replication_timeout配置参数，
reporting errors
  in PL/pgSQL，893
RESET，1290
restart_after_crash配置参数，432
restore_command 恢复参数，508
RESTRICT
  with DROP，75

外键动作，56
RETURN NEXT
in PL/pgSQL，877
RETURN QUERY
in PL/pgSQL，877
RETURNING，79
RETURNING INTO
in PL/pgSQL，872
reverse，161
REVOKE，60，1291
right，161
role
applicable，718
enabled，733
ROLLBACK，1295
rollback
psql，
ROLLBACK PREPARED，1296
ROLLBACK TO SAVEPOINT，1297
round，153
row，7，49
ROW，44
row_number，232
row_to_json，220
rpad，161
rtrim，162
rule，839
and views，840
for DELETE，847
for INSERT，847
for SELECT，841
for UPDATE，847

S

SAVEPOINT，1299
schema，61
creating，62
current，63
public，62
removing，62
当前，241
SCO
安装，365
SCO OpenServer
IPC 配置，382
search path，63
当前，241
search_path，63
search_path配置参数，424
SECURITY LABEL，1301
seg，2482
segment_size配置参数，433
SELECT，9，80，1304
选择列表，90
SELECT INTO，1322
in PL/pgSQL，872
sepgsql，2485
sepgsql.debug_audit configuration parameter，
sepgsql.permissive configuration parameter，
sequence
and serial type，103
seq_page_cost配置参数，411
serial，103
serial2，103
serial4，103
serial8，103
server_encoding配置参数，433
server_version_num配置参数，433
server_version配置参数，433
session_replication_role配置参数，426
session_user，241
SET，249，1324
SET CONSTRAINTS，1327
SET ROLE，1328
SET SESSION AUTHORIZATION，1330
SET TRANSACTION，1332
SET XML OPTION，427
setseed，154
setval，220
setweight，207，
set_bit，168
set_byte，168
set_config，249
set_masklen，205
shared_buffers配置参数，399
shared_preload_libraries，802
shared_preload_libraries配置参数，401
SHMMAX，379
shobj_description，248
SHOW，249，1334
SIGHUP，393，441，444
SIGINT，385
sign，153
SIGQUIT，385
SIGTERM，385
SIMILAR TO，171
sin，154
sliced bread (见 TOAST)
smallint，101
smallserial，103
Solaris
IPC 配置，382
共享库，
启动脚本，376
在其上安装，367
SOME，230，233，235
SP-GiST (见索引)
SPI，950
示例，2493
SPI_connect，951
SPI_copytuple，994
spi_cursor_close
in PL/Perl，

SPI_cursor_close，980
SPI_cursor_fetch，976
SPI_cursor_find，975
SPI_cursor_move，977
SPI_cursor_open，971
SPI_cursor_open_with_args，972
SPI_cursor_open_with_paramlist，974
SPI_exec，958
SPI_execp，970
SPI_execute，955
SPI_execute_plan，968
SPI_execute_plan_with_paramlist，969
SPI_execute_with_args，959
spi_exec_prepared
  in PL/Perl，
spi_exec_query
  in PL/Perl，
spi_fetchrow
  in PL/Perl，
SPI_finish，952
SPI_fname，983
SPI_fnumber，984
spi_freeplan
  in PL/Perl，
SPI_freeplan，999
SPI_freetuple，997
SPI_freetuptable，998
SPI_getargcount，965
SPI_getargtypeid，966
SPI_getbinval，986
SPI_getnspname，990
SPI_getrelname，989
SPI_gettype，987
SPI_gettypeid，988
SPI_getvalue，985
SPI_is_cursor_plan，967
SPI_keepplan，981
spi_lastoid，
SPI_modifytuple，996
SPI_palloc，991
SPI_pfree，993
SPI_pop，954
spi_prepare
  in PL/Perl，
SPI_prepare，961
SPI_prepare_cursor，963
SPI_prepare_params，964
SPI_push，953
spi_query
  in PL/Perl，
spi_query_prepared
  in PL/Perl，
SPI_repalloc，992
SPI_returntuple，995
SPI_saveplan，982
SPI_scroll_cursor_fetch，978
SPI_scroll_cursor_move，979
split_part，162
SQL/CLI，1703
SQL/Foundation，1703
SQL/Framework，1703
SQL/JRT，1703
SQL/MED，1703
SQL/OLB，1703
SQL/PSM，1703
SQL/Schemata，1703
SQL/XML，1703
sql_inheritance配置参数，431
sqrt，153
ssh，391
SSI，314
SSL，389，599
  在 libpq 中，558
  在libpq中，562
sslinfo，2495
ssl_ca_file配置参数，397
ssl_cert_file配置参数，398
ssl_ciphers配置参数，
ssl_crl_file配置参数，398
ssl_key_file配置参数，398
ssl_renegotiation_limit配置参数，398
ssl配置参数，397
SSPI，445
STABLE，778
standard_conforming_strings配置参数，431
standby_mode 恢复参数，509
START TRANSACTION，1336
statement_timeout配置参数，426
statement_timestamp，192
statistics
  of the planner，335
stats_temp_directory配置参数，422
stddev，231
stddev_pop，231
stddev_samp，231
STONITH，491
storage parameters，1172
string (见字符串)
string_agg，230
string_to_array，225
strip，207，
strpos，162
subquery，12
substr，162
substring，156，167，171，171
sum，230
superuser，5
superuser_reserved_connections配置参数，396
suppress_redundant_updates_trigger，257
synchronize_seqscans配置参数，431
synchronous_commit配置参数，404
synchronous_standby_names配置参数，408
syntax
  SQL，29
syslog_facility配置参数，416

syslog_ident配置参数，416
system catalog
  schema，64

T

table，7，49
  creating，49
  inheritance，65
  modifying，58
  partitioning，68
  removing，50
  renaming，60
TABLE command，1304
tablefunc，2497
tableoid，57
tan，154
Tcl，916
tcn，2506
tcp_keepalives_count配置参数，397
tcp_keepalives_idle配置参数，397
tcp_keepalives_interval配置参数，397
template0，458
template1，457，458
temp_buffers配置参数，399
temp_file_limit配置参数，400
temp_tablespaces配置参数，425
test，538
test_parser，2507
text，104，205
tid，145
time，108，110
  constants，112
  输出格式，112
    (参见 formatting)
time zone，113
timeofday，192
timestamp，108，111
timestamptz，108
timezone_abbreviations配置参数，427
TimeZone配置参数，427
TOAST，1635
  per-column storage settings，
  versus large objects，614
  与用户定义类型，808
to_ascii，162
to_char，182
  与区域设置，463
to_date，183
to_hex，162
to_number，183
to_timestamp，183
to_tsquery，208，285
to_tsvector，208，284
trace_locks配置参数，434
trace_lock_oidmin配置参数，435
trace_lock_table配置参数，435
trace_lwlocks配置参数，435
trace_notify配置参数，434
trace_recovery_messages 配置参数，434
trace_sort配置参数，434
trace_userlocks配置参数，435
track_activities配置参数，422
track_activity_query_size配置参数，422
track_counts配置参数，422
track_functions配置参数，422
track_io_timing配置参数，422
transaction，16
transaction_timestamp，192
transform_null_equals配置参数，431
translate，162
trigger，147
  in PL/pgSQL，894
triggered_change_notification，2506
trigger_file 恢复参数，510
trim，156，167
Tru64 UNIX
  共享库，
true，116
trunc，153，206
TRUNCATE，1337
trusted
  PL/Perl，932
tsearch2，2509
tsquery（数据类型），124
tsvector 连接，
tsvector_update_trigger，209
tsvector_update_trigger_column，209
tsvector（数据类型），123
ts_debug，209，304
ts_headline，208，288
ts_lexize，209，308
ts_parse，209，307
ts_rank，208，
ts_rank_cd，208，
ts_rewrite，208，291
ts_stat，210，293
ts_token_type，210，307
txid_current，248
txid_current_snapshot，248
txid_snapshot_xip，248
txid_snapshot_xmax，248
txid_snapshot_xmin，248
txid_visible_in_snapshot，248
type cast，41

U

UESCAPE，30，32
unaccent，2511，2512
Unicode 转义
  在字符串常量中，32
  在标识符中，30
UNION，91
  结果类型的确定，267
unique constraint，53
Unix 域套接字，556
UnixWare

IPC 配置，383
共享库，
安装，365
unix_socket_directory配置参数，
unix_socket_group配置参数，396
unix_socket_permissions配置参数，396
UNLISTEN，1339
unnest，225
unqualified name，63
UPDATE，13，78，1340
RETURNING，79
update_process_title 配置参数，422
upper，156，228
与区域设置，463
upper_inc，228
upper_inf，228
URI，555
user，241
current，241
user mapping，74
User name maps，443
UUID，125，352
uuid-ossp，2513

V

VACUUM，1344
vacuumdb，1444
vacuumlo，2531
vacuum_cost_delay 配置参数，401
vacuum_cost_limit配置参数，402
vacuum_cost_page_dirty配置参数，402
vacuum_cost_page_hit配置参数，402
vacuum_cost_page_miss配置参数，402
vacuum_defer_cleanup_age 配置参数，408
vacuum_freeze_min_age配置参数，426
vacuum_freeze_table_age配置参数，426
VALUES，93，1346
结果类型的确定，267
varchar，104
variadic function，771
variance，231
var_pop，231
var_samp，231
version，5，241
view，15
implementation through rules，840
updating，850
Visibility Map，1637
VM (见 Visibility Map)
void，147
VOLATILE，778
volatility
functions，778
VPATH，349

W

WAL，532
wal_block_size配置参数，433
wal_buffers配置参数，405
wal_debug配置参数，435
wal_keep_segments 配置参数，407
wal_level配置参数，403
wal_receiver_status_interval配置参数，
wal_segment_size配置参数，433
wal_sync_method配置参数，404
wal_writer_delay配置参数，405
WHERE，86
WHILE
in PL/pgSQL，883
width，201
width_bucket，154
window function，17
WITH
in SELECT，1304
在 SELECT 中，94
work_mem配置参数，400

X

xid，145
xmax，57
xmin，57
XML，126
XML export，216
XML 选项，126，427
xml2，2514
xmlagg，213，230
xmlbinary配置参数，427
xmlcomment，210
xmlconcat，211
xmlelement，211
XMLEXISTS，214
xmlforest，212
xmloption配置参数，427
xmlparse，126
xmlpi，213
xmlroot，213
xmlserialize，126
xml_is_well_formed，215
xml_is_well_formed_content，215
xml_is_well_formed_document，215
XPath，215
xpath_exists，216

Y

yacc，348

Z

zero_damaged_pages配置参数，435
zlib，347，354