Vastbase G100

用户手册

(Version 2.2)

北京海量数据技术股份有限公司

目录

1. 关于本文档 1
2. 概述 2
   2.1. 数据库逻辑结构图 2
   2.2. 数据查询请求处理过程 3
   2.3. 管理事务 3
   2.4. 相关概念 5
3. 数据库使用 6
   3.1. 从这里开始 6
   3.2. 连接数据库 8
      3.2.1. 配置服务端远程连接 8
      3.2.2. 使用 vsql 连接 9
      3.2.3. 应用程序接口 10
   3.3. 关闭数据库 11
      3.3.1. 使用 sql 语句关闭数据库 11
      3.3.2. 使用 vb_ctl 工具关闭 11
   3.4. 创建和管理数据库 11
   3.5. 规划存储模型 13
   3.6. 列存下的数据压缩 16
   3.7. 创建和管理表空间 16
   3.8. 创建和管理表 18
      3.8.1. 创建表 18
      3.8.2. 向表中插入数据 19
      3.8.3. 更新表中数据 22
      3.8.4. 查看数据 23
      3.8.5. 删除表中数据 24
      3.8.6. 闪回表 24
      3.8.7. 全局临时表 25
      3.8.8. 创建和管理分区表 25
         3.8.8.1. 支持 default 分区 30
   3.9. 查看系统表 31
   3.10. 配置数据库参数 33
   3.11. 其他操作 33
      3.11.1. 创建和管理 schema 33
      3.11.2. 创建和管理索引 36
      3.11.3. 创建和管理视图 39

3.11.4. 创建和管理序列 40
3.11.5. 创建和管理约束 42
3.11.5.1. 检查约束 42
3.11.5.2. 非空约束 43
3.11.5.3. 唯一约束 43
3.11.5.4. 主键 44
3.11.5.5. 外键 44
3.11.5.6. 约束启用禁用 46
3.11.6. vb_basebackup 备份为 tar 格式 46
3.11.7. vb_basebackup 备份表空间 46
3.11.7.1. 备份 46
3.11.7.2. 恢复 47
4. 应用程序开发教程 48
4.1. 开发规范 48
4.2. JDBC 48
4.2.1. Vastbase JDBC 版本 48
4.2.2. Vastbase JDBC 与 JDK 的兼容性 48
4.2.3. Vastbase JDBC 主要类与接口 48
4.2.4. 设置 JDBC 驱动 49
4.2.4.1. 设置类路径 49
4.2.4.2. 为 JDBC 准备数据库 49
4.2.5. 使用 JDBC 连接数据库 49
4.2.5.1. 导入 49
4.2.5.2. 加载驱动 49
4.2.5.3. 连接数据库 50
4.2.5.4. 连接参数 50
4.2.6. 执行 SQL 语句并处理结果 51
4.2.6.1. 执行查询 51
4.2.6.2. 使用游标获取结果 51
4.2.6.3. 使用 ResultSet 接口 53
4.2.6.4. 执行更新 53
4.2.6.5. 创建或修改数据库对象 54
4.2.7. 调用存储过程 54
4.2.7.1. 获取结果集 54
4.2.8. 处理大数据类型 55
4.2.8.1. 二进制类型 55
4.2.8.2. 字符串类型 56

4.2.9. 使用连接池 57
4.3. ODBC 57
4.3.1. Vastbase ODBC 主要接口 57
4.3.2. 设置 ODBC 驱动 58
4.3.2.1. 安装 openGauss 或 Vastbase 58
4.3.2.2. 安装 unixODBC 58
4.3.2.3. 替换客户端驱动程序 59
4.3.2.4. 配置数据源 59
4.3.2.5. 配置环境变量 59
4.3.2.6. 测试数据源配置 60
4.3.3. 使用 ODBC 连接数据库 60
4.3.3.1. 连接数据库 60
4.3.3.2. 连接参数 62
4.3.4. 执行 SQL 语句并处理结果 63
4.3.4.1. 执行查询 63
4.3.4.2. 使用游标获取结果 64
4.3.4.3. 执行更新 65
4.3.4.4. 创建或修改数据库对象 66
4.3.5. 调用存储过程 66
4.3.6. 处理大数据类型 67
4.3.6.1. 二进制类型 67
4.3.6.2. 字符串类型 68
4.4. 基于 libpq 开发 69
4.5. 调试 69
5. 管理数据库安全 74
5.1. 客户端接入认证 74
5.1.1. 配置客户端接入认证 74
5.1.2. 配置文件参考 76
5.1.3. SSL 证书管理 88
5.1.3.1. 证书生成 88
5.1.4. 用 SSL 进行安全的 TCP/IP 连接 93
5.1.5. 查看数据库连接数 99
5.2. 管理用户及权限 102
5.2.1. 默认权限机制 102
5.2.2. 管理员 102
5.2.3. 三权分立 103
5.2.4. 用户 104

5.2.5. 角色 105
5.2.6. Schema 105
5.2.7. 用户权限设置 107
5.2.8. 行级访问控制 107
5.2.9. 设置安全策略 109
5.2.9.1. 设置帐户安全策略 109
5.2.9.2. 设置帐号有效期 111
5.2.9.3. 设置密码安全策略 112
5.3. 设置数据库审计 118
5.3.1. 审计概述 118
5.3.2. 查看审计结果 123
5.3.3. 维护审计日志 125
5.3.4. 设置文件权限安全策略 127
5.4. 强制访问控制 129
5.4.1. 敏感标记 129
5.4.1.1. 等级分类 129
5.4.1.2. 非等级类别 129
5.4.1.3. 敏感标记 129
5.4.2. 访问策略 130
5.4.2.1. 表强制访问控制策略 130
5.5. 安全审计 132
5.5.1. 概述 132
5.5.1.1. 可审计对象与事件类型 132
5.5.1.2. 审计响应动作 133
5.5.2. 开启安全审计 133
5.5.3. 审计策略维护 134
5.5.3.1. 资源池 134
5.5.3.2. 审计策略 134
5.5.4. 查看审计日志 135
5.6. 存储加密功能 135
6. 接口参考 136
6.1. JDBC 136
6.1.1. java.sql.Connection 136
6.1.2. java.sql.CallableStatement 137
6.1.3. java.sql.DatabaseMetaData 138
6.1.4. java.sql.Driver 140
6.1.5. java.sql.PreparedStatement 141

6.1.6. java.sql.ResultSet....142
6.1.7. java.sql.ResultSetMetaData....143
6.1.8. java.sql.Statement....143
6.1.9. javax.sql.ConnectionPoolDataSource....144
6.1.10. javax.sql.DataSource....145
6.1.11. javax.sql.PooledConnection....145
6.1.12. javax.naming.Context....145
6.1.13. javax.naming.spi.InitialContextFactory....146
6.1.14. CopyManager....146
6.2. ODBC....148
6.2.1. SQLAllocEnv....148
6.2.2. SQLAllocConnect....148
6.2.3. SQLAllocHandle....148
6.2.4. SQLAllocStmt....150
6.2.5. SQLBindCol....150
6.2.6. SQLBindParameter....151
6.2.7. SQLColAttribute....152
6.2.8. SQLConnect....154
6.2.9. SQLDisconnect....155
6.2.10. SQLExecDirect....156
6.2.11. SQLExecute....157
6.2.12. SQLFetch....158
6.2.13. SQLFreeStmt....159
6.2.14. SQLFreeConnect....159
6.2.15. SQLFreeHandle....159
6.2.16. SQLFreeEnv....160
6.2.17. SQLPrepare....160
6.2.18. SQLGetData....161
6.2.19. SQLGetDiagRec....163
6.2.20. SQLSetConnectAttr....165
6.2.21. SQLSetEnvAttr....166
6.2.22. SQLSetStmtAttr....167
6.2.23. 示例....168
6.3. libpq....175
6.3.1. 数据库连接控制函数....175
6.3.1.1. PQconnectdbParams....175
6.3.1.2. PQconnectdb....176

6.3.1.3. PQconninfoParse 176
6.3.1.4. PQconnectStart 177
6.3.1.5. PQerrorMessage 178
6.3.1.6. PQsetdbLogin 178
6.3.1.7. PQfinish 179
6.3.1.8. PQreset 180
6.3.1.9. PQstatus 181
6.3.2. 数据库执行语句函数 182
6.3.2.1. PQclear 182
6.3.2.2. PQexec 183
6.3.2.3. PQexecParams 184
6.3.2.4. PQexecParamsBatch 185
6.3.2.5. PQexecPrepared 186
6.3.2.6. PQexecPreparedBatch 187
6.3.2.7. PQfname 188
6.3.2.8. PQgetvalue 188
6.3.2.9. PQnfields 189
6.3.2.10. PQntuples 190
6.3.2.11. PQprepare 190
6.3.2.12. PQresultStatus 192
6.3.3. 异步命令处理 193
6.3.3.1. PQsendQuery 194
6.3.3.2. PQsendQueryParams 194
6.3.3.3. PQsendPrepare 195
6.3.3.4. PQsendQueryPrepared 196
6.3.3.5. PQflush 198
6.3.4. 取消正在处理的查询 198
6.3.4.1. PQgetCancel 198
6.3.4.2. PQfreeCancel 199
6.3.4.3. PQcancel 200
6.3.5. 示例 201
6.3.6. 链接字符 209
7. 导入数据 212
7.1. 通过 INSERT 语句直接写入数据 212
7.2. 使用 COPY FROM STDIN 导入数据 213
7.2.1. 关于 COPY FROM STDIN 导入数据 213
7.2.2. CopyManager 类简介 213

7.2.3. 示例 1：通过本地文件导入导出数据 214
7.2.4. 示例 2：从 MySQL 向 Vastbase 进行数据迁移 216
7.3. 使用 vsql 元命令导入数据 218
7.4. 使用 vb_restore 命令导入数据 221
7.5. 更新表中数据 226
7.5.1. 使用 DML 命令更新表 226
7.5.2. 使用合并方式更新和插入数据 227
7.6. 深层复制 230
7.6.1. 使用 CREATE TABLE 执行深层复制 230
7.6.2. 使用 CREATE TABLE LIKE 执行深层复制 230
7.6.3. 通过创建临时表并截断原始表来执行深层复制 231
7.7. 分析表 231
7.8. 对表执行 VACUUM 232
7.9. 管理并发写入操作 233
7.9.1. 事务隔离说明 233
7.9.2. 写入和读写操作 233
7.9.3. 并发写入事务的潜在死锁情况 234
7.9.4. 并发写入示例 234
7.9.4.1. 相同表的 INSERT 和 DELETE 并发 234
7.9.4.2. 相同表的并发 INSERT 235
7.9.4.3. 相同表的并发 UPDATE 235
7.9.4.4. 数据导入和查询的并发 236
8. 导出数据 238
8.1. 使用 vb_dump 和 vb_dumpall 命令导出数据 238
8.1.1. 概述 238
8.1.2. 导出单个数据库 240
8.1.2.1. 导出数据库 240
8.1.2.2. 导出模式 242
8.1.2.3. 导出表 244
8.1.3. 导出所有数据库 247
8.1.3.1. 导出所有数据库 247
8.1.3.2. 导出全局对象 249
9. 性能调优 250
9.1. 总体调优思路 250
9.2. 确定性能调优范围 252
9.2.1. 硬件瓶颈点分析 253
9.2.1.1. CPU 253

9.2.1.2. 内存 255
9.2.1.3. I/O 256
9.2.1.4. 网络 258
9.2.2. 查询最耗性能的 SQL 259
9.2.3. 分析作业是否被阻塞 260
9.3. 系统调优指南 261
9.3.1. 操作系统参数调优 261
9.3.2. 数据库系统参数调优 264
9.3.2.1. 数据库内存参数调优 264
9.3.3. 配置 LLVM 265
9.3.3.1. LLVM 适用场景与限制 265
9.3.3.2. 其他因素对 LLVM 性能的影响 266
9.3.3.3. LLVM 使用建议 266
9.3.4. 资源利用-空闲事务会话超时 267
9.3.5. 资源利用-降级容错 267
9.3.6. 资源利用-表空间阈值 267
9.4. SQL 调优指南 268
9.4.1. Query 执行流程 268
9.4.2. SQL 执行计划介绍 270
9.4.2.1. SQL 执行计划概述 270
9.4.2.2. 详解 271
9.4.3. 调优流程 274
9.4.4. 更新统计信息 274
9.4.5. 审视和修改表定义 275
9.4.5.1. 审视和修改表定义概述 275
9.4.5.2. 选择存储模型 276
9.4.5.3. 使用分区表 276
9.4.5.4. 选择数据类型 276
9.4.6. 典型 SQL 调优点 277
9.4.6.1. SQL 自诊断 277
9.4.6.2. 子查询调优 278
9.4.6.3. 统计信息调优 287
9.4.6.4. 算子级调优 289
9.4.7. 经验总结：SQL 语句改写规则 291
9.4.8. SQL 调优关键参数调整 292
9.4.9. 使用 Plan Hint 进行调优 293
9.4.9.1. Plan Hint 调优概述 293

9.4.9.2. Join 顺序的 Hint 298
9.4.9.3. Join 方式的 Hint 299
9.4.9.4. 行数的 Hint 300
9.4.9.5. Scan 方式的 Hint 301
9.4.9.6. 子链接块名的 hint 302
9.4.9.7. Hint 的错误、冲突及告警 303
10. 配置运行参数 305
10.1. 查看参数当前取值 305
10.2. 重设参数 306
11. SQL 参考 309
11.1. Vastbase SQL 309
11.2. 关键字 310
11.3. 数据类型 343
11.3.1. 数值类型 343
11.3.2. 货币类型 349
11.3.3. 布尔类型 349
11.3.4. 字符类型 350
11.3.5. 二进制类型 353
11.3.6. 日期/时间类型 354
11.3.7. 几何类型 361
11.3.8. 网络地址类型 363
11.3.9. 位串类型 365
11.3.10. 文本搜索类型 366
11.3.11. UUID 类型 368
11.3.12. JSON 类型 369
11.3.13. XML 类型 369
11.3.14. 对象标识符类型 369
11.3.15. 伪类型 371
11.3.16. 列存表支持的数据类型 373
11.4. 常量与宏 374
11.5. 函数和操作符 376
11.5.1. 逻辑操作符 376
11.5.2. 比较操作符 376
11.5.3. 字符处理函数和操作符 377
11.5.4. 二进制字符串函数和操作符 400
11.5.5. 位串函数和操作符 402
11.5.6. 模式匹配操作符 404

11.5.7. 数字操作函数和操作符 409
11.5.8. 时间和日期处理函数和操作符 420
11.5.9. 类型转换函数 438
11.5.10. 几何函数和操作符 446
11.5.11. 网络地址函数和操作符 459
11.5.12. 文本检索函数和操作符 466
11.5.13. JSON 函数 473
11.5.14. SEQUENCE 函数 474
11.5.15. 数组函数和操作符 476
11.5.16. 范围函数和操作符 482
11.5.17. 聚集函数 487
11.5.18. 窗口函数 499
11.5.19. 安全函数 510
11.5.20. 返回集合的函数 514
11.5.21. 条件表达式函数 516
11.5.22. 系统信息函数 519
11.5.23. 系统管理函数 538
11.5.23.1. 配置设置函数 538
11.5.23.2. 通用文件访问函数 539
11.5.23.3. 服务器信号函数 541
11.5.23.4. 备份恢复控制函数 542
11.5.23.5. 快照同步函数 546
11.5.23.6. 数据库对象函数 547
11.5.23.7. 咨询锁函数 552
11.5.23.8. 逻辑复制函数 556
11.5.23.9. 其它函数 558
11.5.24. 统计信息函数 562
11.5.25. 触发器函数 585
11.5.26. 终止用户会话函数 586
11.5.27. 其他系统函数 586
11.6. 表达式 647
11.6.1. 简单表达式 647
11.6.2. 条件表达式 649
11.6.3. 子查询表达式 653
11.6.4. 数组表达式 656
11.6.5. 行表达式 658
11.7. 类型转换 658

11.7.1. 概述 658
11.7.2. 操作符 660
11.7.3. 函数 662
11.7.4. 值存储 664
11.7.5. UNION，CASE 和相关构造 664
11.8. 全文检索 667
11.8.1. 介绍 667
11.8.1.1. 全文检索概述 667
11.8.1.2. 文档概念 668
11.8.1.3. 基本文本匹配 669
11.8.1.4. 分词器 670
11.8.2. 表和索引 670
11.8.2.1. 搜索表 670
11.8.2.2. 创建索引 672
11.8.2.3. 索引使用约束 673
11.8.2.4. pg_zhtrgm 674
11.8.3. 控制文本搜索 675
11.8.3.1. 解析文档 675
11.8.3.2. 解析查询 676
11.8.3.3. 排序查询结果 677
11.8.3.4. 高亮搜索结果 679
11.8.4. 附加功能 680
11.8.4.1. 处理 tsvector 680
11.8.4.2. 处理查询 681
11.8.4.3. 查询重写 682
11.8.4.4. 收集文献统计 683
11.8.5. 解析器 684
11.8.6. 词典 687
11.8.6.1. 词典概述 687
11.8.6.2. 停用词 688
11.8.6.3. Simple 词典 688
11.8.6.4. Synonym 词典 690
11.8.6.5. Thesaurus 词典 692
11.8.6.6. Ispell 词典 693
11.8.6.7. Snowball 词典 694
11.8.7. 配置示例 695
11.8.8. 测试和调试文本搜索 696

11.8.8.1. 分词器测试 696
11.8.8.2. 解析器测试 697
11.8.8.3. 词典测试 698
11.8.9. 限制约束 699
11.9. 并行查询 699
11.10. 全局分区索引 700
11.11. 系统操作 700
11.12. 事务控制 701
11.13. DDL 语法一览表 701
11.14. DML 语法一览表 705
11.15. DCL 语法一览表 706
11.16. SQL 语法 708
11.16.1. ABORT 708
11.16.2. ALTER DATABASE 709
11.16.3. ALTER DATA SOURCE 711
11.16.4. ALTER DEFAULT PRIVILEGES 713
11.16.5. ALTER DIRECTORY 716
11.16.6. ALTER FUNCTION 716
11.16.7. ALTER GROUP 720
11.16.8. ALTER INDEX 721
11.16.9. ALTER LARGE OBJECT 723
11.16.10. ALTER ROLE 724
11.16.11. ALTER ROW LEVEL SECURITY POLICY 726
11.16.12. ALTER SCHEMA 728
11.16.13. ALTER SEQUENCE 729
11.16.14. ALTER SESSION 730
11.16.15. ALTER SYNONYM 732
11.16.16. ALTER SYSTEM KILL SESSION 733
11.16.17. ALTER TABLE 734
11.16.18. ALTER TABLE PARTITION 743
11.16.19. ALTER TABLESPACE 747
11.16.20. ALTER TEXT SEARCH CONFIGURATION 749
11.16.21. ALTER TEXT SEARCH DICTIONARY 752
11.16.22. ALTER TRIGGER 754
11.16.23. ALTER TYPE 755
11.16.24. ALTER USER 757
11.16.25. ALTER VIEW 759

11.16.26. ANALYZE | ANALYSE 761
11.16.27. BEGIN 764
11.16.28. CALL 765
11.16.29. CHECKPOINT 766
11.16.30. CLOSE 767
11.16.31. CLUSTER 768
11.16.32. COMMENT 770
11.16.33. COMMIT | END 773
11.16.34. COMMIT PREPARED 774
11.16.35. COPY 775
11.16.36. CREATE DATABASE 789
11.16.37. CREATE DATA SOURCE 792
11.16.38. CREATE DIRECTORY 794
11.16.39. CREATE FUNCTION 795
11.16.40. CREATE GROUP 801
11.16.41. CREATE INDEX 802
11.16.42. CREATE ROW LEVLEL SECURITY POLICY 809
11.16.43. CREATE PROCEDURE 812
11.16.44. CREATE ROLE 816
11.16.45. CREATE SCHEMA 821
11.16.46. CREATE SEQUENCE 822
11.16.47. CREATE SYNONYM 825
11.16.48. CREATE TABLE 827
11.16.49. CREATE TABLE AS 846
11.16.50. CREATE TABLE PARTITION 848
11.16.51. CREATE TABLESPACE 862
11.16.52. CREATE TEXT SEARCH CONFIGURATION 864
11.16.53. CREATE TEXT SEARCH DICTIONARY 866
11.16.54. CREATE TRIGGER 870
11.16.55. CREATE TYPE 875
11.16.56. CREATE USER 881
11.16.57. CREATE VIEW 884
11.16.58. CURSOR 885
11.16.59. DEALLOCATE 887
11.16.60. DECLARE 887
11.16.61. DELETE 889
11.16.62. DO 891

11.16.63. DROP DATABASE....892
11.16.64. DROP DATA SOURCE....893
11.16.65. DROP DIRECTORY....894
11.16.66. DROP FUNCTION....895
11.16.67. DROP GROUP....896
11.16.68. DROP INDEX....897
11.16.69. DROP OWNED....898
11.16.70. DROP ROW LEVEL SECURITY POLICY....898
11.16.71. DROP PROCEDURE....899
11.16.72. DROP ROLE....900
11.16.73. DROP SCHEMA....901
11.16.74. DROP SEQUENCE....902
11.16.75. DROP SYNONYM....902
11.16.76. DROP TABLE....903
11.16.77. DROP TABLESPACE....904
11.16.78. DROP TEXT SEARCH CONFIGURATION....905
11.16.79. DROP TEXT SEARCH DICTIONARY....906
11.16.80. DROP TRIGGER....907
11.16.81. DROP TYPE....908
11.16.82. DROP USER....909
11.16.83. DROP VIEW....910
11.16.84. EXECUTE....911
11.16.85. EXPLAIN....912
11.16.86. EXPLAIN PLAN....916
11.16.87. FETCH....918
11.16.88. GRANT....922
11.16.89. INSERT....929
11.16.90. LOCK....933
11.16.91. MOVE....936
11.16.92. MERGE INTO....938
11.16.93. PREPARE....940
11.16.94. PREPARE TRANSACTION....941
11.16.95. REASSIGN OWNED....942
11.16.96. REINDEX....943
11.16.97. RELEASE SAVEPOINT....945
11.16.98. RESET....946
11.16.99. REVOKE....948

11.16.100. ROLLBACK....950
11.16.101. ROLLBACK PREPARED....951
11.16.102. ROLLBACK TO SAVEPOINT....951
11.16.103. SAVEPOINT....952
11.16.104. SELECT INTO....967
11.16.105. SET....968
11.16.106. SET CONSTRAINTS....970
11.16.107. SET ROLE....971
11.16.108. SET SESSION AUTHORIZATION....973
11.16.109. SET TRANSACTION....974
11.16.110. SHOW....975
11.16.111. START TRANSACTION....976
11.16.112. TRUNCATE....977
11.16.113. UPDATE....980
11.16.114. VACUUM....982
11.16.115. VALUES....985
11.17. Oracle 语法兼容....986
11.17.1. 兼容 Insert All&First 子句语法....986
11.17.2. 兼容 Merge Into 子句语法....989
11.17.3. 兼容 with function 子句语法....991
11.17.4. 兼容 sample 抽样采集语法....992
11.17.5. 兼容 Q’ 转义字符语法....994
11.17.6. 兼容虚拟列语法....997
11.17.7. 兼容 VPD 功能....998
11.17.8. 兼容 Oracle 宏变量....1000
11.17.9. 兼容 ROWNUM....1001
11.17.10. 兼容^=运算符....1001
11.17.11. 兼容序列支持 restart 语法....1002
11.17.12. 兼容 ROWID....1002
11.17.13. 兼容内置包-DBMS_UTILITY....1002
11.17.14. 兼容内置函数-MONTHS_BETWEEN 函数....1003
11.17.15. 兼容内置函数-table 函数....1004
11.17.16. 兼容 PL/pgSQL-cursor for select into....1004
11.17.17. 兼容 PL/pgSQL-PIPELINED 与 PIPE ROW....1005
11.17.18. 兼容 INSERT 支持别名....1006
11.17.19. 兼容内置包-DBMS_SQL....1006
11.17.20. 兼容序列支持 no cache、nocache、order、no order、noorder....1009

11.17.21. 兼容内置包-UTL_FILE 1011
11.17.22. 兼容内置包-DBMS_ALERT 1012
11.17.23. 数组支持 record 类型 1014
11.17.24. 视图管理 1015
11.17.25. 兼容正则表达式 1017
11.17.26. 兼容分析函数 1019
11.17.27. 兼容内置函数-数值函数 1021
11.17.28. 兼容内置函数-空值比较函数 1022
11.17.29. 兼容内置包-DBMS_OUTPUT 1023
11.17.30. 兼容 WITH AS 子语句 1024
11.17.31. 兼容创建 Package 1025
11.17.32. 兼容（+）操作符的左外连接和右外连接查询 1026
11.17.33. 兼容 PL/pgSQL-存储过程自治事务 1027
11.18. OCI/OCCI 接口 1027
11.19. 附录 1029
11.19.1. GIN 索引 1029
11.19.1.1. 介绍 1029
11.19.1.2. 扩展性 1029
11.19.1.3. 实现 1031
11.19.1.4. GIN 提示与技巧 1032
11.19.2. 扩展函数 1033
11.19.3. 扩展语法 1033
12. 用户自定义函数 1037
12.1. PL/pgsql 语言函数 1037
13. 存储过程 1038
13.1. 存储过程 1038
13.2. 数据类型 1038
13.3. 数据类型转换 1038
13.4. 数组和 record 1039
13.4.1. 数组 1039
13.4.2. record 1040
13.5. 声明语法 1042
13.5.1. 基本结构 1042
13.5.2. 匿名块 1043
13.5.3. 子程序 1044
13.6. 基本语句 1044
13.6.1. 定义变量 1044

13.6.2. 赋值语句 1045
13.6.3. 调用语句 1046
13.7. 动态语句 1047
13.7.1. 执行动态查询语句 1047
13.7.2. 执行动态非查询语句 1049
13.7.3. 动态调用存储过程 1050
13.7.4. 动态调用匿名块 1051
13.8. commit/rollback 1052
13.9. 控制语句 1053
13.9.1. 返回语句 1053
13.9.1.1. RETURN 1053
13.9.1.2. RETURN NEXT 及 RETURN QUERY 1053
13.9.2. 条件语句 1054
13.9.3. 循环语句 1056
13.9.4. 分支语句 1060
13.9.5. 空语句 1061
13.9.6. 错误捕获语句 1061
13.9.7. GOTO 语句 1063
13.10. 其他语句 1065
13.10.1. 锁操作 1065
13.10.2. 游标操作 1065
13.11. 游标 1066
13.11.1. 游标概述 1066
13.11.2. 显式游标 1066
13.11.3. 隐式游标 1069
13.11.4. 游标循环 1070
13.12. Retry 管理 1071
13.13. 调试 1071
14. 系统表和系统视图 1075
14.1. 系统表和系统视图概述 1075
14.2. 系统表 1075
14.2.1. GS_OPT_MODEL 1075
14.2.2. GS_WLM_INSTANCE_HISTORY 1077
14.2.3. GS_WLM_OPERATOR_INFO 1078
14.2.4. GS_WLM_PLAN_ENCODING_TABLE 1079
14.2.5. GS_WLM_PLAN_OPERATOR_INFO 1080
14.2.6. GS_WLM_USER_RESOURCE_HISTORY 1081

14.2.7. PG_AGGREGATE ... 1082
14.2.8. PG_AM ... 1083
14.2.9. PG_AMOP ... 1085
14.2.10. PG_AMPROC ... 1086
14.2.11. PG_APP_WORKLOADGROUP_MAPPING ... 1087
14.2.12. PG_ATTRDEF ... 1087
14.2.13. PG_ATTRIBUTE ... 1088
14.2.14. PG_AUTHID ... 1090
14.2.15. PG_AUTH_HISTORY ... 1091
14.2.16. PG_AUTH_MEMBERS ... 1091
14.2.17. PG_CAST ... 1092
14.2.18. PG_CLASS ... 1092
14.2.19. PG_COLLATION ... 1096
14.2.20. PG_CONSTRAINT ... 1097
14.2.21. PG_CONVERSION ... 1099
14.2.22. PG_DATABASE ... 1100
14.2.23. PG_DB_ROLE_SETTING ... 1101
14.2.24. PG_DEFAULT_ACL ... 1101
14.2.25. PG_DEPEND ... 1102
14.2.26. PG_DESCRIPTION ... 1103
14.2.27. PG_DIRECTORY ... 1103
14.2.28. PG_ENUM ... 1104
14.2.29. PG_EXTENSION ... 1104
14.2.30. PG_EXTENSION_DATA_SOURCE ... 1105
14.2.31. PG_FOREIGN_DATA_WRAPPER ... 1106
14.2.32. PG_FOREIGN_SERVER ... 1106
14.2.33. PG_FOREIGN_TABLE ... 1107
14.2.34. PG_INDEX ... 1107
14.2.35. PG_INHERITS ... 1109
14.2.36. PG_JOB ... 1109
14.2.37. PG_JOB_PROC ... 1111
14.2.38. PG_LANGUAGE ... 1111
14.2.39. PG_LARGEOBJECT ... 1112
14.2.40. PG_LARGEOBJECT_METADATA ... 1113
14.2.41. PG_NAMESPACE ... 1113
14.2.42. PG_OBJECT ... 1113
14.2.43. PG_OPCLASS ... 1114

14.2.44. PG_OPERATOR 1115
14.2.45. PG_OPFAMILY 1116
14.2.46. PG_PARTITION 1117
14.2.47. PG_PLTEMPLATE 1119
14.2.48. PG_PROC 1119
14.2.49. PG_RANGE 1121
14.2.50. PG_RESOURCE_POOL 1122
14.2.51. PG_REWRITE 1123
14.2.52. PG_RLSPOLICY 1124
14.2.53. PG_SECLABEL 1124
14.2.54. PG_SHDEPEND 1125
14.2.55. PG_SHDESCRIPTION 1126
14.2.56. PG_SHSECLABEL 1126
14.2.57. PG_STATISTIC 1127
14.2.58. PG_STATISTIC_EXT 1128
14.2.59. PG_TABLESPACE 1129
14.2.60. PG_TRIGGER 1130
14.2.61. PG_TS_CONFIG 1131
14.2.62. PG_TS_CONFIG_MAP 1132
14.2.63. PG_TS_DICT 1132
14.2.64. PG_TS_PARSER 1133
14.2.65. PG_TS_TEMPLATE 1133
14.2.66. PG_TYPE 1134
14.2.67. PG_USER_MAPPING 1137
14.2.68. PG_USER_STATUS 1137
14.2.69. PG_WORKLOAD_GROUP 1138
14.2.70. PLAN_TABLE_DATA 1138
14.3. 系统视图 1139
14.3.1. GS_SESSION_CPU_STATISTICS 1139
14.3.2. GS_SESSION_MEMORY_STATISTICS 1140
14.3.3. GS_SQL_COUNT 1141
14.3.4. GS_WLM_OPERATOR_HISTORY 1142
14.3.5. GS_WLM_OPERATOR_STATISTICS 1142
14.3.6. GS_WLM_PLAN_OPERATOR_HISTORY 1144
14.3.7. GS_WLM_REBUILD_USER_RESOURCE_POOL 1144
14.3.8. GS_WLM_RESOURCE_POOL 1144
14.3.9. GS_WLM_SESSION_HISTORY 1145

14.3.10. GS_WLM_SESSION_INFO_ALL....1149
14.3.11. GS_WLM_USER_INFO....1149
14.3.12. GS_WLM_SESSION_STATISTICS....1149
14.3.13. GS_STAT_DB_CU....1152
14.3.14. GS_STAT_SESSION_CU....1153
14.3.15. MPP_TABLES....1153
14.3.16. PG_AVAILABLE_EXTENSION_VERSIONS....1154
14.3.17. PG_AVAILABLE_EXTENSIONS....1154
14.3.18. PG_CURSORS....1155
14.3.19. PG_EXT_STATS....1155
14.3.20. PG_GET_INVALID_BACKENDS....1157
14.3.21. PG_GET_SENDERS_CATCHUP_TIME....1157
14.3.22. PG_GROUP....1158
14.3.23. PG_INDEXES....1158
14.3.24. PG_LOCKS....1159
14.3.25. PG_NODE_ENV....1160
14.3.26. PG_OS_THREADS....1161
14.3.27. PG_PREPARED_STATEMENTS....1161
14.3.28. PG_PREPARED_XACTS....1162
14.3.29. PG_REPLICATION_SLOTS....1162
14.3.30. PG_RLSPOLICIES....1163
14.3.31. PG_ROLES....1163
14.3.32. PG_RULES....1165
14.3.33. PG_RUNNING_XACTS....1165
14.3.34. PG_SECLABELS....1166
14.3.35. PG_SESSION_WLMSTAT....1167
14.3.36. PG_SESSION_IOSTAT....1168
14.3.37. PG_SETTINGS....1169
14.3.38. PG_SHADOW....1170
14.3.39. PG_STATS....1171
14.3.40. PG_STAT_ACTIVITY....1173
14.3.41. PG_STAT_ALL_INDEXES....1175
14.3.42. PG_STAT_ALL_TABLES....1176
14.3.43. PG_STAT_BAD_BLOCK....1177
14.3.44. PG_STAT_BGWRITER....1178
14.3.45. PG_STAT_DATABASE....1179
14.3.46. PG_STAT_DATABASE_CONFLICTS....1180

14.3.47. PG_STAT_USER_FUNCTIONS....1181
14.3.48. PG_STAT_USER_INDEXES....1181
14.3.49. PG_STAT_USER_TABLES....1182
14.3.50. PG_STAT_REPLICATION....1183
14.3.51. PG_STAT_SYS_INDEXES....1184
14.3.52. PG_STAT_SYS_TABLES....1185
14.3.53. PG_STAT_XACT_ALL_TABLES....1186
14.3.54. PG_STAT_XACT_SYS_TABLES....1187
14.3.55. PG_STAT_XACT_USER_FUNCTIONS....1187
14.3.56. PG_STAT_XACT_USER_TABLES....1188
14.3.57. PG_STATIO_ALL_INDEXES....1188
14.3.58. PG_STATIO_ALL_SEQUENCES....1189
14.3.59. PG_STATIO_ALL_TABLES....1189
14.3.60. PG_STATIO_SYS_INDEXES....1190
14.3.61. PG_STATIO_SYS_SEQUENCES....1191
14.3.62. PG_STATIO_SYS_TABLES....1191
14.3.63. PG_STATIO_USER_INDEXES....1192
14.3.64. PG_STATIO_USER_SEQUENCES....1192
14.3.65. PG_STATIO_USER_TABLES....1193
14.3.66. PG_THREAD_WAIT_STATUS....1193
14.3.67. PG_TABLES....1207
14.3.68. PG_TDE_INFO....1207
14.3.69. PG_TIMEZONE_NAMES....1208
14.3.70. PG_TOTAL_USER_RESOURCE_INFO....1208
14.3.71. PG_USER....1210
14.3.72. PG_USER_MAPPINGS....1211
14.3.73. PG_VIEWS....1212
14.3.74. PG_WLM_STATISTICS....1212
14.3.75. PLAN_TABLE....1213
14.3.76. GS_FILE_STAT....1214
14.3.77. GS_OS_RUN_INFO....1215
14.3.78. GS_REDO_STAT....1215
14.3.79. GS_SESSION_MEMORY....1216
14.3.80. GS_SESSION_MEMORY_DETAIL....1216
14.3.81. GS_SESSION_STAT....1217
14.3.82. GS_SESSION_TIME....1217
14.3.83. GS_TOTAL_MEMORY_DETAIL....1218

14.3.84. PG_TIMEZONE_ABBREVS 1219
14.3.85. PG_TOTAL_USER_RESOURCE_INFO_OID 1219
14.3.86. PG_VARIABLE_INFO 1220
14.3.87. GS_INSTANCE_TIME 1220
14.3.88. USER_ALL_TABLES 1221
14.3.89. USER_CONS_COLUMNS 1221
14.3.90. USER_CONSTRAINTS 1222
14.3.91. USER_IND_COLUMNS 1222
14.3.92. USER_INDEXES 1222
14.3.93. USER_OBJECTS 1223
14.3.94. USER_PART_KEY_COLUMNS 1223
14.3.95. USER_PART_TABLES 1224
14.3.96. USER_PROCEDURES 1225
14.3.97. USER_ROLE_PRIVS 1225
14.3.98. USER_SEQUENCES 1225
14.3.99. USER_SOURCE 1225
14.3.100. USER_SUBPART_KEY_COLUMNS 1226
14.3.101. USER_SYNONYMS 1226
14.3.102. USER_TAB_COLUMNS 1226
14.3.103. USER_TAB_PARTITIONS 1227
14.3.104. USER_TAB_SUBPARTITIONS 1227
14.3.105. USER_TABLES 1228
14.3.106. USER_TRIGGERS 1229
14.3.107. USER_TYPES 1229
14.3.108. USER_USERS 1229
14.3.109. USER_VIEWS 1230
14.3.110. DBA_ALL_TABLES 1230
14.3.111. DBA_SEGMENTS 1230
14.3.112. DBA_CONS_COLUMNS 1231
14.3.113. DBA_CONSTRAINTS 1231
14.3.114. DBA_IND_COLUMNS 1232
14.3.115. DBA_INDEXES 1232
14.3.116. DBA_OBJECTS 1233
14.3.117. DBA_PART_KEY_COLUMNS 1233
14.3.118. DBA_PART_TABLES 1234
14.3.119. DBA_ROLE_PRIVS 1235
14.3.120. DBA_ROLES 1235
14.3.121. DBA_SEQUENCES 1235
14.3.122. DBA_SUBPART_KEY_COLUMNS 1235
14.3.123. DBA_SYNONYMS 1236
14.3.124. DBA_TAB_COLUMNS 1236

14.3.125. DBA_TAB_PARTITIONS 1236
14.3.126. DBA_TAB_SUBPARTITIONS 1237
14.3.127. DBA_TABLES 1238
14.3.128. DBA_TRIGGERS 1238
14.3.129. DBA_TYPES 1238
14.3.130. DBA_USERS 1239
14.3.131. DBA_VIEWS 1239
14.3.132. ALL_ALL_TABLES 1239
14.3.133. ALL_CONS_COLUMNS 1239
14.3.134. ALL_CONSTRAINTS 1240
14.3.135. ALL_IND_COLUMNS 1240
14.3.136. ALL_INDEXES 1241
14.3.137. ALL_OBJECTS 1241
14.3.138. ALL_PART_KEY_COLUMNS 1242
14.3.139. ALL_PART_TABLES 1242
14.3.140. ALL_SEQUENCES 1243
14.3.141. ALL_SUBPART_KEY_COLUMNS 1244
14.3.142. ALL_TAB_COLUMNS 1244
14.3.143. ALL_TAB_PARTITIONS 1245
14.3.144. ALL_TAB_SUBPARTITIONS 1246
14.3.145. ALL_TAB_COLS 1247
14.3.146. ALL_TABLES 1249
14.3.147. ALL_TRIGGERS 1249
14.3.148. ALL_TYPES 1250
14.3.149. ALL_USERS 1250
14.3.150. ALL_VIEWS 1250
14.3.151. V$DATABASE 1251
14.3.152. V$INSTANCE 1253
14.3.153. V$LICENSE 1253
14.3.154. V$VERSION 1254
14.3.155. V$OBSOLETE_PARAMETER 1254
14.3.156. V$OPTION 1254
14.3.157. V$PARAMETER 1254
14.3.158. V$TABLESPACE 1255
14.3.159. V$OBJECT_USAGE 1255
14.3.160. V$DBLINK 1256
15. DBLINK 1257
16. 外部数据封装器 1258
16.1. JDBC FDW 1258
17. DBE_PERF Schema 1259
17.1. OS 1259
17.1.1. OS_RUNTIME 1259
17.1.2. GLOBAL_OS_RUNTIME 1260
17.1.3. OS_THREADS 1260
17.1.4. GLOBAL_OS_THREADS 1260
17.2. Instance 1261
17.2.1. INSTANCE_TIME 1261
17.2.2. GLOBAL_INSTANCE_TIME 1262
17.3. Memory 1262

17.3.1. MEMORY_NODE_DETAIL 1262
17.3.2. GLOBAL_MEMORY_NODE_DETAIL 1263
17.3.3. SHARED_MEMORY_DETAIL 1264
17.3.4. GLOBAL_SHARED_MEMORY_DETAIL 1265
17.4. File 1265
17.4.1. FILE_IOSTAT 1265
17.4.2. SUMMARY_FILE_IOSTAT 1266
17.4.3. GLOBAL_FILE_IOSTAT 1267
17.4.4. FILE_REDO_IOSTAT 1268
17.4.5. SUMMARY_FILE_REDO_IOSTAT 1268
17.4.6. GLOBAL_FILE_REDO_IOSTAT 1269
17.4.7. LOCAL_REL_IOSTAT 1269
17.4.8. GLOBAL_REL_IOSTAT 1270
17.4.9. SUMMARY_REL_IOSTAT 1270
17.5. Object 1271
17.5.1. STAT_USER_TABLES 1271
17.5.2. SUMMARY_STAT_USER_TABLES 1272
17.5.3. GLOBAL_STAT_USER_TABLES 1273
17.5.4. STAT_USER_INDEXES 1274
17.5.5. SUMMARY_STAT_USER_INDEXES 1275
17.5.6. GLOBAL_STAT_USER_INDEXES 1275
17.5.7. STAT_SYS_TABLES 1276
17.5.8. SUMMARY_STAT_SYS_TABLES 1277
17.5.9. GLOBAL_STAT_SYS_TABLES 1278
17.5.10. STAT_SYS_INDEXES 1279
17.5.11. SUMMARY_STAT_SYS_INDEXES 1280
17.5.12. GLOBAL_STAT_SYS_INDEXES 1280
17.5.13. STAT_ALL_TABLES 1281
17.5.14. SUMMARY_STAT_ALL_TABLES 1282
17.5.15. GLOBAL_STAT_ALL_TABLES 1283
17.5.16. STAT_ALL_INDEXES 1284
17.5.17. SUMMARY_STAT_ALL_INDEXES 1285
17.5.18. GLOBAL_STAT_ALL_INDEXES 1285
17.5.19. STAT_DATABASE 1286
17.5.20. SUMMARY_STAT_DATABASE 1287
17.5.21. GLOBAL_STAT_DATABASE 1289
17.5.22. STAT_DATABASE_CONFLICTS 1290

17.5.23. SUMMARY_STAT_DATABASE_CONFLICTS 1290
17.5.24. GLOBAL_STAT_DATABASE_CONFLICTS 1291
17.5.25. STAT_XACT_ALL_TABLES 1291
17.5.26. SUMMARY_STAT_XACT_ALL_TABLES 1292
17.5.27. GLOBAL_STAT_XACT_ALL_TABLES 1293
17.5.28. STAT_XACT_SYS_TABLES 1294
17.5.29. SUMMARY_STAT_XACT_SYS_TABLES 1294
17.5.30. GLOBAL_STAT_XACT_SYS_TABLES 1295
17.5.31. STAT_XACT_USER_TABLES 1296
17.5.32. SUMMARY_STAT_XACT_USER_TABLES 1296
17.5.33. GLOBAL_STAT_XACT_USER_TABLES 1297
17.5.34. STAT_XACT_USER_FUNCTIONS 1298
17.5.35. SUMMARY_STAT_XACT_USER_FUNCTIONS 1298
17.5.36. GLOBAL_STAT_XACT_USER_FUNCTIONS 1298
17.5.37. STAT_BAD_BLOCK 1299
17.5.38. SUMMARY_STAT_BAD_BLOCK 1300
17.5.39. GLOBAL_STAT_BAD_BLOCK 1300
17.5.40. STAT_USER_FUNCTIONS 1301
17.5.41. SUMMARY_STAT_USER_FUNCTIONS 1301
17.5.42. GLOBAL_STAT_USER_FUNCTIONS 1302
17.6. Workload 1302
17.6.1. WORKLOAD_SQL_COUNT 1302
17.6.2. SUMMARY_WORKLOAD_SQL_COUNT 1303
17.6.3. WORKLOAD_TRANSACTION 1303
17.6.4. SUMMARY_WORKLOAD_TRANSACTION 1304
17.6.5. GLOBAL_WORKLOAD_TRANSACTION 1305
17.6.6. WORKLOAD_SQL_ELAPSE_TIME 1306
17.6.7. SUMMARY_WORKLOAD_SQL_ELAPSE_TIME 1307
17.7. Session/Thread 1308
17.7.1. SESSION_STAT 1308
17.7.2. GLOBAL_SESSION_STAT 1308
17.7.3. SESSION_TIME 1309
17.7.4. GLOBAL_SESSION_TIME 1309
17.7.5. SESSION_MEMORY 1309
17.7.6. GLOBAL_SESSION_MEMORY 1310
17.7.7. SESSION_MEMORY_DETAIL 1310
17.7.8. GLOBAL_SESSION_MEMORY_DETAIL 1311

17.7.9. SESSION_STAT_ACTIVITY 1311
17.7.10. GLOBAL_SESSION_STAT_ACTIVITY 1313
17.7.11. THREAD_WAIT_STATUS 1315
17.7.12. GLOBAL_THREAD_WAIT_STATUS 1316
17.7.13. LOCAL_THREADPOOL_STATUS 1317
17.7.14. GLOBAL_THREADPOOL_STATUS 1318
17.7.15. SESSION_CPU_RUNTIME 1318
17.7.16. SESSION_MEMORY_RUNTIME 1319
17.7.17. STATEMENT_IOSTAT_COMPLEX_RUNTIME 1320
17.8. Transaction 1320
17.8.1. TRANSACTIONS_RUNNING_XACTS 1320
17.8.2. SUMMARY_TRANSACTIONS_RUNNING_XACTS 1321
17.8.3. GLOBAL_TRANSACTIONS_RUNNING_XACTS 1322
17.8.4. TRANSACTIONS_PREPARED_XACTS 1322
17.8.5. SUMMARY_TRANSACTIONS_PREPARED_XACTS 1323
17.8.6. GLOBAL_TRANSACTIONS_PREPARED_XACTS 1323
17.9. Query 1324
17.9.1. STATEMENT 1324
17.9.2. SUMMARY_STATEMENT 1325
17.9.3. STATEMENT_COUNT 1327
17.9.4. GLOBAL_STATEMENT_COUNT 1328
17.9.5. SUMMARY_STATEMENT_COUNT 1329
17.9.6. GLOBAL_STATEMENT_COMPLEX_HISTORY 1331
17.9.7. GLOBAL_STATEMENT_COMPLEX_HISTORY_TABLE 1335
17.9.8. GLOBAL_STATEMENT_COMPLEX_RUNTIME 1335
17.9.9. STATEMENT_RESPONSETIME_PERCENTILE 1338
17.9.10. STATEMENT_USER_COMPLEX_HISTORY 1338
17.9.11. STATEMENT_COMPLEX_RUNTIME 1338
17.9.12. STATEMENT_COMPLEX_HISTORY_TABLE 1341
17.9.13. STATEMENT_COMPLEX_HISTORY 1341
17.9.14. STATEMENT_WLMSTAT_COMPLEX_RUNTIME 1341
17.10. Cache/IO 1343
17.10.1. STATIO_USER_TABLES 1343
17.10.2. SUMMARY_STATIO_USER_TABLES 1344
17.10.3. GLOBAL_STATIO_USER_TABLES 1344
17.10.4. STATIO_USER_INDEXES 1345
17.10.5. SUMMARY_STATIO_USER_INDEXES 1346

17.10.6. GLOBAL_STATIO_USER_INDEXES 1346
17.10.7. STATIO_USER_SEQUENCES 1347
17.10.8. SUMMARY_STATIO_USER_SEQUENCES 1347
17.10.9. GLOBAL_STATIO_USER_SEQUENCES 1347
17.10.10. STATIO_SYS_TABLES 1348
17.10.11. SUMMARY_STATIO_SYS_TABLES 1349
17.10.12. GLOBAL_STATIO_SYS_TABLES 1349
17.10.13. STATIO_SYS_INDEXES 1350
17.10.14. SUMMARY_STATIO_SYS_INDEXES 1350
17.10.15. GLOBAL_STATIO_SYS_INDEXES 1351
17.10.16. STATIO_SYS_SEQUENCES 1351
17.10.17. SUMMARY_STATIO_SYS_SEQUENCES 1352
17.10.18. GLOBAL_STATIO_SYS_SEQUENCES 1352
17.10.19. STATIO_ALL_TABLES 1353
17.10.20. SUMMARY_STATIO_ALL_TABLES 1354
17.10.21. GLOBAL_STATIO_ALL_TABLES 1354
17.10.22. STATIO_ALL_INDEXES 1355
17.10.23. SUMMARY_STATIO_ALL_INDEXES 1355
17.10.24. GLOBAL_STATIO_ALL_INDEXES 1356
17.10.25. STATIO_ALL_SEQUENCES 1356
17.10.26. SUMMARY_STATIO_ALL_SEQUENCES 1357
17.10.27. GLOBAL_STATIO_ALL_SEQUENCES 1357
17.10.28. GLOBAL_STAT_DB_CU 1358
17.10.29. GLOBAL_STAT_SESSION_CU 1358
17.11. Utility 1359
17.11.1. REPLICATION_STAT 1359
17.11.2. GLOBAL_REPLICATION_STAT 1360
17.11.3. REPLICATION_SLOTS 1361
17.11.4. GLOBAL_REPLICATION_SLOTS 1361
17.11.5. BGWRITER_STAT 1362
17.11.6. GLOBAL_BGWRITER_STAT 1363
17.11.7. GLOBAL_CKPT_STATUS 1364
17.11.8. GLOBAL_DOUBLE_WRITE_STATUS 1364
17.11.9. GLOBAL_PAGEWRITER_STATUS 1365
17.11.10. GLOBAL_RECORD_RESET_TIME 1365
17.11.11. GLOBAL_REDO_STATUS 1366
17.11.12. GLOBAL_RECOVERY_STATUS 1367

17.11.13. CLASS_VITAL_INFO ... 1368
17.11.14. USER_LOGIN ... 1368
17.11.15. SUMMARY_USER_LOGIN ... 1368
17.12. Lock ... 1369
17.12.1. LOCKS ... 1369
17.12.2. GLOBAL_LOCKS ... 1370
17.13. Wait Events ... 1372
17.13.1. WAIT_EVENTS ... 1372
17.13.2. GLOBAL_WAIT_EVENTS ... 1372
17.14. Configuration ... 1373
17.14.1. CONFIG_SETTINGS ... 1373
17.14.2. GLOBAL_CONFIG_SETTINGS ... 1374
17.15. Operator ... 1375
17.15.1. OPERATOR_HISTORY_TABLE ... 1375
17.15.2. OPERATOR_HISTORY ... 1377
17.15.3. OPERATOR_RUNTIME ... 1377
17.15.4. GLOBAL_OPERATOR_HISTORY ... 1378
17.15.5. GLOBAL_OPERATOR_HISTORY_TABLE ... 1380
17.15.6. GLOBAL_OPERATOR_RUNTIME ... 1380
17.16. Workload Manager ... 1381
17.16.1. WLM_USER_RESOURCE_CONFIG ... 1381
17.16.2. WLM_USER_RESOURCE_RUNTIME ... 1382
18. WDR Snapshot Schema ... 1384
18.1. WDR Snapshot 原信息表 ... 1384
18.1.1. SNAPSHOT ... 1384
18.1.2. TABLES_SNAP_TIMESTAMP ... 1384
18.2. WDR Snapshot 数据表 ... 1385
18.3. 开启 WDR Snapshot ... 1385
18.3.1. 相关参数 ... 1385
18.3.2. 开启 WDR Snapshot 示例 ... 1385
18. 4. WDR Snapshot 生成性能报告 ... 1385
18.4.1. 前提条件 ... 1386
18.4.2. 操作步骤 ... 1386
18.4.3. 示例 ... 1387
19. GUC 参数说明 ... 1388
19.1. GUC 使用说明 ... 1388
19.2. 文件位置 ... 1388

19.3. 连接和认证 1390
19.3.1. 连接设置 1390
19.3.2. 安全和认证（postgresql.conf） 1393
19.3.3. 通信库参数 1401
19.4. 资源消耗 1405
19.4.1. 内存 1405
19.4.2. 磁盘空间 1411
19.4.3. 内核资源使用 1412
19.4.4. 基于开销的清理延迟 1413
19.4.5. 后端写进程 1415
19.4.6. 异步 IO 1416
19.5. 并行导入 1418
19.6. 预写式日志 1419
19.6.1. 设置 1419
19.6.2. 检查点 1423
19.6.3. 日志回放 1425
19.6.4. 归档 1427
19.7. 双机复制 1428
19.7.1. 发送端服务器 1428
19.7.2. 主服务器 1430
19.7.3. 备服务器 1433
19.8. 查询规划 1436
19.8.1. 优化器方法配置 1436
19.8.2. 优化器开销常量 1442
19.8.3. 基因查询优化器 1444
19.8.4. 其他优化器选项 1446
19.9. 错误报告和日志 1454
19.9.1. 记录日志的位置 1454
19.9.2. 记录日志的时间 1457
19.9.3. 记录日志的内容 1460
19.9.4. 使用 CSV 格式写日志 1468
19.10. 告警检测 1470
19.11. 运行时统计 1471
19.11.1. 查询和索引统计收集器 1471
19.11.2. 性能统计 1474
19.12. 负载管理 1474
19.13. 自动清理 1481

19.14. 客户端连接缺省设置 1485
19.14.1. 语句行为 1485
19.14.2. 区域和格式化 1490
19.14.3. 其他缺省 1494
19.15. 锁管理 1495
19.16. 版本和平台兼容性 1498
19.16.1. 历史版本兼容性 1498
19.16.2. 平台和客户端兼容性 1501
19.17. 容错性 1502
19.18. 连接池参数 1504
19.19. Vastbase 事务 1505
19.20. 开发人员选项 1507
19.21. 审计 1521
19.21.1. 审计开关 1521
19.21.2. 用户和权限审计 1523
19.21.3. 操作审计 1524
19.22. 事务监控 1530
19.23. 升级参数 1531
19.24. 其它选项 1531
19.25. 等待事件 1539
19.26. Query 1540
19.27. 系统性能快照 1541
20. 常见故障定位指南 1543
20.1. core 问题定位 1543
20.1.1. 磁盘满故障引起的 core 问题 1543
20.1.2. GUC 参数 log_directory 设置不正确引起的 core 问题 1543
20.2. 备机处于 need repair(WAL)状态问题 1544

1. 关于本文档

概述

本章介绍如何设计、创建、查询和维护数据库。包括 SQL 语句、存储过程、系统表和视图等。

读者对象

本文档是为基于 Vastbase G100 进行 C/Java 应用程序开发的程序员而写的，提供了必要的参考信息。

作为应用程序开发人员，至少需要了解以下知识：

- 操作系统知识。这是一切的基础。
- C/Java 语言。这是做应用程序开发的基础。
- 熟悉 C/Java 的一种 IDE。这是高效完成开发任务的必备条件。
- SQL 语法。这是操作数据库的必备能力。

符号约定

在本文中可能出现下列标志，它们所代表的含义如下。

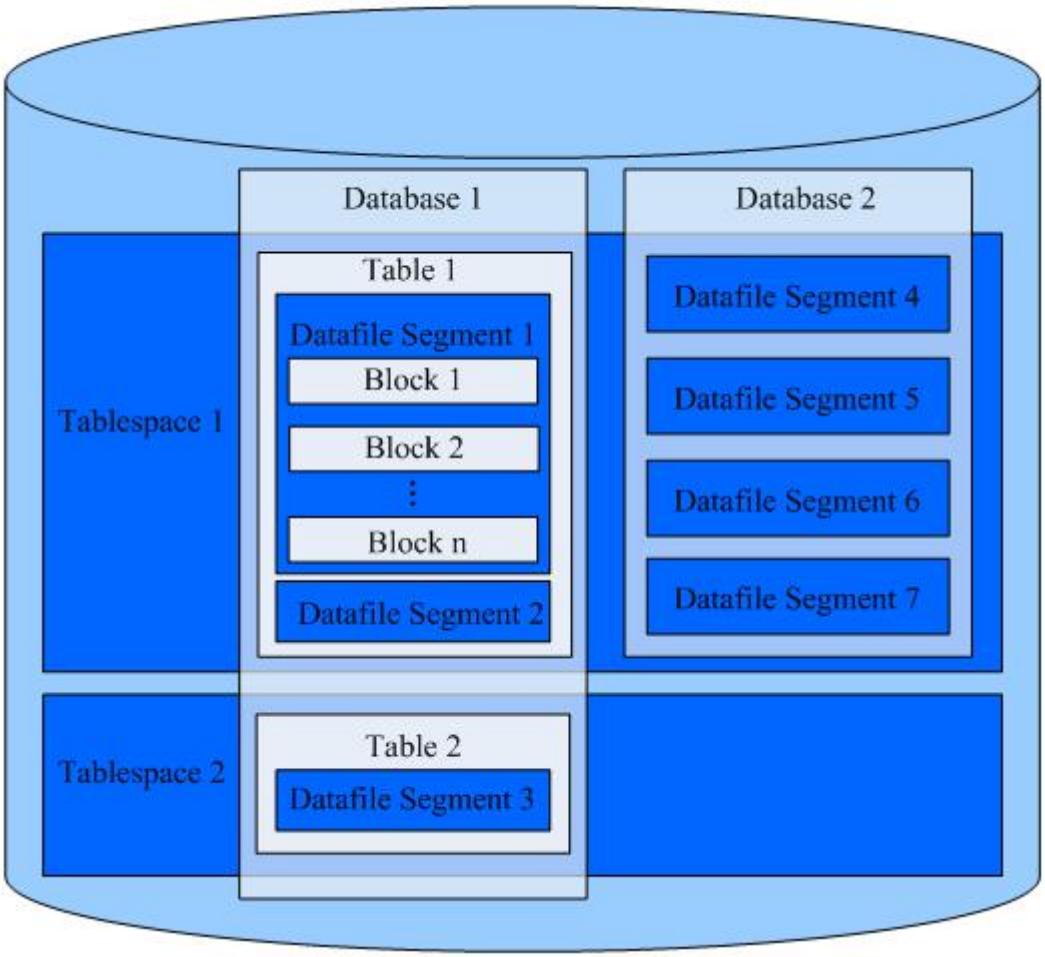
符号	说明
危险	表示如不避免则将会导致死亡或严重伤害的具有高等级风险的危害。
警告	表示如不避免则可能导致死亡或严重伤害的具有中等级风险的危害。
注意	表示如不避免则可能导致轻微或中度伤害的具有低等级风险的危害。
须知	用于传递设备或环境安全警示信息。如不避免则可能会导致设备损坏、数据丢失、设备性能降低或其它不可预知的结果。 “须知”不涉及人身伤害。
说明	对正文中重点信息的补充说明。 “说明”不是安全警示信息，不涉及人身、设备及环境伤害信息。

2. 概述

2.1. 数据库逻辑结构图

Vastbase 的数据库节点负责存储数据，其存储介质也是磁盘，本节主要从逻辑视角介绍数据库节点都有哪些对象，以及这些对象之间的关系。数据库逻辑结构如图 2-1。

图 2-1. 数据库逻辑结构图



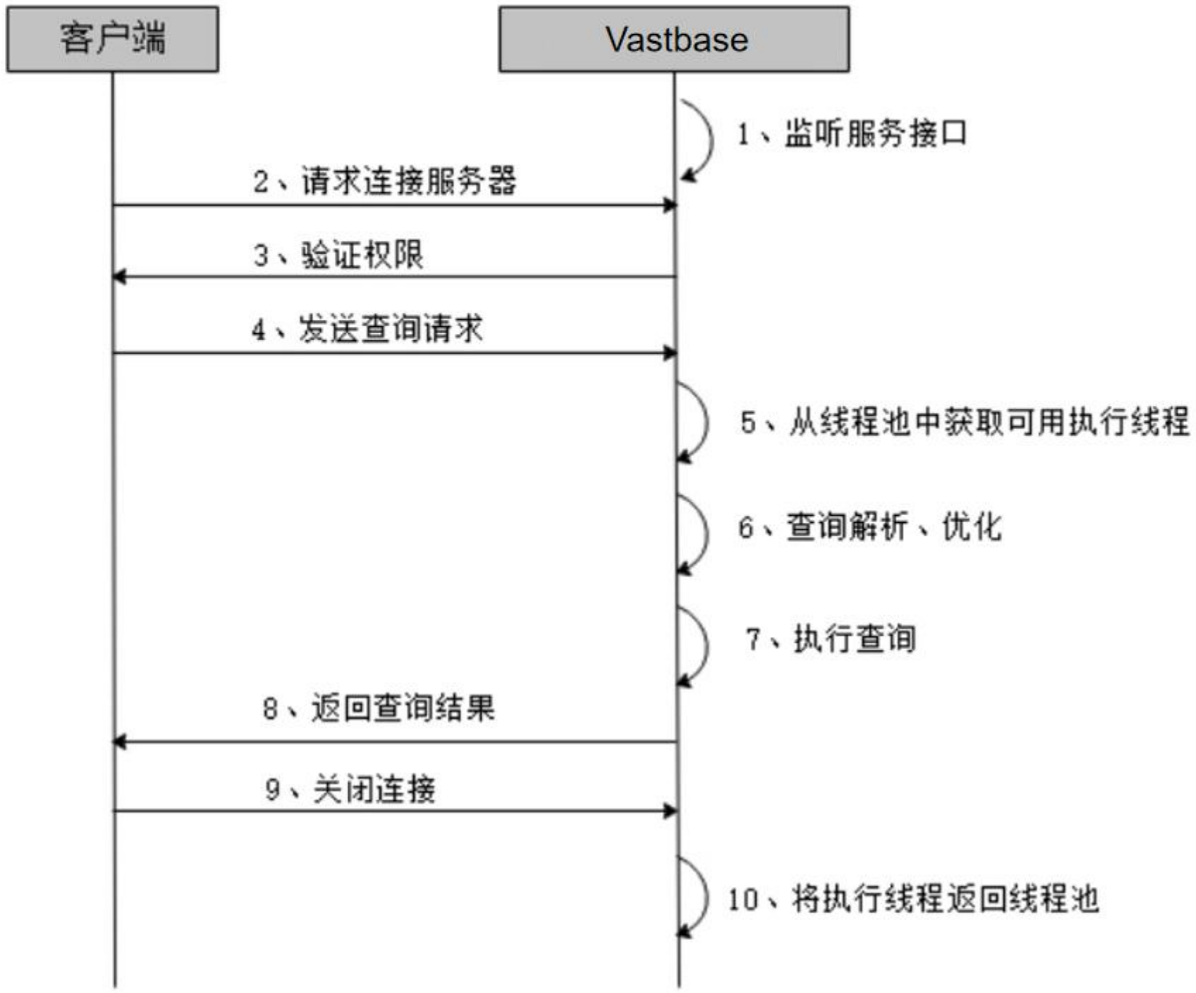
说明

- Tablespace，即表空间，是一个目录，可以存在多个，里面存储的是它所包含的数据库的各种物理文件。每个表空间可以对应多个 Database。
- Database，即数据库，用于管理各类数据对象，各数据库间相互隔离。数据库管理的对象可分布在多个 Tablespace 上。

- Datafile Segment，即数据文件，通常每张表只对应一个数据文件。如果某张表的数据大于 1GB，则会分为多个数据文件存储。
- Table，即表，每张表只能属于一个数据库，也只能对应到一个 Tablespace。每张表对应的数据文件必须在同一个 Tablespace 中。
- Block，即数据块，是数据库管理的基本单位，默认大小为 8KB。

2.2. 数据查询请求处理过程

图 2-2. Vastbase 服务响应流程



2.3. 管理事务

事务是用户定义的一个数据库操作序列，这些操作要么全做要么全不做，是一个不可分割的工作单位。Vastbase 数据库支持的事务控制命令有启动、设置、提交、回滚事务。Vastbase 数据库支持的事务隔离级别有读已提交和可重复读。

事务控制

以下是数据库支持的事务命令：

- 启动事务

  用户可以使用 11.16.111START TRANSACTION 和 11.16.27BEGIN 语法启动事务。

- 设置事务

  用户可以使用 SET TRANSACTION 或者 SET LOCAL TRANSACTION 语法设置事务特性，详细操作请参考 11.16.109SET TRANSACTION。

- 提交事务

  用户可以使用 COMMIT 或者 END 可完成提交事务的功能，即提交事务的所有操作，详细操作请参考 11.16.33COMMIT | END。

- 回滚事务

  回滚是在事务运行的过程中发生了某种故障，事务不能继续执行，系统将事务中对数据库的所有已完成的操作全部撤销，详细操作请参考 11.16.100ROLLBACK。

事务隔离级别

事务隔离级别，它决定多个事务并发操作同一个对象时的处理方式。

说明

在事务中第一个数据修改语句（SELECT, INSERT，DELETE，UPDATE，FETCH，COPY）执行之后，事务隔离级别就不能再次设置。

- READ COMMITTED：读已提交隔离级别，事务只能读到已提交的数据而不会读到未提交的数据，这是缺省值。

  实际上，SELECT 查询会查看到在查询开始运行的瞬间该数据库的一个快照。不过，SELECT 能查看到其自身所在事务中先前更新的执行结果。即使先前更新尚未提交。请注意，在同一个事务里两个相邻的 SELECT 命令可能会查看到不同的快照，因为其它事务会在第一个 SELECT 执行期间提交。

  因为在读已提交模式里，每个新的命令都是从一个新的快照开始的，而这个快照包含所有到该时刻为止已提交的事务，因此同一事务中后面的命令将看到任何已提交的其它事务的效果。这里关心的问题是在单个命令里是否看到数据库里绝对一致的视图。

  读已提交模式提供的部分事务隔离对于许多应用而言是足够的，并且这个模式速度快，使用简单。不过，对于做复杂查询和更新的应用，可能需要保证数据库有比读已提交模式更加严格的一致性视图。

- REPEATABLE READ：事务可重复读隔离级别，事务只能读到事务开始之前已提交的数据，不能读到未提交的数据以及事务执行期间其它并发事务提交的修改（但是，查询能查看到自身所在事

务中先前更新的执行结果，即使先前更新尚未提交）。这个级别和读已提交是不一样的，因为可重复读事务中的查询看到的是事务开始时的快照，不是该事务内部当前查询开始时的快照，就是说，单个事务内部的 select 命令总是查看到同样的数据，查看不到自身事务开始之后其他并发事务修改后提交的数据。使用该级别的应用必须准备好重试事务，因为可能会发生串行化失败。

2.4. 相关概念

数据库

数据库用于管理各类数据对象，与其他数据库隔离。创建数据对象时可以指定对应的表空间，如果不指定相应的表空间，相关的对象会默认保存在 PG_DEFAULT 空间中。数据库管理的对象可分布在多个表空间上。

表空间

在 Vastbase 中，表空间是一个目录，可以存在多个，里面存储的是它所包含的数据库的各种物理文件。由于表空间是一个目录，仅是起到了物理隔离的作用，其管理功能依赖于文件系统。

模式

Vastbase 的模式是对数据库做一个逻辑分割。所有的数据库对象都建立在模式下面。Vastbase 的模式和用户是弱绑定的，所谓的弱绑定是指虽然创建用户的同时会自动创建一个同名模式，但用户也可以单独创建模式，并且为用户指定其他的模式。

用户和角色

Vastbase 使用用户和角色来控制对数据库的访问。根据角色自身的设置不同，一个角色可以看做是一个数据库用户，或者一组数据库用户。在 Vastbase 中角色和用户之间的区别只在于角色默认是没有 LOGIN 权限的。在 Vastbase 中一个用户唯一对应一个角色，不过可以使用角色叠加来更灵活地进行管理。

事务管理

在事务管理上，Vastbase 采取了 MVCC（多版本并发控制）结合两阶段锁的方式，其特点是读写之间不阻塞。Vastbase 统一存放，而是和当前元组的版本放在了一起。Vastbase 期清除历史版本数据 Vastbase 引入了一个 VACUUM 线程。一般情况下用户不用关注它，除非要做性能调优。此外，Vastbase 是自动提交事务。

3. 数据库使用

3.1. 从这里开始

本节描述使用数据库的基本操作。通过此节您可以完成创建数据库、创建表及向表中插入数据和查询表中数据等操作。

前提条件

Vastbase 正常运行。

操作步骤

步骤 1 以操作系统用户 vastbase 登录数据库主节点。

步骤 2 连接数据库。

```
vsql -d vastbase -p 5432
```

当结果显示为如下信息，则表示连接成功。

```
vsql ((Vastbase 2.2.0 build ) compiled at 2021-01-26 19:22:33 commit 0 last mr  )
Non-SSL connection (SSL connection is recommended when requiring high-security)
Type "help" for help.

vastbase=#
```

其中，vastbase 为 Vastbase 安装完成后默认生成的数据库。初始可以连接到此数据库进行新数据库的创建。5432 为数据库主节点的端口号，需根据 Vastbase 的实际情况做替换。

引申信息：

- 使用数据库前，需先使用客户端程序或工具连接到数据库，然后就可以通过客户端程序或工具执行 SQL 来使用数据库了。
- vsql 是本产品提供的命令行方式的数据库连接工具。更多的数据库连接办法可参考 3.2 连接数据库。

步骤 3 创建数据库用户。

默认只有 Vastbase 安装时创建的管理员用户可以访问初始数据库，您还可以创建其他数据库用户帐号。

```
vastbase=# CREATE USER joe WITH PASSWORD "Bigdata@123";
```

当结果显示为如下信息，则表示创建成功。

```
CREATE ROLE
```

如上创建了一个用户名为 joe，密码为 Bigdata@123 的用户。

引申信息：关于数据库用户的更多信息请参考 5.2 管理用户及权限。

步骤 4 创建数据库。

```
vastbase=#  CREATE DATABASE db_tpcc OWNER joe;
```

当结果显示为如下信息，则表示创建成功。

```
CREATE DATABASE
```

db_tpcc 数据库创建完成后，就可以按如下方法退出 vastbase 数据库，使用新用户连接到 db_tpcc 数据库执行创建表等操作。您也可以选择继续在默认的 vastbase 数据库下进行后续的体验。

```
vastbase=#  \q
vsql -d db_tpcc -p 5432 -U joe -W Bigdata@123
vsql ((Vastbase 2.2.0 build ) compiled at 2021-01-26 19:22:33 commit 0 last mr  )
Non-SSL connection (SSL connection is recommended when requiring high-security)
Type "help" for help.

db_tpcc=>
```

创建 SCHEMA

```
db_tpcc=> CREATE SCHEMA joe AUTHORIZATION joe;
```

当结果显示为如下信息，则表示创建 SCHEMA 成功。

```
CREATE SCHEMA
```

引申信息：数据库默认创建在 pg_default 表空间下。若要指定表空间，可以使用如下语句：

```
vastbase=#  CREATE DATABASE db_tpcc WITH TABLESPACE = hr_local;
CREATE DATABASE
```

其中 hr_local 为表空间名称，关于如何创建表空间，请参考 3.7 创建和管理表空间。

步骤 5 创建表。

- 创建一个名称为 mytable，只有一列的表。字段名为 firstcol，字段类型为 integer。

```
db_tpcc=>  CREATE TABLE mytable (firstcol int);
CREATE TABLE
```

- 向表中插入数据：

```
db_tpcc=> INSERT INTO mytable values (100);
```

当结果显示为如下信息，则表示插入数据成功。

```
INSERT 0 1
```

- 查看表中数据：

```
db_tpcc=> SELECT * from mytable;
 firstcol
----------
      100
(1 row)
```

引申信息：

- 默认情况下，新的数据库对象是创建在“$user”模式下的，例如刚刚新建的表。关于模式的更多信息请参考 3.11.1 创建和管理 schema。
- 关于创建表的更多信息请参见 3.8 创建和管理表。
- 除了创建的表以外，数据库还包含很多系统表。这些系统表包含 Vastbase 安装信息以及 Vastbase 上运行的各种查询和进程的信息。可以通过查询系统表来收集有关数据库的信息。请参见规划存储模型 3.5

Vastbase 支持行列混合存储，为各种复杂场景下的交互分析提供较高的查询性能，关于存储模型的选择，请参考 3.5 规划存储模型。

3.2. 连接数据库

连接数据库的客户端工具包括 vsql、应用程序接口（如 ODBC 和 JDBC）。

- vsql 是 Vastbase 自带的客户端工具。3.2.2 使用 vsql 连接数据库，可以交互式地输入、编辑、执行 SQL 语句。
- 用户可以使用标准的数据库 3.2.3 应用程序接口（如 ODBC 和 JDBC），开发基于 Vastbase 的应用程序。

3.2.1. 配置服务端远程连接

进行远程连接前，需要在部署了数据库主节点的机器上设置允许客户端访问数据库，并配置远程连接。

操作步骤

以下步骤需要在 Vastbase 所在主机上执行。

步骤 1 以操作系统用户 vastbase 登录数据库主节点。

步骤 2 配置客户端认证方式，请参考 5.1.1 配置客户端接入认证。

步骤 3 配置 listen_addresses，listen_addresses 即远程客户端连接使用的数据库主节点 ip 或者主机名。

1. 进入数据库，使用如下 SQL 命令查看数据库主节点目前的 listen_addresses 配置。

```
show listen_addresses;
```

查询到的信息类似如下：

```
listen_addresses
------------------
 localhost
```

2. 使用如下命令把要添加的 IP 追加到 listen_addresses 后面，多个配置项之间用英文逗号分隔。例如，追加 IP 地址 172.16.103.84。

```
alter system set listen_addresses='localhost,172.16.103.84';
```

步骤 4 执行如下命令重启 Vastbase。

```
vb_ctl stop -D 数据库目录
vb_ctl start -D 数据库目录
```

3.2.2. 使用 vsql 连接

vsql 是 Vastbase 提供的在命令行下运行的数据库连接工具。此工具除了具备操作数据库的基本功能，还提供了若干高级特性，便于用户使用。本节只介绍如何使用 vsql 连接数据库，关于 vsql 使用方法的更多信息请参考《工具参考》中“客户端工具 > vsql”章节。

注意事项

缺省情况下，客户端连接数据库后处于空闲状态时会根据参数 session_timeout 的默认值自动断开连接。如果要关闭超时设置，设置参数 session_timeout 为 0 即可。

本地连接数据库

步骤 1 以操作系统用户 vastbase 登录数据库主节点。

步骤 2 连接数据库。

执行如下命令连接数据库。

```
vsql -d vastbase -p 5432
```

连接成功后，系统显示类似如下信息：

```
vsql ((Vastbase 2.2.0 build 290d125f) compiled at 2020-05-08 02:59:43 commit 2143 last mr
131
Non-SSL connection (SSL connection is recommended when requiring high-security)
Type "help" for help.


vastbase=#
```

vastbase 用户是管理员用户，因此系统显示“DBNAME=#”。若使用普通用户身份登录和连接数据库，系统显示“DBNAME=>”。

“Non-SSL connection”表示未使用 SSL 方式连接数据库。如果需要高安全性时，请 5.1.4 用 SSL 进行安全的 TCP/IP 连接。

步骤 3 首次登录需要修改密码。原始密码为安装 Vastbase 数据库手动输入的密码，具体请参见《安装指南》，此处需将原始密码修改为自定义的密码，例如 Mypwd123，命令如下：

```
vastbase=# ALTER ROLE vastbase IDENTIFIED BY 'Mypwd123' REPLACE 'XuanYuan@2012';
```

步骤 4 退出数据库。

```
vastbase=# \q
```

远程连接数据库

步骤 1 完成远程连接配置，操作步骤参见 3.2.1 配置服务端远程连接。

步骤 2 在客户端机器（10.10.0.30）上，连接数据库。

```
vsql -d vastbase -h 10.10.0.11 -U jack -p 5432 -W Test@123
```

vastbase 为需要连接的数据库名称，10.10.0.11 为数据库主节点所在的服务器 IP 地址，jack 为连接数据库的用户，5432 为数据库主节点的端口号，Test@123 为连接数据库用户 jack 的密码。

说明

- 连接 Vastbase 的机器与 Vastbase 不在同一网段时，-h 指定的 IP 地址应为 Manager 界面上所设的 coo.cooListenIp2（应用访问 IP）的取值。
- 禁止使用 vastbase 用户进行远程连接数据库。

3.2.3. 应用程序接口

用户可以使用标准的数据库应用程序接口（如 ODBC 和 JDBC），开发基于 Vastbase 的应用程序。

支持的应用程序接口

每个应用程序是一个独立的开发项目。应用程序通过 API 与数据库进行交互，在避免了应用程序直接操作数据库系统的同时，增强了应用程序的可移植性、扩展性和可维护性。表 3-1 为 Vastbase 所支持的应用程序接口及其下载地址。

表 3-1. 数据库应用程序接口

API	下载地址
ODBC	• Linux 下： 驱动程序：Vastbase-x.x-ODBC.tar.gz unixODBC 源码包： http://sourceforge.net/projects/unixodbc/files/unixODBC/2.3.0/unixODBC-2.3.0.tar.gz/download
JDBC	• 驱动程序：Vastbase-x.x-EULER-64bit-Jdbc.tar.gz • 驱动类：org.postgresql.Driver

使用 JDBC 和 ODBC 接口连接数据库属远程连接，因此需要 Vastbase 已经做了支持 3.2.1 配置服务端远程连接。

3.3. 关闭数据库

3.3.1. 使用 sql 语句关闭数据库

Vastbase 提供关闭数据库的 SQL，且允许指定关闭模式，执行该 SQL 可以指定模式关闭数据库实例。SQL 语法如下：

```
SHUTDOWN [SMART|FAST|IMMEDIATE]
```

模式说明：

- SMART：等所有的连接终止后，关闭数据库。如果客户端连接不终止，则无法关闭数据库。
- FAST：快速关闭数据库，断开客户端的连接，让已有的事物回滚，然后正常关闭数据库。
- IMMEDIATE：立即关闭数据库，立即停止数据库进程，直接退出，下次启动时会进行实例恢复。

3.3.2. 使用 vb_ctl 工具关闭

vb_ctl 是 Vastbase 提供的数据库服务控制工具，可以用来启停数据库服务和查询数据库状态。主要供 Vastbase 管理模块调用。这里主要介绍关闭数据库的用法，关于 vb_ctl 使用方法的更多信息请参考《工具参考》中“客户端工具 > vb_ctl”章节。

vb_ctl 关闭数据库语法如下：

```
vb_ctl stop -D 数据库目录 -p 端口号
```

3.4. 创建和管理数据库

前提条件

用户必须拥有数据库创建的权限或者是数据库的系统管理员权限才能创建数据库，赋予创建数据库的权限参见 5.2 管理用户及权限。

背景信息

- 初始时，Vastbase 包含两个模板数据库 template0、template1，以及两个默认的用户数据库 postgres 与 vastbase。

- CREATE DATABASE 实际上通过拷贝模板数据库来创建新数据库。默认情况下，拷贝template1。请避免使用客户端或其他手段连接及操作两个模板数据库。
- Vastbase 允许创建的数据库总数目上限为 128 个。
- 数据库系统中会有多个数据库，但是客户端程序一次只能连接一个数据库。也不能在不同的数据库之间相互查询。一个 Vastbase 中存在多个数据库时，需要通过-d 参数指定相应的数据库实例进行连接。

注意事项

如果数据库的编码为 SQL_ASCII（可以通过“show server_encoding”命令查看当前数据库存储编码），则在创建数据库对象时，可能造成出现半个字符的情况。如果对象名中含有多字节字符（例如中文），超过数据库对象名长度限制（63 字节）的时候，数据库将会将最后一个字节（而不是字符）截断。

针对这种情况，请遵循以下条件：

- 保证数据库对象的名称不超过限定长度。
- 使用例如 utf-8 编码集做为数据库的默认存储编码集（server_encoding）。
- 不要使用多字节字符做为对象名。
- 创建的数据库总数目不得超过 128 个。
- 如果出现因为误操作导致在多字节字符的中间截断而无法删除数据库对象的现象，请使用截断前的数据库对象名进行删除操作，或将该对象从各个数据库节点的相应系统表中依次删掉。

操作步骤

步骤 1 创建数据库

1. 使用如下命令创建一个新的表空间 tpcds_local。

```
vastbase=# CREATE TABLESPACE tpcds_local RELATIVE LOCATION 'tablespace/tablespace_1' ;
CREATE TABLESPACE
```

2. 使用如下命令创建一个新的数据库 db_tpcc。

```
vastbase=# CREATE DATABASE db_tpcc WITH TABLESPACE = tpcds_local;
CREATE DATABASE
```

说明

- 数据库名称遵循 SQL 标识符的一般规则。当前角色自动成为此新数据库的所有者。
- 如果一个数据库系统用于承载相互独立的用户和项目，建议把它们放在不同的数据库里。
- 如果项目或者用户是相互关联的，并且可以相互使用对方的资源，则应该把它们放在同一个数据库里，但可以规划在不同的模式中。模式只是一个纯粹的逻辑结构，某个模式的访问权限由权限系统模块控制。
- 创建数据库时，若数据库名称长度超过 63 字节，server 端会对数据库名称进行截断，保留前 63 个字节，因此建议数据库名称长度不要超过 63 个字节。

步骤 2 查看数据库

- 使用\l 元命令查看数据库系统的数据库列表。

```
vastbase=# \l
```

- 使用如下命令通过系统表 pg_database 查询数据库列表。

```
vastbase=# SELECT datname FROM pg_database;
```

步骤 3 修改数据库

用户可以使用如下命令修改数据库属性（比如：owner、名称和默认的配置属性）。

- 使用以下命令为数据库设置默认的模式搜索路径。

```
vastbase=# ALTER DATABASE db_tpcc SET search_path TO pa_catalog,public;
ALTER DATABASE
```

- 使用如下命令修改数据库表空间。

```
vastbase=# ALTER DATABASE db_tpcc SET TABLESPACE tpcds;
ALTER DATABASE
```

- 使用如下命令为数据库重新命名。

```
vastbase=# ALTER DATABASE db_tpcc RENAME TO human_tpcds;
ALTER DATABASE
```

步骤 4 删除数据库

用户可以使用 11.16.63DROP DATABASE 命令删除数据库。这个命令删除了数据库中的系统目录，并且删除了带有数据的磁盘上的数据库目录。用户必须是数据库的 owner 或者系统管理员才能删除数据库。当有人连接数据库时，删除操作会失败。删除数据库前请保证没有用户连接到该数据库。

使用如下命令删除数据库：

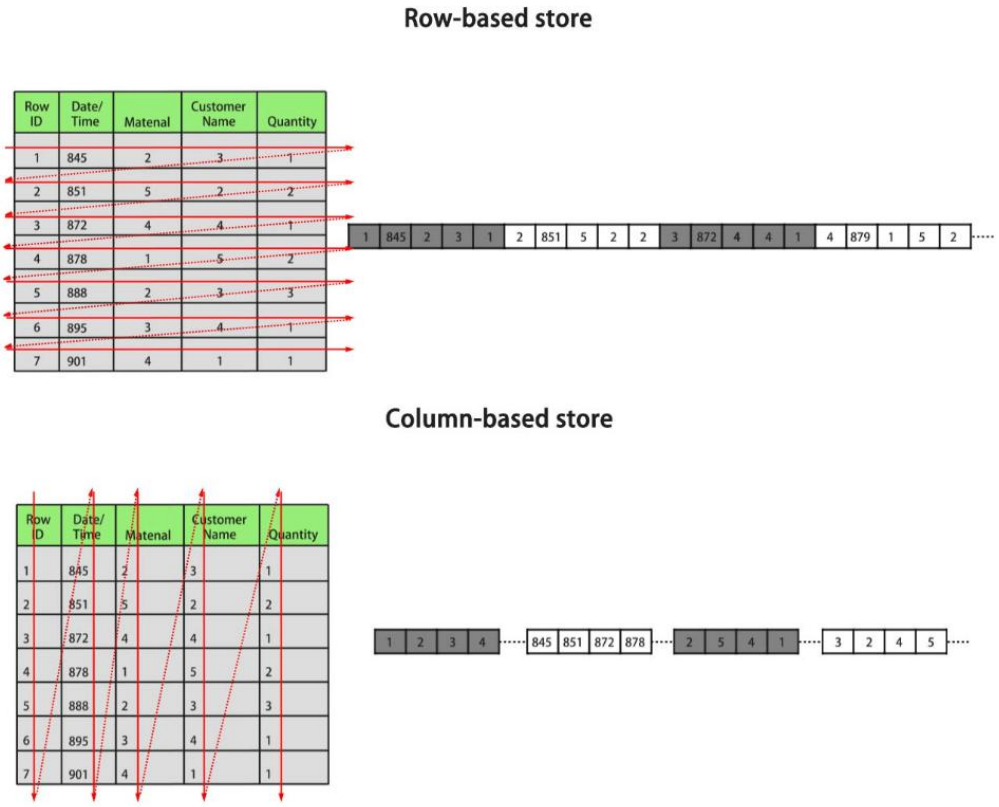
```
vastbase=# DROP DATABASE human_tpcds;
DROP DATABASE
```

3.5. 规划存储模型

Vastbase 支持行列混合存储。行、列存储模型各有优劣，建议根据实际情况选择。通常 Vastbase 用于 TP 场景的数据库，默认使用行存储，仅对执行复杂查询且数据量大的 AP 场景时，才使用列存储。

行存储是指将表按行存储到硬盘分区上，列存储是指将表按列存储到硬盘分区上。默认情况下，创建的表为行存储。行存储和列存储的差异请参见图 3-1。

图 3-1 行存储和列存储的差异



上图中，左上为行存表，右上为行存表在硬盘上的存储方式。左下为列存表，右下为列存表在硬盘上的存储方式。

行、列存储有如下优缺点：

存储模型	优点	缺点
行存	数据被保存在一起。INSERT/UPDATE 容易。	选择(Selection)时即使只涉及某几列，所有数据也都会被读取。
列存	• 查询时只有涉及到的列会被读取。 • 投影(Projection)很高效。 • 任何列都能作为索引。	• 选择完成时，被选择的列要重新组装。 • INSERT/UPDATE 比较麻烦。

一般情况下，如果表的字段比较多（大宽表），查询中涉及到的列不多的情况下，适合列存储。如果表的字段个数比较少，查询大部分字段，那么选择行存储比较好。

存储类型	适用场景
行存	• 点查询(返回记录少，基于索引的简单查询)。 • 增、删、改操作较多的场景。

存储类型	适用场景
列存	• 统计分析类查询（关联、分组操作较多的场景）。 • 即席查询（查询条件不确定，行存表扫描难以使用索引）。

行存表

默认创建表的类型。数据按行进行存储，即一行数据是连续存储。适用于对数据需要经常更新的场景。

```
vastbase=# CREATE TABLE customer_t1
(
  state_ID   CHAR(2),
  state_NAME VARCHAR2(40),
  area_ID    NUMBER
);

--删除表
vastbase=# DROP TABLE customer_t1;
```

列存表

数据按列进行存储，即一列所有数据是连续存储的。单列查询 IO 小，比行存表占用更少的存储空间。适合数据批量插入、更新较少和以查询为主统计分析类的场景。列存表不适合点查询。

```
vastbase=# CREATE TABLE customer_t2
(
  state_ID   CHAR(2),
  state_NAME VARCHAR2(40),
  area_ID    NUMBER
)
WITH (ORIENTATION = COLUMN);

--删除表
vastbase=# DROP TABLE customer_t2;
```

行存表和列存表的选择

- ❖ 更新频繁程度

  数据如果频繁更新，选择行存表。

- ❖ 插入频繁程度

  频繁的少量插入，选择行存表。一次插入大批量数据，选择列存表。

- ❖ 表的列数

  表的列数很多，选择列存表。

- ❖ 查询的列数

  如果每次查询时，只涉及了表的少数（<50%总列数）几个列，选择列存表。

❖ 压缩率

列存表比行存表压缩率高。但高压缩率会消耗更多的 CPU 资源。

3.6. 列存下的数据压缩

对于非活跃的早期数据可以通过压缩来减少空间占用，降低采购和运维成本。Vastbase 列存储压缩支持 Delta Value Encoding、Dictionary、RLE 、LZ4、ZLIB 等压缩算法，且能够根据数据特征自适应的选择压缩算法，平均压缩比 7:1。压缩数据可直接访问，对业务透明，极大缩短历史数据访问的准备时间。

3.7. 创建和管理表空间

背景信息

通过使用表空间，管理员可以控制一个数据库安装的磁盘布局。这样有以下优点：

❖ 如果初始化数据库所在的分区或者卷空间已满，又不能逻辑上扩展更多空间，可以在不同的分区上创建和使用表空间，直到系统重新配置空间。

❖ 表空间允许管理员根据数据库对象的使用模式安排数据位置，从而提高性能。

– 一个频繁使用的索引可以放在性能稳定且运算速度较快的磁盘上，比如一种固态设备。

– 一个存储归档的数据，很少使用的或者对性能要求不高的表可以存储在一个运算速度较慢的磁盘上。

❖ 管理员通过表空间可以设置占用的磁盘空间。用以在和其他数据共用分区的时候，防止表空间占用相同分区上的其他空间。

❖ 表空间可以控制数据库数据占用的磁盘空间，当表空间所在磁盘的使用率达到 90%时，数据库将被设置为只读模式，当磁盘使用率降到 90%以下时，数据库将恢复到读写模式。

建议用户使用数据库时，通过后台监控程序或者 Database Manager 进行磁盘空间使用率监控，以免出现数据库只读情况。

❖ 表空间对应于一个文件系统目录，假定数据库节点数据目录/pg_location/mount1/path1 是用户拥有读写权限的空目录。

使用表空间配额管理会使性能有 30%左右的影响，MAXSIZE 指定每个数据库节点的配额大小，误差范围在 500MB 以内。请根据实际的情况确认是否需要设置表空间的最大值。

操作步骤

❖ 创建表空间

步骤 1 执行如下命令创建用户 jack。

```
vastbase=# CREATE USER jack IDENTIFIED BY 'Bigdata@123';
```

当结果显示为如下信息，则表示创建成功。

```
CREATE ROLE
```

步骤 2 执行如下命令创建表空间。

```
vastbase=# CREATE TABLESPACE fastspace RELATIVE LOCATION 'tablespace/tablespace_1';
```

当结果显示为如下信息，则表示创建成功。

```
CREATE TABLESPACE
```

其中“fastspace”为新创建的表空间，“数据库节点数据目录*/pg_location/tablespace/tablespace_1*”是用户拥有读写权限的空目录。

步骤 3 数据库系统管理员执行如下命令将“fastspace”表空间的访问权限赋予数据用户 jack。

```
vastbase=# GRANT CREATE ON TABLESPACE fastspace TO jack;
```

当结果显示为如下信息，则表示赋予成功。

```
GRANT
```

❖ 在表空间中创建对象

如果用户拥有表空间的 CREATE 权限，就可以在表空间上创建数据库对象，比如：表和索引等。以创建表为例。

– 方式 1：执行如下命令在指定表空间创建表。

```
vastbase=# CREATE TABLE foo(i int) TABLESPACE fastspace;
```

当结果显示为如下信息，则表示创建成功。

```
CREATE TABLE
```

– 方式 2：先使用 set default_tablespace 设置默认表空间，再创建表。

```
vastbase=# SET default_tablespace = 'fastspace';
SET
vastbase=# CREATE TABLE foo2(i int);
CREATE TABLE
```

假设设置“fastspace”为默认表空间，然后创建表 foo2。

❖ 查询表空间

– 方式 1：检查 pg_tablespace 系统表。如下命令可查到系统和用户定义的全部表空间。

```
vastbase=# SELECT spcname FROM pg_tablespace;
```

– 方式 2：使用 vsql 程序的元命令查询表空间。

```
vastbase=# \db
```

❖ 查询表空间使用率

步骤 4 查询表空间的当前使用情况。

```
vastbase=#  SELECT PG_TABLESPACE_SIZE('fastspace');
```

返回如下信息：

```
 pg_tablespace_size
--------------------
            2146304
(1 row)
```

其中 2146304 表示表空间的大小，单位为字节。

步骤 5 计算表空间使用率。

表空间使用率=PG_TABLESPACE_SIZE/表空间所在目录的磁盘大小。

❖ 修改表空间

执行如下命令对表空间 fastspace 重命名为 fspace。

```
vastbase=# ALTER TABLESPACE fastspace RENAME TO fspace;
ALTER TABLESPACE
```

❖ 删除表空间

– 执行如下命令删除用户 jack。

```
vastbase=# DROP USER jack CASCADE;
DROP ROLE
```

– 执行如下命令删除表 foo 和 foo2。

```
vastbase=# DROP TABLE foo;
vastbase=# DROP TABLE foo2;
```

当结果显示为如下信息，则表示删除成功。

```
DROP TABLE
```

– 执行如下命令删除表空间 fspace。

```
vastbase=# DROP TABLESPACE fspace;
DROP TABLESPACE
```

说明

用户必须是表空间的 owner 或者系统管理员才能删除表空间。

3.8. 创建和管理表

3.8.1. 创建表

背景信息

表是建立在数据库中的，在不同的数据库中可以存放相同的表。甚至可以通过使用模式在同一个数据库中创建相同名称的表。创建表前请先 3.5 规划存储模型。

创建表

执行如下命令创建表。

```
vastbase=# CREATE TABLE customer_t1
(
   c_customer_sk          integer,
   c_customer_id          char(5),
```

```
    c_first_name            char(6),
    c_last_name             char(8)
);
```

当结果显示为如下信息，则表示创建成功。

```
CREATE TABLE
```

其中 c_customer_sk 、c_customer_id、c_first_name 和 c_last_name 是表的字段名，integer、char(5)、char(6)和 char(8)分别是这四字段名称的类型。

3.8.2. 向表中插入数据

在创建一个表后，表中并没有数据，在使用这个表之前，需要向表中插入数据。本小节介绍如何使用 11.16.89INSERT 命令插入一行或多行数据，及从指定表插入数据。如果有大量数据需要批量导入表中，请参考 7 导入数据。

背景信息

服务端与客户端使用不同的字符集时，两者字符集中单个字符的长度也会不同，客户端输入的字符串会以服务端字符集的格式进行处理，所以产生的最终结果可能会与预期不一致。

表 3-2. 客户端和服务端设置字符集的输出结果对比

操作过程	服务端和客户端编码一致	服务端和客户端编码不一致
存入和取出过程中没有对字符串进行操作	输出预期结果	输出预期结果（输入与显示的客户端编码必须一致）。
存入取出过程对字符串有做一定的操作（如字符串函数操作）	输出预期结果	根据对字符串具体操作可能产生非预期结果。
存入过程中对超长字符串有截断处理	输出预期结果	字符集中字符编码长度是否一致，如果不一致可能会产生非预期的结果。

上述字符串函数操作和自动截断产生的效果会有叠加效果，例如：在客户端与服务端字符集不一致的场景下，如果既有字符串操作，又有字符串截断，在字符串被处理完以后的情况下继续截断，这样也会产生非预期的效果。详细的示例请参见表 3-3。

说明

数据库 DBCOMPATIBILITY 设为兼容 TD 模式，且 td_compatible_truncation 参数设置为 on 的情况下，才会对超长字符串进行截断。

执行如下命令建立示例中需要使用的表 table1、table2。

```
vastbase=# CREATE TABLE table1(id int, a char(6), b varchar(6),c varchar(6));
vastbase=# CREATE TABLE table2(id int, a char(20), b varchar(20),c varchar(20));
```

表 3-3. 示例

编号	服务端字符集	客户端字符集	是否启用自动截断	示例	结果	说明
1	SQL_ASCII	UTF8	是	vastbase=# **INSERT INTO** *table1* **VALUES**(*1,reverse('123 A A 78'),reverse('123 A A 78'),reverse('123 A A 78')*);	id |a|b|c ----+------+------+------ 1 | 87| 87| 87	字符串在服务端翻转后，并进行截断，由于服务端和客户端的字符集不一致，字符 A 在客户端由多个字节表示，结果产生异常。
2	SQL_ASCII	UTF8	是	vastbase=# **INSERT INTO** *table1* **VALUES**(*2,reverse('123 A 78'),reverse('123 A 78'),reverse('123 A 78')*);	id |a|b|c ----+------+------+------ 2 | 873| 873| 873	字符串翻转后，又进行了自动截断，所以产生了非预期的效果。
3	SQL_ASCII	UTF8	是	vastbase=# **INSERT INTO** *table1* **VALUES**(*3,'87A 123','87A 123','87A 123'*);	id | a | b | c ----+-------+-------+------- 3 | 87A1 | 87A1 | 87A1	字符串类型的字段长度是客户端字符编码长度的整数倍，所以截断后产生结果正常。

编号	服务端字符集	客户端字符集	是否启用自动截断	示例	结果	说明
4	SQL_ASCII	UTF8	否	vastbase=# **INSERT INTO** *table2* **VALUES**(*1,reverse('123 A A 78'),reverse('123 A A 78'),reverse('123 A A 78')*); vastbase=# **INSERT INTO** *table2* **VALUES**(*2,reverse('123 A 78'),reverse('123 A 78'),reverse('123 A 78')*);	id |a|b|c ----+-------------------+--------+-------- 1 | 87□321| 87□321 | 87□321 2 | 87321| 87321| 87321	与示例 1 类似，多字节字符翻转之后不再表示原来的字符。

操作步骤

向表中插入数据前，意味着表已创建成功。创建表的步骤请参考 3.8 创建和管理表。

- 向表 customer_t1 中插入一行：

  数据值是按照这些字段在表中出现的顺序列出的，并且用逗号分隔。通常数据值是文本（常量），但也允许使用标量表达式。

```
vastbase=# INSERT INTO customer_t1(c_customer_sk, c_customer_id, c_first_name) VALUES (3769, 'hello', 'Grace');
```

  如果用户已经知道表中字段的顺序，也可无需列出表中的字段。例如以下命令与上面的命令效果相同。

```
vastbase=# INSERT INTO customer_t1 VALUES (3769, 'hello', 'Grace');
```

  如果用户不知道所有字段的数值，可以忽略其中的一些。没有数值的字段将被填充为字段的缺省值。例如：

```
vastbase=# INSERT INTO customer_t1 (c_customer_sk, c_first_name) VALUES (3769, 'Grace');

vastbase=# INSERT INTO customer_t1 VALUES (3769, 'hello');
```

  用户也可以对独立的字段或者整个行明确缺省值：

```
vastbase=# INSERT INTO customer_t1 (c_customer_sk, c_customer_id, c_first_name) VALUES (3769, 'hello', DEFAULT);
```

```
vastbase=# INSERT INTO customer_t1 DEFAULT VALUES;
```

❖ 如果需要在表中插入多行，请使用以下命令：

```
vastbase=# INSERT INTO customer_t1 (c_customer_sk, c_customer_id, c_first_name) VALUES
   (6885, 'maps', 'Joes'),
   (4321, 'tpcds', 'Lily'),
   (9527, 'world', 'James');
```

如果需要向表中插入多条数据，除此命令外，也可以多次执行插入一行数据命令实现。但是建议使用此命令可以提升效率。

❖ 如果从指定表插入数据到当前表，例如在数据库中创建了一个表 customer_t1 的备份表 customer_t2，现在需要将表 customer_t1 中的数据插入到表 customer_t2 中，则可以执行如下命令。

```
vastbase=# CREATE TABLE customer_t2
(
   c_customer_sk           integer,
   c_customer_id           char(5),
   c_first_name            char(6),
   c_last_name             char(8)
);

vastbase=# INSERT INTO customer_t2 SELECT * FROM customer_t1;
```

说明

从指定表插入数据到当前表时，若指定表与当前表对应的字段数据类型之间不存在隐式转换，则这两种数据类型必须相同。

❖ 删除备份表

```
vastbase=# DROP TABLE customer_t2 CASCADE;
```

说明

在删除表的时候，若当前需删除的表与其他表有依赖关系，需先删除关联的表，然后再删除当前表。

UPSERT 功能

UPSERT 功能用于往目标表中插入或更新数据。当插入数据因为主键或约束产生冲突时，更新数据，不冲突则插入数据。例：

```
CREATE TABLE test01 (col1 INT PRIMARY KEY, col2 INT);
insert into test01 values(1,1);
INSERT INTO test01(col1) VALUES(1) ON DUPLICATE KEY UPDATE col2 = 2;
```

3.8.3. 更新表中数据

修改已经存储在数据库中数据的行为叫做更新。用户可以更新单独一行，所有行或者指定的部分行。还可以独立更新每个字段，而其他字段则不受影响。

使用 UPDATE 命令更新现有行，需要提供以下三种信息：

❖ 表的名称和要更新的字段名

- 字段的新值
- 要更新哪些行

SQL 通常不会为数据行提供唯一标识，因此无法直接声明需要更新哪一行。但是可以通过声明一个被更新的行必须满足的条件。只有在表里存在主键的时候，才可以通过主键指定一个独立的行。

建立表和插入数据的步骤请参考 3.8.1 创建表和 3.8.2 向表中插入数据。

需要将表 customer_t1 中 c_customer_sk 为 9527 的地域重新定义为 9876：

```
vastbase=# UPDATE customer_t1 SET c_customer_sk = 9876 WHERE c_customer_sk = 9527;
```

这里的表名称也可以使用模式名修饰，否则会从默认的模式路径找到这个表。SET 后面紧跟字段和新的字段值。新的字段值不仅可以是常量，也可以是变量表达式。

比如，把所有 c_customer_sk 的值增加 100：

```
vastbase=# UPDATE customer_t1 SET c_customer_sk = c_customer_sk + 100;
```

在这里省略了 WHERE 子句，表示表中的所有行都要被更新。如果出现了 WHERE 子句，那么只有匹配其条件的行才会被更新。

在 SET 子句中的等号是一个赋值，而在 WHERE 子句中的等号是比较。WHERE 条件不一定是相等测试，许多其他的操作符也可以使用。

用户可以在一个 UPDATE 命令中更新更多的字段，方法是在 SET 子句中列出更多赋值，比如：

```
vastbase=# UPDATE customer_t1 SET  c_customer_id = 'Admin', c_first_name = 'Local' WHERE c_customer_sk
= 4421;
```

批量更新或删除数据后，会在数据文件中产生大量的删除标记，查询过程中标记删除的数据也是需要扫描的。故多次批量更新/删除后，标记删除的数据量过大会严重影响查询的性能。建议在批量更新/删除业务会反复执行的场景下，定期执行 VACUUM FULL 以保持查询性能。

3.8.4. 查看数据

- 使用系统表 pg_tables 查询数据库所有表的信息。

```
vastbase=# SELECT * FROM pg_tables;
```

- 使用 vsql 的\d+命令查询表的属性。

```
vastbase=# \d+ customer_t1;
```

- 执行如下命令查询表 customer_t1 的数据量。

```
vastbase=# SELECT count(*) FROM customer_t1;
```

- 执行如下命令查询表 customer_t1 的所有数据。

```
vastbase=# SELECT * FROM customer_t1;
```

- 执行如下命令只查询字段 c_customer_sk 的数据。

```
vastbase=# SELECT c_customer_sk FROM customer_t1;
```

- 执行如下命令过滤字段 c_customer_sk 的重复数据。

```
vastbase=# SELECT DISTINCT( c_customer_sk ) FROM customer_t1;
```

- 执行如下命令查询字段 c_customer_sk 为 3869 的所有数据。

```
vastbase=# SELECT * FROM customer_t1 WHERE c_customer_sk = 3869;
```

- 执行如下命令按照字段 c_customer_sk 进行排序。

```
vastbase=# SELECT * FROM customer_t1 ORDER BY c_customer_sk;
```

3.8.5. 删除表中数据

在使用表的过程中，可能会需要删除已过期的数据，删除数据必须从表中整行的删除。

SQL 不能直接访问独立的行，只能通过声明被删除行匹配的条件进行。如果表中有一个主键，用户可以指定准确的行。用户可以删除匹配条件的一组行或者一次删除表中的所有行。

使用 DELETE 命令删除行，如果删除表 customer_t1 中所有 c_customer_sk 为 3869 的记录：

```
vastbase=# DELETE FROM customer_t1 WHERE c_customer_sk = 3869;
```

如果执行如下命令之一，会删除表中所有的行。

```
vastbase=# DELETE FROM customer_t1;
或:
vastbase=# TRUNCATE TABLE customer_t1;
```

说明

全表删除的场景下，建议使用 truncate，不建议使用 delete。

删除创建的表：

```
vastbase=# DROP TABLE customer_t1;
```

3.8.6. 闪回表

在使用表的过程中，可能会误删或误改数据，此时可能需要使用闪回表、闪回查询功能。

修改 postgresql.conf，添加参数 max_flashback_time，例如将值设为 20，即最多允许闪回至 20 秒之前。修改完成后重启数据库。

❖闪回查询 15 秒以前的表数据，使用 vsql 客户端工具执行以下命令：

```
CREATE TABLE t person (id bigint, name varchar2(20));
INSERT INTO t person VALUES (1,'bob');
INSERT INTO t person VALUES (2,'tim');
SELECT * FROM t_person;
DELETE FROM t_person WHERE ID = 2;
SELECT pg_sleep(10);
SELECT * FROM t_person;
SELECT * FROM t_person flashback(15);
SELECT * FROM t_person;
```

❖闪回表，对数据表的操作进行回退，使用 vsql 客户端工具执行以下命令：

```
TRUNCATE TABLE t_person;
INSERT INTO t_person VALUES (1,'bob');
INSERT INTO t_person VALUES (2,'tim');
INSERT INTO t_person VALUES (3,'kiki');
SELECT * FROM t_person;
SELECT pg_sleep(10);
DELETE FROM t_person WHERE id = 3;
SELECT * FROM t_person;
SELECT pg_flashback('public.t_person',15);
SELECT * FROM t_person;
```

3.8.7. 全局临时表

功能描述

全局临时表像普通表一样，有结构，但是对数据的管理上不一样，临时表存储事务或会话的中间结果集，临时表中保存的数据只对当前会话可见，所有会话都看不到其他会话的数据，即使其他会话提交了，也看不到。临时表不存在并发行为，因为他们对于当前会话都是独立的。

创建临时表时，Vastbase 只创建了表的结构（在数据字典中定义），并没有初始化存储空间，当某一会话使用临时表时，Vastbase 会从当前用户的临时表空间分配一块空间。也就是说只有向临时表中插入数据时，才会给临时表分配存储空间。

临时表分事务级临时表和会话级临时表：

- 事务级临时表只对当前事务有效，通过语句：ON COMMIT DELETE ROWS 指定。
- 会话级临时表对当前会话有效，通过语句：ON COMMIT PRESERVE ROWS 语句指定。 这是不指定 ON COMMIT 选项的缺省行为。

示例

```
CREATE GLOBAL TEMP TABLE temptest(tcol int)
ON COMMIT PRESERVE ROWS;
INSERT INTO temptest VALUES (1);
update temptest set tcol=10;
SELECT * FROM temptest;
delete from temptest;
drop table temptest;
```

3.8.8. 创建和管理分区表

背景信息

Vastbase 数据库支持的分区表：Range 分区表，List 分区表，Hash 分区表。

分区表和普通表相比具有以下优点：

- 改善查询性能：对分区对象的查询可以仅搜索自己关心的分区，提高检索效率。
- 增强可用性：如果分区表的某个分区出现故障，表在其他分区的数据仍然可用。
- 方便维护：如果分区表的某个分区出现故障，需要修复数据，只修复该分区即可。
- 均衡 I/O：可以把不同的分区映射到不同的
- 磁盘以平衡 I/O，改善整个系统性能。

普通表若要转成分区表，需要新建分区表，然后把普通表中的数据导入到新建的分区表中。因此在初始设计表时，请根据业务提前规划是否使用分区表。

Range 分区：将数据基于范围映射到每一个分区，这个范围是由创建分区表时指定的分区键决定的。这种分区方式是最为常用的，并且分区键经常采用日期，例如将销售数据按照月份进行分区。

操作步骤

按照以下方式对 Range 分区表进行操作。

- 创建表空间

```
vastbase=# CREATE TABLESPACE example1 RELATIVE LOCATION 'tablespace1/tablespace_1';
vastbase=# CREATE TABLESPACE example2 RELATIVE LOCATION 'tablespace2/tablespace_2';
vastbase=# CREATE TABLESPACE example3 RELATIVE LOCATION 'tablespace3/tablespace_3';
vastbase=# CREATE TABLESPACE example4 RELATIVE LOCATION 'tablespace4/tablespace_4';
```

当结果显示为如下信息，则表示创建成功。

```
CREATE TABLESPACE
```

- 创建分区表

```
vastbase=# CREATE TABLE tpcds.customer_address
(
    ca_address_sk      integer               NOT NULL   ,
    ca_address_id      character(16)          NOT NULL   ,
    ca_street_number    character(10)                     ,
    ca_street_name     character varying(60)              ,
    ca_street_type     character(15)                     ,
    ca_suite_number    character(10)                     ,
    ca_city           character varying(60)              ,
    ca_county         character varying(30)              ,
    ca_state          character(2)                      ,
    ca_zip            character(10)                     ,
    ca_country         character varying(20)              ,
    ca_gmt_offset      numeric(5,2)                      ,
    ca_location_type    character(20)
)
TABLESPACE example1

PARTITION BY RANGE (ca_address_sk)
(
       PARTITION P1 VALUES LESS THAN(5000),
       PARTITION P2 VALUES LESS THAN(10000),
       PARTITION P3 VALUES LESS THAN(15000),
       PARTITION P4 VALUES LESS THAN(20000),
       PARTITION P5 VALUES LESS THAN(25000),
       PARTITION P6 VALUES LESS THAN(30000),
       PARTITION P7 VALUES LESS THAN(40000),
       PARTITION P8 VALUES LESS THAN(MAXVALUE) TABLESPACE example2
)
ENABLE ROW MOVEMENT;
```

当结果显示为如下信息，则表示创建成功。

```
CREATE TABLE
```

说明

创建列存分区表的数量建议不超过 1000 个。

- 插入数据

将表 tpcds.customer_address 的数据插入到表 tpcds.web_returns_p2 中。

例如在数据库中创建了一个表 tpcds.customer_address 的备份表 tpcds.web_returns_p2，现在需要将表 tpcds.customer_address 中的数据插入到表 tpcds.web_returns_p2 中，则可以执行如下命令。

```
vastbase=# CREATE TABLE tpcds.web_returns_p2
(
    ca_address_sk      integer               NOT NULL   ,
    ca_address_id      character(16)          NOT NULL   ,
    ca_street_number    character(10)                     ,
    ca_street_name     character varying(60)              ,
    ca_street_type     character(15)                     ,
    ca_suite_number    character(10)                     ,
    ca_city           character varying(60)              ,
    ca_county         character varying(30)              ,
    ca_state          character(2)                      ,
    ca_zip            character(10)                     ,
    ca_country         character varying(20)             ,
    ca_gmt_offset      numeric(5,2)                      ,
    ca_location_type    character(20)
)
TABLESPACE example1
PARTITION BY RANGE (ca_address_sk)
(
       PARTITION P1 VALUES LESS THAN(5000),
       PARTITION P2 VALUES LESS THAN(10000),
       PARTITION P3 VALUES LESS THAN(15000),
       PARTITION P4 VALUES LESS THAN(20000),
       PARTITION P5 VALUES LESS THAN(25000),
       PARTITION P6 VALUES LESS THAN(30000),
       PARTITION P7 VALUES LESS THAN(40000),
       PARTITION P8 VALUES LESS THAN(MAXVALUE) TABLESPACE example2
)
ENABLE ROW MOVEMENT;
CREATE TABLE
vastbase=# INSERT INTO tpcds.web_returns_p2 SELECT * FROM tpcds.customer_address;
INSERT 0 0
```

- 修改分区表行迁移属性

```
vastbase=# ALTER TABLE tpcds.web_returns_p2 DISABLE ROW MOVEMENT;
ALTER TABLE
```

- 删除分区

删除分区 P8。

```
vastbase=# ALTER TABLE tpcds.web_returns_p2 DROP PARTITION P8;
ALTER TABLE
```

- 增加分区

增加分区 P8，范围为 40000<= P8<=MAXVALUE。

```
vastbase=# ALTER TABLE tpcds.web_returns_p2 ADD PARTITION P8 VALUES LESS THAN (MAXVALUE);
ALTER TABLE
```

- 重命名分区
  - 重命名分区 P8 为 P_9。

```
vastbase=# ALTER TABLE tpcds.web_returns_p2 RENAME PARTITION P8 TO P_9;
ALTER TABLE
```

- 重命名分区 P_9 为 P8。

```
vastbase=# ALTER TABLE tpcds.web_returns_p2 RENAME PARTITION FOR (40000) TO P8;
ALTER TABLE
```

❖ 修改分区的表空间

- 修改分区 P6 的表空间为 example3。

```
vastbase=#  ALTER TABLE tpcds.web_returns_p2 MOVE PARTITION P6 TABLESPACE example3;
ALTER TABLE
```

- 修改分区 P4 的表空间为 example4。

```
vastbase=#  ALTER TABLE tpcds.web_returns_p2 MOVE PARTITION P4 TABLESPACE example4;
ALTER TABLE
```

❖ 查询分区

查询分区 P6。

```
vastbase=# SELECT * FROM tpcds.web_returns_p2 PARTITION (P6);
vastbase=# SELECT * FROM tpcds.web_returns_p2 PARTITION FOR (35888);
```

❖ 删除分区表和表空间

```
vastbase=# DROP TABLE tpcds.web_returns_p2;
vastbase=# DROP TABLE tpcds.customer_address;
DROP TABLE
vastbase=# DROP TABLESPACE example1;
vastbase=# DROP TABLESPACE example2;
vastbase=# DROP TABLESPACE example3;
vastbase=# DROP TABLESPACE example4;
DROP TABLESPACE
```

按照以下方式对 Hash 分区表进行操作。

❖ 创建分区表

```
vastbase=# CREATE TABLE t_hash
vastbase-# (c1 integer,
vastbase(# c2 date,
vastbase(# c3 text)
vastbase-# PARTITION BY HASH (c1)
vastbase-# (
vastbase(# PARTITION t_hash_p1 VALUES (MODULUS 4, REMAINDER 0),
vastbase(# PARTITION t hash p2 VALUES (MODULUS 4, REMAINDER 1)
vastbase(# );
CREATE TABLE
vastbase=# SELECT relname, parttype, partstrategy, listitem FROM pg_partition;
  relname  | parttype | partstrategy | listitem
-----------+----------+--------------+----------
 t_hash    | r        | h            |
 t_hash_p2 | p        | h            |
 t_hash_p1 | p        | h            |
```

❖ 新增分区

```
vastbase=# alter table t_hash add partition t_hash_p3 VALUES (MODULUS 4, REMAINDER 2);
ALTER TABLE
vastbase=# SELECT relname, parttype, partstrategy, listitem FROM pg_partition;
  relname  | parttype | partstrategy | listitem
-----------+----------+--------------+----------
```

```
 t_hash    | r       | h          |
 t_hash_p2 | p       | h          |
 t_hash_p1 | p       | h          |
 t_hash_p3 | p       | h          |
(4 rows)
```

- 删除分区

```
vastbase=# alter table t_hash drop partition t_hash_p3;
ALTER TABLE
vastbase=# SELECT relname, parttype, partstrategy, listitem FROM pg_partition;
  relname  | parttype | partstrategy | listitem
-----------+----------+--------------+----------
 t_hash    | r       | h          |
 t_hash_p2 | p       | h          |
 t_hash_p1 | p       | h          |
(3 rows)
```

- 重命名分区

```
vastbase=# alter table t_hash rename partition t_hash_p1 to t_hash_p1_rename;
ALTER TABLE
vastbase=# SELECT relname, parttype, partstrategy, listitem FROM pg_partition;
    relname     | parttype | partstrategy | listitem
----------------+----------+--------------+----------
 t_hash         | r       | h          |
 t_hash_p2      | p       | h          |
 t_hash_p1_rename | p       | h          |
(3 rows)
```

- 插入数据

```
vastbase=# insert into t_hash values(1,'2020-07-29','test');
INSERT 0 1
```

按照以下方式对 List 分区表进行操作。

- 创建分区表

```
vastbase=# CREATE TABLE t_list
vastbase-# (c1 integer,
vastbase(# c2 date,
vastbase(# c3 text)
vastbase-# PARTITION BY LIST (c2)
vastbase-# (
vastbase(# PARTITION p1 VALUES ('2019-10-12'),
vastbase(# PARTITION p2 VALUES ('2019-10-13'),
vastbase(# PARTITION p3 VALUES ('2019-10-14')
vastbase(# );
CREATE TABLE

vastbase=# SELECT relname, parttype, partstrategy, listitem FROM pg_partition;
    relname     | parttype | partstrategy |   listitem
----------------+----------+--------------+--------------
 t_hash         | r       | h          |
 t_hash_p2      | p       | h          |
 t_hash_p1_rename | p       | h          |
 t_list         | r       | l          |
 p3             | p       | l          | {2019-10-14}
 p2             | p       | l          | {2019-10-13}
```

```
 p1             | p       | l          | {2019-10-12}
(7 rows)
(3 rows)
```

- 修改分区表行迁移属性

```
vastbase=# alter table t_list enable row movement;
ALTER TABLE
```

- 新增分区

```
vastbase=# ALTER TABLE t_list ADD PARTITION P4 VALUES ('2019-10-15');
ALTER TABLE
vastbase=# SELECT relname, parttype, partstrategy, listitem FROM pg_partition;
    relname     | parttype | partstrategy |   listitem
------------------+----------+--------------+--------------
 t_hash          | r       | h          |
 t_hash_p2       | p       | h          |
 t_hash_p1_rename | p       | h          |
 t_list          | r       | l          |
 p3              | p       | l          | {2019-10-14}
 p2              | p       | l          | {2019-10-13}
 p1              | p       | l          | {2019-10-12}
 p4              | p       | l          | {2019-10-15}
(8 rows)
```

- 删除分区

```
vastbase=# ALTER TABLE t_list DROP PARTITION p4;
ALTER TABLE
```

- 插入数据

```
vastbase=# insert into t_list values(1,'2019-10-13','test');
INSERT 0 1
vastbase=# SELECT * FROM t_list PARTITION (p2);
 c1 |        c2         |  c3
----+---------------------+------
  1 | 2019-10-13 00:00:00 | test
(1 row)

vastbase=# SELECT * FROM t_list PARTITION (p1);
 c1 | c2 | c3
----+----+----
(0 rows)
```

- 更新数据

```
vastbase=# update t_list set c2='2019-10-12' where c1=1;
UPDATE 1
vastbase=# SELECT * FROM t_list PARTITION (p2);
 c1 | c2 | c3
----+----+----
(0 rows)
vastbase=# SELECT * FROM t_list PARTITION (p1);
 c1 |        c2         |  c3
----+---------------------+------
  1 | 2019-10-12 00:00:00 | test
(1 row)
```

3.8.8.1.支持 default 分区

功能描述

针对 Range、List 和 Hash 类型的分区表，支持定义和使用 default 分区。

示例

```
create table t_partition_list1(col number,name varchar2(20))
partition by list(col)(
partition t_list_p1 values(1,3,5,7,9) ,
partition t_list_p2 values(2,4,6,8,10) ,
partition t_list_default values(default));
insert into t_partition_list1 values(1,'t_list_p1');
insert into t_partition_list1 values(2,'t_list_p2');
insert into t_partition_list1 values(31,'t_list_default');

select * from t_partition_list1 partition(t_list_p1);
select * from t_partition_list1 partition(t_list_p2);
select * from t_partition_list1 partition(t_list_default);
```

3.9. 查看系统表

除了创建的表以外，数据库还包含很多系统表。这些系统表包含 Vastbase 安装信息以及 Vastbase 上运行的各种查询和进程的信息。可以通过查询系统表来收集有关数据库的信息。

“14 系统表和系统视图”中每个表的说明指出了表是对所有用户可见还是只对初始化用户可见。必须以初始化用户身份登录才能查询只对初始化用户可见的表。

Vastbase 提供了以下类型的系统表和视图：

- 继承自 PG 的系统表和视图

  这类系统表和视图具有 PG 前缀。

- Vastbase 新增的系统表和视图

  这类系统表和视图具有 GS 前缀。

查看数据库中包含的表

例如，在 PG_TABLES 系统表中查看 public schema 中包含的所有表。

```
SELECT distinct(tablename) FROM pg_tables WHERE SCHEMANAME = 'public';
```

结果类似如下这样：

```
    tablename
-------------------
 err_hr_staffs
 test
 err_hr_staffs_ft3
 web_returns_p1
 mig_seq_table
 films4
(6 rows)
```

查看数据库用户

通过 PG_USER 可以查看数据库中所有用户的列表，还可以查看用户 ID（USESYSID）和用户权限。

```
SELECT * FROM pg_user;
           usename            | usesysid | usecreatedb | usesuper | usecatupd | userepl |  passwd  |
valbegin | valuntil |   respool
   | parent | spacelimit | useconfig
-----------------------------------+----------+-------------+----------+-----------+---------+---
-------+----------+----------+----------
----+--------+------------+-----------
dfc22b86afbd9a745668c3ecd0f15ec18 |    17107 | f           | f        | f         | f       | ******** |
|         | default_p
ool |     0 |          |
guest                             |    17103 | f           | f        | f         | f       | ******** |         |
| default_p
ool |     0 |          |
vastbase                          |       10 | t           | t        | t         | t       | ******** |
|         | default_p
ool |     0 |          |
vastbase                          |    16404 | f           | f        | f         | f       | ******** |
|         | default_p
ool |     0 |          |
lily                              |    16482 | f           | f        | f         | f       | ******** |         |
| default_p
ool |     0 |          |
jack                              |    16478 | f           | f        | f         | f       | ******** |         |
| default_p
ool |     0 |          |


(6 rows)
```

查看和停止正在运行的查询语句

通过视图 14.3.40PG_STAT_ACTIVITY 可以查看正在运行的查询语句。方法如下：

步骤 1 设置参数 track_activities 为 on。

```
SET track_activities = on;
```

当此参数为 on 时，数据库系统才会收集当前活动查询的运行信息。

步骤 2 查看正在运行的查询语句。以查看正在运行的查询语句所连接的数据库名、执行查询的用户、查询状态及查询对应的 PID 为例：

```
SELECT datname, usename, state,pid FROM pg_stat_activity;
 datname  | usename | state  |      pid
----------+---------+--------+-----------------
 vastbase | Ruby    | active | 140298793514752
 vastbase | Ruby    | active | 140298718004992
 vastbase | Ruby    | idle   | 140298650908416
 vastbase | Ruby    | idle   | 140298625742592
 vastbase | vastbase | active | 140298575406848
(5 rows)
```

如果 state 字段显示为 idle，则表明此连接处于空闲，等待用户输入命令。

如果仅需要查看非空闲的查询语句，则使用如下命令查看：

```
SELECT datname, usename, state FROM pg_stat_activity WHERE state != 'idle';
```

步骤 3 若需要取消运行时间过长的查询，通过 PG_TERMINATE_BACKEND 函数，根据线程 ID 结束会话。

```
SELECT PG_TERMINATE_BACKEND(139834759993104);
```

显示类似如下信息，表示结束会话成功。

```
PG_TERMINATE_BACKEND
----------------------
 t
(1 row)
```

显示类似如下信息，表示用户执行了结束当前会话的操作。

```
FATAL:  terminating connection due to administrator command
FATAL:  terminating connection due to administrator command
```

说明

vsql 客户端使用 PG_TERMINATE_BACKEND 函数结束当前会话后台线程时，客户端不会退出而是自动重连。即还会返回“The connection to the server was lost. Attempting reset: Succeeded.”

FATAL: terminating connection due to administrator command
FATAL: terminating connection due to administrator command
The connection to the server was lost. Attempting reset: Succeeded.

3.10. 配置数据库参数

功能描述

通过 SQL 语句直接修改数据库配置参数。详细用法请参考 10.2 重设参数。

示例

```
alter system set max_connections=300;
alter system set max_connections=default;
```

3.11. 其他操作

3.11.1. 创建和管理 schema

背景信息

schema 又称作模式。通过管理 schema，允许多个用户使用同一数据库而不相互干扰，可以将数据库对象组织成易于管理的逻辑组，同时便于将第三方应用添加到相应的 schema 下而不引起冲突。 管理 schema 包括：创建 schema、使用 schema、删除 schema、设置 schema 的搜索路径以及 schema 的权限控制。

注意事项

- Vastbase 包含一个或多个已命名数据库。用户和用户组在 Vastbase 范围内是共享的，但是其数据并不共享。任何与服务器连接的用户都只能访问连接请求里声明的那个数据库。
- 一个数据库可以包含一个或多个已命名的 schema，schema 又包含表及其他数据库对象，包括数据类型、函数、操作符等。同一对象名可以在不同的 schema 中使用而不会引起冲突。例如，schema1 和 schema2 都可以包含一个名为 mytable 的表。
- 和数据库不同，schema 不是严格分离的。用户根据其对 schema 的权限，可以访问所连接数据库的 schema 中的对象。进行 schema 权限管理首先需要对数据库的权限控制进行了解。
- 不能创建以 PG_为前缀的 schema 名，该类 schema 为数据库系统预留的。
- 在每次创建新用户时，系统会在当前登录的数据库中为新用户创建一个同名 Schema。对于其他数据库，若需要同名 Schema，则需要用户手动创建。
- 通过未修饰的表名(名称中只含有表名,没有“schema 名”)引用表时,系统会通过 search_path（搜索路径）来判断该表是哪个 schema 下的表。pg_temp 和 pg_catalog 始终会作为搜索路径顺序中的前两位，无论二者是否出现在 search_path 中，或者出现在 search_path 中的任何位置。search_path（搜索路径）是一个 schema 名列表，在其中找到的第一个表就是目标表，如果没有找到则报错。（某个表即使存在，如果它的 schema 不在 search_path 中，依然会查找失败）在搜索路径中的第一个 schema 叫做"当前 schema"。它是搜索时查询的第一个 schema，同时在没有声明 schema 名时，新创建的数据库对象会默认存放在该 schema 下。
- 每个数据库都包含一个 pg_catalog schema，它包含系统表和所有内置数据类型、函数、操作符。pg_catalog 是搜索路径中的一部分，始终在临时表所属的模式后面，并在 search_path 中所有模式的前面，即具有第二搜索优先级。这样确保可以搜索到数据库内置对象。如果用户需要使用和系统内置对象重名的自定义对象时，可以在操作自定义对象时带上自己的模式。

操作步骤

- 创建管理用户及权限 schema
  - 执行如下命令来创建一个 schema。

```
vastbase=# CREATE SCHEMA myschema;
```

当结果显示为如下信息，则表示成功创建一个名为 myschema 的 schema。

```
CREATE SCHEMA
```

如果需要在模式中创建或者访问对象，其完整的对象名称由模式名称和具体的对象名称组成。中间由符号“.”隔开。例如：myschema.table。

  - 执行如下命令在创建 schema 时指定 owner。

```
vastbase=# CREATE SCHEMA myschema AUTHORIZATION vastbase;
```

当结果显示为如下信息，则表示成功创建一个属于 vastbase 用户，名为 myschema 的 schema。

```
CREATE SCHEMA
```

❖ 使用 schema

在特定 schema 下创建对象或者访问特定 schema 下的对象，需要使用有 schema 修饰的对象名。该名称包含 schema 名以及对象名，他们之间用“.”号分开。

– 执行如下命令在 myschema 下创建 mytable 表。

```
vastbase=# CREATE TABLE myschema.mytable(id int, name varchar(20));
CREATE TABLE
```

如果在数据库中指定对象的位置，就需要使用有 schema 修饰的对象名称。

– 执行如下命令查询 myschema 下 mytable 表的所有数据。

```
vastbase=# SELECT * FROM myschema.mytable;
id | name
----+------
(0 rows)
```

❖ schema 的搜索路径

可以设置 search_path 配置参数指定寻找对象可用 schema 的顺序。在搜索路径列出的第一个 schema 会变成默认的 schema。如果在创建对象时不指定 schema，则会创建在默认的 schema 中。

– 执行如下命令查看搜索路径。

```
vastbase=# SHOW SEARCH_PATH;
 search_path
----------------
 "$user",public
(1 row)
```

– 执行如下命令将搜索路径设置为 myschema、public，首先搜索 myschema。

```
vastbase=# SET SEARCH_PATH TO myschema, public;
SET
```

❖ schema 的权限控制

默认情况下，用户只能访问属于自己的 schema 中的数据库对象。如果需要访问其他 schema 的对象，则被访问的对象的 schema 的所有者应该赋予用户对该 schema 的 USAGE 权限。

通过将模式的 CREATE 权限授予某用户，被授权用户就可以在此模式中创建对象。注意默认情况下，所有角色都拥有在 public 模式上的 USAGE 权限，但是普通用户没有在 public 模式上的 CREATE 权限。普通用户能够连接到一个指定数据库并在它的 public 模式中创建对象是不安全的，如果普通用户具有在 public 模式上的 CREATE 权限则建议通过如下语句撤销该权限。

– 撤销 PUBLIC 在 public 模式下创建对象的权限，下面语句中第一个“public”是模式，第二个“PUBLIC”指的是所有角色。

```
vastbase=# REVOKE CREATE ON SCHEMA public FROM PUBLIC;
REVOKE
```

– 使用以下命令查看现有的 schema：

```
vastbase=# SELECT current_schema();
 current_schema
----------------
```

```
 myschema
(1 row)
```

- 执行如下命令创建用户 jack，并将 myschema 的 usage 权限赋给用户 jack。

```
vastbase=# CREATE USER jack IDENTIFIED BY 'Bigdata@123';
CREATE ROLE
vastbase=# GRANT USAGE ON schema myschema TO jack;
GRANT
```

- 将用户 jack 对于 myschema 的 usage 权限收回。

```
vastbase=# REVOKE USAGE ON schema myschema FROM jack;
REVOKE
```

❖ 删除 schema

- 当 schema 为空时，即该 schema 下没有数据库对象，使用 DROP SCHEMA 命令进行删除。例如删除名为 nullschema 的空 schema。

```
vastbase=# DROP SCHEMA IF EXISTS nullschema;
DROP SCHEMA
```

- 当 schema 非空时，如果要删除一个 schema 及其包含的所有对象，需要使用 CASCADE 关键字。例如删除 myschema 及该 schema 下的所有对象。

```
vastbase=# DROP SCHEMA myschema CASCADE;
DROP SCHEMA
```

- 执行如下命令删除用户 jack。

```
vastbase=# DROP USER jack;
DROP ROLE
```

3.11.2. 创建和管理索引

背景信息

索引可以提高数据的访问速度，但同时也增加了插入、更新和删除操作的处理时间。所以是否要为表增加索引，索引建立在哪些字段上，是创建索引前必须要考虑的问题。需要分析应用程序的业务处理、数据使用、经常被用作查询的条件或者被要求排序的字段来确定是否建立索引。

索引建立在数据库表中的某些列上。因此，在创建索引时，应该仔细考虑在哪些列上创建索引。

❖ 在经常需要搜索查询的列上创建索引，可以加快搜索的速度。

❖ 在作为主键的列上创建索引，强制该列的唯一性和组织表中数据的排列结构。

❖ 在经常需要根据范围进行搜索的列上创建索引，因为索引已经排序，其指定的范围是连续的。

❖ 在经常需要排序的列上创建索引，因为索引已经排序，这样查询可以利用索引的排序，加快排序查询时间。

❖ 在经常使用 WHERE 子句的列上创建索引，加快条件的判断速度。

❖ 为经常出现在关键字 ORDER BY、GROUP BY、DISTINCT 后面的字段建立索引。

说明

- 索引创建成功后，系统会自动判断何时引用索引。当系统认为使用索引比顺序扫描更快时，就会使用索引。
- 索引创建成功后，必须和表保持同步以保证能够准确地找到新数据，这样就增加了数据操作的负荷。因此请定期删除无用的索引。

操作步骤

创建分区表的步骤请参考 3.8.8 创建和管理分区表

- 创建索引
  - 创建分区表索引 tpcds_web_returns_p2_index1，不指定索引分区的名称。

```
vastbase=# CREATE INDEX tpcds_web_returns_p2_index1 ON tpcds.web_returns_p2
(ca_address_id) LOCAL;
```

当结果显示为如下信息，则表示创建成功。

```
CREATE INDEX
```

  - 创建分区索引 tpcds_web_returns_p2_index2，并指定索引分区的名称。

```
vastbase=# CREATE INDEX tpcds web returns p2 index2 ON tpcds.web returns p2
(ca address sk) LOCAL
(
   PARTITION web returns p2 P1 index,
   PARTITION web returns p2 P2 index TABLESPACE example3,
   PARTITION web returns p2 P3 index TABLESPACE example4,
   PARTITION web returns p2 P4 index,
   PARTITION web_returns_p2_P5_index,
   PARTITION web_returns_p2_P6_index,
   PARTITION web_returns_p2_P7_index,
   PARTITION web_returns_p2_P8_index
) TABLESPACE example2;
```

当结果显示为如下信息，则表示创建成功。

```
CREATE INDEX
```

- 修改索引分区的表空间
  - 修改索引分区 web_returns_p2_P2_index 的表空间为 example1。

```
vastbase=# ALTER INDEX tpcds.tpcds_web_returns_p2_index2 MOVE PARTITION
web_returns_p2_P2_index TABLESPACE example1;
```

当结果显示为如下信息，则表示修改成功。

```
ALTER INDEX
```

  - 修改索引分区 web_returns_p2_P3_index 的表空间为 example2。

```
vastbase=# ALTER INDEX tpcds.tpcds_web_returns_p2_index2 MOVE PARTITION
web_returns_p2_P3_index TABLESPACE example2;
```

当结果显示为如下信息，则表示修改成功。

```
ALTER INDEX
```

- 重命名索引分区

执行如下命令对索引分区 *web_returns_p2_P8_index* 重命名 *web_returns_p2_P8_index*_new。

```
vastbase=# ALTER INDEX tpcds.tpcds_web_returns_p2_index2 RENAME PARTITION
web_returns_p2_P8_index TO web_returns_p2_P8_index_new;
```

当结果显示为如下信息，则表示重命名成功。

```
ALTER INDEX
```

- 查询索引

- 执行如下命令查询系统和用户定义的所有索引。

```
vastbase=# SELECT RELNAME FROM PG_CLASS WHERE RELKIND='i';
```

- 执行如下命令查询指定索引的信息。

```
vastbase=# \di+ tpcds.tpcds_web_returns_p2_index2
```

❖ 删除索引

```
vastbase=# DROP INDEX tpcds.tpcds_web_returns_p2_index1;
vastbase=# DROP INDEX tpcds.tpcds_web_returns_p2_index2;
```

当结果显示为如下信息，则表示删除成功。

```
DROP INDEX
```

Vastbase 支持 4 种创建索引的方式请参见表 3-4

表 3-4.

索引方式	描述
唯一索引	可用于约束索引属性值的唯一性，或者属性组合值的唯一性。如果一个表声明了唯一约束或者主键，则 Vastbase 自动在组成主键或唯一约束的字段上创建唯一索引（可能是多字段索引），以实现这些约束。目前，Vastbase 只有 B-Tree 可以创建唯一索引。
多字段索引	一个索引可以定义在表中的多个属性上。目前，Vastbase 中的 B-Tree 支持多字段索引，且最多可在 32 个字段上创建索引。
部分索引	建立在一个表的子集上的索引，这种索引方式只包含满足条件表达式的元组。
表达式索引	索引建立在一个函数或者从表中一个或多个属性计算出来的表达式上。表达式索引只有在查询时使用与创建时相同的表达式才会起作用。

❖ 创建一个普通表。

```
vastbase=# CREATE TABLE tpcds.customer_address_bak AS TABLE tpcds.customer_address;
INSERT 0 0
```

❖ 创建普通索引

如果对于 tpcds.customer_address_bak 表，需要经常进行以下查询。

```
vastbase=# SELECT ca_address_sk FROM tpcds.customer_address_bak WHERE ca_address_sk=14888;
```

通常，数据库系统需要逐行扫描整个 tpcds.customer_address_bak 表以寻找所有匹配的元组。如果表 tpcds.customer_address_bak 的规模很大，但满足 WHERE 条件的只有少数几个（可能是零个或一个），则这种顺序扫描的性能就比较差。如果让数据库系统在 ca_address_sk 属性上维护一个索引，用于快速定位匹配的元组，则数据库系统只需要在搜索树上查找少数的几层就可以找到匹配的元组，这将会大大提高数据查询的性能。同样，在数据库中进行更新和删除操作时，索引也可以提升这些操作的性能。

使用以下命令创建索引。

```
vastbase=# CREATE INDEX index_wr_returned_date_sk ON tpcds.customer_address_bak
(ca_address_sk);
CREATE INDEX
```

- 创建多字段索引

假如用户需要经常查询表 tpcds.customer_address_bak 中 ca_address_sk 是 5050，且 ca_street_number 小于 1000 的记录，使用以下命令进行查询。

```
vastbase=# SELECT ca_address_sk,ca_address_id FROM tpcds.customer_address_bak WHERE
ca_address_sk = 5050 AND ca_street_number < 1000;
```

使用以下命令在字段 ca_address_sk 和 ca_street_number 上定义一个多字段索引。

```
vastbase=# CREATE INDEX more_column_index ON
tpcds.customer_address_bak(ca_address_sk ,ca_street_number );
CREATE INDEX
```

- 创建部分索引

如果只需要查询 ca_address_sk 为 5050 的记录，可以创建部分索引来提升查询效率。

```
vastbase=# CREATE INDEX part_index ON tpcds.customer_address_bak(ca_address_sk) WHERE
ca_address_sk = 5050;
CREATE INDEX
```

- 创建表达式索引

假如经常需要查询 ca_street_number 小于 1000 的信息，执行如下命令进行查询。

```
vastbase=# SELECT * FROM tpcds.customer_address_bak WHERE trunc(ca_street_number) < 1000;
```

可以为上面的查询创建表达式索引：

```
vastbase=# CREATE INDEX para_index ON tpcds.customer_address_bak (trunc(ca_street_number));
CREATE INDEX
```

- 删除 tpcds.customer_address_bak 表。

```
vastbase=# DROP TABLE tpcds.customer_address_bak;
DROP TABLE
```

3.11.3. 创建和管理视图

背景信息

当用户对数据库中的一张或者多张表的某些字段的组合感兴趣，而又不想每次键入这些查询时，用户就可以定义一个视图，以便解决这个问题。

视图与基本表不同，不是物理上实际存在的，是一个虚表。数据库中仅存放视图的定义，而不存放视图对应的数据，这些数据仍存放在原来的基本表中。若基本表中的数据发生变化，从视图中查询出的数据也随之改变。从这个意义上讲，视图就像一个窗口，透过它可以看到数据库中用户感兴趣的数据及变化。视图每次被引用的时候都会运行一次。

管理视图

- 创建视图

  执行如下命令创建新视图 MyView。

```
vastbase=# CREATE OR REPLACE VIEW MyView AS SELECT * FROM tpcds.customer_address WHERE
trunc(ca_street_number) < 1000;
CREATE VIEW
```

  说明

  CREATE VIEW 中的 OR REPLACE 可有可无，当存在 OR REPLACE 时，表示若以前存在该视图就进行替换。

- 查询视图

  执行如下命令查询 MyView 视图。

```
vastbase=# SELECT * FROM MyView;
```

- 查看某视图的具体信息

  执行如下命令查询 dba_users 视图的详细信息。

```
vastbase=# \d+ dba_users
                 View "pg_catalog.dba_users"
  Column  |       Type         | Modifiers | Storage  | Description
----------+--------------------+-----------+----------+-------------
 username | varchar(64) |          | extended |
View definition:
 SELECT pg_authid.rolename::varchar(64) AS username
   FROM pg_authid;
```

- 更新视图数据

```
vastbase=# create table t_view(id int,name text);
vastbase=# insert into t_view values(1,'test1');
vastbase=# insert into t_view values(2,'test2');
vastbase=# insert into t_view values(3,'test3');
vastbase=# CREATE VIEW t_view_v AS SELECT * FROM t_view WHERE id > 2;
vastbase=# update t_view_v set name='test66' where id =3;
```

- 删除视图

  执行如下命令删除 MyView 视图。

```
vastbase=# DROP VIEW MyView;
DROP VIEW
```

3.11.4. 创建和管理序列

背景信息

序列 Sequence 是用来产生唯一整数的数据库对象。序列的值是按照一定规则自增的整数。因为自增所以不重复，因此说 Sequence 具有唯一标识性。这也是 Sequence 常被用作主键的原因。

通过序列使某字段成为唯一标识符的方法有两种：

- 一种是声明字段的类型为表 11-4，由数据库在后台自动创建一个对应的 Sequence。

❖ 另一种是使用 11.16.46CREATE SEQUENCE 自定义一个新的 Sequence，然后将 nextval('sequence_name')函数读取的序列值，指定为某一字段的默认值，这样该字段就可以作为唯一标识符。

操作步骤

方法一： 声明字段类型为序列整型来定义标识符字段。例如：

```
vastbase=# CREATE TABLE T1
(
   id   serial,
   name  text
);
```

当结果显示为如下信息，则表示创建成功。

```
NOTICE:  CREATE TABLE will create implicit sequence "t1_id_seq" for serial column "t1.id"
CREATE TABLE
```

方法二： 创建序列，并通过 nextval('sequence_name')函数指定为某一字段的默认值。

1. 创建序列

```
vastbase=# CREATE SEQUENCE seq1 cache 100;
```

当结果显示为如下信息，则表示创建成功。

```
CREATE SEQUENCE
```

2. 指定为某一字段的默认值，使该字段具有唯一标识属性。

```
vastbase=# CREATE TABLE T2
(
   id   int not null default nextval('seq1'),
   name text
);
```

当结果显示为如下信息，则表示默认值指定成功。

```
CREATE TABLE
```

3. 指定序列与列的归属关系。

将序列和一个表的指定字段进行关联。这样，在删除那个字段或其所在表的时候会自动删除已关联的序列。

```
vastbase=# ALTER SEQUENCE seq1 OWNED BY T2.id;
```

当结果显示为如下信息，则表示指定成功。

```
ALTER SEQUENCE
```

说明

除了为序列指定了 cache，方法二所实现的功能基本与方法一类似。但是一旦定义 cache，序列将会产生空洞(序列值为不连贯的数值，如：1.4.5)，并且不能保序。另外为某序列指定从属列后，该列删除，对应的 sequence 也会被删除。 虽然数据库并不限制序列只能为一列产生默认值，但最好不要多列共用同一个序列。

当前版本只支持在定义表的时候指定自增列，或者指定某列的默认值为 nextval('seqname')， 不支持在已有表中增加自增列或者增加默认值为 nextval('seqname')的列。

3.11.5. 创建和管理约束

数据类型是一种限制可以存储在表中的数据类型的方法。然而，对于许多应用，它们提供的约束太粗糙。例如，包含产品价格的列应该只接受正值。但是没有标准数据类型只接受正数。另一个问题是您可能希望相对于其他列或行约束列数据。例如，在包含产品信息的表中，每个产品编号只应有一行。

为此，SQL 允许您定义列和表的约束。约束使您可以根据需要尽可能多地控制表中的数据。如果用户尝试将数据存储在违反约束的列中，则会引发错误。即使值来自默认值定义，这也适用。

3.11.5.1. 检查约束

检查约束是最通用的约束类型。它允许您指定某列中的值必须满足布尔值（真值）表达。例如，要求正面的产品价格，您可以使用：

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

如您所见，约束定义位于数据类型之后，就像默认值定义一样。可以按任何顺序列出默认值和约束。检查约束由关键字 CHECK 和括号中的表达式组成。检查约束表达式应该涉及这样约束的列，否则约束不会有太多意义。

您还可以为约束指定单独的名称。这澄清了错误消息，并允许您在需要更改时引用约束。语法是：

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

因此，要指定命名约束，请使用关键字 CONSTRAINT ，后跟标识符，后跟约束定义。 （如果没有以这种方式指定约束名称，系统会为您选择一个名称。）

检查约束也可以引用多个列。假设您存储正常价格和折扣价格，并且您希望确保折扣价格低于正常价格：

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

前两个约束应该看起来很熟悉。第三个使用新语法。它未附加到特定列，而是显示为逗号分隔列列表中的单独项。列定义和这些约束定义可以按混合顺序列出。

我们说前两个约束是列约束，而第三个约束是表约束，因为它与任何一个列定义分开编写。列约束也可以写为表约束，而反向不一定是可能的，因为列约束应该仅引用它所附加的列。 （Vastbase 不强制执行该规则，但如果您希望表定义与其他数据库系统一起使用，则应遵循该规则。）上述示例也可以写为：

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

甚至：

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

可以使用与列约束相同的方式将名称分配给表约束：

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

应当注意，如果检查表达式的计算结果为真或空值，则满足检查约束。由于如果任何操作数为 null，大多数表达式将计算为空值，因此它们不会阻止受约束列中的空值。要确保列不包含空值，可以使用下一节中描述的非空约束。

3.11.5.2. 非空约束

非空约束只是指定列不能采用空值。语法示例：

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

非空约束始终写为列约束。非空约束在功能上等同于创建检查约束 CHECK (column_name IS NOT NULL) ，但在 Vastbase 中创建显式非空约束更有效。缺点是不能为以这种方式创建的非空约束提供显式名称。

当然，列可以有多个约束。

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

NOT NULL 约束具有反转： NULL 约束。这并不意味着列必须为 NULL ，这肯定是无用的。相反，这只是选择列可能为 NULL 的默认行为。 NULL 约束在 SQL 标准中不存在，不应在便携式应用程序中使用。（它只是添加到 Vastbase 以与其他一些数据库系统兼容。）然而，有些用户喜欢它，因为它可以很容易地在脚本文件中切换约束。

例如，可以在需要的位置插入 NOT 关键字。

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

说明

在大多数数据库设计中，大多数列应标记为非空。

3.11.5.3. 唯一约束

唯一约束可确保列或列组中包含的数据在表中的所有行中都是唯一的。语法是：

声明为列约束：

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

声明为表约束：

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

要为一组列定义唯一约束，请将其写为表约束，并使用逗号分隔列名：

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

这指定所指示列中的值的组合在整个表中是唯一的，但是任何一列都不需要（通常不是）唯一的。

可以为唯一约束指定自己的名称：

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

添加唯一约束将自动在约束中列出的列或列组上创建唯一的 B 树索引。仅覆盖某些行的唯一性限制不能写为唯一约束，但可以通过创建唯一的 partial index 来强制执行此类限制。

通常，如果表中有多行，其中包含的所有列的值都违反了唯一约束约束是平等的。但是，在此比较中，两个空值永远不会被视为相等。这意味着即使存在唯一约束，也可以在至少一个约束列中存储包含空值的重复行。此行为符合 SQL 标准，但可能其他 SQL 数据库不遵循此规则。因此在开发可移植的应用程序时要小心。

3.11.5.4. 主键

主键约束表示列或列组可用作表中行的唯一标识符。这要求值既是唯一的又不是 null。因此，以下两个表定义接受相同的数据：

```
vastbase=# CREATE TABLE products ( product_no integer UNIQUE NOT NULL,  name text,  price numeric );
vastbase=# CREATE TABLE products (product_no integer PRIMARY KEY,  name text,  price numeric );
```

主键可以跨越多个列;语法类似于唯一约束：

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

添加主键将自动在主键中列出的列或列组上创建唯一的 B 树索引，并强制将列标记为 NOT NULL 。

一个表最多只能有一个主键。（可以有任意数量的唯一和非空约束，它们在功能上几乎是相同的，但只有一个可以被识别为主键。）关系数据库理论规定每个表必须有一个主键。 Vastbase 不强制执行此规则，但通常最好遵循它。

主键对于客户端应用程序很有用。例如，允许修改行值的 GUI 应用程序可能需要知道表的主键才能唯一地标识行。如果已经声明了主键，数据库系统还可以使用各种方式使用主键。例如，主键定义引用其表的外键的默认目标列。

3.11.5.5. 外键

外键约束指定列（或列组）中的值必须与另一个表的某行中出现的值匹配。这维持了两个相关表之间的引用完整性。

假设已经创建一个产品表：

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

再假设还有一个存储这些产品订单的表。我们希望确保订单表仅包含实际存在的产品订单。所以我们在 orders 表中定义了一个引用 products 表的外键约束：

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

如果没有创建 products 表，那么无法创建 orders 表。

我们说在这种情况下，orders 表是引用表，products 表是被引用表。同样，有引用和引用列。

上述命令可以缩短为：

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

因为在没有列列表的情况下，引用表的主键用作引用列。

外键还可以约束和引用一组列。像往常一样，它需要以表约束形式编写。

这是一个人为的语法示例：

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

当然，受约束列的数量和类型需要与引用列的数量和类型相匹配。

可以为外键约束指定自己的名称。

一个表可以有多个外键约束。这用于实现表之间的多对多关系。假设有关于产品和订单的表格，但现在希望允许一个订单包含可能的许多产品（上述结构不允许）。那么可以使用此表结构：

```
vastbase=# CREATE TABLE products ( product_no integer PRIMARY KEY, name text, price numeric );
vastbase=# CREATE TABLE orders ( order_id integer PRIMARY KEY, shipping_address text,    ... );
vastbase=# CREATE TABLE order_items (product_no integer REFERENCES products, order_id integer
REFERENCES orders,  quantity integer,   PRIMARY KEY (product_no, order_id) );
```

请注意，主键与最后一个表中的外键重叠。

我们知道外键不允许创建与任何产品无关的订单。如果在创建引用它的订单后删除了产品，SQL 支持处理它。

为了说明这一点，让我们在上面的多对多关系示例中实现以下策略：当有人想要删除仍然被订单引用的产品时（通过 order_items ），我们不允许它。如果有人删除了订单，订单商品也会被删除：

```
vastbase=# CREATE TABLE products (    product_no integer PRIMARY KEY,    name text,    price
numeric );  CREATE TABLE orders (    order_id integer PRIMARY KEY,    shipping_address text,    ... );
CREATE TABLE order_items (    product_no integer REFERENCES products ON DELETE RESTRICT,    order_id
integer REFERENCES orders ON DELETE CASCADE,    quantity integer,    PRIMARY KEY (product_no,
order_id) );
```

限制和级联删除是两种最常见的选项。 RESTRICT 防止删除引用的行。 NO ACTION 表示如果在检查约束时仍存在任何引用行，则会引发错误；如果您未指定任何内容，则这是默认行为。 （这两个选项之间的本质区别在于 NO ACTION 允许将检查推迟到事务的后期，而 RESTRICT 则不会。） CASCADE 指定当删除引用的行时，引用它的行应自动删除为好。还有另外两个选项：SET NULL 和 SET DEFAULT 。当删除引用的行时，这些引用将引用行中的引用列分别设置为空值或其默认值。请注意，这些不能免除您观察任何约束。例如，如果操作指定 SET DEFAULT 但默认值不满足外键约束，则操作将失败。

类似于 ON DELETE ，还有 ON UPDATE ，当引用的列被更改（更新）时会调用它。可能的行动是相同的。在这种情况下， CASCADE 表示应将引用列的更新值复制到引用行中。

通常，如果引用行的任何引用列为空，则引用行不必满足外键约束。如果将 MATCH FULL 添加到外键声明中，则仅当所有引用列都为空时，引用行才会转义满足约束（因此，保证空值和非空值的混合使 MATCH FULL 约束失败）。如果您不希望引用行以避免满足外键约束，请将引用列声明为 NOT NULL 。

外键必须引用作为主键或形成唯一约束的列。这意味着引用的列总是有索引（主键或唯一约束下面的索引）;因此，检查引用行是否匹配将是有效的。由于引用表中的行的 DELETE 或引用列的 UPDATE 将需要扫描引用表以查找与旧值匹配的行，因此通常也应该对引用列 Build 索引。因为并不总是需要这个，并且有很多关于如何索引的选择，所以外键约束的声明不会自动在引用列上创建索引。

另请参阅 CREATE TABLE 的参考文档中的外键约束语法的说明。

3.11.5.6. 约束启用禁用

约束可以被启用禁用，通过 SQL 可以修改表上的约束的状态，约束禁用后该约束不生效，启用约束时会校验表中当前的数据是否满足约束条件，若不满足则约束无法启用。语法是：

```
启用约束语法
vastbase=# ALTER TABLE table_name MODIFY CONSTRAINT constraint_name ENABLE;
禁用约束语法
vastbase=# ALTER TABLE table_name MODIFY CONSTRAINT constraint_name DISABLE;
```

3.11.6. vb_basebackup 备份为 tar 格式

功能描述

使 vb_basebackup 支持将数据库备份为 tar 格式，并可进一步进行 gzip 压缩。

参数说明

- -D：备份文件输出的目录，必选项
- -F：指定输出格式：plain | tar
- -z：将输出的 tar 格式文件进一步压缩为.gz 格式

示例

```
vb_basebackup -D /home/vastbase/data/bak -h 127.0.0.1 -p 5432 -Ftar -P -v -z -Xf
```

3.11.7. vb_basebackup 备份表空间

功能描述

使 vb_basebackup 支持表空间的备份恢复。

3.11.7.1. 备份

vb_basebackup 有两种输出格式，分别为 tar 格式与 plain 格式 ，使这两种格式都支持表空间的备份：

➢ tar 格式备份支持

将表空间备份文件输出在-D 参数指定的目录下，输出文件名以表空间，命令用法如下：

```
vb_basebackup -D /home/vbtest/data/bak -h 127.0.0.1 -p 5432 -Ftar -P -v -Xf
```

参数说明：

-D :备份文件输出路径，必选项

-F： 指定输出格式

➢ plain 格式备份支持

添加-T 参数指定表空间目录映射，将表空间备份到指定目录，命令用法如下：

```
vb_basebackup -D /home/vbtest/oribackuptest/base   \
-T/home/vbtest/data/tbs_mydb=/home/vbtest/oribackuptest/tbs_mydb  \
-h 127.0.0.1 -p 5432 -Fplain -P -v
```

参数说明：

-D：主数据目录备份文件输出路径，必选项

-T： “=”号左侧为需备份的表空间路径，右侧为表空间备份输出路径；多个表空间可以多次使用-T 参数。

3.11.7.2. 恢复

使 Vastbase 支持，恢复时，不止恢复主数据目录也恢复表空间。用法如下：

➢ tar 格式，将表空间备份文件解压到原表空间位置

➢ plain 格式，直接使用，不用修改表空间

4. 应用程序开发教程

4.1. 开发规范

如果用户在 APP 的开发中，使用了连接池机制，那么需要遵循如下规范：

- 如果在连接中设置了 GUC 参数，那么在将连接归还连接池之前，必须使用“SET SESSION AUTHORIZATION DEFAULT;RESET ALL;”将连接的状态清空。
- 如果使用了临时表，那么在将连接归还连接池之前，必须将临时表删除。

否则，连接池里面的连接就是有状态的，会对用户后续使用连接池进行操作的正确性带来影响。

4.2. JDBC

Vastbase 数据库通过提供 JDBC 驱动为使用 JAVA 语言的应用程序提供访问 Vastbase 的接口。

4.2.1. Vastbase JDBC 版本

如果要使用 Vastbase JDBC 驱动访问 Vastbase，建议使用对应版本的 Vastbase JDBC 驱动。驱动名称：vastbase-$version.jar，例如 vastbase-2.2.jar。下文中统称为 vastbase.jar

4.2.2. Vastbase JDBC 与 JDK 的兼容性

Vastbase JDBC 驱动兼容 JDK1.8 及以上版本。

4.2.3. Vastbase JDBC 主要类与接口

Vastbase JDBC 驱动由具有建立和管理与 Vastbase 数据库连接、执行 SQL 语句和对结果集进行储存管理的若干功能类组成。

- Driver 类(org.postgresql.Driver)

当注册驱动的时候或配置软件以使用 Vastbase JDBC 驱动的时候，应该使用这个类名。

示例：

```
Class.forName("org.postgresql.Driver");
```

- DriverManager 类(java.sql.DriverManager)

跟踪可用的驱动程序，在数据库与相应的驱动程序之间建立连接。在应用服务器外使用 JDBC 时，DriverManager 类管理连接的确立。要告诉 DriverManager 应该与哪个 JDBC 驱动进行连接，最简单的方法就是使用实现了接口 java.sql.Driver 的类的 Class.forName()方法。在 Vastbase JDBC 驱动中，

这个类的名字叫做 org.postgresql.Driver。用这种方法，就可以在连接一个数据库时使用一个外部配置文件来给驱动提供类名和驱动参数。

- Connection 接口(org.postgresql.PGConnection)

与特定数据库的连接（会话）。在连接上下文中执行 SQL 语句并返回结果。

示例：

```
Connection con=DriverManager.getConnection("jdbc:postgresql://host:port/dbname ",
"user","password");
```

- Statement 接口(org.postgresql.PGStatement)

用于执行 SQL 语句并返回结果。

- ResultSet 接口(org.postgresql.PGResultSetMetaData)

存储执行 SQL 语句产生的结果集。示例：

```
Statement st = con.createStatement();
ResultSet rs = st.executeQuery("select c_custkey from customer");
```

4.2.4. 设置 JDBC 驱动

4.2.4.1.设置类路径

要使用驱动， 必须将驱动 jar 包含在类路径里，可以将 jar 包路径添加到 CLASSPATH 环境变量中，或者使用 java 命令行标记的方式将驱动 jar 包引入.

比如，有一个使用 Vastbase JDBC 驱动的应用安装在/usr/local/lib/myapp.jar，而 Vastbase JDBC 驱动安装在/usr/local/vastbase/share/java/vastbase.jar．可以用以下方式运行应用：

```
export CLASSPATH=/usr/local/lib/MyApp.jar:/usr/local/vastbase/share/java/vastbase.jar:.
java MyApp;
```

4.2.4.2.为 JDBC 准备数据库

因为 java 只支持使用 TCP/IP 连接，所以 Vastbase 必须配置为可接受 TCP/IP 连接。可以通过修改 listen addresses 来进行配置。同样，也需要配置主机认证文件，保证客户端程序可以访问数据库。

4.2.5. 使用 JDBC 连接数据库

4.2.5.1.导入

任何使用 JDBC 的程序都需要导入 java.sql 包：

```
import java.sql.*
```

4.2.5.2.加载驱动

在试图与数据库建立连接之前，首先需要加载驱动，加载驱动有两种方法：

- 在代码中，用 Class.forName(“org.postgresql.Driver”)方法显示加载驱动；
- 在 JVM 启动时作为参数传递，比如：java -Djdbc.drivers=org.postgresql.Driver

4.2.5.3.连接数据库

4.2.5.3.1.URL 格式

使用 Vastbase JDBC 驱动访问 Vastbase 的 URL 格式如下：

- jdbc:postgresql://host:port/database
- jdbc:postgresql://host:port/
- jdbc:postgresql://host/database
- jdbc:postgresql://host/
- jdbc:postgresql:/

URL 中各参数的含义如下：

- host

服务端的主机名，默认为 localhost,如果要指定 IPV6 的地址 ，必须用方括号括起主机参数，例如：jdbc:postgresql://[::1]:5432/vastbase.

- port

服务端监听的端口，默认为 5432.

- database

数据库名称，默认连接的数据库是与用户同名的数据库。比如连接用户为 vuser,如果不指定 database 参数，则默认连接到 vuser 这个数据库。

4.2.5.3.2.获取 JDBC 连接

使用 DriverManager.getConnection()获取连接：

```
Connection connection = DriverManager.getConnection(url, username,password);
```

4.2.5.3.3.关闭 JDBC 连接

关闭连接时调用 Connection 的 close()方法即可:

```
Connection connection = DriverManager.getConnection(url, username,password);
connection.close();
```

4.2.5.4.连接参数

名称	类型	定义
user	String	连接数据库的用户
password	String	数据库用户的密码

sendBufferSize	int	在连接流上设置 SO_SNDBUF
recvBufferSize	int	在连接流上设置 SO_RCVBUF
tcpKeepAlive	boolean	启用或禁用 TCP keep-alive
socketTimeout	int	socket 读取操作的超时时间
ApplicationName	String	指定正在使用连接的应用程序名称
currentSchema	String	指定设置到 search-path 中的 schema
readOnly	boolean	将连接设置为只读模式

4.2.6. 执行 SQL 语句并处理结果

向数据库发出执行 SQL 的请求时，需要使用 Statement 或 PreparedStatement，使用 Statement 或 PrepareStatement 实例发出查询，服务端执行完成后将返回一个 ResultSet 实例，该实例包括查询的结果。

4.2.6.1.执行查询

4.2.6.1.1.使用 Statement 执行查询

```
Statement st = conn.createStatement();
ResultSet rs = st.executeQuery("SELECT * FROM mytable WHERE columnfoo = 500");
while (rs.next())
{
   System.out.print("Column 1 returned ");
   System.out.println(rs.getString(1));
}
rs.close();
```

4.2.6.1.2.使用 PreparedStatement 执行查询

```
int foovalue = 500;
PreparedStatement st = conn.prepareStatement("SELECT * FROM mytable WHERE columnfoo = ?");
st.setInt(1, foovalue);
ResultSet rs = st.executeQuery();
while (rs.next())
{
   System.out.print("Column 1 returned ");
   System.out.println(rs.getString(1));
}
rs.close();
st.close();
```

4.2.6.2.使用游标获取结果

默认情况下，驱动程序将一次性获取查询的所有结果，当某个查询的结果集很庞大时，一次获取所有结果很不合理，因此 JDBC 驱动程序提供了一种基于数据库游标的 ResultSet 方法，只获取少量的行缓存在客户端。通过重新定位游标来获取下一批数据。

基于游标的 ResultSet 不能在所有情况下使用。有许多限制会使驱动程序一次性获取整个结果集：

- 连接不能使用自动提交模式 ，因为后端会在事务结束时关闭游标，因此在自动提交模式下，事务结束后，在从游标获取数据之前，游标已经被关闭。
- Statement 必须使用 ResultSet.TYPE_FORWARD_ONLY 类型创建，这意味着不能在结果集中进行跳转或向后滚动。这也是默认值，因此不需要在代码中额外指定。
- 给定的查询语句必须是单条语句，不能是用分号连接在一起的多个语句。
- 必须设置 fetch size，且 fetch size 不能为 0.

示例：

```
// make sure autocommit is off
conn.setAutoCommit(false);
Statement st = conn.createStatement();

// Turn use of the cursor on.
st.setFetchSize(50);
ResultSet rs = st.executeQuery("SELECT * FROM mytable");
while (rs.next())
{
   System.out.print("a row was returned.");
}
rs.close();

// Turn the cursor off.
st.setFetchSize(0);
rs = st.executeQuery("SELECT * FROM mytable");
while (rs.next())
{
   System.out.print("many rows were returned.");
}
rs.close();

// Close the statement.
st.close();
```

使用 Statement 和 PreparedStatement 接口

在使用 Statement 和 PreparedStatement 时，需要注意以下问题：

- 可以多次使用一个 Statement 实例，可以在打开连接时创建 Statement 然后在连接的整个生命周期内使用它。但是必须注意在给定时间内每个 Statement 或 PreparedStatement 只能存在一个 ResultSet.
- 如果在处理 ResultSet 时需要执行查询，可以创建另一个 Statement 来执行。
- 如果使用多线程 ，并且有多个线程正在使用数据库，则每个线程必须使用单独的 Statement.
- 使用完 Statement 或 PreparedStatement 应该及时关闭。
- 在 JDBC 中，问号(?)是 PreparedStatement 的占位符，但是有一些操作符包含一个问号，为避

免 SQL 语句中的问号被当成占位符，可以使用两个问号(??)进行转义。也可以在 Statement 中使用上述转义，但是不是必须的，因为在 Statement 中单个问号也可以用作操作符。

4.2.6.3.使用 ResultSet 接口

使用 ResultSet 接口时必须考虑下面的问题：

- 在读取任何数值的时候，必须调用 next()． 如果还有结果则返回真（true）。
- 在 JDBC 规范里，对一个字段应该只访问一次． 遵循这个规则是最安全的，不过目前 Vastbase JDBC 驱动允许对一个字段访问任意次．
- 在结束对一个 ResultSet 的处理后，必须调用 close()来关闭它。
- 使用那个创建 ResultSet 的 Statement 做另一个查询请求， 当前打开的 ResultSet 实例将自动关闭．

4.2.6.4.执行更新

要更改数据(执行一个 insert，update 或者 delete)，可以使用 executeUpdate() 方法．executeUpdate()方法类似于发出 select 语句的 executeQuery()方法，不过，它不返回 ResultSet，它返回的是 insert,update 或者 delete 语句影响的行数．

4.2.6.4.1.删除

```
int foovalue = 500;
PreparedStatement st = conn.prepareStatement("DELETE FROM mytable WHERE columnfoo = ?");
st.setInt(1, foovalue);
int rowsDeleted = st.executeUpdate();
System.out.println(rowsDeleted + " rows deleted");
st.close();
```

4.2.6.4.2.更新

```
int foovalue = 500;
PreparedStatement st = conn.prepareStatement("UPDATE mytable  SET columninfo=' updated'  WHERE
columnfoo = ?");
st.setInt(1, foovalue);
int rowsUpdated = st.executeUpdate();
System.out.println(rowsUpdated + " rows updated");
st.close();
```

4.2.6.4.3.插入

插入单行：

```
PreparedStatement st = conn.prepareStatement("INSERT INTO mytable(col_1,col_2) values(?,?)");
st.setInt(1, 0);
st.setString(2, "row1");
int rowsInsert = st.executeUpdate();
System.out.println(rowsInsert + " rows insert");
st.close();
```

插入多行：

```
PreparedStatement st = conn.prepareStatement("INSERT INTO mytable(col_1,col_2) values(?,?)");
st.setInt(1, 0);
st.setString(2, "row1");
st.addBatch();
st.setInt(1,1);
st.setString(2, "row2");
st.addBatch();
st.executeBatch();
st.close();
```

4.2.6.5.创建或修改数据库对象

要创建、更改或者删除一个类似表或者视图这样的数据库对象，可以使用 execute() 方法，execute() 方法类于发出 select 语句的 executeQuery()方法 ，不过它不返回结果.

例如 DROP 表:

```
Statement st = conn.createStatement();
st.execute("DROP TABLE mytable");
st.close();
```

4.2.7. 调用存储过程

下面的例子展示了如何调用存储过程:

```
CallableStatement upperProc = conn.prepareCall("{? = call upper( ? ) }");
upperProc.registerOutParameter(1, Types.VARCHAR);
upperProc.setString(2, "lowercase to uppercase");
upperProc.execute();
String upperCased = upperProc.getString(1);
upperProc.close();
```

4.2.7.1.获取结果集

函数可以通过两种方式返回结果集，根据函数被调用的方式决定结果集的返回方式。

- 方式一:

```
Statement stmt = conn.createStatement();
stmt.execute("CREATE OR REPLACE FUNCTION setoffunc() RETURNS SETOF int AS "
   + "' SELECT 1 UNION SELECT 2;' LANGUAGE sql");
ResultSet rs = stmt.executeQuery("SELECT * FROM setoffunc()");
while (rs.next())
{
   // do something
}
rs.close();
stmt.close();
```

- 方式二:

```
// Setup function to call.
Statement stmt = conn.createStatement();
stmt.execute("CREATE OR REPLACE FUNCTION refcursorfunc() RETURNS refcursor"
```

```
   + " LANGUAGE plpgsql AS '"
   + " DECLARE "
   + "    mycurs refcursor; "
   + " BEGIN "
   + "    OPEN mycurs FOR SELECT 1 UNION SELECT 2; "
   + "    RETURN mycurs; "
   + " END;' ");
stmt.close();

// We must be inside a transaction for cursors to work.
conn.setAutoCommit(false);

// Procedure call.
CallableStatement proc = conn.prepareCall("{? = call refcursorfunc() }");
proc.registerOutParameter(1, Types.OTHER);
proc.execute();
ResultSet results = (ResultSet) proc.getObject(1);
while (results.next())
{
   // do something with the results.
}
results.close();
proc.close();
```

4.2.8. 处理大数据类型

4.2.8.1.二进制类型

Vastbase 提供两种不同的方法存储二进制数据。 二进制数据可以使用二进制数据类型 BYTEA 存储在表中， BYTEA 的字段可以存储最多 1G 字节的二进制数据。

使用 BYTEA 类型存储二进制数据，可以用 getBytes()，setBytes()，getBinaryStream()，setBinaryStream() 进行读写。

例如：

```
CREATE TABLE images (imgname text, img bytea);
```

4.2.8.1.1.插入记录

```
File file = new File("myimage.gif");
FileInputStream fis = new FileInputStream(file);
PreparedStatement ps = conn.prepareStatement("INSERT INTO images VALUES (?, ?)");
ps.setString(1, file.getName());
ps.setBinaryStream(2, fis);
ps.executeUpdate();
ps.close();
fis.close();
```

在这里，setBinaryStream()将一组字节从流传输到 BYTEA 类型的列中。如果数据已经在 byte[]中，那么也可以使用 setBytes()方法来实现数据写入。

4.2.8.1.2.读取记录

```
PreparedStatement ps = conn.prepareStatement("SELECT img FROM images WHERE imgname = ?");
ps.setString(1, "myimage.gif");
ResultSet rs = ps.executeQuery();
while (rs.next())
{
   byte[] imgBytes = rs.getBytes(1);
   // use the data in some way here
}
rs.close();
ps.close();
```

在这里，二进制数据被读取为 byte[]。也可以调用 getBinaryStream()处理 。

4.2.8.2.字符串类型

Vastbase 中 TEXT 类型与 VARCHAR 类型都是可变长的字符串类型，区别在于 VARCHAR 类型通过 VARCHAR(n)中的 n 来限制最大长度，而 TEXT 类型没有。TEXT 类型与 VARCHAR 类型几乎没有性能差别，TEXT 类型最多可存储 1G 数据 。

使用 TEXT 类型存储数据时，可用 getString()，getCharacterStream(), setString(), setCharacterStream()来进行读写。

例如：

```
CREATE TABLE images (id int, msg text);
```

4.2.8.2.1.插入记录

```
File file = new File("myimage.gif");
Reader reader = new InputStreamReader(new FileInputStream(file));
PreparedStatement ps = conn.prepareStatement("INSERT INTO images VALUES (?, ?)");
ps.setInt(1, 12);
ps.setCharacterStream (2, reader);
ps.executeUpdate();
ps.close();
```

如果数据内容已经在 String 中，那么也可以使用 setString()方法来实现数据写入。

4.2.8.2.2.读取记录

```
PreparedStatement ps = conn.prepareStatement("SELECT msg FROM images WHERE id= ?");
ps.setInt(1, 12);
ResultSet rs = ps.executeQuery();
while (rs.next())
{
   String imgBytes = rs.getString(1);
   // use the data in some way here
}
rs.close();
ps.close();
```

在这里，TEXT 数据被读取为 String，也可以调用 getCharacterStream()处理 。

4.2.9. 使用连接池

Vastbase jdbc 连接可使用连接池，比如阿里巴巴开源连接池 Druid:

```
Properties properties = new Properties();
//add properties...

DruidDataSource ds = new DruidDataSource();
ds.setDriverClassName("org.postgresql.Driver");
ds.setUrl("jdbc:postgresql://localhost:5432/vastbase");
ds.setDbType("postgresql");
ds.setConnectProperties(properties);
ds.setUsername("vuser");
ds.setPassword("vuser");
ds.setInitialSize(2);
ds.setMinIdle(2);
ds.setMaxActive(5);
//set other properties

 //get connection
 Connection conn = ds.getConnection();

 //execute SQL
 String sql = "INSERT INTO student VALUES (NULL, ?, ?, ?);";
 PreparedStatement pstmt = conn.prepareStatement(sql);
 pstmt.setString(1, "sam");
 pstmt.setInt(2, 35);
 pstmt.setDouble(3, 88.5);
 int i = pstmt.executeUpdate();
 System.out.println("insert rows:  " + i);

 pstmt.close();
//release connection and give back to pool
 conn.close();
```

4.3. ODBC

Vastbase 数据库通过提供 ODBC 驱动为使用 C/C++语言的应用程序提供访问 Vastbase 的接口。但同时也兼容 openGauss 的 ODBC 驱动，可以通过 openGauss 的 ODBC 驱动访问 Vastbase。

4.3.1. Vastbase ODBC 主要接口

Vastbase ODBC 驱动由具有建立和管理与 Vastbase 数据库连接、执行 SQL 语句和对结果集进行储存管理的若干功能类组成。

- SQLConnect

与特定数据库的连接（会话）。在连接上下文中执行 SQL 语句并返回结果。

示例：

```
SQLConnect(V_OD_hdbc,              // 连接句柄，通过 SQLAllocHandle 获得
(SQLCHAR*) "Test",  // 要连接数据源的名称
SQL_NTS,                  // ServerName 的长度
 (SQLCHAR*) "userName", //  数据源中数据库用户名
SQL_NTS,                          // UserName 的长度
 (SQLCHAR*) "password",  // 数据源中数据库用户密码
SQL_NTS);                        // Authentication 的长度
```

- SQLAllocHandle

用于分配环境、连接、语句或描述符的句柄，包括连接句柄、语句句柄等。

```
SQLAllocHandle(SQL_HANDLE_DBC,   // 由 SQLAllocHandle 分配的句柄类型
V_OD_Env,                  // 将要分配的新句柄的类型
&V_OD_hdbc);            // 输出参数：一个缓冲区的指针，此缓冲区以新分配的数据结构存放返回的句柄
```

- SQLSetConnectAttr

设置控制连接各方面的属性。

```
SQLSetConnectAttr(V_OD_hdbc,                                // 连接句柄
SQL_ATTR_AUTOCOMMIT,   // 设置属性
SQL_AUTOCOMMIT_ON,      // 指向对应 Attribute 的值。依赖于 Attribute 的值，ValuePtr 是 32 位无符号整型值或指向以空结束的字符串。
0);                                          // 如果 ValuePtr 指向字符串或二进制缓冲区，这个参数是*ValuePtr 长度，如果 ValuePtr 指向整型，忽略 StringLength
```

4.3.2. 设置 ODBC 驱动

安装 ODBC 驱动前需要先装好 openGauss 数据库或 Vastbase 数据库和 unixODBC,并配置环境变量。

4.3.2.1.安装 openGauss 或 Vastbase

按照安装说明安装 openGauss 或 Vastbase，并配置环境变量（根据实际情况修改路径）：

```
export GAUSSHOME="/usr/local/vastbase"
export PATH="/usr/local/vastbase/bin:$PATH"
export LD_LIBRARY_PATH="/usr/local/vastbase/lib:$LD_LIBRARY_PATH"
export PGDATA="/home/vastbase/data"
export PGPORT="5432"
export PGDATABASE="vastbase"
```

4.3.2.2.安装 unixODBC

要使用驱动， 必须先安装 unixODBC。获取 unixODBC 源码包参考地址为：
http://sourceforge.net/projects/unixodbc/files/unixODBC/2.3.0/unixODBC-2.3.0.tar.gz/download。

目前不支持 unixODBC-2.2.1 版本。以 unixODBC-2.3.0 版本为例，在客户端执行如下命令安装 unixODBC。默认安装到“/usr/local”目录下，生成数据源文件到 “/usr/local/etc”目录下，库文件生成在“/usr/local/lib”目录。

```
tar -zxvf unixODBC-2.3.0.tar.gz
cd unixODBC-2.3.0
#修改 configure 文件(如果不存在，那么请修改 configure.ac)，找到 LIB_VERSION
#将它的值修改为"1:0:0"，这样将编译出*.so.1 的动态库，与 psqlodbcw.so 的依赖关系相同。
vim configure
./configure --enable-gui=no
#如果要在鲲鹏服务器上编译，请追加一个 configure 参数: --build=aarch64-unknown-linux-gnu
make
make install
```

4.3.2.3.替换客户端驱动程序

1．解压 ODBC 驱动包；

2．将解压路径下 odbc/lib 目录下的 psqlodbcw.la 和 psqlodbcw.so 拷贝到“/usr/local/lib”目录下，或者将 odbc/lib 目录的绝对路径加入 LD_LIBRARY_PATH 环境变量；

3．将解压后 lib 目录下的库拷贝到“/usr/local/lib”目录下，或者将 lib 目录的绝对路径加入 LD_LIBRARY_PATH 环境变量。

4.3.2.4.配置数据源

1．配置 ODBC 驱动文件。在“/usr/local/etc/odbcinst.ini”文件中添加以下内容。

```
vim /usr/local/etc/odbcinst.ini
[GaussMPP]                                  # 驱动器名称，对应数据源 DSN 中的驱动名
Driver64=/usr/local/lib/psqlodbcw.so     # 驱动动态库的路径
setup=/usr/local/lib/psqlodbcw.so          # 驱动安装路径，与 Driver64 中动态库的路径一致
```

2．配置数据源文件。该步骤并不是必须的，取决于获取连接的方式，详见下章“连接数据库”小节。在“/usr/local/etc/odbc.ini ”文件中添加以下内容。

```
[Test]                              # 数据源的名称。
Description=Test                 # 数据源说明
Driver=GaussMPP              # 驱动名，对应 odbcinst.ini 中的 DriverName
Servername=127.0.0.1       # 服务器的 IP 地址
Database=vastbase             # 要连接的数据库的名称
Username=vastbase          # 数据库用户名称
Password= Aa@123456    # 数据库用户密码
Port=5432                        # 服务器的端口号
Sslmode=allow                # 开启 SSL 模式:disable 否，allow 可能 ,prefer 可能,requrie 是,verify-ca 是,
verify-full 是
```

4.3.2.5.配置环境变量

```
vim ~/.bashrc
# 在配置文件中追加以下内容。
export LD_LIBRARY_PATH=/usr/local/lib/:$LD_LIBRARY_PATH
```

```
export ODBCSYSINI=/usr/local/etc
export ODBCINI=/usr/local/etc/odbc.ini
# 执行如下命令使配置生效
source ~/.bashrc
```

4.3.2.6.测试数据源配置

上述配置中该数据源的 DSN 为 Test，可以通过 isql 来验证配置是否正确。命令如下。

```
isql -v Test
```

如果数据源配置中未指定用户名和密码，则用 isql 测试连接时需要指定用户密码：

```
isql -v Test vastbase Aa@123456
```

如果显示以下信息，表示配置正确，连接成功。

```
[vastbase@localhost root]$ isql -v Test
+---------------------------------------+
| Connected!                            |
|                                       |
| sql-statement                         |
| help [tablename]                      |
| quit                                  |
|                                       |
+---------------------------------------+
SQL>
```

4.3.3. 使用 ODBC 连接数据库

4.3.3.1.连接数据库

4.3.3.1.1.获取 ODBC 连接

获取连接有两种方法。

1．通过调用 SQLConnect()函数连接到数据库：

```
SQLHENV      V_OD_Env;  // 环境句柄
SQLHDBC      V_OD_hdbc;  // 连接句柄
SQLRETURN retcode;  // 返回值
SQLINTEGER   V_OD_erg; //调用 接口返回值
 /* 1.分配环境句柄 */
V_OD_erg = SQLAllocHandle(SQL_HANDLE_ENV,SQL_NULL_HANDLE,&V_OD_Env);
if ((V_OD_erg != SQL_SUCCESS) && (V_OD_erg != SQL_SUCCESS_WITH_INFO))
{
printf("Error AllocHandle\n");
exit(0);
}
 /* 2.设置 ODBC 环境属性（版本信息）*/
SQLSetEnvAttr(V_OD_Env, SQL_ATTR_ODBC_VERSION, (void*)SQL_OV_ODBC3, 0);
/* 3.分配连接句柄 */
V_OD_erg = SQLAllocHandle(SQL_HANDLE_DBC, V_OD_Env, &V_OD_hdbc);
if ((V_OD_erg != SQL_SUCCESS) && (V_OD_erg != SQL_SUCCESS_WITH_INFO))
```

```
{
SQLFreeHandle(SQL_HANDLE_ENV, V_OD_Env);
exit(0);
}
 /* 4.设置连接属性 */
SQLSetConnectAttr(V_OD_hdbc, SQL_ATTR_AUTOCOMMIT, SQL_AUTOCOMMIT_ON, 0);
/* 5. 连接数据源，这里的“userName”与“password”分别表示连接数据库的用户名和用户密码，请根据实际情况修改。
如果 odbc.ini 文件中已经配置了用户名密码，那么这里可以留空（""）。 但是不建议这么做，因为一旦 odbc.ini 权限管理不善，
将导致数据库用户密码泄露. */
V_OD_erg = SQLConnect(V_OD_hdbc, (SQLCHAR*) "Test", SQL_NTS,  (SQLCHAR*) "vastbase", SQL_NTS,
(SQLCHAR*) "Aa@123456", SQL_NTS);
 if ((V_OD_erg != SQL_SUCCESS) && (V_OD_erg != SQL_SUCCESS_WITH_INFO))
{
printf("Error SQLConnect %d\n",V_OD_erg);
SQLFreeHandle(SQL_HANDLE_ENV, V_OD_Env);
exit(0);
}
 printf("Connected !\n");
```

2. 通过调用 SQLDriverConnect()函数连接到数据库：

```
SQLHENV      V_OD_Env;  // 环境句柄
SQLHDBC      V_OD_hdbc;  // 连接句柄
SQLRETURN retcode;  // 返回值
SQLINTEGER   V_OD_erg; //调用 接口返回值
 /* 1.分配环境句柄 */
V_OD_erg = SQLAllocHandle(SQL_HANDLE_ENV,SQL_NULL_HANDLE,&V_OD_Env);
if ((V_OD_erg != SQL_SUCCESS) && (V_OD_erg != SQL_SUCCESS_WITH_INFO))
{
printf("Error AllocHandle\n");
exit(0);
}
 /* 2.设置 ODBC 环境属性（版本信息） */
SQLSetEnvAttr(V_OD_Env, SQL_ATTR_ODBC_VERSION, (void*)SQL_OV_ODBC3, 0);
/* 3.分配连接句柄 */
V_OD_erg = SQLAllocHandle(SQL_HANDLE_DBC, V_OD_Env, &V_OD_hdbc);
if ((V_OD_erg != SQL_SUCCESS) && (V_OD_erg != SQL_SUCCESS_WITH_INFO))
{
SQLFreeHandle(SQL_HANDLE_ENV, V_OD_Env);
exit(0);
}
 /* 4.设置连接属性 */
SQLSetConnectAttr(V_OD_hdbc, SQL_ATTR_AUTOCOMMIT, SQL_AUTOCOMMIT_ON, 0);
/* 5. 连接数据源 */
V_OD_erg = SQLDriverConnect(V_OD_hdbc, NULL,
(SQLCHAR*)"Driver=GaussMPP;Servername=127.0.0.1;Port=5432;Database=postgres;UserName=vastbase;Pas
sword=Aa@123456", SQL_NTS, str, sizeof(str), &strl,SQL_DRIVER_COMPLETE);
 if ((V_OD_erg != SQL_SUCCESS) && (V_OD_erg != SQL_SUCCESS_WITH_INFO))
{
printf("Error SQLConnect %d\n",V_OD_erg);
SQLFreeHandle(SQL_HANDLE_ENV, V_OD_Env);
exit(0);
}
 printf("Connected !\n");
```

4.3.3.1.2.关闭 ODBC 连接

关闭连接时调用 SQLDisconnect()函数关闭数据库连接:

```
rc = SQLDisconnect(V_OD_hdbc);
If (!SQL_SUCCEEDED(rc))
{
  printf("SQLDisconnect failed");
  fflush(stdout);
  exit(1);
}
rc = SQLFreeHandle(SQL_HANDLE_DBC,V_OD_hdbc);
if (!SQL_SUCCEEDED(rc))
{
  printf("SQLFreeHandle failed");
  fflush(stdout);
  exit(1);
}
V_OD_hdbc = NULL;
rc = SQLFreeHandle(SQL_HANDLE_ENV, V_OD_Env);
if (!SQL_SUCCEEDED(rc))
{
  printf("SQLFreeHandle failed");
  fflush(stdout);
  exit(1);
}
V_OD_Env= NULL;
```

4.3.3.2.连接参数

名称	缩写	类型	定义
Description		string	数据源说明
Servicename		string	数据源服务主机名或 IP
Port		int	数据库端口
Username		string	数据库用户
Password		string	数据库用户密码
UseDeclareFetch	B6	bool	是否使用 Declare 和 Fetch
Fetch	A7	int	批量读取的最大行数
Socket	A8	int	Socket buffer size
ReadOnly	A0	bool	数据库是否只读
MaxVarcharSize	B0	int	Varchar 的最大长度
KeepaliveTime	D1	int	TCP KEEPALIVE 设置：空闲时间
KeepaliveInterval	D2	int	TCP KEEPALIVE 设置：间隔

连接参数可以在连接串中指定：

```
ret = SQLDriverConnect(conn, NULL,
(SQLCHAR*)"Driver=PostgreSQL;Servername=192.168.114.28;Port=5411;Database=atlasdb;
UserName=atlasdb;Password=atlasdb;ReadOnly=Yes;Fetch=200", SQL_NTS, str, sizeof(str), &strl,
SQL_DRIVER_COMPLETE);
```

4.3.4. 执行 SQL 语句并处理结果

向数据库发出执行 SQL 的请求时，需要使用语句句柄，并调用 SQLExecDirect 或 SQLExecute 发出查询，服务端执行完成后查询的结果将保存在语句句柄中。

4.3.4.1.执行查询

4.3.4.1.1.直接执行

调用 SQLAllocHandle()函数获取句柄，调用 SQLExecDirect()函数执行查询。

```
HSTMT V_OD_hstmt = SQL_NULL_HSTMT; //语句句柄
ret = SQLAllocHandle(SQL_HANDLE_STMT, V_OD_hdbc, &V_OD_hstmt);
if (!SQL_SUCCEEDED(ret))
{
  printf("failed to allocate stmt handle");
  return;
}
/*执行语句*/
ret = SQLExecDirect(V_OD_hstmt, (SQLCHAR *) "select * from test_stu", SQL_NTS);
/*获取结果*/
ret = SQLFetch(V_OD_hstmt);
if (ret == SQL_NO_DATA)
  return;
if (ret == SQL_SUCCESS)
{
char buf[40];
SQLLEN ind;
/*获取列数据*/
  ret = SQLGetData(V_OD_hstmt, 1, SQL_C_CHAR, buf, sizeof(buf), &ind);
}
```

4.3.4.1.2.预编译方式执行

调用 SQLAllocHandle()函数获取句柄，使用 SQLPrepare 预编译语句，使用 SQLBindParameter 绑定参数，最后调用 SQLExecute 执行查询

```
/*初始化句柄*/
ret = SQLAllocHandle(SQL_HANDLE_STMT, V_OD_hdbc, &V_OD_hstmt);
if (!SQL_SUCCEEDED(ret))
{
  printf("failed to allocate stmt handle");
  exit(1);
}
/*预编译 SQL*/
ret = SQLPrepare(V_OD_hstmt, (SQLCHAR *) "SELECT * FROM test_stu WHERE id = ?", SQL_NTS);
CHECK_STMT_RESULT(ret, "SQLPrepare failed", V_OD_hstmt);
SQLLEN cbParam1 = SQL_NTS;
/*绑定参数*/
ret = SQLBindParameter(V_OD_hstmt, 1, SQL_PARAM_INPUT,
              SQL_C_SLONG,  /* value type */
```

```
              SQL_INTEGER,   /* param type */
               20,       /* column size */
               0,      /* dec digits */
               "1",    /* param value ptr */
               0,      /* buffer len */
               &cbParam1   /* StrLen_or_IndPtr */);
CHECK_STMT_RESULT(ret, "SQLBindParameter failed", V_OD_hstmt);
/*执行*/
ret = SQLExecute(V_OD_hstmt);
CHECK_STMT_RESULT(ret, "SQLExecute failed", V_OD_hstmt);
/*获取结果*/
/*获取结果*/
ret = SQLFetch(V_OD_hstmt);
if (ret == SQL_NO_DATA)
  return;
if (ret == SQL_SUCCESS)
{
char buf[40];
SQLLEN ind;
/*获取列数据*/
  ret = SQLGetData(V_OD_hstmt, 1, SQL_C_CHAR, buf, sizeof(buf), &ind);
}
```

4.3.4.2.使用游标获取结果

默认情况下，驱动程序将一次性获取查询的所有结果，当某个查询的结果集很庞大时，一次获取所有结果很不合理，因此 ODBC 驱动程序提供了一种基于数据库游标的结果集处理方法，只获取少量的行缓存在客户端。通过重新定位游标来获取下一批数据。

游标不能在所有情况下使用。有许多限制会使驱动程序一次性获取整个结果集：

- 连接不能使用自动提交模式 ，因为后端会在事务结束时关闭游标，因此在自动提交模式下，事务结束后，在从游标获取数据之前，游标已经被关闭。
- UseDeclareFetch 参数设置为 true，默认缓存 100 行数据，为 false 时将不会使用游标，直接获取整个结果集。
- 给定的查询语句必须是单条语句，不能是用分号连接在一起的多个语句。
- 设置 fetch size，且 fetch size 不能为 0.

示例：

```
int rc;
int i;
/*关闭自动提交*/
rc = SQLSetConnectAttr(V_OD_hdbc, SQL_AUTOCOMMIT, (SQLPOINTER) SQL_AUTOCOMMIT_OFF, 0);
  CHECK_CONN_RESULT(rc, "SQL_AUTOCOMMIT off failed", V_OD_hdbc);
/*执行 SQL*/
rc = SQLExecDirect(V_OD_hstmt, (SQLCHAR *) "SELECT 'foo' || g FROM generate_series(1, 3210) g",
SQL_NTS);
CHECK_STMT_RESULT(rc, "SQLExecDirect failed", V_OD_hstmt);
/*获取结果*/
for (;; i++)
{
```

```
char buf[40];
  SQLLEN ind;
  rc = SQLFetch(V_OD_hstmt);
  if (rc == SQL_NO_DATA)
    break;
  if (rc != SQL_SUCCESS)
  {
    char sqlstate[32] = "";
    SQLINTEGER nativeerror;
    SQLSMALLINT textlen;
    SQLGetDiagRec(SQL_HANDLE_STMT, V_OD_hstmt, 1, sqlstate, &nativeerror,
              NULL, 0, &textlen);
            /*在读取游标的过程中，如果调用用 SQLEndTran 提交或回滚了事务，游标将被关闭*/
    if (strcmp(sqlstate, "HY010") == 0)
    {
      printf("SQLFetch failed with HY010 (which probably means that the cursor was closed at
commit/rollback)");
      break;
    }
    else
      CHECK_STMT_RESULT(rc, "SQLGetDiagRec failed", V_OD_hstmt);
  }

  rc = SQLGetData(V_OD_hstmt, 1, SQL_C_CHAR, buf, sizeof(buf), &ind);
  CHECK_STMT_RESULT(rc, "SQLGetData failed", V_OD_hstmt);
}
```

4.3.4.3.执行更新

要更改数据(执行一个 insert，update 或者 delete)，可以使用 SQLExecDirect 函数执行完整的 SQL，也可以使用预编译的方式绑定参数后再执行。下面的示例将以预编译的方式执行。

4.3.4.3.1.删除

```
rc = SQLAllocHandle(SQL_HANDLE_STMT, V_OD_hdbc, &V_OD_hstmt);
rc = SQLPrepare(V_OD_hstmt, (SQLCHAR *) "delete from test_stu where id= ? ", SQL_NTS);
cbParam1 = SQL_NTS;
rc = SQLBindParameter(V_OD_hstmt, 1, SQL_PARAM_INPUT,
SQL_C_SLONG, /* value type */
SQL_INTEGER, /* param type */
20,    /* column size */
0,   /* dec digits */
“1”,   /* param value ptr */
0,   /* buffer len */
&cbParam1  /* StrLen_or_IndPtr */);
CHECK_STMT_RESULT(rc, "SQLBindParameter failed", V_OD_hstmt);
rc = SQLExecute(V_OD_hstmt);
CHECK_STMT_RESULT(rc, "SQLExecute failed", V_OD_hstmt);
```

4.3.4.3.2.更新

```
rc = SQLAllocHandle(SQL_HANDLE_STMT, V_OD_hdbc, &V_OD_hstmt);
rc = SQLPrepare(V_OD_hstmt, (SQLCHAR *) "update stu_id set name = ‘Lisa’ where id = ? ", SQL_NTS);
cbParam1 = SQL_NTS;
rc = SQLBindParameter(V_OD_hstmt, 1, SQL_PARAM_INPUT,
          SQL_C_SLONG, /* value type */
```

```
            SQL_INTEGER, /* param type */
            20,    /* column size */
                0,   /* dec digits */
            "100",   /* param value ptr */
            0,   /* buffer len */
            &cbParam1  /* StrLen_or_IndPtr */);
CHECK_STMT_RESULT(rc, "SQLBindParameter failed", V_OD_hstmt);
rc = SQLExecute(V_OD_hstmt);
CHECK_STMT_RESULT(rc, "SQLExecute failed", V_OD_hstmt);
```

4.3.4.3.3.插入

```
rc = SQLAllocHandle(SQL_HANDLE_STMT, V_OD_hdbc, &V_OD_hstmt);
rc = SQLPrepare(V_OD_hstmt, (SQLCHAR *) "INSERT INTO test_stu VALUES (2, ?) ", SQL_NTS);
cbParam1 = SQL_NTS;
rc = SQLBindParameter(V_OD_hstmt, 1, SQL_PARAM_INPUT,
            SQL_C_CHAR,/* value type */
            SQL_CHAR,  /* param type */
            20,    /* column size */
            0,   /* dec digits */
            "Jennifer",  /* param value ptr */
            0,   /* buffer len */
            &cbParam1  /* StrLen_or_IndPtr */);
CHECK_STMT_RESULT(rc, "SQLBindParameter failed", V_OD_hstmt);
rc = SQLExecute(V_OD_hstmt);
CHECK_STMT_RESULT(rc, "SQLExecute failed", V_OD_hstmt);
```

4.3.4.4.创建或修改数据库对象

要创建、更改或者删除一个类似表或者视图这样的数据库对象， 可以调用 SQLExecDirect() 方法. 例如创建表：

```
HSTMT V_OD_hstmt = SQL_NULL_HSTMT; //语句句柄
ret = SQLAllocHandle(SQL_HANDLE_STMT, V_OD_hdbc, &V_OD_hstmt);
if (!SQL_SUCCEEDED(ret))
{
  printf("failed to allocate stmt handle");
  return;
}
ret = SQLExecDirect(V_OD_hstmt, (SQLCHAR *) "create table test(c1 int,c2 int)", SQL_NTS);
```

4.3.5. 调用存储过程

下面的例子展示了如何调用存储过程：

```
/*获取语句句柄*/
rc = SQLAllocHandle(SQL_HANDLE_STMT, V_OD_hdbc, &V_OD_hstmt);
/*预编译SQL，调用函数时使用{call funcname(?,...)} */
SQLPrepare(V_OD_hstmt, (SQLCHAR *) "{ call length(?) }", SQL_NTS);
cbParams[paramno] = SQL_NTS;
/*绑定参数 */
rc = SQLBindParameter(V_OD_hstmt, paramno, SQL_PARAM_INPUT,
SQL_C_CHAR,/* value type */
SQL_CHAR,    /* param type */
```

```
20,      /* column size */
0,     /* dec digits */
"testvalue",    /* param value ptr */
0,     /* buffer len */
&cbParams[paramno]     /* StrLen_or_IndPtr */);
CHECK_STMT_RESULT(rc, "SQLBindParameter failed", V_OD_hstmt);
/*执行*/
rc = SQLExecute(V_OD_hstmt);
CHECK_STMT_RESULT(rc, "SQLExecute failed", V_OD_hstmt);
```

4.3.6. 处理大数据类型

4.3.6.1.二进制类型

Vastbase 提供两种不同的方法存储二进制数据。 二进制数据可以使用二进制数据类型 BYTEA 存储在表中，或者使用大对象特性以一种特殊的格式将二进制数据存储在一个独立的表中， 然后通过在表中保存一个类型为 OID 的值来引用该表。

为了判断哪种方法比较合适，必须理解每种方法的局限。 BYTEA 数据类型并不适合存储非常大数量的二进制数据。虽然类型为 BYTEA 的字段可以存储最多 1G 字节的二进制数据， 但是这样它会要求巨大的内存(RAM)来处理这样巨大的数据。用于存储二进制数据的大对象方法更适合存储非常大的数据， 但也有自己的局限，具体来说，删除一个引用大数据的行时并不会删除大对象，删除大对象需要单独操作。大对象还有一些安全性的问题， 因为连接到数据库的任何用户都可以查看或修改任何大对象，即使他们没有权限查看或修改包含大对象引用的行。

4.3.6.1.1.BYTEA

使用 BYTEA 类型存储二进制数据，例如：

```
CREATE TABLE lo_test_tab (id int, img bytea);
```

4.3.6.1.1.1. 插入记录

```
/*二进制数据*/
char param1[20] = { 1, 2, 3, 4, 5, 6, 7, 8 };
SQLLEN cbParam1;
/*分配语句句柄*/
rc = SQLAllocHandle(SQL_HANDLE_STMT, V_OD_hdbc, &V_OD_hstmt);
if (!SQL_SUCCEEDED(rc))
{
  printf("failed to allocate stmt handle");
  exit(1);
}
/*预编译SQL*/
rc = SQLPrepare(V_OD_hstmt, (SQLCHAR *) "INSERT INTO lo_test_tab VALUES (1, ?)", SQL_NTS);
CHECK_STMT_RESULT(rc, "SQLPrepare failed", V_OD_hstmt);
/*绑定参数*/
cbParam1 = 8;
rc = SQLBindParameter(V_OD_hstmt, 1, SQL_PARAM_INPUT,
           SQL_C_BINARY,/* value type */
           SQL_LONGVARBINARY,  /* param type */
           200,     /* column size */
           0,     /* dec digits */
           param1,     /* param value ptr */
```

```
          0,     /* buffer len */
          &cbParam1     /* StrLen_or_IndPtr */);
CHECK_STMT_RESULT(rc, "SQLBindParameter failed", V_OD_hstmt);

/* 执行*/
rc = SQLExecute(V_OD_hstmt);
CHECK_STMT_RESULT(rc, "SQLExecute failed", V_OD_hstmt);
```

在这里，SQLBindParameter 时指定参数类型为 SQL_LONGVARBINARY，再将参数值的指针(char *类型)传入函数，即可将二进制数据写入。

4.3.6.1.1.2. 读取记录

```
char buf[100];
SQLLEN ind;
/*分配句柄*/
rc = SQLAllocHandle(SQL_HANDLE_STMT, V_OD_hdbc, &V_OD_hstmt);
/*执行查询*/
SQLExecDirect(V_OD_hstmt, (SQLCHAR *) "SELECT id, img FROM lo_test_tab WHERE id = 1", SQL_NTS);
/*获取结果*/
rc = SQLFetch(V_OD_hstmt);
CHECK_STMT_RESULT(rc, "SQLFetch failed", V_OD_hstmt);
/*读取数据*/
rc = SQLGetData(V_OD_hstmt, 2, SQL_C_BINARY, buf, sizeof(buf), &ind);
CHECK_STMT_RESULT(rc, "SQLGetData failed", V_OD_hstmt);
```

4.3.6.2.字符串类型

Vastbase 中 TEXT 类型与 VARCHAR 类型都是可变长的字符串类型，区别在于 VARCHAR 类型通过 VARCHAR(n)中的 n 来限制最大长度，而 TEXT 类型没有。TEXT 类型与 VARCHAR 类型几乎没有性能差别，TEXT 类型最多可存储 1G 数据 。

使用 TEXT 类型存储数据时，可用 getString()，getCharacterStream(), setString(), setCharacterStream()来进行读写。

例如：

```
CREATE TABLE images (id int, msg text);
```

4.3.6.2.1.插入记录

```
str=”text value for test”;
SQLLEN cbParam1 = SQL_NTS;
/*分配语句句柄*/
rc = SQLAllocHandle(SQL_HANDLE_STMT, V_OD_hdbc, &V_OD_hstmt);
if (!SQL_SUCCEEDED(rc))
{
  printf("failed to allocate stmt handle");
  exit(1);
}
/*预编译SQL*/
rc = SQLPrepare(V_OD_hstmt, (SQLCHAR *) "INSERT INTO images  VALUES (1, ?)", SQL_NTS);
CHECK_STMT_RESULT(rc, "SQLPrepare failed", V_OD_hstmt);
/*绑定参数*/
cbParam1 = 8;
rc = SQLBindParameter(V_OD_hstmt, 1, SQL_PARAM_INPUT,
          SQL_C_CHAR,/* value type */
```

```
            SQL_CHAR,  /* param type */
            200,     /* column size */
            0,     /* dec digits */
            str,   /* param value ptr */
            0,     /* buffer len */
            &cbParam1     /* StrLen_or_IndPtr */);
CHECK_STMT_RESULT(rc, "SQLBindParameter failed", V_OD_hstmt);

/* 执行*/
rc = SQLExecute(V_OD_hstmt);
CHECK_STMT_RESULT(rc, "SQLExecute failed", V_OD_hstmt);
```

4.3.6.2.2.读取记录

```
char buf[100];
SQLLEN ind;
/*分配句柄*/
rc = SQLAllocHandle(SQL_HANDLE_STMT, V_OD_hdbc, &V_OD_hstmt);
/*执行查询*/
SQLExecDirect(V_OD_hstmt, (SQLCHAR *) "SELECT id, img FROM images  WHERE id = 1", SQL_NTS);
/*获取结果*/
rc = SQLFetch(V_OD_hstmt);
CHECK_STMT_RESULT(rc, "SQLFetch failed", V_OD_hstmt);
/*读取数据*/
rc = SQLGetData(V_OD_hstmt, 2, SQL_C_CHAR, buf, sizeof(buf), &ind);
CHECK_STMT_RESULT(rc, "SQLGetData failed", V_OD_hstmt);
```

4.4. 基于 libpq 开发

Vastbase 未对此接口在应用程序开发场景下的使用做验证。因此对使用此接口做应用程序开发存在的风险未知，故不推荐用户使用此套接口做应用程序开发。推荐用户使用 ODBC 或 JDBC 接口来替代。

4.5. 调试

用户可以根据自己的需要，通过修改实例数据目录下的 postgresql.conf 文件中特定的配置参数来控制日志的输出，从而更好的了解数据库的运行状态。

可调整的配置参数请参见表 4-1。

表 4-1. 配置参数

参数名称	描述	取值范围	备注
client_min_messages	配置发送到客户端信息的级别。	• DEBUG5 • DEBUG4 • DEBUG3 • DEBUG2	设置级别后，发送到客户端的信息包含所设级别及以下所有低级别会发送的信息。级别越低，发送的信息越少。

参数名称	描述	取值范围	备注
		• DEBUG1 • LOG • NOTICE • WARNING • ERROR • FATAL • PANIC 默认值：NOTICE 。	
log_min_messages	配置写到服务器日志里信息的级别。	• DEBUG5 • DEBUG4 • DEBUG3 • DEBUG2 • DEBUG1 • INFO • NOTICE • WARNING • ERROR • LOG • FATAL • PANIC 默认值：WARNING。	指定某一级别后，写到日志的信息包含所有更高级别会输出的信息。级别越高，服务器日志的信息越少。
log_min_error_statement	配置写到服务器日志中错误 SQL 语句的级别。	• DEBUG5 • DEBUG4 • DEBUG3 • DEBUG2	所有导致一个特定级别（或者更高级别）错误的 SQL 语句都将记录在服务器日志中。 只有系统管理员可以修改该参数。

参数名称	描述	取值范围	备注
		• DEBUG1 • INFO • NOTICE • WARNING • ERROR • FATAL • PANIC 缺省值：ERROR。	
log_min_duration_statement	配置语句执行持续的最短时间。如果某个语句的持续时间大于或者等于设置的毫秒数，则会在日志中记录该语句及其持续时间。打开这个选项可以方便地跟踪需要优化的查询。	INT 类型。 默认值：-1。 单位：毫秒。	设置为-1 表示关闭这个功能。 只有系统管理员可以修改该参数。
log_connections/log_disconnections	配置是否在每次会话连接或结束时向服务器日志里打印一条信息。	• on：每次会话连接或结束时向日志里打印一条信息。 • off：每次会话连接或结束时不向日志里打印信息。 默认值：off。	-
log_duration	配置是否记录每个已完成语句的持续时间。	• on：记录每个已完成语句的持续时间。 • off：不记	只有系统管理员可以修改该参数。

参数名称	描述	取值范围	备注
		录已完成语句的持续时间。 默认值：off。	
log_statement	配置日志中记录哪些 SQL 语句。	• none：不记录任何 SQL 语句。 • ddl：记录数据定义语句。 • mod：记录数据定义语句和数据操作语句。 • all ：记录所有语句。 默认值：none。	只有系统管理员可以修改该参数。
log_hostname	配置是否记录主机名。	• on：记录主机名。 • off：不记录主机名。 默认值：off。	缺省时，连接日志只记录所连接主机的 IP 地址。打开这个选项会同时记录主机名。 该参数同时影响 5.3.2 查看审计结果、14.3.9GS_WLM_SESSION_HISTORY、14.3.40PG_STAT_ACTIVITY 和 log_line_prefix 参数。

上表有关参数级别的说明请参见表 4-2。

表 4-2. 日志级别参数说明

级别	说明
DEBUG[1-5]	提供开发人员使用的信息。5 级为最高级别，依次类推，1 级为最低级别。

级别	说明
INFO	提供用户隐含要求的信息。如在 VACUUM VERBOSE 过程中的信息。
NOTICE	提供可能对用户有用的信息。如长标识符的截断，作为主键一部分创建的索引。
WARNING	提供给用户的警告。如在事务块范围之外的 COMMIT。
ERROR	报告导致当前命令退出的错误。
LOG	报告一些管理员感兴趣的信息。如检查点活跃性。
FATAL	报告导致当前会话终止的原因。
PANIC	报告导致所有会话退出的原因。

5. 管理数据库安全

5.1. 客户端接入认证

5.1.1. 配置客户端接入认证

背景信息

如果主机需要远程连接数据库，必须在数据库系统的配置文件中增加此主机的信息，并且进行客户端接入认证。配置文件（默认名称为 pg_hba.conf）存放在数据库的数据目录里。hba（host-based authentication）表示是基于主机的认证。

- 本产品支持如下三种认证方式，这三种方式都需要配置“pg_hba.conf”文件。
  - 基于主机的认证：服务器端根据客户端的 IP 地址、用户名及要访问的数据库来查看配置文件从而判断用户是否通过认证。
  - 口令认证：包括远程连接的加密口令认证和本地连接的非加密口令认证。
  - SSL 加密：使用 OpenSSL（开源安全通信库）提供服务器端和客户端安全连接的环境。
- “pg_hba.conf”文件的格式是一行写一条信息，表示一个认证规则，空白和注释（以#开头）被忽略。
- 每个认证规则是由若干空格和/，空格和制表符分隔的字段组成。如果字段用引号包围，则它可以包含空白。一条记录不能跨行存在。

操作步骤

步骤 1 以操作系统用户 vastbase 登录数据库主节点。

步骤 2 在数据库主节点实例对应的“pg_hba.conf”文件中添加规则，配置客户端认证方式，允许客户端以“jack”用户连接到本机，此处远程连接禁止使用“vastbase”用户（即数据库初始化用户）。

例如，下面示例中，配置允许 IP 地址为 10.10.0.30 的客户端访问本机。

```
host all jack 10.10.0.30/32 md5;
```

说明

- 使用“jack”用户前，需先本地连接数据库，并在数据库中使用如下语句建立“jack”用户：

```
vastbase=# CREATE USER jack PASSWORD 'Test@123';
```

- jack 表示连接数据库的用户。
- 10.10.0.30/32 表示只允许 IP 地址为 10.10.0.30 的主机连接。此处的 IP 地址不能为 Vastbase 内的 IP，在使用过程中，请根据用户的网络进行配置修改。32 表示子网掩码为 1 的位数，即 255.255.255.255
- md5 表示连接时 jack 用户的密码使用 md5 算法加密。

“pg_hba.conf”文件中的每条记录可以是下面四种格式之一，四种格式的参数说明请参见 5.1.2 配置文件参考。

```
local     DATABASE USER METHOD [OPTIONS]
host      DATABASE USER ADDRESS METHOD [OPTIONS]
hostssl   DATABASE USER ADDRESS METHOD [OPTIONS]
hostnossl DATABASE USER ADDRESS METHOD [OPTIONS]
```

因为认证时系统是为每个连接请求顺序检查“pg_hba.conf”里的记录的，所以这些记录的顺序是非常关键的。

说明

- 可配置“时间段”字段：

（1）表示一段时间范围，例如"[2019-10-01 09:30:28, 2020-10-01 23:59:59)"，其中 [] 表示闭区间，() 表示开区间。区间中的每个元素要求是合法的 timestamptz 类型，可以通过逗号分隔的方式指定多个时间段。

（2）兼容旧的配置方式，意味着在没有配置时间段时，表示允许任意时刻连接。

（3）[2020-8-14,) 表示从 2020-8-14 且包含 2020-8-14 到永久； [2020-8-14,] 与其含义一致。

（4） [,] 、(,) 、(,] 、[,) 含义都表示任意时刻

（5）访问控制只针对登录那一瞬间，因此连接成功后，超出这个时间不会被断开。

- 示例如下：

允许 192.168.1.111 客户端通过 Unix domain 方式在 2020-10-01 至 2021-11-01 和 20210101 至 20210501 时间段登录

```
local   all   all   192.168.1.111   "[2020-10-01, 2021-11-01]","[20210101,20210501]"   md5
```

说明

在配置“pg_hba.conf”文件时，请依据通讯需求按照格式内容从上至下配置记录，优先级高的需求需要配置在前面。Vastbase 和扩容配置的 IP 优先级最高，用户手动配置的 IP 请放在这二者之后，如果已经进行的客户配置和扩容节点的 IP 在同一网段，请在扩容前删除，扩容成功后再进行配置。

因此对于认证规则的配置建议如下：

- 靠前的记录有比较严格的连接参数和比较弱的认证方法。
- 靠后的记录有比较宽松的连接参数和比较强的认证方法。

说明

- 一个用户要想成功连接到特定的数据库，不仅需要通过 pg_hba.conf 中的规则检查，还必须要有该数据库上的 CONNECT 权限。如果希望控制某些用户只能连接到指定数据库，赋予/撤销 CONNECT 权限通常比在 pg_hba.conf 中设置规则更为简单。
- 对应 Vastbase 外部客户端连接，trust 为不安全的认证方式，请将认证方式设置为 sha256。

异常处理

用户认证失败有很多原因，通过服务器返回给客户端的提示信息，可以看到用户认证失败的原因。常见的错误提示请参见表 5-1。

表 5-1. 错误提示

问题现象	解决方法
用户名或密码错误： FATAL: invalid username/password,login denied	这条信息说明用户名或者密码错误，请检查输入是否有误。
连接的数据库不存在： FATAL: database "TESTDB" does not exist	这条信息说明尝试连接的数据库不存在，请检查连接的数据库名输入是否有误。
未找到客户端匹配记录： FATAL: no pg_hba.conf entry for host "10.10.0.60", user "ANDYM", database "TESTDB"	这条信息说明已经连接了服务器，但服务器拒绝了连接请求，因为没有在它的 pg_hba.conf 配置文件里找到匹配的记录。请联系数据库管理员在 pg_hba.conf 配置文件加入用户的信息。

示例

```
TYPE  DATABASE        USER            ADDRESS                 METHOD

"local" is for Unix domain socket connections only
#表示只允许以安装时-U 参数指定的用户从服务器本机进行连接。
local   all             all                                     trust
IPv4 local connections:
#表示允许 vastbase 用户从 10.10.0.50 主机上连接到任意数据库，使用 sha256 算法对密码进行加密。
host    all             vastbase            10.10.0.50/32           sha256
#表示允许任何用户从 10.10.0.0/24 网段的主机上连接到任意数据库，使用 sha256 算法对密码进行加密，并且经过 SSL 加密传输。
hostssl    all             all             10.10.0.0/24            sha256
#表示允许任何用户从 10.10.0.0/24 网段的主机上连接到任意数据库，使用 Kerberos 认证方式，当前版本暂不支持客户端
kerberos 认证。
host    all             all             10.10.0.0/24            gss        include_realm=1
krb_realm=HADOOP.COM
```

5.1.2. 配置文件参考

表 5-2. 参数说明

参数名称	描述	取值范围
local	表示这条记录只接受通过 Unix 域套接字进行的连接。没有这种类型的记录，就不允许 Unix 域套接字的连接。 只有在从服务器本机使用 vsql 连接且在不指定-U 参数的情况下，才是通过 Unix 域套接字连接。	-

参数名称	描述	取值范围
host	表示这条记录既接受一个普通的 TCP/IP 套接字连接，也接受一个经过 SSL 加密的 TCP/IP 套接字连接。	-
hostssl	表示这条记录只接受一个经过 SSL 加密的 TCP/IP 套接字连接。	5.1.3. 用 SSL 进行安全的连接，需要配置申请数字证书并配置相关参数，详细信息请参见 5.1.3SSL 证书管理 Vastbase 默认配置了通过 openssl 生成的安全证书、私钥。并且提供证书替换的接口，方便用户进行证书的替换。 5.1.3.1. 证书生成 **操作场景** 在测试环境下，用户可以用通过以下方式进行数字证书测试。在客户的运行环境中，请使用从 CA 认证中心申请的数字证书。 **前提条件** Linux 环境安装了 openssl 组件。 **自认证证书生成过程** **步骤 1** 搭建 CA 环境。 --假设用户为 vastbase 已存在，搭建 CA 的路径为 test --以 root 用户身份登录 Linux 环境，切换到用户 vastbase mkdir test cd /etc/pki/tls --copy 配置文件

参数名称	描述	取值范围
		openssl.cnf 到 test 下 cp openssl.cnf ~/test cd ~/test --到 test 文件夹下，开始搭建 CA 环境 --创建文件夹 demoCA./demoCA/newcerts./demoCA/private mkdir ./demoCA ./demoCA/newcerts ./demoCA/private chmod 777 ./demoCA/private --创建 serial 文件,写入 01 echo '01'>./demoCA/serial --创建文件 index.txt touch ./demoCA/index.txt --修改 openssl.cnf 配置文件中的参数 dir = ./demoCA default_md = sha256 --至此 CA 环境搭建完成 **步骤 2** 生成根私钥。 --生成 CA 私钥 openssl genrsa -aes256 -out demoCA/private/cakey.pem 2048 Generating RSA private key, 2048 bit long modulus .....+++ .....+++ e is 65537 (0x10001) --设置根私钥的保护密码,最少要求 4 个字符，假设为 Test@123 Enter pass phrase for demoCA/private/cakey.pem: --再次输入私钥密码 Test@123 Verifying - Enter pass phrase for demoCA/private/cakey.pem: **步骤 3** 生成根证书请求文件。 --生成 CA 根证书申请文件 server.req openssl req -config openssl.cnf -new -key demoCA/private/cakey.pem -out demoCA/careq.pem Enter pass phrase for demoCA/private/cakey.pem

参数名称	描述	取值范围
		: --输入根私钥密码 Test@123 You are about to be asked to enter information that will be incorporated into your certificate request. What you are about to enter is what is called a Distinguished Name or a DN. There are quite a few fields but you can leave some blank For some fields there will be a default value, If you enter '.', the field will be left blank. ----- --以下名称请牢记,生成服务器证书和客户端证书时填写的信息需要与此处的一致 Country Name (2 letter code) [AU]:CN State or Province Name (full name) [Some-State]:shanxi Locality Name (eg, city) []:xian Organization Name (eg, company) [Internet Widgits Pty Ltd]:Abc Organizational Unit Name (eg, section) []:hello --Common Name 可以随意命名 Common Name (eg, YOUR name) []:world --Email 可以选择性填写 Email Address []: Please enter the following 'extra' attributes to be sent with your certificate request A challenge password []: An optional company name []:
		步骤 4 生成自签发根证书。 --生成根证书时，需要修改 openssl.cnf 文件，设置 basicConstraints=CA:TRUE

参数名称	描述	取值范围
		vi openssl.cnf --生成 CA 自签发根证书 openssl ca -config openssl.cnf -out demoCA/cacert.pem -keyfile demoCA/private/cakey.pem -selfsign -infiles demoCA/careq.pem Using configuration from openssl.cnf Enter pass phrase for demoCA/private/cakey.pem : --输入根私钥密码 Test@123 Check that the request matches the signature Signature ok Certificate Details: Serial Number: 1 (0x1) Validity Not Before: Feb 28 02:17:11 2017 GMT Not After : Feb 28 02:17:11 2018 GMT Subject: countryName = CN stateOrProvinceName = shanxi organizationName = Abc organizationalUnitName = hello commonName = world X509v3 extensions: X509v3 Basic Constraints: CA:FALSE Netscape Comment: OpenSSL Generated Certificate X509v3 Subject Key Identifier: F9:91:50:B2:42:8C:A8:D3: 41:B0:E4:42:CB:C2:BE:8D: B7:8C:17:1F

参数名称	描述	取值范围
		X509v3 Authority Key Identifier: keyid:F9:91:50:B2:42:8C: A8:D3:41:B0:E4:42:CB:C2: BE:8D:B7:8C:17:1F Certificate is to be certified until Feb 28 02:17:11 2018 GMT (365 days) Sign the certificate? [y/n]:y 1 out of 1 certificate requests certified, commit? [y/n]y Write out database with 1 new entries Data Base Updated --至此 CA 根证书自签发完成，根 证书 demoCA/cacert.pem。 **步骤 5** 生成服务端证书私钥。 --生成服务器私钥文件 server.key openssl genrsa -aes256 -out server.key 2048 Generating a 2048 bit RSA private key .....++++++ ..++++++ e is 65537 (0x10001) Enter pass phrase for server.key: --服务器私钥的保护密码,最少要 求 4 个字符，假设为 Test@123 Verifying - Enter pass phrase for server.key: --再次确认服务器私钥的保护密 码，即为 Test@123 **步骤 6** 生成服务端证书请求文件。 --生成服务器证书请求文件 server.req openssl req -config openssl.cnf -new -key server.key -out server.req Enter pass phrase for server.key:

参数名称	描述	取值范围
		You are about to be asked to enter information that will be incorporated into your certificate request. What you are about to enter is what is called a Distinguished Name or a DN. There are quite a few fields but you can leave some blank For some fields there will be a default value, If you enter '.', the field will be left blank. ----- --以下填写的信息与创建CA时的信息一致 Country Name (2 letter code) [AU]:CN State or Province Name (full name) [Some-State]:shanxi Locality Name (eg, city) []:xian Organization Name (eg, company) [Internet Widgits Pty Ltd]:Abc Organizational Unit Name (eg, section) []:hello --Common Name可以随意命名 Common Name (eg, YOUR name) []:world Email Address []: --以下信息可以选择性填写 Please enter the following 'extra' attributes to be sent with your certificate request A challenge password []: An optional company name []: **步骤 7** 生成服务端证书。 --生成服务端/客户端证书时，修改openssl.cnf文件，设置basicConstraints=CA:FALSE vi openssl.cnf --修改demoCA/index.txt.attr中

参数名称	描述	取值范围
		属性为 no。 vi demoCA/index.txt.attr --对生成的服务器证书请求文件进行签发，签发后将生成正式的服务器证书 server.crt openssl ca -config openssl.cnf -in server.req -out server.crt -days 3650 -md sha256 Using configuration from /etc/ssl/openssl.cnf Enter pass phrase for ./demoCA/private/cakey.pem: Check that the request matches the signature Signature ok Certificate Details: Serial Number: 2 (0x2) Validity Not Before: Feb 27 10:11:12 2017 GMT Not After : Feb 25 10:11:12 2027 GMT Subject: countryName = CN stateOrProvinceName = shanxi organizationName = Abc organizationalUnitName = hello commonName = world X509v3 extensions: X509v3 Basic Constraints: CA:FALSE Netscape Comment: OpenSSL Generated Certificate X509v3 Subject Key Identifier: EB:D9:EE:C0:D2:14:48:AD:

参数名称	描述	取值范围
		EB:BB:AD:B6:29:2C:6C:72:96:5C:38:35 X509v3 Authority Key Identifier: keyid:84:F6:A1:65:16:1F:28:8A:B7:0D:CB:7E:19:76:2A:8B:F5:2B:5C:6A Certificate is to be certified until Feb 25 10:11:12 2027 GMT (3650 days) --选择 y 对证书进行签发 Sign the certificate? [y/n]:y --选择 y，证书签发结束 1 out of 1 certificate requests certified, commit? [y/n]y Write out database with 1 new entries Data Base Updated 去掉私钥密码保护，方法如下： --去掉服务器私钥的密码保护 openssl rsa -in server.key -out server.key --如果不去掉服务器私钥的密码保护需要使用 vb_guc 工具对存储密码进行加密保护 vb_guc encrypt -M server -K Test@123 -D ./ --vb_guc 加密后会生成 server.key.cipher,server.key.rand 两个私钥密码保护文件 **步骤 8** 客户端证书，私钥的生成。 生成客户端证书和客户端私钥的方法和要求与服务器相同。 --生成客户端私钥 openssl genrsa -aes256 -out client.key 2048 --生成客户端证书请求文件 openssl req -config openssl.cnf -new -key client.key -out

参数名称	描述	取值范围
		client.req --对生成的客户端证书请求文件进行签发，签发后将生成正式的客户端证书 client.crt openssl ca -config openssl.cnf -in client.req -out client.crt -days 3650 -md sha256 去掉私钥密码保护，方法如下： --去掉客户端私钥的密码 openssl rsa -in client.key -out client.key --如果不去掉客户端私钥的密码保护需要使用 vb_guc 工具对存储密码进行加密保护 vb_guc encrypt -M client -K Test@123 -D ./ vb_guc 加密后会生成 client.key.cipher,client.key.rand 两个私钥密码保护文件。 将客户端密钥转化为 DER 格式，方法如下： openssl pkcs8 -topk8 -outform DER -in client.key -out client.key.pk8 -nocrypt **步骤 9** 吊销证书列表的生成。 如果需要吊销列表，可按照如下方法生成： --首先创建 crlnumber 文件 echo '00'>./demoCA/crlnumber --吊销服务器证书 openssl ca -config openssl.cnf -revoke server.crt --生成证书吊销列表 sslcrl-file.crl openssl ca -config openssl.cnf -gencrl -out sslcrl-file.crl

参数名称	描述	取值范围
		用 SSL 进行安全的 TCP/IP 连接。
hostnossl	表示这条记录只接受一个普通的 TCP/IP 套接字连接。	-
DATABASEGUC	声明记录所匹配且允许访问的数据库。	• all：表示该记录匹配所有数据库。 • sameuser：表示如果请求访问的数据库和请求的用户同名，则匹配。 • samerole：表示请求的用户必须是与数据库同名角色中的成员。 • samegroup：与 samerole 作用完全一致，表示请求的用户必须是与数据库同名角色中的成员。 • 一个包含数据库名的文件或者文件中的数据库列表：文件可以通过在文件名前面加前缀@来声明。文件中的数据库列表以逗号或者换行符分隔。 • 特定的数据库名称或者用逗号分隔的数据库列表。 说明 值 replication 表示如果请求一个复制链接，则匹配，但复制链接不表示任何特定的数据库。如需使用名为 replication 的数据库，需在 database 列使用记录“replication”作为数据库名。
USER	声明记录所匹配且允许访问的数据库用户。	• all：表明该记录匹配所有用户。 • +用户角色：表示匹配任何直接或者间接属于这个角色的成员。 说明 +表示前缀符号。 • 一个包含用户名的文件或者文件中的用户列表：文件可以通过在文件名前面加前缀@来声明。文件中的用户列表以逗号或者换行符分隔。 • 特定的数据库用户名或者用逗号分

参数名称	描述	取值范围
		隔的用户列表。
ADDRESS	指定与记录匹配且允许访问的 IP 地址范围。	支持 IPv4 和 IPv6，可以使用如下两种形式来表示： • IP 地址/掩码长度。例如，10.10.0.0/24 • IP 地址子网掩码。例如，10.10.0.0 255.255.255.0 说明 以 IPv4 格式给出的 IP 地址会匹配那些拥有对应地址的 IPv6 连接，比如 127.0.0.1 将匹配 IPv6 地址 ::ffff:127.0.0.1
METHOD	声明连接时使用的认证方法。	本产品支持如下几种认证方式，详细解释请参见表 5-3： • trust • reject • md5（不推荐使用，默认不支持，可通过 password_encryption_type 参数配置） • sha256 • cert • gss（仅用于 Vastbase 内部节点间认证）

表 5-3. 认证方式

认证方式	说明
trust	采用这种认证模式时，本产品只完全信任从服务器本机使用 vsql 且不指定-U 参数的连接，此时不需要口令。 trust 认证对于单用户工作站的本地连接是非常合适和方便的，通常不适用于多用户环境。如果想使用这种认证方法，可利用文件系统权限限制对服务器的 Unix 域套接字文件的访问。要使用这种限制有两个方法： • 设置参数 unix_socket_permissions 和 unix_socket_group。

认证方式	说明
	• 设置参数 unix_socket_directory，将 Unix 域套接字文件放在一个经过恰当限制的目录里。 须知 设置文件系统权限只能 Unix 域套接字连接，它不会限制本地 TCP/IP 连接。为保证本地 TCP/IP 安全，Vastbase 不允许远程连接使用 trust 认证方法。
reject	无条件地拒绝连接。常用于过滤某些主机。
md5	要求客户端提供一个 md5 加密的口令进行认证。 须知 不推荐使用 md5 认证，因为 md5 为不安全的加密算法，存在网络安全风险。Vastbase 保留 md5 认证和密码存储，是为了便于第三方工具的使用（比如 TPCC 评测工具）。
sha256	要求客户端提供一个 sha256 算法加密的口令进行认证，该口令在传送过程中结合 salt（服务器发送给客户端的随机数）的单向 sha256 加密，增强了安全性。
cert	客户端证书认证模式，此模式需进行 SSL 连接配置且需要客户端提供有效的 SSL 证书，不需要提供用户密码。 须知 该认证方式只支持 hostssl 类型的规则。
gss	使用基于 gssapi 的 kerberos 认证。 须知 该认证方式依赖 kerberos server 等组件，仅支持 Vastbase 内部通信认证。当前版本暂不支持外部客户端通过 kerberos 认证连接。 开启 Vastbase 内部 kerberos 认证会使增加内部节点建连时间，即影响首次涉及内部建连的 SQL 操作性能，内部连接建立好后， 后续操作不受影响。

5.1.3. SSL 证书管理

Vastbase 默认配置了通过 openssl 生成的安全证书、私钥。并且提供证书替换的接口，方便用户进行证书的替换。

5.1.3.1. 证书生成

操作场景

在测试环境下，用户可以用通过以下方式进行数字证书测试。在客户的运行环境中，请使用从 CA 认证中心申请的数字证书。

前提条件

Linux 环境安装了 openssl 组件。

自认证证书生成过程

步骤 10 搭建 CA 环境。

```
--假设用户为 vastbase 已存在，搭建 CA 的路径为 test
--以 root 用户身份登录 Linux 环境，切换到用户 vastbase
mkdir test
cd /etc/pki/tls
--copy 配置文件 openssl.cnf 到 test 下
cp openssl.cnf ~/test
cd ~/test
--到 test 文件夹下，开始搭建 CA 环境
--创建文件夹 demoCA./demoCA/newcerts./demoCA/private
mkdir ./demoCA ./demoCA/newcerts ./demoCA/private
chmod 777 ./demoCA/private
--创建 serial 文件，写入 01
echo '01'>./demoCA/serial
--创建文件 index.txt
touch ./demoCA/index.txt
--修改 openssl.cnf 配置文件中的参数
dir  = ./demoCA
default_md     = sha256
--至此 CA 环境搭建完成
```

步骤 11 生成根私钥。

```
--生成 CA 私钥
openssl genrsa -aes256 -out demoCA/private/cakey.pem 2048
Generating RSA private key, 2048 bit long modulus
.................+++
.................+++
e is 65537 (0x10001)
--设置根私钥的保护密码，最少要求 4 个字符，假设为 Test@123
Enter pass phrase for demoCA/private/cakey.pem:
--再次输入私钥密码 Test@123
Verifying - Enter pass phrase for demoCA/private/cakey.pem:
```

步骤 12 生成根证书请求文件。

```
--生成 CA 根证书申请文件 server.req
openssl req -config openssl.cnf -new -key demoCA/private/cakey.pem -out demoCA/careq.pem
Enter pass phrase for demoCA/private/cakey.pem:
--输入根私钥密码 Test@123
You are about to be asked to enter information that will be incorporated
into your certificate request.
What you are about to enter is what is called a Distinguished Name or a DN.
```

```
There are quite a few fields but you can leave some blank
For some fields there will be a default value,
If you enter '.', the field will be left blank.
-----

--以下名称请牢记，生成服务器证书和客户端证书时填写的信息需要与此处的一致
Country Name (2 letter code) [AU]:CN
State or Province Name (full name) [Some-State]:shanxi
Locality Name (eg, city) []:xian
Organization Name (eg, company) [Internet Widgits Pty Ltd]:Abc
Organizational Unit Name (eg, section) []:hello
--Common Name 可以随意命名
Common Name (eg, YOUR name) []:world
--Email 可以选择性填写
Email Address []:

Please enter the following 'extra' attributes
to be sent with your certificate request
A challenge password []:
An optional company name []:
```

步骤 13 生成自签发根证书。

```
--生成根证书时，需要修改 openssl.cnf 文件，设置 basicConstraints=CA:TRUE
vi openssl.cnf
--生成 CA 自签发根证书
openssl ca -config openssl.cnf -out demoCA/cacert.pem -keyfile demoCA/private/cakey.pem
-selfsign -infiles demoCA/careq.pem
Using configuration from openssl.cnf
Enter pass phrase for demoCA/private/cakey.pem:
--输入根私钥密码 Test@123
Check that the request matches the signature
Signature ok
Certificate Details:
       Serial Number: 1 (0x1)
       Validity
          Not Before: Feb 28 02:17:11 2017 GMT
          Not After : Feb 28 02:17:11 2018 GMT
       Subject:
          countryName             = CN
          stateOrProvinceName      = shanxi
          organizationName        = Abc
          organizationalUnitName    = hello
          commonName              = world
       X509v3 extensions:
          X509v3 Basic Constraints:
             CA:FALSE
          Netscape Comment:
             OpenSSL Generated Certificate
          X509v3 Subject Key Identifier:
             F9:91:50:B2:42:8C:A8:D3:41:B0:E4:42:CB:C2:BE:8D:B7:8C:17:1F
          X509v3 Authority Key Identifier:
             keyid:F9:91:50:B2:42:8C:A8:D3:41:B0:E4:42:CB:C2:BE:8D:B7:8C:17:1F

Certificate is to be certified until Feb 28 02:17:11 2018 GMT (365 days)
Sign the certificate? [y/n]:y
```

```
1 out of 1 certificate requests certified, commit? [y/n]y
Write out database with 1 new entries
Data Base Updated
--至此 CA 根证书自签发完成，根证书 demoCA/cacert.pem。
```

步骤 14 生成服务端证书私钥。

```
--生成服务器私钥文件 server.key
openssl genrsa -aes256 -out server.key 2048
Generating a 2048 bit RSA private key
.......++++++
..++++++
e is 65537 (0x10001)
Enter pass phrase for server.key:
--服务器私钥的保护密码，最少要求 4 个字符，假设为 Test@123
Verifying - Enter pass phrase for server.key:
--再次确认服务器私钥的保护密码，即为 Test@123
```

步骤 15 生成服务端证书请求文件。

```
--生成服务器证书请求文件 server.req
openssl req -config openssl.cnf -new -key server.key -out server.req
Enter pass phrase for server.key:
You are about to be asked to enter information that will be incorporated
into your certificate request.
What you are about to enter is what is called a Distinguished Name or a DN.
There are quite a few fields but you can leave some blank
For some fields there will be a default value,
If you enter '.', the field will be left blank.
-----

--以下填写的信息与创建 CA 时的信息一致
Country Name (2 letter code) [AU]:CN
State or Province Name (full name) [Some-State]:shanxi
Locality Name (eg, city) []:xian
Organization Name (eg, company) [Internet Widgits Pty Ltd]:Abc
Organizational Unit Name (eg, section) []:hello
--Common Name 可以随意命名
Common Name (eg, YOUR name) []:world
Email Address []:
--以下信息可以选择性填写
Please enter the following 'extra' attributes
to be sent with your certificate request
A challenge password []:
An optional company name []:
```

步骤 16 生成服务端证书。

```
--生成服务端/客户端证书时，修改 openssl.cnf 文件，设置 basicConstraints=CA:FALSE
vi openssl.cnf
--修改 demoCA/index.txt.attr 中属性为 no。
vi demoCA/index.txt.attr

--对生成的服务器证书请求文件进行签发，签发后将生成正式的服务器证书 server.crt
openssl ca  -config openssl.cnf -in server.req -out server.crt -days 3650 -md sha256
Using configuration from /etc/ssl/openssl.cnf
```

```
Enter pass phrase for ./demoCA/private/cakey.pem:
Check that the request matches the signature
Signature ok
Certificate Details:
      Serial Number: 2 (0x2)
      Validity
         Not Before: Feb 27 10:11:12 2017 GMT
         Not After : Feb 25 10:11:12 2027 GMT
      Subject:
         countryName            = CN
         stateOrProvinceName      = shanxi
         organizationName         = Abc
         organizationalUnitName    = hello
         commonName             = world
      X509v3 extensions:
         X509v3 Basic Constraints:
            CA:FALSE
         Netscape Comment:
            OpenSSL Generated Certificate
         X509v3 Subject Key Identifier:
            EB:D9:EE:C0:D2:14:48:AD:EB:BB:AD:B6:29:2C:6C:72:96:5C:38:35
         X509v3 Authority Key Identifier:
            keyid:84:F6:A1:65:16:1F:28:8A:B7:0D:CB:7E:19:76:2A:8B:F5:2B:5C:6A

Certificate is to be certified until Feb 25 10:11:12 2027 GMT (3650 days)
--选择 y 对证书进行签发
Sign the certificate? [y/n]:y

--选择 y，证书签发结束
1 out of 1 certificate requests certified, commit? [y/n]y
Write out database with 1 new entries
Data Base Updated
```

去掉私钥密码保护，方法如下：

```
--去掉服务器私钥的密码保护
openssl rsa -in server.key -out server.key
--如果不去掉服务器私钥的密码保护需要使用 vb_guc 工具对存储密码进行加密保护
vb_guc encrypt -M server -K Test@123 -D ./
--vb_guc 加密后会生成 server.key.cipher,server.key.rand 两个私钥密码保护文件
```

步骤 17 客户端证书，私钥的生成。

生成客户端证书和客户端私钥的方法和要求与服务器相同。

```
--生成客户端私钥
openssl genrsa -aes256 -out client.key 2048
--生成客户端证书请求文件
openssl req -config openssl.cnf -new -key client.key -out client.req
--对生成的客户端证书请求文件进行签发，签发后将生成正式的客户端证书 client.crt
openssl ca -config openssl.cnf -in client.req -out client.crt -days 3650 -md sha256
```

去掉私钥密码保护，方法如下：

```
--去掉客户端私钥的密码
openssl rsa -in client.key -out client.key
--如果不去掉客户端私钥的密码保护需要使用 vb_guc 工具对存储密码进行加密保护
vb_guc encrypt -M client -K Test@123 -D ./
vb_guc 加密后会生成 client.key.cipher,client.key.rand 两个私钥密码保护文件。
```

将客户端密钥转化为 DER 格式，方法如下：

```
openssl pkcs8 -topk8 -outform DER -in client.key -out client.key.pk8 -nocrypt
```

步骤 18 吊销证书列表的生成。

如果需要吊销列表，可按照如下方法生成：

```
--首先创建 crlnumber 文件
echo '00'>./demoCA/crlnumber
--吊销服务器证书
openssl ca -config openssl.cnf -revoke server.crt
--生成证书吊销列表 sslcrl-file.crl
openssl ca -config openssl.cnf -gencrl -out sslcrl-file.crl
```

5.1.4. 用 SSL 进行安全的 TCP/IP 连接

背景信息

Vastbase 支持 SSL 标准协议（TLS 1.2），SSL 协议是安全性更高的协议标准，它们加入了数字签名和数字证书来实现客户端和服务器的双向身份验证，保证了通信双方更加安全的数据传输。

前提条件

从 CA 认证中心申请到正式的服务器、客户端的证书和密钥。（假设服务器的私钥为 server.key，证书为 server.crt，客户端的私钥为 client.key，证书为 client.crt，CA 根证书名称为 cacert.pem。）

注意事项

- 当用户远程连接到数据库主节点时，需要使用 sha256 的认证方式。
- 当内部服务器之间连接时，需要使用 trust 的认证方式，支持 IP 白名单认证。

操作步骤

Vastbase 部署完成后，用户需要配置客户端的相关参数。配置 SSL 认证相关的数字证书参数，具体要求请参见表 5-4。

- 配置客户端参数。

  已从 CA 认证中心申请到客户端默认证书，私钥，根证书以及私钥密码加密文件。假设证书、私钥和根证书都放在“/home/vastbase”目录。

  双向认证需配置如下参数：

  ```
  export PGSSLCERT="/home/vastbase/client.crt"
  export PGSSLKEY="/home/vastbase/client.key"
  export PGSSLMODE="verify-ca"
  export PGSSLROOTCERT="/home/vastbase/cacert.pem"
  ```

  单向认证需要配置如下参数：

```
export PGSSLMODE="verify-ca"
export PGSSLROOTCERT="/home/vastbase/cacert.pem"
```

❖ 修改客户端密钥的权限。

客户端根证书，密钥，证书以及密钥密码加密文件的权限，需保证权限为 600。如果权限不满足要求，则客户端无法以 SSL 连接到 Vastbase。

```
chmod 600 client.key
chmod 600 client.crt
chmod 600 client.key.cipher
chmod 600 client.key.rand
chmod 600 cacert.pem
```

须知

从安全性考虑，建议使用双向认证方式。

配置客户端环境变量，必须包含文件的绝对路径。

表 5-4. 认证方式

认证方式	含义	配置客户端环境变量	维护建议
双向认证 (推荐)	客户端验证服务器证书的有效性，同时服务器端也要验证客户端证书的有效性，只有认证成功，连接才能建立。	设置如下环境变量： • PGSSLCERT • PGSSLKEY • PGSSLROOTCERT • PGSSLMODE	该方式应用于安全性要求较高的场景。使用此方式时，建议设置客户端的 PGSSLMODE 变量为 verify-ca。确保了网络数据的安全性。
单向认证	客户端只验证服务器证书的有效性，而服务器端不验证客户端证书的有效性。服务器加载证书信息并发送给客户端，客户端使用根证书来验证服务器端证书的有效性。	设置如下环境变量： • PGSSLROOTCERT • PGSSLMODE	为防止基于 TCP 链接的欺骗，建议使用 SSL 证书认证功能。除配置客户端根证书外，建议客户端使用 PGSSLMODE 变量为 verify-ca 方式连接。

相关参考

在服务器端的 postgresql.conf 文件中配置相关参数，详细信息请参见表 5-5。

表 5-5. 服务器参数

参数	描述	取值范围
ssl	表示是否启动 SSL 功能。	• on：开启 SSL 功能。 • off：关闭 SSL 功能。 默认值：on
require_ssl	设置服务器端是否强制要求 SSL 连接。该参数只有当参数 ssl 为 on 时才有效。	• on：服务器端强制要求 SSL 连接。 • off：服务器端对是否通过 SSL 连接不作强制要求。 默认值：off
ssl_cert_file	指定服务器证书文件，包含服务器端的公钥。服务器证书用以表明服务器身份的合法性，公钥将发送给对端用来对数据进行加密。	请以实际的证书名为准。必须使用绝对路径。 默认值：server.crt
ssl_key_file	指定服务器私钥文件，用以对公钥加密的数据进行解密。	请以实际的服务器私钥名称为准。必须使用绝对路径。 默认值：server.key
ssl_ca_file	CA 服务器的根证书。此参数可选择配置，需要验证客户端证书的合法性时才需要配置。	请以实际的 CA 服务器根证书名称为准。 默认值：cacert.pem
ssl_crl_file	证书吊销列表，如果客户端证书在该列表中，则当前客户端证书被视为无效证书。	请以实际的证书吊销列表名称为准。 默认值：空，表示没有吊销列表。
ssl_ciphers	SSL 通讯使用的加密算法。	本产品支持的加密算法的详细信息请参见表 5-7。 默认值：ALL，表示允许对端使用产品支持的所有加密算法，但不包含 ADH、LOW、EXP、MD5 算法。

在客户端配置 SSL 认证相关的环境变量，详细信息请参见表 5-6。

说明

客户端环境变量的路径以“*/home/vastbase*”为例，在实际操作中请使用实际路径进行替换。

表 5-6. 客户端参数

环境变量	描述	取值范围
PGSSLCERT	指定客户端证书文件，包含客户端的公钥。客户端证书用以表明客户端身份的合法性，公钥将发送给对端用来对数据进行加密。	必须包含文件的绝对路径，如： **export PGSSLCERT='*/home/vastbase/client.crt*'** 默认值：空
PGSSLKEY	指定客户端私钥文件，用以对公钥加密的数据进行解密。	必须包含文件的绝对路径，如： **export PGSSLKEY='*/home/vastbase/client.key*'** 默认值：空
PGSSLMODE	设置是否和服务器进行 SSL 连接协商，以及指定 SSL 连接的优先级。	取值及含义： • disable：只尝试非 SSL 连接。 • allow：首先尝试非 SSL 连接，如果连接失败，再尝试 SSL 连接。 • prefer：首先尝试 SSL 连接，如果连接失败，将尝试非 SSL 连接。 • require：只尝试 SSL 连接。如果存在 CA 文件，则按设置成 verify-ca 的方式验证。 • verify-ca：只尝试 SSL 连接，并且验证服务器是否具有由可信任的证书机构签发的证书。 • verify-full：只尝试 SSL 连接，并且验证服务器是否具有由可信任的证书机构签发的证书，以及验证服务器主机名是否与证书中的一致。 默认值：prefer
PGSSLROOTCERT	指定为客户端颁发证书的根证书文件，根证书用于验证服务器证书的有效性。	必须包含文件的绝对路径，如： **export PGSSLROOTCERT='*/home/vastbase/certca.pem*'** 默认值：空
PGSSLCRL	指定证书吊销列表文件，用于验证服	必须包含文件的绝对路径，如：

环境变量	描述	取值范围
	务器证书是否在废弃证书列表中，如果在，则服务器证书将会被视为无效证书。	**export PGSSLCRL='***/home/vastbase/sslcrl-file.crl***'** 默认值：空

服务器端参数 ssl、require_ssl 与客户端参数 sslmode 配置组合结果如下：

SSL（服务器）	SSLMODE（客户端）	REQUIRE_SSL（服务器）	结果
on	disable	on	由于服务器端要求使用 SSL，但客户端针对该连接禁用了 SSL，因此无法建立连接。
	disable	off	连接未加密。
	allow	on	连接经过加密。
	allow	off	连接未加密。
	prefer	on	连接经过加密。
	prefer	off	连接经过加密。
	require	on	连接经过加密。
	require	off	连接经过加密。
	verify-ca	on	连接经过加密，且验证了服务器证书。
	verify-ca	off	连接经过加密，且验证了服务器证书。
	verify-full	on	连接经过加密，且验证了服务器证书和主机名。
	verify-full	off	连接经过加密，且验证了服务器证书和主机名。
off	disable	on	连接未加密。
	disable	off	连接未加密。
	allow	on	连接未加密。
	allow	off	连接未加密。
	prefer	on	连接未加密。
	prefer	off	连接未加密。

SSL（服务器）	SSLMODE（客户端）	REQUIRE_SSL（服务器）	结果
	require	on	由于客户端要求使用 SSL，但服务器端禁用了 SSL，因此无法建立连接。
	require	off	由于客户端要求使用 SSL，但服务器端禁用了 SSL，因此无法建立连接。
	verify-ca	on	由于客户端要求使用 SSL，但服务器端禁用了 SSL，因此无法建立连接。
	verify-ca	off	由于客户端要求使用 SSL，但服务器端禁用了 SSL，因此无法建立连接。
	verify-full	on	由于客户端要求使用 SSL，但服务器端禁用了 SSL，因此无法建立连接。
	verify-full	off	由于客户端要求使用 SSL，但服务器端禁用了 SSL，因此无法建立连接。

SSL 传输支持一系列不同强度的加密和认证算法。用户可以通过修改 postgresql.conf 中的 ssl_ciphers 参数指定数据库服务器使用的加密算法。目前本产品 SSL 支持的加密算法如表 5-7 所示。

表 5-7. 加密算法

加密强度	安全程度	加密算法描述
stronger	high	DHE-RSA-AES256-GCM-SHA384
stronger	high	DHE-RSA-AES128-GCM-SHA256
stronger	high	DHE-DSS-AES256-GCM-SHA384
stronger	high	DHE-DSS-AES128-GCM-SHA256
stronger	medium	DHE-RSA-AES256-SHA256
stronger	medium	DHE-RSA-AES128-SHA256
stronger	medium	DHE-DSS-AES256-SHA256
stronger	medium	DHE-DSS-AES128-SHA256
stronger	high	DHE-RSA-AES256-CCM
stronger	high	DHE-RSA-AES128-CCM
stronger	medium	DHE-RSA-AES256-SHA
stronger	medium	DHE-RSA-AES128-SHA

加密强度	安全程度	加密算法描述
stronger	medium	DHE-DSS-AES256-SHA
stronger	medium	DHE-DSS-AES128-SHA

说明

- SSL 目前只支持加密强度在 strong 以上的加密算法。
- 配置参数 ssl_ciphers 的默认值为 ALL，表示支持上表中的所有加密算法。如果对加密算法没有特殊要求，建议用户使用该默认值。
- 如指定以上多种加密，加密算法之间需要使用分号分割。

  如在 postgresql.conf 设置

  ssl_ciphers='DHE-RSA-AES256-GCM-SHA384;DHE-RSA-AES256-SHA256;DHE-RSA-AES256-CCM'
- 如果要使用上表中和 DSS 相关的加密算法（如 DHE-DSS-AES256-GCM-SHA384、DHE-DSS-AES256-SHA256、DHE-DSS-AES256-SHA 等）必须加载使用 DSA 算法签名的证书文件。如何使用 openssl 产生 DSA 算法签名的证书文件，请参见 openssl 官方文档。
- SSL 连接认证不仅增加了登录（创建 SSL 环境）及退出过程（清理 SSL 环境）的时间消耗，同时需要消耗额外的时间用于加解密所需传输的内容，因此对性能有一定影响。特别的，对于频繁的登录登出，短时查询等场景有较大的影响。
- 在证书有效期小于 7 天的时候，连接登录会在日志中产生告警提醒。

5.1.5. 查看数据库连接数

背景信息

当用户连接数达到上限后，无法建立新的连接。因此，当数据库管理员发现某用户无法连接到数据库时，需要查看是否连接数达到了上限。控制数据库连接的主要以下几种选项。

- 全局的最大连接数：由运行参数 max_connections 指定，默认值为 5000。
- 某用户的连接数：在创建用户时由 CREATE ROLE 命令的 CONNECTION LIMIT connlimit 子句直接设定，也可以在设定以后用 ALTER ROLE 的 CONNECTION LIMIT connlimit 子句修改。
- 某数据库的连接数：在创建数据库时，由 CREATE DATABASE 的 CONNECTION LIMIT connlimit 参数指定。

操作步骤

步骤 1 以操作系统用户 vastbase 登录数据库主节点。

步骤 2 使用如下命令连接数据库。

```
vsql -d vastbase -p 5432
```

vastbase 为需要连接的数据库名称，5432 为数据库主节点的端口号。

连接成功后，系统显示类似如下信息：

```
vsql ((Vastbase 1.0 build 290d125f) compiled at 2020-05-08 02:59:43 commit 2143 last mr 131)
Non-SSL connection (SSL connection is recommended when requiring high-security)
```

```
Type "help" for help.

vastbase=#
```

步骤 3 查看全局会话连接数限制。

```
vastbase=# SHOW max_connections;
 max_connections
-----------------
 800
(1 row)
```

其中 800 是最大会话连接数。

步骤 4 查看已使用的会话连接数。

具体命令请参见表 5-8。

须知

除了创建的时候用双引号引起的数据库和用户名称外，以下命令中用到的数据库名称和用户名称，其中包含的英文字母必须使用小写。

表 5-8. 查看会话连接数

描述	命令
查看指定用户的会话连接数上限。	执行如下命令查看连接到指定用户 vastbase 的会话连接数上限。其中-1 表示没有对用户 vastbase 设置连接数的限制。 vastbase=# **SELECT ROLNAME,ROLCONNLIMIT FROM PG_ROLES WHERE ROLNAME='vastbase';** rolname | rolconnlimit ----------+-------------- vastbase | -1 (1 row)
查看指定用户已使用的会话连接数。	执行如下命令查看指定用户 vastbase 已使用的会话连接数。其中，1 表示 vastbase 已使用的会话连接数。 vastbase=# CREATE OR REPLACE VIEW DV_SESSIONS AS SELECT sa.sessionid AS SID, 0::integer AS SERIAL#, sa.usesysid AS USER#, ad.rolname AS USERNAME FROM pg_stat_get_activity(NULL) AS sa LEFT JOIN pg_authid ad ON(sa.usesysid = ad.oid) WHERE sa.application_name <> 'JobScheduler'; vastbase=# **SELECT COUNT(*) FROM DV_SESSIONS WHERE USERNAME='vastbase';** count -------

描述	命令
	1 (1 row)
查看指定数据库的会话连接数上限。	执行如下命令查看连接到指定数据库 vastbase 的会话连接数上限。其中 -1 表示没有对数据库 vastbase 设置连接数的限制。 vastbase=# **SELECT DATNAME,DATCONNLIMIT FROM PG_DATABASE WHERE DATNAME='vastbase';** datname | datconnlimit ----------+-------------- vastbase | -1 (1 row)
查看指定数据库已使用的会话连接数。	执行如下命令查看指定数据库 vastbase 上已使用的会话连接数。其中，1 表示数据库 vastbase 上已使用的会话连接数。 vastbase=# **SELECT COUNT(*) FROM PG_STAT_ACTIVITY WHERE DATNAME='vastbase';** count ------- 1 (1 row)
查看所有用户已使用会话连接数。	执行如下命令查看所有用户已使用的会话连接数。 vastbase=# CREATE OR REPLACE VIEW DV_SESSIONS AS SELECT sa.sessionid AS SID, 0::integer AS SERIAL#, sa.usesysid AS USER#, ad.rolname AS USERNAME FROM pg_stat_get_activity(NULL) AS sa LEFT JOIN pg_authid ad ON(sa.usesysid = ad.oid) WHERE sa.application_name <> 'JobScheduler'; vastbase=# **SELECT COUNT(*) FROM DV_SESSIONS;** count ------- 10 (1 row)

5.2. 管理用户及权限

5.2.1. 默认权限机制

数据库对象创建后，进行对象创建的用户就是该对象的所有者。Vastbase 安装后的默认情况下，未开启 5.2.3 三权分立，数据库系统管理员具有与对象所有者相同的权限。也就是说对象创建后，默认只有对象所有者或者系统管理员可以查询、修改和销毁对象，以及通过 11.16.88GRANT 将对象的权限授予其他用户。

为使其他用户能够使用对象，必须向用户或包含该用户的角色授予必要的权限。

Vastbase 支持以下的权限：SELECT、INSERT、UPDATE、DELETE、TRUNCATE、REFERENCES、CREATE、CONNECT、EXECUTE 和 USAGE。不同的权限与不同的对象类型关联。有关各权限的详细信息，请参见 11.16.88GRANT。

要撤消已经授予的权限，可以使用 11.16.99REVOKE。对象所有者的权限（例如 ALTER、 DROP、GRANT 和 REVOKE）是隐式的，无法授予或撤消。即只要拥有对象就可以执行对象所有者的这些隐式权限。对象所有者可以撤消自己的普通权限，例如，使表对自己以及其他人只读。

系统表和系统视图要么只对系统管理员可见，要么对所有用户可见。标识了需要系统管理员权限的系统表和视图只有系统管理员可以查询。 有关信息，请参考 14 系统表和系统视图。

数据库提供对象隔离的特性，对象隔离特性开启时，用户只能查看有权限访问的对象(表、视图、字段、函数)，系统管理员不受影响。有关信息，请参考 11.16.2ALTER DATABASE。

5.2.2. 管理员

初始用户

Vastbase 安装过程中自动生成的帐户称为初始用户。初始用户也是系统管理员和安全策略管理员，拥有系统的最高权限，能够执行所有的操作。该帐户与进行 Vastbase 安装的操作系统用户同名。

初始用户会绕过所有权限检查。建议仅将此初始用户作为 DBA 管理用途，而非业务应用。

系统管理员

系统管理员是指具有 SYSADMIN 属性的帐户，默认安装情况下具有与对象所有者相同的权限，但不包括 dbe_perf 模式的对象权限。

要创建新的系统管理员，请以初始用户或者系统管理员用户身份连接数据库，并使用带 SYSADMIN 选项的 11.16.56CREATE USER 语句或 11.16.24ALTER USER 语句进行设置。

```
vastbase=# CREATE USER sysadmin WITH SYSADMIN password "Bigdata@123";
```

或者

```
vastbase=# ALTER USER joe SYSADMIN;
```

ALTER USER 时，要求用户已存在。

5.2.3. 三权分立

5.2.1 默认权限机制和 5.2.2 管理员两节的描述基于的是 Vastbase 创建之初的默认情况。从前面的介绍可以看出，默认情况下拥有 SYSADMIN 属性的系统管理员，具备系统最高权限。

在实际业务管理中，为了避免系统管理员拥有过度集中的权利带来高风险，可以设置三权分立。将系统管理员的部分权限分立给安全管理员和审计管理员，形成系统管理员、安全管理员和审计管理员三权分立。

三权分立后，系统管理员将不再具有 SSOADMIN 属性（安全管理员）和 AUDITADMIN 属性（审计管理员）能力。即不再拥有创建角色和用户的权限，并不再拥有查看和维护数据库审计日志的权限。关于 SSOADMIN 属性和 AUDITADMIN 属性的更多信息请参考 11.16.44CREATE ROLE。

三权分立后，系统管理员只会对自己作为所有者的对象有权限。

初始用户的权限不受三权分立设置影响。因此建议仅将此初始用户作为 DBA 管理用途，而非业务应用。

在数据库初始化阶段，新增三个默认用户：管理员 vbadmin、审计员 vbaudit 和安全员 vbsso。三个默认用户的关键属性（pg_authid 数据字典）如下表：

用户属性	管理员	审计员	安全员
ROLSUPER	TRUE	FALSE	FALSE
ROLINHERIT	TRUE	TRUE	TRUE
ROLCREATEROLE	TRUE	FALSE	FALSE
ROLCREATEDB	TRUE	FALSE	FALSE
ROLCATUPDATE	TRUE	FALSE	FALSE
ROLCANLOGIN	TRUE	TRUE	TRUE
ROLREPLICATION	TRUE	FALSE	FALSE
ROLSSOADMIN	FALSE	FALSE	TRUE
ROLAUDITADMIN	FALSE	TRUE	FALSE
ROLSYSADMIN	TRUE	FALSE	FALSE
ROLUSEFT	TRUE	TRUE	FALSE

Vastbase 可以通过 ALTER ROLE/USER 命令修改用户属性。在启用三权分立功能情况下，不允许修改其他类型用户的属性，不允许修改用户属性使之同时具备安全员、审计员和管理员中两种或以上权限。例如不允许管理员修改安全员的用户属性。

新增安全员属性，相关语法示例如下：

```
CREATE USER user1 PASSWORD 'user1@abcd' ssoadmin;
```

以上 SQL 语句创建了一个名称为 user1 的安全员。

不允许同时指定 sysadmin、auditadmin 和 ssoadmin 中两种或以上属性，若用户属性为 auditadmin 或者 ssoadmin，则不允许指定 createrole 和 createdb 属性。

例如以下创建用户语句是不合法的：

```
CREATE USER user1 PASSWORD 'user1@abcd' ssoadmin createrole;
```

5.2.4. 用户

使用 CREATE USER 和 ALTER USER 可以创建和管理数据库用户。Vastbase 包含一个或多个已命名数据库。用户和角色在整个 Vastbase 范围内是共享的，但是其数据并不共享。即用户可以连接任何数据库，但当连接成功后，任何用户都只能访问连接请求里声明的那个数据库。

非 5.2.3 三权分立下，Vastbase 用户帐户只能由系统管理员或拥有 CREATEROLE 属性的安全管理员创建和删除。三权分立时，用户帐户只能由初始用户和安全管理员创建。

在用户登录 Vastbase 时会对其进行身份验证。用户可以拥有数据库和数据库对象（例如表），并且可以向用户和角色授予对这些对象的权限以控制谁可以访问哪个对象。除系统管理员外，具有 CREATEDB 属性的用户可以创建数据库并授予对这些数据库的权限。

创建、修改和删除用户

- 要创建用户，请使用 SQL 语句 11.16.56CREATE USER。

  例如：创建用户 joe，并设置用户拥有 CREATEDB 属性。

  ```
  vastbase=# CREATE USER joe WITH CREATEDB PASSWORD "Bigdata@123";
  CREATE ROLE
  ```

- 要创建系统管理员，请使用带有 SYSADMIN 选项的 11.16.56CREATE USER 语句 。
- 要删除现有用户，请使用 11.16.82DROP USER。
- 要更改用户帐户（例如，重命名用户或更改密码），请使用 11.16.24ALTER USER。
- 要查看用户列表，请查询视图 14.3.71PG_USER：

  ```
  vastbase=# SELECT * FROM pg_user;
  ```

- 要查看用户属性，请查询系统表 14.2.14PG_AUTHID：

  ```
  vastbase=# SELECT * FROM pg_authid;
  ```

私有用户

对于有多个业务部门，各部门间使用不同的数据库用户进行业务操作，同时有一个同级的数据库维护部门使用数据库管理员进行维护操作的场景下，业务部门可能希望在未经授权的情况下，管理员用户只能对各部门的数据进行控制操作 (DROP、ALTER、TRUNCATE) , 但是不能进行访问操作 (INSERT、DELETE、UPDATE、SELECT、COPY）。即针对管理员用户，表对象的控制权和访问权要能够分离，提高普通用户数据安全性。

5.2.3 三权分立情况下，管理员对其他用户放在属于各自模式下的表无权限。但是，这种无权限包含了无控制权限，因此不能满足上面的诉求。为此，Vastbase 提供了私有用户方案。即在非三权分立模式下，创建具有 INDEPENDENT 属性的私有用户。

```
vastbase=# CREATE USER user_independent WITH INDEPENDENT IDENTIFIED BY "1234@abc";
```

针对该用户的对象，系统管理员和拥有 CREATEROLE 属性的安全管理员在未经其授权前，只能进行控制操作（DROP、ALTER、TRUNCATE），无权进行 INSERT、DELETE、SELECT、UPDATE、COPY、GRANT、REVOKE、ALTER OWNER 操作。

5.2.5. 角色

角色是一组用户的集合。通过 GRANT 把角色授予用户后，用户即具有了角色的所有权限。推荐使用角色进行高效权限分配。例如，可以为设计、开发和维护人员创建不同的角色，将角色 GRANT 给用户后，再向每个角色中的用户授予其工作所需数据的差异权限。在角色级别授予或撤消权限时，这些更改将作用到角色下的所有成员。

Vastbase 提供了一个隐式定义的拥有所有角色的组 PUBLIC，所有创建的用户和角色默认拥有 PUBLIC 所拥有的权限。关于 PUBLIC 默认拥有的权限请参考 11.16.88GRANT。要撤销或重新授予用户和角色对 PUBLIC 的权限， 可通过在 GRANT 和 REVOKE 指定关键字 PUBLIC 实现。

要查看所有角色，请查询系统表 PG_ROLES：

```
SELECT * FROM PG_ROLES;
```

创建、修改和删除角色

非 5.2.3 三权分立时，只有系统管理员和具有 CREATEROLE 属性的用户才能创建、修改或删除角色。三权分立下，只有初始用户和具有 CREATEROLE 属性的用户才能创建、修改或删除角色。

- 要创建角色，请使用 11.16.44CREATE ROLE。
- 要在现有角色中添加或删除用户，请使用 11.16.10ALTER ROLE。
- 要删除角色，请使用 11.16.72DROP ROLE。DROP ROLE 只会删除角色，并不会删除角色中的成员用户帐户。

5.2.6. Schema

Schema 又称作模式。通过管理 Schema，允许多个用户使用同一数据库而不相互干扰，可以将数据库对象组织成易于管理的逻辑组，同时便于将第三方应用添加到相应的 Schema 下而不引起冲突。

每个数据库包含一个或多个 Schema。数据库中的每个 Schema 包含表和其他类型的对象。数据库创建初始，默认具有一个名为 public 的 Schema，且所有用户都拥有此 Schema 的权限。可以通过 Schema 分组数据库对象。Schema 类似于操作系统目录，但 Schema 不能嵌套。

相同的数据库对象名称可以应用在同一数据库的不同 Schema 中，而没有冲突。例如，a_schema 和 b_schema 都可以包含名为 mytable 的表。具有所需权限的用户可以访问数据库的多个 Schema 中的对象。

在数据库创建用户时，系统会自动帮助用户创建一个同名 Schema。

数据库对象是创建在数据库搜索路径中的第一个 Schema 内的。有关默认情况下的第一个 Schema 情况及如何变更 Schema 顺序等更多信息，请参见搜索路径。

创建、修改和删除 Schema

❖ 要创建 Schema，请使用 11.16.45CREATE SCHEMA。默认初始用户和系统管理员可以创建 Schema，其他用户需要具备数据库的 CREATE 权限才可以在该数据库中创建 Schema，赋权方式请参考 11.16.88GRANT 中将数据库的访问权限赋予指定的用户或角色中的语法。

❖ 要更改 Schema 名称或者所有者，请使用 11.16.12ALTER SCHEMA。Schema 所有者可以更改 Schema。

❖ 要删除 Schema 及其对象，请使用 11.16.73DROP SCHEMA。Schema 所有者可以删除 Schema。

❖ 要在 Schema 内创建表，请以 schema_name.table_name 格式创建表。不指定 schema_name 时，对象默认创建到搜索路径中的第一个 Schema 内。

❖ 要查看 Schema 所有者，请对系统表 PG_NAMESPACE 和 PG_USER 执行如下关联查询。语句中的 schema_name 请替换为实际要查找的 Schema 名称。

```
vastbase=# SELECT s.nspname,u.usename AS nspowner FROM pg_namespace s, pg_user u WHERE
nspname='schema_name' AND s.nspowner = u.usesysid;
```

❖ 要查看所有 Schema 的列表，请查询 PG_NAMESPACE 系统表。

```
vastbase=# SELECT * FROM pg_namespace;
```

❖ 要查看属于某 Schema 下的表列表，请查询系统视图 PG_TABLES。例如，以下查询会返回 Schema PG_CATALOG 中的表列表。

```
vastbase=# SELECT distinct(tablename),schemaname from pg tables where schemaname =
'pg_catalog';
```

搜索路径

搜索路径定义在 search_path 参数中，参数取值形式为采用逗号分隔的 Schema 名称列表。如果创建对象时未指定目标 Schema，则将该对象会被添加到搜索路径中列出的第一个 Schema 中。当不同 Schema 中存在同名的对象时，查询对象未指定 Schema 的情况下，将从搜索路径中包含该对象的第一个 Schema 中返回对象。

❖ 要查看当前搜索路径，请使用 11.16.110SHOW。

```
vastbase=# SHOW SEARCH_PATH;
 search_path
----------------
 "$user",public
(1 row)
```

search_path 参数的默认值为："$user"，public。$user 表示与当前会话用户名同名的 Schema 名，如果这样的模式不存在，$user 将被忽略。所以默认情况下，用户连接数据库后，如果数据库下存在同名 Schema，则对象会添加到同名 Schema 下，否则对象被添加到 Public Schema 下。

❖ 要更改当前会话的默认 Schema，请使用 SET 命令。

执行如下命令将搜索路径设置为 myschema、public，首先搜索 myschema。

```
vastbase=# SET SEARCH_PATH TO myschema, public;
SET
```

5.2.7. 用户权限设置

- 给用户直接授予某对象的权限，请使用 11.16.88GRANT。

  将 Schema 中的表或者视图对象授权给其他用户或角色时，需要将表或视图所属 Schema 的 USAGE 权限同时授予该用户或角色。否则用户或角色将只能看到这些对象的名称，并不能实际进行对象访问。

  例如，下面示例将 Schema tpcds 的权限赋给用户 joe 后，将表 tpcds.web_returns 的 select 权限赋给用户 joe。

```
vastbase=# GRANT USAGE ON SCHEMA tpcds TO joe;
vastbase=# GRANT SELECT ON TABLE tpcds.web_returns to joe;
```

- 给用户指定角色，使用户继承角色所拥有的对象权限。

  a. 创建角色。

     新建一个角色 lily，同时给角色指定系统权限 CREATEDB：

```
vastbase=# CREATE ROLE lily WITH CREATEDB PASSWORD "Bigdata@123";
```

  b. 给角色赋予对象权限，请使用 11.16.88GRANT。

     例如，将模式 tpcds 的权限赋给角色 lily 后，将表 tpcds.web_returns 的 select 权限赋给角色 lily。

```
vastbase=# GRANT USAGE ON SCHEMA tpcds TO lily;
vastbase=# GRANT SELECT ON TABLE tpcds.web_returns to lily;
```

  c. 将角色的权限赋予用户。

```
vastbase=# GRANT lily to joe;
```

说明

当将角色的权限赋予用户时，角色的属性并不会传递到用户。

- 回收用户权限，请使用 11.16.99REVOKE。

5.2.8. 行级访问控制

行级访问控制特性将数据库访问控制精确到数据表行级别，使数据库达到行级访问控制的能力。不同用户执行相同的 SQL 查询操作，读取到的结果是不同的。

用户可以在数据表创建行访问控制(Row Level Security)策略，该策略是指针对特定数据库用户、特定 SQL 操作生效的表达式。当数据库用户对数据表访问时，若 SQL 满足数据表特定的 Row Level Security 策略，在查询优化阶段将满足条件的表达式，按照属性(PERMISSIVE | RESTRICTIVE)类型，通过 AND 或 OR 方式拼接，应用到执行计划上。

行级访问控制的目的是控制表中行级数据可见性，通过在数据表上预定义 Filter，在查询优化阶段将满足条件的表达式应用到执行计划上，影响最终的执行结果。当前受影响的 SQL 语句包括 SELECT，UPDATE，DELETE。

场景一：某表中汇总了不同用户的数据，但是不同用户只能查看自身相关的数据信息，不能查看其他用户的数据信息。

```
--创建用户 alice, bob, peter
vastbase=# CREATE ROLE alice PASSWORD 'Vastbase@123';
vastbase=# CREATE ROLE bob PASSWORD 'Vastbase@123';
vastbase=# CREATE ROLE peter PASSWORD 'Vastbase@123';

--创建表 all_data，包含不同用户数据信息
vastbase=# CREATE TABLE all_data(id int, role varchar(100), data varchar(100));

--向数据表插入数据
vastbase=# INSERT INTO all_data VALUES(1, 'alice', 'alice data');
vastbase=# INSERT INTO all_data VALUES(2, 'bob', 'bob data');
vastbase=# INSERT INTO all_data VALUES(3, 'peter', 'peter data');

--将表 all_data 的读取权限赋予 alice，bob 和 peter 用户
vastbase=# GRANT SELECT ON all_data TO alice, bob, peter;

--打开行访问控制策略开关
vastbase=# ALTER TABLE all_data ENABLE ROW LEVEL SECURITY;

--创建行访问控制策略，当前用户只能查看用户自身的数据
vastbase=# CREATE ROW LEVEL SECURITY POLICY all_data_rls ON all_data USING(role = CURRENT_USER);

--查看表详细信息
vastbase=# \d+ all_data
                        Table "public.all_data"
 Column |         Type          | Modifiers | Storage  | Stats target | Description
--------+-----------------------+-----------+----------+--------------+-------------
 id     | integer               |           | plain    |              |
 role   | character varying(100) |          | extended |              |
 data   | character varying(100) |          | extended |              |
Row Level Security Policies:
   POLICY "all_data_rls"
     USING (((role)::name = "current_user"()))
Has OIDs: no
Location Nodes: ALL DATANODES
Options: orientation=row, compression=no, enable_rowsecurity=true

--切换至用户 alice，执行 SQL"SELECT * FROM public.all_data"
vastbase=# SELECT * FROM public.all_data;
 id | role  |    data
----+-------+------------
  1 | alice | alice data
(1 row)

vastbase=# EXPLAIN(COSTS OFF) SELECT * FROM public.all_data;
                     QUERY PLAN
-----------------------------------------------------------------
```

```
 Streaming (type: GATHER)
   Node/s: All datanodes
   ->  Seq Scan on all_data
         Filter: ((role)::name = 'alice'::name)
 Notice: This query is influenced by row level security feature
(5 rows)

--切换至用户peter，执行SQL"SELECT * FROM public.all_data"
vastbase=# SELECT * FROM public.all_data;
 id | role  |    data
----+-------+------------
  3 | peter | peter data
(1 row)

vastbase=#  EXPLAIN(COSTS OFF) SELECT * FROM public.all_data;
                      QUERY PLAN
-----------------------------------------------------------------
 Streaming (type: GATHER)
   Node/s: All datanodes
   ->  Seq Scan on all_data
         Filter: ((role)::name = 'peter'::name)
 Notice: This query is influenced by row level security feature
(5 rows)
```

5.2.9. 设置安全策略

5.2.9.1. 设置帐户安全策略

背景信息

Vastbase 为帐户提供了自动锁定和解锁帐户、手动锁定和解锁异常帐户和删除不再使用的帐户等一系列的安全措施，保证数据安全。

自动锁定和解锁帐户

- 为了保证帐户安全，如果用户输入密码次数超过一定次数（failed_login_attempts），系统将自动锁定该帐户，默认值 5。次数设置越小越安全，但是在使用过程中会带来不便。
- 当帐户被锁定时间超过设定值（password_lock_time），则当前帐户自动解锁，默认值为 1 天。时间设置越长越安全，但是在使用过程中会带来不便。

说明

- 参数 password_lock_time 的整数部分表示天数，小数部分可以换算成时、分、秒。
- 当 failed_login_attempts 设置为 0 时，表示不限制密码错误次数。当 password_lock_time 设置为 0 时，表示即使超过密码错误次数限制导致帐户锁定，也会在短时间内自动解锁。因此，只有两个配置参数都为正数时，才可以进行常规的密码失败检查、帐户锁定和解锁操作。
- 这两个参数的默认值都符合安全标准，用户可以根据需要重新设置参数，提高安全等级。建议用户使用默认值。

配置 failed_login_attempts 参数。

1. 以操作系统用户 vastbase 登录数据库主节点。

2. 使用如下命令连接数据库。

```
vsql -d vastbase -p 5432
```

vastbase 为需要连接的数据库名称，5432 为数据库主节点的端口号。

连接成功后，系统显示类似如下信息：

```
vsql ((Vastbase 1.0 build 290d125f) compiled at 2020-05-08 02:59:43 commit 2143 last mr 131
Non-SSL connection (SSL connection is recommended when requiring high-security)
Type "help" for help.

vastbase=#
```

3. 查看已配置的参数。

```
vastbase=# SHOW failed_login_attempts;
 failed_login_attempts
-----------------------
 5
(1 row)
```

如果显示结果不为 5，执行“\q”命令退出数据库。

4. 执行如下命令设置成默认值 5。

```
vastbase=# alter system set failed_login_attempts=5;
```

配置 password_lock_time 参数。

5. 以操作系统用户 vastbase 登录数据库主节点。

6. 使用如下命令连接数据库。

```
vsql -d vastbase -p 5432
```

vastbase 为需要连接的数据库名称，5432 为数据库主节点的端口号。

连接成功后，系统显示类似如下信息：

```
vsql ((Vastbase 1.0 build 290d125f) compiled at 2020-05-08 02:59:43 commit 2143 last mr 131
Non-SSL connection (SSL connection is recommended when requiring high-security)
Type "help" for help.

vastbase=#
```

7. 查看已配置的参数。

```
vastbase=# SHOW password_lock_time;
 password_lock_time
-----------------------
 1
(1 row)
```

如果显示结果不为 1，执行“\q”命令退出数据库。

8. 执行如下命令设置成默认值 1。

```
vastbase=# alter system set password_lock_time=1;
```

手动锁定和解锁帐户

若管理员发现某帐户被盗、非法访问等异常情况，可手动锁定该帐户。

当管理员认为帐户恢复正常后，可手动解锁该帐户。

以手动锁定和解锁用户 joe 为例，用户的创建请参见 5.2.4 用户，命令格式如下：

- 手动锁定

```
vastbase=# ALTER USER joe ACCOUNT LOCK;
ALTER ROLE
```

- 手动解锁

```
vastbase=# ALTER USER joe ACCOUNT UNLOCK;
ALTER ROLE
```

删除不再使用的帐户

当确认帐户不再使用，管理员可以删除帐户。该操作不可恢复。

当删除的用户正处于活动状态时，此会话状态不会立马断开，用户在会话状态断开后才会被完全删除。

以删除帐户 joe 为例，命令格式如下：

```
vastbase=# DROP USER joe  CASCADE;
DROP ROLE
```

5.2.9.2.设置帐号有效期

注意事项

创建新用户时，需要限制用户的操作期限（有效开始时间和有效结束时间）。

不在有效操作期内的用户需要重新设定帐号的有效操作期。

操作步骤

步骤 1 以操作系统用户 vastbase 登录数据库主节点。

步骤 2 使用如下命令连接数据库。

```
vsql -d vastbase -p 5432
```

vastbase 为需要连接的数据库名称，5432 为数据库主节点的端口号。

连接成功后，系统显示类似如下信息：

```
vsql ((Vastbase 1.0 build 290d125f) compiled at 2020-05-08 02:59:43 commit 2143 last mr 131
Non-SSL connection (SSL connection is recommended when requiring high-security)
Type "help" for help.

vastbase=#
```

步骤 3 创建用户并制定用户的有效开始时间和有效结束时间。

```
vastbase=# CREATE USER joe WITH PASSWORD 'Bigdata@123' VALID BEGIN '2015-10-10 08:00:00' VALID
UNTIL '2016-10-10 08:00:00';
```

显示如下信息表示创建用户成功。

```
CREATE ROLE
```

步骤 4 用户已不在有效使用期内，需要重新设定帐号的有效期，这包括有效开始时间和有效结束时间。

```
vastbase=# ALTER USER joe WITH VALID BEGIN '2016-11-10 08:00:00' VALID UNTIL '2017-11-10
08:00:00';
```

显示如下信息表示重新设定成功。

```
ALTER ROLE
```

说明

若在“CREATE ROLE”或“ALTER ROLE”语法中不指定“VALID BEGIN”，表示不对用户的开始操作时间做限定；若不指定“VALID UNTIL”，表示不对用户的结束操作时间做限定；若两者均不指定，表示该用户一直有效。

5.2.9.3.设置密码安全策略

操作步骤

用户密码存储在系统表 pg_authid 中，为防止用户密码泄露，Vastbase 对用户密码进行加密存储，所采用的加密算法由配置参数 password_encryption_type 决定。

- 当参数 password_encryption_type 设置为 0 时，表示采用 md5 方式对密码加密。md5 为不安全的加密算法，不建议使用。
- 当参数 password_encryption_type 设置为 1 时，表示采用 sha256 和 md5 方式对密码加密，为默认配置。
- 当参数 password_encryption_type 设置为 2 时，表示采用 sha256 方式对密码加密。

步骤 1 以操作系统用户 vastbase 登录数据库主节点。

步骤 2 使用如下命令连接数据库。

```
vsql -d vastbase -p 5432
```

vastbase 为需要连接的数据库名称，5432 为数据库主节点的端口号。

连接成功后，系统显示类似如下信息：

```
vsql ((Vastbase 1.0 build 290d125f) compiled at 2020-05-08 02:59:43 commit 2143 last mr 131
Non-SSL connection (SSL connection is recommended when requiring high-security)
Type "help" for help.

vastbase=#
```

步骤 3 查看已配置的加密算法。

```
vastbase=# SHOW password_encryption_type;
 password_encryption_type
--------------------------
 1
(1 row)
```

如果显示结果为 0 或 2，执行“\q”命令退出数据库。

步骤 4 执行如下命令将其设置为安全的加密算法。

```
vastbase=# alter system set password_encryption_type=1;
```

须知

为防止用户密码泄露，在执行 CREATE USER/ROLE 命令创建数据库用户时，不能指定 UNENCRYPTED 属性，即新创建的用户的密码只能是加密存储的。

步骤 5 配置密码安全参数。

❖ 密码复杂度

初始化数据库、创建用户、修改用户时需要指定密码。密码必须要符合复杂度（password_policy）的要求，否则会提示用户重新输入密码。

- 参数 password_policy 设置为 1 时表示采用密码复杂度校验，默认值。
- 参数 password_policy 设置为 0 时表示不采用任何密码复杂度校验，设置为 0 会存在安全风险，不建议设置为 0，即使需要设置也要将所有 Vastbase 节点中的 password_policy 都设置为 0 才能生效。

配置 password_policy 参数。

a. 使用如下命令连接数据库。

```
vsql -d vastbase -p 5432
```

vastbase 为需要连接的数据库名称，5432 为数据库主节点的端口号。

连接成功后，系统显示类似如下信息：

```
vsql ((Vastbase 2.2.0 build ) compiled at 2021-01-26 19:22:33 commit 0 last
mr  )
Non-SSL connection (SSL connection is recommended when requiring high-security)
Type "help" for help.

vastbase=#
```

b. 查看已配置的参数。

```
vastbase=# SHOW password_policy;
 password_policy
---------------------
 1
(1 row)
```

如果显示结果不为 1，执行“\q”命令退出数据库。

c. 执行如下命令设置成默认值 1。

```
vastbase=# alter system set password_policy=1
```

帐户密码的复杂度要求如下：

- 包含大写字母（A-Z）的最少个数（password_min_uppercase）
- 包含小写字母（a-z）的最少个数（password_min_lowercase）
- 包含数字（0-9）的最少个数（password_min_digital）
- 包含特殊字符的最少个数（password_min_special）（特殊字符的列表请参见表 5-9）
- 密码的最小长度（password_min_length）

- 密码的最大长度（password_max_length）
- 至少包含上述四类字符中的三类。
- 不能和用户名、用户名倒写相同，本要求为非大小写敏感。
- 不能和当前密码、当前密码的倒写相同。

❖ 密码重用

用户修改密码时，只有超过不可重用天数（password_reuse_time）或不可重用次数（password_reuse_max）的密码才可以使用。参数配置说明如表 5-10 所示。

说明

不可重用天数默认值为 90 天，不可重用次数默认值是 3。这两个参数值越大越安全，但是在使用过程中会带来不便，其默认值符合安全标准，用户可以根据需要重新设置参数，提高安全等级。

配置 password_reuse_time 参数。

a. 使用如下命令连接数据库。

```
vsql -d vastbase -p 5432
```

vastbase 为需要连接的数据库名称，5432 为数据库主节点的端口号。

连接成功后，系统显示类似如下信息：

```
vsql ((Vastbase 2.2.0 build ) compiled at 2021-01-26 19:22:33 commit 0 last
mr  )
Non-SSL connection (SSL connection is recommended when requiring high-security)
Type "help" for help.

vastbase=#
```

b. 查看已配置的参数。

```
vastbase=# SHOW password_reuse_time;
 password_reuse_time
---------------------
 90
(1 row)
```

如果显示结果不为 90，执行“\q”命令退出数据库。

c. 执行如下命令设置成默认值 90。

```
vastbase=# alter system set password_reuse_time=90
```

说明

不建议设置为 0，即使需要设置也要将所有 Vastbase 节点中的 password_reuse_time 都设置为 0 才能生效。

配置 password_reuse_max 参数。

a. 使用如下命令连接数据库。

```
vsql -d vastbase -p 5432
```

vastbase 为需要连接的数据库名称，5432 为数据库主节点的端口号。

连接成功后，系统显示类似如下信息：

```
vsql ((Vastbase 2.2.0 build ) compiled at 2021-01-26 19:22:33 commit 0 last
mr  )
Non-SSL connection (SSL connection is recommended when requiring high-security)
Type "help" for help.

vastbase=#
```

b. 查看已配置的参数。

```
vastbase=# SHOW password_reuse_max;
 password_reuse_max
--------------------
 3
(1 row)
```

如果显示结果不为 3，执行“\q”命令退出数据库。

c. 执行如下命令设置成默认值 3。

```
vastbase=# alter system set password_reuse_max = 3
```

❖ 密码有效期限

数据库用户的密码都有密码有效期（password_effect_time），当达到密码到期提醒天数（password_notify_time）时，系统会在用户登录数据库时提示用户修改密码。

说明

考虑到数据库使用特殊性及业务连续性，密码过期后用户还可以登录数据库，但是每次登录都会提示修改密码，直至修改为止。

配置 password_effect_time 参数。

a. 使用如下命令连接数据库。

```
vsql -d vastbase -p 5432
```

vastbase 为需要连接的数据库名称，5432 为数据库主节点的端口号。

连接成功后，系统显示类似如下信息：

```
vsql ((Vastbase 2.2.0 build ) compiled at 2021-01-26 19:22:33 commit 0 last
mr  )
Non-SSL connection (SSL connection is recommended when requiring high-security)
Type "help" for help.

vastbase=#
```

b. 查看已配置的参数。

```
vastbase=# SHOW password_effect_time;
 password_effect_time
----------------------
 90
(1 row)
```

如果显示结果不为 90，执行“\q”命令退出数据库。

c. 执行如下命令设置成默认值 90（不建议设置为 0）。

```
vastbase=# alter system set password_effect_time = 90
```

配置 password_notify_time 参数。

a. 使用如下命令连接数据库。

```
vsql -d vastbase -p 5432
```

vastbase 为需要连接的数据库名称，5432 为数据库主节点的端口号。

连接成功后，系统显示类似如下信息：

```
vsql ((Vastbase 2.2.0 build ) compiled at 2021-01-26 19:22:33 commit 0 last
mr  )
Non-SSL connection (SSL connection is recommended when requiring high-security)
Type "help" for help.

vastbase=#
```

b. 查看已配置的参数。

```
vastbase=# SHOW password_notify_time;
 password_notify_time
----------------------
 7
(1 row)
```

c. 如果显示结果不为 7，执行如下命令设置成默认值 7（不建议设置为 0）。

```
vastbase=# alter system set password_notify_time = 7;
```

❖ 密码修改

– 在安装数据库时，会新建一个和初始化用户重名的操作系统用户，为了保证帐户安全，请定期修改操作系统用户的密码。

以修改用户 user1 密码为例，命令格式如下：

```
passwd user1
```

根据提示信息完成修改密码操作。

– 建议系统管理员和普通用户都要定期修改自己的帐户密码，避免帐户密码被非法窃取。

以修改用户 user1 密码为例，以系统管理员用户连接数据库并执行如下命令：

```
vastbase=# ALTER USER user1 IDENTIFIED BY "1234@abc" REPLACE "5678@def";
ALTER ROLE
```

说明

1234@abc、5678@def 分别代表用户 user1 的新密码和原始密码，这些密码要符合规则，否则会执行失败。

– 管理员可以修改自己的或者其他帐户的密码。通过修改其他帐户的密码，解决用户密码遗失所造成无法登录的问题。

以修改用户 joe 帐户密码为例，命令格式如下：

```
vastbase=# ALTER USER joe IDENTIFIED BY "abc@1234";
ALTER ROLE
```

说明

- 系统管理员之间不允许互相修改对方密码。
- 系统管理员可以修改普通用户密码且不需要用户原密码。
- 系统管理员修改自己密码但需要管理员原密码。

❖ 密码验证

设置当前会话的用户和角色时，需要验证密码。如果输入密码与用户的存储密码不一致，则会报错。

以设置用户 joe 为例，命令格式如下：

```
vastbase=# SET ROLE joe PASSWORD "abc@1234";
ERROR:  Invalid username/password,set role denied.
```

表 5-9. 特殊字符

编号	字符	编号	字符	编号	字符	编号	字符
1	~	9	*	17	|	25	<
2	!	10	(	18	[	26	.
3	@	11	)	19	{	27	>
4	#	12	-	20	}	28	/
5	$	13	_	21	]	29	?
6	%	14	=	22	;	-	-
7	^	15	+	23	:	-	-
8	&	16	\	24	,	-	-

表 5-10. 不可重用天数和不可重用次数参数说明

参数	取值范围	配置说明
不可重用天数（password_reuse_time）	正数或 0，其中整数部分表示天数，小数部分可以换算成时，分，秒。 默认值为 90。	• 如果参数变小，则后续修改密码按新的参数进行检查。 • 如果参数变大（比如由 a 变大为 b），因为 b 天之前的历史密码可能已经删除，所以 b 天之前的密码仍有可能被重用。则后续修改密码按新的参数进行检查。 说明 时间以绝对时间为准，历史密码记录的都是当时的时间，

参数	取值范围	配置说明
		不识别时间的修改。
不可重用次数（password_reuse_max）	正整数或 0。 默认值为 3，表示不检查重用次数。	• 如果参数变小，则后续修改密码按新的参数进行检查。 • 如果参数变大（比如由 a 变大为 b），因为 b 次之前的历史密码可能已经删除，所以 b 次之前的密码仍有可能被重用。则后续修改密码按新的参数进行检查。

5.3. 设置数据库审计

5.3.1. 审计概述

背景信息

数据库安全对数据库系统来说至关重要。Vastbase 将用户对数据库的所有操作写入审计日志。数据库安全管理员可以利用这些日志信息，重现导致数据库现状的一系列事件，找出非法操作的用户、时间和内容等。

关于审计功能，用户需要了解以下几点内容：

❖ 审计总开关 audit_enabled 支持动态加载。在数据库运行期间修改该配置项的值会立即生效，无需重启数据库。默认值为 on，表示开启审计功能。

❖ 除了审计总开关，各个审计项也有对应的开关。只有开关开启，对应的审计功能才能生效。

❖ 各审计项的开关支持动态加载。在数据库运行期间修改审计开关的值，不需要重启数据库便可生效。

目前，Vastbase 支持以下审计项如表 5-11 所示。

表 5-11. 配置审计项

配置项	描述
用户登录、注销审计	参数：audit_login_logout 默认值为 7，表示开启用户登录、退出的审计功能。设置为 0 表示关闭用户登录、退出的审计功能。不推荐设置除 0 和 7 之外的值。
数据库启动、停止、恢复和切换审计	参数：audit_database_process

配置项	描述
	默认值为 1，表示开启数据库启动、停止、恢复和切换的审计功能。
用户锁定和解锁审计	参数：audit_user_locked 默认值为 1，表示开启审计用户锁定和解锁功能。
用户访问越权审计	参数：audit_user_violation 默认值为 0，表示关闭用户越权操作审计功能。
授权和回收权限审计	参数：audit_grant_revoke 默认值为 1，表示开启审计用户权限授予和回收功能。
数据库对象的 CREATE，ALTER，DROP 操作审计	参数：audit_system_object 默认值为 12295，表示只对 DATABASE、SCHEMA、USER、DATA SOURCE 这四类数据库对象的 CREATE、ALTER、DROP 操作进行审计。
具体表的 INSERT、UPDATE 和 DELETE 操作审计	参数：audit_dml_state 默认值为 0，表示关闭具体表的 DML 操作（SELECT 除外）审计功能。
SELECT 操作审计	参数：audit_dml_state_select 默认值为 0，表示关闭 SELECT 操作审计功能。
COPY 审计	参数：audit_copy_exec 默认值为 0，表示关闭 copy 操作审计功能。
存储过程和自定义函数的执行审计	参数：audit_function_exec 默认值为 0，表示不记录存储过程和自定义函数的执行审计日志。
SET 审计	参数：audit_set_parameter 默认值为 1，表示记录 set 操作审计日志

安全相关参数及默认值请参见表 5-12。

表 5-12. 安全相关参数及默认值

参数名	默认值	说明
ssl	off	指定是否启用 SSL 连接。
require_ssl	off	指定服务器端是否强制要求 SSL 连接。

参数名	默认值	说明
ssl_ciphers	ALL	指定 SSL 支持的加密算法列表。
ssl_cert_file	server.crt	指定包含 SSL 服务器证书的文件的名称。
ssl_key_file	server.key	指定包含 SSL 私钥的文件名称。
ssl_ca_file	cacert.pem	指定包含 CA 信息的文件的名称。
ssl_crl_file	NULL	指定包含 CRL 信息的文件的名称。
password_policy	1	指定是否进行密码复杂度检查。
password_reuse_time	60	指定是否对新密码进行可重用天数检查。
password_reuse_max	0	指定是否对新密码进行可重用次数检查。
password_lock_time	1	指定帐户被锁定后自动解锁的时间。
failed_login_attempts	10	如果输入密码错误的次数达到此参数值时，当前帐户被锁定。
password_encryption_type	2	指定采用何种加密方式对用户密码进行加密存储。
password_min_uppercase	0	密码中至少需要包含大写字母的个数。
password_min_lowercase	0	密码中至少需要包含小写字母的个数。
password_min_digital	0	密码中至少需要包含数字的个数。
password_min_special	0	密码中至少需要包含特殊字符的个数。
password_min_length	8	密码的最小长度。 说明 在设置此参数时，请将其设置成不大于password_max_length，否则进行涉及密码的操作会一直出现密码长度错误的提示
password_max_length	32	密码的最大长度。 说明 在设置此参数时，请将其设置成不小于password_min_length，否则进行涉及密码的操作会一直出现密码长度错误的提示。

参数名	默认值	说明
password_effect_time	90	密码的有效期限。
password_notify_time	7	密码到期提醒的天数。
audit_enabled	on	控制审计进程的开启和关闭。
audit_directory	pg_audit	审计文件的存储目录。
audit_data_format	binary	审计日志文件的格式，当前仅支持二进制格式（binary）。
audit_rotation_interval	1d	指定创建一个新审计日志文件的时间间隔。当现在的时间减去上次创建一个审计日志的时间超过了此参数值时，服务器将生成一个新的审计日志文件。
audit_rotation_size	10MB	指定审计日志文件的最大容量。当审计日志消息的总量超过此参数值时，服务器将生成一个新的审计日志文件。
audit_resource_policy	on	控制审计日志的保存策略，以空间还是时间限制为优先策略，on 表示以空间为优先策略。
audit_file_remain_time	90	表示需记录审计日志的最短时间要求，该参数在 audit_resource_policy 为 off 时生效。
audit_space_limit	1GB	审计文件占用磁盘空间的最大值。
audit_file_remain_threshold	1048576	审计目录下审计文件的最大数量。
audit_login_logout	7	指定是否审计数据库用户的登录（包括登录成功和登录失败）、注销。
audit_database_process	1	指定是否审计数据库启动、停止、切换和恢复的操作。
audit_user_locked	1	指定是否审计数据库用户的锁定和解锁。
audit_user_violation	0	指定是否审计数据库用户的越权访问操作。
audit_grant_revoke	1	指定是否审计数据库用户权限授予和回收的操作。

参数名	默认值	说明
audit_system_object	12295	指定是否审计数据库对象的 CREATE、DROP、ALTER 操作。
audit_dml_state	0	指定是否审计具体表的 INSERT、UPDATE、DELETE 操作。
audit_dml_state_select	0	指定是否审计 SELECT 操作。
audit_copy_exec	0	指定是否审计 COPY 操作。
audit_function_exec	0	指定在执行存储过程、匿名块或自定义函数（不包括系统自带函数）时是否记录审计信息。
audit_set_parameter	1	指定是否审计 SET 操作。
enableSeparationOfDuty	off	指定是否开启三权分立。
session_timeout	10min	建立连接会话后，如果超过此参数的设置时间，则会自动断开连接。
auth_iteration_count	10000	认证加密信息生成过程中使用的迭代次数。

操作步骤

步骤 1 以操作系统用户 vastbase 登录数据库主节点。

步骤 2 使用如下命令连接数据库。

```
vsql -d vastbase -p 5432
```

vastbase 为需要连接的数据库名称，5432 为数据库主节点的端口号。

连接成功后，系统显示类似如下信息：

```
vsql ((Vastbase 1.0 build 290d125f) compiled at 2020-05-08 02:59:43 commit 2143 last mr 131
Non-SSL connection (SSL connection is recommended when requiring high-security)
Type "help" for help.

vastbase=#
```

步骤 3 检查审计总开关状态。

1. 用 show 命令显示审计总开关 audit_enabled 的值。

```
vastbase=# SHOW audit_enabled;
```

如果显示为 off，执行“\q”命令退出数据库。

2. 修改 postgres.conf 的 audit_enabled=”on”,重启数据库，使得参数生效。

步骤 4 配置具体的审计项。

说明

- 只有开启审计功能，用户的操作才会被记录到审计文件中。
- 各审计项的默认参数都符合安全标准，用户可以根据需要开启其他审计功能，但会对性能有一定影响。

以开启对数据库所有对象的增删改操作的审计开关为例，其他配置项的修改方法与此相同，修改 postgres.conf 的 audit_system_object=12295,重启数据库，使得参数生效。

其中 audit_system_object 代表审计项开关，12295 为该审计开关的值。

5.3.2. 查看审计结果

前提条件

- 审计功能总开关已开启。
- 需要审计的审计项开关已开启。
- 数据库正常运行，并且对数据库执行了一系列增、删、改、查操作，保证在查询时段内有审计结果产生。
- 数据库各个节点审计日志单独记录。

背景信息

- 只有拥有 AUDITADMIN 属性的用户才可以查看审计记录。有关数据库用户及创建用户的办法请参见 5.2.4 用户。
- 审计查询命令是数据库提供的 sql 函数 pg_query_audit，其原型为：

```
pg_query_audit(timestamptz startime,timestamptz endtime,audit_log)
```

参数 startime 和 endtime 分别表示审计记录的开始时间和结束时间，audit_log 表示所查看的审计日志信息所在的物理文件路径，当不指定 audit_log 时，默认查看连接当前实例的审计日志信息。

通过 sql 函数 pgxc_query_audit 可以查询数据库主节点的审计日志，其原型为：

```
pgxc_query_audit(timestamptz startime,timestamptz endtime)
```

说明

startime 和 endtime 的差值代表要查询的时间段，其有效值为从 startime 日期中的 00:00:00 开始到 endtime 日期中的 23:59:59 之间的任何值。请正确指定这两个参数，否则将查不到需要的审计信息。

操作步骤

步骤 1 以操作系统用户 vastbase 登录数据库主节点。

步骤 2 使用如下命令连接数据库。

```
vsql -d vastbase -p 5432
```

vastbase 为需要连接的数据库名称，5432 为数据库主节点的端口号。

连接成功后，系统显示类似如下信息：

```
vsql ((Vastbase 2.0 build 290d125f) compiled at 2020-05-08 02:59:43 commit 2143 last mr 131
Non-SSL connection (SSL connection is recommended when requiring high-security)
Type "help" for help.

vastbase=#
```

步骤 3 查询审计记录。

```
vastbase=# SELECT * FROM pg_query_audit('2015-07-15 08:00:00','2015-07-15 09:47:33');
```

查询结果如下：

```
        time         |     type      | result | username |    database    | client_conninfo |
object_name   |                      detail_info                       | node_name |
thread_id          | local_port | remote_port
-----------------------+---------------+--------+----------+----------------+-----------------
--------+----------------+------------------------------------------------------------------------
+-----------+----------------------------------+------------+-------------
 2015-07-15 08:03:55+08 | login_success | ok     | vastbase | vastbase       | gs_clean@::1
| vastbase       | login db(vastbase) success,the current user is:vastbase       | cn_5003   |
1398089029977776@490233835920483 | 9000       | 55805
```

该条记录表明，用户 vastbase 在 2015-07-15 08:03:55+08 登录数据库 vastbase。其中 client_conninfo 字段在 log_hostname 启动且 IP 连接时，字符@后显示反向 DNS 查找得到的主机名。

步骤 4 查询数据库主节点审计记录。

```
vastbase=# SELECT * FROM pgxc_query_audit('2019-01-10 17:00:00','2019-01-10 19:00:00') where
type = 'login_success' and username = 'user1';
```

查询结果如下：

```
        time         |     type      | result | username | database | client_conninfo |
object_name |                 detail_info                  | node_name   |
thread_id          |
local_port | remote_port
-----------------------+---------------+--------+----------+----------+-----------------
-+-------------+----------------------------------------------------------+--------------+--
--------------------------------+-
-----------+-------------
 2019-01-10 18:06:08+08 | login_success | ok     | user1    | vastbase | vsql@[local]    |
vastbase    | login db(vastbase) success,the current user is:user1 | coordinator1 |
139965149210368@600429968516954 |
17560      | null
 2019-01-10 18:06:22+08 | login_success | ok     | user1    | vastbase | vsql@[local]    |
vastbase    | login db(vastbase) success,the current user is:user1 | coordinator1 |
139965149210368@600429982697548 |
17560      | null
 2019-01-10 18:06:54+08 | login_success | ok     | user1    | vastbase | vsql@[local]    |
vastbase    | login db(vastbase) success,the current user is:user1 | coordinator2 |
140677694355200@600430014804280 |
17562      | null
(3 rows)
```

查询结果显示，用户 user1 在数据库主节点 1 和数据库主节点 2 的成功登录记录。

5.3.3. 维护审计日志

前提条件

用户必须拥有审计权限。

背景信息

- 与审计日志相关的配置参数及其含义请参见表格表 5-13。

表 5-13. 表格

配置项	含义	默认值
audit_directory	审计文件的存储目录。	pg_audit
audit_resource_policy	审计日志的保存策略。	on（表示使用空间配置策略）
audit_space_limit	审计文件占用的磁盘空间总量。	1GB
audit_file_remain_time	审计日志文件的最小保存时间。	90
audit_file_remain_threshold	审计目录下审计文件的最大数量。	1048576

说明

- 审计日志删除命令为数据库提供的 sql 函数 pg_delete_audit，其原型为：

```
pg_delete_audit(timestamp startime,timestamp endtime)
```

其中参数 startime 和 endtime 分别表示审计记录的开始时间和结束时间，填写的参数类型为 timestamp，如只输入日期，则默认为指定日期的零时零分零秒，如：

```
SELECT pg_delete_audit('2012-09-20 ','2012-09-21');
```

等价于

```
SELECT pg_delete_audit('2012-09-20 00:00:00','2012-09-21 00:00:00');
```

- 目前常用的记录审计内容的方式有两种：记录到数据库的表中、记录到 OS 文件中。这两种方式的优缺点比较如表格 0 所示。
- 表格

方式	优点	缺点

方式	优点	缺点
记录到表中	不需要用户维护审计日志。	由于表是数据库的对象，如果一个数据库用户具有一定的权限，就能够访问到审计表。如果该用户非法操作审计表，审计记录的准确性难以得到保证。
记录到 OS 文件中	比较安全，即使一个帐户可以访问数据库，但不一定有访问 OS 这个文件的权限。	需要用户维护审计日志。

从数据库安全角度出发，Vastbase 采用记录到 OS 文件的方式来保存审计结果，保证了审计结果的可靠性。

操作步骤

步骤 1 以操作系统用户 vastbase 登录数据库主节点。

步骤 2 使用如下命令连接数据库。

```
vsql -d vastbase -p 5432
```

vastbase 为需要连接的数据库名称，5432 为数据库主节点的端口号。

连接成功后，系统显示类似如下信息：

```
vsql ((Vastbase 2.0 build 290d125f) compiled at 2020-05-08 02:59:43 commit 2143 last mr 131
Non-SSL connection (SSL connection is recommended when requiring high-security)
Type "help" for help.

vastbase=#
```

步骤 3 选择日志维护方式进行维护。

- 设置自动删除审计日志

审计文件占用的磁盘空间或者审计文件的个数超过指定的最大值时，系统将删除最早的审计文件，并记录审计文件删除信息到审计日志中。

说明

审计文件占用的磁盘空间大小默认值为 1024MB，用户可以根据磁盘空间大小重新设置参数。

配置审计文件占用磁盘空间的大小（audit_space_limit）。

a. 查看已配置的参数。

```
vastbase=# SHOW audit_space_limit;
 audit_space_limit
-------------------
 1GB
(1 row)
```

如果显示结果不为 1GB（1024MB），执行“\q”命令退出数据库。

b. 修改 postgres.conf 的 audit_space_limit=1024MB,重启数据库，使得参数生效。

配置审计文件个数的最大值（audit_file_remain_threshold）。

a. 查看已配置的参数。

```
vastbase=# SHOW audit_file_remain_threshold;
 audit_file_remain_threshold
-----------------------------
 1048576
(1 row)
```

如果显示结果不为 1048576，执行“\q”命令退出数据库。

b. 建议修改 postgres.conf 的设置 audit_file_remain_threshold 为默认值 1048576。

❖ 手动备份审计文件

当审计文件占用的磁盘空间或者审计文件的个数超过配置文件指定的值时，系统将会自动删除较早的审计文件，因此建议用户周期性地对比较重要的审计日志进行保存。

a. 使用 show 命令获得审计文件所在目录（audit_directory）。

```
vastbase=# SHOW audit_directory;
```

b. 将审计目录整个拷贝出来进行保存。

❖ 手动删除审计日志

当不再需要某时段的审计记录时，可以使用审计接口命令 pg_delete_audit 进行手动删除。

以删除 2012/9/20 到 2012/9/21 之间的审计记录为例：

```
vastbase=# SELECT pg_delete_audit('2012-09-20 ','2012-09-21');
```

5.3.4. 设置文件权限安全策略

背景信息

数据库在安装过程中，会自动对其文件权限（包括运行过程中生成的文件，如日志文件等）进行设置。其权限规则如下：

❖ 数据库程序目录的权限为 0750。

❖ 数据库数据文件目录的权限为 0700。

❖ 数据库的数据文件、审计日志和其他数据库程序生成的数据文件的权限为 0600，运行日志的权限默认不高于 0640。

❖ 普通操作系统用户不允许修改和删除数据库文件和日志文件。

数据库程序目录及文件权限

数据库安装后，部分程序目录及文件权限如表 5-13 所示。

表 5-14. 文件及目录权限

文件/目录	父目录	权限
bin	-	0700
lib	-	0700
share	-	0700
data（数据库节点/数据库主节点）	-	0700
base	实例数据目录	0700
global	实例数据目录	0700
pg_audit	实例数据目录（可配置）	0700
pg_log	实例数据目录（可配置）	0700
pg_xlog	实例数据目录	0700
postgresql.conf	实例数据目录	0600
pg_hba.conf	实例数据目录	0600
postmaster.opts	实例数据目录	0600
pg_ident.conf	实例数据目录	0600
vb_initdb	bin	0700
vb_dump	bin	0700
vb_ctl	bin	0700
vb_guc	bin	0700
vsql	bin	0700
archive_status	pg_xlog	0700
libpq.so.5.5	lib	0600

建议

数据库在安装过程中，会自动对其文件权限（包括运行过程中生成的文件，如日志文件等）进行设置，适合大多数情况下的权限要求。如果用户产品对相关权限有特殊要求，建议用户安装后定期检查相关权限设置，确保完全符合产品要求。

5.4. 强制访问控制

功能描述

强制访问控制，用于将系统中的信息分密级和类进行管理，以保证每个用户只能访问到那些被标明可以由他访问的信息一种访问约束机制。通俗的来说，在强制访问控制下，用户（或其他主体）与文件（或其他客体）都被标记了固定的安全属性（如安全级、访问权限等），在每次访问发生时，系统检测安全属性以便确定一个用户是否有权限访问该文件。

5.4.1. 敏感标记

为实现强制访问控制功能，数据库管理系统应维护主体和客体相关的敏感标记，为主体和客体划分安全级。这些敏感标记是等级分类和非等级类别的组合，是实施强制访问控制的依据。

5.4.1.1.等级分类

Vastbase 内置了 16 个等级分类，命名为 Li，其中（1≤i≤16），等级标记满足偏序关系（若 i≤j，则 Li ≤ Lj）。

5.4.1.2.非等级类别

Vastbase 内置了 32 个非等级类别，命名为 Gi，其中（1≤i≤32）。非等级标记无法进行大小比较，但可以进行集合运算。例如非等级标记“G1,G3”包含“G1”。

5.4.1.3.敏感标记

Vastbase 中敏感标记由等级分类和非等级类别两部分组成，两者中间用冒号（“:”）分隔，形式如下：

```
等级分类组:非等级类别组
```

其中等级分类组由多个等级分类组成，非等级类别组由多个非等级类别组成。等级分类和非等级类别之间用逗号（“,”）分隔，用点号（“.”）表示区间。

例如敏感标记“L1,L3.L7:G2,G4.G6,G27”表示的等级分类和非等级类别分别为：

- 等级分类：L_1,L_3~L_7
- 非等级类别：G_2,G_4~G_6,G_{27}

5.4.2. 访问策略

Vastbase 中只有自主访问控制（DAC）校验通过的情况下才能进行强制访问控制（MAC）校验，强制访问控制的读写规则如下。

5.4.2.1.表强制访问控制策略

1.插入（INSERT）策略

主体敏感等级标记小于等于表敏感等级标记，且主体敏感非等级标记是表敏感非等级标记子集则允许插入。

2.修改（UPDATE）策略

主体敏感标记与元组的敏感标记一致则允许修改元组。

3.删除（DELETE）策略

同修改（UPDATE）策略。

4.查询（SELECT）策略

主体敏感等级标记大于等于元组敏感等级标记，且主体敏感非等级标记是元组敏感非等级标记超集则允许查询。

示例

```
1. 登陆进入 vastbase 数据库，创建用户
create user user1 with password 'vastbase@123';
create user user2 with password 'vastbase@123';
create user user3 with password 'vastbase@123';
create user user4 with password 'vastbase@123';
create user user5 with password 'vastbase@123';
create user user6 with password 'vastbase@123';
create user user7 with password 'vastbase@123';
```

```
create user user8 with password 'vastbase@123';
create user user9 with password 'vastbase@123';
create table t1(c1 int,c2 varchar(50));
```
2. 给用户授权
```
GRANT ALL ON TABLE t1 TO user1;
GRANT ALL ON TABLE t1 TO user2;
GRANT ALL ON TABLE t1 TO user3;
GRANT ALL ON TABLE t1 TO user4;
GRANT ALL ON TABLE t1 TO user5;
GRANT ALL ON TABLE t1 TO user6;
GRANT ALL ON TABLE t1 TO user7;
GRANT ALL ON TABLE t1 TO user8;
GRANT ALL ON TABLE t1 TO user9;
```
3.创建标签并为主体和客体授予标签
```
CREATE SECURITY LABEL label1 'L12:G7,G2,G32,G15.G20';
CREATE SECURITY LABEL label2 'L5:G7,G2,G15.G20';
CREATE SECURITY LABEL label3 'L10:G2,G7';
CREATE SECURITY LABEL label4 'L15:G1,G2,G4,G7.G10,G32,G15.G20';
CREATE SECURITY LABEL label5 'L12:G1,G2,G7.G10,G32,G15.G20';
CREATE SECURITY LABEL label6 'L1:G7,G2,G32,G15.G20';
CREATE SECURITY LABEL label7 'L13:G2,G32,G15.G20';
```
4.对数据库对象设置敏感标记
```
SECURITY LABEL ON TABLE t1 IS 'label1';
SECURITY LABEL ON role user1 IS 'label2';
SECURITY LABEL ON role user2 IS 'label3';
SECURITY LABEL ON role user3 IS 'label4';
SECURITY LABEL ON role user4 IS 'label2';
SECURITY LABEL ON role user5 IS 'label3';
SECURITY LABEL ON role user6 IS 'label5';
SECURITY LABEL ON role user7 IS 'label6';
SECURITY LABEL ON role user8 IS 'label1';
```
5. 根据情况分别用 user1,user2,user3,user4,user5,user6,user7,user8 用户登数据库，对表 t1 中的记录进行增删改查操作

1)主体等级小于等于客体的等级，主体非等级是客体的非等级的子集时，主体(用户)对客体(表)insert 数据
```
\c - user1
insert into t1 values(1,'aaa');
insert into t1 values(2,'bbb');
insert into t1 values(3,'ccc');
insert into t1 values(4,'ddd');
```
主体等级大于等于客体的等级，主体非等级是客体的非等级的超集时，主体(用户)对客体(表)进行 select
```
\c - user6
select * from t1;
```
4)主体等级等于客体的等级，主体非等级等于客体的非等级时，主体(用户)对客体(表)进行 update 操作
```
\c - user4
update t1 set c2='mmm';
select * from t1;
```
5)主体等级等于客体的等级，主体非等级等于客体的非等级时，主体(用户)对客体(表)进行 delete 操作
```
\c - user1
delete from t1 where c1=4;
```

5.5. 安全审计

5.5.1. 概述

安全审计功能是对数据库用户行为进行记录、分析和阻断的功能。提供 SQL 命令维护审计策略，审计策略维护好之后，对应的审计事件发生时依据审计策略中的响应动作进行处理。

5.5.1.1.可审计对象与事件类型

- 对象类型

目前 Vastbase 支持的审计对象类型包括：模式（schema）、表（table）、视图（view）和函数（function），只有 DDL 和 DML 语句操作的对象是上述对象时才可被审计。

- 事件类型

Vastbase 中可审计的事件主要分为两大类：权限（PRIVILEGE）和访问（ACCESS）。

- PRIVILEGE 类型：

Vastbase 支持的 PRIVILEGE 权限审计类型如下表所示。

序号	PRIVILEGE 类型	说明
1.	ALTER	修改对象属性 DDL 语句
2.	ANALYZE	表分析语句
3.	COMMENT	为对象添加注释
4.	CREATE	创建对象语句
5.	DROP	删除对象语句
6.	GRANT	授权语句
7.	REVOKE	权限回收语句
8.	SET	修改会话属性（参数）
9.	SHOW	展示会话属性（参数）
10.	LOGIN_ANY	任何登录事件
11.	LOGIN_FAILURE	登录失败事件
12.	LOGIN_SUCCESS	登录成功事件
13.	LOGOUT	登出（退出）事件
14.	CHECKPOINT	检查点事件
15.	COMMIT	提交
16.	ROLLBACK	回滚

- ACCESS 类型：

Vastbase 支持的 ACCESS 权限审计类型如下表所示。

序号	ACCESS 类型	说明

1.	COPY	COPY 命令
2.	DELETE	DELETE 语句
3.	EXECUTE	EXECUTE 语句（函数的执行）
4.	INSERT	INSERT 语句
5.	REINDEX	REINDEX 语句
6.	SELECT	SELECT 语句
7.	TRUNCATE	TRUNCATE 语句
8.	UPDATE	UPDATE 语句

5.5.1.2.审计响应动作

Vastbase 审计支持记录审计日志（普通日志和告警日志）和阻断会话，当审计事件发生时，根据审计策略判断是写日志还是阻断。

5.5.2. 开启安全审计

Vastbase 中通过配置参数 enable_security_policy 参数控制是否启用审计功能，在开启了审计功能情况下，若用户操作满足审计策略则会采取策略设定的响应动作。具体步骤如下：

```
1.使用 root 操作系统用户编辑文件/etc/rsyslog.conf，在文件末尾添加以下内容：
template(name="vbaudit" type="list") {
property(name="timereported" dateformat="year")
constant(value="-")
property(name="timereported" dateformat="month")
constant(value="-")
property(name="timereported" dateformat="day")
constant(value=" ")
property(name="timereported" dateformat="hour")
constant(value=":")
property(name="timereported" dateformat="minute")
constant(value=":")
property(name="timereported" dateformat="second")
constant(value="|")
property(name="msg")
constant(value="\n")
}
local0.*       /var/log/vbaudit.log;vbaudit

2.使用 root 操作系统用户重启 rsyslog 服务
systemctl stop rsyslog.service
systemctl start rsyslog.service
查看服务
systemctl status rsyslog.service

3.使用 root 操作系统用户创建日志文件并授权
```

```
touch /var/log/vbaudit.log
chmod 777 /var/log/vbaudit.log

4.修改 postgres.conf 参数
enable_security_policy=on
shared_preload_libraries = 'pg_stat_statements,security_plugin'

5.重启数据库使参数生效
vb_ctl restart
```

5.5.3. 审计策略维护

Vastbase 审计策略由两部分组成：资源池和审计约束。资源池指的是待审计的对象，审计约束则限制了可审计的操作，例如限定了用户名称、IP、操作类型等。

5.5.3.1.资源池

Vastbase 提供 SQL 命令维护资源池，相关语法如下：

```
CREATE RESOURCE LABEL [IF NOT EXISTS] label_name [ADD object_type (objects)];
DROP RESOURCE LABEL [IF EXISTS] resource_labels;
ALTER RESOURCE LABEL label_name {ADD | REMOVE} resource_type (objects);
```

其中 label_name 为资源池名称，ADD 子句是可选的，用来指定资源池中的对象，object_type 为对象类型，目前支持的类型详见 5.5.1.1 可审计对象与事件类型，objects 是对象名称列表，多个对象之间用逗号分隔。resource_labels 是已经创建的资源池名称（若有多个资源池，则用逗号分隔）。

例如：创建一个名称为 l1 的资源池，该资源池中有两类对象：表和视图，其中表名称为 t1、t2，视图名称为 v1。

```
CREATE RESOURCE LABEL l1 ADD TABLE (t1, t2), VIEW (v1);
```

从资源池 l1 中删除视图 v1。

```
ALTER RESOURCE LABEL l1 REMOVE VIEW (v1);
```

5.5.3.2.审计策略

Vastbase 提供 SQL 命令维护审计策略，相关语法如下：

```
CREATE AUDIT POLICY [IF NOT EXISTS] name {PRIVILEGES privilege_list | ACCESS access_list} FILTER
ON expression [{normally|warning|block}];
DROP AUDIT POLICY [IF EXISTS] names;
ALTER AUDIT POLICY [ IF EXISTS ] policy_name { ADD | REMOVE } { [ privilege_audit_clause ] [
access_audit_clause ] };
ALTER AUDIT POLICY [ IF EXISTS ] policy_name DROP FILTER;
```

其中 name 为策略的名称，privilege_list 和 access_list 分别对应审计的操作类型，FILTER 子句则限定了操作的发起 IP、用户和客户端工具类型。

响应动作可填写 normally、warning 或 block，对应为记录普通级别日志、记录警告级别日志或阻断会话。不填默认为记录普通级别日志，

例如：创建了一个名称为 p_tbl 的审计策略，该策略会审计通过 gsql 客户端工具或者（从 127.0.0.1 发起且用户为 vastbase）对资源池 tbl_label 中的对象进行插入或者 update 的操作。

```
   CREATE AUDIT POLICY p_tbl ACCESS insert ON LABEL (tbl_label), update ON LABEL (tbl_label) FILTER
ON IP ('127.0.0.1') AND ROLES (vastbase) OR APP (gsql);
```

5.5.4. 查看审计日志

Vastbase 中审计日志产生后以文本记录到审计日志文件/var/log/vbaudit.log 中，可在操作系统中进行查看。或者可以通过查询数据库中的表 audit.vb_audit_log 来查看审计日志。

示例

```
1.创建用户
CREATE USER lst PASSWORD 'Bigdata@123';
grant all privileges to lst;
2.创建审计策略
create table t1(id int,col text);
insert into t1 values(1,'1');
insert into t1 values(2,'2');
CREATE AUDIT POLICY p_tbl1 PRIVILEGES DROP FILTER ON IP ('127.0.0.1') AND ROLES (lst) normally;
3.触发审计，切换到用户 lst 删除表
\c - lst
drop table t1;
4.查看审计日志
select * from audit.vb_audit_log;
```

5.6. 存储加密功能

功能描述

vastbase 数据库采用软加密方式实现数据加密功能，当用户向数据库中写入数据时，数据自动加密；当用户从数据库中查询数据时，数据被自动解密。

示例

```
先对 key 进行编码：
echo -n "0123456789012345" | base64
结果为：MDEyMzQ1Njc4OTAxMjM0NQ==

初始化实例：
gs_initdb -w Aa@123456 -D $PGDATA --nodename='sgnode' -K MDEyMzQ1Njc4OTAxMjM0NQ==
密钥配置
```

```
gs_encrypt -v ABCDEFGHIJKLMNOP -B MDEyMzQ1Njc4OTAxMjM0NQ==TRANS_ENCRYPT_SAMPLE_STRING
执行结果为
HDlPicpUuhkRFAGLhcIH2RFVGMp9LgDziwqUR3CVbRR/ZHvocH3odowS7TDXUE1SQUJDREVGR0hJSktMTU5PUA==

参数配置
transparent_encrypted_string =
'HDlPicpUuhkRFAGLhcIH2RFVGMp9LgDziwqUR3CVbRR/ZHvocH3odowS7TDXUE1SQUJDREVGR0hJSktMTU5PUA=='

启动数据库（确保端口没被占用）
vb_ctl start

创建表并插入数据
vastbase=# create table test_pwd (id int,name text);
CREATE TABLE
vastbase=# insert into test_pwd values (1,'加密');
vastbase=# select * from test pwd;
 id | name
----+------
  1 | 加密
(1 row)

进入数据目录（$PGDATA/base/）查看是否加密成功：
[klk@db1 15659]$ grep -iR '加密'  ##没返回则表示加密成功
[klk@db1 15659]$
```

6. 接口参考

6.1. JDBC

JDBC 接口是一套提供给用户的 API 方法，本节将对部分常用接口做具体描述，若涉及其他接口可参考 JDK1.8（软件包）/JDBC4.0 中相关内容。

6.1.1. java.sql.Connection

java.sql.Connection 是数据库连接接口。

表 6-1. 对 java.sql.Connection 接口的支持情况

方法名	返回值类型	支持 JDBC 4
close()	void	Yes
commit()	void	Yes
createStatement()	Statement	Yes
getAutoCommit()	Boolean	Yes

方法名	返回值类型	支持 JDBC 4
getClientInfo()	Properties	Yes
getClientInfo(String name)	String	Yes
getTransactionIsolation()	int	Yes
isClosed()	Boolean	Yes
isReadOnly()	Boolean	Yes
prepareStatement(String sql)	PreparedStatement	Yes
rollback()	void	Yes
setAutoCommit(boolean autoCommit)	void	Yes
setClientInfo(Properties properties)	void	Yes
setClientInfo(String name,String value)	void	Yes

须知

接口内部默认使用自动提交模式，若通过 setAutoCommit(false)关闭自动提交，将会导致后面执行的语句都受到显式事务包裹，数据库中不支持事务中执行的语句不能在此模式下执行。

6.1.2. java.sql.CallableStatement

java.sql.CallableStatement 是存储过程执行接口。

表 6-2. 对 java.sql.CallableStatement 的支持情况

方法名	返回值类型	支持 JDBC 4
registerOutParameter(int parameterIndex, int type)	void	Yes
wasNull()	Boolean	Yes
getString(int parameterIndex)	String	Yes
getBoolean(int parameterIndex)	Boolean	Yes
getByte(int parameterIndex)	byte	Yes
getShort(int parameterIndex)	short	Yes
getInt(int parameterIndex)	int	Yes
getLong(int parameterIndex)	long	Yes
getFloat(int parameterIndex)	float	Yes
getDouble(int parameterIndex)	double	Yes

方法名	返回值类型	支持 JDBC 4
getBigDecimal(int parameterIndex)	BigDecimal	Yes
getBytes(int parameterIndex)	byte[]	Yes
getDate(int parameterIndex)	Date	Yes
getTime(int parameterIndex)	Time	Yes
getTimestamp(int parameterIndex)	Timestamp	Yes
getObject(int parameterIndex)	Object	Yes

说明

- 不允许含有 OUT 参数的语句执行批量操作。
- 以下方法是从 java.sql.Statement 继承而来：close，execute，executeQuery，executeUpdate，getConnection，getResultSet，getUpdateCount，isClosed，setMaxRows， setFetchSize。
- 以下方法是从 java.sql.PreparedStatement 继承而来：addBatch，clearParameters，execute，executeQuery，executeUpdate，getMetaData，setBigDecimal，setBoolean，setByte，setBytes，setDate，setDouble，setFloat，setInt，setLong，setNull，setObject，setString，setTime，setTimestamp 。

6.1.3. java.sql.DatabaseMetaData

java.sql.DatabaseMetaData 是数据库对象定义接口。

表 6-3. 对 java.sql.DatabaseMetaData 的支持情况

方法名	返回值类型	支持 JDBC 4
getTables(String catalog, String schemaPattern, String tableNamePattern, String[] types)	ResultSet	Yes
getColumns(String catalog, String schemaPattern, String tableNamePattern, String columnNamePattern)	ResultSet	Yes
getTableTypes()	ResultSet	Yes
getUserName()	String	Yes
isReadOnly()	boolean	Yes
nullsAreSortedHigh()	boolean	Yes
nullsAreSortedLow()	boolean	Yes
nullsAreSortedAtStart()	boolean	Yes
nullsAreSortedAtEnd()	boolean	Yes
getDatabaseProductName()	String	Yes
getDatabaseProductVersion()	String	Yes

方法名	返回值类型	支持 JDBC 4
getDriverName()	String	Yes
getDriverVersion()	String	Yes
getDriverMajorVersion()	int	Yes
getDriverMinorVersion()	int	Yes
usesLocalFiles()	boolean	Yes
usesLocalFilePerTable()	boolean	Yes
supportsMixedCaseIdentifiers()	boolean	Yes
storesUpperCaseIdentifiers()	boolean	Yes
storesLowerCaseIdentifiers()	boolean	Yes
supportsMixedCaseQuotedIdentifiers()	boolean	Yes
storesUpperCaseQuotedIdentifiers()	boolean	Yes
storesLowerCaseQuotedIdentifiers()	boolean	Yes
storesMixedCaseQuotedIdentifiers()	boolean	Yes
supportsAlterTableWithAddColumn()	boolean	Yes
supportsAlterTableWithDropColumn()	boolean	Yes
supportsColumnAliasing()	boolean	Yes
nullPlusNonNullIsNull()	boolean	Yes
supportsConvert()	boolean	Yes
supportsConvert(int fromType, int toType)	boolean	Yes
supportsTableCorrelationNames()	boolean	Yes
supportsDifferentTableCorrelationNames()	boolean	Yes
supportsExpressionsInOrderBy()	boolean	Yes
supportsOrderByUnrelated()	boolean	Yes
supportsGroupBy()	boolean	Yes
supportsGroupByUnrelated()	boolean	Yes
supportsGroupByBeyondSelect()	boolean	Yes
supportsLikeEscapeClause()	boolean	Yes
supportsMultipleResultSets()	boolean	Yes
supportsMultipleTransactions()	boolean	Yes
supportsNonNullableColumns()	boolean	Yes
supportsMinimumSQLGrammar()	boolean	Yes

方法名	返回值类型	支持 JDBC 4
supportsCoreSQLGrammar()	boolean	Yes
supportsExtendedSQLGrammar()	boolean	Yes
supportsANSI92EntryLevelSQL()	boolean	Yes
supportsANSI92IntermediateSQL()	boolean	Yes
supportsANSI92FullSQL()	boolean	Yes
supportsIntegrityEnhancementFacility()	boolean	Yes
supportsOuterJoins()	boolean	Yes
supportsFullOuterJoins()	boolean	Yes
supportsLimitedOuterJoins()	boolean	Yes
isCatalogAtStart()	boolean	Yes
supportsSchemasInDataManipulation()	boolean	Yes
supportsSavepoints()	boolean	Yes
supportsResultSetHoldability(int holdability)	boolean	Yes
getResultSetHoldability()	int	Yes
getDatabaseMajorVersion()	int	Yes
getDatabaseMinorVersion()	int	Yes
getJDBCMajorVersion()	int	Yes
getJDBCMinorVersion()	int	Yes

6.1.4. java.sql.Driver

java.sql.Driver 是数据库驱动接口。

表 6-4. 对 java.sql.Driver 的支持情况

方法名	返回值类型	支持 JDBC 4
acceptsURL(String url)	Boolean	Yes
connect(String url, Properties info)	Connection	Yes
jdbcCompliant()	Boolean	Yes
getMajorVersion()	int	Yes
getMinorVersion()	int	Yes

6.1.5. java.sql.PreparedStatement

java.sql.PreparedStatement 是预处理语句接口。

表 6-5. 对 java.sql.PreparedStatement 的支持情况

方法名	返回值类型	支持 JDBC 4
clearParameters()	void	Yes
execute()	Boolean	Yes
executeQuery()	ResultSet	Yes
excuteUpdate()	int	Yes
getMetaData()	ResultSetMetaData	Yes
setBoolean(int parameterIndex, boolean x)	void	Yes
setBigDecimal(int parameterIndex, BigDecimal x)	void	Yes
setByte(int parameterIndex, byte x)	void	Yes
setBytes(int parameterIndex, byte[] x)	void	Yes
setDate(int parameterIndex, Date x)	void	Yes
setDouble(int parameterIndex, double x)	void	Yes
setFloat(int parameterIndex, float x)	void	Yes
setInt(int parameterIndex, int x)	void	Yes
setLong(int parameterIndex, long x)	void	Yes
setShort(int parameterIndex, short x)	void	Yes
setString(int parameterIndex, String x)	void	Yes
addBatch()	void	Yes
executeBatch()	int[]	Yes
clearBatch()	void	Yes

说明

- addBatch()、execute()必须在 clearBatch()之后才能执行。
- 调用 executeBatch()方法并不会清除 batch。用户必须显式使用 clearBatch()清除 。
- 在添加了一个 batch 的绑定变量后，用户若想重用这些值(再次添加一个 batch)，无需再次使用 set*()方法 。
- 以下方法是从 java.sql.Statement 继承而来：close，execute，executeQuery，executeUpdate，getConnection，getResultSet，getUpdateCount，isClosed，setMaxRows， setFetchSize。

6.1.6. java.sql.ResultSet

java.sql.ResultSet 是执行结果集接口。

表 6-6. 对 java.sql.ResultSet 的支持情况

方法名	返回值类型	支持 JDBC 4
findColumn(String columnLabel)	int	Yes
getBigDecimal(int columnIndex)	BigDecimal	Yes
getBigDecimal(String columnLabel)	BigDecimal	Yes
getBoolean(int columnIndex)	Boolean	Yes
getBoolean(String columnLabel)	Boolean	Yes
getByte(int columnIndex)	byte	Yes
getBytes(int columnIndex)	byte[]	Yes
getByte(String columnLabel)	byte	Yes
getBytes(String columnLabel)	byte[]	Yes
getDate(int columnIndex)	Date	Yes
getDate(String columnLabel)	Date	Yes
getDouble(int columnIndex)	double	Yes
getDouble(String columnLabel)	double	Yes
getFloat(int columnIndex)	float	Yes
getFloat(String columnLabel)	float	Yes
getInt(int columnIndex)	int	Yes
getInt(String columnLabel)	int	Yes
getLong(int columnIndex)	long	Yes
getLong(String columnLabel)	long	Yes
getShort(int columnIndex)	short	Yes
getShort(String columnLabel)	short	Yes

方法名	返回值类型	支持 JDBC 4
getString(int columnIndex)	String	Yes
getString(String columnLabel)	String	Yes
getTime(int columnIndex)	Time	Yes
getTime(String columnLabel)	Time	Yes
getTimestamp(int columnIndex)	Timestamp	Yes
getTimestamp(String columnLabel)	Timestamp	Yes
isAfterLast()	Boolean	Yes
isBeforeFirst()	Boolean	Yes
isFirst()	Boolean	Yes
next()	Boolean	Yes

说明

- 一个 Statement 不能有多个处于“open”状态的 ResultSet。
- 用于遍历结果集(ResultSet)的游标(Cursor)在被提交后不能保持“open”的状态。

6.1.7. java.sql.ResultSetMetaData

java.sql.ResultSetMetaData 是对 ResultSet 对象相关信息的具体描述。

表 6-7. 对 java.sql.ResultSetMetaData 的支持情况

方法名	返回值类型	支持 JDBC 4
getColumnCount()	int	Yes
getColumnName(int column)	String	Yes
getColumnType(int column)	int	Yes
getColumnTypeName(int column)	String	Yes

6.1.8. java.sql.Statement

java.sql.Statement 是 SQL 语句接口。

表 6-8. 对 java.sql.Statement 的支持情况

方法名	返回值类型	支持 JDBC 4

方法名	返回值类型	支持 JDBC 4
close()	void	Yes
execute(String sql)	Boolean	Yes
executeQuery(String sql)	ResultSet	Yes
executeUpdate(String sql)	int	Yes
getConnection()	Connection	Yes
getResultSet()	ResultSet	Yes
getQueryTimeout()	int	Yes
getUpdateCount()	int	Yes
isClosed()	Boolean	Yes
setQueryTimeout(int seconds)	void	Yes
setFetchSize(int rows)	void	Yes
cancel()	void	Yes

说明

- 通过 setFetchSize 可以减少结果集在客户端的内存占用情况。它的原理是通过将结果集打包成游标，然后分段处理，所以会加大数据库与客户端的通信量，会有性能损耗。
- 由于数据库游标是事务内有效，所以，在设置 setFetchSize 的同时，需要将连接设置为非自动提交模式，setAutoCommit(false)。同时在业务数据需要持久化到数据库中时，在连接上执行提交操作。

6.1.9. javax.sql.ConnectionPoolDataSource

javax.sql.ConnectionPoolDataSource 是数据源连接池接口。

表 6-9. 对 javax.sql.ConnectionPoolDataSource 的支持情况

方法名	返回值类型	支持 JDBC 4
getLoginTimeout()	int	Yes
getLogWriter()	PrintWriter	Yes
getPooledConnection()	PooledConnection	Yes
getPooledConnection(String user,String password)	PooledConnection	Yes
setLoginTimeout(int seconds)	void	Yes
setLogWriter(PrintWriter out)	void	Yes

6.1.10. javax.sql.DataSource

javax.sql.DataSource 是数据源接口。

表 6-10. 对 javax.sql.DataSource 接口的支持情况

方法名	返回值类型	支持 JDBC 4
getConneciton()	Connection	Yes
getConnection(String username,String password)	Connection	Yes
getLoginTimeout()	int	Yes
getLogWriter()	PrintWriter	Yes
setLoginTimeout(int seconds)	void	Yes
setLogWriter(PrintWriter out)	void	Yes

6.1.11. javax.sql.PooledConnection

javax.sql.PooledConnection 是由连接池创建的连接接口。

表 6-11. 对 javax.sql.PooledConnection 的支持情况

方法名	返回值类型	支持 JDBC 4
addConnectionEventListener (ConnectionEventListener listener)	void	Yes
close()	void	Yes
getConnection()	Connection	Yes
removeConnectionEventListener (ConnectionEventListener listener)	void	Yes
addStatementEventListener (StatementEventListener listener)	void	Yes
removeStatementEventListener (StatementEventListener listener)	void	Yes

6.1.12. javax.naming.Context

javax.naming.Context 是连接配置的上下文接口。

表 6-12. 对 javax.naming.Context 的支持情况

方法名	返回值类型	支持 JDBC 4
bind(Name name, Object obj)	void	Yes
bind(String name, Object obj)	void	Yes
lookup(Name name)	Object	Yes
lookup(String name)	Object	Yes
rebind(Name name, Object obj)	void	Yes
rebind(String name, Object obj)	void	Yes
rename(Name oldName, Name newName)	void	Yes
rename(String oldName, String newName)	void	Yes
unbind(Name name)	void	Yes
unbind(String name)	void	Yes

6.1.13. javax.naming.spi.InitialContextFactory

javax.naming.spi.InitialContextFactory 是初始连接上下文工厂接口。

表 6-13. 对 javax.naming.spi.InitialContextFactory 的支持情况

方法名	返回值类型	支持 JDBC 4
getInitialContext(Hashtable<?,?> environment)	Context	Yes

6.1.14. CopyManager

CopyManager 是 Vastbase JDBC 驱动中提供的一个 API 接口类，用于批量向 Vastbase 中导入数据。

CopyManager 的继承关系

CopyManager 类位于 org.postgresql.copy Package 中，继承自 java.lang.Object 类，该类的声明如下：

```
public class CopyManager
extends Object
```

构造方法

public CopyManager(BaseConnection connection)
throws SQLException

常用方法

表 6-14. CopyManager 常用方法

返回值	方法	描述	THROWS
CopyIn	copyIn(String sql)	-	SQLException
long	copyIn(String sql, InputStream from)	使用 COPY FROM STDIN 从 InputStream 中快速向数据库中的表加载数据。	SQLException,IOException
long	copyIn(String sql, InputStream from, int bufferSize)	使用 COPY FROM STDIN 从 InputStream 中快速向数据库中的表加载数据。	SQLException,IOException
long	copyIn(String sql, Reader from)	使用 COPY FROM STDIN 从 Reader 中快速向数据库中的表加载数据。	SQLException,IOException
long	copyIn(String sql, Reader from, int bufferSize)	使用 COPY FROM STDIN 从 Reader 中快速向数据库中的表加载数据。	SQLException,IOException
CopyOut	copyOut(String sql)	-	SQLException
long	copyOut(String sql, OutputStream to)	将一个 COPY TO STDOUT 的结果集从数据库发送到 OutputStream 类中。	SQLException,IOException
long	copyOut(String sql, Writer to)	将一个 COPY TO STDOUT 的结果集从数据库发送到 Writer 类	SQLException,IOException

返回值	方法	描述	THROWS
		中。	

6.2. ODBC

ODBC 接口是一套提供给用户的 API 函数，本节将对部分常用接口做具体描述，若涉及其他接口可参考 msdn（网址：https://msdn.microsoft.com/en-us/library/windows/desktop/ms714177(v=vs.85).aspx）中 ODBC Programmer's Reference 项的相关内容。

6.2.1. SQLAllocEnv

在 ODBC 3.x 版本中，ODBC 2.x 的函数 SQLAllocEnv 已被 SQLAllocHandle 代替。有关详细信息请参阅 6.2.3SQLAllocHandle。

6.2.2. SQLAllocConnect

在 ODBC 3.x 版本中，ODBC 2.x 的函数 SQLAllocConnect 已被 SQLAllocHandle 代替。有关详细信息请参阅 6.2.3SQLAllocHandle。

6.2.3. SQLAllocHandle

功能描述

分配环境、连接、语句或描述符的句柄，它替代了 ODBC 2.x 函数 SQLAllocEnv、SQLAllocConnect 及 SQLAllocStmt。

原型

```
SQLRETURN SQLAllocHandle(SQLSMALLINT   HandleType,
                         SQLHANDLE     InputHandle,
                         SQLHANDLE     *OutputHandlePtr);
```

参数

表 6-15. SQLAllocHandle 参数

关键字	参数说明
HandleType	由 SQLAllocHandle 分配的句柄类型。必须为下列值之一：

关键字	参数说明
	• SQL_HANDLE_ENV（环境句柄） • SQL_HANDLE_DBC（连接句柄） • SQL_HANDLE_STMT（语句句柄） • SQL_HANDLE_DESC（描述句柄） 申请句柄顺序为，先申请环境句柄，再申请连接句柄，最后申请语句句柄，后申请的句柄都要依赖它前面申请的句柄。
InputHandle	将要分配的新句柄的类型。 • 如果 HandleType 为 SQL_HANDLE_ENV，则这个值为 SQL_NULL_HANDLE。 • 如果 HandleType 为 SQL_HANDLE_DBC，则这一定是一个环境句柄。 • 如果 HandleType 为 SQL_HANDLE_STMT 或 SQL_HANDLE_DESC，则它一定是一个连接句柄。
OutputHandlePtr	输出参数：一个缓冲区的指针，此缓冲区以新分配的数据结构存放返回的句柄。

返回值

- SQL_SUCCESS：表示调用正确，
- SQL_SUCCESS_WITH_INFO：表示会有一些警告信息，
- SQL_ERROR：表示比较严重的错误，如：内存分配失败、建立连接失败等、
- SQL_INVALID_HANDLE：表示调用无效句柄。其他 API 的返回值同理。

注意事项

当分配的句柄并非环境句柄时，如果 SQLAllocHandle 返回的值为 SQL_ERROR，则它会将 OutputHandlePtr 的值设置为 SQL_NULL_HDBC、SQL_NULL_HSTMT 或 SQL_NULL_HDESC。之后，通过调用带有适当参数的 6.2.19SQLGetDiagRec，其中 HandleType 和 Handle 被设置为 IntputHandle 的值，可得到相关的 SQLSTATE 值，通过 SQLSTATE 值可以查出调用此函数的具体信息。

示例

参见：6.2.23 示例

6.2.4. SQLAllocStmt

在 ODBC 3.x 版本中，ODBC 2.x 的函数 SQLAllocStmt 已被 SQLAllocHandle 代替。有关详细信息请参阅 6.2.3SQLAllocHandle。

6.2.5. SQLBindCol

功能描述

将应用程序数据缓冲区绑定到结果集的列中。

原型

```
SQLRETURN SQLBindCol(SQLHSTMT      StatementHandle,
                     SQLUSMALLINT   ColumnNumber,
                     SQLSMALLINT    TargetType,
                     SQLPOINTER     TargetValuePtr,
                     SQLLEN     BufferLength,
                     SQLLEN     *StrLen_or_IndPtr);
```

参数

表 6-16. SQLBindCol 参数

关键字	参数说明
StatementHandle	语句句柄。
ColumnNumber	要绑定结果集的列号。起始列号为 0，以递增的顺序计算列号，第 0 列是书签列。若未设置书签页，则起始列号为 1。
TargetType	缓冲区中 C 数据类型的标识符。
TargetValuePtr	输出参数：指向与列绑定的数据缓冲区的指针。SQLFetch 函数返回这个缓冲区中的数据。如果此参数为一个空指针，则 StrLen_or_IndPtr 是一个有效值。
BufferLength	TargetValuePtr 指向缓冲区的长度，以字节为单位。
StrLen_or_IndPtr	输出参数：缓冲区的长度或指示器指针。若为空值，则未使用任何长度或指示器值。

返回值

- SQL_SUCCESS：表示调用正确。

- SQL_SUCCESS_WITH_INFO：表示会有一些警告信息。
- SQL_ERROR：表示比较严重的错误，如：内存分配失败、建立连接失败等。
- SQL_INVALID_HANDLE：表示调用无效句柄。其他 API 的返回值同理。

注意事项

当 SQLBindCol 返回 SQL_ERROR 或 SQL_SUCCESS_WITH_INFO 时，通过调用 6.2.19SQLGetDiagRec 函数，并将 HandleType 和 Handle 参数设置为 SQL_HANDLE_STMT 和 StatementHandle，可得到一个相关的 SQLSTATE 值，通过 SQLSTATE 值可以查出调用此函数的具体信息。

示例

参见：6.2.23 示例

6.2.6. SQLBindParameter

功能描述

将一条 SQL 语句中的一个参数标志和一个缓冲区绑定起来。

原型

```
SQLRETURN SQLBindParameter(SQLHSTMT      StatementHandle,
                      SQLUSMALLINT   ParameterNumber,
                      SQLSMALLINT    InputOutputType,
                      SQLSMALLINT    ValuetType,
                      SQLSMALLINT    ParameterType,
                      SQLULEN    ColumnSize,
                      SQLSMALLINT    DecimalDigits,
                      SQLPOINTER     ParameterValuePtr,
                      SQLLEN     BufferLength,
                      SQLLEN     *StrLen_or_IndPtr);
```

参数

表 6-17. SQLBindParameter

关键词	参数说明
StatementHandle	语句句柄。
ParameterNumber	参数序号，起始为 1，依次递增。
InputOutputType	输入输出参数类型。
ValueType	参数的 C 数据类型。

关键词	参数说明
ParameterType	参数的 SQL 数据类型。
ColumnSize	列的大小或相应参数标记的表达式。
DecimalDigits	列的十进制数字或相应参数标记的表达式。
ParameterValuePtr	指向存储参数数据缓冲区的指针。
BufferLength	ParameterValuePtr 指向缓冲区的长度，以字节为单位。
StrLen_or_IndPtr	缓冲区的长度或指示器指针。若为空值，则未使用任何长度或指示器值。

返回值

- SQL_SUCCESS：表示调用正确。
- SQL_SUCCESS_WITH_INFO：表示会有一些警告信息。
- SQL_ERROR：表示比较严重的错误，如：内存分配失败、建立连接失败等。
- SQL_INVALID_HANDLE：表示调用无效句柄。其他 API 的返回值同理。

注意事项

当 SQLBindParameter 返回 SQL_ERROR 或 SQL_SUCCESS_WITH_INFO 时，通过调用 6.2.19SQLGetDiagRec 函数，并将 HandleType 和 Handle 参数设置为 SQL_HANDLE_STMT 和 StatementHandle，可得到一个相关的 SQLSTATE 值，通过 SQLSTATE 值可以查出调用此函数的具体信息。

示例

参见：6.2.23 示例

6.2.7. SQLColAttribute

功能描述

返回结果集中一列的描述符信息。

原型

```
SQLRETURN SQLColAttibute(SQLHSTMT       StatementHandle,
                    SQLUSMALLINT    ColumnNumber,
                    SQLUSMALLINT    FieldIdentifier,
                    SQLPOINTER     CharacterAtrributePtr,
```

```
                    SQLSMALLINT    BufferLength,
                    SQLSMALLINT    *StringLengthPtr,
                    SQLLEN      *NumericAttributePtr);
```

参数

表 6-18. SQLColAttribute 参数

关键字	参数说明
StatementHandle	语句句柄。
ColumnNumber	要检索字段的列号，起始为 1，依次递增。
FieldIdentifier	IRD 中 ColumnNumber 行的字段。
CharacterAttributePtr	输出参数：一个缓冲区指针，返回 FieldIdentifier 字段值。
BufferLength	• 如果 FieldIdentifier 是一个 ODBC 定义的字段，而且 CharacterAttributePtr 指向一个字符串或二进制缓冲区，则此参数为该缓冲区的长度。 • 如果 FieldIdentifier 是一个 ODBC 定义的字段，而且 CharacterAttributePtr 指向一个整数，则会忽略该字段。
StringLengthPtr	输出参数：缓冲区指针，存放*CharacterAttributePtr 中字符类型数据的字节总数，对于非字符类型，忽略 BufferLength 的值。
NumericAttributePtr	输出参数：指向一个整型缓冲区的指针，返回 IRD 中 ColumnNumber 行 FieldIdentifier 字段的值。

返回值

- SQL_SUCCESS：表示调用正确。
- SQL_SUCCESS_WITH_INFO：表示会有一些警告信息。
- SQL_ERROR：表示比较严重的错误，如：内存分配失败、建立连接失败等。
- SQL_INVALID_HANDLE：表示调用无效句柄。其他 API 的返回值同理。

注意事项

当 SQLColAttribute 返回 SQL_ERROR 或 SQL_SUCCESS_WITH_INFO 时，通过调用 6.2.19SQLGetDiagRec 函数，并将 HandleType 和 Handle 参数设置为 SQL_HANDLE_STMT 和 StatementHandle，可得到一个相关的 SQLSTATE 值，通过 SQLSTATE 值可以查出调用此函数的具体信息。

示例

参见：6.2.23 示例

6.2.8. SQLConnect

功能描述

在驱动程序和数据源之间建立连接。连接上数据源之后，可以通过连接句柄访问到所有有关连接数据源的信息，包括程序运行状态、事务处理状态和错误信息。

原型

```
SQLRETURN  SQLConnect(SQLHDBC       ConnectionHandle,
                      SQLCHAR       *ServerName,
                      SQLSMALLINT    NameLength1,
                      SQLCHAR       *UserName,
                      SQLSMALLINT    NameLength2,
                      SQLCHAR       *Authentication,
                      SQLSMALLINT    NameLength3);
```

参数

表 6-19. SQLConnect 参数

关键字	参数说明
ConnectionHandle	连接句柄，通过 SQLAllocHandle 获得。
ServerName	要连接数据源的名称。
NameLength1	ServerName 的长度。
UserName	数据源中数据库用户名。
NameLength2	UserName 的长度。
Authentication	数据源中数据库用户密码。
NameLength3	Authentication 的长度。

返回值

- SQL_SUCCESS：表示调用正确。
- SQL_SUCCESS_WITH_INFO：表示会有一些警告信息。
- SQL_ERROR：表示比较严重的错误，如：内存分配失败、建立连接失败等。

- SQL_INVALID_HANDLE：表示调用无效句柄。其他 API 的返回值同理。
- SQL_STILL_EXECUTING：表示语句正在执行。

注意事项

当调用 SQLConnect 函数返回 SQL_ERROR 或 SQL_SUCCESS_WITH_INFO 时，通过调用 6.2.19SQLGetDiagRec 函数，并将 HandleType 和 Handle 参数设置为 SQL_HANDLE_DBC 和 ConnectionHandle，可得到一个相关的 SQLSTATE 值，通过 SQLSTATE 值可以查出调用此函数的具体信息。

示例

参见：6.2.23 示例

6.2.9. SQLDisconnect

功能描述

关闭一个与特定连接句柄相关的连接。

原型

```
SQLRETURN SQLDisconnect(SQLHDBC    ConnectionHandle);
```

参数

表 6-20. SQLDisconnect 参数

关键字	参数说明
ConnectionHandle	连接句柄，通过 SQLAllocHandle 获得。

返回值

- SQL_SUCCESS：表示调用正确。
- SQL_SUCCESS_WITH_INFO：表示会有一些警告信息。
- SQL_ERROR：表示比较严重的错误，如：内存分配失败、建立连接失败等。
- SQL_INVALID_HANDLE：表示调用无效句柄。其他 API 的返回值同理。

注意事项

当调用 SQLDisconnect 函数返回 SQL_ERROR 或 SQL_SUCCESS_WITH_INFO 时，通过调用 6.2.19SQLGetDiagRec 函数，并将 HandleType 和 Handle 参数设置为 SQL_HANDLE_DBC 和 ConnectionHandle，可得到一个相关的 SQLSTATE 值，通过 SQLSTATE 值可以查出调用此函数的具体信息。

示例

参见：6.2.23 示例

6.2.10. SQLExecDirect

功能描述

使用参数的当前值，执行一条准备好的语句。对于一次只执行一条 SQL 语句，SQLExecDirect 是最快的执行方式。

原型

```
SQLRETURN SQLExecDirect(SQLHSTMT        StatementHandle,
                        SQLCHAR        *StatementText,
                        SQLINTEGER      TextLength);
```

参数

表 6-21. SQLExecDirect 参数

关键字	参数说明
StatementHandle	语句句柄，通过 SQLAllocHandle 获得。
StatementText	要执行的 SQL 语句。不支持一次执行多条语句。
TextLength	StatementText 的长度。

返回值

- SQL_SUCCESS：表示调用正确。
- SQL_SUCCESS_WITH_INFO：表示会有一些警告信息。
- SQL_NEED_DATA：在执行 SQL 语句前没有提供足够的参数。
- SQL_ERROR：表示比较严重的错误，如：内存分配失败、建立连接失败等。
- SQL_INVALID_HANDLE：表示调用无效句柄。其他 API 的返回值同理。

- SQL_STILL_EXECUTING：表示语句正在执行。
- SQL_NO_DATA：表示 SQL 语句不返回结果集。

注意事项

当调用 SQLExecDirect 函数返回 SQL_ERROR 或 SQL_SUCCESS_WITH_INFO 时，通过调用 6.2.19SQLGetDiagRec 函数，并将 HandleType 和 Handle 参数设置为 SQL_HANDLE_STMT 和 StatementHandle，可得到一个相关的 SQLSTATE 值，通过 SQLSTATE 值可以查出调用此函数的具体信息。

示例

参见：6.2.23 示例

6.2.11. SQLExecute

功能描述

如果语句中存在参数标记的话，SQLExecute 函数使用参数标记参数的当前值，执行一条准备好的 SQL 语句。

原型

```
SQLRETURN SQLExecute(SQLHSTMT    StatementHandle);
```

参数

表 6-22. SQLExecute 参数

关键字	参数说明
StatementHandle	要执行语句的语句句柄。

返回值

- SQL_SUCCESS：表示调用正确。
- SQL_SUCCESS_WITH_INFO：表示会有一些警告信息。
- SQL_NEED_DATA：表示在执行 SQL 语句前没有提供足够的参数。
- SQL_ERROR：表示比较严重的错误，如：内存分配失败、建立连接失败等。
- SQL_NO_DATA：表示 SQL 语句不返回结果集。
- SQL_INVALID_HANDLE：表示调用无效句柄。其他 API 的返回值同理。
- SQL_STILL_EXECUTING：表示语句正在执行。

注意事项

当 SQLExecute 函数返回 SQL_ERROR 或 SQL_SUCCESS_WITH_INFO 时，可通过调用 6.2.19SQLGetDiagRec 函数，并将 HandleType 和 Handle 参数设置为 SQL_HANDLE_STMT 和 StatementHandle，可得到一个相关的 SQLSTATE 值，通过 SQLSTATE 值可以查出调用此函数的具体信息。

示例

参见：6.2.23 示例

6.2.12. SQLFetch

功能描述

从结果集中取下一个行集的数据，并返回所有被绑定列的数据。

原型

```
SQLRETURN SQLFetch(SQLHSTMT    StatementHandle);
```

参数

表 6-23. SQLFetch 参数

关键字	参数说明
StatementHandle	语句句柄，通过 SQLAllocHandle 获得。

返回值

- SQL_SUCCESS：表示调用正确。
- SQL_SUCCESS_WITH_INFO：表示会有一些警告信息。
- SQL_ERROR：表示比较严重的错误，如：内存分配失败、建立连接失败等。
- SQL_NO_DATA：表示 SQL 语句不返回结果集。
- SQL_INVALID_HANDLE：表示调用无效句柄。其他 API 的返回值同理。
- SQL_STILL_EXECUTING：表示语句正在执行。

注意事项

当调用 SQLFetch 函数返回 SQL_ERROR 或 SQL_SUCCESS_WITH_INFO 时，通过调用 6.2.19SQLGetDiagRec 函数，并将 HandleType 和 Handle 参数设置为 SQL_HANDLE_STMT 和

StatementHandle，可得到一个相关的 SQLSTATE 值，通过 SQLSTATE 值可以查出调用此函数的具体信息。

示例

参见：6.2.23 示例

6.2.13. SQLFreeStmt

在 ODBC 3.x 版本中，ODBC 2.x 的函数 SQLFreeStmt 已被 SQLFreeHandle 代替。有关详细信息请参阅 6.2.15SQLFreeHandle。

6.2.14. SQLFreeConnect

在 ODBC 3.x 版本中，ODBC 2.x 的函数 SQLFreeConnect 已被 SQLFreeHandle 代替。有关详细信息请参阅 6.2.15SQLFreeHandle。

6.2.15. SQLFreeHandle

功能描述

释放与指定环境、连接、语句或描述符相关联的资源，它替代了 ODBC 2.x 函数 SQLFreeEnv、SQLFreeConnect 及 SQLFreeStmt。

原型

```
SQLRETURN SQLFreeHandle(SQLSMALLINT   HandleType,
                        SQLHANDLE     Handle);
```

参数

表 6-24. SQLFreeHandle 参数

关键字	参数说明
HandleType	SQLFreeHandle 要释放的句柄类型。必须为下列值之一： • SQL_HANDLE_ENV • SQL_HANDLE_DBC • SQL_HANDLE_STMT • SQL_HANDLE_DESC 如果 HandleType 不是这些值之一，SQLFreeHandle 返回

关键字	参数说明
	SQL_INVALID_HANDLE。
Handle	要释放的句柄。

返回值

- SQL_SUCCESS：表示调用正确。
- SQL_SUCCESS_WITH_INFO：表示会有一些警告信息。
- SQL_ERROR：表示比较严重的错误，如：内存分配失败、建立连接失败等。
- SQL_INVALID_HANDLE：表示调用无效句柄。其他 API 的返回值同理。

注意事项

如果 SQLFreeHandle 返回 SQL_ERROR，句柄仍然有效。

示例

参见：6.2.23 示例

6.2.16. SQLFreeEnv

在 ODBC 3.x 版本中，ODBC 2.x 的函数 SQLFreeEnv 已被 SQLFreeHandle 代替。有关详细信息请参阅 6.2.15SQLFreeHandle。

6.2.17. SQLPrepare

功能描述

准备一个将要进行的 SQL 语句。

原型

```
SQLRETURN SQLPrepare(SQLHSTMT     StatementHandle,
                     SQLCHAR      *StatementText,
                     SQLINTEGER    TextLength);
```

参数

表 6-25. SQLPrepare 参数

关键字	参数说明

关键字	参数说明
StatementHandle	语句句柄。
StatementText	SQL 文本串。
TextLength	StatementText 的长度。

返回值

- SQL_SUCCESS：表示调用正确。
- SQL_SUCCESS_WITH_INFO：表示会有一些警告信息。
- SQL_ERROR：表示比较严重的错误，如：内存分配失败、建立连接失败等。
- SQL_INVALID_HANDLE：表示调用无效句柄。其他 API 的返回值同理。
- SQL_STILL_EXECUTING：表示语句正在执行。

注意事项

当 SQLPrepare 返回的值为 SQL_ERROR 或 SQL_SUCCESS_WITH_INFO 时，通过调用 6.2.19SQLGetDiagRec 函数，并将 HandleType 和 Handle 参数分别设置为 SQL_HANDLE_STMT 和 StatementHandle，可得到一个相关的 SQLSTATE 值，通过 SQLSTATE 值可以查出调用此函数的具体信息。

示例

参见：6.2.23 示例

6.2.18. SQLGetData

功能描述

SQLGetData 返回结果集中某一列的数据。可以多次调用它来部分地检索不定长度的数据。

原型

```
SQLRETURN SQLGetData(SQLHSTMT       StatementHandle,
                     SQLUSMALLINT   Col_or_Param_Num,
                     SQLSMALLINT    TargetType,
                     SQLPOINTER     TargetValuePtr,
                     SQLLEN         BufferLength,
                     SQLLEN         *StrLen_or_IndPtr);
```

参数

表 6-26. SQLGetData 参数

关键字	参数说明
StatementHandle	语句句柄，通过 SQLAllocHandle 获得。
Col_or_Param_Num	要返回数据的列号。结果集的列按增序从 1 开始编号。书签列的列号为 0。
TargetType	TargetValuePtr 缓冲中的 C 数据类型的类型标识符。若 TargetType 为 SQL_ARD_TYPE，驱动使用 ARD 中 SQL_DESC_CONCISE_TYPE 字段的类型标识符。若为 SQL_C_DEFAULT，驱动根据源的 SQL 数据类型选择缺省的数据类型。
TargetValuePtr	输出参数：指向返回数据所在缓冲区的指针。
BufferLength	TargetValuePtr 所指向缓冲区的长度。
StrLen_or_IndPtr	输出参数：指向缓冲区的指针，在此缓冲区中返回长度或标识符的值。

返回值

- SQL_SUCCESS：表示调用正确。
- SQL_SUCCESS_WITH_INFO：表示会有一些警告信息。
- SQL_ERROR：表示比较严重的错误，如：内存分配失败、建立连接失败等。
- SQL_NO_DATA：表示 SQL 语句不返回结果集。
- SQL_INVALID_HANDLE：表示调用无效句柄。其他 API 的返回值同理。
- SQL_STILL_EXECUTING：表示语句正在执行。

注意事项

当调用 SQLGetData 函数返回 SQL_ERROR 或 SQL_SUCCESS_WITH_INFO 时，通过调用 6.2.19SQLGetDiagRec 函数，并将 HandleType 和 Handle 参数分别设置为 SQL_HANDLE_STMT 和 StatementHandle，可得到一个相关的 SQLSTATE 值，通过 SQLSTATE 值可以查出调用此函数的具体信息。

示例

参见：6.2.23 示例

6.2.19. SQLGetDiagRec

功能描述

返回诊断记录的多个字段的当前值，其中诊断记录包含错误、警告及状态信息。

原型

```
SQLRETURN  SQLGetDiagRec(SQLSMALLINT    HandleType
                         SQLHANDLE      Handle,
                         SQLSMALLINT    RecNumber,
                         SQLCHAR       *SQLState,
                         SQLINTEGER     *NativeErrorPtr,
                         SQLCHAR       *MessageText,
                         SQLSMALLINT    BufferLength
                         SQLSMALLINT    *TextLengthPtr);
```

参数

表 6-27. SQLGetDiagRec 参数

关键字	参数说明
HandleType	句柄类型标识符，它说明诊断所要求的句柄类型。必须为下列值之一： • SQL_HANDLE_ENV • SQL_HANDLE_DBC • SQL_HANDLE_STMT • SQL_HANDLE_DESC
Handle	诊断数据结构的句柄，其类型由 HandleType 来指出。如果 HandleType 是 SQL_HANDLE_ENV，Handle 可以是共享的或非共享的环境句柄。
RecNumber	指出应用从查找信息的状态记录。状态记录从 1 开始编号。
SQLState	输出参数：指向缓冲区的指针，该缓冲区存储着有关 RecNumber 的五字符的 SQLSTATE 码。
NativeErrorPtr	输出参数：指向缓冲区的指针，该缓冲区存储着本地的错误码。
MessageText	指向缓冲区的指针，该缓冲区存储着诊断信息文本串。
BufferLength	MessageText 的长度。
TextLengthPtr	输出参数：指向缓冲区的指针，返回 MessageText 中的字节总数。如果返回字节数大于 BufferLength，则 MessageText 中的诊断信息文本被截断成 BufferLength 减去

关键字	参数说明
	NULL 结尾字符的长度。

返回值

- SQL_SUCCESS：表示调用正确。
- SQL_SUCCESS_WITH_INFO：表示会有一些警告信息。
- SQL_ERROR：表示比较严重的错误，如：内存分配失败、建立连接失败等。
- SQL_INVALID_HANDLE：表示调用无效句柄。其他 API 的返回值同理。

注意事项

SQLGetDiagRec 不发布自己的诊断记录。它用下列返回值来报告它自己的执行结果：

- SQL_SUCCESS：函数成功返回诊断信息。
- SQL_SUCCESS_WITH_INFO：*MessageText 太小以致不能容纳所请求的诊断信息。没有诊断记录生成。
- SQL_INVALID_HANDLE：由 HandType 和 Handle 所指出的句柄是不合法句柄。
- SQL_ERROR：RecNumber 小于等于 0 或 BufferLength 小于 0。

如果调用 ODBC 函数返回 SQL_ERROR 或 SQL_SUCCESS_WITH_INFO，可调用 SQLGetDiagRec 返回诊断信息值 SQLSTATE，SQLSTATE 值的如下表。

表 6-28. SQLSTATE 值

SQLSATATE	错误	描述
HY000	一般错误	未定义特定的 SQLSTATE 所产生的一个错误。
HY001	内存分配错误	驱动程序不能分配所需要的内存来支持函数的执行或完成。
HY008	取消操作	调用 SQLCancel 取消执行语句后，依然在 StatementHandle 上调用函数。
HY010	函数系列错误	在为执行中的所有数据参数或列发送数据前就调用了执行函数。
HY013	内存管理错误	不能处理函数调用，可能由当前内存条件差引起。
HYT01	连接超时	数据源响应请求之前，连接超时。
IM001	驱动程序不支持此函数	调用了 StatementHandle 相关的驱动程序不支持的函数

示例

参见：6.2.23 示例

6.2.20. SQLSetConnectAttr

功能描述

设置控制连接各方面的属性。

原型

```
SQLRETURN SQLSetConnectAttr(SQLHDBC      ConnectionHandle
                        SQLINTEGER   Attribute,
                        SQLPOINTER   ValuePtr,
                        SQLINTEGER   StringLength);
```

参数

表 6-29. SQLSetConnectAttr 参数

关键字	参数说明
ConnectionHandle	连接句柄。
Attribute	设置属性。
ValuePtr	指向对应 Attribute 的值。依赖于 Attribute 的值，ValuePtr 是 32 位无符号整型值或指向以空结束的字符串。注意，如果 ValuePtr 参数是驱动程序指定值。ValuePtr 可能是有符号的整数。
StringLength	如果 ValuePtr 指向字符串或二进制缓冲区，这个参数是*ValuePtr 长度，如果 ValuePtr 指向整型，忽略 StringLength。

返回值

- SQL_SUCCESS：表示调用正确。
- SQL_SUCCESS_WITH_INFO：表示会有一些警告信息。
- SQL_ERROR：表示比较严重的错误，如：内存分配失败、建立连接失败等。
- SQL_INVALID_HANDLE：表示调用无效句柄。其他 API 的返回值同理。

注意事项

当 SQLSetConnectAttr 的返回值为 SQL_ERROR 或 SQL_SUCCESS_WITH_INFO 时，通过借助 SQL_HANDLE_DBC 的 HandleType 和 ConnectionHandle 的 Handle，调用 6.2.19SQLGetDiagRec 可得到相关的 SQLSTATE 值，通过 SQLSTATE 值可以查出调用此函数的具体信息。

示例

参见：6.2.23 示例

6.2.21. SQLSetEnvAttr

功能描述

设置控制环境各方面的属性。

原型

```
SQLRETURN SQLSetEnvAttr(SQLHENV      EnvironmentHandle
                    SQLINTEGER    Attribute,
                    SQLPOINTER    ValuePtr,
                    SQLINTEGER    StringLength);
```

参数

表 6-30. SQLSetEnvAttr 参数

关键字	参数说明
EnvironmentHandle	环境句柄。
Attribute	需设置的环境属性，可为如下值： • SQL_ATTR_ODBC_VERSION：指定 ODBC 版本。 • SQL_CONNECTION_POOLING：连接池属性。 • SQL_OUTPUT_NTS：指明驱动器返回字符串的形式。
ValuePtr	指向对应 Attribute 的值。依赖于 Attribute 的值，ValuePtr 可能是 32 位整型值，或为以空结束的字符串。
StringLength	如果 ValuePtr 指向字符串或二进制缓冲区，这个参数是*ValuePtr 长度，如果 ValuePtr 指向整型，忽略 StringLength。

返回值

- SQL_SUCCESS：表示调用正确。
- SQL_SUCCESS_WITH_INFO：表示会有一些警告信息。
- SQL_ERROR：表示比较严重的错误，如：内存分配失败、建立连接失败等。
- SQL_INVALID_HANDLE：表示调用无效句柄。其他 API 的返回值同理。

注意事项

当 SQLSetEnvAttr 的返回值为 SQL_ERROR 或 SQL_SUCCESS_WITH_INFO 时，通过借助 SQL_HANDLE_ENV 的 HandleType 和 EnvironmentHandle 的 Handle，调用 6.2.19SQLGetDiagRec 可得到相关的 SQLSTATE 值，通过 SQLSTATE 值可以查出调用此函数的具体信息。

示例

参见：6.2.23 示例

6.2.22. SQLSetStmtAttr

功能描述

设置相关语句的属性。

原型

```
SQLRETURN SQLSetStmtAttr(SQLHSTMT     StatementHandle
                    SQLINTEGER    Attribute,
                    SQLPOINTER    ValuePtr,
                    SQLINTEGER    StringLength);
```

参数

表 6-31. SQLSetStmtAttr 参数

关键字	参数说明
StatementHandle	语句句柄。
Attribute	需设置的属性。
ValuePtr	指向对应 Attribute 的值。依赖于 Attribute 的值，ValuePtr 可能是 32 位无符号整型值，或指向以空结束的字符串，二进制缓冲区，或者驱动定义值。注意，如果 ValuePtr 参数是驱动程序指定值。ValuePtr 可能是有符号的整数。
StringLength	如果 ValuePtr 指向字符串或二进制缓冲区，这个参数是*ValuePtr 长度，如果

关键字	参数说明
	ValuePtr 指向整型，忽略 StringLength。

返回值

- SQL_SUCCESS：表示调用正确。
- SQL_SUCCESS_WITH_INFO：表示会有一些警告信息。
- SQL_ERROR：表示比较严重的错误，如：内存分配失败、建立连接失败等。
- SQL_INVALID_HANDLE：表示调用无效句柄。其他 API 的返回值同理。

注意事项

当 SQLSetStmtAttr 的返回值为 SQL_ERROR 或 SQL_SUCCESS_WITH_INFO 时，通过借助 SQL_HANDLE_STMT 的 HandleType 和 StatementHandle 的 Handle，调用 6.2.19SQLGetDiagRec 可得到相关的 SQLSTATE 值，通过 SQLSTATE 值可以查出调用此函数的具体信息。

示例

参见：6.2.23 示例

6.2.23. 示例

常用功能示例代码

```
// 此示例演示如何通过 ODBC 方式获取 Vastbase 中的数据。
// DBtest.c (compile with: libodbc.so)
#include <stdlib.h>
#include <stdio.h>
#include <sqlext.h>
#ifdef WIN32
#include <windows.h>
#endif
SQLHENV      V_OD_Env;       // Handle ODBC environment
SQLHSTMT     V_OD_hstmt;     // Handle statement
SQLHDBC      V_OD_hdbc;      // Handle connection
char         typename[100];
SQLINTEGER   value = 100;
SQLINTEGER   V_OD_erg,V_OD_buffer,V_OD_err,V_OD_id;
int main(int argc,char *argv[])
{
    // 1. 申请环境句柄
    V_OD_erg = SQLAllocHandle(SQL_HANDLE_ENV,SQL_NULL_HANDLE,&V_OD_Env);
    if ((V_OD_erg != SQL_SUCCESS) && (V_OD_erg != SQL_SUCCESS_WITH_INFO))
    {
        printf("Error AllocHandle\n");
        exit(0);
```

```
    }
    // 2. 设置环境属性（版本信息）
    SQLSetEnvAttr(V_OD_Env, SQL_ATTR_ODBC_VERSION, (void*)SQL_OV_ODBC3, 0);
    // 3. 申请连接句柄
    V_OD_erg = SQLAllocHandle(SQL_HANDLE_DBC, V_OD_Env, &V_OD_hdbc);
    if ((V_OD_erg != SQL_SUCCESS) && (V_OD_erg != SQL_SUCCESS_WITH_INFO))
    {
        SQLFreeHandle(SQL_HANDLE_ENV, V_OD_Env);
        exit(0);
    }
    // 4. 设置连接属性
    SQLSetConnectAttr(V_OD_hdbc, SQL_ATTR_AUTOCOMMIT, SQL_AUTOCOMMIT_ON, 0);
    // 5. 连接数据源，这里的“userName”与“password”分别表示连接数据库的用户名和用户密码，请根据实际情况修改。
    // 如果 odbc.ini 文件中已经配置了用户名密码，那么这里可以留空（""）；但是不建议这么做，因为一旦 odbc.ini 权限
管理不善，将导致数据库用户密码泄露。
    V_OD_erg = SQLConnect(V_OD_hdbc, (SQLCHAR*) "vastbase", SQL_NTS,
                    (SQLCHAR*) "userName", SQL_NTS,  (SQLCHAR*) "password", SQL_NTS);
    if ((V_OD_erg != SQL_SUCCESS) && (V_OD_erg != SQL_SUCCESS_WITH_INFO))
    {
       printf("Error SQLConnect %d\n",V_OD_erg);
       SQLFreeHandle(SQL_HANDLE_ENV, V_OD_Env);
       exit(0);
    }
    printf("Connected !\n");
    // 6. 设置语句属性
    SQLSetStmtAttr(V_OD_hstmt,SQL_ATTR_QUERY_TIMEOUT,(SQLPOINTER *)3,0);
    // 7. 申请语句句柄
    SQLAllocHandle(SQL_HANDLE_STMT, V_OD_hdbc, &V_OD_hstmt);
    // 8. 直接执行 SQL 语句。
    SQLExecDirect(V_OD_hstmt,"drop table IF EXISTS customer_t1",SQL_NTS);
    SQLExecDirect(V_OD_hstmt,"CREATE TABLE customer_t1(c_customer_sk INTEGER, c_customer_name
VARCHAR(32));",SQL_NTS);
    SQLExecDirect(V_OD_hstmt,"insert into customer_t1 values(25,li)",SQL_NTS);
    // 9. 准备执行
    SQLPrepare(V_OD_hstmt,"insert into customer_t1 values(?)",SQL_NTS);
    // 10. 绑定参数
    SQLBindParameter(V_OD_hstmt,1,SQL_PARAM_INPUT,SQL_C_SLONG,SQL_INTEGER,0,0,
                 &value,0,NULL);
    // 11. 执行准备好的语句
    SQLExecute(V_OD_hstmt);
    SQLExecDirect(V_OD_hstmt,"select id from testtable",SQL_NTS);
    // 12. 获取结果集某一列的属性
    SQLColAttribute(V_OD_hstmt,1,SQL_DESC_TYPE,typename,100,NULL,NULL);
    printf("SQLColAtrribute %s\n",typename);
    // 13. 绑定结果集
    SQLBindCol(V_OD_hstmt,1,SQL_C_SLONG, (SQLPOINTER)&V_OD_buffer,150,
            (SQLLEN *)&V_OD_err);
    // 14. 通过 SQLFetch 取结果集中数据
    V_OD_erg=SQLFetch(V_OD_hstmt);
    // 15. 通过 SQLGetData 获取并返回数据。
    while(V_OD_erg != SQL_NO_DATA)
    {
       SQLGetData(V_OD_hstmt,1,SQL_C_SLONG,(SQLPOINTER)&V_OD_id,0,NULL);
```

```
        printf("SQLGetData ----ID = %d\n",V_OD_id);
        V_OD_erg=SQLFetch(V_OD_hstmt);
    };
    printf("Done !\n");
    // 16. 断开数据源连接并释放句柄资源
    SQLFreeHandle(SQL_HANDLE_STMT,V_OD_hstmt);
    SQLDisconnect(V_OD_hdbc);
    SQLFreeHandle(SQL_HANDLE_DBC,V_OD_hdbc);
    SQLFreeHandle(SQL_HANDLE_ENV, V_OD_Env);
    return(0);
}
```

批量绑定示例代码

```
/*********************************************************************
* 请在数据源中打开 UseBatchProtocol，同时指定数据库中参数 support_batch_bind
* 为 on
* CHECK_ERROR 的作用是检查并打印错误信息。
* 此示例将与用户交互式获取 DSN、模拟的数据量，忽略的数据量，并将最终数据入库到 test_odbc_batch_insert 中
*********************************************************************/
#include <stdio.h>
#include <stdlib.h>
#include <sql.h>
#include <sqlext.h>
#include <string.h>

#include "util.c"

void Exec(SQLHDBC hdbc, SQLCHAR* sql)
{
   SQLRETURN retcode;               // Return status
   SQLHSTMT hstmt = SQL_NULL_HSTMT;    // Statement handle
   SQLCHAR    loginfo[2048];

   // Allocate Statement Handle
   retcode = SQLAllocHandle(SQL_HANDLE_STMT, hdbc, &hstmt);
   CHECK_ERROR(retcode, "SQLAllocHandle(SQL_HANDLE_STMT)",
             hstmt, SQL_HANDLE_STMT);

   // Prepare Statement
   retcode = SQLPrepare(hstmt, (SQLCHAR*) sql, SQL_NTS);
   sprintf((char*)loginfo, "SQLPrepare log: %s", (char*)sql);
   CHECK_ERROR(retcode, loginfo, hstmt, SQL_HANDLE_STMT);

   // Execute Statement
   retcode = SQLExecute(hstmt);
   sprintf((char*)loginfo, "SQLExecute stmt log: %s", (char*)sql);
   CHECK_ERROR(retcode, loginfo, hstmt, SQL_HANDLE_STMT);

   // Free Handle
   retcode = SQLFreeHandle(SQL_HANDLE_STMT, hstmt);
   sprintf((char*)loginfo, "SQLFreeHandle stmt log: %s", (char*)sql);
   CHECK_ERROR(retcode, loginfo, hstmt, SQL_HANDLE_STMT);
}
```

```
int main ()
{
   SQLHENV  henv  = SQL_NULL_HENV;
   SQLHDBC  hdbc  = SQL_NULL_HDBC;
   int     batchCount = 1000;
   SQLLEN   rowsCount = 0;
   int     ignoreCount = 0;

   SQLRETURN   retcode;
   SQLCHAR    dsn[1024] = {'\0'};
   SQLCHAR    loginfo[2048];

   // 交互获取数据源名称
   getStr("Please input your DSN", (char*)dsn, sizeof(dsn), 'N');
   // 交互获取批量绑定的数据量
   getInt("batchCount", &batchCount, 'N', 1);
   do
   {
      // 交互获取批量绑定的数据中，不要入库的数据量
      getInt("ignoreCount", &ignoreCount, 'N', 1);
      if (ignoreCount > batchCount)
      {
         printf("ignoreCount(%d) should be less than batchCount(%d)\n", ignoreCount, batchCount);
      }
   }while(ignoreCount > batchCount);

   retcode = SQLAllocHandle(SQL_HANDLE_ENV, SQL_NULL_HANDLE, &henv);
   CHECK_ERROR(retcode, "SQLAllocHandle(SQL_HANDLE_ENV)",
            henv, SQL_HANDLE_ENV);

   // Set ODBC Verion
   retcode = SQLSetEnvAttr(henv, SQL_ATTR_ODBC_VERSION,
                           (SQLPOINTER*)SQL_OV_ODBC3, 0);
   CHECK_ERROR(retcode, "SQLSetEnvAttr(SQL_ATTR_ODBC_VERSION)",
            henv, SQL_HANDLE_ENV);

   // Allocate Connection
   retcode = SQLAllocHandle(SQL_HANDLE_DBC, henv, &hdbc);
   CHECK_ERROR(retcode, "SQLAllocHandle(SQL_HANDLE_DBC)",
            henv, SQL_HANDLE_DBC);

   // Set Login Timeout
   retcode = SQLSetConnectAttr(hdbc, SQL_LOGIN_TIMEOUT, (SQLPOINTER)5, 0);
   CHECK_ERROR(retcode, "SQLSetConnectAttr(SQL_LOGIN_TIMEOUT)",
            hdbc, SQL_HANDLE_DBC);

   // Set Auto Commit
   retcode = SQLSetConnectAttr(hdbc, SQL_ATTR_AUTOCOMMIT,
                           (SQLPOINTER)(1), 0);
   CHECK_ERROR(retcode, "SQLSetConnectAttr(SQL_ATTR_AUTOCOMMIT)",
            hdbc, SQL_HANDLE_DBC);

   // Connect to DSN
   sprintf(loginfo, "SQLConnect(DSN:%s)", dsn);
```

```
retcode = SQLConnect(hdbc, (SQLCHAR*) dsn, SQL_NTS,
                     (SQLCHAR*) NULL, 0, NULL, 0);
CHECK_ERROR(retcode, loginfo, hdbc, SQL_HANDLE_DBC);

// init table info.
Exec(hdbc, "drop table if exists test_odbc_batch_insert");
Exec(hdbc, "create table test_odbc_batch_insert(id int primary key, col varchar2(50))");

// 下面的代码根据用户输入的数据量，构造出将要入库的数据：
{
    SQLRETURN retcode;
    SQLHSTMT hstmtinesrt = SQL_NULL_HSTMT;
    int        i;
    SQLCHAR     *sql = NULL;
    SQLINTEGER   *ids  = NULL;
    SQLCHAR     *cols = NULL;
    SQLLEN      *bufLenIds = NULL;
    SQLLEN      *bufLenCols = NULL;
    SQLUSMALLINT *operptr = NULL;
    SQLUSMALLINT *statusptr = NULL;
    SQLULEN     process = 0;

    // 这里是按列构造，每个字段的内存连续存放在一起。
    ids = (SQLINTEGER*)malloc(sizeof(ids[0]) * batchCount);
    cols = (SQLCHAR*)malloc(sizeof(cols[0]) * batchCount * 50);
    // 这里是每个字段中，每一行数据的内存长度。
    bufLenIds = (SQLLEN*)malloc(sizeof(bufLenIds[0]) * batchCount);
    bufLenCols = (SQLLEN*)malloc(sizeof(bufLenCols[0]) * batchCount);
    // 该行是否需要被处理，SQL_PARAM_IGNORE 或 SQL_PARAM_PROCEED
    operptr = (SQLUSMALLINT*)malloc(sizeof(operptr[0]) * batchCount);
    memset(operptr, 0, sizeof(operptr[0]) * batchCount);
    // 该行的处理结果。
    // 注：由于数据库中处理方式是同一语句隶属同一事务中，所以如果出错，那么待处理数据都将是出错的，并不会部分入库。
    statusptr = (SQLUSMALLINT*)malloc(sizeof(statusptr[0]) * batchCount);
    memset(statusptr, 88, sizeof(statusptr[0]) * batchCount);

    if (NULL == ids || NULL == cols || NULL == bufLenCols || NULL == bufLenIds)
    {
        fprintf(stderr, "FAILED:\tmalloc data memory failed\n");
        goto exit;
    }

    for (int i = 0; i < batchCount; i++)
    {
        ids[i] = i;
        sprintf(cols + 50 * i, "column test value %d", i);
        bufLenIds[i] = sizeof(ids[i]);
        bufLenCols[i] = strlen(cols + 50 * i);
        operptr[i] = (i < ignoreCount) ? SQL_PARAM_IGNORE : SQL_PARAM_PROCEED;
    }

    // Allocate Statement Handle
    retcode = SQLAllocHandle(SQL_HANDLE_STMT, hdbc, &hstmtinesrt);
    CHECK_ERROR(retcode, "SQLAllocHandle(SQL_HANDLE_STMT)",
              hstmtinesrt, SQL_HANDLE_STMT);
```

```
      // Prepare Statement
      sql = (SQLCHAR*)"insert into test_odbc_batch_insert values(?, ?)";
      retcode = SQLPrepare(hstmtinesrt, (SQLCHAR*) sql, SQL_NTS);
      sprintf((char*)loginfo, "SQLPrepare log: %s", (char*)sql);
      CHECK_ERROR(retcode, loginfo, hstmtinesrt, SQL_HANDLE_STMT);

      retcode = SQLSetStmtAttr(hstmtinesrt, SQL_ATTR_PARAMSET_SIZE, (SQLPOINTER)batchCount,
sizeof(batchCount));
      CHECK_ERROR(retcode, "SQLSetStmtAttr", hstmtinesrt, SQL_HANDLE_STMT);

      retcode = SQLBindParameter(hstmtinesrt, 1, SQL_PARAM_INPUT, SQL_C_SLONG, SQL_INTEGER,
sizeof(ids[0]), 0,&(ids[0]), 0, bufLenIds);
      CHECK_ERROR(retcode, "SQLBindParameter for id", hstmtinesrt, SQL_HANDLE_STMT);

      retcode = SQLBindParameter(hstmtinesrt, 2, SQL_PARAM_INPUT, SQL_C_CHAR, SQL_CHAR, 50, 50, cols,
50, bufLenCols);
      CHECK_ERROR(retcode, "SQLBindParameter for cols", hstmtinesrt, SQL_HANDLE_STMT);

      retcode = SQLSetStmtAttr(hstmtinesrt, SQL_ATTR_PARAMS_PROCESSED_PTR, (SQLPOINTER)&process,
sizeof(process));
      CHECK_ERROR(retcode, "SQLSetStmtAttr for SQL_ATTR_PARAMS_PROCESSED_PTR", hstmtinesrt,
SQL_HANDLE_STMT);

      retcode = SQLSetStmtAttr(hstmtinesrt, SQL_ATTR_PARAM_STATUS_PTR, (SQLPOINTER)statusptr,
sizeof(statusptr[0]) * batchCount);
      CHECK_ERROR(retcode, "SQLSetStmtAttr for SQL_ATTR_PARAM_STATUS_PTR", hstmtinesrt,
SQL_HANDLE_STMT);

      retcode = SQLSetStmtAttr(hstmtinesrt, SQL_ATTR_PARAM_OPERATION_PTR, (SQLPOINTER)operptr,
sizeof(operptr[0]) * batchCount);
      CHECK_ERROR(retcode, "SQLSetStmtAttr for SQL_ATTR_PARAM_OPERATION_PTR", hstmtinesrt,
SQL_HANDLE_STMT);

      retcode = SQLExecute(hstmtinesrt);
      sprintf((char*)loginfo, "SQLExecute stmt log: %s", (char*)sql);
      CHECK_ERROR(retcode, loginfo, hstmtinesrt, SQL_HANDLE_STMT);

      retcode = SQLRowCount(hstmtinesrt, &rowsCount);
      CHECK_ERROR(retcode, "SQLRowCount execution", hstmtinesrt, SQL_HANDLE_STMT);

      if (rowsCount != (batchCount - ignoreCount))
      {
         sprintf(loginfo, "(batchCount - ignoreCount)(%d) != rowsCount(%d)", (batchCount -
ignoreCount), rowsCount);
         CHECK_ERROR(SQL_ERROR, loginfo, NULL, SQL_HANDLE_STMT);
      }
      else
      {
         sprintf(loginfo, "(batchCount - ignoreCount)(%d) == rowsCount(%d)", (batchCount -
ignoreCount), rowsCount);
         CHECK_ERROR(SQL_SUCCESS, loginfo, NULL, SQL_HANDLE_STMT);
      }

      // check row number returned
```

```
      if (rowsCount != process)
      {
         sprintf(loginfo, "process(%d) != rowsCount(%d)", process, rowsCount);
         CHECK_ERROR(SQL_ERROR, loginfo, NULL, SQL_HANDLE_STMT);
      }
      else
      {
         sprintf(loginfo, "process(%d) == rowsCount(%d)", process, rowsCount);
         CHECK_ERROR(SQL_SUCCESS, loginfo, NULL, SQL_HANDLE_STMT);
      }

      for (int i = 0; i < batchCount; i++)
      {
         if (i < ignoreCount)
         {
            if (statusptr[i] != SQL_PARAM_UNUSED)
            {
               sprintf(loginfo, "statusptr[%d](%d) != SQL_PARAM_UNUSED", i, statusptr[i]);
               CHECK_ERROR(SQL_ERROR, loginfo, NULL, SQL_HANDLE_STMT);
            }
         }
         else if (statusptr[i] != SQL_PARAM_SUCCESS)
         {
            sprintf(loginfo, "statusptr[%d](%d) != SQL_PARAM_SUCCESS", i, statusptr[i]);
            CHECK_ERROR(SQL_ERROR, loginfo, NULL, SQL_HANDLE_STMT);
         }
      }

      retcode = SQLFreeHandle(SQL_HANDLE_STMT, hstmtinesrt);
      sprintf((char*)loginfo, "SQLFreeHandle hstmtinesrt");
      CHECK_ERROR(retcode, loginfo, hstmtinesrt, SQL_HANDLE_STMT);
   }


exit:
   printf ("\nComplete.\n");

   // Connection
   if (hdbc != SQL_NULL_HDBC) {
      SQLDisconnect(hdbc);
      SQLFreeHandle(SQL_HANDLE_DBC, hdbc);
   }

   // Environment
   if (henv != SQL_NULL_HENV)
      SQLFreeHandle(SQL_HANDLE_ENV, henv);

   return 0;
}
```

6.3. libpq

6.3.1. 数据库连接控制函数

数据库连接控制函数控制与 vastbase 服务器链接的事情。一个应用程序一次可以与多个服务器建立链接，如一个客户端链接多个数据库的场景。每个链接都是用一个从函数 PQconnectdb、PQconnectdbParams 或 PQsetdbLogin 获得的 PGconn 对象表示。注意，这些函数总是返回一个非空的对象指针，除非内存分配失败，会返回一个空的指针。链接建立的接口保存在 PGconn 对象中，可以调用 PQstatus 函数来检查一下返回值看看连接是否成功。

6.3.1.1. PQconnectdbParams

功能描述

与数据库服务器建立一个新的连接。

原型

```
PGconn *PQconnectdbParams(const char * const *keywords,
                    const char * const *values,
                    int expand_dbname);
```

参数

表 6-32. PQconnectdbParams 参数

关键字	参数说明
keywords	定义为一个字符串的数组，每个都成为一个关键字。
values	给每个关键字一个值。
expand_dbname	当 expand_dbname 是非零的时，允许将 dbname 的关键字值看做一个连接字符串。只有第一个出现的 dbname 是这样展开的，任何随后的 dbname 值作为纯数据库名处理。

返回值

PGconn *：指向包含链接的对象指针，内存在函数内部申请。

注意事项

这个函数用从两个 NULL 结束的数组中来的参数打开一个新的数据库连接。与 PQsetdbLogin 不同的是，可以不必更换函数签名（名字）就可以扩展参数集，所以建议应用程序中使用这个函数（或者它的类似的非阻塞变种 PQconnectStartParams 和 PQconnectPoll）。

示例

请参见 6.3.5 示例章节。

6.3.1.2. PQconnectdb

功能描述

与数据库服务器建立一个新的连接。

原型

```
PGconn *PQconnectdb(const char *conninfo);
```

参数

表 6-33. PQconnectdb 参数

关键字	参数说明
conninfo	链接字符串，字符串中的字段见 6.3.6 链接字符章节。

返回值

PGconn *：指向包含链接的对象指针，内存在函数内部申请。

注意事项

- 这个函数用从一个字符串 conninfo 来的参数与数据库打开一个新的链接。
- 传入的参数可以为空，表明使用所有缺省的参数，或者可以包含一个或更多个用空白间隔的参数设置，或者它可以包含一个 URL。

示例

请参见 6.3.5 示例章节。

6.3.1.3. PQconninfoParse

功能描述

根据连接，返回已解析的连接选项。

原型

```
PQconninfoOption* PQconninfoParse(const char* conninfo, char** errmsg);
```

参数

表 3-1

关键字	参数说明
conninfo	被传递的字符串。可以为空，这样将会使用默认参数。也可以包含由空格分隔的一个或多个参数设置，还可以包含一个 URI。
errmsg	错误信息。

返回值

PQconninfoOption 类型指针。

6.3.1.4. PQconnectStart

功能描述

与数据库服务器建立一次非阻塞的连接。

原型

```
PGconn* PQconnectStart(const char* conninfo);
```

参数

表 3-2

关键字	参数说明
conninfo	连接信息字符串。可以为空，这样将会使用默认参数。也可以包含由空格分隔的一个或多个参数设置，还可以包含一个 URI。

返回值

PGconn 类型指针。

6.3.1.5. PQerrorMessage

功能描述

返回连接上的错误信息。

原型

```
char* PQerrorMessage(const PGconn* conn);
```

参数

表 3-3

关键字	参数说明
conn	连接句柄。

返回值

char 类型指针。

示例

参见：6.3.5 示例

6.3.1.6. PQsetdbLogin

功能描述

与数据库服务器建立一个新的链接。

原型

```
PGconn *PQsetdbLogin(const char *pghost,
                     const char *pgport,
                     const char *pgoptions,
                     const char *pgtty,
                     const char *dbName,
```

```
                const char *login,
                const char *pwd);
```

参数

表 6-34. PQsetdbLogin 参数

关键字	参数说明
pghost	要链接的主机名，详见 6.3.6 链接字符章节描述的 host 字段。
pgport	主机服务器的端口号，详见 6.3.6 链接字符描述的 port 字段。
pgoptions	添加命令行选项以在运行时发送到服务器，详见 6.3.6 链接字符描述的 options 字段。
pgtty	忽略（以前，这个选项声明服务器日志的输出方向）
dbName	要链接的数据库名，详见 6.3.6 链接字符描述的 dbname 字段。
login	要链接的用户名，详见 6.3.6 链接字符章节描述的 user 字段。
pwd	如果服务器要求口令认证，所用的口令，详见 6.3.6 链接字符描述的 password 字段。

返回值

PGconn *：指向包含链接的对象指针，内存在函数内部申请。

注意事项

- 该函数为 PQconnectdb 前身，参数个数固定，未定义参数被调用时使用缺省值，若需要给固定参数设置缺省值，则可赋值 NULL 或者空字符串。
- 若 dbName 中包含“=”或链接 URL 的有效前缀，则该 dbName 被看做一个 conninfo 字符串并传递至 PQconnectdb 中，其余参数与 PQconnectdbParams 保持一致。

示例

请参见 6.3.5 示例章节。

6.3.1.7. PQfinish

功能描述

关闭与服务器的连接，同时释放被 PGconn 对象使用的存储器。

原型

```
void PQfinish(PGconn *conn);
```

参数

表 6-35. PQfinish 参数

关键字	参数说明
conn	指向包含链接的对象指针。

注意事项

若 PQstatus 判断服务器链接尝试失败，应用程序调用 PQfinish 释放被 PGconn 对象使用的存储器，PQfinish 调用后 PGconn 指针不可再次使用。

示例

请参见 6.3.5 示例章节。

6.3.1.8. PQreset

功能描述

重置与服务器的通讯端口。

原型

```
void PQreset(PGconn *conn);
```

参数

表 6-36. PQreset 参数

关键字	参数说明
conn	指向包含链接的对象指针。

注意事项

此函数将关闭与服务器的连接并且试图与同一个服务器重建新的连接，并使用所有前面使用过的参数。该函数在链接异常后进行故障恢复时很有用。

示例

请参见 6.3.5 示例章节。

6.3.1.9. PQstatus

功能描述

返回链接的状态。

原型

```
ConnStatusType PQstatus(const PGconn *conn);
```

参数

表 6-37. PQstatus 参数

关键字	参数说明
conn	指向包含链接的对象指针。

返回值

ConnStatusType：链接状态的枚举，包括：

```
CONNECTION_STARTED
等待进行连接。

CONNECTION_MADE
连接成功；等待发送。

CONNECTION_AWAITING_RESPONSE
等待来自服务器的响应。

CONNECTION_AUTH_OK
已收到认证；等待后端启动结束。

CONNECTION_SSL_STARTUP
协商 SSL 加密。

CONNECTION_SETENV
```

```
协商环境驱动的参数设置。

CONNECTION_OK
链接正常。

CONNECTION_BAD
链接故障。
```

注意事项

状态可以是多个值之一。但是，在异步连接过程之外只能看到其中两个：CONNECTION_OK 和 CONNECTION_BAD。与数据库的良好连接状态为 CONNECTION_OK。状态表示连接尝试失败 CONNECTION_BAD。通常，“正常”状态将一直保持到 PQfinish，但通信失败可能会导致状态 CONNECTION_BAD 过早变为。在这种情况下，应用程序可以尝试通过调用进行恢复 PQreset。

示例

请参见 6.3.5 示例章节。

6.3.2. 数据库执行语句函数

与数据库服务器的连接成功建立，便可以使用这里描述的函数执行 SQL 查询和命令。

6.3.2.1. PQclear

功能描述

释放与 PGresult 相关联的存储空间，任何不再需要的查询结果都应该用 PQclear 释放掉。

原型

```
void PQclear(PGresult *res);
```

参数

表 6-38. PQclear 参数

关键字	参数说明
res	包含查询结果的对象指针。

注意事项

PGresult 不会自动释放，当提交新的查询时它并不消失，甚至断开连接后也不会。要删除它，必须调用 PQclear，否则则会有内存泄漏。

示例

请参见 6.3.5 示例章节。

6.3.2.2. PQexec

功能描述

向服务器提交一条命令并等待结果。

原型

```
PGresult *PQexec(PGconn *conn, const char *command);
```

参数

表 6-39. PQexec 参数

关键字	参数说明
conn	指向包含链接的对象指针。
command	需要执行的查询字符串。

返回值

PGresult：包含查询结果的对象指针。

注意事项

应该调用 PQresultStatus 函数来检查任何错误的返回值（包括空指针的值，在这种情况下它将返回 PGRES_FATAL_ERROR）。使用 PQerrorMessage 获取有关错误的更多信息。

须知

命令字符串可以包括多个 SQL 命令（用分号分隔）。在一个 PQexec 调用中发送的多个查询是在一个事务里处理的，除非在查询字符串里有明确的 BEGIN/COMMIT 命令把整个字符串分隔成多个事务。请注意，返回的 PGresult 结构只描述字符串里执行的最后一条命令的结果，如果有一个命令失败，那么字符串处理的过程就会停止，并且返回的 PGresult 会描述错误条件。

示例

请参见 6.3.5 示例章节。

6.3.2.3. PQexecParams

功能描述

执行一个绑定参数的命令。

原型

```
PGresult* PQexecParams(PGconn* conn,
                  const char* command,
                  int nParams,
                  const Oid* paramTypes,
                  const char* const* paramValues,
                  const int* paramLengths,
                  const int* paramFormats,
                  int resultFormat);
```

参数

表 3-4

关键字	参数说明
conn	连接句柄。
command	SQL 文本串。
nParams	绑定参数的个数
paramTypes	绑定参数类型。
paramValues	绑定参数的值。
paramLengths	参数长度。
paramFormats	参数格式（文本或二进制）。

关键字	参数说明
resultFormat	返回结果格式（文本或二进制）。

返回值

PGresult 类型指针。

6.3.2.4. PQexecParamsBatch

功能描述

执行一个批量绑定参数的命令。

原型

```
PGresult* PQexecParamsBatch(PGconn* conn,
                        const char* command,
                        int nParams,
                        int nBatch,
                        const Oid* paramTypes,
                        const char* const* paramValues,
                        const int* paramLengths,
                        const int* paramFormats,
                        int resultFormat);
```

参数

表 3-5

关键字	参数说明
conn	连接句柄。
command	SQL 文本串。
nParams	绑定参数的个数
nBatch	批量操作数。
paramTypes	绑定参数类型。
paramValues	绑定参数的值。
paramLengths	参数长度。

关键字	参数说明
paramFormats	参数格式（文本或二进制）。
resultFormat	返回结果格式（文本或二进制）。

返回值

PGresult 类型指针。

6.3.2.5. PQexecPrepared

功能描述

发送一个请求来用给定参数执行一个预备语句，并且等待结果。

原型

```
PGresult* PQexecPrepared(PGconn* conn,
                    const char* stmtName,
                    int nParams,
                    const char* const* paramValues,
                    const int* paramLengths,
                    const int* paramFormats,
                    int resultFormat);
```

参数

表 3-6

关键字	参数说明
conn	连接句柄。
stmtName	stmt 名称，可以用""或者 NULL 来引用未命名语句，否则它必须是一个现有预备语句的名字。
nParams	参数个数。
paramValues	参数的实际值。
paramLengths	参数的实际数据长度。
paramFormats	参数的格式（文本或二进制）。
resultFormat	结果的格式（文本或二进制）。

返回值

PGresult 类型指针。

6.3.2.6. PQexecPreparedBatch

功能描述

发送一个请求来用给定的批量参数执行一个预备语句，并且等待结果。

原型

```
PGresult* PQexecPreparedBatch(PGconn* conn,
                        const char* stmtName,
                        int nParams,
                        int nBatchCount,
                        const char* const* paramValues,
                        const int* paramLengths,
                        const int* paramFormats,
                        int resultFormat);
```

参数

表 3-7

关键字	参数说明
conn	连接句柄。
stmtName	stmt 名称，可以用""或者 NULL 来引用未命名语句，否则它必须是一个现有预备语句的名字。
nParams	参数个数。
nBatchCount	批量数。
paramValues	参数的实际值。
paramLengths	参数的实际数据长度。
paramFormats	参数的格式（文本或二进制）。
resultFormat	结果的格式（文本或二进制）。

返回值

PGresult 类型指针。

6.3.2.7. PQfname

功能描述

返回与给定列号相关联的列名。列号从 0 开始。调用者不应该直接释放该结果。它将在相关的 PGresult 句柄被传递给 PQclear 之后被释放。

原型

```
char *PQfname(const PGresult *res,
              int column_number);
```

参数

表 3-8

关键字	参数说明
res	操作结果句柄。
column_number	列数。

返回值

char 类型指针。

示例

参见：6.3.5 示例

6.3.2.8. PQgetvalue

功能描述

返回一个 PGresult 的一行的单一域值。行和列号从 0 开始。调用者不应该直接释放该结果。它将在相关的 PGresult 句柄被传递给 PQclear 之后被释放。

原型

```
char *PQgetvalue(const PGresult *res,
                 int row_number,
                 int column_number);
```

参数

表 3-9

关键字	参数说明
res	操作结果句柄。
row_number	行数。
column_number	列数。

返回值

对于文本格式的数据，PQgetvalue 返回的值是该域值的一种空值结束的字符串表示。
对于二进制格式的数据，该值是由该数据类型的 typsend 和 typreceive 函数决定的二进制表示。
如果该域值为空，则返回一个空串。

示例

参见：6.3.5 示例

6.3.2.9. PQnfields

功能描述

返回查询结果中每一行的列（域）数。

原型

```
int PQnfields(const PGresult *res);
```

参数

表 3-10

关键字	参数说明
res	操作结果句柄。

返回值

int 类型数字。

示例

参见：6.3.5 示例

6.3.2.10. PQntuples

功能描述

返回查询结果中的行（元组）数。因为它返回一个整数结果，在 32 位操作系统上大型的结果集可能使返回值溢出。

原型

```
int PQntuples(const PGresult *res);
```

参数

表 3-11

关键字	参数说明
res	操作结果句柄。

返回值

int 类型数字

示例

参见：6.3.5 示例

6.3.2.11. PQprepare

功能描述

用给定的参数提交请求，创建一个预备语句，然后等待结束。

原型

```
PGresult *PQprepare(PGconn *conn,
                const char *stmtName,
                const char *query,
                int nParams,
                const Oid *paramTypes);
```

参数

表 6-40. PQprepare 参数

关键字	参数说明
conn	指向包含链接的对象指针。
stmtName	需要执行的 stmt 名称。
query	需要执行的查询字符串。
nParams	参数个数。
paramTypes	声明参数类型的数组。

返回值

PGresult：包含查询结果的对象指针。

注意事项

- PQprepare 创建一个为 PQexecPrepared 执行用的预备语句，本特性支持命令的重复执行，不需要每次都进行解析和规划。PQprepare 仅在协议 3.0 及以后的连接中支持，使用协议 2.0 时，PQprepare 将失败。
- 该函数从查询字符串创建一个名为 stmtName 的预备语句，该查询字符串必须包含一个 SQL 命令。stmtName 可以是""来创建一个未命名的语句，在这种情况下，任何预先存在的未命名的语句都将被自动替换；否则，如果在当前会话中已经定义了语句名称，则这是一个错误。如果使用了任何参数，那么在查询中将它们称为$1,$2 等。nParams 是在 paramTypes[]数组中预先指定类型的参数的数量。（当 nParams 为 0 时，数组指针可以为 NULL） paramTypes[]通过 OID 指定要分配给参数符号的数据类型。如果 paramTypes 为 NULL ，或者数组中的任何特定元素为零，服务器将按照对非类型化字面字符串的相同方式为参数符号分配数据类型。另外，查询可以使用数字高于 nParams 的参数符号；还将推断这些符号的数据类型。

须知

通过执行 SQLPREPARE 语句，还可以创建与 PQexecPrepared 一起使用的预备语句。此外，虽然没有用于删除预备语句的 libpq 函数，但是 SQL DEALLOCATE 语句可用于此目的。

示例

请参见 6.3.5 示例章节。

6.3.2.12. PQresultStatus

功能描述

返回命令的结果状态。

原型

```
ExecStatusType PQresultStatus(const PGresult *res);
```

参数

表 6-41. PQresultStatus 参数

关键字	参数说明
res	包含查询结果的对象指针。

返回值

PQresultStatus：命令执行结果的枚举，包括：

```
PQresultStatus 可以返回下面数值之一：
PGRES_EMPTY_QUERY
发送给服务器的字串是空的。

PGRES_COMMAND_OK
成功完成一个不返回数据的命令。

PGRES_TUPLES_OK
成功执行一个返回数据的查询（比如 SELECT 或者 SHOW）。

PGRES_COPY_OUT
（从服务器）Copy Out （拷贝出）数据传输开始。

PGRES_COPY_IN
Copy In（拷贝入）（到服务器）数据传输开始。
```

```
PGRES_BAD_RESPONSE
服务器的响应无法理解。

PGRES_NONFATAL_ERROR
发生了一个非致命错误（通知或者警告）。

PGRES_FATAL_ERROR
发生了一个致命错误。

PGRES_COPY_BOTH
拷贝入/出（到和从服务器）数据传输开始。这个特性当前只用于流复制， 所以这个状态不会在普通应用中发生。

PGRES_SINGLE_TUPLE
PGresult 包含一个来自当前命令的结果元组。 这个状态只在查询选择了单行模式时发生
```

注意事项

- 请注意，恰好检索到零行的 SELECT 命令仍然显示 PGRES_TUPLES_OK。PGRES_COMMAND_OK 用于永远不能返回行的命令（插入或更新，不带返回子句等）。PGRES_EMPTY_QUERY 响应可能表明客户端软件存在 bug。
- 状态为 PGRES_NONFATAL_ERROR 的结果永远不会由 PQexec 或其他查询执行函数直接返回，此类结果将传递给通知处理程序。

示例

请参见 6.3.5 示例章节。

6.3.3. 异步命令处理

PQexec 函数对普通的同步应用里提交命令已经足够使用。但是它却有几个缺陷，而这些缺陷可能对某些用户很重要：

- PQexec 等待命令结束，而应用可能还有其它的工作要做（比如维护用户界面等），此时 PQexec 可不想阻塞在这里等待响应。
- 因为客户端应用在等待结果的时候是处于挂起状态的，所以应用很难判断它是否该尝试结束正在进行的命令。
- PQexec 只能返回一个 PGresult 结构。如果提交的命令字符串包含多个 SQL 命令，除了最后一个 PGresult 以外都会被 PQexec 丢弃。
- PQexec 总是收集命令的整个结果，将其缓存在一个 PGresult 中。虽然这为应用简化了错误处理逻辑，但是对于包含多行的结果是不切实际的。

不想受到这些限制的应用可以改用下面的函数，这些函数也是构造 PQexec 的函数：PQsendQuery 和 PQgetResult。PQsendQueryParams，PQsendPrepare，PQsendQueryPrepared 也可以和 PQgetResult 一起使用。

6.3.3.1. PQsendQuery

功能描述

向服务器提交一个命令而不等待结果。如果查询成功发送则返回 1，否则返回 0。

原型

```
int PQsendQuery(PGconn *conn, const char *command);
```

参数

表 6-42. PQsendQuery 参数

关键字	参数说明
conn	指向包含链接的对象指针。
command	需要执行的查询字符串.

返回值

int：执行结果为 1 表示成功，0 表示失败，失败原因存到 conn->errorMessage 中。

注意事项

在成功调用 PQsendQuery 后，调用 PQgetResult 一次或者多次获取结果。PQgetResult 返回空指针表示命令已执行完成，否则不能再次调用 PQsendQuery（在同一连接上）。

示例

请参见 6.3.5 示例章节。

6.3.3.2. PQsendQueryParams

功能描述

给服务器提交一个命令和分隔的参数，而不等待结果。

原型

```
int PQsendQueryParams(PGconn *conn,
                      const char *command,
```

```
                int nParams,
                const Oid *paramTypes,
                const char * const *paramValues,
                const int *paramLengths,
                const int *paramFormats,
                int resultFormat);
```

参数

表 6-43. PQsendQueryParams 参数

关键字	参数说明
conn	指向包含链接的对象指针。
command	需要执行的查询字符串。
nParams	参数个数。
paramTypes	参数类型。
paramValues	参数值。
paramLengths	参数长度。
paramFormats	参数格式。
resultFormat	结果的格式。

返回值

int：执行结果为 1 表示成功，0 表示失败，失败原因存到 conn->errorMessage 中。

注意事项

该函数等效于 PQsendQuery，只是查询参数可以和查询字符串分开声明。函数的参数处理和 PQexecParams 一样，和 PQexecParams 类似，它不能在 2.0 版本的协议连接上工作，并且它只允许在查询字符串里出现一条命令。

示例

请参见 6.3.5 示例章节。

6.3.3.3. PQsendPrepare

功能描述

发送一个请求，创建一个给定参数的预备语句，而不等待结束。

原型

```
int PQsendPrepare(PGconn *conn,
                  const char *stmtName,
                  const char *query,
                  int nParams,
                  const Oid *paramTypes);
```

参数

表 6-44. PQsendPrepare 参数

关键字	参数说明
conn	指向包含链接的对象指针。
stmtName	需要执行的 stmt 名称。
query	需要执行的查询字符串。
nParams	参数个数。
paramTypes	声明参数类型的数组。

返回值

int：执行结果为 1 表示成功，0 表示失败，失败原因存到 conn->errorMessage 中。

注意事项

该函数为 PQprepare 的异步版本：如果能够分派请求，则返回 1，否则返回 0。调用成功后，调用 PQgetResult 判断服务端是否成功创建了 preparedStatement。函数的参数与 PQprepare 一样处理。与 PQprepare 一样，它也不能在 2.0 协议的连接上工作。

示例

请参见 6.3.5 示例章节。

6.3.3.4. PQsendQueryPrepared

功能描述

发送一个请求执行带有给出参数的预备语句，不等待结果。

原型

```
int PQsendQueryPrepared(PGconn *conn,
                    const char *stmtName,
                    int nParams,
                    const char * const *paramValues,
                    const int *paramLengths,
                    const int *paramFormats,
                    int resultFormat);
```

参数

表 6-45. PQsendQueryPrepared 参数

关键字	参数说明
conn	指向包含链接信息的对象指针。
stmtName	需要执行的 stmt 名称。
nParams	参数个数。
paramValues	参数值。
paramLengths	参数长度。
paramFormats	参数格式。
resultFormat	结果的格式。

返回值

int：执行结果为 1 表示成功，0 表示失败，失败原因存到 conn->error_message 中。

注意事项

该函数类似于 PQsendQueryParams，但是要执行的命令是通过命名一个预先准备的语句来指定的，而不是提供一个查询字符串。该函数的参数与 PQexecPrepared 一样处理。和 PQexecPrepared 一样，它也不能在 2.0 协议的连接上工作。

示例

请参见 6.3.5 示例章节。

6.3.3.5. PQflush

功能描述

尝试将任何排队的输出数据刷新到服务器。

原型

```
int PQflush(PGconn *conn);
```

参数

表 6-46. PQflush 参数

关键字	参数说明
conn	指向包含链接信息的对象指针

返回值

int：如果成功（或者如果发送队列为空），则返回 0；如果由于某种原因失败，则返回-1；如果发送队列中的所有数据都发送失败，则返回 1。（此情况只有在连接为非阻塞时才能发生），失败原因存到 conn->error_message 中。

注意事项

在非阻塞连接上发送任何命令或数据之后，调用 PQflush。如果返回 1，则等待套接字变为读或写就绪。如果为写就绪状态，则再次调用 PQflush。如果已经读到，调用 PQconsumeInput，然后再次调用 PQflush。重复，直到 PQflush 返回 0。(必须检查读就绪，并用 PQconsumeInput 排出输入，因为服务器可以阻止试图向我们发送数据，例如。通知信息，直到我们读完它才会读我们的数据。)一旦 PQflush 返回 0，就等待套接字准备好，然后按照上面描述读取响应。

示例

请参见 6.3.5 示例章节。

6.3.4. 取消正在处理的查询

客户端应用可以使用本节描述的函数，要求取消一个仍在被服务器处理的命令。

6.3.4.1. PQgetCancel

功能描述

创建一个数据结构，其中包含取消通过特定数据库连接发出的命令所需的信息。

原型

```
PGcancel *PQgetCancel(PGconn *conn);
```

参数

表 6-47. PQgetCancel 参数

关键字	参数说明
conn	指向包含链接信息的对象指针。

返回值

PGcancel：指向包含 cancel 信息对象的指针。

注意事项

PQgetCancel 创建一个给定 PGconn 连接对象的 PGcancel 对象。如果给定的 conn 是 NULL 或无效连接，它将返回 NULL。PGcancel 对象是一个不透明的结构，应用程序不能直接访问它；它只能传递给 PQcancel 或 PQfreeCancel。

示例

请参见 6.3.5 示例章节。

6.3.4.2. PQfreeCancel

功能描述

释放 PQgetCancel 创建的数据结构。

原型

```
void PQfreeCancel(PGcancel *cancel);
```

参数

表 6-48. PQfreeCancel 参数

关键字	参数说明
cancel	指向包含 cancel 信息的对象指针。

注意事项

PQfreeCancel 释放一个由前面的 PQgetCancel 创建的数据对象。

示例

请参见 6.3.5 示例章节。

6.3.4.3. PQcancel

功能描述

要求服务器放弃处理当前命令。

原型

```
int PQcancel(PGcancel *cancel, char *errbuf, int errbufsize);
```

参数

表 6-49. PQcancel 参数

关键字	参数说明
cancel	指向包含 cancel 信息的对象指针。
errbuf	出错保存错误信息的 buffer。
errbufsize	保存错误信息的 buffer 大小。

返回值

int：执行结果为 1 表示成功，0 表示失败，失败原因存到 errbuf 中。

注意事项

- 成功发送并不保证请求将产生任何效果。如果取消有效，当前命令将提前终止并返回错误结果。如果取消失败（例如，因为服务器已经处理完命令），无返回结果。
- 如果 errbuf 是信号处理程序中的局部变量，则可以安全地从信号处理程序中调用 PQcancel。就 PQcancel 而言，PGcancel 对象是只读的，因此它也可以从一个线程中调用，这个线程与操作 PGconn 对象线程是分离的。

示例

请参见 6.3.5 示例章节。

6.3.5. 示例

常用功能示例代码 1

```
// 此示例演示如何通过 ODBC 方式获取 Vastbase 中的数据。
// DBtest.c (compile with: libodbc.so)
#include <stdlib.h>
#include <stdio.h>
#include <sqlext.h>
#ifdef WIN32
#include <windows.h>
#endif
SQLHENV      V_OD_Env;       // Handle ODBC environment
SQLHSTMT     V_OD_hstmt;     // Handle statement
SQLHDBC      V_OD_hdbc;      // Handle connection
char         typename[100];
SQLINTEGER   value = 100;
SQLINTEGER   V_OD_erg,V_OD_buffer,V_OD_err,V_OD_id;
int main(int argc,char *argv[])
{
    // 1. 申请环境句柄
    V_OD_erg = SQLAllocHandle(SQL_HANDLE_ENV,SQL_NULL_HANDLE,&V_OD_Env);
    if ((V_OD_erg != SQL_SUCCESS) && (V_OD_erg != SQL_SUCCESS_WITH_INFO))
    {
        printf("Error AllocHandle\n");
        exit(0);
    }
    // 2. 设置环境属性（版本信息）
    SQLSetEnvAttr(V_OD_Env, SQL_ATTR_ODBC_VERSION, (void*)SQL_OV_ODBC3, 0);
    // 3. 申请连接句柄
    V_OD_erg = SQLAllocHandle(SQL_HANDLE_DBC, V_OD_Env, &V_OD_hdbc);
    if ((V_OD_erg != SQL_SUCCESS) && (V_OD_erg != SQL_SUCCESS_WITH_INFO))
    {
        SQLFreeHandle(SQL_HANDLE_ENV, V_OD_Env);
        exit(0);
    }
    // 4. 设置连接属性
    SQLSetConnectAttr(V_OD_hdbc, SQL_ATTR_AUTOCOMMIT, SQL_AUTOCOMMIT_ON, 0);
```

```
    // 5. 连接数据源，这里的“userName”与“password”分别表示连接数据库的用户名和用户密码，请根据实际情况修改。
    // 如果 odbc.ini 文件中已经配置了用户名密码，那么这里可以留空（""）；但是不建议这么做，因为一旦 odbc.ini 权限
管理不善，将导致数据库用户密码泄露。
    V_OD_erg = SQLConnect(V_OD_hdbc, (SQLCHAR*) "vastbase", SQL_NTS,
                    (SQLCHAR*) "userName", SQL_NTS,  (SQLCHAR*) "password", SQL_NTS);
    if ((V_OD_erg != SQL_SUCCESS) && (V_OD_erg != SQL_SUCCESS_WITH_INFO))
    {
       printf("Error SQLConnect %d\n",V_OD_erg);
       SQLFreeHandle(SQL_HANDLE_ENV, V_OD_Env);
       exit(0);
    }
    printf("Connected !\n");
    // 6. 设置语句属性
    SQLSetStmtAttr(V_OD_hstmt,SQL_ATTR_QUERY_TIMEOUT,(SQLPOINTER *)3,0);
    // 7. 申请语句句柄
    SQLAllocHandle(SQL_HANDLE_STMT, V_OD_hdbc, &V_OD_hstmt);
    // 8. 直接执行 SQL 语句。
    SQLExecDirect(V_OD_hstmt,"drop table IF EXISTS customer_t1",SQL_NTS);
    SQLExecDirect(V_OD_hstmt,"CREATE TABLE customer_t1(c_customer_sk INTEGER, c_customer_name
VARCHAR(32));",SQL_NTS);
    SQLExecDirect(V_OD_hstmt,"insert into customer_t1 values(25,li)",SQL_NTS);
    // 9. 准备执行
    SQLPrepare(V_OD_hstmt,"insert into customer_t1 values(?)",SQL_NTS);
    // 10. 绑定参数
    SQLBindParameter(V_OD_hstmt,1,SQL_PARAM_INPUT,SQL_C_SLONG,SQL_INTEGER,0,0,
                  &value,0,NULL);
    // 11. 执行准备好的语句
    SQLExecute(V_OD_hstmt);
    SQLExecDirect(V_OD_hstmt,"select id from testtable",SQL_NTS);
    // 12. 获取结果集某一列的属性
    SQLColAttribute(V_OD_hstmt,1,SQL_DESC_TYPE,typename,100,NULL,NULL);
    printf("SQLColAtrribute %s\n",typename);
    // 13. 绑定结果集
    SQLBindCol(V_OD_hstmt,1,SQL_C_SLONG, (SQLPOINTER)&V_OD_buffer,150,
            (SQLLEN *)&V_OD_err);
    // 14. 通过 SQLFetch 取结果集中数据
    V_OD_erg=SQLFetch(V_OD_hstmt);
    // 15. 通过 SQLGetData 获取并返回数据。
    while(V_OD_erg != SQL_NO_DATA)
    {
       SQLGetData(V_OD_hstmt,1,SQL_C_SLONG,(SQLPOINTER)&V_OD_id,0,NULL);
       printf("SQLGetData ----ID = %d\n",V_OD_id);
       V_OD_erg=SQLFetch(V_OD_hstmt);
    };
    printf("Done !\n");
    // 16. 断开数据源连接并释放句柄资源
    SQLFreeHandle(SQL_HANDLE_STMT,V_OD_hstmt);
    SQLDisconnect(V_OD_hdbc);
    SQLFreeHandle(SQL_HANDLE_DBC,V_OD_hdbc);
    SQLFreeHandle(SQL_HANDLE_ENV, V_OD_Env);
    return(0);
}
```

常用功能示例代码 2

```
#include <stdio.h>
#include <stdlib.h>
#include <libpq-fe.h>

static void exit_nicely(PGconn *conn)
{
    PQfinish(conn);
    exit(1);
}

int main(int argc, char **argv)
{
    const char *conninfo;
    PGconn     *conn;
    PGresult   *res;
    int         nFields;
    int         i,j;

    /*
     * 如果用户在命令行上提供了一个参数，将它用作连接信息串。
     * 否则默认用设置 dbname=postgres 并且为所有其他链接参数使用环境变量或默认值。
     */
    if (argc > 1)
        conninfo = argv[1];
    else
        conninfo = "dbname = postgres";

    /* 建立到数据库的一个连接 */
    conn = PQconnectdb(conninfo);

    /* 检查看后端连接是否成功建立 */
    if (PQstatus(conn) != CONNECTION_OK)
    {
        fprintf(stderr, "Connection to database failed: %s",
                PQerrorMessage(conn));
        exit_nicely(conn);
    }

    /*
     * 我们的测试案例这里涉及使用一个游标，对它我们必须用在一个事务块内。
     * 我们可以在一个单一的 "select * from pg_database" 的 PQexec() 中做整个事情，
     * 但是作为一个好的例子它太琐碎。
     */

    /* 开始一个事务块 */
    res = PQexec(conn, "BEGIN");
    if (PQresultStatus(res) != PGRES_COMMAND_OK)
    {
        fprintf(stderr, "BEGIN command failed: %s", PQerrorMessage(conn));
        PQclear(res);
        exit_nicely(conn);
    }
```

```
    /* 任何时候不再需要 PGresult 时，应该 PQclear 它来避免内存泄露 */
    PQclear(res);

    /* 从 pg_database 取得行，它是数据库的系统目录 */
    res = PQexec(conn, "DECLARE myportal CURSOR FOR select * from pg_database");
    if (PQresultStatus(res) != PGRES_COMMAND_OK)
    {
        fprintf(stderr, "DECLARE CURSOR failed: %s", PQerrorMessage(conn));
        PQclear(res);
        exit_nicely(conn);
    }
    PQclear(res);

    res = PQexec(conn, "FETCH ALL in myportal");
    if (PQresultStatus(res) != PGRES_TUPLES_OK)
    {
        fprintf(stderr, "FETCH ALL failed: %s", PQerrorMessage(conn));
        PQclear(res);
        exit_nicely(conn);
    }

    /* 首先，打印出属性名 */
    nFields = PQnfields(res);
    for (i = 0; i < nFields; i++)
        printf("%-15s", PQfname(res, i));
    printf("\n\n");

    /* 接下来，打印出行 */
    for (i = 0; i < PQntuples(res); i++)
    {
        for (j = 0; j < nFields; j++)
            printf("%-15s", PQgetvalue(res, i, j));
        printf("\n");
    }

    PQclear(res);

    /* 关闭入口，我们不需要考虑检查错误 */
    res = PQexec(conn, "CLOSE myportal");
    PQclear(res);

    /* 结束事务 */
    res = PQexec(conn, "END");
    PQclear(res);

    /* 关闭到数据库的连接并且清理 */
    PQfinish(conn);

    return 0;
}
```

批量绑定示例代码

```
/*********************************************************************
* 请在数据源中打开 UseBatchProtocol，同时指定数据库中参数 support_batch_bind
* 为 on
* CHECK_ERROR 的作用是检查并打印错误信息。
* 此示例将与用户交互式获取 DSN、模拟的数据量，忽略的数据量，并将最终数据入库到 test_odbc_batch_insert 中
*********************************************************************/
#include <stdio.h>
#include <stdlib.h>
#include <sql.h>
#include <sqlext.h>
#include <string.h>

#include "util.c"

void Exec(SQLHDBC hdbc, SQLCHAR* sql)
{
   SQLRETURN retcode;               // Return status
   SQLHSTMT hstmt = SQL_NULL_HSTMT;    // Statement handle
   SQLCHAR    loginfo[2048];

   // Allocate Statement Handle
   retcode = SQLAllocHandle(SQL_HANDLE_STMT, hdbc, &hstmt);
   CHECK_ERROR(retcode, "SQLAllocHandle(SQL_HANDLE_STMT)",
             hstmt, SQL_HANDLE_STMT);

   // Prepare Statement
   retcode = SQLPrepare(hstmt, (SQLCHAR*) sql, SQL_NTS);
   sprintf((char*)loginfo, "SQLPrepare log: %s", (char*)sql);
   CHECK_ERROR(retcode, loginfo, hstmt, SQL_HANDLE_STMT);

   // Execute Statement
 retcode = SQLExecute(hstmt);
   sprintf((char*)loginfo, "SQLExecute stmt log: %s", (char*)sql);
   CHECK_ERROR(retcode, loginfo, hstmt, SQL_HANDLE_STMT);

   // Free Handle
retcode = SQLFreeHandle(SQL_HANDLE_STMT, hstmt);
   sprintf((char*)loginfo, "SQLFreeHandle stmt log: %s", (char*)sql);
   CHECK_ERROR(retcode, loginfo, hstmt, SQL_HANDLE_STMT);
}

int main ()
{
   SQLHENV  henv  = SQL_NULL_HENV;
   SQLHDBC  hdbc  = SQL_NULL_HDBC;
   int     batchCount = 1000;
   SQLLEN   rowsCount = 0;
   int     ignoreCount = 0;

   SQLRETURN   retcode;
   SQLCHAR    dsn[1024] = {'\0'};
   SQLCHAR    loginfo[2048];
```

```
// 交互获取数据源名称
getStr("Please input your DSN", (char*)dsn, sizeof(dsn), 'N');
// 交互获取批量绑定的数据量
getInt("batchCount", &batchCount, 'N', 1);
do
{
    // 交互获取批量绑定的数据中，不要入库的数据量
    getInt("ignoreCount", &ignoreCount, 'N', 1);
    if (ignoreCount > batchCount)
    {
       printf("ignoreCount(%d) should be less than batchCount(%d)\n", ignoreCount, batchCount);
    }
}while(ignoreCount > batchCount);

retcode = SQLAllocHandle(SQL_HANDLE_ENV, SQL_NULL_HANDLE, &henv);
CHECK_ERROR(retcode, "SQLAllocHandle(SQL_HANDLE_ENV)",
          henv, SQL_HANDLE_ENV);

// Set ODBC Verion
retcode = SQLSetEnvAttr(henv, SQL_ATTR_ODBC_VERSION,
                            (SQLPOINTER*)SQL_OV_ODBC3, 0);
CHECK_ERROR(retcode, "SQLSetEnvAttr(SQL_ATTR_ODBC_VERSION)",
          henv, SQL_HANDLE_ENV);

// Allocate Connection
retcode = SQLAllocHandle(SQL_HANDLE_DBC, henv, &hdbc);
CHECK_ERROR(retcode, "SQLAllocHandle(SQL_HANDLE_DBC)",
          henv, SQL_HANDLE_DBC);

// Set Login Timeout
retcode = SQLSetConnectAttr(hdbc, SQL_LOGIN_TIMEOUT, (SQLPOINTER)5, 0);
CHECK_ERROR(retcode, "SQLSetConnectAttr(SQL_LOGIN_TIMEOUT)",
          hdbc, SQL_HANDLE_DBC);

// Set Auto Commit
retcode = SQLSetConnectAttr(hdbc, SQL_ATTR_AUTOCOMMIT,
                            (SQLPOINTER)(1), 0);
CHECK_ERROR(retcode, "SQLSetConnectAttr(SQL_ATTR_AUTOCOMMIT)",
          hdbc, SQL_HANDLE_DBC);

// Connect to DSN
sprintf(loginfo, "SQLConnect(DSN:%s)", dsn);
retcode = SQLConnect(hdbc, (SQLCHAR*) dsn, SQL_NTS,
                     (SQLCHAR*) NULL, 0, NULL, 0);
CHECK_ERROR(retcode, loginfo, hdbc, SQL_HANDLE_DBC);

// init table info.
Exec(hdbc, "drop table if exists test_odbc_batch_insert");
Exec(hdbc, "create table test_odbc_batch_insert(id int primary key, col varchar2(50))");

// 下面的代码根据用户输入的数据量，构造出将要入库的数据：
{
    SQLRETURN retcode;
    SQLHSTMT hstmtinesrt = SQL_NULL_HSTMT;
    int        i;
```

```
      SQLCHAR     *sql = NULL;
      SQLINTEGER   *ids  = NULL;
      SQLCHAR     *cols = NULL;
      SQLLEN      *bufLenIds = NULL;
      SQLLEN      *bufLenCols = NULL;
      SQLUSMALLINT *operptr = NULL;
      SQLUSMALLINT *statusptr = NULL;
      SQLULEN     process = 0;

      // 这里是按列构造，每个字段的内存连续存放在一起。
      ids = (SQLINTEGER*)malloc(sizeof(ids[0]) * batchCount);
      cols = (SQLCHAR*)malloc(sizeof(cols[0]) * batchCount * 50);
      // 这里是每个字段中，每一行数据的内存长度。
      bufLenIds = (SQLLEN*)malloc(sizeof(bufLenIds[0]) * batchCount);
      bufLenCols = (SQLLEN*)malloc(sizeof(bufLenCols[0]) * batchCount);
      // 该行是否需要被处理，SQL_PARAM_IGNORE 或 SQL_PARAM_PROCEED
      operptr = (SQLUSMALLINT*)malloc(sizeof(operptr[0]) * batchCount);
      memset(operptr, 0, sizeof(operptr[0]) * batchCount);
      // 该行的处理结果。
      // 注：由于数据库中处理方式是同一语句隶属同一事务中，所以如果出错，那么待处理数据都将是出错的，并不会部分入库。
      statusptr = (SQLUSMALLINT*)malloc(sizeof(statusptr[0]) * batchCount);
      memset(statusptr, 88, sizeof(statusptr[0]) * batchCount);

      if (NULL == ids || NULL == cols || NULL == bufLenCols || NULL == bufLenIds)
      {
         fprintf(stderr, "FAILED:\tmalloc data memory failed\n");
         goto exit;
      }

      for (int i = 0; i < batchCount; i++)
      {
         ids[i] = i;
         sprintf(cols + 50 * i, "column test value %d", i);
         bufLenIds[i] = sizeof(ids[i]);
         bufLenCols[i] = strlen(cols + 50 * i);
         operptr[i] = (i < ignoreCount) ? SQL_PARAM_IGNORE : SQL_PARAM_PROCEED;
      }

      // Allocate Statement Handle
      retcode = SQLAllocHandle(SQL_HANDLE_STMT, hdbc, &hstmtinesrt);
      CHECK_ERROR(retcode, "SQLAllocHandle(SQL_HANDLE_STMT)",
                hstmtinesrt, SQL_HANDLE_STMT);

      // Prepare Statement
      sql = (SQLCHAR*)"insert into test_odbc_batch_insert values(?, ?)";
      retcode = SQLPrepare(hstmtinesrt, (SQLCHAR*) sql, SQL_NTS);
      sprintf((char*)loginfo, "SQLPrepare log: %s", (char*)sql);
      CHECK_ERROR(retcode, loginfo, hstmtinesrt, SQL_HANDLE_STMT);

      retcode = SQLSetStmtAttr(hstmtinesrt, SQL_ATTR_PARAMSET_SIZE, (SQLPOINTER)batchCount,
sizeof(batchCount));
      CHECK_ERROR(retcode, "SQLSetStmtAttr", hstmtinesrt, SQL_HANDLE_STMT);

      retcode = SQLBindParameter(hstmtinesrt, 1, SQL_PARAM_INPUT, SQL_C_SLONG, SQL_INTEGER,
sizeof(ids[0]), 0,&(ids[0]), 0, bufLenIds);
```

```
      CHECK_ERROR(retcode, "SQLBindParameter for id", hstmtinesrt, SQL_HANDLE_STMT);

      retcode = SQLBindParameter(hstmtinesrt, 2, SQL_PARAM_INPUT, SQL_C_CHAR, SQL_CHAR, 50, 50, cols,
50, bufLenCols);
      CHECK_ERROR(retcode, "SQLBindParameter for cols", hstmtinesrt, SQL_HANDLE_STMT);

      retcode = SQLSetStmtAttr(hstmtinesrt, SQL_ATTR_PARAMS_PROCESSED_PTR, (SQLPOINTER)&process,
sizeof(process));
      CHECK_ERROR(retcode, "SQLSetStmtAttr for SQL_ATTR_PARAMS_PROCESSED_PTR", hstmtinesrt,
SQL_HANDLE_STMT);

      retcode = SQLSetStmtAttr(hstmtinesrt, SQL_ATTR_PARAM_STATUS_PTR, (SQLPOINTER)statusptr,
sizeof(statusptr[0]) * batchCount);
      CHECK_ERROR(retcode, "SQLSetStmtAttr for SQL_ATTR_PARAM_STATUS_PTR", hstmtinesrt,
SQL_HANDLE_STMT);

      retcode = SQLSetStmtAttr(hstmtinesrt, SQL_ATTR_PARAM_OPERATION_PTR, (SQLPOINTER)operptr,
sizeof(operptr[0]) * batchCount);
      CHECK_ERROR(retcode, "SQLSetStmtAttr for SQL_ATTR_PARAM_OPERATION_PTR", hstmtinesrt,
SQL_HANDLE_STMT);

      retcode = SQLExecute(hstmtinesrt);
      sprintf((char*)loginfo, "SQLExecute stmt log: %s", (char*)sql);
      CHECK_ERROR(retcode, loginfo, hstmtinesrt, SQL_HANDLE_STMT);

      retcode = SQLRowCount(hstmtinesrt, &rowsCount);
      CHECK_ERROR(retcode, "SQLRowCount execution", hstmtinesrt, SQL_HANDLE_STMT);

      if (rowsCount != (batchCount - ignoreCount))
      {
         sprintf(loginfo, "(batchCount - ignoreCount)(%d) != rowsCount(%d)", (batchCount -
ignoreCount), rowsCount);
         CHECK_ERROR(SQL_ERROR, loginfo, NULL, SQL_HANDLE_STMT);
      }
      else
      {
         sprintf(loginfo, "(batchCount - ignoreCount)(%d) == rowsCount(%d)", (batchCount -
ignoreCount), rowsCount);
         CHECK_ERROR(SQL_SUCCESS, loginfo, NULL, SQL_HANDLE_STMT);
      }

      if (rowsCount != process)
      {
         sprintf(loginfo, "process(%d) != rowsCount(%d)", process, rowsCount);
         CHECK_ERROR(SQL_ERROR, loginfo, NULL, SQL_HANDLE_STMT);
      }
      else
      {
         sprintf(loginfo, "process(%d) == rowsCount(%d)", process, rowsCount);
         CHECK_ERROR(SQL_SUCCESS, loginfo, NULL, SQL_HANDLE_STMT);
      }

      for (int i = 0; i < batchCount; i++)
      {
         if (i < ignoreCount)
```

```
        {
            if (statusptr[i] != SQL_PARAM_UNUSED)
            {
                sprintf(loginfo, "statusptr[%d](%d) != SQL_PARAM_UNUSED", i, statusptr[i]);
                CHECK_ERROR(SQL_ERROR, loginfo, NULL, SQL_HANDLE_STMT);
            }
        }
        else if (statusptr[i] != SQL_PARAM_SUCCESS)
        {
            sprintf(loginfo, "statusptr[%d](%d) != SQL_PARAM_SUCCESS", i, statusptr[i]);
            CHECK_ERROR(SQL_ERROR, loginfo, NULL, SQL_HANDLE_STMT);
        }
    }

    retcode = SQLFreeHandle(SQL_HANDLE_STMT, hstmtinesrt);
    sprintf((char*)loginfo, "SQLFreeHandle hstmtinesrt");
    CHECK_ERROR(retcode, loginfo, hstmtinesrt, SQL_HANDLE_STMT);
}


exit:
    printf ("\nComplete.\n");

    // Connection
    if (hdbc != SQL_NULL_HDBC) {
        SQLDisconnect(hdbc);
        SQLFreeHandle(SQL_HANDLE_DBC, hdbc);
    }

    // Environment
    if (henv != SQL_NULL_HENV)
        SQLFreeHandle(SQL_HANDLE_ENV, henv);

    return 0;
}
```

6.3.6. 链接字符

表 6-50. 链接字符串

字符串	描述
host	要链接的主机名。如果主机名以斜杠开头，则它声明使用 Unix 域套接字通讯而不是 TCP/IP 通讯；该值就是套接字文件所存储的目录。如果没有声明 host，那么默认是与位于/tmp 目录（或者安装 vastbase 的时候声明的套接字目录）里面的 Unix-域套接字链接。在没有 Unix 域套接字的机器上，默认与 localhost 链接。
hostaddr	与之链接的主机的 IP 地址，是标准的 IPv4 地址格式，比如，172.28.40.9。如果机器支持 IPv6，那么也可以使用 IPv6 的地址。如果声明了一个非空的字符串，那么使用 TCP/IP 通讯机制。

字符串	描述
	使用 hostaddr 取代 host 可以让应用避免一次主机名查找，这一点对于那些有时间约束的应用来说可能是非常重要的。不过，GSSAPI 或 SSPI 认证方法要求主机名（host）。因此，应用下面的规则： 1. 如果声明了不带 hostaddr 的 host 那么就强制进行主机名查找。 2. 如果声明中没有 host，hostaddr 的值给出服务器网络地址；如果认证方法要求主机名，那么链接尝试将失败。 3. 如果同时声明了 host 和 hostaddr，那么 hostaddr 的值作为服务器网络地址。host 的值将被忽略，除非认证方法需要它，在这种情况下它将被用作主机名。 须知 要注意如果 host 不是网络地址 hostaddr 处的服务器名，那么认证很有可能失败。 如果主机名（host）和主机地址都没有，那么 libpq 将使用一个本地的 Unix 域套接字进行链接；或者是在没有 Unix 域套接字的机器上，它将尝试与 localhost 链接。
port	主机服务器的端口号，或者在 Unix 域套接字链接时的套接字扩展文件名。
user	要链接的用户名，缺省是与运行该应用的用户操作系统名同名的用户。
dbname	数据库名，缺省和用户名相同。
password	如果服务器要求口令认证，所用的口令。
connect_timeout	链接的最大等待时间，以秒计（用十进制整数字符串书写），0 或者不声明表示无穷。不建议把链接超时的值设置得小于 2 秒。
client_encoding	为这个链接设置 client_encoding 配置参数。除了对应的服务器选项接受的值，你可以使用 auto 从客户端中的当前环境中确定正确的编码（Unix 系统上是 LC_CTYPE 环境变量）。
options	添加命令行选项以在运行时发送到服务器。
application_name	为 application_name 配置参数指定一个值，表明当前用户身份。
keepalives	控制客户端侧的 TCP 保持激活是否使用。缺省值是 1，意思为打开，但是如果不想要保持激活，你可以更改为 0，意思为关闭。通过 Unix 域套接字做的链接忽略这个参数。
keepalives_idle	在 TCP 应该发送一个保持激活的信息给服务器之后，控制不活动的秒数。0 值表示使用系统缺省。通过 Unix 域套接字做的链接或者如果禁用了保持激活则忽略这个参数。
keepalives_interval	在 TCP 保持激活信息没有被应该传播的服务器承认之后，控制秒数。0 值表示使用

字符串	描述
	系统缺省。通过 Unix 域套接字做的链接或者如果禁用了保持激活则忽略这个参数。
keepalives_count	添加命令行选项以在运行时发送到服务器。例如，设置为-c comm_debug_mode=off 设置 guc 参数 comm_debug_mode 参数的会话的值为 off。

7. 导入数据

Vastbase 提供了灵活的数据入库方式：INSERT、COPY 以及 vsql 元命令\copy。各方式具有不同的特点，具体请参见表 7-1。

表 7-1. 导入方式特点说明

方式	特点
INSERT	通过 INSERT 语句插入一行或多行数据，及从指定表插入数据。
COPY	通过 COPY FROM STDIN 语句直接向 Vastbase 写入数据。 通过 JDBC 驱动的 CopyManager 接口从其他数据库向 Vastbase 写入数据时，具有业务数据无需落地成文件的优势。
vsql 工具的元命令\copy	与直接使用 SQL 语句 COPY 不同，该命令读取/写入的文件只能是 vsql 客户端所在机器上的本地文件。 说明 \COPY 只适合小批量、格式良好的数据导入，不会对非法字符做预处理，也无容错能力，无法适用于含有异常数据的场景。导入数据应优先选择 COPY。

7.1. 通过 INSERT 语句直接写入数据

用户可以通过以下方式执行 11.16.89INSERT 语句直接向 Vastbase 写入数据：

- 使用 Vastbase 提供的客户端工具向 Vastbase 写入数据。

  请参见 3.8.2 向表中插入数据。

- 通过 JDBC/ODBC 驱动连接数据库执行 INSERT 语句向 Vastbase 写入数据。

  详细内容请参见 3.2 连接数据库。

Vastbase 支持完整的数据库事务级别的增删改操作。INSERT 是最简单的一种数据写入方式，这种方式适合数据写入量不大， 并发度不高的场景。

7.2. 使用 COPY FROM STDIN 导入数据

7.2.1. 关于 COPY FROM STDIN 导入数据

这种方式适合数据写入量不太大， 并发度不太高的场景。

用户可以使用以下方式通过 COPY FROM STDIN 语句直接向 Vastbase 写入数据。

- 通过键盘输入向 Vastbase 写入数据。详细请参见 11.16.35COPY。
- 通过 JDBC 驱动的 CopyManager 接口从文件或者数据库向 Vastbase 写入数据。此方法支持 COPY 语法中 copy option 的所有参数。

7.2.2. CopyManager 类简介

CopyManager 是 Vastbase JDBC 驱动中提供的一个 API 接口类，用于批量向 Vastbase 中导入数据。

CopyManager 的继承关系

CopyManager 类位于 org.postgresql.copy Package 中，继承自 java.lang.Object 类，该类的声明如下：

```
public class CopyManager
extends Object
```

构造方法

public CopyManager(BaseConnection connection)
throws SQLException

常用方法

表 7-2. CopyManager 常用方法

返回值	方法	描述	THROWS
CopyIn	copyIn(String sql)	-	SQLException
long	copyIn(String sql, InputStream from)	使用 COPY FROM STDIN 从 InputStream 中快速向数据库中的表导入数据。	SQLException,IOException
long	copyIn(String sql, InputStream from, int bufferSize)	使用 COPY FROM STDIN 从 InputStream	SQLException,IOException

返回值	方法	描述	THROWS
		中快速向数据库中的表导入数据。	
long	copyIn(String sql, Reader from)	使用 COPY FROM STDIN 从 Reader 中快速向数据库中的表导入数据。	SQLException,IOException
long	copyIn(String sql, Reader from, int bufferSize)	使用 COPY FROM STDIN 从 Reader 中快速向数据库中的表导入数据。	SQLException,IOException
CopyOut	copyOut(String sql)	-	SQLException
long	copyOut(String sql, OutputStream to)	将一个 COPY TO STDOUT 的结果集从数据库发送到 OutputStream 类中。	SQLException,IOException
long	copyOut(String sql, Writer to)	将一个 COPY TO STDOUT 的结果集从数据库发送到 Writer 类中。	SQLException,IOException

7.2.3.示例 1：通过本地文件导入导出数据

在使用 JAVA 语言基于 Vastbase 进行二次开发时，可以使用 CopyManager 接口，通过流方式，将数据库中的数据导出到本地文件或者将本地文件导入数据库中，文件格式支持 CSV、TEXT 等格式。

样例程序如下，执行时需要加载 Vastbase 的 JDBC 驱动。

```
import java.sql.Connection;
import java.sql.DriverManager;
import java.io.IOException;
import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.sql.SQLException;
import org.postgresql.copy.CopyManager;
import org.postgresql.core.BaseConnection;

public class Copy{

    public static void main(String[] args)
```

```
    {
    String urls = new String("jdbc:postgresql://localhost:5432/postgres"); //数据库URL
    String username = new String("username");          //用户名
    String password = new String("passwd");           //密码
    String tablename = new String("migration_table"); //定义表信息
    String tablename1 = new String("migration_table_1"); //定义表信息
    String driver = "org.postgresql.Driver";
    Connection conn = null;

    try {
       Class.forName(driver);
       conn = DriverManager.getConnection(urls, username, password);
    } catch (ClassNotFoundException e) {
        e.printStackTrace(System.out);
    } catch (SQLException e) {
        e.printStackTrace(System.out);
    }

    // 将表migration_table中数据导出到本地文件d:/data.txt
    try {
       copyToFile(conn, "d:/data.txt", "(SELECT * FROM migration_table)");
    } catch (SQLException e) {
       // TODO Auto-generated catch block
       e.printStackTrace();
    } catch (IOException e) {
       // TODO Auto-generated catch block
       e.printStackTrace();
    }
    //将d:/data.txt中的数据导入到migration_table_1中。
    try {
       copyFromFile(conn, "d:/data.txt", tablename1);
    } catch (SQLException e) {
       // TODO Auto-generated catch block
       e.printStackTrace();
    } catch (IOException e) {
       // TODO Auto-generated catch block
       e.printStackTrace();
    }

    // 将表migration_table_1中的数据导出到本地文件d:/data1.txt
    try {
       copyToFile(conn, "d:/data1.txt", tablename1);
    } catch (SQLException e) {
       // TODO Auto-generated catch block
       e.printStackTrace();
    } catch (IOException e) {
       // TODO Auto-generated catch block
       e.printStackTrace();
    }
}

public static void copyFromFile(Connection connection, String filePath, String tableName)
      throws SQLException, IOException {
```

```
    FileInputStream fileInputStream = null;

    try {
       CopyManager copyManager = new CopyManager((BaseConnection)connection);
       fileInputStream = new FileInputStream(filePath);
       copyManager.copyIn("COPY " + tableName + " FROM STDIN with (" + "DELIMITER"+"'"+ delimiter
+  "'" + "ENCODING " + "'" + encoding + "')", fileInputStream);
    } finally {
       if (fileInputStream != null) {
          try {
             fileInputStream.close();
          } catch (IOException e) {
             e.printStackTrace();
          }
       }
    }
 }
    public static void copyToFile(Connection connection, String filePath, String tableOrQuery)
        throws SQLException, IOException {

     FileOutputStream fileOutputStream = null;

     try {
        CopyManager copyManager = new CopyManager((BaseConnection)connection);
        fileOutputStream = new FileOutputStream(filePath);
        copyManager.copyOut("COPY " + tableOrQuery + " TO STDOUT", fileOutputStream);
     } finally {
        if (fileOutputStream != null) {
           try {
              fileOutputStream.close();
           } catch (IOException e) {
              e.printStackTrace();
           }
        }
     }
  }
}
```

7.2.4.示例 2：从 MySQL 向 Vastbase 进行数据迁移

下面示例演示如何通过 CopyManager 从 MySQL 向 Vastbase 进行数据迁移的过程。

```
import java.io.StringReader;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.Statement;

import org.postgresql.copy.CopyManager;
import org.postgresql.core.BaseConnection;

public class Migration{

   public static void main(String[] args) {
```

```
        String url = new String("jdbc:postgresql://localhost:5432/postgres"); //数据库 URL
        String user = new String("username");          //Vastbase 用户名
        String pass = new String("passwd");           //Vastbase 密码
        String tablename = new String("migration_table_1"); //定义表信息
        String delimiter = new String("|");            //定义分隔符
        String encoding = new String("UTF8");          //定义字符集
        String driver = "org.postgresql.Driver";
        StringBuffer buffer = new StringBuffer();      //定义存放格式化数据的缓存

        try {
            //获取源数据库查询结果集
            ResultSet rs = getDataSet();

            //遍历结果集，逐行获取记录
            //将每条记录中各字段值，按指定分隔符分割，由换行符结束，拼成一个字符串
            //把拼成的字符串，添加到缓存 buffer
            while (rs.next()) {
                buffer.append(rs.getString(1) + delimiter
                        + rs.getString(2) + delimiter
                        + rs.getString(3) + delimiter
                        + rs.getString(4)
                        + "\n");
            }
            rs.close();

            try {
                //建立目标数据库连接
                Class.forName(driver);
                Connection conn = DriverManager.getConnection(url, user, pass);
                BaseConnection baseConn = (BaseConnection) conn;
                baseConn.setAutoCommit(false);

                //初始化表信息
                String sql = "Copy " + tablename + " from STDIN with (DELIMITER " + "'" + delimiter +
"'" +","+ " ENCODING " + "'" + encoding + "'");

                //提交缓存 buffer 中的数据
                CopyManager cp = new CopyManager(baseConn);
                StringReader reader = new StringReader(buffer.toString());
                cp.copyIn(sql, reader);
                baseConn.commit();
                reader.close();
                baseConn.close();
            } catch (ClassNotFoundException e) {
                e.printStackTrace(System.out);
            } catch (SQLException e) {
                e.printStackTrace(System.out);
            }

        } catch (Exception e) {
            e.printStackTrace();
        }
    }
```

```
    //********************************
    // 从源数据库返回查询结果集
    //********************************
    private static ResultSet getDataSet() {
        ResultSet rs = null;
        try {
            Class.forName("com.mysql.jdbc.Driver").newInstance();
            Connection conn =
DriverManager.getConnection("jdbc:mysql://10.119.179.227:3306/jack?useSSL=false&allowPublicKeyRet
rieval=true", "jack", "Vastbase@123");
            Statement stmt = conn.createStatement();
            rs = stmt.executeQuery("select * from migration_table");
        } catch (SQLException e) {
            e.printStackTrace();
        } catch (Exception e) {
            e.printStackTrace();
        }
        return rs;
    }
}
```

7.3. 使用 vsql 元命令导入数据

DWS 的 vsql 工具提供了元命令\copy 进行数据导入。

\copy 命令

\copy 命令格式以及说明参见表 7-3。

表 7-3. \copy 元命令说明

语法	说明
\copy { table [(column_list)] | (query) } { from | to } { filename | stdin | stdout | pstdin | pstdout } [with] [binary] [delimiter [as] 'character'] [null [as] 'string'] [csv [header] [quote [as] 'character'] [escape [as] 'character'] [force quote column_list | *] [force not null column_list]]	在任何 vsql 客户端登录数据库成功后，可以使用该命令进行数据的导入/导出。但是与 SQL 的 COPY 命令不同，该命令读取/写入的文件是本地文件，而非数据库服务器端文件；所以，要操作的文件的可访问性、权限等，都是受限于本地用户的权限。 说明 \COPY 只适合小批量、格式良好的数据导入，不会对非法字符做预处理，也无容错能力，无法适用于含有异常数据的场景。导入数据应优先选择 COPY。

参数说明

- table

  表的名称（可以有模式修饰）。

  取值范围：已存在的表名。

- column_list

  可选的待拷贝字段列表。

  取值范围：任意字段。如果没有声明字段列表，将使用所有字段。

- query

  其结果将被拷贝。

  取值范围：一个必须用圆括弧包围的 SELECT 或 VALUES 命令。

- filename

  文件的绝对路径。执行 copy 命令的用户必须有此路径的写权限。

- stdin

  声明输入是来自标准输入。

- stdout

  声明输出打印到标准输出。

- pstdin

  声明输入是来自 vsql 的标准输入。

- pstout

  声明输出打印到 vsql 的标准输出。

- binary

  使用二进制格式存储和读取，而不是以文本的方式。在二进制模式下，不能声明 DELIMITER，NULL，CSV 选项。指定 binary 类型后，不能再通过 option 或 copy_option 指定 CSV、FIXED、TEXT 等类型。

- delimiter [as] 'character'

  指定数据文件行数据的字段分隔符。

说明

- 分隔符不能是\r 和\n。
- 分隔符不能和 null 参数相同，CSV 格式数据的分隔符不能和 quote 参数相同。
- TEXT 格式数据的分隔符不能包含： \.abcdefghijklmnopqrstuvwxyz0123456789。
- 数据文件中单行数据长度需<1GB，如果分隔符较长且数据列较多的情况下，会影响导出有效数据的长度。
- 分隔符推荐使用多字符和不可见字符。多字符例如'$^&'；不可见字符例如 0x07，0x08，0x1b 等。

取值范围：支持多字符分隔符，但分隔符不能超过 10 个字节。

默认值：

- TEXT 格式的默认分隔符是水平制表符（tab）。
- CSV 格式的默认分隔符为“,”。
- FIXED 格式没有分隔符。

❖ null [as] 'string'

用来指定数据文件中空值的表示。

取值范围：

- null 值不能是\r 和\n，最大为 100 个字符。
- null 值不能和分隔符、quote 参数相同。

默认值：

- CSV 格式下默认值是一个没有引号的空字符串。
- 在 TEXT 格式下默认值是\N。

❖ header

指定导出数据文件是否包含标题行，标题行一般用来描述表中每个字段的信息。header 只能用于 CSV，FIXED 格式的文件中。

在导入数据时，如果 header 选项为 on，则数据文本第一行会被识别为标题行，会忽略此行。如果 header 为 off，而数据文件中第一行会被识别为数据。

在导出数据时，如果 header 选项为 on，则需要指定 fileheader。fileheader 是指定导出数据包含标题行的定义文件。如果 header 为 off，则导出数据文件不包含标题行。

取值范围：true/on，false/off。

默认值：false

❖ quote [as] 'character'

CSV 格式文件下的引号字符。

默认值：双引号。

说明

- quote 参数不能和分隔符、null 参数相同。
- quote 参数只能是单字节的字符。
- 推荐不可见字符作为 quote，例如 0x07，0x08，0x1b 等。

❖ escape [as] 'character'

CSV 格式下，用来指定逃逸字符，逃逸字符只能指定为单字节字符。

默认值：双引号。当与 quote 值相同时，会被替换为'\0'。

❖ force quote column_list | *

在 CSV COPY TO 模式下，强制在每个声明的字段周围对所有非 NULL 值都使用引号包围。NULL 输出不会被引号包围。

取值范围：已存在的字段。

- force not null column_list

在 CSV COPY FROM 模式下，指定的字段输入不能为空。

取值范围：已存在的字段。

任务示例

1. 创建目标表 a。

```
vastbase=# CREATE TABLE a(a int);
```

2. 导入数据。

a. 从 stdin 拷贝数据到目标表 a。

```
vastbase=# \copy a from stdin;
```

出现>>符号提示时，输入数据，输入\.时结束。

```
Enter data to be copied followed by a newline.
End with a backslash and a period on a line by itself.
>> 1
>> 2
>> \.
```

查询导入目标表 a 的数据。

```
vastbase=# SELECT * FROM a;
 a
---
 1
 2
(2 rows)
```

b. 从本地文件拷贝数据到目标表 a。假设存在本地文件/home/vastbase/2.csv。

- 分隔符为 ‘,’ 。
- 在导入过程中，若数据源文件比外表定义的列数多，则忽略行尾多出来的列。

```
vastbase=# \copy a FROM '/home/vastbase/2.csv' WITH
(delimiter',',IGNORE_EXTRA_DATA 'on');
```

7.4. 使用 vb_restore 命令导入数据

操作场景

vb_restore 是 Vastbase 提供的与 vb_dump 配套的导入工具。通过该工具，可将 vb_dump 导出的文件导入至数据库。vb_restore 支持导入的文件格式包含自定义归档格式、目录归档格式和 tar 归档格式。

vb_restore 具备如下两种功能。

- 导入至数据库

  如果指定了数据库，则数据将被导入到指定的数据库中。其中，并行导入必须指定连接数据库的密码。

- 导入至脚本文件

  如果未指定导入数据库，则创建包含重建数据库所需的 SQL 语句脚本，并将其写入至文件或者标准输出。该脚本文件等效于 vb_dump 导出的纯文本格式文件。

vb_restore 工具在导入时，允许用户选择需要导入的内容，并支持在数据导入前对等待导入的内容进行排序。

操作步骤

说明

vb_restore 默认是以追加的方式进行数据导入。为避免多次导入造成数据异常，在进行导入时，建议选择使用"-c" 和"-e"参数。"-c"表示在重新创建数据库对象前，清理（删除）已存在于将要还原的数据库中的数据库对象；"-e"表示当发送 SQL 语句到数据库时如果出现错误请退出，默认状态下会继续，且在导入后会显示一系列错误信息。

步骤 1 以操作系统用户 vastbase 登录数据库主节点。

步骤 2 使用 vb_restore 命令，从 vastbase 整个数据库内容的导出文件中，将数据库的所有对象的定义导入到 backupdb。

```
vb_restore -W Bigdata@123 -U jack /home/vastbase/backup/MPPDB_backup.tar -p 5432 -d backupdb
-s -e -c
```

表 7-4. 常用参数说明

参数	参数说明	举例
-U	连接数据库的用户名。	-U jack
-W	指定用户连接的密码。 • 如果主机的认证策略是 trust，则不会对数据库管理员进行密码验证，即无需输入-W 选项； • 如果没有-W 选项，并且不是数据库管理员，会提示用户输入密码。	-W Bigdata@123
-d	连接数据库 dbname，并直接将数据导入到该数据库中。	-d backupdb
-p	指定服务器所监听的 TCP 端口或本地 Unix 域套接字后缀，以确保连接。	-p 5432

参数	参数说明	举例
-e	当发送 SQL 语句到数据库时如果出现错误，退出当前出现错误的任务，并执行其他导入任务。即默认状态下会忽略错误任务并继续执行导入，且在导入后会显示一系列错误信息。	-
-c	在重新创建数据库对象前，清理（删除）已存在于将要导入的数据库中的数据库对象。	-
-s	只导入模式定义，不导入数据。当前的序列值也不会被导入。	-

其他参数说明请参见《工具参考》中“服务端工具 > vb_restore”章节。

示例

示例一：执行 vb_restore，导入指定 MPPDB_backup.dmp 文件（自定义归档格式）中 vastbase 数据库的数据和对象定义。

```
vb_restore -W Bigdata@123 backup/MPPDB_backup.dmp -p 5432 -d backupdb
vb_restore[2017-07-21 19:16:26]: restore operation successful
vb_restore: total time: 13053  ms
```

示例二：执行 vb_restore，导入指定 MPPDB_backup.tar 文件（tar 归档格式）中 vastbase 数据库的数据和对象定义。

```
vb_restore backup/MPPDB_backup.tar -p 5432 -d backupdb
vb_restore[2017-07-21 19:21:32]: restore operation successful
vb_restore[2017-07-21 19:21:32]: total time: 21203  ms
```

示例三：执行 vb_restore，导入指定 MPPDB_backup 目录文件（目录归档格式）中 vastbase 数据库的数据和对象定义。

```
vb_restore backup/MPPDB_backup -p 5432 -d backupdb
vb_restore[2017-07-21 19:26:46]: restore operation successful
vb_restore[2017-07-21 19:26:46]: total time: 21003  ms
```

示例四：执行 vb_restore，将 vastbase 数据库的所有对象的定义导入至 backupdb 数据库。导入前，vastbase 存在完整的定义和数据，导入后，backupdb 数据库只存在所有对象定义，表没有数据。

```
vb_restore -W Bigdata@123 /home/vastbase/backup/MPPDB_backup.tar -p 5432 -d backupdb -s -e -c
vb_restore[2017-07-21 19:46:27]: restore operation successful
vb_restore[2017-07-21 19:46:27]: total time: 32993  ms
```

示例五：执行 vb_restore，导入 MPPDB_backup.dmp 文件中 PUBLIC 模式的所有定义和数据。在导入时会先删除已经存在的对象，如果原对象存在跨模式的依赖则需手工强制干预。

```
vb_restore backup/MPPDB_backup.dmp -p 5432 -d backupdb -e -c -n PUBLIC
vb_restore: [archiver (db)] Error while PROCESSING TOC:
vb_restore: [archiver (db)] Error from TOC entry 313; 1259 337399 TABLE table1 vastbasea
```

```
vb_restore: [archiver (db)] could not execute query: ERROR:  cannot drop table table1 because other
objects depend on it
DETAIL:  view t1.v1 depends on table table1
HINT:  Use DROP ... CASCADE to drop the dependent objects too.
Command was: DROP TABLE public.table1;
```

手工删除依赖，导入完成后再重新创建。

```
vb_restore backup/MPPDB_backup.dmp -p 5432 -d backupdb -e -c -n PUBLIC
vb_restore[2017-07-21 19:52:26]: restore operation successful
vb_restore[2017-07-21 19:52:26]: total time: 2203  ms
```

示例六：执行 vb_restore，导入 MPPDB_backup.dmp 文件中 hr 模式下表 staffs 的定义。在导入之前，staffs 表不存在。

```
vb_restore backup/MPPDB_backup.dmp -p 5432 -d backupdb -e -c -s -n hr -t staffs
vb_restore[2017-07-21 19:56:29]: restore operation successful
vb_restore[2017-07-21 19:56:29]: total time: 21000  ms
```

示例七：执行 vb_restore，导入 MPPDB_backup.dmp 文件中 hr 模式下表 staffs 的数据。在导入之前，staffs 表不存在数据。

```
vb_restore backup/MPPDB_backup.dmp -p 5432 -d backupdb -e -a -n hr -t staffs
vb_restore[2017-07-21 20:12:32]: restore operation successful
vb_restore[2017-07-21 20:12:32]: total time: 20203  ms
```

示例八：执行 vb_restore，导入指定表 hr.staffs 的定义。在导入之前，hr.staffs 表的数据是存在的。

```
human_resource=# select * from hr.staffs;
 staff_id | first_name  | last_name  | email   |    phone_number    |     hire_date     |
employment_id |  salary  | commission_pct | manager_id | section_id
----------+-------------+------------+----------+--------------------+---------------------+----
-----------+----------+----------------+------------+------------
     200 | Jennifer    | Whalen     | JWHALEN  | 515.123.4444       | 1987-09-17 00:00:00 | AD_ASST
|  4400.00 |             |       101 |       10
     201 | Michael     | Hartstein  | MHARTSTE | 515.123.5555       | 1996-02-17 00:00:00 | MK_MAN
| 13000.00 |             |       100 |       20

vsql -d human_resource -p 5432

vsql ((Vastbase 2.2.0 build ) compiled at 2021-01-26 19:22:33 commit 0 last mr  )
Non-SSL connection (SSL connection is recommended when requiring high-security)
Type "help" for help.

human_resource=# drop table hr.staffs CASCADE;
NOTICE:  drop cascades to view hr.staff_details_view
DROP TABLE

vb_restore -W Bigdata@123 /home/vastbase/backup/MPPDB_backup.tar -p 5432 -d human_resource -n hr -t
staffs -s -e
restore operation successful
total time: 904  ms

human_resource=# select * from hr.staffs;
 staff_id | first_name | last_name | email | phone_number | hire_date | employment_id | salary |
commission_pct | manager_id | section_id
----------+------------+-----------+-------+--------------+-----------+---------------+--------+-
---------------+------------+------------
(0 rows)
```

示例九：执行 vb_restore，导入 staffs 和 areas 两个指定表的定义和数据。在导入之前，staffs 和 areas 表不存在。

```
human_resource=# \d
                          List of relations
 Schema |       Name       | Type  |  Owner   |           Storage
--------+------------------+-------+----------+-------------------------------
 hr     | employment_history | table | vastbase | {orientation=row,compression=no}
 hr     | employments       | table | vastbase | {orientation=row,compression=no}
 hr     | places           | table | vastbase | {orientation=row,compression=no}
 hr     | sections         | table | vastbase | {orientation=row,compression=no}
 hr     | states           | table | vastbase | {orientation=row,compression=no}
(5 rows)

vb_restore -W Bigdata@123 /home/vastbase/backup/MPPDB_backup.tar -p 5432 -d human_resource -n hr -t
staffs -n hr -t areas
restore operation successful
total time: 724  ms

human_resource=# \d
                          List of relations
 Schema |       Name       | Type  |  Owner   |           Storage
--------+------------------+-------+----------+-------------------------------
 hr     | areas            | table | vastbase | {orientation=row,compression=no}
 hr     | employment_history | table | vastbase | {orientation=row,compression=no}
 hr     | employments       | table | vastbase | {orientation=row,compression=no}
 hr     | places           | table | vastbase | {orientation=row,compression=no}
 hr     | sections         | table | vastbase | {orientation=row,compression=no}
 hr     | staffs           | table | vastbase | {orientation=row,compression=no}
 hr     | states           | table | vastbase | {orientation=row,compression=no}
(7 rows)

human_resource=# select * from hr.areas;
 area_id |      area_name
---------+------------------------
       4 | Middle East and Africa
       1 | Europe
       2 | Americas
       3 | Asia
(4 rows)
```

示例十：执行 vb_restore，导入 hr 的模式，包含模式下的所有对象定义和数据。

```
vb_restore -W Bigdata@123  /home/vastbase/backup/MPPDB_backup1.sql -p 5432 -d backupdb -n hr -e -c
restore operation successful
total time: 702  ms
```

示例十一：执行 vb_restore，同时导入 hr 和 hr1 两个模式，仅导入模式下的所有对象定义。

```
vb_restore -W Bigdata@123 /home/vastbase/backup/MPPDB_backup2.dmp -p 5432 -d backupdb -n hr -n hr1
-s
restore operation successful
total time: 665  ms
```

示例十二：执行 vb_restore，将 human_resource 数据库导出文件进行解密并导入至 backupdb 数据库中。

```
vastbase=# create database backupdb;
CREATE DATABASE
```

```
vb_restore /home/vastbase/backup/MPPDB_backup.tar -p 5432 -d backupdb --with-key=1234567812345678
restore operation successful
total time: 23472  ms

vsql -d backupdb -p 5432 -r

vsql ((Vastbase 2.2.0 build ) compiled at 2021-01-26 19:22:33 commit 0 last mr  )
Non-SSL connection (SSL connection is recommended when requiring high-security)
Type "help" for help.

backupdb=# select * from hr.areas;
 area_id |      area_name
---------+------------------------
       4 | Middle East and Africa
       1 | Europe
       2 | Americas
       3 | Asia
(4 rows)
```

7.5. 更新表中数据

7.5.1. 使用 DML 命令更新表

Vastbase 支持标准的数据库操作语言（DML）命令，对表进行更新。

操作步骤

假设存在表 customer_t，表结构如下：

```
vastbase=# CREATE TABLE customer_t
( c_customer_sk           integer,
  c_customer_id           char(5),
  c_first_name            char(6),
  c_last_name             char(8)
) ;
```

可以使用如下 DML 命令对表进行数据更新。

- 使用 INSERT 向表中插入数据。
  - 向表 customer_t 中插入一行。

    ```
    vastbase=# INSERT INTO customer_t (c_customer_sk, c_customer_id, c_first_name,c_last_name)
    VALUES (3769, 5, 'Grace','White');
    ```

  - 向表 customer_t 中插入多行数据。

    ```
    vastbase=# INSERT INTO customer_t (c_customer_sk, c_customer_id, c_first_name,c_last_name)
    VALUES
    (6885, 1, 'Joes', 'Hunter'),
    (4321, 2, 'Lily','Carter'),
    (9527, 3, 'James', 'Cook'),
    (9500, 4, 'Lucy', 'Baker');
    ```

    更多关于 INSERT 的使用方法，请参见 3.8.2 向表中插入数据。

- 使用 UPDATE 更新表中数据。修改字段 c_customer_id 值为 0。

```
vastbase=# UPDATE customer_t SET c_customer_id = 0;
```

更多关于 UPDATE 的使用方法，请参见 11.16.113UPDATE。

- 使用 DELETE 删除表中的行。

可以使用 WHERE 子句指定需要删除的行，若不指定即删除表中所有的行，只保留数据结构。

```
vastbase=# DELETE FROM customer_t WHERE c_last_name = 'Baker';
```

更多关于 DELETE 的使用方法，请参见 11.16.61DELETE。

- 使用 TRUNCATE 命令快速从表中删除所有的行。

```
vastbase=# TRUNCATE TABLE customer_t;
```

更多关于 TRUNCATE 的使用方法，请参见 11.16.112TRUNCATE。

删除表时，DELETE 语句每次删除一行数据而 TRUNCATE 语句是通过释放表存储的数据页来删除数据，使用 TRUNCATE 语句比使用 DELETE 语句更加快速。

使用 DELETE 语句删除表时，仅删除数据，不释放存储空间。使用 TRUNCATE 语句删除表时，删除数据且释放存储空间。

7.5.2.使用合并方式更新和插入数据

在用户需要将一个表中所有的数据或大量的数据添加至现有表的场景下，Vastbase 提供了 MERGE INTO 语句通过两个表合并的方式高效地将新数据添加到现有表。

MERGE INTO 语句将目标表和源表中数据针对关联条件进行匹配，若关联条件匹配时对目标表进行 UPDATE，关联条件不匹配时对目标表执行 INSERT。此方法可以很方便地用来将两个表合并执行 UPDATE 和 INSERT，避免多次执行。

前提条件

进行 MERGE　INTO 操作的用户需要同时拥有目标表的 UPDATE 和 INSERT 权限，以及源表的 SELECT 权限。

操作步骤

步骤 1 创建源表 products，并插入数据。

```
vastbase=# CREATE TABLE products
( product_id INTEGER,
  product_name VARCHAR2(60),
  category VARCHAR2(60)
);

vastbase=# INSERT INTO products VALUES
(1502, 'olympus camera', 'electrncs'),
(1601, 'lamaze', 'toys'),
```

```
(1666, 'harry potter', 'toys'),
(1700, 'wait interface', 'books');
```

步骤 2 创建目标表 newproducts,并插入数据。

```
vastbase=# CREATE TABLE newproducts
( product_id INTEGER,
  product_name VARCHAR2(60),
  category VARCHAR2(60)
);

vastbase=# INSERT INTO newproducts VALUES
(1501, 'vivitar 35mm', 'electrncs'),
(1502, 'olympus ', 'electrncs'),
(1600, 'play gym', 'toys'),
(1601, 'lamaze', 'toys'),
(1666, 'harry potter', 'dvd');
```

步骤 3 使用 MERGE INTO 语句将源表 products 的数据合并至目标表 newproducts。

```
vastbase=# MERGE INTO newproducts np
USING products p
ON (np.product_id = p.product_id )
WHEN MATCHED THEN
  UPDATE SET np.product_name = p.product_name, np.category = p.category
WHEN NOT MATCHED THEN
  INSERT VALUES (p.product_id, p.product_name, p.category) ;
```

上述语句中使用的参数说明，请见表格。更多信息，请参见 11.16.92MERGE INTO。

参数	说明	举例
INTO 子句	指定需要更新或插入数据的目标表。 • 目标表支持指定别名。 • 目标表支持复制表，但复制表不能带有含 volatile 函数的列（如自增列）。	取值：newproducts np 说明：名为 newproducts，别名为 np 的目标表。
USING 子句	指定源表。源表支持指定别名。 目标表是复制表时，源表也需要是复制表。	取值：products p 名为 products，别名为 p 的源表。
ON 子句	指定目标表和源表的关联条件。 关联条件中的字段不支持更新。	取值：np.product_id = p.product_id 说明：指定的关联条件为，目标表 newproducts 的 product_id 字段和源表 products 的 product_id 字段相等。
WHEN MATCHED 子句	当源表和目标表中数据针对关联条	取值：WHEN MATCHED THEN UPDATE SET

参数	说明	举例
	件可以匹配上时，选择 WHEN MATCHED 子句进行 UPDATE 操作。 • 仅支持指定一个 WHEN MATCHED 子句。 • WHEN MATCHED 子句可缺省，缺省时，对于满足 ON 子句条件的行，不进行任何操作。 • 若目标表中存在分布列，则该列不支持更新。	np.product_name = p.product_name, np.category = p.category 说明：当满足 ON 子句条件时，将目标表 newproducts 的 product_name、category 字段的值替换为源表 products 相对应字段的值。
WHEN NOT MATCHED 子句	当源表和目标表中数据针对关联条件无法匹配时，选择 WHEN NOT MATCHED 子句进行 INSERT 操作。 • 仅支持指定一个 WHEN NOT MATCHED 子句。 • WHEN NOT MATCHED 子句可缺省。 • 不支持 INSERT 子句中包含多个 VALUES。 • WHEN MATCHED 和 WHEN NOT MATCHED 子句顺序可以交换，可以缺省其中一个，但不能同时缺省。	取值：WHEN NOT MATCHED THEN INSERT VALUES (p.product_id, p.product_name, p.category) 说明：将源表 products 中，不满足 ON 子句条件的行插入目标表 products。

步骤 4 查询合并后的目标表 newproducts。

```
vastbase=# SELECT * FROM newproducts;
```

返回信息如下：

```
 product_id |  product_name  | category
------------+----------------+-----------
      1501 | vivitar 35mm   | electrncs
      1502 | olympus camera | electrncs
      1666 | harry potter   | toys
      1600 | play gym       | toys
      1601 | lamaze         | toys
      1700 | wait interface | books
(6 rows)
```

7.6. 深层复制

数据导入后，如果需要修改表的分区键、或者将行存表改列存、添加 PCK（Partial Cluster Key）约束等场景下，可以使用深层复制的方式对表进行调整。深层复制是指重新创建表，然后使用批量插入填充表的过程。

Vastbase 提供了三种深层复制的方式供用户选择。

7.6.1. 使用 CREATE TABLE 执行深层复制

该方法使用 CREATE TABLE 语句创建原始表的副本，将原始表的数据填充至副本并重命名副本，完成原始表的复制。

在创建新表时，可以指定表以及列属性，比如主键。

操作步骤

执行如下步骤对表 customer_t 进行深层复制。

步骤 1 使用 CREATE TABLE 语句创建表 customer_t 的副本 customer_t_copy。

```
vastbase=# CREATE TABLE customer_t_copy
( c_customer_sk           integer,
  c_customer_id           char(5),
  c_first_name            char(6),
  c_last_name             char(8)
) ;
```

步骤 2 使用 INSERT INTO...SELECT 语句向副本填充原始表中的数据。

```
vastbase=# INSERT INTO customer_t_copy (SELECT * FROM customer_t);
```

步骤 3 删除原始表。

```
vastbase=# DROP TABLE customer_t;
```

步骤 4 使用 ALTER TABLE 语句将副本重命名为原始表名称。

```
vastbase=# ALTER TABLE customer_t_copy RENAME TO customer_t;
```

7.6.2. 使用 CREATE TABLE LIKE 执行深层复制

该方法使用 CREATE TABLE LIKE 语句创建原始表的副本，将原始表的数据填充至副本并重命名副本，完成原始表的复制。该方法不继承父表的主键属性，您可以使用 ALTER TABLE 语句来添加它们。

操作步骤

步骤 1 使用 CREATE TABLE LIKE 语句创建表 customer_t 的副本 customer_t_copy。

```
vastbase=# CREATE TABLE customer_t_copy (LIKE customer_t);
```

步骤 2 使用 INSERT INTO...SELECT 语句向副本填充原始表中的数据。

```
vastbase=# INSERT INTO customer_t_copy (SELECT * FROM customer_t);
```

步骤 3 删除原始表。

```
vastbase=# DROP TABLE customer_t;
```

步骤 4 使用 ALTER TABLE 语句将副本重命名为原始表名称。

```
vastbase=# ALTER TABLE customer_t_copy RENAME TO customer_t;
```

7.6.3.通过创建临时表并截断原始表来执行深层复制

该方法使用 CREATE TABLE ... AS 语句创建原始表的临时表，然后截断原始表并从临时表填充它完成原始表的深层复制。

在新建表需要保留父表的主键属性，或如果父表具有依赖项的情况下，建议使用此方法。

操作步骤

步骤 1 使用 CREATE TABLE AS 语句创建表 customer_t 的临时表副本 customer_t_temp。

```
vastbase=# CREATE TABLE customer_t_temp AS SELECT * FROM customer_t;
```

说明

与使用永久表相比，使用临时表可以提高性能，但存在丢失数据的风险。临时表只在当前会话可见，本会话结束后将自动删除。如果数据丢失是不可接受的，请使用永久表。

步骤 2 截断当前表 customer_t。

```
vastbase=# TRUNCATE customer_t;
```

步骤 3 使用 INSERT INTO...SELECT 语句从副本中向原始表中填充数据。

```
vastbase=# INSERT INTO customer_t (SELECT * FROM customer_t_temp);
```

步骤 4 删除临时表副本 customer_t_temp。

```
vastbase=# DROP TABLE customer_t_temp;
```

7.7.分析表

执行计划生成器需要使用表的统计信息，以生成最有效的查询执行计划，提高查询性能。因此数据导入完成后，建议执行 ANALYZE 语句生成最新的表统计信息。统计结果存储在系统表 PG_STATISTIC 中。

分析表

ANALYZE 支持的表类型有行/列存表。ANALYZE 同时也支持对本地表的指定列进行信息统计。下面以表的 ANALYZE 为例，更多关于 ANALYZE 的信息，请参见 11.16.26ANALYZE | ANALYSE。

步骤 1 更新表统计信息。

以表 product_info 为例，ANALYZE 命令如下：

```
vastbase=# ANALYZE product_info;
ANALYZE
```

表自动分析

Vastbase 提供了 GUC 参数 autovacuum 用于控制数据库自动清理功能的启动。

autovacuum 设置为 on 时，系统定时启动 autovacuum 线程来进行表自动分析，如果表中数据量发生较大变化达到阈值时，会触发表自动分析，即 autoanalyze。

- 对于空表而言，当表中插入数据的行数大于参数 autovacuum_analyze_threshold 时，会触发表自动进行 ANALYZE。
- 对于表中已有数据的情况，阈值设定为参数表达式：autovacuum_vacuum_threshold+autovacuum_vacuum_scale_factor*reltuples，其中 reltuples 是表的总行数。

autovacuum 自动清理功能的生效还依赖于下面两个 GUC 参数：

- track_counts 参数需要设置为 on，表示开启收集收据库统计数据功能。
- autovacuum_max_workers 参数需要大于 0，该参数表示能同时运行的自动清理线程的最大数量。

须知

- autoanalyze 只支持默认采样方式，不支持百分比采样方式。
- 多列统计信息仅支持百分比采样，因此 autoanalyze 不收集多列统计信息。
- autoanalyze 支持行存表和列存表，不支持外表、临时表、unlogged 表和 toast 表。

7.8. 对表执行 VACUUM

如果导入过程中，进行了大量的更新或删除行时，应运行 VACUUM FULL 命令，然后运行 ANALYZE 命令。大量的更新和删除操作，会产生大量的磁盘页面碎片，从而逐渐降低查询的效率。VACUUM FULL 可以将磁盘页面碎片恢复并交还操作系统。

步骤 1 对表执行 VACUUM FULL。

以表 product_info 为例，VACUUM FULL 命令如下：

```
vastbase=# VACUUM FULL product_info
VACUUM
```

7.9. 管理并发写入操作

7.9.1. 事务隔离说明

Vastbase 基于 MVCC（多版本并发控制）并结合两阶段锁的方式进行事务管理，其特点是读写之间不阻塞。SELECT 是纯读操作，UPDATE 和 DELETE 是读写操作。

- 读写操作和纯读操作之间并不会发生冲突，读写操作之间也不会发生冲突。每个并发事务在事务开始时创建事务快照，并发事务之间不能检测到对方的更改。
  - 读已提交隔离级别中，如果事务 T1 提交后，事务 T2 就可以看到事务 T1 更改的结果。
  - 可重复读级别中，如果事务 T1 提交事务前事务 T2 开始执行，则事务 T1 提交后，事务 T2 依旧看不到事务 T1 更改的结果，保证了一个事务开始后，查询的结果前后一致，不受其他事务的影响。
- 读写操作，支持的是行级锁，不同的事务可以并发更新同一个表，只有更新同一行时才需等待，后发生的事务会等待先发生的事务提交后，再执行更新操作。
  - READ COMMITTED：读已提交隔离级别，事务只能读到已提交的数据而不会读到未提交的数据，这是缺省值。
  - REPEATABLE READ： 事务只能读到事务开始之前已提交的数据，不能读到未提交的数据以及事务执行期间其它并发事务提交的修改。

7.9.2. 写入和读写操作

关于写入和读写操作的命令：

- INSERT，可向表中插入一行或多行数据。
- UPDATE，可修改表中现有数据。
- DELETE，可删除表中现有数据。
- COPY，导入数据。

INSERT 和 COPY 是纯写入的操作。并发写入操作，需要等待，对同一个表的操作，当事务 T1 的 INSERT 或 COPY 未解除锁定时，事务 T2 的 INSERT 或 COPY 需等待，事务 T1 解除锁定时，事务 T2 正常继续。

UPDATE 和 DELETE 是读写操作（先查询出要操作的行）。UPDATE 和 DELETE 执行前需要先查询数据，由于并发事务彼此不可见，所以 UPDATE 和 DELETE 操作是读取事务发生前提交的数据的快照。写入操作，是行级锁，当事务 T1 和事务 T2 并发更新同一行时，后发生的事务 T2 会等待，根据设置的等待时长，若超时事务 T1 未提交则事务 T2 执行失败；当事务 T1 和事务 T2 并发更新的行不同时，事务 T1 和事务 2 都会执行成功。

7.9.3.并发写入事务的潜在死锁情况

只要事务涉及多个表的或者同一个表相同行的更新时，同时运行的事务就可能在同时尝试写入时变为死锁状态。事务会在提交或回滚时一次性解除其所有锁定，而不会逐一放弃锁定。 例如，假设事务 T1 和 T2 在大致相同的时间开始：

- ❖ 如果 T1 开始对表 A 进行写入且 T2 开始对表 B 进行写入，则两个事务均可继续而不会发生冲突；但是，如果 T1 完成了对表 A 的写入操作并需要开始对表 B 进行写入，此时操作的行数正好与 T2 一致，它将无法继续，因为 T2 仍保持对表 B 对应行的锁定，此时 T2 开始更新表 A 中与 T1 相同的行数，此时也将无法继续，产生死锁，在锁等待超时内，前面事务提交释放锁，后面的事务可以继续执行更新，等待时间超时后，事务抛错，有一个事务退出。
- ❖ 如果 T1，T2 都对表 A 进行写入，此时 T1 更新 1-5 行的数据，T2 更新 6-10 行的数据，两个事务不会发生冲突，但是，如果 T1 完成后开始对表 A 的 6-10 行数据进行更新，T2 完成后开始更新 1-5 行的数据，此时两个事务无法继续，在锁等待超时内，前面事务提交释放锁，后面的事务可以继续执行更新，等待时间超时后，事务抛错，有一个事务退出。

7.9.4.并发写入示例

本章节以表 test 为例，分别介绍相同表的 INSERT 和 DELETE 并发，相同表的并发 INSERT，相同表的并发 UPDATE，以及数据导入和查询的并发的执行详情。

```
CREATE TABLE test(id int, name char(50), address varchar(255));
```

7.9.4.1. 相同表的 INSERT 和 DELETE 并发

事务 T1：

```
START TRANSACTION;
INSERT INTO test VALUES(1,'test1','test123');
COMMIT;
```

事务 T2：

```
START TRANSACTION;
DELETE test WHERE NAME='test1';
COMMIT;
```

场景 1：

开启事务 T1，不提交的同时开启事务 T2，事务 T1 执行 INSERT 完成后，执行事务 T2 的 DELETE，此时显示 DELETE 0，由于事务 T1 未提交，事务 2 看不到事务插入的数据；

场景 2：

- ❖ READ COMMITTED 级别

  执行命令 BEGIN TRANSACTION ISOLATION LEVEL REPEATABLE READ;开启事务 T1，不提交的同时开启事务 T2。在事务 T1 和 T2 中各执行 select txid_current();建立快照。事务 T1 执

行 INSERT 完成后，提交事务 T1，事务 T2 再执行 DELETE 语句时，此时显示 DELETE 1，事务 T1 提交完成后，事务 T2 可以看到此条数据，可以删除成功。

- REPEATABLE READ 级别

  执行命令 BEGIN TRANSACTION ISOLATION LEVEL REPEATABLE READ;开启事务 T1，不提交的同时开启事务 T2。在事务 T1 和 T2 中各执行 select txid_current();建立快照。事务 T1 执行 INSERT 完成后，提交事务 T1，事务 T2 再执行 DELETE 语句时，此时显示 DELETE 0，事务 T1 提交完成后，事务 T2 依旧看不到事务 T1 的数据，一个事务中前后查询到的数据是一致的。

7.9.4.2. 相同表的并发 INSERT

事务 T1：

```
START TRANSACTION;
INSERT INTO test VALUES(2,'test2','test123');
COMMIT;
```

事务 T2：

```
START TRANSACTION;
INSERT INTO test VALUES(3,'test3','test123');
COMMIT;
```

场景 1：

开启事务 T1，不提交的同时开启事务 T2，事务 T1 执行 INSERT 完成后，执行事务 T2 的 INSERT 语句，可以执行成功，读已提交和可重复读隔离级别下，此时在事务 T1 中执行 SELECT 语句，看不到事务 T2 中插入的数据，事务 T2 中执行查询语句看不到事务 T1 中插入的数据。

场景 2：

- READ COMMITTED 级别

  执行命令 BEGIN TRANSACTION ISOLATION LEVEL REPEATABLE READ;开启事务 T1，不提交的同时开启事务 T2。在事务 T1 和 T2 中各执行 select txid_current();建立快照。事务 T1 执行 INSERT 完成后直接提交，事务 T2 中执行 INSERT 语句后执行查询语句，可以看到事务 T1 中插入的数据。

- REPEATABLE READ 级别

  执行命令 BEGIN TRANSACTION ISOLATION LEVEL REPEATABLE READ;开启事务 T1，不提交的同时开启事务 T2。在事务 T1 和 T2 中各执行 select txid_current();建立快照。事务 T1 执行 INSERT 完成后直接提交，事务 T2 中执行 INSERT 语句后执行查询语句，看不到事务 T1 中插入的数据。

7.9.4.3. 相同表的并发 UPDATE

事务 T1：

```
START TRANSACTION;
UPDATE test SET address='test1234' WHERE name='test1';
COMMIT;
```

事务 T2：

```
START TRANSACTION;
UPDATE test SET address='test1234' WHERE name='test2';
COMMIT;
```

事务 T3：

```
START TRANSACTION;
UPDATE test SET address='test1234' WHERE name='test1';
COMMIT;
```

场景 1：

开启事务 T1，不提交的同时开启事务 T2，事务 T1 开始执行 UPDATE，事务 T2 开始执行 UPDATE，事务 T1 和事务 T2 都执行成功。更新不同行时，更新操作拿的是行级锁，不会发生冲突，两个事务都可以执行成功。

场景 2：

开启事务 T1，不提交的同时开启事务 T3，事务 T1 开始执行 UPDATE，事务 T3 开始执行 UPDATE，事务 T1 执行成功，事务 T3 等待超时后会出错。更新相同行时，事务 T1 未提交时，未释放锁，导致事务 T3 执行不成功。

7.9.4.4. 数据导入和查询的并发

事务 T1：

```
START TRANSACTION;
COPY test FROM '...';
COMMIT;
```

事务 T2：

```
START TRANSACTION;
SELECT * FROM test;
COMMIT;
```

场景 1：

开启事务 T1，不提交的同时开启事务 T2，事务 T1 开始执行 COPY，事务 T2 开始执行 SELECT，事务 T1 和事务 T2 都执行成功。事务 T2 中查询看不到事务 T1 新 COPY 进来的数据。

场景 2：

- READ COMMITTED 级别

  执行命令 BEGIN TRANSACTION ISOLATION LEVEL REPEATABLE READ;开启事务 T1，不提交的同时开启事务 T2。在事务 T1 和 T2 中各执行 select txid_current();建立快照。事务 T1 开始执行 COPY，然后提交，事务 T2 查询，可以看到事务 T1 中 COPY 的数据。

- REPEATABLE READ 级别

执行命令 BEGIN TRANSACTION ISOLATION LEVEL REPEATABLE READ;开启事务 T1，不提交的同时开启事务 T2。在事务 T1 和 T2 中各执行 select txid_current();建立快照。事务 T1 开始执行 COPY，然后提交，事务 T2 查询，看不到事务 T1 中 COPY 的数据。

8. 导出数据

8.1. 使用 vb_dump 和 vb_dumpall 命令导出数据

8.1.1. 概述

Vastbase 提供的 vb_dump 和 vb_dumpall 工具，能够帮助用户导出需要的数据库对象或其相关信息。通过导入工具将导出的数据信息导入至需要的数据库，可以完成数据库信息的迁移。vb_dump 支持导出单个数据库或其内的对象，而 vb_dumpall 支持导出 Vastbase 中所有数据库或各库的公共全局对象。详细的使用场景见表 8-1。

表 8-1. 适用场景

适用场景	支持的导出粒度	支持的导出格式	配套的导入方法
导出单个数据库	8.1.2.1 导出数据库。 • 导出全量信息。 使用导出的全量信息可以创建一个与当前库相同的数据库，且库中数据也与当前库相同。 • 仅导出库中所有对象的定义，包含库定义、函数定义、模式定义、表定义、索引定义和存储过程定义等。 使用导出的对象定义，可以快速创建一个相同的数据库，但是库中并无原数据库的数据。 • 仅导出数据。	• 纯文本格式 • 自定义归档格式 • 目录归档格式 • tar 归档格式	• 纯文本格式数据文件导入请参见 7.3 使用 vsql 元命令导入数据。 • 自定义归档格式、目录归档格式和 tar 归档格式数据文件导入请参见 7.4 使用 vb_restore 命令导入数据。
	8.1.2.2 导出模式。 • 导出模式的全量信息。 • 仅导出模式中数据。 • 仅导出对象的定义，包含表定义、存储过程定义和索引定义等。		

适用场景	支持的导出粒度	支持的导出格式	配套的导入方法
	8.1.2.3 导出表。 • 导出表的全量信息。 • 仅导出表中数据。 • 仅导出表的定义。		
导出所有数据库	8.1.3.1 导出所有数据库。 • 导出全量信息。 使用导出的全量信息可以创建与 Vastbase 相同的一个 Vastbase，拥有相同数据库和公共全局对象，且库中数据也与当前各库相同。 • 仅导出各数据库中的对象定义，包含表空间、库定义、函数定义、模式定义、表定义、索引定义和存储过程定义等。 使用导出的对象定义，可以快速创建与 Vastbase 相同的一个 Vastbase，拥有相同的数据库和表空间，但是库中并无原数据库的数据。 • 仅导出数据。	纯文本格式	数据文件导入请参见 7.3 使用 vsql 元命令导入数据。
	8.1.3.2 导出全局对象 • 仅导出表空间信息。 • 仅导出角色信息。 • 导出角色与表空间。		

vb_dump 和 vb_dumpall 通过-U 指定执行导出的用户帐户。如果当前使用的帐户不具备导出所要求的权限时，会无法导出数据。此时，可在导出命令中设置--role 参数来指定具备权限的角色。在执行命令后，vb_dump 和 vb_dumpall 会使用--role 参数指定的角色，完成导出动作。可使用该功能的场景请参见表 8-1。

vb_dump 和 vb_dumpall 通过对导出的数据文件加密，导入时对加密的数据文件进行解密，可以防止数据信息泄露，为数据库的安全提供保证。

vb_dump 和 vb_dumpall 工具在进行数据导出时，其他用户可以访问 Vastbase 数据库（读或写）。

vb_dump 和 vb_dumpall 工具支持导出完整一致的数据。例如，T1 时刻启动 vb_dump 导出 A 数据库，或者启动 vb_dumpall 导出 Vastbase 数据库，那么导出数据结果将会是 T1 时刻 A 数据库或者该 Vastbase 数据库的数据状态，T1 时刻之后对 A 数据库或 Vastbase 数据库的修改不会被导出。

注意事项

- 禁止修改导出的文件和内容，否则可能无法恢复成功。
- 如果数据库中包含的对象数量（数据表、视图、索引）在 50 万以上，为了提高性能且避免出现内存问题，建议修改配置文件 postgres.conf 增加或修改如下参数(如果参数值大于如下建议值，则无需设置）。

```
max_prepared_transactions = 1000
max_locks_per_transaction = 512
```

- 为了保证数据一致性和完整性，导出工具会对需要转储的表设置共享锁。如果表在别的事务中设置了共享锁，vb_dump 和 vb_dumpall 会等待锁释放后锁定表。如果无法在指定时间内锁定某个表，转储会失败。用户可以通过指定--lock-wait-timeout 选项，自定义等待锁超时时间。
- 由于 vb_dumpall 读取所有数据库中的表，因此必须以 Vastbase 管理员身份进行连接，才能导出完整文件。在使用 vsql 执行脚本文件导入时，同样需要管理员权限，以便添加用户和组，以及创建数据库。

8.1.2.导出单个数据库

8.1.2.1. 导出数据库

Vastbase 支持使用 vb_dump 工具导出某个数据库级的内容，包含数据库的数据和所有对象定义。可根据需要自定义导出如下信息：

- 导出数据库全量信息，包含数据和所有对象定义。

  使用导出的全量信息可以创建一个与当前库相同的数据库，且库中数据也与当前库相同。

- 仅导出所有对象定义，包括：库定义、函数定义、模式定义、表定义、索引定义和存储过程定义等。

  使用导出的对象定义，可以快速创建一个相同的数据库，但是库中并无原数据库的数据。

- 仅导出数据，不包含所有对象定义。

操作步骤

步骤 1 以操作系统用户 vastbase 登录数据库主节点。

步骤 2 使用 vb_dump 导出 vastbase 数据库。

```
vb_dump -W Bigdata@123 -U jack -f /home/vastbase/backup/vastbase_backup.tar -p 5432 vastbase
-F t
```

表 8-2. 常用参数说明

参数	参数说明	举例
-U	连接数据库的用户名。 说明 不指定连接数据库的用户名时，默认以安装时创建的初始系统管理员连接。	-U jack
-W	指定用户连接的密码。 • 如果主机的认证策略是 trust，则不会对数据库管理员进行密码验证，即无需输入-W 选项； • 如果没有-W 选项，并且不是数据库管理员，会提示用户输入密码。	-W Bigdata@123
-f	将导出文件发送至指定目录文件夹。如果这里省略，则使用标准输出。	-f /home/vastbase/backup/postgres_backup.tar
-p	指定服务器所监听的 TCP 端口或本地 Unix 域套接字后缀，以确保连接。	-p 5432
dbname	需要导出的数据库名称	vastbase
-F	选择导出文件格式。-F 参数值如下： • p：纯文本格式 • c：自定义归档 • d：目录归档格式 • t：tar 归档格式	-F t

其他参数说明请参见《工具参考》中“服务端工具 > vb_dump”章节。

示例

示例一：执行 vb_dump，导出 vastbase 数据库全量信息，并对导出文件进行压缩，导出文件格式为 sql 文本格式。

```
vb_dump -W Bigdata@123 -f /home/vastbase/backup/postgres_backup.sql -p 5432 vastbase -Z 8 -F p
vb_dump[port='5432'][vastbase][2017-07-21 15:36:13]: dump database vastbase successfully
vb_dump[port='5432'][vastbase][2017-07-21 15:36:13]: total time: 3793  ms
```

示例二：执行 vb_dump，仅导出 vastbase 数据库中的数据，不包含数据库对象定义，导出文件格式为自定义归档格式。

```
vb_dump -W Bigdata@123 -f /home/vastbase/backup/vastbase_data_backup.dmp -p 5432 vastbase -a -F c
vb_dump[port='5432'][vastbase][2017-07-21 15:36:13]: dump database vastbase successfully
vb_dump[port='5432'][vastbase][2017-07-21 15:36:13]: total time: 3793  ms
```

示例三：执行 vb_dump，仅导出 vastbase 数据库所有对象的定义，导出文件格式为 sql 文本格式。

```
--导出前，表 nation 有数据
vastbase=# select n_nationkey,n_name,n_regionkey from nation limit 3;
 n_nationkey |         n_name          | n_regionkey
-------------+-------------------------+-------------
           0 | ALGERIA                 |           0
           3 | CANADA                  |           1
          11 | IRAQ                    |           4
(3 rows)

vb_dump -W Bigdata@123 -f /home/vastbase/backup/vastbase_def_backup.sql -p 5432 vastbase -s -F p
vb_dump[port='5432'][vastbase][2017-07-20 15:04:14]: dump database vastbase successfully
vb_dump[port='5432'][vastbase][2017-07-20 15:04:14]: total time: 472 ms
```

示例四：执行 vb_dump，仅导出 vastbase 数据库的所有对象的定义，导出文件格式为文本格式，并对导出文件进行加密。

```
vb_dump -W Bigdata@123 -f /home/vastbase/backup/vastbase_def_backup.sql -p 5432 vastbase
--with-encryption AES128 --with-key 1234567812345678 -s -F p
vb_dump[port='5432'][vastbase][2018-11-14 11:25:18]: dump database vastbase successfully
vb_dump[port='5432'][vastbase][2018-11-14 11:25:18]: total time: 1161  ms
```

8.1.2.2. 导出模式

Vastbase 目前支持使用 vb_dump 工具导出模式级的内容，包含模式的数据和定义。用户可通过灵活的自定义方式导出模式内容，不仅支持选定一个模式或多个模式的导出，还支持排除一个模式或者多个模式的导出。可根据需要自定义导出如下信息：

- 导出模式全量信息，包含数据和对象定义。
- 仅导出数据，即模式包含表中的数据，不包含对象定义。
- 仅导出模式对象定义，包括：表定义、存储过程定义和索引定义等。

操作步骤

步骤 1 以操作系统用户 vastbase 登录数据库主节点。

步骤 2 使用 vb_dump 同时导出 hr 和 public 模式。

```
vb_dump -W Bigdata@123 -U jack -f /home/vastbase/backup/MPPDB_schema_backup -p 5432
human_resource -n hr -n public -F d
```

表 8-3. 常用参数说明

参数	参数说明	举例
-U	连接数据库的用户名。	-U jack
-W	指定用户连接的密码。	-W Bigdata@123

参数	参数说明	举例
	• 如果主机的认证策略是 trust，则不会对数据库管理员进行密码验证，即无需输入-W 选项。 • 如果没有-W 选项，并且不是数据库管理员，会提示用户输入密码。	
-f	将导出文件发送至指定目录文件夹。如果这里省略，则使用标准输出。	-f /home/vastbase/backup/MPPDB_schema_backup
-p	指定服务器所监听的 TCP 端口或本地 Unix 域套接字后缀，以确保连接。	-p 5432
dbname	需要导出的数据库名称	human_resource
-n	只导出与模式名称匹配的模式，此选项包括模式本身和所有它包含的对象。 • 单个模式：-n *schemaname* • 多个模式：多次输入-n *schemaname*	• 单个模式：-n hr • 多个模式：-n hr -n public
-F	选择导出文件格式。-F 参数值如下： • p：纯文本格式 • c：自定义归档 • d：目录归档格式 • t：tar 归档格式	-F d

其他参数说明请参见《工具参考》中“服务端工具 > vb_dump”章节。

示例

示例一：执行 vb_dump，导出 hr 模式全量信息，并对导出文件进行压缩，导出文件格式为文本格式。

```
vb_dump -W Bigdata@123 -f /home/vastbase/backup/MPPDB_schema_backup.sql -p 5432 human_resource -n hr
-Z 6 -F p
vb_dump[port='5432'][human_resource][2017-07-21 16:05:55]: dump database human_resource
successfully
vb_dump[port='5432'][human_resource][2017-07-21 16:05:55]: total time: 2425  ms
```

示例二：执行 vb_dump，仅导出 hr 模式的数据，导出文件格式为 tar 归档格式。

```
vb_dump -W Bigdata@123 -f /home/vastbase/backup/MPPDB_schema_data_backup.tar -p 5432 human_resource
-n hr -a -F t
vb_dump[port='5432'][human_resource][2018-11-14 15:07:16]: dump database human_resource
successfully
vb_dump[port='5432'][human_resource][2018-11-14 15:07:16]: total time: 1865  ms
```

示例三：执行 vb_dump，仅导出 hr 模式的定义，导出文件格式为目录归档格式。

```
vb_dump -W Bigdata@123 -f /home/vastbase/backup/MPPDB_schema_def_backup -p 5432 human_resource -n hr
-s -F d
vb_dump[port='5432'][human_resource][2018-11-14 15:11:34]: dump database human_resource
successfully
vb_dump[port='5432'][human_resource][2018-11-14 15:11:34]: total time: 1652  ms
```

示例四：执行 vb_dump，导出 human_resource 数据库时，排除 hr 模式，导出文件格式为自定义归档格式。

```
vb_dump -W Bigdata@123 -f /home/vastbase/backup/MPPDB_schema_backup.dmp -p 5432 human_resource -N hr
-F c
vb_dump[port='5432'][human_resource][2017-07-21 16:06:31]: dump database human_resource
successfully
vb_dump[port='5432'][human_resource][2017-07-21 16:06:31]: total time: 2522  ms
```

示例五：执行 vb_dump，同时导出 hr 和 public 模式，且仅导出模式定义，并对导出文件进行加密，导出文件格式为 tar 归档格式。

```
vb_dump -W Bigdata@123 -f /home/vastbase/backup/MPPDB_schema_backup1.tar -p 5432 human_resource -n
hr -n public -s --with-encryption AES128 --with-key 1234567812345678 -F t
vb_dump[port='5432'][human_resource][2017-07-21 16:07:16]: dump database human_resource
successfully
vb_dump[port='5432'][human_resource][2017-07-21 16:07:16]: total time: 2132  ms
```

示例六：执行 vb_dump，导出 human_resource 数据库时，排除 hr 和 public 模式，导出文件格式为自定义归档格式。

```
vb_dump -W Bigdata@123 -f /home/vastbase/backup/MPPDB_schema_backup2.dmp -p 5432 human_resource -N
hr -N public -F c
vb_dump[port='5432'][human_resource][2017-07-21 16:07:55]: dump database human_resource
successfully
vb_dump[port='5432'][human_resource][2017-07-21 16:07:55]: total time: 2296  ms
```

示例七：执行 vb_dump，导出 public 模式下所有表（视图、序列和外表）和 hr 模式中 staffs 表，包含数据和表定义，导出文件格式为自定义归档格式。

```
vb_dump -W Bigdata@123 -f /home/vastbase/backup/MPPDB_backup3.dmp -p 5432 human_resource -t public.*
-t hr.staffs -F c
vb_dump[port='5432'][human_resource][2018-12-13 09:40:24]: dump database human_resource
successfully
vb_dump[port='5432'][human_resource][2018-12-13 09:40:24]: total time: 896  ms
```

8.1.2.3. 导出表

Vastbase 支持使用 vb_dump 工具导出表级的内容，包含表定义和表数据。视图、序列和外表属于特殊的表。用户可通过灵活的自定义方式导出表内容，不仅支持选定一个表或多个表的导出，还支持排除一个表或者多个表的导出。可根据需要自定义导出如下信息：

- 导出表全量信息，包含表数据和表定义。

- ❖ 仅导出数据，不包含表定义。
- ❖ 仅导出表定义。

操作步骤

步骤 1 以操作系统用户 vastbase 登录数据库主节点。

步骤 2 使用 vb_dump 同时导出指定表 hr.staffs 和 hr.employments。

```
vb_dump -W Bigdata@123 -U jack -f /home/vastbase/backup/MPPDB_table_backup -p    5432
human_resource -t hr.staffs -t hr.employments -F d
```

表 8-4. 常用参数说明

参数	参数说明	举例
-U	连接数据库的用户名。	-U jack
-W	指定用户连接的密码。 • 如果主机的认证策略是 trust，则不会对数据库管理员进行密码验证，即无需输入-W 选项。 • 如果没有-W 选项，并且不是数据库管理员，会提示用户输入密码。	-W Bigdata@123
-f	将导出文件发送至指定目录文件夹。如果这里省略，则使用标准输出。	-f /home/vastbase/backup/MPPDB_table_backup
-p	指定服务器所监听的 TCP 端口或本地 Unix 域套接字后缀，以确保连接。	-p 5432
dbname	需要导出的数据库名称	human_resource
-t	指定导出的表（或视图、序列、外表），可以使用多个-t 选项来选择多个表，也可以使用通配符指定多个表对象。当使用通配符指定多个表对象时，注意给 pattern 打引号，防止 shell 扩展通配符。 • 单个表：-t *schema.table* • 多个表：多次输入-t *schema.table*	• 单个表：-t hr.staffs • 多个表：-t hr.staffs -t hr.employments
-F	选择导出文件格式。-F 参数值如下： • p：纯文本格式	-F d

参数	参数说明	举例
	• c：自定义归档 • d：目录归档格式 • t：tar 归档格式	

其他参数说明请参见《工具参考》中“服务端工具 > vb_dump”章节。

示例

示例一：执行 vb_dump，导出表 hr.staffs 的定义和数据，并对导出文件进行压缩，导出文件格式为文本格式。

```
vb_dump -W Bigdata@123 -f /home/vastbase/backup/MPPDB_table_backup.sql -p 5432 human_resource -t
hr.staffs -Z 6 -F p
vb_dump[port='5432'][human_resource][2017-07-21 17:05:10]: dump database human_resource
successfully
vb_dump[port='5432'][human_resource][2017-07-21 17:05:10]: total time: 3116  ms
```

示例二：执行 vb_dump，只导出表 hr.staffs 的数据，导出文件格式为 tar 归档格式。

```
vb_dump -W Bigdata@123 -f /home/vastbase/backup/MPPDB_table_data_backup.tar -p 5432 human_resource
-t hr.staffs -a -F t
vb_dump[port='5432'][human_resource][2017-07-21 17:04:26]: dump database human_resource
successfully
vb_dump[port='5432'][human_resource][2017-07-21 17:04:26]: total time: 2570  ms
```

示例三：执行 vb_dump，导出表 hr.staffs 的定义，导出文件格式为目录归档格式。

```
vb_dump -W Bigdata@123 -f /home/vastbase/backup/MPPDB_table_def_backup -p 5432 human_resource -t
hr.staffs -s -F d
vb_dump[port='5432'][human_resource][2017-07-21 17:03:09]: dump database human_resource
successfully
vb_dump[port='5432'][human_resource][2017-07-21 17:03:09]: total time: 2297  ms
```

示例四：执行 vb_dump，不导出表 hr.staffs，导出文件格式为自定义归档格式。

```
vb_dump -W Bigdata@123 -f /home/vastbase/backup/MPPDB_table_backup4.dmp -p 5432 human_resource -T
hr.staffs -F c
vb_dump[port='5432'][human_resource][2017-07-21 17:14:11]: dump database human_resource
successfully
vb_dump[port='5432'][human_resource][2017-07-21 17:14:11]: total time: 2450  ms
```

示例五：执行 vb_dump，同时导出两个表 hr.staffs 和 hr.employments，导出文件格式为文本格式。

```
vb_dump -W Bigdata@123 -f /home/vastbase/backup/MPPDB_table_backup1.sql -p 5432 human_resource -t
hr.staffs -t hr.employments -F p
vb_dump[port='5432'][human_resource][2017-07-21 17:19:42]: dump database human_resource
successfully
vb_dump[port='5432'][human_resource][2017-07-21 17:19:42]: total time: 2414  ms
```

示例六：执行 vb_dump，导出时，排除两个表 hr.staffs 和 hr.employments，导出文件格式为文本格式。

```
vb_dump -W Bigdata@123 -f /home/vastbase/backup/MPPDB_table_backup2.sql -p 5432 human_resource -T
hr.staffs -T hr.employments -F p
vb_dump[port='5432'][human_resource][2017-07-21 17:21:02]: dump database human_resource
```

```
successfully
vb_dump[port='5432'][human_resource][2017-07-21 17:21:02]: total time: 3165  ms
```

示例七：执行 vb_dump，导出表 hr.staffs 的定义和数据，只导出表 hr.employments 的定义，导出文件格式为 tar 归档格式。

```
vb_dump -W Bigdata@123 -f /home/vastbase/backup/MPPDB_table_backup3.tar -p 5432 human_resource -t
hr.staffs -t hr.employments --exclude-table-data hr.employments -F t
vb_dump[port='5432'][human_resource][2018-11-14 11:32:02]: dump database human_resource
successfully
vb_dump[port='5432'][human_resource][2018-11-14 11:32:02]: total time: 1645  ms
```

示例八：执行 vb_dump，导出表 hr.staffs 的定义和数据，并对导出文件进行加密，导出文件格式为文本格式。

```
vb_dump -W Bigdata@123 -f /home/vastbase/backup/MPPDB_table_backup4.sql -p 5432 human_resource -t
hr.staffs --with-encryption AES128 --with-key 1212121212121212 -F p
vb_dump[port='5432'][human_resource][2018-11-14 11:35:30]: dump database human_resource
successfully
vb_dump[port='5432'][human_resource][2018-11-14 11:35:30]: total time: 6708  ms
```

示例九：执行 vb_dump，导出 public 模式下所有表（包括视图、序列和外表）和 hr 模式中 staffs 表，包含数据和表定义，导出文件格式为自定义归档格式。

```
vb_dump -W Bigdata@123 -f /home/vastbase/backup/MPPDB_table_backup5.dmp -p 5432 human_resource -t
public.* -t hr.staffs -F c
vb_dump[port='5432'][human_resource][2018-12-13 09:40:24]: dump database human_resource
successfully
vb_dump[port='5432'][human_resource][2018-12-13 09:40:24]: total time: 896  ms
```

示例十： 执行 vb_dump，仅导出依赖于 t1 模式下的 test1 表对象的视图信息，导出文件格式为目录归档格式。

```
vb_dump -W Bigdata@123 -U jack -f /home/vastbase/backup/MPPDB_view_backup6 -p 5432 human_resource -t
t1.test1 --include-depend-objs --exclude-self -F d
vb_dump[port='5432'][jack][2018-11-14 17:21:18]: dump database human_resource successfully
vb_dump[port='5432'][jack][2018-11-14 17:21:23]: total time: 4239  ms
```

8.1.3.导出所有数据库

8.1.3.1. 导出所有数据库

Vastbase 支持使用 vb_dumpall 工具导出所有数据库的全量信息，包含 Vastbase 中每个数据库信息和公共的全局对象信息。可根据需要自定义导出如下信息：

- ❖ 导出所有数据库全量信息，包含 Vastbase 中每个数据库信息和公共的全局对象信息（包含角色和表空间信息）。

  使用导出的全量信息可以创建与 Vastbase 相同的一个 Vastbase，拥有相同数据库和公共全局对象，且库中数据也与当前各库相同。

- ❖ 仅导出数据，即导出每个数据库中的数据，且不包含所有对象定义和公共的全局对象信息。
- ❖ 仅导出所有对象定义，包括：表空间、库定义、函数定义、模式定义、表定义、索引定义和存储过程定义等。

使用导出的对象定义，可以快速创建与 Vastbase 相同的一个 Vastbase，拥有相同的数据库和表空间，但是库中并无原数据库的数据。

操作步骤

步骤 1 以操作系统用户 vastbase 登录数据库主节点。

步骤 2 使用 vb_dumpall 一次导出所有数据库信息。

```
vb_dumpall -W Bigdata@123 -U vastbase -f /home/vastbase/backup/MPPDB_backup.sql -p 5432
```

表 8-5. 常用参数说明

参数	参数说明	举例
-U	连接数据库的用户名，需要是 Vastbase 管理员用户。	-U vastbase
-W	指定用户连接的密码。 • 如果主机的认证策略是 trust，则不会对数据库管理员进行密码验证，即无需输入-W 选项； • 如果没有-W 选项，并且不是数据库管理员，会提示用户输入密码。	-W Bigdata@123
-f	将导出文件发送至指定目录文件夹。如果这里省略，则使用标准输出。	-f /home/vastbase/backup/MPPDB_backup.sql
-p	指定服务器所监听的 TCP 端口或本地 Unix 域套接字后缀，以确保连接。	-p 5432

其他参数说明请参见《工具参考》中“服务端工具 > vb_dumpall”章节。

示例

示例一：执行 vb_dumpall，导出所有数据库全量信息（vastbase 用户为管理员用户），导出文件为文本格式。执行命令后，会有很长的打印信息，最终出现 total time 即代表执行成功。示例中将不体现中间的打印信息。

```
vb_dumpall -W Bigdata@123 -U vastbase -f /home/vastbase/backup/MPPDB_backup.sql -p 5432
vb_dumpall[port='5432'][2017-07-21 15:57:31]: dumpall operation successful
vb_dumpall[port='5432'][2017-07-21 15:57:31]: total time: 9627  ms
```

示例二：执行 vb_dumpall，仅导出所有数据库定义（vastbase 用户为管理员用户），导出文件为文本格式。执行命令后，会有很长的打印信息，最终出现 total time 即代表执行成功。示例中将不体现中间的打印信息。

```
vb_dumpall -W Bigdata@123 -U vastbase -f /home/vastbase/backup/MPPDB_backup.sql -p 5432 -s
vb_dumpall[port='5432'][2018-11-14 11:28:14]: dumpall operation successful
vb_dumpall[port='5432'][2018-11-14 11:28:14]: total time: 4147  ms
```

示例三：执行 vb_dumpall，仅导出所有数据库中数据，并对导出文件进行加密，导出文件为文本格式。执行命令后，会有很长的打印信息，最终出现 total time 即代表执行成功。示例中将不体现中间的打印信息。

```
vb_dumpall -f /home/vastbase/backup/MPPDB_backup.sql -p 5432 -a --with-encryption AES128 --with-key
1234567812345678
vb_dumpall[port='5432'][2018-11-14 11:32:26]: dumpall operation successful
vb_dumpall[port='5432'][2018-11-14 11:23:26]: total time: 4147  ms
```

8.1.3.2. 导出全局对象

Vastbase 支持使用 vb_dumpall 工具导出所有数据库公共的全局对象，包含数据库用户和组，表空间及属性（例如：适用于数据库整体的访问权限）信息。

操作步骤

步骤 1 以操作系统用户 vastbase 登录数据库主节点。

步骤 2 使用 vb_dumpall 导出表空间对象信息。

```
vb_dumpall -W Bigdata@123 -U vastbase -f /home/vastbase/backup/MPPDB_tablespace.sql -p 5432
-t
```

表 8-6. 常用参数说明

参数	参数说明	举例
-U	连接数据库的用户名，需要是 Vastbase 管理员用户。	-U vastbase
-W	指定用户连接的密码。 • 如果主机的认证策略是 trust，则不会对数据库管理员进行密码验证，即无需输入-W 选项； • 如果没有-W 选项，并且不是数据库管理员，会提示用户输入密码。	-W Bigdata@123
-f	将导出文件发送至指定目录文件夹。如果这里省略，则使用标准输出。	-f /home/vastbase/backup/MPPDB_tablespace.sql

参数	参数说明	举例
-p	指定服务器所监听的 TCP 端口或本地 Unix 域套接字后缀，以确保连接。	-p 5432
-t	或者--tablespaces-only，只转储表空间，不转储数据库或角色。	-

其他参数说明请参见《工具参考》中“服务端工具 > vb_dumpall”章节。

示例

示例一：执行 vb_dumpall，导出所有数据库的公共全局表空间信息和用户信息（vastbase 用户为管理员用户），导出文件为文本格式。

```
vb_dumpall -W Bigdata@123 -U vastbase -f /home/vastbase/backup/MPPDB_globals.sql -p 5432 -g
vb_dumpall[port='5432'][2018-11-14 19:06:24]: dumpall operation successful
vb_dumpall[port='5432'][2018-11-14 19:06:24]: total time: 1150  ms
```

示例二： 执行 vb_dumpall，导出所有数据库的公共全局表空间信息（vastbase 用户为管理员用户），并对导出文件进行加密，导出文件为文本格式。

```
vb_dumpall -W Bigdata@123 -U vastbase -f /home/vastbase/backup/MPPDB_tablespace.sql -p 5432 -t
--with-encryption AES128 --with-key 1212121212121212
vb_dumpall[port='5432'][2018-11-14 19:00:58]: dumpall operation successful
vb_dumpall[port='5432'][2018-11-14 19:00:58]: total time: 186  ms
```

示例三：执行 vb_dumpall，导出所有数据库的公共全局用户信息（vastbase 用户为管理员用户），导出文件为文本格式。

```
vb_dumpall -W Bigdata@123 -U vastbase -f /home/vastbase/backup/MPPDB_user.sql -p 5432 -r
vb_dumpall[port='5432'][2018-11-14 19:03:18]: dumpall operation successful
vb_dumpall[port='5432'][2018-11-14 19:03:18]: total time: 162  ms
```

9. 性能调优

9.1. 总体调优思路

Vastbase 的总体性能调优思路为性能瓶颈点分析、关键参数调整以及 SQL 调优。在调优过程中，通过系统资源、吞吐量、负载等因素来帮助定位和分析性能问题，使系统性能达到可接受的范围。

Vastbase 性能调优过程需要综合考虑多方面因素，因此，调优人员应对系统软件架构、软硬件配置、数据库配置参数、并发控制、查询处理和数据库应用有广泛而深刻的理解。

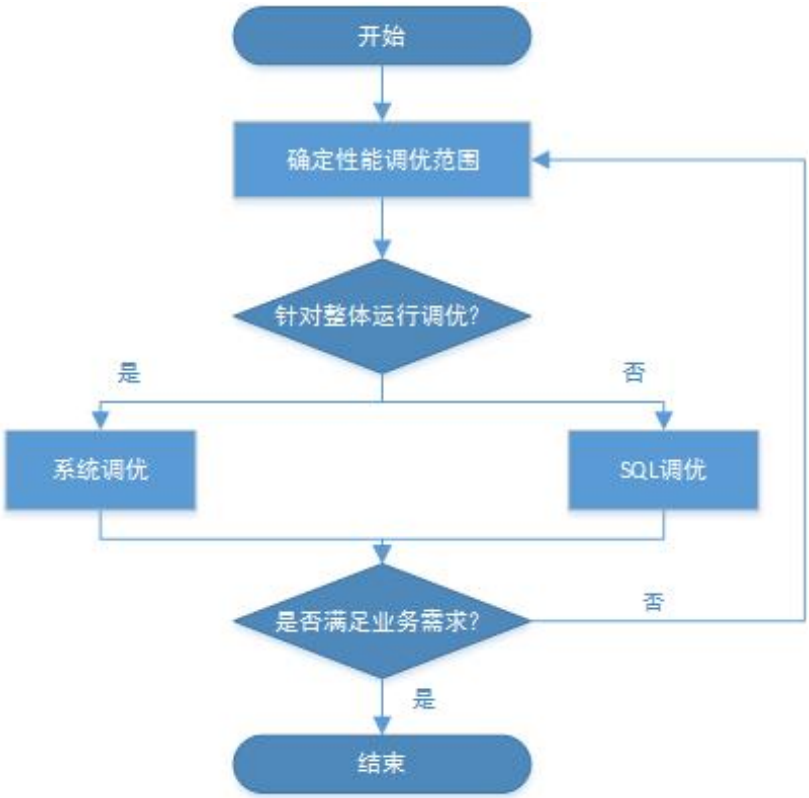
须知

性能调优过程有时候需要重启 Vastbase，可能会中断当前业务。因此，业务上线后，当性能调优操作需要重启 Vastbase 时，操作窗口时间需向管理部门提出申请，经批准后方可执行。

调优流程

调优流程如图 3-2 所示。

图 3-2 Vastbase 性能调优流程



调优各阶段说明，如表 9-1 所示。

表 9-1. Vastbase 性能调优流程说明

阶段	描述
9.2 确定性能调优范围	获取 Vastbase 节点的 CPU、内存、I/O 和网络资源使用情况，确认这些资源是否已被充分利用，是否存在瓶颈点。
9.3 系统调优指南	进行操作系统级以及数据库系统级的调优，更充分地利用机器的 CPU、内存、I/O 和网络资源，避免资源冲突，提升整个系统查询的吞吐量。
9.3 系统调优指南	审视业务所用 SQL 语句是否存在可优化空间，包括：

阶段	描述
	• 通过 ANALYZE 语句生成表统计信息：ANALYZE 语句可收集与数据库中表内容相关的统计信息，统计结果存储在系统表 PG_STATISTIC 中。执行计划生成器会使用这些统计数据，以确定最有效的执行计划。 • 分析执行计划：EXPLAIN 语句可显示 SQL 语句的执行计划，EXPLAIN PERFORMANCE 语句可显示 SQL 语句中各算子的执行时间。 • 查找问题根因并进行调优：通过分析执行计划，找到可能存在的原因，进行针对性的调优，通常为调整数据库级 SQL 调优参数。 • 编写更优的 SQL：介绍一些复杂查询中的中间临时数据缓存、结果集缓存、结果集合并等场景中的更优 SQL 语法。

9.2. 确定性能调优范围

数据库性能调优通常发生在用户对业务的执行效率不满意，期望通过调优加快业务执行的情况下。正如“性能因素”小节所述，数据库性能受影响因素多，从而性能调优是一项复杂的工程，有些时候无法系统性地说明和解释，而是依赖于 DBA 的经验判断。尽管如此，此处还是期望能尽量系统性的对性能调优方法加以说明，方便应用开发人员和刚接触 Vastbase 的 DBA 参考。

性能因素

多个性能因素会影响数据库性能，了解这些因素可以帮助定位和分析性能问题。

- 系统资源

  数据库性能在很大程度上依赖于磁盘的 I/O 和内存使用情况。为了准确设置性能指标，用户需要了解 Vastbase 部署硬件的基本性能。CPU，硬盘，磁盘控制器，内存和网络接口等这些硬件性能将显著影响数据库的运行速度。

- 负载

  负载等于数据库系统的需求总量，它会随着时间变化。总体负载包含用户查询，应用程序，并行作业，事务以及数据库随时传递的系统命令。比如：多用户在执行多个查询时会提高负载。负载会显著地影响数据库的性能。了解工作负载高峰期可以帮助用户更合理地利用系统资源，更有效地完成系统任务。

- 吞吐量

  使用系统的吞吐量来定义处理数据的整体能力。数据库的吞吐量以每秒的查询次数、每秒的处理事务数量或平均响应时间来测量。数据库的处理能力与底层系统（磁盘 I/O，CPU 速度，存储器带宽等）有密切的关系，所以当设置数据库吞吐量目标时，需要提前了解硬件的性能。

- 竞争

  竞争是指两组或多组负载组件尝试使用冲突的方式使用系统的情况。比如，多条查询视图同一时间更新相同的数据，或者多个大量的负载争夺系统资源。随着竞争的增加，吞吐量下降。

- 优化

  数据库优化可以影响到整个系统的性能。在执行 SQL 制定、数据库配置参数、表设计、数据分布等操作时，启用数据库查询优化器打造最有效的执行计划。

调优范围确定

性能调优主要通过查看 Vastbase 节点的 CPU、内存、I/O 和网络这些硬件资源的使用情况，确认这些资源是否已被充分利用，是否存在瓶颈点，然后针对性调优。

- 如果某个资源已达瓶颈，则：
  a. 检查关键的操作系统参数和数据库参数是否合理设置，进行 9.3 系统调优指南
  b. 通过查询最耗时的 SQL 语句、跑不出来的 SQL 语句，找出耗资源的 SQL，进行 9.4SQL 调优指南
- 如果所有资源均未达瓶颈，则表明性能仍有提升潜力。可以查询最耗时的 SQL 语句，或者跑不出来的 SQL 语句，进行针对性的 9.4SQL 调优指南

9.2.1. 硬件瓶颈点分析

获取 Vastbase 节点的 CPU、内存、I/O 和网络资源使用情况，确认这些资源是否已被充分利用，是否存在瓶颈点。

9.2.1.1. CPU

通过 top 命令查看 Vastbase 内节点 CPU 使用情况，分析是否存在由于 CPU 负载过高导致的性能瓶颈。

查看 CPU 状况

查询服务器 CPU 的使用情况主要通过以下方式：

在所有存储节点，逐一执行 **top** 命令，查看 CPU 占用情况。执行该命令后，按“1”键，可查看每个 CPU 核的使用率。

```
top - 17:05:04 up 32 days, 20:34,  5 users,  load average: 0.02, 0.02, 0.00
Tasks: 124 total,   1 running, 123 sleeping,   0 stopped,   0 zombie
Cpu0  :  0.0%us,  0.3%sy,  0.0%ni, 69.7%id,  0.0%wa,  0.0%hi,  0.0%si,  0.0%st
Cpu1  :  0.3%us,  0.3%sy,  0.0%ni, 69.3%id,  0.0%wa,  0.0%hi,  0.0%si,  0.0%st
Cpu2  :  0.3%us,  0.3%sy,  0.0%ni, 69.3%id,  0.0%wa,  0.0%hi,  0.0%si,  0.0%st
Cpu3  :  0.3%us,  0.3%sy,  0.0%ni, 69.3%id,  0.0%wa,  0.0%hi,  0.0%si,  0.0%st
```

```
Mem:   8038844k total,  7165272k used,   873572k free,   530444k buffers
Swap:  4192924k total,     4920k used,  4188004k free,  4742904k cached

  PID USER      PR  NI  VIRT  RES  SHR S %CPU %MEM    TIME+  COMMAND

35184 vastbase  20   0  822m 421m 128m S    0  5.4   5:28.15 vastbase
    1 root      20   0 13592  820  784 S    0  0.0   1:16.62 init
```

分析时，请主要关注进程占用的 CPU 利用率。

其中，统计信息中“us”表示用户空间占用 CPU 百分比，“sy”表示内核空间占用 CPU 百分比，“id”表示空闲 CPU 百分比。如果“id”低于 10%，即表明 CPU 负载较高，可尝试通过降低本节点任务量等手段降低 CPU 负载。

性能参数分析

步骤 1 使用“top -H”命令查看 CPU，显示内容如下所示。

```
   14 root      20   0     0    0    0 S    0  0.0   0:16.41 events/3
top - 14:22:49 up 5 days, 21:51,  2 users,  load average: 0.08, 0.08, 0.06
Tasks: 312 total,   1 running, 311 sleeping,   0 stopped,   0 zombie
Cpu(s):  1.3%us,  0.7%sy,  0.0%ni, 95.0%id,  2.4%wa,  0.5%hi,  0.2%si,  0.0%st
Mem:   8038844k total,  5317668k used,  2721176k free,   180268k buffers
Swap:  4192924k total,        0k used,  4192924k free,  2886860k cached

  PID USER      PR  NI  VIRT  RES  SHR S %CPU %MEM    TIME+  COMMAND

 3105 root      20   0 50492  11m 2708 S    3  0.1  22:22.56 acc-snf

 4015 gdm       20   0  232m  23m  11m S    0  0.3  11:34.70 gdm-simple-gree
51001 vastbase  20   0 12140 1484  948 R    0  0.0   0:00.94 top

54885 vastbase  20   0  615m 396m 116m S    0  5.1   0:09.44 vastbase


    1 root      20   0 13592  944  792 S    0  0.0   0:08.54 init
```

步骤 2 根据查询结果中“Cpu(s)”分析是系统 CPU（sy）还是用户 CPU（us）占用过高。

- 如果是系统 CPU 占用过高，需要查找异常系统进程进行处理。
- 如果是“USER”为 vastbase 的 vastbase 进程 CPU 占用过高，请根据目前运行的业务查询内容，对业务 SQL 进行优化。请根据以下步骤，并结合当前正在运行的业务特征进行分析，是否该程序处于死循环逻辑。

a. 使用“top -H -p pid”查找进程内占用的 CPU 百分比较高的线程，进行分析。

```
top -H -p 54952
```

查询结果如下所示，top 中可以看到占用 CPU 很高的线程，下面以线程 54775 为主，分析其为何占用 CPU 过高。

```
top - 14:23:27 up 5 days, 21:52,  2 users,  load average: 0.04, 0.07, 0.05
Tasks:  13 total,   0 running,  13 sleeping,   0 stopped,   0 zombie
Cpu(s):  0.9%us,  0.4%sy,  0.0%ni, 97.3%id,  1.1%wa,  0.2%hi,  0.1%si,
0.0%st
```

```
Mem:   8038844k total,  5322180k used,  2716664k free,   180316k buffers
Swap:  4192924k total,        0k used,  4192924k free,  2889860k cached

  PID USER      PR  NI  VIRT  RES  SHR S %CPU %MEM    TIME+  COMMAND
54775 vastbase  20   0  684m 424m 131m S    0  5.4   0:00.32 vastbase
54951 vastbase  20   0  684m 424m 131m S    0  5.4   0:00.84 vastbase
54732 vastbase  20   0  684m 424m 131m S    0  5.4   0:00.24 vastbase
54758 vastbase  20   0  684m 424m 131m S    0  5.4   0:00.00 vastbase
54759 vastbase  20   0  684m 424m 131m S    0  5.4   0:00.02 vastbase
54773 vastbase  20   0  684m 424m 131m S    0  5.4   0:02.79 vastbase
54780 vastbase  20   0  684m 424m 131m S    0  5.4   0:00.04 vastbase
54781 vastbase  20   0  684m 424m 131m S    0  5.4   0:00.21 vastbase
54782 vastbase  20   0  684m 424m 131m S    0  5.4   0:00.02 vastbase
54798 vastbase  20   0  684m 424m 131m S    0  5.4   0:16.70 vastbase
54952 vastbase  20   0  684m 424m 131m S    0  5.4   0:07.51 vastbase
54953 vastbase  20   0  684m 424m 131m S    0  5.4   0:00.81 vastbase
54954 vastbase  20   0  684m 424m 131m S    0  5.4   0:06.54 vastbase
```

b. 使用“gstack ”查看进程内各线程的函数调用栈。查找上一步骤中占用 CPU 较高的线程 ID 对应的线程号。

```
gstack  54954
```

查询结果如下所示，其中线程 ID54775 对应线程号是 10。

```
192.168.0.11:~ # gstack 54954
Thread 10 (Thread 0x7f95a5fff710 (LWP 54775)):
#0  0x00007f95c41d63c6 in poll () from /lib64/libc.so.6
#1  0x0000000000d3d2d3 in WaitLatchOrSocket(Latch volatile*, int, int, long)
()
#2  0x000000000095ed25 in XLogPageRead(XLogRecPtr*, int, bool, bool) ()
#3  0x000000000095f6dd in ReadRecord(XLogRecPtr*, int, bool) ()
#4  0x000000000096aef0 in StartupXLOG() ()
#5  0x0000000000d5607a in StartupProcessMain() ()
#6  0x00000000009e19f9 in AuxiliaryProcessMain(int, char**) ()
#7  0x0000000000d50135 in SubPostmasterMain(int, char**) ()
#8  0x0000000000d504ec in MainStarterThreadFunc(void*) ()
#9  0x00007f95c79b85f0 in start_thread () from /lib64/libpthread.so.0
#10 0x00007f95c41df84d in clone () from /lib64/libc.so.6
#11 0x0000000000000000 in ?? ()
```

9.2.1.2. 内存

通过 top 命令查看 Vastbase 节点内存使用情况，分析是否存在由于内存占用率过高导致的性能瓶颈。

查看内存状况

查询服务器内存的使用情况主要通过以下方式：

执行 **top** 命令，查看内存占用情况。执行该命令后，按“Shift+M”键，可按照内存大小排序。

```
top - 11:38:26 up 2 days, 17:59, 10 users,  load average: 0.01, 0.05, 0.15
Tasks: 685 total,   1 running, 684 sleeping,   0 stopped,   0 zombie
```

```
%Cpu(s):  0.2 us,  0.2 sy,  0.0 ni, 99.7 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
KiB Mem : 19740646+total, 23503420 free, 15947100 used, 15795595+buff/cache
KiB Swap:  8242172 total,  8242172 free,       0 used. 13366219+avail Mem

  PID USER PR  NI    VIRT    RES    SHR S  %CPU %MEM     TIME+ COMMAND
29838 vastbase  20   0 1373104 456904 175248 S   3.6  0.2  98:53.16 vastbase
27789 vastbase  20   0  150732   4136   3216 S   0.0  0.0   0:00.00 vsql
45659 vastbase  20   0  117164   4052   1860 S   0.0  0.0   0:00.24 bash
 8087 vastbase  20   0  117164   4000   1848 S   0.0  0.0   0:00.05 bash
27459 vastbase  20   0  117160   4000   1848 S   0.0  0.0   0:00.04 bash
33619 vastbase  20   0  117120   3852   1740 S   0.0  0.0   0:00.04 bash
27282 vastbase  20   0  117120   3840   1728 S   0.0  0.0   0:00.03 bash
 9923 vastbase  20   0  158064   2932   1612 R   0.3  0.0   0:00.04 top
```

分析时，请主要关注 vastbase 进程占用的内存百分比（%MEM）、整系统的剩余内存。

显示信息中的主要属性解释如下：

- total：物理内存总量。
- used：已使用的物理内存总量。
- free：空闲内存总量。
- buffers：进程使用的虚拟内存总量。
- %MEM：进程占用的内存百分比。
- VIRT：进程使用的虚拟内存总量，VIRT=SWAP+RES。
- SWAP：进程使用的虚拟内存中已被换出到交换分区的量。
- RES：进程使用的虚拟内存中未被换出的量。
- SHR：共享内存大小。

性能参数分析

步骤 1 以 root 用户执行“free”命令查看 cache 的占用情况。

```
free
```

查询结果如下所示：

```
             total       used       free     shared    buffers     cached
Mem:       8038844    6336184    1702660          0     375896    2880912
-/+ buffers/cache:    3079376    4959468
Swap:      4192924          0    4192924
```

步骤 2 若“cache”占用过高，请清除系统缓存。

步骤 3 若用户内存占用过高，需查看执行计划，重点分析以下内容。

- 是否有不合理的 join 顺序。例如，多表关联时，执行计划中优先关联的两表的中间结果集比较大，导致最终执行代价比较大。

9.2.1.3. I/O

通过 iostat、pidstat 命令或 Vastbase 健康检查工具查看 Vastbase 内节点 I/O 繁忙度和吞吐量，分析是否存在由于 I/O 导致的性能瓶颈。

查看 I/O 状况

查询服务器 I/O 的方法主要有以下两种方式：

- 使用 iostat 命令查看 I/O 情况。此命令主要关注单个硬盘的 I/O 使用率和每秒读取、写入的数量。

```
iostat -xm 1  //1 为间隔时间
Device:         rrqm/s   wrqm/s     r/s     w/s    rMB/s    wMB/s avgrq-sz avgqu-sz   await
r_await w_await  svctm  %util
sdc               0.01   519.62    2.35   44.10     0.31     2.17   109.66     0.68   14.62    2.80
15.25   0.31   1.42
sdb               0.01   515.95    5.84   44.78     0.89     2.16   123.51     0.72   14.19    1.55
15.84   0.31   1.55
sdd               0.02   519.93    2.36   43.91     0.32     2.17   110.16     0.65   14.12    2.58
14.74   0.30   1.38
sde               0.02   520.26    2.34   45.17     0.31     2.18   107.46     0.80   16.86    2.92
17.58   0.34   1.63
sda              12.07    15.72    3.97    5.01     0.07     0.08    34.11     0.28   30.64   10.11
46.92   0.98   0.88
```

“rMB/s”为每秒读取的 MB 数，“wMB/s”为每秒写入的 MB 数，“%util”为硬盘使用率。

- 使用 pidstat 命令查看 I/O 情况。此命令主要关注单个进程每秒读取、写入的数量。

```
pidstat -d 1 10  //1 为间隔时间，10 表示查看占用 I/O 最多的 Top10 进程
03:17:12 PM   UID      PID   kB_rd/s   kB_wr/s kB_ccwr/s  Command
03:17:13 PM  1006    36134      0.00  59436.00      0.00  vastbase
```

“kB_rd/s”为每秒读取的 kB 数，“kB_wr/s”为每秒写入的 kB 数。

性能参数分析

步骤 1 检查磁盘空间使用率，建议不要超过 60%。

```
df -T
```

步骤 2 若 I/O 持续过高，建议尝试以下方式降低 I/O。

- 降低并发数。
- 对造成 IO 较大的查询进行调优，如由于页面回收问题造成的 IO 消耗，建议使用单表 Vacuum 操作降低其 IO 开销。

```
vacuum  tablename;
```

说明

建议用户在系统空闲时进行 VACUUM FULL 操作，VACUUM FULL 操作会造成短时间内 I/O 负载重，反而不利于降低 I/O。

9.2.1.4. 网络

通过 sar 或 ifconfig 命令查看 Vastbase 内节点网络使用情况，分析是否存在由于网络导致的性能瓶颈。

查看网络状况

查询服务器网络状况的方法主要有以下两种方式：

- 使用 root 用户身份登录服务器，执行如下命令查看服务器网络连接。

```
SIA1000056771:~ # ifconfig
eth0      Link encap:Ethernet  HWaddr 28:6E:D4:86:7D:D5
          inet addr:10.180.123.163  Bcast:10.180.123.255  Mask:255.255.254.0
          inet6 addr: fe80::2a6e:d4ff:fe86:7dd5/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:5669314 errors:0 dropped:0 overruns:0 frame:0
          TX packets:4955927 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:508077795 (484.5 Mb)  TX bytes:818004366 (780.1 Mb)

lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
          inet6 addr: ::1/128 Scope:Host
          UP LOOPBACK RUNNING  MTU:16436  Metric:1
          RX packets:711938 errors:0 dropped:0 overruns:0 frame:0
          TX packets:711938 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:164158862 (156.5 Mb)  TX bytes:164158862 (156.5 Mb)
```

  - “errors”表示收包错误的总数量。
  - “dropped”表示数据包已经进入了 Ring Buffer，但是由于内存不够等系统原因，导致在拷贝到内存的过程中被丢弃的总数量。
  - “overruns”表示 Ring Buffer 队列中被丢弃的报文数目，由于 Ring Buffer(aka Driver Queue)传输的 IO 大于 kernel 能够处理的 IO 导致。

  分析时，如果发现上述三个值持续增长，则表示网络负载过大或者存在网卡、内存等硬件故障。

- 使用 sar 命令查看服务器网络连接。

```
sar -n DEV 1  //1为间隔时间
Average: IFACE  rxpck/s  txpck/s   rxkB/s   txkB/s rxcmp/s txcmp/s rxmcst/s %ifutil
Average:    lo  1926.94  1926.94  25573.92  25573.92    0.00    0.00    0.00    0.00
Average:  A1-0     0.00     0.00     0.00     0.00    0.00    0.00    0.00    0.00
Average:  A1-1     0.00     0.00     0.00     0.00    0.00    0.00    0.00    0.00
Average:  NIC0     5.17     1.48     0.44     0.92    0.00    0.00    0.00    0.00
Average:  NIC1     0.00     0.00     0.00     0.00    0.00    0.00    0.00    0.00
Average:  A0-0  8173.06 92420.66  97102.22 133305.09    0.00    0.00    0.00    0.00
Average:  A0-1 11431.37  9373.06 156950.45   494.40    0.00    0.00    0.00    0.00
Average:  B3-0     0.00     0.00     0.00     0.00    0.00    0.00    0.00    0.00
Average:  B3-1     0.00     0.00     0.00     0.00    0.00    0.00    0.00    0.00
```

  “rxkB/s”为每秒接收的 kB 数，“txkB/s”为每秒发送的 kB 数。

分析时，请主要关注每个网卡的传输量和是否达到传输上限。

检查完后，按“Ctrl+Z”键退出查看。

9.2.2.查询最耗性能的 SQL

系统中有些 SQL 语句运行了很长时间还没有结束，这些语句会消耗很多的系统性能，请根据本章内容查询长时间运行的 SQL 语句。

操作步骤

步骤 1 以操作系统用户 vastbase 登录数据库节点。

步骤 2 使用如下命令连接数据库。

```
vsql -d vastbase -p 5432
```

vastbase 为需要连接的数据库名称，5432 为数据库节点的端口号。

连接成功后，系统显示类似如下信息：

```
vsql ((Vastbase 1.0 build 290d125f) compiled at 2020-05-08 02:59:43 commit 2143 last mr 131
Non-SSL connection (SSL connection is recommended when requiring high-security)
Type "help" for help.

vastbase=#
```

步骤 3 查询系统中长时间运行的查询语句。

```
SELECT current_timestamp - query_start AS runtime, datname, usename, query FROM
pg_stat_activity where state != 'idle' ORDER BY 1 desc;
```

查询后会按执行时间从长到短顺序返回查询语句列表，第一条结果就是当前系统中执行时间最长的查询语句。返回结果中包含了系统调用的 SQL 语句和用户执行 SQL 语句，请根据实际找到用户执行时间长的语句。

若当前系统较为繁忙，可以通过限制 current_timestamp - query_start 大于某一阈值来查看执行时间超过此阈值的查询语句。

```
SELECT query FROM pg_stat_activity WHERE current_timestamp - query_start > interval '1 days';
```

步骤 4 设置参数 track_activities 为 on。

```
SET track_activities = on;
```

当此参数为 on 时，数据库系统才会收集当前活动查询的运行信息。

步骤 5 查看正在运行的查询语句。

以查看视图 pg_stat_activity 为例：

```
SELECT datname, usename, state FROM pg_stat_activity;
 datname  | usename | state  |
----------+---------+--------+
 vastbase |  vastbase  | idle   |
 vastbase |  vastbase  | active |
(2 rows)
```

如果 state 字段显示为 idle，则表明此连接处于空闲，等待用户输入命令。

如果仅需要查看非空闲的查询语句，则使用如下命令查看：

```
SELECT datname, usename, state FROM pg_stat_activity WHERE state != 'idle';
```

步骤 6 分析长时间运行的查询语句状态。

- 若查询语句处于正常状态，则等待其执行完毕。
- 若查询语句阻塞，则通过如下命令查看当前处于阻塞状态的查询语句：

```
SELECT datname, usename, state, query FROM pg_stat_activity WHERE waiting = true;
```

查询结果中包含了当前被阻塞的查询语句，该查询语句所请求的锁资源可能被其他会话持有，正在等待持有会话释放锁资源。

说明

只有当查询阻塞在系统内部锁资源时，waiting 字段才显示为 true。尽管等待锁资源是数据库系统最常见的阻塞行为，但是在某些场景下查询也会阻塞在等待其他系统资源上，例如写文件、定时器等。但是这种情况的查询阻塞，不会在视图 pg_stat_activity 中体现。

9.2.3.分析作业是否被阻塞

数据库系统运行时，在某些业务场景下查询语句会被阻塞，导致语句运行时间过长，可以强制结束有问题的会话。

操作步骤

步骤 1 以操作系统用户 vastbase 登录数据库节点。

步骤 2 使用如下命令连接数据库。

```
vsql -d vastbase -p 5432
```

vastbase 为需要连接的数据库名称，5432 为数据库节点的端口号。

连接成功后，系统显示类似如下信息：

```
vsql ((Vastbase 1.0 build 290d125f) compiled at 2020-05-08 02:59:43 commit 2143 last mr 131
Non-SSL connection (SSL connection is recommended when requiring high-security)
Type "help" for help.

vastbase=#
```

步骤 3 查看阻塞的查询语句及阻塞查询的表、模式信息。

```
SELECT w.query as waiting_query,
w.pid as w_pid,
w.usename as w_user,
l.query as locking_query,
l.pid as l_pid,
l.usename as l_user,
t.schemaname || '.' || t.relname as tablename
from pg_stat_activity w join pg_locks l1 on w.pid = l1.pid
and not l1.granted join pg_locks l2 on l1.relation = l2.relation
and l2.granted join pg_stat_activity l on l2.pid = l.pid join pg_stat_user_tables t on
l1.relation = t.relid
where w.waiting;
```

该查询返回线程 ID、用户信息、查询状态，以及导致阻塞的表、模式信息。

步骤 4 使用如下命令结束相应的会话。其中，139834762094352 为线程 ID。

```
SELECT PG_TERMINATE_BACKEND(139834762094352);
```

显示类似如下信息，表示结束会话成功。

```
 PG_TERMINATE_BACKEND
----------------------
 t
(1 row)
```

显示类似如下信息，表示用户正在尝试结束当前会话，此时仅会重连会话，而不是结束会话。

```
FATAL:  terminating connection due to administrator command
FATAL:  terminating connection due to administrator command
The connection to the server was lost. Attempting reset: Succeeded.
```

说明

vsql 客户端使用 PG_TERMINATE_BACKEND 函数终止本会话后台线程时，客户端不会退出而是自动重连。

9.3. 系统调优指南

系统调优是指进行操作系统级以及数据库系统级的调优，更充分地利用机器的 CPU、内存、I/O 和网络资源，避免资源冲突，提升整个系统查询的吞吐量。

9.3.1. 操作系统参数调优

在性能调优过程中，可以根据实际业务情况修改关键操作系统(OS)配置参数，以提升 Vastbase 数据库的性能。

前提条件

需要用户使用 gs_check 检查操作系统参数结果是否和建议值保持一致，如果不一致，用户可根据实际业务情况去手动修改。

内存相关参数设置

配置“sysctl.conf”文件，修改内存相关参数 vm.extfrag_threshold 为 1000（参考值），如果文件中没有内存相关参数，可以手动添加。

```
vim /etc/sysctl.conf
```

修改完成后，请执行如下命令，使参数生效。

```
sysctl -p
```

网络相关参数设置

❖ 配置“sysctl.conf”文件，修改网络相关参数，如果文件中没有网络相关参数，可以手动添加。详细说明请参见表 9-2。

```
vim /etc/sysctl.conf
```

在修改完成后，请执行如下命令，使参数生效。

```
sysctl -p
```

表 9-2. 网络相关参数

参数名	参考值	说明
net.ipv4.tcp_timestamps	1	表示开启 TCP 连接中 TIME-WAIT sockets 的快速回收，默认为 0，表示关闭，1 表示打开。
net.ipv4.tcp_mem	94500000 915000000 927000000	第一个数字表示，当 tcp 使用的 page 少于 94500000 时，kernel 不对其进行任何的干预。 第二个数字表示，当 tcp 使用的 page 超过 915000000 时，kernel 会进入“memory pressure”压力模式。 第三个数字表示，当 tcp 使用的 pages 超过 927000000 时，就会报：Out of socket memory。
net.ipv4.tcp_max_orphans	3276800	最大孤儿套接字（orphan sockets）数。
net.ipv4.tcp_fin_timeout	60	表示系统默认的 TIMEOUT 时间。
net.ipv4.ip_local_port_range	26000 65535	TCP 和 UDP 能够使用的 port 段。

❖ 设置 10GE 网卡最大传输单元（MTU），使用 ifconfig 命令设置。10GE 网卡推荐设置为 8192，可提升网络带宽利用率。

须知

在数据库节点配置参数设置 comm_tcp_mode=false 时，一定要保证各个 Vastbase 的 MTU 大小相等，否则可能有通信问题。

示例：

```
#ifconfig ethx mtu 8192
#ifconfig ethx
ethx   Link encap:Ethernet  HWaddr XX:XX:XX:XX:XX:XX
inet addr:xxx.xxx.xxx.xxx  Bcast:xxx.xxx.xxx.xxx  Mask:xxx.xxx.xxx.0
```

```
inet6 addr: fxxx::9xxx:bxxx:xxxa:1d18/64 Scope:Link
UP BROADCAST RUNNING MULTICAST  MTU:8192  Metric:1
RX packets:179849803 errors:0 dropped:0 overruns:0 frame:0
TX packets:40492292 errors:0 dropped:0 overruns:0 carrier:0
collisions:0 txqueuelen:1000
RX bytes:17952090386 (17120.4 Mb)  TX bytes:171359670290 (163421.3 Mb)
```

说明

- ethx 为 10GE 数据库内部使用的业务网卡。
- 第一条命令设置 MTU，第二条命令验证是否设置成功，粗体部分为 MTU 的值。
- 需使用 root 用户设置。

❖ 设置 10GE 网卡接收(rx)、发送队列(tx)长度，使用 ethtool 工具设置。10GE 网卡推荐设置为 4096，可提升网络带宽利用率。

示例：

```
# ethtool -G ethx rx 4096 tx 4096
# ethtool -g ethx
Ring parameters for ethx:
Pre-set maximums:
RX:           4096
RX Mini:       0
RX Jumbo:      0
TX:           4096
Current hardware settings:
RX:           4096
RX Mini:       0
RX Jumbo:      0
TX:           4096
```

说明

- ethx 为 10GE 数据库内部使用的业务网卡。
- 第一条命令设置网卡接收、发送队列长度，第二条命令验证是否设置成功，示例的输出表示设置成功。
- 需使用 root 用户设置。

I/O 相关参数设置

设置 hugepage 属性。在启动文件配置中追加下列参数，来关闭透明大页。

```
echo 'never' > /sys/kernel/mm/transparent_hugepage/enabled
echo 'never' > /sys/kernel/mm/transparent_hugepage/defrag
```

修改完成后，请执行如下命令，查看结果。

```
#grep Huge /proc/meminfo
AnonHugePages:         0 kB     ---透明大页关闭，则显示 0
ShmemHugePages:        0 kB
HugePages_Total:       0
HugePages_Free:        0
HugePages_Rsvd:        0
HugePages_Surp:        0
Hugepagesize:     524288 kB
Hugetlb:               0 kB
```

9.3.2.数据库系统参数调优

为了保证数据库尽可能高性能地运行，建议依据硬件资源情况和业务实际进行数据库系统参数——GUC 参数的设置。这里主要介绍 GUC 参数对性能的影响，关于参数的详细设置方法请参见"管理员指南"。

9.3.2.1. 数据库内存参数调优

数据库的复杂查询语句性能非常强的依赖于数据库系统内存的配置参数。数据库系统内存的配置参数主要包括逻辑内存管理的控制参数和执行算子是否下盘的参数。

逻辑内存管理参数

逻辑内存管理参数为 max_process_memory，主要功能是控制数据库节点上可用内存的最大峰值，该参数的数值设置公式参考 max_process_memory。

执行作业最终可用的内存为：

max_process_memory － shared memory（包括 shared_buffers）－ cstore_buffers

所以影响执行作业可用内存参数的主要两个参数为 shared_buffers 及 cstore_buffers。

逻辑内存管理有专门的视图查询数据库节点中各大块内存区域已使用内存及峰值信息。可连接到单个数据库节点，通过“pg_total_memory_detail”查询该节点上内存区域信息；或者连接到数据库主节点，通过“pgxc_total_memory_detail”查询节点上内存区域信息。

参数 work_mem 依据查询特点和并发来确定，一旦 work_mem 限定的物理内存不够，算子运算数据将写入临时表空间，带来 5-10 倍的性能下降，查询响应时间从秒级下降到分钟级。

- 对于串行无并发的复杂查询场景，平均每个查询有 5-10 关联操作，建议 work_mem=50%内存/10。
- 对于串行无并发的简单查询场景，平均每个查询有 2-5 个关联操作，建议 work_mem=50%内存/5。
- 对于并发场景，建议 work_mem=串行下的 work_mem/物理并发数。

执行算子是否下盘的参数

参数 work_mem 可以判断执行作业可下放到磁盘的算子是否已使用内存量触发下盘点。。当前可下盘算子有六类（向量化及非向量化共 10 种）：Hash(VecHashJoin)，Agg(VecAgg)，Sort(VecSort)，Material(VecMaterial)，SetOp(VecSetOp)，WindowAgg(VecWindowAgg)。该参数设置通常是一个权衡，即要保证并发的吞吐量，又要保证单查询作业的性能，故需要根据实际执行情况（结合 Explain Performance 输出）进行调优。

9.3.3.配置 LLVM

LLVM(Low Level Virtual Machine)动态编译技术可以为每个查询生成定制化的机器码用于替换原本的通用函数。通过减少实际查询时冗余的条件逻辑判断、虚函数调用并提高数据局域性，从而达到提升查询整体性能的目的。

由于 LLVM 需要消耗额外的时间预生成 IR 中间态表示并编译成机器码，因此在小数据量场景或查询本身耗时较少时，可能引起性能的劣化。

9.3.3.1. LLVM 适用场景与限制

适用场景

- 支持 LLVM 的表达式

  查询语句中存在以下的表达式支持 LLVM 优化：

  a. Case...when... 表达式
  b. In 表达式
  c. Bool 表达式 (And/Or/Not)
  d. BooleanTest 表达式 (IS_NOT_KNOWN/IS_UNKNOWN/IS_TRUE/IS_NOT_TRUE/IS_FALSE/IS_NOT_FALSE)
  e. NullTest 表达式 (IS_NOT_NULL/IS_NULL)
  f. Operator 表达式
  g. Function 表达式 (lpad, substring, btrim, rtrim, length)
  h. Nullif 表达式

  表达式计算支持的数据类型包括 bool, tinyint, smallint, int, bigint, float4, float8, numeric, date, time, timetz, timestamp, timestamptz, interval, bpchar, varchar, text, oid。

  仅当表达式出现在向量化执行引擎中 Scan 节点的 filter、Hash Join 节点中的 complicate hash condition、hash join filter、hash join target, Nested Loop 节点中的 filter、join filter, Merge Join 节点的 merge join filter, merge join target, Group 节点中的 filter 表达式时，才会考虑是否使用 LLVM 动态编译优化。

- 支持 LLVM 的算子：

  a. Join ： HashJoin
  b. Agg ： HashAgg
  c. Sort

其中 HashJoin 算子仅支持 Hash Inner Join，对应的 hash cond 仅支持 int4、bigint、bpchar 类型的比较；HashAgg 算子仅支持针对 bigint、numeric 类型的 sum 及 avg 操作，且 group by 语句仅支持 int4、bigint、bpchar，text，varchar，timestamp 类型操作，同时支持 count(*) 聚集操作。Sort 算子仅支持对 int4，bigint，numeric，bpchar，text，varchar 数据类型的比较操作。除此之外，无法使用 LLVM 动态编译优化，具体可通过 explain performance 工具进行显示。

非适用场景

- 不支持小数据量表使用 LLVM 动态编译优化。
- 不支持生成非向量化执行路径的查询作业。

9.3.3.2. 其他因素对 LLVM 性能的影响

LLVM 优化效果不仅依赖于数据库内部具体的实现，还与当前所选择的硬件环境等有关。

- 表达式调用 C-函数个数

  数据库内部针对表达式计算并未实现全 codegen，即在整个表达式计算中部分表达式实现了 codegen，部分直接调用原本的 C 代码。如果整个表达式计算中后者占据了主要部分，使用 LLVM 动态编译优化，可能会导致性能劣化。通过设置 log_min_message 的级别为 DEBUG1 可以查看到哪些表达式直接调用了 C 代码实现。

- 内存资源

  LLVM 特性的一个重要思想是保障数据的局域特性，即数据应尽可能的存放在寄存器中。同时应减少数据加载，因此在使用 LLVM 优化时应设置足够大的 work_mem，保证对应使用 LLVM 优化的执行代码整个过程在内存中实现，否则可能引起性能劣化。

- 优化器代价估算

  LLVM 特性实现了简易的代价估算模型，即依据当前参与节点运算的表大小决定当前节点是否考虑使用 LLVM 动态编译优化。如果优化器低估了实际参与运算的行数，则原本可获得收益的未正常获得收益。反之亦然。

9.3.3.3. LLVM 使用建议

目前 LLVM 在数据库内核侧已默认打开，用户可结合上述的分析进行配置，总体建议如下：

- 设置合理的 work_mem，在允许的条件下尽可能设置较大的 work_mem，如果出现大量下盘，则建议关闭 LLVM 动态编译优化(通过设置 enable_codegen=off 实现)。

❖ 设置合理的 codegen_cost_threshold(默认值为 10000)，确保小数据量场景下避免使用 LLVM 动态编译优化。当 codegen_cost_threshold 的值设定后，因使用 LLVM 动态编译优化引入性能劣化，则建议增加 codegen_cost_threshold 的取值。

❖ 对于表达式计算使用 LLVM 动态编译优化，如果存在大量的调用 C-函数的场景，建议关闭 LLVM 动态编译优化。

说明

在资源许可的情况下，数据量越大，可获得的性能提升效果越好。

9.3.4.资源利用-空闲事务会话超时

功能描述

本功能支持指定空闲事务超时时间（以毫秒为单位），若超时则终止该会话。

示例

```
set idle_in_transaction_session_timeout=1000;
```

9.3.5.资源利用-降级容错

功能描述

当 Vastbase 数据库服务器 CPU 利用率过高，流程出现阻塞等情况时，可以设置 Vastbase 服务器进程调度优先级(即 nice 值，nice 值越大优先级越低)，由于进程优先级非系统超级用户只能降不能升，所以需慎用。

示例

```
1.使用超级管理员登陆数据库
2.获取当前优先级：select get_nice();
3.设置优先级比当前高：select set_nice(10);
```

9.3.6.资源利用-表空间阈值

功能描述

为防止表空间增长到最大值（MAXSIZE）而限制用户操作的突发事件，特采用表空间最低配额阈值（THRESHOLD）来告知用户表空间的使用信息，以便提前处理表空间不足的情况。

示例

```
create tablespace testname relative location 'derectory' [MAXSIZE '1234m'] [with_option_clause]
[THRESHOLD '100'];
```

9.4. SQL 调优指南

SQL 调优的唯一目的是“资源利用最大化”，即 CPU、内存、磁盘 IO 三种资源利用最大化。所有调优手段都是围绕资源使用开展的。所谓资源利用最大化是指 SQL 语句尽量高效，节省资源开销，以最小的代价实现最大的效益。比如做典型点查询的时候，可以用 seqscan+filter(即读取每一条元组和点查询条件进行匹配)实现，也可以通过 indexscan 实现，显然 indexscan 可以以更小的代价实现相同的效果。

根据硬件资源和客户的业务特征确定合理的 Vastbase 部署方案和表定义是数据库在多数情况下满足性能要求的基础。下文的调优说明假设您已根据“软件安装”指引在安装过程中按照合理的 Vastbase 方案完成了安装，且已经根据“开发设计建议”的指引进行了数据库设计。

9.4.1. Query 执行流程

SQL 引擎从接受 SQL 语句到执行 SQL 语句需要经历的步骤如图 3-3 和表 9-3 所示。其中，红色字体部分为 DBA 可以介入实施调优的环节。

图 3-3 SQL 引擎执行查询类 SQL 语句的流程

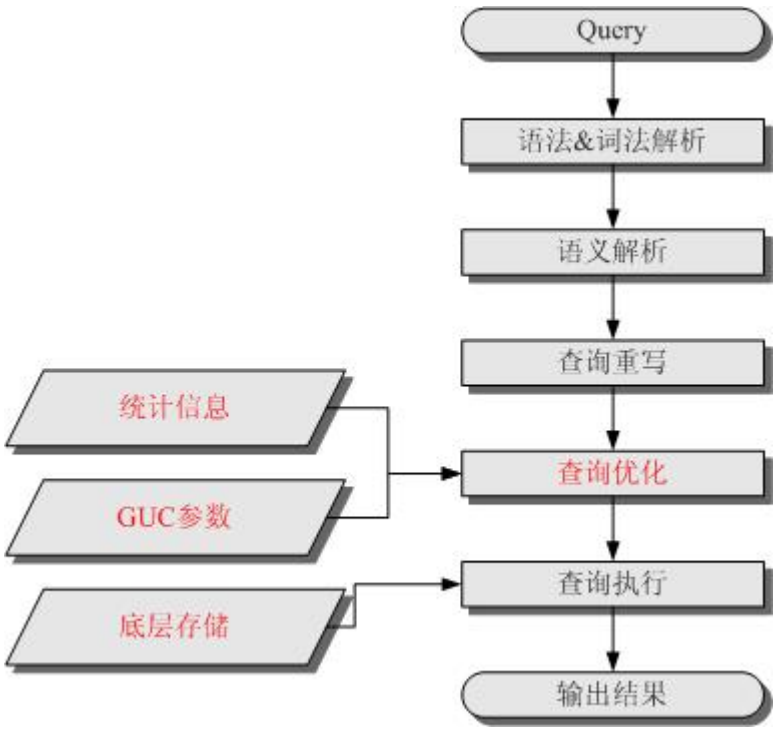


表 9-3. SQL 引擎执行查询类 SQL 语句的步骤说明

步骤	说明
1、语法&词法解析	按照约定的 SQL 语句规则，把输入的 SQL 语句从字符串转化为格式化结构(Stmt)。
2、语义解析	将“语法&词法解析”输出的格式化结构转化为数据库可以识别的对象。
3、查询重写	根据规则把“语义解析”的输出等价转化为执行上更为优化的结构。
4、查询优化	根据“查询重写”的输出和数据库内部的统计信息规划 SQL 语句具体的执行方式，也就是执行计划。统计信息和 GUC 参数对查询优化（执行计划）的影响，请参见调优手段之统计信息和调优手段之 GUC 参数。
5、查询执行	根据“查询优化”规划的执行路径执行 SQL 查询语句。底层存储方式的选择合理性，将影响查询执行效率。详见调优手段之底层存储。

调优手段之统计信息

Vastbase 优化器是典型的基于代价的优化（Cost-Based Optimization，简称 CBO)。在这种优化器模型下，数据库根据表的元组数、字段宽度、NULL 记录比率、distinct 值、MCV 值、HB 值等表的特征值，以及一定的代价计算模型，计算出每一个执行步骤的不同执行方式的输出元组数和执行代价(cost)，进而选出整体执行代价最小/首元组返回代价最小的执行方式进行执行。这些特征值就是统计信息。从上面描述可以看出统计信息是查询优化的核心输入，准确的统计信息将帮助规划器选择最合适的查询规划，一般来说我们通过 analyze 语法收集整个表或者表的若干个字段的统计信息，周期性地运行 ANALYZE，或者在对表的大部分内容做了更改之后马上运行它是个好习惯。

调优手段之 GUC 参数

查询优化的主要目的是为查询语句选择高效的执行方式。

如下 SQL 语句:

```
select count(1)
from customer inner join store_sales on (ss_customer_sk = c_customer_sk);
```

在执行 customer inner join store_sales 的时候，Vastbase 支持 Nested Loop、Merge Join 和 Hash Join 三种不同的 Join 方式。优化器会根据表 customer 和表 store_sales 的统计信息估算结果集的大小以及每种 join 方式的执行代价，然后对比选出执行代价最小的执行计划。

正如前面所说，执行代价计算都是基于一定的模型和统计信息进行估算，当因为某些原因代价估算不能反映真实的 cost 的时候，我们就需要通过 guc 参数设置的方式让执行计划倾向更优规划。

调优手段之底层存储

Vastbase 的表支持行存表、列存表，底层存储方式的选择严格依赖于客户的具体业务场景。一般来说计算型业务查询场景(以关联、聚合操作为主)建议使用列存表；点查询、大批量 UPDATE/DELETE 业务场景适合行存表。

对于每种存储方式还有对应的存储层优化手段，这部分会在后续的调优章节深入介绍。

调优手段之 SQL 重写

除了上述干预 SQL 引擎所生成执行计划的执行性能外，根据数据库的 SQL 执行机制以及大量的实践发现，有些场景下，在保证客户业务 SQL 逻辑的前提下，通过一定规则由 DBA 重写 SQL 语句，可以大幅度的提升 SQL 语句的性能。

这种调优场景对 DBA 的要求比较高，需要对客户业务有足够的了解，同时也需要扎实的 SQL 语句基本功，后续会介绍几个常见的 SQL 改写场景。

9.4.2. SQL 执行计划介绍

9.4.2.1. SQL 执行计划概述

SQL 执行计划是一个节点树，显示 Vastbase 执行一条 SQL 语句时执行的详细步骤。每一个步骤为一个数据库运算符。

使用 EXPLAIN 命令可以查看优化器为每个查询生成的具体执行计划。EXPLAIN 给每个执行节点都输出一行，显示基本的节点类型和优化器为执行这个节点预计的开销值。如图所示。

```
postgres=# explain select * from t1, t2 where t1.c1=t2.c2;
                           QUERY PLAN
-----------------------------------------------------------------
 Hash Join  (cost=58.35..355.67 rows=23091 width=16)
   Hash Cond: (t1.c1 = t2.c2)
   ->  Seq Scan on t1  (cost=0.00..31.49 rows=2149 width=8)
   ->  Hash  (cost=31.49..31.49 rows=2149 width=8)
         ->  Seq Scan on t2  (cost=0.00..31.49 rows=2149 width=8)
(5 rows)
```

- ❖ 最底层节点是表扫描节点，它扫描表并返回原始数据行。不同的表访问模式有不同的扫描节点类型：顺序扫描、索引扫描等。最底层节点的扫描对象也可能是非表行数据（不是直接从表中读取的数据），如 VALUES 子句和返回行集的函数，它们有自己的扫描节点类型。
- ❖ 如果查询需要连接、聚集、排序、或者对原始行做其它操作，那么就会在扫描节点之上添加其它节点。 并且这些操作通常都有多种方法，因此在这些位置也有可能出现不同的执行节点类型。
- ❖ 第一行(最上层节点)是执行计划总执行开销的预计。这个数值就是优化器试图最小化的数值。

执行计划显示信息

除了设置不同的执行计划显示格式外，还可以通过不同的 EXPLAIN 用法，显示不同详细程度的执行计划信息。常见有如下几种，关于更多用法请参见 11.16.85EXPLAIN 语法说明。

- ❖ EXPLAIN statement： 只生成执行计划，不实际执行。其中 statement 代表 SQL 语句。
- ❖ EXPLAIN ANALYZE statement：生成执行计划，进行执行，并显示执行的概要信息。显示中加入了实际的运行时间统计，包括在每个规划节点内部花掉的总时间(以毫秒计)和它实际返回的行数。
- ❖ EXPLAIN PERFORMANCE statement：生成执行计划，进行执行，并显示执行期间的全部信息。

为了测量运行时在执行计划中每个节点的开销，EXPLAIN ANALYZE 或 EXPLAIN PERFORMANCE 会在当前查询执行上增加性能分析的开销。在一个查询上运行 EXPLAIN ANALYZE 或 EXPLAIN PERFORMANCE 有时会比普通查询明显的花费更多的时间。超支的数量依赖于查询的本质和使用的平台。

因此，当定位 SQL 运行慢问题时，如果 SQL 长时间运行未结束，建议通过 EXPLAIN 命令查看执行计划，进行初步定位。如果 SQL 可以运行出来，则推荐使用 EXPLAIN ANALYZE 或 EXPLAIN PERFORMANCE 查看执行计划及其实际的运行信息，以便更精准地定位问题原因。

9.4.2.2. 详解

如 9.4.2.1SQL 执行计划概述节中所说，EXPLAIN 会显示执行计划，但并不会实际执行 SQL 语句。EXPLAIN ANALYZE 和 EXPLAIN PERFORMANCE 两者都会实际执行 SQL 语句并返回执行信息。在这一节将详细解释执行计划及执行信息。

执行计划

以如下 SQL 语句为例：

```
SELECT * FROM t1, t2 WHERE t1.c1 = t2.c2;
```

执行 EXPLAIN 的输出为：

```
postgres=# explain select * from t1, t2 where t1.c1=t2.c2;
                           QUERY PLAN
-----------------------------------------------------------------
 Hash Join  (cost=58.35..355.67 rows=23091 width=16)
   Hash Cond: (t1.c1 = t2.c2)
   ->  Seq Scan on t1  (cost=0.00..31.49 rows=2149 width=8)
   ->  Hash  (cost=31.49..31.49 rows=2149 width=8)
         ->  Seq Scan on t2  (cost=0.00..31.49 rows=2149 width=8)
(5 rows)
```

执行计划层级解读（自下而上纵向）：

1. 第一层：Seq Scan on t2

表扫描算子，用 Seq Scan 的方式扫描表 t2。这一层的作用是把表 t2 的数据从 buffer 或者磁盘上读上来输送给上层节点参与计算。

2. 第二层：Hash

Hash 算子，作用是把下层计算输送上来的算子计算 hash 值，为后续 hash join 操作做数据准备

3. 第三层：Seq Scan on t1

表扫描算子，用 Seq Scan 的方式扫描表 t1。这一层的作用是把表 t1 的数据从 buffer 或者磁盘上读上来输送给上层节点参与 hash join 计算。

4. 第四层：Hash Join

join 算子，主要作用是将 t1 表和 t2 表的数据通过 hash join 的方式连接，并输出结果数据

执行计划中的关键字说明：

5. 表访问方式

- Seq Scan

  全表顺序扫描。

- Index Scan

  优化器决定使用两步的规划：最底层的规划节点访问一个索引，找出匹配索引条件的行的位置，然后从下层索引匹配的行位置读取上层表中的那些行。独立地抓取数据行比顺序地读取它们的开销高很多，但是因为并非所有表的页面都被访问了，这么做实际上仍然比一次顺序扫描开销要少。使用两层规划的原因是，上层规划节点在读取索引标识出来的行位置之前，会先将它们按照物理位置排序，这样可以最小化独立抓取的开销。

  如果在 WHERE 里面使用的好几个字段上都有索引，那么优化器可能会使用索引的 AND 或 OR 的组合。但是这么做要求访问两个索引，因此与只使用一个索引，而把另外一个条件只当作过滤器相比，这个方法未必是更优。

  索引扫描可以分为以下几类，他们之间的差异在于索引的排序机制。

  ■ Bitmap Index Scan

  使用位图索引抓取数据页。

  ■ Index Scan using index_name

  当使用简单的索引搜索，会使用索引的顺序去获取表行数据，这会使得读取这些数据的开销更大。而当前的数据行比较少，因此额外的行位置排序操作是没有价值的。最常用的方法是只获取一行数据去查看这类计划，并且那些查询是需要使用 ORDER BY 条件去匹配索引顺序的。当使用这种方法，为了满足 ORDER BY，就可以省去冗余的排序步骤。
  表连接方式

- Nested Loop

嵌套循环，适用于被连接的数据子集较小的查询。在嵌套循环中，外表驱动内表，外表返回的每一行都要在内表中检索找到它匹配的行，因此整个查询返回的结果集不能太大（不能大于 10000），要把返回子集较小的表作为外表，而且在内表的连接字段上建议要有索引。

- (Sonic) Hash Join

  哈希连接，适用于数据量大的表的连接方式。优化器使用两个表中较小的表，利用连接键在内存中建立 hash 表，然后扫描较大的表并探测散列，找到与散列匹配的行。Sonic 和非 Sonic 的 Hash Join 的区别在于所使用 hash 表结构不同，不影响执行的结果集。

- Merge Join

  归并连接，通常情况下执行性能差于哈希连接。如果源数据已经被排序过，在执行融合连接时，并不需要再排序，此时融合连接的性能优于哈希连接。

6. 运算符

- sort

  对结果集进行排序。

- filter

  EXPLAIN 输出显示 WHERE 子句当作一个"filter"条件附属于顺序扫描计划节点。这意味着规划节点为它扫描的每一行检查该条件，并且只输出符合条件的行。预计的输出行数降低了，因为有 WHERE 子句。不过，扫描仍将必须访问所有 10000 行，因此开销没有降低；实际上它还增加了一些（确切的说，通过 10000 * cpu_operator_cost）以反映检查 WHERE 条件的额外 CPU 时间。

- LIMIT

  LIMIT 限定了执行结果的输出记录数。如果增加了 LIMIT，那么不是所有的行都会被检索到。

执行信息

以如下 SQL 语句为例：

```
select sum(t2.c1) from t1,t2 where t1.c1=t2.c2 group by t1.c2;
```

执行 EXPLAIN PERFORMANCE 输出为：

```
postgres=# explain performance select sum(t2.c1) from t1,t2 where t1.c1=t2.c2 group by t1.c2;
                                                        QUERY PLAN
-----------------------------------------------------------------------------------------------------------------------
 HashAggregate  (cost=471.13..473.13 rows=200 width=16) (actual time=0.068..0.068 rows=0 loops=1)
   Output: sum(t2.c1), t1.c2
   Group By Key: t1.c2
   (CPU: ex c/r=0, ex row=0, ex cyc=164552, inc cyc=175720)
   ->  Hash Join  (cost=58.35..355.67 rows=23091 distinct=[200, 200] width=8) (actual time=0.004..0.004 rows=0 loops=1)
         Output: t1.c2, t2.c1
         Hash Cond: (t1.c1 = t2.c2)
         (CPU: ex c/r=0, ex row=0, ex cyc=7384, inc cyc=11168)
         ->  Seq Scan on public.t1  (cost=0.00..31.49 rows=2149 width=8) (actual time=0.001..0.001 rows=0 loops=1)
               Output: t1.c1, t1.c2
               (CPU: ex c/r=0, ex row=0, ex cyc=3784, inc cyc=3784)
         ->  Hash  (cost=31.49..31.49 rows=2149 width=8) (Actual time: never executed)
               Output: t2.c1, t2.c2
                Buckets: 0  Batches: 0  Memory Usage: 0kB
               (CPU: ex c/r=0, ex row=0, ex cyc=0, inc cyc=0)
               ->  Seq Scan on public.t2  (cost=0.00..31.49 rows=2149 width=8) (Actual time: never executed)
                     Output: t2.c1, t2.c2
                     (CPU: ex c/r=0, ex row=0, ex cyc=0, inc cyc=0)
 Total runtime: 1.087 ms
(19 rows)
```

9.4.3.调优流程

对慢 SQL 语句进行分析，通常包括以下步骤：

操作步骤

步骤 1 收集 SQL 中涉及到的所有表的统计信息。在数据库中，统计信息是规划器生成计划的源数据。没有收集统计信息或者统计信息陈旧往往会造成执行计划严重劣化，从而导致性能问题。从经验数据来看，10%左右性能问题是因为没有收集统计信息。具体请参见 9.4.4 更新统计信息。

步骤 2 通过查看执行计划来查找原因。如果 SQL 长时间运行未结束，通过 EXPLAIN 命令查看执行计划，进行初步定位。如果 SQL 可以运行出来，则推荐使用 EXPLAIN ANALYZE 或 EXPLAIN PERFORMANCE 查看执行计划及实际运行情况，以便更精准地定位问题原因。有关执行计划的详细介绍请参见 9.4.2SQL 执行计划介绍。

步骤 3 9.4.5 审视和修改表定义。

步骤 4 针对 EXPLAIN 或 EXPLAIN PERFORMANCE 信息，定位 SQL 慢的具体原因以及改进措施，具体参见 9.4.6 典型 SQL 调优点。

步骤 5 通常情况下，有些 SQL 语句可以通过查询重写转换成等价的，或特定场景下等价的语句。重写后的语句比原语句更简单，且可以简化某些执行步骤达到提升性能的目的。查询重写方法在各个数据库中基本是通用的。9.4.7 经验总结：SQL 语句改写规则介绍了几种常用的通过改写 SQL 进行调优的方法。

9.4.4.更新统计信息

在数据库中，统计信息是规划器生成计划的源数据。没有收集统计信息或者统计信息陈旧往往会造成执行计划严重劣化，从而导致性能问题。

背景信息

ANALYZE 语句可收集与数据库中表内容相关的统计信息，统计结果存储在系统表 PG_STATISTIC 中。查询优化器会使用这些统计数据，以生成最有效的执行计划。

建议在执行了大批量插入/删除操作后，例行对表或全库执行 ANALYZE 语句更新统计信息。目前默认收集统计信息的采样比例是 30000 行（即：guc 参数 default_statistics_target 默认设置为 100），如果表的总行数超过一定行数（大于 1600000），建议设置 guc 参数 default_statistics_target 为-2，即按 2%收集样本估算统计信息。

对于在批处理脚本或者存储过程中生成的中间表，也需要在完成数据生成之后显式的调用 ANALYZE。

对于表中多个列有相关性且查询中有同时基于这些列的条件或分组操作的情况，可尝试收集多列统计信息，以便查询优化器可以更准确地估算行数，并生成更有效的执行计划。

操作步骤

使用以下命令更新某个表或者整个 database 的统计信息。

```
ANALYZE tablename;                    --更新单个表的统计信息
ANALYZE;                             --更新全库的统计信息
```

使用以下命令进行多列统计信息相关操作。

```
ANALYZE tablename ((column_1, column_2));                    --收集 tablename 表的 column_1、column_2 列的多列统计信息

ALTER TABLE tablename ADD STATISTICS ((column_1, column_2));    --添加 tablename 表的 column_1、column_2 列的多列统计信息声明
ANALYZE tablename;                                          --收集单列统计信息，并收集已声明的多列统计信息

ALTER TABLE tablename DELETE STATISTICS ((column_1, column_2)); --删除 tablename 表的 column_1、column_2 列的多列统计信息或其声明
```

须知

在使用 ALTER TABLE tablename ADD STATISTICS 语句添加了多列统计信息声明后，系统并不会立刻收集多列统计信息，而是在下次对该表或全库进行 ANALYZE 时，进行多列统计信息的收集。

如果想直接收集多列统计信息，请使用 ANALYZE 命令进行收集。

说明

使用 EXPLAIN 查看各 SQL 的执行计划时，如果发现某个表 SEQ SCAN 的输出中 rows=10，rows=10 是系统给的默认值，有可能该表没有进行 ANALYZE，需要对该表执行 ANALYZE。

9.4.5. 审视和修改表定义

9.4.5.1. 审视和修改表定义概述

好的表定义至少需要达到以下几个目标：

- 减少扫描数据数据量。通过分区的剪枝机制可以实现该点。
- 尽量极少随机 IO。通过聚簇/局部聚簇可以实现该点。

表定义在数据库设计阶段创建，在 SQL 调优过程中进行审视和修改。

9.4.5.2. 选择存储模型

进行数据库设计时，表设计上的一些关键项将严重影响后续整库的查询性能。表设计对数据存储也有影响：好的表设计能够减少 I/O 操作及最小化内存使用，进而提升查询性能。

表的存储模型选择是表定义的第一步。客户业务属性是表的存储模型的决定性因素，依据下面表格选择适合当前业务的存储模型。

存储模型	适用场景
行存	点查询(返回记录少，基于索引的简单查询)。 增删改比较多的场景。
列存	统计分析类查询 (group , join 多的场景)。

9.4.5.3. 使用分区表

分区表是把逻辑上的一张表根据某种方案分成几张物理块进行存储。这张逻辑上的表称之为分区表，物理块称之为分区。分区表是一张逻辑表，不存储数据，数据实际是存储在分区上的。分区表和普通表相比具有以下优点：

- 改善查询性能：对分区对象的查询可以仅搜索自己关心的分区，提高检索效率。
- 增强可用性：如果分区表的某个分区出现故障，表在其他分区的数据仍然可用。
- 方便维护：如果分区表的某个分区出现故障，需要修复数据，只修复该分区即可。

Vastbase 支持的分区表为范围分区表。

范围分区表：将数据基于范围映射到每一个分区。这个范围是由创建分区表时指定的分区键决定的。分区键经常采用日期，例如将销售数据按照月份进行分区。

9.4.5.4. 选择数据类型

高效数据类型，主要包括以下三方面：

- 尽量使用执行效率比较高的数据类型

一般来说整型数据运算(包括=、＞、＜、≧、≦、≠等常规的比较运算，以及 group by)的效率比字符串、浮点数要高。比如某客户场景中对列存表进行点查询，filter 条件在一个 numeric 列上，执行时间为 10+s；修改 numeric 为 int 类型之后，执行时间缩短为 1.8s 左右。

- 尽量使用短字段的数据类型

  长度较短的数据类型不仅可以减小数据文件的大小，提升 IO 性能；同时也可以减小相关计算时的内存消耗，提升计算性能。比如对于整型数据，如果可以用 smallint 就尽量不用 int，如果可以用 int 就尽量不用 bigint。

- 使用一致的数据类型

  表关联列尽量使用相同的数据类型。如果表关联列数据类型不同，数据库必须动态地转化为相同的数据类型进行比较，这种转换会带来一定的性能开销。

9.4.6. 典型 SQL 调优点

SQL 调优是一个不断分析与尝试的过程：试跑 Query，判断性能是否满足要求；如果不满足要求，则通过 9.4.2SQL 执行计划介绍分析原因并进行针对性优化；然后重新试跑和优化，直到满足性能目标。

9.4.6.1. SQL 自诊断

用户在执行查询或者执行 INSERT/DELETE/UPDATE/CREATE TABLE AS 语句时，可能会遇到性能问题。这种情况下，通过查询 14.3.12GS_WLM_SESSION_STATISTICS，14.3.9GS_WLM_SESSION_HISTORY 视图的 warning 字段可以获得对应查询可能导致性能问题的告警信息，为性能调优提供参考。

SQL 自诊断的告警类型与 resource_track_level 的设置有关系。如果 resource_track_level 设置为 query，则可以诊断多列/单列统计信息未收集和 SQL 不下推的告警。如果 resource_track_level 设置为 operator，则可以诊断所有的告警场景。

SQL 自诊断的诊断范围与 resource_track_cost 的设置有关系。当 SQL 的代价大于 resource_track_cost 时，SQL 才会被诊断。SQL 的代价可以通过 explain 来确认。

告警场景

目前支持对多列/单列统计信息未收集导致性能问题的场景上报告警。

如果存在单列或者多列统计信息未收集，则上报相关告警。调优方法可以参考 9.4.4 更新统计信息和 9.4.6.3 统计信息调优。

告警信息示例：

整表的统计信息未收集：

```
Statistic Not Collect:
    schema_test.t1
```

单列统计信息未收集：

```
Statistic Not Collect:
    schema_test.t2(c1,c2)
```

多列统计信息未收集:

```
Statistic Not Collect:
    schema_test.t3((c1,c2))
```

单列和多列统计信息未收集：

```
Statistic Not Collect:
    schema_test.t4(c1,c2)    schema_test.t4((c1,c2))
```

规格约束

- 告警字符串长度上限为 2048。如果告警信息超过这个长度（例如存在大量未收集统计信息的超长表名，列名等信息）则不告警，只上报 warning：

  ```
  WARNING, "Planner issue report is truncated, the rest of planner issues will be skipped"
  ```

- 如果 query 存在 limit 节点（即查询语句中包含 limit），则不会上报 limit 节点以下的 Operator 级别的告警。

9.4.6.2. 子查询调优

子查询背景介绍

应用程序通过 SQL 语句来操作数据库时会使用大量的子查询，这种写法比直接对两个表做连接操作在结构上和思路上更清晰，尤其是在一些比较复杂的查询语句中，子查询有更完整、更独立的语义，会使 SQL 对业务逻辑的表达更清晰更容易理解，因此得到了广泛的应用。

Vastbase 根据子查询在 SQL 语句中的位置把子查询分成了子查询、子链接两种形式。

- 子查询 SubQuery：对应于查询解析树中的范围表 RangeTblEntry，更通俗一些指的是出现在 FROM 语句后面的独立的 SELECT 语句。
- 子链接 SubLink：对应于查询解析树中的表达式，更通俗一些指的是出现在 where/on 子句、targetlist 里面的语句。

  综上，对于查询解析树而言，SubQuery 的本质是范围表、而 SubLink 的本质是表达式。针对 SubLink 场景而言，由于 SubLink 可以出现在约束条件、表达式中，按照 Vastbase 对 sublink 的实现，sublink 可以分为以下几类：

  - exist_sublink：对应 EXIST、NOT EXIST 语句
  - any_sublink：对应 op ALL(select...)语句，其中 OP 可以是 IN,<,>,=操作符
  - all_sublink：对应 op ALL(select...)语句，其中 OP 可以是 IN,<,>,=操作符
  - rowcompare_sublink：对应 record op (select ...)语句
  - expr_sublink：对应(SELECT with single targetlist item ...)语句
  - array_sublink：对应 ARRAY(select...)语句
  - cte_sublink：对应 with query(...)语句

其中 OLAP、HTAP 场景中常用的 sublink 为 exist_sublink、any_sublink，在 Vastbase 的优化引擎中对其应用场景做了优化（子链接提升），由于 SQL 语句中子查询的使用的灵活性，会带来 SQL 子查询过于复杂造成性能问题。子查询从大类上来看，分为非相关子查询和相关子查询：

- 非相关子查询 None-Correlated SubQuery

  子查询的执行不依赖于外层父查询的任何属性值。这样子查询具有独立性，可独自求解，形成一个子查询计划先于外层的查询求解。

  例如：

```
select t1.c1,t1.c2
from t1
where t1.c1 in (
   select c2
   from t2
   where t2.c2 IN (2,3,4)
);
                        QUERY PLAN
------------------------------------------------------------------
 Hash Join
   Hash Cond: (t1.c1 = t2.c2)
   ->  Seq Scan on t1
         Filter: (c1 = ANY ('{2,3,4}'::integer[]))
   ->  Hash
         ->  HashAggregate
               Group By Key: t2.c2
               ->  Seq Scan on t2
                     Filter: (c2 = ANY ('{2,3,4}'::integer[]))
(9 rows)
```

- 相关子查询 Correlated-SubQuery

  子查询的执行依赖于外层父查询的一些属性值(如下列示例 t2.c1 = t1.c1 条件中的 t1.c1)作为内层查询的一个 AND-ed 条件。这样的子查询不具备独立性，需要和外层查询按分组进行求解。

  例如：

```
select t1.c1,t1.c2
from t1
where t1.c1 in (
   select c2
   from t2
   where t2.c1 = t1.c1 AND t2.c2 in (2,3,4)
);
                        QUERY PLAN
------------------------------------------------------------------------
 Seq Scan on t1
   Filter: (SubPlan 1)
   SubPlan 1
     ->  Seq Scan on t2
           Filter: ((c1 = t1.c1) AND (c2 = ANY ('{2,3,4}'::integer[])))
```

```
(5 rows)
```

Vastbase 对 SubLink 的优化

针对 SubLink 的优化策略主要是让内层的子查询提升(pullup)，能够和外表直接做关联查询，从而避免生成 SubPlan+Broadcast 内表的执行计划。判断子查询是否存在性能风险，可以通过 explain 查询语句查看 Sublink 的部分是否被转换成 SubPlan 的执行计划。

例如：

```
select t1.c1,t1.c2
from t1
where t1.c1 in (
    select c2
    from t2
    where t2.c1 = t1.c1
);
```



```
QUERY PLAN
-----------------------------------------------------
Streaming (type: GATHER)
  Node/s: All datanodes
  ->  Seq Scan on t1
        Filter: (SubPlan 1)
        SubPlan 1
          ->  Result
                Filter: (t2.c1 = t1.c1)
                ->  Materialize
                      ->  Streaming(type: BROADCAST)
                            Spawn on: All datanodes
                            ->  Seq Scan on t2
(11 rows)
```

箭头右侧执行计划应替换成下面的执行计划

```
QUERY PLAN
-------------------------------
Seq Scan on t1
Filter: (SubPlan 1)
SubPlan 1
->   Seq Scan on t2
Filter: (c1 = t1.c1)
(5 rows)
```

- ❖ 目前 Vastbase 支持的 Sublink-Release 场景
  - – IN-Sublink 无相关条件
    - ■ 不能包含上一层查询的表中的列（可以包含更高层查询表中的列）。
    - ■ 不能包含易变函数。

```
select t1.c1,t1.c2
from t1
where t1.c1 in (
    select c2
    from t2
    where t2.c1 = 1
);
```

```
QUERY PLAN
-------------------------------------------------
Streaming (type: GATHER)
  Node/s: All datanodes
  ->  Nested Loop Semi Join
        Join Filter: (t1.c1 = t2.c2)
        ->  Seq Scan on t1
        ->  Materialize
              ->  Streaming(type: REDISTRIBUTE)
                    Spawn on: datanode1
                    ->  Seq Scan on t2
                          Filter: (c1 = 1)
(10 rows)
```

箭头右侧执行计划应替换成下面的执行计划

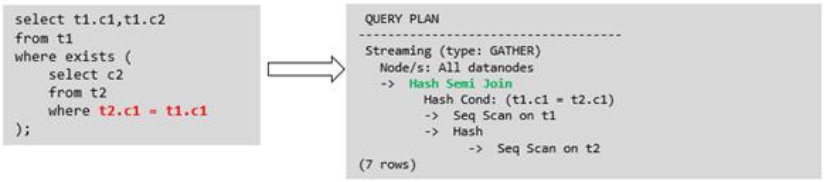
```
QUERY PLAN
--------------------------------------
Hash Join
```

```
Hash Cond: (t1.c1 = t2.c2)
->  Seq Scan on t1
->  Hash
->  HashAggregate
Group By Key: t2.c2
->  Seq Scan on t2
Filter: (c1 = 1)
(8 rows)
```

– Exist-Sublink 包含相关条件

Where 子句中必须包含上一层查询的表中的列，子查询的其它部分不能含有上层查询的表中的列。其它限制如下。

- 子查询必须有 from 子句。
- 子查询不能含有 with 子句。
- 子查询不能含有聚集函数。
- 子查询里不能包含集合操作、排序、limit、windowagg、having 操作。
- 不能包含易变函数。



箭头右侧执行计划应替换成下面的执行计划

```
QUERY PLAN
----------------------------------
Hash Join
Hash Cond: (t1.c1 = t2.c1)
->  Seq Scan on t1
->  Hash
->  HashAggregate
Group By Key: t2.c1
->  Seq Scan on t2
(7 rows)
```

– 包含聚集函数的等值相关子查询的提升

子查询的 where 条件中必须含有来自上一层的列，而且此列必须和子查询本层涉及表中的列做相等判断，且这些条件必须用 and 连接。其它地方不能包含上层的列。其它限制条件如下。

- 子查询中 where 条件包含的表达式(列名)必须是表中的列。
- 子查询的 Select 关键字后，必须有且仅有一个输出列，此输出列必须是聚集函数(如 max)，并且聚集函数的参数(t2.c2)不能是来自外层表(t1)中的列。聚集函数不能是 count。

例如，下列示例可以提升。

```
select * from t1 where c1 >(
        select max(t2.c1) from t2 where t2.c1=t1.c1
);
```

下列示例不能提升，因为子查询没有聚集函数。

```
select * from t1 where c1 >(
        select   t2.c1 from t2 where t2.c1=t1.c1
);
```

下列示例不能提升，因为子查询有两个输出列。

```
select * from t1 where (c1,c2) >(
        select   max(t2.c1),min(t2.c2) from t2 where t2.c1=t1.c1
);
```

- 子查询必须是 from 子句。
- 子查询中不能有 groupby、having、集合操作。
- 子查询只能是 inner join。

例如：下列示例不能提升。

```
select * from t1 where c1 >(
        select max(t2.c1) from t2 full join t3 on (t2.c2=t3.c2) where t2.c1=t1.c1
);
```

- 子查询的 targetlist 中不能包含返回 set 的函数。

子查询的 where 条件中必须含有来自上一层的列，而且此列必须和子查询层涉及表中的列做相等判断，且这些条件必须用 and 连接。其它地方不能包含上层的上层中的列。例如：下列示例中的最内层子链接可以提升。

```
select * from t3 where t3.c1=(
          select t1.c1
          from t1 where c1 >(
```

```
                    select max(t2.c1) from t2 where t2.c1=t1.c1
));
```

基于上面的示例，再加一个条件，则不能提升，因为最内侧子查询引用了上层中的列。示例如下：

```
select * from t3 where t3.c1=(
            select t1.c1
            from t1 where c1 >(
                        select max(t2.c1) from t2 where t2.c1=t1.c1 and t3.c1>t2.c2

));
```

- 提升 OR 子句中的 SubLink

当 WHERE 过滤条件中有 OR 连接的 EXIST 相关 SubLink，

例如：

```
select a, c from t1
where t1.a = (select avg(a) from t3 where t1.b = t3.b) or
exists (select * from t4 where t1.c = t4.c);
```

将 OR-ed 连接的 EXIST 相关子查询 OR 字句的提升过程：

i. 提取 where 条件中，or 子句中的 opExpr。为：t1.a = (select avg(a) from t3 where t1.b = t3.b)

ii. 这个 op 操作中包含 subquery，判断是否可以提升，如果可以提升，重写 subquery 为：select avg(a), t3.b from t3 group by t3.b，生成 not null 条件 t3.b is not null，并将这个 opexpr 用这个 not null 条件替换。此时 SQL 变为：

```
select a, c
from t1 left join (select avg(a) avg, t3.b from t3 group by t3.b)  as t3 on (t1.a = avg
and t1.b = t3.b)
where t3.b is not null or exists (select * from t4 where t1.c = t4.c);
```

iii. 再次提取 or 子句中的 exists sublink，exists (select * from t4 where t1.c = t4.c)，判断是否可以提升，如果可以提升，转换 subquery 为：select t4.c from t4 group by t4.c 生成 NotNull 条件 t4.c is not null 提升查询，SQL 变为：

```
select a, c
from t1 left join (select avg(a) avg, t3.b from t3 group by t3.b)  as t3 on (t1.a = avg and
t1.b = t3.b)
left join (select t4.c from t4 group by t4.c) where t3.b is not null or t4.c is not null;
```

❖ 目前 Vastbase 不支持的 Sublink-Release 场景

除了以上场景之外都不支持 Sublink 提升，因此关联子查询会被计划成 SubPlan+Broadcast 的执行计划，当 inner 表的数据量较大时则会产生性能风险。

如果相关子查询中跟外层的两张表做 join，那么无法提升该子查询，需要通过将父 SQL 创建成 with 子句，然后再跟子查询中的表做相关子查询查询。

例如：

```
select distinct t1.a, t2.a
from t1 left join t2 on t1.a=t2.a and not exists (select a,b from test1 where test1.a=t1.a and
test1.b=t2.a);
```

改写为

```
with temp as
(
      select * from (select t1.a as a, t2.a as b from t1 left join t2 on t1.a=t2.a)

)
select distinct a,b
from temp
where not exists (select a,b from test1 where temp.a=test1.a and temp.b=test1.b);
```

- 出现在 targetlist 里的相关子查询无法提升(不含 count)

  例如：

  ```
  explain (costs off)
  select (select c2 from t2 where t1.c1 = t2.c1) ssq, t1.c2
  from t1
  where t1.c2 > 10;
  ```

  执行计划为：

  ```
  explain (costs off)
  select (select c2 from t2 where t1.c1 = t2.c1) ssq, t1.c2
  from t1
  where t1.c2 > 10;
          QUERY PLAN
  -------------------------------
   Seq Scan on t1
     Filter: (c2 > 10)
     SubPlan 1
       ->  Seq Scan on t2
             Filter: (t1.c1 = c1)
  (5 rows)
  ```

  由于相关子查询出现在 targetlist(查询返回列表)里，对于 t1.c1=t2.c1 不匹配的场景仍然需要输出值，因此使用 left-outerjoin 关联 T1&T2 确保 t1.c1=t2.c1 在不匹配时，子 SSQ 能够返回不匹配的补空值。

说明

SSQ 和 CSSQ 的解释如下：

- SSQ：ScalarSubQuery 一般指返回 1 行 1 列 scalar 值的 sublink，简称 SSQ。
- CSSQ：Correlated-ScalarSubQuery 和 SSQ 相同不过是指包含相关条件的 SSQ。

上述 SQL 语句可以改写为：

```
with ssq as
(
   select t2.c2 from t2
)
select ssq.c2, t1.c2
from t1 left join ssq on t1.c1 = ssq.c2
where t1.c2 > 10;
```

改写后的执行计划为：

```
         QUERY PLAN
--------------------------------
 Hash Right Join
   Hash Cond: (ssq.c2 = t1.c1)
   CTE ssq
     ->  Seq Scan on t2
   ->  CTE Scan on ssq
   ->  Hash
         ->  Seq Scan on t1
               Filter: (c2 > 10)
(8 rows)
```

可以看到出现在 SSQ 返回列表里的相关子查询 SSQ，已经被提升成 Right Join，从而避免当内表 T2 较大时出现 SubPlan 计划导致性能变差。

- 出现在 targetlist 里的相关子查询无法提升(带 count)

例如：

```
select (select count(*) from t2 where t2.c1=t1.c1) cnt, t1.c1, t3.c1
from t1,t3
where t1.c1=t3.c1 order by cnt, t1.c1;
```

执行计划为

```
              QUERY PLAN
-----------------------------------------
 Sort
   Sort Key: ((SubPlan 1)), t1.c1
   ->  Hash Join
         Hash Cond: (t1.c1 = t3.c1)
         ->  Seq Scan on t1
         ->  Hash
               ->  Seq Scan on t3
         SubPlan 1
           ->  Aggregate
                 ->  Seq Scan on t2
                       Filter: (c1 = t1.c1)
(11 rows)
```

由于相关子查询出现在 targetlist(查询返回列表)里，对于 t1.c1=t2.c1 不匹配的场景仍然需要输出值，因此使用 left-outerjoin 关联 T1&T2 确保 t1.c1=t2.c1 在不匹配时子 SSQ 能够返回不匹配的补空值，但是这里带了 count 语句及时在 t1.c1=t2.t1 不匹配时需要输出 0，因此可以使用一个 case-when NULL then 0 else count(*)来代替。

上述 SQL 语句可以改写为：

```
with ssq as
(
   select count(*) cnt, c1 from t2 group by c1
)
select case when
          ssq.cnt is null then 0
          else ssq.cnt
     end cnt, t1.c1, t3.c1
from t1 left join ssq on ssq.c1 = t1.c1,t3
where t1.c1 = t3.c1
order by ssq.cnt, t1.c1;
```

改写后的执行计划为

```
             QUERY PLAN
-----------------------------------------
 Sort
   Sort Key: ssq.cnt, t1.c1
   CTE ssq
     ->  HashAggregate
           Group By Key: t2.c1
           ->  Seq Scan on t2
   ->  Hash Join
         Hash Cond: (t1.c1 = t3.c1)
         ->  Hash Left Join
               Hash Cond: (t1.c1 = ssq.c1)
               ->  Seq Scan on t1
               ->  Hash
                     ->  CTE Scan on ssq
         ->  Hash
               ->  Seq Scan on t3
(15 rows)
```

- 相关条件为不等值场景

例如：

```
select t1.c1, t1.c2
from t1
where t1.c1 = (select agg() from t2.c2 > t1.c2);
```

对于非等值相关条件的 SubLink 目前无法提升，从语义上可以通过做 2 次 join（一次 CorrelationKey，一次 rownum 自关联）达到提升改写的目的。

改写方案有两种。

- 子查询改写方式

```
select t1.c1, t1.c2
from t1, (
   select t1.rowid, agg() aggref
   from t1,t2
   where t1.c2 > t2.c2 group by t1.rowid
) dt /* derived table */
where t1.rowid = dt.rowid AND t1.c1 = dt.aggref;
```

- CTE 改写方式

```
WITH dt as
(
   select t1.rowid, agg() aggref
   from t1,t2
   where t1.c2 > t2.c2 group by t1.rowid
)
select t1.c1, t1.c2
from t1, derived_table
where t1.rowid = derived_table.rowid AND
t1.c1 = derived_table.aggref;
```

须知

- 对于 AGG 类型为 count(*)时需要进行 CASE-WHEN 对没有 match 的场景补 0 处理，非 COUNT(*)场景 NULL 处理。
- CTE 改写方式如果有 sharescan 支持性能上能够更优。

更多优化示例

示例：修改 select 语句，将子查询修改为和主表的 join，或者修改为可以提升的 subquery，但是在修改前后需要保证语义的正确性。

```
explain (costs off) select * from t1 where t1.c1 in (select t2.c1 from t2 where t1.c1 = t2.c2);
         QUERY PLAN
--------------------------------
 Seq Scan on t1
   Filter: (SubPlan 1)
   SubPlan 1
     ->  Seq Scan on t2
           Filter: (t1.c1 = c2)
(5 rows)
```

上面事例计划中存在一个 subPlan，为了消除这个 subPlan 可以修改语句为：

```
explain (costs off) select * from t1 where exists (select t2.c1 from t2 where t1.c1 = t2.c2 and t1.c1
= t2.c1);
             QUERY PLAN
-----------------------------------------
 Hash Join
   Hash Cond: (t1.c1 = t2.c2)
   ->  Seq Scan on t1
   ->  Hash
         ->  HashAggregate
               Group By Key: t2.c2, t2.c1
               ->  Seq Scan on t2
                     Filter: (c2 = c1)
(8 rows)
```

从计划可以看出，subPlan 消除了，计划变成了两个表的 hash join，这样会大大提高执行效率。

9.4.6.3. 统计信息调优

统计信息调优介绍

Vastbase 是基于代价估算生成的最优执行计划。优化器需要根据 analyze 收集的统计信息行数估算和代价估算，因此统计信息对优化器行数估算和代价估算起着至关重要的作用。通过 analyze 收集全局统计信息，主要包括：pg_class 表中的 relpages 和 reltuples；pg_statistic 表中的 stadistinct、stanullfrac、stanumbersN、stavaluesN、histogram_bounds 等。

实例分析 1：未收集统计信息导致查询性能差

在很多场景下，由于查询中涉及到的表或列没有收集统计信息，会对查询性能有很大的影响。

表结构如下所示：

```
CREATE TABLE LINEITEM
(
L_ORDERKEY        BIGINT       NOT NULL
, L_PARTKEY       BIGINT       NOT NULL
, L_SUPPKEY       BIGINT       NOT NULL
, L_LINENUMBER    BIGINT       NOT NULL
, L_QUANTITY      DECIMAL(15,2) NOT NULL
, L_EXTENDEDPRICE  DECIMAL(15,2) NOT NULL
, L_DISCOUNT      DECIMAL(15,2) NOT NULL
, L_TAX           DECIMAL(15,2) NOT NULL
, L_RETURNFLAG    CHAR(1)      NOT NULL
, L_LINESTATUS    CHAR(1)      NOT NULL
, L_SHIPDATE      DATE         NOT NULL
, L_COMMITDATE    DATE         NOT NULL
, L_RECEIPTDATE   DATE         NOT NULL
, L_SHIPINSTRUCT  CHAR(25)     NOT NULL
, L_SHIPMODE      CHAR(10)     NOT NULL
, L_COMMENT       VARCHAR(44)  NOT NULL
) with (orientation = column, COMPRESSION = MIDDLE);

CREATE TABLE ORDERS
(
O_ORDERKEY       BIGINT       NOT NULL
, O_CUSTKEY      BIGINT       NOT NULL
, O_ORDERSTATUS  CHAR(1)      NOT NULL
, O_TOTALPRICE   DECIMAL(15,2) NOT NULL
, O_ORDERDATE    DATE NOT NULL
, O_ORDERPRIORITY CHAR(15)     NOT NULL
, O_CLERK        CHAR(15)     NOT NULL
, O_SHIPPRIORITY  BIGINT      NOT NULL
, O_COMMENT      VARCHAR(79)  NOT NULL
)with (orientation = column, COMPRESSION = MIDDLE);
```

查询语句如下所示：

```
explain verbose select
count(*) as numwait
from
lineitem l1,
orders
where
o_orderkey = l1.l_orderkey
and o_orderstatus = 'F'
```

```
and l1.l_receiptdate > l1.l_commitdate
and not exists (
select
*
from
lineitem l3
where
l3.l_orderkey = l1.l_orderkey
and l3.l_suppkey <> l1.l_suppkey
and l3.l_receiptdate > l3.l_commitdate
)
order by
numwait desc;
```

当出现该问题时，可以通过如下方法确认查询中涉及到的表或列有没有做过 analyze 收集统计信息。

1. 通过 explain verbose 执行 query 分析执行计划时会提示 WARNING 信息，如下所示：

```
WARNING:Statistics in some tables or columns(public.lineitem.l_receiptdate,
public.lineitem.l_commitdate, public.lineitem.l_orderkey, public.lineitem.l_suppkey,
public.orders.o_orderstatus, public.orders.o_orderkey) are not collected.
HINT:Do analyze for them in order to generate optimized plan.
```

2. 可以通过在 pg_log 目录下的日志文件中查找以下信息来确认是当前执行的 query 是否由于没有收集统计信息导致查询性能变差。

```
2017-06-14 17:28:30.336 CST 140644024579856 20971684 [BACKEND] LOG:Statistics in some tables
or columns(public.lineitem.l_receiptdate, public.lineitem.l_commitdate,
public.lineitem.l_orderkey, public.linei
tem.l_suppkey, public.orders.o_orderstatus, public.orders.o_orderkey) are not collected.
2017-06-14 17:28:30.336 CST 140644024579856 20971684 [BACKEND] HINT:Do analyze for them in order
to generate optimized plan.
```

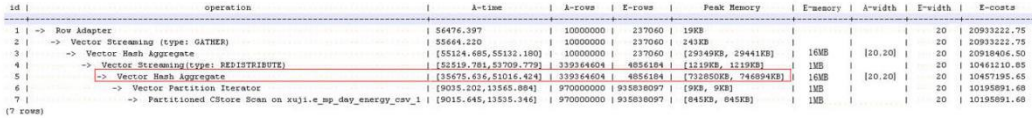
当通过以上方法查看到哪些表或列没有做 analyze，可以通过对 WARNING 或日志中上报的表或列做 analyze 可以解决由于为收集统计信息导致查询变慢的问题。

9.4.6.4. 算子级调优

算子级调优介绍

一个查询语句要经过多个算子步骤才会输出最终的结果。由于各别算子耗时过长导致整体查询性能下降的情况比较常见。这些算子是整个查询的瓶颈算子。通用的优化手段是 EXPLAIN ANALYZE/PERFORMANCE 命令查看执行过程的瓶颈算子，然后进行针对性优化。

如下面的执行过程信息中，Hashagg 算子的执行时间占总时间的：(51016-13535)/ 56476 ≈66%，此处 Hashagg 算子就是这个查询的瓶颈算子，在进行性能优化时应当优先考虑此算子的优化。



id	operation	A-time	A-rows	E-rows	Peak Memory	E-memory	A-width	E-width	E-costs
1	-> Row Adapter	56476.397	10000000	237060	19KB			20	20933222.75
2	-> Vector Streaming (type: GATHER)	55664.220	10000000	237060	243KB			20	20933222.75
3	-> Vector Hash Aggregate	[55124.685,55132.180]	10000000	237060	[29349KB, 29441KB]	16MB	[20,20]	20	20918406.50
4	-> Vector Streaming(type: REDISTRIBUTE)	[52519.781,53709.779]	339364604	4856184	[1219KB, 1219KB]	1MB		20	10461210.85
5	-> Vector Hash Aggregate	[35675.636,51016.424]	339364604	4856184	[732850KB, 746894KB]	16MB	[20,20]	20	10457195.65
6	-> Vector Partition Iterator	[9035.202,13565.884]	970000000	935838097	[9KB, 9KB]	1MB		20	10195891.68
7	-> Partitioned CStore Scan on xuji.e_mp_day_energy_csv_1	[9015.645,13535.346]	970000000	935838097	[845KB, 845KB]	1MB		20	10195891.68

(7 rows)

算子级调优示例

示例 1：基表扫描时，对于点查或者范围扫描等过滤大量数据的查询，如果使用 SeqScan 全表扫描会比较耗时，可以在条件列上建立索引选择 IndexScan 进行索引扫描提升扫描效率。

```
vastbase=#  explain (analyze on, costs off) select * from store_sales where ss_sold_date_sk = 2450944;
 id |            operation         |       A-time       | A-rows | Peak Memory  | A-width
----+------------------------------+--------------------+--------+--------------+---------
  1 | ->  Streaming (type: GATHER)   | 3666.020           |   3360 | 195KB        |
  2 |    ->  Seq Scan on store_sales | [3594.611,3594.611] |   3360 | [34KB, 34KB] |
(2 rows)

 Predicate Information (identified by plan id)
-----------------------------------------------
   2 --Seq Scan on store_sales
       Filter: (ss_sold_date_sk = 2450944)
       Rows Removed by Filter: 4968936
vastbase=#  create index idx on store_sales_row(ss_sold_date_sk);
CREATE INDEX
vastbase=#  explain (analyze on, costs off) select * from store_sales_row where ss_sold_date_sk =
2450944;
 id |                 operation                  |     A-time      | A-rows | Peak Memory  | A-width
----+--------------------------------------------+-----------------+--------+--------------+-
---------
  1 | ->  Streaming (type: GATHER)               | 81.524          |   3360 | 195KB        |
  2 |    ->  Index Scan using idx on store_sales_row | [13.352,13.352] |   3360 | [34KB, 34KB] |
(2 rows)
```

上述例子中，全表扫描返回 3360 条数据，过滤掉大量数据，在 ss_sold_date_sk 列上建立索引后，使用 IndexScan 扫描效率显著提高，从 3.6 秒提升到 13 毫秒。

示例 2：如果从执行计划中看，两表 join 选择了 NestLoop，而实际行数比较大时，NestLoop Join 可能执行比较慢。如下的例子中 NestLoop 耗时 181 秒，如果设置参数 enable_mergejoin=off 关掉 Merge Join，同时设置参数 enable_nestloop=off 关掉 NestLoop，让优化器选择 HashJoin，则 Join 耗时提升至 200 多毫秒。

```
postgres=#  explain analyze select count(*) from store_sales ss, item i where ss.ss_item_sk = i.i_item_sk;
 id |                operation                 |       A-time       |  A-rows   | E-rows  |  Peak Memory   | E-memory | A-width | E-width |   E-costs
----+------------------------------------------+--------------------+-----------+---------+----------------+----------+---------+---------+-------------
  1 | ->  Row Adapter                          | 184300.301         |         1 |       1 | 11KB           |          |         |       0 | 48629179.77
  2 |   ->  Vector Aggregate                   | 184300.280         |         1 |       1 | 181KB          |          |         |       0 | 48629179.77
  3 |     ->  Vector Streaming (type: GATHER)  | 184300.186         |         4 |       4 | 188KB          |          |         |       0 | 48629179.77
  4 |       ->  Vector Aggregate               | [165575.384,184252.368] |         4 |       4 | [140KB, 140KB] | 1MB      |         |       0 | 48629179.61
  5 |         ->  Vector Nest Loop (6,7)       | [162918.848,181438.162] |   2880404 | 2880404 | [74KB, 74KB]   | 1MB      |         |       0 | 48627379.35
  6 |           ->  CStore Scan on store_sales ss | [15.660,16.229]    |   2880404 | 2880404 | [490KB, 490KB] | 1MB      |         |       4 | 16683.10
  7 |           ->  Vector Materialize         | [118314.521,132478.454] | 1296821302 |   18000 | [869KB, 900KB] | 16MB     | [8,8]   |       4 | 3890.00
  8 |             ->  CStore Scan on item i    | [0.234,0.243]      |     18000 |   18000 | [476KB, 476KB] | 1MB      |         |       4 | 3867.50
(8 rows)


 postgres=#  set enable_nestloop=off;
 SET
 postgres=#  set enable_mergejoin=off;
 SET
 ppostgres=# explain analyze select count(*) fpostgres=# ales ss, item i where ss.ss_item_sk = i.i_item_sk;
  id |                operation                 |       A-time       | A-rows  | E-rows  |  Peak Memory   | E-memory | A-width | E-width | E-costs
 ----+------------------------------------------+--------------------+---------+---------+----------------+----------+---------+---------+-----------
   1 | ->  Row Adapter                          | 291.066            |       1 |       1 | 11KB           |          |         |       0 | 32308.66
   2 |   ->  Vector Aggregate                   | 291.052            |       1 |       1 | 181KB          |          |         |       0 | 32308.66
   3 |     ->  Vector Streaming (type: GATHER)  | 290.973            |       4 |       4 | 188KB          |          |         |       0 | 32308.66
   4 |       ->  Vector Aggregate               | [220.792,234.532]  |       4 |       4 | [140KB, 140KB] | 1MB      |         |       0 | 32308.50
   5 |         ->  Vector Hash Join (6,7)       | [209.987,223.345]  | 2880404 | 2880404 | [236KB, 241KB] | 16MB     | [8,8]   |       0 | 30508.24
   6 |           ->  CStore Scan on store_sales ss | [13.132,13.717]  | 2880404 | 2880404 | [490KB, 490KB] | 1MB      |         |       4 | 16683.10
   7 |           ->  CStore Scan on item i      | [0.214,0.246]      |   18000 |   18000 | [477KB, 477KB] | 1MB      |         |       4 | 3867.50
 (7 rows)
```

示例 3：通常情况下 Agg 选择 HashAgg 性能较好，如果大结果集选择了 Sort+GroupAgg，则需要设置 enable_sort=off，HashAgg 耗时明显优于 Sort+GroupAgg。

```
postgres=#  explain analyze select count(*) from store_sales group by ss_item_sk;
 id |                operation                 |       A-time       | A-rows  | E-rows  |      Peak Memory      | E-memory | A-width | E-width | E-costs
----+------------------------------------------+--------------------+---------+---------+-----------------------+----------+---------+---------+----------
  1 | ->  Row Adapter                          | 1977.385           |   18000 |   17644 | 20KB                  |          |         |       4 | 92875.24
  2 |   ->  Vector Streaming (type: GATHER)    | 1973.617           |   18000 |   17644 | 1946KB                |          |         |       4 | 92875.24
  3 |     ->  Vector Sort Aggregate            | [1784.800,1883.243] |   18000 |   17644 | [273KB, 273KB]        | 1MB      |         |       4 | 92186.02
  4 |       ->  Vector Sort                    | [1752.270,1848.357] | 2880404 | 2880404 | [128466KB, 135135KB] | 16MB     | [8,8]   |       4 | 88541.40
  5 |         ->  CStore Scan on store_sales   | [12.483,13.548]    | 2880404 | 2880404 | [490KB, 490KB]        | 1MB      |         |       4 | 16683.10
(5 rows)
```

```
postgres=#  set enable_sort=off;
SET
postgres=#  explain analyze select count(*) from store_sales group by ss_item_sk;
 id |              operation               |       A-time        | A-rows  | E-rows  |        Peak Memory        | E-memory | A-width | E-width | E-costs
----+--------------------------------------+---------------------+---------+---------+---------------------------+----------+---------+---------+----------
  1 | ->  Row Adapter                      | 838.218             |   18000 |   17644 | 20KB                      |          |         |       4 | 21016.93
  2 |    ->  Vector Streaming (type: GATHER) | 834.264           |   18000 |   17644 | 228KB                     |          |         |       4 | 21016.93
  3 |       ->  Vector Hash Aggregate      | [585.017,758.204]   |   18000 |   17644 | [262552KB, 262564KB]      | 16MB     | [8,8]   |       4 | 20327.72
  4 |          ->  CStore Scan on store_sales | [12.540,13.941]  | 2880404 | 2880404 | [490KB, 490KB]            | 1MB      |         |       4 | 16683.10
(4 rows)
```

9.4.7.经验总结：SQL 语句改写规则

根据数据库的 SQL 执行机制以及大量的实践，总结发现：通过一定的规则调整 SQL 语句，在保证结果正确的基础上，能够提高 SQL 执行效率。如果遵守这些规则，常常能够大幅度提升业务查询效率。

- 使用 union all 代替 union

  union 在合并两个集合时会执行去重操作，而 union all 则直接将两个结果集合并、不执行去重。执行去重会消耗大量的时间，因此，在一些实际应用场景中，如果通过业务逻辑已确认两个集合不存在重叠，可用 union all 替代 union 以便提升性能。

- join 列增加非空过滤条件

  若 join 列上的 NULL 值较多，则可以加上 is not null 过滤条件，以实现数据的提前过滤，提高 join 效率。

- not in 转 not exists

  not in 语句需要使用 nestloop anti join 来实现，而 not exists 则可以通过 hash anti join 来实现。在 join 列不存在 null 值的情况下，not exists 和 not in 等价。因此在确保没有 null 值时，可以通过将 not in 转换为 not exists，通过生成 hash join 来提升查询效率。

  如下所示，如果 t2.d2 字段中没有 null 值(t2.d2 字段在表定义中 not null)查询可以修改为

  ```
  SELECT * FROM t1 WHERE  NOT EXISTS (SELECT * FROM t2 WHERE t1.c1=t2.d2);
  ```

  产生的计划如下：

  ```
  QUERY PLAN
  ------------------------------
  Hash Anti Join
  Hash Cond: (t1.c1 = t2.d2)
  ->  Seq Scan on t1
  ->  Hash
  ->  Seq Scan on t2
  (5 rows)
  ```

- 选择 hashagg。

  查询中 GROUP BY 语句如果生成了 groupagg+sort 的 plan 性能会比较差，可以通过加大 work_mem 的方法生成 hashagg 的 plan，因为不用排序而提高性能。

- 尝试将函数替换为 case 语句。

  Vastbase 函数调用性能较低，如果出现过多的函数调用导致性能下降很多，可以根据情况把可下推函数的函数改成 CASE 表达式。

- 避免对索引使用函数或表达式运算。

对索引使用函数或表达式运算会停止使用索引转而执行全表扫描。

- 尽量避免在 where 子句中使用!=或<>操作符、null 值判断、or 连接、参数隐式转换。
- 对复杂 SQL 语句进行拆分。

  对于过于复杂并且不易通过以上方法调整性能的 SQL 可以考虑拆分的方法，把 SQL 中某一部分拆分成独立的 SQL 并把执行结果存入临时表，拆分常见的场景包括但不限于：

  - 作业中多个 SQL 有同样的子查询，并且子查询数据量较大。
  - Plan cost 计算不准，导致子查询 hash bucket 太小，比如实际数据 1000W 行，hash bucket 只有 1000。
  - 函数（如 substr,to_number）导致大数据量子查询选择度计算不准。

9.4.8. SQL 调优关键参数调整

本节将介绍影响 Vastbase SQL 调优性能的关键数据库主节点配置参数，配置方法参见 10 配置运行参数。

表 9-4. 数据库主节点配置参数

参数/参考值	描述
enable_nestloop=on	控制查询优化器对嵌套循环连接(Nest Loop Join)类型的使用。当设置为“on”后，优化器优先使用 Nest Loop Join；当设置为“off”后，优化器在存在其他方法时将优先选择其他方法。 说明 如果只需要在当前数据库连接(即当前 Session)中临时更改该参数值，则只需要在 SQL 语句中执行如下命令： SET enable_nestloop to off; 此参数默认设置为“on”，但实际调优中应根据情况选择是否关闭。一般情况下，在三种 join 方式（Nested Loop、Merge Join 和 Hash Join）里，Nested Loop 性能较差，实际调优中可以选择关闭。
enable_bitmapscan=on	控制查询优化器对位图扫描规划类型的使用。设置为“on”，表示使用；设置为“off”，表示不使用。 说明 如果只需要在当前数据库连接(即当前 Session)中临时更改该参数值，则只需要在 SQL 语句中执行命令如下命令： SET enable_bitmapscan to off; bitmapscan 扫描方式适用于“where a > 1 and b > 1”且 a 列和 b 列都有索引这种查询条件，但有时其性能不如 indexscan。因此，现场调优如发现查询

参数/参考值	描述
	性能较差且计划中有 bitmapscan 算子，可以关闭 bitmapscan，看性能是否有提升。
enable_hashagg=on	控制优化器对 Hash 聚集规划类型的使用。
enable_hashjoin=on	控制优化器对 Hash 连接规划类型的使用。
enable_mergejoin=on	控制优化器对融合连接规划类型的使用。
enable_indexscan=on	控制优化器对索引扫描规划类型的使用。
enable_indexonlyscan=on	控制优化器对仅索引扫描规划类型的使用。
enable_seqscan=on	控制优化器对顺序扫描规划类型的使用。完全消除顺序扫描是不可能的，但是关闭这个变量会让优化器在存在其他方法的时候优先选择其他方法。
enable_sort=on	控制优化器使用的排序步骤。该设置不可能完全消除明确的排序，但是关闭这个变量可以让优化器在存在其他方法的时候优先选择其他方法。
rewrite_rule	控制优化器是否启用 LAZY_AGG 和 MAGIC_SET 重写规则。

9.4.9.使用 Plan Hint 进行调优

9.4.9.1. Plan Hint 调优概述

Plan Hint 为用户提供了直接影响执行计划生成的手段，用户可以通过指定 join 顺序，join、scan 方法，指定结果行数，等多个手段来进行执行计划的调优，以提升查询的性能。

功能描述

Plan Hint 仅支持在 SELECT 关键字后通过如下形式指定：

```
/*+ <plan hint>*/
```

可以同时指定多个 hint，之间使用空格分隔。hint 只能 hint 当前层的计划，对于子查询计划的 hint，需要在子查询的 select 关键字后指定 hint。

例如：

```
select /*+ <plan_hint1> <plan_hint2> */ * from t1, (select /*+ <plan_hint3> */ from t2) where 1=1;
```

其中<plan_hint1>，<plan_hint2>为外层查询的 hint，<plan_hint3>为内层子查询的 hint。

须知

如果在视图定义（CREATE VIEW）时指定 hint，则在该视图每次被应用时会使用该 hint。

当使用 random plan 功能（参数 plan_mode_seed 不为 0）时，查询指定的 plan hint 不会被使用。

支持范围

当前版本 Plan Hint 支持的范围如下，后续版本会进行增强。

- 指定 Join 顺序的 Hint - leading hint
- 指定 Join 方式的 Hint，仅支持除 semi/anti join，unique plan 之外的常用 hint。
- 指定结果集行数的 Hint
- 指定 Scan 方式的 Hint，仅支持常用的 tablescan，indexscan 和 indexonlyscan 的 hint。
- 指定子链接块名的 Hint

注意事项

不支持 Agg、Sort、Setop 和 Subplan 的 hint。

示例

本章节使用同一个语句进行示例，便于 Plan Hint 支持的各方法作对比，示例语句及不带 hint 的原计划如下所示：

```
create table store
(
   s_store_sk             integer            not null,
   s_store_id             char(16)           not null,
   s_rec_start_date        date                      ,
   s_rec_end_date          date                     ,
   s_closed_date_sk        integer                   ,
   s_store_name           varchar(50)                ,
   s_number_employees      integer                   ,
   s_floor_space          integer                   ,
   s_hours                char(20)                  ,
   s_manager              varchar(40)               ,
   s_market_id            integer                   ,
   s_geography_class       varchar(100)               ,
   s_market_desc          varchar(100)               ,
   s_market_manager        varchar(40)               ,
   s_division_id          integer                   ,
   s_division_name         varchar(50)               ,
   s_company_id           integer                   ,
   s_company_name          varchar(50)               ,
   s_street_number         varchar(10)               ,
   s_street_name          varchar(60)               ,
   s_street_type          char(15)                  ,
   s_suite_number          char(10)                  ,
   s_city                 varchar(60)               ,
   s_county               varchar(30)               ,
```

```
    s_state                char(2)                    ,
    s_zip                  char(10)                   ,
    s_country              varchar(20)                ,
    s_gmt_offset           decimal(5,2)               ,
    s_tax_precentage       decimal(5,2)               ,
    primary key (s_store_sk)
);
create table store_sales
(
    ss_sold_date_sk         integer                   ,
    ss_sold_time_sk         integer                   ,
    ss_item_sk              integer            not null,
    ss_customer_sk          integer                   ,
    ss_cdemo_sk             integer                   ,
    ss_hdemo_sk             integer                   ,
    ss_addr_sk              integer                   ,
    ss_store_sk             integer                   ,
    ss_promo_sk             integer                   ,
    ss_ticket_number        integer            not null,
    ss_quantity             integer                   ,
    ss_wholesale_cost       decimal(7,2)              ,
    ss_list_price           decimal(7,2)              ,
    ss_sales_price          decimal(7,2)              ,
    ss_ext_discount_amt     decimal(7,2)              ,
    ss_ext_sales_price      decimal(7,2)              ,
    ss_ext_wholesale_cost   decimal(7,2)              ,
    ss_ext_list_price       decimal(7,2)              ,
    ss_ext_tax              decimal(7,2)              ,
    ss_coupon_amt           decimal(7,2)              ,
    ss_net_paid             decimal(7,2)              ,
    ss_net_paid_inc_tax     decimal(7,2)              ,
    ss_net_profit           decimal(7,2)              ,
    primary key (ss_item_sk, ss_ticket_number)
);
create table store_returns
(
    sr_returned_date_sk     integer                   ,
    sr_return_time_sk       integer                   ,
    sr_item_sk              integer            not null,
    sr_customer_sk          integer                   ,
    sr_cdemo_sk             integer                   ,
    sr_hdemo_sk             integer                   ,
    sr_addr_sk              integer                   ,
    sr_store_sk             integer                   ,
    sr_reason_sk            integer                   ,
    sr_ticket_number        integer            not null,
    sr_return_quantity      integer                   ,
    sr_return_amt           decimal(7,2)              ,
    sr_return_tax           decimal(7,2)              ,
    sr_return_amt_inc_tax   decimal(7,2)              ,
    sr_fee                  decimal(7,2)              ,
    sr_return_ship_cost     decimal(7,2)              ,
    sr_refunded_cash        decimal(7,2)              ,
    sr_reversed_charge      decimal(7,2)              ,
    sr_store_credit         decimal(7,2)              ,
```

```
   sr_net_loss             decimal(7,2)               ,
   primary key (sr_item_sk, sr_ticket_number)
);
create table customer
(
   c_customer_sk           integer             not null,
   c_customer_id           char(16)            not null,
   c_current_cdemo_sk       integer                    ,
   c_current_hdemo_sk       integer                    ,
   c_current_addr_sk       integer                    ,
   c_first_shipto_date_sk   integer                    ,
   c_first_sales_date_sk    integer                    ,
   c_salutation            char(10)                   ,
   c_first_name            char(20)                   ,
   c_last_name             char(30)                   ,
   c_preferred_cust_flag    char(1)                    ,
   c_birth_day             integer                    ,
   c_birth_month           integer                    ,
   c_birth_year            integer                    ,
   c_birth_country         varchar(20)                ,
   c_login                 char(13)                   ,
   c_email_address         char(50)                   ,
   c_last_review_date      char(10)                   ,
   primary key (c_customer_sk)
);
create table promotion
(
   p_promo_sk              integer             not null,
   p_promo_id              char(16)            not null,
   p_start_date_sk         integer                    ,
   p_end_date_sk           integer                    ,
   p_item_sk               integer                    ,
   p_cost                  decimal(15,2)              ,
   p_response_target       integer                    ,
   p_promo_name            char(50)                   ,
   p_channel_dmail         char(1)                    ,
   p_channel_email         char(1)                    ,
   p_channel_catalog       char(1)                    ,
   p_channel_tv            char(1)                    ,
   p_channel_radio         char(1)                    ,
   p_channel_press         char(1)                    ,
   p_channel_event         char(1)                    ,
   p_channel_demo          char(1)                    ,
   p_channel_details       varchar(100)               ,
   p_purpose               char(15)                   ,
   p_discount_active       char(1)                    ,
   primary key (p_promo_sk)
);
create table customer_address
(
   ca_address_sk           integer             not null,
   ca_address_id           char(16)            not null,
   ca_street_number        char(10)                   ,
   ca_street_name          varchar(60)                ,
   ca_street_type          char(15)                   ,
```

```
   ca_suite_number         char(10)                  ,
   ca_city                 varchar(60)               ,
   ca_county               varchar(30)               ,
   ca_state                char(2)                   ,
   ca_zip                  char(10)                  ,
   ca_country              varchar(20)               ,
   ca_gmt_offset           decimal(5,2)              ,
   ca_location_type        char(20)                  ,
   primary key (ca_address_sk)
);
create table item
(
   i_item_sk               integer            not null,
   i_item_id               char(16)           not null,
   i_rec_start_date        date                      ,
   i_rec_end_date          date                      ,
   i_item_desc             varchar(200)              ,
   i_current_price         decimal(7,2)              ,
   i_wholesale_cost        decimal(7,2)              ,
   i_brand_id              integer                   ,
   i_brand                 char(50)                  ,
   i_class_id              integer                   ,
   i_class                 char(50)                  ,
   i_category_id           integer                   ,
   i_category              char(50)                  ,
   i_manufact_id           integer                   ,
   i_manufact              char(50)                  ,
   i_size                  char(20)                  ,
   i_formulation           char(20)                  ,
   i_color                 char(20)                  ,
   i_units                 char(10)                  ,
   i_container             char(10)                  ,
   i_manager_id            integer                   ,
   i_product_name          char(50)                  ,
   primary key (i_item_sk)
);
explain
select i_product_name product_name
,i_item_sk item_sk
,s_store_name store_name
,s_zip store_zip
,ad2.ca_street_number c_street_number
,ad2.ca_street_name c_street_name
,ad2.ca_city c_city
,ad2.ca_zip c_zip
,count(*) cnt
,sum(ss_wholesale_cost) s1
,sum(ss_list_price) s2
,sum(ss_coupon_amt) s3
FROM   store_sales
,store_returns
,store
,customer
,promotion
,customer_address ad2
```

```
,item
WHERE  ss_store_sk = s_store_sk AND
ss_customer_sk = c_customer_sk AND
ss_item_sk = i_item_sk and
ss_item_sk = sr_item_sk and
ss_ticket_number = sr_ticket_number and
c_current_addr_sk = ad2.ca_address_sk and
ss_promo_sk = p_promo_sk and
i_color in ('maroon','burnished','dim','steel','navajo','chocolate') and
i_current_price between 35 and 35 + 10 and
i_current_price between 35 + 1 and 35 + 15
group by i_product_name
,i_item_sk
,s_store_name
,s_zip
,ad2.ca_street_number
,ad2.ca_street_name
,ad2.ca_city
,ad2.ca_zip
;
                                                                              QUERY PLAN
----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------
 HashAggregate  (cost=23.52..23.53 rows=1 width=880)
   Group By Key: item.i_product_name, item.i_item_sk, store.s_store_name, store.s_zip, ad2.ca_street_number, ad2.ca_street_name, ad2.ca_city, ad2.ca_zip
   ->  Nested Loop  (cost=4.27..23.49 rows=1 width=776)
         ->  Nested Loop  (cost=4.27..22.80 rows=1 width=416)
               ->  Nested Loop  (cost=4.27..22.39 rows=1 width=420)
                     ->  Nested Loop  (cost=4.27..21.98 rows=1 width=420)
                           ->  Nested Loop  (cost=4.27..21.57 rows=1 width=262)
                                 Join Filter: (item.i_item_sk = store_sales.ss_item_sk)
                                 ->  Nested Loop  (cost=4.27..20.78 rows=2 width=216)
                                       ->  Seq Scan on item  (cost=0.00..11.16 rows=1 width=208)
                                             Filter: ((i_current_price >= 35::numeric) AND (i_current_price <= 45::numeric) AND (i_current_price >= 36::numeric) AND (i_current_price <= 50::numeric) A
ND (i_color = ANY ('{maroon,burnished,dim,steel,navajo,chocolate}'::bpchar[])))
                                       ->  Bitmap Heap Scan on store_returns  (cost=4.27..9.61 rows=2 width=8)
                                             Recheck Cond: (sr_item_sk = item.i_item_sk)
                                             ->  Bitmap Index Scan on store_returns_pkey  (cost=0.00..4.27 rows=2 width=0)
                                                   Index Cond: (sr_item_sk = item.i_item_sk)
                                 ->  Index Scan using store_sales_pkey on store_sales  (cost=0.00..0.38 rows=1 width=62)
                                       Index Cond: ((ss_item_sk = store_returns.sr_item_sk) AND (ss_ticket_number = store_returns.sr_ticket_number))
                           ->  Index Scan using store_pkey on store  (cost=0.00..0.40 rows=1 width=166)
                                 Index Cond: (s_store_sk = store_sales.ss_store_sk)
                     ->  Index Scan using customer_pkey on customer  (cost=0.00..0.40 rows=1 width=8)
                           Index Cond: (c_customer_sk = store_sales.ss_customer_sk)
               ->  Index Only Scan using promotion_pkey on promotion  (cost=0.00..0.40 rows=1 width=4)
                     Index Cond: (p_promo_sk = store_sales.ss_promo_sk)
         ->  Index Scan using customer_address_pkey on customer_address ad2  (cost=0.00..0.68 rows=1 width=368)
               Index Cond: (ca_address_sk = customer.c_current_addr_sk)
(25 rows)
```

9.4.9.2. Join 顺序的 Hint

功能描述

指明 join 的顺序，包括内外表顺序。

语法格式

- ❖ 仅指定 join 顺序，不指定内外表顺序。

```
leading(join_table_list)
```

- ❖ 同时指定 join 顺序和内外表顺序，内外表顺序仅在最外层生效。

```
leading((join_table_list))
```

参数说明

join_table_list 为表示表 join 顺序的 hint 字符串，可以包含当前层的任意个表（别名），或对于子查询提升的场景，也可以包含子查询的 hint 别名，同时任意表可以使用括号指定优先级，表之间使用空格分隔。

须知

表只能用单个字符串表示，不能带 schema。

表如果存在别名，需要优先使用别名来表示该表。

join table list 中指定的表需要满足以下要求，否则会报语义错误。

- list 中的表必须在当前层或提升的子查询中存在。
- list 中的表在当前层或提升的子查询中必须是唯一的。如果不唯一，需要使用不同的别名进行区分。
- 同一个表只能在 list 里出现一次。
- 如果表存在别名，则 list 中的表需要使用别名。

例如：

leading(t1 t2 t3 t4 t5)表示：t1, t2, t3, t4, t5 先 join，五表 join 顺序及内外表不限。

leading((t1 t2 t3 t4 t5))表示：t1 和 t2 先 join，t2 做内表；再和 t3 join，t3 做内表；再和 t4 join，t4 做内表；再和 t5 join，t5 做内表。

leading(t1 (t2 t3 t4) t5)表示：t2, t3, t4 先 join，内外表不限；再和 t1, t5 join，内外表不限。

leading((t1 (t2 t3 t4) t5))表示：t2, t3, t4 先 join，内外表不限；在最外层，t1 再和 t2, t3, t4 的 join 表 join，t1 为外表，再和 t5 join，t5 为内表。

leading((t1 (t2 t3) t4 t5)) leading((t3 t2))表示：这里第一个 leading 指定 5 张表 join 的顺序，第二个 leading 指定 t2, t3 join 的内外表关系，所以这里是 t2, t3 先 join，t2 做内表；然后再和 t1 join，t2, t3 的 join 表做内表；然后再依次跟 t4, t5 做 join，t4, t5 做内表。

示例

对示例中原语句使用如下 hint:

```
explain
select /*+ leading((((((store_sales store) promotion) item) customer) ad2) store_returns)
leading((store store_sales))*/ i_product_name product_name ...
```

该 hint 表示：表之间的 join 关系是：store_sales 和 store 先 join， store_sales 做内表，然后依次跟 promotion, item, customer, ad2, store_returns 做 join。生成计划如下所示：

```
WARNING:  Duplicated or conflict hint: Leading(store_sales store), will be discarded.
                                                                QUERY PLAN
-----------------------------------------------------------------------------------------------------------------------------------------------------
----------------------------------------------------------------------
 HashAggregate  (cost=55.24..55.25 rows=1 width=880)
   Group By Key: item.i_product_name, item.i_item_sk, store.s_store_name, store.s_zip, ad2.ca_street_number, ad2.ca_street_name, ad2.ca_city, ad2.ca_zip
   ->  Nested Loop  (cost=29.93..55.21 rows=1 width=776)
         ->  Nested Loop  (cost=29.93..54.80 rows=1 width=784)
               ->  Nested Loop  (cost=29.93..54.11 rows=1 width=424)
                     ->  Nested Loop  (cost=29.93..53.70 rows=1 width=424)
                           Join Filter: (store_sales.ss_item_sk = item.i_item_sk)
                           ->  Seq Scan on item  (cost=0.00..11.16 rows=1 width=208)
                                 Filter: ((i_current_price >= 35::numeric) AND (i_current_price <= 45::numeric) AND (i_current_price >= 36::numeric) AND (i_current_price <= 50::numeric) AND (i_color
= ANY ('{maroon,burnished,dim,steel,navajo,chocolate}'::bpchar[])))
                           ->  Hash Join  (cost=29.93..41.99 rows=44 width=216)
                                 Hash Cond: (promotion.p_promo_sk = store_sales.ss_promo_sk)
                                 ->  Seq Scan on promotion  (cost=0.00..11.18 rows=118 width=4)
                                 ->  Hash  (cost=29.00..29.00 rows=74 width=220)
                                       ->  Hash Join  (cost=17.61..29.00 rows=74 width=220)
                                             Hash Cond: (store.s_store_sk = store_sales.ss_store_sk)
                                             ->  Seq Scan on store  (cost=0.00..10.44 rows=44 width=166)
                                             ->  Hash  (cost=13.38..13.38 rows=338 width=62)
                                                   ->  Seq Scan on store_sales  (cost=0.00..13.38 rows=338 width=62)
                     ->  Index Scan using customer_pkey on customer  (cost=0.00..0.40 rows=1 width=8)
                           Index Cond: (c_customer_sk = store_sales.ss_customer_sk)
               ->  Index Scan using customer_address_pkey on customer_address ad2  (cost=0.00..0.68 rows=1 width=368)
                     Index Cond: (ca_address_sk = customer.c_current_addr_sk)
         ->  Index Only Scan using store_returns_pkey on store_returns  (cost=0.00..0.41 rows=1 width=8)
               Index Cond: ((sr_item_sk = store_sales.ss_item_sk) AND (sr_ticket_number = store_sales.ss_ticket_number))
(24 rows)
```

图中计划顶端 warning 的提示详见 9.4.9.7Hint 的错误、冲突及告警的说明。

9.4.9.3. Join 方式的 Hint

功能描述

指明 Join 使用的方法，可以为 Nested Loop，Hash Join 和 Merge Join。

语法格式

```
[no] nestloop|hashjoin|mergejoin(table_list)
```

参数说明

- ❖ no 表示 hint 的 join 方式不使用。
- ❖ table_list 为表示 hint 表集合的字符串，该字符串中的表与 join_table_list 相同，只是中间不允许出现括号指定 join 的优先级。

例如：

no nestloop(t1 t2 t3)表示：生成 t1，t2，t3 三表连接计划时，不使用 nestloop。三表连接计划可能是 t2 t3 先 join，再跟 t1 join，或 t1 t2 先 join，再跟 t3 join。此 hint 只 hint 最后一次 join 的 join 方式，对于两表连接的方法不 hint。如果需要，可以单独指定，例如：任意表均不允许 nestloop 连接，且希望 t2 t3 先 join，则增加 hint：no nestloop(t2 t3)。

步骤 1 示例

对示例中原语句使用如下 hint:

```
explain
select /*+ nestloop(store_sales store_returns item) */ i_product_name product_name ...
```

该 hint 表示：生成 store_sales，store_returns 和 item 三表的结果集时，最后的两表关联使用 nestloop。生成计划如下所示：

```
                                                  QUERY PLAN
-----------------------------------------------------------------------------------------------------------------------
 HashAggregate  (cost=23.52..23.53 rows=1 width=880)
   Group By Key: item.i_product_name, item.i_item_sk, store.s_store_name, store.s_zip, ad2.ca_street_number, ad2.ca_street_name, ad2.ca_city, ad2.ca_zip
   ->  Nested Loop  (cost=4.27..23.49 rows=1 width=776)
         ->  Nested Loop  (cost=4.27..22.80 rows=1 width=416)
               ->  Nested Loop  (cost=4.27..22.39 rows=1 width=420)
                     ->  Nested Loop  (cost=4.27..21.98 rows=1 width=420)
                           ->  Nested Loop  (cost=4.27..21.57 rows=1 width=262)
                                 Join Filter: (item.i_item_sk = store_sales.ss_item_sk)
                                 ->  Nested Loop  (cost=4.27..20.78 rows=2 width=216)
                                       ->  Seq Scan on item  (cost=0.00..11.16 rows=1 width=208)
                                             Filter: ((i_current_price >= 35::numeric) AND (i_current_price <= 45::numeric) AND (i_current_price >= 36::numeric) AND (i_current_price <= 50::numeric) A
ND (i_color = ANY ('{maroon,burnished,dim,steel,navajo,chocolate}'::bpchar[])))
                                       ->  Bitmap Heap Scan on store_returns  (cost=4.27..9.61 rows=2 width=8)
                                             Recheck Cond: (sr_item_sk = item.i_item_sk)
                                             ->  Bitmap Index Scan on store_returns_pkey  (cost=0.00..4.27 rows=2 width=0)
                                                   Index Cond: (sr_item_sk = item.i_item_sk)
                                 ->  Index Scan using store_sales_pkey on store_sales  (cost=0.00..0.38 rows=1 width=62)
                                       Index Cond: ((ss_item_sk = store_returns.sr_item_sk) AND (ss_ticket_number = store_returns.sr_ticket_number))
                           ->  Index Scan using store_pkey on store  (cost=0.00..0.40 rows=1 width=166)
                                 Index Cond: (s_store_sk = store_sales.ss_store_sk)
                     ->  Index Scan using customer_pkey on customer  (cost=0.00..0.40 rows=1 width=8)
                           Index Cond: (c_customer_sk = store_sales.ss_customer_sk)
               ->  Index Only Scan using promotion_pkey on promotion  (cost=0.00..0.40 rows=1 width=4)
                     Index Cond: (p_promo_sk = store_sales.ss_promo_sk)
         ->  Index Scan using customer_address_pkey on customer_address ad2  (cost=0.00..0.68 rows=1 width=368)
               Index Cond: (ca_address_sk = customer.c_current_addr_sk)
(25 rows)
```

9.4.9.4. 行数的 Hint

功能描述

指明中间结果集的大小，支持绝对值和相对值的 hint。

语法格式

```
rows(table_list #|+|-|* const)
```

参数说明

- #,+,-,*，进行行数估算 hint 的四种操作符号。#表示直接使用后面的行数进行 hint。+,-,*表示对原来估算的行数进行加、减、乘操作，运算后的行数最小值为 1 行。table_list 为 hint 对应的单表或多表 join 结果集，与 9.4.9.3Join 方式的 Hint 中 table_list 相同。
- const 可以是任意非负数，支持科学计数法。

例如：

rows(t1 #5)表示：指定 t1 表的结果集为 5 行。

rows(t1 t2 t3 *1000)表示：指定 t1, t2, t3 join 完的结果集的行数乘以 1000。

建议

- 推荐使用两个表*的 hint。对于两个表的采用*操作符的 hint，只要两个表出现在 join 的两端，都会触发 hint。例如：设置 hint 为 rows(t1 t2 * 3)，对于(t1 t3 t4)和(t2 t5 t6)join 时，由于 t1 和 t2 出现在 join 的两端，所以其 join 的结果集也会应用该 hint 规则乘以 3。
- rows hint 支持在单表、多表、function table 及 subquery scan table 的结果集上指定 hint。

示例

对示例中原语句使用如下 hint:

```
explain
select /*+ rows(store_sales store_returns *50) */ i_product_name product_name ...
```

该 hint 表示：store_sales，store_returns 关联的结果集估算行数在原估算行数基础上乘以 50。生成计划如下所示：

```
                                                                                             QUERY PLAN
---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------
-------------------------------------------------------------------------------
 HashAggregate  (cost=23.52..23.53 rows=1 width=880)
   Group By Key: item.i_product_name, item.i_item_sk, store.s_store_name, store.s_zip, ad2.ca_street_number, ad2.ca_street_name, ad2.ca_city, ad2.ca_zip
   ->  Nested Loop  (cost=4.27..23.49 rows=1 width=776)
         ->  Nested Loop  (cost=4.27..22.80 rows=1 width=416)
               ->  Nested Loop  (cost=4.27..22.39 rows=1 width=420)
                     ->  Nested Loop  (cost=4.27..21.98 rows=1 width=420)
                           ->  Nested Loop  (cost=4.27..21.57 rows=1 width=262)
                                 Join Filter: (item.i_item_sk = store_sales.ss_item_sk)
                                 ->  Nested Loop  (cost=4.27..20.78 rows=2 width=216)
                                       ->  Seq Scan on item  (cost=0.00..11.16 rows=1 width=208)
                                             Filter: ((i_current_price >= 35::numeric) AND (i_current_price <= 45::numeric) AND (i_current_price >= 36::numeric) AND (i_current_price <= 50::numeric) A
ND (i_color = ANY ('{maroon,burnished,dim,steel,navajo,chocolate}'::bpchar[])))
                                       ->  Bitmap Heap Scan on store_returns  (cost=4.27..9.61 rows=2 width=8)
                                             Recheck Cond: (sr_item_sk = item.i_item_sk)
                                             ->  Bitmap Index Scan on store_returns_pkey  (cost=0.00..4.27 rows=2 width=0)
                                                   Index Cond: (sr_item_sk = item.i_item_sk)
                                 ->  Index Scan using store_sales_pkey on store_sales  (cost=0.00..0.38 rows=1 width=62)
                                       Index Cond: ((ss_item_sk = store_returns.sr_item_sk) AND (ss_ticket_number = store_returns.sr_ticket_number))
                           ->  Index Scan using store_pkey on store  (cost=0.00..0.40 rows=1 width=166)
                                 Index Cond: (s_store_sk = store_sales.ss_store_sk)
                     ->  Index Scan using customer_pkey on customer  (cost=0.00..0.40 rows=1 width=8)
                           Index Cond: (c_customer_sk = store_sales.ss_customer_sk)
               ->  Index Only Scan using promotion_pkey on promotion  (cost=0.00..0.40 rows=1 width=4)
                     Index Cond: (p_promo_sk = store_sales.ss_promo_sk)
         ->  Index Scan using customer_address_pkey on customer_address ad2  (cost=0.00..0.68 rows=1 width=368)
               Index Cond: (ca_address_sk = customer.c_current_addr_sk)
(25 rows)
```

9.4.9.5. Scan 方式的 Hint

功能描述

指明 scan 使用的方法，可以是 tablescan、indexscan 和 indexonlyscan。

语法格式

```
[no] tablescan|indexscan|indexonlyscan(table [index])
```

参数说明

- ❖ no 表示 hint 的 scan 方式不使用。
- ❖ table 表示 hint 指定的表，只能指定一个表，如果表存在别名应优先使用别名进行 hint。
- ❖ index 表示使用 indexscan 或 indexonlyscan 的 hint 时，指定的索引名称，当前只能指定一个。

说明

对于 indexscan 或 indexonlyscan，只有 hint 的索引属于 hint 的表时，才能使用该 hint。

scan hint 支持在行列存表、obs 表、子查询表上指定。

示例

为了 hint 使用索引扫描，需要首先在表 item 的 i_item_sk 列上创建索引，名称为 i。

```
create index i on item(i_item_sk);
```

对示例中原语句使用如下 hint:

```
explain
select /*+ indexscan(item i) */ i_product_name product_name ...
```

该 hint 表示：item 表使用索引 i 进行扫描。生成计划如下所示：

```
                                                                 QUERY PLAN
------------------------------------------------------------------------------------------------------------------------
 HashAggregate  (cost=38.79..38.80 rows=1 width=880)
   Group By Key: item.i_product_name, item.i_item_sk, store.s_store_name, store.s_zip, ad2.ca_street_number, ad2.ca_street_name, ad2.ca_city, ad2.ca_zip
   ->  Nested Loop  (cost=18.45..38.76 rows=1 width=776)
         ->  Nested Loop  (cost=18.45..38.07 rows=1 width=416)
               ->  Nested Loop  (cost=18.45..37.66 rows=1 width=420)
                     ->  Nested Loop  (cost=18.45..37.25 rows=1 width=420)
                           ->  Nested Loop  (cost=18.45..36.84 rows=1 width=262)
                                 Join Filter: (store_sales.ss_item_sk = item.i_item_sk)
                                 ->  Hash Join  (cost=18.45..35.62 rows=3 width=62)
                                       Hash Cond: ((store_returns.sr_item_sk = store_sales.ss_item_sk) AND (store_returns.sr_ticket_number = store_sales.ss_ticket_number))
                                       ->  Seq Scan on store_returns  (cost=0.00..14.08 rows=408 width=8)
                                       ->  Hash  (cost=13.38..13.38 rows=338 width=62)
                                             ->  Seq Scan on store_sales  (cost=0.00..13.38 rows=338 width=62)
                                 ->  Index Scan using i on item  (cost=0.00..0.40 rows=1 width=208)
                                       Index Cond: (i_item_sk = store_returns.sr_item_sk)
                                       Filter: ((i_current_price >= 35::numeric) AND (i_current_price <= 45::numeric) AND (i_current_price >= 36::numeric) AND (i_current_price <= 50::numeric) AND (i_
color = ANY ('{maroon,burnished,dim,steel,navajo,chocolate}'::bpchar[])))
                           ->  Index Scan using store_pkey on store  (cost=0.00..0.40 rows=1 width=166)
                                 Index Cond: (s_store_sk = store_sales.ss_store_sk)
                     ->  Index Scan using customer_pkey on customer  (cost=0.00..0.40 rows=1 width=8)
                           Index Cond: (c_customer_sk = store_sales.ss_customer_sk)
               ->  Index Only Scan using promotion_pkey on promotion  (cost=0.00..0.40 rows=1 width=4)
                     Index Cond: (p_promo_sk = store_sales.ss_promo_sk)
         ->  Index Scan using customer_address_pkey on customer_address ad2  (cost=0.00..0.68 rows=1 width=368)
               Index Cond: (ca_address_sk = customer.c_current_addr_sk)
(24 rows)
```

9.4.9.6. 子链接块名的 hint

功能描述

指明子链接块的名称。

语法格式

```
blockname (table)
```

参数说明

- ❖ table 表示为该子链接块 hint 的别名的名称。

说明

- blockname hint 仅在对应的子链接块提升时才会被上层查询使用。目前支持的子链接提升包括 IN 子链接提升、EXISTS 子链接提升和包含 Agg 等值相关子链接提升。该 hint 通常会和前面章节提到的 hint 联合使用。

- 对于 FROM 关键字后的子查询，则需要使用子查询的别名进行 hint，blockname hint 不会被用到。
- 如果子链接中含有多个表，则提升后这些表可与外层表以任意优化顺序连接，hint 也不会被用到。

示例

```
explain select /*+nestloop(store_sales tt) */ * from store_sales where ss_item_sk in (select
/*+blockname(tt)*/ i_item_sk from item group by 1);
```

该 hint 表示：子链接的别名为 tt，提升后与上层的 store_sales 表关联时使用 nestloop。生成计划如下所示：

```
                                    QUERY PLAN
-----------------------------------------------------------------------------------------
 Nested Loop  (cost=10.53..68.39 rows=169 width=212)
   ->  HashAggregate  (cost=10.53..10.95 rows=42 width=4)
         Group By Key: item.i_item_sk
         ->  Seq Scan on item  (cost=0.00..10.42 rows=42 width=4)
   ->  Index Scan using store_sales_pkey on store_sales  (cost=0.00..1.34 rows=2 width=212)
         Index Cond: (ss_item_sk = item.i_item_sk)
(6 rows)
```

9.4.9.7. Hint 的错误、冲突及告警

Plan Hint 的结果会体现在计划的变化上，可以通过 explain 来查看变化。

Hint 中的错误不会影响语句的执行，只是不能生效，该错误会根据语句类型以不同方式提示用户。对于 explain 语句，hint 的错误会以 warning 形式显示在界面上，对于非 explain 语句，会以 debug1 级别日志显示在日志中，关键字为 PLANHINT。

hint 的错误分为以下类型：

- 语法错误

  语法规则树归约失败，会报错，指出出错的位置。

  例如：hint 关键字错误，leading hint 或 join hint 指定 2 个表以下，其它 hint 未指定表等。一旦发现语法错误，则立即终止 hint 的解析，所以此时只有错误前面的解析完的 hint 有效。

  例如：

  ```
  leading((t1 t2)) nestloop(t1) rows(t1 t2 #10)
  ```

  nestloop(t1)存在语法错误，则终止解析，可用 hint 只有之前解析的 leading((t1 t2))。

- 语义错误
  - 表不存在，存在多个，或在 leading 或 join 中出现多次，均会报语义错误。
  - scanhint 中的 index 不存在，会报语义错误。
  - 另外，如果子查询提升后，同一层出现多个名称相同的表，且其中某个表需要被 hint，hint 会存在歧义，无法使用，需要为相同表增加别名规避。
- hint 重复或冲突

  如果存在 hint 重复或冲突，只有第一个 hint 生效，其它 hint 均会失效，会给出提示。
  - hint 重复是指，hint 的方法及表名均相同。例如：nestloop(t1 t2) nestloop(t1 t2)。
  - hint 冲突是指，table list 一样的 hint，存在不一样的 hint，hint 的冲突仅对于每一类 hint

方法检测冲突。

例如：nestloop (t1 t2) hashjoin (t1 t2)，则后面与前面冲突，此时 hashjoin 的 hint 失效。注意：nestloop(t1 t2)和 no mergejoin(t1 t2)不冲突。

须知

leading hint 中的多个表会进行拆解。例如：leading ((t1 t2 t3))会拆解成：leading((t1 t2)) leading(((t1 t2) t3))，此时如果存在 leading((t2 t1))，则两者冲突，后面的会被丢弃。（例外：指定内外表的 hint 若与不指定内外表的 hint 重复，则始终丢弃不指定内外表的 hint。）

- 子链接提升后 hint 失效

  子链接提升后的 hint 失效，会给出提示。通常出现在子链接中存在多个表连接的场景。提升后，子链接中的多个表不再作为一个整体出现在 join 中。

- 列类型不支持重分布
  - 对于 skew hint 来说，目的是为了进行重分布时的调优，所以当 hint 列的类型不支持重分布时，hint 将无效。
- hint 未被使用
  - 非等值 join 使用 hashjoin hint 或 mergejoin hint
  - 不包含索引的表使用 indexscan hint 或 indexonlyscan hint
  - 通常只有在索引列上使用过滤条件才会生成相应的索引路径，全表扫描将不会使用索引，因此使用 indexscan hint 或 indexonlyscan hint 将不会使用
  - indexonlyscan 只有输出列仅包含索引列才会使用，否则指定时 hint 不会被使用
  - 多个表存在等值连接时，仅尝试有等值连接条件的表的连接，此时没有关联条件的表之间的路径将不会生成，所以指定相应的 leading，join，rows hint 将不使用，例如：t1 t2 t3 表 join，t1 和 t2, t2 和 t3 有等值连接条件，则 t1 和 t3 不会优先连接，leading(t1 t3)不会被使用。
  - 生成 stream 计划时，如果表的分布列与 join 列相同，则不会生成 redistribute 的计划；如果不同，且另一表分布列与 join 列相同，只能生成 redistribute 的计划，不会生成 broadcast 的计划，指定相应的 hint 则不会被使用。
  - 如果子链接未被提升，则 blockname hint 不会被使用。
  - 对于 skew hint，hint 未被使用可能由于：
    - 计划中不需要进行重分布。
    - hint 指定的列为包含分布键。
    - hint 指定倾斜信息有误或不完整，如对于 join 优化未指定值。
    - 倾斜优化的 GUC 参数处于关闭状态。

10. 配置运行参数

10.1. 查看参数当前取值

Vastbase 安装后，有一套默认的运行参数，为了使 Vastbase 与业务的配合度更高，用户需要根据业务场景和数据量的大小进行参数调整。

操作步骤

步骤 1 以操作系统用户 vastbase 登录数据库主节点。

步骤 2 使用如下命令连接数据库。

```
vsql -d vastbase -p 5432
```

vastbase 为需要连接的数据库名称，5432 为数据库主节点的端口号。

连接成功后，系统显示类似如下信息：

```
vsql ((Vastbase 2.2.0 build ) compiled at 2021-01-26 19:22:33 commit 0 last mr  )
Non-SSL connection (SSL connection is recommended when requiring high-security)
Type "help" for help.

vastbase=#
```

步骤 3 查看数据库运行参数当前取值。

- 方法一：使用 SHOW 命令。
  - 使用如下命令查看单个参数：

    ```
    vastbase=# SHOW server_version;
    ```

    server_version 显示数据库版本信息的参数。

  - 使用如下命令查看所有参数：

    ```
    vastbase=#  SHOW ALL;
    ```

    使用此命令输出结果较多，会分屏显示，可以按空格键分页，按 Q 键退出。

- 方法二：使用 pg_settings 视图。
  - 使用如下命令查看单个参数：

    ```
    vastbase=# SELECT * FROM pg_settings WHERE NAME='server_version';
    ```

  - 使用如下命令查看所有参数：

    ```
    vastbase=# SELECT * FROM pg_settings;
    ```

示例

查看服务器的版本号。

```
vastbase=# SHOW server_version;
 server_version
```

```
---------------
 9.2.4
(1 row)
```

10.2. 重设参数

背景信息

Vastbase 提供了多种修改参数的方法，用户可以方便的针对数据库、用户、会话进行设置。

- 参数名称不区分大小写。
- 参数取值有整型、浮点型、字符串、布尔型和枚举型五类。
  - 布尔值可以是（on，off）、（true，false）、（yes，no）或者（1，0），且不区分大小写。
  - 枚举类型的取值是在系统表 pg_settings 的 enumvals 字段取值定义的。
- 对于有单位的参数，在设置时请指定单位，否则将使用默认的单位。
  - 参数的默认单位在系统表 pg_settings 的 unit 字段定义的。
  - 内存单位有：KB（千字节）、MB（兆字节）和 GB（吉字节）。
  - 时间单位：ms（毫秒）、s（秒）、min（分钟）、h（小时）和 d（天）。

具体参数说明请参见 19GUC 参数说明。

参数设置

Vastbase 提供了六类参数，具体分类和设置方式请参考表 10-1：

表 10-1. 参数分类

参数类型	说明	设置方式
INTERNAL	固定参数，在创建数据库的时候确定，用户无法修改，只能通过 show 语法或者 pg_settings 视图进行查看。	无
POSTMASTER	数据库服务端参数，在数据库启动时确定，可以通过配置文件指定。	支持表 10-2 中的方式一。
SIGHUP	数据库全局参数，可在数据库启动时设置或者在数据库启动后，发送指令重新加载。	支持表 10-2 中的方式一、方式二。
BACKEND	会话连接参数。在创建会话连接时指定，连接建立后无法修改。连接断掉后参数失效。内部使用参数，不推荐用户设置。	支持表 10-2 中的方式一、方式二。 说明 设置该参数后，下一次建立会话连接时生效。
SUSET	数据库管理员参数。可在数据库启动时、数据	支持表 10-2 中的方式一、方式二设置。

参数类型	说明	设置方式
	库启动后或者数据库管理员通过 SQL 进行设置。	
USERSET	普通用户参数。可被任何用户在任何时刻设置。	支持表 10-2 中的方式一、方式二设置。

Vastbase 提供了两种方式来修改参数，具体操作请参考表 10-2：

表 10-2. 参数设置方式

序号	设置方法
方式一	1. 编辑参数文件 手工编辑参数文件 postgresql.conf 修改参数值，如果没有该参数则手动添加 2.重启数据库使参数生效。 说明： 重启 vastbase 操作会导致用户执行操作中断，请在执行之前规划好合适的执行窗口 vb_ctl stop && vb_ctl srart
方式二	修改指定数据库，用户，会话级别的参数。 • 设置数据库级别的参数 vastbase=# **ALTER DATABASE** *dbname* **SET** *paraname* **TO** *value;* 在下次会话中生效。 • 设置用户级别的参数 vastbase=# **ALTER USER** *username* **SET** *paraname* **TO** *value;* 在下次会话中生效。 • 设置会话级别的参数 vastbase=# **SET** *paraname* **TO** *value;* 修改本次会话中的取值。退出会话后，设置将失效。

操作步骤

使用方式一设置数据库参数，以在数据库主节点设置 archive_mode 参数为例。

步骤 1 以操作系统用户 vastbase 登录数据库主节点。

步骤 2 查看 archive_mode 参数。

```
cat /vastbase/data/dbnode/postgresql.conf | grep archive_mode
archive_mode = on
```

on 表示日志要进行归档操作。

步骤 3 设置 archive_mode 参数为 off，直接编辑参数文件 postgresql.conf,将参数设为 off 关闭日志的归档操作。

```
archive_mode=off
```

步骤 4 重启数据库使参数生效。

```
vb_ctl stop && vb_ctl start
```

步骤 5 检查参数设置的正确性。

```
vastbase=# SHOW archive_mode;
 archive_mode
--------------
 off
(1 row)
vastbase=#
```

使用方式二设置参数，以设置 explain_perf_mode 参数为例。

步骤 1 以操作系统用户 vastbase 登录数据库主节点。

步骤 2 使用如下命令连接数据库。

```
vsql -d vastbase -p 5432
```

vastbase 为需要连接的数据库名称，5432 为数据库主节点的端口号。

连接成功后，系统显示类似如下信息：

```
vsql ((Vastbase 2.2.0 build ) compiled at 2021-01-26 19:22:33 commit 0 last mr  )
Non-SSL connection (SSL connection is recommended when requiring high-security)
Type "help" for help.

vastbase=#
```

步骤 3 查看 explain_perf_mode 参数。

```
vastbase=# SHOW explain_perf_mode;
 explain_perf_mode
-------------------
 normal
(1 row)
```

步骤 4 设置 explain_perf_mode 参数。

使用以下任意方式进行设置：

- 设置数据库级别的参数

```
vastbase=# ALTER DATABASE vastbase SET explain_perf_mode TO pretty;
```

当结果显示为如下信息，则表示设置成功。

```
ALTER DATABASE
```

在下次会话中生效。

- 设置用户级别的参数

```
vastbase=# ALTER USER vuser SET explain_perf_mode TO pretty;
```

当结果显示为如下信息，则表示设置成功。

```
ALTER ROLE
```

在下次会话中生效。

- 设置会话级别的参数

```
vastbase=# SET explain_perf_mode TO pretty;
```

当结果显示为如下信息，则表示设置成功。

```
SET
```

步骤 5 检查参数设置的正确性。

```
vastbase=# SHOW explain_perf_mode;
 explain_perf_mode
--------------
 pretty
(1 row)
```

11. SQL 参考

11.1. Vastbase SQL

什么是 SQL

SQL 是用于访问和处理数据库的标准计算机语言。

SQL 提供了各种任务的语句，包括：

- 查询数据。
- 在表中插入，更新和删除行。
- 创建，替换，更改和删除对象。
- 控制对数据库及其对象的访问。
- 保证数据库的一致性和完整性。

SQL 语言由用于处理数据库和数据库对象的命令和函数组成。该语言还会强制实施有关数据类型、表达式和文本使用的规则。因此在 11SQL 参考章节，除了 SQL 语法参考外，还会看到有关数据类型、表达式、函数和操作符等信息。

SQL 发展简史

SQL 发展简史如下：

- 1986 年，ANSI X3.135-1986，ISO/IEC 9075:1986，SQL-86
- 1989 年，ANSI X3.135-1989，ISO/IEC 9075:1989，SQL-89
- 1992 年，ANSI X3.135-1992，ISO/IEC 9075:1992，SQL-92（SQL2）
- 1999 年，ISO/IEC 9075:1999，SQL:1999（SQL3）
- 2003 年，ISO/IEC 9075:2003，SQL:2003（SQL4）

- 2011 年，ISO/IEC 9075:200N，SQL:2011（SQL5）

Vastbase 支持的 SQL 标准

Vastbase 默认支持 SQL2、SQL3 和 SQL4 的主要特性。

11.2. 关键字

SQL 里有保留字和非保留字之分。根据标准，保留字决不能用做其他标识符。非保留字只是在特定的环境里有特殊的含义，而在其他环境里是可以用做标识符的。

标识符的命名需要遵守如下规范：

- 标识符需要为字母、下划线、数字（0-9）或美元符号（$）。
- 标识符必须以字母（a-z）或下划线（_）开头。

表 11-1. SQL 关键字

关键字	VASTBASE	SQL:1999	SQL-92
ABORT	非保留	-	-
ABS	-	非保留	-
ABSOLUTE	非保留	保留	保留
ACCESS	非保留	-	-
ACCOUNT	非保留	-	-
ACTION	非保留	保留	保留
ADA	-	非保留	非保留
ADD	非保留	保留	保留
ADMIN	非保留	保留	-
AFTER	非保留	保留	-
AGGREGATE	非保留	保留	-
ALIAS	-	保留	-
ALL	保留	保留	保留
ALLOCATE	-	保留	保留

关键字	VASTBASE	SQL:1999	SQL-92
ALSO	非保留	-	-
ALTER	非保留	保留	保留
ALWAYS	非保留	-	-
ANALYSE	保留	-	-
ANALYZE	保留	-	-
AND	保留	保留	保留
ANY	保留	保留	保留
APP	非保留	-	-
ARE	-	保留	保留
ARRAY	保留	保留	-
AS	保留	保留	保留
ASC	保留	保留	保留
ASENSITIVE	-	非保留	-
ASSERTION	非保留	保留	保留
ASSIGNMENT	非保留	非保留	-
ASYMMETRIC	保留	非保留	-
AT	非保留	保留	保留
ATOMIC	-	非保留	-
ATTRIBUTE	非保留	-	-
AUDIT	非保留	-	-
AUTHID	保留	-	-
AUTHORIZATION	保留(可以是函数或类型)	保留	保留
AUTOEXTEND	非保留	-	-

关键字	VASTBASE	SQL:1999	SQL-92
AUTOMAPPED	非保留	-	-
AVG	-	非保留	保留
BACKWARD	非保留	-	-
BARRIER	非保留	-	-
BEFORE	非保留	保留	-
BEGIN	非保留	保留	保留
BEGIN_NON_ANOYBLOCK	非保留	-	-
BETWEEN	非保留(不能是函数或类型)	非保留	保留
BIGINT	非保留(不能是函数或类型)	-	-
BINARY	保留(可以是函数或类型)	保留	-
BINARY_DOUBLE	非保留(不能是函数或类型)	-	-
BINARY_INTEGER	非保留(不能是函数或类型)	-	-
BIT	非保留(不能是函数或类型)	保留	保留
BITVAR	-	非保留	-
BIT_LENGTH	-	非保留	保留
BLOB	非保留	保留	-
BLOCK	非保留	-	-
BODY	非保留	-	-
BOOLEAN	非保留(不能是函数或类型)	保留	-

关键字	VASTBASE	SQL:1999	SQL-92
BOTH	保留	保留	保留
BUCKETS	保留	-	-
BREADTH	-	保留	-
BY	非保留	保留	保留
C	-	非保留	非保留
CACHE	非保留	-	-
CALL	非保留	保留	-
CALLED	非保留	非保留	-
CARDINALITY	-	非保留	-
CASCADE	非保留	保留	保留
CASCADED	非保留	保留	保留
CASE	保留	保留	保留
CAST	保留	保留	保留
CATALOG	非保留	保留	保留
CATALOG_NAME	-	非保留	非保留
CHAIN	非保留	非保留	-
CHAR	非保留(不能是函数或类型)	保留	保留
CHARACTER	非保留(不能是函数或类型)	保留	保留
CHARACTERISTICS	非保留	-	-
CHARACTER_LENGTH	-	非保留	保留
CHARACTER_SET_CATALOG	-	非保留	非保留
CHARACTER_SET_NAME	-	非保留	非保留
CHARACTER_SET_SCHEMA	-	非保留	非保留

关键字	VASTBASE	SQL:1999	SQL-92
CHAR_LENGTH	-	非保留	保留
CHECK	保留	保留	保留
CHECKED	-	非保留	-
CHECKPOINT	非保留	-	-
CLASS	非保留	保留	-
CLEAN	非保留	-	-
CLASS_ORIGIN	-	非保留	非保留
CLOB	非保留	保留	-
CLOSE	非保留	保留	保留
CLUSTER	非保留	-	-
COALESCE	非保留(不能是函数或类型)	非保留	保留
COBOL	-	非保留	非保留
COLLATE	保留	保留	保留
COLLATION	保留(可以是函数或类型)	保留	保留
COLLATION_CATALOG	-	非保留	非保留
COLLATION_NAME	-	非保留	非保留
COLLATION_SCHEMA	-	非保留	非保留
COLUMN	保留	保留	保留
COLUMN_NAME	-	非保留	非保留
COMMAND_FUNCTION	-	非保留	非保留
COMPLETE	非保留	-	-
COMMAND_FUNCTION_CODE	-	非保留	-
COMMENT	非保留	-	-

关键字	VASTBASE	SQL:1999	SQL-92
COMMENTS	非保留	-	-
COMMIT	非保留	保留	保留
COMMITTED	非保留	非保留	非保留
COMPRESS	非保留	-	-
COMPLETION	-	保留	-
CONCURRENTLY	保留(可以是函数或类型)	-	-
CONDITION	-	-	-
CONDITION_NUMBER	-	非保留	非保留
CONFIGURATION	非保留	-	-
CONNECT	-	保留	保留
CONNECTION	非保留	保留	保留
CONNECTION_NAME	-	非保留	非保留
CONSTRAINT	保留	保留	保留
CONSTRAINTS	非保留	保留	保留
CONSTRAINT_CATALOG	-	非保留	非保留
CONSTRAINT_NAME	-	非保留	非保留
CONSTRAINT_SCHEMA	-	非保留	非保留
CONSTRUCTOR	-	保留	-
CONTAINS	-	非保留	-
CONTENT	非保留	-	-
CONTINUE	非保留	保留	保留
CONVERSION	非保留	-	-
CONVERT	-	非保留	保留

关键字	VASTBASE	SQL:1999	SQL-92
COORDINATOR	非保留	-	-
COPY	非保留	-	-
CORRESPONDING	-	保留	保留
COST	非保留	-	-
COUNT	-	非保留	保留
CREATE	保留	保留	保留
CROSS	保留(可以是函数或类型)	保留	保留
CSV	非保留	-	-
CUBE	-	保留	-
CURRENT	非保留	保留	保留
CURRENT_CATALOG	保留	-	-
CURRENT_DATE	保留	保留	保留
CURRENT_PATH	-	保留	-
CURRENT_ROLE	保留	保留	-
CURRENT_SCHEMA	保留(可以是函数或类型)	-	-
CURRENT_TIME	保留	保留	保留
CURRENT_TIMESTAMP	保留	保留	保留
CURRENT_USER	保留	保留	保留
CURSOR	非保留	保留	保留
CURSOR_NAME	-	非保留	非保留
CYCLE	非保留	保留	-
DATA	非保留	保留	非保留
DATABASE	非保留	-	-

关键字	VASTBASE	SQL:1999	SQL-92
DATAFILE	非保留	-	-
DATE	非保留(不能是函数或类型)	保留	保留
DATE_FORMAT	非保留	-	-
DATETIME_INTERVAL_CODE	-	非保留	非保留
DATETIME_INTERVAL_PRECISION	-	非保留	非保留
DAY	非保留	保留	保留
DBCOMPATIBILITY	非保留	-	-
DBTIMEZONE	保留	-	-
DEALLOCATE	非保留	保留	保留
DEC	非保留(不能是函数或类型)	保留	保留
DECIMAL	非保留(不能是函数或类型)	保留	保留
DECLARE	非保留	保留	保留
DECODE	非保留(不能是函数或类型)	-	-
DEFAULT	保留	保留	保留
DEFAULTS	非保留	-	-
DEFERRABLE	保留	保留	保留
DEFERRED	非保留	保留	保留
DEFINED	-	非保留	-
DEFINER	非保留	非保留	-
DELETE	非保留	保留	保留
DELIMITER	非保留	-	-

关键字	VASTBASE	SQL:1999	SQL-92
DELIMITERS	非保留	-	-
DELTA	非保留	-	-
DEPTH	-	保留	-
DEREF	-	保留	-
DESC	保留	保留	保留
DESCRIBE	-	保留	保留
DESCRIPTOR	-	保留	保留
DESTROY	-	保留	-
DESTRUCTOR	-	保留	-
DETERMINISTIC	非保留	保留	-
DIAGNOSTICS	-	保留	保留
DICTIONARY	非保留	保留	-
DIRECT	非保留	-	-
DIRECTORY	非保留	-	-
DISABLE	非保留	-	-
DISCARD	非保留	-	-
DISCONNECT	-	保留	保留
DISPATCH	-	非保留	-
DISTINCT	保留	保留	保留
DISTRIBUTE	非保留	-	-
DISTRIBUTION	非保留	-	-
DO	保留	-	-
DOCUMENT	非保留	-	-
DOMAIN	非保留	保留	保留

关键字	VASTBASE	SQL:1999	SQL-92
DOUBLE	非保留	保留	保留
DROP	非保留	保留	保留
DUPLICATE	非保留	-	-
DYNAMIC	-	保留	-
DYNAMIC_FUNCTION	-	非保留	非保留
DYNAMIC_FUNCTION_CODE	-	非保留	-
EACH	非保留	保留	-
ELSE	保留	保留	保留
ENABLE	非保留	-	-
ENCODING	非保留	-	-
ENCRYPTED	非保留	-	-
END	保留	保留	保留
END-EXEC	-	保留	保留
ENFORCED	非保留	-	-
ENUM	非保留	-	-
EOL	非保留	-	-
EQUALS	-	保留	-
ESCAPE	非保留	保留	保留
ESCAPING	非保留	-	-
EVERY	非保留	保留	-
EXCEPT	保留	保留	保留
EXCEPTION	-	保留	保留
EXCHANGE	非保留	-	-
EXCLUDE	非保留	-	-

关键字	VASTBASE	SQL:1999	SQL-92
EXCLUDING	非保留	-	-
EXCLUSIVE	非保留	-	-
EXEC	-	保留	保留
EXECUTE	非保留	保留	保留
EXISTING	-	非保留	-
EXISTS	非保留(不能是函数或类型)	非保留	保留
EXPLAIN	非保留	-	-
EXTENSION	非保留	-	-
EXTERNAL	非保留	保留	保留
EXTRACT	非保留(不能是函数或类型)	非保留	保留
FALSE	保留	保留	保留
FAMILY	非保留	-	-
FAST	非保留	-	-
FETCH	保留	保留	保留
FENCED	保留	-	-
FILEHEADER	非保留	-	-
FILE_TYPE	非保留	-	-
FILL_MISSING_FIELDS	非保留	-	-
FILTER	非保留	-	-
FINAL	-	非保留	-
FIRST	非保留	保留	保留
FIXED	非保留	保留	保留
FLASHBACK	非保留	-	-

关键字	VASTBASE	SQL:1999	SQL-92
FLOAT	非保留(不能是函数或类型)	保留	保留
FOLLOWING	非保留	-	-
FOR	保留	保留	保留
FORCE	非保留	-	-
FOREIGN	保留	保留	保留
FORMATTER	非保留	-	-
FORTRAN	-	非保留	非保留
FORWARD	非保留	-	-
FOUND	-	保留	保留
FREE	-	保留	-
FREEZE	保留(可以是函数或类型)	-	-
FROM	保留	保留	保留
FULL	保留(可以是函数或类型)	保留	保留
FUNCTION	非保留	保留	-
FUNCTIONS	非保留	-	-
G	-	非保留	-
GENERAL	-	保留	-
GENERATED	-	非保留	-
GET	-	保留	保留
GLOBAL	非保留	保留	保留
GO	-	保留	保留
GOTO	-	保留	保留

关键字	VASTBASE	SQL:1999	SQL-92
GRANT	保留	保留	保留
GRANTED	非保留	非保留	-
GREATEST	非保留(不能是函数或类型)	-	-
GROUP	保留	保留	保留
GROUPING	-	保留	-
HANDLER	非保留	-	-
HAVING	保留	保留	保留
HEADER	非保留	-	-
HIERARCHY	-	非保留	-
HOLD	非保留	非保留	-
HOST	-	保留	-
HOUR	非保留	保留	保留
IDENTIFIED	非保留	-	-
IDENTITY	非保留	保留	保留
IF	非保留	-	-
IGNORE	-	保留	-
ILIKE	保留(可以是函数或类型)	-	-
IMMEDIATE	非保留	保留	保留
IMMUTABLE	非保留	-	-
IMPLEMENTATION	-	非保留	-
IMPLICIT	非保留	-	-
IN	保留	保留	保留
INCLUDING	非保留	-	-

关键字	VASTBASE	SQL:1999	SQL-92
INCREMENT	非保留	-	-
INDEX	非保留	-	-
INDEXES	非保留	-	-
INDICATOR	-	保留	保留
INFIX	-	非保留	-
INHERIT	非保留	-	-
INHERITS	非保留	-	-
INITIAL	非保留	-	-
INITIALIZE	-	保留	-
INITIALLY	保留	保留	保留
INITRANS	非保留	-	-
INLINE	非保留	-	-
INNER	保留(可以是函数或类型)	保留	保留
INOUT	非保留(不能是函数或类型)	保留	-
INPUT	非保留	保留	保留
INSENSITIVE	非保留	非保留	保留
INSERT	非保留	保留	保留
INSTANCE	-	非保留	-
INSTANTIABLE	-	非保留	-
INSTEAD	非保留	-	-
INT	非保留(不能是函数或类型)	保留	保留
INTEGER	非保留(不能是函数	保留	保留

关键字	VASTBASE	SQL:1999	SQL-92
	或类型)		
INTERSECT	保留	保留	保留
INTERVAL	非保留(不能是函数或类型)	保留	保留
INTO	保留	保留	保留
INVOKER	非保留	非保留	-
IP	非保留	-	-
IS	保留	保留	保留
ISNULL	非保留	-	-
ISOLATION	非保留	保留	保留
ITERATE	-	保留	-
JOIN	保留(可以是函数或类型)	保留	保留
K	-	非保留	-
KEY	非保留	保留	保留
KEY_MEMBER	-	非保留	-
KEY_TYPE	-	非保留	-
KILL	非保留	-	-
LABEL	非保留	-	-
LANGUAGE	非保留	保留	保留
LARGE	非保留	保留	-
LAST	非保留	保留	保留
LATERAL	-	保留	-
LC_COLLATE	非保留	-	-
LC_CTYPE	非保留	-	-

关键字	VASTBASE	SQL:1999	SQL-92
LEADING	保留	保留	保留
LEAKPROOF	非保留	-	-
LEAST	非保留(不能是函数或类型)	-	-
LEFT	保留(可以是函数或类型)	保留	保留
LENGTH	-	非保留	非保留
LESS	保留	保留	-
LEVEL	非保留	保留	保留
LIKE	保留(可以是函数或类型)	保留	保留
LIMIT	保留	保留	-
LIST	非保留	-	-
LISTEN	非保留	-	-
LOAD	非保留	-	-
LOCAL	非保留	保留	保留
LOCALTIME	保留	保留	-
LOCALTIMESTAMP	保留	保留	-
LOCATION	非保留	-	-
LOCATOR	-	保留	-
LOCK	非保留	-	-
LOG	非保留	-	-
LOGGING	非保留	-	-
LOGIN	非保留	-	-
LOGIN_ANY	非保留	-	-

关键字	VASTBASE	SQL:1999	SQL-92
LOGIN_FAILURE	非保留	-	-
LOGIN_SUCCESS	非保留	-	-
LOGOUT	非保留	-	-
LOOP	非保留	-	-
LOWER	-	非保留	保留
M	-	非保留	-
MAP	-	保留	-
MAPPING	非保留	-	-
MASKING	非保留	-	-
MATCH	非保留	保留	保留
MATCHED	非保留	-	-
MATERIALIZED	非保留	-	-
MAX	-	非保留	保留
MAXEXTENTS	非保留	-	-
MAXSIZE	非保留	-	-
MAXTRANS	非保留	-	-
MAXVALUE	保留	-	-
MERGE	非保留	-	-
MESSAGE_LENGTH	-	非保留	非保留
MESSAGE_OCTET_LENGTH	-	非保留	非保留
MESSAGE_TEXT	-	非保留	非保留
METHOD	-	非保留	-
MIN	-	非保留	保留
MINEXTENTS	非保留	-	-

关键字	VASTBASE	SQL:1999	SQL-92
MINUS	保留	-	-
MINUTE	非保留	保留	保留
MINVALUE	非保留	-	-
MOD	-	非保留	-
MODE	非保留	-	-
MODIFIES	-	保留	-
MODIFY	保留	保留	-
MODULE	-	保留	保留
MODULUS	非保留	-	-
MONTH	非保留	保留	保留
MORE	-	非保留	非保留
MOVE	非保留	-	-
MOVEMENT	非保留	-	-
MUMPS	-	非保留	非保留
NAME	非保留	非保留	非保留
NAMES	非保留	保留	保留
NATIONAL	非保留(不能是函数或类型)	保留	保留
NATURAL	保留(可以是函数或类型)	保留	保留
NCHAR	非保留(不能是函数或类型)	保留	保留
NCLOB	-	保留	-
NEW	-	保留	-
NEXT	非保留	保留	保留

关键字	VASTBASE	SQL:1999	SQL-92
NLSSORT	保留	-	-
NO	非保留	保留	保留
NOCACHE	非保留	-	-
NOCOMPRESS	非保留	-	-
NOCYCLE	保留	-	-
NODE	非保留	-	-
NOLOGGING	非保留	-	-
NOLOGIN	非保留	-	-
NOMAXVALUE	非保留	-	-
NOMINVALUE	非保留	-	-
NONE	非保留(不能是函数或类型)	保留	-
NOORDER	非保留	-	-
NORMALLY	非保留	-	-
NOT	保留	保留	保留
NOTHING	非保留	-	-
NOTIFY	非保留	-	-
NOTNULL	保留(可以是函数或类型)	-	-
NOWAIT	非保留	-	-
NULL	保留	保留	保留
NULLABLE	-	非保留	非保留
NULLIF	非保留(不能是函数或类型)	非保留	保留
NULLS	非保留	-	-

关键字	VASTBASE	SQL:1999	SQL-92
NUMBER	非保留(不能是函数或类型)	非保留	非保留
NUMERIC	非保留(不能是函数或类型)	保留	保留
NUMSTR	非保留	-	-
NVARCHAR2	非保留(不能是函数或类型)	-	-
NVL	非保留(不能是函数或类型)	-	-
OBJECT	非保留	保留	-
OCTET_LENGTH	-	非保留	保留
OF	非保留	保留	保留
OFF	非保留	保留	-
OFFSET	保留	-	-
OIDS	非保留	-	-
OLD	-	保留	-
ON	保留	保留	保留
ONLY	保留	保留	保留
OPEN	-	保留	保留
OPERATION	-	保留	-
OPERATOR	非保留	-	-
OPTIMIZATION	非保留	-	-
OPTION	非保留	保留	保留
OPTIONS	非保留	非保留	-
OR	保留	保留	保留

关键字	VASTBASE	SQL:1999	SQL-92
ORDER	保留	保留	保留
ORDINALITY	-	保留	-
OUT	非保留(不能是函数或类型)	保留	-
OUTER	保留(可以是函数或类型)	保留	保留
OUTPUT	-	保留	保留
OVER	非保留	-	-
OVERLAPS	保留(可以是函数或类型)	非保留	保留
OVERLAY	非保留(不能是函数或类型)	非保留	-
OVERRIDING	-	非保留	-
OWNED	非保留	-	-
OWNER	非保留	-	-
PAD	-	保留	保留
PARAMETER	-	保留	-
PARAMETERS	-	保留	-
PARAMETER_MODE	-	非保留	-
PARAMETER_NAME	-	非保留	-
PARAMETER_ORDINAL_POSITION	-	非保留	-
PARAMETER_SPECIFIC_CATALOG	-	非保留	-
PARAMETER_SPECIFIC_NAME	-	非保留	-
PARAMETER_SPECIFIC_SCHEMA	-	非保留	-
PARSER	非保留	-	-

关键字	VASTBASE	SQL:1999	SQL-92
PARTIAL	非保留	保留	保留
PARTITION	非保留	-	-
PARTITIONS	非保留	-	-
PASCAL	-	非保留	非保留
PASSING	非保留	-	-
PASSWORD	非保留	-	-
PATH	-	保留	-
PCTFREE	非保留	-	-
PER	非保留	-	-
PERCENT	非保留	-	-
PERFORMANCE	保留	-	-
PERM	非保留	-	-
PIPELINED	非保留	-	-
PLACING	保留	-	-
PLANS	非保留	-	-
PLI	-	非保留	非保留
POOL	非保留	-	-
POSITION	非保留(不能是函数或类型)	非保留	保留
POSTFIX	-	保留	-
PRECEDING	非保留	-	-
PRECISION	非保留(不能是函数或类型)	保留	保留
PREFERRED	非保留	-	-
PREFIX	非保留	保留	-

关键字	VASTBASE	SQL:1999	SQL-92
PREORDER	-	保留	-
PREPARE	非保留	保留	保留
PREPARED	非保留	-	-
PRESERVE	非保留	保留	保留
PRIMARY	保留	保留	保留
PRIOR	保留	保留	保留
PRIVATE	非保留	-	-
PRIVILEGE	非保留	-	-
PRIVILEGES	非保留	保留	保留
PROCEDURAL	非保留	-	-
PROCEDURE	保留	保留	保留
PROFILE	非保留	-	-
PUBLIC	-	保留	保留
QUERY	非保留	-	-
QUOTE	非保留	-	-
RANGE	非保留	-	-
RAW	非保留	-	-
READ	非保留	保留	保留
READS	-	保留	-
REAL	非保留(不能是函数或类型)	保留	保留
REASSIGN	非保留	-	-
REBUILD	非保留	-	-
RECHECK	非保留	-	-

关键字	VASTBASE	SQL:1999	SQL-92
RECURSIVE	非保留	保留	-
REF	非保留	保留	-
REFERENCES	保留	保留	保留
REFERENCING	-	保留	-
REFRESH	非保留	-	-
REINDEX	非保留	-	-
REJECT	保留	-	-
RELATIVE	非保留	保留	保留
RELEASE	非保留	-	-
RELOPTIONS	非保留	-	-
REMAINDER	非保留	-	-
REMOTE	非保留	-	-
REMOVE	非保留	-	-
RENAME	非保留	-	-
REPEATABLE	非保留	非保留	非保留
REPLACE	非保留	-	-
REPLICA	非保留	-	-
RESET	非保留	-	-
RESIZE	非保留	-	-
RESOURCE	非保留	-	-
RESTART	非保留	-	-
RESTRICT	非保留	保留	保留
RESULT	-	保留	-
RETURN	非保留	保留	-

关键字	VASTBASE	SQL:1999	SQL-92
RETURNED_LENGTH	-	非保留	非保留
RETURNED_OCTET_LENGTH	-	非保留	非保留
RETURNED_SQLSTATE	-	非保留	非保留
RETURNING	保留	-	-
RETURNS	非保留	保留	-
REUSE	非保留	-	-
REVOKE	非保留	保留	保留
RIGHT	保留(可以是函数或类型)	保留	保留
ROLE	非保留	保留	-
ROLES	非保留	-	-
ROLLBACK	非保留	保留	保留
ROLLUP	-	保留	-
ROUTINE	-	保留	-
ROUTINE_CATALOG	-	非保留	-
ROUTINE_NAME	-	非保留	-
ROUTINE_SCHEMA	-	非保留	-
ROW	非保留(不能是函数或类型)	保留	-
ROWS	非保留	-	-
ROWID	非保留	-	-
ROWNUM	保留	-	-
ROWS	非保留	保留	保留
ROW_COUNT	-	非保留	非保留
RULE	非保留	-	-

关键字	VASTBASE	SQL:1999	SQL-92
SAMPLE	作为函数名或类型名的关键字	-	-
SAVEPOINT	非保留	保留	-
SCALE	-	非保留	非保留
SCHEMA	非保留	保留	保留
SCHEMA_NAME		非保留	非保留
SCOPE	-	保留	-
SCROLL	非保留	保留	保留
SEARCH	非保留	保留	-
SECOND	非保留	保留	保留
SECTION	-	保留	保留
SECURITY	非保留	非保留	-
SEED	非保留	-	
SELECT	保留	保留	保留
SELF	-	非保留	-
SENSITIVE	-	非保留	-
SEQUENCE	非保留	保留	-
SEQUENCES	非保留	-	-
SERIALIZABLE	非保留	非保留	非保留
SERVER	非保留	-	-
SERVER_NAME	-	非保留	非保留
SESSION	非保留	保留	保留
SESSIONTIMEZONE	保留	-	-
SESSION_USER	保留	保留	保留

关键字	VASTBASE	SQL:1999	SQL-92
SET	非保留	保留	保留
SETOF	非保留(不能是函数或类型)	-	-
SETS	-	保留	-
SHARE	非保留	-	-
SHIPPABLE	非保留	-	-
SHOW	非保留	-	-
SHUTDOWN	非保留	-	-
SIMILAR	保留(可以是函数或类型)	非保留	-
SIMPLE	非保留	非保留	-
SIZE	非保留	保留	保留
SMALLDATETIME	非保留(不能是函数或类型)	-	-
SMALLINT	非保留(不能是函数或类型)	保留	保留
SNAPSHOT	非保留	-	-
SOME	保留	保留	保留
SOURCE	-	非保留	-
SPACE	-	保留	保留
SPECIFIC	-	保留	-
SPECIFICTYPE	-	保留	-
SPECIFIC_NAME	-	非保留	-
SPILL	非保留	-	-
SPLIT	保留	-	-

关键字	VASTBASE	SQL:1999	SQL-92
SQL	-	保留	保留
SQLCODE	-	-	保留
SQLERROR	-	-	保留
SQLEXCEPTION	-	保留	-
SQLSTATE	-	保留	保留
SQLWARNING	-	保留	-
STABLE	非保留	-	-
STANDALONE	非保留	-	-
START	非保留	保留	-
STATE	-	保留	-
STATEMENT	非保留	保留	-
STATIC	-	保留	-
STATISTICS	非保留	-	-
STDIN	非保留	-	-
STDOUT	非保留	-	-
STORAGE	非保留	-	-
STORE	非保留	-	-
STORED	非保留	-	-
STRICT	非保留	-	-
STRIP	非保留	-	-
STRUCTURE	-	保留	-
STYLE	-	非保留	-
SUBCLASS_ORIGIN	-	非保留	非保留
SUBLIST	-	非保留	-

关键字	VASTBASE	SQL:1999	SQL-92
SUBSTRING	非保留(不能是函数或类型)	非保留	保留
SUM	-	非保留	保留
SUPERUSER	非保留	-	-
SYMMETRIC	保留	非保留	-
SYNONYM	非保留	-	-
SYS_REFCURSOR	非保留	-	-
SYSDATE	保留	-	-
SYSID	非保留	-	-
SYSTEM	非保留	非保留	-
SYSTEM_USER	-	保留	保留
SYSTIMESTAMP	非保留	-	-
TABLE	保留	保留	保留
TABLES	非保留	-	-
TABLESAMPLE	函数名或类型名的关键字	-	-
TABLE_NAME	-	非保留	非保留
TEMP	非保留	-	-
TEMPLATE	非保留	-	-
TEMPORARY	非保留	保留	保留
TERMINATE	-	保留	-
TEXT	非保留	-	-
THAN	非保留	保留	-
THEN	保留	保留	保留
TIME	非保留(不能是函数	保留	保留

关键字	VASTBASE	SQL:1999	SQL-92
	或类型)		
TIMESTAMP	非保留(不能是函数或类型)	保留	保留
TIMESTAMPDIFF	非保留(不能是函数或类型)	-	-
TIMEZONE_HOUR	-	保留	保留
TIMEZONE_MINUTE	-	保留	保留
TINYINT	非保留(不能是函数或类型)	-	-
TO	保留	保留	保留
TRAILING	保留	保留	保留
TRANSACTION	非保留	保留	保留
TRANSACTIONS_COMMITTED	-	非保留	-
TRANSACTIONS_ROLLED_BACK	-	非保留	-
TRANSACTION_ACTIVE	-	非保留	-
TRANSFORM	-	非保留	-
TRANSFORMS	-	非保留	-
TRANSLATE	-	非保留	保留
TRANSLATION	-	保留	保留
TREAT	非保留(不能是函数或类型)	保留	-
TRIGGER	非保留	保留	-
TRIGGER_CATALOG	-	非保留	-
TRIGGER_NAME	-	非保留	-
TRIGGER_SCHEMA	-	非保留	-

关键字	VASTBASE	SQL:1999	SQL-92
TRIM	非保留(不能是函数或类型)	非保留	保留
TRUE	保留	保留	保留
TRUNCATE	非保留	-	-
TRUSTED	非保留	-	-
TSFIELD	非保留	-	-
TSTAG	非保留	-	-
TSTIME	非保留	-	-
TYPE	非保留	非保留	非保留
TYPES	非保留	-	-
UESCAPE	-	-	-
UNBOUNDED	非保留	-	-
UNCOMMITTED	非保留	非保留	非保留
UNDER	-	保留	-
UNENCRYPTED	非保留	-	-
UNION	保留	保留	保留
UNIQUE	保留	保留	保留
UNKNOWN	非保留	保留	保留
UNLIMITED	非保留	-	-
UNLISTEN	非保留	-	-
UNLOCK	非保留	-	-
UNLOGGED	非保留	-	-
UNNAMED	-	非保留	非保留
UNNEST	-	保留	-

关键字	VASTBASE	SQL:1999	SQL-92
UNTIL	非保留	-	-
UNUSABLE	非保留	-	-
UPDATE	非保留	保留	保留
UPPER	-	非保留	保留
USAGE	-	保留	保留
USER	保留	保留	保留
USER_DEFINED_TYPE_CATALOG	-	非保留	-
USER_DEFINED_TYPE_NAME	-	非保留	-
USER_DEFINED_TYPE_SCHEMA	-	非保留	-
USING	保留	保留	保留
VACUUM	非保留	-	-
VALID	非保留	-	-
VALIDATE	非保留	-	-
VALIDATION	非保留	-	-
VALIDATOR	非保留	-	-
VALUE	非保留	保留	保留
VALUES	非保留(不能是函数或类型)	保留	保留
VARCHAR	非保留(不能是函数或类型)	保留	保留
VARCHAR2	非保留(不能是函数或类型)	-	-
VARIABLE	-	保留	-
VARIADIC	保留	-	-
VARYING	非保留	保留	保留

关键字	VASTBASE	SQL:1999	SQL-92
VERBOSE	保留(可以是函数或类型)	-	-
VERSION	非保留	-	-
VIEW	非保留	保留	保留
VOLATILE	非保留	-	-
WARNING	非保留	-	-
WHEN	保留	保留	保留
WHENEVER	-	保留	保留
WHERE	保留	保留	保留
WHITESPACE	非保留	-	-
WINDOW	保留	-	-
WITH	保留	保留	保留
WITHIN	非保留	-	-
WITHOUT	非保留	保留	-
WORK	非保留	保留	保留
WORKLOAD	非保留	-	-
WRAPPER	非保留	-	-
WRITE	非保留	保留	保留
XML	非保留	-	-
XMLATTRIBUTES	非保留(不能是函数或类型)	-	-
XMLCONCAT	非保留(不能是函数或类型)	-	-
XMLELEMENT	非保留(不能是函数或类型)	-	-

关键字	VASTBASE	SQL:1999	SQL-92
XMLEXISTS	非保留(不能是函数或类型)	-	-
XMLFOREST	非保留(不能是函数或类型)	-	-
XMLPARSE	非保留(不能是函数或类型)	-	-
XMLPI	非保留(不能是函数或类型)	-	-
XMLROOT	非保留(不能是函数或类型)	-	-
XMLSERIALIZE	非保留(不能是函数或类型)	-	-
YEAR	非保留	保留	保留
YES	非保留	-	-
ZONE	非保留	保留	保留

11.3. 数据类型

Vastbase 支持某些数据类型间的隐式转换，具体转化关系请参见 14.2.17PG_CAST。

11.3.1. 数值类型

表 11-2 列出了所有的可用类型。数字操作符和相关的内置函数请参见 11.5.7 数字操作函数和操作符。

表 11-2. 整数类型

名称	描述	存储空间	范围
TINYINT	微整数，别名为 INT1。	1 字节	0 ~ 255
SMALLINT	小范围整数，别名为 INT2。	2 字节	-32,768 ~ +32,767

名称	描述	存储空间	范围
INTEGER	常用的整数，别名为 INT4。	4 字节	-2,147,483,648 ~ +2,147,483,647
BINARY_INTEGER	常用的整数 INTEGER 的别名。	4 字节	-2,147,483,648 ~ +2,147,483,647
BIGINT	大范围的整数，别名为 INT8。	8 字节	-9,223,372,036,854,775,808 ~ +9,223,372,036,854,775,807

示例：

```
--创建具有 TINYINT 类型数据的表。
vastbase=# CREATE TABLE int_type_t1
         (
          IT_COL1 TINYINT
         );

--向 TINYINT 表中插入数据。
vastbase=# INSERT INTO int_type_t1 VALUES(10);

--查看数据。
vastbase=# SELECT * FROM int_type_t1;
 it_col1
---------
 10
(1 row)

--删除表。
vastbase=# DROP TABLE int_type_t1;

--创建具有 TINYINT,INTEGER,BIGINT 类型数据的表。
vastbase=# CREATE TABLE int_type_t2
(
   a TINYINT,
   b TINYINT,
   c INTEGER,
   d BIGINT
);

--插入数据。
vastbase=# INSERT INTO int_type_t2 VALUES(100, 10, 1000, 10000);

--查看数据。
vastbase=# SELECT * FROM int_type_t2;
  a  | b  |  c   |   d
-----+----+------+-------
 100 | 10 | 1000 | 10000
(1 row)
```

```
--删除表。
vastbase=# DROP TABLE int_type_t2;
```

说明

- TINYINT、SMALLINT、INTEGER 和 BIGINT 类型存储各种范围的数字，也就是整数。试图存储超出范围以外的数值将会导致错误。
- 常用的类型是 INTEGER，因为它提供了在范围、存储空间、性能之间的最佳平衡。一般只有取值范围确定不超过 SMALLINT 的情况下，才会使用 SMALLINT 类型。而只有在 INTEGER 的范围不够的时候才使用 BIGINT，因为前者相对快得多。

表 11-3. 任意精度型

名称	描述	存储空间	范围
NUMERIC[(p[,s])], DECIMAL[(p[,s])]	精度 p 取值范围为[1,1000]，标度 s 取值范围为[0,p]。 说明 p 为总位数，s 为小数位数。	用户声明精度。每四位（十进制位）占用两个字节，然后在整个数据上加上八个字节的额外开销。	未指定精度的情况下，小数点前最大 131,072 位，小数点后最大 16,383 位。
NUMBER[(p[,s])]	NUMERIC 类型的别名。	用户声明精度。每四位（十进制位）占用两个字节，然后在整个数据上加上八个字节的额外开销。	未指定精度的情况下，小数点前最大 131,072 位，小数点后最大 16,383 位。

示例：

```
--创建表。
vastbase=# CREATE TABLE decimal_type_t1
(
   DT_COL1 DECIMAL(10,4)
);

--插入数据。
vastbase=# INSERT INTO decimal_type_t1 VALUES(123456.122331);

--查询表中的数据。
vastbase=# SELECT * FROM decimal_type_t1;
  dt_col1
-------------
 123456.1223
(1 row)

--删除表。
vastbase=# DROP TABLE decimal_type_t1;
--创建表。
vastbase=# CREATE TABLE numeric_type_t1
```

```
(
   NT_COL1 NUMERIC(10,4)
);

--插入数据。
vastbase=# INSERT INTO numeric_type_t1 VALUES(123456.12354);

--查询表中的数据。
vastbase=# SELECT * FROM numeric_type_t1;
   nt_col1
-------------
 123456.1235
(1 row)

--删除表。
vastbase=# DROP TABLE numeric_type_t1;
```

说明

- 与整数类型相比，任意精度类型需要更大的存储空间，其存储效率、运算效率以及压缩比效果都要差一些。在进行数值类型定义时，优先选择整数类型。当且仅当数值超出整数可表示最大范围时，再选用任意精度类型。
- 使用 Numeric/Decimal 进行列定义时，建议指定该列的精度 p 以及标度 s。

表 11-4. 序列整型

名称	描述	存储空间	范围
SMALLSERIAL	二字节序列整型。	2 字节	1 ~ 32,767
SERIAL	四字节序列整型。	4 字节	1 ~ 2,147,483,647
BIGSERIAL	八字节序列整型。	8 字节	1 ~ 9,223,372,036,854,775,807

示例：

```
--创建表。
vastbase=# CREATE TABLE smallserial_type_tab(a SMALLSERIAL);

--插入数据。
vastbase=# INSERT INTO smallserial_type_tab VALUES(default);

--再次插入数据。
vastbase=# INSERT INTO smallserial_type_tab VALUES(default);

--查看数据。
vastbase=# SELECT * FROM smallserial_type_tab;
 a
---
 1
 2
(2 rows)

--创建表。
```

```
vastbase=# CREATE TABLE serial_type_tab(b SERIAL);

--插入数据。
vastbase=# INSERT INTO serial_type_tab VALUES(default);

--再次插入数据。
vastbase=# INSERT INTO serial_type_tab VALUES(default);

--查看数据。
vastbase=# SELECT * FROM serial_type_tab;
 b
---
 1
 2
(2 rows)

--创建表。
vastbase=# CREATE TABLE bigserial_type_tab(c BIGSERIAL);

--插入数据。
vastbase=# INSERT INTO bigserial_type_tab VALUES(default);

--插入数据。
vastbase=# INSERT INTO bigserial_type_tab VALUES(default);

--查看数据。
vastbase=# SELECT * FROM bigserial_type_tab;
 c
---
 1
 2
(2 rows)

--删除表。
vastbase=# DROP TABLE smallserial_type_tab;

vastbase=# DROP TABLE serial_type_tab;

vastbase=# DROP TABLE bigserial_type_tab;
```

说明

SMALLSERIAL，SERIAL 和 BIGSERIAL 类型不是真正的类型，只是为在表中设置唯一标识做的概念上的便利。因此，创建一个整数字段，并且把它的缺省数值安排为从一个序列发生器读取。应用了一个 NOT NULL 约束以确保 NULL 不会被插入。在大多数情况下用户可能还希望附加一个 UNIQUE 或 PRIMARY KEY 约束避免意外地插入重复的数值，但这个不是自动的。最后，将序列发生器从属于那个字段，这样当该字段或表被删除的时候也一并删除它。目前只支持在创建表时候指定 SERIAL 列，不可以在已有的表中，增加 SERIAL 列。另外临时表也不支持创建 SERIAL 列。因为 SERIAL 不是真正的类型，也不可以将表中存在的列类型转化为 SERIAL。

表 11-5. 浮点类型

名称	描述	存储空间	范围

名称	描述	存储空间	范围
REAL, FLOAT4	单精度浮点数,不精准。	4 字节	6 位十进制数字精度。
DOUBLE PRECISION, FLOAT8, FLOAT	双精度浮点数,不精准。	8 字节	1E-307~1E+308, 15 位十进制数字精度。
FLOAT[(p)]	浮点数，不精准。精度 p 取值范围为[1,53]。 说明 p 为精度，表示总位数。	4 字节或 8 字节	根据精度 p 不同选择 REAL 或 DOUBLE PRECISION 作为内部表示。如不指定精度，内部用 DOUBLE PRECISION 表示。
BINARY_DOUBLE	是 DOUBLE PRECISION 的别名。	8 字节	1E-307~1E+308, 15 位十进制数字精度。
DEC[(p[,s])]	精度 p 取值范围为[1,1000]，标度 s 取值范围为[0,p]。 说明 p 为总位数，s 为小数位位数。	用户声明精度。每四位（十进制位）占用两个字节，然后在整个数据上加上八个字节的额外开销。	未指定精度的情况下，小数点前最大 131,072 位，小数点后最大 16,383 位。
INTEGER[(p[,s])]	精度 p 取值范围为[1,1000]，标度 s 取值范围为[0,p]。	用户声明精度。每四位（十进制位）占用两个字节，然后在整个数据上加上八个字节的额外开销。	未指定精度的情况下，小数点前最大 131,072 位，小数点后最大 16,383 位。

示例：

```
--创建表。
vastbase=# CREATE TABLE float_type_t2
(
    FT_COL1 INTEGER,
    FT_COL2 FLOAT4,
    FT_COL3 FLOAT8,
    FT_COL4 FLOAT,
    FT_COL5 FLOAT(3),
    FT_COL6 BINARY_DOUBLE,
    FT_COL7 DECIMAL(10,4),
    FT_COL8 INTEGER(6,3)
);
```

```
--插入数据。
vastbase=# INSERT INTO float_type_t2 VALUES(10,10.365456,123456.1234,123456.1234,10.3214, 321.321,
123.123654, 123.123654);

--查看数据。
vastbase=# SELECT * FROM float_type_t2 ;
 ft_col1 | ft_col2 |   ft_col3   |   ft_col4    | ft_col5 | ft_col6  | ft_col7  | ft_col8
---------+---------+-------------+--------------+---------+----------+----------+---------
      10 | 10.3655 | 123456.1234 | 123456.1234 | 10.3214 | 321.321  | 123.1237 | 123.124
(1 row)

--删除表。
vastbase=# DROP TABLE float_type_t2;
```

11.3.2. 货币类型

货币类型存储带有固定小数精度的货币金额。

表 11-6 中显示的范围假设有两位小数。可以以任意格式输入，包括整型、浮点型或者典型的货币格式（如“$1,000.00”）。根据区域字符集，输出一般是最后一种形式。

表 11-6. 货币类型

名称	存储容量	描述	范围
money	8 字节	货币金额	-92233720368547758.08 到 +92233720368547758.07

numeric，int 和 bigint 类型的值可以转化为 money 类型。如果从 real 和 double precision 类型转换到 money 类型，可以先转化为 numeric 类型，再转化为 money 类型，例如：

```
vastbase=# SELECT '12.34'::float8::numeric::money;
```

这种用法是不推荐使用的。浮点数不应该用来处理货币类型，因为小数点的位数可能会导致错误。

money 类型的值可以转换为 numeric 类型而不丢失精度。转换为其他类型可能丢失精度，并且必须通过以下两步来完成：

```
vastbase=# SELECT '52093.89'::money::numeric::float8;
```

当一个 money 类型的值除以另一个 money 类型的值时，结果是 double precision（也就是，一个纯数字，而不是 money 类型）；在运算过程中货币单位相互抵消。

11.3.3. 布尔类型

表 11-7. 布尔类型

名称	描述	存储空间	取值
BOOLEAN	布尔类型	1 字节。	• true：真 • false：假

名称	描述	存储空间	取值
			• null：未知（unknown）

“真”值的有效文本值是：
TRUE、't'、'true'、'y'、'yes'、'1'。
“假”值的有效文本值是：
FALSE、'f'、'false'、'n'、'no'、'0'。
使用 TRUE 和 FALSE 是比较规范的用法（也是 SQL 兼容的用法）。

示例

显示用字母 t 和 f 输出 Boolean 值。

```
--创建表。
vastbase=# CREATE TABLE bool_type_t1
(
   BT_COL1 BOOLEAN,
   BT_COL2 TEXT
);

--插入数据。
vastbase=# INSERT INTO bool_type_t1 VALUES (TRUE, 'sic est');

vastbase=# INSERT INTO bool_type_t1 VALUES (FALSE, 'non est');

--查看数据。
vastbase=# SELECT * FROM bool_type_t1;
 bt_col1 | bt_col2
---------+---------
 t       | sic est
 f       | non est
(2 rows)

vastbase=# SELECT * FROM bool_type_t1 WHERE bt_col1 = 't';
 bt_col1 | bt_col2
---------+---------
 t       | sic est
(1 row)

--删除表。
vastbase=# DROP TABLE bool_type_t1;
```

11.3.4. 字符类型

Vastbase 支持的字符类型请参见表 11-8。字符串操作符和相关的内置函数请参见 11.5.3 字符处理函数和操作符。

表 11-8. 字符类型

名称	描述	存储空间
CHAR(n) CHARACTER(n) NCHAR(n)	定长字符串，不足补空格。n 是指字节长度，如不带精度 n，默认精度为 1。	最大为 10MB。
VARCHAR(n) CHARACTER VARYING(n)	变长字符串。n 是指字节长度。	最大为 10MB。
VARCHAR2(n)	变长字符串。是 VARCHAR(n)类型的别名。n 是指字节长度。	最大为 10MB。
NVARCHAR2(n)	变长字符串。n 是指字节长度。	最大为 10MB。
CLOB	文本大对象。是 TEXT 类型的别名。	最大为 1GB-8203 字节（即 1073733621 字节）。
TEXT	变长字符串。	最大为 1GB-8203 字节（即 1073733621 字节）。
VARCHAR	变长字符串。	最大为 1GB-8203 字节（即 1073733621 字节）。

说明

除了每列的大小限制以外，每个元组的总大小也不可超过 1GB-8203 字节（即 1073733621 字节）。

在 Vastbase 里另外还有两种定长字符类型。在表 11-9 里显示。name 类型只用在内部系统表中，作为存储标识符，不建议普通用户使用。该类型长度当前定为 64 字节（63 可用字符加结束符）。类型"char"只用了一个字节的存储空间。他在系统内部主要用于系统表，主要作为简单化的枚举类型使用。

表 11-9. 特殊字符类型

名称	描述	存储空间
name	用于对象名的内部类型。	64 字节。
"char"	单字节内部类型。	1 字节。

示例

```
--创建表。
vastbase=# CREATE TABLE char_type_t1
(
   CT_COL1 CHARACTER(4)
);

--插入数据。
vastbase=# INSERT INTO char_type_t1 VALUES ('ok');

--查询表中的数据。
vastbase=# SELECT ct_col1, char_length(ct_col1) FROM char_type_t1;
 ct_col1 | char_length
---------+-------------
 ok      |           4
(1 row)

--删除表。
vastbase=# DROP TABLE char_type_t1;
--创建表。
vastbase=# CREATE TABLE char_type_t2
(
   CT_COL1 VARCHAR(5)
);

--插入数据。
vastbase=# INSERT INTO char_type_t2 VALUES ('ok');

vastbase=# INSERT INTO char_type_t2 VALUES ('good');

--插入的数据长度超过类型规定的长度报错。
vastbase=# INSERT INTO char_type_t2 VALUES ('too long');
ERROR:  value too long for type character varying(4)
CONTEXT:  referenced column: ct_col1

--明确类型的长度，超过数据类型长度后会自动截断。
vastbase=# INSERT INTO char_type_t2 VALUES ('too long'::varchar(5));

--查询数据。
vastbase=# SELECT ct_col1, char_length(ct_col1) FROM char_type_t2;
 ct_col1 | char_length
---------+-------------
 ok      |           2
 good    |           5
 too l   |           5
(3 rows)

--删除数据。
vastbase=# DROP TABLE char_type_t2;
```

11.3.5. 二进制类型

Vastbase 支持的二进制类型请参见表 11-10。

表 11-10. 二进制类型

名称	描述	存储空间
BLOB	二进制大对象 说明 列存不支持 BLOB 类型	最大为 1GB-8203 字节（即 1073733621 字节）。
RAW	变长的十六进制类型 说明 列存不支持 RAW 类型	4 字节加上实际的十六进制字符串。最大为 1GB-8203 字节（即 1073733621 字节）。
BYTEA	变长的二进制字符串	4 字节加上实际的二进制字符串。最大为 1GB-8203 字节（即 1073733621 字节）。

说明

除了每列的大小限制以外，每个元组的总大小也不可超过 1GB-8203 字节（即 1073733621 字节）。

示例:

```
--创建表。
vastbase=# CREATE TABLE blob_type_t1
(
   BT_COL1 INTEGER,
   BT_COL2 BLOB,
   BT_COL3 RAW,
   BT_COL4 BYTEA
) ;

--插入数据。
vastbase=# INSERT INTO blob_type_t1 VALUES(10,empty_blob(),
HEXTORAW('DEADBEEF'),E'\\xDEADBEEF');

--查询表中的数据。
vastbase=# SELECT * FROM blob_type_t1;
 bt_col1 | bt_col2 | bt_col3  |  bt_col4
---------+---------+----------+------------
      10 |         | DEADBEEF | \xdeadbeef
(1 row)

--删除表。
vastbase=# DROP TABLE blob_type_t1;
```

11.3.6. 日期/时间类型

Vastbase 支持的日期/时间类型请参见表 11-11。该类型的操作符和内置函数请参见 11.5.8 时间和日期处理函数和操作符。

说明

如果其他的数据库时间格式和 Vastbase 的时间格式不一致，可通过修改配置参数 DateStyle 的值来保持一致。

表 11-11. 日期/时间类型

名称	描述	存储空间
DATE	日期和时间。	4 字节（实际存储空间大小为 8 字节）
TIME [(p)] [WITHOUT TIME ZONE]	只用于一日内时间。 p 表示小数点后的精度，取值范围为 0~6。	8 字节
TIME [(p)] [WITH TIME ZONE]	只用于一日内时间，带时区。 p 表示小数点后的精度，取值范围为 0~6。	12 字节
TIMESTAMP[(p)] [WITHOUT TIME ZONE]	日期和时间。 p 表示小数点后的精度，取值范围为 0~6。	8 字节
TIMESTAMP[(p)][WITH TIME ZONE]	日期和时间，带时区。TIMESTAMP 的别名为 TIMESTAMPTZ。 p 表示小数点后的精度，取值范围为 0~6。	8 字节
SMALLDATETIME	日期和时间，不带时区。 精确到分钟，秒位大于等于 30 秒进一位。	8 字节
INTERVAL DAY (l) TO SECOND (p)	时间间隔，X 天 X 小时 X 分 X 秒。 • l：天数的精度，取值范围为 0~6。兼容性考虑，目前未实现具体功能。 • p：秒数的精度，取值范围为 0~6。小数末尾的零不显示。	16 字节
INTERVAL [FIELDS] [(p)]	时间间隔。 • fields：可以是 YEAR，MONTH，DAY，HOUR，MINUTE，SECOND，DAY TO HOUR，DAY TO MINUTE，DAY TO SECOND，HOUR	12 字节

名称	描述	存储空间
	TO MINUTE，HOUR TO SECOND，MINUTE TO SECOND。 • p：秒数的精度，取值范围为 0~6，且 fields 为 SECOND，DAY TO SECOND，HOUR TO SECOND 或 MINUTE TO SECOND 时，参数 p 才有效。小数末尾的零不显示。	
reltime	相对时间间隔。格式为： X years X mons X days XX:XX:XX。 • 采用儒略历计时，规定一年为 365.25 天，一个月为 30 天，计算输入值对应的相对时间间隔，输出采用 POSTGRES 格式。	4 字节

示例：

```
--创建表。
vastbase=# CREATE TABLE date_type_tab(coll date);

--插入数据。
vastbase=# INSERT INTO date_type_tab VALUES (date '12-10-2010');

--查看数据。
vastbase=# SELECT * FROM date_type_tab;
      coll
---------------------
 2010-12-10 00:00:00
(1 row)

--删除表。
vastbase=# DROP TABLE date_type_tab;

--创建表。
vastbase=# CREATE TABLE time_type_tab (da time without time zone ,dai time with time zone,dfgh timestamp
without time zone,dfga timestamp with time zone, vbg smalldatetime);

--插入数据。
vastbase=# INSERT INTO time_type_tab VALUES ('21:21:21','21:21:21 pst','2010-12-12','2013-12-11
pst','2003-04-12 04:05:06');

--查看数据。
vastbase=# SELECT * FROM time_type_tab;
    da    |     dai     |        dfgh         |         dfga          |         vbg
----------+-------------+---------------------+-----------------------+---------------------
 21:21:21 | 21:21:21-08 | 2010-12-12 00:00:00 | 2013-12-11 16:00:00+08 | 2003-04-12 04:05:00
(1 row)
```

```
--删除表。
vastbase=# DROP TABLE time_type_tab;

--创建表。
vastbase=# CREATE TABLE day_type_tab (a int,b INTERVAL DAY(3) TO SECOND (4));

--插入数据。
vastbase=# INSERT INTO day_type_tab VALUES (1, INTERVAL '3' DAY);

--查看数据。
vastbase=# SELECT * FROM day_type_tab;
 a |   b
---+--------
 1 | 3 days
(1 row)

--删除表。
vastbase=# DROP TABLE day type tab;

--创建表。
vastbase=# CREATE TABLE year_type_tab(a int, b interval year (6));

--插入数据。
vastbase=# INSERT INTO year_type_tab VALUES(1,interval '2' year);

--查看数据。
vastbase=# SELECT * FROM year_type_tab;
 a |    b
---+---------
 1 | 2 years
(1 row)

--删除表。
vastbase=# DROP TABLE year_type_tab;
```

日期输入

日期和时间的输入几乎可以是任何合理的格式，包括 ISO-8601 格式、SQL-兼容格式、传统 POSTGRES 格式或者其它的形式。系统支持按照日、月、年的顺序自定义日期输入。如果把 DateStyle 参数设置为 MDY 就按照“月-日-年”解析，设置为 DMY 就按照“日-月-年”解析，设置为 YMD 就按照“年-月-日”解析。

日期的文本输入需要加单引号包围，语法如下：

type [(p)] 'value'

可选的精度声明中的 p 是一个整数，表示在秒域中小数部分的位数。表 11-12 显示了 date 类型的输入方式。

表 11-12. 日期输入方式

例子	描述

例子	描述
1999-01-08	ISO 8601 格式（建议格式），任何方式下都是 1999 年 1 月 8 号。
January 8, 1999	在任何 datestyle 输入模式下都无歧义。
1/8/1999	有歧义，在 MDY 模式下是一月八号，在 DMY 模式下是八月一号。
1/18/1999	MDY 模式下是一月十八日，其它模式下被拒绝。
01/02/03	• MDY 模式下的 2003 年 1 月 2 日。 • DMY 模式下的 2003 年 2 月 1 日。 • YMD 模式下的 2001 年 2 月 3 日。
1999-Jan-08	任何模式下都是 1 月 8 日。
Jan-08-1999	任何模式下都是 1 月 8 日。
08-Jan-1999	任何模式下都是 1 月 8 日。
99-Jan-08	YMD 模式下是 1 月 8 日，否则错误。
08-Jan-99	一月八日，除了在 YMD 模式下是错误的之外。
Jan-08-99	一月八日，除了在 YMD 模式下是错误的之外。
19990108	ISO 8601；任何模式下都是 1999 年 1 月 8 日。
990108	ISO 8601；任何模式下都是 1999 年 1 月 8 日。
1999.008	年和年里的第几天。
J2451187	儒略日。
January 8, 99 BC	公元前 99 年。

示例：

```
--创建表。
vastbase=# CREATE TABLE date_type_tab(coll date);

--插入数据。
vastbase=# INSERT INTO date_type_tab VALUES (date '12-10-2010');

--查看数据。
vastbase=# SELECT * FROM date_type_tab;
       coll
---------------------
 2010-12-10 00:00:00
```

```
(1 row)

--查看日期格式。
vastbase=# SHOW datestyle;
 DateStyle
-----------
 ISO, MDY
(1 row)

--设置日期格式。
vastbase=# SET datestyle='YMD';
SET

--插入数据。
vastbase=# INSERT INTO date_type_tab VALUES(date '2010-12-11');

--查看数据。
vastbase=# SELECT * FROM date_type_tab;
       coll
---------------------
 2010-12-10 00:00:00
 2010-12-11 00:00:00
(2 rows)

--删除表。
vastbase=# DROP TABLE date_type_tab;
```

时间

时间类型包括 time [(p)] without time zone 和 time [(p)] with time zone。如果只写 time 等效于 time without time zone。

如果在 time without time zone 类型的输入中声明了时区，则会忽略这个时区。

时间输入类型的详细信息请参见表 11-13，时区输入类型的详细信息请参加表 11-14。

表 11-13. 时间输入

例子	描述
05:06.8	ISO 8601
4:05:06	ISO 8601
4:05	ISO 8601
40506	ISO 8601
4:05 AM	与 04:05 一样，AM 不影响数值
4:05 PM	与 16:05 一样，输入小时数必须<= 12
04:05:06.789-8	ISO 8601
04:05:06-08:00	ISO 8601

例子	描述
04:05-08:00	ISO 8601
040506-08	ISO 8601
04:05:06 PST	缩写的时区
2003-04-12 04:05:06 America/New_York	用名称声明的时区

表 11-14. 时区输入

例子	描述
PST	太平洋标准时间（Pacific Standard Time）
America/New_York	完整时区名称
-8:00	ISO 8601 与 PST 的偏移
-800	ISO 8601 与 PST 的偏移
-8	ISO 8601 与 PST 的偏移

示例：

```
vastbase=# SELECT time '04:05:06';
  time
----------
 04:05:06
(1 row)

vastbase=# SELECT time '04:05:06 PST';
  time
----------
 04:05:06
(1 row)

vastbase=# SELECT time with time zone '04:05:06 PST';
  timetz
-------------
 04:05:06-08
(1 row)
```

特殊值

Vastbase 支持几个特殊值，在读取的时候将被转换成普通的日期/时间值，请参考表 11-15。

表 11-15. 特殊值

输入字符串	适用类型	描述
epoch	date，timestamp	1970-01-01 00:00:00+00 （Unix 系统零时）
infinity	timestamp	比任何其他时间戳都晚
-infinity	timestamp	比任何其他时间戳都早
now	date，time，timestamp	当前事务的开始时间
today	date，timestamp	今日午夜
tomorrow	date，timestamp	明日午夜
yesterday	date，timestamp	昨日午夜
allballs	time	00:00:00.00 UTC

时间段输入

reltime 的输入方式可以采用任何合法的时间段文本格式，包括数字形式（含负数和小数）及时间形式，其中时间形式的输入支持 SQL 标准格式、ISO-8601 格式、POSTGRES 格式等。另外，文本输入需要加单引号。

时间段输入的详细信息请参考表 11-16。

表 11-16. 时间段输入

输入示例	输出结果	描述
60	2 mons	采用数字表示时间段，默认单位是 day，可以是小数或负数。特别的，负数时间段，在语义上，可以理解为“早于多久”。
31.25	1 mons 1 days 06:00:00	
-365	-12 mons -5 days	
1 years 1 mons 8 days 12:00:00	1 years 1 mons 8 days 12:00:00	采用 POSTGRES 格式表示时间段，可以正负混用，不区分大小写，输出结果为将输入时间段计算并转换得到的简化 POSTGRES 格式时间段。
-13 months -10 hours	-1 years -25 days -04:00:00	
-2 YEARS +5 MONTHS 10 DAYS	-1 years -6 mons -25 days -06:00:00	
P-1.1Y10M	-3 mons -5 days -06:00:00	采用 ISO-8601 格式表示时间段，可以正负混用，不区分大小写，输出结果为将输入时间段计算并转换得到的简化 POSTGRES 格式时间段。
-12H	-12:00:00	

示例：

```
--创建表。
vastbase=# CREATE TABLE reltime_type_tab(col1 character(30), col2 reltime);

--插入数据。
vastbase=# INSERT INTO reltime_type_tab VALUES ('90', '90');
vastbase=# INSERT INTO reltime_type_tab VALUES ('-366', '-366');
vastbase=# INSERT INTO reltime_type_tab VALUES ('1975.25', '1975.25');
vastbase=# INSERT INTO reltime_type_tab VALUES ('-2 YEARS +5 MONTHS 10 DAYS', '-2 YEARS +5 MONTHS 10
DAYS');
vastbase=# INSERT INTO reltime_type_tab VALUES ('30 DAYS 12:00:00', '30 DAYS 12:00:00');
vastbase=# INSERT INTO reltime_type_tab VALUES ('P-1.1Y10M', 'P-1.1Y10M');

--查看数据。
vastbase=# SELECT * FROM reltime_type_tab;
           col1            |             col2
-------------------------------+-----------------------------------
 1975.25                     | 5 years 4 mons 29 days
 -2 YEARS +5 MONTHS 10 DAYS    | -1 years -6 mons -25 days -06:00:00
 P-1.1Y10M                   | -3 mons -5 days -06:00:00
 -366                        | -1 years -18:00:00
 90                          | 3 mons
 30 DAYS 12:00:00              | 1 mon 12:00:00
(6 rows)

--删除表。
vastbase=# DROP TABLE reltime_type_tab;
```

11.3.7. 几何类型

Vastbase 支持的几何类型请参见表 11-17。最基本的类型：点，是其它类型的基础。

表 11-17. 几何类型

名称	存储空间	说明	表现形式
point	16 字节	平面中的点	(x,y)
lseg	32 字节	（有限）线段	((x1,y1),(x2,y2))
box	32 字节	矩形	((x1,y1),(x2,y2))
path	16+16n 字节	闭合路径（与多边形类似）	((x1,y1),...)
path	16+16n 字节	开放路径	[(x1,y1),...]
polygon	40+16n 字节	多边形（与闭合路径相似）	((x1,y1),...)
circle	24 字节	圆	<(x,y),r> （圆心和半径）

Vastbase 提供了一系列的函数和操作符用来进行各种几何计算，如拉伸、转换、旋转、计算相交等。详细信息请参考 11.5.10 几何函数和操作符。

点

点是几何类型的基本二维构造单位。用下面语法描述 point 的数值：

```
( x , y )
x , y
```

x 和 y 是用浮点数表示的点的坐标。

点输出使用第一种语法。

线段

线段（lseg）是用一对点来代表的。用下面的语法描述 lseg 的数值：

```
[ ( x1 , y1 ) , ( x2 , y2 ) ]
( ( x1 , y1 ) , ( x2 , y2 ) )
( x1 , y1 ) , ( x2 , y2 )
x1 , y1   ,   x2 , y2
```

(x1,y1)和(x2,y2)表示线段的端点。

线段输出使用第一种语法。

矩形

矩形是用一对对角点来表示的。用下面的语法描述 box 的值：

```
( ( x1 , y1 ) , ( x2 , y2 ) )
( x1 , y1 ) , ( x2 , y2 )
x1 , y1   ,   x2 , y2
```

(x1,y1)和(x2,y2)表示矩形的一对对角点。

矩形的输出使用第二种语法。

任何两个对角都可以出现在输入中，但按照那样的顺序，右上角和左下角的值会被重新排序以存储。

路径

路径由一系列连接的点组成。路径可能是开放的，也就是认为列表中第一个点和最后一个点没有连接，也可能是闭合的，这时认为第一个和最后一个点连接起来。

用下面的语法描述 path 的数值：

```
[ ( x1 , y1 ) , ... , ( xn , yn ) ]
( ( x1 , y1 ) , ... , ( xn , yn ) )
( x1 , y1 ) , ... , ( xn , yn )
( x1 , y1   , ... ,   xn , yn )
x1 , y1   , ... ,   xn , yn
```

点表示组成路径的线段的端点。方括弧（[]）表明一个开放的路径，圆括弧（()）表明一个闭合的路径。当最外层的括号被省略，如在第三至第五语法，会假定一个封闭的路径。

路径的输出使用第一种或第二种语法输出。

多边形

多边形由一系列点代表（多边形的顶点）。多边形可以认为与闭合路径一样，但是存储方式不一样而且有自己的一套支持函数。

用下面的语法描述 polygon 的数值：

```
( ( x1 , y1 ) , ... , ( xn , yn ) )
( x1 , y1 ) , ... , ( xn , yn )
( x1 , y1   , ... ,   xn , yn )
x1 , y1   , ... ,   xn , yn
```

点表示多边形的端点。

多边形输出使用第一种语法。

圆

圆由一个圆心和半径标识。用下面的语法描述 circle 的数值：

```
< ( x , y ) , r >
( ( x , y ) , r )
( x , y ) , r
x , y   , r
```

(x,y)表示圆心，r 表示半径。

圆的输出用第一种格式。

11.3.8. 网络地址类型

Vastbase 提供用于存储 IPv4、IPv6、MAC 地址的数据类型。

用这些数据类型存储网络地址比用纯文本类型好，因为这些类型提供输入错误检查和特殊的操作和功能（请参见 11.5.11 网络地址函数和操作符）。

表 11-18. 网络地址类型

名称	存储空间	描述
cidr	7 或 19 字节	IPv4 或 IPv6 网络
inet	7 或 19 字节	IPv4 或 IPv6 主机和网络
macaddr	6 字节	MAC 地址

在对 inet 或 cidr 数据类型进行排序的时候，IPv4 地址总是排在 IPv6 地址前面，包括那些封装或者是映射在 IPv6 地址里的 IPv4 地址，比如::10.2.3.4 或::ffff:10.4.3.2。

cidr

cidr（无类别域间路由，Classless Inter-Domain Routing）类型，保存一个 IPv4 或 IPv6 网络地址。声明网络格式为 address/y，address 表示 IPv4 或者 IPv6 地址，y 表示子网掩码的二进制位数。如果省略 y，则掩码部分使用已有类别的网络编号系统进行计算，但要求输入的数据已经包括了确定掩码所需的所有字节。

表 11-19. cidr 类型输入举例

CIDR 输入	CIDR 输出	ABBREV（CIDR）
192.168.100.128/25	192.168.100.128/25	192.168.100.128/25
192.168/24	192.168.0.0/24	192.168.0/24
192.168/25	192.168.0.0/25	192.168.0.0/25
192.168.1	192.168.1.0/24	192.168.1/24
192.168	192.168.0.0/24	192.168.0/24
10.1.2	10.1.2.0/24	10.1.2/24
10.1	10.1.0.0/16	10.1/16
10	10.0.0.0/8	10/8
10.1.2.3/32	10.1.2.3/32	10.1.2.3/32
2001:4f8:3:ba::/64	2001:4f8:3:ba::/64	2001:4f8:3:ba::/64
2001:4f8:3:ba:2e0:81ff:fe22:d1f1/128	2001:4f8:3:ba:2e0:81ff:fe22:d1f1/128	2001:4f8:3:ba:2e0:81ff:fe22:d1f1
::ffff:1.2.3.0/120	::ffff:1.2.3.0/120	::ffff:1.2.3/120
::ffff:1.2.3.0/128	::ffff:1.2.3.0/128	::ffff:1.2.3.0/128

inet

inet 类型在一个数据区域内保存主机的 IPv4 或 IPv6 地址，以及一个可选子网。主机地址中网络地址的位数表示子网（“子网掩码”）。如果子网掩码是 32 并且地址是 IPv4，则这个值不表示任何子网，只表示一台主机。在 IPv6 里，地址长度是 128 位，因此 128 位表示唯一的主机地址。

该类型的输入格式是 address/y，address 表示 IPv4 或者 IPv6 地址，y 是子网掩码的二进制位数。如果省略/y，则子网掩码对 IPv4 是 32，对 IPv6 是 128，所以该值表示只有一台主机。如果该值表示只有一台主机，/y 将不会显示。

inet 和 cidr 类型之间的基本区别是 inet 接受子网掩码，而 cidr 不接受。

macaddr

macaddr 类型存储 MAC 地址，也就是以太网卡硬件地址（尽管 MAC 地址还用于其它用途）。可以接受下列格式：

```
'08:00:2b:01:02:03'
'08-00-2b-01-02-03'
'08002b:010203'
'08002b-010203'
'0800.2b01.0203'
'08002b010203'
```

这些示例都表示同一个地址。对于数据位 a 到 f，大小写都行。输出时都是以第一种形式展示。

11.3.9. 位串类型

位串就是一串 1 和 0 的字符串。它们可以用于存储位掩码。

Vastbase 支持两种位串类型：bit(n)和 bit varying(n)，这里的 n 是一个正整数。

bit 类型的数据必须准确匹配长度 n，如果存储短或者长的数据都会报错。bit varying 类型的数据是最长为 n 的变长类型，超过 n 的类型会被拒绝。一个没有长度的 bit 等效于 bit(1)，没有长度的 bit varying 表示没有长度限制。

说明

如果用户明确地把一个位串值转换成 bit(n)，则此位串右边的内容将被截断或者在右边补齐零，直到刚好 n 位，而不会抛出任何错误。

如果用户明确地把一个位串数值转换成 bit varying(n)，如果它超过了 n 位，则它的右边将被截断。

```
--创建表。
vastbase=# CREATE TABLE bit_type_t1
(
   BT_COL1 INTEGER,
   BT_COL2 BIT(3),
   BT_COL3 BIT VARYING(5)
) ;

--插入数据。
vastbase=# INSERT INTO bit_type_t1 VALUES(1, B'101', B'00');

--插入数据的长度不符合类型的标准会报错。
vastbase=# INSERT INTO bit_type_t1 VALUES(2, B'10', B'101');
ERROR:  bit string length 2 does not match type bit(3)
CONTEXT:  referenced column: bt_col2

--将不符合类型长度的数据进行转换。
vastbase=# INSERT INTO bit_type_t1 VALUES(2, B'10'::bit(3), B'101');

--查看数据。
vastbase=# SELECT * FROM bit_type_t1;
 bt_col1 | bt_col2 | bt_col3
---------+---------+---------
       1 | 101     | 00
```

```
     2 | 100    | 101
(2 rows)

--删除表。
vastbase=# DROP TABLE bit_type_t1;
```

11.3.10.文本搜索类型

Vastbase 提供了两种数据类型用于支持全文检索。tsvector 类型表示为文本搜索优化的文件格式，tsquery 类型表示文本查询。

tsvector

tsvector 类型表示一个检索单元，通常是一个数据库表中一行的文本字段或者这些字段的组合，tsvector 类型的值是一个标准词位的有序列表，标准词位就是把同一个词的变型体都标准化相同的，在输入的同时会自动排序和消除重复。to_tsvector 函数通常用于解析和标准化文档字符串。

tsvector 的值是唯一分词的分类列表，把一句话的词格式化为不同的词条，在进行分词处理的时候 tsvector 会自动去掉分词中重复的词条，按照一定的顺序录入。如：

```
vastbase=# SELECT 'a fat cat sat on a mat and ate a fat rat'::tsvector;
                    tsvector
----------------------------------------------------
 'a' 'and' 'ate' 'cat' 'fat' 'mat' 'on' 'rat' 'sat'
(1 row)
```

从上面的例子可以看出，通过 tsvector 把一个字符串按照空格进行分词，分词的顺序是按照长短和字母排序的。但是如果词条中需要包含空格或标点符号，可以用引号标记：

```
vastbase=# SELECT $$the lexeme '    ' contains spaces$$::tsvector;
              tsvector
----------------------------------------
 '    ' 'contains' 'lexeme' 'spaces' 'the'
(1 row)
```

如果在词条中使用引号，可以使用双$$符号作为标记：

```
vastbase=# SELECT $$the lexeme 'Joe''s' contains a quote$$::tsvector;
                 tsvector
-----------------------------------------------
 'Joe''s' 'a' 'contains' 'lexeme' 'quote' 'the'
(1 row)
```

词条位置常量也可以放到词汇中：

```
vastbase=# SELECT 'a:1 fat:2 cat:3 sat:4 on:5 a:6 mat:7 and:8 ate:9 a:10 fat:11 rat:12'::tsvector;
                             tsvector
-------------------------------------------------------------------------------
 'a':1,6,10 'and':8 'ate':9 'cat':3 'fat':2,11 'mat':7 'on':5 'rat':12 'sat':4
(1 row)
```

位置常量通常表示文档中源字的位置。位置信息可以用于进行排名。位置常量的范围是 1 到 16383，最大值默认是 16383。相同词的重复位会被忽略掉。

拥有位置的词汇甚至可以用一个权来标记，这个权可以是 A，B，C 或 D。默认的是 D，因此输出中不会出现：

```
vastbase=# SELECT 'a:1A fat:2B,4C cat:5D'::tsvector;
        tsvector
----------------------------
 'a':1A 'cat':5 'fat':2B,4C
(1 row)
```

权可以用来反映文档结构，如：标记标题与主体文字的区别。全文检索排序函数可以为不同的权标记分配不同的优先级。

下面的示例是 tsvector 类型标准用法。如：

```
vastbase=# SELECT 'The Fat Rats'::tsvector;
     tsvector
--------------------
 'Fat' 'Rats' 'The'
(1 row)
```

但是对于英文全文检索应用来说，上面的单词会被认为非规范化的，所以需要通过 to_tsvector 函数对这些单词进行规范化处理：

```
vastbase=# SELECT to_tsvector('english', 'The Fat Rats');
  to_tsvector
-----------------
 'fat':2 'rat':3
(1 row)
```

tsquery

tsquery 类型表示一个检索条件，存储用于检索的词汇，并且使用布尔操作符&（AND），|（OR）和!（NOT）来组合他们，括号用来强调操作符的分组。to_tsquery 函数及 plainto_tsquery 函数会将单词转换为 tsquery 类型前进行规范化处理。

```
vastbase=# SELECT 'fat & rat'::tsquery;
   tsquery
---------------
 'fat' & 'rat'
(1 row)

vastbase=# SELECT 'fat & (rat | cat)'::tsquery;
        tsquery
---------------------------
 'fat' & ( 'rat' | 'cat' )
(1 row)

vastbase=# SELECT 'fat & rat & ! cat'::tsquery;
      tsquery
------------------------
 'fat' & 'rat' & !'cat'
(1 row)
```

在没有括号的情况下，！（非）结合的最紧密，而&（和）结合的比|（或）紧密。

tsquery 中的词汇可以用一个或多个权字母来标记，这些权字母限制这次词汇只能与带有匹配权的 tsvector 词汇进行匹配。

```
vastbase=# SELECT 'fat:ab & cat'::tsquery;
    tsquery
------------------
```

```
 'fat':AB & 'cat'
(1 row)
```

同样，tsquery 中的词汇可以用*标记来指定前缀匹配：

```
vastbase=# SELECT 'super:*'::tsquery;
  tsquery
-----------
 'super':*
(1 row)
```

这个查询可以匹配 tsvector 中以“super”开始的任意单词。

请注意，前缀首先被文本搜索分词器处理，这也就意味着下面的结果为真：

```
vastbase=# SELECT to_tsvector( 'postgraduate' ) @@ to_tsquery( 'postgres:*' ) AS RESULT;
 result
----------
 t
(1 row)
```

因为 vastbase 经过处理后得到 postgr：

```
vastbase=# SELECT to_tsquery('postgres:*');
 to_tsquery
------------
 'postgr':*
(1 row)
```

这样就匹配 postgraduate 了。

'Fat:ab & Cats'规范化转为 tsquery 类型结果如下：

```
vastbase=# SELECT to_tsquery('Fat:ab & Cats');
    to_tsquery
------------------
 'fat':AB & 'cat'
(1 row)
```

11.3.11.UUID 类型

UUID 数据类型用来存储 RFC 4122，ISO/IEF 9834-8:2005 以及相关标准定义的通用唯一标识符（UUID）。这个标识符是一个由算法产生的 128 位标识符，确保它不可能使用相同算法在已知的模块中产生的相同标识符。

因此，对分布式系统而言，这种标识符比序列能更好的保证唯一性，因为序列只能在单一数据库中保证是唯一。

UUID 是一个小写十六进制数字的序列，由分字符分成几组，一组 8 位数字+三组 4 位数字+一组 12 位数字，总共 32 个数字代表 128 位，标准的 UUID 示例如下：

```
a0eebc99-9c0b-4ef8-bb6d-6bb9bd380a11
```

Vastbase 同样支持以其他方式输入：大写字母和数字、由花括号包围的标准格式、省略部分或所有连字符、在任意一组四位数字之后加一个连字符。示例：

```
A0EEBC99-9C0B-4EF8-BB6D-6BB9BD380A11
{a0eebc99-9c0b-4ef8-bb6d-6bb9bd380a11}
a0eebc999c0b4ef8bb6d6bb9bd380a11
a0ee-bc99-9c0b-4ef8-bb6d-6bb9-bd38-0a11
```

一般是以标准格式输出。

11.3.12.JSON 类型

JSON 数据类型可以用来存储 JSON（JavaScript Object Notation）数据。数据可以存储为 text，但是 JSON 数据类型更有利于检查每个存储的数值是可用的 JSON 值。

JSON 类型相关的支持函数请参见 11.5.13JSON 函数。

11.3.13.XML 类型

xml 类型用于存储 XML 数据。使用字符串也可以存储 XML 数据，但不能保证其合法性。支持 xml 类型后，数据库会对数据进行合法性检查，同时提供函数进行类型安全性检查。

xml 类型中存储数据有两种：documents 和 content。content 可以有多个顶级元素，documents 只能有一个顶级元素。默认情况下是 content。

示例：

```
查看当前的存储类型:
vastbase=# show xmloption;
 xmloption
-----------
 content
(1 row)

修改当前的存储类型:
vastbase=# SET xmloption TO document;
SET

vastbase=# CREATE TABLE T1(x1 xml);
CREATE TABLE

vastbase=#insert into t1 select  xml'<title> hello world</title>';
vastbase=#select * from t1;
          xml
----------------------------
 <title> hello world</title>
(1 row)
```

11.3.14.对象标识符类型

Vastbase 在内部使用对象标识符（OID）作为各种系统表的主键。系统不会给用户创建的表增加一个 OID 系统字段，OID 类型代表一个对象标识符。

目前 OID 类型用一个四字节的无符号整数实现。因此不建议在创建的表中使用 OID 字段做主键。

表 11-20. 对象标识符类型

名称	引用	描述	示例
OID	-	数字化的对象标识符。	564182
CID	-	命令标识符。它是系统字段 cmin 和 cmax 的数据类型。命令标识符是 32 位的量。	-
XID	-	事务标识符。它是系统字段 xmin 和 xmax 的数据类型。事务标识符也是 32 位的量。	-
TID	-	行标识符。它是系统表字段 ctid 的数据类型。行 ID 是一对数值（块号，块内的行索引），它标识该行在其所在表内的物理位置。	-
REGCONFIG	pg_ts_config	文本搜索配置。	english
REGDICTIONARY	pg_ts_dict	文本搜索字典。	simple
REGOPER	pg_operator	操作符名。	-
REGOPERATOR	pg_operator	带参数类型的操作符。	*(integer,integer)或 -(NONE,integer)
REGPROC	pg_proc	函数名称。	sum
REGPROCEDURE	pg_proc	带参数类型的函数。	sum(int4)
REGCLASS	pg_class	关系名。	pg_type
REGTYPE	pg_type	数据类型名。	integer

OID 类型：主要作为数据库系统表中字段使用。

示例：

```
vastbase=# SELECT oid FROM pg_class WHERE relname = 'pg_type';
 oid
------
 1247
(1 row)
```

OID 别名类型 REGCLASS：主要用于对象 OID 值的简化查找。

示例：

```
vastbase=# SELECT attrelid,attname,atttypid,attstattarget FROM pg_attribute WHERE attrelid =
'pg_type'::REGCLASS;
 attrelid |  attname   | atttypid | attstattarget
----------+------------+----------+---------------
    1247 | xc_node_id    |      23 |           0
    1247 | tableoid      |      26 |           0
    1247 | cmax          |      29 |           0
    1247 | xmax          |      28 |           0
    1247 | cmin          |      29 |           0
    1247 | xmin          |      28 |           0
    1247 | oid           |      26 |           0
    1247 | ctid          |      27 |           0
    1247 | typname       |      19 |          -1
    1247 | typnamespace  |      26 |          -1
    1247 | typowner      |      26 |          -1
    1247 | typlen        |      21 |          -1
    1247 | typbyval      |      16 |          -1
    1247 | typtype       |      18 |          -1
    1247 | typcategory   |      18 |          -1
    1247 | typispreferred |      16 |          -1
    1247 | typisdefined  |      16 |          -1
    1247 | typdelim      |      18 |          -1
    1247 | typrelid      |      26 |          -1
    1247 | typelem       |      26 |          -1
    1247 | typarray      |      26 |          -1
    1247 | typinput      |      24 |          -1
    1247 | typoutput     |      24 |          -1
    1247 | typreceive    |      24 |          -1
    1247 | typsend       |      24 |          -1
    1247 | typmodin      |      24 |          -1
    1247 | typmodout     |      24 |          -1
    1247 | typanalyze    |      24 |          -1
    1247 | typalign      |      18 |          -1
    1247 | typstorage    |      18 |          -1
    1247 | typnotnull    |      16 |          -1
    1247 | typbasetype   |      26 |          -1
    1247 | typtypmod     |      23 |          -1
    1247 | typndims      |      23 |          -1
    1247 | typcollation  |      26 |          -1
    1247 | typdefaultbin |     194 |          -1
    1247 | typdefault    |      25 |          -1
    1247 | typacl        |    1034 |          -1
(38 rows)
```

11.3.15.伪类型

Vastbase 数据类型中包含一系列特殊用途的类型，这些类型按照类别被称为伪类型。伪类型不能作为字段的数据类型，但是可以用于声明函数的参数或者结果类型。

当一个函数不仅是简单地接受并返回某种 SQL 数据类型的情况下伪类型是很有用的。表 11-21 列出了所有的伪类型。

表 11-21. 伪类型

名称	描述
any	表示函数接受任何输入数据类型。
anyelement	表示函数接受任何数据类型。
anyarray	表示函数接受任意数组数据类型。
anynonarray	表示函数接受任意非数组数据类型。
anyenum	表示函数接受任意枚举数据类型。
anyrange	表示函数接受任意范围数据类型。
cstring	表示函数接受或者返回一个空结尾的 C 字符串。
internal	表示函数接受或者返回一种服务器内部的数据类型。
language_handler	声明一个过程语言调用句柄返回 language_handler。
fdw_handler	声明一个外部数据封装器返回 fdw_handler。
record	标识函数返回一个未声明的行类型。
trigger	声明一个触发器函数返回 trigger。
void	表示函数不返回数值。
opaque	一个已经过时的类型，以前用于所有上面这些用途。

声明用 C 编写的函数（不管是内置的还是动态装载的）都可以接受或者返回任何这样的伪数据类型。当伪类型作为参数类型使用时，用户需要保证函数的正常运行。

用过程语言编写的函数只能使用实现语言允许的伪类型。目前，过程语言都不允许使用作为参数类型的伪类型，并且只允许使用 void 和 record 作为结果类型。一些多态的函数还支持使用 anyelement，anyarray，anynonarray anyenum 和 anyrange 类型。

伪类型 internal 用于声明那种只能在数据库系统内部调用的函数，他们不能直接在 SQL 查询里调用。如果函数至少有一个 internal 类型的参数，则不能从 SQL 里调用他。建议不要创建任何声明返回 internal 的函数，除非他至少有一个 internal 类型的参数。

示例：

```
--创建表
vastbase=# create table t1 (a int);

--插入两条数据
vastbase=# insert into t1 values(1),(2);
```

```
--创建函数showall()。
vastbase=# CREATE OR REPLACE FUNCTION showall() RETURNS SETOF record
AS $$ SELECT count(*) from t1; $$
LANGUAGE SQL;

--调用函数showall()。
vastbase=# SELECT showall();
 showall
---------
 (2)
(1 row)

--删除函数。
vastbase=# DROP FUNCTION showall();

--删除表
vastbase=# drop table t1;
```

11.3.16.列存表支持的数据类型

列存表支持的数据类型如表 11-22 所示。

表 11-22. 列存表支持的数据类型

类别	数据类型	长度	是否支持
Numeric Types	smallint	2	支持
	integer	4	支持
	bigint	8	支持
	decimal	-1	支持
	numeric	-1	支持
	real	4	支持
	double precision	8	支持
	smallserial	2	支持
	serial	4	支持
	bigserial	8	支持
Monetary Types	money	8	支持
Character Types	character varying(n), varchar(n)	-1	支持

类别	数据类型	长度	是否支持
	character(n), char(n)	n	支持
	character、char	1	支持
	text	-1	支持
	nvarchar2	-1	支持
	name	64	不支持
Date/Time Types	timestamp with time zone	8	支持
	timestamp without time zone	8	支持
	date	4	支持
	time without time zone	8	支持
	time with time zone	12	支持
	interval	16	支持
big object	clob	-1	支持
	blob	-1	不支持
other types	...	...	不支持

11.4. 常量与宏

Vastbase 支持的常量和宏请参见表 11-23。

表 11-23. 常量和宏

参数	描述	示例
CURRENT_CATALOG	当前数据库	vastbase=# SELECT CURRENT_CATALOG; current_database ---------- vastbase (1 row)
CURRENT_ROLE	当前用户	vastbase=# SELECT CURRENT_ROLE; current_user ----------

参数	描述	示例
		vastbase (1 row)
CURRENT_SCHEMA	当前数据库模式	vastbase=# SELECT CURRENT_SCHEMA; current_schema ---------- public (1 row)
CURRENT_USER	当前用户	vastbase=# SELECT CURRENT_USER; current_user ---------- vastbase (1 row)
LOCALTIMESTAMP	当前会话时间（无时区）	vastbase=# SELECT LOCALTIMESTAMP; timestamp ---------- 2015-10-10 15:37:30.968538 (1 row)
NULL	空值	-
SESSION_USER	当前系统用户	vastbase=# SELECT SESSION_USER; session_user ---------- vastbase (1 row)
SYSDATE	当前系统日期	vastbase=# SELECT SYSDATE; sysdate ---------- 2015-10-10 15:48:53 (1 row)
USER	当前用户，此用户为 CURRENT_USER 的别名。	vastbase=# SELECT USER; current_user ---------- vastbase (1 row)

11.5. 函数和操作符

11.5.1. 逻辑操作符

常用的逻辑操作符有 AND、OR 和 NOT，他们的运算结果有三个值，分别为 TRUE、FALSE 和 NULL，其中 NULL 代表未知。他们运算优先级顺序为：NOT>AND>OR。

运算规则请参见表 11-24，表中的 A 和 B 代表逻辑表达式。

表 11-24. 运算规则表

A	B	A AND B 的结果	A OR B 的结果	NOT A 的结果
TRUE	TRUE	TRUE	TRUE	FALSE
TRUE	FALSE	FALSE	TRUE	FALSE
TRUE	NULL	NULL	TRUE	FALSE
FALSE	FALSE	FALSE	FALSE	TRUE
FALSE	NULL	FALSE	NULL	TRUE
NULL	NULL	NULL	NULL	NULL

说明

操作符 AND 和 OR 具有交换性，即交换左右两个操作数，不影响其结果。

11.5.2. 比较操作符

所有数据类型都可用比较操作符进行比较，并返回一个布尔类型的值。

比较操作符均为双目操作符，被比较的两个数据类型必须是相同的数据类型或者是可以进行隐式转换的类型。

Vastbase 提供的比较操作符请参见表 11-25。

表 11-25. 比较操作符

操作符	描述
<	小于
>	大于
<=	小于或等于
>=	大于或等于

操作符	描述
=	等于
<> 或 !=	不等于

比较操作符可以用于所有相关的数据类型。所有比较操作符都是双目操作符，返回布尔类型数值。像 1<2<3 这样的表达式是非法的。（因为布尔值和 3 之间不能做比较。）

11.5.3. 字符处理函数和操作符

Vastbase 提供的字符处理函数和操作符主要用于字符串与字符串、字符串与非字符串之间的连接，以及字符串的模式匹配操作。

- bit_length(string)

  描述：字符串的位数。

  返回值类型：int

  示例：

```
vastbase=# SELECT bit_length('world');
 bit_length
------------
         40
(1 row)
```

- btrim(string text [, characters text])

  描述：从 string 开头和结尾删除只包含 characters 中字符（缺省是空白）的最长字符串。

  返回值类型：text

  示例：

```
vastbase=# SELECT btrim('sring' , 'ing');
 btrim
-------
 sr
(1 row)
```

- char_length(string)或 character_length(string)

  描述：字符串中的字符个数。

  返回值类型：int

  示例：

```
vastbase=# SELECT char_length('hello');
 char_length
-------------
           5
(1 row)
```

- instr(text,text,int,int)

  描述：instr(string1,string2,int1,int2)返回在 string1 中从 int1 位置开始匹配到第 int2 次 string2 的位置，第一个 int 表示开始匹配起始位置，第二个 int 表示匹配的次数。

  返回值类型：int

  示例：

```
vastbase=# SELECT instr( 'abcdabcdabcd', 'bcd', 2, 2 );
 instr
-------
     6
(1 row)
```

- lengthb(text/bpchar)

  描述：获取指定字符串的字节数。

  返回值类型：int

  示例：

```
vastbase=# SELECT lengthb('hello');
 lengthb
---------
       5
(1 row)
```

- left(str text, n int)

  描述：返回字符串的前 n 个字符。当 n 是负数时，返回除最后|n|个字符以外的所有字符。

  返回值类型：text

  示例：

```
vastbase=# SELECT left('abcde', 2);
 left
------
 ab
(1 row)
```

- length(string bytea, encoding name)

  描述：指定 encoding 编码格式的 string 的字符数。在这个编码格式中，string 必须是有效的。

  返回值类型：int

  示例：

```
vastbase=# SELECT length('jose', 'UTF8');
 length
--------
      4
(1 row)
```

- lpad(string text, length int [, fill text])

  描述：通过填充字符 fill（缺省时为空白），把 string 填充为 length 长度。如果 string 已经比 length 长则将其尾部截断。

返回值类型：text

示例：

```
vastbase=# SELECT lpad('hi', 5, 'xyza');
 lpad
-------
 xyzhi
(1 row)
```

- octet_length(string)

描述：字符串中的字节数。

返回值类型：int

示例：

```
vastbase=# SELECT octet_length('jose');
 octet_length
--------------
            4
(1 row)
```

- overlay(string placing string FROM int [for int])

描述：替换子字符串。FROM int 表示从第一个 string 的第几个字符开始替换，for int 表示替换第一个 string 的字符数目。

返回值类型：text

示例：

```
vastbase=# SELECT overlay('hello' placing 'world' from 2 for 3 );
 overlay
---------
 hworldo
(1 row)
```

- position(substring in string)

描述：指定子字符串的位置。

返回值类型：int

示例：

```
vastbase=# SELECT position('ing' in 'string');
 position
----------
        4
(1 row)
```

- pg_client_encoding()

描述：当前客户端编码名称。

返回值类型：name

示例：

```
vastbase=# SELECT pg_client_encoding();
 pg_client_encoding
--------------------
 UTF8
(1 row)
```

- quote_ident(string text)

描述：返回适用于 SQL 语句的标识符形式（使用适当的引号进行界定）。只有在必要的时候才会添加引号（字符串包含非标识符字符或者会转换大小写的字符）。返回值中嵌入的引号都写了两次。

返回值类型：text

示例：

```
vastbase=# SELECT quote_ident('hello world');
 quote_ident
--------------
 "hello world"
(1 row)
```

- quote_literal(string text)

描述：返回适用于在 SQL 语句里当作文本使用的形式（使用适当的引号进行界定）。

返回值类型：text

示例：

```
vastbase=# SELECT quote_literal('hello');
 quote_literal
---------------
 'hello'
(1 row)
```

如果出现如下写法，text 文本将进行转义。

```
vastbase=# SELECT quote_literal(E'O\'hello');
 quote_literal
---------------
 'O''hello'
(1 row)
```

如果出现如下写法，反斜杠会写入两次。

```
vastbase=# SELECT quote_literal('O\hello');
 quote_literal
---------------
 E'O\\hello'
(1 row)
```

如果参数为 NULL，返回空。如果参数可能为 null，通常使用函数 quote_nullable 更适用。

```
vastbase=# SELECT quote_literal(NULL);
 quote_literal
---------------

(1 row)
```

- quote_literal(value anyelement)

描述：将给定的值强制转换为 text，加上引号作为文本。

返回值类型：text

示例：

```
vastbase=# SELECT quote_literal(42.5);
 quote_literal
---------------
 '42.5'
(1 row)
```

如果出现如下写法，定值将进行转义。

```
vastbase=# SELECT quote_literal(E'O\'42.5');
 quote_literal
---------------
 'O''42.5'
(1 row)
```

如果出现如下写法，反斜杠会写入两次。

```
vastbase=# SELECT quote_literal('O\42.5');
 quote_literal
---------------
 E'O\\42.5'
(1 row)
```

- quote_nullable(string text)

描述：返回适用于在 SQL 语句里当作字符串使用的形式（使用适当的引号进行界定）。

返回值类型：text

示例：

```
vastbase=# SELECT quote_nullable('hello');
 quote_nullable
----------------
 'hello'
(1 row)
```

如果出现如下写法，text 文本将进行转义。

```
vastbase=# SELECT quote_nullable(E'O\'hello');
 quote_nullable
----------------
 'O''hello'
(1 row)
```

如果出现如下写法，反斜杠会写入两次。

```
vastbase=# SELECT quote_nullable('O\hello');
 quote_nullable
----------------
 E'O\\hello'
(1 row)
```

如果参数为 NULL，返回 NULL。

```
vastbase=# SELECT quote_nullable(NULL);
 quote_nullable
```

```
----------------
 NULL
(1 row)
```

- quote_nullable(value anyelement)

  描述：将给定的参数值转化为 text，加上引号作为文本。

  返回值类型：text

  示例：

```
vastbase=# SELECT quote_nullable(42.5);
 quote_nullable
----------------
 '42.5'
(1 row)
```

  如果出现如下写法，定值将进行转义。

```
vastbase=# SELECT quote_nullable(E'O\'42.5');
 quote_nullable
----------------
 'O''42.5'
(1 row)
```

  如果出现如下写法，反斜杠会写入两次。

```
vastbase=# SELECT quote_nullable('O\42.5');
 quote_nullable
----------------
 E'O\\42.5'
(1 row)
```

  如果参数为 NULL，返回 NULL。

```
vastbase=# SELECT quote nullable(NULL);
 quote nullable
----------------
 NULL
(1 row)
```

- substring(string [from int] [for int])

  描述：截取子字符串，from int 表示从第几个字符开始截取，for int 表示截取几个字节。

  返回值类型：text

  示例：

```
vastbase=# SELECT substring('Thomas' from 2 for 3);
 substring
-----------
 hom
(1 row)
```

- substring(string from pattern)

  描述：截取匹配 POSIX 正则表达式的子字符串。如果没有匹配它返回空值，否则返回文本中匹配模式的那部分。

  返回值类型：text

示例：

```
vastbase=# SELECT substring('Thomas' from '...$');
 substring
-----------
 mas
(1 row)
vastbase=# SELECT substring('foobar' from 'o(.)b');
 result
--------
 o
(1 row)
vastbase=# SELECT substring('foobar' from '(o(.)b)');
 result
--------
 oob
(1 row)
```

说明

如果 POSIX 正则表达式模式包含任何圆括号，那么将返回匹配第一对子表达式（对应第一个左圆括号的） 的文本。如果你想在表达式里使用圆括号而又不想导致这个例外，那么你可以在整个表达式外边放上一对圆括号。

- substring(string from pattern for escape)

  描述：截取匹配 SQL 正则表达式的子字符串。声明的模式必须匹配整个数据串，否则函数失败并返回空值。为了标识在成功的时候应该返回的模式部分，模式必须包含逃逸字符的两次出现，并且后面要跟上双引号（"）。匹配这两个标记之间的模式的文本将被返回。

  返回值类型：text

  示例：

```
vastbase=# SELECT substring('Thomas' from '%#"o_a#"_' for '#');
 substring
-----------
 oma
(1 row)
```

- rawcat(raw,raw)

  描述：字符串拼接函数。

  返回值类型：raw

  示例：

```
vastbase=# SELECT rawcat('ab','cd');
 rawcat
--------
 ABCD
(1 row)
```

- regexp_like(text,text,text)

  描述：正则表达式的模式匹配函数。

  返回值类型：bool

示例：

```
vastbase=# SELECT regexp_like('str','[ac]');
 regexp_like
-------------
 f
(1 row)
```

❖ regexp_substr(text,text)

描述：正则表达式的抽取子串函数。与 substr 功能相似，正则表达式出现多个并列的括号时，也全部处理。

返回值类型：text

示例：

```
vastbase=# SELECT regexp_substr('str','[ac]');
 regexp_substr
---------------

(1 row)
```

❖ regexp_matches(string text, pattern text [, flags text])

描述：返回 string 中所有匹配 POSIX 正则表达式的子字符串。如果 pattern 不匹配，该函数不返回行。如果模式不包含圆括号子表达式，则每一个被返回的行都是一个单一元素的文本数组，其中包括匹配整个模式的子串。如果模式包含圆括号子表达式，该函数返回一个文本数组，它的第 n 个元素是匹配模式的第 n 个圆括号子表达式的子串。

flags 参数为可选参数，包含零个或多个改变函数行为的单字母标记。i 表示进行大小写无关的匹配，g 表示替换每一个匹配的子字符串而不仅仅是第一个。

须知

如果提供了最后一个参数，但参数值是空字符串（''），且数据库 SQL 兼容模式设置为 ORA 的情况下，会导致返回结果为空集。这是因为 ORA 兼容模式将''作为 NULL 处理，避免此类行为的方式有如下几种：

- 将数据库 SQL 兼容模式改为 TD；
- 不提供最后一个参数，或最后一个参数不为空字符串。

返回值类型：setof text[]

示例：

```
vastbase=# SELECT regexp_matches('foobarbequebaz', '(bar)(beque)');
 regexp_matches
----------------
 {bar,beque}
(1 row)
vastbase=# SELECT regexp_matches('foobarbequebaz', 'barbeque');
 regexp_matches
----------------
 {barbeque}
```

```
(1 row)
vastbase=#  SELECT regexp_matches('foobarbequebazilbarfbonk', '(b[^b]+)(b[^b]+)', 'g');
   result
--------------
 {bar,beque}
 {bazil,barf}
(2 rows)
```

- regexp_split_to_array(string text, pattern text [, flags text])

描述：用 POSIX 正则表达式作为分隔符，分隔 string。和 regexp_split_to_table 相同，不过 regexp_split_to_array 会把它的结果以一个 text 数组的形式返回。

返回值类型：text[]

示例：

```
vastbase=# SELECT regexp_split_to_array('hello world', E'\\s+');
 regexp_split_to_array
-----------------------
 {hello,world}
(1 row)
```

- regexp_split_to_table(string text, pattern text [, flags text])

描述：用 POSIX 正则表达式作为分隔符，分隔 string。如果没有与 pattern 的匹配，该函数返回 string。如果有至少有一个匹配，对每一个匹配它都返回从上一个匹配的末尾（或者串的开头）到这次匹配开头之间的文本。当没有更多匹配时，它返回从上一次匹配的末尾到串末尾之间的文本。

flags 参数包含零个或多个改变函数行为的单字母标记。i 表示进行大小写无关的匹配，g 表示替换每一个匹配的子字符串而不仅仅是第一个。

返回值类型：setof text

示例：

```
vastbase=# SELECT regexp_split_to_table('hello world', E'\\s+');
 regexp_split_to_table
-----------------------
 hello
 world
(2 rows)
```

- repeat(string text, number int)

描述：将 string 重复 number 次。

返回值类型：text。

示例：

```
vastbase=# SELECT repeat('Pg', 4);
  repeat
----------
 PgPgPgPg
(1 row)
```

说明

由于数据库内存分配机制限制单次内存分配不可超过1GB，因此number最大值不应超过(1G-x)/lengthb(string) - 1。x 为头信息长度，通常大于 4 字节，其具体值在不同的场景下存在差异。

- replace(string text, from text, to text)

  描述：把字符串 string 里出现地所有子字符串 from 的内容替换成子字符串 to 的内容。

  返回值类型：text

  示例：

```
vastbase=# SELECT replace('abcdefabcdef', 'cd', 'XXX');
   replace
----------------
 abXXXefabXXXef
(1 row)
```

- reverse(str)

  描述：返回颠倒的字符串。

  返回值类型：text

  示例：

```
vastbase=# SELECT reverse('abcde');
 reverse
---------
 edcba
(1 row)
```

- right(str text, n int)

  描述：返回字符串中的后 n 个字符。当 n 是负值时，返回除前|n|个字符以外的所有字符。

  返回值类型：text

  示例：

```
vastbase=# SELECT right('abcde', 2);
 right
-------
 de
(1 row)

vastbase=# SELECT right('abcde', -2);
 right
-------
 cde
(1 row)
```

- rpad(string text, length int [, fill text])

  描述：使用填充字符 fill（缺省时为空白），把 string 填充到 length 长度。如果 string 已经比 length 长则将其从尾部截断。

  返回值类型：text

示例：

```
vastbase=# SELECT rpad('hi', 5, 'xy');
 rpad
-------
 hixyx
(1 row)
```

- rtrim(string text [, characters text])

描述：从字符串 string 的结尾删除只包含 characters 中字符（缺省是个空白）的最长的字符串。

返回值类型：text

示例：

```
vastbase=# SELECT rtrim('trimxxxx', 'x');
 rtrim
-------
 trim
(1 row)
```

- substrb(text,int,int)

描述：提取子字符串，第一个 int 表示提取的起始位置，第二个表示提取几位字符。

返回值类型：text

示例：

```
vastbase=# SELECT substrb('string',2,3);
 substrb
---------
 tri
(1 row)
```

- substrb(text,int)

描述：提取子字符串，int 表示提取的起始位置。

返回值类型：text

示例：

```
vastbase=# SELECT substrb('string',2);
 substrb
---------
 tring
(1 row)
```

- substr(bytea,from,count)

描述：从参数 bytea 中抽取子字符串。from 表示抽取的起始位置，count 表示抽取的子字符串长度。

返回值类型：text

示例：

```
vastbase=# SELECT substr('string',2,3);
 substr
--------
 tri
(1 row)
```

- string || string

  描述：连接字符串。

  返回值类型：text

  示例：

```
vastbase=# SELECT 'MPP'||'DB' AS RESULT;
 result
--------
 MPPDB
(1 row)
```

- string || non-string 或 non-string || string

  描述：连接字符串和非字符串。

  返回值类型：text

  示例：

```
vastbase=# SELECT 'Value: '||42 AS RESULT;
  result
-----------
 Value: 42
(1 row)
```

- split_part(string text, delimiter text, field int)

  描述：根据 delimiter 分隔 string 返回生成的第 field 个子字符串（从出现第一个 delimiter 的 text 为基础）。

  返回值类型：text

  示例：

```
vastbase=# SELECT split_part('abc~@~def~@~ghi', '~@~', 2);
 split_part
------------
 def
(1 row)
```

- strpos(string, substring)

  描述：指定的子字符串的位置。和 position(substring in string)一样，不过参数顺序相反。

  返回值类型：int

  示例：

```
vastbase=# SELECT strpos('source', 'rc');
 strpos
--------
      4
(1 row)
```

- to_hex(number int or bigint)

  描述：把 number 转换成十六进制表现形式。

  返回值类型：text

  示例：

```
vastbase=# SELECT to_hex(2147483647);
  to_hex
----------
 7fffffff
(1 row)
```

- translate(string text, from text, to text)

  描述：把在 string 中包含的任何匹配 from 中字符的字符转化为对应的在 to 中的字符。如果 from 比 to 长，删掉在 from 中出现的额外的字符。

  返回值类型：text

  示例：

```
vastbase=# SELECT translate('12345', '143', 'ax');
 translate
-----------
 a2x5
(1 row)
```

- length(string)

  描述：获取参数 string 中字符的数目。

  返回值类型：integer

  示例：

```
vastbase=# SELECT length('abcd');
 length
--------
      4
(1 row)
```

- lengthb(string)

  描述：获取参数 string 中字节的数目。与字符集有关，同样的中文字符，在 GBK 与 UTF8 中，返回的字节数不同。

  返回值类型：integer

  示例：

```
vastbase=# SELECT lengthb('Chinese');
 lengthb
---------
       7
(1 row)
```

- substr(string,from)

  描述：

从参数 string 中抽取子字符串。

from 表示抽取的起始位置。

- from 为 0 时，按 1 处理。
- from 为正数时，抽取从 from 到末尾的所有字符。
- from 为负数时，抽取字符串的后 n 个字符，n 为 from 的绝对值。

返回值类型：varchar

示例：

from 为正数时：

```
vastbase=# SELECT substr('ABCDEF',2);
 substr
--------
 BCDEF
(1 row)
```

from 为负数时：

```
vastbase=# SELECT substr('ABCDEF',-2);
 substr
--------
 EF
(1 row)
```

❖ substr(string,from,count)

描述：

从参数 string 中抽取子字符串。

from 表示抽取的起始位置。

count 表示抽取的子字符串长度。

- from 为 0 时，按 1 处理。
- from 为正数时，抽取从 from 开始的 count 个字符。
- from 为负数时，抽取从倒数第 n 个开始的 count 个字符，n 为 from 的绝对值。
- count 小于 1 时，返回 null。

返回值类型：varchar

示例：

from 为正数时：

```
vastbase=# SELECT substr('ABCDEF',2,2);
 substr
--------
 BC
(1 row)
```

from 为负数时：

```
vastbase=# SELECT substr('ABCDEF',-3,2);
 substr
--------
 DE
(1 row)
```

- substrb(string,from)

  描述：该函数和 SUBSTR(string,from)函数功能一致，但是计算单位为字节。

  返回值类型：bytea

  示例：

```
vastbase=# SELECT substrb('ABCDEF',-2);
 substrb
---------
 EF
(1 row)
```

- substrb(string,from,count)

  描述：该函数和 SUBSTR(string,from,count)函数功能一致，但是计算单位为字节。

  返回值类型：bytea

  示例：

```
vastbase=# SELECT substrb('ABCDEF',2,2);
 substrb
---------
 BC
(1 row)
```

- trim([leading |trailing |both] [characters] from string)

  描述：从字符串 string 的开头、结尾或两边删除只包含 characters 中字符（缺省是一个空白）的最长的字符串。

  返回值类型：varchar

  示例：

```
vastbase=# SELECT trim(BOTH 'x' FROM 'xTomxx');
 btrim
-------
 Tom
(1 row)
vastbase=# SELECT trim(LEADING 'x' FROM 'xTomxx');
 ltrim
-------
 Tomxx
(1 row)
vastbase=# SELECT trim(TRAILING 'x' FROM 'xTomxx');
 rtrim
-------
 xTom
(1 row)
```

- rtrim(string [, characters])

描述：从字符串 string 的结尾删除只包含 characters 中字符（缺省是个空白）的最长的字符串。

返回值类型：varchar

示例：

```
vastbase=# SELECT rtrim('TRIMxxxx','x');
 rtrim
-------
 TRIM
(1 row)
```

- ltrim(string [, characters])

描述：从字符串 string 的开头删除只包含 characters 中字符（缺省是一个空白）的最长的字符串。

返回值类型：varchar

示例：

```
vastbase=# SELECT ltrim('xxxxTRIM','x');
 ltrim
-------
 TRIM
(1 row)
```

- upper(string)

描述：把字符串转化为大写。

返回值类型：varchar

示例：

```
vastbase=# SELECT upper('tom');
 upper
-------
 TOM
(1 row)
```

- lower(string)

描述：把字符串转化为小写。

返回值类型：varchar

示例：

```
vastbase=# SELECT lower('TOM');
 lower
-------
 tom
(1 row)
```

- rpad(string varchar, length int [, fill varchar])

描述：使用填充字符 fill（缺省时为空白），把 string 填充到 length 长度。如果 string 已经比 length 长则将其从尾部截断。

length 参数在 Vastbase 中表示字符长度。一个汉字长度计算为一个字符。

返回值类型：varchar

示例：

```
vastbase=# SELECT rpad('hi',5,'xyza');
 rpad
-------
 hixyz
(1 row)
vastbase=# SELECT rpad('hi',5,'abcdefg');
 rpad
-------
 hiabc
(1 row)
```

- instr(string,substring[,position,occurrence])

描述：从字符串 string 的 position（缺省时为 1）所指的位置开始查找并返回第 occurrence（缺省时为 1）次出现子串 substring 的位置的值。

  - 当 position 为 0 时，返回 0。
  - 当 position 为负数时，从字符串倒数第 n 个字符往前逆向搜索。n 为 position 的绝对值。

本函数以字符为计算单位，如一个汉字为一个字符。

返回值类型：integer

示例：

```
vastbase=# SELECT instr('corporate floor','or', 3);
 instr
-------
     5
(1 row)
vastbase=# SELECT instr('corporate floor','or',-3,2);
 instr
-------
     2
(1 row)
```

- initcap(string)

描述：将字符串中的每个单词的首字母转化为大写，其他字母转化为小写。

返回值类型：text

示例：

```
vastbase=# SELECT initcap('hi THOMAS');
  initcap
-----------
 Hi Thomas
(1 row)
```

- ascii(string)

描述：参数 string 的第一个字符的 ASCII 码。

返回值类型：integer

示例：

```
vastbase=# SELECT ascii('xyz');
 ascii
-------
   120
(1 row)
```

- replace(string varchar, search_string varchar, replacement_string varchar)

描述：把字符串 string 中所有子字符串 search_string 替换成子字符串 replacement_string。

返回值类型：varchar

示例：

```
vastbase=# SELECT replace('jack and jue','j','bl');
    replace
----------------
 black and blue
(1 row)
```

- lpad(string varchar, length int[, repeat_string varchar])

描述：在 string 的左侧添上一系列的 repeat_string（缺省为空白）来组成一个总长度为 n 的新字符串。

如果 string 本身的长度比指定的长度 length 长，则本函数将把 string 截断并把前面长度为 length 的字符串内容返回。

返回值类型：varchar

示例：

```
vastbase=# SELECT lpad('PAGE 1',15,'*.');
      lpad
-----------------
 *.*.*.*.*.*PAGE 1
(1 row)
vastbase=# SELECT lpad('hello world',5,'abcd');
 lpad
-------
 hello
(1 row)
```

- concat(str1,str2)

描述：将字符串 str1 和 str2 连接并返回。

返回值类型：varchar

示例：

```
vastbase=# SELECT concat('Hello', ' World!');
    concat
--------------
 Hello World!
(1 row)
```

```
vastbase=# SELECT concat('Hello', NULL);
 concat
--------
 Hello
(1 row)
```

- chr(integer)

  描述：给出 ASCII 码的字符。

  返回值类型：varchar

  示例：

```
vastbase=# SELECT chr(65);
 chr
-----
 A
(1 row)
```

- regexp_substr(source_char, pattern)

  描述：正则表达式的抽取子串函数。

  返回值类型：varchar

  示例：

```
vastbase=# SELECT regexp_substr('500 Hello World, Redwood Shores, CA', ',[^,]+,')
"REGEXPR_SUBSTR";
  REGEXPR_SUBSTR
--------------------
 , Redwood Shores,
(1 row)
```

- regexp_replace(string, pattern, replacement [,flags])

  描述：替换匹配 POSIX 正则表达式的子字符串。 如果没有匹配 pattern，那么返回不加修改的 string 串。 如果有匹配，则返回的 string 串里面的匹配子串将被 replacement 串替换掉。

  replacement 串可以包含\n， 其中\n 是 1 到 9， 表明 string 串里匹配模式里第 n 个圆括号子表达式的子串应该被插入， 并且它可以包含\&表示应该插入匹配整个模式的子串。

  可选的 flags 参数包含零个或多个改变函数行为的单字母标记。i 表示进行大小写无关的匹配，g 表示替换每一个匹配的子字符串而不仅仅是第一个。

  返回值类型：varchar

  示例：

```
vastbase=# SELECT regexp_replace('Thomas', '.[mN]a.', 'M');
 regexp_replace
----------------
 ThM
(1 row)
vastbase=# SELECT regexp_replace('foobarbaz','b(..)', E'X\\1Y', 'g') AS RESULT;
  result
-------------
```

```
 fooXarYXazY
(1 row)
```

- concat_ws(sep text, str"any" [, str"any" [, ...]])

  描述：以第一个参数为分隔符，链接第二个以后的所有参数。

  返回值类型：text

  示例：

```
vastbase=# SELECT concat_ws(',', 'ABCDE', 2, NULL, 22);
 concat_ws
------------
 ABCDE,2,22
(1 row)
```

- convert(string bytea, src_encoding name, dest_encoding name)

  描述：以 dest_encoding 指定的目标编码方式转化字符串 bytea。src_encoding 指定源编码方式，在该编码下，string 必须是合法的。

  返回值类型：bytea

  示例：

```
vastbase=# SELECT convert('text_in_utf8', 'UTF8', 'GBK');
        convert
----------------------------
 \x746578745f696e5f75746638
(1 row)
```

说明

如果源编码格式到目标编码格式的转化规则不存在，则字符串不进行任何转换直接返回，如 GBK 和 LATIN1 之间的转换规则是不存在的，具体转换规则可以通过查看系统表 pg_conversion 获得。

示例：

```
vastbase=# show server_encoding;
 server_encoding
-----------------
 LATIN1
(1 row)

vastbase=# SELECT convert_from('some text', 'GBK');
 convert_from
--------------
 some text
(1 row)

db_latin1=# SELECT convert_to('some text', 'GBK');
     convert_to
----------------------
 \x736f6d652074657874
(1 row)

db latin1=# SELECT convert('some text', 'GBK', 'LATIN1');
      convert
----------------------
```

```
 \x736f6d652074657874
(1 row)
```

- convert_from(string bytea, src_encoding name)

描述：以数据库的编码方式转化字符串 bytea。

src_encoding 指定源编码方式，在该编码下，string 必须是合法的。

返回值类型：text

示例：

```
vastbase=# SELECT convert_from('text_in_utf8', 'UTF8');
 convert_from
--------------
 text_in_utf8
(1 row)
```

- convert_to(string text, dest_encoding name)

描述：将字符串转化为 dest_encoding 的编码格式。

返回值类型：bytea

示例：

```
vastbase=# SELECT convert_to('some text', 'UTF8');
     convert_to
-----------------------
 \x736f6d652074657874
(1 row)
```

- string [NOT] LIKE pattern [ESCAPE escape-character]

描述：模式匹配函数。

如果 pattern 不包含百分号或者下划线，该模式只代表它本身，这时候 LIKE 的行为就像等号操作符。在 pattern 里的下划线（_）匹配任何单个字符；而一个百分号（%）匹配零或多个任何字符。

要匹配下划线或者百分号本身，在 pattern 里相应的字符必须前导逃逸字符。缺省的逃逸字符是反斜杠，但是用户可以用 ESCAPE 子句指定一个。要匹配逃逸字符本身，写两个逃逸字符。

返回值类型：Boolean

示例：

```
vastbase=# SELECT 'AA_BBCC' LIKE '%A@_B%' ESCAPE '@' AS RESULT;
 result
--------
 t
(1 row)
vastbase=# SELECT 'AA_BBCC' LIKE '%A@_B%' AS RESULT;
 result
--------
 f
(1 row)
```

```
vastbase=# SELECT 'AA@_BBCC' LIKE '%A@_B%' AS RESULT;
 result
--------
 t
(1 row)
```

❖ REGEXP_LIKE(source_string, pattern [, match_parameter])

描述：正则表达式的模式匹配函数。

source_string 为源字符串，pattern 为正则表达式匹配模式。match_parameter 为匹配选项，可取值为：

- 'i'：大小写不敏感。
- 'c'：大小写敏感。
- 'n'：允许正则表达式元字符“.”匹配换行符。
- 'm'：将 source_string 视为多行。

若忽略 match_parameter 选项，默认为大小写敏感，“.”不匹配换行符，source_string 视为单行。

返回值类型：Boolean

示例：

```
vastbase=# SELECT regexp_like('ABC', '[A-Z]');
 regexp_like
-------------
 t
(1 row)
vastbase=# SELECT regexp_like('ABC', '[D-Z]');
 regexp_like
-------------
 f
(1 row)
vastbase=# SELECT regexp_like('ABC', '[A-Z]','i');
 regexp_like
-------------
 t
(1 row)
vastbase=# SELECT regexp_like('ABC', '[A-Z]');
 regexp_like
-------------
 t
(1 row)
```

❖ format(formatstr text [, str"any" [, ...]])

描述：格式化字符串。

返回值类型：text

示例：

```
vastbase=# SELECT format('Hello %s, %1$s', 'World');
      format
--------------------
```

```
 Hello World, World
(1 row)
```

❖ md5(string)

描述：将 string 使用 MD5 加密，并以 16 进制数作为返回值。

说明

MD5 的安全性较低，不建议使用。

返回值类型：text

示例：

```
vastbase=# SELECT md5('ABC');
              md5
----------------------------------
 902fbdd2b1df0c4f70b4a5d23525e932
(1 row)
```

❖ decode(string text, format text)

描述：将二进制数据从文本数据中解码。

返回值类型：bytea

示例：

```
vastbase=# SELECT decode('MTIzAAE=', 'base64');
   decode
--------------
 \x3132330001
(1 row)
```

❖ encode(data bytea, format text)

描述：将二进制数据编码为文本数据。

返回值类型：text

示例：

```
vastbase=# SELECT encode(E'123\\000\\001', 'base64');
  encode
----------
 MTIzAAE=
(1 row)
```

说明

- 若字符串中存在换行符，如字符串由一个换行符和一个空格组成，在 Vastbase 中 LENGTH 和 LENGTHB 的值为 2。
- 对于 CHAR(n) 类型，Vastbase 中 n 是指字符个数。因此，对于多字节编码的字符集， LENGTHB 函数返回的长度可能大于 n。

11.5.4. 二进制字符串函数和操作符

字符串操作符

SQL 定义了一些字符串函数，在这些函数里使用关键字而不是逗号来分隔参数。

- octet_length(string)

  描述：二进制字符串中的字节数。

  返回值类型：int

  示例：

```
vastbase=# SELECT octet_length(E'jo\\000se'::bytea) AS RESULT;
 result
--------
      5
(1 row)
```

- overlay(string placing string from int [for int])

  描述：替换子串。

  返回值类型：bytea

  示例：

```
vastbase=# SELECT overlay(E'Th\\000omas'::bytea placing E'\\002\\003'::bytea from 2 for 3) AS
RESULT;
    result
----------------
 \x5402036d6173
(1 row)
```

- position(substring in string)

  描述：特定子字符串的位置。

  返回值类型：int

  示例：

```
vastbase=# SELECT position(E'\\000om'::bytea in E'Th\\000omas'::bytea) AS RESULT;
 result
--------
      3
(1 row)
```

- substring(string [from int] [for int])

  描述：截取子串。

  返回值类型：bytea

  示例：

```
vastbase=# SELECT substring(E'Th\\000omas'::bytea from 2 for 3) AS RESULT;
  result
----------
```

```
 \x68006f
(1 row)
```

- trim([both] bytes from string)

  描述：从 string 的开头和结尾删除只包含 bytes 中字节的最长字符串。

  返回值类型：bytea

  示例：

```
vastbase=# SELECT trim(E'\\000'::bytea from E'\\000Tom\\000'::bytea) AS RESULT;
 result
----------
 \x546f6d
(1 row)
```

二进制字符串函数

Vastbase 也提供了函数调用所使用的常用语法。

- btrim(string bytea,bytes bytea)

  描述：从 string 的开头和结尾删除只包含 bytes 中字节的最长的字符串。

  返回值类型：bytea

  示例：

```
vastbase=# SELECT btrim(E'\\000trim\\000'::bytea, E'\\000'::bytea) AS RESULT;
  result
------------
 \x7472696d
(1 row)
```

- get_bit(string, offset)

  描述：从字符串中抽取位。

  返回值类型：int

  示例：

```
vastbase=# SELECT get_bit(E'Th\\000omas'::bytea, 45) AS RESULT;
 result
--------
      1
(1 row)
```

- get_byte(string, offset)

  描述：从字符串中抽取字节。

  返回值类型：int

  示例：

```
vastbase=# SELECT get_byte(E'Th\\000omas'::bytea, 4) AS RESULT;
 result
--------
```

```
   109
(1 row)
```

- set_bit(string,offset, newvalue)

  描述：设置字符串中的位。

  返回值类型：bytea

  示例：

```
vastbase=# SELECT set_bit(E'Th\\000omas'::bytea, 45, 0) AS RESULT;
     result
------------------
 \x5468006f6d4173
(1 row)
```

- set_byte(string,offset, newvalue)

  描述：设置字符串中的字节。

  返回值类型：bytea

  示例：

```
vastbase=# SELECT set_byte(E'Th\\000omas'::bytea, 4, 64) AS RESULT;
     result
------------------
 \x5468006f406173
(1 row)
```

11.5.5. 位串函数和操作符

位串操作符

除了常用的比较操作符之外，还可以使用以下的操作符。&，|和#的位串操作数必须等长。在位移的时候，保留原始的位串长度（并以 0 填充）。

- ||

  描述：位串之间进行连接。

  示例：

```
vastbase=# SELECT B'10001' || B'011' AS RESULT;
  result
----------
 10001011
(1 row)
```

说明

单字段内部连续连接操作不能超过 180 次，若超过需拆分为多个连续连接的字符串，它们之间再做连接操作。

例如：str1||str2||str3||str4 拆分为（str1||str2)||(str3||str4)。

- &

  描述：位串之间进行“与”操作。

示例：

```
vastbase=# SELECT B'10001' & B'01101' AS RESULT;
 result
--------
 00001
(1 row)
```

- |

描述：位串之间进行“或”操作。

示例：

```
vastbase=# SELECT B'10001' | B'01101' AS RESULT;
 result
--------
 11101
(1 row)
```

- #

描述：位串之间如果不一致进行“或”操作。如果两个位串中对应位置都为 1 或者 0，则该位置返回为 0。

示例：

```
vastbase=# SELECT B'10001' # B'01101' AS RESULT;
 result
--------
 11100
(1 row)
```

- ~

描述：位串之间进行“非”操作。

示例：

```
vastbase=# SELECT ~B'10001'AS RESULT;
 result
----------
 01110
(1 row)
```

- <<

描述：位串进行左移操作。

示例：

```
vastbase=# SELECT B'10001' << 3 AS RESULT;
 result
----------
 01000
(1 row)
```

- >>

描述：位串进行右移操作。

示例：

```
vastbase=# SELECT B'10001' >> 2 AS RESULT;
 result
----------
 00100
(1 row)
```

下面的 SQL 标准函数除了可以用于字符串之外，也可以用于位串：length，bit_length，octet_length，position，substring，overlay。

下面的函数用于位串和二进制字符串：get_bit，set_bit。当用于位串时，这些函数位数从字符串的第一位（最左边）作为 0 位。

另外，可以在整数和 bit 之间来回转换。示例：

```
vastbase=# SELECT 44::bit(10) AS RESULT;
  result
------------
 0000101100
(1 row)

vastbase=# SELECT 44::bit(3) AS RESULT;
 result
--------
 100
(1 row)

vastbase=# SELECT cast(-44 as bit(12)) AS RESULT;
   result
--------------
 111111010100
(1 row)

vastbase=# SELECT '1110'::bit(4)::integer AS RESULT;
 result
--------
     14
(1 row)
```

说明

只是转换为“bit”的意思是转换成 bit(1)，因此只会转换成整数的最低位。

11.5.6. 模式匹配操作符

数据库提供了三种独立的实现模式匹配的方法：SQL LIKE 操作符、SIMILAR TO 操作符和 POSIX-风格的正则表达式。除了这些基本的操作符外，还有一些函数可用于提取或替换匹配子串并在匹配位置分离一个串。

- LIKE

  描述：判断字符串是否能匹配上 LIKE 后的模式字符串。如果字符串与提供的模式匹配，则 LIKE 表达式返回为真（NOT LIKE 表达式返回假），否则返回为假（NOT LIKE 表达式返回真）。

  匹配规则：

a. 此操作符只有在它的模式匹配整个串的时候才能成功。如果要匹配在串内任何位置的序列，该模式必须以百分号开头和结尾。

b. 下划线 （_）代表（匹配）任何单个字符； 百分号（%）代表任意串的通配符。

c. 要匹配文本里的下划线或者百分号，在提供的模式里相应字符必须前导逃逸字符。逃逸字符的作用是禁用元字符的特殊含义，缺省的逃逸字符是反斜线，也可以用 ESCAPE 子句指定一个不同的逃逸字符。

d. 要匹配逃逸字符本身，写两个逃逸字符。例如要写一个包含反斜线的模式常量，那你就要在 SQL 语句里写两个反斜线。

说明

参数 standard_conforming_strings 设置为 off 时，在文串常量中写的任何反斜线都需要被双写。因此，写一个匹配单个反斜线的模式实际上要在语句里写四个反斜线。（你可以通过用 ESCAPE 选择一个不同的逃逸字符来避免这种情况，这样反斜线就不再是 LIKE 的特殊字符了。但仍然是字符文本分析器的特殊字符，所以你还是需要两个反斜线。）我们也可以通过写 ESCAPE ''的方式不选择逃逸字符，这样可以有效地禁用逃逸机制，但是没有办法关闭下划线和百分号在模式中的特殊含义。

e. 关键字 ILIKE 可以用于替换 LIKE，区别是 LIKE 大小写敏感，ILIKE 大小写不敏感。

f. 操作符~~等效于 LIKE，操作符~~*等效于 ILIKE。

示例：

```
vastbase=# SELECT 'abc' LIKE 'abc' AS RESULT;
 result
-----------
 t
(1 row)
vastbase=# SELECT 'abc' LIKE 'a%' AS RESULT;
 result
-----------
 t
(1 row)
vastbase=# SELECT 'abc' LIKE '_b_' AS RESULT;
 result
-----------
 t
(1 row)
vastbase=# SELECT 'abc' LIKE 'c' AS RESULT;
 result
-----------
 f
(1 row)
```

- SIMILAR TO

描述：SIMILAR TO 操作符根据自己的模式是否匹配给定串而返回真或者假。他和 LIKE 非常类似，只不过他使用 SQL 标准定义的正则表达式理解模式。

匹配规则：

a. 和 LIKE 一样，此操作符只有在它的模式匹配整个串的时候才能成功。如果要匹配在串内任何位置的序列，该模式必须以百分号开头和结尾。

b. 下划线 （_）代表（匹配）任何单个字符； 百分号（%）代表任意串的通配符。

c. SIMILAR TO 也支持下面这些从 POSIX 正则表达式借用的模式匹配元字符。

元字符	含义
|	表示选择（两个候选之一）
*	表示重复前面的项零次或更多次
+	表示重复前面的项一次或更多次
?	表示重复前面的项零次或一次
{m}	表示重复前面的项刚好 m 次
{m,}	表示重复前面的项 m 次或更多次
{m,n}	表示重复前面的项至少 m 次并且不超过 n 次
()	把多个项组合成一个逻辑项
[...]	声明一个字符类，就像 POSIX 正则表达式一样

d. 前导逃逸字符可以禁止所有这些元字符的特殊含义。逃逸字符的使用规则和 LIKE 一样。

正则表达式函数：

支持使用函数 substring(string from pa...截取匹配 SQL 正则表达式的子字符串。

示例：

```
vastbase=# SELECT 'abc' SIMILAR TO 'abc' AS RESULT;
 result
-----------
 t
(1 row)
vastbase=# SELECT 'abc' SIMILAR TO 'a' AS RESULT;
 result
-----------
 f
(1 row)
vastbase=# SELECT 'abc' SIMILAR TO '%(b|d)%' AS RESULT;
 result
```

```
-----------
 t
(1 row)
vastbase=# SELECT 'abc' SIMILAR TO '(b|c)%'  AS RESULT;
 result
-----------
 f
(1 row)
```

❖ POSIX 正则表达式

描述：正则表达式是一个字符序列，它是定义一个串集合 （一个正则集）的缩写。 如果一个串是正则表达式描述的正则集中的一员时， 我们就说这个串匹配该正则表达式。 POSIX 正则表达式提供了比 LIKE 和 SIMILAR TO 操作符更强大的含义。表 11-26 列出了所有可用于 POSIX 正则表达式模式匹配的操作符。

表 11-26. 正则表达式匹配操作符

操作符	描述	例子
~	匹配正则表达式，大小写敏感	'thomas' ~ '.*thomas.*'
~*	匹配正则表达式，大小写不敏感	'thomas' ~* '.*Thomas.*'
！ ~	不匹配正则表达式，大小写敏感	'thomas' !~ '.*Thomas.*'
！ ~*	不匹配正则表达式，大小写不敏感	'thomas' !~* '.*vadim.*'

匹配规则：

a. 与 LIKE 不同，正则表达式允许匹配串里的任何位置，除非该正则表达式显式地挂接在串的开头或者结尾。

b. 除了上文提到的元字符外， POSIX 正则表达式还支持下列模式匹配元字符。

元字符	含义
^	表示串开头的匹配
$	表示串末尾的匹配
.	匹配任意单个字符

正则表达式函数：

POSIX 正则表达式支持下面函数。

– substring(string from pa...函数提供了抽取一个匹配 POSIX 正则表达式模式的子串的方法。

- regexp_replace(string, p...函数提供了将匹配 POSIX 正则表达式模式的子串替换为新文本的功能。
- regexp_matches(string te...函数返回一个文本数组，该数组由匹配一个 POSIX 正则表达式模式得到的所有被捕获子串构成。
- regexp_split_to_table(st...函数把一个 POSIX 正则表达式模式当作一个定界符来分离一个串。
- regexp_split_to_array(st...和 regexp_split_to_table 类似，是一个正则表达式分离函数，不过它的结果以一个 text 数组的形式返回。

说明

正则表达式分离函数会忽略零长度的匹配，这种匹配发生在串的开头或结尾或者正好发生在前一个匹配之后。这和正则表达式匹配的严格定义是相悖的，后者由 regexp_matches 实现，但是通常前者是实际中最常用的行为。

示例：

```
vastbase=#  SELECT 'abc' ~ 'Abc' AS RESULT;
result
--------
 f
(1 row)
vastbase=# SELECT 'abc' ~* 'Abc' AS RESULT;
 result
--------
 t
(1 row)
vastbase=# SELECT 'abc' !~ 'Abc' AS RESULT;
 result
--------
 t
(1 row)
vastbase=# SELECT 'abc'!~* 'Abc' AS RESULT;
 result
--------
 f
(1 row)
vastbase=# SELECT 'abc' ~ '^a' AS RESULT;
 result
--------
 t
(1 row)
vastbase=# SELECT 'abc' ~ '(b|d)'AS RESULT;
 result
--------
 t
(1 row)
vastbase=# SELECT 'abc' ~ '^(b|c)'AS RESULT;
 result
--------
 f
(1 row)
```

虽然大部分的正则表达式搜索都能很快地执行，但是正则表达式仍可能被人为地弄成需要任意长的时间和任意量的内存进行处理。不建议从非安全模式来源接受正则表达式搜索模式，如果必须这样做，建议加上语句超时限制。使用 SIMILAR TO 模式的搜索具有同样的安全性危险，因为 SIMILAR TO 提供了很多和 POSIX-风格正则表达式相同的能力。LIKE 搜索比其他两种选项简单得多，因此在接受非安全模式来源搜索时要更安全些。

11.5.7. 数字操作函数和操作符

数字操作符

- \+

  描述：加

  示例：

```
vastbase=# SELECT 2+3 AS RESULT;
 result
--------
      5
(1 row)
```

- \-

  描述：减

  示例：

```
vastbase=# SELECT 2-3 AS RESULT;
 result
--------
     -1
(1 row)
```

- *

  描述：乘

  示例：

```
vastbase=# SELECT 2*3 AS RESULT;
 result
--------
      6
(1 row)
```

- /

  描述：除（除法操作符不会取整）

  示例：

```
vastbase=# SELECT 4/2 AS RESULT;
 result
--------
      2
(1 row)
```

```
vastbase=# SELECT 4/3 AS RESULT;
     result
------------------
 1.33333333333333
(1 row)
```

- +/-

描述：正/负

示例：

```
vastbase=# SELECT -2 AS RESULT;
 result
--------
    -2
(1 row)
```

- %

描述：模（求余）

示例：

```
vastbase=# SELECT 5%4 AS RESULT;
 result
--------
     1
(1 row)
```

- @

描述：绝对值

示例：

```
vastbase=# SELECT @ -5.0 AS RESULT;
 result
--------
   5.0
(1 row)
```

- ^

描述：幂（指数运算）

示例：

```
vastbase=# SELECT 2.0^3.0 AS RESULT;
      result
--------------------
 8.0000000000000000
(1 row)
```

- |/

描述：平方根

示例：

```
vastbase=# SELECT |/ 25.0 AS RESULT;
 result
--------
```

```
     5
(1 row)
```

- ||/

  描述：立方根

  示例：

```
vastbase=# SELECT ||/ 27.0 AS RESULT;
 result
--------
     3
(1 row)
```

- !

  描述：阶乘

  示例：

```
vastbase=# SELECT 5! AS RESULT;
 result
--------
    120
(1 row)
```

- !!

  描述：阶乘（前缀操作符）

  示例：

```
vastbase=# SELECT !!5 AS RESULT;
 result
--------
    120
(1 row)
```

- &

  描述：二进制 AND

  示例：

```
vastbase=# SELECT 91&15  AS RESULT;
 result
--------
    11
(1 row)
```

- |

  描述：二进制 OR

  示例：

```
vastbase=# SELECT 32|3  AS RESULT;
 result
--------
    35
(1 row)
```

- #

  描述：二进制 XOR

  示例：

```
vastbase=# SELECT 17#5  AS RESULT;
 result
--------
    20
(1 row)
```

- ~

  描述：二进制 NOT

  示例：

```
vastbase=# SELECT ~1 AS RESULT;
 result
--------
    -2
(1 row)
```

- <<

  描述：二进制左移

  示例：

```
vastbase=# SELECT 1<<4 AS RESULT;
 result
--------
    16
(1 row)
```

- >>

  描述：二进制右移

  示例：

```
vastbase=# SELECT 8>>2 AS RESULT;
 result
--------
     2
(1 row)
```

数字操作函数

- abs(x)

  描述：绝对值。

  返回值类型：和输入相同。

  示例：

```
vastbase=# SELECT abs(-17.4);
 abs
------
```

```
 17.4
(1 row)
```

- acos(x)

  描述：反余弦。

  返回值类型：double precision

  示例：

```
vastbase=# SELECT acos(-1);
      acos
------------------
 3.14159265358979
(1 row)
```

- asin(x)

  描述：反正弦。

  返回值类型：double precision

  示例：

```
vastbase=# SELECT asin(0.5);
      asin
------------------
 .523598775598299
(1 row)
```

- atan(x)

  描述：反正切。

  返回值类型：double precision

  示例：

```
vastbase=# SELECT atan(1);
      atan
------------------
 .785398163397448
(1 row)
```

- atan2(y, x)

  描述：y/x 的反正切。

  返回值类型：double precision

  示例：

```
vastbase=# SELECT atan2(2, 1);
     atan2
------------------
 1.10714871779409
(1 row)
```

- bitand(integer, integer)

  描述：计算两个数字与运算(&)的结果。

返回值类型：bigint 类型数字。

示例：

```
vastbase=# SELECT bitand(127, 63);
 bitand
--------
     63
(1 row)
```

- cbrt(dp)

描述：立方根。

返回值类型：double precision

示例：

```
vastbase=# SELECT cbrt(27.0);
 cbrt
------
    3
(1 row)
```

- ceil(x)

描述：不小于参数的最小的整数。

返回值类型：整数。

示例：

```
vastbase=# SELECT ceil(-42.8);
 ceil
------
  -42
(1 row)
```

- ceiling(dp or numeric)

描述：不小于参数的最小整数（ceil 的别名）。

返回值类型：与输入相同。

示例：

```
vastbase=# SELECT ceiling(-95.3);
 ceiling
---------
     -95
(1 row)
```

- cos(x)

描述：余弦。

返回值类型：double precision

示例：

```
vastbase=# SELECT cos(-3.1415927);
       cos
```

```
-------------------
 -.999999999999999
(1 row)
```

- cot(x)

  描述：余切。

  返回值类型：double precision

  示例：

```
vastbase=# SELECT cot(1);
       cot
-------------------
 .642092615934331
(1 row)
```

- degrees(dp)

  描述：把弧度转为角度。

  返回值类型：double precision

  示例：

```
vastbase=# SELECT degrees(0.5);
     degrees
-------------------
 28.6478897565412
(1 row)
```

- div(y numeric, x numeric)

  描述：y 除以 x 的商的整数部分。

  返回值类型：numeric

  示例：

```
vastbase=# SELECT div(9,4);
 div
-----
   2
(1 row)
```

- exp(x)

  描述：自然指数。

  返回值类型：与输入相同。

  示例：

```
vastbase=# SELECT exp(1.0);
        exp
--------------------
 2.7182818284590452
(1 row)
```

- floor(x)

  描述：不大于参数的最大整数。

返回值类型：与输入相同。

示例：

```
vastbase=# SELECT floor(-42.8);
 floor
-------
   -43
(1 row)
```

- radians(dp)

描述：把角度转为弧度。

返回值类型：double precision

示例：

```
vastbase=# SELECT radians(45.0);
    radians
------------------
 .785398163397448
(1 row)
```

- random()

描述：0.0 到 1.0 之间的随机数。

返回值类型：double precision

示例：

```
vastbase=# SELECT random();
     random
------------------
 .824823560658842
(1 row)
```

- ln(x)

描述：自然对数。

返回值类型：与输入相同。

示例：

```
vastbase=# SELECT ln(2.0);
       ln
-------------------
 .6931471805599453
(1 row)
```

- log(x)

描述：以 10 为底的对数。

返回值类型：与输入相同。

示例：

```
vastbase=# SELECT log(100.0);
       log
```

```
--------------------
 2.0000000000000000
(1 row)
```

- log(b numeric, x numeric)

  描述：以 b 为底的对数。

  返回值类型：numeric

  示例：

```
vastbase=# SELECT log(2.0, 64.0);
       log
--------------------
 6.0000000000000000
(1 row)
```

- mod(x,y)

  描述：x/y 的余数（模）。如果 x 是 0，则返回 0。

  返回值类型：与参数类型相同。

  示例：

```
vastbase=# SELECT mod(9,4);
 mod
-----
   1
(1 row)
vastbase=# SELECT mod(0,9);
 mod
-----
   0
(1 row)
```

- pi()

  描述： “π”常量。

  返回值类型：double precision

  示例：

```
vastbase=# SELECT pi();
       pi
------------------
 3.14159265358979
(1 row)
```

- power(a double precision, b double precision)

  描述：a 的 b 次幂。

  返回值类型：double precision

  示例：

```
vastbase=# SELECT power(9.0, 3.0);
       power
----------------------
```

```
 729.0000000000000000
(1 row)
```

- round(x)

  描述：离输入参数最近的整数。

  返回值类型：与输入相同。

  示例：

```
vastbase=# SELECT round(42.4);
 round
-------
    42
(1 row)

vastbase=# SELECT round(42.6);
 round
-------
    43
(1 row)
```

- round(v numeric, s int)

  描述：保留小数点后 s 位，s 后一位进行四舍五入。

  返回值类型：numeric

  示例：

```
vastbase=# SELECT round(42.4382, 2);
 round
-------
 42.44
(1 row)
```

- setseed(dp)

  描述：为随后的 random()调用设置种子(-1.0 到 1.0 之间，包含)。

  返回值类型：void

  示例：

```
vastbase=# SELECT setseed(0.54823);
 setseed
---------

(1 row)
```

- sign(x)

  描述：输出此参数的符号。

  返回值类型：-1 表示负数，0 表示 0，1 表示正数。

  示例：

```
vastbase=# SELECT sign(-8.4);
 sign
------
```

```
  -1
(1 row)
```

- sin(x)

  描述：正弦。

  返回值类型：double precision

  示例：

```
vastbase=# SELECT sin(1.57079);
      sin
------------------
 .999999999979986
(1 row)
```

- sqrt(x)

  描述：平方根。

  返回值类型：与输入相同。

  示例：

```
vastbase=# SELECT sqrt(2.0);
      sqrt
-------------------
 1.414213562373095
(1 row)
```

- tan(x)

  描述：正切。

  返回值类型：double precision

  示例：

```
vastbase=# SELECT tan(20);
      tan
------------------
 2.23716094422474
(1 row)
```

- trunc(x)

  描述：截断（取整数部分）。

  返回值类型：与输入相同。

  示例：

```
vastbase=# SELECT trunc(42.8);
 trunc
-------
    42
(1 row)
```

- trunc(v numeric, s int)

  描述：截断为 s 位小数。

返回值类型：numeric

示例：

```
vastbase=# SELECT trunc(42.4382, 2);
 trunc
-------
 42.43
(1 row)
```

- width_bucket(op numeric, b1 numeric, b2 numeric, count int)

描述：返回一个桶，这个桶是在一个有 count 个桶，上界为 b1 下界为 b2 的等深柱图中 operand 将被赋予的那个桶。

返回值类型：int

示例：

```
vastbase=# SELECT width_bucket(5.35, 0.024, 10.06, 5);
 width_bucket
--------------
            3
(1 row)
```

- width_bucket(op dp, b1 dp, b2 dp, count int)

描述：返回一个桶，这个桶是在一个有 count 个桶，上界为 b1 下界为 b2 的等深柱图中 operand 将被赋予的那个桶。

返回值类型：int

示例：

```
vastbase=# SELECT width_bucket(5.35, 0.024, 10.06, 5);
 width_bucket
--------------
            3
(1 row)
```

11.5.8. 时间和日期处理函数和操作符

时间日期操作符

⚠ 警告

用户在使用时间和日期操作符时，对应的操作数请使用明确的类型前缀修饰，以确保数据库在解析操作数的时候能够与用户预期一致，不会产生用户非预期的结果。

比如下面示例没有明确数据类型就会出现异常错误。

```
SELECT date '2001-10-01' - '7' AS RESULT;
```

表 11-27. 时间和日期操作符

操作符	示例
+	vastbase=# SELECT date '2001-09-28' + integer '7' AS RESULT; result --------------------- 2001-10-05 00:00:00 (1 row)
	vastbase=# SELECT date '2001-09-28' + interval '1 hour' AS RESULT; result --------------------- 2001-09-28 01:00:00 (1 row)
	vastbase=# SELECT date '2001-09-28' + time '03:00' AS RESULT; result --------------------- 2001-09-28 03:00:00 (1 row)
	vastbase=# SELECT interval '1 day' + interval '1 hour' AS RESULT; result ---------------- 1 day 01:00:00 (1 row)
	vastbase=# SELECT timestamp '2001-09-28 01:00' + interval '23 hours' AS RESULT; result --------------------- 2001-09-29 00:00:00 (1 row)
	vastbase=# SELECT time '01:00' + interval '3 hours' AS RESULT; result ---------- 04:00:00 (1 row)
-	vastbase=# SELECT date '2001-10-01' - date '2001-09-28' AS RESULT; result -------- 3 days (1 row)
	vastbase=# SELECT date '2001-10-01' - integer '7' AS RESULT; result --------------------- 2001-09-24 00:00:00 (1 row)

操作符	示例
	vastbase=# SELECT date '2001-09-28' - interval '1 hour' AS RESULT; result --------------------- 2001-09-27 23:00:00 (1 row)
	vastbase=# SELECT time '05:00' - time '03:00' AS RESULT; result ---------- 02:00:00 (1 row)
	vastbase=# SELECT time '05:00' - interval '2 hours' AS RESULT; result ---------- 03:00:00 (1 row)
	vastbase=# SELECT timestamp '2001-09-28 23:00' - interval '23 hours' AS RESULT; result --------------------- 2001-09-28 00:00:00 (1 row)
	vastbase=# SELECT interval '1 day' - interval '1 hour' AS RESULT; result ---------- 23:00:00 (1 row)
	vastbase=# SELECT timestamp '2001-09-29 03:00' - timestamp '2001-09-27 12:00' AS RESULT; result ---------------- 1 day 15:00:00 (1 row)
*	vastbase=# SELECT 900 * interval '1 second' AS RESULT; result ---------- 00:15:00 (1 row)
	vastbase=# SELECT 21 * interval '1 day' AS RESULT; result --------- 21 days (1 row)
	vastbase=# SELECT double precision '3.5' * interval '1 hour' AS RESULT; result

操作符	示例
	---------- 03:30:00 (1 row)
/	vastbase=# SELECT interval '1 hour' / double precision '1.5' AS RESULT; result ---------- 00:40:00 (1 row)

时间/日期函数

- age(timestamp, timestamp)

  描述：将两个参数相减，并以年、月、日作为返回值。若相减值为负，则函数返回亦为负。

  返回值类型：interval

  示例：

```
vastbase=# SELECT age(timestamp '2001-04-10', timestamp '1957-06-13');
         age
-------------------------
 43 years 9 mons 27 days
(1 row)
```

- age(timestamp)

  描述：当前时间和参数相减。

  返回值类型：interval

  示例：

```
vastbase=# SELECT age(timestamp '1957-06-13');
         age
-------------------------
 60 years 2 mons 18 days
(1 row)
```

- clock_timestamp()

  描述：实时时钟的当前时间戳。

  返回值类型：timestamp with time zone

  示例：

```
vastbase=# SELECT clock_timestamp();
       clock_timestamp
-------------------------------
 2017-09-01 16:57:36.636205+08
(1 row)
```

- current_date

描述：当前日期。

返回值类型：date

示例：

```
vastbase=# SELECT current_date;
   date
------------
 2017-09-01
(1 row)
```

- current_time

描述：当前时间。

返回值类型：time with time zone

示例：

```
vastbase=# SELECT current_time;
     timetz
--------------------
 16:58:07.086215+08
(1 row)
```

- current_timestamp

描述：当前日期及时间。

返回值类型：timestamp with time zone

示例：

```
vastbase=# SELECT current_timestamp;
     pg_systimestamp
-------------------------------
 2017-09-01 16:58:19.22173+08
(1 row)
```

- date_part(text, timestamp)

描述：

获取小时的值。

等效于 extract(field from timestamp)。

返回值类型：double precision

示例：

```
vastbase=# SELECT date_part('hour', timestamp '2001-02-16 20:38:40');
 date_part
-----------
       20
(1 row)
```

- date_part(text, interval)

描述：获取月份的值。如果大于 12，则取与 12 的模。等效于 extract(field from timestamp)。

返回值类型：double precision

示例：

```
vastbase=# SELECT date_part('month', interval '2 years 3 months');
 date_part
-----------
         3
(1 row)
```

- date_trunc(text, timestamp)

描述：截取到参数 text 指定的精度。

返回值类型：timestamp

示例：

```
vastbase=# SELECT date_trunc('hour', timestamp  '2001-02-16 20:38:40');
     date_trunc
---------------------
 2001-02-16 20:00:00
(1 row)
```

- trunc(timestamp)

描述：默认按天截取。

示例：

```
vastbase=# SELECT trunc(timestamp  '2001-02-16 20:38:40');
trunc
---------------------
2001-02-16 00:00:00
(1 row)
```

- extract(field from timestamp)

描述：获取小时的值。

返回值类型：double precision

示例：

```
vastbase=# SELECT extract(hour from timestamp '2001-02-16 20:38:40');
 date_part
-----------
        20
(1 row)
```

- extract(field from interval)

描述：获取月份的值。如果大于 12，则取与 12 的模。

返回值类型：double precision

示例：

```
vastbase=# SELECT extract(month from interval '2 years 3 months');
 date_part
-----------
```

```
       3
(1 row)
```

- isfinite(date)

  描述：测试是否为有效日期。

  返回值类型：Boolean

  示例：

```
vastbase=# SELECT isfinite(date '2001-02-16');
 isfinite
----------
 t
(1 row)
```

- isfinite(timestamp)

  描述：测试判断是否为有效时间。

  返回值类型：Boolean

  示例：

```
vastbase=# SELECT isfinite(timestamp '2001-02-16 21:28:30');
 isfinite
----------
 t
(1 row)
```

- isfinite(interval)

  描述：测试是否为有效区间。

  返回值类型：Boolean

  示例：

```
vastbase=# SELECT isfinite(interval '4 hours');
 isfinite
----------
 t
(1 row)
```

- justify_days(interval)

  描述：将时间间隔以月（30 天为一月）为单位。

  返回值类型：interval

  示例：

```
vastbase=# SELECT justify_days(interval '35 days');
 justify_days
--------------
 1 mon 5 days
(1 row)
```

- justify_hours(interval)

  描述：将时间间隔以天（24 小时为一天）为单位。

返回值类型：interval

示例：

```
vastbase=# SELECT JUSTIFY_HOURS(INTERVAL '27 HOURS');
 justify_hours
----------------
 1 day 03:00:00
(1 row)
```

- justify_interval(interval)

描述：结合 justify_days 和 justify_hours，调整 interval。

返回值类型：interval

示例：

```
vastbase=# SELECT JUSTIFY_INTERVAL(INTERVAL '1 MON -1 HOUR');
 justify_interval
------------------
 29 days 23:00:00
(1 row)
```

- localtime

描述：当前时间。

返回值类型：time

示例：

```
vastbase=# SELECT localtime AS RESULT;
    result
----------------
 16:05:55.664681
(1 row)
```

- localtimestamp

描述：当前日期及时间。

返回值类型：timestamp

示例：

```
vastbase=# SELECT localtimestamp;
       timestamp
----------------------------
 2017-09-01 17:03:30.781902
(1 row)
```

- now()

描述：当前日期及时间。

返回值类型：timestamp with time zone

示例：

```
vastbase=# SELECT now();
           now
```

```
-------------------------------
 2017-09-01 17:03:42.549426+08
(1 row)
```

- numtodsinterval(num, interval_unit)

  描述：将数字转换为 interval 类型。num 为 numeric 类型数字，interval_unit 为固定格式字符串（'DAY' | 'HOUR' | 'MINUTE' | 'SECOND'）。

  可以通过设置参数 IntervalStyle 为 a，兼容该函数 interval 输出格式。

  示例：

```
vastbase=# SELECT numtodsinterval(100, 'HOUR');
 numtodsinterval
-----------------
 100:00:00
(1 row)

vastbase=# SET intervalstyle = a;
SET
vastbase=# SELECT numtodsinterval(100, 'HOUR');
      numtodsinterval
-------------------------------
 +000000004 04:00:00.000000000
(1 row)
```

- pg_sleep(seconds)

  描述：服务器线程延迟时间，单位为秒。

  返回值类型：void

  示例：

```
vastbase=# SELECT pg_sleep(10);
 pg_sleep
----------

(1 row)
```

- statement_timestamp()

  描述：当前日期及时间。

  返回值类型：timestamp with time zone

  示例：

```
vastbase=# SELECT statement_timestamp();
     statement_timestamp
-------------------------------
 2017-09-01 17:04:39.119267+08
(1 row)
```

- sysdate

  描述：当前日期及时间。

  返回值类型：timestamp

示例：

```
vastbase=# SELECT sysdate;
     sysdate
---------------------
 2017-09-01 17:04:49
(1 row)
```

- timeofday()

描述：当前日期及时间（像 clock_timestamp，但是返回时为 text）。

返回值类型：text

示例：

```
vastbase=# SELECT timeofday();
           timeofday
-------------------------------------
 Fri Sep 01 17:05:01.167506 2017 CST
(1 row)
```

- transaction_timestamp()

描述：当前日期及时间，与 current_timestamp 等效。

返回值类型：timestamp with time zone

示例：

```
vastbase=# SELECT transaction_timestamp();
    transaction_timestamp
-------------------------------
 2017-09-01 17:05:13.534454+08
(1 row)
```

- add_months(d,n)

描述：用于计算时间点 d 再加上 n 个月的时间。

返回值类型：timestamp

示例：

```
vastbase=# SELECT add_months(to_date('2017-5-29', 'yyyy-mm-dd'), 11) FROM dual;
    add_months
---------------------
 2018-04-29 00:00:00
(1 row)
```

- last_day(d)

描述：用于计算时间点 d 当月最后一天的时间。

返回值类型：timestamp

示例：

```
vastbase=# select last_day(to_date('2017-01-01', 'YYYY-MM-DD')) AS cal_result;
    cal_result
---------------------
```

```
 2017-01-31 00:00:00
(1 row)
```

- next_day(x,y)

  描述：用于计算时间点 x 开始的下一个星期几（y）的时间。

  返回值类型：timestamp

  示例：

```
vastbase=# select next_day(timestamp '2017-05-25 00:00:00','Sunday')AS cal_result;
    cal_result
---------------------
 2017-05-28 00:00:00
(1 row)
```

TIMESTAMPDIFF

TIMESTAMPDIFF(*unit , timestamp_expr1, timestamp_expr2*)

timestampdiff 函数是计算两个日期时间之间(timestamp_expr2-timestamp_expr1)的差值，并以 unit 形式返回结果。timestamp_expr1 , timestamp_expr2 必须是一个 timestamp、timestamptz、date 类型的值表达式。unit 表示的是两个日期差的单位。

说明

该函数仅在 Vastbase 兼容 MY 类型时（即 dbcompatibility = 'B'）有效，其他类型不支持该函数。

- year

  年份。

```
vastbase=#  SELECT TIMESTAMPDIFF(YEAR, '2018-01-01', '2020-01-01');
 timestamp_diff
----------------
              2
(1 row)
```

- quarter

  季度。

```
vastbase=#  SELECT TIMESTAMPDIFF(QUARTER, '2018-01-01', '2020-01-01');
 timestamp_diff
----------------
              8
(1 row)
```

- month

  月份。

```
vastbase=#  SELECT TIMESTAMPDIFF(MONTH, '2018-01-01', '2020-01-01');
 timestamp_diff
----------------
             24
(1 row)
```

- week

星期。

```
vastbase=#  SELECT TIMESTAMPDIFF(WEEK, '2018-01-01', '2020-01-01');
 timestamp_diff
----------------
          104
(1 row)
```

- day

  天。

```
vastbase=#  SELECT TIMESTAMPDIFF(DAY, '2018-01-01', '2020-01-01');
 timestamp_diff
----------------
          730
(1 row)
```

- hour

  小时。

```
vastbase=#  SELECT TIMESTAMPDIFF(HOUR, '2020-01-01 10:10:10', '2020-01-01 11:11:11');
 timestamp_diff
----------------
           1
(1 row)
```

- minute

  分钟。

```
vastbase=#  SELECT TIMESTAMPDIFF(MINUTE, '2020-01-01 10:10:10', '2020-01-01 11:11:11');
 timestamp_diff
----------------
          61
(1 row)
```

- second

  秒。

```
vastbase=#  SELECT TIMESTAMPDIFF(SECOND, '2020-01-01 10:10:10', '2020-01-01 11:11:11');
 timestamp_diff
----------------
         3661
(1 row)
```

- microseconds

  秒域（包括小数部分）乘以 1,000,000。

```
vastbase=#  SELECT TIMESTAMPDIFF(MICROSECOND, '2020-01-01 10:10:10.000000', '2020-01-01
10:10:10.111111');
 timestamp_diff
----------------
         111111
```

```
(1 row)
```

EXTRACT

EXTRACT(*field* FROM *source*)

extract 函数从日期或时间的数值里抽取子域，比如年、小时等。source 必须是一个 timestamp、time 或 interval 类型的值表达式（类型为 date 的表达式转换为 timestamp，因此也可以用）。field 是一个标识符或者字符串，它指定从源数据中抽取的域。extract 函数返回类型为 double precision 的数值。field 的取值范围如下所示。

- century

  世纪。

  第一个世纪从 0001-01-01 00:00:00 AD 开始。这个定义适用于所有使用阳历的国家。没有 0 世纪，直接从公元前 1 世纪到公元 1 世纪。

  示例：

```
vastbase=# SELECT EXTRACT(CENTURY FROM TIMESTAMP '2000-12-16 12:21:13');
 date_part
-----------
        20
(1 row)
```

- day
  - 如果 source 为 timestamp，表示月份里的日期（1-31）。

```
vastbase=# SELECT EXTRACT(DAY FROM TIMESTAMP '2001-02-16 20:38:40');
 date_part
-----------
        16
(1 row)
```

  - 如果 source 为 interval，表示天数。

```
vastbase=# SELECT EXTRACT(DAY FROM INTERVAL '40 days 1 minute');
 date_part
-----------
        40
(1 row)
```

- decade

  年份除以 10。

```
vastbase=# SELECT EXTRACT(DECADE FROM TIMESTAMP '2001-02-16 20:38:40');
 date_part
-----------
       200
(1 row)
```

- dow

  每周的星期几，星期天（0）到星期六（6）。

```
vastbase=# SELECT EXTRACT(DOW FROM TIMESTAMP '2001-02-16 20:38:40');
 date_part
```

```
-----------
          5
(1 row)
```

- doy

一年的第几天（1~365/366）。

```
vastbase=# SELECT EXTRACT(DOY FROM TIMESTAMP '2001-02-16 20:38:40');
 date_part
-----------
         47
(1 row)
```

- epoch
  - 如果 source 为 timestamp with time zone，表示自 1970-01-01 00:00:00-00 UTC 以来的秒数（结果可能是负数）；

    如果 source 为 date 和 timestamp，表示自 1970-01-01 00:00:00-00 当地时间以来的秒数；

    如果 source 为 interval，表示时间间隔的总秒数。

```
vastbase=# SELECT EXTRACT(EPOCH FROM TIMESTAMP WITH TIME ZONE '2001-02-16 20:38:40.12-08');
  date_part
--------------
 982384720.12
(1 row)
vastbase=# SELECT EXTRACT(EPOCH FROM INTERVAL '5 days 3 hours');
 date_part
-----------
    442800
(1 row)
```

  - 将 epoch 值转换为时间戳的方法。

```
vastbase=# SELECT TIMESTAMP WITH TIME ZONE 'epoch' + 982384720.12 * INTERVAL '1 second'
AS RESULT;
         result
---------------------------
 2001-02-17 12:38:40.12+08
(1 row)
```

- hour

小时域（0-23）。

```
vastbase=# SELECT EXTRACT(HOUR FROM TIMESTAMP '2001-02-16 20:38:40');
 date_part
-----------
         20
(1 row)
```

- isodow

一周的第几天（1-7）。

星期一为 1，星期天为 7。

说明

除了星期天外，都与 dow 相同。

```
vastbase=# SELECT EXTRACT(ISODOW FROM TIMESTAMP '2001-02-18 20:38:40');
 date_part
-----------
         7
(1 row)
```

- isoyear

  日期中的 ISO 8601 标准年（不适用于间隔）。

  每个带有星期一开始的周中包含 1 月 4 日的 ISO 年，所以在年初的 1 月或 12 月下旬的 ISO 年可能会不同于阳历的年。详细信息请参见后续的 week 描述。

```
vastbase=# SELECT EXTRACT(ISOYEAR FROM DATE '2006-01-01');
 date_part
-----------
      2005
(1 row)
vastbase=# SELECT EXTRACT(ISOYEAR FROM DATE '2006-01-02');
 date_part
-----------
      2006
(1 row)
```

- microseconds

  秒域（包括小数部分）乘以 1,000,000。

```
vastbase=# SELECT EXTRACT(MICROSECONDS FROM TIME '17:12:28.5');
 date_part
-----------
  28500000
(1 row)
```

- millennium

  千年。

  20 世纪（19xx 年）里面的年份在第二个千年里。第三个千年从 2001 年 1 月 1 日零时开始。

```
vastbase=# SELECT EXTRACT(MILLENNIUM FROM TIMESTAMP '2001-02-16 20:38:40');
 date_part
-----------
         3
(1 row)
```

- milliseconds

  秒域（包括小数部分）乘以 1000。请注意它包括完整的秒。

```
vastbase=# SELECT EXTRACT(MILLISECONDS FROM TIME '17:12:28.5');
 date_part
-----------
     28500
(1 row)
```

- minute

分钟域（0-59）。

```
vastbase=# SELECT EXTRACT(MINUTE FROM TIMESTAMP '2001-02-16 20:38:40');
 date_part
-----------
        38
(1 row)
```

- month

如果 source 为 timestamp，表示一年里的月份数（1-12）。

```
vastbase=# SELECT EXTRACT(MONTH FROM TIMESTAMP '2001-02-16 20:38:40');
 date_part
-----------
         2
(1 row)
```

如果 source 为 interval，表示月的数目，然后对 12 取模（0-11）。

```
vastbase=# SELECT EXTRACT(MONTH FROM INTERVAL '2 years 13 months');
 date_part
-----------
         1
(1 row)
```

- quarter

该天所在的该年的季度（1-4）。

```
vastbase=# SELECT EXTRACT(QUARTER FROM TIMESTAMP '2001-02-16 20:38:40');
 date_part
-----------
         1
(1 row)
```

- second

秒域，包括小数部分（0-59）。

```
vastbase=# SELECT EXTRACT(SECOND FROM TIME '17:12:28.5');
 date_part
-----------
      28.5
(1 row)
```

- timezone

与 UTC 的时区偏移量，单位为秒。正数对应 UTC 东边的时区，负数对应 UTC 西边的时区。

- timezone_hour

时区偏移量的小时部分。

- timezone_minute

时区偏移量的分钟部分。

- week

该天在所在的年份里是第几周。ISO 8601 定义一年的第一周包含该年的一月四日（ISO-8601 的周从星期一开始）。换句话说，一年的第一个星期四在第一周。

在 ISO 定义里，一月的头几天可能是前一年的第 52 或者第 53 周，十二月的后几天可能是下一年第一周。比如，2005-01-01 是 2004 年的第 53 周，而 2006-01-01 是 2005 年的第 52 周，2012-12-31 是 2013 年的第一周。建议 isoyear 字段和 week 一起使用以得到一致的结果。

```
vastbase=# SELECT EXTRACT(WEEK FROM TIMESTAMP '2001-02-16 20:38:40');
 date_part
-----------
         7
(1 row)
```

- year

  年份域。

```
vastbase=# SELECT EXTRACT(YEAR FROM TIMESTAMP '2001-02-16 20:38:40');
 date_part
-----------
      2001
(1 row)
```

date_part

date_part 函数是在传统的 Ingres 函数的基础上制作的（该函数等效于 SQL 标准函数 extract）：

date_part('*field*', *source*)

这里的 field 参数必须是一个字符串，而不是一个名称。有效的 field 与 extract 一样，详细信息请参见 EXTRACT。

示例：

```
vastbase=# SELECT date_part('day', TIMESTAMP '2001-02-16 20:38:40');
 date_part
-----------
        16
(1 row)
vastbase=# SELECT date_part('hour', INTERVAL '4 hours 3 minutes');
 date_part
-----------
         4
(1 row)
```

表 11-28 显示了可以用于格式化日期和时间值的模式。

表 11-28. 用于日期/时间格式化的模式

类别	模式	描述
小时	HH	一天的小时数（01-12）
	HH12	一天的小时数（01-12）
	HH24	一天的小时数（00-23）
分钟	MI	分钟（00-59）

类别	模式	描述
秒	SS	秒（00-59）
	FF	微秒（000000-999999）
	SSSSS	午夜后的秒（0-86399）
上、下午	AM 或 A.M.	上午标识
	PM 或 P.M.	下午标识
年	Y,YYY	带逗号的年（4 和更多位）
	SYYYY	公元前四位年
	YYYY	年（4 和更多位）
	YYY	年的后三位
	YY	年的后两位
	Y	年的最后一位
	IYYY	ISO 年（4 位或更多位）
	IYY	ISO 年的最后三位
	IY	ISO 年的最后两位
	I	ISO 年的最后一位
	RR	年的后两位（可在 21 世纪存储 20 世纪的年份）
	RRRR	可接收 4 位年或两位年。若是两位，则和 RR 的返回值相同，若是四位，则和 YYYY 相同。
	• BC 或 B.C. • AD 或 A.D.	纪元标识。BC（公元前），AD（公元后）。
月	MONTH	全长大写月份名（空白填充为 9 字符）
	MON	大写缩写月份名（3 字符）
	MM	月份数（01-12）
	RM	罗马数字的月份（I-XII ； I=JAN）（大写）

类别	模式	描述
天	DAY	全长大写日期名（空白填充为 9 字符）
	DY	缩写大写日期名（3 字符）
	DDD	一年里的日（001-366）
	DD	一个月里的日（01-31）
	D	一周里的日（1-7 ； 周日是 1）
周	W	一个月里的周数（1-5）（第一周从该月第一天开始）
	WW	一年里的周数（1-53）（第一周从该年的第一天开始）
	IW	ISO 一年里的周数（第一个星期四在第一周里）
世纪	CC	世纪（2 位）（21 世纪从 2001-01-01 开始）
儒略日	J	儒略日（自公元前 4712 年 1 月 1 日来的天数）
季度	Q	季度

说明

上表中 RR 计算年的规则如下：

- 输入的两位年份在 00~49 之间：

  当前年份的后两位在 00~49 之间，返回值年份的前两位和当前年份的前两位相同；

  当前年份的后两位在 50~99 之间，返回值年份的前两位是当前年份的前两位加 1。

- 输入的两位年份在 50~99 之间：

  当前年份的后两位在 00~49 之间，返回值年份的前两位是当前年份的前两位减 1；

  当前年份的后两位在 50~99 之间，返回值年份的前两位和当前年份的前两位相同。

11.5.9. 类型转换函数

类型转换函数

- cast(x as y)

  描述：类型转换函数，将 x 转换成 y 指定的类型。

  示例：

```
vastbase=# SELECT cast('22-oct-1997' as timestamp);
     timestamp
---------------------
```

```
 1997-10-22 00:00:00
(1 row)
```

- hextoraw(string)

  描述：将一个十六进制构成的字符串转换为二进制。

  返回值类型：raw

  示例：

```
vastbase=# SELECT hextoraw('7D');
 hextoraw
----------
 7D
(1 row)
```

- numtoday(numeric)

  描述：将数字类型的值转换为指定格式的时间戳。

  返回值类型：timestamp

  示例：

```
vastbase=# SELECT numtoday(2);
 numtoday
----------
 2 days
(1 row)
```

- pg_systimestamp()

  描述：获取系统时间戳。

  返回值类型：timestamp with time zone

  示例：

```
vastbase=# SELECT pg_systimestamp();
      pg_systimestamp
-------------------------------
 2015-10-14 11:21:28.317367+08
(1 row)
```

- rawtohex(string)

  描述：将一个二进制构成的字符串转换为十六进制的字符串。

  结果为输入字符的 ACSII 码，以十六进制表示。

  返回值类型：varchar

  示例：

```
vastbase=# SELECT rawtohex('1234567');
   rawtohex
----------------
 31323334353637
(1 row)
```

- to_char (datetime/interval [, fmt])

描述：将一个 DATE、TIMESTAMP、TIMESTAMP WITH TIME ZONE 或者 TIMESTAMP WITH LOCAL TIME ZONE 类型的 DATETIME 或者 INTERVAL 值按照 fmt 指定的格式转换为 VARCHAR 类型。

- 可选参数 fmt 可以为以下几类：日期、时间、星期、季度和世纪。每类都可以有不同的模板，模板之间可以合理组合，常见的模板有：HH、MM、SS、YYYY、MM、DD。
- 模板可以有修饰词，常用的修饰词是 FM，可以用来抑制前导的零或尾随的空白。

返回值类型：varchar

示例：

```
vastbase=# SELECT to_char(current_timestamp,'HH12:MI:SS');
 to_char
----------
 10:19:26
(1 row)
vastbase=# SELECT to_char(current_timestamp,'FMHH12:FMMI:FMSS');
 to_char
----------
 10:19:46
(1 row)
```

- to_char(double precision, text)

描述：将双精度类型的值转换为指定格式的字符串。

返回值类型：text

示例：

```
vastbase=# SELECT to_char(125.8::real, '999D99');
 to_char
---------
  125.80
(1 row)
```

- to_char (integer/number[, fmt])

描述：将一个整型或者浮点类型的值转换为指定格式的字符串。

- 可选参数 fmt 可以为以下几类：十进制字符、“分组”符、正负号和货币符号，每类都可以有不同的模板，模板之间可以合理组合，常见的模板有：9、0、，（千分隔符）、.（小数点）。
- 模板可以有类似 FM 的修饰词，但 FM 不抑制由模板 0 指定而输出的 0。
- 要将整型类型的值转换成对应 16 进制值的字符串，使用模板 X 或 x。

返回值类型：varchar

示例：

```
vastbase=# SELECT to_char(1485,'9,999');
 to_char
---------
  1,485
(1 row)
```

```
vastbase=# SELECT to_char( 1148.5,'9,999.999');
  to_char
------------
  1,148.500
(1 row)
vastbase=# SELECT to_char(148.5,'990999.909');
   to_char
-------------
    0148.500
(1 row)
vastbase=# SELECT to_char(123,'XXX');
 to_char
---------
   7B
(1 row)
```

- to_char(interval, text)

  描述：将时间间隔类型的值转换为指定格式的字符串。

  返回值类型：text

  示例：

```
vastbase=# SELECT to_char(interval '15h 2m 12s', 'HH24:MI:SS');
 to_char
----------
 15:02:12
(1 row)
```

- to_char(int, text)

  描述：将整数类型的值转换为指定格式的字符串。

  返回值类型：text

  示例：

```
vastbase=# SELECT to_char(125, '999');
 to_char
---------
  125
(1 row)
```

- to_char(numeric, text)

  描述：将数字类型的值转换为指定格式的字符串。

  返回值类型：text

  示例：

```
vastbase=# SELECT to_char(-125.8, '999D99S');
 to_char
---------
 125.80-
(1 row)
```

- to_char (string)

  描述：将 CHAR、VARCHAR、VARCHAR2、CLOB 类型转换为 VARCHAR 类型。

如使用该函数对 CLOB 类型进行转换，且待转换 CLOB 类型的值超出目标类型的范围，则返回错误。

返回值类型：varchar

示例：

```
vastbase=# SELECT to_char('01110');
 to_char
---------
 01110
(1 row)
```

- to_char(timestamp, text)

描述：将时间戳类型的值转换为指定格式的字符串。

返回值类型：text

示例：

```
vastbase=# SELECT to_char(current_timestamp, 'HH12:MI:SS');
 to_char
----------
 10:55:59
(1 row)
```

- to_clob(char/nchar/varchar/nvarchar/varchar2/nvarchar2/text/raw)

描述：将 RAW 类型或者文本字符集类型 CHAR、NCHAR、VARCHAR、VARCHAR2、NVARCHAR2、TEXT 转成 CLOB 类型。

返回值类型：clob

示例：

```
vastbase=# SELECT to_clob('ABCDEF'::RAW(10));
 to_clob
---------
 ABCDEF
(1 row)
vastbase=# SELECT to_clob('hello111'::CHAR(15));
 to_clob
----------
 hello111
(1 row)
vastbase=# SELECT to_clob('gauss123'::NCHAR(10));
 to_clob
----------
 gauss123
(1 row)
vastbase=# SELECT to_clob('gauss234'::VARCHAR(10));
 to_clob
----------
 gauss234
(1 row)
vastbase=# SELECT to_clob('gauss345'::VARCHAR2(10));
 to_clob
```

```
----------
 gauss345
(1 row)
vastbase=# SELECT to_clob('gauss456'::NVARCHAR2(10));
 to_clob
----------
 gauss456
(1 row)
vastbase=# SELECT to_clob('World222!'::TEXT);
  to_clob
-----------
 World222!
(1 row)
```

- to_date(text)

  描述：将文本类型的值转换为指定格式的时间戳。

  返回值类型：timestamp

  示例：

```
vastbase=# SELECT to_date('2015-08-14');
      to_date
---------------------
 2015-08-14 00:00:00
(1 row)
```

- to_date(text, text)

  描述：将字符串类型的值转换为指定格式的日期。

  返回值类型：timestamp

  示例：

```
vastbase=# SELECT to_date('05 Dec 2000', 'DD Mon YYYY');
      to_date
---------------------
 2000-12-05 00:00:00
(1 row)
```

- to_date(string, fmt)

  描述：

  将字符串 string 按 fmt 指定格式转化为 DATE 类型的值。

  该函数不能直接支持 CLOB 类型，但是 CLOB 类型的参数能够通过隐式转换实现。

  返回值类型：date

  示例：

```
vastbase=# SELECT TO_DATE('05 Dec 2010','DD Mon YYYY');
      to_date
---------------------
 2010-12-05 00:00:00
(1 row)
```

- to_number (expr [, fmt])

描述：将 expr 按指定格式转换为一个 NUMBER 类型的值。

类型转换格式请参考表 11-29。

转换十六进制字符串为十进制数字时，最多支持 16 个字节的十六进制字符串转换为无符号数。

转换十六进制字符串为十进制数字时，格式字符串中不允许出现除'x'或'X'以外的其他字符，否则报错。

返回值类型：number

示例：

```
vastbase=# SELECT to_number('12,454.8-', '99G999D9S');
 to_number
-----------
  -12454.8
(1 row)
```

- to_number(text, text)

描述：将字符串类型的值转换为指定格式的数字。

返回值类型：numeric

示例：

```
vastbase=# SELECT to_number('12,454.8-', '99G999D9S');
 to_number
-----------
  -12454.8
(1 row)
```

- to_timestamp(double precision)

描述：把 UNIX 纪元转换成时间戳。

返回值类型：timestamp with time zone

示例：

```
vastbase=# SELECT to_timestamp(1284352323);
     to_timestamp
------------------------
 2010-09-13 12:32:03+08
(1 row)
```

- to_timestamp(string [,fmt])

描述：将字符串 string 按 fmt 指定的格式转换成时间戳类型的值。不指定 fmt 时，按参数 nls_timestamp_format 所指定的格式转换。

Vastbase 的 to_timestamp 中，

- 如果输入的年份 YYYY=0，系统报错。
- 如果输入的年份 YYYY<0，在 fmt 中指定 SYYYY，则正确输出公元前绝对值 n 的年份。

fmt 中出现的字符必须与日期/时间格式化的模式相匹配，否则报错。

返回值类型：timestamp without time zone

示例：

```
vastbase=# SHOW nls_timestamp_format;
   nls_timestamp_format
----------------------------
 DD-Mon-YYYY HH:MI:SS.FF AM
(1 row)

vastbase=# SELECT to_timestamp('12-sep-2014');
   to_timestamp
---------------------
 2014-09-12 00:00:00
(1 row)
vastbase=# SELECT to_timestamp('12-Sep-10 14:10:10.123000','DD-Mon-YY HH24:MI:SS.FF');
     to_timestamp
-------------------------
 2010-09-12 14:10:10.123
(1 row)
vastbase=# SELECT to_timestamp('-1','SYYYY');
     to_timestamp
------------------------
 0001-01-01 00:00:00 BC
(1 row)
vastbase=# SELECT to_timestamp('98','RR');
   to_timestamp
---------------------
 1998-01-01 00:00:00
(1 row)
vastbase=# SELECT to_timestamp('01','RR');
   to_timestamp
---------------------
 2001-01-01 00:00:00
(1 row)
```

- to_timestamp(text, text)

描述：将字符串类型的值转换为指定格式的时间戳。

返回值类型：timestamp

示例：

```
vastbase=# SELECT to_timestamp('05 Dec 2000', 'DD Mon YYYY');
   to_timestamp
---------------------
 2000-12-05 00:00:00
(1 row)
```

表 11-29. 数值格式化的模版模式

模式	描述
9	带有指定数值位数的值
0	带前导零的值

模式	描述
.（句点）	小数点
,（逗号）	分组（千）分隔符
PR	尖括号内负值
S	带符号的数值（使用区域设置）
L	货币符号（使用区域设置）
D	小数点（使用区域设置）
G	分组分隔符（使用区域设置）
MI	在指明的位置的负号（如果数字 < 0）
PL	在指明的位置的正号（如果数字 > 0）
SG	在指明的位置的正/负号
RN	罗马数字（输入在 1 和 3999 之间）
TH 或 th	序数后缀
V	移动指定位（小数）

11.5.10.几何函数和操作符

几何操作符

- +

  描述：平移。

  示例：

```
vastbase=# SELECT box '((0,0),(1,1))' + point '(2.0,0)' AS RESULT;
  result
-------------
 (3,1),(2,0)
(1 row)
```

- -

  描述：平移。

  示例：

```
vastbase=# SELECT box '((0,0),(1,1))' - point '(2.0,0)' AS RESULT;
   result
---------------
 (-1,1),(-2,0)
(1 row)
```

- *

描述：伸展/旋转。

示例：

```
vastbase=# SELECT box '((0,0),(1,1))' * point '(2.0,0)' AS RESULT;
  result
-------------
 (2,2),(0,0)
(1 row)
```

- /

描述：收缩/旋转。

示例：

```
vastbase=# SELECT box '((0,0),(2,2))' / point '(2.0,0)' AS RESULT;
  result
-------------
 (1,1),(0,0)
(1 row)
```

- #

描述：两个图形交面。

示例：

```
vastbase=# SELECT box '((1,-1),(-1,1))' # box '((1,1),(-2,-2))' AS RESULT;
 result
---------------
 (1,1),(-1,-1)
(1 row)
```

- #

描述：图形的路径数目或多边形顶点数。

示例：

```
vastbase=# SELECT # path'((1,0),(0,1),(-1,0))' AS RESULT;
 result
--------
      3
(1 row)
```

- @-@

描述：图形的长度或者周长。

示例：

```
vastbase=# SELECT @-@ path '((0,0),(1,0))' AS RESULT;
 result
--------
```

```
     2
(1 row)
```

- @@

描述：图形的中心。

示例：

```
vastbase=# SELECT @@ circle '((0,0),10)' AS RESULT;
 result
--------
 (0,0)
(1 row)
```

- <->

描述：两个图形之间的距离。

示例：

```
vastbase=# SELECT circle '((0,0),1)' <-> circle '((5,0),1)' AS RESULT;
 result
--------
     3
(1 row)
```

- &&

描述：两个图形是否重叠（有一个共同点就为真）。

示例：

```
vastbase=# SELECT box '((0,0),(1,1))' && box '((0,0),(2,2))' AS RESULT;
 result
--------
 t
(1 row)
```

- <<

描述：图形是否全部在另一个图形的左边（没有相同的横坐标）。

示例：

```
vastbase=# SELECT circle '((0,0),1)' << circle '((5,0),1)' AS RESULT;
 result
--------
 t
(1 row)
```

- >>

描述：图形是否全部在另一个图形的右边（没有相同的横坐标）。

示例：

```
vastbase=# SELECT circle '((5,0),1)' >> circle '((0,0),1)' AS RESULT;
 result
--------
 t
(1 row)
```

- &<

  描述：图形的最右边是否不超过在另一个图形的最右边。

  示例：

```
vastbase=# SELECT box '((0,0),(1,1))' &< box '((0,0),(2,2))' AS RESULT;
 result
--------
 t
(1 row)
```

- &>

  描述：图形的最左边是否不超过在另一个图形的最左边。

  示例：

```
vastbase=# SELECT box '((0,0),(3,3))' &> box '((0,0),(2,2))' AS RESULT;
 result
--------
 t
(1 row)
```

- <<|

  描述：图形是否全部在另一个图形的下边（没有相同的纵坐标）。

  示例：

```
vastbase=# SELECT box '((0,0),(3,3))' <<| box '((3,4),(5,5))' AS RESULT;
 result
--------
 t
(1 row)
```

- |>>

  描述：图形是否全部在另一个图形的上边（没有相同的纵坐标）。

  示例：

```
vastbase=# SELECT box '((3,4),(5,5))' |>> box '((0,0),(3,3))' AS RESULT;
 result
--------
 t
(1 row)
```

- &<|

  描述：图形的最上边是否不超过另一个图形的最上边。

  示例：

```
vastbase=# SELECT box '((0,0),(1,1))' &<| box '((0,0),(2,2))' AS RESULT;
 result
--------
 t
(1 row)
```

- |&>

  描述：图形的最下边是否不超过另一个图形的最下边。

示例：

```
vastbase=# SELECT box '((0,0),(3,3))' |&> box '((0,0),(2,2))' AS RESULT;
 result
--------
 t
(1 row)
```

- <^

描述：图形是否低于另一个图形（允许两个图形有接触）。

示例：

```
vastbase=# SELECT box '((0,0),(-3,-3))' <^ box '((0,0),(2,2))' AS RESULT;
 result
--------
 t
(1 row)
```

- >^

描述：图形是否高于另一个图形（允许两个图形有接触）。

示例：

```
vastbase=# SELECT box '((0,0),(2,2))' >^ box '((0,0),(-3,-3))'  AS RESULT;
 result
--------
 t
(1 row)
```

- ?#

描述：两个图形是否相交。

示例：

```
vastbase=# SELECT lseg '((-1,0),(1,0))' ?# box '((-2,-2),(2,2))' AS RESULT;
 result
--------
 t
(1 row)
```

- ?-

描述：图形是否处于水平位置。

示例：

```
vastbase=# SELECT ?- lseg '((-1,0),(1,0))' AS RESULT;
 result
--------
 t
(1 row)
```

- ?-

描述：图形是否水平对齐。

示例：

```
vastbase=# SELECT point '(1,0)' ?- point '(0,0)' AS RESULT;
 result
--------
 t
(1 row)
```

- ?|

  描述：图形是否处于竖直位置。

  示例：

```
vastbase=# SELECT ?| lseg '((-1,0),(1,0))' AS RESULT;
 result
--------
 f
(1 row)
```

- ?|

  描述：图形是否竖直对齐。

  示例：

```
vastbase=# SELECT point '(0,1)' ?| point '(0,0)' AS RESULT;
 result
--------
 t
(1 row)
```

- ?-|

  描述：两条线是否垂直。

  示例：

```
vastbase=# SELECT lseg '((0,0),(0,1))' ?-| lseg '((0,0),(1,0))' AS RESULT;
 result
--------
 t
(1 row)
```

- ?||

  描述：两条线是否平行。

  示例：

```
vastbase=# SELECT lseg '((-1,0),(1,0))' ?|| lseg '((-1,2),(1,2))' AS RESULT;
 result
--------
 t
(1 row)
```

- @>

  描述：图形是否包含另一个图形。

  示例：

```
vastbase=# SELECT circle '((0,0),2)' @> point '(1,1)' AS RESULT;
 result
--------
```

```
 t
(1 row)
```

- <@

  描述：图形是否被包含于另一个图形。

  示例：

```
vastbase=# SELECT point '(1,1)' <@ circle '((0,0),2)' AS RESULT;
 result
--------
 t
(1 row)
```

- ~=

  描述：两个图形是否相同?

  示例：

```
vastbase=# SELECT polygon '((0,0),(1,1))' ~= polygon '((1,1),(0,0))' AS RESULT;
 result
--------
 t
(1 row)
```

几何函数

- area(object)

  描述：计算图形的面积。

  返回类型：double precision

  示例：

```
vastbase=# SELECT area(box '((0,0),(1,1))') AS RESULT;
 result
--------
      1
(1 row)
```

- center(object)

  描述：计算图形的中心。

  返回类型：point

  示例：

```
vastbase=# SELECT center(box '((0,0),(1,2))') AS RESULT;
 result
---------
 (0.5,1)
(1 row)
```

- diameter(circle)

  描述：计算圆的直径。

返回类型：double precision

示例：

```
vastbase=# SELECT diameter(circle '((0,0),2.0)') AS RESULT;
 result
--------
      4
(1 row)
```

- height(box)

描述：矩形的竖直高度。

返回类型：double precision

示例：

```
vastbase=# SELECT height(box '((0,0),(1,1))') AS RESULT;
 result
--------
      1
(1 row)
```

- isclosed(path)

描述：图形是否为闭合路径。

返回类型：Boolean

示例：

```
vastbase=# SELECT isclosed(path '((0,0),(1,1),(2,0))') AS RESULT;
 result
--------
 t
(1 row)
```

- isopen(path)

描述：图形是否为开放路径。

返回类型：Boolean

示例：

```
vastbase=# SELECT isopen(path '[(0,0),(1,1),(2,0)]') AS RESULT;
 result
--------
 t
(1 row)
```

- length(object)

描述：计算图形的长度。

返回类型：double precision

示例：

```
vastbase=# SELECT length(path '((-1,0),(1,0))') AS RESULT;
 result
```

```
--------
     4
(1 row)
```

- npoints(path)

  描述：计算路径的顶点数。

  返回类型：int

  示例：

```
vastbase=# SELECT npoints(path '[(0,0),(1,1),(2,0)]') AS RESULT;
 result
--------
     3
(1 row)
```

- npoints(polygon)

  描述：计算多边形的顶点数。

  返回类型：int

  示例：

```
vastbase=# SELECT npoints(polygon '((1,1),(0,0))') AS RESULT;
 result
--------
     2
(1 row)
```

- pclose(path)

  描述：把路径转换为闭合路径。

  返回类型：path

  示例：

```
vastbase=# SELECT pclose(path '[(0,0),(1,1),(2,0)]') AS RESULT;
      result
--------------------
 ((0,0),(1,1),(2,0))
(1 row)
```

- popen(path)

  描述：把路径转换为开放路径。

  返回类型：path

  示例：

```
vastbase=# SELECT popen(path '((0,0),(1,1),(2,0))') AS RESULT;
      result
--------------------
 [(0,0),(1,1),(2,0)]
(1 row)
```

- radius(circle)

  描述：计算圆的半径。

返回类型：double precision

示例：

```
vastbase=# SELECT radius(circle '((0,0),2.0)') AS RESULT;
 result
--------
      2
(1 row)
```

- width(box)

描述：计算矩形的水平尺寸。

返回类型：double precision

示例：

```
vastbase=# SELECT width(box '((0,0),(1,1))') AS RESULT;
 result
--------
      1
(1 row)
```

几何类型转换函数

- box(circle)

描述：将圆转换成矩形

返回类型：box

示例：

```
vastbase=# SELECT box(circle '((0,0),2.0)') AS RESULT;
                            result
-------------------------------------------------------------------------
 (1.41421356237309,1.41421356237309),(-1.41421356237309,-1.41421356237309)
(1 row)
```

- box(point, point)

描述：将点转换成矩形

返回类型：box

示例：

```
vastbase=# SELECT box(point '(0,0)', point '(1,1)') AS RESULT;
   result
-------------
 (1,1),(0,0)
(1 row)
```

- box(polygon)

描述：将多边形转换成矩形

返回类型：box

示例：

```
vastbase=# SELECT box(polygon '((0,0),(1,1),(2,0))') AS RESULT;
  result
-------------
 (2,1),(0,0)
(1 row)
```

- circle(box)

  描述：矩形转换成圆

  返回类型：circle

  示例：

```
vastbase=# SELECT circle(box '((0,0),(1,1))') AS RESULT;
          result
------------------------------
 <(0.5,0.5),0.707106781186548>
(1 row)
```

- circle(point, double precision)

  描述：将圆心和半径转换成圆

  返回类型：circle

  示例：

```
vastbase=# SELECT circle(point '(0,0)', 2.0) AS RESULT;
  result
-----------
 <(0,0),2>
(1 row)
```

- circle(polygon)

  描述：将多边形转换成圆

  返回类型：circle

  示例：

```
vastbase=# SELECT circle(polygon '((0,0),(1,1),(2,0))') AS RESULT;
                result
-----------------------------------------
 <(1,0.333333333333333),0.924950591148529>
(1 row)
```

- lseg(box)

  描述：矩形对角线转化成线段

  返回类型：lseg

  示例：

```
vastbase=# SELECT lseg(box '((-1,0),(1,0))') AS RESULT;
    result
----------------
 [(1,0),(-1,0)]
(1 row)
```

- lseg(point, point)

描述：点转换成线段

返回类型：lseg

示例：

```
vastbase=# SELECT lseg(point '(-1,0)', point '(1,0)') AS RESULT;
    result
---------------
 [(-1,0),(1,0)]
(1 row)
```

- path(polygon)

描述：多边形转换成路径

返回类型：path

示例：

```
vastbase=# SELECT path(polygon '((0,0),(1,1),(2,0))') AS RESULT;
     result
--------------------
 ((0,0),(1,1),(2,0))
(1 row)
```

- point(double precision, double precision)

描述：节点

返回类型：point

示例：

```
vastbase=# SELECT point(23.4, -44.5) AS RESULT;
   result
--------------
 (23.4,-44.5)
(1 row)
```

- point(box)

描述：矩形的中心

返回类型：point

示例：

```
vastbase=# SELECT point(box '((-1,0),(1,0))') AS RESULT;
 result
--------
 (0,0)
(1 row)
```

- point(circle)

描述：圆心

返回类型：point

示例：

```
vastbase=# SELECT point(circle '((0,0),2.0)') AS RESULT;
 result
--------
 (0,0)
(1 row)
```

- point(lseg)

  描述：线段的中心

  返回类型：point

  示例：

```
vastbase=# SELECT point(lseg '((-1,0),(1,0))') AS RESULT;
 result
--------
 (0,0)
(1 row)
```

- point(polygon)

  描述：多边形的中心

  返回类型：point

  示例：

```
vastbase=# SELECT point(polygon '((0,0),(1,1),(2,0))') AS RESULT;
       result
----------------------
 (1,0.333333333333333)
(1 row)
```

- polygon(box)

  描述：矩形转换成 4 点多边形

  返回类型：polygon

  示例：

```
vastbase=# SELECT polygon(box '((0,0),(1,1))') AS RESULT;
        result
---------------------------
 ((0,0),(0,1),(1,1),(1,0))
(1 row)
```

- polygon(circle)

  描述：圆转换成 12 点多边形

  返回类型：polygon

  示例：

```
vastbase=# SELECT polygon(circle '((0,0),2.0)') AS RESULT;

result

------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------
```

```
----------------------------------------------------------------------------------------------
-------------------------

((-2,0),(-1.73205080756888,1),(-1,1.73205080756888),(-1.22464679914735e-16,2),(1,1.7320508
0756888),(1.73205080756888,1),(2,2.44929359829471e-16),(1.73205080756888,-0.99999999999999
9),(1,-1.73205080756888),(3.67394039744206e-16,-2),(-0.999999999999999,-1.73205080756888),
(-1.73205080756888,-1))
(1 row)
```

❖ polygon(npts, circle)

描述：圆转换成 npts 点多边形

返回类型：polygon

示例：

```
vastbase=# SELECT polygon(12, circle '((0,0),2.0)') AS RESULT;

result

----------------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------------
-------------------------

((-2,0),(-1.73205080756888,1),(-1,1.73205080756888),(-1.22464679914735e-16,2),(1,1.7320508
0756888),(1.73205080756888,1),(2,2.44929359829471e-16),(1.73205080756888,-0.99999999999999
9),(1,-1.73205080756888),(3.67394039744206e-16,-2),(-0.999999999999999,-1.73205080756888),
(-1.73205080756888,-1))
(1 row)
```

❖ polygon(path)

描述：路径转换成多边形

返回类型：polygon

示例：

```
vastbase=# SELECT polygon(path '((0,0),(1,1),(2,0))') AS RESULT;
      result
--------------------
 ((0,0),(1,1),(2,0))
(1 row)
```

11.5.11.网络地址函数和操作符

cidr 和 inet 操作符

操作符<<，<<=，>>，>>=对子网进行测试。它们只考虑两个地址的网络部分（忽略任何主机部分），然后判断其中一个网络是等于另外一个网络，还是另外一个网络的子网。

❖ <

描述：小于

示例：

```
vastbase=# SELECT inet '192.168.1.5' < inet '192.168.1.6' AS RESULT;
 result
--------
 t
(1 row)
```

- <=

描述：小于或等于

示例：

```
vastbase=# SELECT inet '192.168.1.5' <= inet '192.168.1.5' AS RESULT;
 result
--------
 t
(1 row)
```

- =

描述：等于

示例：

```
vastbase=# SELECT inet '192.168.1.5' = inet '192.168.1.5' AS RESULT;
 result
--------
 t
(1 row)
```

- >=

描述：大于或等于

示例：

```
vastbase=# SELECT inet '192.168.1.5' >= inet '192.168.1.5' AS RESULT;
 result
--------
 t
(1 row)
```

- >

描述：大于

示例：

```
vastbase=# SELECT inet '192.168.1.5' > inet '192.168.1.4' AS RESULT;
 result
--------
 t
(1 row)
```

- <>

描述：不等于

示例：

```
vastbase=# SELECT inet '192.168.1.5' <> inet '192.168.1.4' AS RESULT;
 result
--------
 t
(1 row)
```

- <<

描述：包含于

示例：

```
vastbase=# SELECT inet '192.168.1.5' << inet '192.168.1/24' AS RESULT;
 result
--------
 t
(1 row)
```

- <<=

描述：包含于或等于

示例：

```
vastbase=# SELECT inet '192.168.1/24' <<= inet '192.168.1/24' AS RESULT;
 result
--------
 t
(1 row)
```

- >>

描述：包含

示例：

```
vastbase=# SELECT inet '192.168.1/24' >> inet '192.168.1.5' AS RESULT;
 result
--------
 t
(1 row)
```

- >>=

描述：包含或等于

示例：

```
vastbase=# SELECT inet '192.168.1/24' >>= inet '192.168.1/24' AS RESULT;
 result
--------
 t
(1 row)
```

- ~

描述：位非

示例：

```
vastbase=# SELECT ~ inet '192.168.1.6' AS RESULT;
    result
---------------
```

```
 63.87.254.249
(1 row)
```

- &

描述：两个网络地址的每一位都进行“与”操作。

示例：

```
vastbase=# SELECT inet '192.168.1.6' & inet '10.0.0.0' AS RESULT;
 result
---------
 0.0.0.0
(1 row)
```

- |

描述：两个网络地址的每一位都进行“或”操作。

示例：

```
vastbase=# SELECT inet '192.168.1.6' | inet '10.0.0.0' AS RESULT;
   result
-------------
 202.168.1.6
(1 row)
```

- +

描述：加

示例：

```
vastbase=# SELECT inet '192.168.1.6' + 25 AS RESULT;
    result
--------------
 192.168.1.31
(1 row)
```

- -

描述：减

示例：

```
vastbase=# SELECT inet '192.168.1.43' - 36 AS RESULT;
   result
-------------
 192.168.1.7
(1 row)
```

- -

描述：减

示例：

```
vastbase=# SELECT inet '192.168.1.43' - inet '192.168.1.19' AS RESULT;
 result
--------
     24
(1 row)
```

cidr 和 inet 函数

函数 abbrev，host，text 主要是为了提供可选的显示格式。

- abbrev(inet)

  描述：缩写显示格式文本。

  返回类型：text

  示例：

```
vastbase=# SELECT abbrev(inet '10.1.0.0/16') AS RESULT;
   result
-------------
 10.1.0.0/16
(1 row)
```

- abbrev(cidr)

  描述：缩写显示格式文本。

  返回类型：text

  示例：

```
vastbase=# SELECT abbrev(cidr '10.1.0.0/16') AS RESULT;
 result
---------
 10.1/16
(1 row)
```

- broadcast(inet)

  描述：网络广播地址。

  返回类型：inet

  示例：

```
vastbase=# SELECT broadcast('192.168.1.5/24') AS RESULT;
     result
------------------
 192.168.1.255/24
(1 row)
```

- family(inet)

  描述：抽取地址族，4 为 IPv4，6 为 IPv6。

  返回类型：int

  示例：

```
vastbase=# SELECT family('::1') AS RESULT;
 result
--------
      6
(1 row)
```

- host(inet)

描述：将主机地址类型抽出为文本。

返回类型：text

示例：

```
vastbase=# SELECT host('192.168.1.5/24') AS RESULT;
  result
-------------
 192.168.1.5
(1 row)
```

- hostmask(inet)

描述：为网络构造主机掩码。

返回类型：inet

示例：

```
vastbase=# SELECT hostmask('192.168.23.20/30') AS RESULT;
 result
---------
 0.0.0.3
(1 row)
```

- masklen(inet)

描述：抽取子网掩码长度。

返回类型：int

示例：

```
vastbase=# SELECT masklen('192.168.1.5/24') AS RESULT;
 result
--------
     24
(1 row)
```

- netmask(inet)

描述：为网络构造子网掩码。

返回类型：inet

示例：

```
vastbase=# SELECT netmask('192.168.1.5/24') AS RESULT;
    result
---------------
 255.255.255.0
(1 row)
```

- network(inet)

描述：抽取地址的网络部分。

返回类型：cidr

示例：

```
vastbase=# SELECT network('192.168.1.5/24') AS RESULT;
    result
----------------
 192.168.1.0/24
(1 row)
```

- set_masklen(inet, int)

  描述：为 inet 数值设置子网掩码长度。

  返回类型：inet

  示例：

```
vastbase=# SELECT set_masklen('192.168.1.5/24', 16) AS RESULT;
    result
----------------
 192.168.1.5/16
(1 row)
```

- set_masklen(cidr, int)

  描述：为 cidr 数值设置子网掩码长度。

  返回类型：cidr

  示例：

```
vastbase=# SELECT set_masklen('192.168.1.0/24'::cidr, 16) AS RESULT;
    result
----------------
 192.168.0.0/16
(1 row)
```

- text(inet)

  描述：把 IP 地址和掩码长度抽取为文本。

  返回类型：text

  示例：

```
vastbase=# SELECT text(inet '192.168.1.5') AS RESULT;
    result
----------------
 192.168.1.5/32
(1 row)
```

任何 cidr 值都能以显式或者隐式的方式转换为 inet 值，因此上述能够操作 inet 值的函数也同样能够操作 cidr 值。inet 值也可以转换为 cidr 值，此时 inet 子网掩码右侧的所有位都将转换为零，以创建一个有效的 cidr 值。另外，用户还可以使用常规的类型转换语法将一个文本字符串转换为 inet 或 cidr 值。例如：inet(expression)或 colname::cidr。

macaddr 函数

函数 trunc(macaddr)返回一个 MAC 地址，该地址的最后三个字节设置为零。

trunc(macaddr)

描述：把后三个字节置为零。

返回类型：macaddr

示例：

```
vastbase=# SELECT trunc(macaddr '12:34:56:78:90:ab') AS RESULT;
     result
-------------------
 12:34:56:00:00:00
(1 row)
```

macaddr 类型还支持标准关系操作符（>，<=等）用于词法排序，和按位运算符（~，&和|）非，与和或。

11.5.12.文本检索函数和操作符

文本检索操作符

- @@

  描述：tsvector 类型的词汇与 tsquery 类型的词汇是否匹配

  示例：

  ```
  vastbase=# SELECT to_tsvector('fat cats ate rats') @@ to_tsquery('cat & rat') AS RESULT;
   result
  --------
   t
  (1 row)
  ```

- @@@

  描述：@@的同义词

  示例：

  ```
  vastbase=# SELECT to_tsvector('fat cats ate rats') @@@ to_tsquery('cat & rat') AS RESULT;
   result
  --------
   t
  (1 row)
  ```

- ||

  描述：连接两个 tsvector 类型的词汇

  示例：

  ```
  vastbase=# SELECT 'a:1 b:2'::tsvector || 'c:1 d:2 b:3'::tsvector AS RESULT;
          result
  --------------------------
   'a':1 'b':2,5 'c':3 'd':4
  (1 row)
  ```

- &&

  描述：将两个 tsquery 类型的词汇进行“与”操作

  示例：

```
vastbase=# SELECT 'fat | rat'::tsquery && 'cat'::tsquery AS RESULT;
        result
---------------------------
 ( 'fat' | 'rat' ) & 'cat'
(1 row)
```

- ||

  描述：将两个 tsquery 类型的词汇进行“或”操作

  示例：

```
vastbase=# SELECT 'fat | rat'::tsquery || 'cat'::tsquery AS RESULT;
        result
---------------------------
 ( 'fat' | 'rat' ) | 'cat'
(1 row)
```

- !!

  描述：tsquery 类型词汇的非关系

  示例：

```
vastbase=# SELECT !! 'cat'::tsquery AS RESULT;
 result
--------
 !'cat'
(1 row)
```

- @>

  描述：一个 tsquery 类型的词汇是否包含另一个 tsquery 类型的词汇

  示例：

```
vastbase=# SELECT 'cat'::tsquery @> 'cat & rat'::tsquery AS RESULT;
 result
--------
 f
(1 row)
```

- <@

  描述：一个 tsquery 类型的词汇是否被包含另一个 tsquery 类型的词汇

  示例：

```
vastbase=# SELECT 'cat'::tsquery <@ 'cat & rat'::tsquery AS RESULT;
 result
--------
 t
(1 row)
```

除了上述的操作符，还为 tsvector 类型和 tsquery 类型的数据定义了普通的 B-tree 比较操作符（=，<等）。

文本检索函数

- get_current_ts_config()

描述：获取文本检索的默认配置。

返回类型：regconfig

示例：

```
vastbase=# SELECT get_current_ts_config();
 get_current_ts_config
-----------------------
 english
(1 row)
```

- length(tsvector)

描述：tsvector 类型词汇的单词数。

返回类型：integer

示例：

```
vastbase=# SELECT length('fat:2,4 cat:3 rat:5A'::tsvector);
 length
--------
      3
(1 row)
```

- numnode(tsquery)

描述：tsquery 类型的单词加上操作符的数量。

返回类型：integer

示例：

```
vastbase=# SELECT numnode('(fat & rat) | cat'::tsquery);
 numnode
---------
       5
(1 row)
```

- plainto_tsquery([config regconfig ,] query text)

描述：产生 tsquery 类型的词汇，并忽略标点

返回类型：tsquery

示例：

```
vastbase=# SELECT plainto_tsquery('english', 'The Fat Rats');
 plainto_tsquery
-----------------
 'fat' & 'rat'
(1 row)
```

- querytree(query tsquery)

描述：获取 tsquery 类型的词汇可加索引的部分。

返回类型：text

示例：

```
vastbase=# SELECT querytree('foo & ! bar'::tsquery);
 querytree
-----------
 'foo'
(1 row)
```

- setweight(tsvector, "char")

  描述：给 tsvector 类型的每个元素分配权值。

  返回类型：tsvector

  示例：

```
vastbase=# SELECT setweight('fat:2,4 cat:3 rat:5B'::tsvector, 'A');
         setweight
-------------------------------
 'cat':3A 'fat':2A,4A 'rat':5A
(1 row)
```

- strip(tsvector)

  描述：删除 tsvector 类型单词中的 position 和权值。

  返回类型：tsvector

  示例：

```
vastbase=# SELECT strip('fat:2,4 cat:3 rat:5A'::tsvector);
      strip
-------------------
 'cat' 'fat' 'rat'
(1 row)
```

- to_tsquery([config regconfig ,] query text)

  描述：标准化单词，并转换为 tsquery 类型。

  返回类型：tsquery

  示例：

```
vastbase=# SELECT to_tsquery('english', 'The & Fat & Rats');
  to_tsquery
---------------
 'fat' & 'rat'
(1 row)
```

- to_tsvector([config regconfig ,] document text)

  描述：去除文件信息，并转换为 tsvector 类型。

  返回类型：tsvector

  示例：

```
vastbase=# SELECT to_tsvector('english', 'The Fat Rats');
   to_tsvector
-----------------
 'fat':2 'rat':3
(1 row)
```

- ts_headline([config regconfig,] document text, query tsquery [, options text])

描述：高亮显示查询的匹配项。

返回类型：text

示例：

```
vastbase=# SELECT ts_headline('x y z', 'z'::tsquery);
 ts_headline
--------------
 x y <b>z</b>
(1 row)
```

- ts_rank([weights float4[],] vector tsvector, query tsquery [, normalization integer])

描述：文档查询排名。

返回类型：float4

示例：

```
vastbase=# SELECT ts_rank('hello world'::tsvector, 'world'::tsquery);
 ts_rank
----------
 .0607927
(1 row)
```

- ts_rank_cd([weights float4[],] vector tsvector, query tsquery [, normalization integer])

描述：排序文件查询使用覆盖密度。

返回类型：float4

示例：

```
vastbase=# SELECT ts_rank_cd('hello world'::tsvector, 'world'::tsquery);
 ts_rank_cd
------------
        .1
(1 row)
```

- ts_rewrite(query tsquery, target tsquery, substitute tsquery)

描述：替换目标 tsquery 类型的单词。

返回类型：tsquery

示例：

```
vastbase=# SELECT ts_rewrite('a & b'::tsquery, 'a'::tsquery, 'foo|bar'::tsquery);
     ts_rewrite
------------------------
 'b' & ( 'foo' | 'bar' )
(1 row)
```

- ts_rewrite(query tsquery, select text)

描述：使用 SELECT 命令的结果替代目标中 tsquery 类型的单词。

返回类型：tsquery

示例：

```
vastbase=# SELECT ts_rewrite('world'::tsquery, 'select ''world''::tsquery,
''hello''::tsquery');
 ts_rewrite
------------
 'hello'
(1 row)
```

文本检索调试函数

- ts_debug([config regconfig,] document text, OUT alias text, OUT description text, OUT token text, OUT dictionaries regdictionary[], OUT dictionary regdictionary, OUT lexemes text[])

  描述：测试一个配置。

  返回类型：setof record

  示例：

```
vastbase=# SELECT ts_debug('english', 'The Brightest supernovaes');
                              ts_debug
-------------------------------------------------------------------------------
 (asciiword,"Word, all ASCII",The,{english_stem},english_stem,{})
 (blank,"Space symbols"," ",{},,)
 (asciiword,"Word, all ASCII",Brightest,{english_stem},english_stem,{brightest})
 (blank,"Space symbols"," ",{},,)
 (asciiword,"Word, all ASCII",supernovaes,{english_stem},english_stem,{supernova})
(5 rows)
```

- ts_lexize(dict regdictionary, token text)

  描述：测试一个数据字典。

  返回类型：text[]

  示例：

```
vastbase=# SELECT ts_lexize('english_stem', 'stars');
 ts_lexize
-----------
 {star}
(1 row)
```

- ts_parse(parser_name text, document text, OUT tokid integer, OUT token text)

  描述：测试一个解析。

  返回类型：setof record

  示例：

```
vastbase=# SELECT ts_parse('default', 'foo - bar');
 ts_parse
-----------
 (1,foo)
 (12," ")
 (12,"- ")
```

```
 (1,bar)
(4 rows)
```

- ts_parse(parser_oid oid, document text, OUT tokid integer, OUT token text)

  描述：测试一个解析。

  返回类型：setof record

  示例：

```
vastbase=# SELECT ts_parse(3722, 'foo - bar');
 ts_parse
-----------
 (1,foo)
 (12," ")
 (12,"- ")
 (1,bar)
(4 rows)
```

- ts_token_type(parser_name text, OUT tokid integer, OUT alias text, OUT description text)

  描述：获取分析器定义的记号类型。

  返回类型：setof record

  示例：

```
vastbase=# SELECT ts_token_type('default');
                   ts_token_type
-------------------------------------------------------------
 (1,asciiword,"Word, all ASCII")
 (2,word,"Word, all letters")
 (3,numword,"Word, letters and digits")
 (4,email,"Email address")
 (5,url,URL)
 (6,host,Host)
 (7,sfloat,"Scientific notation")
 (8,version,"Version number")
 (9,hword_numpart,"Hyphenated word part, letters and digits")
 (10,hword_part,"Hyphenated word part, all letters")
 (11,hword_asciipart,"Hyphenated word part, all ASCII")
 (12,blank,"Space symbols")
 (13,tag,"XML tag")
 (14,protocol,"Protocol head")
 (15,numhword,"Hyphenated word, letters and digits")
 (16,asciihword,"Hyphenated word, all ASCII")
 (17,hword,"Hyphenated word, all letters")
 (18,url_path,"URL path")
 (19,file,"File or path name")
 (20,float,"Decimal notation")
 (21,int,"Signed integer")
 (22,uint,"Unsigned integer")
 (23,entity,"XML entity")
(23 rows)
```

- ts_token_type(parser_oid oid, OUT tokid integer, OUT alias text, OUT description text)

描述：获取分析器定义的记号类型。

返回类型：setof record

示例：

```
vastbase=# SELECT ts_token_type(3722);
                   ts_token_type
------------------------------------------------------------
 (1,asciiword,"Word, all ASCII")
 (2,word,"Word, all letters")
 (3,numword,"Word, letters and digits")
 (4,email,"Email address")
 (5,url,URL)
 (6,host,Host)
 (7,sfloat,"Scientific notation")
 (8,version,"Version number")
 (9,hword_numpart,"Hyphenated word part, letters and digits")
 (10,hword_part,"Hyphenated word part, all letters")
 (11,hword_asciipart,"Hyphenated word part, all ASCII")
 (12,blank,"Space symbols")
 (13,tag,"XML tag")
 (14,protocol,"Protocol head")
 (15,numhword,"Hyphenated word, letters and digits")
 (16,asciihword,"Hyphenated word, all ASCII")
 (17,hword,"Hyphenated word, all letters")
 (18,url_path,"URL path")
 (19,file,"File or path name")
 (20,float,"Decimal notation")
 (21,int,"Signed integer")
 (22,uint,"Unsigned integer")
 (23,entity,"XML entity")
(23 rows)
```

- ts_stat(sqlquery text, [weights text,] OUT word text, OUT ndoc integer, OUT nentry integer)

描述：获取 tsvector 列的统计数据。

返回类型：setof record

示例：

```
vastbase=# SELECT ts_stat('select ''hello world''::tsvector');
  ts_stat
-------------
 (world,1,1)
 (hello,1,1)
(2 rows)
```

11.5.13.JSON 函数

JSON 函数表示可以用于 JSON 类型（请参考 11.3.12JSON 类型）数据的函数。

- array_to_json(anyarray [, pretty_bool])

描述：返回 JSON 类型的数组。一个多维数组成为一个 JSON 数组的数组。如果 pretty_bool 为 true，将在一维元素之间添加换行符。

返回类型：json

示例：

```
vastbase=# SELECT array_to_json('{{1,5},{99,100}}'::int[]);
array_to_json
------------------
[[1,5],[99,100]]
(1 row)
```

- row_to_json(record [, pretty_bool])

描述：返回 JSON 类型的行。如果 pretty_bool 为 true，将在第一级元素之间添加换行符。

返回类型：json

示例：

```
vastbase=# SELECT row_to_json(row(1,'foo'));
    row_to_json
---------------------
 {"f1":1,"f2":"foo"}
(1 row)
```

11.5.14. SEQUENCE 函数

序列函数为用户从序列对象中获取后续的序列值提供了简单的多用户安全的方法。

- nextval(regclass)

描述：递增序列并返回新值。

说明

为了避免从同一个序列获取值的并发事务被阻塞， nextval 操作不会回滚；也就是说，一旦一个值已经被抓取， 那么就认为它已经被用过了，并且不会再被返回。 即使该操作处于事务中，当事务之后中断，或者如果调用查询结束不使用该值，也是如此。这种情况将在指定值的顺序中留下未使用的"空洞"。 因此，Vastbase 序列对象不能用于获得"无间隙"序列。

须知

nextval 函数只能在主机上执行，备机不支持执行此函数。

返回类型：bigint

nextval 函数有两种调用方式（其中第二种调用方式目前不支持 Sequence 命名中有特殊字符"."的情况），如下：

示例 1：

```
vastbase=# select nextval('seqDemo');
 nextval
---------
```

```
      2
(1 row)
```

示例 2：

```
vastbase=# select seqDemo.nextval;
 nextval
---------
      2
(1 row)
```

- currval(regclass)

返回当前会话里最近一次 nextval 返回的指定的 sequence 的数值。如果当前会话还没有调用过指定的 sequence 的 nextval，那么调用 currval 将会报错。需要注意的是，这个函数在默认情况下是不支持的，需要通过设置 enable_beta_features 为 true 之后，才能使用这个函数。

返回类型：bigint

currval 函数有两种调用方式（其中第二种调用方式目前不支持 Sequence 命名中有特殊字符"."的情况），如下：

示例 1：

```
vastbase=# select currval('seq1');
 currval
---------
      2
(1 row)
```

示例 2：

```
vastbase=# select seq1.currval;
 currval
---------
      2
(1 row)
```

- lastval()

描述：返回当前会话里最近一次 nextval 返回的数值。这个函数等效于 currval，只是它不用序列名为参数，它抓取当前会话里面最近一次 nextval 使用的序列。如果当前会话还没有调用过 nextval，那么调用 lastval 将会报错。

需要注意的是，这个函数在默认情况下是不支持的，需要通过设置 enable_beta_features 或者 lastval_supported 为 true 之后，才能使用这个函数。

返回类型：bigint

示例：

```
vastbase=# select lastval();
 lastval
---------
      2
(1 row)
```

- setval(regclass, bigint)

描述：设置序列的当前数值。

返回类型：bigint

示例：

```
vastbase=# select setval('seqDemo',1);
 setval
--------
      1
(1 row)
```

- setval(regclass, bigint, Boolean)

描述：设置序列的当前数值以及 is_called 标志。

返回类型：bigint

示例：

```
vastbase=# select setval('seqDemo',1,true);
 setval
--------
      1
(1 row)
```

说明

Setval 后当前会话及 GTM 上会立刻生效，但如果其他会话有缓存的序列值，只能等到缓存值用尽才能感知 Setval 的作用。所以为了避免序列值冲突，setval 要谨慎使用。

因为序列是非事务的，setval 造成的改变不会由于事务的回滚而撤销。

须知

nextval 函数只能在主机上执行，备机不支持执行此函数。

11.5.15.数组函数和操作符

数组操作符

- =

描述：两个数组是否相等

示例：

```
vastbase=# SELECT ARRAY[1.1,2.1,3.1]::int[] = ARRAY[1,2,3] AS RESULT ;
 result
--------
 t
(1 row)
```

- <>

描述：两个数组是否不相等

示例：

```
vastbase=# SELECT ARRAY[1,2,3] <> ARRAY[1,2,4] AS RESULT;
 result
--------
 t
(1 row)
```

- <

描述：一个数组是否小于另一个数组

示例：

```
vastbase=# SELECT ARRAY[1,2,3] < ARRAY[1,2,4] AS RESULT;
 result
--------
 t
(1 row)
```

- >

描述：一个数组是否大于另一个数组

示例：

```
vastbase=# SELECT ARRAY[1,4,3] > ARRAY[1,2,4] AS RESULT;
 result
--------
 t
(1 row)
```

- <=

描述：一个数组是否小于或等于另一个数组

示例：

```
vastbase=# SELECT ARRAY[1,2,3] <= ARRAY[1,2,3] AS RESULT;
 result
--------
 t
(1 row)
```

- >=

描述：一个数组是否大于或等于另一个数组

示例：

```
vastbase=# SELECT ARRAY[1,4,3] >= ARRAY[1,4,3] AS RESULT;
 result
--------
 t
(1 row)
```

- @>

描述：一个数组是否包含另一个数组

示例：

```
vastbase=# SELECT ARRAY[1,4,3] @> ARRAY[3,1] AS RESULT;
 result
--------
 t
(1 row)
```

- <@

描述：一个数组是否被包含于另一个数组

示例：

```
vastbase=# SELECT ARRAY[2,7] <@ ARRAY[1,7,4,2,6] AS RESULT;
 result
--------
 t
(1 row)
```

- &&

描述：一个数组是否和另一个数组重叠（有共同元素）

示例：

```
vastbase=# SELECT ARRAY[1,4,3] && ARRAY[2,1] AS RESULT;
 result
--------
 t
(1 row)
```

- ||

描述：数组与数组进行连接

示例：

```
vastbase=# SELECT ARRAY[1,2,3] || ARRAY[4,5,6] AS RESULT;
   result
---------------
 {1,2,3,4,5,6}
(1 row)
vastbase=# SELECT ARRAY[1,2,3] || ARRAY[[4,5,6],[7,8,9]] AS RESULT;
        result
---------------------------
 {{1,2,3},{4,5,6},{7,8,9}}
(1 row)
```

- ||

描述：元素与数组进行连接

示例：

```
vastbase=# SELECT 3 || ARRAY[4,5,6] AS RESULT;
  result
-----------
 {3,4,5,6}
(1 row)
```

- ||

描述：数组与元素进行连接

示例：

```
vastbase=# SELECT ARRAY[4,5,6] || 7 AS RESULT;
  result
-----------
 {4,5,6,7}
(1 row)
```

数组比较是使用默认的 B-tree 比较函数对所有元素逐一进行比较的。多维数组的元素按照行顺序进行访问。如果两个数组的内容相同但维数不等，决定排序顺序的首要因素是维数。

数组函数

- array_append(anyarray, anyelement)

  描述：向数组末尾添加元素，只支持一维数组。

  返回类型：anyarray

  示例：

```
vastbase=# SELECT array_append(ARRAY[1,2], 3) AS RESULT;
 result
---------
 {1,2,3}
(1 row)
```

- array_prepend(anyelement, anyarray)

  描述：向数组开头添加元素，只支持一维数组。

  返回类型：anyarray

  示例：

```
vastbase=# SELECT array_prepend(1, ARRAY[2,3]) AS RESULT;
 result
---------
 {1,2,3}
(1 row)
```

- array_cat(anyarray, anyarray)

  描述：连接两个数组，支持多维数组。

  返回类型：anyarray

  示例：

```
vastbase=# SELECT array_cat(ARRAY[1,2,3], ARRAY[4,5]) AS RESULT;
   result
-------------
 {1,2,3,4,5}
(1 row)

vastbase=# SELECT array_cat(ARRAY[[1,2],[4,5]], ARRAY[6,7]) AS RESULT;
     result
---------------------
```

```
 {{1,2},{4,5},{6,7}}
(1 row)
```

- array_ndims(anyarray)

  描述：返回数组的维数。

  返回类型：int

  示例：

```
vastbase=# SELECT array_ndims(ARRAY[[1,2,3], [4,5,6]]) AS RESULT;
 result
--------
      2
(1 row)
```

- array_dims(anyarray)

  描述：返回数组维数的文本表示。

  返回类型：text

  示例：

```
vastbase=# SELECT array_dims(ARRAY[[1,2,3], [4,5,6]]) AS RESULT;
   result
------------
 [1:2][1:3]
(1 row)
```

- array_length(anyarray, int)

  描述：返回数组维度的长度。

  返回类型：int

  示例：

```
vastbase=# SELECT array_length(array[1,2,3], 1) AS RESULT;
 result
--------
      3
(1 row)
```

- array_lower(anyarray, int)

  描述：返回数组维数的下界。

  返回类型：int

  示例：

```
vastbase=# SELECT array_lower('[0:2]={1,2,3}'::int[], 1) AS RESULT;
 result
--------
      0
(1 row)
```

- array_upper(anyarray, int)

  描述：返回数组维数的上界。

返回类型：int

示例：

```
vastbase=# SELECT array_upper(ARRAY[1,8,3,7], 1) AS RESULT;
 result
--------
      4
(1 row)
```

- array_to_string(anyarray, text [, text])

描述：使用第一个 text 作为数组的新分隔符，使用第二个 text 替换数组值为 null 的值。

返回类型：text

示例：

```
vastbase=# SELECT array_to_string(ARRAY[1, 2, 3, NULL, 5], ',', '*') AS RESULT;
  result
-----------
 1,2,3,*,5
(1 row)
```

- string_to_array(text, text [, text])

描述：使用第二个 text 指定分隔符，使用第三个可选的 text 作为 NULL 值替换模板，如果分隔后的子串与第三个可选的 text 完全匹配，则将其替换为 NULL。

返回类型：text[]

示例：

```
vastbase=# SELECT string_to_array('xx~^~yy~^~zz', '~^~', 'yy') AS RESULT;
    result
--------------
 {xx,NULL,zz}
(1 row)
vastbase=# SELECT string_to_array('xx~^~yy~^~zz', '~^~', 'y') AS RESULT;
   result
------------
 {xx,yy,zz}
(1 row)
```

- unnest(anyarray)

描述：扩大一个数组为一组行。

返回类型：setof anyelement

示例：

```
vastbase=# SELECT unnest(ARRAY[1,2]) AS RESULT;
 result
--------
      1
      2
(2 rows)
```

在 string_to_array 中，如果分隔符参数是 NULL，输入字符串中的每个字符将在结果数组中变成一个独立的元素。如果分隔符是一个空白字符串，则整个输入的字符串将变为一个元素的数组。否则输入字符串将在每个分隔字符串处分开。

在 string_to_array 中，如果省略 null 字符串参数或为 NULL，将字符串中没有输入内容的子串替换为 NULL。

在 array_to_string 中，如果省略 null 字符串参数或为 NULL，运算中将跳过在数组中的任何 null 元素，并且不会在输出字符串中出现。

11.5.16.范围函数和操作符

范围操作符

- =

  描述：等于

  示例：

```
vastbase=# SELECT int4range(1,5) = '[1,4]'::int4range AS RESULT;
 result
--------
 t
(1 row)
```

- <>

  描述：不等于

  示例：

```
vastbase=# SELECT numrange(1.1,2.2) <> numrange(1.1,2.3) AS RESULT;
 result
--------
 t
(1 row)
```

- <

  描述：小于

  示例：

```
vastbase=# SELECT int4range(1,10) < int4range(2,3) AS RESULT;
 result
--------
 t
(1 row)
```

- >

  描述：大于

  示例：

```
vastbase=# SELECT int4range(1,10) > int4range(1,5) AS RESULT;
 result
```

```
--------
 t
(1 row)
```

- <=

  描述：小于或等于

  示例：

```
vastbase=# SELECT numrange(1.1,2.2) <= numrange(1.1,2.2) AS RESULT;
 result
--------
 t
(1 row)
```

- >=

  描述：大于或等于

  示例：

```
vastbase=# SELECT numrange(1.1,2.2) >= numrange(1.1,2.0) AS RESULT;
 result
--------
 t
(1 row)
```

- @>

  描述：包含范围

  示例：

```
vastbase=# SELECT int4range(2,4) @> int4range(2,3) AS RESULT;
 result
--------
 t
(1 row)
```

- @>

  描述：包含元素

  示例：

```
vastbase=# SELECT '[2011-01-01,2011-03-01)'::tsrange @> '2011-01-10'::timestamp AS RESULT;
 result
--------
 t
(1 row)
```

- <@

  描述：范围包含于

  示例：

```
vastbase=# SELECT int4range(2,4) <@ int4range(1,7) AS RESULT;
 result
--------
 t
(1 row)
```

- <@

  描述：元素包含于

  示例：

```
vastbase=# SELECT 42 <@ int4range(1,7) AS RESULT;
 result
--------
 f
(1 row)
```

- &&

  描述：重叠（有共同点）

  示例：

```
vastbase=# SELECT int8range(3,7) && int8range(4,12) AS RESULT;
 result
--------
 t
(1 row)
```

- <<

  描述：范围值是否比另一个范围值的最小值还小（没有交集）

  示例：

```
vastbase=# SELECT int8range(1,10) << int8range(100,110) AS RESULT;
 result
--------
 t
(1 row)
```

- >>

  描述：范围值是否比另一个范围值的最大值还大（没有交集）

  示例：

```
vastbase=# SELECT int8range(50,60) >> int8range(20,30) AS RESULT;
 result
--------
 t
(1 row)
```

- &<

  描述：范围值的最大值是否不超过另一个范围值的最大值。

  示例：

```
vastbase=# SELECT int8range(1,20) &< int8range(18,20) AS RESULT;
 result
--------
 t
(1 row)
```

- &>

  描述：范围值的最小值是否不小于另一个范围值的最小值。

示例：

```
vastbase=# SELECT int8range(7,20) &> int8range(5,10) AS RESULT;
 result
--------
 t
(1 row)
```

- -|-

描述：相邻

示例：

```
vastbase=# SELECT numrange(1.1,2.2) -|- numrange(2.2,3.3) AS RESULT;
 result
--------
 t
(1 row)
```

- +

描述：并集

示例：

```
vastbase=# SELECT numrange(5,15) + numrange(10,20) AS RESULT;
 result
--------
 [5,20)
(1 row)
```

- *

描述：交集

示例：

```
vastbase=# SELECT int8range(5,15) * int8range(10,20) AS RESULT;
 result
---------
 [10,15)
(1 row)
```

- -

描述：差集

示例：

```
vastbase=# SELECT int8range(5,15) - int8range(10,20) AS RESULT;
 result
--------
 [5,10)
(1 row)
```

简单的比较操作符<，>，<=和>=先比较下界，只有下界相等时才比较上界。

<<、>>和-|-操作符当包含空范围时也会返回 false；也就是，不认为空范围在其他范围之前或之后。

并集和差集操作符的执行结果无法包含两个不相交的子范围。

范围函数

- lower(anyrange)

  描述：范围的下界

  返回类型：范围元素类型

  示例：

```
vastbase=# SELECT lower(numrange(1.1,2.2)) AS RESULT;
 result
--------
    1.1
(1 row)
```

- upper(anyrange)

  描述：范围的上界

  返回类型：范围元素类型

  示例：

```
vastbase=# SELECT upper(numrange(1.1,2.2)) AS RESULT;
 result
--------
    2.2
(1 row)
```

- isempty(anyrange)

  描述：范围是否为空

  返回类型：Boolean

  示例：

```
vastbase=# SELECT isempty(numrange(1.1,2.2)) AS RESULT;
 result
--------
 f
(1 row)
```

- lower_inc(anyrange)

  描述：是否包含下界

  返回类型：Boolean

  示例：

```
vastbase=# SELECT lower_inc(numrange(1.1,2.2)) AS RESULT;
 result
--------
 t
(1 row)
```

- upper_inc(anyrange)

  描述：是否包含上界

返回类型：Boolean

示例：

```
vastbase=# SELECT upper_inc(numrange(1.1,2.2)) AS RESULT;
 result
--------
 f
(1 row)
```

- lower_inf(anyrange)

  描述：下界是否为无穷

  返回类型：Boolean

  示例：

```
vastbase=# SELECT lower_inf('(,)'::daterange) AS RESULT;
 result
--------
 t
(1 row)
```

- upper_inf(anyrange)

  描述：上界是否为无穷

  返回类型：Boolean

  示例：

```
vastbase=# SELECT upper_inf('(,)'::daterange) AS RESULT;
 result
--------
 t
(1 row)
```

如果范围是空或者需要的界限是无穷的，lower 和 upper 函数将返回 null。lower_inc、upper_inc、lower_inf 和 upper_inf 函数均对空范围返回 false。

11.5.17.聚集函数

聚集函数

- sum(expression)

  描述：所有输入行的 expression 总和。

  返回类型：

  通常情况下输入数据类型和输出数据类型是相同的，但以下情况会发生类型转换：

  - 对于 SMALLINT 或 INT 输入，输出类型为 BIGINT。
  - 对于 BIGINT 输入，输出类型为 NUMBER 。
  - 对于浮点数输入，输出类型为 DOUBLE PRECISION。

示例：

```
vastbase=# SELECT SUM(ss_ext_tax) FROM tpcds.STORE_SALES;
  sum
--------------
 213267594.69
(1 row)
```

- max(expression)

描述：所有输入行中 expression 的最大值。

参数类型：任意数组、数值、字符串、日期/时间类型。

返回类型：与参数数据类型相同

示例：

```
vastbase=# SELECT MAX(inv_quantity_on_hand) FROM tpcds.inventory;
```

- min(expression)

描述：所有输入行中 expression 的最小值。

参数类型：任意数组、数值、字符串、日期/时间类型。

返回类型：与参数数据类型相同

示例：

```
vastbase=# SELECT MIN(inv_quantity_on_hand) FROM tpcds.inventory;
 min
-----
   0
(1 row)
```

- avg(expression)

描述：所有输入值的均值（算术平均）。

返回类型：

对于任何整数类型输入，结果都是 NUMBER 类型。

对于任何浮点输入，结果都是 DOUBLE PRECISION 类型。

否则和输入数据类型相同。

示例：

```
vastbase=# SELECT AVG(inv_quantity_on_hand) FROM tpcds.inventory;
        avg
----------------------
 500.0387129084044604
(1 row)
```

- count(expression)

描述：返回表中满足 expression 不为 NULL 的行数。

返回类型：BIGINT

示例：

```
vastbase=# SELECT COUNT(inv_quantity_on_hand) FROM tpcds.inventory;
  count
----------
 11158087
(1 row)
```

- count(*)

  描述：返回表中的记录行数。

  返回类型：BIGINT

  示例：

```
vastbase=# SELECT COUNT(*) FROM tpcds.inventory;
  count
----------
 11745000
(1 row)
```

- array_agg(expression)

  描述：将所有输入值（包括空）连接成一个数组。

  返回类型：参数类型的数组

  示例：

```
vastbase=# SELECT ARRAY_AGG(sr_fee) FROM tpcds.store_returns WHERE sr_customer_sk = 2;
   array_agg
---------------
 {22.18,63.21}
(1 row)
```

- string_agg(expression, delimiter)

  描述：将输入值连接成为一个字符串，用分隔符分开。

  返回类型：和参数数据类型相同。

  示例：

```
vastbase=# SELECT string_agg(sr_item_sk, ',') FROM tpcds.store_returns where sr_item_sk < 3;
       string_agg
--------------------------------------------------------------------------------
----------------------------
 1,2,1,2,2,1,1,2,2,1,2,1,2,1,1,1,2,1,1,1,1,1,2,1,1,1,1,1,2,2,1,1,1,1,1,1,1,1,2,
2,1,1,1,1,1,1,2,2,1,1,2,1,1,1
(1 row)
```

- listagg(expression [, delimiter]) WITHIN GROUP(ORDER BY order-list)

  描述：将聚集列数据按 WITHIN GROUP 指定的排序方式排列，并用 delimiter 指定的分隔符拼接成一个字符串。

  - expression：必选。指定聚集列名或基于列的有效表达式，不支持 DISTINCT 关键字和 VARIADIC 参数。
  - delimiter：可选。指定分隔符，可以是字符串常数或基于分组列的确定性表达式，缺省时

表示分隔符为空。

– order-list：必选。指定分组内的排序方式。

返回类型：text

示例：

聚集列是文本字符集类型。

```
vastbase=# SELECT deptno, listagg(ename, ',') WITHIN GROUP(ORDER BY ename) AS employees FROM
emp GROUP BY deptno;
 deptno |            employees
--------+-------------------------------------
     10 | CLARK,KING,MILLER
     20 | ADAMS,FORD,JONES,SCOTT,SMITH
     30 | ALLEN,BLAKE,JAMES,MARTIN,TURNER,WARD
(3 rows)
```

聚集列是整型。

```
vastbase=# SELECT deptno, listagg(mgrno, ',') WITHIN GROUP(ORDER BY mgrno NULLS FIRST) AS mgrnos
FROM emp GROUP BY deptno;
 deptno |          mgrnos
--------+-------------------------------
     10 | 7782,7839
     20 | 7566,7566,7788,7839,7902
     30 | 7698,7698,7698,7698,7698,7839
(3 rows)
```

聚集列是浮点类型。

```
vastbase=# SELECT job, listagg(bonus, '($); ') WITHIN GROUP(ORDER BY bonus DESC) || '($)' AS
bonus FROM emp GROUP BY job;
   job    |                  bonus
----------+-----------------------------------------------
 CLERK    | 10234.21($); 2000.80($); 1100.00($); 1000.22($)
 PRESIDENT | 23011.88($)
 ANALYST   | 2002.12($); 1001.01($)
 MANAGER   | 10000.01($); 2399.50($); 999.10($)
 SALESMAN  | 1000.01($); 899.00($); 99.99($); 9.00($)
(5 rows)
```

聚集列是时间类型。

```
vastbase=# SELECT deptno, listagg(hiredate, ', ') WITHIN GROUP(ORDER BY hiredate DESC) AS
hiredates FROM emp GROUP BY deptno;
 deptno |                                                hiredates
--------+-------------------------------------------------------------------------------------
-----------------------------------------
     10 | 1982-01-23 00:00:00, 1981-11-17 00:00:00, 1981-06-09 00:00:00
     20 | 2001-04-02 00:00:00, 1999-12-17 00:00:00, 1987-05-23 00:00:00, 1987-04-19 00:00:00,
1981-12-03 00:00:00
     30 | 2015-02-20 00:00:00, 2010-02-22 00:00:00, 1997-09-28 00:00:00, 1981-12-03 00:00:00,
1981-09-08 00:00:00, 1981-05-01 00:00:00
(3 rows)
```

聚集列是时间间隔类型。

```
vastbase=# SELECT deptno, listagg(vacationTime, '; ') WITHIN GROUP(ORDER BY vacationTime DESC)
AS vacationTime FROM emp GROUP BY deptno;
 deptno |                            vacationtime
--------+-------------------------------------------------------------------------------
---
    10 | 1 year 30 days; 40 days; 10 days
    20 | 70 days; 36 days; 9 days; 5 days
    30 | 1 year 1 mon; 2 mons 10 days; 30 days; 12 days 12:00:00; 4 days 06:00:00; 24:00:00
(3 rows)
```

分隔符缺省时，默认为空。

```
vastbase=# SELECT deptno, listagg(job) WITHIN GROUP(ORDER BY job) AS jobs FROM emp GROUP BY
deptno;
 deptno |                jobs
--------+---------------------------------------------
    10 | CLERKMANAGERPRESIDENT
    20 | ANALYSTANALYSTCLERKCLERKMANAGER
    30 | CLERKMANAGERSALESMANSALESMANSALESMANSALESMAN
(3 rows)
```

listagg 作为窗口函数时，OVER 子句不支持 ORDER BY 的窗口排序，listagg 列为对应分组的有序聚集。

```
vastbase=# SELECT deptno, mgrno, bonus, listagg(ename,'; ') WITHIN GROUP(ORDER BY hiredate)
OVER(PARTITION BY deptno) AS employees FROM emp;
 deptno | mgrno |  bonus   |              employees
--------+-------+----------+---------------------------------------
    10 |  7839 | 10000.01 | CLARK; KING; MILLER
    10 |       | 23011.88 | CLARK; KING; MILLER
    10 |  7782 | 10234.21 | CLARK; KING; MILLER
    20 |  7566 |  2002.12 | FORD; SCOTT; ADAMS; SMITH; JONES
    20 |  7566 |  1001.01 | FORD; SCOTT; ADAMS; SMITH; JONES
    20 |  7788 |  1100.00 | FORD; SCOTT; ADAMS; SMITH; JONES
    20 |  7902 |  2000.80 | FORD; SCOTT; ADAMS; SMITH; JONES
    20 |  7839 |   999.10 | FORD; SCOTT; ADAMS; SMITH; JONES
    30 |  7839 |  2399.50 | BLAKE; TURNER; JAMES; MARTIN; WARD; ALLEN
    30 |  7698 |     9.00 | BLAKE; TURNER; JAMES; MARTIN; WARD; ALLEN
    30 |  7698 |  1000.22 | BLAKE; TURNER; JAMES; MARTIN; WARD; ALLEN
    30 |  7698 |    99.99 | BLAKE; TURNER; JAMES; MARTIN; WARD; ALLEN
    30 |  7698 |  1000.01 | BLAKE; TURNER; JAMES; MARTIN; WARD; ALLEN
    30 |  7698 |   899.00 | BLAKE; TURNER; JAMES; MARTIN; WARD; ALLEN
(14 rows)
```

- covar_pop(Y, X)

  描述：总体协方差。

  返回类型：double precision

  示例：

```
vastbase=# SELECT COVAR_POP(sr_fee, sr_net_loss) FROM tpcds.store_returns WHERE sr_customer_sk
< 1000;
    covar_pop
------------------
 829.749627587403
(1 row)
```

- covar_samp(Y, X)

  描述：样本协方差。

  返回类型：double precision

  示例：

```
vastbase=# SELECT COVAR_SAMP(sr_fee, sr_net_loss) FROM tpcds.store_returns WHERE
sr_customer_sk < 1000;
   covar_samp
------------------
 830.052235037289
(1 row)
```

- stddev_pop(expression)

  描述：总体标准差。

  返回类型：对于浮点类型的输入返回 double precision，其他输入返回 numeric。

  示例：

```
vastbase=# SELECT STDDEV_POP(inv_quantity_on_hand) FROM tpcds.inventory WHERE
inv_warehouse_sk = 1;
   stddev_pop
------------------
 289.224294957556
(1 row)
```

- stddev_samp(expression)

  描述：样本标准差。

  返回类型：对于浮点类型的输入返回 double precision，其他输入返回 numeric。

  示例：

```
vastbase=# SELECT STDDEV_SAMP(inv_quantity_on_hand) FROM tpcds.inventory WHERE
inv_warehouse_sk = 1;
  stddev_samp
------------------
 289.224359757315
(1 row)
```

- var_pop(expression)

  描述：总体方差（总体标准差的平方）

  返回类型：对于浮点类型的输入返回 double precision 类型，其他输入返回 numeric 类型。

  示例：

```
vastbase=# SELECT VAR_POP(inv_quantity_on_hand) FROM tpcds.inventory WHERE inv_warehouse_sk
= 1;
     var_pop
--------------------
 83650.692793695475
(1 row)
```

- var_samp(expression)

描述：样本方差（样本标准差的平方）

返回类型：对于浮点类型的输入返回 double precision 类型，其他输入返回 numeric 类型。

示例：

```
vastbase=# SELECT VAR_SAMP(inv_quantity_on_hand) FROM tpcds.inventory WHERE inv_warehouse_sk
= 1;
     var_samp
--------------------
 83650.730277028768
(1 row)
```

- bit_and(expression)

描述：所有非 NULL 输入值的按位与(AND)，如果全部输入值皆为 NULL，那么结果也为 NULL 。

返回类型：和参数数据类型相同。

示例：

```
vastbase=# SELECT BIT_AND(inv_quantity_on_hand) FROM tpcds.inventory WHERE inv_warehouse_sk
= 1;
 bit_and
---------
       0
(1 row)
```

- bit_or(expression)

描述：所有非 NULL 输入值的按位或(OR)，如果全部输入值皆为 NULL，那么结果也为 NULL。

返回类型：和参数数据类型相同

示例：

```
vastbase=# SELECT BIT_OR(inv_quantity_on_hand) FROM tpcds.inventory WHERE inv_warehouse_sk =
1;
 bit_or
--------
   1023
(1 row)
```

- bool_and(expression)

描述：如果所有输入值都是真，则为真，否则为假。

返回类型：bool

示例：

```
vastbase=# SELECT bool_and(100 <2500);
 bool_and
----------
 t
(1 row)
```

- bool_or(expression)

描述：如果所有输入值只要有一个为真，则为真，否则为假。

返回类型：bool

示例：

```
vastbase=# SELECT bool_or(100 <2500);
 bool_or
----------
 t
(1 row)
```

- corr(Y, X)

描述：相关系数

返回类型：double precision

示例：

```
vastbase=# SELECT CORR(sr_fee, sr_net_loss) FROM tpcds.store_returns WHERE sr_customer_sk <
1000;
      corr
--------------------
 .0381383624904186
(1 row)
```

- every(expression)

描述：等效于 bool_and。

返回类型：bool

示例：

```
vastbase=# SELECT every(100 <2500);
 every
-------
 t
(1 row)
```

- rank(expression)

描述：根据 expression 对不同组内的元组进行跳跃排序。

返回类型：BIGINT

示例：

```
vastbase=# SELECT d_moy, d_fy_week_seq, rank() OVER(PARTITION BY d_moy ORDER BY d_fy_week_seq)
FROM tpcds.date_dim WHERE d_moy < 4 AND d_fy_week_seq < 7 ORDER BY 1,2;
  d_moy | d_fy_week_seq | rank
-------+---------------+------
    1 |             1 |    1
    1 |             1 |    1
    1 |             1 |    1
    1 |             1 |    1
    1 |             1 |    1
    1 |             1 |    1
    1 |             1 |    1
    1 |             2 |    8
    1 |             2 |    8
    1 |             2 |    8
    1 |             2 |    8
    1 |             2 |    8
```

```
    1 |           2 |   8
    1 |           2 |   8
    1 |           3 |  15
    1 |           3 |  15
    1 |           3 |  15
    1 |           3 |  15
    1 |           3 |  15
    1 |           3 |  15
    1 |           3 |  15
    1 |           4 |  22
    1 |           4 |  22
    1 |           4 |  22
    1 |           4 |  22
    1 |           4 |  22
    1 |           4 |  22
    1 |           4 |  22
    1 |           5 |  29
    1 |           5 |  29
    2 |           5 |   1
    2 |           5 |   1
    2 |           5 |   1
    2 |           5 |   1
    2 |           5 |   1
    2 |           6 |   6
    2 |           6 |   6
    2 |           6 |   6
    2 |           6 |   6
    2 |           6 |   6
    2 |           6 |   6
    2 |           6 |   6
(42 rows)
```

- regr_avgx(Y, X)

  描述：自变量的平均值（sum(X)/N)

  返回类型：double precision

  示例：

```
vastbase=# SELECT REGR_AVGX(sr_fee, sr_net_loss) FROM tpcds.store_returns WHERE sr_customer_sk
< 1000;
   regr_avgx
------------------
 578.606576740795
(1 row)
```

- regr_avgy(Y, X)

  描述：因变量的平均值（sum(Y)/N)

  返回类型：double precision

  示例：

```
vastbase=# SELECT REGR_AVGY(sr_fee, sr_net_loss) FROM tpcds.store_returns WHERE sr_customer_sk
< 1000;
   regr_avgy
------------------
```

```
 50.0136711629602
(1 row)
```

- regr_count(Y, X)

  描述：两个表达式都不为 NULL 的输入行数。

  返回类型：bigint

  示例：

```
vastbase=# SELECT REGR_COUNT(sr_fee, sr_net_loss) FROM tpcds.store_returns WHERE
sr_customer_sk < 1000;
 regr_count
------------
       2743
(1 row)
```

- regr_intercept(Y, X)

  描述：根据所有输入的点(X, Y)按照最小二乘法拟合成一个线性方程，然后返回该直线的 Y 轴截距。

  返回类型：double precision

  示例：

```
vastbase=# SELECT REGR_INTERCEPT(sr_fee, sr_net_loss) FROM tpcds.store_returns WHERE
sr_customer_sk < 1000;
  regr_intercept
------------------
 49.2040847848607
(1 row)
```

- regr_r2(Y, X)

  描述：相关系数的平方

  返回类型：double precision

  示例：

```
vastbase=# SELECT REGR_R2(sr_fee, sr_net_loss) FROM tpcds.store_returns WHERE sr_customer_sk
< 1000;
     regr_r2
--------------------
 .00145453469345058
(1 row)
```

- regr_slope(Y, X)

  描述：根据所有输入的点(X, Y)按照最小二乘法拟合成一个线性方程， 然后返回该直线的斜率。

  返回类型：double precision

  示例：

```
vastbase=# SELECT REGR_SLOPE(sr_fee, sr_net_loss) FROM tpcds.store_returns WHERE
sr_customer_sk < 1000;
    regr_slope
--------------------
```

```
 .00139920009665259
(1 row)
```

- regr_sxx(Y, X)

  描述：sum(X^2) - sum(X)^2/N （自变量的“平方和”）

  返回类型：double precision

  示例：

```
vastbase=# SELECT REGR_SXX(sr_fee, sr_net_loss) FROM tpcds.store_returns WHERE sr_customer_sk
< 1000;
    regr_sxx
------------------
 1626645991.46135
(1 row)
```

- regr_sxy(Y, X)

  描述：sum(X*Y) - sum(X) * sum(Y)/N （自变量和因变量的“乘方积”）

  返回类型：double precision

  示例：

```
vastbase=# SELECT REGR_SXY(sr_fee, sr_net_loss) FROM tpcds.store_returns WHERE sr_customer_sk
< 1000;
    regr_sxy
------------------
 2276003.22847225
(1 row)
```

- regr_syy(Y, X)

  描述：sum(Y^2) - sum(Y)^2/N（因变量的"平方和"）

  返回类型：double precision

  示例：

```
vastbase=# SELECT REGR_SYY(sr_fee, sr_net_loss) FROM tpcds.store_returns WHERE sr_customer_sk
< 1000;
   regr_syy
-----------------
 2189417.6547314
(1 row)
```

- stddev(expression)

  描述：stddev_samp 的别名。

  返回类型：对于浮点类型的输入返回 double precision，其他输入返回 numeric。

  示例：

```
vastbase=# SELECT STDDEV(inv_quantity_on_hand) FROM tpcds.inventory WHERE inv_warehouse_sk =
1;
     stddev
------------------
 289.224359757315
(1 row)
```

❖ variance(expexpression,ression)

描述：var_samp 的别名。

返回类型：对于浮点类型的输入返回 double precision 类型，其他输入返回 numeric 类型。

示例：

```
vastbase=# SELECT VARIANCE(inv_quantity_on_hand) FROM tpcds.inventory WHERE inv_warehouse_sk
= 1;
     variance
--------------------
 83650.730277028768
(1 row)
```

❖ checksum(expression)

描述：返回所有输入值的 CHECKSUM 值。使用该函数可以用来验证 Vastbase 数据库（不支持 Vastbase 之外的其他数据库）的备份恢复或者数据迁移操作前后表中的数据是否相同。在备份恢复或者数据迁移操作前后都需要用户通过手工执行 SQL 命令的方式获取执行结果，通过对比获取的执行结果判断操作前后表中的数据是否相同。

说明

- 对于大表，CHECKSUM 函数可能会需要很长时间。
- 如果某两表的 CHECKSUM 值不同，则表明两表的内容是不同的。由于 CHECKSUM 函数中使用散列函数不能保证无冲突，因此两个不同内容的表可能会得到相同的 CHECKSUM 值，存在这种情况的可能性较小。对于列进行的 CHECKSUM 也存在相同的情况。
- 对于时间类型 timestamp, timestamptz 和 smalldatetime，计算 CHECKSUM 值时请确保时区设置一致。
  - 若计算某列的 CHECKSUM 值，且该列类型可以默认转为 TEXT 类型，则 expression 为列名。
  - 若计算某列的 CHECKSUM 值，且该列类型不能默认转为 TEXT 类型，则 expression 为列名::TEXT。
  - 若计算所有列的 CHECKSUM 值，则 expression 为表名::TEXT。

可以默认转换为 TEXT 类型的类型包括：char, name, int8, int2, int1, int4, raw, pg_node_tree, float4, float8, bpchar, varchar, nvarchar2, date, timestamp, timestamptz, numeric, smalldatetime，其他类型需要强制转换为 TEXT。

返回类型：numeric。

示例：

表中可以默认转为 TEXT 类型的某列的 CHECKSUM 值。

```
vastbase=# SELECT CHECKSUM(inv_quantity_on_hand) FROM tpcds.inventory;
    checksum
-------------------
 24417258945265247
(1 row)
```

表中不能默认转为 TEXT 类型的某列的 CHECKSUM 值。注意此时 CHECKSUM 参数是列名::TEXT。

```
vastbase=# SELECT CHECKSUM(inv_quantity_on_hand::TEXT) FROM tpcds.inventory;
    checksum
--------------------
 24417258945265247
(1 row)
```

表中所有列的 CHECKSUM 值。注意此时 CHECKSUM 参数是表名::TEXT，且表名前不加 Schema。

```
vastbase=# SELECT CHECKSUM(inventory::TEXT) FROM tpcds.inventory;
    checksum
--------------------
 25223696246875800
(1 row)
```

11.5.18.窗口函数

窗口函数

列存表目前只支持 rank(expression)和 row_number(expression)两个函数。

窗口函数与 OVER 语句一起使用。OVER 语句用于对数据进行分组，并对组内元素进行排序。窗口函数用于给组内的值生成序号。

说明

窗口函数中的 order by 后面必须跟字段名，若 order by 后面跟数字，该数字会被按照常量处理，因此对目标列没有起到排序的作用。

- RANK()

描述：RANK 函数为各组内值生成跳跃排序序号，其中，相同的值具有相同序号。

返回值类型：BIGINT

示例：

```
vastbase=# SELECT d_moy, d_fy_week_seq, rank() OVER(PARTITION BY d_moy ORDER BY d_fy_week_seq)
FROM tpcds.date_dim WHERE d_moy < 4 AND d_fy_week_seq < 7 ORDER BY 1,2;
  d_moy | d_fy_week_seq | rank
-------+---------------+------
    1 |             1 |    1
    1 |             1 |    1
    1 |             1 |    1
    1 |             1 |    1
    1 |             1 |    1
    1 |             1 |    1
    1 |             1 |    1
    1 |             2 |    8
    1 |             2 |    8
    1 |             2 |    8
    1 |             2 |    8
```

```
    1 |           2 |    8
    1 |           2 |    8
    1 |           2 |    8
    1 |           3 |   15
    1 |           3 |   15
    1 |           3 |   15
    1 |           3 |   15
    1 |           3 |   15
    1 |           3 |   15
    1 |           3 |   15
    1 |           4 |   22
    1 |           4 |   22
    1 |           4 |   22
    1 |           4 |   22
    1 |           4 |   22
    1 |           4 |   22
    1 |           4 |   22
    1 |           5 |   29
    1 |           5 |   29
    2 |           5 |    1
    2 |           5 |    1
    2 |           5 |    1
    2 |           5 |    1
    2 |           5 |    1
    2 |           6 |    6
    2 |           6 |    6
    2 |           6 |    6
    2 |           6 |    6
    2 |           6 |    6
    2 |           6 |    6
    2 |           6 |    6
(42 rows)
```

- ROW_NUMBER()

描述：ROW_NUMBER 函数为各组内值生成连续排序序号，其中，相同的值其序号也不相同。

返回值类型：BIGINT

示例：

```
vastbase=# SELECT d_moy, d_fy_week_seq, Row_number() OVER(PARTITION BY d_moy ORDER BY
d_fy_week_seq) FROM tpcds.date_dim  WHERE d_moy < 4 AND d_fy_week_seq < 7 ORDER BY 1,2;
 d_moy | d_fy_week_seq | row_number
-------+---------------+------------
    1 |           1 |         1
    1 |           1 |         2
    1 |           1 |         3
    1 |           1 |         4
    1 |           1 |         5
    1 |           1 |         6
    1 |           1 |         7
    1 |           2 |         8
    1 |           2 |         9
    1 |           2 |        10
    1 |           2 |        11
    1 |           2 |        12
```

```
    1 |           2 |        13
    1 |           2 |        14
    1 |           3 |        15
    1 |           3 |        16
    1 |           3 |        17
    1 |           3 |        18
    1 |           3 |        19
    1 |           3 |        20
    1 |           3 |        21
    1 |           4 |        22
    1 |           4 |        23
    1 |           4 |        24
    1 |           4 |        25
    1 |           4 |        26
    1 |           4 |        27
    1 |           4 |        28
    1 |           5 |        29
    1 |           5 |        30
    2 |           5 |         1
    2 |           5 |         2
    2 |           5 |         3
    2 |           5 |         4
    2 |           5 |         5
    2 |           6 |         6
    2 |           6 |         7
    2 |           6 |         8
    2 |           6 |         9
    2 |           6 |        10
    2 |           6 |        11
    2 |           6 |        12
(42 rows)
```

- DENSE_RANK()

描述：DENSE_RANK 函数为各组内值生成连续排序序号，其中，相同的值具有相同序号。

返回值类型：BIGINT

示例：

```
vastbase=# SELECT d_moy, d_fy_week_seq, dense_rank() OVER(PARTITION BY d_moy ORDER BY
d_fy_week_seq) FROM tpcds.date_dim WHERE d_moy < 4 AND d_fy_week_seq < 7 ORDER BY 1,2;
 d_moy | d_fy_week_seq | dense_rank
-------+---------------+------------
    1 |           1 |         1
    1 |           1 |         1
    1 |           1 |         1
    1 |           1 |         1
    1 |           1 |         1
    1 |           1 |         1
    1 |           1 |         1
    1 |           2 |         2
    1 |           2 |         2
    1 |           2 |         2
    1 |           2 |         2
    1 |           2 |         2
    1 |           2 |         2
```

```
    1 |           2 |         2
    1 |           3 |         3
    1 |           3 |         3
    1 |           3 |         3
    1 |           3 |         3
    1 |           3 |         3
    1 |           3 |         3
    1 |           3 |         3
    1 |           4 |         4
    1 |           4 |         4
    1 |           4 |         4
    1 |           4 |         4
    1 |           4 |         4
    1 |           4 |         4
    1 |           4 |         4
    1 |           5 |         5
    1 |           5 |         5
    2 |           5 |         1
    2 |           5 |         1
    2 |           5 |         1
    2 |           5 |         1
    2 |           5 |         1
    2 |           6 |         2
    2 |           6 |         2
    2 |           6 |         2
    2 |           6 |         2
    2 |           6 |         2
    2 |           6 |         2
    2 |           6 |         2
(42 rows)
```

- PERCENT_RANK()

描述：PERCENT_RANK 函数为各组内对应值生成相对序号，即根据公式 (rank - 1) / (total rows - 1)计算所得的值。其中 rank 为该值依据 RANK 函数所生成的对应序号，totalrows 为该分组内的总元素个数。

返回值类型：DOUBLE PRECISION

示例：

```
vastbase=# SELECT d_moy, d_fy_week_seq, percent_rank() OVER(PARTITION BY d_moy ORDER BY
d_fy_week_seq) FROM tpcds.date_dim WHERE d_moy < 4 AND d_fy_week_seq < 7 ORDER BY 1,2;
 d_moy | d_fy_week_seq |   percent_rank
-------+---------------+------------------
    1 |           1 |              0
    1 |           1 |              0
    1 |           1 |              0
    1 |           1 |              0
    1 |           1 |              0
    1 |           1 |              0
    1 |           1 |              0
    1 |           2 | .241379310344828
    1 |           2 | .241379310344828
    1 |           2 | .241379310344828
    1 |           2 | .241379310344828
```

```
    1 |           2 | .241379310344828
    1 |           2 | .241379310344828
    1 |           2 | .241379310344828
    1 |           3 | .482758620689655
    1 |           3 | .482758620689655
    1 |           3 | .482758620689655
    1 |           3 | .482758620689655
    1 |           3 | .482758620689655
    1 |           3 | .482758620689655
    1 |           3 | .482758620689655
    1 |           4 | .724137931034483
    1 |           4 | .724137931034483
    1 |           4 | .724137931034483
    1 |           4 | .724137931034483
    1 |           4 | .724137931034483
    1 |           4 | .724137931034483
    1 |           4 | .724137931034483
    1 |           5 |  .96551724137931
    1 |           5 |  .96551724137931
    2 |           5 |             0
    2 |           5 |             0
    2 |           5 |             0
    2 |           5 |             0
    2 |           5 |             0
    2 |           6 | .454545454545455
    2 |           6 | .454545454545455
    2 |           6 | .454545454545455
    2 |           6 | .454545454545455
    2 |           6 | .454545454545455
    2 |           6 | .454545454545455
    2 |           6 | .454545454545455
(42 rows)
```

❖ CUME_DIST()

描述：CUME_DIST 函数为各组内对应值生成累积分布序号。即根据公式(小于等于当前值的数据行数)/(该分组总行数 totalrows)计算所得的相对序号。

返回值类型：DOUBLE PRECISION

示例：

```
vastbase=# SELECT d_moy, d_fy_week_seq, cume_dist() OVER(PARTITION BY d_moy ORDER BY
d_fy_week_seq) FROM tpcds.date_dim e_dim WHERE d_moy < 4 AND d_fy_week_seq < 7 ORDER BY 1,2;
 d_moy | d_fy_week_seq |    cume_dist
-------+---------------+-----------------
    1 |           1 | .233333333333333
    1 |           1 | .233333333333333
    1 |           1 | .233333333333333
    1 |           1 | .233333333333333
    1 |           1 | .233333333333333
    1 |           1 | .233333333333333
    1 |           1 | .233333333333333
    1 |           2 | .466666666666667
    1 |           2 | .466666666666667
    1 |           2 | .466666666666667
    1 |           2 | .466666666666667
```

```
    1 |           2 | .466666666666667
    1 |           2 | .466666666666667
    1 |           2 | .466666666666667
    1 |           3 |              .7
    1 |           3 |              .7
    1 |           3 |              .7
    1 |           3 |              .7
    1 |           3 |              .7
    1 |           3 |              .7
    1 |           3 |              .7
    1 |           4 | .933333333333333
    1 |           4 | .933333333333333
    1 |           4 | .933333333333333
    1 |           4 | .933333333333333
    1 |           4 | .933333333333333
    1 |           4 | .933333333333333
    1 |           4 | .933333333333333
    1 |           5 |               1
    1 |           5 |               1
    2 |           5 | .416666666666667
    2 |           5 | .416666666666667
    2 |           5 | .416666666666667
    2 |           5 | .416666666666667
    2 |           5 | .416666666666667
    2 |           6 |               1
    2 |           6 |               1
    2 |           6 |               1
    2 |           6 |               1
    2 |           6 |               1
    2 |           6 |               1
    2 |           6 |               1
(42 rows)
```

- NTILE(num_buckets integer)

  描述：NTILE 函数根据 num_buckets integer 将有序的数据集合平均分配到 num_buckets 所指定数量的桶中，并将桶号分配给每一行。分配时应尽量做到平均分配。

  返回值类型：INTEGER

  示例：

```
vastbase=# SELECT d_moy, d_fy_week_seq, ntile(3) OVER(PARTITION BY d_moy ORDER BY d_fy_week_seq)
FROM tpcds.date_dim WHERE d_moy < 4 AND d_fy_week_seq < 7 ORDER BY 1,2;
 d_moy | d_fy_week_seq | ntile
-------+---------------+-------
     1 |             1 |     1
     1 |             1 |     1
     1 |             1 |     1
     1 |             1 |     1
     1 |             1 |     1
     1 |             1 |     1
     1 |             1 |     1
     1 |             2 |     1
     1 |             2 |     1
     1 |             2 |     1
     1 |             2 |     2
```

```
    1 |           2 |     2
    1 |           2 |     2
    1 |           2 |     2
    1 |           3 |     2
    1 |           3 |     2
    1 |           3 |     2
    1 |           3 |     2
    1 |           3 |     2
    1 |           3 |     2
    1 |           3 |     3
    1 |           4 |     3
    1 |           4 |     3
    1 |           4 |     3
    1 |           4 |     3
    1 |           4 |     3
    1 |           4 |     3
    1 |           4 |     3
    1 |           5 |     3
    1 |           5 |     3
    2 |           5 |     1
    2 |           5 |     1
    2 |           5 |     1
    2 |           5 |     1
    2 |           5 |     2
    2 |           6 |     2
    2 |           6 |     2
    2 |           6 |     2
    2 |           6 |     3
    2 |           6 |     3
    2 |           6 |     3
    2 |           6 |     3
(42 rows)
```

- LAG(value any [, offset integer [, default any]])

  描述：LAG 函数为各组内对应值生成滞后值。即当前值对应的行数往前偏移 offset 位后所得行的 value 值作为序号。若经过偏移后行数不存在，则对应结果取为 default 值。若无指定，在默认情况下，offset 取为 1，default 值取为 NULL。

  返回值类型：与参数数据类型相同

  示例：

```
vastbase=# SELECT d_moy, d_fy_week_seq, lag(d_moy,3,null) OVER(PARTITION BY d_moy ORDER BY
d_fy_week_seq) FROM tpcds.date_dim WHERE d_moy < 4 AND d_fy_week_seq < 7 ORDER BY 1,2;
 d_moy | d_fy_week_seq | lag
-------+---------------+-----
    1 |           1 |
    1 |           1 |
    1 |           1 |
    1 |           1 |   1
    1 |           1 |   1
    1 |           1 |   1
    1 |           1 |   1
    1 |           2 |   1
    1 |           2 |   1
```

```
    1 |           2 |   1
    1 |           2 |   1
    1 |           2 |   1
    1 |           2 |   1
    1 |           2 |   1
    1 |           3 |   1
    1 |           3 |   1
    1 |           3 |   1
    1 |           3 |   1
    1 |           3 |   1
    1 |           3 |   1
    1 |           3 |   1
    1 |           4 |   1
    1 |           4 |   1
    1 |           4 |   1
    1 |           4 |   1
    1 |           4 |   1
    1 |           4 |   1
    1 |           4 |   1
    1 |           5 |   1
    1 |           5 |   1
    2 |           5 |
    2 |           5 |
    2 |           5 |
    2 |           5 |   2
    2 |           5 |   2
    2 |           6 |   2
    2 |           6 |   2
    2 |           6 |   2
    2 |           6 |   2
    2 |           6 |   2
    2 |           6 |   2
    2 |           6 |   2
(42 rows)
```

❖ LEAD(value any [, offset integer [, default any]])

描述：LEAD 函数为各组内对应值生成提前值。即当前值对应的行数向后偏移 offset 位后所得行的 value 值作为序号。若经过向后偏移后行数超过当前组内的总行数，则对应结果取为 default 值。若无指定，在默认情况下，offset 取为 1，default 值取为 NULL。

返回值类型：与参数数据类型相同。

示例：

```
vastbase=# SELECT d_moy, d_fy_week_seq, lead(d_fy_week_seq,2) OVER(PARTITION BY d_moy ORDER
BY d_fy_week_seq) FROM  tpcds.date_dim WHERE d_moy < 4 AND d_fy_week_seq < 7 ORDER BY 1,2;
d_moy | d_fy_week_seq | lead
-------+---------------+------
    1 |           1 |    1
    1 |           1 |    1
    1 |           1 |    1
    1 |           1 |    1
    1 |           1 |    1
    1 |           1 |    2
    1 |           1 |    2
```

```
    1 |           2 |    2
    1 |           2 |    2
    1 |           2 |    2
    1 |           2 |    2
    1 |           2 |    2
    1 |           2 |    3
    1 |           2 |    3
    1 |           3 |    3
    1 |           3 |    3
    1 |           3 |    3
    1 |           3 |    3
    1 |           3 |    3
    1 |           3 |    4
    1 |           3 |    4
    1 |           4 |    4
    1 |           4 |    4
    1 |           4 |    4
    1 |           4 |    4
    1 |           4 |    4
    1 |           4 |    5
    1 |           4 |    5
    1 |           5 |
    1 |           5 |
    2 |           5 |    5
    2 |           5 |    5
    2 |           5 |    5
    2 |           5 |    6
    2 |           5 |    6
    2 |           6 |    6
    2 |           6 |    6
    2 |           6 |    6
    2 |           6 |    6
    2 |           6 |    6
    2 |           6 |
    2 |           6 |
(42 rows)
```

- FIRST_VALUE(value any)

描述：FIRST_VALUE 函数取各组内的第一个值作为返回结果。

返回值类型：与参数数据类型相同。

示例：

```
vastbase=# SELECT d_moy, d_fy_week_seq, first_value(d_fy_week_seq) OVER(PARTITION BY d_moy
ORDER BY d_fy_week_seq) FROM tpcds.date_dim WHERE d_moy < 4 AND d_fy_week_seq < 7 ORDER BY 1,2;
 d_moy | d_fy_week_seq | first_value
-------+---------------+-------------
    1 |           1 |         1
    1 |           1 |         1
    1 |           1 |         1
    1 |           1 |         1
    1 |           1 |         1
    1 |           1 |         1
    1 |           1 |         1
    1 |           2 |         1
```

```
    1 |           2 |         1
    1 |           2 |         1
    1 |           2 |         1
    1 |           2 |         1
    1 |           2 |         1
    1 |           2 |         1
    1 |           3 |         1
    1 |           3 |         1
    1 |           3 |         1
    1 |           3 |         1
    1 |           3 |         1
    1 |           3 |         1
    1 |           3 |         1
    1 |           4 |         1
    1 |           4 |         1
    1 |           4 |         1
    1 |           4 |         1
    1 |           4 |         1
    1 |           4 |         1
    1 |           4 |         1
    1 |           5 |         1
    1 |           5 |         1
    2 |           5 |         5
    2 |           5 |         5
    2 |           5 |         5
    2 |           5 |         5
    2 |           5 |         5
    2 |           6 |         5
    2 |           6 |         5
    2 |           6 |         5
    2 |           6 |         5
    2 |           6 |         5
    2 |           6 |         5
    2 |           6 |         5
(42 rows)
```

❖ LAST_VALUE(value any)

描述：LAST_VALUE 函数取各组内的最后一个值作为返回结果。

返回值类型：与参数数据类型相同。

示例：

```
vastbase=# SELECT d_moy, d_fy_week_seq, last_value(d_moy) OVER(PARTITION BY d_moy ORDER BY
d_fy_week_seq) FROM tpcds.date_dim WHERE d_moy < 4 AND d_fy_week_seq < 6 ORDER BY 1,2;
 d_moy | d_fy_week_seq | last_value
-------+---------------+------------
    1 |           1 |         1
    1 |           1 |         1
    1 |           1 |         1
    1 |           1 |         1
    1 |           1 |         1
    1 |           1 |         1
    1 |           1 |         1
    1 |           2 |         1
    1 |           2 |         1
```

```
    1 |           2 |         1
    1 |           2 |         1
    1 |           2 |         1
    1 |           2 |         1
    1 |           2 |         1
    1 |           2 |         1
    1 |           3 |         1
    1 |           3 |         1
    1 |           3 |         1
    1 |           3 |         1
    1 |           3 |         1
    1 |           3 |         1
    1 |           3 |         1
    1 |           4 |         1
    1 |           4 |         1
    1 |           4 |         1
    1 |           4 |         1
    1 |           4 |         1
    1 |           4 |         1
    1 |           4 |         1
    1 |           5 |         1
    1 |           5 |         1
    2 |           5 |         2
    2 |           5 |         2
    2 |           5 |         2
    2 |           5 |         2
    2 |           5 |         2
(35 rows)
```

❖ NTH_VALUE(value any, nth integer)

描述：NTH_VALUE 函数返回该组内的第 nth 行作为结果。若该行不存在，则默认返回 NULL。

返回值类型：与参数数据类型相同。

示例：

```
vastbase=# SELECT d_moy, d_fy_week_seq, nth_value(d_fy_week_seq,6) OVER(PARTITION BY d_moy
ORDER BY d_fy_week_seq) FROM tpcds.date_dim WHERE d_moy < 4 AND d_fy_week_seq < 6 ORDER BY 1,2;
 d_moy | d_fy_week_seq | nth_value
-------+---------------+-----------
    1 |           1 |         1
    1 |           1 |         1
    1 |           1 |         1
    1 |           1 |         1
    1 |           1 |         1
    1 |           1 |         1
    1 |           1 |         1
    1 |           2 |         1
    1 |           2 |         1
    1 |           2 |         1
    1 |           2 |         1
    1 |           2 |         1
    1 |           2 |         1
    1 |           2 |         1
    1 |           3 |         1
    1 |           3 |         1
```

```
    1 |           3 |        1
    1 |           3 |        1
    1 |           3 |        1
    1 |           3 |        1
    1 |           3 |        1
    1 |           4 |        1
    1 |           4 |        1
    1 |           4 |        1
    1 |           4 |        1
    1 |           4 |        1
    1 |           4 |        1
    1 |           4 |        1
    1 |           5 |        1
    1 |           5 |        1
    2 |           5 |
    2 |           5 |
    2 |           5 |
    2 |           5 |
    2 |           5 |
(35 rows)
```

11.5.19.安全函数

安全函数

- gs_encrypt_aes128(encryptstr,keystr)

  描述：以 keystr 为密钥对 encryptstr 字符串进行加密，返回加密后的字符串。keystr 的长度范围为 1~16 字节。支持的加密数据类型：目前数据库支持的数值类型，字符类型，二进制类型中的 RAW，日期/时间类型中的 DATE、TIMESTAMP、SMALLDATETIME。

  返回值类型：text

  返回值长度：至少为 92 字节，不超过 4*[(Len+68)/3] 字节，其中 Len 为加密前数据长度（单位为字节）。

  示例：

```
vastbase=# SELECT gs_encrypt_aes128('MPPDB','1234');

                          gs_encrypt_aes128
--------------------------------------------------------------------------------
gwditQLQG8NhFw4OuoKhhQJoXojhFlYkjeG0aYdSCtLCnIUgkNwvYI04KbuhmcGZp8jWizBdR1vU9CspjuzI0lbz12
A=
(1 row)
```

说明

由于该函数的执行过程需要传入解密口令，为了安全起见，vsql 工具不会将该函数记录入执行历史。即无法在 vsql 里通过上下翻页功能找到该函数的执行历史。

- gs_decrypt_aes128(decryptstr,keystr)

描述：以 keystr 为密钥对 decrypt 字符串进行解密，返回解密后的字符串。解密使用的 keystr 必须保证与加密时使用的 keystr 一致才能正常解密。keystr 不得为空。

说明

此参数需要结合 gs_encrypt_aes128 加密函数共同使用。

返回值类型：text

示例：

```
vastbase=# SELECT
gs_decrypt_aes128('gwditQLQG8NhFw4OuoKhhQJoXojhFlYkjeG0aYdSCtLCnIUgkNwvYI04KbuhmcGZp8jWizB
dR1vU9CspjuzI0lbz12A=','1234');
 gs_decrypt_aes128
-------------------
 MPPDB
(1 row)
```

说明

由于该函数的执行过程需要传入解密口令，为了安全起见，vsql 工具不会将该函数记录入执行历史；即无法在 vsql 里通过上下翻页功能找到该函数的执行历史。

- gs_password_deadline

描述：显示当前帐户密码离过期还距离多少天。

返回值类型：interval

示例：

```
vastbase=# SELECT gs_password_deadline();
  gs_password_deadline
-------------------------
 83 days 17:44:32.196094
(1 row)
```

- login_audit_messages

描述：查看登录用户的登录信息。

返回值类型：元组

示例：

– 查看上一次登录认证通过的日期、时间和 IP 等信息。

```
vastbase=# SELECT * FROM login_audit_messages(true);
  username  | database |      logintime       |     mytype     | result |  client_conninfo
------------+----------+----------------------+----------------+--------+-----------
---------
    vastbase    | vastbase | 2017-06-02 15:28:34+08 | login_success | ok     | vsql@[local]
(1 row)
```

– 查看上一次登录认证失败的日期、时间和 IP 等信息。

```
vastbase=# SELECT * FROM login_audit_messages(false) ORDER BY logintime desc limit 1;
  username  | database |      logintime       |     mytype     | result |
client_conninfo
------------+----------+----------------------+----------------+--------+------------
```

```
-------------
(0 rows)
```

- 查看自从最后一次认证通过以来失败的尝试次数、日期和时间。

```
vastbase=# SELECT * FROM login_audit_messages(false);
 username  | database |      logintime       |    mytype     | result |
client_conninfo
-----------+----------+----------------------+---------------+--------+-----------
-------------
(0 rows)
```

❖ login_audit_messages_pid

描述：查看登录用户的登录信息。与 login_audit_messages 的区别在于结果基于当前 backendid 向前查找。所以不会因为同一用户的后续登录，而影响本次登录的查询结果。也就是查询不到该用户后续登录的信息。

返回值类型：元组

示例：

- 查看上一次登录认证通过的日期、时间和 IP 等信息。

```
vastbase=# SELECT * FROM login_audit_messages(true);
 username  | database |      logintime       |    mytype     | result |  client_conninfo
-----------+----------+----------------------+---------------+--------+--------
    vastbase    | postgres | 2017-06-02 15:28:34+08 | login_success | ok     | vsql@[local]
(1 row)
```

- 查看上一次登录认证失败的日期、时间和 IP 等信息。

```
vastbase=# SELECT * FROM login audit messages(false) ORDER BY logintime desc limit 1;
 username  | database |      logintime       |    mytype     | result |    client conninfo
-----------+----------+----------------------+---------------+--------+--------
(0 rows)
```

- 查看自从最后一次认证通过以来失败的尝试次数、日期和时间。

```
vastbase=# SELECT * FROM login_audit_messages(false);
 username  | database |      logintime       |    mytype     | result |    client_conninfo
-----------+----------+----------------------+---------------+--------+--------
(0 rows)
```

❖ inet_server_addr

描述：显示服务器 IP 信息。

返回值类型：inet

示例：

```
vastbase=# SELECT inet_server_addr();
 inet_server_addr
------------------
 10.10.0.13
(1 row)
```

说明

- 上面是以客户端在 10.10.0.50 上，服务器端在 10.10.0.13 上为例。
- 如果是通过本地连接，使用此接口显示为空。

❖ inet_client_addr

描述：显示客户端 IP 信息。

返回值类型：inet

示例：

```
vastbase=# SELECT inet_client_addr();
 inet_client_addr
------------------
 10.10.0.50
(1 row)
```

说明

- 上面是以客户端在 10.10.0.50 上，服务器端在 10.10.0.13 上为例。
- 如果是通过本地连接，使用此接口显示为空。

❖ pg_query_audit

描述：查看数据库主节点审计日志。

返回值类型：record

函数返回字段如下：

名称	类型	描述
time	timestamp with time zone	操作时间
type	text	操作类型
result	text	操作结果
username	text	执行操作的用户名
database	text	数据库名称
client_conninfo	text	客户端连接信息
object_name	text	操作对象名称
detail_info	text	执行操作详细信息
node_name	text	节点名称
thread_id	text	线程 id
local_port	text	本地端口
remote_port	text	远端端口

函数使用方法及示例请参考 5.3.2 查看审计结果。

- pgxc_query_audit

  描述：查看数据库主节点审计日志。

  返回值类型：record

  函数返回字段同 pg_query_audit 函数。

  函数使用方法及示例请参考 5.3.2 查看审计结果。

- pg_delete_audit

描述：删除指定时间段的审计日志。 返回值类型：void 函数使用方法及示例请参考 5.3.3 维护审计日志。

11.5.20.返回集合的函数

序列号生成函数

- generate_series(start, stop)

  描述：生成一个数值序列，从 start 到 stop，步长为 1。

  参数类型：int、bigint、numeric

  返回值类型：setof int、setof bigint、setof numeric（与参数类型相同）

- generate_series(start, stop, step)

  描述：生成一个数值序列，从 start 到 stop，步长为 step。

  参数类型：int、bigint、numeric

  返回值类型：setof int、setof bigint、setof numeric（与参数类型相同）

- generate_series(start, stop, step interval)

  描述：生成一个数值序列，从 start 到 stop，步长为 step。

  参数类型：timestamp 或 timestamp with time zone

  返回值类型：setof timestamp 或 setof timestamp with time zone（与参数类型相同）

如果 step 是正数且 start 大于 stop，则返回零行。相反，如果 step 是负数且 start 小于 stop，则也返回零行。如果输入是 NULL，同样产生零行。如果 step 为零则是一个错误。

示例：

```
vastbase=# SELECT * FROM generate_series(2,4);
 generate_series
-----------------
               2
               3
               4
(3 rows)

vastbase=# SELECT * FROM generate_series(5,1,-2);
```

```
 generate_series
-----------------
               5
               3
               1
(3 rows)

vastbase=# SELECT * FROM generate_series(4,3);
 generate_series
-----------------
(0 rows)

--这个示例应用于 date-plus-integer 操作符。
vastbase=# SELECT current_date + s.a AS dates FROM generate_series(0,14,7) AS s(a);
   dates
------------
 2017-06-02
 2017-06-09
 2017-06-16
(3 rows)

vastbase=# SELECT * FROM generate_series('2008-03-01 00:00'::timestamp, '2008-03-04 12:00', '10
hours');
   generate_series
---------------------
 2008-03-01 00:00:00
 2008-03-01 10:00:00
 2008-03-01 20:00:00
 2008-03-02 06:00:00
 2008-03-02 16:00:00
 2008-03-03 02:00:00
 2008-03-03 12:00:00
 2008-03-03 22:00:00
 2008-03-04 08:00:00
(9 rows)
```

下标生成函数

- generate_subscripts(array anyarray, dim int)

  描述：生成一系列包括给定数组的下标。

  返回值类型：setof int

- generate_subscripts(array anyarray, dim int, reverse boolean)

  描述：生成一系列包括给定数组的下标。当 reverse 为真时，该系列则以相反的顺序返回。

  返回值类型：setof int

generate_subscripts 是一个为给定数组中的指定维度生成有效下标集的函数。如果数组中没有所请求的维度或者 NULL 数组，返回零行（但是会给数组元素为空的返回有效下标）。示例：

```
--基本用法。
vastbase=# SELECT generate_subscripts('{NULL,1,NULL,2}'::int[], 1) AS s;
 s
```

```
---
 1
 2
 3
 4
(4 rows)
--unnest 一个 2D 数组。
vastbase=# CREATE OR REPLACE FUNCTION unnest2(anyarray)
RETURNS SETOF anyelement AS $$
SELECT $1[i][j]
   FROM generate_subscripts($1,1) g1(i),
        generate_subscripts($1,2) g2(j);
$$ LANGUAGE sql IMMUTABLE;

vastbase=# SELECT * FROM unnest2(ARRAY[[1,2],[3,4]]);
 unnest2
---------
       1
       2
       3
       4
(4 rows)

--删除函数。
vastbase=# DROP FUNCTION unnest2;
```

11.5.21.条件表达式函数

条件表达式函数

- coalesce(expr1, expr2, ..., exprn)

  描述：

  返回参数列表中第一个非 NULL 的参数值。

  COALESCE(expr1, expr2) 等价于 CASE WHEN expr1 IS NOT NULL THEN expr1 ELSE expr2 END。

  示例：

```
vastbase=# SELECT coalesce(NULL,'hello');
 coalesce
----------
 hello
(1 row)
```

  备注：

  - 如果表达式列表中的所有表达式都等于 NULL，则本函数返回 NULL。
  - 它常用于在显示数据时用缺省值替换 NULL。
  - 和 CASE 表达式一样，COALESCE 不会计算不需要用来判断结果的参数；即在第一个非空参数右边的参数不会被计算。

- decode(base_expr, compare1, value1, Compare2,value2, ... default)

  描述：把 base_expr 与后面的每个 compare(n) 进行比较，如果匹配返回相应的 value(n)。如果没有发生匹配，则返回 default。

  示例：

```
vastbase=# SELECT decode('A','A',1,'B',2,0);
 case
------
 1
(1 row)
```

- nullif(expr1, expr2)

  描述：当且仅当 expr1 和 expr2 相等时，NULLIF 才返回 NULL，否则它返回 expr1。

  nullif(expr1, expr2) 逻辑上等价于 CASE WHEN expr1 = expr2 THEN NULL ELSE expr1 END。

  示例：

```
vastbase=# SELECT nullif('hello','world');
 nullif
--------
 hello
(1 row)
```

  备注：

  如果两个参数的数据类型不同，则：

  - 若两种数据类型之间存在隐式转换，则以其中优先级较高的数据类型为基准将另一个参数隐式转换成该类型，转换成功则进行计算，转换失败则返回错误。如：

```
vastbase=# SELECT nullif('1234'::VARCHAR,123::INT4);
 nullif
--------
   1234
(1 row)
vastbase=# SELECT nullif('1234'::VARCHAR,'2012-12-24'::DATE);
ERROR:  invalid input syntax for type timestamp: "1234"
```

  - 若两种数据类型之间不存在隐式转换，则返回错误 。如：

```
vastbase=# SELECT nullif(TRUE::BOOLEAN,'2012-12-24'::DATE);
ERROR:  operator does not exist: boolean = timestamp without time zone
LINE 1: SELECT nullif(TRUE::BOOLEAN,'2012-12-24'::DATE) FROM DUAL;
^
HINT:  No operator matches the given name and argument type(s). You might need to add explicit type casts.
```

- nvl(expr1 , expr2)

  描述：

  - 如果 expr1 为 NULL 则返回 expr2。
  - 如果 expr1 非 NULL，则返回 expr1。

  示例：

```
vastbase=# SELECT nvl('hello','world');
 nvl
-------
 hello
(1 row)
```

备注：参数 expr1 和 expr2 可以为任意类型，当 NVL 的两个参数不属于同类型时，看第二个参数是否可以向第一个参数进行隐式转换，如果可以则返回第一个参数类型。如果第二个参数不能向第一个参数进行隐式转换而第一个参数可以向第二个参数进行隐式转换，则返回第二个参数的类型。如果两个参数之间不存在隐式类型转换并且也不属于同一类型则报错。

- greatest(expr1 [, ...])

描述：获取并返回参数列表中值最大的表达式的值。

返回值类型：

示例：

```
vastbase=# SELECT greatest(1*2,2-3,4-1);
 greatest
----------
        3
(1 row)
vastbase=# SELECT greatest('HARRY', 'HARRIOT', 'HAROLD');
 greatest
----------
 HARRY
(1 row)
```

- least(expr1 [, ...])

描述：获取并返回参数列表中值最小的表达式的值。

示例：

```
vastbase=# SELECT least(1*2,2-3,4-1);
 least
-------
    -1
(1 row)
vastbase=# SELECT least('HARRY','HARRIOT','HAROLD');
 least
--------
 HAROLD
(1 row)
```

- EMPTY_BLOB()

描述：使用 EMPTY_BLOB 在 INSERT 或 UPDATE 语句中初始化一个 BLOB 变量，取值为 NULL。

返回值类型：BLOB

示例：

```
--新建表
vastbase=# CREATE TABLE blob_tb(b blob,id int);
--插入数据
```

```
vastbase=# INSERT INTO blob_tb VALUES (empty_blob(),1);
--删除表
vastbase=# DROP TABLE blob_tb;
```

11.5.22.系统信息函数

会话信息函数

- current_catalog

  描述：当前数据库的名称（在标准 SQL 中称"catalog"）。

  返回值类型：name

  示例：

```
vastbase=# SELECT current_catalog;
 current_database
------------------
 vastbase
(1 row)
```

- current_database()

  描述：当前数据库的名称。

  返回值类型：name

  示例：

```
vastbase=# SELECT current_database();
 current_database
------------------
 vastbase
(1 row)
```

- current_query()

  描述：由客户端提交的当前执行语句（可能包含多个声明）。

  返回值类型：text

  示例：

```
vastbase=# SELECT current_query();
     current_query
-------------------------
 SELECT current_query();
(1 row)
```

- current_schema[()]

  描述：当前模式的名称。

  返回值类型：name

  示例：

```
vastbase=# SELECT current_schema();
 current_schema
```

```
----------------
 public
(1 row)
```

备注：current_schema 返回在搜索路径中第一个顺位有效的模式名。（如果搜索路径为空则返回 NULL，没有有效的模式名也返回 NULL）。如果创建表或者其他命名对象时没有声明目标模式，则将使用这些对象的模式。

- current_schemas(Boolean)

描述：搜索路径中的模式名称。

返回值类型：name[]

示例：

```
vastbase=# SELECT current_schemas(true);
  current_schemas
---------------------
 {pg_catalog,public}
(1 row)
```

备注：

current_schemas(Boolean)返回搜索路径中所有模式名称的数组。布尔选项决定像 pg_catalog 这样隐含包含的系统模式是否包含在返回的搜索路径中。

说明

搜索路径可以通过运行时设置更改。命令是：

SET search_path TO schema [, schema, ...]

- current_user

描述：当前执行环境下的用户名。

返回值类型：name

示例：

```
vastbase=# SELECT current_user;
 current_user
--------------
 vastbase
(1 row)
```

备注：current_user 是用于权限检查的用户标识。通常，他表示会话用户，但是可以通过 11.16.107SET ROLE 改变他。在函数执行的过程中随着属性 SECURITY DEFINER 的改变，其值也会改变。

- pg_current_sessionid()

描述：当前执行环境下的会话 ID。

返回值类型：text

示例：

```
vastbase=# SELECT pg_current_sessionid();
   pg_current_sessionid
----------------------------
 1579228402.140190434944768
(1 row)
```

备注：pg_current_sessionid()是用于获取当前执行环境下的会话 ID。其组成结构为：时间戳.会话 ID，当线程池模式开启（enable_thread_pool=on）时，会话 ID 为 SessionID；而线程池模式关闭时，会话 ID 为 ThreadID。

- inet_client_addr()

描述：连接的远端地址。inet_client_addr 返回当前客户端的 IP 地址。

说明

此函数只有在远程连接模式下有效。

返回值类型：inet

示例：

```
vastbase=# SELECT inet_client_addr();
 inet_client_addr
------------------
 10.10.0.50
(1 row)
```

- inet_client_port()

描述：连接的远端端口。inet_client_port 返回当前客户端的端口号。

说明

此函数只有在远程连接模式下有效。

返回值类型：int

示例：

```
vastbase=# SELECT inet_client_port();
 inet_client_port
------------------
            33143
(1 row)
```

- inet_server_addr()

描述：连接的本地地址。inet_server_addr 返回服务器接收当前连接用的 IP 地址。

说明

此函数只有在远程连接模式下有效。

返回值类型：inet

示例：

```
vastbase=# SELECT inet server addr();
 inet server addr
------------------
```

```
 10.10.0.13
(1 row)
```

- inet_server_port()

  描述：连接的本地端口。inet_server_port 返回接收当前连接的端口号。如果是通过 Unix-domain socket 连接的，则所有这些函数都返回 NULL。

说明

此函数只有在远程连接模式下有效。

返回值类型：int

示例：

```
vastbase=# SELECT inet_server_port();
 inet_server_port
------------------
             5432
(1 row)
```

- pg_backend_pid()

  描述：当前会话连接的服务进程的进程 ID。

  返回值类型：int

  示例：

```
vastbase=# SELECT pg_backend_pid();
 pg_backend_pid
-----------------
 140229352617744
(1 row)
```

- pg_conf_load_time()

  描述：配置加载时间。pg_conf_load_time 返回最后加载服务器配置文件的时间戳。

  返回值类型：timestamp with time zone

  示例：

```
vastbase=# SELECT pg_conf_load_time();
     pg_conf_load_time
-------------------------------
 2017-09-01 16:05:23.89868+08
(1 row)
```

- pg_my_temp_schema()

  描述：会话的临时模式的 OID，不存在则为 0。

  返回值类型：oid

  示例：

```
vastbase=# SELECT pg_my_temp_schema();
 pg_my_temp_schema
-------------------
```

```
             0
(1 row)
```

备注：pg_my_temp_schema 返回当前会话中临时模式的 OID，如果不存在（没有创建临时表)的话则返回 0。如果给定的 OID 是其它会话中临时模式的 OID，pg_is_other_temp_schema 则返回 true。

- pg_is_other_temp_schema(oid)

  描述：是否为另一个会话的临时模式。

  返回值类型：Boolean

  示例：

```
vastbase=# SELECT pg_is_other_temp_schema(25356);
 pg_is_other_temp_schema
-------------------------
 f
(1 row)
```

- pg_listening_channels()

  描述：会话正在监听的信道名称。

  返回值类型：setof text

  示例：

```
vastbase=# SELECT pg_listening_channels();
 pg_listening_channels
-----------------------
(0 rows)
```

  备注：pg_listening_channels 返回当前会话正在监听的一组信道名称。

- pg_postmaster_start_time()

  描述：服务器启动时间。pg_postmaster_start_time 返回服务器启动时的 timestamp with time zone。

  返回值类型：timestamp with time zone

  示例：

```
vastbase=# SELECT pg_postmaster_start_time();
  pg_postmaster_start_time
-------------------------------
 2017-08-30 16:02:54.99854+08
(1 row)
```

- pg_trigger_depth()

  描述：触发器的嵌套层次。

  返回值类型：int

  示例：

```
vastbase=# SELECT pg_trigger_depth();
 pg_trigger_depth
------------------
                0
(1 row)
```

- pgxc_version()

  描述：Postgres-XC 版本信息。

  返回值类型：text

  示例：

```
vastbase=# SELECT pgxc_version();
                                        pgxc_version
-------------------------------------------------------------------------------------------
-------------------
 Postgres-XC 1.1 on x86_64-unknown-linux-gnu, based on PostgreSQL 9.2.4, compiled by g++ (GCC)
5.4.0, 64-bit
(1 row)
```

- session_user

  描述：会话用户名。

  返回值类型：name

  示例：

```
vastbase=# SELECT session_user;
 session_user
--------------
 vastbase
(1 row)
```

  备注：session_user 通常是连接当前数据库的初始用户，不过系统管理员可以用 11.16.108SET SESSION AUTHORIZATION 修改这个设置。

- user

  描述：等价于 current_user。

  返回值类型：name

  示例：

```
vastbase=# SELECT user;
 current_user
--------------
 vastbase
(1 row)
```

- version()

  描述：版本信息。version 返回一个描述服务器版本信息的字符串。

  返回值类型：text

  示例：

```
vastbase=# SELECT version();
                                                    version
-------------------------------------------------------------------------------------------------------------
-------------------------------------------
 PostgreSQL 9.2.4 (Vastbase-1.0.0 build 66e54e4d) compiled at 2020-01-02 13:02:26 commit 7218
last mr 10175  on x86_64-unknown-linux-gnu, compiled by g++ (GCC) 8.2.0, 64-bit

(1 row)
```

访问权限查询函数

- has_any_column_privilege(user, table, privilege)

  描述：指定用户是否有访问表任何列的权限。

  返回类型：Boolean

- has_any_column_privilege(table, privilege)

  描述：当前用户是否有访问表任何列的权限。

  返回类型：Boolean

  备注：has_any_column_privilege 检查用户是否以特定方式访问表的任何列。其参数可能与 has_table_privilege 类似，除了访问权限类型必须是 SELECT、INSERT、UPDATE 或 REFERENCES 的一些组合。

说明

拥有表的表级别权限则隐含的拥有该表每列的列级权限，因此如果与 has_table_privilege 参数相同，has_any_column_privilege 总是返回 true。但是如果授予至少一列的列级权限也返回成功。

- has_column_privilege(user, table, column, privilege)

  描述：指定用户是否有访问列的权限。

  返回类型：Boolean

- has_column_privilege(table, column, privilege)

  描述：当前用户是否有访问列的权限。

  返回类型：Boolean

  备注：has_column_privilege 检查用户是否以特定方式访问一列。其参数类似于 has_table_privilege，可以通过列名或属性号添加列。想要的访问权限类型必须是 SELECT、INSERT、UPDATE 或 REFERENCES 的一些组合。

说明

拥有表的表级别权限则隐含的拥有该表每列的列级权限。

- has_database_privilege(user, database, privilege)

  描述：指定用户是否有访问数据库的权限。

返回类型：Boolean

- has_database_privilege(database, privilege)

  描述：当前用户是否有访问数据库的权限。

  返回类型：Boolean

  备注：has_database_privilege 检查用户是否能以在特定方式访问数据库。其参数类似 has_table_privilege。访问权限类型必须是 CREATE、CONNECT、TEMPORARY 或 TEMP（等价于 TEMPORARY）的一些组合。

- has_directory_privilege(user, database, privilege)

  描述：指定用户是否有访问 directory 的权限。

  返回类型：Boolean

- has_directory_privilege(database, privilege)

  描述：当前用户是否有访问 directory 的权限。

  返回类型：Boolean

- has_foreign_data_wrapper_privilege(user, fdw, privilege)

  描述：指定用户是否有访问外部数据封装器的权限。

  返回类型：Boolean

- has_foreign_data_wrapper_privilege(fdw, privilege)

  描述：当前用户是否有访问外部数据封装器的权限。

  返回类型：Boolean

  备注：has_foreign_data_wrapper_privilege 检查用户是否能以特定方式访问外部数据封装器。其参数类似 has_table_privilege。访问权限类型必须是 USAGE。

- has_function_privilege(user, function, privilege)

  描述：指定用户是否有访问函数的权限。

  返回类型：Boolean

- has_function_privilege(function, privilege)

  描述：当前用户是否有访问函数的权限。

  返回类型：Boolean

  备注：has_function_privilege 检查一个用户是否能以指定方式访问一个函数。其参数类似 has_table_privilege。使用文本字符而不是 OID 声明一个函数时，允许输入的类型和 regprocedure 数据类型一样（请参考 11.3.14 对象标识符类型）。访问权限类型必须是 EXECUTE。

- has_language_privilege(user, language, privilege)

  描述：指定用户是否有访问语言的权限。

  返回类型：Boolean

- has_language_privilege(language, privilege)

  描述：当前用户是否有访问语言的权限。

  返回类型：Boolean

  备注：has_language_privilege 检查用户是否能以特定方式访问一个过程语言。其参数类似 has_table_privilege。访问权限类型必须是 USAGE。

- has_schema_privilege(user, schema, privilege)

  描述：指定用户是否有访问模式的权限。

  返回类型：Boolean

- has_schema_privilege(schema, privilege)

  描述：当前用户是否有访问模式的权限。

  返回类型：Boolean

  备注：has_schema_privilege 检查用户是否能以特定方式访问一个模式。其参数类似 has_table_privilege。访问权限类型必须是 CREATE 或 USAGE 的一些组合。

- has_server_privilege(user, server, privilege)

  描述：指定用户是否有访问外部服务的权限。

  返回类型：Boolean

- has_server_privilege(server, privilege)

  描述：当前用户是否有访问外部服务的权限。

  返回类型：Boolean

  备注：has_server_privilege 检查用户是否能以指定方式访问一个外部服务器。其参数类似 has_table_privilege。访问权限类型必须是 USAGE。

- has_table_privilege(user, table, privilege)

  描述：指定用户是否有访问表的权限。

  返回类型：Boolean

- has_table_privilege(table, privilege)

  描述：当前用户是否有访问表的权限。

  返回类型：Boolean

备注：has_table_privilege 检查用户是否以特定方式访问表。用户可以通过名称或 OID（pg_authid.oid）来指定，public 表明 PUBLIC 伪角色，或如果缺省该参数，则使用 current_user。该表可以通过名称或者 OID 声明。如果用名称声明，则在必要时可以用模式进行修饰。如果使用文本字符串来声明所希望的权限类型，这个文本字符串必须是 SELECT、INSERT、UPDATE、DELETE、TRUNCATE、REFERENCES 或 TRIGGER 之一的值。可以给权限类型添加 WITH GRANT OPTION，用来测试权限是否拥有授权选项。也可以用逗号分隔列出的多个权限类型，如果拥有任何所列出的权限，则结果便为 true。

示例：

```
vastbase=# SELECT has_table_privilege('tpcds.web_site', 'select');
 has_table_privilege
---------------------
 t
(1 row)

vastbase=# SELECT has_table_privilege('vastbase', 'tpcds.web_site', 'select,INSERT WITH GRANT
OPTION ');
 has_table_privilege
---------------------
 t
(1 row)
```

- has_tablespace_privilege(user, tablespace, privilege)

  描述：指定用户是否有访问表空间的权限。

  返回类型：Boolean

- has_tablespace_privilege(tablespace, privilege)

  描述：当前用户是否有访问表空间的权限。

  返回类型：Boolean

  备注：has_tablespace_privilege 检查用户是否能以特定方式访问一个表空间。其参数类似 has_table_privilege。访问权限类型必须是 CREATE。

- pg_has_role(user, role, privilege)

  描述：指定用户是否有角色的权限。

  返回类型：Boolean

- pg_has_role(role, privilege)

  描述：当前用户是否有角色的权限。

  返回类型：Boolean

  备注：pg_has_role 检查用户是否能以特定方式访问一个角色。其参数类似 has_table_privilege，除了 public 不能用做用户名。访问权限类型必须是 MEMBER 或 USAGE

的一些组合。 MEMBER 表示的是角色中的直接或间接成员关系（也就是 SET ROLE 的权限），而 USAGE 表示无需通过 SET ROLE 也直接拥有角色的使用权限。

模式可见性查询函数

每个函数执行检查数据库对象类型的可见性。对于函数和操作符，如果在前面的搜索路径中没有相同的对象名称和参数的数据类型，则此对象是可见的。对于操作符类，则要同时考虑名称和相关索引的访问方法。

所有这些函数都需要使用 OID 来标识要需要检查的对象。如果用户想通过名称测试对象，则使用 OID 别名类型（regclass、regtype、regprocedure、regoperator、regconfig 或 regdictionary）将会很方便。

比如，如果一个表所在的模式在搜索路径中，并且在前面的搜索路径中没有同名的表，则这个表是可见的。它等效于表可以不带明确模式修饰进行引用。比如，要列出所有可见表的名称：

```
vastbase=# SELECT relname FROM pg_class WHERE pg_table_is_visible(oid);
```

- pg_collation_is_visible(collation_oid)

  描述：该排序是否在搜索路径中可见。

  返回类型：Boolean

- pg_conversion_is_visible(conversion_oid)

  描述：该转换是否在搜索路径中可见。

  返回类型：Boolean

- pg_function_is_visible(function_oid)

  描述：该函数是否在搜索路径中可见。

  返回类型：Boolean

- pg_opclass_is_visible(opclass_oid)

  描述：该操作符类是否在搜索路径中可见。

  返回类型：Boolean

- pg_operator_is_visible(operator_oid)

  描述：该操作符是否在搜索路径中可见。

  返回类型：Boolean

- pg_opfamily_is_visible(opclass_oid)

  描述：该操作符族是否在搜索路径中可见。

  返回类型：Boolean

- pg_table_is_visible(table_oid)

  描述：该表是否在搜索路径中可见。

返回类型：Boolean

- pg_ts_config_is_visible(config_oid)

  描述：该文本检索配置是否在搜索路径中可见。

  返回类型：Boolean

- pg_ts_dict_is_visible(dict_oid)

  描述：该文本检索词典是否在搜索路径中可见。

  返回类型：Boolean

- pg_ts_parser_is_visible(parser_oid)

  描述：该文本搜索解析是否在搜索路径中可见。

  返回类型：Boolean

- pg_ts_template_is_visible(template_oid)

  描述：该文本检索模板是否在搜索路径中可见。

  返回类型：Boolean

- pg_type_is_visible(type_oid)

  描述：该类型（或域）是否在搜索路径中可见。

  返回类型：Boolean

系统表信息函数

- format_type(type_oid, typemod)

  描述：获取数据类型的 SQL 名称

  返回类型：text

  备注：format_type 通过某个数据类型的类型 OID 以及可能的类型修饰词，返回其 SQL 名称。如果不知道具体的修饰词，则在类型修饰词的位置传入 NULL。类型修饰词一般只对有长度限制的数据类型有意义。format_type 所返回的 SQL 名称中包含数据类型的长度值，其大小是：实际存储长度 len - sizeof(int32)，单位字节。原因是数据存储时需要 32 位的空间来存储用户对数据类型的自定义长度信息，即实际存储长度要比用户定义长度多 4 个字节。在下例中，format_type 返回的 SQL 名称为“character varying(6)”，6 表示 varchar 类型的长度值是 6 字节，因此该类型的实际存储长度为 10 字节。

```
vastbase=# SELECT format_type((SELECT oid FROM pg_type WHERE typname='varchar'), 10);
    format_type
---------------------
 character varying(6)
(1 row)
```

- pg_check_authid(role_oid)

描述：检查是否存在给定 oid 的角色名

返回类型：bool

- pg_describe_object(catalog_id, object_id, object_sub_id)

  描述：获取数据库对象的描述

  返回类型：text

  备注：pg_describe_object 返回由目录 OID，对象 OID 和一个（或许 0 个）子对象 ID 指定的数据库对象的描述。这有助于确认存储在 pg_depend 系统表中对象的身份。

- pg_get_constraintdef(constraint_oid)

  描述：获取约束的定义

  返回类型：text

- pg_get_constraintdef(constraint_oid, pretty_bool)

  描述：获取约束的定义

  返回类型：text

  备注：pg_get_constraintdef 和 pg_get_indexdef 分别从约束或索引上使用创建命令进行重构。

- pg_get_expr(pg_node_tree, relation_oid)

  描述：反编译表达式的内部形式，假设其中的任何 Vars 都引用第二个参数指定的关系。

  返回类型：text

- pg_get_expr(pg_node_tree, relation_oid, pretty_bool)

  描述：反编译表达式的内部形式，假设其中的任何 Vars 都引用第二个参数指定的关系。

  返回类型：text

  备注：pg_get_expr 反编译一个独立表达式的内部形式，比如一个字段的缺省值。在检查系统表的内容的时候很有用。如果表达式可能包含关键字，则指定他们引用相关的 OID 作为第二个参数；如果没有关键字，零就足够了。

- pg_get_functiondef(func_oid)

  描述：获取函数的定义

  返回类型：text

- pg_get_function_arguments(func_oid)

  描述：获取函数定义的参数列表（带默认值）

  返回类型：text

备注：pg_get_function_arguments 返回一个函数的参数列表，需要在 CREATE FUNCTION 中使用这种格式。

- pg_get_function_identity_arguments(func_oid)

  描述：获取参数列表来确定一个函数（不带默认值）

  返回类型：text

  备注：pg_get_function_identity_arguments 返回需要的参数列表用来标识函数，这种形式需要在 ALTER FUNCTION 中使用，并且这种形式省略了默认值。

- pg_get_function_result(func_oid)

  描述：获取函数的 RETURNS 子句

  返回类型：text

  备注：pg_get_function_result 为函数返回适当的 RETURNS 子句。

- pg_get_indexdef(index_oid)

  描述：获取索引的 CREATE INDEX 命令

  返回类型：text

- pg_get_indexdef(index_oid, column_no, pretty_bool)

  描述：获取索引的 CREATE INDEX 命令，或者如果 column_no 不为零，则只获取一个索引字段的定义。

  返回类型：text

  备注：pg_get_functiondef 为函数返回一个完整的 CREATE OR REPLACE FUNCTION 语句。

- pg_get_keywords()

  描述：获取 SQL 关键字和类别列表

  返回类型：setof record

  备注：pg_get_keywords 返回一组关于描述服务器识别 SQL 关键字的记录。word 列包含关键字。catcode 列包含一个分类代码：U 表示通用的，C 表示列名，T 表示类型或函数名，或 R 表示保留。catdesc 列包含了一个可能本地化描述分类的字符串。

- pg_get_userbyid(role_oid)

  描述：获取给定 OID 的角色名

  返回类型：name

  备注：pg_get_userbyid 通过角色的 OID 抽取对应的用户名。

- pg_get_viewdef(view_name)

  描述：为视图获取底层的 SELECT 命令

返回类型：text

- pg_get_viewdef(view_name, pretty_bool)

  描述：为视图获取底层的 SELECT 命令，如果 pretty_bool 为 true，行字段可以包含 80 列。

  返回类型：text

  备注：pg_get_viewdef 重构出定义视图的 SELECT 查询。这些函数大多数都有两种形式，其中带有 pretty_bool 参数，且参数为 true 时，是"适合打印"的结果，这种格式更容易读。另一种是缺省的格式，更有可能被将来的不同版本用同样的方法解释。如果是用于转储，那么尽可能避免使用适合打印的格式。给 pretty-print 参数传递 false 生成的结果和没有这个参数的变种生成的结果是完全一样。

- pg_get_viewdef(view_oid)

  描述：为视图获取底层的 SELECT 命令

  返回类型：text

- pg_get_viewdef(view_oid, pretty_bool)

  描述：为视图获取底层的 SELECT 命令，如果 pretty_bool 为 true，行字段可以包含 80 列。

  返回类型：text

- pg_get_viewdef(view_oid, wrap_column_int)

  描述：为视图获取底层的 SELECT 命令；行字段被换到指定的列数，打印是隐含的。

  返回类型：text

- pg_get_tabledef(table_oid)

  描述：根据 table_oid 获取表定义

  返回类型：text

- pg_get_tabledef(table_name)

  描述：根据 table_name 获取表定义

  返回类型：text

  备注：pg_get_tabledef 重构出表定义的 CREATE 语句，包含了表定义本身、索引信息、comments 信息。对于表对象依赖的 group、schema、tablespace、server 等信息，需要用户自己去创建，表定义里不会有这些对象的创建语句。

- pg_options_to_table(reloptions)

  描述：获取存储选项名称/值对的集合

  返回类型：setof record

备注：pg_options_to_table 当通过 pg_class.reloptions 或 pg_attribute.attoptions 时返回存储选项名称/值对（option_name/option_value）的集合。

- pg_tablespace_databases(tablespace_oid)

  描述：获取在指定的表空间中有对象的数据库 OID 集合

  返回类型：setof oid

  备注：pg_tablespace_databases 允许检查表空间的状况，返回在该表空间中保存了对象的数据库 OID 集合。如果这个函数返回数据行，则该表空间就是非空的，因此不能删除。要显示该表空间中的特定对象，用户需要连接 pg_tablespace_databases 标识的数据库与查询 pg_class 系统表。

- pg_tablespace_location(tablespace_oid)

  描述：获取表空间所在的文件系统的路径

  返回类型：text

- pg_typeof(any)

  描述：获取任何值的数据类型

  返回类型：regtype

  备注：pg_typeof 返回传递给他的值的数据类型 OID。这可能有助于故障排除或动态构造 SQL 查询。声明此函数返回 regtype，这是一个 OID 别名类型（请参考 11.3.14 对象标识符类型）；这意味着它是一个为了比较而显示类型名称的 OID。

  示例：

```
vastbase=# SELECT pg_typeof(33);
 pg_typeof
-----------
 integer
(1 row)

vastbase=# SELECT typlen FROM pg_type WHERE oid = pg_typeof(33);
 typlen
--------
      4
(1 row)
```

- collation for (any)

  描述：获取参数的排序

  返回类型：text

  备注：表达式 collation for 返回传递给他的值的排序。示例：

```
vastbase=# SELECT collation for (description) FROM pg_description LIMIT 1;
 pg_collation_for
------------------
```

```
 "default"
(1 row)
```

值可能是引号括起来的并且模式限制的。如果没有为参数表达式排序，则返回一个 null 值。如果参数不是排序的类型，则抛出一个错误。

- getdistributekey(table_name)

  描述：获取一个 hash 表的分布列。

  返回类型：text

  示例：

```
vastbase=# SELECT getdistributekey('item');
 getdistributekey
------------------
 i_item_sk
(1 row)
```

注释信息函数

- col_description(table_oid, column_number)

  描述：获取一个表字段的注释

  返回类型：text

  备注：col_description 返回一个表中字段的注释，通过表 OID 和字段号来声明。

- obj_description(object_oid, catalog_name)

  描述：获取一个数据库对象的注释

  返回类型：text

  备注：带有两个参数的 obj_description 返回一个数据库对象的注释，该对象是通过其 OID 和其所属的系统表名称声明。比如，obj_description(123456,'pg_class')将返回 OID 为 123456 的表的注释。只带一个参数的 obj_description 只要求对象 OID。

  obj_description 不能用于表字段，因为字段没有自己的 OID。

- obj_description(object_oid)

  描述：获取一个数据库对象的注释

  返回类型：text

- shobj_description(object_oid, catalog_name)

  描述：获取一个共享数据库对象的注释

  返回类型：text

  备注：shobj_description 和 obj_description 差不多，不同之处仅在于前者用于共享对象。一些系统表是通用于 Vastbase 中所有数据库的全局表，因此这些表的注释也是全局存储的。

事务 ID 和快照

以下的函数在一个输出形式中提供服务器事务信息。这些函数的主要用途是为了确定在两个快照之间有哪个事务提交。

- pgxc_is_committed(transaction_id)

  描述：如果提交或忽略给定的 XID（gxid）。NULL 表示的状态是未知的（运行，准备，冻结等）。

  返回类型：bool

- txid_current()

  描述：获取当前事务 ID。

  返回类型：bigint

- txid_current_snapshot()

  描述：获取当前快照。

  返回类型：txid_snapshot

- txid_snapshot_xip(txid_snapshot)

  描述：在快照中获取正在进行的事务 ID。

  返回类型：setof bigint

- txid_snapshot_xmax(txid_snapshot)

  描述：获取快照的 xmax。

  返回类型：bigint

- txid_snapshot_xmin(txid_snapshot)

  描述：获取快照的 xmin。

  返回类型：bigint

- txid_visible_in_snapshot(bigint, txid_snapshot)

  描述：在快照中事务 ID 是否可见（不使用子事务 ID）。

  返回类型：Boolean

- get_local_prepared_xact()

  描述：获取当前节点两阶段残留事务信息，包括事务 id，两阶段 gid 名称，prepared 的时间，owner 的 oid，database 的 oid 及当前节点的 node_name。

  返回类型：xid, text, timestamptz, oid, oid，text

- get_remote_prepared_xacts()

描述：获取所有远程节点两阶段残留事务信息，包括事务 id，两阶段 gid 名称，prepared 的时间，owner 的名称，database 的名称及 node_name。

返回类型：xid, text, timestamptz, name, name，text

- global_clean_prepared_xacts(text, text)

  描述：并发清理两阶段残留事务，仅 gs_clean 工具可以调用清理，其他用户调用均返回 false。

  返回类型：Boolean

- pgxc_stat_get_wal_senders_status()

  描述：返回节点事务日志备机接收状态。

  返回值如下：

表 11-30. pgxc_stat_get_wal_senders_status 返回参数说明

字段名	描述
nodename	主节点名。
source_ip	主节点 IP。
source_port	主节点端口。
dest_ip	备节点 IP。
dest_port	备节点端口。
sender_pid	发送线程 PID。
local_role	主节点类型。
peer_role	备节点类型。
peer_state	备节点状态。
state	wal sender 状态。
sender_sent_location	主节点发送位置。
sender_write_location	主节点落盘位置。
sender_replay_location	主节点 redo 位置。
receiver_received_location	备节点接收位置。
receiver_write_location	备节点落盘位置。

字段名	描述
receiver_flush_location	备节点 flush 磁盘位置。
receiver_replay_location	备节点 redo 位置。

内部事务 ID 类型（xid）是 32 位，每 40 亿事务一次循环。这些函数使用的数据类型 txid_snapshot，存储在特定时刻事务 ID 可见性的信息。其组件描述在表 11-31。

表 11-31. 快照组件

名称	描述
xmin	最早的事务 ID（txid）仍然活动。所有较早事务将是已经提交可见的，或者是直接回滚。
xmax	作为尚未分配的 txid。所有大于或等于此 txids 的都是尚未开始的快照时间，因此不可见。
xip_list	当前快照中活动的 txids。这个列表只包含在 xmin 和 xmax 之间活动的 txids；有可能活动的 txids 高于 xmax。 介于大于等于 xmin、小于 xmax，并且不在这个列表中的 txid，在这个时间快照已经完成的，因此按照提交状态查看他是可见还是回滚。这个列表不包含子事务的 txids。

txid_snapshot 的文本表示为：xmin:xmax:xip_list。
示例：10:20:10,14,15 意思为：xmin=10, xmax=20, xip_list=10, 14, 15。

11.5.23.系统管理函数

11.5.23.1. 配置设置函数

配置设置函数是可以用于查询以及修改运行时配置参数的函数。

- current_setting(setting_name)

  描述：当前的设置值。

  返回值类型：text

  备注：current_setting 用于以查询形式获取 setting_name 的当前值。和 SQL 语句 SHOW 是等效的。比如：

```
vastbase=# SELECT current_setting('datestyle');

 current_setting
-----------------
 ISO, MDY
(1 row)
```

- set_config(setting_name, new_value, is_local)

  描述：设置参数并返回新值。

  返回值类型：text

  备注：set_config 将参数 setting_name 设置为 new_value，如果 is_local 为 true，则新值将只应用于当前事务。如果希望新值应用于当前会话，可以使用 false，和 SQL 语句 SET 是等效的。比如：

```
vastbase=# SELECT set_config('log_statement_stats', 'off', false);

 set_config
------------
 off
(1 row)
```

11.5.23.2. 通用文件访问函数

通用文件访问函数提供了对数据库服务器上的文件的本地访问接口。只有 Vastbase 目录和 log_directory 目录里面的文件可以访问。使用相对路径访问 Vastbase 目录里面的文件，以及匹配 log_directory 配置而设置的路径访问日志文件。只有数据库初始化用户才能使用这些函数。

- pg_ls_dir(dirname text)

  描述：列出目录中的文件。

  返回值类型：setof text

  备注：pg_ls_dir 返回指定目录里面的除了特殊项“.”和“..”之外所有名称。

  示例：

```
vastbase=# SELECT pg_ls_dir('./');
     pg_ls_dir
----------------------
 .postgresql.conf.swp
 postgresql.conf
 pg_tblspc
 PG_VERSION
 pg_ident.conf
 core
 server.crt
 pg_serial
 pg_twophase
 postgresql.conf.lock
 pg_stat_tmp
 pg_notify
 pg_subtrans
 pg_ctl.lock
 pg_xlog
 pg_clog
 base
```

```
pg_snapshots
postmaster.opts
postmaster.pid
server.key.rand
server.key.cipher
pg_multixact
pg_errorinfo
server.key
pg_hba.conf
pg_replslot
.pg_hba.conf.swp
cacert.pem
pg_hba.conf.lock
global
vastbase.state
(32 rows)
```

- pg_read_file(filename text, offset bigint, length bigint)

  描述：返回一个文本文件的内容。

  返回值类型：text

  备注：pg_read_file 返回一个文本文件的一部分，从 offset 开始，最多返回 length 字节（如果先达到文件结尾，则小于这个数值）。如果 offset 是负数，则它是相对于文件结尾回退的长度。如果省略了 offset 和 length，则返回整个文件。

  示例：

```
vastbase=# SELECT pg_read_file('postmaster.pid',0,100);
          pg_read_file
---------------------------------------
 53078                          +
 /srv/BigData/hadoop/data1/dbnode+
 1500022474                      +
 5432                           +
 /var/run/FusionInsight             +
 localhost                       +
  2
(1 row)
```

- pg_read_binary_file(filename text [, offset bigint, length bigint,missing_ok boolean])

  描述：返回一个二进制文件的内容。

  返回值类型：bytea

  备注：pg_read_binary_file 的功能与 pg_read_file 类似，除了结果的返回值为 bytea 类型不一致，相应地不会执行编码检查。与 convert_from 函数结合，这个函数可以用来读取用指定编码的一个文件。

```
vastbase=# SELECT convert_from(pg_read_binary_file('filename'), 'UTF8');
```

- pg_stat_file(filename text)

  描述：返回一个文本文件的状态信息。

  返回值类型：record

备注：pg_stat_file 返回一条记录，其中包含：文件大小、最后访问时间戳、最后更改时间戳、最后文件状态修改时间戳以及标识传入参数是否为目录的 Boolean 值。典型的用法：

```
vastbase=# SELECT * FROM pg_stat_file('filename');
vastbase=# SELECT (pg_stat_file('filename')).modification;
```

示例：

```
vastbase=# SELECT convert_from(pg_read_binary_file('postmaster.pid'), 'UTF8');
          convert_from
-------------------------------------
 4881                          +
 /srv/BigData/vastbase/data1/dbnode+
 1496308688                     +
 25108                          +
 /opt/huawei/Bigdata/vastbase/vastbase_tmp +
 *                              +
  25108001  43352069              +

(1 row)
vastbase=# SELECT * FROM pg_stat_file('postmaster.pid');

 size |        access        |     modification     |        change
| creation | isdir
------+----------------------+----------------------+-----------------------
+----------+-------
  117 | 2017-06-05 11:06:34+08 | 2017-06-01 17:18:08+08 | 2017-06-01 17:18:08+08
|          | f
(1 row)
vastbase=# SELECT (pg_stat_file('postmaster.pid')).modification;
     modification
-----------------------
 2017-06-01 17:18:08+08
(1 row)
```

11.5.23.3. 服务器信号函数

服务器信号函数向其他服务器进程发送控制信号。只有系统管理员才能使用这些函数。

- pg_cancel_backend(pid int)

  描述：取消一个后端的当前查询。

  返回值类型：Boolean

  备注：pg_cancel_backend 向由 pid 标识的后端进程发送一个查询取消（SIGINT）信号。一个活动的后端进程的 PID 可以从 pg_stat_activity 视图的 pid 字段找到，或者在服务器上用 ps 列出数据库进程。

- pg_reload_conf()

  描述：导致所有服务器进程重新装载它们的配置文件。

  返回值类型：Boolean

备注：pg_reload_conf 给服务器发送一个 SIGHUP 信号，导致所有服务器进程重新装载配置文件。

- pg_rotate_logfile()

  描述：滚动服务器的日志文件。

  返回值类型：Boolean

  备注：pg_rotate_logfile 给日志文件管理器发送信号，告诉它立即切换到一个新的输出文件。这个函数只有在 redirect_stderr 用于日志输出的时候才有用，否则根本不存在日志文件管理器子进程。

- pg_terminate_backend(pid int)

  描述：终止一个后台线程。

  返回值类型：Boolean

  备注：如果成功，函数返回 true，否则返回 false。

  示例：

```
vastbase=# SELECT pid from pg_stat_activity;
     pid
-----------------
 140657876268816
(1 rows)

vastbase=# SELECT pg_terminate_backend(140657876268816);
 pg_terminate_backend
----------------------
 t
(1 row)
```

11.5.23.4. 备份恢复控制函数

备份控制函数

备份控制函数可帮助进行在线备份。

- pg_create_restore_point(name text)

  描述：为执行恢复创建一个命名点。（需要管理员角色）

  返回值类型：text

  备注：pg_create_restore_point 创建了一个可以用作恢复目的、有命名的事务日志记录，并返回相应的事务日志位置。在恢复过程中，recovery_target_name 可以通过这个名称定位对应的日志恢复点，并从此处开始执行恢复操作。避免使用相同的名称创建多个恢复点，因为恢复操作将在第一个匹配（恢复目标）的名称上停止。

- pg_current_xlog_location()

描述：获取当前事务日志的写入位置。

返回值类型：text

备注：pg_current_xlog_location 使用与前面那些函数相同的格式显示当前事务日志的写入位置。如果是只读操作，不需要系统管理员权限。

- pg_current_xlog_insert_location()

描述：获取当前事务日志的插入位置。

返回值类型：text

备注：pg_current_xlog_insert_location 显示当前事务日志的插入位置。插入点是事务日志在某个瞬间的“逻辑终点”，而实际的写入位置则是从服务器内部缓冲区写出时的终点。写入位置是可以从服务器外部检测到的终点，如果要归档部分完成事务日志文件，则该操作即可实现。插入点主要用于服务器调试目的。如果是只读操作，不需要系统管理员权限。

- pg_start_backup(label text [, fast boolean])

描述：开始执行在线备份。（需要管理员角色或复制的角色）

返回值类型：text

备注：pg_start_backup 接受一个用户定义的备份标签（通常这是备份转储文件存放地点的名称）。这个函数向 Vastbase 的数据目录写入一个备份标签文件，然后以文本方式返回备份的事务日志起始位置。

```
vastbase=# SELECT pg_start_backup('label_goes_here');
 pg_start_backup
-----------------
 0/3000020
(1 row)
```

- pg_stop_backup()

描述：完成执行在线备份。（需要管理员角色或复制的角色）

返回值类型：text

备注：pg_stop_backup 删除 pg_start_backup 创建的标签文件，并且在事务日志归档区里创建一个备份历史文件。这个历史文件包含给予 pg_start_backup 的标签、备份的事务日志起始与终止位置、备份的起始和终止时间。返回值是备份的事务日志终止位置。计算出中止位置后，当前事务日志的插入点将自动前进到下一个事务日志文件，这样，结束的事务日志文件可以被立即归档从而完成备份。

- pg_switch_xlog()

描述：切换到一个新的事务日志文件。（需要管理员角色）

返回值类型：text

备注：pg_switch_xlog 移动到下一个事务日志文件，以允许将当前日志文件归档（假定使用连续归档）。返回值是刚完成的事务日志文件的事务日志结束位置+1。如果从最后一次事务日志切换以来没有活动的事务日志，则 pg_switch_xlog 什么事也不做，直接返回当前事务日志文件的开始位置。

- pg_xlogfile_name(location text)

  描述：将事务日志的位置字符串转换为文件名。

  返回值类型：text

  备注：pg_xlogfile_name 仅抽取事务日志文件名称。如果给定的事务日志位置恰好位于事务日志文件的交界上，这两个函数都返回前一个事务日志文件的名称。这对于管理事务日志归档来说是非常有利的，因为前一个文件是当前最后一个需要归档的文件。

- pg_xlogfile_name_offset(location text)

  描述：将事务日志的位置字符串转换为文件名并返回在文件中的字节偏移量。

  返回值类型：text,integer

  备注：可以使用 pg_xlogfile_name_offset 从前述函数的返回结果中抽取相应的事务日志文件名称和字节偏移量。例如：

```
vastbase=# SELECT * FROM pg_xlogfile_name_offset(pg_stop_backup());
NOTICE:  pg_stop_backup cleanup done, waiting for required WAL segments to be archived
NOTICE:  pg_stop_backup complete, all required WAL segments have been archived
      file_name        | file_offset
--------------------------+-------------
000000010000000000000003  |         272
(1 row)
```

- pg_xlog_location_diff(location text, location text)

  描述：计算两个事务日志位置之间在字节上的区别。

  返回值类型：numeric

- pg_cbm_tracked_location()

  描述：用于查询 cbm 解析到的 lsn 位置。

  返回值类型：text

- pg_cbm_get_merged_file(startLSNArg text, endLSNArg text)

  描述：用于将指定 lsn 范围之内的 cbm 文件合并成一个 cbm 文件，并返回合并完的 cbm 文件名。

  返回值类型：text

- pg_cbm_get_changed_block(startLSNArg text, endLSNArg text)

  描述：用于将指定 lsn 范围之内的 cbm 文件合并成一个表，并返回表的各行记录。

  返回值类型：records

备注：pg_cbm_get_changed_block 返回的表字段包含：合并起始的 lsn，合并截止的 lsn，表空间 oid，库 oid，表的 relfilenode，表的 fork number，表是否被删除，表是否被创建，表是否被截断，表被截断后的页面数，有多少页被修改以及被修改的页号的列表。

- pg_cbm_recycle_file(targetLSNArg text)

  描述：删除不再使用的 cbm 文件，并返回删除后的第一条 lsn。

  返回值类型：text

- pg_cbm_force_track(targetLSNArg text,timeOut int)

  描述：强制执行一次 cbm 追踪到指定的 xlog 位置，并返回实际追踪结束点的 xlog 位置。

  返回值类型：text

- pg_enable_delay_ddl_recycle(is_full_backup boolean, backup_key text)

  描述：开启延迟 DDL 功能，并返回开启点的 xlog 位置。

  返回值类型：text

- pg_disable_delay_ddl_recycle(barrierLSNArg text, isForce bool, is_full_backup boolean, backup_key text)

  描述：关闭延迟 DDL 功能，并返回本次延迟 DDL 生效的 xlog 范围。

  返回值类型：records

- pg_enable_delay_xlog_recycle()

  描述：开启延迟 xlog 回收功能，数据库主节点修复使用。

  返回值类型：void

- pg_disable_delay_xlog_recycle()

  描述：关闭延迟 xlog 回收功能，数据库主节点修复使用。

  返回值类型：void

恢复控制函数

恢复信息函数提供了当前备机状态的信息。这些函数可能在恢复期间或正常运行中执行。

- pg_is_in_recovery()

  描述：如果恢复仍然在进行中则返回 true。

  返回值类型：bool

- pg_last_xlog_receive_location()

  描述：获取最后接收事务日志的位置并通过流复制将其同步到磁盘。当流复制正在进行时，事务日志将持续递增。如果恢复已完成，则最后一次获取的 WAL 记录会被静态保持并在恢复过程中同步到磁盘。如果流复制不可用，或还没有开始，这个函数返回 NULL。

返回值类型：text

- pg_last_xlog_replay_location()

  描述：获取最后一个事务日志在恢复时重放的位置。如果恢复仍在进行，事务日志将持续递增。如果已经完成恢复，则将保持在恢复期间最后接收 WAL 记录的值。如果未进行恢复但服务器正常启动时，则这个函数返回 NULL。

  返回值类型：text

- pg_last_xact_replay_timestamp()

  描述：获取最后一个事务在恢复时重放的时间戳。这是为在主节点上生成事务提交或终止 WAL 记录的时间。如果在恢复时没有事务重放，则这个函数返回 NULL。如果恢复仍在进行，则事务日志将持续递增。如果恢复已经完成，则将保持在恢复期间最后接收 WAL 记录的值。如果服务器无需恢复就已正常启动，则这个函数返回 NULL。

  返回值类型：timestamp with time zone

恢复控制函数控制恢复的进程。这些函数可能只在恢复时被执行。

- pg_is_xlog_replay_paused()

  描述：如果恢复暂停则返回 true。

  返回值类型：bool

- pg_xlog_replay_pause()

  描述：立即暂停恢复。

  返回值类型：void

- pg_xlog_replay_resume()

  描述：如果恢复处于暂停状态，则重新启动。

  返回值类型：void

当恢复暂停时，没有发生数据库更改。如果是在热备里，所有新的查询将看到一致的数据库快照，并且不会有进一步的查询冲突产生，直到恢复继续。

如果不能使用流复制，则暂停状态将无限的延续。当流复制正在进行时，将连续接收 WAL 记录，最终将填满可用磁盘空间，这个进度取决于暂停的持续时间，WAL 生成的速度和可用的磁盘空间。

11.5.23.5. 快照同步函数

快照同步函数是导出当前快照的标识符。

pg_export_snapshot()

描述：保存当前的快照并返回它的标识符。

返回值类型：text

备注：函数 pg_export_snapshot 保存当前的快照并返回一个文本字符串标识此快照。这个字符串必须传递给想要导入快照的客户端。可用在 set transaction snapshot snapshot_id 时导入 snapshot，但是应用的前提是该事务设置了 SERIALIZABLE 或 REPEATABLE READ 隔离级别。而 Vastbase 目前是不支持这两种隔离级别的。该函数的输出不可用做 set transaction snapshot 的输入。

pg_export_snapshot_and_csn()

描述：保存当前的快照并返回它的标识符。比 pg_export_snapshot()多返回一列 CSN，表示当前快照的 CSN。

返回值类型：text

11.5.23.6. 数据库对象函数

数据库对象尺寸函数

数据库对象尺寸函数计算数据库对象使用的实际磁盘空间。

- pg_column_size(any)

  描述：存储一个指定的数值需要的字节数（可能压缩过）。

  返回值类型：int

  备注：pg_column_size 显示用于存储某个独立数据值的空间。

```
vastbase=# SELECT pg_column_size(1);
 pg_column_size
----------------
              4
(1 row)
```

- pg_database_size(oid)

  描述：指定 OID 代表的数据库使用的磁盘空间。

  返回值类型：bigint

- pg_database_size(name)

  描述：指定名称的数据库使用的磁盘空间。

  返回值类型：bigint

  备注：pg_database_size 接受一个数据库的 OID 或者名称，然后返回该对象使用的全部磁盘空间。

  示例：

```
vastbase=# SELECT pg_database_size(' vastbase ');
 pg_database_size
------------------
         51590112
(1 row)
```

- pg_relation_size(oid)

描述：指定 OID 代表的表或者索引所使用的磁盘空间。

返回值类型：bigint

- get_db_source_datasize()

  描述：估算当前数据库非压缩态的数据总容量

  返回值类型：bigint

  备注：（1）调用该函数前需要做 analyze；（2）通过估算列存的压缩率计算非压缩态的数据总容量。

  示例：

```
vastbase=# analyze;
ANALYZE
vastbase=# select get_db_source_datasize();
 get_db_source_datasize
------------------------
             35384925667
(1 row)
```

- pg_relation_size(text)

  描述：指定名称的表或者索引使用的磁盘空间。表名称可以用模式名修饰。

  返回值类型：bigint

- pg_relation_size(relation regclass, fork text)

  描述：指定表或索引的指定分叉树（'main'，'fsm'或'vm'）使用的磁盘空间。

  返回值类型：bigint

- pg_relation_size(relation regclass)

  描述：pg_relation_size(..., 'main')的简写。

  返回值类型：bigint

  备注：pg_relation_size 接受一个表、索引、压缩表的 OID 或者名称，然后返回它们的字节大小。

- pg_partition_size(oid,oid)

  描述：指定 OID 代表的分区使用的磁盘空间。其中，第一个 oid 为表的 OID，第二个 oid 为分区的 OID。

  返回值类型：bigint

- pg_partition_size(text, text)

  描述：指定名称的分区使用的磁盘空间。其中，第一个 text 为表名，第二个 text 为分区名。

  返回值类型：bigint

- pg_partition_indexes_size(oid,oid)

描述：指定 OID 代表的分区的索引使用的磁盘空间。其中，第一个 oid 为表的 OID，第二个 oid 为分区的 OID。

返回值类型：bigint

- pg_partition_indexes_size(text,text)

  描述：指定名称的分区的索引使用的磁盘空间。其中，第一个 text 为表名，第二个 text 为分区名。

  返回值类型：bigint

- pg_indexes_size(regclass)

  描述：附加到指定表的索引使用的总磁盘空间。

  返回值类型：bigint

- pg_size_pretty(bigint)

  描述：将以 64 位整数表示的字节值转换为具有单位的易读格式。

  返回值类型：text

- pg_size_pretty(numeric)

  描述：将以数值表示的字节值转换为具有单位的易读格式。

  返回值类型：text

  备注：pg_size_pretty 用于把其他函数的结果格式化成一种易读的格式，可以根据情况使用 KB 、MB 、GB 、TB。

- pg_table_size(regclass)

  描述：指定的表使用的磁盘空间，不计索引（但是包含 TOAST，自由空间映射和可见性映射）。

  返回值类型：bigint

- pg_tablespace_size(oid)

  描述：指定 OID 代表的表空间使用的磁盘空间。

  返回值类型：bigint

- pg_tablespace_size(name)

  描述：指定名称的表空间使用的磁盘空间。

  返回值类型：bigint

  备注：

  pg_tablespace_size 接受一个数据库的 OID 或者名称，然后返回该对象使用的全部磁盘空间。

- pg_total_relation_size(oid)

  描述：指定 OID 代表的表使用的磁盘空间，包括索引和压缩数据。

返回值类型：bigint

- pg_total_relation_size(regclass)

  描述：指定的表使用的总磁盘空间，包括所有的索引和 TOAST 数据。

  返回值类型：bigint

- pg_total_relation_size(text)

  描述：指定名称的表所使用的全部磁盘空间，包括索引和压缩数据。表名称可以用模式名修饰。

  返回值类型：bigint

  备注：pg_total_relation_size 接受一个表或者一个压缩表的 OID 或者名称，然后返回以字节计的数据和所有相关的索引和压缩表的尺寸。

- datalength(any)

  描述：计算一个指定的数据需要的字节数（不考虑数据的管理空间和数据压缩，数据类型转换等情况）。

  返回值类型：int

  备注：datalength 用于计算某个独立数据值的空间。

  示例：

```
vastbase=# SELECT datalength(1);
datalength
------------
4
(1 row)
```

目前支持的数据类型及计算方式见下表：

数据类型			存储空间
数值类型	整数类型	TINYINT	1
		SMALLINT	2
		INTEGER	4
		BINARY_INTEGER	4
		BIGINT	8
	任意精度型	DECIMAL	每 4 位十进制数占两个字节，小数点前后数字分别计算
		NUMERIC	每 4 位十进制数占两个字节，小数点前后数字分别计算
		NUMBER	每 4 位十进制数占两个字节，小数点

			前后数字分别计算
	序列整型	SMALLSERIAL	2
		SERIAL	4
		BIGSERIAL	8
	浮点类型	FLOAT4	4
		DOUBLE PRECISION	8
		FLOAT8	8
		BINARY_DOUBLE	8
		FLOAT[(p)]	每 4 位十进制数占两个字节，小数点前后数字分别计算
		DEC[(p[,s])]	每 4 位十进制数占两个字节，小数点前后数字分别计算
		INTEGER[(p[,s])]	每 4 位十进制数占两个字节，小数点前后数字分别计算
布尔类型	布尔类型	BOOLEAN	1
字符类型	字符类型	CHAR	n
		CHAR(n)	n
		CHARACTER(n)	n
		NCHAR(n)	n
		VARCHAR(n)	n
		CHARACTER	字符实际字节数
		VARYING(n)	字符实际字节数
		VARCHAR2(n)	字符实际字节数
		NVARCHAR2(n)	字符实际字节数
		TEXT	字符实际字节数
		CLOB	字符实际字节数
时间类型	时间类	DATE	8
		TIME	8

	型	TIMEZ	12
		TIMESTAMP	8
		TIMESTAMPZ	8
		SMALLDATETIME	8
		INTERVAL DAY TO SECOND	16
		INTERVAL	16
		RELTIME	4
		ABSTIME	4
		TINTERVAL	12

数据库对象位置函数

- pg_relation_filenode(relation regclass)

  描述：指定关系的文件节点数。

  返回值类型：oid

  备注：pg_relation_filenode 接受一个表、索引、序列或压缩表的 OID 或者名称，并且返回当前分配给它的"filenode"数。文件节点是关系使用的文件名称的基本组件。对大多数表来说，结果和 pg_class.relfilenode 相同，但对确定的系统目录来说， relfilenode 为 0 而且这个函数必须用来获取正确的值。如果传递一个没有存储的关系，比如一个视图，那么这个函数返回 NULL。

- pg_relation_filepath(relation regclass)

  描述：指定关系的文件路径名。

  返回值类型：text

  备注：pg_relation_filepath 类似于 pg_relation_filenode，但是它返回关系的整个文件路径名（相对于 Vastbase 的数据目录 PGDATA）。

11.5.23.7. 咨询锁函数

咨询锁函数用于管理咨询锁（Advisory Lock）。

- pg_advisory_lock(key bigint)

  描述：获取会话级别的排它咨询锁。

  返回值类型：void

备注：pg_advisory_lock 锁定应用程序定义的资源，该资源可以用一个 64 位或两个不重叠的 32 位键值标识。如果已经有另外的会话锁定了该资源，则该函数将阻塞到该资源可用为止。这个锁是排它的。多个锁定请求将会被压入栈中，因此，如果同一个资源被锁定了三次，它必须被解锁三次以将资源释放给其他会话使用。

- pg_advisory_lock(key1 int, key2 int)

  描述：获取会话级别的排它咨询锁。

  返回值类型：void

  备注：只允许 sysadmin 对键值对(65535, 65535)加会话级别的排它咨询锁，普通用户无权限。

- pg_advisory_lock_shared(key bigint)

  描述：获取会话级别的共享咨询锁。

  返回值类型：void

- pg_advisory_lock_shared(key1 int, key2 int)

  描述：获取会话级别的共享咨询锁。

  返回值类型：void

  备注：pg_advisory_lock_shared 类似于 pg_advisory_lock，不同之处仅在于共享锁会话可以和其他请求共享锁的会话共享资源，但排它锁除外。

- pg_advisory_unlock(key bigint)

  描述：释放会话级别的排它咨询锁。

  返回值类型：Boolean

- pg_advisory_unlock(key1 int, key2 int)

  描述：释放会话级别的排它咨询锁。

  返回值类型：Boolean

  备注：pg_advisory_unlock 释放先前取得的排它咨询锁。如果释放成功则返回 true。如果实际上并未持有指定的锁，将返回 false 并在服务器中产生一条 SQL 警告信息。

- pg_advisory_unlock_shared(key bigint)

  描述：释放会话级别的共享咨询锁。

  返回值类型：Boolean

- pg_advisory_unlock_shared(key1 int, key2 int)

  描述：释放会话级别的共享咨询锁。

  返回值类型：Boolean

备注：pg_advisory_unlock_shared 类似于 pg_advisory_unlock，不同之处在于该函数释放的是共享咨询锁。

- pg_advisory_unlock_all()

  描述：释放当前会话持有的所有咨询锁。

  返回值类型：void

  备注：pg_advisory_unlock_all 将会释放当前会话持有的所有咨询锁，该函数在会话结束的时候被隐含调用，即使客户端异常地断开连接也是一样。

- pg_advisory_xact_lock(key bigint)

  描述：获取事务级别的排它咨询锁。

  返回值类型：void

- pg_advisory_xact_lock(key1 int, key2 int)

  描述：获取事务级别的排它咨询锁。

  返回值类型：void

  备注：pg_advisory_xact_lock 类似于 pg_advisory_lock，不同之处在于锁是自动在当前事务结束时释放，而且不能被显式的释放。只允许 sysadmin 对键值对(65535, 65535)加事务级别的排它咨询锁，普通用户无权限。

- pg_advisory_xact_lock_shared(key bigint)

  描述：获取事务级别的共享咨询锁。

  返回值类型：void

- pg_advisory_xact_lock_shared(key1 int, key2 int)

  描述：获取事务级别的共享咨询锁。

  返回值类型：void

  备注：pg_advisory_xact_lock_shared 类似于 pg_advisory_lock_shared，不同之处在于锁是在当前事务结束时自动释放，而且不能被显式的释放。

- pg_try_advisory_lock(key bigint)

  描述：尝试获取会话级排它咨询锁。

  返回值类型：Boolean

  备注：pg_try_advisory_lock 类似于 pg_advisory_lock，不同之处在于该函数不会阻塞以等待资源的释放。它要么立即获得锁并返回 true，要么返回 false 表示目前不能锁定。

- pg_try_advisory_lock(key1 int, key2 int)

  描述：尝试获取会话级排它咨询锁。

返回值类型：Boolean

备注：只允许 sysadmin 对键值对(65535, 65535)加会话级别的排它咨询锁，普通用户无权限。

- pg_try_advisory_lock_shared(key bigint)

  描述：尝试获取会话级共享咨询锁。

  返回值类型：Boolean

- pg_try_advisory_lock_shared(key1 int, key2 int)

  描述：尝试获取会话级共享咨询锁。

  返回值类型：Boolean

  备注：pg_try_advisory_lock_shared 类似于 pg_try_advisory_lock，不同之处在于该函数尝试获得共享锁而不是排它锁。

- pg_try_advisory_xact_lock(key bigint)

  描述：尝试获取事务级别的排它咨询锁。

  返回值类型：Boolean

- pg_try_advisory_xact_lock(key1 int, key2 int)

  描述：尝试获取事务级别的排它咨询锁。

  返回值类型：Boolean

  备注：pg_try_advisory_xact_lock 类似于 pg_try_advisory_lock，不同之处在于如果得到锁，在当前事务的结束时自动释放，而且不能被显式的释放。只允许 sysadmin 对键值对(65535, 65535)加事务级别的排它咨询锁，普通用户无权限。

- pg_try_advisory_xact_lock_shared(key bigint)

  描述：尝试获取事务级别的共享咨询锁。

  返回值类型：Boolean

- pg_try_advisory_xact_lock_shared(key1 int, key2 int)

  描述：尝试获取事务级别的共享咨询锁。

  返回值类型：Boolean

  备注：pg_try_advisory_xact_lock_shared 类似于 pg_try_advisory_lock_shared，不同之处在于如果得到锁，在当前事务结束时自动释放，而且不能被显式的释放。

- lock_cluster_ddl()

  描述：尝试对 Vastbase 内所有存活的数据库主节点获取会话级别的排他咨询锁。

  返回值类型：Boolean

  备注：只允许 sysadmin 调用，普通用户无权限。

- unlock_cluster_ddl()

  描述：尝试对数据库主节点会话级别的排他咨询锁。

  返回值类型：Boolean

11.5.23.8. 逻辑复制函数

- pg_create_logical_replication_slot('slot_name', 'plugin_name')

  描述：创建逻辑复制槽。

  参数说明：

  - slot_name

    流复制槽名称。

    取值范围：字符串，不支持除字母，数字,以及（_?-.）以外的字符。

  - plugin_name

    插件名称。

    取值范围：字符串，当前只支持“vastbase_decoding”。

  返回值类型：name, text

  备注：第一个返回值表示 slot_name，第二个返回值表示该逻辑复制槽解码的起始 LSN 位置。

- pg_drop_replication_slot('slot_name')

  描述：删除流复制槽。

  参数说明：

  - slot_name

    流复制槽名称。

    取值范围：字符串，不支持除字母，数字,以及（_?-.）以外的字符。

  返回值类型：void

- pg_logical_slot_peek_changes('slot_name', 'LSN', upto_nchanges, 'options_name', 'options_value')

  描述：解码并不推进流复制槽（下次解码可以再次获取本次解出的数据）。

  参数说明：

  - slot_name

    流复制槽名称。

    取值范围：字符串，不支持除字母，数字,以及（_?-.）以外的字符。

– LSN

日志的 LSN，表示只解码小于等于此 LSN 的日志。

取值范围：字符串（LSN，格式为 xlogid/xrecoff），如'1/2AAFC60'。为 NULL 时表示不对解码截止的日志位置做限制。

– upto_nchanges

解码条数（包含 begin 和 commit）。假设一共有三条事务，分别包含 3、5、7 条记录，如果 upto_nchanges 为 4，那么会解码出前两个事务共 8 条记录。解码完第二条事务时发现解码条数记录大于等于 upto_nchanges，会停止解码。

取值范围：非负整数。

说明

LSN 和 upto_nchanges 中任一参数达到限制，解码都会结束。

– options：此项为可选参数。

■ include-xids

解码出的 data 列是否包含 xid 信息。

取值范围：0 或 1，默认值为 1。

✧0：设为 0 时，解码出的 data 列不包含 xid 信息。

✧1：设为 1 时，解码出的 data 列包含 xid 信息。

■ skip-empty-xacts

解码时是否忽略空事务信息。

取值范围：0 或 1，默认值为 0。

✧0：设为 0 时，解码时不忽略空事务信息。

✧1：设为 1 时，解码时会忽略空事务信息。

■ include-timestamp

解码信息是否包含 commit 时间戳。

取值范围：0 或 1，默认值为 0。

✧0：设为 0 时，解码信息不包含 commit 时间戳。

✧1：设为 1 时，解码信息包含 commit 时间戳。

返回值类型：text, uint, text

备注：函数返回解码结果，每一条解码结果包含三列，对应上述返回值类型，分别表示 LSN 位置、xid 和解码内容。

❖ pg_logical_slot_get_changes('slot_name', 'LSN', upto_nchanges, 'options_name', 'options_value')

描述：解码并推进流复制槽。

参数说明： 与 pg_logical_slot_peek_changes 一致，详细内容请参见 pg_logical_slot_peek_ch...。

- pg_replication_slot_advance ('slot_name', 'LSN')

  描述：直接推进流复制槽到指定 LSN，不输出解码结果。

  参数说明：

  - slot_name

    流复制槽名称。

    取值范围：字符串，不支持除字母，数字,以及（_?-.）以外的字符。

  - LSN

    推进到的日志 LSN 位置，下次解码时只会输出提交位置比该 LSN 大的事务结果。如果输入的 LSN 比当前流复制槽记录的推进位置还要小，则直接返回；如果输入的 LSN 比当前最新物理日志 LSN 还要大，则推进到当前最新物理日志 LSN。

    取值范围：字符串（LSN，格式为 xlogid/xrecoff）。

  返回值类型：name, text

  备注：返回值分别对应 slot_name 和实际推进至的 LSN。

11.5.23.9. 其它函数

- pgxc_pool_check()

  描述：检查连接池中缓存的连接数据是否与 pgxc_node 一致。

  返回值类型：Boolean

- pgxc_pool_reload()

  描述：更新连接池中缓存的连接信息。

  返回值类型：Boolean

- pgxc_lock_for_backup()

  描述：为备份操作给 Vastbase 加锁，这些备份是为在新增节点上做恢复。

  返回值类型：Boolean

- plan_seed()

  描述：获取前一次查询语句的 seed 值（内部使用）。

  返回值类型：int

- pg_stat_get_env()

  描述：提供获取当前节点的环境变量信息。

  返回值类型：record

- pg_stat_get_thread()

  描述：提供当前节点下所有线程的状态信息。

  返回值类型：record

- pg_stat_get_sql_count()

  描述：提供当前节点中所有用户执行的 SELECT/UPDATE/INSERT/DELETE/MERGE INTO 语句的计数结果。

  返回值类型：record

- copy_error_log_create()

  描述：创建 COPY FROM 容错机制所需要的错误表（public.pgxc_copy_error_log）。

  返回值类型：Boolean

说明

- 此函数会尝试创建 public.pgxc_copy_error_log 表，表的详细信息请参见表 11-32。
- 在 relname 列上创建 B-tree 索引，并 REVOKE ALL on public.pgxc_copy_error_log FROM public 对错误表进行权限控制（与 COPY 语句权限一致）。
- 由于尝试创建的 public.pgxc_copy_error_log 定义是一张行存表，因此 Vastbase 上必须支持行存表的创建才能够正常运行此函数，并使用后续的 COPY 容错功能。需要特别注意的是，enable_hadoop_env 这个 GUC 参数开启后会禁止在 Vastbase 内创建行存表（Vastbase 默认为 off）。
- 此函数自身权限为 Sysadmin 及以上（与错误表、COPY 权限一致）。
- 若创建前 public.pgxc_copy_error_log 表已存在或者 copy_error_log_relname_idx 索引已存在，则此函数会报错回滚。

表 11-32. 错误表 public.pgxc_copy_error_log 信息

列名称	类型	描述
relname	varchar	表名称。以模式名.表名形式显示。
begintime	timestamp with time zone	出现数据格式错误的时间。
filename	character varying	出现数据格式错误的数据源文件名。
rownum	bigint	在数据源文件中，出现数据格式错误的行号。
rawrecord	text	在数据源文件中，出现数据格式错误的原始记录。
detail	text	详细错误信息。

❖ pg_stat_get_data_senders()

描述：提供当前活跃的数据复制发送线程的详细信息。

返回值类型：record

❖ generate_wdr_report(begin_snap_id Oid, end_snap_id Oid, int report_type, int report_scope, int node_name)

描述：基于两个 snapshot 生成系统诊断报告。

返回值类型：record

表 11-33. generate_wdr_report 参数说明

参数	说明	取值范围
begin_snap_id	生成某段时间内性能诊断报告的开始 snapshotid。	-
end_snap_id	结束 snapshot 的 id，默认 end_snap_id 大于 begin_snap_id。	-
report_type	指定生成 report 的类型。	• summary • detail • all，即同时包含 summary 和 detail。
report_scope	指定生成 report 的范围。	node：Vastbase 中的节点。
node_name	在“report_scope”指定为“node”时，需要把该参数指定为对应节点的名称。	-

❖ create_wdr_snapshot(stat text, dbid Oid)

描述：手工生成系统诊断快照。

返回值类型：record

表 11-34. create_wdr_snapshot 参数说明

参数	描述
stat	表示每次 snapshot 时节点的名称，默认为收集所有节点的状态信息。
dbid	表示数据库的 ID，收集指定数据库的状态信息。

- wdr_xdb_query(db_name_str text, query text)

  描述：提供本地跨数据库执行 query 的能力。例如: 在连接到 vastbase 库时，访问 test 库下的表。

  ```
  select col1 from wdr_xdb_query('dbname=test','select col1 from t1') as dd(col1 int);
  ```

  返回值类型：record

- pg_wlm_jump_queue(pid int)

  描述：调整任务到数据库主节点队列的最前端。

  返回值类型：boolean

  - true：成功。
  - false：失败。

- gs_wlm_switch_cgroup(pid int, cgroup text)

  描述：调整作业的优先级到新控制组。

  返回值类型：boolean

  - true：成功。
  - false：失败。

- pv_session_memctx_detail(threadid tid, MemoryContextName text)

  描述：将线程 tid 的 MemoryContextName 内存上下文信息记录到“/tmp/dumpmem”目录下的“threadid_timestamp.log”文件中。其中 threadid 可通过表 PV_SESSION_MEMORY_DETAIL 中的 sessid 后获得。当 MemoryContextName 为 '' 时，会记录所有的内存上下文信息。

  返回值类型：boolean

  - true：成功。
  - false：失败。

- pg_shared_memctx_detail(MemoryContextName text)

  描述：将 MemoryContextName 内存上下文信息记录到“/tmp/dumpmem”目录下的“threadid_timestamp.log”文件中。

  返回值类型：boolean

  - true：成功。
  - false：失败。

说明

pgxc_lock_for_backup 是在使用 vb_dump 或 vb_dumpall 工具备份 Vastbase 前，用来给 Vastbase 加锁的。当给 Vastbase 加锁后，不允许有改变系统结构的操作。该函数不影响 DML 语句。

11.5.24.统计信息函数

统计信息函数根据访问对象分为两种类型：针对某个数据库进行访问的函数，以数据库中每个表或索引的 OID 作为参数，标识需要报告的数据库；针对某个服务器进行访问的函数，以一个服务器进程号为参数，其范围从 1 到当前活跃服务器的数目。

- pg_stat_get_db_numbackends(oid)

  描述：处理该数据库活跃的服务器进程数目。

  返回值类型：integer

- pg_stat_get_db_xact_commit(oid)

  描述：数据库中已提交事务的数量。

  返回值类型：bigint

- pg_stat_get_db_xact_rollback(oid)

  描述：数据库中回滚事务的数量。

  返回值类型：bigint

- pg_stat_get_db_blocks_fetched(oid)

  描述：数据库中磁盘块抓取请求的总数。

  返回值类型：bigint

- pg_stat_get_db_blocks_hit(oid)

  描述：数据库在缓冲区中找到的磁盘块抓取请求的总数。

  返回值类型：bigint

- pg_stat_get_db_tuples_returned(oid)

  描述：为数据库返回的 Tuple 数。

  返回值类型：bigint

- pg_stat_get_db_tuples_fetched(oid)

  描述：为数据库中获取的 Tuple 数。

  返回值类型：bigint

- pg_stat_get_db_tuples_inserted(oid)

  描述：在数据库中插入 Tuple 数。

  返回值类型：bigint

- pg_stat_get_db_tuples_updated(oid)

  描述：在数据库中更新的 Tuple 数。

返回值类型：bigint

- pg_stat_get_db_tuples_deleted(oid)

  描述：数据库中删除 Tuple 数。

  返回值类型：bigint

- pg_stat_get_db_conflict_lock(oid)

  描述：数据库中锁冲突的数量。

  返回值类型：bigint

- pg_stat_get_db_deadlocks(oid)

  描述：数据库中死锁的数量。

  返回值类型：bigint

- pg_stat_get_numscans(oid)

  描述：如果参数是一个表，则顺序扫描读取的行数目。如果参数是一个索引，则返回索引行的数目。

  返回值类型：bigint

- pg_stat_get_tuples_returned(oid)

  描述：如果参数是一个表，则顺序扫描读取的行数目。如果参数是一个索引，则返回的索引行的数目。

  返回值类型：bigint

- pg_stat_get_tuples_fetched(oid)

  描述：如果参数是一个表，则位图扫描抓取的行数目。如果参数是一个索引，则用简单索引扫描抓取的行数目。

  返回值类型：bigint

- pg_stat_get_tuples_inserted(oid)

  描述：插入表中行的数量。

  返回值类型：bigint

- pg_stat_get_tuples_updated(oid)

  描述：在表中已更新行的数量。

  返回值类型：bigint

- pg_stat_get_tuples_deleted(oid)

  描述：从表中删除行的数量。

  返回值类型：bigint

- pg_stat_get_tuples_changed(oid)

  描述：该表上一次 analyze 或 autoanalyze 之后插入、更新、删除行的总数量。

  返回值类型：bigint

- pg_stat_get_tuples_hot_updated(oid)

  描述：热更新的行数表。

  返回值类型：bigint

- pg_stat_get_live_tuples(oid)

  描述：活行数表。

  返回值类型：bigint

- pg_stat_get_dead_tuples(oid)

  描述：死行数表。

  返回值类型：bigint

- pg_stat_get_blocks_fetched(oid)

  描述：表或者索引的磁盘块抓取请求的数量。

  返回值类型：bigint

- pg_stat_get_blocks_hit(oid)

  描述：在缓冲区中找到的表或者索引的磁盘块请求数目。

  返回值类型：bigint

- pg_stat_get_partition_tuples_inserted(oid)

  描述：插入相应表分区中行的数量。

  返回值类型：bigint

- pg_stat_get_partition_tuples_updated(oid)

  描述：在相应表分区中已更新行的数量。

  返回值类型：bigint

- pg_stat_get_partition_tuples_deleted(oid)

  描述：从相应表分区中删除行的数量。

  返回值类型：bigint

- pg_stat_get_partition_tuples_changed(oid)

  描述：该表分区上一次 analyze 或 autoanalyze 之后插入、更新、删除行的总数量。

  返回值类型：bigint

- pg_stat_get_partition_live_tuples(oid)

  描述：活行数表分区。

  返回值类型：bigint

- pg_stat_get_partition_dead_tuples(oid)

  描述：死行数表分区。

  返回值类型：bigint

- pg_stat_get_xact_tuples_inserted(oid)

  描述：表相关的活跃子事务中插入的 tuple 数。

  返回值类型：bigint

- pg_stat_get_xact_tuples_deleted(oid)

  描述：表相关的活跃子事务中删除的 tuple 数。

  返回值类型：bigint

- pg_stat_get_xact_tuples_hot_updated(oid)

  描述：表相关的活跃子事务中热更新的 tuple 数。

  返回值类型：bigint

- pg_stat_get_xact_tuples_updated(oid)

  描述：表相关的活跃子事务中更新的 tuple 数。

  返回值类型：bigint

- pg_stat_get_xact_partition_tuples_inserted(oid)

  描述：表分区相关的活跃子事务中插入的 tuple 数。

  返回值类型：bigint

- pg_stat_get_xact_partition_tuples_deleted(oid)

  描述：表分区相关的活跃子事务中删除的 tuple 数。

  返回值类型：bigint

- pg_stat_get_xact_partition_tuples_hot_updated(oid)

  描述：表分区相关的活跃子事务中热更新的 tuple 数。

  返回值类型：bigint

- pg_stat_get_xact_partition_tuples_updated(oid)

  描述：表分区相关的活跃子事务中更新的 tuple 数。

  返回值类型：bigint

- pg_stat_get_last_vacuum_time(oid)

  描述：用户在该表上最后一次手动启动清理或者 autovacuum 线程启动清理的时间。

  返回值类型：timestamptz

- pg_stat_get_last_autovacuum_time(oid)

  描述：autovacuum 守护进程在该表上最后一次启动清理的时间。

  返回值类型：timestamptz

- pg_stat_get_vacuum_count(oid)

  描述：用户在该表上手动启动清理的次数。

  返回值类型：bigint

- pg_stat_get_autovacuum_count(oid)

  描述：autovacuum 守护进程在该表上启动清理的次数。

  返回值类型：bigint

- pg_stat_get_last_analyze_time(oid)

  描述：用户在该表上最后一次手动启动分析或者 autovacuum 线程启动分析的时间。

  返回值类型：timestamptz

- pg_stat_get_last_autoanalyze_time(oid)

  描述：autovacuum 守护进程在该表上最后一次启动分析的时间。

  返回值类型：timestamptz

- pg_stat_get_analyze_count(oid)

  描述：用户在该表上手动启动分析的次数。

  返回值类型：bigint

- pg_stat_get_autoanalyze_count(oid)

  描述：autovacuum 守护进程在该表上启动分析的次数。

  返回值类型：bigint

- pg_total_autovac_tuples(bool,bool)

  描述：返回 total autovac 相关的 tuple 记录，如 nodename,nspname,relname 以及各类 tuple 的 IUD 信息，入参分别为：是否查询 relation 信息，是否查询 local 信息。

  返回值类型：setofrecord

- pg_autovac_status(oid)

描述：返回和 autovac 状态相关的参数信息，如 nodename,nspname,relname,analyze,vacuum 设置，analyze/vacuum 阈值, analyze/vacuum tuple 数等。

返回值类型：setofrecord

- pg_autovac_timeout(oid)

  描述：返回某个表做 autovac 连续超时的次数，表信息非法或 node 信息异常返回 NULL。

  返回值类型：bigint

- pg_autovac_dbnode(oid)

  描述：返回对某个表做 autovac 的 dbnode 名称，表信息非法或 node 信息异常返回 NULL。

  返回值类型：text

- pg_stat_get_last_data_changed_time(oid)

  描述：insert/update/delete, exchange/truncate/drop partition 在该表上最后一次操作的时间，14.3.42PG_STAT_ALL_TABLES 视图 last_data_changed 列的数据是通过该函数求值，在表数量很大的场景中，通过视图获取表数据最后修改时间的性能较差，建议直接使用该函数获取表数据的最后修改时间。

  返回值类型：timestamptz

- pg_stat_set_last_data_changed_time(oid)

  描述：手动设置该表上最后一次 insert/update/delete, exchange/truncate/drop partition 操作的时间。

  返回值类型：void

- pg_backend_pid()

  描述：当前会话的服务器线程的线程 ID。

  返回值类型：integer

- pg_stat_get_activity(integer)

  描述：返回一个关于带有特殊 PID 的后台进程的记录信息，当参数为 NULL 时，则返回每个活动的后台进程的记录。返回结果是 14.3.40PG_STAT_ACTIVITY 视图中的一个子集，不包含 connection_info 列。

  返回值类型：setofrecord

- pg_stat_get_activity_with_conninfo(integer)

  描述：返回一个关于带有特殊 PID 的后台进程的记录信息，当参数为 NULL 时，则返回每个活动的后台进程的记录。返回结果是 14.3.40PG_STAT_ACTIVITY 视图中的一个子集。

  返回值类型：setofrecord

- pg_user_iostat(text)

  描述：显示和当前用户执行作业正在运行时的 IO 负载管理相关信息。

  返回值类型：record

  函数返回字段说明如下：

名称	类型	描述
userid	oid	用户 id。
min_curr_iops	int4	当前该用户 io 在数据库节点中的最小值。对于行存，以万次/s 为单位；对于列存，以次/s 为单位。
max_curr_iops	int4	当前该用户 io 在数据库节点中的最大值。对于行存，以万次/s 为单位；对于列存，以次/s 为单位。
min_peak_iops	int4	该用户 io 峰值中，数据库节点的最小值。对于行存，以万次/s 为单位；对于列存，以次/s 为单位。
max_peak_iops	int4	该用户 io 峰值中，数据库节点的最大值。对于行存，以万次/s 为单位；对于列存，以次/s 为单位。
io_limits	int4	用户指定的资源池所设置的 io_limits。对于行存，以万次/s 为单位；对于列存，以次/s 为单位。
io_priority	text	该用户所设 io_priority。对于行存，以万次/s 为单位；对于列存，以次/s 为单位。

- pg_stat_get_function_calls(oid)

  描述：函数已被调用次数。

  返回值类型：bigint

- pg_stat_get_function_time(oid)

  描述：该函数花费的总挂钟时间，单位为微秒。包括在这个函数调用所花费的时间。

  返回值类型：bigint

- pg_stat_get_function_self_time(oid)

  描述：只有在此功能所花费的时间。在所谓的功能所花费的时间被排除在外。

  返回值类型：bigint

- pg_stat_get_backend_idset()

描述：设置当前活动的服务器进程数（从 1 到活动服务器进程的数量）。

返回值类型：setofinteger

- pg_stat_get_backend_pid(integer)

  描述：给定的服务器线程的线程 ID。

  返回值类型：bigint

- pg_stat_get_backend_dbid(integer)

  描述：给定服务器进程的数据库 ID。

  返回值类型：oid

- pg_stat_get_backend_userid(integer)

  描述：给定服务器进程的用户 ID。

  返回值类型：oid

- pg_stat_get_backend_activity(integer)

  描述：给定服务器进程的当前活动查询，仅在调用者是系统管理员或被查询会话的用户，并且打开 track_activities 的时候才能获得结果。

  返回值类型：text

- pg_stat_get_backend_waiting(integer)

  描述：如果给定服务器进程在等待某个锁，并且调用者是系统管理员或被查询会话的用户，并且打开 track_activities 的时候才返回真。

  返回值类型：Boolean

- pg_stat_get_backend_activity_start(integer)

  描述：给定服务器进程当前正在执行的查询的起始时间，仅在调用者是系统管理员或被查询会话的用户，并且打开 track_activities 的时候才能获得结果。

  返回值类型：timestampwithtimezone

- pg_stat_get_backend_xact_start(integer)

  描述：给定服务器进程当前正在执行的事务的开始时间，但只有当前用户是系统管理员或被查询会话的用户，并且打开 track_activities 的时候才能获得结果。

  返回值类型：timestampwithtimezone

- pg_stat_get_backend_start(integer)

  描述：给定服务器进程启动的时间，如果当前用户不是系统管理员或被查询的后端的用户，则返回 NULL。

  返回值类型：timestampwithtimezone

- pg_stat_get_backend_client_addr(integer)

  描述：连接到给定客户端后端的 IP 地址。如果是通过 Unix 域套接字连接的则返回 NULL；如果当前用户不是系统管理员或被查询会话的用户，也返回 NULL。

  返回值类型：inet

- pg_stat_get_backend_client_port(integer)

  描述：连接到给定客户端后端的 TCP 端口。如果是通过 Unix 域套接字连接的则返回-1；如果当前用户不是系统管理员或被查询会话的用户，也返回 NULL。

  返回值类型：integer

- pg_stat_get_bgwriter_timed_checkpoints()

  描述：后台写进程开启定时检查点的时间（因为 checkpoint_timeout 时间已经过期了）。

  返回值类型：bigint

- pg_stat_get_bgwriter_requested_checkpoints()

  描述：后台写进程开启基于后端请求的检查点的时间，因为已经超过了 checkpoint_segments 或因为已经执行了 CHECKPOINT。

  返回值类型：bigint

- pg_stat_get_bgwriter_buf_written_checkpoints()

  描述：在检查点期间后台写进程写入的缓冲区数目。

  返回值类型：bigint

- pg_stat_get_bgwriter_buf_written_clean()

  描述：为日常清理脏块，后台写进程写入的缓冲区数目。

  返回值类型：bigint

- pg_stat_get_bgwriter_maxwritten_clean()

  描述：后台写进程停止清理扫描的时间，因为已经写入了更多的缓冲区（相比 bgwriter_lru_maxpages 参数声明的缓冲区数）。

  返回值类型：bigint

- pg_stat_get_buf_written_backend()

  描述：后端进程写入的缓冲区数，因为它们需要分配一个新的缓冲区。

  返回值类型：bigint

- pg_stat_get_buf_alloc()

  描述：分配的总缓冲区数。

  返回值类型：bigint

- pg_stat_clear_snapshot()

  描述：清理当前的统计快照。

  返回值类型：void

- pg_stat_reset()

  描述：为当前数据库重置统计计数器为 0（需要系统管理员权限）。

  返回值类型：void

- pg_stat_reset_shared(text)

  描述：重置 shared cluster 每个节点当前数据统计计数器为 0（需要系统管理员权限）。

  返回值类型：void

- pg_stat_reset_single_table_counters(oid)

  描述：为当前数据库中的一个表或索引重置统计为 0（需要系统管理员权限）。

  返回值类型：void

- pg_stat_reset_single_function_counters(oid)

  描述：为当前数据库中的一个函数重置统计为 0（需要系统管理员权限）。

  返回值类型：void

- pg_stat_session_cu(int, int, int)

  描述：获取当前节点所运行 session 的 CU 命中统计信息。

  返回值类型：record

- gs_get_stat_session_cu(text, int, int, int)

  描述：获取 Vastbase 所有运行 session 的 CU 命中统计信息。

  返回值类型：record

- gs_get_stat_db_cu(text, text, int, int, int)

  描述：获取 Vastbase 一个数据库的 CU 命中统计信息。

  返回值类型：record

- pg_stat_get_cu_mem_hit(oid)

  描述：获取当前节点当前数据库中一个列存表的 CU 内存命中次数。

  返回值类型：bigint

- pg_stat_get_cu_hdd_sync(oid)

  描述：获取当前节点当前数据库中一个列存表从磁盘同步读取 CU 次数。

  返回值类型：bigint

- pg_stat_get_cu_hdd_asyn(oid)

  描述：获取当前节点当前数据库中一个列存表从磁盘异步读取 CU 次数。

  返回值类型：bigint

- pg_stat_get_db_cu_mem_hit(oid)

  描述：获取当前节点一个数据库 CU 内存命中次数。

  返回值类型：bigint

- pg_stat_get_db_cu_hdd_sync(oid)

  描述：获取当前节点一个数据库从磁盘同步读取 CU 次数。

  返回值类型：bigint

- pgxc_get_wlm_current_instance_info(text, int default null)

  描述：在数据库主节点上查询当前的资源使用情况，读取内存中还未存到 5.19.2.3 GS_WLM_INSTANCE_HISTORY 系统表的数据。入参分别为节点名称（可以输入 ALL、C、D、实例名称)、每个节点返回的大数量。返回值为 GS_WLM_INSTANCE_HISTORY。

  返回值类型：setofrecord

- pgxc_get_wlm_history_instance_info(text, TIMESTAMP, TIMESTAMP, int default null)

  描述：在数据库主节点上查询历史资源使用情况，读取 GS_WLM_INSTANCE_HISTORY 系统表的数据。入参分别为节点名称(可以输入 ALL、C、D、实例名称)、起始区间时间、结束区间时间和每个实例返回的大数 量。返回值为 GS_WLM_INSTANCE_HISTORY。

  返回值类型：setofrecord

- pg_stat_get_db_cu_hdd_asyn(oid)

  描述：获取当前节点一个数据库从磁盘异步读取 CU 次数。

  返回值类型：bigint

- pg_stat_bad_block(text, int, int, int, int, int, timestamp with time zone, timestamp with time zone)

  描述：获取当前节点自启动后，读取出现 Page/CU 的损坏信息。

  例: select * from pg_stat_bad_block();

  返回值类型：record

- pg_stat_bad_block_clear()

  描述：清理节点记录的读取出现的 Page/CU 损坏信息（需要系统管理员权限）。

  返回值类型：void

- gs_respool_exception_info(pool text)

描述：查看某个资源池关联的查询规则信息。

返回值类型：record

- gs_control_group_info(pool text)

  描述：查看资源池关联的控制组信息

  返回值类型：record

  返回信息如下：

属性	属性值	描述
name	class_a:workload_a1	class 和 workload 名称
class	class_a	Class 控制组名称
workload	workload_a1	Workload 控制组名称
type	DEFWD	控制组类型（Top、CLASS、BAKWD、DEFWD、TSWD）
gid	87	控制组 id
shares	30	占父节点 CPU 资源的百分比
limits	0	占父节点 CPU 核数的百分比
rate	0	Timeshare 中的分配比例
cpucores	0-3	CPU 核心数

- get_instr_workload_info(integer)

  描述：获取数据库主节点上事务量信息，事务时间信息。

  返回值类型：record

属性	属性值	描述
resourcepool_oid	10	资源池的 oid(逻辑同负载等价)
commit_counter	4	前端事务 commit 数量
rollback_counter	1	前端事务 rollback 数量
resp_min	949	前端事务最小响应时间（单位：微秒）
resp_max	201891	前端事务最大响应时间（单位：微秒）

属性	属性值	描述
resp_avg	43564	前端事务平均响应时间(单位：微秒)
resp_total	217822	前端事务总响应时间（单位：微秒）
bg_commit_counter	910	后端事务 commit 数量
bg_rollback_counter	0	后端事务 rollback 数量
bg_resp_min	97	后端事务最小响应时间（单位：微秒）
bg_resp_max	678080687	后端事务最大响应时间（单位：微秒）
bg_resp_avg	327847884	后端事务平均响应时间（单位：微秒）
bg_resp_total	298341575300	后端事务总响应时间（单位：微秒）

- pv_instance_time()

  描述：获取当前节点上各个关键阶段的时间消耗。

  返回值类型：record

Stat_name 属性	属性值	描述
DB_TIME	1062385	所有线程端到端的墙上时间（WALL TIME）消耗总和(单位： 微秒)
CPU_TIME	311777	所有线程 CPU 时间消耗总和(单位： 微秒)
EXECUTION_TIME	380037	消耗在执行器上的时间总和(单位： 微秒)
PARSE_TIME	6033	消耗在 SQL 解析上的时间总和(单位： 微秒)
PLAN_TIME	173356	消耗在执行计划生成上的时间总和(单位： 微秒)
REWRITE_TIME	2274	消耗在查询重写上的时间总和(单位： 微秒)
PL_EXECUTION_TIME	0	消耗在 PL/SQL 执行上的时间总和(单位： 微秒)
PL_COMPILATION_TIME	557	消耗在 SQL 编译上的时间总和(单位： 微秒)
NET_SEND_TIME	1673	消耗在网络发送上的时间总和(单位： 微秒)
DATA_IO_TIME	426622	消耗在数据读写上的时间总和(单位： 微秒)

- DBE_PERF.get_global_instance_time()

  描述：提供 Vastbase 各个关键阶段的时间消耗，仅在数据库主节点上支持查询，查询该函数必须具有 sysadmin 权限。

  返回值类型：record

- get_instr_unique_sql()

  描述：获取当前结点的执行语句（归一化 SQL）信息，查询该函数必须具有 sysadmin 权限。

  返回值类型：record

- reset_unique_sql(text, text, bigint)

  描述：重置系统执行语句（归一化 SQL）信息，执行该函数必须具有 sysadmin 权限。第一个参数取值范围“global/local”，global 表示清理所有节点上的信息，local 表示只清理当前节点；第二参数取值范围“ALL/BY_USERID/BY_CNID”，ALL 表示清理所有信息，BY_USERID 表示通过指定 USERID 清理只属于该用户的 sql 信息，BY_CNID 表示清理系统中涉及到该数据库主节点的 sql 信息；第三个参数表示具体的 CNID 和 USERID，如果第二个参数为 ALL，第三个参数不起作用，可以取任意值。

  返回值类型：boolean

- get_instr_wait_event(NULL)

  描述：获取当前节点 event 等待的统计信息。

  返回值类型：record

- get_instr_user_login()

  描述：获取当前结点的用户登入登出次数信息，查询该函数必须具有 sysadmin 权限。

  返回值类型：record

- get_instr_rt_percentile()

  描述：获取 CCN 结点 SQL 响应时间 P80，P95 分布信息，Vastbase 统一的信息在 CCN 节点上，其他节点查询为 0。

  返回值类型：record

- get_node_stat_reset_time()

  描述：获取当前结点的统计信息重置（重启，主备倒换，数据库删除）时间。

  返回值类型：record

- DBE_PERF.get_global_os_runtime()

  描述：显示当前操作系统运行的状态信息，仅在数据库主节点上支持查询，查询该函数必须具有 sysadmin 权限。

  返回值类型：record

- DBE_PERF.get_global_os_threads()

  描述：提供 Vastbase 中所有正常节点下的线程状态信息，仅在数据库主节点上支持查询，查询该函数必须具有 sysadmin 权限。

  返回值类型：record

- DBE_PERF.get_summary_workload_sql_count()

  描述：提供 Vastbase 中不同负载 SELECT，UPDATE，INSERT，DELETE，DDL， DML，DCL 计数信息，查询该函数必须具有 sysadmin 权限。

  返回值类型：record

- DBE_PERF.get_summary_workload_sql_elapse_time()

  描述：提供 Vastbase 中不同负载 SELECT，UPDATE，INSERT，DELETE，响应时间信息（TOTAL,AVG, MIN, MAX），查询该函数必须具有 sysadmin 权限。

  返回值类型：record

- DBE_PERF.get_global_workload_transaction()

  描述：获取 Vastbase 内所有节点上的事务量信息，事务时间信息，查询该函数必须具有 sysadmin 权限。

  返回值类型：record

- DBE_PERF.get_global_session_stat()

  描述：获取 Vastbase 节点上的会话状态信息，查询该函数必须具有 sysadmin 权限。

  返回值类型：record

说明

状态信息有如下 17 项，commit,rollback,sql,table_scan,blocks_fetched,physical_read_operation, shared_blocks_dirtied,local_blocks_dirtied,shared_blocks_read,local_blocks_read, blocks_read_time,blocks_write_time,sort_imemory,sort_idisk,cu_mem_hit, cu_hdd_sync_read,cu_hdd_asyread

- DBE_PERF.get_global_session_time()

  描述：提供 Vastbase 各节点各个关键阶段的时间消耗，查询该函数必须具有 sysadmin 权限。

  返回值类型：record

- DBE_PERF.get_global_session_memory()

  描述：汇聚各节点的 Session 级别的内存使用情况，包含执行作业在数据节点上 Postgres 线程和 Stream 线程分配的所有内存，单位为 MB，查询该函数必须具有 sysadmin 权限。

  返回值类型：record

- DBE_PERF.get_global_session_memory_detail()

描述：汇聚各节点的线程的内存使用情况，以 MemoryContext 节点来统计，查询该函数必须具有 sysadmin 权限。

返回值类型：record

- gs_session_memory_detail_tp

  描述：统计线程的内存使用情况，以 MemoryContext 节点来统计。当开启线程池（enable_thread_pool = on）时，该视图包含所有的线程和会话的内存使用情况。

  返回值类型：record

- DBE_PERF.get_global_session_stat_activity()

  描述：汇聚 Vastbase 内各节点上正在运行的线程相关的信息，查询该函数必须具有 sysadmin 权限。

  返回值类型：record

- DBE_PERF.get_global_thread_wait_status()

  描述：汇聚所有结点上工作线程（backend thread）以及辅助线程（auxiliary thread）的阻塞等待情况，查询该函数必须具有 sysadmin 权限。

  返回值类型：record

- DBE_PERF.get_wlm_user_resource_runtime()

  描述：显示所有用户资源使用情况，参数 use_workload_manager 为 on 时才有效，查询该函数必须具有 sysadmin 权限。

  返回值类型：record

- DBE_PERF.get_global_operator_history_table()

  描述：汇聚当前用户数据库主节点上执行作业结束后的算子相关记录（持久化），查询该函数必须具有 sysadmin 权限。

  返回值类型：record

- DBE_PERF.get_global_operator_history()

  描述：汇聚当前用户数据库主节点上执行作业结束后的算子相关记录，查询该函数必须具有 sysadmin 权限。

  返回值类型：record

- DBE_PERF.get_global_operator_runtime()

  描述：汇聚当前用户数据库主节点上执行作业实时的算子相关记录，查询该函数必须具有 sysadmin 权限。

  返回值类型：record

- DBE_PERF.get_global_statement_complex_history()

描述：汇聚当前用户数据库主节点上复杂查询的历史记录，查询该函数必须具有 sysadmin 权限。

返回值类型：record

- DBE_PERF.get_global_statement_complex_history_table()

  描述：汇聚当前用户数据库主节点上复杂查询的历史记录（持久化），查询该函数必须具有 sysadmin 权限。

  返回值类型：record

- DBE_PERF.get_global_statement_complex_runtime()

  描述：汇聚当前用户数据库主节点上复杂查询的实时信息，查询该函数必须具有 sysadmin 权限。

  返回值类型：record

- DBE_PERF.get_global_memory_node_detail()

  描述：汇聚所有节点某个数据库节点内存使用情况，查询该函数必须具有 sysadmin 权限。

  返回值类型：record

- DBE_PERF.get_global_shared_memory_detail()

  描述：汇聚所有节点已产生的共享内存上下文的使用信息，查询该函数必须具有 sysadmin 权限。

  返回值类型：record

- DBE_PERF.get_global_statio_all_indexes

  描述：汇聚所有节点当前数据库中的每个索引行，显示特定索引的 I/O 的统计，查询该函数必须具有 sysadmin 权限。

  返回值类型：record

- DBE_PERF.get_local_toastname_and_toastindexname()

  描述：提供本地 toast 表的 name 和 index 和其关联表的对应关系。

  返回值类型：record

- DBE_PERF.get_summary_statio_all_indexes()

  描述：统计所有节点当前数据库中的每个索引行，显示特定索引的 I/O 的统计，查询该函数必须具有 sysadmin 权限。

  返回值类型：record

- DBE_PERF.get_global_statio_all_sequences()

  描述：提供命名空间中所有 sequences 的 IO 状态信息，查询该函数必须具有 sysadmin 权限。

返回值类型：record

- DBE_PERF.get_global_statio_all_tables()

  描述：汇聚各节点的数据库中每个表 I/O 的统计，查询该函数必须具有 sysadmin 权限。

  返回值类型：record

- DBE_PERF.get_summary_statio_all_tables()

  描述：统计 Vastbase 内数据库中每个表 I/O 的统计，查询该函数必须具有 sysadmin 权限。

  返回值类型：record

- DBE_PERF.get_local_toast_relation()

  描述：提供本地 toast 表的 name 和其关联表的对应关系，查询该函数必须具有 sysadmin 权限

  返回值类型：record

- DBE_PERF.get_global_statio_sys_indexes()

  描述：汇聚各节点的命名空间中所有系统表索引的 IO 状态信息，查询该函数必须具有 sysadmin 权限。

  返回值类型：record

- DBE_PERF.get_summary_statio_sys_indexes()

  描述：统计各节点的命名空间中所有系统表索引的 IO 状态信息，查询该函数必须具有 sysadmin 权限。

  返回值类型：record

- DBE_PERF.get_global_statio_sys_sequences()

  描述：提供命名空间中所有系统表为 sequences 的 IO 状态信息，查询该函数必须具有 sysadmin 权限。

  返回值类型：record

- DBE_PERF.get_global_statio_sys_tables()

  描述：提供各节点的命名空间中所有系统表的 IO 状态信息，查询该函数必须具有 sysadmin 权限。

  返回值类型：record

- DBE_PERF.get_summary_statio_sys_tables()

  描述：Vastbase 内汇聚命名空间中所有系统表的 IO 状态信息，查询该函数必须具有 sysadmin 权限。

  返回值类型：record

- DBE_PERF.get_global_statio_user_indexes()

  描述:各节点的命名空间中所有用户关系表索引的IO状态信息,查询该函数必须具有sysadmin权限。

  返回值类型：record

- DBE_PERF.get_summary_statio_user_indexes()

  描述：Vastbase 内汇聚命名空间中所有用户关系表索引的 IO 状态信息，查询该函数必须具有 sysadmin 权限。

  返回值类型：record

- DBE_PERF.get_global_statio_user_sequences()

  描述：显示各节点的命名空间中所有用户的 sequences 的 IO 状态信息，查询该函数必须具有 sysadmin 权限。

  返回值类型：record

- DBE_PERF.get_global_statio_user_tables()

  描述:显示各节点的命名空间中所有用户关系表的IO状态信息,查询该函数必须具有sysadmin权限。

  返回值类型：record

- DBE_PERF.get_summary_statio_user_tables()

  描述：Vastbase 内汇聚命名空间中所有用户关系表的 IO 状态信息，查询该函数必须具有 sysadmin 权限。

  返回值类型：record

- DBE_PERF.get_stat_db_cu()

  描述：视图查询 Vastbase 各个节点，每个数据库的 CU 命中情况，查询该函数必须具有 sysadmin 权限。

  返回值类型：record

- DBE_PERF.get_global_stat_all_indexes()

  描述：汇聚所有结点数据库中每个索引的统计信息，查询该函数必须具有 sysadmin 权限。

  返回值类型：record

- DBE_PERF.get_summary_stat_all_indexes()

  描述：统计所有结点数据库中每个索引的统计信息，查询该函数必须具有 sysadmin 权限。

  返回值类型：record

- DBE_PERF.get_global_stat_sys_tables()

描述：汇聚各节点 pg_catalog、information_schema 模式的所有命名空间中系统表的统计信息，查询该函数必须具有 sysadmin 权限。

返回值类型：record

- DBE_PERF.get_summary_stat_sys_tables()

  描述：统计各节点 pg_catalog、information_schema 模式的所有命名空间中系统表的统计信息，查询该函数必须具有 sysadmin 权限。

  返回值类型：record

- DBE_PERF.get_global_stat_sys_indexes()

  描述：汇聚各节点 pg_catalog、information_schema 模式中所有系统表的索引状态信息，查询该函数必须具有 sysadmin 权限。

  返回值类型：record

- DBE_PERF.get_summary_stat_sys_indexes()

  描述：统计各节点 pg_catalog、information_schema 模式中所有系统表的索引状态信息，查询该函数必须具有 sysadmin 权限。

  返回值类型：record

- DBE_PERF.get_global_stat_user_tables()

  描述：汇聚所有命名空间中用户自定义普通表的状态信息，查询该函数必须具有 sysadmin 权限。

  返回值类型：record

- DBE_PERF.get_summary_stat_user_tables()

  描述：统计所有命名空间中用户自定义普通表的状态信息，查询该函数必须具有 sysadmin 权限。

  返回值类型：record

- DBE_PERF.get_global_stat_user_indexes()

  描述：汇聚所有数据库中用户自定义普通表的索引状态信息，查询该函数必须具有 sysadmin 权限。

  返回值类型：record

- DBE_PERF.get_summary_stat_user_indexes()

  描述：统计所有数据库中用户自定义普通表的索引状态信息，查询该函数必须具有 sysadmin 权限。

  返回值类型：record

- DBE_PERF.get_global_stat_database()

描述：汇聚所有节点数据库统计信息，查询该函数必须具有 sysadmin 权限。

返回值类型：record

- DBE_PERF.get_global_stat_database_conflicts()

  描述：统计所有节点数据库统计信息，查询该函数必须具有 sysadmin 权限。

  返回值类型：record

- DBE_PERF.get_global_stat_xact_all_tables()

  描述：汇聚命名空间中所有普通表和 toast 表的事务状态信息，查询该函数必须具有 sysadmin 权限。

  返回值类型：record

- DBE_PERF.get_summary_stat_xact_all_tables()

  描述：统计命名空间中所有普通表和 toast 表的事务状态信息，查询该函数必须具有 sysadmin 权限。

  返回值类型：record

- DBE_PERF.get_global_stat_xact_sys_tables()

  描述：汇聚所有节点命名空间中系统表的事务状态信息，查询该函数必须具有 sysadmin 权限。

  返回值类型：record

- DBE_PERF.get_summary_stat_xact_sys_tables()

  描述：统计所有节点命名空间中系统表的事务状态信息，查询该函数必须具有 sysadmin 权限。

  返回值类型：record

- DBE_PERF.get_global_stat_xact_user_tables()

  描述：汇聚所有节点命名空间中用户表的事务状态信息，查询该函数必须具有 sysadmin 权限。

  返回值类型：record

- DBE_PERF.get_summary_stat_xact_user_tables()

  描述：统计所有节点命名空间中用户表的事务状态信息，查询该函数必须具有 sysadmin 权限。

  返回值类型：record

- DBE_PERF.get_global_stat_user_functions()

  描述：汇聚所有节点命名空间中用户定义函数的事务状态信息，查询该函数必须具有 sysadmin 权限。

  返回值类型：record

- DBE_PERF.get_global_stat_xact_user_functions()

描述：统计所有节点命名空间中用户定义函数的事务状态信息，查询该函数必须具有 sysadmin 权限。

返回值类型：record

- DBE_PERF.get_global_stat_bad_block()

  描述：汇聚所有节点表、索引等文件的读取失败信息，查询该函数必须具有 sysadmin 权限。

  返回值类型：record

- DBE_PERF.get_global_file_redo_iostat()

  描述：统计所有节点表、索引等文件的读取失败信息，查询该函数必须具有 sysadmin 权限。

  返回值类型：record

- DBE_PERF.get_global_file_iostat()

  描述：汇聚所有节点数据文件 IO 的统计，查询该函数必须具有 sysadmin 权限。

  返回值类型：record

- DBE_PERF.get_global_locks()

  描述：汇聚所有节点的锁信息，查询该函数必须具有 sysadmin 权限。

  返回值类型：record

- DBE_PERF.get_global_replication_slots()

  描述：汇聚所有节点上逻辑复制信息，查询该函数必须具有 sysadmin 权限。

  返回值类型：record

- DBE_PERF.get_global_bgwriter_stat()

  描述：汇聚所有节点后端写进程活动的统计信息，查询该函数必须具有 sysadmin 权限。

  返回值类型：record

- DBE_PERF.get_global_replication_stat()

  描述：汇聚各节点日志同步状态信息，如发起端发送日志位置，收端接收日志位置等，查询该函数必须具有 sysadmin 权限。

  返回值类型：record

- DBE_PERF.get_global_transactions_running_xacts()

  描述：汇聚各节点运行事务的信息，查询该函数必须具有 sysadmin 权限。

  返回值类型：record

- DBE_PERF.get_summary_transactions_running_xacts()

  描述：统计各节点运行事务的信息，查询该函数必须具有 sysadmin 权限。

  返回值类型：record

- DBE_PERF.get_global_transactions_prepared_xacts()

  描述：汇聚各节点当前准备好进行两阶段提交的事务的信息，查询该函数必须具有 sysadmin 权限。

  返回值类型：record

- DBE_PERF.get_summary_transactions_prepared_xacts()

  描述：统计各节点当前准备好进行两阶段提交的事务的信息，查询该函数必须具有 sysadmin 权限。

  返回值类型：record

- DBE_PERF.get_summary_statement()

  描述：汇聚各节点历史执行语句状态信息，查询该函数必须具有 sysadmin 权限。

  返回值类型：record

- DBE_PERF.get_global_statement_count()

  描述：汇聚各节点 SELECT，UPDATE，INSERT，DELETE，响应时间信息（TOTAL,AVG, MIN, MAX），查询该函数必须具有 sysadmin 权限。

  返回值类型：record

- DBE_PERF.get_global_config_settings()

  描述：汇聚各节点 GUC 参数配置信息，查询该函数必须具有 sysadmin 权限。

  返回值类型：record

- DBE_PERF.get_global_wait_events()

  描述：汇聚各节点 wait events 状态信息，查询该函数必须具有 sysadmin 权限。

  返回值类型：record

- DBE_PERF.get_statement_responsetime_percentile()

  描述：获取 VastbaseSQL 响应时间 P80，P95 分布信息，查询该函数必须具有 sysadmin 权限。

  返回值类型：record

- DBE_PERF.get_summary_user_login()

  描述：统计 Vastbase 各节点用户登入登出次数信息，查询该函数必须具有 sysadmin 权限。

  返回值类型：record

- DBE_PERF.get_global_record_reset_time()

  描述：汇聚 Vastbase 统计信息重置（重启，主备倒换，数据库删除）时间，查询该函数必须具有 sysadmin 权限。

返回值类型：record

- gs_wlm_user_resource_info(name text)

  描述：查询具体某个用户的资源限额和资源使用情况。

  返回值类型：record

- get_local_rel_iostat()

  描述：查询当前节点的数据文件 IO 状态累计值。

  返回值类型：record

- DBE_PERF.get_global_rel_iostat()

  描述：汇聚所有节点数据文件 IO 的统计，查询该函数必须具有 sysadmin 权限。

  返回值类型：record

示例：

pg_backend_pid 函数显示当前后台服务线程 ID。

```
vastbase=# SELECT pg_backend_pid();
 pg_backend_pid
-----------------
 139706243217168
(1 row)
```

pg_stat_get_backend_pid 函数显示后台线程 ID。

```
vastbase=# SELECT pg_stat_get_backend_pid(1);
 pg_stat_get_backend_pid
-------------------------
       139706243217168
(1 row)
```

11.5.25.触发器函数

- pg_get_triggerdef(oid)

  描述：获取触发器的定义信息。

  参数：待查触发器的 OID。

  返回值类型：text

  示例:

  ```
  vastbase=# SELECT pg_get_triggerdef(oid) FROM pg_trigger;
                                        pg_get_triggerdef
  ----------------------------------------------------------------------------------------------
  ----------------------------
  (0 rows)
  ```

- pg_get_triggerdef(oid, boolean)

描述：获取触发器的定义信息。

参数：待查触发器的 OID 及是否以 pretty 方式展示。

返回值类型：text

示例:

```
vastbase=# SELECT pg_get_triggerdef(oid, true) FROM pg_trigger;
                                       pg_get_triggerdef
-----------------------------------------------------------------------------------------------
---------------------------
(0 rows)
```

11.5.26.终止用户会话函数

功能描述

新增一个内置函数，功能为终止用户会话。

示例

```
数据库处于非线程模式:
alter system set enable_thread_pool to off;
show enable_thread_pool;
1.新建会话 1
2.新建会话 2，查询会话 1 的 pid 及 sid
select * from pg_stat_activity;
3.使用超级用户执行终止会话函数
select pg_terminate_session($pid,$sid);
4.会话 1 执行一条 SQL:
select * from pg_stat_activity;
```

11.5.27.其他系统函数

Vastbase 为内建数据类型提供了大量的函数和操作符。用户也可以自行定义自己所需的函数和操作符。PSQL 命令\df 和\do 可以逐个罗列出现有的函数和操作符。如果用户关心这些函数和操作符的可移植性，那请注意，除了算术和比较操作符以及上述直接标注出的函数，大部分本章中描述的函数和操作符是没有被 SQL 标准规定的。本章所介绍的扩充后的函数在其他 SQL 数据库管理系统中也存在，而且很多时候，这些函数的不同版本实现都是兼容且具备一致性的。

Vastbase 的内建函数和操作符继承自开源 PGXC/PG，本小节所列举的函数，相关细节描述请参见 PGXC/PG 官方文档，链接如下：

http://postgres-xc.sourceforge.net/docs/1_1/functions.html

https://www.postgresql.org/docs/9.2/functions.html

abbrev	abs	abstime	abstimeeq	abstimege	abstimegt	abstimein

abstimele	abstimelt	abstimene	abstimeout	abstimerecv	abstimesend	aclcontains
acldefault	aclexplode	aclinsert	aclitemeq	aclitemin	aclitemout	aclremove
acos	add_months	age	any_in	any_out	anyarray_in	anyarray_out
anyarray_recv	anyarray_send	anyelement_in	anyelement_out	anyenum_in	anyenum_out	anynonarray_in

- abbrev
- abs
- abstime
- abstimeeq
- abstimege
- abstimegt
- abstimein
- abstimele
- abstimelt
- abstimene
- abstimeout
- abstimerecv
- abstimesend
- aclcontains
- acldefault
- aclexplode
- aclinsert
- aclitemeq
- aclitemin
- aclitemout
- aclremove
- acos
- add_months
- age
- any_in
- any_out
- anyarray_in
- anyarray_out
- anyarray_recv
- anyarray_send

- anyelement_in
- anyelement_out
- anyenum_in
- anyenum_out
- anynonarray_in
- anynonarray_out
- anyrange_in
- anyrange_out
- anytextcat
- area
- areajoinsel
- areasel
- array_agg
- array_agg_finalfn
- array_agg_transfn
- array_append
- array_cat
- array_dims
- array_eq
- array_extend
- array_fill
- array_ge
- array_gt
- array_in
- array_larger
- array_le
- array_length
- array_lower
- array_lt
- array_ndims
- array_ne
- array_out
- array_prepend
- array_recv
- array_send
- array_smaller
- array_to_json

- array_to_string
- array_typanalyze
- array_upper
- arraycontained
- arraycontains
- arraycontjoinsel
- arraycontsel
- arrayoverlap
- ascii
- asin
- atan
- atan2
- avg
- bigint_tid
- bit
- bit_and
- bit_in
- bit_length
- bit_or
- bit_out
- bit_recv
- bit_send
- bitand
- bitcat
- bitcmp
- biteq
- bitge
- bitgt
- bitle
- bitlt
- bitne
- bitnot
- bitor
- bitshiftleft
- bitshiftright
- bittypmodin
- bittypmodout

- bitxor
- bool
- bool_and
- bool_int1
- bool_int2
- bool_int8
- bool_or
- booland_statefunc
- booleq
- boolge
- boolgt
- boolin
- boolle
- boollt
- boolne
- boolor_statefunc
- boolout
- boolrecv
- boolsend
- box
- box_above
- box_above_eq
- box_add
- box_below
- box_below_eq
- box_center
- box_contain
- box_contain_pt
- box_contained
- box_distance
- box_div
- box_eq
- box_ge
- box_gt
- box_in
- box_intersect
- box_le

- box_left
- box_lt
- box_mul
- box_out
- box_overabove
- box_overbelow
- box_overlap
- box_overleft
- box_overright
- box_recv
- box_right
- box_same
- box_send
- box_sub
- bpchar
- bpchar_date
- bpchar_float4
- bpchar_float8
- bpchar_int4
- bpchar_int8
- bpchar_larger
- bpchar_numeric
- bpchar_pattern_ge
- bpchar_pattern_gt
- bpchar_pattern_le
- bpchar_pattern_lt
- bpchar_smaller
- bpchar_sortsupport
- bpchar_timestamp
- bpcharcmp
- bpchareq
- bpcharge
- bpchargt
- bpchariclike
- bpcharicnlike
- bpcharicregexeq
- bpcharicregexne

- bpcharin
- bpcharle
- bpcharlike
- bpcharlt
- bpcharne
- bpcharnlike
- bpcharout
- bpcharrecv
- bpcharregexeq
- bpcharregexne
- bpcharsend
- bpchartypmodin
- bpchartypmodout
- broadcast
- btabstimecmp
- btarraycmp
- btbeginscan
- btboolcmp
- btbpchar_pattern_cmp
- btbuild
- btbuildempty
- btbulkdelete
- btcanreturn
- btcharcmp
- btcostestimate
- btendscan
- btfloat48cmp
- btfloat4cmp
- btfloat4sortsupport
- btfloat84cmp
- btfloat8cmp
- btfloat8sortsupport
- btgetbitmap
- btgettuple
- btinsert
- btint24cmp
- btint28cmp

- btint2cmp
- btint2sortsupport
- btint42cmp
- btint48cmp
- btint4cmp
- btint4sortsupport
- btint82cmp
- btint84cmp
- btint8cmp
- btint8sortsupport
- btmarkpos
- btmerge
- btnamecmp
- btnamesortsupport
- btoidcmp
- btoidsortsupport
- btoidvectorcmp
- btoptions
- btrecordcmp
- btreltimecmp
- btrescan
- btrestrpos
- btrim
- bttext_pattern_cmp
- bttextcmp
- bttextsortsupport
- bttidcmp
- bttintervalcmp
- btvacuumcleanup
- bucketabstime
- bucketbool
- bucketbpchar
- bucketbytea
- bucketcash
- bucketchar
- bucketdate
- bucketfloat4

- bucketfloat8
- bucketint1
- bucketint2
- bucketint2vector
- bucketint4
- bucketint8
- bucketinterval
- bucketname
- bucketnumeric
- bucketnvarchar2
- bucketoid
- bucketoidvector
- bucketraw
- bucketreltime
- bucketsmalldatetime
- buckettext
- buckettime
- buckettimestamp
- buckettimestamptz
- buckettimetz
- bucketuuid
- bucketvarchar
- bytea_sortsupport
- bytea_string_agg_finalfn
- bytea_string_agg_transfn
- byteacat
- byteacmp
- byteaeq
- byteage
- byteagt
- byteain
- byteale
- bytealike
- bytealt
- byteane
- byteanlike
- byteaout

- bytearecv
- byteasend
- cash_cmp
- cash_div_cash
- cash_div_flt4
- cash_div_flt8
- cash_div_int1
- cash_div_int2
- cash_div_int4
- cash_div_int8
- cash_eq
- cash_ge
- cash_gt
- cash_in
- cash_le
- cash_lt
- cash_mi
- cash_mul_flt4
- cash_mul_flt8
- cash_mul_int1
- cash_mul_int2
- cash_mul_int4
- cash_mul_int8
- cash_ne
- cash_out
- cash_pl
- cash_recv
- cash_send
- cash_words
- cashlarger
- cashsmaller
- cbrt
- cbtreebuild
- cbtreecanreturn
- cbtreecostestimate
- cbtreegetbitmap
- cbtreegettuple

- cbtreeoptions
- ceil
- ceiling
- center
- cginbuild
- cgingetbitmap
- char
- char_length
- character_length
- chareq
- charge
- chargt
- charin
- charle
- charlt
- charne
- charout
- charrecv
- charsend
- check_engine_status
- checksum
- checksumtext_agg_transfn
- chr
- cideq
- cidin
- cidout
- cidr
- cidr_in
- cidr_out
- cidr_recv
- cidr_send
- cidrecv
- cidsend
- circle
- circle_above
- circle_add_pt
- circle_below

- circle_center
- circle_contain
- circle_contain_pt
- circle_contained
- circle_distance
- circle_div_pt
- circle_eq
- circle_ge
- circle_gt
- circle_in
- circle_le
- circle_left
- circle_lt
- circle_mul_pt
- circle_ne
- circle_out
- circle_overabove
- circle_overbelow
- circle_overlap
- circle_overleft
- circle_overright
- circle_recv
- circle_right
- circle_same
- circle_send
- circle_sub_pt
- clock_timestamp
- close_lb
- close_ls
- close_lseg
- close_pb
- close_pl
- close_ps
- close_sb
- close_sl
- col_description
- comm_client_info

- complex_array_in
- concat
- concat_ws
- contjoinsel
- contsel
- convert
- convert_from
- convert_to
- convert_to_nocase
- corr
- cos
- cot
- count
- covar_pop
- covar_samp
- create_wdr_snapshot
- cstore_tid_out
- cstring_in
- cstring_out
- cstring_recv
- cstring_send
- cume_dist
- cupointer_bigint
- current_database
- current_query
- current_schema
- current_schemas
- current_setting
- current_user
- currtid
- currtid2
- currval
- cursor_to_xml
- cursor_to_xmlschema
- database_to_xml
- database_to_xml_and_xmlschema
- database_to_xmlschema

- datalength
- date
- date_bpchar
- date_cmp
- date_cmp_timestamp
- date_cmp_timestamptz
- date_eq
- date_eq_timestamp
- date_eq_timestamptz
- date_ge
- date_ge_timestamp
- date_ge_timestamptz
- date_gt
- date_gt_timestamp
- date_gt_timestamptz
- date_in
- date_larger
- date_le
- date_le_timestamp
- date_le_timestamptz
- date_list_agg_noarg2_transfn
- date_list_agg_transfn
- date_lt
- date_lt_timestamp
- date_lt_timestamptz
- date_mi
- date_mi_interval
- date_mii
- date_ne
- date_ne_timestamp
- date_ne_timestamptz
- date_out
- date_part
- date_pl_interval
- date_pli
- date_recv
- date_send

- date_smaller
- date_sortsupport
- date_text
- date_trunc
- date_varchar
- daterange
- daterange_canonical
- daterange_subdiff
- datetime_pl
- datetimetz_pl
- dcbrt
- decode
- definer_current_user
- degrees
- dense_rank
- dexp
- diagonal
- diameter
- disable_conn
- dispell_init
- dispell_lexize
- dist_cpoly
- dist_lb
- dist_pb
- dist_pc
- dist_pl
- dist_ppath
- dist_ps
- dist_sb
- dist_sl
- div
- dlog1
- dlog10
- domain_in
- domain_recv
- dpow
- dround

- dsimple_init
- dsimple_lexize
- dsqrt
- dsynonym_init
- dsynonym_lexize
- dtrunc
- elem_contained_by_range
- empty_blob
- encode
- encode_plan_node
- enum_cmp
- enum_eq
- enum_first
- enum_ge
- enum_gt
- enum_in
- enum_larger
- enum_last
- enum_le
- enum_lt
- enum_ne
- enum_out
- enum_range
- enum_recv
- enum_send
- enum_smaller
- eqjoinsel
- eqsel
- every
- exec_hadoop_sql
- exec_on_extension
- exp
- f4toi1
- f8toi1
- factorial
- family
- fdw_handler_in

- fdw_handler_out
- first_value
- float4
- float4_accum
- float4_bpchar
- float4_list_agg_noarg2_transfn
- float4_list_agg_transfn
- float4_text
- float4_varchar
- float48div
- float48eq
- float48ge
- float48gt
- float48le
- float48lt
- float48mi
- float48mul
- float48ne
- float48pl
- float4abs
- float4div
- float4eq
- float4ge
- float4gt
- float4in
- float4larger
- float4le
- float4lt
- float4mi
- float4mul
- float4ne
- float4out
- float4pl
- float4recv
- float4send
- float4smaller
- float4um

- float4up
- float8
- float8_accum
- float8_avg
- float8_bpchar
- float8_collect
- float8_corr
- float8_covar_pop
- float8_covar_samp
- float8_interval
- float8_list_agg_noarg2_transfn
- float8_list_agg_transfn
- float8_regr_accum
- float8_regr_avgx
- float8_regr_avgy
- float8_regr_collect
- float8_regr_intercept
- float8_regr_r2
- float8_regr_slope
- float8_regr_sxx
- float8_regr_sxy
- float8_regr_syy
- float8_stddev_pop
- float8_stddev_samp
- float8_text
- float8_var_pop
- float8_var_samp
- float8_varchar
- float84div
- float84eq
- float84ge
- float84gt
- float84le
- float84lt
- float84mi
- float84mul
- float84ne

- float84pl
- float8abs
- float8div
- float8eq
- float8ge
- float8gt
- float8in
- float8larger
- float8le
- float8lt
- float8mi
- float8mul
- float8ne
- float8out
- float8pl
- float8recv
- float8send
- float8smaller
- float8um
- float8up
- floor
- flt4_mul_cash
- flt8_mul_cash
- fmgr_c_validator
- fmgr_internal_validator
- fmgr_sql_validator
- format
- format_type
- generate_series
- generate_subscripts
- generate_wdr_report
- get_gtm_lite_status
- get_hostname
- get_nodename
- get_prepared_pending_xid
- get_schema_oid
- getbucket

- getdatabaseencoding
- getpgusername
- gin_clean_pending_list
- gin_cmp_prefix
- gin_cmp_tslexeme
- gin_extract_tsquery
- gin_extract_tsvector
- gin_tsquery_consistent
- gin_tsquery_triconsistent
- ginarrayconsistent
- ginarrayextract
- ginarraytriconsistent
- ginbeginscan
- ginbuild
- ginbuildempty
- ginbulkdelete
- gincostestimate
- ginendscan
- gingetbitmap
- gininsert
- ginmarkpos
- ginmerge
- ginoptions
- ginqueryarrayextract
- ginrescan
- ginrestrpos
- ginvacuumcleanup
- gist_box_compress
- gist_box_consistent
- gist_box_decompress
- gist_box_penalty
- gist_box_picksplit
- gist_box_same
- gist_box_union
- gist_circle_compress
- gist_circle_consistent
- gist_point_compress

- gist_point_consistent
- gist_point_distance
- gist_poly_compress
- gist_poly_consistent
- gistbeginscan
- gistbuild
- gistbuildempty
- gistbulkdelete
- gistcostestimate
- gistendscan
- gistgetbitmap
- gistgettuple
- gistinsert
- gistmarkpos
- gistmerge
- gistoptions
- gistrescan
- gistrestrpos
- gistvacuumcleanup
- global_comm_get_client_info
- global_comm_get_recv_stream
- global_comm_get_send_stream
- global_comm_get_status
- gs_all_control_group_info
- gs_all_nodegroup_control_group_info
- gs_cgroup_map_ng_conf
- gs_extend_library
- gs_fault_inject
- gs_get_next_xid_csn
- gs_get_nodegroup_tablecount
- gs_password_notifytime
- gs_stat_get_wlm_plan_operator_info
- gs_switch_relfilenode
- gs_total_nodegroup_memory_detail
- gs_wlm_get_resource_pool_info
- gs_wlm_get_session_info
- gs_wlm_get_user_info

- gs_wlm_get_user_session_info
- gs_wlm_get_workload_records
- gs_wlm_node_clean
- gs_wlm_node_recover
- gs_wlm_readjust_user_space
- gs_wlm_readjust_user_space_through_username
- gs_wlm_readjust_user_space_with_reset_flag
- gs_wlm_rebuild_user_resource_pool
- gs_wlm_session_respool
- gtsquery_compress
- gtsquery_consistent
- gtsquery_decompress
- gtsquery_penalty
- gtsquery_picksplit
- gtsquery_same
- gtsquery_union
- gtsvector_compress
- gtsvector_consistent
- gtsvector_decompress
- gtsvector_penalty
- gtsvector_picksplit
- gtsvector_same
- gtsvector_union
- gtsvectorin
- gtsvectorout
- has_any_column_privilege
- has_column_privilege
- has_database_privilege
- has_directory_privilege
- has_foreign_data_wrapper_privilege
- has_function_privilege
- has_language_privilege
- has_nodegroup_privilege
- has_schema_privilege
- has_sequence_privilege
- has_server_privilege
- has_table_privilege

- has_tablespace_privilege
- has_type_privilege
- hash_aclitem
- hash_array
- hash_numeric
- hash_range
- hashbeginscan
- hashbpchar
- hashbuild
- hashbuildempty
- hashbulkdelete
- hashchar
- hashcostestimate
- hashendscan
- hashenum
- hashfloat4
- hashfloat8
- hashgetbitmap
- hashgettuple
- hashinet
- hashinsert
- hashint1
- hashint2
- hashint2vector
- hashint4
- hashint8
- hashmacaddr
- hashmarkpos
- hashmerge
- hashname
- hashoid
- hashoidvector
- hashoptions
- hashrescan
- hashrestrpos
- hashtext
- hashvacuumcleanup

- hashvarlena
- height
- hextoraw
- host
- hostmask
- i1tof4
- i1tof8
- i1toi2
- i1toi4
- i1toi8
- i2toi1
- i4toi1
- i8toi1
- iclikejoinsel
- iclikesel
- icnlikejoinsel
- icnlikesel
- icregexeqjoinsel
- icregexeqsel
- icregexnejoinsel
- icregexnesel
- inet_client_addr
- inet_client_port
- inet_in
- inet_out
- inet_recv
- inet_send
- inet_server_addr
- inet_server_port
- inetand
- inetmi
- inetmi_int8
- inetnot
- inetor
- inetpl
- initcap
- instr

- int1_avg_accum
- int1_bool
- int1_bpchar
- int1_mul_cash
- int1_numeric
- int1_nvarchar2
- int1_text
- int1_varchar
- int1abs
- int1and
- int1cmp
- int1div
- int1eq
- int1ge
- int1gt
- int1in
- int1inc
- int1larger
- int1le
- int1lt
- int1mi
- int1mod
- int1mul
- int1ne
- int1not
- int1or
- int1out
- int1pl
- int1recv
- int1send
- int1shl
- int1shr
- int1smaller
- int1um
- int1up
- int1xor
- int2

- int2_accum
- int2_avg_accum
- int2_bool
- int2_bpchar
- int2_list_agg_noarg2_transfn
- int2_list_agg_transfn
- int2_mul_cash
- int2_sum
- int2_text
- int2_varchar
- int24div
- int24eq
- int24ge
- int24gt
- int24le
- int24lt
- int24mi
- int24mul
- int24ne
- int24pl
- int28div
- int28eq
- int28ge
- int28gt
- int28le
- int28lt
- int28mi
- int28mul
- int28ne
- int28pl
- int2abs
- int2and
- int2div
- int2eq
- int2ge
- int2gt
- int2in

- int2larger
- int2le
- int2lt
- int2mi
- int2mod
- int2mul
- int2ne
- int2not
- int2or
- int2out
- int2pl
- int2recv
- int2send
- int2shl
- int2shr
- int2smaller
- int2um
- int2up
- int2vectoreq
- int2vectorin
- int2vectorout
- int2vectorrecv
- int2vectorsend
- int2xor
- int4
- int4_accum
- int4_avg_accum
- int4_bpchar
- int4_list_agg_noarg2_transfn
- int4_list_agg_transfn
- int4_mul_cash
- int4_sum
- int4_text
- int4_varchar
- int42div
- int42eq
- int42ge

- int42gt
- int42le
- int42lt
- int42mi
- int42mul
- int42ne
- int42pl
- int48div
- int48eq
- int48ge
- int48gt
- int48le
- int48lt
- int48mi
- int48mul
- int48ne
- int48pl
- int4abs
- int4and
- int4div
- int4eq
- int4ge
- int4gt
- int4in
- int4inc
- int4larger
- int4le
- int4lt
- int4mi
- int4mod
- int4mul
- int4ne
- int4not
- int4or
- int4out
- int4pl
- int4range

- int4range_canonical
- int4range_subdiff
- int4recv
- int4send
- int4shl
- int4shr
- int4smaller
- int4um
- int4up
- int4xor
- int8
- int8_accum
- int8_avg
- int8_avg_accum
- int8_avg_collect
- int8_bool
- int8_bpchar
- int8_list_agg_noarg2_transfn
- int8_list_agg_transfn
- int8_mul_cash
- int8_sum
- int8_sum_to_int8
- int8_text
- int8_varchar
- int82div
- int82eq
- int82ge
- int82gt
- int82le
- int82lt
- int82mi
- int82mul
- int82ne
- int82pl
- int84div
- int84eq
- int84ge

- int84gt
- int84le
- int84lt
- int84mi
- int84mul
- int84ne
- int84pl
- int8abs
- int8and
- int8div
- int8eq
- int8ge
- int8gt
- int8in
- int8inc
- int8inc_any
- int8inc_float8_float8
- int8larger
- int8le
- int8lt
- int8mi
- int8mod
- int8mul
- int8ne
- int8not
- int8or
- int8out
- int8pl
- int8pl_inet
- int8range
- int8range_canonical
- int8range_subdiff
- int8recv
- int8send
- int8shl
- int8shr
- int8smaller

- int8um
- int8up
- int8xor
- integer_pl_date
- inter_lb
- inter_sb
- inter_sl
- internal_in
- internal_out
- interval
- interval_accum
- interval_avg
- interval_cmp
- interval_collect
- interval_div
- interval_eq
- interval_ge
- interval_gt
- interval_hash
- interval_in
- interval_larger
- interval_le
- interval_list_agg_noarg2_transfn
- interval_list_agg_transfn
- interval_lt
- interval_mi
- interval_mul
- interval_ne
- interval_out
- interval_pl
- interval_pl_date
- interval_pl_time
- interval_pl_timestamp
- interval_pl_timestamptz
- interval_pl_timetz
- interval_recv
- interval_send

- interval_smaller
- interval_transform
- interval_um
- intervaltypmodin
- intervaltypmodout
- intinterval
- isclosed
- isempty
- isfinite
- ishorizontal
- isopen
- isparallel
- isperp
- isvertical
- json_in
- json_out
- json_recv
- json_send
- justify_days
- justify_hours
- justify_interval
- kill_snapshot
- lag
- language_handler_in
- language_handler_out
- last_day
- last_value
- lastval
- lead
- left
- length
- lengthb
- like
- like_escape
- likejoinsel
- likesel
- line

- line_distance
- line_eq
- line_horizontal
- line_in
- line_interpt
- line_intersect
- line_out
- line_parallel
- line_perp
- line_recv
- line_send
- line_vertical
- list_agg_finalfn
- list_agg_noarg2_transfn
- list_agg_transfn
- listagg
- ln
- lo_close
- lo_creat
- lo_create
- lo_export
- lo_import
- lo_lseek
- lo_open
- lo_tell
- lo_truncate
- lo_unlink
- local_ckpt_stat
- local_double_write_stat
- local_pagewriter_stat
- local_recovery_status
- local_redo_stat
- local_rto_stat
- log
- loread
- lower
- lower_inc

- lower_inf
- lowrite
- lpad
- lseg
- lseg_center
- lseg_distance
- lseg_eq
- lseg_ge
- lseg_gt
- lseg_horizontal
- lseg_in
- lseg_interpt
- lseg_intersect
- lseg_le
- lseg_length
- lseg_lt
- lseg_ne
- lseg_out
- lseg_parallel
- lseg_perp
- lseg_recv
- lseg_send
- lseg_vertical
- ltrim
- macaddr_and
- macaddr_cmp
- macaddr_eq
- macaddr_ge
- macaddr_gt
- macaddr_in
- macaddr_le
- macaddr_lt
- macaddr_ne
- macaddr_not
- macaddr_or
- macaddr_out
- macaddr_recv

- macaddr_send
- makeaclitem
- masklen
- max
- md5
- min
- mktinterval
- mod
- model_train_opt
- money
- mul_d_interval
- multiply
- name
- nameeq
- namege
- namegt
- nameiclike
- nameicnlike
- nameicregexeq
- nameicregexne
- namein
- namele
- namelike
- namelt
- namene
- namenlike
- nameout
- namerecv
- nameregexeq
- nameregexne
- namesend
- neqjoinsel
- neqsel
- netmask
- network
- network_cmp
- network_eq

- network_ge
- network_gt
- network_le
- network_lt
- network_ne
- network_sub
- network_subeq
- network_sup
- network_supeq
- next_day
- nextval
- ngram_end
- ngram_lextype
- ngram_nexttoken
- ngram_start
- nlikejoinsel
- nlikesel
- node_oid_name
- notlike
- now
- npoints
- nth_value
- ntile
- numeric
- numeric_abs
- numeric_accum
- numeric_add
- numeric_avg
- numeric_avg_accum
- numeric_avg_collect
- numeric_bpchar
- numeric_cmp
- numeric_collect
- numeric_div
- numeric_div_trunc
- numeric_eq
- numeric_exp

- numeric_fac
- numeric_ge
- numeric_gt
- numeric_in
- numeric_inc
- numeric_int1
- numeric_larger
- numeric_le
- numeric_list_agg_noarg2_transfn
- numeric_list_agg_transfn
- numeric_ln
- numeric_log
- numeric_lt
- numeric_mod
- numeric_mul
- numeric_ne
- numeric_out
- numeric_power
- numeric_recv
- numeric_send
- numeric_smaller
- numeric_sortsupport
- numeric_sqrt
- numeric_stddev_pop
- numeric_stddev_samp
- numeric_sub
- numeric_text
- numeric_transform
- numeric_uminus
- numeric_uplus
- numeric_var_pop
- numeric_var_samp
- numeric_varchar
- numerictypmodin
- numerictypmodout
- numnode
- numrange

- numrange_subdiff
- numtoday
- numtodsinterval
- nvarchar2
- nvarchar2in
- nvarchar2out
- nvarchar2recv
- nvarchar2send
- nvarchar2typmodin
- nvarchar2typmodout
- obj_description
- octet_length
- oid
- oideq
- oidge
- oidgt
- oidin
- oidlarger
- oidle
- oidlt
- oidne
- oidout
- oidrecv
- oidsend
- oidsmaller
- oidvectoreq
- oidvectorge
- oidvectorgt
- oidvectorin
- oidvectorin_extend
- oidvectorle
- oidvectorlt
- oidvectorne
- oidvectorout
- oidvectorout_extend
- oidvectorrecv
- oidvectorrecv_extend

- oidvectorsend
- oidvectorsend_extend
- oidvectortypes
- on_pb
- on_pl
- on_ppath
- on_ps
- on_sb
- on_sl
- opaque_in
- opaque_out
- ordered_set_transition
- overlaps
- overlay
- path
- path_add
- path_add_pt
- path_center
- path_contain_pt
- path_distance
- path_div_pt
- path_in
- path_inter
- path_length
- path_mul_pt
- path_n_eq
- path_n_ge
- path_n_gt
- path_n_le
- path_n_lt
- path_npoints
- path_out
- path_recv
- path_send
- path_sub_pt
- percent_rank
- percentile_cont

- percentile_cont_float8_final
- percentile_cont_interval_final
- pg_autovac_coordinator
- pg_available_extension_versions
- pg_available_extensions
- pg_buffercache_pages
- pg_cancel_invalid_query
- pg_char_to_encoding
- pg_check_xidlimit
- pg_clean_region_info
- pg_collation_for
- pg_comm_delay
- pg_comm_recv_stream
- pg_comm_send_stream
- pg_comm_status
- pg_control_group_config
- pg_create_physical_replication_slot
- pg_current_sessid
- pg_current_sessionid
- pg_current_userid
- pg_cursor
- pg_encoding_max_length
- pg_encoding_to_char
- pg_extension_config_dump
- pg_extension_update_paths
- pg_filenode_relation
- pg_get_replication_slot_name
- pg_get_replication_slots
- pg_get_xidlimit
- pg_lock_status
- pg_log_comm_status
- pg_logical_slot_get_binary_changes
- pg_logical_slot_peek_binary_changes
- pg_node_tree_in
- pg_node_tree_out
- pg_node_tree_recv
- pg_node_tree_send

- pg_notify
- pg_parse_clog
- pg_partition_filenode
- pg_pool_ping
- pg_pool_validate
- pg_prepared_statement
- pg_prepared_xact
- pg_relation_compression_ratio
- pg_relation_with_compression
- pg_resume_bkp_flag
- pg_sequence_parameters
- pg_show_all_settings
- pg_stat_file_recursive
- pg_stat_get_activity_for_temptable
- pg_stat_get_activity_ng
- pg_stat_get_bgwriter_stat_reset_time
- pg_stat_get_buf_fsync_backend
- pg_stat_get_cgroup_info
- pg_stat_get_checkpoint_sync_time
- pg_stat_get_checkpoint_write_time
- pg_stat_get_db_blk_read_time
- pg_stat_get_db_blk_write_time
- pg_stat_get_db_conflict_all
- pg_stat_get_db_conflict_bufferpin
- pg_stat_get_db_conflict_snapshot
- pg_stat_get_db_conflict_startup_deadlock
- pg_stat_get_db_conflict_tablespace
- pg_stat_get_db_stat_reset_time
- pg_stat_get_db_temp_bytes
- pg_stat_get_db_temp_files
- pg_stat_get_file_stat
- pg_stat_get_function_total_time
- pg_stat_get_mem_mbytes_reserved
- pg_stat_get_partition_tuples_hot_updated
- pg_stat_get_pooler_status
- pg_stat_get_realtime_info_internal
- pg_stat_get_redo_stat

- pg_stat_get_role_name
- pg_stat_get_session_wlmstat
- pg_stat_get_status
- pg_stat_get_stream_replications
- pg_stat_get_wal_receiver
- pg_stat_get_wal_senders
- pg_stat_get_wlm_ec_operator_info
- pg_stat_get_wlm_instance_info
- pg_stat_get_wlm_instance_info_with_cleanup
- pg_stat_get_wlm_node_resource_info
- pg_stat_get_wlm_operator_info
- pg_stat_get_wlm_realtime_ec_operator_info
- pg_stat_get_wlm_realtime_operator_info
- pg_stat_get_wlm_realtime_session_info
- pg_stat_get_wlm_session_info
- pg_stat_get_wlm_session_info_internal
- pg_stat_get_wlm_session_iostat_info
- pg_stat_get_wlm_statistics
- pg_stat_get_workload_struct_info
- pg_stat_get_xact_blocks_fetched
- pg_stat_get_xact_blocks_hit
- pg_stat_get_xact_function_calls
- pg_stat_get_xact_function_self_time
- pg_stat_get_xact_function_total_time
- pg_stat_get_xact_numscans
- pg_stat_get_xact_tuples_fetched
- pg_stat_get_xact_tuples_returned
- pg_stat_reset
- pg_sync_cstore_delta
- pg_tde_info
- pg_test_err_contain_err
- pg_timezone_abbrevs
- pg_timezone_names
- pgxc_get_csn
- pgxc_get_thread_wait_status
- pgxc_lock_for_sp_database
- pgxc_max_datanode_size

- pgxc_node_str
- pgxc_pool_connection_status
- pgxc_snapshot_status
- pgxc_stat_dirty_tables
- pgxc_unlock_for_sp_database
- pi
- plainto_tsquery
- plancache_clean
- plancache_status
- point
- point_above
- point_add
- point_below
- point_distance
- point_div
- point_eq
- point_horiz
- point_in
- point_left
- point_mul
- point_ne
- point_out
- point_recv
- point_right
- point_send
- point_sub
- point_vert
- poly_above
- poly_below
- poly_center
- poly_contain
- poly_contain_pt
- poly_contained
- poly_distance
- poly_in
- poly_left
- poly_npoints

- poly_out
- poly_overabove
- poly_overbelow
- poly_overlap
- poly_overleft
- poly_overright
- poly_recv
- poly_right
- poly_same
- poly_send
- polygon
- position
- positionjoinsel
- positionsel
- postgresql_fdw_validator
- pound_end
- pound_lextype
- pound_nexttoken
- pound_start
- pow
- power
- prepare_statement_status
- prsd_end
- prsd_headline
- prsd_lextype
- prsd_nexttoken
- prsd_start
- psortbuild
- psortcanreturn
- psortcostestimate
- psortgetbitmap
- psortgettuple
- psortoptions
- pt_contained_circle
- pt_contained_poly
- pv_builtin_functions
- pv_compute_pool_workload

- pv_os_run_info
- pv_session_memory
- pv_session_memory_detail
- pv_session_stat
- pv_session_time
- pv_thread_memory_detail
- pv_total_memory_detail
- query_to_xml
- query_to_xml_and_xmlschema
- query_to_xmlschema
- querytree
- quote_ident
- quote_literal
- quote_nullable
- radians
- radius
- random
- range_adjacent
- range_after
- range_before
- range_cmp
- range_contained_by
- range_contains
- range_contains_elem
- range_eq
- range_ge
- range_gist_compress
- range_gist_consistent
- range_gist_decompress
- range_gist_penalty
- range_gist_picksplit
- range_gist_same
- range_gist_union
- range_gt
- range_in
- range_intersect
- range_le

- range_lt
- range_minus
- range_ne
- range_out
- range_overlaps
- range_overleft
- range_overright
- range_recv
- range_send
- range_typanalyze
- range_union
- rank
- rawcat
- rawcmp
- raweq
- rawge
- rawgt
- rawin
- rawle
- rawlike
- rawlt
- rawne
- rawnlike
- rawout
- rawrecv
- rawsend
- rawtohex
- read_disable_conn_file
- record_eq
- record_ge
- record_gt
- record_in
- record_le
- record_lt
- record_ne
- record_out
- record_recv

- record_send
- regclass
- regclassin
- regclassout
- regclassrecv
- regclasssend
- regconfigin
- regconfigout
- regconfigrecv
- regconfigsend
- regdictionaryin
- regdictionaryout
- regdictionaryrecv
- regdictionarysend
- regexeqjoinsel
- regexeqsel
- regexnejoinsel
- regexnesel
- regexp_matches
- regexp_replace
- regexp_split_to_array
- regexp_split_to_table
- regoperatorin
- regoperatorout
- regoperatorrecv
- regoperatorsend
- regoperin
- regoperout
- regoperrecv
- regopersend
- regprocedurein
- regprocedureout
- regprocedurerecv
- regproceduresend
- regprocin
- regprocout
- regprocrecv

- regprocsend
- regr_avgx
- regr_avgy
- regr_count
- regr_intercept
- regr_r2
- regr_slope
- regr_sxx
- regr_sxy
- regr_syy
- regtypein
- regtypeout
- regtyperecv
- regtypesend
- reltime
- reltimeeq
- reltimege
- reltimegt
- reltimein
- reltimele
- reltimelt
- reltimene
- reltimeout
- reltimerecv
- reltimesend
- remote_ckpt_stat
- remote_double_write_stat
- remote_pagewriter_stat
- remote_recovery_status
- remote_redo_stat
- remote_rto_stat
- repeat
- replace
- report_fatal
- reset_unique_sql
- reverse
- RI_FKey_cascade_del

- RI_FKey_cascade_upd
- RI_FKey_check_ins
- RI_FKey_check_upd
- RI_FKey_noaction_del
- RI_FKey_noaction_upd
- RI_FKey_restrict_del
- RI_FKey_restrict_upd
- RI_FKey_setdefault_del
- RI_FKey_setdefault_upd
- RI_FKey_setnull_del
- RI_FKey_setnull_upd
- right
- round
- row_number
- row_to_json
- rpad
- rtrim
- scalargtjoinsel
- scalargtsel
- scalarltjoinsel
- scalarltsel
- schema_to_xml
- schema_to_xml_and_xmlschema
- schema_to_xmlschema
- session_user
- sessionid2pid
- set_bit
- set_byte
- set_config
- set_hashbucket_info
- set_masklen
- set_working_grand_version_num_manually
- setseed
- setval
- setweight
- shell_in
- shell_out

- shobj_description
- sign
- signal_backend
- similar_escape
- sin
- slope
- smalldatetime_cmp
- smalldatetime_eq
- smalldatetime_ge
- smalldatetime_gt
- smalldatetime_hash
- smalldatetime_in
- smalldatetime_larger
- smalldatetime_le
- smalldatetime_lt
- smalldatetime_ne
- smalldatetime_out
- smalldatetime_recv
- smalldatetime_send
- smalldatetime_smaller
- smgreq
- smgrin
- smgrne
- smgrout
- spg_kd_choose
- spg_kd_config
- spg_kd_inner_consistent
- spg_kd_picksplit
- spg_quad_choose
- spg_quad_config
- spg_quad_inner_consistent
- spg_quad_leaf_consistent
- spg_quad_picksplit
- spg_text_choose
- spg_text_config
- spg_text_inner_consistent
- spg_text_leaf_consistent

- spg_text_picksplit
- spgbeginscan
- spgbuild
- spgbuildempty
- spgbulkdelete
- spgcanreturn
- spgcostestimate
- spgendscan
- spggetbitmap
- spggettuple
- spginsert
- spgmarkpos
- spgmerge
- spgoptions
- spgrescan
- spgrestrpos
- spgvacuumcleanup
- split_part
- sqrt
- statement_timestamp
- stddev
- stddev_pop
- stddev_samp
- string_agg
- string_agg_finalfn
- string_agg_transfn
- string_to_array
- strip
- strpos
- substr
- substrb
- substring
- substring_inner
- sum
- suppress_redundant_updates_trigger
- sysdate
- table_data_skewness

- table_distribution
- table_to_xml
- table_to_xml_and_xmlschema
- table_to_xmlschema
- tablespace_oid_name
- tan
- text
- text_date
- text_float4
- text_float8
- text_ge
- text_gt
- text_int1
- text_int2
- text_int4
- text_int8
- text_larger
- text_le
- text_lt
- text_numeric
- text_pattern_ge
- text_pattern_gt
- text_pattern_le
- text_pattern_lt
- text_smaller
- text_timestamp
- textanycat
- textcat
- texteq
- texticlike
- texticnlike
- texticregexeq
- texticregexne
- textin
- textlen
- textlike
- textne

- textnlike
- textout
- textrecv
- textregexeq
- textregexne
- textsend
- thesaurus_init
- thesaurus_lexize
- threadpool_status
- tideq
- tidge
- tidgt
- tidin
- tidlarger
- tidle
- tidlt
- tidne
- tidout
- tidrecv
- tidsend
- tidsmaller
- time
- time_cmp
- time_eq
- time_ge
- time_gt
- time_hash
- time_in
- time_larger
- time_le
- time_lt
- time_mi_interval
- time_mi_time
- time_ne
- time_out
- time_pl_interval
- time_recv

- time_send
- time_smaller
- time_transform
- timedate_pl
- timemi
- timenow
- timeofday
- timepl
- timestamp
- timestamp_cmp
- timestamp_cmp_date
- timestamp_cmp_timestamptz
- timestamp_diff
- timestamp_eq
- timestamp_eq_date
- timestamp_eq_timestamptz
- timestamp_ge
- timestamp_ge_date
- timestamp_ge_timestamptz
- timestamp_gt
- timestamp_gt_date
- timestamp_gt_timestamptz
- timestamp_hash
- timestamp_in
- timestamp_larger
- timestamp_le
- timestamp_le_date
- timestamp_le_timestamptz
- timestamp_list_agg_noarg2_transfn
- timestamp_list_agg_transfn
- timestamp_lt
- timestamp_lt_date
- timestamp_lt_timestamptz
- timestamp_mi
- timestamp_mi_interval
- timestamp_ne
- timestamp_ne_date

- timestamp_ne_timestamptz
- timestamp_out
- timestamp_pl_interval
- timestamp_recv
- timestamp_send
- timestamp_smaller
- timestamp_sortsupport
- timestamp_text
- timestamp_transform
- timestamp_varchar
- timestamptypmodin
- timestamptypmodout
- timestamptz
- timestamptz_cmp
- timestamptz_cmp_date
- timestamptz_cmp_timestamp
- timestamptz_eq
- timestamptz_eq_date
- timestamptz_eq_timestamp
- timestamptz_ge
- timestamptz_ge_date
- timestamptz_ge_timestamp
- timestamptz_gt
- timestamptz_gt_date
- timestamptz_gt_timestamp
- timestamptz_in
- timestamptz_larger
- timestamptz_le
- timestamptz_le_date
- timestamptz_le_timestamp
- timestamptz_list_agg_noarg2_transfn
- timestamptz_list_agg_transfn
- timestamptz_lt
- timestamptz_lt_date
- timestamptz_lt_timestamp
- timestamptz_mi
- timestamptz_mi_interval

- timestamptz_ne
- timestamptz_ne_date
- timestamptz_ne_timestamp
- timestamptz_out
- timestamptz_pl_interval
- timestamptz_recv
- timestamptz_send
- timestamptz_smaller
- timestamptztypmodin
- timestamptztypmodout
- timestampzone_text
- timetypmodin
- timetypmodout
- timetz
- timetz_cmp
- timetz_eq
- timetz_ge
- timetz_gt
- timetz_hash
- timetz_in
- timetz_larger
- timetz_le
- timetz_lt
- timetz_mi_interval
- timetz_ne
- timetz_out
- timetz_pl_interval
- timetz_recv
- timetz_send
- timetz_smaller
- timetzdate_pl
- timetztypmodin
- timetztypmodout
- timezone
- tinterval
- tintervalct
- tintervalend

- tintervaleq
- tintervalge
- tintervalgt
- tintervalin
- tintervalle
- tintervalleneq
- tintervallenge
- tintervallengt
- tintervallenle
- tintervallenlt
- tintervallenne
- tintervallt
- tintervalne
- tintervalout
- tintervalov
- tintervalrecv
- tintervalrel
- tintervalsame
- tintervalsend
- tintervalstart
- to_ascii
- to_char
- to_date
- to_hex
- to_number
- to_timestamp
- to_tsquery
- to_tsvector
- to_tsvector_for_batch
- total_cpu
- total_memory
- track_model_train_opt
- transaction_timestamp
- translate
- trigger_in
- trigger_out
- trunc

- ts_headline
- ts_lexize
- ts_match_qv
- ts_match_tq
- ts_match_tt
- ts_match_vq
- ts_parse
- ts_rank
- ts_rank_cd
- ts_rewrite
- ts_stat
- ts_token_type
- ts_typanalyze
- tsmatchjoinsel
- tsmatchsel
- tsq_mcontained
- tsq_mcontains
- tsquery_and
- tsquery_cmp
- tsquery_eq
- tsquery_ge
- tsquery_gt
- tsquery_le
- tsquery_lt
- tsquery_ne
- tsquery_not
- tsquery_or
- tsqueryin
- tsqueryout
- tsqueryrecv
- tsquerysend
- tsrange
- tsrange_subdiff
- tstzrange
- tstzrange_subdiff
- tsvector_cmp
- tsvector_concat

- tsvector_eq
- tsvector_ge
- tsvector_gt
- tsvector_le
- tsvector_lt
- tsvector_ne
- tsvector_update_trigger
- tsvector_update_trigger_column
- tsvectorin
- tsvectorout
- tsvectorrecv
- tsvectorsend
- txid_current
- txid_current_snapshot
- txid_snapshot_in
- txid_snapshot_out
- txid_snapshot_recv
- txid_snapshot_send
- txid_snapshot_xip
- txid_snapshot_xmax
- txid_snapshot_xmin
- txid_visible_in_snapshot
- unique_key_recheck
- unknownin
- unknownout
- unknownrecv
- unknownsend
- unnest
- update_pgjob
- upper
- upper_inc
- upper_inf
- uuid_cmp
- uuid_eq
- uuid_ge
- uuid_gt
- uuid_hash

- uuid_in
- uuid_le
- uuid_lt
- uuid_ne
- uuid_out
- uuid_recv
- uuid_send
- var_pop
- var_samp
- varbit
- varbit_in
- varbit_out
- varbit_recv
- varbit_send
- varbit_transform
- varbitcmp
- varbiteq
- varbitge
- varbitgt
- varbitle
- varbitlt
- varbitne
- varbittypmodin
- varbittypmodout
- varchar
- varchar_date
- varchar_float4
- varchar_float8
- varchar_int4
- varchar_int8
- varchar_numeric
- varchar_timestamp
- varchar_transform
- varcharin
- varcharout
- varcharrecv
- varcharsend

- varchartypmodin
- varchartypmodout
- variance
- version
- void_in
- void_out
- void_recv
- void_send
- wdr_xdb_query
- width
- width_bucket
- xideq
- xideq4
- xideqint4
- xideqint8
- xidin
- xidin4
- xidlt
- xidlt4
- xidout
- xidout4
- xidrecv
- xidrecv4
- xidsend
- xidsend4
- xml
- xml_in
- xml_is_well_formed
- xml_is_well_formed_content
- xml_is_well_formed_document
- xml_out
- xml_recv
- xml_send
- xmlagg
- xmlcomment
- xmlconcat2
- xmlexists

- xmlvalidate
- xpath
- xpath_exists
- zhprs_end
- zhprs_getlexeme
- zhprs_lextype
- zhprs_start

11.6. 表达式

11.6.1. 简单表达式

逻辑表达式

逻辑表达式的操作符和运算规则，请参见 11.5.1 逻辑操作符。

比较表达式

常用的比较操作符，请参见 11.7.2 操作符。

除比较操作符外，还可以使用以下句式结构：

- BETWEEN 操作符

  a BETWEEN x AND y 等效于 a >= x AND a <= y

  a NOT BETWEEN x AND y 等效于 a < x OR a > y

- 检查一个值是不是 null，可使用：

  expression IS NULL

  expression IS NOT NULL

  或者与之等价的句式结构，但不是标准的：

  expression ISNULL

  expression NOTNULL

须知

不要写 expression=NULL 或 expression<>(!=)NULL，因为 NULL 代表一个未知的值，不能通过该表达式判断两个未知值是否相等。

示例

```
vastbase=# SELECT 2 BETWEEN 1 AND 3 AS RESULT;
 result
----------
 t
(1 row)

vastbase=# SELECT 2 >= 1 AND 2 <= 3 AS RESULT;
 result
----------
 t
(1 row)

vastbase=# SELECT 2 NOT BETWEEN 1 AND 3 AS RESULT;
 result
----------
 f
(1 row)

vastbase=# SELECT 2 < 1 OR 2 > 3 AS RESULT;
 result
----------
 f
(1 row)

vastbase=# SELECT 2+2 IS NULL AS RESULT;
 result
----------
 f
(1 row)

vastbase=# SELECT 2+2 IS NOT NULL AS RESULT;
 result
----------
 t
(1 row)

vastbase=# SELECT 2+2 ISNULL AS RESULT;
 result
----------
 f
(1 row)

vastbase=# SELECT 2+2 NOTNULL AS RESULT;
 result
----------
 t
(1 row)

vastbase=# SELECT 2+2 IS DISTINCT FROM NULL AS RESULT;
 result
----------
 t
(1 row)
```

```
vastbase=# SELECT 2+2 IS NOT DISTINCT FROM NULL AS RESULT;
 result
----------
 f
(1 row)
```

11.6.2. 条件表达式

在执行 SQL 语句时，可通过条件表达式筛选出符合条件的数据。

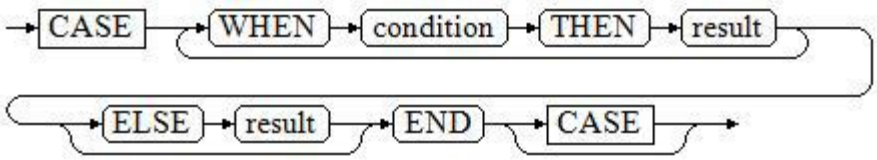
条件表达式主要有以下几种：

- CASE

  CASE 表达式是条件表达式，类似于其他编程语言中的 CASE 语句。

  CASE 表达式的语法图请参考图 11-1。

  图 11-2. case::=

  CASE 子句可以用于合法的表达式中。condition 是一个返回 BOOLEAN 数据类型的表达式：

  - 如果结果为真，CASE 表达式的结果就是符合该条件所对应的 result。
  - 如果结果为假，则以相同方式处理随后的 WHEN 或 ELSE 子句。
  - 如果各 WHEN condition 都不为真，表达式的结果就是在 ELSE 子句执行的 result。如果省略了 ELSE 子句且没有匹配的条件，结果为 NULL。

  示例：

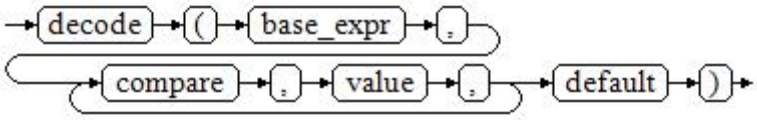
```
vastbase=# CREATE TABLE tpcds.case_when_t1(CW_COL1 INT);

vastbase=# INSERT INTO tpcds.case_when_t1 VALUES (1), (2), (3);

vastbase=# SELECT * FROM tpcds.case_when_t1;
 a
---
 1
 2
 3
(3 rows)

vastbase=# SELECT CW_COL1, CASE WHEN CW_COL1=1 THEN 'one' WHEN CW_COL1=2 THEN 'two' ELSE 'other'
END FROM tpcds.case_when_t1 ORDER BY 1;
 cw_col1 | case
---------+-------
       1 | one
```

```
     2 | two
     3 | other
(3 rows)

vastbase=# DROP TABLE tpcds.case_when_t1;
```

❖ DECODE

DECODE 的语法图请参见图 11-2。

图 11-3. decode::=



将表达式 base_expr 与后面的每个 compare(n) 进行比较，如果匹配返回相应的 value(n)。如果没有发生匹配，则返回 default。

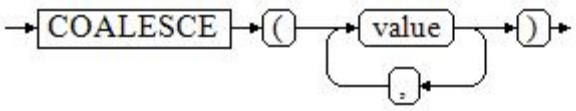
示例请参见 11.5.21 条件表达式函数。

```
vastbase=# SELECT DECODE('A','A',1,'B',2,0);
 case
------
    1
(1 row)
```

❖ COALESCE

COALESCE 的语法图请参见图 11-3。

图 11-4. coalesce::=



COALESCE 返回它的第一个非 NULL 的参数值。如果参数都为 NULL，则返回 NULL。它常用于在显示数据时用缺省值替换 NULL。和 CASE 表达式一样，COALESCE 只计算用来判断结果的参数，即在第一个非空参数右边的参数不会被计算。

示例

```
vastbase=# CREATE TABLE tpcds.c_tabl(description varchar(10), short_description varchar(10),
last_value varchar(10))
;

vastbase=# INSERT INTO tpcds.c_tabl VALUES('abc', 'efg', '123');
vastbase=# INSERT INTO tpcds.c_tabl VALUES(NULL, 'efg', '123');

vastbase=# INSERT INTO tpcds.c_tabl VALUES(NULL, NULL, '123');
```

```
vastbase=# SELECT description, short_description, last_value, COALESCE(description,
short_description, last_value) FROM tpcds.c_tabl ORDER BY 1, 2, 3, 4;
 description | short_description | last_value | coalesce
-------------+-------------------+------------+----------
 abc         | efg               | 123        | abc
             | efg               | 123        | efg
             |                   | 123        | 123
(3 rows)

vastbase=# DROP TABLE tpcds.c_tabl;
```

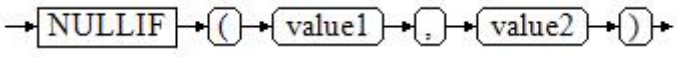
如果 description 不为 NULL，则返回 description 的值，否则计算下一个参数 short_description；如果 short_description 不为 NULL，则返回 short_description 的值，否则计算下一个参数 last_value；如果 last_value 不为 NULL，则返回 last_value 的值，否则返回（none）。

```
vastbase=# SELECT COALESCE(NULL,'Hello World');
  coalesce
---------------
 Hello World
(1 row)
```

- NULLIF

NULLIF 的语法图请参见图 11-4。

图 11-5.　nullif::=



只有当 value1 和 value2 相等时，NULLIF 才返回 NULL。否则它返回 value1。

示例

```
vastbase=# CREATE TABLE tpcds.null_if_t1 (
   NI_VALUE1 VARCHAR(10),
   NI_VALUE2 VARCHAR(10)
);

vastbase=# INSERT INTO tpcds.null_if_t1 VALUES('abc', 'abc');
vastbase=# INSERT INTO tpcds.null_if_t1 VALUES('abc', 'efg');

vastbase=# SELECT NI_VALUE1, NI_VALUE2, NULLIF(NI_VALUE1, NI_VALUE2) FROM tpcds.null_if_t1
ORDER BY 1, 2, 3;

 ni_value1 | ni_value2 | nullif
-----------+-----------+--------
 abc       | abc       |
 abc       | efg       | abc
(2 rows)
vastbase=# DROP TABLE tpcds.null_if_t1;
```

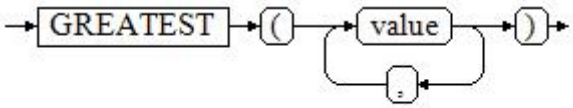
如果 value1 等于 value2 则返回 NULL，否则返回 value1。

```
vastbase=# SELECT NULLIF('Hello','Hello World');
 nullif
--------
 Hello
(1 row)
```

❖ GREATEST（最大值），LEAST（最小值）

GREATEST 的语法图请参见图 11-5。

图 11-6. greatest::=

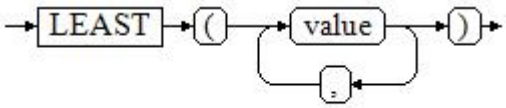


从一个任意数字表达式的列表里选取最大的数值。

```
vastbase=# SELECT greatest(9000,155555,2.01);
 greatest
----------
   155555
(1 row)
```

LEAST 的语法图请参见图 11-6。

图 11-7. least::=



从一个任意数字表达式的列表里选取最小的数值。

以上的数字表达式必须都可以转换成一个普通的数据类型，该数据类型将是结果类型。

列表中的 NULL 值将被忽略。只有所有表达式的结果都是 NULL 的时候，结果才是 NULL。

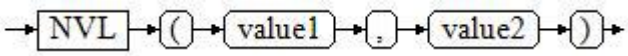
```
vastbase=# SELECT least(9000,2);
 least
-------
     2
(1 row)
```

示例请参见 11.5.21 条件表达式函数。

❖ NVL

NVL 的语法图请参见图 11-7。

图 11-8.　nvl::=



如果 value1 为 NULL 则返回 value2，如果 value1 非 NULL，则返回 value1。

示例：

```
vastbase=# SELECT nvl(null,1);
NVL
-----
 1
(1 row)

vastbase=# SELECT nvl ('Hello World' ,1);
     nvl
---------------
 Hello World
(1 row)
```

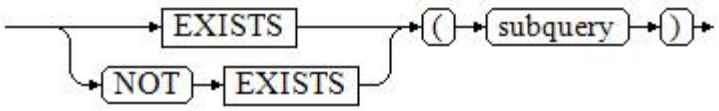
11.6.3. 子查询表达式

子查询表达式主要有以下几种：

- EXISTS/NOT EXISTS

EXISTS/NOT EXISTS 的语法图请参见图 11-9。

图 11-9.　EXISTS/NOT EXISTS::=



EXISTS 的参数是一个任意的 SELECT 语句，或者说子查询。系统对子查询进行运算以判断它是否返回行。如果它至少返回一行，则 EXISTS 结果就为"真"；如果子查询没有返回任何行，EXISTS 的结果是"假"。

这个子查询通常只是运行到能判断它是否可以生成至少一行为止，而不是等到全部结束。

示例：

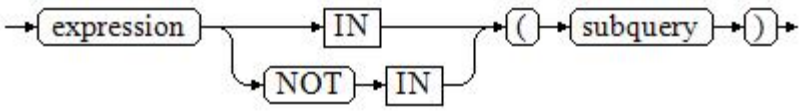
```
vastbase=# SELECT sr_reason_sk,sr_customer_sk FROM tpcds.store_returns WHERE EXISTS (SELECT
d_dom FROM tpcds.date_dim WHERE d_dom = store_returns.sr_reason_sk and sr_customer_sk <10);
sr_reason_sk | sr_customer_sk
-------------+---------------
          13 |              2
          22 |              5
          17 |              7
```

```
        25 |           7
         3 |           7
        31 |           5
         7 |           7
        14 |           6
        20 |           4
         5 |           6
        10 |           3
         1 |           5
        15 |           2
         4 |           1
        26 |           3
(15 rows)
```

❖ IN/NOT IN

IN/NOT IN 的语法请参见图 11-9。

图 11-10. IN/NOT IN::=



右边是一个圆括弧括起来的子查询，它必须只返回一个字段。左边表达式对子查询结果的每一行进行一次计算和比较。如果找到任何相等的子查询行，则 IN 结果为"真"。如果没有找到任何相等行，则结果为"假"（包括子查询没有返回任何行的情况）。

表达式或子查询行里的 NULL 遵照 SQL 处理布尔值和 NULL 组合时的规则。如果两个行对应的字段都相等且非空，则这两行相等；如果任意对应字段不等且非空，则这两行不等；否则结果是未知（NULL）。如果每一行的结果都是不等或 NULL ，并且至少有一个 NULL ，则 IN 的结果是 NULL 。

示例：

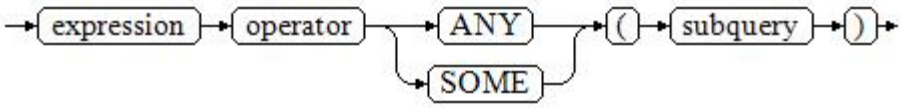
```
vastbase=# SELECT sr_reason_sk,sr_customer_sk  FROM tpcds.store_returns WHERE sr_customer_sk
IN (SELECT d_dom FROM tpcds.date_dim WHERE d_dom < 10);
sr_reason_sk | sr_customer_sk
--------------+----------------
        10 |           3
        26 |           3
        22 |           5
        31 |           5
         1 |           5
        32 |           5
        32 |           5
         4 |           1
        15 |           2
        13 |           2
        33 |           4
        20 |           4
```

```
         33 |            8
          5 |            6
         14 |            6
         17 |            7
          3 |            7
         25 |            7
          7 |            7
(19 rows)
```

❖ ANY/SOME

ANY/SOME 的语法图请参见图 11-10。

图 11-11. any/some::=



右边是一个圆括弧括起来的子查询，它必须只返回一个字段。左边表达式使用 operator 对子查询结果的每一行进行一次计算和比较，其结果必须是布尔值。如果至少获得一个真值，则 ANY 结果为“真”。如果全部获得假值，则结果是“假”（包括子查询没有返回任何行的情况）。SOME 是 ANY 的同义词。IN 与 ANY 可以等效替换 。

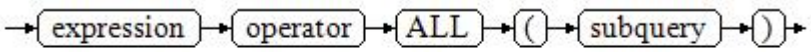
示例：

```
vastbase=# SELECT sr_reason_sk,sr_customer_sk  FROM tpcds.store_returns WHERE sr_customer_sk
< ANY (SELECT d_dom FROM tpcds.date_dim WHERE d_dom < 10);
sr_reason_sk | sr_customer_sk
--------------+----------------
         26 |            3
         17 |            7
         32 |            5
         32 |            5
         13 |            2
         31 |            5
         25 |            7
          5 |            6
          7 |            7
         10 |            3
          1 |            5
         14 |            6
          4 |            1
          3 |            7
         22 |            5
         33 |            4
         20 |            4
         33 |            8
         15 |            2
(19 rows)
```

❖ ALL

ALL 的语法请参见图 11-11。

图 11-12. all::=



右边是一个圆括弧括起来的子查询，它必须只返回一个字段。左边表达式使用 operator 对子查询结果的每一行进行一次计算和比较，其结果必须是布尔值。如果全部获得真值，ALL 结果为"真"（包括子查询没有返回任何行的情况）。如果至少获得一个假值，则结果是"假"。

示例：

```
vastbase=# SELECT sr_reason_sk,sr_customer_sk  FROM tpcds.store_returns WHERE sr_customer_sk
< all(SELECT d_dom FROMOM tpcds.date_dim WHERE d_dom < 10);
 sr_reason_sk | sr_customer_sk
--------------+----------------
(0 rows)
```

11.6.4. 数组表达式

IN

expression **IN** *(value [, ...])*

右侧括号中的是一个表达式列表。左侧表达式的结果与表达式列表的内容进行比较。如果列表中的内容符合左侧表达式的结果，则 IN 的结果为 true。如果没有相符的结果，则 IN 的结果为 false。

示例如下：

```
vastbase=# SELECT 8000+500 IN (10000, 9000) AS RESULT;
 result
----------
 f
(1 row)
```

说明

如果表达式结果为 null，或者表达式列表不符合表达式的条件且右侧表达式列表返回结果至少一处为空，则 IN 的返回结果为 null，而不是 false。这样的处理方式和 SQL 返回空值的布尔组合规则是一致的。

NOT IN

expression **NOT IN** *(value [, ...])*

右侧括号中的是一个表达式列表。左侧表达式的结果与表达式列表的内容进行比较。如果在列表中的内容没有符合左侧表达式结果的内容，则 NOT IN 的结果为 true。如果有符合的内容，则 NOT IN 的结果为 false。

示例如下：

```
vastbase=# SELECT 8000+500 NOT IN (10000, 9000) AS RESULT;
 result
----------
 t
(1 row)
```

说明

如果查询语句返回结果为空，或者表达式列表不符合表达式的条件且右侧表达式列表返回结果至少一处为空，则 NOT IN 的返回结果为 null，而不是 false。这样的处理方式和 SQL 返回空值的布尔组合规则是一致的。

提示：在所有情况下 X NOT IN Y 等价于 NOT(X IN Y)。

ANY/SOME (array)

expression operator **ANY** *(array expression)*

expression operator **SOME** *(array expression)*

```
vastbase=# SELECT 8000+500 < SOME (array[10000,9000]) AS RESULT;
 result
----------
 t
(1 row)

vastbase=# SELECT 8000+500 < ANY (array[10000,9000]) AS RESULT;
 result
----------
 t
(1 row)
```

右侧括号中的是一个数组表达式，它必须产生一个数组值。左侧表达式的结果使用操作符对数组表达式的每一行结果都进行计算和比较，比较结果必须是布尔值。

- 如果对比结果至少获取一个真值，则 ANY 的结果为 true。
- 如果对比结果没有真值，则 ANY 的结果为 false。

说明

如果结果没有真值，并且数组表达式生成至少一个值为 null，则 ANY 的值为 NULL，而不是 false。这样的处理方式和 SQL 返回空值的布尔组合规则是一致的。

SOME 是 ANY 的同义词。

ALL (array)

expression operator **ALL** *(array expression)*

右侧括号中的是一个数组表达式，它必须产生一个数组值。左侧表达式的结果使用操作符对数组表达式的每一行结果都进行计算和比较，比较结果必须是布尔值。

- 如果所有的比较结果都为真值（包括数组不含任何元素的情况），则 ALL 的结果为 true。
- 如果存在一个或多个比较结果为假值，则 ALL 的结果为 false。

如果数组表达式产生一个 NULL 数组，则 ALL 的结果为 NULL。如果左边表达式的值为 NULL ，则 ALL 的结果通常也为 NULL(某些不严格的比较操作符可能得到不同的结果)。另外，如果右边的数组表达

式中包含 null 元素并且比较结果没有假值，则 ALL 的结果将是 NULL(某些不严格的比较操作符可能得到不同的结果)， 而不是真。这样的处理方式和 SQL 返回空值的布尔组合规则是一致的。

```
vastbase=# SELECT 8000+500 < ALL (array[10000,9000]) AS RESULT;
 result
----------
 t
(1 row)
```

11.6.5. 行表达式

语法：

row_constructor operator row_constructor

两边都是一个行构造器，两行值必须具有相同数目的字段，每一行都进行比较，行比较允许使用=，<>，<，<=，>=等操作符，或其中一个相似的语义符。

=<>和别的操作符使用略有不同。如果两行值的所有字段都是非空并且相等，则认为两行是相等的；如果两行值的任意字段为非空并且不相等，则认为两行是不相等的；否则比较结果是未知的（null）。

对于<，<=，>，> =的情况下，行中元素从左到右依次比较，直到遇到一对不相等的元素或者一对为空的元素。如果这对元素中存在至少一个 null 值，则比较结果是未知的（null），否则这对元素的比较结果为最终的结果。

示例：

```
vastbase=# SELECT ROW(1,2,NULL) < ROW(1,3,0) AS RESULT;
 result
----------
 t
(1 row)
```

11.7. 类型转换

11.7.1. 概述

背景信息

在 SQL 语言中，每个数据都与一个决定其行为和用法的数据类型相关。Vastbase 提供一个可扩展的数据类型系统，该系统比其它 SQL 实现更具通用性和灵活性。因而，Vastbase 中大多数类型转换是由通用规则来管理的，这种做法允许使用混合类型的表达式。

Vastbase 扫描/分析器只将词法元素分解成五个基本种类：整数、浮点数、字符串、标识符和关键字。大多数非数字类型首先表现为字符串。SQL 语言的定义允许将常量字符串声明为具体的类型。例，下面查询：

```
vastbase=# SELECT text 'Origin' AS "label", point '(0,0)' AS "value";
 label  | value
--------+-------
```

```
Origin | (0,0)
(1 row)
```

示例中有两个文本常量，类型分别为 text 和 point。如果没有为字符串文本声明类型，则该文本首先被定义成一个 unknown 类型。

在 Vastbase 分析器里，有四种基本的 SQL 结构需要独立的类型转换规则：

- 函数调用

  多数 SQL 类型系统是建筑在一套丰富的函数上的。函数调用可以有一个或多个参数。因为 SQL 允许函数重载，所以不能通过函数名直接找到要调用的函数，分析器必须根据函数提供的参数类型选择正确的函数。

- 操作符

  SQL 允许在表达式上使用前缀或后缀（单目）操作符，也允许表达式内部使用双目操作符（两个参数）。像函数一样，操作符也可以被重载，因此操作符的选择也和函数一样取决于参数类型。

- 值存储

  INSERT 和 UPDATE 语句将表达式结果存入表中。语句中的表达式类型必须和目标字段的类型一致或者可以转换为一致。

- UNION，CASE 和相关构造

  因为联合 SELECT 语句中的所有查询结果必须在一列里显示出来，所以每个 SELECT 子句中的元素类型必须相互匹配并转换成一个统一类型。类似地，一个 CASE 构造的结果表达式必须转换成统一的类型，这样整个 CASE 表达式会有一个统一的输出类型。同样的要求也存在于 ARRAY 构造以及 GREATEST 和 LEAST 函数中。

系统表 pg_cast 存储了有关数据类型之间的转换关系以及如何执行这些转换的信息。详细信息请参见 14.2.17PG_CAST。

语义分析阶段会决定表达式的返回值类型并选择适当的转换行为。数据类型的基本类型分类，包括：Boolean，numeric，string，bitstring，datetime，timespan，geometric 和 network。每种类型都有一种或多种首选类型用于解决类型选择的问题。根据首选类型和可用的隐含转换，就可能保证有歧义的表达式（那些有多个候选解析方案的）得到有效的方式解决。

所有类型转换规则都是建立在下面几个基本原则上的：

- 隐含转换决不能有奇怪的或不可预见的输出。
- 如果一个查询不需要隐含的类型转换，分析器和执行器不应该进行更多的额外操作。这就是说，任何一个类型匹配、格式清晰的查询不应该在分析器里耗费更多的时间，也不应该向查询中引入任何不必要的隐含类型转换调用。
- 另外，如果一个查询在调用某个函数时需要进行隐式转换，当用户定义了一个有正确参数的函数后，解释器应该选择使用新函数。

11.7.2. 操作符

操作符类型解析

1. 从系统表 pg_operator 中选出要考虑的操作符。如果可以找到一个参数类型以及参数个数都一致的操作符，那么这个操作符就是最终使用的操作符。如果找到了多个备选的操作符，我们将从中选择一个最合适的。
2. 寻找最优匹配。
    a. 抛弃那些输入类型不匹配并且也不能隐式转换成匹配的候选操作符。unknown 文本在这种情况下可以转换成任何东西。如果只剩下一个候选项，则用之，否则继续下一步。
    b. 遍历所有候选操作符，保留那些输入类型匹配最准确的。此时，域被看作和他们的基本类型相同。如果没有一个操作符能被保留，则保留所有候选。如果只剩下一个候选项，则用之，否则继续下一步。
    c. 遍历所有候选操作符，保留那些需要类型转换时接受(属于输入数据类型的类型范畴的)首选类型位置最多的操作符。如果没有接受首选类型的操作符，则保留所有候选。如果只剩下一个候选项，则用之，否则继续下一步。
    d. 如果有任何输入参数是 unknown 类型，检查剩余的候选操作符对应参数位置的类型范畴。在每一个能够接受字符串类型范畴的位置使用 string 类型（这种对字符串的偏爱是合适的，因为 unknown 文本确实像字符串）。另外，如果所有剩下的候选操作符都接受相同的类型范畴，则选择该类型范畴，否则抛出一个错误（因为在没有更多线索的条件下无法作出正确的选择）。现在抛弃不接受选定的类型范畴的候选操作符，然后，如果任意候选操作符在某个给定的参数位置接受一个首选类型，则抛弃那些在该参数位置接受非首选类型的候选操作符。如果没有一个操作符能被保留，则保留所有候选。如果只剩下一个候选项，则用之，否则继续下一步。
    e. 如果同时有 unknown 和已知类型的参数，并且所有已知类型的参数都是相同的类型，那么假设 unknown 参数也是那种类型，并检查哪个候选操作符在 unknown 参数位置接受那个类型。如果只有一个操作符符合，那么使用它。否则，产生一个错误。

示例

示例 1：阶乘操作符类型解析。在系统表中里只有一个阶乘操作符（后缀!），它以 bigint 作为参数。扫描器给下面查询表达式的参数赋予 bigint 的初始类型：

```
vastbase=# SELECT 40 ! AS "40 factorial";

               40 factorial
```

```
-------------------------------------------------
 815915283247897734345611269596115894272000000000
(1 row)
```

分析器对参数做类型转换，查询等效于：

```
vastbase=# SELECT CAST(40 AS bigint) ! AS "40 factorial";
```

示例 2：字符串连接操作符类型分析。一种字符串风格的语法既可以用于字符串也可以用于复杂的扩展类型。未声明类型的字符串将被所有可能的候选操作符匹配。有一个未声明的参数的例子：

```
vastbase=# SELECT text 'abc' || 'def' AS "text and unknown";
 text and unknown
------------------
 abcdef
(1 row)
```

本例中分析器寻找两个参数都是 text 的操作符。确实有这样的操作符，两个参数都是 text 类型。

下面是连接两个未声明类型的值：

```
vastbase=# SELECT 'abc' || 'def' AS "unspecified";
 unspecified
-------------
 abcdef
(1 row)
```

说明

因为查询中没有声明任何类型，所以本例中对类型没有任何初始提示。因此，分析器查找所有候选操作符，发现既存在接受字符串类型范畴的操作符也存在接受位串类型范畴的操作符。因为字符串类型范畴是首选，所以选择字符串类型范畴的首选类型 text 作为解析未知类型文本的声明类型。

示例 3：绝对值和取反操作符类型分析。Vastbase 操作符表里面有几条记录对应于前缀操作符@，它们都用于为各种数值类型实现绝对值操作。其中之一用于 float8 类型，它是数值类型范畴中的首选类型。因此，在面对 unknown 输入的时候， Vastbase 会使用该类型：

```
vastbase=# SELECT @ '-4.5' AS "abs";
 abs
-----
 4.5
(1 row)
```

此处，系统在应用选定的操作符之前隐式的转换 unknown 类型的文字为 float8 类型。

示例 4：数组包含操作符类型分析。这里是解决一个操作符带有一个已知和一个未知类型输入的例子：

```
vastbase=# SELECT array[1,2] <@ '{1,2,3}' as "is subset";
 is subset
-----------
 t
(1 row)
```

说明

Vastbase 操作符表有几条记录对应于中缀操作符<@，但是只有两个可以在左侧接受一个整数数组的操作符是数组包含(anyarray <@ anyarray) 和范围包含(anyelement <@ anyrange)的。因为没有多态的伪类型(参阅 11.3.15 伪类型)是首选的，所以解析器不能解决这个基础上的歧义。然而，最后一个解析规则告诉用户，假设未知类型的文字是和另外一个输入相同的类型，也就是，整数数组。现在只有两个操作符中的一个可以匹配，所以选择数组包含。（如果用户选择了范围包含，用户将得到一个错误，因为字符串没有正确的格式成为范围的文字。）

11.7.3. 函数

函数类型解析

1. 从系统表 pg_proc 中选择所有可能被选到的函数。如果使用了一个不带模式修饰的函数名称，那么认为该函数是那些在当前搜索路径中的函数。如果给出一个带修饰的函数名，那么只考虑指定模式中的函数。

如果搜索路径中找到了多个不同参数类型的函数。将从中选择一个合适的函数。

2. 查找和输入参数类型完全匹配的函数。如果找到一个，则用之。如果输入的实参类型都是 unknown 类型，则不会找到匹配的函数。
3. 如果未找到完全匹配，请查看该函数是否为一个特殊的类型转换函数。
4. 寻找最优匹配。

    a. 抛弃那些输入类型不匹配并且也不能隐式转换成匹配的候选函数。unknown 文本在这种情况下可以转换成任何东西。如果只剩下一个候选项，则用之，否则继续下一步。

    b. 遍历所有候选函数，保留那些输入类型匹配最准确的。此时，域被看作和它们的基本类型相同。如果没有一个函数能准确匹配，则保留所有候选。如果只剩下一个候选项，则用之，否则继续下一步。

    c. 遍历所有候选函数，保留那些需要类型转换时接受首选类型位置最多的函数。如果没有接受首选类型的函数，则保留所有候选。如果只剩下一个候选项，则用之，否则继续下一步。

    d. 如果有任何输入参数是 unknown 类型，检查剩余的候选函数对应参数位置的类型范畴。在每一个能够接受字符串类型范畴的位置使用 string 类型（这种对字符串的偏爱是合适的，因为 unknown 文本确实像字符串）。另外，如果所有剩下的候选函数都接受相同的类型范畴，则选择该类型范畴，否则抛出一个错误（因为在没有更多线索的条件下无法作出正确的选择）。现在抛弃不接受选定的类型范畴的候选函数，然后，如果任意候选函数在那个范畴接受一个首选类型，则抛弃那些在该参数位置接受非首选类型的候选函数。如果没有一个候选符合这些测试则保留所有候选。如果只有一个候选函数符合，则使用它；否则，继续下一步。

    e. 如果同时有 unknown 和已知类型的参数，并且所有已知类型的参数有相同的类型，假设 unknown 参数也是这种类型，检查哪个候选函数可以在 unknown 参数位置接受这种类型。如果正好一个候选符合，那么使用它。否则，产生一个错误。

示例

示例 1：圆整函数参数类型解析。只有一个 round 函数有两个参数（第一个是 numeric，第二个是 integer）。所以下面的查询自动把第一个类型为 integer 的参数转换成 numeric 类型。

```
vastbase=# SELECT round(4, 4);
 round
--------
 4.0000
(1 row)
```

实际上它被分析器转换成：

```
vastbase=# SELECT round(CAST (4 AS numeric), 4);
```

因为带小数点的数值常量初始时被赋予 numeric 类型，因此下面的查询将不需要类型转换，并且可能会略微高效一些：

```
vastbase=# SELECT round(4.0, 4);
```

示例 2：子字符串函数类型解析。有好几个 substr 函数，其中一个接受 text 和 integer 类型。如果用一个未声明类型的字符串常量调用它，系统将选择接受 string 类型范畴的首选类型（也就是 text 类型）的候选函数。

```
vastbase=# SELECT substr('1234', 3);
 substr
--------
    34
(1 row)
```

如果该字符串声明为 varchar 类型，就像从表中取出来的数据一样，分析器将试着将其转换成 text 类型：

```
vastbase=# SELECT substr(varchar '1234', 3);
 substr
--------
    34
(1 row)
```

被分析器转换后实际上变成：

```
vastbase=# SELECT substr(CAST (varchar '1234' AS text), 3);
```

说明

分析器从 pg_cast 表中了解到 text 和 varchar 是二进制兼容的，意思是说一个可以传递给接受另一个的函数而不需要做任何物理转换。因此，在这种情况下，实际上没有做任何类型转换。

而且，如果以 integer 为参数调用函数，分析器将试图将其转换成 text 类型：

```
vastbase=# SELECT substr(1234, 3);
substr
--------
 34
(1 row)
```

被分析器转换后实际上变成：

```
vastbase=# SELECT substr(CAST (1234 AS text), 3);
 substr
--------
    34
(1 row)
```

11.7.4. 值存储

值存储数据类型解析

1. 查找与目标字段准确的匹配。
2. 试着将表达式直接转换成目标类型。如果已知这两种类型之间存在一个已注册的转换函数，那么直接调用该转换函数即可。如果表达式是一个未知类型文本，该文本字符串的内容将交给目标类型的输入转换过程。
3. 检查一下看目标类型是否有长度转换。长度转换是一个从某类型到自身的转换。如果在 pg_cast 表里面找到一个，那么在存储到目标字段之前先在表达式上应用。这样的转换函数总是接受一个额外的类型为 integer 的参数，它接收目标字段的 atttypmod 值（实际上是其声明长度，atttypmod 的解释随不同的数据类型而不同），并且它可能接受一个 Boolean 类型的第三个参数，表示转换是显式的还是隐式的。转换函数负责施加那些长度相关的语义，比如长度检查或者截断。

示例

character 存储类型转换。对一个目标列定义为 character(20)的语句，下面的语句显示存储值的长度正确：

```
vastbase=# CREATE TABLE tpcds.value_storage_t1 (
   VS_COL1 CHARACTER(20)
);
vastbase=# INSERT INTO tpcds.value_storage_t1 VALUES('abcdef');
vastbase=# SELECT VS_COL1, octet_length(VS_COL1) FROM tpcds.value_storage_t1;
     vs_col1       | octet_length
---------------------+--------------
 abcdef              |         20
(1 row)
)

vastbase=# DROP TABLE tpcds.value_storage_t1;
```

说明

这里真正发生的事情是两个 unknown 文本缺省解析成 text，这样就允许||操作符解析成 text 连接。然后操作符的 text 结果转换成 bpchar("空白填充的字符型"， character 类型内部名称)以匹配目标字段类型。不过，从 text 到 bpchar 的转换是二进制兼容的，这样的转换是隐含的并且实际上不做任何函数调用。最后，在系统表里找到长度转换函数 bpchar(bpchar, integer, Boolean) 并且应用于该操作符的结果和存储的字段长。这个类型相关的函数执行所需的长度检查和额外的空白填充。

11.7.5. UNION，CASE 和相关构造

SQL UNION 构造必须把那些可能不太相似的类型匹配起来成为一个结果集。解析算法分别应用于联合查询的每个输出字段。INTERSECT 和 EXCEPT 构造对不相同的类型使用和 UNION 相同的算法进行解

析。CASE、ARRAY、VALUES、GREATEST 和 LEAST 构造也使用同样的算法匹配它的部件表达式并且选择一个结果数据类型。

UNION，CASE 和相关构造解析

- 如果所有输入都是相同的类型，并且不是 unknown 类型，那么解析成这种类型。
- 如果所有输入都是 unknown 类型则解析成 text 类型（字符串类型范畴的首选类型）。否则，忽略 unknown 输入。
- 如果输入不属于同一个类型范畴，失败。（unknown 类型除外）
- 如果输入类型是同一个类型范畴，则选择该类型范畴的首选类型。（例外：union 操作会选择第一个分支的类型作为所选类型。）

说明

系统表 pg_type 中 typcategory 表示数据类型范畴， typispreferred 表示是否是 typcategory 分类中的首选类型。

- 把所有输入转换为所选的类型（对于字符串保持原有长度）。如果从给定的输入到所选的类型没有隐式转换则失败。
- 若输入中含 json、txid_snapshot、sys_refcursor 或几何类型，则不能进行 union。

对于 case 和 coalesce，在 TD 兼容模式下的处理

- 如果所有输入都是相同的类型，并且不是 unknown 类型，那么解析成这种类型。
- 如果所有输入都是 unknown 类型则解析成 text 类型。
- 如果输入字符串（包括 unknown，unknown 当 text 来处理）和数字类型，那么解析成字符串类型，如果是其他不同的类型范畴，则报错。
- 如果输入类型是同一个类型范畴，则选择该类型的优先级较高的类型。
- 把所有输入转换为所选的类型。如果从给定的输入到所选的类型没有隐式转换则失败。

示例

示例 1：Union 中的待定类型解析。这里，unknown 类型文本'b'将被解析成 text 类型。

```
vastbase=# SELECT text 'a' AS "text" UNION SELECT 'b';
 text
------
 a
 b
(2 rows)
```

示例 2：简单 Union 中的类型解析。文本 1.2 的类型为 numeric，而且 integer 类型的 1 可以隐含地转换为 numeric，因此使用这个类型。

```
vastbase=# SELECT 1.2 AS "numeric" UNION SELECT 1;
 numeric
---------
       1
     1.2
(2 rows)
```

示例 3：转置 Union 中的类型解析。这里，因为类型 real 不能被隐含转换成 integer，但是 integer 可以隐含转换成 real，那么联合的结果类型将是 real。

```
vastbase=# SELECT 1 AS "real" UNION SELECT CAST('2.2' AS REAL);
 real
------
    1
  2.2
(2 rows)
```

示例 4：TD 模式下，coalesce 参数输入 int 和 varchar 类型，那么解析成 varchar 类型。ORA 模式下会报错。

```
--在 A 模式下，创建 A 兼容模式的数据库 a_1。
vastbase=# CREATE DATABASE a_1 dbcompatibility = 'A';

--切换数据库为 a_1。
vastbase=# \c a_1

--创建表 t1。
a_1=# CREATE TABLE t1(a int, b varchar(10));

--查看 coalesce 参数输入 int 和 varchar 类型的查询语句的执行计划。
a_1=# EXPLAIN SELECT coalesce(a, b) FROM t1;
ERROR:  COALESCE types integer and varchar cannot be matched
LINE 1: EXPLAIN SELECT coalesce(a, b) FROM t1;
                               ^
CONTEXT:  referenced column: coalesce

--删除表。
a_1=# DROP TABLE t1;

--切换数据库为 vastbase。
a_1=# \c vastbase

--在 TD 模式下，创建 TD 兼容模式的数据库 td_1。
vastbase=# CREATE DATABASE td_1 dbcompatibility = 'C';

--切换数据库为 td_1。
vastbase=# \c td_1

--创建表 t2。
td_1=# CREATE TABLE t2(a int, b varchar(10));

--查看 coalesce 参数输入 int 和 varchar 类型的查询语句的执行计划。
td_1=# EXPLAIN VERBOSE select coalesce(a, b) from t2;
                                QUERY PLAN
-------------------------------------------------------------------------------------------
 Seq Scan on public.t2  (cost=0.00..24.18 rows=1134 width=42)
   Output: (COALESCE((a)::varchar, b)
(2 rows)

--删除表。
td_1=# DROP TABLE t2;
```

```
--切换数据库为 vastbase。
td_1=# \c vastbase

--删除 A 和 TD 模式的数据库。
vastbase=# DROP DATABASE a_1;
vastbase=# DROP DATABASE td_1;
```

11.8. 全文检索

11.8.1. 介绍

11.8.1.1. 全文检索概述

文本搜索操作符在数据库中已存在多年。Vastbase 为文本数据类型提供~、~*、LIKE 和 ILIKE 操作符；但它们缺乏现代信息系统所要求的许多必要属性。这些缺憾可以通过使用索引及词典进行解决。

文本检索缺乏信息系统所要求的必要属性：

- 没有语义支持，即使是英语。

  由于要识别派生词并不是那么容易，因此正则表达式也不能满足要求。如，satisfies 和 satisfy，当使用正则表达式寻找 satisfy 时，并不会查询到包含 satisfies 的文档。用户可以使用 OR 搜索多种派生形式，但过程非常繁琐。并且有些词会有上千的派生词，因此容易出错。

- 没有对搜索结果的分类（排序）。当搜索出成千的文档时，查找效率很低。
- 由于没有索引的支持，每一次的搜索需要遍历所有的文档，整体搜索比较缓慢。

使用全文索引可以对文档进行预处理，并且可以使后续的搜索更快速。预处理过程包括：

- 将文档解析成 token。

  为每个文档标记不同类别的 token 是非常有必要的，例如：数字、文字、复合词、电子邮件地址，这样就可以做不同的处理。原则上 token 的类别依赖于具体的应用，但对于大多数的应用来说，可以使用一组预定义的 token 类。

- 将 token 转换为词素。

  词素像 token 一样是一个字符串，但它已经标准化处理，这样同一个词的不同形式是一样的。例如，标准化通常包括：将大写字母转换成小写字母、删除后缀（如英语中的 s 或者 es）。这将允许通过搜索找到同一个词的不同形式，不需要繁琐地输入所有可能的变形样式。同时，这一步通常会删除停用词。这些停用词通常因为太常见而对搜索无用。（总之，token 是文档文本的原片段，而词素被认为是有用的索引和搜索词。）Vastbase 使用词典执行这一步，且提供了各种标准的词典。

- 保存搜索优化后的预处理文档。

比如，每个文档可以呈现为标准化词素的有序组合。伴随词素，通常还需要存储词素位置信息以用于邻近排序。因此文档包含的查询词越密集其排序越高。

词典能够对 token 如何标准化做到细粒度控制。使用合适的词典，可以定义不被索引的停用词。

数据类型 tsvector 用于存储预处理文档，tsquery 用于存储查询条件，详细请参见 11.3.10 文本搜索类型。为这些数据类型提供的函数和操作符请参见 11.5.12 文本检索函数和操作符。其中最重要的是匹配运算符@@，将在 11.8.1.3 基本文本匹配中介绍。

11.8.1.2. 文档概念

文档是全文搜索系统的搜索单元，例如：杂志上的一篇文章或电子邮件消息。文本搜索引擎必须能够解析文档，而且可以存储父文档的关联词素（关键词）。后续，这些关联词素用来搜索包含查询词的文档。

在 Vastbase 中，文档通常是一个数据库表中一行的文本字段，或者这些字段的可能组合（级联）。文档可能存储在多个表中或者需动态获取。换句话说，一个文档由被索引化的不同部分构成，因此无法存储为一个整体。比如：

```
create table date_dim(
d_dow int,
d_dom int,
d_fy_week_seq int
);

insert into date_dim values(5,6,1);
insert into date_dim values(0,8,1);
insert into date_dim values(2,3,1);
insert into date_dim values(3,4,1);
insert into date_dim values(4,5,1);
insert into date_dim values(1,2,1);
insert into date_dim values(6,7,1);
insert into date_dim values(4,5,2);
insert into date_dim values(1,2,2);
insert into date_dim values(6,7,2);

vastbase=# SELECT d_dow || '-' || d_dom || '-' || d_fy_week_seq  AS identify_serials FROM tpcds.date_dim
WHERE d_fy_week_seq = 1;
identify_serials
------------------
 5-6-1
 0-8-1
 2-3-1
 3-4-1
 4-5-1
 1-2-1
 6-7-1
(7 rows)
```

须知

实际上，在这些示例查询中，应该使用 coalesce 防止一个独立的 NULL 属性导致整个文档的 NULL 结果。

另外一种可能是：文档在文件系统中作为简单的文本文件存储。在这种情况下，数据库可以用于存储全文索引并且执行搜索，同时可以使用一些唯一标识从文件系统中检索文档。然而，从数据库外部检索文件需要拥有系统管理员权限或者特殊函数支持。因此，还是将所有数据保存在数据库中比较方便。同时，将所有数据保存在数据库中可以方便地访问文档元数据以便于索引和显示。

为了实现文本搜索目的，必须将每个文档减少至预处理后的 tsvector 格式。搜索和相关性排序都是在 tsvector 形式的文档上执行的。原始文档只有在被选中要呈现给用户时才会被检索。因此，我们常将 tsvector 说成文档，但是很显然其实它只是完整文档的一种紧凑表示。

11.8.1.3. 基本文本匹配

Vastbase 的全文检索基于匹配算子@@，当一个 tsvector(document)匹配到一个 tsquery(query)时，则返回 true。其中，tsvector(document)和 tsquery(query)两种数据类型可以任意排序。

```
vastbase=# SELECT 'a fat cat sat on a mat and ate a fat rat'::tsvector @@ 'cat & rat'::tsquery AS RESULT;
 result
----------
 t
(1 row)
vastbase=# SELECT 'fat & cow'::tsquery @@ 'a fat cat sat on a mat and ate a fat rat'::tsvector AS RESULT;
 result
----------
 f
(1 row)
```

正如上面例子表明，tsquery 不仅是文本，且比 tsvector 包含的要多。tsquery 包含已经标注化为词条的搜索词，同时可能是使用 AND、OR、或 NOT 操作符连接的多个术语。详细请参见 11.3.10 文本搜索类型。函数 to_tsquery 和 plainto_tsquery 对于将用户书写文本转换成适合的 tsquery 是非常有用的，比如将文本中的词标准化。类似地，to_tsvector 用于解析和标准化文档字符串。因此，实际中文本搜索匹配看起来更像这样：

```
vastbase=# SELECT to_tsvector('fat cats ate fat rats') @@ to_tsquery('fat & rat') AS RESULT;
result
----------
 t
(1 row)
```

需要注意的是，下面这种方式是不可行的：

```
vastbase=# SELECT 'fat cats ate fat rats'::tsvector @@ to_tsquery('fat & rat')AS RESULT;
result
----------
 f
(1 row)
```

由于 tsvector 没有对 rats 进行标准化，所以 rats 不匹配 rat。

@@操作符也支持 text 输入，允许一个文本字符串的显示转换为 tsvector 或者在简单情况下忽略 tsquery。可用形式是：

```
tsvector @@ tsquery
tsquery  @@ tsvector
text @@ tsquery
text @@ text
```

我们已经看到了前面两种，形式 text @@ tsquery 等价于 to_tsvector(text) @@ tsquery，而 text @@ text 等价于 to_tsvector(text) @@ plainto_tsquery(text)。

11.8.1.4. 分词器

全文检索功能还可以做更多事情：忽略索引某个词（停用词），处理同义词和使用复杂解析，例如：不仅基于空格的解析。这些功能通过文本搜索分词器控制。Vastbase 支持多语言的预定义的分词器，并且可以创建分词器（vsql 的\dF 命令显示了所有可用分词器）。

在安装期间选择一个合适的分词器，并且在 postgresql.conf 中相应的设置 default_text_search_config。如果为了 Vastbase 使用同一个文本搜索分词器可以使用 postgresql.conf 中的值。如果需要在 Vastbase 中使用不同分词器，可以使用 ALTER DATABASE ... SET 在任一数据库进行配置。用户也可以在每个会话中设置 default_text_search_config。

每个依赖于分词器的文本搜索函数有一个可选的配置参数，用以明确声明所使用的分词器。仅当忽略这个参数的时候，才使用 default_text_search_config。

为了更方便的建立自定义文本搜索分词器，可以通过简单的数据库对象建立分词器。 Vastbase 文本搜索功能提供了四种类型与分词器相关的数据库对象：

- 文本搜索解析器将文档分解为 token，并且分类每个 token（例如：词和数字）。
- 文本搜索词典将 token 转换成规范格式并且丢弃停用词。
- 文本搜索模板提供潜在的词典功能：一个词典指定一个模板，并且为模板设置参数。
- 文本搜索分词器选择一个解析器，并且使用一系列词典规范化语法分析器产生的 token。

11.8.2. 表和索引

11.8.2.1. 搜索表

在不使用索引的情况下也可以进行全文检索。

- 一个简单查询：将 body 字段中包含 america 的每一行打印出来。

```
vastbase=# DROP SCHEMA IF EXISTS tsearch CASCADE;

vastbase=# CREATE SCHEMA tsearch;

vastbase=# CREATE TABLE tsearch.pgweb(id int, body text, title text, last_mod_date date);

vastbase=# INSERT INTO tsearch.pgweb VALUES(1, 'China, officially the People''s Republic of
```

```
China (PRC), located in Asia, is the world''s most populous state.', 'China', '2010-1-1');

vastbase=# INSERT INTO tsearch.pgweb VALUES(2, 'America is a rock band, formed in England in
1970 by multi-instrumentalists Dewey Bunnell, Dan Peek, and Gerry Beckley.', 'America',
'2010-1-1');

vastbase=# INSERT INTO tsearch.pgweb VALUES(3, 'England is a country that is part of the United
Kingdom. It shares land borders with Scotland to the north and Wales to the west.', 'England',
'2010-1-1');

vastbase=# INSERT INTO tsearch.pgweb VALUES(4, 'Australia, officially the Commonwealth of
Australia, is a country comprising the mainland of the Australian continent, the island of
Tasmania, and numerous smaller islands.', 'Australia', '2010-1-1');

vastbase=# INSERT INTO tsearch.pgweb VALUES(5, 'Russia, also officially known as the Russian
Federation, is a sovereign state in northern Eurasia.', 'Russia', '2010-1-1');

vastbase=# INSERT INTO tsearch.pgweb VALUES(6, 'Japan is an island country in East Asia.',
'Japan', '2010-1-1');

vastbase=# INSERT INTO tsearch.pgweb VALUES(7, 'Germany, officially the Federal Republic of
Germany, is a sovereign state and federal parliamentary republic in central-western Europe.',
'Germany', '2010-1-1');

vastbase=# INSERT INTO tsearch.pgweb VALUES(8, 'France, is a sovereign state comprising
territory in western Europe and several overseas regions and territories.', 'France',
'2010-1-1');

vastbase=# INSERT INTO tsearch.pgweb VALUES(9, 'Italy officially the Italian Republic, is a
unitary parliamentary republic in Europe.', 'Italy', '2010-1-1');

vastbase=# INSERT INTO tsearch.pgweb VALUES(10, 'India, officially the Republic of India, is
a country in South Asia.', 'India', '2010-1-1');

vastbase=# INSERT INTO tsearch.pgweb VALUES(11, 'Brazil, officially the Federative Republic
of Brazil, is the largest country in both South America and Latin America.', 'Brazil', '2010-1-1');

vastbase=# INSERT INTO tsearch.pgweb VALUES(12, 'Canada is a country in the northern half of
North America.', 'Canada', '2010-1-1');

vastbase=# INSERT INTO tsearch.pgweb VALUES(13, 'Mexico, officially the United Mexican States,
is a federal republic in the southern part of North America.', 'Mexico', '2010-1-1');

vastbase=# SELECT id, body, title FROM tsearch.pgweb WHERE to_tsvector('english', body) @@
to_tsquery('english', 'america');
 id |                                            body
|  title
----+-------------------------------------------------------------------------------------------
-----------------------------------+---------
  2 | America is a rock band, formed in England in 1970 by multi-instrumentalists Dewey Bunnell,
Dan Peek, and Gerry Beckley. | America
 12 | Canada is a country in the northern half of North America.
| Canada
 13 | Mexico, officially the United Mexican States, is a federal republic in the southern part
of North America.           | Mexico
```

```
 11 | Brazil, officially the Federative Republic of Brazil, is the largest country in both South
America and Latin America.   | Brazil
(4 rows)
```

像 America 这样的相关词也会被找到，因为这些词都被处理成了相同标准的词条。

上面的查询指定 english 配置来解析和规范化字符串。当然也可以省略此配置，通过 default_text_search_config 进行配置设置：

```
vastbase=# SHOW default_text_search_config;
 default_text_search_config
----------------------------
 pg_catalog.english
(1 row)

vastbase=# SELECT id, body, title FROM tsearch.pgweb WHERE to_tsvector(body) @@
to_tsquery('america');
 id |                                                 body
|  title
----+-------------------------------------------------------------------------------------------
-----------------------------------+---------
 11 | Brazil, officially the Federative Republic of Brazil, is the largest country in both South
America and Latin America.   | Brazil
  2 | America is a rock band, formed in England in 1970 by multi-instrumentalists Dewey Bunnell,
Dan Peek, and Gerry Beckley. | America
 12 | Canada is a country in the northern half of North America.
| Canada
 13 | Mexico, officially the United Mexican States, is a federal republic in the southern part
of North America.            | Mexico
(4 rows)
```

- 一个复杂查询：检索出在 title 或者 body 字段中包含 north 和 america 的最近 10 篇文档：

```
vastbase=# SELECT title FROM tsearch.pgweb WHERE to_tsvector(title || ' ' || body) @@
to_tsquery('north & america') ORDER BY last_mod_date DESC LIMIT 10;
 title
--------
 Mexico
 Canada
(2 rows)
```

为了清晰，举例中没有调用 coalesce 函数在两个字段中查找包含 NULL 的行。

以上例子均在没有索引的情况下进行查询。对于大多数应用程序来说，这个方法很慢。因此除了偶尔的特定搜索，文本搜索在实际使用中通常需要创建索引。

11.8.2.2. 创建索引

为了加速文本搜索，可以创建 GIN 索引。

```
vastbase=# CREATE INDEX pgweb_idx_1 ON tsearch.pgweb USING gin(to_tsvector('english', body));
```

to_tsvector()函数有两个版本。只输一个参数的版本和输两个参数的版本。只输一个参数时，系统默认采用 default_text_search_config 所指定的分词器。

请注意：创建索引时必须使用 to_tsvector 的两参数版本。只有指定了分词器名称的全文检索函数才可以在索引表达式中使用。这是因为索引的内容必须不受 default_text_search_config 的影响，否则索引内容可能不一致。由于 default_text_search_config 的值可以随时调整，从而导致不同条目生成的 tsvector 采用了不同的分词器，并且没有办法区分究竟使用了哪个分词器。正确地转储和恢复这样的索引也是不可能的。

因为在上述创建索引中 to_tsvector 使用了两个参数，只有当查询时也使用了两个参数，且参数值与索引中相同时，才会使用该索引。也就是说，WHERE to_tsvector('english', body) @@ 'a & b' 可以使用索引，但 WHERE to_tsvector(body) @@ 'a & b'不能使用索引。这确保只使用这样的索引——索引各条目是使用相同的分词器创建的。

索引中的分词器名称由另一列指定时可以建立更复杂的表达式索引。例如：

```
vastbase=# CREATE INDEX pgweb_idx_2 ON tsearch.pgweb USING gin(to_tsvector('ngram', body));
```

其中 body 是 pgweb 表中的一列。当对索引的各条目使用了哪个分词器进行记录时，允许在同一索引中存在混合分词器。在某些场景下这将是有用的。例如，文档集合中包含不同语言的文档时。再次强调，打算使用索引的查询必须措辞匹配，例如，WHERE to_tsvector(config_name, body) @@ 'a & b'与索引中的 to_tsvector 措辞匹配。

索引甚至可以连接列：

```
vastbase=# CREATE INDEX pgweb_idx_3 ON tsearch.pgweb USING gin(to_tsvector('english', title || ' '
|| body));
```

另一个方法是创建一个单独的 tsvector 列控制 to_tsvector 的输出。下面的例子是 title 和 body 的连接， 当其它是 NULL 的时候，使用 coalesce 确保一个字段仍然会被索引：

```
vastbase=# ALTER TABLE tsearch.pgweb ADD COLUMN textsearchable_index_col tsvector;
vastbase=# UPDATE tsearch.pgweb SET textsearchable_index_col = to_tsvector('english',
coalesce(title,'') || ' ' || coalesce(body,''));
```

然后为加速搜索创建一个 GIN 索引：

```
vastbase=# CREATE INDEX textsearch_idx_4 ON tsearch.pgweb USING gin(textsearchable_index_col);
```

现在，就可以执行一个快速全文搜索了：

```
vastbase=# SELECT title
FROM tsearch.pgweb
WHERE textsearchable_index_col @@ to_tsquery('north & america')
ORDER BY last_mod_date DESC
LIMIT 10;

 title
--------
 Canada
 Mexico
(2 rows)
```

相比于一个表达式索引，单独列方法的一个优势是：它没有必要在查询时明确指定分词器以便能使用索引。正如上面例子所示，查询可以依赖于 default_text_search_config。另一个优势是搜索比较快速，因为它没有必要重新利用 to_tsvector 调用来验证索引匹配。表达式索引方法更容易建立，且它需要较少的磁盘空间，因为 tsvector 形式没有明确存储。

11.8.2.3. 索引使用约束

下面是一个使用索引的例子：

```
vastbase=# create table table1 (c_int int,c_bigint bigint,c_varchar varchar,c_text text)
with(orientation=row);

vastbase=# create text search configuration ts_conf_1(parser=POUND);
vastbase=# create text search configuration ts_conf_2(parser=POUND) with(split_flag='%');

vastbase=# set default_text_search_config='ts_conf_1';
vastbase=# create index idx1 on table1 using gin(to_tsvector(c_text));

vastbase=# set default_text_search_config='ts_conf_2';
vastbase=# create index idx2 on table1 using gin(to_tsvector(c_text));

vastbase=# select c_varchar,to_tsvector(c_varchar) from table1 where to_tsvector(c_text) @@
plainto_tsquery('￥#@……&**')  and to_tsvector(c_text) @@  plainto_tsquery('某公司 ')  and c_varchar
is not null order by 1 desc limit 3;
```

该例子的关键点是表 table1 的同一个列 c_text 上建立了两个 gin 索引：idx1 和 idx2，但这两个索引是在不同 default_text_search_config 的设置下建立的。该例子和同一张表的同一个列上建立普通索引的不同之处在于：

- gin 索引使用了不同的 parser（即分隔符不同），那么 idx1 和 idx2 的索引数据是不同的；
- 在同一张表的同一个列上建立的多个普通索引的索引数据是相同的。

因此当执行同一个查询时，使用 idx1 和 idx2 查询出的结果是不同的。

使用约束

通过上面的例子，索引使用满足如下条件时：

- 在同一个表的同一个列上建立了多个 gin 索引；
- 这些 gin 索引使用了不同的 parser（即分隔符不同）；
- 在查询中使用了该列，且执行计划中使用索引进行扫描；

为了避免使用不同 gin 索引导致查询结果不同的问题，需要保证在物理表的一列上只有一个 gin 索引可用。

11.8.2.4. pg_zhtrgm

创建扩展 pg_zhtrgm，可以简化全文索引的使用方式，并且该扩展不仅支持对英文字符全文检索，还允许对中文字符进行全文检索。

```
vastbase=# create extension pg_zhtrgm;
```

创建中文数据表如下：

```
vastbase=# create table t_full_text(id int, info text);
vastbase=# insert into t_full_text values (1, ' Vastbase 是一种特性非常齐全的自由软件的对象-关系型数据库管理系统（ORDBMS），Vastbase 支持大部分的 SQL 标准并且提供了很多其他现代特性，如复杂查询、外键、触发器、视图、事务完整性、多版本并发控制等。同样，PostgreSQL 也可以用许多方法扩展，例如通过增加新的数据类型、函数、操作符、聚集函数、索引方法、过程语言等。另外，因为许可证的灵活，任何人都可以以任何目的免费使用、修改和分发 PostgreSQL。');
```

在此表上创建以 pg_zhgrtm 提供的 gin_zhtrgm_ops、gist_zhtrgm_ops 操作符类所支持的 gin 索引或 gist 索引，该索引创建不再需要借助 to_tsvector 函数，同样地，使用该索引进行查询时也不再需要借助 to_tsquery 函数。

```
create index idx_full_text on t_full_text using gin(info gin_zhtrgm_ops);
create index idx_full_text on t_full_text using gist(info gist_zhtrgm_ops);
```

此类全文索引仅支持模糊查询，不支持等值匹配。对应地操作符如下：

操作符	返回类型	描述
text % text	boolean	如果其参数的相似度大于 pg_trgm.similarity_threshold 设置的当前相似度阈值，则返回 true。
text <% text	boolean	如果第一个参数的子集与第二个参数的有序集合的相似程度大于系统设定值，则返回 true。当前的相似性阈值通过 pg_trgm.word_similarity_threshold 参数设定。
text %> text	boolean	<%运算符的换向器。
text <-> text	real	返回参数之间的“距离”，即 1 减去 similarity()值。
text <<-> text	real	返回参数之间的“距离”，即 1 减去 word_similarity()值。
text <->> text	real	<<->运算符的换向器。

那么，可以很方便地进行单字、词语、短句的全文搜索：

```
select * from t_full_text where info like '%函%';
select * from t_full_text where info like '%视图%';
select * from t_full_text where info like '%查询%' and info not like '%外键%';
select * from t_full_text where info like '%Vastbase是一种特性非常齐全的自由软件的对象%';
```

11.8.3. 控制文本搜索

11.8.3.1. 解析文档

Vastbase 中提供了 to_tsvector 函数把文档处理成 tsvector 数据类型。

```
to_tsvector([ config regconfig, ] document text) returns tsvector
```

to_tsvector 将文本文档解析为 token，再将 token 简化到词素，并返回一个 tsvector。其中 tsvector 中列出了词素及它们在文档中的位置。文档是根据指定的或默认的文本搜索分词器进行处理的。这里有一个简单的例子：

```
vastbase=# SELECT to_tsvector('english', 'a fat  cat sat on a mat - it ate a fat rats');
                to_tsvector
-----------------------------------------------------
 'ate':9 'cat':3 'fat':2,11 'mat':7 'rat':12 'sat':4
```

通过以上例子可发现结果 tsvector 不包含词 a、on 或者 it，rats 变成 rat，并且忽略标点符号-。

to_tsvector 函数内部调用一个解析器，将文档的文本分解成 token 并给每个 token 指定一个类型。对于每个 token，有一系列词典可供查询。词典系列因 token 类型的不同而不同。识别 token 的第一本词典将发出一个或多个标准词素来表示 token。例如：

- rats 变成 rat 因为词典认为词 rats 是 rat 的复数形式。
- 有些词被作为停用词（请参考 11.8.6.2 停用词），这样它们就会被忽略，因为它们出现得太过频繁以致于搜索中没有用处。比如例子中的 a、on 和 it。
- 如果没有词典识别 token，那么它也被忽略。在这个例子中，符号“-”被忽略，因为词典没有给它分配 token 类型（空间符号），即空间记号永远不会被索引。

语法解析器、词典和要索引的 token 类型由选定的文本搜索分词器决定。可以在同一个数据库中有多种不同的分词器，以及提供各种语言的预定义分词器。在以上例子中，使用缺省分词器 english。

函数 setweight 可以给 tsvector 的记录加权重，权重是字母 A、B、C、D 之一。这通常用于标记来自文档不同部分的记录，比如标题、正文。之后，这些信息可以用于排序搜索结果。

因为 to_tsvector(NULL)会返回空，当字段可能是空的时候，建议使用 coalesce。以下是推荐的为结构化文档创建 tsvector 的方法：

```
vastbase=# CREATE TABLE tsearch.tt (id int, title text, keyword text, abstract text, body text, ti
tsvector);

vastbase=# INSERT INTO tsearch.tt(id, title, keyword, abstract, body) VALUES (1, 'China', 'Beijing',
'China','China, officially the People''s Republic of China (PRC), located in Asia, is the world''s
most populous state.');

vastbase=# UPDATE tsearch.tt SET ti =
   setweight(to_tsvector(coalesce(title,'')), 'A')    ||
   setweight(to_tsvector(coalesce(keyword,'')), 'B')  ||
   setweight(to_tsvector(coalesce(abstract,'')), 'C') ||
   setweight(to_tsvector(coalesce(body,'')), 'D');
vastbase=# DROP TABLE tsearch.tt;
```

上例使用 setweight 标记已完成的 tsvector 中的每个词的来源，并且使用 tsvector 连接操作符||合并标记过的 tsvector 值，11.8.4.1 处理 tsvector 一节详细介绍了这些操作。

11.8.3.2. 解析查询

Vastbase 提供了函数 to_tsquery 和 plainto_tsquery 将查询转换为 tsquery 数据类型，to_tsquery 提供比 plainto_tsquery 更多的功能，但对其输入要求更严格。

```
to_tsquery([ config regconfig, ] querytext text) returns tsquery
```

to_tsquery 从 querytext 中创建一个 tsquery，querytext 必须由布尔运算符& (AND)，| (OR)和! (NOT)分割的单个 token 组成。这些运算符可以用圆括弧分组。换句话说，to_tsquery 输入必须遵循 tsquery 输入的通用规则，具体请参见 11.3.10 文本搜索类型。不同的是基本 tsquery 以 token 表面值作为输入，而 to_tsquery 使用指定或默认分词器将每个 token 标准化成词素，并依据分词器丢弃属于停用词的 token。例如：

```
vastbase=# SELECT to_tsquery('english', 'The & Fat & Rats');
  to_tsquery
---------------
 'fat' & 'rat'
(1 row)
```

像在基本 tsquery 中的输入一样，weight(s)可以附加到每个词素来限制它只匹配那些有相同 weight(s)的 tsvector 词素。比如：

```
vastbase=# SELECT to_tsquery('english', 'Fat | Rats:AB');
   to_tsquery
------------------
 'fat' | 'rat':AB
(1 row)
```

同时，*也可以附加到词素来指定前缀匹配：

```
vastbase=# SELECT to_tsquery('supern:*A & star:A*B');
      to_tsquery
--------------------------
 'supern':*A & 'star':*AB
(1 row)
```

这样的词素将匹配 tsquery 中指定字符串和权重的项。

```
plainto_tsquery([ config regconfig, ] querytext text) returns tsquery
```

plainto_tsquery 将未格式化的文本 querytext 变换为 tsquery。类似于 to_tsvector，文本被解析并且标准化，然后在存在的词之间插入&(AND)布尔算子。

比如：

```
vastbase=# SELECT plainto_tsquery('english', 'The Fat Rats');
 plainto_tsquery
-----------------
 'fat' & 'rat'
(1 row)
```

请注意，plainto_tsquery 无法识别布尔运算符、权重标签，或在其输入中的前缀匹配标签：

```
vastbase=# SELECT plainto_tsquery('english', 'The Fat & Rats:C');
  plainto_tsquery
---------------------
 'fat' & 'rat' & 'c'
(1 row)
```

在这里，所有输入的标点符号作为空格符号丢弃。

11.8.3.3. 排序查询结果

排序试图针对特定查询衡量文档的相关度，从而将众多的匹配文档中相关度最高的文档排在最前。Vastbase 提供了两个预置的排序函数。函数考虑了词法，距离，和结构信息；也就是，他们考虑查询词在文档中出现的频率、紧密程度、以及他们出现的地方在文档中的重要性。然而，相关性的概念是模糊的，并且是跟应用强相关的。不同的应用程序可能需要额外的信息来排序，比如，文档的修改时间，内置的排序函数等。也可以开发自己的排序函数或者采用附加因素组合这些排序函数的结果来满足特定需求。

两个预置的排序函数：

```
ts_rank([ weights float4[], ] vector tsvector, query tsquery [, normalization integer ]) returns float4
```

基于词素匹配率对 vector 进行排序：

```
ts_rank_cd([ weights float4[], ] vector tsvector, query tsquery [, normalization integer ]) returns
float4
```

该函数需要位置信息的输入。因此它不能在"剥离"tsvector 值的情况下运行—它将总是返回零。

对于这两个函数，可选的 weights 参数提供给词加权重的能力，词的权重大小取决于所加的权值。权重阵列指定在排序时为每类词汇加多大的权重。

```
{D-weight, C-weight, B-weight, A-weight}
```

如果没有提供 weights，则使用缺省值：{0.1, 0.2, 0.4, 1.0}。

通常的权重是用来标记文档特殊领域的词，如标题或最初的摘要，所以相对于文章主体中的词它们有着更高或更低的重要性。

由于较长的文档有更多的机会包含查询词，因此有必要考虑文档的大小。例如，包含有 5 个搜索词的一百字文档比包含有 5 个搜索词的一千字文档相关性更高。两个预置的排序函数都采用了一个整型的标准化选项来定义文档长度是否影响排序及如何影响。这个整型选项控制多个行为，所以它是一个屏蔽字：可以使用|指定一个或多个行为（例如，2|4）。

- 0（缺省）表示：跟长度大小没有关系
- 1 表示：排名（rank）除以(文档长度的对数+1)
- 2 表示：排名除以文档的长度
- 4 表示：排名除以两个扩展词间的调和平均距离。只能使用 ts_rank_cd 实现
- 8 表示：排名除以文档中单独词的数量
- 16 表示：排名除以单独词数量的对数+1
- 32 表示：排名除以排名本身+1

当指定多个标志位时，会按照所列的顺序依次进行转换。

需要特别注意的是，排序函数不使用任何全局信息，所以不可能产生一个某些情况下需要的 1%或 100%的理想标准值。标准化选项 32 (rank/(rank+1))可用于所有规模的从零到一之间的排序，当然，这只是一个表面变化；它不会影响搜索结果的排序。

下面是一个例子，仅选择排名前十的匹配：

```
vastbase=# SELECT id, title, ts_rank_cd(to_tsvector(body), query) AS rank
FROM tsearch.pgweb, to_tsquery('america') query
WHERE query @@ to_tsvector(body)
ORDER BY rank DESC
LIMIT 10;
 id |  title  | rank
----+---------+------
 11 | Brazil  |   .2
  2 | America |   .1
 12 | Canada  |   .1
 13 | Mexico  |   .1
(4 rows)
```

这是使用标准化排序的相同例子：

```
vastbase=# SELECT id, title, ts_rank_cd(to_tsvector(body), query, 32 /* rank/(rank+1) */ ) AS rank
FROM tsearch.pgweb, to_tsquery('america') query
WHERE  query @@ to_tsvector(body)
ORDER BY rank DESC
LIMIT 10;
 id |  title  |   rank
----+---------+----------
 11 | Brazil  |  .166667
  2 | America | .0909091
```

```
 12 | Canada  | .0909091
 13 | Mexico  | .0909091
(4 rows)
```

下面是使用中文分词法排序查询的例子：

```
vastbase=# CREATE TABLE tsearch.ts_ngram(id int, body text);
vastbase=# INSERT INTO tsearch.ts_ngram VALUES(1, '中文');
vastbase=# INSERT INTO tsearch.ts_ngram VALUES(2, '中文检索');
vastbase=# INSERT INTO tsearch.ts_ngram VALUES(3, '检索中文');
--精确匹配
vastbase=# SELECT id, body, ts_rank_cd(to_tsvector('ngram',body), query) AS rank FROM tsearch.ts_ngram,
to_tsquery('中文') query WHERE query @@ to_tsvector(body);
 id | body | rank
----+------+------
  1 | 中文 |   .1
(1 row)

--模糊匹配
vastbase=# SELECT id, body, ts_rank_cd(to_tsvector('ngram',body), query) AS rank FROM tsearch.ts_ngram,
to_tsquery('中文') query WHERE query @@ to_tsvector('ngram',body);
 id |   body   | rank
----+----------+------
  3 | 检索中文 |   .1
  1 | 中文     |   .1
  2 | 中文检索 |   .1
(3 rows)
```

排序要遍历每个匹配的 tsvector，因此资源消耗多，可能会因为 I/O 限制导致排序慢。可是这是很难避免的，因为实际查询中通常会有大量的匹配。

11.8.3.4. 高亮搜索结果

搜索结果的理想显示是：列出每篇文档中与搜索相关的部分，并标识为什么与查询相关。搜索引擎能够显示标识了搜索词的文档片段。Vastbase 提供了函数 ts_headline 支持这部分功能。

```
ts_headline([ config regconfig, ] document text, query tsquery [, options text ]) returns text
```

ts_headline 的输入是带有查询条件的文档，其返回文档中的摘录，在摘录中查询词是高亮显示的。用来解析文档的分词器由 config 参数指定。如果省略 config，则使用 default_text_search_config 的值所指定的分词器。

指定 options 字符串时，需由一个或多个 option=value 对组成，且必须用逗号分隔。options 可以是下面的选项：

- StartSel，StopSel：分隔文档中出现的查询词，以区别于其他摘录词。当包含有空格或逗号时，必须用双引号将字符串引起来。
- MaxWords，MinWords：定义摘录的最长和最短值。
- ShortWord：在摘录的开始和结束会丢弃此长度或更短的词。默认值 3 会消除常见的英语冠词。
- HighlightAll：布尔标志。如果为真，整个文档将作为摘录。忽略前面三个参数的值。

❖ MaxFragments：要显示的文本摘录或片段的最大数量。默认值 0 表示选择非片段的摘录生成方法。 大于 0 的值表示选择基于片段的摘录生成。此方法查找带有尽可能多查询词的文本片段，并显示查询词周围的上下文片段。因此，查询词临近每个片段的中间，且查询词两边都有词。每个片段至多有 MaxWords，并且长度为 ShortWord 或更短的词在每一个片段开始和结束被丢弃。如果在文档中没有找到所有的查询词，则文档中开头将显示 MinWords 单片段。

❖ FragmentDelimiter：当有一个以上的片段时，通过该字符串分隔这些片段。

不声明选项时，采用下面的缺省值：

```
StartSel=<b>, StopSel=</b>,
MaxWords=35, MinWords=15, ShortWord=3, HighlightAll=FALSE,
MaxFragments=0, FragmentDelimiter=" ... "
```

例如：

```
vastbase=# SELECT ts_headline('english',
'The most common type of search
is to find all documents containing given query terms
and return them in order of their similarity to the
query.',
to_tsquery('english', 'query & similarity'));
                    ts_headline
------------------------------------------------------------
 containing given <b>query</b> terms
 and return them in order of their <b>similarity</b> to the
 <b>query</b>.
(1 row)

vastbase=# SELECT ts_headline('english',
'The most common type of search
is to find all documents containing given query terms
and return them in order of their similarity to the
query.',
to_tsquery('english', 'query & similarity'),
'StartSel = <, StopSel = >');
                   ts_headline
-------------------------------------------------------
 containing given <query> terms
 and return them in order of their <similarity> to the
 <query>.
(1 row)
```

ts_headline 使用原始文档，而不是 tsvector 摘录，因此使用起来会慢，应慎重使用。

11.8.4. 附加功能

11.8.4.1. 处理 tsvector

Vastbase 提供了用来操作 tsvector 类型的函数和操作符。

❖ tsvector || tsvector

tsvector 连接操作符返回一个新的 tsvector 类型，它综合了两个 tsvector 中词素和位置信息，并保留词素的位置信息和权重标签。右侧的 tsvector 的起始位置位于左侧 tsvector 的最后位置，因此，新生成的 tsvector 几乎等同于将两个原始文档字串连接后进行 to_tsvector 操作。（这个等价是不准确的，因为任何从左边 tsvector 中删除的停用词都不会影响结果，但是，在使用文本连接时，则会影响词素在右侧 tsvector 中的位置。）

相较于对文本进行连接后再执行 to_tsvector 操作，使用 tsvector 类型进行连接操作的优势在于，可以对文档的不同部分使用不同配置进行解析。因为 setweight 函数会对给定的 tsvector 中的语素进行统一设置，如果想要对文档的不同部分设置不同的权重，需要在连接之前对文本进行解析和权重设置。

- setweight(vector tsvector, weight "char") returns tsvector

  setweight 返回一个输入 tsvector 的副本，其中每一个位置都使用给定的权重做了标记。权值可以为 A、B、C 或 D（D 是 tsvector 副本的默认权重，并且不在输出中呈现）。当对 tsvector 进行连接操作时，这些权重标签将会被保留，文档不同部分以不同的权重进行排序。

须知

权重标签作用于位置，而不是词素。如果传入的 tsvector 已经被剥离了位置信息，那么 setweight 函数将什么都不做。

- length(vector tsvector) returns integer

  返回 vector 中的词素的数量。

- strip(vector tsvector) returns tsvector

  返回一个 tsvector 类型，其中包含输入的 tsvector 的同义词，但不包含任何位置和权重信息。虽然在相关性排序中，这里返回的 tsvector 要比未拆分的 tsvector 的作用小很多，但它通常都比未拆分的 tsvector 小的多。

11.8.4.2. 处理查询

Vastbase 提供了函数和操作符用来操作 tsquery 类型的查询。

- tsquery && tsquery

  返回两个给定查询 tsquery 的与结果。

- tsquery || tsquery

  返回两个给定查询 tsquery 的或结果。

- !! tsquery

  返回给定查询 tsquery 的非结果。

- numnode(query tsquery) returns integer

  返回 tsquery 中的节点数目（词素加操作符），这个函数在检查查询是否有效（返回值大于 0），或者只包含停用词（返回值等于 0）时，是有用的。例如：

```
vastbase=# SELECT numnode(plainto_tsquery('the any'));
NOTICE:  text-search query contains only stop words or doesn't contain lexemes, ignored
CONTEXT:  referenced column: numnode
 numnode
---------
       0

vastbase=# SELECT numnode('foo & bar'::tsquery);
 numnode
---------
       3
```

- querytree(query tsquery) returns text

  返回可用于索引搜索的 tsquery 部分，该函数对于检测非索引查询是有用的（例如只包含停用词或否定项）。例如:

```
vastbase=# SELECT querytree(to_tsquery('!defined'));
 querytree
-----------
 T
(1 row)
```

11.8.4.3. 查询重写

ts_rewrite 函数族可以从 tsquery 中搜索一个特定的目标子查询，并在该子查询每次出现的地方都替换为另一个子查询。 实际上这只是通过字串替换而得到的一个特定 tsquery 版本。目标子查询和替换查询组合起来可以被认为是一个重写规则。一组类似的重写规则可以为搜索提供强大的帮助。例如，可以使用同义词扩大搜索范围（例如，new york, big apple, nyc, gotham）或限制搜索范围在用户直接感兴趣的热点话题上。

- ts_rewrite (query tsquery, target tsquery, substitute tsquery) returns tsquery

  ts_rewrite 的这种形式只适用于一个单一的重写规则：任何出现目标子查询的地方都被无条件替换。例如：

```
vastbase=# SELECT ts_rewrite('a & b'::tsquery, 'a'::tsquery, 'c'::tsquery);
 ts_rewrite
------------
 'b' & 'c'
```

- ts_rewrite (query tsquery, select text) returns tsquery

  ts_rewrite 的这种形式接受一个起始查询和 SQL 查询命令。 这里的查询命令是文本字串形式，必须产生两个 tsquery 列。查询结果的每一行，第一个字段的值（目标子查询） 都会被第二个字段（替代子查询）替换。

说明

当多个规则需要重写时，重写顺序非常重要； 因此在实践中需要使用 ORDER BY 将源查询按照某些字段进行排序。

例如：举一个现实生活中天文学上的例子。我们将使用表驱动的重写规则扩大 supernovae 的查询范围：

```
vastbase=# CREATE TABLE tsearch.aliases (id int, t tsquery, s tsquery);

vastbase=# INSERT INTO tsearch.aliases VALUES(1, to tsquery('supernovae'),
to tsquery('supernovae|sn'));

vastbase=# SELECT ts rewrite(to tsquery('supernovae & crab'), 'SELECT t, s FROM
tsearch.aliases');

        ts_rewrite
---------------------------------
 'crab' & ( 'supernova' | 'sn' )
```

可以通过更新表修改重写规则：

```
vastbase=# UPDATE tsearch.aliases
SET s = to_tsquery('supernovae|sn & !nebulae')
WHERE t = to_tsquery('supernovae');

vastbase=# SELECT ts_rewrite(to_tsquery('supernovae & crab'), 'SELECT t, s FROM
tsearch.aliases');

             ts_rewrite
--------------------------------------------
 'crab' & ( 'supernova' | 'sn' & !'nebula' )
```

需要重写的规则越多，重写操作就越慢。因为它要检查每一个可能匹配的规则。为了过滤明显的非候选规则，可以使用 tsquery 类型的操作符来实现。在下面的例子中， 我们只选择那些可能与原始查询匹配的规则：

```
vastbase=# SELECT ts_rewrite('a & b'::tsquery, 'SELECT t,s FROM tsearch.aliases WHERE ''a &
b''::tsquery @> t');

 ts_rewrite
------------
 'b' & 'a'
(1 row)
vastbase=# DROP TABLE tsearch.aliases;
```

11.8.4.4. 收集文献统计

函数 ts_stat 可用于检查配置和查找候选停用词。

```
ts_stat(sqlquery text, [ weights text, ]
      OUT word text, OUT ndoc integer,
      OUT nentry integer) returns setof record
```

sqlquery 是一个包含 SQL 查询语句的文本，该 SQL 查询将返回一个 tsvector。ts_stat 执行 SQL 查询语句并返回一个包含 tsvector 中每一个不同的语素（词）的统计信息。返回信息包括：

- word text：词素。
- ndoc integer：词素在文档（tsvector）中的编号。
- nentry integer：词素出现的频率。

如果设置了权重条件，只有标记了对应权重的词素才会统计频率。例如，在一个文档集中检索使用频率最高的十个单词：

```
vastbase=# SELECT * FROM ts_stat('SELECT to_tsvector(''english'', sr_reason_sk) FROM
tpcds.store_returns WHERE sr_customer_sk < 10') ORDER BY nentry DESC, ndoc DESC, word LIMIT 10;
  word | ndoc | nentry
------+------+--------
 32   |    2 |      2
 33   |    2 |      2
 1    |    1 |      1
 10   |    1 |      1
 13   |    1 |      1
 14   |    1 |      1
 15   |    1 |      1
 17   |    1 |      1
 20   |    1 |      1
 22   |    1 |      1
(10 rows)
```

同样的情况，但是只计算权重为 A 或者 B 的单词使用频率：

```
vastbase=# SELECT * FROM ts_stat('SELECT to_tsvector(''english'', sr_reason_sk) FROM
tpcds.store_returns WHERE sr_customer_sk < 10', 'a') ORDER BY nentry DESC, ndoc DESC, word LIMIT 10;
 word | ndoc | nentry
------+------+--------
(0 rows)
```

11.8.5. 解析器

文本搜索解析器负责将原文档文本分解为多个 token，并标识每个 token 的类型。这里的类型集由解析器本身定义。注意，解析器并不修改文本，它只是确定合理的单词边界。由于这一限制，人们更需要定制词典，而不是为每个应用程序定制解析器。

目前 Vastbase 提供了三个内置的解析器，分别为 pg_catalog.default/pg_catalog.ngram/pg_catalog.pound，其中 pg_catalog.default 适用于英文分词场景，pg_catalog.ngram/pg_catalog.pound 是为了支持中文全文检索功能新增的两种解析器，适用于中文及中英混合分词场景。

内置解析器 pg_catalog.default，它能识别 23 种 token 类型，显示在表 11-35 中。

表 11-35. 默认解析器类型

别名	描述	示例
asciiword	Word, all ASCII letters	elephant
word	Word, all letters	mañana
numword	Word, letters and digits	beta1

别名	描述	示例
asciihword	Hyphenated word, all ASCII	up-to-date
hword	Hyphenated word, all letters	lógico-matemática
numhword	Hyphenated word, letters and digits	postgresql-beta1
hword_asciipart	Hyphenated word part, all ASCII	postgresql in the context postgresql-beta1
hword_part	Hyphenated word part, all letters	lógico or matemática in the context lógico-matemática
hword_numpart	Hyphenated word part, letters and digits	beta1 in the context postgresql-beta1
email	Email address	foo@example.com
protocol	Protocol head	http://
url	URL	example.com/stuff/index.html
host	Host	example.com
url_path	URL path	/stuff/index.html, in the context of a URL
file	File or path name	/usr/local/foo.txt, if not within a URL
sfloat	Scientific notation	-1.23E+56
float	Decimal notation	-1.234
int	Signed integer	-1234
uint	Unsigned integer	1234
version	Version number	8.3.0
tag	XML tag	<a href="dictionaries.html">
entity	XML entity	&
blank	Space symbols	(any whitespace or punctuation not otherwise recognized)

注意：对于解析器来说，一个“字母”的概念是由数据库的语言区域设置，即 lc_ctype 设置决定的。只包含基本 ASCII 字母的词被报告为一个单独的 token 类型，因为这类词有时需要被区分出来。大多数欧洲语言中，对 token 类型 word 和 asciiword 的处理方法是类似的。

email 不支持某些由 RFC 5322 定义的有效电子邮件字符。具体来说，可用于 email 用户名的非字母数字字符仅包含句号、破折号和下划线。

解析器可能对同一内容进行重叠 token。例如，包含连字符的单词将作为一个整体进行报告，其组件也会分别被报告：

```
vastbase=# SELECT alias, description, token FROM ts_debug('english','foo-bar-beta1');
    alias     |            description             |    token
```

```
-----------------+-----------------------------------------+---------------
 numhword       | Hyphenated word, letters and digits     | foo-bar-beta1
 hword_asciipart | Hyphenated word part, all ASCII         | foo
 blank          | Space symbols                           | -
 hword_asciipart | Hyphenated word part, all ASCII         | bar
 blank          | Space symbols                           | -
 hword_numpart   | Hyphenated word part, letters and digits | beta1
```

这种行为是有必要的，因为它支持搜索整个复合词和各组件。这里是另一个例子：

```
vastbase=# SELECT alias, description, token FROM
ts_debug('english','http://example.com/stuff/index.html');
  alias   |  description  |          token
----------+---------------+------------------------------
 protocol | Protocol head | http://
 url      | URL           | example.com/stuff/index.html
 host     | Host          | example.com
 url_path | URL path      | /stuff/index.html
```

N-gram 是一种机械分词方法，适用于无语义中文分词场景。N-gram 分词法可以保证分词的完备性，但是为了照顾所有可能，把很多不必要的词也加入到索引中，导致索引项增加。N-gram 支持中文编码包括 GBK、UTF-8。内置 6 种 token 类型，如表 11-36 所示。

表 11-36. token 类型

别名	描述
zh_words	chinese words
en_word	english word
numeric	numeric data
alnum	alnum string
grapsymbol	graphic symbol
multisymbol	multiple symbol

Pound 是一种固定格式分词方法，适用于无语意但待解析文本以固定分隔符分割开来的中英文分词场景。支持中文编码包括 GBK、UTF8，支持英文编码包括 ASCII。内置 6 种 token 类型，如表 11-37 所示；支持 5 种分隔符，如表 11-38 所示，在用户不进行自定义设置的情况下分隔符默认为“#”。Pound 限制单个 token 长度不能超过 256 个字符。

表 11-37. token 类型

别名	描述
zh_words	chinese words
en_word	english word
numeric	numeric data
alnum	alnum string

别名	描述
grapsymbol	graphic symbol
multisymbol	multiple symbol

表 11-38. 分隔符类型

分隔符	描述
@	Special character
#	Special character
$	Special character
%	Special character
/	Special character

11.8.6. 词典

11.8.6.1. 词典概述

词典用于定义停用词（stop words），即全文检索时不搜索哪些词。

词典还可以用于对同一词的不同形式进行规范化，这样同一个词的不同派生形式都可以进行匹配。规范化后的词称为词位（lexeme）。

除了提高检索质量外，词的规范化和删除停用词可以减少文档 tsvector 格式的大小， 从而提高性能。词的规范化和删除停用词并不总是具有语言学意义，用户可以根据应用环境在词典定义文件中自定义规范化和删除规则。

一个词典是一个程序，接收标记（token）作为输入，并返回：

- 如果 token 在词典中已知，返回对应 lexeme 数组（注意，一个标记可能对应多个 lexeme）。
- 一个 lexeme。一个新 token 会代替输入 token 被传递给后继词典（当前词典可被称为过滤词典）。
- 如果 token 在词典中已知，但它是一个停用词，返回空数组。
- 如果词典不能识别输入的 token，返回 NULL。

Vastbase 提供了多种语言的预定义字典，同时提供了五种预定义的词典模板，分别是 Simple，Synonym，Thesaurus，Ispell，和 Snowball，可用于创建自定义参数的新词典。

在使用全文检索时，建议用户：

- 可以在文本搜索配置中定义一个解析器，以及一组用于处理该解析器的输出标记词典。对于解析器返回的每个标记类型，可以在配置中指定不同的词典列表进行处理。当解析器输出一种类型的

标记后，在对应列表的每个字典中会查阅该标记，直到某个词典识别它。如果它被识别为一个停用词， 或者没有任何词典识别，该 token 将被丢弃，即不被索引或检索到。通常情况下，第一个返回非空结果的词典决定了最终结果，后继词典将不会继续处理。但是一个过滤类型的词典可以依据规则替换输入 token，然后将替换后的 token 传递给后继词典进行处理。

❖ 配置字典列表的一般规则是，第一个位置放置一个应用范围最小的、最具体化定义的词典，其次是更一般化定义的词典， 最后是一个普适定义的词典，比如 Snowball 词干词典或 Simple 词典。在下面例子中，对于一个针对天文学的文本搜索配置 astro_en，可以定义标记类型 asciiword（ASCII 词) 对应的词典列表为：天文术语的 Synonym 同义词词典， Ispell 英语词典和 Snowball 英语词干词典。

```
vastbase=# ALTER TEXT SEARCH CONFIGURATION astro_en
 ADD MAPPING FOR asciiword WITH astro_syn, english_ispell, english_stem;
```

过滤类型的词典可以放置在词典列表中除去末尾的任何地方，放置在末尾时是无效的。使用这些词典对标记进行部分规范化，可以有效简化后继词典的处理。

11.8.6.2. 停用词

停用词是很常见的词，几乎出现在每一个文档中，并且没有区分值。因此，在全文搜索的语境下可忽视它们。停用词处理逻辑和词典类型相关。例如，Ispell 词典会先对标记进行规范化，然后再查看停用词表，而 Snowball 词典会最先检查输入标记是否为停用词。

例如，每个英文文本包含像 a 和 the 的单词，因此没必要将它们存储在索引中。然而，停用词影响 tsvector 中的位置，同时位置也会影响相关度：

```
vastbase=# SELECT to_tsvector('english','in the list of stop words');
      to_tsvector
----------------------------
 'list':3 'stop':5 'word':6
```

位置 1、2、4 是停用词，所以不显示。为包含和不包含停用词的文档计算出的排序是完全不同的：

```
vastbase=# SELECT ts_rank_cd (to_tsvector('english','in the list of stop words'), to_tsquery('list
& stop'));
 ts_rank_cd
------------
       .05

vastbase=# SELECT ts_rank_cd (to_tsvector('english','list stop words'), to_tsquery('list & stop'));
 ts_rank_cd
------------
        .1
```

11.8.6.3. Simple 词典

Simple 词典首先将输入标记转换为小写字母，然后检查停用词表。如果识别为停用词则返回空数组，即表示该标记会被丢弃。否则，输入标记的小写形式作为规范化后的 lexeme 返回。此外，Simple 词典可通过设置参数 Accept 为 false（默认值 true），将非停用词报告为未识别，传递给后继词典继续处理。

注意事项

- 大多数词典的功能依赖于词典定义文件，词典定义文件名仅支持小写字母、数字、下划线组合。
- 临时模式 pg_temp 下不允许创建词典。
- 词典定义文件的字符集编码必须为 UTF-8 格式。实际应用时，如果与数据库的字符编码格式不一致，在读入词典定义文件时会进行编码转换。
- 通常情况下，每个 session 仅读取词典定义文件一次，当且仅当在第一次使用该词典时。需要修改词典文件时，可通过 ALTER TEXT SEARCH DICTIONARY 命令进行词典定义文件的更新和重新加载。

操作步骤

步骤 1 创建 Simple 词典。

```
vastbase=# CREATE TEXT SEARCH DICTIONARY public.simple_dict (
    TEMPLATE = pg_catalog.simple,
    STOPWORDS = english
);
```

其中，停用词表文件全名为 english.stop。关于创建 simple 词典的语法和更多参数，请参见 11.16.53CREATE TEXT SEARCH DICTIONARY。

步骤 2 使用 Simple 词典。

```
vastbase=# SELECT ts_lexize('public.simple_dict','YeS');
 ts_lexize
-----------
 {yes}
(1 row)

vastbase=# SELECT ts_lexize('public.simple_dict','The');
 ts_lexize
-----------
 {}
(1 row)
```

步骤 3 设置参数 ACCEPT=false，使 Simple 词典返回 NULL，而不是返回非停用词的小写形式。

```
vastbase=# ALTER TEXT SEARCH DICTIONARY public.simple_dict ( Accept = false );
ALTER TEXT SEARCH DICTIONARY
vastbase=# SELECT ts_lexize('public.simple_dict','YeS');
 ts_lexize
-----------

(1 row)

vastbase=# SELECT ts_lexize('public.simple_dict','The');
 ts_lexize
-----------
```

```
 {}
(1 row)
```

11.8.6.4. Synonym 词典

Synonym 词典用于定义、识别 token 的同义词并转化，不支持词组（词组形式的同义词可用 Thesaurus 词典定义，详细请参见 11.8.6.5Thesaurus 词典）。

示例

❖ Synonym 词典可用于解决语言学相关问题，例如，为避免使单词"Paris"变成"pari"，可在 Synonym 词典文件中定义一行"Paris paris"，并将该词典放置在预定义的 english_stem 词典之前。

```
vastbase=# SELECT * FROM ts_debug('english', 'Paris');
   alias   |   description   | token |  dictionaries  |  dictionary  | lexemes
-----------+-----------------+-------+----------------+--------------+---------
 asciiword | Word, all ASCII | Paris | {english_stem} | english_stem | {pari}
(1 row)

vastbase=# CREATE TEXT SEARCH DICTIONARY my_synonym (
    TEMPLATE = synonym,
    SYNONYMS = my_synonyms,
    FILEPATH = 'file:///home/dicts/'
);

vastbase=# ALTER TEXT SEARCH CONFIGURATION english
    ALTER MAPPING FOR asciiword
    WITH my_synonym, english_stem;

vastbase=# SELECT * FROM ts_debug('english', 'Paris');
   alias   |   description   | token |      dictionaries       | dictionary | lexemes
-----------+-----------------+-------+-------------------------+------------+---------
 asciiword | Word, all ASCII | Paris | {my_synonym,english_stem} | my_synonym | {paris}
(1 row)

vastbase=# SELECT * FROM ts_debug('english', 'paris');
   alias   |   description   | token |      dictionaries       | dictionary | lexemes
-----------+-----------------+-------+-------------------------+------------+---------
 asciiword | Word, all ASCII | Paris | {my_synonym,english_stem} | my_synonym | {paris}
(1 row)

vastbase=# ALTER TEXT SEARCH DICTIONARY my_synonym ( CASESENSITIVE=true);

vastbase=# SELECT * FROM ts_debug('english', 'Paris');
   alias   |   description   | token |      dictionaries       | dictionary | lexemes
-----------+-----------------+-------+-------------------------+------------+---------
 asciiword | Word, all ASCII | Paris | {my_synonym,english_stem} | my_synonym | {paris}
(1 row)
```

```
vastbase=# SELECT * FROM ts_debug('english', 'paris');
   alias   |   description   | token |      dictionaries       | dictionary | lexemes
-----------+-----------------+-------+-------------------------+------------+---------
 asciiword | Word, all ASCII | Paris | {my_synonym,english_stem} | my_synonym | {pari}
(1 row)
```

其中，同义词词典文件全名为 my_synonyms.syn，所在目录为当前连接数据库主节点的/home/dicts/下。关于创建词典的语法和更多参数，请参见 11.16.21ALTER TEXT SEARCH DICTIONARY。

❖ 星号（*）可用于词典文件中的同义词结尾，表示该同义词是一个前缀。在 to_tsvector()中该星号将被忽略，但在 to_tsquery()中会匹配该前缀并对应输出结果（参照 11.8.4.2 处理查询一节）。

假设词典文件 synonym_sample.syn 内容如下：

```
postgres      pgsql
postgresql    pgsql
postgre pgsql
gogle   googl
indices index*
```

创建并使用词典：

```
vastbase=# CREATE TEXT SEARCH DICTIONARY syn (
   TEMPLATE = synonym,
   SYNONYMS = synonym_sample
);

vastbase=# SELECT ts_lexize('syn','indices');
 ts_lexize
-----------
 {index}
(1 row)

vastbase=# CREATE TEXT SEARCH CONFIGURATION tst (copy=simple);

vastbase=# ALTER TEXT SEARCH CONFIGURATION tst ALTER MAPPING FOR asciiword WITH syn;

vastbase=# SELECT to_tsvector('tst','indices');
 to_tsvector
-------------
 'index':1
(1 row)

vastbase=# SELECT to_tsquery('tst','indices');
 to_tsquery
------------
 'index':*
(1 row)

vastbase=# SELECT 'indexes are very useful'::tsvector;
          tsvector
--------------------------------
 'are' 'indexes' 'useful' 'very'
(1 row)
```

```
vastbase=# SELECT 'indexes are very useful'::tsvector @@ to_tsquery('tst','indices');
 ?column?
----------
 t
(1 row)
```

11.8.6.5. Thesaurus 词典

Thesaurus 词典，也叫做分类词典（缩写为 TZ），是一组定义了词以及词组间关系的集合，包括广义词（BT）、狭义词（NT）、首选词、非首选词、相关词等。根据词典文件中的定义，TZ 词典用一个指定的短语替换对应匹配的所有短语，并且可选择保留原始短语进行索引。TZ 词典实际上是 Synonym 词典的一个扩展，增加了短语支持。

注意事项

- 由于 TZ 词典需要识别短语，所以在处理过程中必须保存当前状态并与解析器进行交互，以决定是否处理下一个 token 或是结束当前识别。此外，TZ 词典配置时需谨慎，如果设置 TZ 词典仅处理 asciiword 类型的 token，则类似 one 7 的分类词典定义将不会生效，因为 uint 类型的 token 不会传给 TZ 词典处理。
- 在索引期间要用到分类词典，因此分类词典参数中的任何变化都要求重新索引。对于其他大多数类型的词典来说，类似添加或删除停用词这种修改并不需要强制重新索引。

操作步骤

步骤 1 创建一个名为 thesaurus_astro 的 TZ 词典。

以一个简单的天文学词典 thesaurus_astro 为例，其中定义了两组天文短语及其同义词如下：

```
supernovae stars : sn
crab nebulae : crab
```

执行如下语句创建 TZ 词典：

```
vastbase=# CREATE TEXT SEARCH DICTIONARY thesaurus_astro (
    TEMPLATE = thesaurus,
    DictFile = thesaurus_astro,
    Dictionary = pg_catalog.english_stem,
    FILEPATH = 'file:///home/dicts/'
);
```

其中，词典定义文件全名为 thesaurus_astro.ths，所在目录为当前连接数据库主节点的/home/dicts/下 。子词典 pg_catalog.english_stem 是预定义的 Snowball 类型的英语词干词典，用于规范化输入词，子词典自身相关配置（例如停用词等）不在此处显示。关于创建词典的语法和更多参数，请参见 11.16.53CREATE TEXT SEARCH DICTIONARY。

步骤 2 创建词典后，将其绑定到对应文本搜索配置中需要处理的 token 类型上：

```
vastbase=# ALTER TEXT SEARCH CONFIGURATION russian
    ALTER MAPPING FOR asciiword, asciihword, hword_asciipart
    WITH thesaurus_astro, english_stem;
```

步骤 3 使用 TZ 词典。

- 测试 TZ 词典。

  ts_lexize 函数对于测试 TZ 词典作用不大，因为该函数是按照单个 token 处理输入。可以使用 plainto_tsquery、to_tsvector、to_tsquery 函数测试 TZ 词典，这些函数能够将输入分解成多个 token（to_tsquery 函数需要将输入加上引号）。

```
vastbase=# SELECT plainto_tsquery('russian','supernova star');
 plainto_tsquery
-----------------
 'sn'
(1 row)

vastbase=# SELECT to_tsvector('russian','supernova star');
 to_tsvector
-------------
 'sn':1
(1 row)

vastbase=# SELECT to_tsquery('russian','''supernova star''');
 to_tsquery
------------
 'sn'
(1 row)
```

  其中，supernova star 匹配了词典 thesaurus_astro 定义中的 supernovae stars，这是因为在 thesaurus_astro 词典定义中指定了 Snowball 类型的子词典 english_stem，该词典移除了 e 和 s。

- 如果同时需要索引原始短语，只要将其同时放置在词典定义文件中对应定义的右侧即可，如下：

```
supernovae stars : sn supernovae stars

vastbase=# ALTER TEXT SEARCH DICTIONARY thesaurus_astro (
   DictFile = thesaurus_astro,
   FILEPATH = 'file:///home/dicts/');

vastbase=# SELECT plainto_tsquery('russian','supernova star');
     plainto_tsquery
-----------------------------
 'sn' & 'supernova' & 'star'
(1 row)
```

11.8.6.6. Ispell 词典

Ispell 词典模板支持词法词典，它可以把一个词的各种语言学形式规范化成相同的词位。比如，一个 Ispell 英语词典可以匹配搜索词 bank 的词尾变化和词形变化，如 banking、banked、banks、banks' 和 bank's 等。

Vastbase 不提供任何预定义的 Ispell 类型词典或词典文件。dict 文件和 affix 文件支持多种开源词典格式，包括 Ispell、MySpell 和 Hunspell 等。

操作步骤

步骤 1 获取词典定义文件和词缀文件。

用户可以使用开源词典，直接获取的开源词典后缀名可能为.aff 和.dic，此时需要将扩展名改为.affix 和.dict。此外，对于某些词典文件，还需要使用下面的命令把字符转换成 UTF-8 编码，比如挪威语词典：

```
iconv -f ISO_8859-1 -t UTF-8 -o nn_no.affix nn_NO.aff
iconv -f ISO_8859-1 -t UTF-8 -o nn_no.dict nn_NO.dic
```

步骤 2 创建 Ispell 词典。

```
vastbase=# CREATE TEXT SEARCH DICTIONARY norwegian_ispell (
   TEMPLATE = ispell,
   DictFile = nn_no,
   AffFile = nn_no,
   FilePath = 'file:///home/dicts'
);
```

其中，词典文件全名为 nn_no.dict 和 nn_no.affix，所在目录为当前连接数据库主节点的/home/dicts/下 。关于创建词典的语法和更多参数，请参见 11.16.53CREATE TEXT SEARCH DICTIONARY。

步骤 3 使用 Ispell 词典进行复合词拆分。

```
vastbase=# SELECT ts_lexize('norwegian_ispell', 'sjokoladefabrikk');
     ts_lexize
----------------------
 {sjokolade,fabrikk}
(1 row)
```

MySpell 不支持复合词，Hunspell 对复合词有较好的支持。Vastbase 仅支持 Hunspell 中基本的复合词操作。通常情况下，Ispell 词典能够识别的词是一个有限集合，其后应该配置一个更广义的词典，例如一个可以识别所有词的 Snowball 词典。

11.8.6.7. Snowball 词典

Snowball 词典模板支持词干分析词典，基于 Martin Porter 的 Snowball 项目，内置有许多语言的词干分析算法。Vastbase 中预定义有多种语言的 Snowball 词典，可通过系统表 14.2.63PG_TS_DICT 查看预定义的词干分析词典以及支持的语言词干分析算法。

无论是否可以简化，Snowball 词典将标示所有输入为已识别，因此它应当被放置在词典列表的最后。把 Snowball 词典放在任何其他词典前面会导致后继词典失效，因为输入 token 不会通过 Snowball 词典进入到下一个词典。

关于 Snowball 词典的语法，请参见 11.16.53CREATE TEXT SEARCH DICTIONARY。

11.8.7. 配置示例

文本搜索配置（Text Search Configuration），指定了将文档转换成 tsvector 过程中所必需的组件：

- 解析器，用于把文本分解成标记 token；
- 词典列表，用于将每个 token 转换成词位 lexeme。

每次 to_tsvector 或 to_tsquery 函数调用时，都需要指定一个文本搜索配置来指定具体的处理过程。GUC 参数 default_text_search_config 指定了默认的文本搜索配置，当文本搜索函数中没有显式指定文本搜索配置参数时，将会使用该默认值进行处理。

Vastbase 中预定义有一些可用的文本搜索配置，用户也可创建自定义的文本搜索配置。此外，为了便于管理文本搜索对象，还提供有多个 vsql 元命令，可以显示有关文本搜索对象的信息（详细请参见《工具参考》中“客户端工具 >元命令参考”章节）。

操作步骤

步骤 1 创建一个文本搜索配置 ts_conf，复制预定义的文本搜索配置 english。

```
vastbase=# CREATE TEXT SEARCH CONFIGURATION ts_conf ( COPY = pg_catalog.english );
CREATE TEXT SEARCH CONFIGURATION
```

步骤 2 创建 Synonym 词典。

假设同义词词典定义文件 pg_dict.syn 内容如下：

```
postgres    pg
pgsql       pg
postgresql  pg
```

执行如下语句创建 Synonym 词典：

```
vastbase=# CREATE TEXT SEARCH DICTIONARY pg_dict (
    TEMPLATE = synonym,
    SYNONYMS = pg_dict,
    FILEPATH = 'file:///home/dicts'
);
```

步骤 3 创建一个 Ispell 词典 english_ispell（词典定义文件来自开源词典）。

```
vastbase=# CREATE TEXT SEARCH DICTIONARY english_ispell (
   TEMPLATE = ispell,
   DictFile = english,
   AffFile = english,
   StopWords = english,
   FILEPATH = 'file:///home/dicts'
);
```

步骤 4 设置文本搜索配置 ts_conf，修改某些类型的 token 对应的词典列表。关于 token 类型的详细信息，请参见 11.8.5 解析器。

```
vastbase=# ALTER TEXT SEARCH CONFIGURATION ts_conf
   ALTER MAPPING FOR asciiword, asciihword, hword_asciipart,
                     word, hword, hword_part
   WITH pg_dict, english_ispell, english_stem;
```

步骤 5 在文本搜索配置中，选择设置不索引或搜索某些 token 类型。

```
vastbase=# ALTER TEXT SEARCH CONFIGURATION ts_conf
   DROP MAPPING FOR email, url, url_path, sfloat, float;
```

步骤 6 使用文本检索调测函数 ts_debug()对所创建的词典配置 ts_conf 进行测试。

```
vastbase=# SELECT * FROM ts_debug('ts_conf', '
PostgreSQL, the highly scalable, SQL compliant, open source object-relational
database management system, is now undergoing beta testing of the next
version of our software.
');
```

步骤 7 可以设置当前 session 使用 ts_conf 作为默认的文本搜索配置。此设置仅在当前 session 有效。

```
vastbase=# \dF+ ts_conf
    Text search configuration "public.ts_conf"
Parser: "pg_catalog.default"
    Token     |          Dictionaries
--------------+-----------------------------------
 asciihword     | pg_dict,english_ispell,english_stem
 asciiword      | pg_dict,english_ispell,english_stem
 file           | simple
 host           | simple
 hword          | pg_dict,english_ispell,english_stem
 hword_asciipart | pg_dict,english_ispell,english_stem
 hword_numpart   | simple
 hword_part     | pg_dict,english_ispell,english_stem
 int            | simple
 numhword       | simple
 numword        | simple
 uint           | simple
 version        | simple
 word           | pg_dict,english_ispell,english_stem

vastbase=# SET default_text_search_config = 'public.ts_conf';
SET
vastbase=# SHOW default_text_search_config;
 default_text_search_config
----------------------------
 public.ts_conf
(1 row)
```

11.8.8. 测试和调试文本搜索

11.8.8.1. 分词器测试

函数 ts_debug 允许简单测试文本搜索分词器。

```
ts_debug([ config regconfig, ] document text,
       OUT alias text,
       OUT description text,
       OUT token text,
       OUT dictionaries regdictionary[],
       OUT dictionary regdictionary,
```

```
      OUT lexemes text[])
      returns setof record
```

ts_debug 显示 document 的每个 token 信息，token 是由解析器生成，由指定的词典进行处理。如果忽略对应参数，则使用 config 指定的分词器或者 default_text_search_config 指定的分词器。

ts_debug 为文本解析器标识的每个 token 返回一行记录。记录中的列分别是：

- alias：text 类型，token 的别名。
- description：text 类型，token 的描述。
- token：text 类型，token 的文本内容。
- dictionaries：regdictionary 数组类型，是分词器为 token 选定的词典。
- dictionary：regdictionary 类型，用来识别 token 的词典。如果为空，则不做识别。
- lexemes：text 数组类型，词典识别 token 时生成的词素。如果为空，则不生成词素。空数组（{}）意味着 token 将被识别成停用词。

一个简单的例子：

```
vastbase=# SELECT * FROM ts_debug('english','a fat  cat sat on a mat - it ate a fat rats');
  alias   |   description   | token |  dictionaries  |  dictionary  | lexemes
-----------+-----------------+-------+----------------+--------------+---------
 asciiword | Word, all ASCII | a     | {english_stem} | english_stem | {}
 blank     | Space symbols   |       | {}             |              |
 asciiword | Word, all ASCII | fat   | {english_stem} | english_stem | {fat}
 blank     | Space symbols   |       | {}             |              |
 asciiword | Word, all ASCII | cat   | {english_stem} | english_stem | {cat}
 blank     | Space symbols   |       | {}             |              |
 asciiword | Word, all ASCII | sat   | {english_stem} | english_stem | {sat}
 blank     | Space symbols   |       | {}             |              |
 asciiword | Word, all ASCII | on    | {english_stem} | english_stem | {}
 blank     | Space symbols   |       | {}             |              |
 asciiword | Word, all ASCII | a     | {english_stem} | english_stem | {}
 blank     | Space symbols   |       | {}             |              |
 asciiword | Word, all ASCII | mat   | {english_stem} | english_stem | {mat}
 blank     | Space symbols   |       | {}             |              |
 blank     | Space symbols   | -     | {}             |              |
 asciiword | Word, all ASCII | it    | {english_stem} | english_stem | {}
 blank     | Space symbols   |       | {}             |              |
 asciiword | Word, all ASCII | ate   | {english_stem} | english_stem | {ate}
 blank     | Space symbols   |       | {}             |              |
 asciiword | Word, all ASCII | a     | {english_stem} | english_stem | {}
 blank     | Space symbols   |       | {}             |              |
 asciiword | Word, all ASCII | fat   | {english_stem} | english_stem | {fat}
 blank     | Space symbols   |       | {}             |              |
 asciiword | Word, all ASCII | rats  | {english_stem} | english_stem | {rat}
(24 rows)
```

11.8.8.2. 解析器测试

函数 ts_parse 可以直接测试文本搜索解析器。

```
ts_parse(parser_name text, document text,
      OUT tokid integer, OUT token text) returns setof record
```

ts_parse 解析指定的 document 并返回一系列的记录，一条记录代表一个解析生成的 token。每条记录包括标识 token 类型的 tokid，及 token 文本。例如：

```
vastbase=# SELECT * FROM ts_parse('default', '123 - a number');
 tokid | token
-------+--------
    22 | 123
    12 |
    12 | -
     1 | a
    12 |
     1 | number
(6 rows)
```

函数 ts_token_type 返回指定解析器的 token 类型及其描述信息。

```
ts_token_type(parser_name text, OUT tokid integer,
          OUT alias text, OUT description text) returns setof record
```

ts_token_type 返回一个表，这个表描述了指定解析器可以识别的每种 token 类型。对于每个 token 类型，表中给出了整数类型的 tokid--用于解析器标记对应的 token 类型；alias——命名分词器命令中的 token 类型；及简单描述。比如：

```
vastbase=# SELECT * FROM ts_token_type('default');
 tokid |     alias      |            description
-------+----------------+-----------------------------------------
     1 | asciiword      | Word, all ASCII
     2 | word           | Word, all letters
     3 | numword        | Word, letters and digits
     4 | email          | Email address
     5 | url            | URL
     6 | host           | Host
     7 | sfloat         | Scientific notation
     8 | version        | Version number
     9 | hword_numpart  | Hyphenated word part, letters and digits
    10 | hword_part     | Hyphenated word part, all letters
    11 | hword_asciipart | Hyphenated word part, all ASCII
    12 | blank          | Space symbols
    13 | tag            | XML tag
    14 | protocol       | Protocol head
    15 | numhword       | Hyphenated word, letters and digits
    16 | asciihword     | Hyphenated word, all ASCII
    17 | hword          | Hyphenated word, all letters
    18 | url_path       | URL path
    19 | file           | File or path name
    20 | float          | Decimal notation
    21 | int            | Signed integer
    22 | uint           | Unsigned integer
    23 | entity         | XML entity
(23 rows)
```

11.8.8.3. 词典测试

函数 ts_lexize 用于进行词典测试。

ts_lexize(dict regdictionary, token text) returns text[]如果输入的 token 可以被词典识别，那么 ts_lexize 返回词素的数组；如果 token 可以被词典识别到它是一个停用词，则返回空数组；如果是一个不可识别的词则返回 NULL。

比如：

```
vastbase=# SELECT ts_lexize('english_stem', 'stars');
 ts_lexize
-----------
 {star}

vastbase=# SELECT ts_lexize('english_stem', 'a');
 ts_lexize
-----------
 {}
```

须知

ts_lexize 函数支持单一 token，不支持文本。

11.8.9. 限制约束

Vastbase 的全文检索功能当前限制约束是：

- 每个分词长度必须小于 2K 字节。
- tsvector 结构（分词+位置）的长度必须小于 1 兆字节。
- tsvector 的位置值必须大于 0，且小于等于 16,383。
- 每个分词在文档中位置数必须小于 256，若超过将舍弃后面的位置信息。
- tsquery 中的关键字及对应运算符最大支持到 32768。

11.9. 并行查询

功能描述

一条查询语句在涉及表中数据量达到阈值时，顺序扫描性能较差，因此使用多个 worker 线程并行查询该表中部分数据。目前支持顺序扫描的并行查询，其他索引扫描不支持并行。

示例

```
create table test_parallel(id int, name text, age int);
insert into test_parallel values(generate_series(1,1000000), 'Vastbase', 34);

analyze test_parallel;
explain select count(*) from test_parallel where id < 100;
explain select count(*) from test_parallel where id < 100;
```

11.10. 全局分区索引

功能描述

支持创建全局分区索引。分区索引可以分为本地索引（local index）和全局索引（global index），建立分区索引可以分散数据库的 IO 压力。就像分区表一样，分区索引提高了可管理性、可用性、性能和可伸缩性。它们既可以独立分区(全局索引)，也可以自动链接到表的分区(本地索引)。

示例

```
create table t_part_1(
id int,
col int
)
partition by range (id)
(
partition p1 values less than (10),
partition p2 values less than (20),
partition p3 values less than (30),
partition p4 values less than (maxvalue)
);
create index idx_tp_1 on t_part_1(id);
```

11.11. 系统操作

Vastbase 通过 SQL 语句执行不同的系统操作，比如：设置变量，显示执行计划和垃圾收集等操作。

设置变量

设置会话或事务中需要使用的各种参数，请参考 11.16.105 SET。

显示执行计划

显示 Vastbase 为 SQL 语句规划的执行计划，请参考 11.16.85 EXPLAIN。

事务日志检查点

预写式日志（WAL）缺省时在事务日志中每隔一段时间放置一个检查点。CHECKPOINT 强迫立即进行检查，而不是等到下一次调度时的检查点。请参考 11.16.29 CHECKPOINT。

垃圾收集

进行垃圾收集以及可选择的对数据库进行分析。请参考 11.16.114VACUUM

收集统计信息

收集与数据库中表内容相关的统计信息。请参考 11.16.26 ANALYZE | ANALYSE。

设置当前事务的约束检查模式

设置当前事务里的约束检查的特性。请参考 11.16.106 SET CONSTRAINTS。

11.12. 事务控制

事务是用户定义的一个数据库操作序列，这些操作要么全做要么全不做，是一个不可分割的工作单位。

启动事务

Vastbase 通过 START TRANSACTION 和 BEGIN 语法启动事务，请参考 11.16.111 START TRANSACTION 和 11.16.27 BEGIN。

设置事务

Vastbase 通过 SET TRANSACTION 或者 SET LOCAL TRANSACTION 语法设置事务，请参考 11.16.109 SET TRANSACTION。

提交事务

Vastbase 通过 COMMIT 或者 END 可完成提交事务的功能，即提交事务的所有操作，请参考 11.16.33 COMMIT | END。

回滚事务

回滚是在事务运行的过程中发生了某种故障，事务不能继续执行，系统将事务中对数据库的所有已完成的操作全部撤销。请参考 11.16.100 ROLLBACK。

说明

数据库中收到的一次执行请求（不在事务块中），如果含有多条语句，将会被打包成一个事务，如果其中有一个语句失败，那么整个请求都将会被回滚。

11.13. DDL 语法一览表

DDL（Data Definition Language 数据定义语言），用于定义或修改数据库中的对象。如：表、索引、视图等。

说明

Vastbase 不支持数据库主节点不完整时进行 DDL 操作。例如：Vastbase 中有 1 个数据库主节点故障时执行新建数据库、表等操作都会失败。

定义数据库

数据库是组织、存储和管理数据的仓库，而数据库定义主要包括：创建数据库、修改数据库属性，以及删除数据库。所涉及的 SQL 语句，请参考表 11-39。

表 11-39. 数据库定义相关 SQL

功能	相关 SQL
创建数据库	11.16.36CREATE DATABASE
修改数据库属性	11.16.2ALTER DATABASE
删除数据库	11.16.63DROP DATABASE

定义模式

模式是一组数据库对象的集合，主要用于控制对数据库对象的访问。所涉及的 SQL 语句，请参考表 11-40。

表 11-40. 模式定义相关 SQL

功能	相关 SQL
创建模式	11.16.45CREATE SCHEMA
修改模式属性	11.16.12ALTER SCHEMA
删除模式	11.16.73DROP SCHEMA

定义表空间

表空间用于管理数据对象，与磁盘上的一个目录对应。所涉及的 SQL 语句，请参考表 11-41。

表 11-41. 表空间定义相关 SQL

功能	相关 SQL
创建表空间	11.16.51CREATE TABLESPACE

功能	相关 SQL
修改表空间属性	11.16.19ALTER TABLESPACE
删除表空间	11.16.77DROP TABLESPACE

定义表

表是数据库中的一种特殊数据结构，用于存储数据对象以及对象之间的关系。所涉及的 SQL 语句，请参考表 11-42。

表 11-42. 表定义相关 SQL

功能	相关 SQL
创建表	11.16.48CREATE TABLE
修改表属性	11.16.17ALTER TABLE
删除表	11.16.76DROP TABLE

定义分区表

分区表是一种逻辑表，数据是由普通表存储的，主要用于提升查询性能。所涉及的 SQL 语句，请参考表 11-43。

表 11-43. 分区表定义相关 SQL

功能	相关 SQL
创建分区表	11.16.50CREATE TABLE PARTITION
创建分区	11.16.18ALTER TABLE PARTITION
修改分区表属性	11.16.18ALTER TABLE PARTITION
删除分区	11.16.18ALTER TABLE PARTITION
删除分区表	11.16.76DROP TABLE

定义索引

索引是对数据库表中一列或多列的值进行排序的一种结构，使用索引可快速访问数据库表中的特定信息。所涉及的 SQL 语句，请参考表 11-44。

表 11-44. 索引定义相关 SQL

功能	相关 SQL
创建索引	11.16.41CREATE INDEX
修改索引属性	11.16.8ALTER INDEX
删除索引	11.16.68DROP INDEX
重建索引	11.16.96REINDEX

定义存储过程

存储过程是一组为了完成特定功能的 SQL 语句集，经编译后存储在数据库中，用户通过指定存储过程的名称并给出参数（如果该存储过程带有参数）来执行它。所涉及的 SQL 语句，请参考表 11-45。

表 11-45. 存储过程定义相关 SQL

功能	相关 SQL
创建存储过程	11.16.43CREATE PROCEDURE
删除存储过程	11.16.71DROP PROCEDURE

定义函数

在 Vastbase 中，它和存储过程类似，也是一组 SQL 语句集，使用上没有差别。所涉及的 SQL 语句，请参考表 11-46。

表 11-46. 函数定义相关 SQL

功能	相关 SQL
创建函数	11.16.39CREATE FUNCTION
修改函数属性	11.16.6ALTER FUNCTION
删除函数	11.16.66DROP FUNCTION

定义视图

视图是从一个或几个基本表中导出的虚表，可用于控制用户对数据访问，请参考表 11-47。

表 11-47. 视图定义相关 SQL

功能	相关 SQL
创建视图	11.16.57CREATE VIEW
删除视图	11.16.83DROP VIEW

定义游标

为了处理 SQL 语句，存储过程进程分配一段内存区域来保存上下文联系。游标是指向上下文区域的句柄或指针。借助游标，存储过程可以控制上下文区域的变化，请参考表 11-48。

表 11-48. 游标定义相关 SQL

功能	相关 SQL
创建游标	11.16.58CURSOR
移动游标	11.16.91MOVE
从游标中提取数据	11.16.87FETCH
关闭游标	11.16.30CLOSE

11.14. DML 语法一览表

DML（Data Manipulation Language 数据操作语言），用于对数据库表中的数据进行操作。如：插入、更新、查询、删除。

插入数据

插入数据是往数据库表中添加一条或多条记录，请参考 11.16.89INSERT。

修改数据

修改数据是修改数据库表中的一条或多条记录，请参考 11.16.113UPDATE。

查询数据

数据库查询语句 SELECT 是用于在数据库中检索适合条件的信息，请参考 SELECT。

删除数据

Vastbase 提供了两种删除表数据的语句：删除表中指定条件的数据，请参考 11.16.61DELETE；或删除表的所有数据，请参考 11.16.112TRUNCATE。

TRUNCATE 快速地从表中删除所有行，它和在每个表上进行无条件的 DELETE 有同样的效果，不过因为它不做表扫描，因而快得多。在大表上最有用。

拷贝数据

Vastbase 提供了在表和文件之间拷贝数据的语句，请参考 11.16.35COPY。

锁定表

Vastbase 提供了多种锁模式用于控制对表中数据的并发访问，请参考 11.16.90LOCK。

调用函数

Vastbase 提供了三个用于调用函数的语句，它们在语法结构上没有差别，请参考 11.16.28CALL。

操作会话

用户与数据库之间建立的连接称为会话，请参考表 11-49。

表 11-49. 会话相关 SQL

功能	相关 SQL
修改会话	11.16.14ALTER SESSION
结束会话	11.16.16ALTER SYSTEM KILL SESSION

11.15. DCL 语法一览表

DCL（Data Control Language 数据控制语言），是用来创建用户角色、设置或更改数据库用户或角色权限的语句。

定义角色

角色是用来管理权限的，从数据库安全的角度考虑，可以把所有的管理和操作权限划分到不同的角色上。所涉及的 SQL 语句，请参考表 11-50。

表 11-50. 角色定义相关 SQL

功能	相关 SQL
创建角色	11.16.44CREATE ROLE
修改角色属性	11.16.10ALTER ROLE
删除角色	11.16.72DROP ROLE

定义用户

用户是用来登录数据库的，通过对用户赋予不同的权限，可以方便地管理用户对数据库的访问及操作。所涉及的 SQL 语句，请参考表 11-51。

表 11-51. 用户定义相关 SQL

功能	相关 SQL
创建用户	11.16.56CREATE USER
修改用户属性	11.16.24ALTER USER
删除用户	11.16.82DROP USER

授权

Vastbase 提供了针对数据对象和角色授权的语句，请参考 11.16.88GRANT。

收回权限

Vastbase 提供了收回权限的语句，请参考 11.16.99REVOKE。

设置默认权限

Vastbase 允许设置应用于将来创建的对象的权限，请参考 11.16.4ALTER DEFAULT PRIVILEGES。

11.16. SQL 语法

11.16.1.ABORT

功能描述

回滚当前事务并且撤销所有当前事务中所做的更改。

作用等同于 11.16.100ROLLBACK，早期 SQL 有用 ABORT，现在推荐使用 ROLLBACK。

注意事项

在事务外部执行 ABORT 语句不会影响事务的执行，但是会产生一个警告信息。

语法格式

```
ABORT [ WORK | TRANSACTION ] ;
```

参数说明

- WORK | TRANSACTION

  可选关键字，除了增加可读性没有其他任何作用。

示例

```
--创建表 customer_demographics_t1。
vastbase=# CREATE TABLE customer_demographics_t1
(
   CD_DEMO_SK             INTEGER            NOT NULL,
   CD_GENDER              CHAR(1)                    ,
   CD_MARITAL_STATUS       CHAR(1)                   ,
   CD_EDUCATION_STATUS     CHAR(20)                  ,
   CD_PURCHASE_ESTIMATE    INTEGER                   ,
   CD_CREDIT_RATING        CHAR(10)                  ,
   CD_DEP_COUNT           INTEGER                   ,
   CD_DEP_EMPLOYED_COUNT    INTEGER                  ,
   CD_DEP_COLLEGE_COUNT    INTEGER
)
WITH (ORIENTATION = COLUMN,COMPRESSION=MIDDLE)
;

--插入记录。
vastbase=# INSERT INTO customer_demographics_t1 VALUES(1920801,'M', 'U', 'DOCTOR DEGREE', 200, 'GOOD',
1, 0,0);

--开启事务。
vastbase=# START TRANSACTION;

--更新字段值。
```

```
vastbase=# UPDATE customer_demographics_t1 SET cd_education_status= 'Unknown';

--终止事务，上面所执行的更新会被撤销掉。
vastbase=# ABORT;

--查询数据。
vastbase=# SELECT * FROM customer_demographics_t1 WHERE cd_demo_sk = 1920801;
cd_demo_sk | cd_gender | cd_marital_status | cd_education_status  | cd_purchase_estimate |
cd_credit_rating | cd_dep_count | cd_dep_employed_count | cd_dep_college_count
------------+-----------+-------------------+----------------------+----------------------+------
------------+--------------+-----------------------+----------------------
   1920801 | M         | U                 | DOCTOR DEGREE        |                200 | GOOD           |
1 |                  0 |                  0
(1 row)

--删除表。
vastbase=# DROP TABLE customer_demographics_t1;
```

相关链接

11.16.109SET TRANSACTION，11.16.33COMMIT | END，11.16.100ROLLBACK

11.16.2.ALTER DATABASE

功能描述

修改数据库的属性，包括它的名称、所有者、连接数限制、对象隔离属性等。

注意事项

- 只有拥有数据库所有者权限的用户才能执行 ALTER DATABASE 命令，系统管理员默认拥有此权限。如果是非系统管理员，针对所要修改属性的不同，对其还有以下权限约束：
  - 修改数据库名称，必须拥有 CREATEDB 权限。
  - 修改数据库所有者，当前用户必须是该 database 的所有者，必须拥有 CREATEDB 权限，且该用户是新所有者角色的成员。
  - 修改数据库默认表空间，该用户必须是该 database 的所有者或系统管理员，必须拥有新表空间的 CREATE 权限。这个语句会从物理上将一个数据库原来缺省表空间上的表和索引移至新的表空间。注意不在缺省表空间的表和索引不受此影响。
  - 修改某个按数据库设置的相关参数，只有数据库所有者或者系统管理员可以改变这些设置。
  - 修改某个数据库对象隔离属性，只有数据库所有者或者系统管理员可以执行此操作。
- 不能重命名当前使用的数据库，如果需要重新命名，须连接至其他数据库上。

语法格式

- 修改数据库的最大连接数。

```
ALTER DATABASE database_name
   [ [ WITH ] CONNECTION LIMIT connlimit ];
```

- 修改数据库名称。

```
ALTER DATABASE database_name
   RENAME TO new_name;
```

- 修改数据库所属者。

```
ALTER DATABASE database_name
   OWNER TO new_owner;
```

- 修改数据库默认表空间。

```
ALTER DATABASE database_name
   SET TABLESPACE new_tablespace;
```

- 修改数据库指定会话参数值。

```
ALTER DATABASE database_name
   SET configuration_parameter { { TO | = } { value | DEFAULT } | FROM CURRENT };
```

- 数据库配置参数重置。

```
ALTER DATABASE database_name RESET
   { configuration_parameter | ALL };
```

- 修改数据库对象隔离属性。

```
ALTER DATABASE database_name [ WITH ] { ENABLE | DISABLE } PRIVATE OBJECT;
```

说明

- 修改数据库的对象隔离属性时须连接至该数据库，否则无法更改。
- 新创建的数据库，对象隔离属性默认是关闭的。当开启数据库对象隔离属性后，普通用户只能查看有权访问的对象（表、函数、视图、字段等）。对象隔离特性对管理员用户不生效，当开启对象隔离特性后，管理员也可以查看到全量的数据库对象。

参数说明

- database_name

  需要修改属性的数据库名称。

  取值范围：字符串，要符合标识符的命名规范。

- connlimit

  数据库可以接收的最大并发连接数（管理员用户连接除外）。

  取值范围：整数，建议填写 1~50 的整数。-1（缺省）表示没有限制。

- new_name

  数据库的新名称。

  取值范围：字符串，要符合标识符的命名规范。

- new_owner

  数据库的新所有者。

取值范围：字符串，有效的用户名。

- new_tablespace

  数据库新的默认表空间，该表空间为数据库中已经存在的表空间。默认的表空间为 pg_default。

  取值范围：字符串，有效的表空间名。

- configuration_parameter

  value

  把指定的数据库会话参数值设置为给定的值。如果 value 是 DEFAULT 或者 RESET，则在新的会话中使用系统的缺省设置。OFF 关闭设置。

  取值范围：字符串，

  - DEFAULT
  - OFF
  - RESET

- FROM CURRENT

  根据当前会话连接的数据库设置该参数的值。

- RESET configuration_parameter

  重置指定的数据库会话参数值。

- RESET ALL

  重置全部的数据库会话参数值。

说明

- 修改数据库默认表空间，会将旧表空间中的所有表和索引转移到新表空间中，该操作不会影响其他非默认表空间中的表和索引。
- 修改的数据库会话参数值，将在下一次会话中生效。

示例

请参考 CREATE DATABASE 的示例。

相关链接

11.16.36CREATE DATABASE，11.16.63DROP DATABASE

11.16.3.ALTER DATA SOURCE

功能描述

修改 Data Source 对象的属性和内容。

属性有：名称和属主；内容有：类型、版本和连接选项。

注意选项

- 只有初始用户/系统管理员/属主才拥有修改 Data Source 的权限。
- 修改属主时，新的属主用户必须是初始用户或系统管理员。
- 当在 OPTIONS 中出现 password 选项时，需要保证 Vastbase 每个节点的$GAUSSHOME/bin 目录下存在 datasource.key.cipher 和 datasource.key.rand 文件，如果不存在这两个文件，请使用 vb_guc 工具生成并使用 gs_ssh 工具发布到每个节点的$GAUSSHOME/bin 目录下。

语法格式

```
ALTER DATA SOURCE src_name
   [TYPE  'type_str']
   [VERSION  {'version_str' | NULL}]
   [OPTIONS ( {[ ADD | SET | DROP ] optname ['optvalue']} [, ...] )];
ALTER DATA SOURCE src_name RENAME TO src_new_name;
ALTER DATA SOURCE src_name OWNER TO new_owner;
```

参数说明

- src_name

  待修改的 Data Source 的名称。

  取值范围：字符串，需要符合标识符的命名规范。

- TYPE

  将 Data Source 原来的 TYPE 修改为指定值。

  取值范围：空串或非空字符串。

- VERSION

  将 Data Source 原来的 VERSION 修改为指定值。

  取值范围：空串或非空字符串或 NULL。

- OPTIONS

  修改 OPTIONS 中的字段：增加（ADD）、修改（SET）、删除（DROP），且字段名称 optname 需唯一，具体要求如下：

  增加字段：ADD 可以省略，待增加字段不能已经存在了；

  修改字段：SET 不可省略，待修改字段必须存在；

  删除字段：DROP 不可省略，待删除字段必须存在，且不能指定 optvalue；

- src_new_name

  新的 Data Source 名称。

取值范围：字符串，需符合标识符命名规范。

- new_user

  对象的新属主。

  取值范围：字符串，有效的用户名。

示例

```
--创建一个空 Data Source 对象。
vastbase=# CREATE DATA SOURCE ds_test1;

--修改名称。
vastbase=# ALTER DATA SOURCE ds_test1 RENAME TO ds_test;

--修改属主。
vastbase=# CREATE USER user_test1 IDENTIFIED BY 'Gs@123456';
vastbase=# ALTER USER user_test1 WITH SYSADMIN;
vastbase=# ALTER DATA SOURCE ds_test OWNER TO user_test1;

--修改 TYPE 和 VERSION。
vastbase=# ALTER DATA SOURCE ds_test TYPE 'MPPDB_TYPE' VERSION 'XXX';

--添加字段。
vastbase=# ALTER DATA SOURCE ds_test OPTIONS (add dsn 'vastbase', username 'test_user');

--修改字段。
vastbase=# ALTER DATA SOURCE ds_test OPTIONS (set dsn 'unknown');

--删除字段。
vastbase=# ALTER DATA SOURCE ds_test OPTIONS (drop username);

--删除 Data Source 和 user 对象。
vastbase=# DROP DATA SOURCE ds_test;
vastbase=# DROP USER user_test1;
```

相关链接

11.16.37CREATE DATA SOURCE，11.16.64DROP DATA SOURCE

11.16.4.ALTER DEFAULT PRIVILEGES

功能描述

设置应用于将来创建的对象的权限（这不会影响分配到已有对象中的权限）。

注意事项

目前只支持表（包括视图）、 函数和类型（包括域）的权限更改。

语法格式

```
ALTER DEFAULT PRIVILEGES
   [ FOR { ROLE | USER } target_role [, ...] ]
   [ IN SCHEMA schema_name [, ...] ]
   abbreviated_grant_or_revoke;
```

- 其中 abbreviated_grant_or_revoke 子句用于指定对哪些对象进行授权或回收权限。

```
grant_on_tables_clause
  | grant_on_functions_clause
  | grant_on_types_clause
  | revoke_on_tables_clause
  | revoke_on_functions_clause
  | revoke_on_types_clause
```

- 其中 grant_on_tables_clause 子句用于对表授权。

```
GRANT { { SELECT | INSERT | UPDATE | DELETE | TRUNCATE | REFERENCES }
   [, ...] | ALL [ PRIVILEGES ] }
   ON TABLES
   TO { [ GROUP ] role_name | PUBLIC } [, ...]
   [ WITH GRANT OPTION ]
```

- 其中 grant_on_functions_clause 子句用于对函数授权。

```
GRANT { EXECUTE | ALL [ PRIVILEGES ] }
   ON FUNCTIONS
   TO { [ GROUP ] role_name | PUBLIC } [, ...]
   [ WITH GRANT OPTION ]
```

- 其中 grant_on_types_clause 子句用于对类型授权。

```
GRANT { USAGE | ALL [ PRIVILEGES ] }
   ON TYPES
   TO { [ GROUP ] role_name | PUBLIC } [, ...]
   [ WITH GRANT OPTION ]
```

- 其中 revoke_on_tables_clause 子句用于回收表对象的权限。

```
REVOKE [ GRANT OPTION FOR ]
   { { SELECT | INSERT | UPDATE | DELETE | TRUNCATE | REFERENCES }
   [, ...] | ALL [ PRIVILEGES ] }
   ON TABLES
   FROM { [ GROUP ] role_name | PUBLIC } [, ...]
   [ CASCADE | RESTRICT | CASCADE CONSTRAINTS ]
```

- 其中 revoke_on_functions_clause 子句用于回收函数的权限。

```
REVOKE [ GRANT OPTION FOR ]
   { EXECUTE | ALL [ PRIVILEGES ] }
   ON FUNCTIONS
   FROM { [ GROUP ] role_name | PUBLIC } [, ...]
   [ CASCADE | RESTRICT | CASCADE CONSTRAINTS ]
```

- 其中 revoke_on_types_clause 子句用于回收类型的权限。

```
REVOKE [ GRANT OPTION FOR ]
   { USAGE | ALL [ PRIVILEGES ] }
   ON TYPES
   FROM { [ GROUP ] role_name | PUBLIC } [, ...]
   [ CASCADE | RESTRICT | CASCADE CONSTRAINTS ]
```

参数说明

- target_role

  已有角色的名称。如果省略 FOR ROLE/USER，则缺省值为当前角色/用户。

  取值范围：已有角色的名称。

- schema_name

  现有模式的名称。

  target_role 必须有 schema_name 的 CREATE 权限。

  取值范围：现有模式的名称。

- role_name

  被授予或者取消权限角色的名称。

  取值范围：已存在的角色名称。

须知

如果想删除一个被赋予了默认权限的角色，有必要恢复改变的缺省权限或者使用 DROP OWNED BY 来为角色脱离缺省的权限记录。

示例

```
--将创建在模式 tpcds 里的所有表（和视图）的 SELECT 权限授予每一个用户。
vastbase=# ALTER DEFAULT PRIVILEGES IN SCHEMA tpcds GRANT SELECT ON TABLES TO PUBLIC;

--创建用户普通用户 jack。
vastbase=# CREATE USER jack PASSWORD 'Bigdata@123';

--将 tpcds 下的所有表的插入权限授予用户 jack。
vastbase=# ALTER DEFAULT PRIVILEGES IN SCHEMA tpcds GRANT INSERT ON TABLES TO jack;

--撤销上述权限。
vastbase=# ALTER DEFAULT PRIVILEGES IN SCHEMA tpcds REVOKE SELECT ON TABLES FROM PUBLIC;
vastbase=# ALTER DEFAULT PRIVILEGES IN SCHEMA tpcds REVOKE INSERT ON TABLES FROM jack;

--删除用户 jack。
vastbase=# DROP USER jack;
```

相关链接

11.16.88GRANT，11.16.99REVOKE

11.16.5. ALTER DIRECTORY

功能描述

对 directory 属性进行修改。

注意事项

- 目前只支持修改 directory 属主。
- 属主仅允许是 sysadmin 权限用户，不允许赋予普通用户。

语法格式

```
ALTER DIRECTORY directory_name
   OWNER TO new_owner;
```

参数描述

directory_name

需要修改的目录名称，范围为已经存在的目录名称。

示例

```
--创建目录。
vastbase=# CREATE OR REPLACE DIRECTORY  dir  as '/tmp/';

--修改目录的 owner。
vastbase=# ALTER DIRECTORY dir OWNER TO system;

--删除外部表。
vastbase=# DROP DIRECTORY dir;
```

相关链接

11.16.38CREATE DIRECTORY，11.16.65DROP DIRECTORY

11.16.6. ALTER FUNCTION

功能描述

修改自定义函数的属性。

注意事项

只有该函数的所有者，才有权限执行该命令，系统管理员默认拥有该权限。如果函数中涉及对临时表相关的操作，则无法使用 ALTER FUNCTION。

语法格式

- 修改自定义函数的附加参数。

```
ALTER FUNCTION function_name ( [ { [ argmode ] [ argname ] argtype} [, ...] ] )
    action [ ... ] [ RESTRICT ];
```

其中附加参数 action 子句语法为。

```
{CALLED ON NULL INPUT | RETURNS NULL ON NULL INPUT | STRICT}
 | {IMMUTABLE | STABLE | VOLATILE}
 | {SHIPPABLE | NOT SHIPPABLE}
 | {NOT FENCED | FENCED}
 | [ NOT ] LEAKPROOF
 | { [ EXTERNAL ] SECURITY INVOKER | [ EXTERNAL ] SECURITY DEFINER }
 | AUTHID { DEFINER | CURRENT USER }
 | COST execution cost
 | ROWS result rows
 | SET configuration parameter { { TO | = } { value | DEFAULT }| FROM CURRENT}
 | RESET {configuration_parameter | ALL}
```

- 修改自定义函数的名称。

```
ALTER FUNCTION funname ( [ { [ argmode ] [ argname ] argtype} [, ...] ] )
    RENAME TO new_name;
```

- 修改自定义函数的所属者。

```
ALTER FUNCTION funname ( [ { [ argmode ] [ argname ] argtype} [, ...] ] )
    OWNER TO new_owner;
```

- 修改自定义函数的模式。

```
ALTER FUNCTION funname ( [ { [ argmode ] [ argname ] argtype} [, ...] ] )
    SET SCHEMA new_schema;
```

参数说明

- function_name

  要修改的函数名称。

  取值范围：已存在的函数名。

- argmode

  标识该参数是输入、输出参数。

  取值范围：IN/OUT/IN OUT

- argname

  参数名称。

  取值范围：字符串，符合标识符命名规范。

- argtype

  参数类型。

  取值范围：有效的类型，请参考 11.3 数据类型。

- CALLED ON NULL INPUT

  表明该函数的某些参数是 NULL 的时候可以按照正常的方式调用。缺省时与指定此参数的作用相同。

- RETURNS NULL ON NULL INPUT

  STRICT

  STRICT 用于指定如果函数的某个参数是 NULL，此函数总是返回 NULL。如果声明了这个参数，则如果存在 NULL 参数时不会执行该函数；而只是自动假设一个 NULL 结果。

  RETURNS NULL ON NULL INPUT 和 STRICT 的功能相同。

- IMMUTABLE

  表示该函数在给出同样的参数值时总是返回同样的结果。

- STABLE

  表示该函数不能修改数据库，对相同参数值，在同一次表扫描里，该函数的返回值不变，但是返回值可能在不同 SQL 语句之间变化。

- VOLATILE

  表示该函数值可以在一次表扫描内改变，不会做任何优化。

- LEAKPROOF

  表示该函数没有副作用，指出参数只包括返回值。LEAKROOF 只能由系统管理员设置。

- EXTERNAL

  （可选）目的是和 SQL 兼容，这个特性适合于所有函数，而不仅是外部函数

- SECURITY INVOKER

  AUTHID CURREN_USER

  表明该函数将以调用它的用户的权限执行。缺省时与指定此参数的作用相同。

  SECURITY INVOKER 和 AUTHID CURREN_USER 的功能相同。

- SECURITY DEFINER

  AUTHID DEFINER

  声明该函数将以创建它的用户的权限执行。

  AUTHID DEFINER 和 SECURITY DEFINER 的功能相同。

- COST execution_cost

  用来估计函数的执行成本。

  execution_cost 以 cpu_operator_cost 为单位。

  取值范围：正数

- ROWS result_rows

  估计函数返回的行数。用于函数返回的是一个集合。

  取值范围：正数，默认值是 1000 行。

- configuration_parameter
  - value

    把指定的数据库会话参数值设置为给定的值。如果 value 是 DEFAULT 或者 RESET，则在新的会话中使用系统的缺省设置。OFF 关闭设置。

    取值范围：字符串

    - DEFAULT
    - OFF
    - RESET

    指定默认值。

  - from current

    取当前会话中的值设置为 configuration_parameter 的值。

- new_name

  函数的新名称。要修改函数的所属模式，必须拥有新模式的 CREATE 权限。

  取值范围：字符串，符合标识符命名规范。

- new_owner

  函数的新所有者。要修改函数的所有者，新所有者必须拥有该函数所属模式的 CREATE 权限。

  取值范围：已存在的用户角色。

- new_schema

  函数的新模式。

  取值范围：已存在的模式。

示例

请参见 CREATE FUNCTION 的示例。

相关链接

11.16.39CREATE FUNCTION，11.16.66DROP FUNCTION

11.16.7.ALTER GROUP

功能描述

修改一个用户组的属性。

注意事项

ALTER GROUP 是 ALTER ROLE 的别名，非 SQL 标准语法，不推荐使用，建议用户直接使用 ALTER ROLE 替代。

语法格式

- 向用户组中添加用户。

```
ALTER GROUP group_name
   ADD USER user_name [, ... ];
```

- 从用户组中删除用户。

```
ALTER GROUP group_name
   DROP USER user_name [, ... ];
```

- 修改用户组的名称。

```
ALTER GROUP group_name
   RENAME TO new_name;
```

参数说明

请参考 ALTER ROLE 的参数说明。

示例

```
--向用户组中添加用户。
vastbase=# ALTER GROUP super_users ADD USER lche, jim;

--从用户组中删除用户。
vastbase=# ALTER GROUP super_users DROP USER jim;

--修改用户组的名称。
vastbase=# ALTER GROUP super_users RENAME TO normal_users;
```

相关链接

11.16.7ALTER GROUP，11.16.67DROP GROUP，11.16.10ALTER ROLE

11.16.8. ALTER INDEX

功能描述

ALTER INDEX 用于修改现有索引的定义。

它有几种子形式：

- IF EXISTS

  如果指定的索引不存在，则发出一个 notice 而不是 error。

- RENAME TO

  只改变索引的名称。对存储的数据没有影响。

- SET TABLESPACE

  这个选项会改变索引的表空间为指定表空间，并且把索引相关的数据文件移动到新的表空间里。

- SET ({ STORAGE_PARAMETER = value } [, ...])

  改变索引的一个或多个索引方法特定的存储参数。 需要注意的是索引内容不会被这个命令立即修改，根据参数的不同，可能需要使用 REINDEX 重建索引来获得期望的效果。

- RESET ({ storage_parameter } [, ...])

  重置索引的一个或多个索引方法特定的存储参数为缺省值。与 SET 一样，可能需要使用 REINDEX 来完全更新索引。

- [MODIFY PARTITION index_partition_name] UNUSABLE

  用于设置表或者索引分区上的索引不可用。

- REBUILD [PARTITION index_partition_name]

  用于重建表或者索引分区上的索引。

- RENAME PARTITION

  用于重命名索引分区

- MOVE PARTITION

  用于修改索引分区的所属表空间。

注意事项

只有索引的所有者有权限执行此命令，系统管理员默认拥有此权限。

语法格式

- 重命名表索引的名称。

```
ALTER INDEX [ IF EXISTS ] index_name
   RENAME TO new_name;
```

- 修改表索引的所属空间。

```
ALTER INDEX [ IF EXISTS ] index_name
    SET TABLESPACE tablespace_name;
```

- 修改表索引的存储参数。

```
ALTER INDEX [ IF EXISTS ] index_name
    SET ( {storage_parameter = value} [, ... ] );
```

- 重置表索引的存储参数。

```
ALTER INDEX [ IF EXISTS ] index_name
    RESET ( storage_parameter [, ... ] ) ;
```

- 设置表索引或索引分区不可用。

```
ALTER INDEX [ IF EXISTS ] index_name
    [ MODIFY PARTITION index_partition_name ] UNUSABLE;
```

说明

列存表不支持该语法。

- 重建表索引或索引分区。

```
ALTER INDEX index_name
    REBUILD [ PARTITION index_partition_name ];
```

- 重命名索引分区。

```
ALTER INDEX [ IF EXISTS ] index_name
    RENAME PARTITION index_partition_name TO new_index_partition_name;
```

- 修改索引分区的所属表空间。

```
ALTER INDEX [ IF EXISTS ] index_name
    MOVE PARTITION index_partition_name TABLESPACE new_tablespace;
```

参数说明

- index_name

  要修改的索引名。

- new_name

  新的索引名。

  取值范围：字符串，且符合标识符命名规范。

- tablespace_name

  表空间的名称。

  取值范围：已存在的表空间。

- storage_parameter

  索引方法特定的参数名。

- value

  索引方法特定的存储参数的新值。根据参数的不同，这可能是一个数字或单词。

- new_index_partition_name

新索引分区名。

- index_partition_name

  索引分区名。

- new_tablespace

  新表空间。

示例

请参见 CREATE INDEX 的示例。

相关链接

11.16.41CREATE INDEX，11.16.68DROP INDEX，11.16.96REINDEX

11.16.9.ALTER LARGE OBJECT

功能描述

ALTER LARGE OBJECT 用于更改一个 large object 的定义。它的唯一的功能是分配一个新的所有者。

注意事项

使用 ALTER LARGE OBJECT 必须是系统管理员或者是其所有者。

语法格式

```
ALTER LARGE OBJECT large_object_oid
   OWNER TO new_owner;
```

参数说明

- large_object_oid

  要被变 large object 的 OID 。

  取值范围：已存在的大对象名。

- OWNER TO new_owner

  large object 新的所有者。

  取值范围：已存在的用户名/角色名。

示例

无。

11.16.10. ALTER ROLE

功能描述

修改角色属性。

注意事项

无。

语法格式

- 修改角色的权限。

```
ALTER ROLE role_name [ [ WITH ] option [ ... ] ];
```

其中权限项子句 option 为。

```
{CREATEDB | NOCREATEDB}
   | {CREATEROLE | NOCREATEROLE}
   | {INHERIT | NOINHERIT}
   | {AUDITADMIN | NOAUDITADMIN}
   | {SYSADMIN | NOSYSADMIN}
   | {MONADMIN | NOMONADMIN}
   | {OPRADMIN | NOOPRADMIN}
   | {POLADMIN | NOPOLADMIN}
   | {USEFT | NOUSEFT}
   | {LOGIN | NOLOGIN}
   | {REPLICATION | NOREPLICATION}
   | {INDEPENDENT | NOINDEPENDENT}
   | {VCADMIN | NOVCADMIN}
   | CONNECTION LIMIT connlimit
   | [ ENCRYPTED | UNENCRYPTED ] PASSWORD 'password'
   | [ ENCRYPTED | UNENCRYPTED ] IDENTIFIED BY 'password' [ REPLACE 'old_password' ]
   | [ ENCRYPTED | UNENCRYPTED ] PASSWORD { 'password' | DISABLE }
   | [ ENCRYPTED | UNENCRYPTED ] IDENTIFIED BY { 'password' [ REPLACE 'old_password' ] | DISABLE }
   | VALID BEGIN 'timestamp'
   | VALID UNTIL 'timestamp'
   | RESOURCE POOL 'respool'
   | PERM SPACE 'spacelimit'
   | ACCOUNT { LOCK | UNLOCK }
   | PGUSER
```

- 修改角色的名称。

```
ALTER ROLE role_name
   RENAME TO new_name;
```

- 设置角色的配置参数。

```
ALTER ROLE role_name [ IN DATABASE database_name ]
   SET configuration_parameter {{ TO | = } { value | DEFAULT } | FROM CURRENT};
```

- 重置角色的配置参数。

```
ALTER ROLE role_name
   [ IN DATABASE database_name ] RESET {configuration_parameter|ALL};
```

参数说明

- role_name

  现有角色名。

  取值范围：已存在的用户名。

- IN DATABASE database_name

  表示修改角色在指定数据库上的参数。

- SET configuration_parameter

  设置角色的参数。ALTER ROLE 中修改的会话参数只针对指定的角色，且在下一次该角色启动的会话中有效。

  取值范围：

  configuration_parameter 和 value 的取值请参见 11.16.105SET。

  DEFAULT：表示清除 configuration_parameter 参数的值，configuration_parameter 参数的值将继承本角色新产生的 SESSION 的默认值。

  FROM CURRENT：取当前会话中的值设置为 configuration_parameter 参数的值。

- RESET configuration_parameter/ALL

  清除 configuration_parameter 参数的值。与 SET configuration_parameter TO DEFAULT 的效果相同。

  取值范围：ALL 表示清除所有参数的值。

- ACCOUNT LOCK | ACCOUNT UNLOCK
  - ACCOUNT LOCK：锁定帐户，禁止登录数据库。
  - ACCOUNT UNLOCK：解锁帐户，允许登录数据库。
- PGUSER

  当前版本不允许修改角色的 PGUSER 属性

其他参数请参见 CREATE ROLE 的参数说明。

示例

请参见 CREATE ROLE 的示例。

相关链接

11.16.44CREATE ROLE，11.16.72DROP ROLE，11.16.105SET

11.16.11. ALTER ROW LEVEL SECURITY POLICY

功能描述

对已存在的行访问控制策略（包括行访问控制策略的名称，行访问控制指定的用户，行访问控制的策略表达式）进行修改。

注意事项

表的所有者或管理员用户才能进行此操作。

语法格式

```
ALTER [ ROW LEVEL SECURITY ] POLICY [ IF EXISTS ] policy_name ON table_name RENAME TO new_policy_name;

ALTER [ ROW LEVEL SECURITY ] POLICY policy_name ON table_name
   [ TO { role_name | PUBLIC } [, ...] ]
   [ USING ( using_expression ) ];
```

参数说明

- policy_name

  行访问控制策略名称。

- table_name

  行访问控制策略的表名。

- new_policy_name

  新的行访问控制策略名称。

- role_name

  行访问控制策略应用的数据库用户，可以指定多个用户，PUBLIC 表示应用到所有用户。

- using_expression

  行访问控制的表达式，返回值为 boolean 类型。

示例

```
--创建数据表 all_data
vastbase=# CREATE TABLE all_data(id int, role varchar(100), data varchar(100));

--创建行访问控制策略，当前用户只能查看用户自身的数据
vastbase=# CREATE ROW LEVEL SECURITY POLICY all_data_rls ON all_data USING(role = CURRENT_USER);
vastbase=# \d+ all_data
                        Table "public.all_data"
 Column |         Type          | Modifiers | Storage  | Stats target | Description
--------+-----------------------+-----------+----------+--------------+-------------
```

```
 id     | integer              |          | plain    |            |
 role   | character varying(100) |          | extended |            |
 data   | character varying(100) |          | extended |            |
Row Level Security Policies:
    POLICY "all_data_rls"
      USING (((role)::name = "current_user"()))
Has OIDs: no
Options: orientation=row, compression=no

--修改行访问控制 all_data_rls 的名称
vastbase=# ALTER ROW LEVEL SECURITY POLICY all_data_rls ON all_data RENAME TO all_data_new_rls;

--修改行访问控制策略影响的用户
vastbase=# ALTER ROW LEVEL SECURITY POLICY all_data_new_rls ON all_data TO alice, bob;
vastbase=# \d+ all_data
                        Table "public.all_data"
 Column |         Type         | Modifiers | Storage  | Stats target | Description
--------+----------------------+-----------+----------+--------------+-------------
 id     | integer              |          | plain    |            |
 role   | character varying(100) |          | extended |            |
 data   | character varying(100) |          | extended |            |
Row Level Security Policies:
    POLICY "all_data_new_rls"
      TO alice,bob
      USING (((role)::name = "current_user"()))
Has OIDs: no
Options: orientation=row, compression=no, enable_rowsecurity=true

--修改行访问控制策略表达式
vastbase=# ALTER ROW LEVEL SECURITY POLICY all_data_new_rls ON all_data USING (id > 100 AND role =
current_user);
vastbase=# \d+ all_data
                        Table "public.all_data"
 Column |         Type         | Modifiers | Storage  | Stats target | Description
--------+----------------------+-----------+----------+--------------+-------------
 id     | integer              |          | plain    |            |
 role   | character varying(100) |          | extended |            |
 data   | character varying(100) |          | extended |            |
Row Level Security Policies:
    POLICY "all data new rls"
      TO alice,bob
      USING (((id > 100) AND ((role)::name = "current_user"())))
Has OIDs: no
Options: orientation=row, compression=no, enable_rowsecurity=true
```

相关链接

11.16.42CREATE ROW LEVLEL SECURITY POLICY，11.16.70DROP ROW LEVEL SECURITY POLICY

11.16.12. ALTER SCHEMA

功能描述

修改模式属性。

注意事项

只有模式的所有者有权限执行 ALTER SCHEMA 命令，系统管理员默认拥有此权限。

语法格式

- 修改模式的名称。

```
ALTER SCHEMA schema_name
   RENAME TO new_name;
```

- 修改模式的所有者。

```
ALTER SCHEMA schema_name
   OWNER TO new_owner;
```

参数说明

- schema_name

  现有模式的名称。

  取值范围：已存在的模式名。

- RENAME TO new_name

  修改模式的名称。

  new_name：模式的新名称。

  取值范围：字符串，要符合标识符命名规范。

- OWNER TO new_owner

  修改模式的所有者。非系统管理员要改变模式的所有者，该用户还必须是新的所有角色的直接或间接成员， 并且该成员必须在此数据库上有 CREATE 权限。

  new_owner：模式的新所有者。

  取值范围：已存在的用户名/角色名。

示例

```
--创建模式 ds。
vastbase=# CREATE SCHEMA ds;

--将当前模式 ds 更名为 ds_new。
vastbase=# ALTER SCHEMA ds RENAME TO ds_new;
```

```
--创建用户 jack。
vastbase=# CREATE USER jack PASSWORD 'Bigdata@123';

--将 DS_NEW 的所有者修改为 jack。
vastbase=# ALTER SCHEMA ds_new OWNER TO jack;

--删除用户 jack 和模式 ds_new。
vastbase=# DROP SCHEMA ds_new;
vastbase=# DROP USER jack;
```

相关链接

11.16.45CREATE SCHEMA，11.16.73DROP SCHEMA

11.16.13. ALTER SEQUENCE

功能描述

修改一个现有的序列的参数。

注意事项

- 使用 ALTER SEQUENCE 的用户必须是该序列的所有者。
- 当前版本仅支持修改拥有者、归属列和最大值。若要修改其他参数，可以删除重建，并用 Setval 函数恢复当前值。
- ALTER SEQUENCE MAXVALUE 不支持在事务、函数和存储过程中使用。
- 修改序列的最大值后，会清空该序列在所有会话的 cache。
- ALTER SEQUENCE 会阻塞 nextval、setval、currval 和 lastval 的调用。

语法格式

修改序列归属列

```
ALTER SEQUENCE [ IF EXISTS ] name
    [MAXVALUE maxvalue | NO MAXVALUE | NOMAXVALUE]
    [ OWNED BY { table_name.column_name | NONE } ] ;
```

修改序列的拥有者

```
ALTER SEQUENCE [ IF EXISTS ] name OWNER TO new_owner;
```

参数说明

- name

  将要修改的序列名称。

- IF EXISTS

当序列不存在时使用该选项不会出现错误消息，仅有一个通知。

- OWNED BY

  将序列和一个表的指定字段进行关联。这样，在删除那个字段或其所在表的时候会自动删除已关联的序列。

  如果序列已经和表有关联后，使用这个选项后新的关联关系会覆盖旧的关联。

  关联的表和序列的所有者必须是同一个用户，并且在同一个模式中。

  使用 OWNED BY NONE 将删除任何已经存在的关联。

- new_owner

  序列新所有者的用户名。用户要修改序列的所有者，必须是新角色的直接或者间接成员，并且那个角色必须有序列所在模式上的 CREATE 权限。

示例

```
--创建一个名为 serial 的递增序列，从 101 开始。
vastbase=# CREATE SEQUENCE serial START 101;

--创建一个表,定义默认值。
vastbase=# CREATE TABLE T1(C1 bigint default nextval('serial'));

--将序列 serial 的归属列变为 T1.C1。
vastbase=# ALTER SEQUENCE serial OWNED BY T1.C1;

--删除序列
vastbase=# DROP SEQUENCE serial cascade;
vastbase=# DROP TABLE T1;
```

相关链接

11.16.46CREATE SEQUENCE，11.16.74DROP SEQUENCE

11.16.14. ALTER SESSION

功能描述

ALTER SESSION 命令用于定义或修改那些对当前会话有影响的条件或参数。修改后的会话参数会一直保持，直到断开当前会话。

注意事项

- 如果执行 SET TRANSACTION 之前没有执行 START TRANSACTION，则事务立即结束，命令无法显示效果。

❖ 可以用 START TRANSACTION 里面声明所需要的 transaction_mode(s)的方法来避免使用 SET TRANSACTION。

语法格式

❖ 设置会话的事务参数。

```
ALTER SESSION SET [ SESSION CHARACTERISTICS AS ] TRANSACTION
   { ISOLATION LEVEL { READ COMMITTED } | { READ ONLY  | READ WRITE } } [, ...] ;
```

❖ 设置会话的其他运行时参数。

```
ALTER SESSION SET
   {{config_parameter { { TO  | =  }  { value | DEFAULT }
     | FROM CURRENT }}
     | TIME ZONE time_zone
     | CURRENT_SCHEMA schema
     | NAMES encoding_name
     | ROLE role_name PASSWORD 'password'
     | SESSION AUTHORIZATION { role_name PASSWORD 'password' | DEFAULT }
     | XML OPTION { DOCUMENT | CONTENT }
   } ;
```

参数说明

修改会话涉及到的参数说明请参见 SET 语法中的参数说明。

示例

```
-- 创建模式 ds。
vastbase=# CREATE SCHEMA ds;

--设置模式搜索路径。
vastbase=# SET SEARCH_PATH TO ds, public;

--设置日期时间风格为传统的 POSTGRES 风格（日在月前）。
vastbase=# SET DATESTYLE TO postgres, dmy;

--设置当前会话的字符编码为 UTF8。
vastbase=# ALTER SESSION SET NAMES 'UTF8';

--设置时区为加州伯克利。
vastbase=# SET TIME ZONE 'PST8PDT';

--设置时区为意大利。
vastbase=# SET TIME ZONE 'Europe/Rome';

--设置当前模式。
vastbase=# ALTER SESSION SET CURRENT_SCHEMA TO tpcds;

--设置 XML OPTION 为 DOCUMENT。
vastbase=# ALTER SESSION SET XML OPTION DOCUMENT;

--创建角色 joe，并设置会话的角色为 joe。
```

```
vastbase=# CREATE ROLE joe WITH PASSWORD 'Bigdata@123';
vastbase=# ALTER SESSION SET SESSION AUTHORIZATION joe PASSWORD 'Bigdata@123';

--切换到默认用户。
vastbase=> ALTER SESSION SET SESSION AUTHORIZATION default;

--删除 ds 模式。
vastbase=# DROP SCHEMA ds;

--删除 joe。
vastbase=# DROP ROLE joe;
```

相关链接

11.16.105SET

11.16.15. ALTER SYNONYM

功能描述

修改 SYNONYM 对象的属性。

注意事项

- 目前仅支持修改 SYNONYM 对象的属主。
- 只有系统管理员有权限修改 SYNONYM 对象的属主信息。
- 新属主必须具有 SYNONYM 对象所在模式的 CREATE 权限。

语法格式

```
ALTER SYNONYM synonym_name
   OWNER TO new_owner;
```

参数描述

- synonym

  待修改的同义词名字，可以带模式名。

  取值范围：字符串，需要符合标识符的命名规范。

- new_owner

  同义词对象的新所有者。

  取值范围：字符串，有效的用户名。

示例

```
--创建同义词 t1。
vastbase=#  CREATE OR REPLACE SYNONYM t1 FOR ot.t1;

--创建新用户 u1。
vastbase=# CREATE USER u1 PASSWORD 'user@111';

--修改同义词 t1 的 owner 为 u1。
vastbase=# ALTER SYNONYM t1 OWNER TO u1;

--删除同义词 t1。
vastbase=# DROP SYNONYM t1;

--删除用户 u1。
vastbase=# DROP USER u1;
```

相关链接

11.16.47CREATE SYNONYM，11.16.75DROP SYNONYM

11.16.16. ALTER SYSTEM KILL SESSION

功能描述

ALTER SYSTEM KILL SESSION 命令用于结束一个会话。

注意事项

无。

语法格式

```
ALTER SYSTEM KILL SESSION 'session_sid, serial' [ IMMEDIATE ];
```

参数说明

- session_sid, serial

  会话的 SID 和 SERIAL（获取方法请参考示例）。

- IMMEDIATE

  表明会话将在命令执行后立即结束。

示例

```
--查询会话信息。
vastbase=#
SELECT sa.sessionid AS sid,0::integer AS serial#,ad.rolname AS username FROM pg_stat_get_activity(NULL)
```

```
AS sa
LEFT JOIN pg_authid ad ON(sa.usesysid = ad.oid)WHERE sa.application_name <> 'JobScheduler';
      sid      | serial# | username
-----------------+---------+----------
 140131075880720 |       0 | vastbase
 140131025549072 |       0 | vastbase
 140131073779472 |       0 | vastbase
 140131071678224 |       0 | vastbase
 140131125774096 |       0 |
 140131127875344 |       0 |
 140131113629456 |       0 |
 140131094742800 |       0 |
(8 rows)

--结束 SID 为 140131075880720 的会话。
vastbase=#  ALTER SYSTEM KILL SESSION '140131075880720,0' IMMEDIATE;
```

11.16.17. ALTER TABLE

功能描述

修改表，包括修改表的定义、重命名表、重命名表中指定的列、重命名表的约束、设置表的所属模式、添加/更新多个列、打开/关闭行访问控制开关。

注意事项

- 只有表的所有者有权限执行 ALTER TABLE 命令，系统管理员默认拥有此权限。
- 不能修改分区表的 tablespace，但可以修改分区的 tablespace。
- 不支持修改存储参数 ORIENTATION。
- SET SCHEMA 操作不支持修改为系统内部模式，当前仅支持用户模式之间的修改。
- 列存表只支持 PARTIAL CLUSTER KEY 表级约束，不支持主外键等表级约束。
- 列存表只支持添加字段 ADD COLUMN、修改字段的数据类型 ALTER TYPE、设置单个字段的收集目标 SET STATISTICS、支持更改表名称、支持更改表空间，支持删除字段 DROP COLUMN。对于添加的字段和修改的字段类型要求是列存支持的 11.3 数据类型。ALTER TYPE 的 USING 选项只支持常量表达式和涉及本字段的表达式，暂不支持涉及其他字段的表达式。
- 列存表支持的字段约束包括 NULL、NOT NULL 和 DEFAULT 常量值；对字段约束的修改当前只支持对 DEFAULT 值的修改（SET DEFAULT）和删除（DROP DEFAULT），暂不支持对非空约束 NULL/NOT NULL 的修改。
- 不支持增加自增列，或者增加 DEFAULT 值中包含 nextval()表达式的列。
- 不支持对外表、临时表开启行访问控制开关。
- 通过约束名删除 PRIMARY KEY 约束时，不会删除 NOT NULL 约束，如果有需要，请手动删除 NOT NULL 约束。
- 使用 JDBC 时，支持通过 PrepareStatement 对 DEFAUTL 值进行参数化设置。

语法格式

❖ 修改表的定义。

```
ALTER TABLE [ IF EXISTS ] { table_name [*] | ONLY table_name | ONLY ( table_name ) }
    action [, ... ];
```

其中具体表操作 action 可以是以下子句之一:

```
column_clause
    | ADD table_constraint [ NOT VALID ]
    | ADD table constraint using index
    | VALIDATE CONSTRAINT constraint name
    | DROP CONSTRAINT [ IF EXISTS ]  constraint name [ RESTRICT | CASCADE ]
    | CLUSTER ON index name
    | SET WITHOUT CLUSTER
    | SET ( {storage parameter = value} [, ... ] )
    | RESET ( storage parameter [, ... ] )
    | OWNER TO new owner
    | SET TABLESPACE new tablespace
    | SET {COMPRESS|NOCOMPRESS}

    | TO { GROUP groupname | NODE ( nodename [, ... ] ) }
    | ADD NODE ( nodename [, ... ] )
    | DELETE NODE ( nodename [, ... ] )
    | DISABLE TRIGGER [ trigger_name | ALL | USER ]
    | ENABLE TRIGGER [ trigger_name | ALL | USER ]
    | ENABLE REPLICA TRIGGER trigger_name
    | ENABLE ALWAYS TRIGGER trigger_name
    | DISABLE ROW LEVEL SECURITY
    | ENABLE ROW LEVEL SECURITY
    | FORCE ROW LEVEL SECURITY
    | NO FORCE ROW LEVEL SECURITY
```

说明

- ADD table_constraint [NOT VALID]

  给表增加一个新的约束。

- ADD table_constraint_using_index

  根据已有唯一索引为表增加主键约束或唯一约束。

- VALIDATE CONSTRAINT constraint_name

  验证一个使用 NOT VALID 选项创建的检查类约束，通过扫描全表来保证所有记录都符合约束条件。如果约束已标记为有效时，什么操作也不会发生。

- DROP CONSTRAINT [IF EXISTS] constraint_name [RESTRICT | CASCADE]

  删除一个表上的约束。

- CLUSTER ON index_name

  为将来的 CLUSTER（聚簇）操作选择默认索引。实际上并没有重新盘簇化处理该表。

- SET WITHOUT CLUSTER

  从表中删除最新使用的 CLUSTER 索引。这样会影响将来那些没有声明索引的 CLUSTER（聚簇）操作。

- SET ({storage_parameter = value} [, ...])

  修改表的一个或多个存储参数。

- RESET (storage_parameter [, ...])

  重置表的一个或多个存储参数。与 SET 一样，根据参数的不同可能需要重写表才能获得想要的效果。

- OWNER TO new_owner

  将表、序列、视图的属主改变成指定的用户。

- SET TABLESPACE new_tablespace

  这种形式将表空间修改为指定的表空间并将相关的数据文件移动到新的表空间。但是表上的所有索引都不会被移动，索引可以通过 ALTER INDEX 语法的 SET TABLESPACE 选项来修改索引的表空间。

- SET {COMPRESS|NOCOMPRESS}

  修改表的压缩特性。表压缩特性的改变只会影响后续批量插入的数据的存储方式，对已有数据的存储毫无影响。也就是说，表压缩特性的修改会导致该表中同时存在着已压缩和未压缩的数据。

- TO { GROUP groupname | NODE (nodename [, ...]) }

  此语法仅在扩展模式（GUC 参数 support_extended_features 为 on 时）下可用。该模式谨慎打开，主要供内部扩容工具使用，一般用户不应使用该模式。

- ADD NODE (nodename [, ...])

  此语法主要供内部扩容工具使用，一般用户不建议使用。

- DELETE NODE (nodename [, ...])

  此语法主要供内部缩容工具使用，一般用户不建议使用。

- DISABLE TRIGGER [trigger_name | ALL | USER]

  禁用 trigger_name 所表示的单个触发器，或禁用所有触发器，或仅禁用用户触发器（此选项不包括内部生成的约束触发器，例如，可延迟唯一性和排除约束的约束触发器）。

说明

应谨慎使用此功能，因为如果不执行触发器，则无法保证原先期望的约束的完整性。

- | ENABLE TRIGGER [trigger_name | ALL | USER]

  启用 trigger_name 所表示的单个触发器，或启用所有触发器，或仅启用用户触发器。

- | ENABLE REPLICA TRIGGER trigger_name

  触发器触发机制受配置变量 session_replication_role 的影响，当复制角色为“origin”（默认值）或“local”时，将触发简单启用的触发器。

  配置为 ENABLE REPLICA 的触发器仅在会话处于“replica”模式时触发。

- | ENABLE ALWAYS TRIGGER trigger_name

  无论当前复制模式如何，配置为 ENABLE ALWAYS 的触发器都将触发。

- | DISABLE/ENABLE ROW LEVEL SECURITY

  开启或关闭表的行访问控制开关。

当开启行访问控制开关时，如果未在该数据表定义相关行访问控制策略，数据表的行级访问将不受影响；如果关闭表的行访问控制开关，即使定义了行访问控制策略，数据表的行访问也不受影响。详细信息参见 11.16.42CREATE ROW LEVLEL SECURITY POLICY 章节。

- | NO FORCE/FORCE ROW LEVEL SECURITY

  强制开启或关闭表的行访问控制开关。

  默认情况，表所有者不受行访问控制特性影响，但当强制开启表的行访问控制开关时，表的所有者（不包含系统管理员用户）会受影响。系统管理员可以绕过所有的行访问控制策略，不受影响。

  - 其中列相关的操作 column_clause 可以是以下子句之一：

```
ADD [ COLUMN ] column_name data_type [ compress_mode ] [ COLLATE collation ]
[ column_constraint [ ... ] ]
| MODIFY column_name data_type
| MODIFY column_name [ CONSTRAINT constraint_name ] NOT NULL [ ENABLE ]
| MODIFY column_name [ CONSTRAINT constraint_name ] NULL
| DROP [ COLUMN ] [ IF EXISTS ] column_name [ RESTRICT | CASCADE ]
| ALTER [ COLUMN ] column_name [ SET DATA ] TYPE data_type [ COLLATE collation ] [ USING
expression ]
| ALTER [ COLUMN ] column_name { SET DEFAULT expression | DROP DEFAULT }
| ALTER [ COLUMN ] column_name { SET | DROP } NOT NULL
| ALTER [ COLUMN ] column_name SET STATISTICS [PERCENT] integer
| ADD STATISTICS (( column_1_name, column_2_name [, ...] ))
| DELETE STATISTICS (( column_1_name, column_2_name [, ...] ))
| ALTER [ COLUMN ] column_name SET ( {attribute_option = value} [, ... ] )
| ALTER [ COLUMN ] column_name RESET ( attribute_option [, ... ] )
| ALTER [ COLUMN ] column_name SET STORAGE { PLAIN | EXTERNAL | EXTENDED | MAIN }
```

说明

- ADD [COLUMN] column_name data_type [compress_mode] [COLLATE collation] [column_constraint [...]]

  向表中增加一个新的字段。用 ADD COLUMN 增加一个字段，所有表中现有行都初始化为该字段的缺省值（如果没有声明 DEFAULT 子句，值为 NULL）。

- ADD ({ column_name data_type [compress_mode] } [, ...])

  向表中增加多列。

- MODIFY ({ column_name data_type | column_name [CONSTRAINT constraint_name] NOT NULL [ENABLE] | column_name [CONSTRAINT constraint_name] NULL } [, ...])

  修改表已存在字段的数据类型。

- DROP [COLUMN] [IF EXISTS] column_name [RESTRICT | CASCADE]

  从表中删除一个字段，和这个字段相关的索引和表约束也会被自动删除。如果任何表之外的对象依赖于这个字段，必须声明 CASCADE ，比如视图。

  DROP COLUMN 命令并不是物理上把字段删除，而只是简单地把它标记为对 SQL 操作不可见。随后对该表的插入和更新将在该字段存储一个 NULL。因此，删除一个字段是很快的，但是它不会立即释放表在磁盘上的空间，因为被删除了的字段占据的空间还没有回收。这些空间将在执行 VACUUM 时而得到回收。

- ALTER [COLUMN] column_name [SET DATA] TYPE data_type [COLLATE collation] [USING expression]

  改变表字段的数据类型。该字段涉及的索引和简单的表约束将被自动地转换为使用新的字段类型，方法是重新分析最初提供的表达式。

  ALTER TYPE 要求重写整个表的特性有时候是一个优点，因为重写的过程消除了表中没用的空间。比如，要想立刻回收被一个已经删除的字段占据的空间，最快的方法是

  ALTER TABLE table ALTER COLUMN anycol TYPE anytype;

  这里的 anycol 是任何在表中还存在的字段，而 anytype 是和该字段的原类型一样的类型。这样的结果是在表上没有任何可见的语意的变化，但是这个命令强迫重写，这样就删除了不再使用的数据。

- ALTER [COLUMN] column_name { SET DEFAULT expression | DROP DEFAULT }

  为一个字段设置或者删除缺省值。请注意缺省值只应用于随后的 INSERT 命令，它们不会修改表中已经存在的行。也可以为视图创建缺省，这个时候它们是在视图的 ON INSERT 规则应用之前插入到 INSERT 句中的。

- ALTER [COLUMN] column_name { SET | DROP } NOT NULL

  修改一个字段是否允许 NULL 值或者拒绝 NULL 值。如果表在字段中包含非 NULL，则只能使用 SET NOT NULL。

- ALTER [COLUMN] column_name SET STATISTICS [PERCENT] integer

  为随后的 ANALYZE 操作设置针对每个字段的统计收集目标。目标的范围可以在 0 到 10000 之内设置。设置为-1 时表示重新恢复到使用系统缺省的统计目标。

- {ADD | DELETE} STATISTICS ((column_1_name, column_2_name [, ...]))

  用于添加和删除多列统计信息声明（不实际进行多列统计信息收集），以便在后续进行全表或全库 analyze 时进行多列统计信息收集。每组多列统计信息最多支持 32 列。不支持添加/删除多列统计信息声明的表：系统表、外表。

- ALTER [COLUMN] column_name SET ({attribute_option = value} [, ...])

  ALTER [COLUMN] column_name RESET (attribute_option [, ...])

  设置/重置属性选项。

  目前，属性选项只定义了 n_distinct 和 n_distinct_inherited。n_distinct 影响表本身的统计值，而 n_distinct_inherited 影响表及其继承子表的统计。目前，只支持 SET/RESET n_distinct 参数，禁止 SET/RESET n_distinct_inherited 参数。

- ALTER [COLUMN] column_name SET STORAGE { PLAIN | EXTERNAL | EXTENDED | MAIN }

  为一个字段设置存储模式。这个设置控制这个字段是内联保存还是保存在一个附属的表里，以及数据是否要压缩。仅支持对行存表的设置；对列存表没有意义，执行时报错。SET STORAGE 本身并不改变表上的任何东西，只是设置将来的表操作时，建议使用的策略。

  ■ 其中列约束 column_constraint 为：

```
[ CONSTRAINT constraint_name ]
    { NOT NULL |
      NULL |
      CHECK ( expression ) |
      DEFAULT default_expr   |
```

```
    UNIQUE index_parameters |
    PRIMARY KEY index_parameters }
  [ DEFERRABLE | NOT DEFERRABLE | INITIALLY DEFERRED | INITIALLY IMMEDIATE ]
```

■ 其中列的压缩可选项 compress_mode 为：

```
[ DELTA | PREFIX | DICTIONARY | NUMSTR | NOCOMPRESS ]
```

– 其中根据已有唯一索引为表增加主键约束或唯一约束 table_constraint_using_index 为：

```
[ CONSTRAINT constraint_name ]
   { UNIQUE | PRIMARY KEY } USING INDEX index_name
   [ DEFERRABLE | NOT DEFERRABLE | INITIALLY DEFERRED | INITIALLY IMMEDIATE ]
```

– 其中表约束 table_constraint 为：

```
[ CONSTRAINT constraint_name ]
   { CHECK ( expression ) |
     UNIQUE ( column_name [, ... ] ) index_parameters |
     PRIMARY KEY ( column_name [, ... ] ) index_parameters |
     PARTIAL CLUSTER KEY ( column_name [, ... ] ) }
   [ DEFERRABLE | NOT DEFERRABLE | INITIALLY DEFERRED | INITIALLY IMMEDIATE ]
```

其中索引参数 index_parameters 为：

```
[ WITH ( {storage_parameter = value} [, ... ] ) ]
   [ USING INDEX TABLESPACE tablespace_name ]
```

❖ 重命名表。对名称的修改不会影响所存储的数据。

```
ALTER TABLE [ IF EXISTS ] table_name
   RENAME TO new_table_name;
```

❖ 重命名表中指定的列。

```
ALTER TABLE [ IF EXISTS ] { table_name [*] | ONLY table_name | ONLY ( table_name )}
   RENAME [ COLUMN ] column_name TO new_column_name;
```

❖ 重命名表的约束。

```
ALTER TABLE [ IF EXISTS ] { table_name [*] | ONLY table_name | ONLY ( table_name ) }
   RENAME CONSTRAINT constraint_name TO new_constraint_name;
```

❖ 设置表的所属模式。

```
ALTER TABLE [ IF EXISTS ] table_name
   SET SCHEMA new_schema;
```

说明

● 这种形式把表移动到另外一个模式。相关的索引、约束都跟着移动。目前序列不支持改变 schema。 若该表拥有序列，需要将序列删除，重建，或者取消拥有关系， 才能将表 schema 更改成功。

● 要修改一个表的模式，用户必须在新模式上拥有 CREATE 权限。要把该表添加为一个父表的新子表，用户必须同时又是父表的所有者。要修改所有者，用户还必须是新的所有角色的直接或间接成员，并且该成员必须在此表的模式上有 CREATE 权限。这些限制规定了该用户不能做出了重建和删除表之外的事情。不过，系统管理员可以以任何方式修改任意表的所有权限。

● 除了 RENAME 和 SET SCHEMA 之外所有动作都可以捆绑在一个经过多次修改的列表中并行使用。比如，可以在一个命令里增加几个字段或修改几个字段的类型。对于大表，此种操作带来的效率提升更明显，原因在于只需要对该大表做一次处理。

● 增加一个 CHECK 或 NOT NULL 约束将会扫描该表，以保证现有的行符合约束要求。

● 用一个非空缺省值增加一个字段或者改变一个字段的现有类型会重写整个表。对于大表来说，这个操作可能会花很长时间，并且它还临时需要两倍的磁盘空间。

❖ 添加多个列。

```
ALTER TABLE [ IF EXISTS ] table_name
   ADD ( { column_name data_type [ compress_mode ] [ COLLATE collation ] [ column_constraint
[ ... ] ]} [, ...] );
```

- 更新多个列。

```
ALTER TABLE [ IF EXISTS ] table_name
   MODIFY ( { column_name data_type | column_name [ CONSTRAINT constraint_name ] NOT NULL
[ ENABLE ] | column_name [ CONSTRAINT constraint_name ] NULL } [, ...] );
```

参数说明

- IF EXISTS

  如果不存在相同名称的表，不会抛出一个错误，而会发出一个通知，告知表不存在。

- table_name [*] | ONLY table_name | ONLY (table_name)

  table_name 是需要修改的表名。

  若声明了 ONLY 选项，则只有那个表被更改。若未声明 ONLY，该表及其所有子表都将会被更改。另外，可以在表名称后面显示地增加*选项来指定包括子表，即表示所有后代表都被扫描，这是默认行为。

- constraint_name

  要删除的现有约束的名称。

- index_name

  索引名称。

- storage_parameter

  表的存储参数的名称。

- new_owner

  表新拥有者的名称。

- new_tablespace

  表所属新的表空间名称。

- column_name，column_1_name， column_2_name

  现存的或新字段的名称。

- data_type

  新字段的类型，或者现存字段的新类型。

- compress_mode

  表字段的压缩可选项，当前仅对行存表有效。该子句指定该字段优先使用的压缩算法。

- collation

字段排序规则名称。可选字段 COLLATE 指定了新字段的排序规则，如果省略，排序规则为新字段的默认类型。

- USING expression

  USING 子句声明如何从旧的字段值里计算新的字段值；如果省略，缺省从旧类型向新类型的赋值转换。如果从旧数据类型到新类型没有隐含或者赋值的转换，则必须提供一个 USING 子句。

说明

ALTER TYPE 的 USING 选项实际上可以声明涉及该行旧值的任何表达式，即它可以引用除了正在被转换的字段之外其他的字段。这样，就可以用 ALTER TYPE 语法做非常普遍性的转换。因为这个灵活性，USING 表达式并没有作用于该字段的缺省值（如果有的话），结果可能不是缺省表达式要求的常量表达式。这就意味着如果从旧类型到新类型没有隐含或者赋值转换的话，即使存在 USING 子句，ALTER TYPE 也可能无法把缺省值转换成新的类型。在这种情况下，应该用 DROP DEFAULT 先删除缺省，执行 ALTER TYPE，然后使用 SET DEFAULT 增加一个合适的新缺省值。类似的考虑也适用于涉及该字段的索引和约束。

- NOT NULL | NULL

  设置列是否允许空值。

- integer

  带符号的整数常值。当使用 PERCENT 时表示按照表数据的百分比收集统计信息，integer 的取值范围为 0-100。

- attribute_option

  属性选项。

- PLAIN | EXTERNAL | EXTENDED | MAIN

  字段存储模式。

  - PLAIN 必需用于定长的数值（比如 integer）并且是内联的、不压缩的。
  - MAIN 用于内联、可压缩的数据。
  - EXTERNAL 用于外部保存、不压缩的数据。使用 EXTERNAL 将令在 text 和 bytea 字段上的子字符串操作更快，但付出的代价是增加了存储空间。
  - EXTENDED 用于外部的压缩数据，EXTENDED 是大多数支持非 PLAIN 存储的数据的缺省。

- CHECK (expression)

  每次将要插入的新行或者将要被更新的行必须使表达式结果为真才能成功，否则会抛出一个异常并且不会修改数据库。

  声明为字段约束的检查约束应该只引用该字段的数值，而在表约束里出现的表达式可以引用多个字段。

  目前，CHECK 表达式不能包含子查询也不能引用除当前行字段之外的变量。

- DEFAULT default_expr

给字段指定缺省值。

缺省表达式的数据类型必须和字段类型匹配。

缺省表达式将被用于任何未声明该字段数值的插入操作。如果没有指定缺省值则缺省值为 NULL 。

- UNIQUE index_parameters

  UNIQUE (column_name [, ...]) index_parameters

  UNIQUE 约束表示表里的一个或多个字段的组合必须在全表范围内唯一。

- PRIMARY KEY index_parameters

  PRIMARY KEY (column_name [, ...]) index_parameters

  主键约束表明表中的一个或者一些字段只能包含唯一（不重复）的非 NULL 值。

- DEFERRABLE | NOT DEFERRABLE | INITIALLY DEFERRED | INITIALLY IMMEDIATE

  设置该约束是否可推迟。

  - DEFERRABLE：可以推迟到事务结尾使用 SET CONSTRAINTS 命令检查。
  - NOT DEFERRABLE：在每条命令之后马上检查。
  - INITIALLY IMMEDIATE：那么每条语句之后就立即检查它。
  - INITIALLY DEFERRED：只有在事务结尾才检查它。

- WITH ({storage_parameter = value} [, ...])

  为表或索引指定一个可选的存储参数。

- tablespace_name

  索引所在表空间的名称。

- COMPRESS|NOCOMPRESS
  - NOCOMPRESS：如果指定关键字 NOCOMPRESS 则不会修改表的现有压缩特性。
  - COMPRESS：如果指定 COMPRESS 关键字，则对该表进行批量插入元组时触发该特性。

- new_table_name

  修改后新的表名称。

- new_column_name

  表中指定列修改后新的列名称。

- new_constraint_name

  修改后表约束的新名称。

- new_schema

  修改后新的模式名称。

- CASCADE

级联删除依赖于被依赖字段或者约束的对象（比如引用该字段的视图）。

- RESTRICT

  如果字段或者约束还有任何依赖的对象，则拒绝删除该字段。这是缺省行为。

- schema_name

  表所在的模式名称。

示例

请参考 CREATE TABLE 的示例。

相关链接

11.16.48CREATE TABLE，11.16.76DROP TABLE

11.16.18. ALTER TABLE PARTITION

功能描述

修改表分区，包括增删分区、切割分区、合成分区，以及修改分区属性等。

注意事项

- 添加分区的表空间不能是 PG_GLOBAL。
- 添加分区的名称不能与该分区表已有分区的名称相同。
- 添加分区的分区键值要和分区表的分区键的类型一致，且要大于分区表中最后一个范围分区的上边界。
- 如果目标分区表中已有分区数达到了最大值（32767），则不能继续添加分区。
- 当分区表只有一个分区时，不能删除该分区。
- 选择分区使用 PARTITION FOR()，括号里指定值个数应该与定义分区时使用的列个数相同，并且一一对应。
- Value 分区表不支持相应的 Alter Partition 操作。
- 列存分区表不支持切割分区。

语法格式

- 修改表分区主语法。

```
ALTER TABLE [ IF EXISTS ] { table_name  [*] | ONLY table_name | ONLY ( table_name  )}
    action [, ... ];
```

其中 action 统指如下分区维护子语法。当存在多个分区维护子句时，保证了分区的连续性，无论这些子句的排序如何，Vastbase 总会先执行 DROP PARTITION 再执行 ADD PARTITION 操作，最后顺序执行其它分区维护操作。

```
move_clause  |
   exchange_clause  |
   row_clause  |
   merge_clause  |
   modify_clause  |
   split_clause  |
   add_clause  |
   drop_clause
```

- move_clause 子语法用于移动分区到新的表空间。

```
MOVE PARTITION { partion_name | FOR ( partition_value [, ...] ) } TABLESPACE tablespacename
```

- exchange_clause 子语法用于把普通表的数据迁移到指定的分区。

```
EXCHANGE PARTITION { ( partition_name ) | FOR ( partition_value [, ...] ) }
   WITH TABLE {[ ONLY ] ordinary_table_name | ordinary_table_name * | ONLY
( ordinary_table_name )}
   [ { WITH | WITHOUT } VALIDATION ] [ VERBOSE ]
```

进行交换的普通表和分区必须满足如下条件：

  - 普通表和分区的列数目相同，对应列的信息严格一致，包括：列名、列的数据类型、列约束、列的 Collation 信息、列的存储参数、列的压缩信息等。
  - 普通表和分区的表压缩信息严格一致。
  - 普通表和分区的分布列信息严格一致。
  - 普通表和分区的索引个数相同，且对应索引的信息严格一致。
  - 普通表和分区的表约束个数相同，且对应表约束的信息严格一致。
  - 普通表不可以是临时表。

完成交换后，普通表和分区的数据被置换，同时普通表和分区的表空间信息被置换。此时，普通表和分区的统计信息变得不可靠，需要对普通表和分区重新执行 analyze。

- row_clause 子语法用于设置分区表的行迁移开关。

```
{ ENABLE | DISABLE } ROW MOVEMENT
```

- merge_clause 子语法用于把多个分区合并成一个分区。

```
MERGE PARTITIONS { partition_name } [, ...] INTO PARTITION partition_name
   [ TABLESPACE tablespacename ]
```

- modify_clause 子语法用于设置分区索引是否可用。

```
MODIFY PARTITION partition_name { UNUSABLE LOCAL INDEXES | REBUILD UNUSABLE LOCAL INDEXES }
```

- split_clause 子语法用于把一个分区切割成多个分区。

```
SPLIT PARTITION { partition_name | FOR ( partition_value [, ...] ) } { split_point_clause
| no_split_point_clause }
```

  - 指定切割点 split_point_clause 的语法为。

```
AT ( partition_value ) INTO ( PARTITION partition_name [ TABLESPACE tablespacename ] ,
PARTITION partition_name [ TABLESPACE tablespacename ] )
```

须知

- 列存分区表不支持切割分区。
- 切割点的大小要位于正在被切割的分区的分区键范围内，指定切割点的方式只能把一个分区切割成两个新分区。

■ 不指定切割点 no_split_point_clause 的语法为。

```
INTO { ( partition_less_than_item [, ...] ) | ( partition_start_end_item [, ...] ) }
```

须知

- 不指定切割点的方式，partition_less_than_item 指定的第一个新分区的分区键要大于正在被切割的分区的前一个分区（如果存在的话）的分区键，partition_less_than_item 指定的最后一个分区的分区键要等于正在被切割的分区的分区键大小。
- 不指定切割点的方式，partition_start_end_item 指定的第一个新分区的起始点（如果存在的话）必须等于正在被切割的分区的前一个分区（如果存在的话）的分区键，partition_start_end_item 指定的最后一个分区的终止点（如果存在的话）必须等于正在被切割的分区的分区键。
- partition_less_than_item 支持的分区键个数最多为 4，而 partition_start_end_item 仅支持 1 个分区键，其支持的数据类型参见 PARTITION BY RANGE(parti...。
- 在同一语句中 partition_less_than_item 和 partition_start_end_item 两者不可同时使用；不同 split 语句之间没有限制。

■ 分区项 partition_less_than_item 的语法为。

```
PARTITION partition_name VALUES LESS THAN ( { partition_value | MAXVALUE }  [, ...] )
    [ TABLESPACE tablespacename ]
```

■ 分区项 partition_start_end_item 的语法为，其约束参见 START END 语法描述。

```
PARTITION partition_name {
        {START(partition_value) END (partition_value) EVERY (interval_value)} |
        {START(partition_value) END ({partition_value | MAXVALUE})} |
        {START(partition_value)} |
        {END({partition_value | MAXVALUE})}
} [TABLESPACE tablespace_name]
```

– add_clause 子语法用于为指定的分区表添加一个或多个分区。

```
ADD {partition_less_than_item | partition_start_end_item}
```

– drop_clause 子语法用于删除分区表中的指定分区。

```
DROP PARTITION  { partition_name | FOR ( partition_value [, ...] )  }
```

❖ 修改表分区名称的语法。

```
ALTER TABLE [ IF EXISTS ] { table_name [*] | ONLY table_name | ONLY ( table_name  )}
   RENAME PARTITION { partion_name | FOR ( partition_value [, ...] ) } TO partition_new_name;
```

参数说明

❖ table_name

分区表名。

取值范围：已存在的分区表名。

- partition_name

  分区名。

  取值范围：已存在的分区名。

- tablespacename

  指定分区要移动到哪个表空间。

  取值范围：已存在的表空间名。

- partition_value

  分区键值。

  通过 PARTITION FOR (partition_value [, ...])子句指定的这一组值，可以唯一确定一个分区。

  取值范围：需要进行重命名的分区的分区键的取值范围。

- UNUSABLE LOCAL INDEXES

  设置该分区上的所有索引不可用。

- REBUILD UNUSABLE LOCAL INDEXES

  重建该分区上的所有索引。

- ENABLE/DISABLE ROW MOVEMET

  行迁移开关。

  如果进行 UPDATE 操作时，更新了元组在分区键上的值，造成了该元组所在分区发生变化，就会根据该开关给出报错信息，或者进行元组在分区间的转移。

  取值范围：

  - ENABLE：打开行迁移开关。
  - DISABLE：关闭行迁移开关。

  默认是打开状态。

- ordinary_table_name

  进行迁移的普通表的名称。

  取值范围：已存在的普通表名。

- { WITH | WITHOUT } VALIDATION

  在进行数据迁移时，是否检查普通表中的数据满足指定分区的分区键范围。

  取值范围：

  - WITH：对于普通表中的数据要检查是否满足分区的分区键范围，如果有数据不满足，则报错。

– WITHOUT：对于普通表中的数据不检查是否满足分区的分区键范围。

默认是 WITH 状态。

由于检查比较耗时，特别是当数据量很大的情况下更甚。所以在保证当前普通表中的数据满足分区的分区键范围时，可以加上 WITHOUT 来指明不进行检查。

❖ VERBOSE

在 VALIDATION 是 WITH 状态时，如果检查出普通表有不满足要交换分区的分区键范围的数据，那么把这些数据插入到正确的分区，如果路由不到任何分区，再报错。

须知

只有在 VALIDATION 是 WITH 状态时，才可以指定 VERBOSE。

❖ partition_new_name

分区的新名称。

取值范围：字符串，要符合标识符的命名规范。

示例

请参考 CREATE TABLE PARTITION 的示例。

相关链接

11.16.50CREATE TABLE PARTITION，11.16.76DROP TABLE

11.16.19. ALTER TABLESPACE

功能描述

修改表空间的属性。

注意事项

❖ 只有表空间的所有者有权限执行 ALTER TABLESPACE 命令，系统管理员默认拥有此权限。

❖ 要修改表空间的所有者 A 为 B，则 A 必须是 B 的直接或者间接成员。

说明

如果 new_owner 与 old_owner 一致，此处不再校验当前执行操作的用户是否具有修改权限，而直接显示 ALTER 成功。

语法格式

- 重命名表空间的语法。

```
ALTER TABLESPACE tablespace_name
   RENAME TO new_tablespace_name;
```

- 设置表空间所有者的语法。

```
ALTER TABLESPACE tablespace_name
   OWNER TO new_owner;
```

- 设置表空间属性的语法。

```
ALTER TABLESPACE tablespace_name
   SET ( {tablespace_option = value} [, ... ] );
```

- 重置表空间属性的语法。

```
ALTER TABLESPACE tablespace_name
   RESET ( { tablespace_option } [, ...] );
```

- 设置表空间限额的语法

```
ALTER TABLESPACE tablespace_name
   RESIZE MAXSIZE { UNLIMITED | 'space_size'};
```

参数说明

- tablespace_name

  要修改的表空间。

  取值范围：已存在的表空间名。

- new_tablespace_name

  表空间的新名称。

  新名称不能以"PG_"开头。

  取值范围：字符串，符合标识符命名规范。

- new_owner

  表空间的新所有者。

  取值范围：已存在的用户名。

- tablespace_option

  设置或者重置表空间的参数。

  取值范围：

  - seq_page_cost：设置优化器计算一次顺序获取磁盘页面的开销。缺省为 1.0。
  - random_page_cost：设置优化器计算一次非顺序获取磁盘页面的开销。缺省为 4.0。

说明

- random_page_cost 是相对于 seq_page_cost 的取值，等于或者小于 seq_page_cost 时毫无意义。
- 默认值为 4.0 的前提条件是，优化器采用索引来扫描表数据，并且表数据在 cache 中命中率可以 90%左右。

- 如果表数据空间要比物理内存小，那么减小该值到一个适当水平；相反地，如果表数据在 cache 中命中率要低于 90%，那么适当增大该值。
- 如果采用了类似于 SSD 的随机访问代价较小的存储器，可以适当减小该值，以反映真正的随机扫描代价。

value 的取值范围：正的浮点类型。

❖ RESIZE MAXSIZE

重新设置表空间限额的数值。

取值范围：

– UNLIMITED，该表空间不设置限额。

– 由 space_size 来确定，其格式参考 11.16.51CREATE TABLESPACE。

说明

修改参数 MAXSIZE 时也可使用：

```
ALTER TABLESPACE tablespace_name RESIZE MAXSIZE
 { 'UNLIMITED' | 'space_size'};
```

示例

请参考 CREATE TABLESPACE 的示例。

相关链接

11.16.51CREATE TABLESPACE，11.16.77DROP TABLESPACE

11.16.20. ALTER TEXT SEARCH CONFIGURATION

功能描述

更改文本搜索配置的定义。用户可以将映射从字串类型调整为字典，或者改变配置的名称或者所有者，或者修改搜索配置的配置参数。

ADD MAPPING FOR 选项为文本搜索配置增加字串类型映射；如果 ADD MAPPING FOR 后面任何一个字串类型的映射已经存在于此文本搜索配置中，那么系统将会报错。

ALTER MAPPING FOR 选项会首先清除已有的字串类型映射，然后添加指定的字串类型映射。

ALTER MAPPING REPLACE ... WITH ... 与 ALTER MAPPING FOR ... REPLACE ... WITH ...选项会直接使用 new_dictionary 替换 old_dictionary。需要注意的是，只有 pg_ts_config_map 系统表中存在 maptokentype 与 old_dictionary 对应关系的元组时，才能更新成功，否则不会成功，也不会有任何提示信息返回。

DROP MAPPING FOR 选项会删除当前文本搜索配置中指定的字串类型映射。 如果没有指定 IF EXISTS 选项，当 DROP MAPPING FOR 选项指定的字串类型映射在文本搜索配置中不存在时，数据库会报错。

注意事项

- 当一个搜索配置已经被引用（如被用来创建索引），则不允许用户修改此文本搜索配置。
- 要使用 ALTER TEXT SEARCH CONFIGURATION，用户必须是配置的所有者。

语法格式

- 增加文本搜索配置字串类型映射语法

```
ALTER TEXT SEARCH CONFIGURATION name
   ADD MAPPING FOR token_type [, ... ] WITH dictionary_name [, ... ];
```

- 修改文本搜索配置字典语法

```
ALTER TEXT SEARCH CONFIGURATION name
   ALTER MAPPING FOR token_type [, ... ] REPLACE old_dictionary WITH new_dictionary;
```

- 修改文本搜索配置字串类型语法

```
ALTER TEXT SEARCH CONFIGURATION name
   ALTER MAPPING FOR token_type [, ... ] WITH dictionary_name [, ... ];
```

- 更改文本搜索配置字典语法

```
ALTER TEXT SEARCH CONFIGURATION name
   ALTER MAPPING REPLACE old_dictionary WITH new_dictionary;
```

- 删除文本搜索配置字串类型映射语法

```
ALTER TEXT SEARCH CONFIGURATION name
   DROP MAPPING [ IF EXISTS ] FOR token_type [, ... ];
```

- 重命名文本搜索配置所有者语法

```
ALTER TEXT SEARCH CONFIGURATION name OWNER TO new_owner;
```

- 重命名文本搜索配置名称语法

```
ALTER TEXT SEARCH CONFIGURATION name RENAME TO new_name;
```

- 重命名文本搜索配置命名空间语法

```
ALTER TEXT SEARCH CONFIGURATION name SET SCHEMA new_schema;
```

- 修改文本搜索配置属性语法

```
ALTER TEXT SEARCH CONFIGURATION name SET ( { configuration_option = value } [, ...] );
```

- 重置文本搜索配置属性语法

```
ALTER TEXT SEARCH CONFIGURATION name RESET ( {configuration_option} [, ...] );
```

参数说明

- name

  已有文本搜索配置的名称（可以有模式修饰）。

- token_type

  与配置的语法解析器关联的字串类型的名称。详细信息参见 11.8.5 解析器。

- dictionary_name

  文本搜索字典名称。 如果有多个字典，则它们会按指定的顺序搜索。

- old_dictionary

映身中拟被替换的文本搜索字典名称。

- new_dictionary

  替换 old_dictionary 的文本搜索字典的名称。

- new_owner

  文本搜索配置的新所有者。

- new_name

  文本搜索配置的新名称。

- new_schema

  文本搜索配置的新模式名。

- configuration_option

  文本搜索配置项。详细信息参见 11.16.52CREATE TEXT SEARCH CONFIGURATION。

- value

  文本搜索配置项的值。

示例

```
--创建文本搜索配置。
vastbase=# CREATE TEXT SEARCH CONFIGURATION english_1 (parser=default);
CREATE TEXT SEARCH CONFIGURATION

--增加文本搜索配置字串类型映射语法。
vastbase=# ALTER TEXT SEARCH CONFIGURATION english_1 ADD MAPPING FOR word WITH simple,english_stem;
ALTER TEXT SEARCH CONFIGURATION

--增加文本搜索配置字串类型映射语法。
vastbase=# ALTER TEXT SEARCH CONFIGURATION english_1 ADD MAPPING FOR email WITH english_stem,
french_stem;
ALTER TEXT SEARCH CONFIGURATION

--查询文本搜索配置相关信息。
vastbase=# SELECT b.cfgname,a.maptokentype,a.mapseqno,a.mapdict,c.dictname FROM pg_ts_config_map
a,pg_ts_config b, pg_ts_dict c WHERE a.mapcfg=b.oid AND a.mapdict=c.oid AND b.cfgname='english_1'
ORDER BY 1,2,3,4,5;
  cfgname  | maptokentype | mapseqno | mapdict |   dictname
-----------+--------------+----------+---------+--------------
 english_1 |            2 |        1 |    3765 | simple
 english_1 |            2 |        2 |   12960 | english_stem
 english_1 |            4 |        1 |   12960 | english_stem
 english_1 |            4 |        2 |   12964 | french_stem
(4 rows)

--增加文本搜索配置字串类型映射语法。
vastbase=# ALTER TEXT SEARCH CONFIGURATION english_1 ALTER MAPPING REPLACE french_stem with german_stem;
ALTER TEXT SEARCH CONFIGURATION
```

```
--查询文本搜索配置相关信息。
vastbase=# SELECT b.cfgname,a.maptokentype,a.mapseqno,a.mapdict,c.dictname FROM pg_ts_config_map
a,pg_ts_config b, pg_ts_dict c WHERE a.mapcfg=b.oid AND a.mapdict=c.oid AND b.cfgname='english_1'
ORDER BY 1,2,3,4,5;
  cfgname  | maptokentype | mapseqno | mapdict |   dictname
-----------+--------------+----------+---------+--------------
 english_1 |            2 |        1 |    3765 | simple
 english_1 |            2 |        2 |   12960 | english_stem
 english_1 |            4 |        1 |   12960 | english_stem
 english_1 |            4 |        2 |   12966 | german_stem
(4 rows)
```

请参见 CREATE TEXT SEARCH CONFIGURATION 的示例。

相关链接

11.16.52CREATE TEXT SEARCH CONFIGURATION， 11.16.78DROP TEXT SEARCH CONFIGURATION

11.16.21. ALTER TEXT SEARCH DICTIONARY

功能描述

修改全文检索词典的相关定义，包括参数、名称、所有者、以及模式等。

注意事项

- 预定义词典不支持 ALTER 操作。
- 只有词典的所有者可以执行 ALTER 操作，系统管理员默认拥有此权限。
- 创建或修改词典之后，任何对于 filepath 路径下用户自定义的词典定义文件的修改，将不会影响到数据库中的词典。如果需要在数据库中使用这些修改，需使用 ALTER TEXT SEARCH DICTIONARY 语句更新对应词典的定义文件。

语法格式

- 修改词典定义。

```
ALTER TEXT SEARCH DICTIONARY name (
    option [ = value ] [, ... ]
);
```

- 重命名词典。

```
ALTER TEXT SEARCH DICTIONARY name RENAME TO new_name;
```

- 设置词典的所属模式。

```
ALTER TEXT SEARCH DICTIONARY name SET SCHEMA new_schema;
```

- 修改词典的所属者。

```
ALTER TEXT SEARCH DICTIONARY name OWNER TO new_owner;
```

参数说明

- name

  已存在的词典名（可指定模式名，否则默认在当前模式下）。

  取值范围：已存在的词典名。

- option

  要修改的参数名。与 template 对应，不同的词典类型具有不同的参数列表，且与指定顺序无关。详细参数说明请见 option。

说明

- 不支持修改词典的 TEMPLATE 参数值。
- 不支持仅修改 FILEPATH 参数而不修改对应的词典定义文件参数。
- 词典定义文件的文件名仅支持小写字母、数据、下划线混合。

- value

  要修改的参数值。如果省略等号（=）和 value，则表示删除该 option 的先前设置，使用默认值。

  取值范围：对应 option 定义。

- new_name

  词典的新名称。

  取值范围：符合标识符命名规范的字符串，且最大长度不超过 63 个字符。

- new_owner

  词典新的所有者。

  取值范围：已存在的用户。

- new_schema

  词典的新模式。

  取值范围：已存在的模式。

示例

```
--更改 Snowball 类型字典的停用词定义，其他参数保持不变。
vastbase=# ALTER TEXT SEARCH DICTIONARY my_dict ( StopWords = newrussian, FilePath =
'file:///home/dicts' );

--更改 Snowball 类型字典的 Language 参数，并删除停用词定义。
vastbase=# ALTER TEXT SEARCH DICTIONARY my_dict ( Language = dutch, StopWords );

--更新词典定义，不实际更改任何内容。
vastbase=# ALTER TEXT SEARCH DICTIONARY my_dict ( dummy );
```

相关链接

11.16.53CREATE TEXT SEARCH DICTIONARY，11.16.79DROP TEXT SEARCH DICTIONARY

11.16.22. ALTER TRIGGER

功能描述

修改触发器定义。

注意事项

只有触发器所在表的所有者可以执行 ALTER TRIGGER 操作，系统管理员默认拥有此权限。

语法格式

```
ALTER TRIGGER trigger_name ON table_name RENAME TO new_name;
```

参数说明

- trigger_name

  要修改的触发器名称。

  取值范围：已存在的触发器。

- table_name

  要修改的触发器所在的表名称。

  取值范围：已存在的含触发器的表。

- new_name

  修改后的新名称。

  取值范围：符合标识符命名规范的字符串，最大长度不超过 63 个字符，且不能与所在表上其他触发器同名。

示例

请参见 11.16.54CREATE TRIGGER 的示例。

相关链接

11.16.54CREATE TRIGGER，11.16.80DROP TRIGGER，11.16.17ALTER TABLE

11.16.23. ALTER TYPE

功能描述

修改一个类型的定义。

语法格式

- 修改类型

```
ALTER TYPE name action [, ... ]
ALTER TYPE name OWNER TO { new_owner | CURRENT_USER | SESSION_USER }
ALTER TYPE name RENAME ATTRIBUTE attribute_name TO new_attribute_name [ CASCADE | RESTRICT ]
ALTER TYPE name RENAME TO new_name
ALTER TYPE name SET SCHEMA new_schema
ALTER TYPE name ADD VALUE [ IF NOT EXISTS ] new_enum_value [ { BEFORE | AFTER }
neighbor_enum_value ]
ALTER TYPE name RENAME VALUE existing_enum_value TO new_enum_value

where action is one of:
    ADD ATTRIBUTE attribute_name data_type [ COLLATE collation ] [ CASCADE | RESTRICT ]
    DROP ATTRIBUTE [ IF EXISTS ] attribute_name [ CASCADE | RESTRICT ]
    ALTER ATTRIBUTE attribute_name [ SET DATA ] TYPE data_type [ COLLATE collation ] [ CASCADE
| RESTRICT ]
```

- 给复合类型增加新的属性。

```
ALTER TYPE name ADD ATTRIBUTE attribute_name data_type [ COLLATE collation ] [ CASCADE |
RESTRICT ]
```

- 从复合类型删除一个属性。

```
ALTER TYPE name DROP ATTRIBUTE [ IF EXISTS ] attribute_name [ CASCADE | RESTRICT ]
```

- 改变一种复合类型中某个属性的类型。

```
ALTER TYPE name ALTER ATTRIBUTE attribute_name [ SET DATA ] TYPE data_type [ COLLATE collation ]
[ CASCADE | RESTRICT ]
```

- 改变类型的所有者。

```
ALTER TYPE name OWNER TO { new_owner | CURRENT_USER | SESSION_USER }
```

- 改变类型的名称或是一个复合类型中的一个属性的名称。

```
ALTER TYPE name RENAME TO new_name
ALTER TYPE name RENAME ATTRIBUTE attribute_name TO new_attribute_name [ CASCADE | RESTRICT ]
```

- 将类型移至一个新的模式中。

```
ALTER TYPE name SET SCHEMA new_schema
```

- 为枚举类型增加一个新值。

```
ALTER TYPE name ADD VALUE [ IF NOT EXISTS ] new_enum_value [ { BEFORE | AFTER }
neighbor_enum_value ]
```

- 重命名枚举类型的一个标签值。

```
ALTER TYPE name RENAME VALUE existing_enum_value TO new_enum_value
```

参数说明

- name

一个需要修改的现有的类型的名称(可以有模式修饰) 。

- new_name

  该类型的新名称。

- new_owner

  新所有者的用户名 。

- new_schema

  该类型的新模式 。

- attribute_name

  拟增加、更改或删除的属性的名称。

- new_attribute_name

  拟改名的属性的新名称。

- data_type

  拟新增属性的数据类型，或是拟更改的属性的新类型名。

- new_enum_value

  枚举类型新增加的标签值，是一个非空的长度不超过 64 个字节的字符串。

- neighbor_enum_value

  一个已有枚举标签值，新值应该被增加在紧接着该枚举值之前或者之后的位置上。

- existing_enum_value

  现有的要重命名的枚举值，是一个非空的长度不超过 64 个字节的字符串

- CASCADE

  自动级联更新需更新类型以及相关联的记录和继承它们的子表。

- RESTRICT

  如果需联动更新类型是已更新类型的关联记录，则拒绝更新。这是缺省选项。

须知

- ADD ATTRIBUTE、DROP ATTRIBUTE 和 ALTER ATTRIBUTE 选项可以组合成一个列表同时处理。 例如，在一条命令中同时增加几个属性或是更改几个属性的类型是可以实现的。
- 要使用 ALTER TYPE，必须是该类型的所有者。 要修改一个类型的模式，还必须在新模式上拥有 CREATE 权限。 要修改所有者，必须是新的所有角色的直接或间接成员， 并且该成员必须在此类型的模式上有 CREATE 权限。 （这些限制强制了修改所有者不会做任何通过删除和重建类型不能做的事情。 不过，系统管理员可以以任何方式修改任意类型的所有权。） 要增加一个属性或是修改一个属性的类型，也必须有该类型的 USAGE 权限。

示例

请参考 CREATE TYPE 的示例。

相关链接

11.16.55CREATE TYPE，11.16.81DROP TYPE

11.16.24. ALTER USER

功能描述

修改数据库用户的属性。

注意事项

ALTER USER 中修改的会话参数只针对指定的用户，且在下一次会话中有效。

语法格式

- 修改用户的权限等信息。

```
ALTER USER user_name [ [ WITH ] option [ ... ] ];
```

其中 option 子句为。

```
{ CREATEDB | NOCREATEDB }
   | { CREATEROLE | NOCREATEROLE }
   | { INHERIT | NOINHERIT }
   | { AUDITADMIN | NOAUDITADMIN }
   | { SYSADMIN | NOSYSADMIN }
   | {MONADMIN | NOMONADMIN}
   | {OPRADMIN | NOOPRADMIN}
   | {POLADMIN | NOPOLADMIN}
   | { USEFT | NOUSEFT }
   | { LOGIN | NOLOGIN }
   | { REPLICATION | NOREPLICATION }
   | {INDEPENDENT | NOINDEPENDENT}
   | {VCADMIN | NOVCADMIN}
   | CONNECTION LIMIT connlimit
   | [ ENCRYPTED | UNENCRYPTED ] PASSWORD { 'password' | DISABLE }
   | [ ENCRYPTED | UNENCRYPTED ] IDENTIFIED BY { 'password' [ REPLACE 'old_password' ] | DISABLE }
   | VALID BEGIN 'timestamp'
   | VALID UNTIL 'timestamp'
   | RESOURCE POOL 'respool'
   | PERM SPACE 'spacelimit'
   | ACCOUNT { LOCK | UNLOCK }
   | PGUSER
```

- 修改用户名。

```
ALTER USER user_name
   RENAME TO new_name;
```

- 修改与用户关联的指定会话参数值。

```
ALTER USER user_name
   SET configuration_parameter { { TO | = } { value | DEFAULT } | FROM CURRENT };
```

- 重置与用户关联的指定会话参数值。

```
ALTER USER user_name
   RESET { configuration_parameter | ALL };
```

参数说明

- user_name

  现有用户名。

  取值范围：已存在的用户名。

- password

  新密码。

  密码规则如下：

  - 不能与当前密码相同。
  - 密码默认不少于 8 个字符。
  - 不能与用户名及用户名倒序相同。
  - 至少包含大写字母（A-Z），小写字母（a-z），数字（0-9），非字母数字字符（限定为~!@#$%^&*()-_=+\|[{}];:,<.>/?）四类字符中的三类字符。

  取值范围：字符串。

- old_password

  旧密码。

- ACCOUNT LOCK | ACCOUNT UNLOCK
  - ACCOUNT LOCK：锁定帐户，禁止登录数据库。
  - ACCOUNT UNLOCK：解锁帐户，允许登录数据库。
- PGUSER

  当前版本不允许修改用户的 PGUSER 属性。

其他参数请参见 11.16.44CREATE ROLE 和 11.16.10ALTER ROLE 的参数说明。

示例

请参考 CREATE USER 的示例。

相关链接

11.16.44CREATE ROLE，11.16.56CREATE USER，11.16.82DROP USER

11.16.25. ALTER VIEW

功能描述

ALTER VIEW 更改视图的各种辅助属性。（如果用户是更改视图的查询定义，要使用 CREATE OR REPLACE VIEW。）

注意事项

- 用户必须是视图的所有者才可以使用 ALTER VIEW。
- 要改变视图的模式，用户必须要有新模式的 CREATE 权限。
- 要改变视图的所有者，用户必须是新所属角色的直接或者间接的成员，并且此角色必须有视图模式的 CREATE 权限。
- 管理员用户可以更改任何视图的所属关系。

语法格式

- 设置视图列的默认值。

```
ALTER VIEW [ IF EXISTS ] view_name
   ALTER [ COLUMN ] column_name SET DEFAULT expression;
```

- 取消列视图列的默认值。

```
ALTER VIEW [ IF EXISTS ] view_name
   ALTER [ COLUMN ] column_name DROP DEFAULT;
```

- 修改视图的所有者。

```
ALTER VIEW [ IF EXISTS ] view_name
   OWNER TO new_owner;
```

- 重命名视图。

```
ALTER VIEW [ IF EXISTS ] view_name
   RENAME TO new_name;
```

- 设置视图的所属模式。

```
ALTER VIEW [ IF EXISTS ] view_name
   SET SCHEMA new_schema;
```

- 设置视图的选项。

```
ALTER VIEW [ IF EXISTS ] view_name
   SET ( { view_option_name [ = view_option_value ] } [, ... ] );
```

- 重置视图的选项。

```
ALTER VIEW [ IF EXISTS ] view_name
   RESET ( view_option_name [, ... ] );
```

参数说明

- IF EXISTS

  使用这个选项，如果视图不存在时不会产生错误，仅有会有一个提示信息。

- view_name

视图名称，可以用模式修饰。

取值范围：字符串，符合标识符命名规范。

- column_name

  可选的名称列表，视图的字段名。如果没有给出，字段名取自查询中的字段名。

  取值范围：字符串，符合标识符命名规范。

- SET/DROP DEFAULT

  设置或删除一个列的缺省值，该参数暂无实际意义。

- new_owner

  视图新所有者的用户名称。

- new_name

  视图的新名称。

- new_schema

  视图的新模式。

- view_option_name [= view_option_value]

  该子句为视图指定一个可选的参数。

  目前 view_option_name 支持的参数仅有 security_barrier，当 VIEW 试图提供行级安全时，应使用该参数。

  取值范围：Boolean 类型，TRUE、FALSE。

示例

```
--创建一个由 c_customer_sk 小于 150 的内容组成的视图。
vastbase=# CREATE VIEW tpcds.customer_details_view_v1 AS
   SELECT * FROM tpcds.customer
   WHERE c_customer_sk < 150;

--修改视图名称。
vastbase=# ALTER VIEW tpcds.customer_details_view_v1 RENAME TO customer_details_view_v2;

--修改视图所属 schema。
vastbase=# ALTER VIEW tpcds.customer_details_view_v2 SET schema public;

--删除视图。
vastbase=# DROP VIEW public.customer_details_view_v2;
```

相关链接

11.16.57CREATE VIEW，11.16.83DROP VIEW

11.16.26. ANALYZE | ANALYSE

功能描述

用于收集与数据库中普通表内容相关的统计信息，统计结果存储在系统表 PG_STATISTIC 下。执行计划生成器会使用这些统计数据，以确定最有效的执行计划。

如果没有指定参数，ANALYZE 会分析当前数据库中的每个表和分区表。同时也可以通过指定 table_name、column 和 partition_name 参数把分析限定在特定的表、列或分区表中。

ANALYZE|ANALYSE VERIFY 用于检测数据库中普通表（行存表、列存表）的数据文件是否损坏。

注意事项

ANALYZE 非临时表不能在一个匿名块、事务块、函数或存储过程内被执行。支持存储过程中 ANALYZE 临时表，不支持统计信息回滚操作。

ANALYZE VERIFY 操作处理的大多为异常场景检测需要使用 RELEASE 版本。ANALYZE VERIFY 场景不触发远程读，因此远程读参数不生效。对于关键系统表出现错误被系统检测出页面损坏时，将直接报错不再继续检测。

语法格式

- 收集表的统计信息。

```
{ ANALYZE | ANALYSE } [ VERBOSE ]
    [ table_name [ ( column_name [, ...] ) ] ];
```

- 收集分区表的统计信息。

```
{ ANALYZE | ANALYSE } [ VERBOSE ]
    [ table_name [ ( column_name [, ...] ) ] ]
    PARTITION ( patrition_name ) ;
```

说明

普通分区表目前支持针对某个分区的统计信息的语法，但功能上不支持针对某个分区的统计信息收集。

- 收集多列统计信息

```
{ANALYZE | ANALYSE} [ VERBOSE ]
    table_name (( column_1_name, column_2_name [, ...] ));
```

说明

- 收集多列统计信息时，请设置 GUC 参数 default_statistics_target 为负数，以使用百分比采样方式。
- 每组多列统计信息最多支持 32 列。
- 不支持收集多列统计信息的表：系统表。

- 检测当前库的数据文件

```
{ANALYZE | ANALYSE} VERIFY {FAST|COMPLETE};
```

说明

- Fast 模式校验时，需要对校验的表有并发的 DML 操作，会导致校验过程中有误报的问题，因为当前 Fast 模式是直接从磁盘上读取，并发有其他线程修改文件时，会导致获取的数据不准确，建议离线操作。

- 支持对全库进行操作，由于涉及的表较多，建议以重定向保存结果 vsql -d database -p port -f "verify.sql"> verify_warning.txt 2>&1。
- 不支持临时表和 unlog 表。
- 对外提示 NOTICE 只核对外可见的表，内部表的检测会包含在它所依赖的外部表，不对外显示和呈现。
- 此命令的处理可容错 ERROR 级别的处理。由于 debug 版本的 Assert 可能会导致 core 无法继续执行命令，建议在 release 模式下操作。
- 对于全库操作时，当关键系统表出现损坏则直接报错，不再继续执行。

❖ 检测表和索引的数据文件

```
{ANALYZE | ANALYSE} VERIFY {FAST|COMPLETE} table_name|index_name [CASCADE];
```

说明

- 支持对普通表的操作和索引表的操作，但不支持对索引表 index 使用 CASCADE 操作。原因是由于 CASCADE 模式用于处理主表的所有索引表，当单独对索引表进行检测时，无需使用 CASCADE 模式。
- 不支持临时表和 unlog 表。
- 对于主表的检测会同步检测主表的内部表，例如 toast 表、cudesc 表等。
- 当提示索引表损坏时，建议使用 reindex 命令进行重建索引操作。

❖ 检测表分区的数据文件

```
{ANALYZE | ANALYSE} VERIFY {FAST|COMPLETE} table_name PARTITION {(patrition_name)}[CASCADE];
```

说明

- 支持对表的单独分区进行检测操作，但不支持对索引表 index 使用 CASCADE 操作。
- 不支持临时表和 unlog 表.

参数说明

❖ VERBOSE

启用显示进度信息。

说明

如果指定了 VERBOSE，ANALYZE 发出进度信息，表明目前正在处理的表。各种有关表的统计信息也会打印出来。

❖ table_name

需要分析的特定表的表名（可能会带模式名），如果省略，将对数据库中的所有表（非外部表）进行分析。

对于 ANALYZE 收集统计信息，目前仅支持行存表、列存表。

取值范围：已有的表名。

❖ column_name，column_1_name，column_2_name

需要分析特定列的列名，默认为所有列。

取值范围：已有的列名。

❖ partition_name

如果 table 为分区表，在关键字 PARTITION 后面指定分区名 partition_name 表示分析该分区表的统计信息。目前语法上支持分区表做 ANALYZE，但功能实现上暂不支持对指定分区统计信息的分析。

取值范围：表的某一个分区名。

- index_name

  需要分析的特定索引表的表名（可能会带模式名）。

  取值范围：已有的表名。

- FAST|COMPLETE

  对于行存表，FAST 模式下主要对于行存表的 CRC 和 page header 进行校验，如果校验失败则会告警；而 COMPLETE 模式下，则主要对行存表的指针、tuple 进行解析校验。 对于列存表，FAST 模式下主要对于列存表的 CRC 和 magic 进行校验，如果校验失败则会告警；而 COMPLETE 模式下，则主要对列存表的 CU 进行解析校验。

- CASCADE

  CASCADE 模式下会对当前表的所有索引进行检测处理。

示例

--- 创建表。

```
vastbase=# CREATE TABLE customer_info
(
WR_RETURNED_DATE_SK      INTEGER                   ,
WR_RETURNED_TIME_SK      INTEGER                   ,
WR_ITEM_SK             INTEGER            NOT NULL,
WR_REFUNDED_CUSTOMER_SK   INTEGER
)
;
```

--- 创建分区表。

```
vastbase=# CREATE TABLE customer_par
(
WR_RETURNED_DATE_SK      INTEGER                   ,
WR_RETURNED_TIME_SK      INTEGER                   ,
WR_ITEM_SK             INTEGER            NOT NULL,
WR_REFUNDED_CUSTOMER_SK   INTEGER
)
PARTITION BY RANGE(WR_RETURNED_DATE_SK)
(
PARTITION P1 VALUES LESS THAN(2452275),
PARTITION P2 VALUES LESS THAN(2452640),
PARTITION P3 VALUES LESS THAN(2453000),
PARTITION P4 VALUES LESS THAN(MAXVALUE)
)
ENABLE ROW MOVEMENT;
```

--- 使用 ANALYZE 语句更新统计信息。

```
vastbase=# ANALYZE customer;
```

--- 使用 ANALYZE VERBOSE 语句更新统计信息，并输出表的相关信息。

```
vastbase=# ANALYZE VERBOSE customer_info;
INFO:  analyzing "cstore.pg_delta_3394584009"(cn_5002 pid=53078)
INFO:  analyzing "public.customer_info"(cn_5002 pid=53078)
INFO:  analyzing "public.customer_info" inheritance tree(cn_5002 pid=53078)
ANALYZE
```

说明

若环境若有故障，需查看数据库主节点的 log。

--- 删除表。

```
vastbase=# DROP TABLE customer;
vastbase=# DROP TABLE customer_par;
```

11.16.27. BEGIN

功能描述

BEGIN 可以用于开始一个匿名块，也可以用于开始一个事务。本节描述用 BEGIN 开始匿名块的语法，以 BEGIN 开始事务的语法见 11.16.111START TRANSACTION。

匿名块是能够动态地创建和执行过程代码的结构，而不需要以持久化的方式将代码作为数据库对象储存在数据库中。

注意事项

无。

语法格式

- 开启匿名块

```
[DECLARE [declare_statements]]
BEGIN
execution_statements
END;
/
```

- 开启事务

```
BEGIN [ WORK | TRANSACTION ]
  [
    {
      ISOLATION LEVEL { READ COMMITTED | SERIALIZABLE | REPEATABLE READ }
      | { READ WRITE | READ ONLY }
    } [, ...]
  ];
```

参数说明

- declare_statements

  声明变量，包括变量名和变量类型，如“sales_cnt int”。

- execution_statements

  匿名块中要执行的语句。

  取值范围：已存在的函数名称。

示例

无

相关链接

11.16.111START TRANSACTION

11.16.28. CALL

功能描述

使用 CALL 命令可以调用已定义的函数和存储过程。

注意事项

无。

语法格式

```
CALL [schema.] {func_name| procedure_name} ( param_expr );
```

参数说明

- schema

  函数或存储过程所在的模式名称。

- func_name

  所调用函数或存储过程的名称。

  取值范围：已存在的函数名称。

- param_expr

  参数列表可以用符号":="或者"=>"将参数名和参数值隔开，这种方法的好处是参数可以以任意顺序排列。若参数列表中仅出现参数值，则参数值的排列顺序必须和函数或存储过程定义时的相同。

  取值范围：已存在的函数参数名称或存储过程参数名称。

说明

参数可以包含入参（参数名和类型之间指定“IN”关键字）和出参（参数名和类型之间指定“OUT”关键字），使用CALL 命令调用函数或存储过程时，对于非重载的函数，参数列表必须包含出参，出参可以传入一个变量或者任一常量，详见示例。对于重载的 package 函数，参数列表里可以忽略出参，忽略出参时可能会导致函数找不到。包含出参时，出参只能是常量。

示例

```
--创建一个函数 func_add_sql，计算两个整数的和，并返回结果。
vastbase=# CREATE FUNCTION func_add_sql(num1 integer, num2 integer) RETURN integer
AS
BEGIN
RETURN num1 + num2;
END;
/

--按参数值传递。
vastbase=# CALL func_add_sql(1, 3);

--使用命名标记法传参。
vastbase=# CALL func_add_sql(num1 => 1,num2 => 3);
vastbase=# CALL func_add_sql(num2 := 2, num1 := 3);

--删除函数。
vastbase=# DROP FUNCTION func_add_sql;

--创建带出参的函数。
vastbase=# CREATE FUNCTION func_increment_sql(num1 IN integer, num2 IN integer, res OUT integer)
RETURN integer
AS
BEGIN
res := num1 + num2;
END;
/

--出参传入常量。
vastbase=# CALL func_increment_sql(1,2,1);

--删除函数。
vastbase=# DROP FUNCTION func_increment_sql;
```

11.16.29. CHECKPOINT

功能描述

检查点（CHECKPOINT）是一个事务日志中的点，所有数据文件都在该点被更新以反映日志中的信息，所有数据文件都将被刷新到磁盘。

设置事务日志检查点。预写式日志（WAL）缺省时在事务日志中每隔一段时间放置一个检查点。可以修改配置文件 postgres.conf，设置相关运行时的参数（checkpoint_segments 和 checkpoint_timeout）来调整这个原子化检查点的间隔。

注意事项

- 只有系统管理员可以调用 CHECKPOINT。
- CHECKPOINT 强迫立即进行检查，而不是等到下一次调度时的检查点。

语法格式

```
CHECKPOINT;
```

参数说明

无。

示例

```
--设置检查点。
vastbase=# CHECKPOINT;
```

11.16.30. CLOSE

功能描述

CLOSE 释放和一个游标关联的所有资源。

注意事项

- 不允许对一个已关闭的游标再做任何操作。
- 一个不再使用的游标应该尽早关闭。
- 当创建游标的事务用 COMMIT 或 ROLLBACK 终止之后，每个不可保持的已打开游标都隐含关闭。
- 当创建游标的事务通过 ROLLBACK 退出之后，每个可以保持的游标都将隐含关闭。
- 当创建游标的事务成功提交，可保持的游标将保持打开，直到执行一个明确的 CLOSE 或者客户端断开。
- Vastbase 没有明确打开游标的 OPEN 语句，因为一个游标在使用 CURSOR 命令定义的时候就打开了。可以通过查询系统视图 pg_cursors 看到所有可用的游标。

语法格式

```
CLOSE { cursor_name | ALL } ;
```

参数说明

- cursor_name

  一个待关闭的游标名称。

- ALL

  关闭所有已打开的游标。

示例

请参考 FETCH 的示例。

相关链接

11.16.87FETCH，11.16.91MOVE

11.16.31. CLUSTER

功能描述

根据一个索引对表进行聚簇排序。

CLUSTER 指定 Vastbase 通过索引名指定的索引聚簇由表名指定的表。 表名上必须已经定义该索引。

当对一个表聚集后，该表将基于索引信息进行物理存储。聚集是一次性操作：当表被更新之后， 更改的内容不会被聚集。也就是说，系统不会试图按照索引顺序对新的存储内容及更新记录进行重新聚集。

在对一个表聚簇之后，Vastbase 会记录在哪个索引上建立了聚集。 CLUSTER table_name 的聚集形式在之前的同一个索引的表上重新聚集。用户也可以用 ALTER TABLE 的 CLUSTER 或 SET WITHOUT CLUSTER 形式来设置索引来用于后续的聚集操作或清除任何之前的设置。

不含参数的 CLUSTER 会将当前用户所拥有的数据库中的先前做过聚簇的所有表重新处理，或者系统管理员调用的这些表。

在对一个表进行聚簇的时候，会在其上请求一个 ACCESS EXCLUSIVE 锁。这样就避免了在 CLUSTER 完成之前对此表执行其它的操作(包括读写)。

注意事项

只有行存 B-tree 索引支持 CLUSTER 操作。

如果用户只是随机访问表中的行，那么表中数据的实际存储顺序是无关紧要的。但是， 如果对某些数据的访问多于其它数据，而且有一个索引将这些数据分组， 那么将使用 CLUSTER 中会有所帮助。如果从一个表中请求一定索引范围的值， 或者是一个索引值对应多行，CLUSTER 也会有助于应用，因为如果索引标识出第一匹配行所在的存储页，所有其它行也可能已经在同一个存储页里了，这样便节省了磁盘访问的时间，加速了查询。

在聚簇过程中，系统先创建一个按照索引顺序建立的表的临时拷贝。同时也建立表上的每个索引的临时拷贝。因此，需要磁盘上有足够的剩余空间， 至少是表大小和索引大小的和。

因为 CLUSTER 记忆聚集信息，可以在第一次的时候手工对表进行聚簇，然后设置一个类似 VACUUM 的时间，这样就可以周期地自动对表进行聚簇操作。

因为优化器记录着有关表的排序的统计，所以建议在新近聚簇的表上运行 ANALYZE。否则，优化器可能会选择很差劲的查询规划。

CLUSTER 不允许在事务中执行。

语法格式

- 对一个表进行聚簇排序。

```
CLUSTER [ VERBOSE ] table_name [ USING index_name ];
```

- 对一个分区进行聚簇排序。

```
CLUSTER [ VERBOSE ] table_name PARTITION ( partition_name ) [ USING index_name ];
```

- 对已做过聚簇的表重新进行聚簇。

```
CLUSTER [ VERBOSE ];
```

参数说明

- VERBOSE

  启用显示进度信息。

- table_name

  表名称。

  取值范围：已存在的表名称。

- index_name

  索引名称。

  取值范围：已存在的索引名称。

- partition_name

  分区名称。

  取值范围：已存在的分区名称。

示例

```
-- 创建一个分区表。
vastbase=# CREATE TABLE tpcds.inventory_p1
(
    INV_DATE_SK             INTEGER             NOT NULL,
    INV_ITEM_SK             INTEGER             NOT NULL,
    INV_WAREHOUSE_SK         INTEGER             NOT NULL,
    INV_QUANTITY_ON_HAND     INTEGER
)
```

```
PARTITION BY RANGE(INV_DATE_SK)
(
      PARTITION P1 VALUES LESS THAN(2451179),
      PARTITION P2 VALUES LESS THAN(2451544),
      PARTITION P3 VALUES LESS THAN(2451910),
      PARTITION P4 VALUES LESS THAN(2452275),
      PARTITION P5 VALUES LESS THAN(2452640),
      PARTITION P6 VALUES LESS THAN(2453005),
      PARTITION P7 VALUES LESS THAN(MAXVALUE)
);

-- 创建索引ds_inventory_p1_index1。
vastbase=# CREATE INDEX ds_inventory_p1_index1 ON tpcds.inventory_p1 (INV_ITEM_SK) LOCAL;

-- 对表tpcds.inventory_p1进行聚集。
vastbase=# CLUSTER tpcds.inventory_p1 USING ds_inventory_p1_index1;

-- 对分区p3进行聚集。
vastbase=# CLUSTER tpcds.inventory_p1 PARTITION (p3) USING ds_inventory_p1_index1;

-- 对数据库中可以进行聚集的表进聚集。
vastbase=# CLUSTER;

--删除索引。
vastbase=# DROP INDEX tpcds.ds_inventory_p1_index1;

--删除分区表。
vastbase=# DROP TABLE tpcds.inventory_p1;
```

优化建议

- cluster
  - 建议在新近聚簇的表上运行 ANALYZE。否则，优化器可能会选择很差劲的查询规划。
  - 不允许在事务中执行 CLUSTER。

11.16.32. COMMENT

功能描述

定义或修改一个对象的注释。

注意事项

- 每个对象只存储一条注释，因此要修改一个注释，对同一个对象发出一条新的 COMMENT 命令即可。要删除注释，在文本字符串的位置写上 NULL 即可。当删除对象时，注释自动被删除掉。
- 目前注释浏览没有安全机制：任何连接到某数据库上的用户都可以看到所有该数据库对象的注释。共享对象（比如数据库、角色、表空间）的注释是全局存储的，连接到任何数据库的任何用户都可以看到它们。因此，不要在注释里存放与安全有关的敏感信息。

❖ 对大多数对象，只有对象的所有者可以设置注释。角色没有所有者，所以 COMMENT ON ROLE 命令仅可以由系统管理员对系统管理员角色执行，有 CREATEROLE 权限的角色也可以为非系统管理员角色设置注释。系统管理员可以对所有对象进行注释。

语法格式

```
COMMENT ON
{
 AGGREGATE agg_name (agg_type [, ...] ) |
 CAST (source_type AS target_type) |
 COLLATION object_name |
 COLUMN { table_name.column_name | view_name.column_name } |
 CONSTRAINT constraint_name ON table_name |
 CONVERSION object_name |
 DATABASE object_name |
 DOMAIN object_name |
 EXTENSION object_name |
 FOREIGN DATA WRAPPER object_name |
 FOREIGN TABLE object_name |
 FUNCTION function_name ( [ {[ argmode ] [ argname ] argtype} [, ...] ] ) |
 INDEX object_name |
 LARGE OBJECT large_object_oid |
 OPERATOR operator_name (left_type, right_type) |
 OPERATOR CLASS object_name USING index_method |
 OPERATOR FAMILY object_name USING index_method |
 [ PROCEDURAL ] LANGUAGE object_name |
 ROLE object_name |
 SCHEMA object_name |
 SERVER object_name |
 TABLE object_name |
 TABLESPACE object_name |
 TEXT SEARCH CONFIGURATION object_name |
 TEXT SEARCH DICTIONARY object_name |
 TEXT SEARCH PARSER object_name |
 TEXT SEARCH TEMPLATE object_name |
 TYPE object_name |
 VIEW object_name
}
  IS 'text';
```

参数说明

❖ agg_name

聚集函数的名称

❖ agg_type

聚集函数参数的类型

❖ source_type

类型转换的源数据类型。

- target_type

  类型转换的目标数据类型。

- object_name

  对象名。

- table_name.column_name

  view_name.column_name

  定义/修改注释的列名称。前缀可加表名称或者视图名称。

- constraint_name

  定义/修改注释的表约束的名称。

- table_name

  表的名称。

- function_name

  定义/修改注释的函数名称。

- argmode,argname,argtype

  函数参数的模式、名称、类型。

- large_object_oid

  定义/修改注释的大对象的 OID 值。

- operator_name

  操作符名称。

- left_type,right_type

  操作参数的数据类型（可以用模式修饰）。当前置或者后置操作符不存在时，可以增加 NONE 选项。

- text

  注释。

示例

```
vastbase=# CREATE TABLE tpcds.customer_demographics_t2
(
   CD_DEMO_SK            INTEGER               NOT NULL,
   CD_GENDER             CHAR(1)                       ,
   CD_MARITAL_STATUS      CHAR(1)                      ,
   CD_EDUCATION_STATUS     CHAR(20)                    ,
   CD_PURCHASE_ESTIMATE    INTEGER                     ,
   CD_CREDIT_RATING       CHAR(10)                     ,
   CD_DEP_COUNT          INTEGER                      ,
```

```
    CD_DEP_EMPLOYED_COUNT     INTEGER                  ,
    CD_DEP_COLLEGE_COUNT      INTEGER
)
WITH (ORIENTATION = COLUMN,COMPRESSION=MIDDLE)
;

-- 为tpcds.customer_demographics_t2.cd_demo_sk列加注释。
vastbase=# COMMENT ON COLUMN tpcds.customer_demographics_t2.cd_demo_sk IS 'Primary key of customer
demographics table.';

--创建一个由c_customer_sk小于150的内容组成的视图。
vastbase=# CREATE VIEW tpcds.customer_details_view_v2 AS
    SELECT *
    FROM tpcds.customer
    WHERE c_customer_sk < 150;

-- 为tpcds.customer_details_view_v2视图加注释。
vastbase=# COMMENT ON VIEW tpcds.customer_details_view_v2 IS 'View of customer detail';

-- 删除view。
vastbase=# DROP VIEW tpcds.customer_details_view_v2;

-- 删除tpcds.customer_demographics_t2。
vastbase=# DROP TABLE tpcds.customer_demographics_t2;
```

11.16.33. COMMIT | END

功能描述

通过 COMMIT 或者 END 可完成提交事务的功能，即提交事务的所有操作。

注意事项

执行 COMMIT 这个命令的时候，命令执行者必须是该事务的创建者或系统管理员，且创建和提交操作可以不在同一个会话中。

语法格式

```
{ COMMIT | END } [ WORK | TRANSACTION ] ;
```

参数说明

- COMMIT | END

  提交当前事务，让所有当前事务的更改为其他事务可见。

- WORK | TRANSACTION

  可选关键字，除了增加可读性没有其他任何作用。

示例

```
--创建表。
vastbase=# CREATE TABLE tpcds.customer_demographics_t2
(
   CD_DEMO_SK             INTEGER            NOT NULL,
   CD_GENDER              CHAR(1)                    ,
   CD_MARITAL_STATUS       CHAR(1)                    ,
   CD_EDUCATION_STATUS     CHAR(20)                   ,
   CD_PURCHASE_ESTIMATE    INTEGER                    ,
   CD_CREDIT_RATING        CHAR(10)                   ,
   CD_DEP_COUNT           INTEGER                    ,
   CD_DEP_EMPLOYED_COUNT    INTEGER                    ,
   CD_DEP_COLLEGE_COUNT     INTEGER
)
WITH (ORIENTATION = COLUMN,COMPRESSION=MIDDLE)
;

--开启事务。
vastbase=# START TRANSACTION;

--插入数据。
vastbase=# INSERT INTO tpcds.customer_demographics_t2 VALUES(1,'M', 'U', 'DOCTOR DEGREE', 1200, 'GOOD',
1, 0, 0);
vastbase=# INSERT INTO tpcds.customer_demographics_t2 VALUES(2,'F', 'U', 'MASTER DEGREE', 300, 'BAD',
1, 0, 0);

--提交事务，让所有更改永久化。
vastbase=# COMMIT;

--查询数据。
vastbase=# SELECT * FROM tpcds.customer_demographics_t2;

--删除表tpcds.customer_demographics_t2。
vastbase=# DROP TABLE tpcds.customer_demographics_t2;
```

相关链接

11.16.100ROLLBACK

11.16.34. COMMIT PREPARED

功能描述

提交一个早先为两阶段提交准备好的事务。

注意事项

- 该功能仅在维护模式(GUC 参数 xc_maintenance_mode 为 on 时)下可用。该模式谨慎打开，一般供维护人员排查问题使用，一般用户不应使用该模式。

- 命令执行者必须是该事务的创建者或系统管理员，且创建和提交操作可以不在同一个会话中。
- 事务功能由数据库自动维护，不应显式使用事务功能。

语法格式

```
COMMIT PREPARED transaction_id ;
COMMIT PREPARED transaction_id WITH CSN;
```

参数说明

- transaction_id

  待提交事务的标识符。它不能和任何当前预备事务已经使用了的标识符同名。

- CSN(commit sequence number)

  待提交事务的序列号。它是一个 64 位递增无符号数。

示例

```
--提交标识符为的 trans_test 的事务。
vastbase=# COMMIT PREPARED 'trans_test';
```

相关链接

11.16.94PREPARE TRANSACTION，11.16.101ROLLBACK PREPARED。

11.16.35. COPY

功能描述

通过 COPY 命令实现在表和文件之间拷贝数据。
COPY FROM 从一个文件拷贝数据到一个表，COPY TO 把一个表的数据拷贝到一个文件。

注意事项

- 执行 COPY FROM FILENAME 或 COPY TO FILENAME 语句需要 SYSADMIN 权限，但默认禁止 SYSADMIN 用户对数据库配置文件，密钥文件，证书文件和审计日志执行 COPY FROM FILENAME 或 COPY TO FILENAME，以防止 SYSADMIN 用户越权查看或修改敏感文件。放开这一权限需要通过更改 enable_copy_server_files 的设定来完成。
- COPY 只能用于表，不能用于视图。
- 对任何要插入数据的表必须有插入权限。
- 如果声明了一个字段列表，COPY 将只在文件和表之间拷贝已声明字段的数据。如果表中有任何不在字段列表里的字段，COPY FROM 将为那些字段插入缺省值。

❖ 如果声明了数据源文件，服务器必须可以访问该文件；如果指定了 STDIN，数据将在客户前端和服务器之间流动，输入时，表的列与列之间使用 TAB 键分隔，在新的一行中以反斜杠和句点（\.）表示输入结束。

❖ 如果数据文件的任意行包含比预期多或者少的字段，COPY FROM 将抛出一个错误。

❖ 数据的结束可以用一个只包含反斜杠和句点（\.）的行表示。如果从文件中读取数据，数据结束的标记是不必要的；如果在客户端应用之间拷贝数据，必须要有结束标记。

❖ COPY FROM 中\N 为空字符串，如果要输入实际数据值\N ，使用\\N。

❖ COPY FROM 不支持在导入过程中对数据做预处理（比如说表达式运算，填充指定默认值等）。如果需要在导入过程中对数据做预处理，用户需先把数据导入到临时表中，然后执行 SQL 语句通过运算插入到表中，但此方法会导致 I/O 膨胀，降低导入性能。

❖ COPY FROM 在遇到数据格式错误时会回滚事务，但没有足够的错误信息，不方便用户从大量的原始数据中定位错误数据。

❖ COPY FROM/TO 适合低并发，本地小数据量导入导出。

语法格式

❖ 从一个文件拷贝数据到一个表。

```
COPY table name [ ( column name [, ...] ) ]
   FROM { 'filename' | STDIN }
   [ [ USING ] DELIMITERS 'delimiters' ]
   [ WITHOUT ESCAPING ]
   [ LOG ERRORS ]
   [ REJECT LIMIT 'limit' ]
   [ WITH ( option [, ...] ) ]
   | copy_option
   | FIXED FORMATTER ( { column_name( offset, length ) } [, ...] ) [ ( option [, ...] ) | copy_option
[  ...] ] ];
```

说明

语法中的 FIXED FORMATTER ({ column_name(offset, length) } [, ...])以及 [(option [, ...]) | copy_option [...]] 可以任意排列组合。

❖ 把一个表的数据拷贝到一个文件。

```
COPY table_name [ ( column_name [, ...] ) ]
   TO { 'filename' | STDOUT }
   [ [ USING ] DELIMITERS 'delimiters' ]
   [ WITHOUT ESCAPING ]
   [ WITH ( option [, ...] ) ]
   | copy_option
   | FIXED FORMATTER ( { column_name( offset, length ) } [, ...] ) [ ( option [, ...] ) | copy_option
[  ...] ] ];

COPY query
   TO { 'filename' | STDOUT }
   [ WITHOUT ESCAPING ]
   [ WITH ( option [, ...] ) ]
   | copy_option
```

```
   | FIXED FORMATTER ( { column_name( offset, length ) } [, ...] ) [ ( option [, ...] ) | copy_option
[  ...] ] ];
```

说明

1. COPY TO 语法形式约束如下：

(query)与[USING] DELIMITER 不兼容，即若 COPY TO 的数据来自于一个 query 的查询结果，那么 COPY TO 语法不能再指定[USING] DELIMITERS 语法子句。

2. 对于 FIXED FORMATTTER 语法后面跟随的 copy_option 是以空格进行分隔的。

3. copy_option 是指 COPY 原生的参数形式，而 option 是兼容外表导入的参数形式。

4. 语法中的 FIXED FORMATTER ({ column_name(offset, length) } [, ...])以及 [(option [, ...]) | copy_option [...]] 可以任意排列组合。

其中可选参数 option 子句语法为：

```
FORMAT 'format_name'
| OIDS [ boolean ]
| DELIMITER 'delimiter_character'
| NULL 'null_string'
| HEADER [ boolean ]
| FILEHEADER 'header_file_string'
| FREEZE [ boolean ]
| QUOTE 'quote_character'
| ESCAPE 'escape_character'
| EOL 'newline_character'
| NOESCAPING [ boolean ]
| FORCE_QUOTE { ( column_name [, ...] ) | * }
| FORCE_NOT_NULL ( column_name [, ...] )
| ENCODING 'encoding_name'
| IGNORE_EXTRA_DATA [ boolean ]
| FILL_MISSING_FIELDS [ boolean ]
| COMPATIBLE_ILLEGAL_CHARS [ boolean ]
| DATE_FORMAT 'date_format_string'
| TIME_FORMAT 'time_format_string'
| TIMESTAMP_FORMAT 'timestamp_format_string'
| SMALLDATETIME_FORMAT 'smalldatetime_format_string'
```

其中可选参数 copy_option 子句语法为：

```
OIDS
| NULL 'null_string'
| HEADER
| FILEHEADER 'header_file_string'
| FREEZE
| FORCE_NOT_NULL column_name [, ...]
| FORCE_QUOTE { column_name [, ...] | * }
| BINARY
| CSV
| QUOTE [ AS ] 'quote character'
| ESCAPE [ AS ] 'escape character'
| EOL 'newline character'
| ENCODING 'encoding name'
| IGNORE EXTRA DATA
| FILL MISSING FIELDS
| COMPATIBLE_ILLEGAL_CHARS
```

```
| DATE_FORMAT 'date_format_string'
| TIME_FORMAT 'time_format_string'
| TIMESTAMP_FORMAT 'timestamp_format_string'
| SMALLDATETIME_FORMAT 'smalldatetime_format_string'
```

参数说明

- query

  其结果将被拷贝。

  取值范围：一个必须用圆括弧包围的 SELECT 或 VALUES 命令。

- table_name

  表的名称（可以有模式修饰）。

  取值范围：已存在的表名。

- column_name

  可选的待拷贝字段列表。

  取值范围：如果没有声明字段列表，将使用所有字段。

- STDIN

  声明输入是来自标准输入。

- STDOUT

  声明输出打印到标准输出。

- FIXED

  打开字段固定长度模式。在字段固定长度模式下，不能声明 DELIMITER，NULL，CSV 选项。指定 FIXED 类型后，不能再通过 option 或 copy_option 指定 BINARY、CSV、TEXT 等类型。

说明

定长格式定义如下：

1. 每条记录的每个字段长度相同。
2. 长度不足的字段以空格填充，数字类型字段左对齐，字符字段右对齐。
3. 字段和字段之间没有分隔符。

- [USING] DELIMITER 'delimiters'

  在文件中分隔各个字段的字符串，分隔符最大长度不超过 10 个字节。

  取值范围：不允许包含\.abcdefghijklmnopqrstuvwxyz0123456789 中的任何一个字符。

  缺省值：在文本模式下，缺省是水平制表符，在 CSV 模式下是一个逗号。

- WITHOUT ESCAPING

  在 TEXT 格式中，不对'\'和后面的字符进行转义。

取值范围：仅支持 TEXT 格式。

- LOG ERRORS

  若指定，则开启对于 COPY FROM 语句中数据类型错误的容错机制，相关错误行的错误记录会记录到此库中 public.pgxc_copy_error_log 表中，备后续查阅。

  取值范围：仅支持导入（即 COPY FROM）时指定。

说明

此容错选项的使用限制如下：

- 此容错机制仅捕捉 COPY FROM 过程中数据库主节点上数据解析过程中相关的数据类型错误（DATA_EXCEPTION）。
- 在每个库第一次使用时 COPY FROM 容错时，请先行检查 public.pgxc_copy_error_log（COPY 错误表）是否存在，若不存在请调用 copy_error_log_create() 函数创建；若存在，请转移此表数据并删除这张表后，调用 copy_error_log_create() 函数创建。更多关于表 public.pgxc_copy_error_log 的字段信息，请参见表 11-32。
- 若在指定了 LOG ERRORS 的 COPY FROM 运行时，public.pgxc_copy_error_log 不存在（未创建或者已删除）或表定义不符合 copy_error_log_create() 中的预设表定义，则会报错。因此请确定此 COPY 错误表是使用 copy_error_log_create() 函数创建的，否则可能导致容错的 COPY FROM 语句无法正常执行。
- COPY 已有的容错选项（如 IGNORE_EXTRA_DATA）开启时，对应类型的错误会按照已有的方式处理而不会报出异常，因此错误表也不会有相应数据。

- LOG ERRORS DATA

  LOG ERRORS DATA 和 LOG ERRORS 的区别:

  a. LOG ERRORS DATA 会填充容错表的 rawrecord 字段。

  b. 只有 supper 权限的用户才能使用 LOG ERRORS DATA 参数选项。

注意

使用“LOG ERRORS DATA”时，若错误内容过于复杂可能存在写入容错表失败的风险，导致任务失败。

- REJECT LIMIT 'limit'

  与 LOG ERROR 选项共同使用，对 COPY FROM 的容错机制设置数值上限，一旦此 COPY FROM 语句错误数据超过选项指定条数，则会按照原有机制报错。

  取值范围：正整数（1-INTMAX），'unlimited'（无最大值限制）

  缺省值：若未指定 LOG ERRORS，则会报错；若指定 LOG ERRORS，则默认为 0。

说明

如上述 LOG ERRORS 中描述的容错机制，REJECT LIMIT 的计数也是按照执行 COPY FROM 的数据库主节点上遇到的解析错误数量计算，而不是数据库节点的错误数量。

- FORMATTER

在固定长度模式中，定义每一个字段在数据文件中的位置。按照 column(offset,length)格式定义每一列在数据文件中的位置。

取值范围：

- offset 取值不能小于 0，以字节为单位。
- length 取值不能小于 0，以字节为单位。

所有列的总长度和不能大于 1GB。

文件中没有出现的列默认以空值代替。

❖ OPTION { option_name ' value ' }

用于指定兼容外表的各类参数。

- FORMAT

  数据源文件的格式。

  取值范围：CSV、TEXT、FIXED、BINARY。

  - CSV 格式的文件，可以有效处理数据列中的换行符，但对一些特殊字符处理有欠缺。
  - TEXT 格式的文件，可以有效处理一些特殊字符，但无法正确处理数据列中的换行符。
  - FIXED 格式的文件，适用于每条数据的数据列都比较固定的数据，长度不足的列会添加空格补齐，过长的列则会自动截断。
  - BINARY 形式的选项会使得所有的数据被存储/读作二进制格式而不是文本。 这比 TEXT 和 CSV 格式的要快一些，但是一个 BINARY 格式文件可移植性比较差。

  缺省值：TEXT

- DELIMITER

  指定数据文件行数据的字段分隔符。

说明

- 分隔符不能是\r 和\n。
- 分隔符不能和 null 参数相同，CSV 格式数据的分隔符不能和 quote 参数相同。
- TEXT 格式数据的分隔符不能包含： 小写字母、数字和特殊字符.\。
- 数据文件中单行数据长度需<1GB，如果分隔符较长且数据列较多的情况下，会影响导出有效数据的长度。
- 分隔符推荐使用多字符和不可见字符。多字符例如'$^&'；不可见字符例如 0x07，0x08，0x1b 等。

  取值范围：支持多字符分隔符，但分隔符不能超过 10 个字节。

  缺省值：

  - TEXT 格式的默认分隔符是水平制表符（tab）。
  - CSV 格式的默认分隔符为 “,” 。

■ FIXED 格式没有分隔符。

– NULL

用来指定数据文件中空值的表示。

取值范围：

■ null 值不能是\r 和\n，最大为 100 个字符。
■ null 值不能和分隔符、quote 参数相同。

缺省值：

■ CSV 格式下默认值是一个没有引号的空字符串。
■ 在 TEXT 格式下默认值是\N。

– HEADER

指定导出数据文件是否包含标题行，标题行一般用来描述表中每个字段的信息。header 只能用于 CSV，FIXED 格式的文件中。

在导入数据时，如果 header 选项为 on，则数据文本第一行会被识别为标题行，会忽略此行。如果 header 为 off，而数据文件中第一行会被识别为数据。

在导出数据时，如果 header 选项为 on，则需要指定 fileheader。如果 header 为 off，则导出数据文件不包含标题行。

取值范围：true/on，false/off。

缺省值：false

– QUOTE

CSV 格式文件下的引号字符。

缺省值：双引号

说明

- quote 参数不能和分隔符、null 参数相同。
- quote 参数只能是单字节的字符。
- 推荐不可见字符作为 quote，例如 0x07，0x08，0x1b 等。

– ESCAPE

CSV 格式下，用来指定逃逸字符，逃逸字符只能指定为单字节字符。

缺省值：双引号。当与 quote 值相同时，会被替换为'\0'。

– EOL 'newline_character'

指定导入导出数据文件换行符样式。

取值范围：支持多字符换行符，但换行符不能超过 10 个字节。常见的换行符，如\r、\n、\r\n（设成 0x0D、0x0A、0x0D0A 效果是相同的），其他字符或字符串，如$、#。

说明

- EOL 参数只能用于 TEXT 格式的导入导出，不支持 CSV 格式和 FIXED 格式导入。为了兼容原有 EOL 参数，仍然支持导出 CSV 格式和 FIXED 格式时指定 EOL 参数为 0x0D 或 0x0D0A。
- EOL 参数不能和分隔符、null 参数相同。
- EOL 参数不能包含：.abcdefghijklmnopqrstuvwxyz0123456789。

– FORCE_QUOTE { (column_name [, ...]) | * }

在 CSV COPY TO 模式下，强制在每个声明的字段周围对所有非 NULL 值都使用引号包围。NULL 输出不会被引号包围。

取值范围：已存在的字段。

– FORCE_NOT_NULL (column_name [, ...])

在 CSV COPY FROM 模式下，指定的字段输入不能为空。

取值范围：已存在的字段。

– ENCODING

指定数据文件的编码格式名称，缺省为当前数据库编码格式。

– IGNORE_EXTRA_DATA

若数据源文件比外表定义列数多，是否会忽略对多出的列。该参数只在数据导入过程中使用。

取值范围：true/on、false/off。

- 参数为 true/on，若数据源文件比外表定义列数多，则忽略行尾多出来的列。
- 参数为 false/off，若数据源文件比外表定义列数多，会显示如下错误信息。

```
extra data after last expected column
```

缺省值：false。

须知

如果行尾换行符丢失，使两行变成一行时，设置此参数为 true 将导致后一行数据被忽略掉。

– COMPATIBLE_ILLEGAL_CHARS

导入非法字符容错参数。此语法仅对 COPY FROM 导入有效。

取值范围：true/on，false/off。

- 参数为 true/on，则导入时遇到非法字符进行容错处理，非法字符转换后入库，不报错，不中断导入。
- 参数为 false/off，导入时遇到非法字符进行报错，中断导入。

缺省值：false/off

说明

导入非法字符容错规则如下：

（1）对于'\0'，容错后转换为空格；

（2）对于其他非法字符，容错后转换为问号；

（3）若 compatible_illegal_chars 为 true/on 标识导入时对于非法字符进行容错处理，则若 NULL、DELIMITER、QUOTE、ESCAPE 设置为空格或问号则会通过如"illegal chars conversion may confuse COPY escape 0x20"等报错信息提示用户修改可能引起混淆的参数以避免导入错误。

- FILL_MISSING_FIELDS

  当数据加载时，若数据源文件中一行的最后一个字段缺失的处理方式。

  取值范围：true/on，false/off。

  缺省值：false/off

- DATE_FORMAT

  导入对于 DATE 类型指定格式。此参数不支持 BINARY 格式，会报“cannot specify bulkload compatibility options in BINARY mode”错误信息。此参数仅对 COPY FROM 导入有效。

  取值范围：合法 DATE 格式。可参考 11.5.8 时间和日期处理函数和操作符。

说明

对于 DATE 类型内建为 TIMESTAMP 类型的数据库，在导入的时候，若需指定格式，可以参考下面的 timestamp_format 参数。

- TIME_FORMAT

  导入对于 TIME 类型指定格式。此参数不支持 BINARY 格式，会报“cannot specify bulkload compatibility options in BINARY mode”错误信息。此参数仅对 COPY FROM 导入有效。

  取值范围：合法 TIME 格式，不支持时区。可参考 11.5.8 时间和日期处理函数和操作符。

- TIMESTAMP_FORMAT

  导入对于 TIMESTAMP 类型指定格式。此参数不支持 BINARY 格式，会报“cannot specify bulkload compatibility options in BINARY mode”错误信息。此参数仅对 COPY FROM 导入有效。

  取值范围：合法 TIMESTAMP 格式，不支持时区。可参考 11.5.8 时间和日期处理函数和操作符。

- SMALLDATETIME_FORMAT

导入对于 SMALLDATETIME 类型指定格式。此参数不支持 BINARY 格式，会报“cannot specify bulkload compatibility options in BINARY mode”错误信息。此参数仅对 COPY FROM 导入有效。

取值范围：合法 SMALLDATETIME 格式。可参考 11.5.8 时间和日期处理函数和操作符。

❖ COPY_OPTION { option_name ' value ' }

用于指定 COPY 原生的各类参数。

– NULL null_string

用来指定数据文件中空值的表示。

须知

在使用 COPY FROM 的时候，任何匹配这个字符串的字符串将被存储为 NULL 值，所以应该确保指定的字符串和 COPY TO 相同。

取值范围：

■ null 值不能是\r 和\n，最大为 100 个字符。

■ null 值不能和分隔符、quote 参数相同。

缺省值：

■ 在 TEXT 格式下默认值是\N。

■ CSV 格式下默认值是一个没有引号的空字符串。

– HEADER

指定导出数据文件是否包含标题行，标题行一般用来描述表中每个字段的信息。header 只能用于 CSV，FIXED 格式的文件中。

在导入数据时，如果 header 选项为 on，则数据文本第一行会被识别为标题行，会忽略此行。如果 header 为 off，而数据文件中第一行会被识别为数据。

在导出数据时，如果 header 选项为 on，则需要指定 fileheader。如果 header 为 off，则导出数据文件不包含标题行。

– FILEHEADER

导出数据时用于定义标题行的文件，一般用来描述每一列的数据信息。

须知

- 仅在 header 为 on 或 true 的情况下有效。
- fileheader 指定的是绝对路径。
- 该文件只能包含一行标题信息，并以换行符结尾，多余的行将被丢弃（标题信息不能包含换行符）。

- 该文件包括换行符在内长度不超过 1M。

- FREEZE

  将 COPY 加载的数据行设置为已经被 frozen，就像这些数据行执行过 VACUUM FREEZE。

  这是一个初始数据加载的性能选项。仅当以下三个条件同时满足时，数据行会被 frozen：

  - 在同一事务中 create 或 truncate 这张表之后执行 COPY。
  - 当前事务中没有打开的游标。
  - 当前事务中没有原有的快照。

说明

COPY 完成后，所有其他会话将会立刻看到这些数据。但是这违反了 MVCC 可见性的一般原则，用户应当了解这样会导致潜在的风险。

- FORCE NOT NULL column_name [, ...]

  在 CSV COPY FROM 模式下，指定的字段不为空。若输入为空，则将视为长度为 0 的字符串。

  取值范围：已存在的字段。

- FORCE QUOTE { column_name [, ...] | * }

  在 CSV COPY TO 模式下，强制在每个声明的字段周围对所有非 NULL 值都使用引号包围。NULL 输出不会被引号包围。

  取值范围：已存在的字段。

- BINARY

  使用二进制格式存储和读取，而不是以文本的方式。在二进制模式下，不能声明 DELIMITER，NULL，CSV 选项。指定 BINARY 类型后，不能再通过 option 或 copy_option 指定 CSV、FIXED、TEXT 等类型。

- CSV

  打开逗号分隔变量（CSV）模式。指定 CSV 类型后，不能再通过 option 或 copy_option 指定 BINARY、FIXED、TEXT 等类型。

- QUOTE [AS] 'quote_character'

  CSV 格式文件下的引号字符。

  缺省值：双引号。

说明

- quote 参数不能和分隔符、null 参数相同。
- quote 参数只能是单字节的字符。
- 推荐不可见字符作为 quote，例如 0x07，0x08，0x1b 等。

- ESCAPE [AS] 'escape_character'

CSV 格式下，用来指定逃逸字符，逃逸字符只能指定为单字节字符。

默认值为双引号。当与 quote 值相同时，会被替换为'\0'。

- EOL 'newline_character'

  指定导入导出数据文件换行符样式。

  取值范围：支持多字符换行符，但换行符不能超过 10 个字节。常见的换行符，如\r、\n、\r\n（设成 0x0D、0x0A、0x0D0A 效果是相同的），其他字符或字符串，如$、#。

说明

- EOL 参数只能用于 TEXT 格式的导入导出，不支持 CSV 格式和 FIXED 格式。为了兼容原有 EOL 参数，仍然支持导出 CSV 格式和 FIXED 格式时指定 EOL 参数为 0x0D 或 0x0D0A。
- EOL 参数不能和分隔符、null 参数相同。
- EOL 参数不能包含：.abcdefghijklmnopqrstuvwxyz0123456789。

- ENCODING 'encoding_name'

  指定文件编码格式名称。

  取值范围：有效的编码格式。

  缺省值：当前编码格式。

- IGNORE_EXTRA_DATA

  指定当数据源文件比外表定义列数多时，忽略行尾多出来的列。该参数只在数据导入过程中使用。

  若不使用该参数，在数据源文件比外表定义列数多，会显示如下错误信息。

```
extra data after last expected column
```

- COMPATIBLE_ILLEGAL_CHARS

  指定导入时对非法字符进行容错处理，非法字符转换后入库。不报错，不中断导入。此参数不支持 BINARY 格式，会报“cannot specify bulkload compatibility options in BINARY mode”错误信息。此参数仅对 COPY FROM 导入有效。

  若不使用该参数，导入时遇到非法字符进行报错，中断导入。

说明

导入非法字符容错规则如下：

（1）对于'\0'，容错后转换为空格；

（2）对于其他非法字符，容错后转换为问号；

（3）若 compatible_illegal_chars 为 true/on 标识，导入时对于非法字符进行容错处理，则若 NULL、DELIMITER、QUOTE、ESCAPE 设置为空格或问号则会通过如"illegal chars conversion may confuse COPY escape 0x20"等报错信息提示用户修改可能引起混淆的参数以避免导入错误。

- FILL_MISSING_FIELDS

  当数据加载时，若数据源文件中一行的最后一个字段缺失的处理方式。

取值范围：true/on，false/off。

缺省值：false/off。

须知

目前 COPY 指定此 Option 实际不会生效，即不会有相应的容错处理效果（不生效）。需要额外注意的是，打开此选项会导致解析器在数据库主节点数据解析阶段（即 COPY 错误表容错的涵盖范围）忽略此数据问题，而到数据库节点重新报错，从而使得 COPY 错误表（打开 LOG ERRORS REJECT LIMIT）在此选项打开的情况下无法成功捕获这类少列的数据异常。因此请不要指定此选项。

- DATE_FORMAT 'date_format_string'

  导入对于 DATE 类型指定格式。此参数不支持 BINARY 格式，会报“cannot specify bulkload compatibility options in BINARY mode”错误信息。此参数仅对 COPY FROM 导入有效。

  取值范围：合法 DATE 格式。可参考 11.5.8 时间和日期处理函数和操作符

说明

对于 DATE 类型内建为 TIMESTAMP 类型的数据库，在导入的时候，若需指定格式，可以参考下面的 timestamp_format 参数。

- TIME_FORMAT 'time_format_string'

  导入对于 TIME 类型指定格式。此参数不支持 BINARY 格式，会报“cannot specify bulkload compatibility options in BINARY mode”错误信息。此参数仅对 COPY FROM 导入有效。

  取值范围：合法 TIME 格式，不支持时区。可参考 11.5.8 时间和日期处理函数和操作符。

- TIMESTAMP_FORMAT 'timestamp_format_string'

  导入对于 TIMESTAMP 类型指定格式。此参数不支持 BINARY 格式，会报“cannot specify bulkload compatibility options in BINARY mode”错误信息。此参数仅对 COPY FROM 导入有效。

  取值范围：合法 TIMESTAMP 格式，不支持时区。可参考 11.5.8 时间和日期处理函数和操作符。

- SMALLDATETIME_FORMAT 'smalldatetime_format_string'

  导入对于 SMALLDATETIME 类型指定格式。此参数不支持 BINARY 格式，会报“cannot specify bulkload compatibility options in BINARY mode”错误信息。此参数仅对 COPY FROM 导入有效。

  取值范围：合法 SMALLDATETIME 格式。可参考 11.5.8 时间和日期处理函数和操作符。

COPY FROM 能够识别的特殊反斜杠序列如下所示。

- \b：反斜杠 （ASCII 8）
- \f：换页（ASCII 12）
- \n：换行符 （ASCII 10）
- \r：回车符 （ASCII 13）
- \t：水平制表符 （ASCII 9）
- \v：垂直制表符 （ASCII 11）
- \digits：反斜杠后面跟着一到三个八进制数，表示 ASCII 值为该数的字符。
- \xdigits：反斜杠 x 后面跟着一个或两个十六进制位声明指定数值编码的字符。

示例

```
--将 tpcds.ship_mode 中的数据拷贝到/home/vastbase/ds_ship_mode.dat 文件中。
vastbase=# COPY tpcds.ship_mode TO '/home/vastbase/ds_ship_mode.dat';

--将 tpcds.ship_mode 输出到 stdout。
vastbase=# COPY tpcds.ship_mode TO stdout;

--创建 tpcds.ship_mode_t1 表。
vastbase=# CREATE TABLE tpcds.ship_mode_t1
(
    SM_SHIP_MODE_SK          INTEGER             NOT NULL,
    SM_SHIP_MODE_ID          CHAR(16)            NOT NULL,
    SM_TYPE                 CHAR(30)                    ,
    SM_CODE                 CHAR(10)                    ,
    SM_CARRIER               CHAR(20)                   ,
    SM_CONTRACT              CHAR(20)
)
WITH (ORIENTATION = COLUMN,COMPRESSION=MIDDLE)
;

--从 stdin 拷贝数据到表 tpcds.ship_mode_t1。
vastbase=# COPY tpcds.ship_mode_t1 FROM stdin;

--从/home/vastbase/ds_ship_mode.dat 文件拷贝数据到表 tpcds.ship_mode_t1。
vastbase=# COPY tpcds.ship_mode_t1 FROM '/home/vastbase/ds_ship_mode.dat';

--从/home/vastbase/ds_ship_mode.dat 文件拷贝数据到表 tpcds.ship_mode_t1，使用参数如下：导入格式为 TEXT
（format 'text'),分隔符为'\t'(delimiter E'\t'),忽略多余列(ignore_extra_data 'true'),不指定转义(noescaping
'true'）。
vastbase=# COPY tpcds.ship_mode_t1 FROM '/home/vastbase/ds_ship_mode.dat' WITH(format 'text',
delimiter E'\t', ignore_extra_data 'true', noescaping 'true');

--从/home/vastbase/ds_ship_mode.dat 文件拷贝数据到表 tpcds.ship_mode_t1，使用参数如下：导入格式为 FIXED
（FIXED），指定定长格式（FORMATTER(SM_SHIP_MODE_SK(0, 2), SM_SHIP_MODE_ID(2,16), SM_TYPE(18,30),
SM_CODE(50,10), SM_CARRIER(61,20), SM_CONTRACT(82,20))），忽略多余列（ignore_extra_data），有数据头
（header）。
vastbase=# COPY tpcds.ship_mode_t1 FROM '/home/vastbase/ds_ship_mode.dat' FIXED
FORMATTER(SM_SHIP_MODE_SK(0, 2), SM_SHIP_MODE_ID(2,16), SM_TYPE(18,30), SM_CODE(50,10),
SM_CARRIER(61,20), SM_CONTRACT(82,20)) header ignore_extra_data;
```

```
--删除 tpcds.ship_mode_t1。
vastbase=# DROP TABLE tpcds.ship_mode_t1;
```

11.16.36. CREATE DATABASE

功能描述

创建一个新的数据库。缺省情况下新数据库将通过复制标准系统数据库 template0 来创建，且仅支持使用 template0 来创建。

注意事项

- 只有拥有 CREATEDB 权限的用户才可以创建新数据库，系统管理员默认拥有此权限。
- 不能在事务块中执行创建数据库语句。
- 在创建数据库过程中，若出现类似“could not initialize database directory”的错误提示，可能是由于文件系统上数据目录的权限不足或磁盘满等原因引起。

语法格式

```
CREATE DATABASE database_name
   [ [ WITH ] { [ OWNER [=] user_name ] |
            [ TEMPLATE [=] template ] |
            [ ENCODING [=] encoding ] |
            [ LC_COLLATE [=] lc_collate ] |
            [ LC_CTYPE [=] lc_ctype ] |
            [ DBCOMPATIBILITY [=] compatibilty_type ] |
            [ TABLESPACE [=] tablespace_name ] |
            [ CONNECTION LIMIT [=] connlimit ]}[...] ];
```

参数说明

- database_name

  数据库名称。

  取值范围：字符串，要符合标识符的命名规范。

- OWNER [=] user_name

  数据库所有者。缺省时，新数据库的所有者是当前用户。

  取值范围：已存在的用户名。

- TEMPLATE [=] template

  模板名。即从哪个模板创建新数据库。Vastbase 采用从模板数据库复制的方式来创建新的数据库。初始时，Vastbase 包含两个模板数据库 template0、template1，以及两个默认的用户数据库 vastbase 和 postgres。

取值范围：仅 template0。

- ENCODING [=] encoding

  指定数据库使用的字符编码，可以是字符串（如'SQL_ASCII'）、整数编号。

  不指定时，默认使用模版数据库的编码。模板数据库 template0 和 template1 的编码默认与操作系统环境相关。template1 不允许修改字符编码，因此若要变更编码，请使用 template0 创建数据库。

  常用取值：GBK、UTF8、Latin1。

须知

- 指定新的数据库字符集编码必须与所选择的本地环境中（LC_COLLATE 和 LC_CTYPE）的设置兼容。
- 当指定的字符编码集为 GBK 时，部分中文生僻字无法直接作为对象名。这是因为 GBK 第二个字节的编码范围在 0x40-0x7E 之间时，字节编码与 ASCII 字符@A-Z[\]^_`a-z{|}重叠。其中@[\]^_'{|}是数据库中的操作符，直接作为对象名时，会语法报错。例如“侅”字，GBK16 进制编码为 0x8240，第二个字节为 0x40，与 ASCII “@” 符号编码相同，因此无法直接作为对象名使用。如果确实要使用，可以在创建和访问对象时，通过增加双引号来规避这个问题。

  - LC_COLLATE [=] lc_collate

    指定新数据库使用的字符集。例如，通过 lc_collate = 'zh_CN.gbk'设定该参数。

    该参数的使用会影响到对字符串的排序顺序（如使用 ORDER BY 执行，以及在文本列上使用索引的顺序）。默认是使用模板数据库的排序顺序。

    取值范围：有效的排序类型。

  - LC_CTYPE [=] lc_ctype

    指定新数据库使用的字符分类。例如，通过 lc_ctype = 'zh_CN.gbk'设定该参数。该参数的使用会影响到字符的分类，如大写、小写和数字。默认是使用模板数据库的字符分类。

    取值范围：有效的字符分类。

  - DBCOMPATIBILITY [=] compatibilty_type

    指定兼容的数据库的类型。

    取值范围：A、B、C。分别表示兼容 O、MY 和 TD。

  - TABLESPACE [=] tablespace_name

    指定数据库对应的表空间。

    取值范围：已存在表空间名。

  - CONNECTION LIMIT [=] connlimit

    数据库可以接受的并发连接数。

须知

- 系统管理员不受此参数的限制。
- connlimit 数据库主节点单独统计，Vastbase 整体的连接数 = connlimit * 当前正常数据库主节点个数。

取值范围：>=-1 的整数。默认值为-1，表示没有限制。

有关字符编码的一些限制：

❖ 若区域设置为 C（或 POSIX），则允许所有的编码类型，但是对于其他的区域设置，字符编码必须和区域设置相同。

❖ 编码和区域设置必须匹配模板数据库，除了将 template0 当作模板。 因为其他数据库可能会包含不匹配指定编码的数据，或者可能包含排序顺序受 LC_COLLATE 和 LC_CTYPE 影响的索引。复制这些数据会导致在新数据库中的索引失效。template0 是不包含任何会受到影响的数据或者索引。

示例

```
--创建 jim 和 tom 用户。
vastbase=# CREATE USER jim PASSWORD 'Bigdata@123';
vastbase=# CREATE USER tom PASSWORD 'Bigdata@123';

--创建一个 GBK 编码的数据库 music（本地环境的编码格式必须也为 GBK）。
vastbase=# CREATE DATABASE music ENCODING 'GBK' template = template0;

--创建数据库 music2，并指定所有者为 jim。
vastbase=# CREATE DATABASE music2 OWNER jim;

--用模板 template0 创建数据库 music3，并指定所有者为 jim。
vastbase=# CREATE DATABASE music3 OWNER jim TEMPLATE template0;

--设置 music 数据库的连接数为 10。
vastbase=# ALTER DATABASE music CONNECTION LIMIT= 10;

--将 music 名称改为 music4。
vastbase=# ALTER DATABASE music RENAME TO music4;

--将数据库 music2 的所属者改为 tom。
vastbase=# ALTER DATABASE music2 OWNER TO tom;

--设置 music3 的表空间为 PG_DEFAULT。
vastbase=# ALTER DATABASE music3 SET TABLESPACE PG_DEFAULT;

--关闭在数据库 music3 上缺省的索引扫描。
vastbase=# ALTER DATABASE music3 SET enable_indexscan TO off;

--重置 enable_indexscan 参数。
vastbase=# ALTER DATABASE music3 RESET enable_indexscan;
```

```
--删除数据库。
vastbase=# DROP DATABASE music2;
vastbase=# DROP DATABASE music3;
vastbase=# DROP DATABASE music4;

--删除 jim 和 tom 用户。
vastbase=# DROP USER jim;
vastbase=# DROP USER tom;

--创建兼容 TD 格式的数据库。
vastbase=# CREATE DATABASE td_compatible_db DBCOMPATIBILITY 'C';

--创建兼容 ORA 格式的数据库。
vastbase=# CREATE DATABASE ora_compatible_db DBCOMPATIBILITY 'A';

--删除兼容 TD、ORA 格式的数据库。
vastbase=# DROP DATABASE td compatible db;
vastbase=# DROP DATABASE ora_compatible_db;
```

相关链接

11.16.2ALTER DATABASE，11.16.63DROP DATABASE

优化建议

- create database

  事务中不支持创建 database。

- ENCODING　LC_COLLATE　LC_CTYPE

  当新建数据库 Encoding、LC-Collate 或 LC_Ctype 与模板数据库（SQL_ASCII）不匹配（为 'GBK' /'UTF8'/'LATIN1'）时，必须指定 template [=] template0。

11.16.37. CREATE DATA SOURCE

功能描述

创建一个新的外部数据源对象，该对象用于定义 Vastbase 要连接的目标库信息。

注意事项

- Data Source 名称在数据库中需唯一，遵循标识符命名规范，长度限制为 63 字节，过长则会被截断。
- 只有系统管理员或初始用户才有权限创建 Data Source 对象。且创建该对象的用户为其默认属主。

❖ 当在 OPTIONS 中出现 password 选项时，需要保证 Vastbase 每个节点的$GAUSSHOME/bin 目录下存在 datasource.key.cipher 和 datasource.key.rand 文件，如果不存在这两个文件，请使用 vb_guc 工具生成并使用 gs_ssh 工具发布到 Vastbase 每个节点的$GAUSSHOME/bin 目录下。

语法格式

```
CREATE DATA SOURCE src_name
   [TYPE 'type_str']
   [VERSION {'version_str' | NULL}]
   [OPTIONS (optname 'optvalue' [, ...])];
```

参数说明

❖ src_name

新建 Data Source 对象的名称，需在数据库内部唯一。

取值范围：字符串，要符标识符的命名规范。

❖ TYPE

新建 Data Source 对象的类型，可缺省。

取值范围：空串或非空字符串。

❖ VERSION

新建 Data Source 对象的版本号，可缺省或 NULL 值。

取值范围：空串或非空字符串或 NULL。

❖ OPTIONS

Data Source 对象的选项字段，创建时可省略，如若指定，其关键字如下：

– optname

选项名称。

取值范围：dsn, username, password, encoding。不区分大小写。

■ dsn 对应 odbc 配置文件中的 DSN。

■ username/password 对应连接目标库的用户名和密码。

Vastbase 在后台会对用户输入的 username/password 加密以保证安全性。该加密所需密钥文件需要使用 vb_guc 工具生成并使用 gs_ssh 工具发布到 Vastbase 每个节点的 $GAUSSHOME/bin 目录下。username/password 不应当包含'encryptOpt'前缀，否则会被认为是加密后的密文。

■ encoding 表示与目标库交互的字符串编码方式（含发送的 SQL 语句和返回的字符类型数据），此处创建对象时不检查 encoding 取值的合法性，

能否正确编解码取决于用户提供的编码方式是否在数据库本身支持的字符编码范围内。

- optvalue

  选项值。

  取值范围：空或者非空字符串。

示例

```
--创建一个空的 Data Source 对象，不含任何信息。
vastbase=# CREATE DATA SOURCE ds_test1;

--创建一个 Data Source 对象，含 TYPE 信息，VERSION 为 NULL。
vastbase=# CREATE DATA SOURCE ds_test2 TYPE 'MPPDB' VERSION NULL;

--创建一个 Data Source 对象，仅含 OPTIONS。
vastbase=#  CREATE DATA SOURCE ds_test3 OPTIONS (dsn 'Vastbase', encoding 'utf8');

--创建一个 Data Source 对象，含 TYPE, VERSION, OPTIONS。
vastbase=# CREATE DATA SOURCE ds_test4 TYPE 'unknown' VERSION '11.2.3' OPTIONS (dsn 'Vastbase', username
'userid', password 'pwd@123456', encoding '');

--删除 Data Source 对象。
vastbase=# DROP DATA SOURCE ds_test1;
vastbase=# DROP DATA SOURCE ds_test2;
vastbase=# DROP DATA SOURCE ds_test3;
vastbase=# DROP DATA SOURCE ds_test4;
```

相关链接

11.16.3ALTER DATA SOURCE, 11.16.64DROP DATA SOURCE

11.16.38. CREATE DIRECTORY

功能描述

使用 CREATE DIRECTORY 语句创建一个目录对象，该目录对象定义了服务器文件系统上目录的别名，用于存放用户使用的数据文件。

注意事项

- 默认只允许初始化用户创建，如果开启 enable_access_server_directory（可参考 enable_access_server_directory），sysadmin 权限的用户也可以创建目录。
- 创建用户默认拥有此路径的 READ 和 WRITE 操作权限。
- 目录的默认 owner 为创建 directory 的用户。
- 以下路径禁止创建：

- 路径含特殊字符。
- 路径是相对路径。
- 路径是符号连接。

❖ 创建目录时会进行以下合法性校验：

- 创建时会检查添加路径是否为操作系统实际存在路径，如不存在会提示用户使用风险。
- 创建时会校验数据库初始化（vastbase）用户对于添加路径的权限(即操作系统目录权限，读/写/执行 - R/W/X)，如果权限不全，会提示用户使用风险。

❖ 在 Vastbase 环境下用户指定的路径需要用户保证各节点上路径的一致性，否则在不同节点上执行会产生找不到路径的问题。

语法格式

```
CREATE [OR REPLACE] DIRECTORY directory_name
AS 'path_name';
```

参数说明

❖ directory_name

目录名称。

取值范围：字符串，要符标识符的命名规范。

❖ path_name

操作系统的路径。

取值范围： 有效的操作系统路径。

示例

```
--创建目录。
vastbase=# CREATE OR REPLACE DIRECTORY  dir  as '/tmp/';
```

相关链接

11.16.5ALTER DIRECTORY，11.16.65DROP DIRECTORY

11.16.39. CREATE FUNCTION

功能描述

创建一个函数。

注意事项

❖ 如果创建函数时参数或返回值带有精度，不进行精度检测。

❖ 创建函数时，函数定义中对表对象的操作建议都显式指定模式，否则可能会导致函数执行异常。

❖ 在创建函数时，函数内部通过 SET 语句设置 current_schema 和 search_path 无效。执行完函数 search_path 和 current_schema 与执行函数前的 search_path 和 current_schema 保持一致。

❖ 如果函数参数中带有出参，SELECT 调用函数必须缺省出参，CALL 调用函数必须指定出参，对于调用重载的带有 PACKAGE 属性的函数，CALL 调用函数可以缺省出参，具体信息参见 11.16.28CALL 的示例。

❖ 兼容 Postgresql 风格的函数或者带有 PACKAGE 属性的函数支持重载。在指定 REPLACE 的时候，如果参数个数、类型、返回值有变化，不会替换原有函数，而是会建立新的函数。

❖ SELECT 调用可以指定不同参数来进行同名函数调用。由于语法不支持调用不带有 PACKAGE 属性的同名函数。

❖ 在创建 function 时，不能在 avg 函数外面嵌套其他 agg 函数，或者其他系统函数。

❖ 新创建的函数默认会给 PUBLIC 授予执行权限（详见 11.16.88GRANT）。用户可以选择收回 PUBLIC 默认执行权限，然后根据需要将执行权限授予其他用户，为了避免出现新函数能被所有人访问的时间窗口，应在一个事务中创建函数并且设置函数执行权限。

语法格式

❖ 兼容 PostgreSQL 风格的创建自定义函数语法。

```
CREATE [ OR REPLACE  ] FUNCTION function name
    ( [  { argname [ argmode  ] argtype [  { DEFAULT  | :=  | =  } expression  ]}  [, ...]  ] )
    [ RETURNS rettype [ DETERMINISTIC  ]  | RETURNS TABLE (  { column_name column_type  }
[, ...] )]
    LANGUAGE lang_name
    [
      {IMMUTABLE  | STABLE  | VOLATILE }
       | {SHIPPABLE | NOT SHIPPABLE}
       | WINDOW
       | [ NOT  ] LEAKPROOF
       | {CALLED ON NULL INPUT  | RETURNS NULL ON NULL INPUT | STRICT }
       | {[ EXTERNAL  ] SECURITY INVOKER | [ EXTERNAL  ] SECURITY DEFINER | AUTHID DEFINER  |
AUTHID CURRENT_USER}
       | {fenced | not fenced}
       | {PACKAGE}

       | COST execution_cost
       | ROWS result_rows
       | SET configuration_parameter { {TO | =} value | FROM CURRENT }}
     ][...]
    {
       AS 'definition'
    }
```

❖ O 风格的创建自定义函数的语法。

```
CREATE [ OR REPLACE  ] FUNCTION function_name
    ( [  { argname [ argmode  ] argtype [  { DEFAULT | := | =  } expression  ] }  [, ...]  ] )
    RETURN rettype [ DETERMINISTIC  ]
    [
```

```
      {IMMUTABLE  | STABLE  | VOLATILE }
      | {SHIPPABLE | NOT SHIPPABLE}
      | {PACKAGE}
      | {FENCED | NOT FENCED}
      | [ NOT  ] LEAKPROOF
      | {CALLED ON NULL INPUT | RETURNS NULL ON NULL INPUT | STRICT }
      | {[ EXTERNAL  ] SECURITY INVOKER  | [ EXTERNAL  ] SECURITY DEFINER |
AUTHID DEFINER | AUTHID CURRENT_USER
}
      | COST execution_cost
      | ROWS result_rows
      | SET configuration_parameter { {TO | =} value  | FROM CURRENT

   ][...]

   {
     IS  | AS
} plsql_body
/
```

参数说明

- function_name

  要创建的函数名称（可以用模式修饰）。

  取值范围：字符串，要符合标识符的命名规范。

- argname

  函数参数的名称。

  取值范围：字符串，要符合标识符的命名规范。

- argmode

  函数参数的模式。

  取值范围：IN，OUT，INOUT 或 VARIADIC。缺省值是 IN。只有 OUT 模式的参数后面能跟 VARIADIC。并且 OUT 和 INOUT 模式的参数不能用在 RETURNS TABLE 的函数定义中。

说明

VARIADIC 用于声明数组类型的参数。

- argtype

  函数参数的类型。

- expression

  参数的默认表达式。

- rettype

  函数返回值的数据类型。

如果存在 OUT 或 IN OUT 参数，可以省略 RETURNS 子句。如果存在，该子句必须和输出参数所表示的结果类型一致：如果有多个输出参数，则为 RECORD，否则与单个输出参数的类型相同。

SETOF 修饰词表示该函数将返回一个集合，而不是单独一项。

- column_name

  字段名称。

- column_type

  字段类型。

- definition

  一个定义函数的字符串常量，含义取决于语言。它可以是一个内部函数名称、一个指向某个目标文件的路径、一个 SQL 查询、一个过程语言文本。

- LANGUAGE lang_name

  用以实现函数的语言的名称。可以是 SQL，internal，或者是用户定义的过程语言名称。为了保证向下兼容，该名称可以用单引号（包围）。若采用单引号，则引号内必须为大写。

- WINDOW

  表示该函数是窗口函数。替换函数定义时不能改变 WINDOW 属性。

须知

自定义窗口函数只支持 LANGUAGE 是 internal，并且引用的内部函数必须是窗口函数。

- IMMUTABLE

  表示该函数在给出同样的参数值时总是返回同样的结果。

- STABLE

  表示该函数不能修改数据库，对相同参数值，在同一次表扫描里，该函数的返回值不变，但是返回值可能在不同 SQL 语句之间变化。

- VOLATILE

  表示该函数值可以在一次表扫描内改变，因此不会做任何优化。

- PACKAGE

  表示该函数是否支持重载。PostgreSQL 风格的函数本身就支持重载，此参数主要是针对其它风格的函数。

  - 不允许 package 函数和非 package 函数重载或者替换。
  - package 函数不支持 VARIADIC 类型的参数。

- 不允许修改函数的 package 属性。

❖ LEAKPROOF

指出该函数的参数只包括返回值。LEAKPROOF 只能由系统管理员设置。

❖ CALLED ON NULL INPUT

表明该函数的某些参数是 NULL 的时候可以按照正常的方式调用。该参数可以省略。

❖ RETURNS NULL ON NULL INPUT

STRICT

STRICT 用于指定如果函数的某个参数是 NULL，此函数总是返回 NULL。如果声明了这个参数，当有 NULL 值参数时该函数不会被执行；而只是自动返回一个 NULL 结果。

RETURNS NULL ON NULL INPUT 和 STRICT 的功能相同。

❖ EXTERNAL

目的是和 SQL 兼容，是可选的，这个特性适合于所有函数，而不仅是外部函数。

❖ SECURITY INVOKER

AUTHID CURRENT_USER

表明该函数将带着调用它的用户的权限执行。该参数可以省略。

SECURITY INVOKER 和 AUTHID CURRENT_USER 的功能相同。

❖ SECURITY DEFINER

AUTHID DEFINER

声明该函数将以创建它的用户的权限执行。

AUTHID DEFINER 和 SECURITY DEFINER 的功能相同。

❖ COST execution_cost

用来估计函数的执行成本。

execution_cost 以 cpu_operator_cost 为单位。

取值范围：正数

❖ ROWS result_rows

估计函数返回的行数。用于函数返回的是一个集合。

取值范围：正数，默认值是 1000 行。

❖ configuration_parameter

- value

  把指定的数据库会话参数值设置为给定的值。如果 value 是 DEFAULT 或者 RESET，则在新的会话中使用系统的缺省设置。OFF 关闭设置。

取值范围：字符串

- DEFAULT
- OFF
- RESET

指定默认值。

- from current

  取当前会话中的值设置为 configuration_parameter 的值。

- plsql_body

  PL/SQL 存储过程体。

须知

当在函数体中创建用户时，日志中会记录密码的明文。因此不建议用户在函数体中创建用户。

示例

```
--定义函数为 SQL 查询。
vastbase=# CREATE FUNCTION func_add_sql(integer, integer) RETURNS integer
    AS 'select $1 + $2;'
    LANGUAGE SQL
    IMMUTABLE
    RETURNS NULL ON NULL INPUT;

--利用参数名用 PL/pgsql 自增一个整数。
vastbase=# CREATE OR REPLACE FUNCTION func_increment_plsql(i integer) RETURNS integer AS $$
        BEGIN
                RETURN i + 1;
        END;
$$ LANGUAGE plpgsql;

--返回 RECORD 类型
CREATE OR REPLACE FUNCTION compute(i int, out result_1 bigint, out result_2 bigint)
returns SETOF RECORD
as $$
begin
    result_1 = i + 1;
    result_2 = i * 10;
return next;
end;
$$language plpgsql;

--返回一个包含多个输出参数的记录。
vastbase=# CREATE FUNCTION func_dup_sql(in int, out f1 int, out f2 text)
    AS $$ SELECT $1, CAST($1 AS text) || ' is text' $$
    LANGUAGE SQL;

vastbase=# SELECT * FROM func_dup_sql(42);
```

```
--计算两个整数的和，并返回结果。若果输入为 null，则返回 null。
vastbase=# CREATE FUNCTION func_add_sql2(num1 integer, num2 integer) RETURN integer
AS
BEGIN
RETURN num1 + num2;
END;
/
--修改函数 add 的执行规则为 IMMUTABLE，即参数不变时返回相同结果。
vastbase=# ALTER FUNCTION func_add_sql2(INTEGER, INTEGER) IMMUTABLE;

--将函数 add 的名称修改为 add_two_number。
vastbase=# ALTER FUNCTION func_add_sql2(INTEGER, INTEGER) RENAME TO add_two_number;

--将函数 add 的属者改为 vastbase。
vastbase=# ALTER FUNCTION add_two_number(INTEGER, INTEGER) OWNER TO vastbase;

--删除函数。
vastbase=# DROP FUNCTION add two number;
vastbase=# DROP FUNCTION func_increment_sql;
vastbase=# DROP FUNCTION func_dup_sql;
vastbase=# DROP FUNCTION func_increment_plsql;
vastbase=# DROP FUNCTION func_add_sql;
```

相关链接

11.16.6ALTER FUNCTION，11.16.66DROP FUNCTION

11.16.40. CREATE GROUP

功能描述

创建一个新用户组。

注意事项

CREATE GROUP 是 CREATE ROLE 的别名，非 SQL 标准语法，不推荐使用，建议用户直接使用 CREATE ROLE 替代。

语法格式

```
CREATE GROUP group_name [ [ WITH ] option [ ... ] ] [ ENCRYPTED | UNENCRYPTED ] { PASSWORD | IDENTIFIED BY } { 'password' | DISABLE };
```

其中可选项 option 子句语法为：

```
{SYSADMIN | NOSYSADMIN}
   | {AUDITADMIN | NOAUDITADMIN}
   | {CREATEDB | NOCREATEDB}
   | {USEFT | NOUSEFT}
   | {CREATEROLE | NOCREATEROLE}
   | {INHERIT | NOINHERIT}
```

```
  | {LOGIN | NOLOGIN}
  | {REPLICATION | NOREPLICATION}
  | {INDEPENDENT | NOINDEPENDENT}
  | {VCADMIN | NOVCADMIN}
  | CONNECTION LIMIT connlimit
  | VALID BEGIN 'timestamp'
  | VALID UNTIL 'timestamp'
  | RESOURCE POOL 'respool'
  | PERM SPACE 'spacelimit'
  | TEMP SPACE 'tmpspacelimit'
  | SPILL SPACE 'spillspacelimit'
  | IN ROLE role_name [, ...]
  | IN GROUP role_name [, ...]
  | ROLE role_name [, ...]
  | ADMIN rol e_name [, ...]
  | USER role_name [, ...]
  | SYSID uid
  | DEFAULT TABLESPACE tablespace_name
  | PROFILE DEFAULT
  | PROFILE profile_name
  | PGUSER
```

参数说明

请参考 CREATE ROLE 的参数说明。

相关链接

11.16.7ALTER GROUP，11.16.67DROP GROUP，11.16.44CREATE ROLE

11.16.41. CREATE INDEX

功能描述

在指定的表上创建索引。

索引可以用来提高数据库查询性能，但是不恰当的使用将导致数据库性能下降。建议仅在匹配如下某条原则时创建索引：

- 经常执行查询的字段。
- 在连接条件上创建索引，对于存在多字段连接的查询，建议在这些字段上建立组合索引。例如，select * from t1 join t2 on t1.a=t2.a and t1.b=t2.b，可以在 t1 表上的 a，b 字段上建立组合索引。
- where 子句的过滤条件字段上（尤其是范围条件）。
- 在经常出现在 order by、group by 和 distinct 后的字段。

在分区表上创建索引与在普通表上创建索引的语法不太一样，使用时请注意，如分区表上不支持并行创建索引、不支持创建部分索引、不支持 NULL FIRST 特性。

注意事项

- 索引自身也占用存储空间、消耗计算资源，创建过多的索引将对数据库性能造成负面影响（尤其影响数据导入的性能，建议在数据导入后再建索引）。因此，仅在必要时创建索引。
- 索引定义里的所有函数和操作符都必须是 immutable 类型的，即它们的结果必须只能依赖于它们的输入参数，而不受任何外部的影响（如另外一个表的内容或者当前时间）。这个限制可以确保该索引的行为是定义良好的。要在一个索引上或 WHERE 中使用用户定义函数，请把它标记为 immutable 类型函数。
- 在分区表上创建唯一索引时，索引项中必须包含分布列和所有分区键。
- 列存表支持的 PSORT 和 B-tree 索引都不支持创建表达式索引、部分索引和唯一索引。
- 列存表支持的 GIN 索引支持创建表达式索引，但表达式不能包含空分词、空列和多列，不支持创建部分索引和唯一索引。

语法格式

- 在表上创建索引。

```
CREATE [ UNIQUE ] INDEX [ [schemaname.]index_name ] ON table_name [ USING method ]
    ({ { column_name | ( expression ) } [ COLLATE collation ] [ opclass ] [ ASC | DESC ] [ NULLS
{ FIRST | LAST } ] }[, ...] )
    [ WITH ( {storage_parameter = value} [, ... ] ) ]
    [ TABLESPACE tablespace_name ]
    [ WHERE predicate ];
```

- 在分区表上创建索引。

```
CREATE [ UNIQUE ] INDEX [ [schemaname.]index_name ] ON table_name [ USING method ]
    ( {{ column_name | ( expression ) } [ COLLATE collation ] [ opclass ] [ ASC | DESC ] [ NULLS
LAST ] }[, ...] )
    LOCAL [ ( { PARTITION index_partition_name [ TABLESPACE index_partition_tablespace ] }
[, ...] ) ]
    [ WITH ( { storage_parameter = value } [, ...] ) ]
    [ TABLESPACE tablespace_name ];
```

参数说明

- UNIQUE

  创建唯一性索引，每次添加数据时检测表中是否有重复值。如果插入或更新的值会引起重复的记录时，将导致一个错误。

  目前只有行存表 B-tree 索引支持唯一索引。

- schema_name

  模式的名称。

  取值范围：已存在模式名。

- index_name

  要创建的索引名，不能包含模式名，索引的模式与表相同。

取值范围：字符串，要符合标识符的命名规范。

- table_name

  需要为其创建索引的表的名称，可以用模式修饰。

  取值范围：已存在的表名。

- USING method

  指定创建索引的方法。

  取值范围：

  - btree：B-tree 索引使用一种类似于 B+树的结构来存储数据的键值，通过这种结构能够快速的查找索引。btree 适合支持比较查询以及查询范围。
  - gin：GIN 索引是倒排索引，可以处理包含多个键的值（比如数组）。
  - gist：Gist 索引适用于几何和地理等多维数据类型和集合数据类型。
  - Psort：Psort 索引。针对列存表进行局部排序索引。

  行存表支持的索引类型：btree（行存表缺省值）、gin、gist。列存表支持的索引类型：Psort（列存表缺省值）、btree、gin。

说明

列存表对 GIN 索引支持仅限于对于 tsvector 类型的支持，即创建列存 GIN 索引入参需要为 to_tsvector 函数（的返回值）。此方法为 GIN 索引比较普遍的使用方式。

- column_name

  表中需要创建索引的列的名称（字段名）。

  如果索引方式支持多字段索引，可以声明多个字段。最多可以声明 32 个字段。

- expression

  创建一个基于该表的一个或多个字段的表达式索引，通常必须写在圆括弧中。如果表达式有函数调用的形式，圆括弧可以省略。

  表达式索引可用于获取对基本数据的某种变形的快速访问。比如，一个在 upper(col)上的函数索引将允许 WHERE upper(col) = 'JIM'子句使用索引。

  在创建表达式索引时，如果表达式中包含 IS NULL 子句，则这种索引是无效的。此时，建议用户尝试创建一个部分索引。

- COLLATE collation

  COLLATE 子句指定列的排序规则（该列必须是可排列的数据类型）。如果没有指定，则使用默认的排序规则。

- opclass

操作符类的名称。对于索引的每一列可以指定一个操作符类，操作符类标识了索引那一列的使用的操作符。例如一个 B-tree 索引在一个四字节整数上可以使用 int4_ops；这个操作符类包括四字节整数的比较函数。实际上对于列上的数据类型默认的操作符类是足够用的。操作符类主要用于一些有多种排序的数据。例如，用户想按照绝对值或者实数部分排序一个复数。能通过定义两个操作符类然后当建立索引时选择合适的类。

- ASC

  指定按升序排序 （默认）。

- DESC

  指定按降序排序。

- NULLS FIRST

  指定空值在排序中排在非空值之前，当指定 DESC 排序时，本选项为默认的。

- NULLS LAST

  指定空值在排序中排在非空值之后，未指定 DESC 排序时，本选项为默认的。

- WITH ({storage_parameter = value} [, ...])

  指定索引方法的存储参数。

  取值范围：

  只有 GIN 索引支持 FASTUPDATE，GIN_PENDING_LIST_LIMIT 参数。GIN 和 Psort 之外的索引都支持 FILLFACTOR 参数。

  - FILLFACTOR

    一个索引的填充因子（fillfactor）是一个介于 10 和 100 之间的百分数。

    取值范围：10~100

  - FASTUPDATE

    GIN 索引是否使用快速更新。

    取值范围：ON，OFF

    默认值：ON

  - GIN_PENDING_LIST_LIMIT

    当 GIN 索引启用 fastupdate 时，设置该索引 pending list 容量的最大值。

    取值范围：64~INT_MAX，单位 KB。

    默认值：gin_pending_list_limit 的默认取决于 GUC 中 gin_pending_list_limit 的值（默认为 4MB）

- TABLESPACE tablespace_name

指定索引的表空间，如果没有声明则使用默认的表空间。

取值范围：已存在的表空间名。

- WHERE predicate

  创建一个部分索引。部分索引是一个只包含表的一部分记录的索引，通常是该表中比其他部分数据更有用的部分。例如，有一个表，表里包含已记账和未记账的定单，未记账的定单只占表的一小部分而且这部分是最常用的部分，此时就可以通过只在未记账部分创建一个索引来改善性能。另外一个可能的用途是使用带有 UNIQUE 的 WHERE 强制一个表的某个子集的唯一性。

  取值范围：predicate 表达式只能引用表的字段，它可以使用所有字段，而不仅是被索引的字段。目前，子查询和聚集表达式不能出现在 WHERE 子句里。

- PARTITION index_partition_name

  索引分区的名称。

  取值范围：字符串，要符合标识符的命名规范。

- TABLESPACE index_partition_tablespace

  索引分区的表空间。

  取值范围：如果没有声明，将使用分区表索引的表空间 index_tablespace。

示例

```
--创建表 tpcds.ship_mode_t1。
vastbase=# create schema tpcds;
vastbase=# CREATE TABLE tpcds.ship_mode_t1
(
    SM_SHIP_MODE_SK          INTEGER             NOT NULL,
    SM_SHIP_MODE_ID          CHAR(16)            NOT NULL,
    SM_TYPE                 CHAR(30)                    ,
    SM_CODE                 CHAR(10)                    ,
    SM_CARRIER               CHAR(20)                    ,
    SM_CONTRACT              CHAR(20)
)
;

--在表 tpcds.ship_mode_t1 上的 SM_SHIP_MODE_SK 字段上创建普通索引。
vastbase=# CREATE UNIQUE INDEX ds_ship_mode_t1_index1 ON tpcds.ship_mode_t1(SM_SHIP_MODE_SK);

--在表 tpcds.ship_mode_t1 上的 SM_SHIP_MODE_SK 字段上创建指定 B-tree 索引。
vastbase=# CREATE INDEX ds_ship_mode_t1_index4 ON tpcds.ship_mode_t1 USING btree(SM_SHIP_MODE_SK);

--在表 tpcds.ship_mode_t1 上 SM_CODE 字段上创建表达式索引。
vastbase=# CREATE INDEX ds_ship_mode_t1_index2 ON tpcds.ship_mode_t1(SUBSTR(SM_CODE,1 ,4));

--在表 tpcds.ship_mode_t1 上的 SM_SHIP_MODE_SK 字段上创建 SM_SHIP_MODE_SK 大于 10 的部分索引。
vastbase=# CREATE UNIQUE INDEX ds_ship_mode_t1_index3 ON tpcds.ship_mode_t1(SM_SHIP_MODE_SK) WHERE
SM_SHIP_MODE_SK>10;
```

```
--重命名一个现有的索引。
vastbase=# ALTER INDEX tpcds.ds_ship_mode_t1_index1 RENAME TO ds_ship_mode_t1_index5;

--设置索引不可用。
vastbase=# ALTER INDEX tpcds.ds_ship_mode_t1_index2 UNUSABLE;

--重建索引。
vastbase=# ALTER INDEX tpcds.ds_ship_mode_t1_index2 REBUILD;

--删除一个现有的索引。
vastbase=# DROP INDEX tpcds.ds_ship_mode_t1_index2;

--删除表。
vastbase=# DROP TABLE tpcds.ship mode t1;

--创建表空间。
vastbase=# CREATE TABLESPACE example1 RELATIVE LOCATION 'tablespace1/tablespace_1';
vastbase=# CREATE TABLESPACE example2 RELATIVE LOCATION 'tablespace2/tablespace_2';
vastbase=# CREATE TABLESPACE example3 RELATIVE LOCATION 'tablespace3/tablespace_3';
vastbase=# CREATE TABLESPACE example4 RELATIVE LOCATION 'tablespace4/tablespace_4';
--创建表tpcds.customer_address_p1。
vastbase=# CREATE TABLE tpcds.customer_address_p1
(
    CA_ADDRESS_SK           INTEGER             NOT NULL,
    CA_ADDRESS_ID           CHAR(16)            NOT NULL,
    CA_STREET_NUMBER         CHAR(10)                    ,
    CA_STREET_NAME          VARCHAR(60)                  ,
    CA_STREET_TYPE          CHAR(15)                    ,
    CA_SUITE_NUMBER          CHAR(10)                   ,
    CA_CITY                VARCHAR(60)                 ,
    CA_COUNTY               VARCHAR(30)                ,
    CA_STATE                CHAR(2)                    ,
    CA_ZIP                 CHAR(10)                    ,
    CA_COUNTRY              VARCHAR(20)                 ,
    CA_GMT_OFFSET           DECIMAL(5,2)                ,
    CA_LOCATION_TYPE         CHAR(20)
)
TABLESPACE example1
PARTITION BY RANGE(CA_ADDRESS_SK)
(
   PARTITION p1 VALUES LESS THAN (3000),
   PARTITION p2 VALUES LESS THAN (5000) TABLESPACE example1,
   PARTITION p3 VALUES LESS THAN (MAXVALUE) TABLESPACE example2
)
ENABLE ROW MOVEMENT;
--创建分区表索引ds_customer_address_p1_index1，不指定索引分区的名称。
vastbase=# CREATE INDEX ds_customer_address_p1_index1 ON tpcds.customer_address_p1(CA_ADDRESS_SK)
LOCAL;
--创建分区表索引ds_customer_address_p1_index2，并指定索引分区的名称。
vastbase=# CREATE INDEX ds_customer_address_p1_index2 ON tpcds.customer_address_p1(CA_ADDRESS_SK)
LOCAL
(
    PARTITION CA_ADDRESS_SK_index1,
    PARTITION CA_ADDRESS_SK_index2 TABLESPACE example3,
```

```
   PARTITION CA_ADDRESS_SK_index3 TABLESPACE example4
)
TABLESPACE example2;

--修改分区表索引CA_ADDRESS_SK_index2的表空间为example1。
vastbase=# ALTER INDEX tpcds.ds_customer_address_p1_index2 MOVE PARTITION CA_ADDRESS_SK_index2
TABLESPACE example1;

--修改分区表索引CA_ADDRESS_SK_index3的表空间为example2。
vastbase=# ALTER INDEX tpcds.ds_customer_address_p1_index2 MOVE PARTITION CA_ADDRESS_SK_index3
TABLESPACE example2;

--重命名分区表索引。
vastbase=# ALTER INDEX tpcds.ds_customer_address_p1_index2 RENAME PARTITION CA_ADDRESS_SK_index1 TO
CA_ADDRESS_SK_index4;

--删除索引和分区表。
vastbase=# DROP INDEX tpcds.ds_customer_address_p1_index1;
vastbase=# DROP INDEX tpcds.ds_customer_address_p1_index2;
vastbase=# DROP TABLE tpcds.customer_address_p1;
--删除表空间。
vastbase=# DROP TABLESPACE example1;
vastbase=# DROP TABLESPACE example2;
vastbase=# DROP TABLESPACE example3;
vastbase=# DROP TABLESPACE example4;

--创建列存表以及列存表GIN索引。
vastbase=# create table cgin_create_test(a int, b text) with (orientation = column);
CREATE TABLE
vastbase=# create index cgin_test on cgin_create_test using gin(to_tsvector('ngram', b));
CREATE INDEX
```

相关链接

11.16.8ALTER INDEX，11.16.68DROP INDEX

优化建议

- create index

  建议仅在匹配如下条件之一时创建索引：

  - 经常执行查询的字段。
  - 在连接条件上创建索引，对于存在多字段连接的查询，建议在这些字段上建立组合索引。例如，select * from t1 join t2 on t1.a=t2.a and t1.b=t2.b，可以在 t1 表上的 a，b 字段上建立组合索引。
  - where 子句的过滤条件字段上（尤其是范围条件）。
  - 在经常出现在 order by、group by 和 distinct 后的字段。

  约束限制：

- 分区表上不支持创建部分索引、不支持 NULL FIRST 特性。
- 在分区表上创建唯一索引时，索引项中必须包含分布列和所有分区键。

11.16.42. CREATE ROW LEVLEL SECURITY POLICY

功能描述

对表创建行访问控制策略。

当对表创建了行访问控制策略，只有打开该表的行访问控制开关(ALTER TABLE ... ENABLE ROW LEVEL SECURITY)，策略才能生效。否则不生效。

当前行访问控制影响数据表的读取操作(SELECT、UPDATE、DELETE)，暂不影响数据表的写入操作(INSERT、MERGE INTO)。表所有者或系统管理员可以在 USING 子句中创建表达式，在客户端执行数据表读取操作时，数据库后台在查询重写阶段会将满足条件的表达式拼接并应用到执行计划中。针对数据表的每一条元组，当 USING 表达式返回 TRUE 时，元组对当前用户可见，当 USING 表达式返回 FALSE 或 NULL 时，元组对当前用户不可见。

行访问控制策略名称是针对表的，同一个数据表上不能有同名的行访问控制策略；对不同的数据表，可以有同名的行访问控制策略。

行访问控制策略可以应用到指定的操作(SELECT、UPDATE、DELETE、ALL)，ALL 表示会影响 SELECT、UPDATE、DELETE 三种操作；定义行访问控制策略时，若未指定受影响的相关操作，默认为 ALL。

行访问控制策略可以应用到指定的用户(角色)，也可应用到全部用户(PUBLIC)；定义行访问控制策略时，若未指定受影响的用户，默认为 PUBLIC。

注意事项

- 支持对行存表、行存分区表、列存表、列存分区表、复制表、unlogged 表、hash 表定义行访问控制策略。
- 不支持外表、临时表定义行访问控制策略。
- 不支持对视图定义行访问控制策略。
- 同一张表上可以创建多个行访问控制策略，一张表最多创建 100 个行访问控制策略。
- 系统管理员不受行访问控制影响，可以查看表的全量数据。
- 通过 SQL 语句、视图、函数、存储过程查询包含行访问控制策略的表，都会受影响。

语法格式

```
CREATE [ ROW LEVEL SECURITY ] POLICY policy_name ON table_name
    [ AS { PERMISSIVE | RESTRICTIVE } ]
    [ FOR { ALL | SELECT | INSERT | UPDATE | DELETE } ]
    [ TO { role_name | PUBLIC | CURRENT_USER | SESSION_USER } [, ...] ]
    USING ( using_expression )
```

参数说明

- policy_name

  行访问控制策略名称，同一个数据表上行访问控制策略名称不能相同。

- table_name

  行访问控制策略的表名。

- command

  当前行访问控制影响的 SQL 操作，可指定操作包括：ALL、SELECT、UPDATE、DELETE。当未指定时，ALL 为默认值，涵盖 SELECT、UPDATE、DELETE 操作。

  当 command 为 SELECT 时，SELECT 类操作受行访问控制的影响，只能查看到满足条件(using_expression 返回值为 TRUE)的元组数据，受影响的操作包括 SELECT，UPDATE ... RETURNING，DELETE ... RETURNING。

  当 command 为 UPDATE 时，UPDATE 类操作受行访问控制的影响，只能更新满足条件(using_expression 返回值为 TRUE)的元组数据，受影响的操作包括 UPDATE, UPDATE ... RETURNING, SELECT ... FOR UPDATE/SHARE。

  当 command 为 DELETE 时，DELETE 类操作受行访问控制的影响，只能删除满足条件(using_expression 返回值为 TRUE)的元组数据，受影响的操作包括 DELETE, DELETE ... RETURNING。

  行访问控制策略与适配的 SQL 语法关系参加下表：

表 11-52. ROW LEVLEL SECURITY 策略与适配 SQL 语法关系

COMMAND	SELECT/ALL POLICY	UPDATE/ALL POLICY	DELETE/ALL POLICY
SELECT	Existing row	No	No
SELECT FOR UPDATE/SHARE	Existing row	Existing row	No
UPDATE	No	Existing row	No
UPDATE RETURNING	Existing row	Existing row	No
DELETE	No	No	Existing row
DELETE RETURNING	Existing row	No	Existing row

- role_name

  行访问控制影响的数据库用户。

  当未指定时，PUBLIC 为默认值，PUBLIC 表示影响所有数据库用户，可以指定多个受影响的数据库用户。

须知

系统管理员不受行访问控制特性影响。

- using_expression

  行访问控制的表达式（返回 boolean 值）。

  条件表达式中不能包含 AGG 函数和窗口（WINDOW）函数。在查询重写阶段，如果数据表的行访问控制开关打开，满足条件的表达式会添加到计划树中。针对数据表的每条元组，会进行表达式计算，只有表达式返回值为 TRUE 时，行数据对用户才可见（SELECT、UPDATE、DELETE）；当表达式返回 FALSE 时，该元组对当前用户不可见，用户无法通过 SELECT 语句查看此元组，无法通过 UPDATE 语句更新此元组，无法通过 DELETE 语句删除此元组。

示例

```
--创建用户 alice
vastbase=# CREATE ROLE alice PASSWORD 'Vastbase@123';

--创建用户 bob
vastbase=# CREATE ROLE bob PASSWORD 'Vastbase@123';

--创建数据表 all_data
vastbase=# CREATE TABLE all_data(id int, role varchar(100), data varchar(100));

--向数据表插入数据
vastbase=# INSERT INTO all_data VALUES(1, 'alice', 'alice data');
vastbase=# INSERT INTO all_data VALUES(2, 'bob', 'bob data');
vastbase=# INSERT INTO all_data VALUES(3, 'peter', 'peter data');

--将表 all_data 的读取权限赋予 alice 和 bob 用户
vastbase=# GRANT SELECT ON all_data TO alice, bob;

--打开行访问控制策略开关
vastbase=# ALTER TABLE all_data ENABLE ROW LEVEL SECURITY;

--创建行访问控制策略，当前用户只能查看用户自身的数据
vastbase=# CREATE ROW LEVEL SECURITY POLICY all_data_rls ON all_data USING(role = CURRENT_USER);

--查看表 all_data 相关信息
vastbase=# \d+ all_data
                        Table "public.all_data"
 Column |         Type          | Modifiers | Storage  | Stats target | Description
--------+-----------------------+-----------+----------+--------------+-------------
 id     | integer               |           | plain    |              |
 role   | character varying(100) |          | extended |              |
 data   | character varying(100) |          | extended |              |
Row Level Security Policies:
    POLICY "all_data_rls"
      USING (((role)::name = "current_user"()))
```

```
Has OIDs: no
Options: orientation=row, compression=no, enable_rowsecurity=true

--当前用户执行 SELECT 操作
vastbase=# SELECT * FROM all_data;
 id | role  |    data
----+-------+------------
  1 | alice | alice data
  2 | bob   | bob data
  3 | peter | peter data
(3 rows)

vastbase=# EXPLAIN(COSTS OFF) SELECT * FROM all_data;
       QUERY PLAN
----------------------------
Seq Scan on all_data
(1 row)

--给用户登录权限
vastbase=# ALTER USER alice LOGIN;

--切换至 alice 用户执行 SELECT 操作
vastbase=# SELECT * FROM all_data;
 id | role  |    data
----+-------+------------
  1 | alice | alice data
(1 row)

vastbase=# EXPLAIN(COSTS OFF) SELECT * FROM all_data;
                     QUERY PLAN
-----------------------------------------------------------------
Seq Scan on all_data
       Filter: ((role)::name = 'alice'::name)
 Notice: This query is influenced by row level security feature
(3 rows)
```

相关链接

11.16.70DROP ROW LEVEL SECURITY POLICY

11.16.43. CREATE PROCEDURE

功能描述

创建一个新的存储过程。

注意事项

- 如果创建存储过程时参数或返回值带有精度，不进行精度检测。
- 创建存储过程时，存储过程定义中对表对象的操作建议都显示指定模式，否则可能会导致存储过程执行异常。

- 在创建存储过程时，存储过程内部通过 SET 语句设置 current_schema 和 search_path 无效。执行完函数 search_path 和 current_schema 与执行函数前的 search_path 和 current_schema 保持一致。
- 如果存储过程参数中带有出参，SELECT 调用存储过程必须缺省出参，CALL 调用存储过程调用非重载函数时必须指定出参，对于重载的 package 函数，out 参数可以缺省，具体信息参见 11.16.28CALL 的示例。
- 存储过程指定 package 属性时支持重载。
- 在创建 procedure 时，不能在 avg 函数外面嵌套其他 agg 函数，或者其他系统函数。

语法格式

```
vastbase=# CREATE [ OR REPLACE ] PROCEDURE procedure_name
   [ ( {[ argmode ] [ argname ] argtype [ { DEFAULT | := | = } expression ]}[,...]) ]
   [
     { IMMUTABLE | STABLE | VOLATILE }
     | { SHIPPABLE | NOT SHIPPABLE }
     | {PACKAGE}
     | [ NOT ] LEAKPROOF
     | { CALLED ON NULL INPUT | RETURNS NULL ON NULL INPUT | STRICT }
     | {[ EXTERNAL ] SECURITY INVOKER | [ EXTERNAL ] SECURITY DEFINER | AUTHID DEFINER | AUTHID
CURRENT_USER}
     | COST execution_cost
     | ROWS result_rows
     | SET configuration_parameter { [ TO | = ] value | FROM CURRENT }
   ][ ... ]
 { IS | AS }
plsql_body
/
```

参数说明

- OR REPLACE

  当存在同名的存储过程时，替换原来的定义。

- procedure_name

  创建的存储过程名称，可以带有模式名。

  取值范围：字符串，要符合标识符的命名规范。

- argmode

  参数的模式。

其他说明

vastbase 数据库支持在存储过程含 commit&rollback 以及自治事物。示例如下：

```
创建含 commit&rollback 的存储过程
```

```
vastbase2=# create table test_pro(id int,a text,c char(20));
vastbase2=# create or replace procedure test_pro1 is
vastbase2$# begin
vastbase2$# insert into test_pro values(1,'b1','c1');
vastbase2$# commit;
vastbase2$# insert into test_pro values(2,'b2','c3');
vastbase2$# insert into test_pro values(3,'b3','c3');
vastbase2$# insert into test_pro values(4,'b4','c4');
vastbase2$# insert into test_pro values(5,'b5','c5');
vastbase2$# insert into test_pro values(6,'b6','c6');
vastbase2$# commit;
vastbase2$# delete from test_pro where id =2;
vastbase2$# rollback;
vastbase2$# delete from test_pro where id =2;
vastbase2$# commit;
vastbase2$# end;
vastbase2$# /
CREATE PROCEDURE
vastbase2=#call test_pro1();
vastbase2=#select * from test_pro;
 id | a   |  c
----+----+-------------
  1 | b1  | c1
  1 | b3  | c3
  1 | b4  | c4
  1 | b5  | c5
  1 | b6  | c6
(5 rows)
```

创建启用自治事物的存储过程

```
vastbase2=# create table at_tb2(id int,val varchar(20));
vastbase2=# create or replace procedure at_test3(i int) AS DECLARE
vastbase2$# PRAGMA AUTONOMOUS_TRANSACTION;
vastbase2$# BEGIN
vastbase2$# START TRANSACTION;
vastbase2$# insert into at_tb2 values(1,'before s1');
vastbase2$# insert into at_tb2 values(2,'after s1');
vastbase2$# if i>10 then
vastbase2$# rollback;
vastbase2$# else
vastbase2$# commit;
vastbase2$# end if;
vastbase2$# end;
vastbase2$# /
CREATE PROCEDURE
vastbase2=# call at_test3(11);
   at_test3
----------------
(1 row)
vastbase2=# select * from at_tb2;
id | val
---+-----
 (0 row)
vastbase2=# call at_test3(6);
   at_test3
----------------
```

```
(1 row)
vastbase2=# select * from at_tb2;
id | val
---+-----
 1  |before s1
 2  |after s1
(2 rows)

--删除表 test_pro.
vastbase2=# DROP TABLE test_pro;

--删除存储过程 test_pro1;
vastbase2=# DROP PROCEDURE test_pro1;

--删除表 at_tb2.
vastbase2=# DROP TABLE at_tb2;

--删除存储过程 at_test3;
vastbase2=# DROP PROCEDURE at_test3;
```

须知

VARIADIC 用于声明数组类型的参数。

取值范围： IN，OUT，INOUT 或 VARIADIC。缺省值是 IN。只有 OUT 模式的参数后面能跟 VARIADIC。并且 OUT 和 INOUT 模式的参数不能用在 RETURNS TABLE 的过程定义中。

- argname

  参数的名称。

  取值范围：字符串，要符合标识符的命名规范。

- argtype

  参数的数据类型。

  取值范围：可用的数据类型。

- IMMUTABLE、STABLE 等

  行为约束可选项。各参数的功能与 CREATE FUNCTION 类似，详细说明见 11.16.39CREATE FUNCTION

- plsql_body

  PL/SQL 存储过程体。

须知

当在存储过程体中进行创建用户等涉及用户密码相关操作时，系统表及 csv 日志中会记录密码的明文。因此不建议用户在存储过程体中进行涉及用户密码的相关操作。

说明

argument_name 和 argmode 的顺序没有严格要求，推荐按照 argument_name、argmode、argument_type 的顺序使用。

相关链接

11.16.71DROP PROCEDURE

优化建议

- analyse | analyze
  - 不支持在事务或匿名块中执行 analyze 。
  - 不支持在函数或存储过程中执行 analyze 操作。

11.16.44. CREATE ROLE

功能描述

创建角色。

角色是拥有数据库对象和权限的实体。在不同的环境中角色可以认为是一个用户，一个组或者兼顾两者。

注意事项

- 在数据库中添加一个新角色，角色无登录权限。
- 创建角色的用户必须具备 CREATE ROLE 的权限或者是系统管理员。

语法格式

```
CREATE ROLE role_name [ [ WITH ] option [ ... ] ] [ ENCRYPTED | UNENCRYPTED ] { PASSWORD | IDENTIFIED BY } { 'password' | DISABLE };
```

其中角色信息设置子句 option 语法为：

```
{SYSADMIN | NOSYSADMIN}
   | {AUDITADMIN | NOAUDITADMIN}
   | {SSOADMIN | NOSSOADMIN}
   | {CREATEDB | NOCREATEDB}
   | {USEFT | NOUSEFT}
   | {CREATEROLE | NOCREATEROLE}
   | {INHERIT | NOINHERIT}
   | {LOGIN | NOLOGIN}
   | {REPLICATION | NOREPLICATION}
```

```
| {INDEPENDENT | NOINDEPENDENT}
| {VCADMIN | NOVCADMIN}
| CONNECTION LIMIT connlimit
| VALID BEGIN 'timestamp'
| VALID UNTIL 'timestamp'
| RESOURCE POOL 'respool'
| PERM SPACE 'spacelimit'
| TEMP SPACE 'tmpspacelimit'
| SPILL SPACE 'spillspacelimit'
| IN ROLE role_name [, ...]
| IN GROUP role_name [, ...]
| ROLE role_name [, ...]
| ADMIN rol e_name [, ...]
| USER role_name [, ...]
| SYSID uid
| DEFAULT TABLESPACE tablespace_name
| PROFILE DEFAULT
| PROFILE profile_name
| PGUSER
```

参数说明

- role_name

  角色名称。

  取值范围：字符串，要符合标识符的命名规范，且最多为 63 个字符。若超过 63 个字符，数据库会截断并保留前 63 个字符当做角色名称。在创建角色时，数据库的时候会给出提示信息。

说明

标识符需要为字母、下划线、数字（0-9）或美元符号（$），且必须以字母（a-z）或下划线（_）开头。

- password

  登录密码。

  密码规则如下：

  - 密码默认不少于 8 个字符。
  - 不能与用户名及用户名倒序相同。
  - 至少包含大写字母（A-Z），小写字母（a-z），数字（0-9），非字母数字字符（限定为~!@#$%^&*()-_=+\|[{}];:,<.>/?）四类字符中的三类字符。

  取值范围：字符串。

- DISABLE

  默认情况下，用户可以更改自己的密码，除非密码被禁用。要禁用用户的密码，请指定 DISABLE。禁用某个用户的密码后，将从系统中删除该密码，此类用户只能通过外部认证来连接数据库，例如：kerberos 认证。只有管理员才能启用或禁用密码。普通用户不能禁用初始用户的密码。要启用密码，请运行 ALTER USER 并指定密码。

- ENCRYPTED | UNENCRYPTED

  控制密码存储在系统表里的口令是否加密。（如果没有指定，那么缺省的行为由配置参数 password_encryption 控制。）按照产品安全要求，密码必须加密存储，所以，UNENCRYPTED 在 Vastbase 中禁止使用。因为系统无法对指定的加密口令字符串进行解密，所以如果目前的口令字符串已经是用 SHA256 加密的格式，则会继续照此存放，而不管是否声明了 ENCRYPTED 或 UNENCRYPTED。这样就允许在 dump/restore 的时候重新加载加密的口令。

- SYSADMIN | NOSYSADMIN

  决定一个新角色是否为“系统管理员”，具有 SYSADMIN 属性的角色拥有系统最高权限。

  缺省为 NOSYSADMIN。

- AUDITADMIN | NOAUDITADMIN

  定义角色是否有审计管理属性。

  缺省为 NOAUDITADMIN。

- SSOADMIN | NOSSOADMIN

  定义角色是否有安全管理属性。

  缺省为 NOSSOADMIN。

- CREATEDB | NOCREATEDB

  决定一个新角色是否能创建数据库。

  新角色没有创建数据库的权限。

  缺省为 NOCREATEDB。

- USEFT | NOUSEFT

  该参数为保留参数，暂未启用。

- CREATEROLE | NOCREATEROLE

  决定一个角色是否可以创建新角色（也就是执行 CREATE ROLE 和 CREATE USER）。 一个拥有 CREATEROLE 权限的角色也可以修改和删除其他角色。

  缺省为 NOCREATEROLE。

- INHERIT | NOINHERIT

  这些子句决定一个角色是否“继承”它所在组的角色的权限。不推荐使用。

- LOGIN | NOLOGIN

  具有 LOGIN 属性的角色才可以登录数据库。一个拥有 LOGIN 属性的角色可以认为是一个用户。

  缺省为 NOLOGIN。

- REPLICATION | NOREPLICATION

定义角色是否允许流复制或设置系统为备份模式。REPLICATION 属性是特定的角色，仅用于复制。

缺省为 NOREPLICATION。

- INDEPENDENT | NOINDEPENDENT

定义私有、独立的角色。具有 INDEPENDENT 属性的角色，管理员对其进行的控制、访问的权限被分离，具体规则如下：

  - 未经 INDEPENDENT 角色授权，系统管理员无权对其表对象进行增、删、查、改、拷贝、授权操作。
  - 未经 INDEPENDENT 角色授权，系统管理员和拥有 CREATEROLE 属性的安全管理员无权修改 INDEPENDENT 角色的继承关系。
  - 系统管理员无权修改 INDEPENDENT 角色的表对象的属主。
  - 系统管理员和拥有 CREATEROLE 属性的安全管理员无权去除 INDEPENDENT 角色的 INDEPENDENT 属性。
  - 系统管理员和拥有 CREATEROLE 属性的安全管理员无权修改 INDEPENDENT 角色的数据库口令，INDEPENDENT 角色需管理好自身口令，口令丢失无法重置。
  - 管理员属性用户不允许定义修改为 INDEPENDENT 属性。

- CONNECTION LIMIT

声明该角色可以使用的并发连接数量。

须知

- 系统管理员不受此参数的限制。
- connlimit 数据库主节点单独统计，Vastbase 整体的连接数 = connlimit * 当前正常数据库主节点个数。

取值范围：整数，>=-1，缺省值为-1，表示没有限制。

- VALID BEGIN

设置角色生效的时间戳。如果省略了该子句，角色无有效开始时间限制。

- VALID UNTIL

设置角色失效的时间戳。如果省略了该子句，角色无有效结束时间限制。

- RESOURCE POOL

设置角色使用的 resource pool 名称，该名称属于系统表：pg_resource_pool

- PERM SPACE

设置用户使用空间的大小。

- TEMP SPACE

设置用户临时表存储空间限额。

- SPILL SPACE

  设置用户算子落盘空间限额。

- IN ROLE

  新角色立即拥有 IN ROLE 子句中列出的一个或多个现有角色拥有的权限。不推荐使用。

- IN GROUP

  IN GROUP 是 IN ROLE 过时的拼法。不推荐使用。

- ROLE

  ROLE 子句列出一个或多个现有的角色，它们将自动添加为这个新角色的成员，拥有新角色所有的权限。

- ADMIN

  ADMIN 子句类似 ROLE 子句，不同的是 ADMIN 后的角色可以把新角色的权限赋给其他角色。

- USER

  USER 子句是 ROLE 子句过时的拼法。

- SYSID

  SYSID 子句将被忽略，无实际意义。

- DEFAULT TABLESPACE

  DEFAULT TABLESPACE 子句将被忽略，无实际意义。

- PROFILE

  PROFILE 子句将被忽略，无实际意义。

- PGUSER

  当前版本该属性没有实际意义，仅为了语法的前向兼容而保留。

示例

```
--创建一个角色，名为 manager，密码为 Bigdata@123。
vastbase=# CREATE ROLE manager IDENTIFIED BY 'Bigdata@123';

--创建一个角色，从 2015 年 1 月 1 日开始生效，到 2026 年 1 月 1 日失效。
vastbase=# CREATE ROLE miriam WITH LOGIN PASSWORD 'Bigdata@123' VALID BEGIN '2015-01-01' VALID UNTIL
'2026-01-01';

--修改角色 manager 的密码为 abcd@123。
vastbase=# ALTER ROLE manager IDENTIFIED BY 'abcd@123' REPLACE 'Bigdata@123';

--修改角色 manager 为系统管理员。
vastbase=# ALTER ROLE manager SYSADMIN;
```

```
--删除角色manager。
vastbase=# DROP ROLE manager;

--删除角色miriam。
vastbase=# DROP ROLE miriam;
```

相关链接

11.16.107SET ROLE，11.16.10ALTER ROLE，11.16.72DROP ROLE，11.16.88GRANT

11.16.45. CREATE SCHEMA

功能描述

创建模式。

访问命名对象时可以使用模式名作为前缀进行访问，如果无模式名前缀，则访问当前模式下的命名对象。创建命名对象时也可用模式名作为前缀修饰。

另外，CREATE SCHEMA 可以包括在新模式中创建对象的子命令，这些子命令和那些在创建完模式后发出的命令没有任何区别。如果使用了 AUTHORIZATION 子句，则所有创建的对象都将被该用户所拥有。

注意事项

- 只要用户对当前数据库有 CREATE 权限，就可以创建模式。
- 系统管理员在普通用户同名 schema 下创建的对象，所有者为 schema 的同名用户（非系统管理员）。

语法格式

- 根据指定的名称创建模式。

```
CREATE SCHEMA schema_name
   [ AUTHORIZATION user_name ] [ schema_element [ ... ] ];
```

- 根据用户名创建模式。

```
CREATE SCHEMA AUTHORIZATION user_name [ schema_element [ ... ] ];
```

参数说明

- schema_name

  模式名称。

须知

模式名不能和当前数据库里其他的模式重名。

模式的名称不可以“pg_”开头。

取值范围：字符串，要符合标识符的命名规范。

- AUTHORIZATION user_name

  指定模式的所有者。当不指定 schema_name 时，把 user_name 当作模式名，此时 user_name 只能是角色名。

  取值范围：已存在的用户名/角色名。

- schema_element

  在模式里创建对象的 SQL 语句。目前仅支持 CREATE TABLE、CREATE VIEW、CREATE INDEX、CREATE PARTITION、GRANT 子句。

  子命令所创建的对象都被 AUTHORIZATION 子句指定的用户所拥有。

说明

如果当前搜索路径上的模式中存在同名对象时，需要明确指定引用对象所在的模式。可以通过命令 SHOW SEARCH_PATH 来查看当前搜索路径上的模式。

示例

```
--创建一个角色 role1。
vastbase=# CREATE ROLE role1 IDENTIFIED BY 'Bigdata@123';

-- 为用户 role1 创建一个同名 schema，子命令创建的表 films 和 winners 的拥有者为 role1。
vastbase=# CREATE SCHEMA AUTHORIZATION role1
    CREATE TABLE films (title text, release date, awards text[])
    CREATE VIEW winners AS
    SELECT title, release FROM films WHERE awards IS NOT NULL;

--删除 schema。
vastbase=# DROP SCHEMA role1 CASCADE;
--删除用户。
vastbase=# DROP USER role1 CASCADE;
```

相关链接

11.16.12ALTER SCHEMA，11.16.73DROP SCHEMA

11.16.46. CREATE SEQUENCE

功能描述

CREATE SEQUENCE 用于向当前数据库里增加一个新的序列。序列的 Owner 为创建此序列的用户。

注意事项

- Sequence 是一个存放等差数列的特殊表。这个表没有实际意义，通常用于为行或者表生成唯一的标识符。
- 如果给出一个模式名，则该序列就在给定的模式中创建，否则会在当前模式中创建。序列名必须和同一个模式中的其他序列、表、索引、视图或外表的名称不同。
- 创建序列后，在表中使用序列的 nextval()函数和 generate_series(1,N)函数对表插入数据，请保证 nextval 的可调用次数大于等于 N+1 次，否则会因为 generate_series()函数会调用 N+1 次而导致报错。

语法格式

```
CREATE SEQUENCE name [ INCREMENT [ BY ] increment ]
    [ MINVALUE minvalue | NO MINVALUE | NOMINVALUE ] [ MAXVALUE maxvalue | NO MAXVALUE | NOMAXVALUE]
    [ START [ WITH ] start ] [ CACHE cache ] [ [ NO ] CYCLE | NOCYCLE ]
    [ OWNED BY { table_name.column_name | NONE } ];
```

参数说明

- name

  将要创建的序列名称。

  取值范围: 仅可以使用小写字母（a~z）、 大写字母（A~Z），数字和特殊字符"#"，"_"，"$"的组合。

- increment

  指定序列的步长。一个正数将生成一个递增的序列，一个负数将生成一个递减的序列。

  缺省值为 1。

- MINVALUE minvalue | NO MINVALUE| NOMINVALUE

  执行序列的最小值。如果没有声明 minvalue 或者声明了 NO MINVALUE，则递增序列的缺省值为 1，递减序列的缺省值为-2^{63}-1。NOMINVALUE 等价于 NO MINVALUE

- MAXVALUE maxvalue | NO MAXVALUE| NOMAXVALUE

  执行序列的最大值。如果没有声明 maxvalue 或者声明了 NO MAXVALUE，则递增序列的缺省值为 2^{63}-1，递减序列的缺省值为-1。NOMAXVALUE 等价于 NO MAXVALUE

- start

  指定序列的起始值。缺省值：对于递增序列为 minvalue，递减序列为 maxvalue。

- cache

  为了快速访问，而在内存中预先存储序列号的个数。

  缺省值为 1，表示一次只能生成一个值，也就是没有缓存。

说明

不建议同时定义 cache 和 maxvalue 或 minvalue。因为定义 cache 后不能保证序列的连续性，可能会产生空洞，造成序列号段浪费。

- CYCLE

  用于使序列达到 maxvalue 或者 minvalue 后可循环并继续下去。

  如果声明了 NO CYCLE，则在序列达到其最大值后任何对 nextval 的调用都会返回一个错误。

  NOCYCLE 的作用等价于 NO CYCLE。

  缺省值为 NO CYCLE。

  若定义序列为 CYCLE，则不能保证序列的唯一性。

- OWNED BY-

  将序列和一个表的指定字段进行关联。这样，在删除那个字段或其所在表的时候会自动删除已关联的序列。关联的表和序列的所有者必须是同一个用户，并且在同一个模式中。需要注意的是，通过指定 OWNED BY，仅仅是建立了表的对应列和 sequence 之间关联关系，并不会在插入数据时在该列上产生自增序列。

  缺省值为 OWNED BY NONE，表示不存在这样的关联。

须知

通过 OWNED BY 创建的 Sequence 不建议用于其他表，如果希望多个表共享 Sequence，该 Sequence 不应该从属于特定表。

示例

创建一个名为 serial 的递增序列，从 101 开始：

```
vastbase=# CREATE SEQUENCE serial
 START 101
 CACHE 20;
```

从序列中选出下一个数字：

```
vastbase=# SELECT nextval('serial');
 nextval
 ---------
     101
```

从序列中选出下一个数字：

```
vastbase=# SELECT nextval('serial');
 nextval
 ---------
     102
```

创建与表关联的序列：

```
vastbase=# CREATE TABLE customer_address
(
```

```
    ca_address_sk             integer               not null,
    ca_address_id             char(16)              not null,
    ca_street_number          char(10)                      ,
    ca_street_name            varchar(60)                   ,
    ca_street_type            char(15)                      ,
    ca_suite_number           char(10)                      ,
    ca_city                   varchar(60)                   ,
    ca_county                 varchar(30)                   ,
    ca_state                  char(2)                       ,
    ca_zip                    char(10)                      ,
    ca_country                varchar(20)                   ,
    ca_gmt_offset             decimal(5,2)                  ,
    ca_location_type          char(20)
);

vastbase=# CREATE SEQUENCE serial1
 START 101
 CACHE 20
OWNED BY customer_address.ca_address_sk;
--删除序列
vastbase=# DROP SEQUENCE serial cascade;
vastbase=# DROP SEQUENCE serial1 cascade;
--删除表
vastbase=# DROP TABLE customer_address;
```

相关链接

11.16.74DROP SEQUENCE，11.16.13ALTER SEQUENCE

11.16.47. CREATE SYNONYM

功能描述

创建一个同义词对象。同义词是数据库对象的别名，用于记录与其他数据库对象名间的映射关系，用户可以使用同义词访问关联的数据库对象。

注意事项

- 定义同义词的用户成为其所有者。
- 若指定模式名称，则同义词在指定模式中创建。否则，在当前模式创建。
- 支持通过同义词访问的数据库对象包括：表、视图、函数和存储过程。
- 使用同义词时，用户需要具有对关联对象的相应权限。
- 支持使用同义词的 DML 语句包括：SELECT、INSERT、UPDATE、DELETE、EXPLAIN、CALL。
- 不支持关联函数或存储过程的 CREATE SYNONYM 语句出现在存储过程中，建议存储过程中使用系统表 pg_synonym 中已存在的同义词对象。

语法格式

```
CREATE [ OR REPLACE ] SYNONYM synonym_name
   FOR object_name;
```

参数说明

- synonym

  创建的同义词名字，可以带模式名。

  取值范围：字符串，要符合标识符的命名规范。

- object_name

  关联的对象名字，可以带模式名。

  取值范围：字符串，要符合标识符的命名规范。

说明

object_name 可以是不存在的对象名称。

示例

```
--创建模式 ot。
vastbase=# CREATE SCHEMA ot;

--创建表 ot.t1 及其同义词 t1。
vastbase=# CREATE TABLE ot.t1(id int, name varchar2(10));
vastbase=# CREATE OR REPLACE SYNONYM t1 FOR ot.t1;

--使用同义词 t1。
vastbase=# SELECT * FROM t1;
vastbase=# INSERT INTO t1 VALUES (1, 'ada'), (2, 'bob');
vastbase=# UPDATE t1 SET t1.name = 'cici' WHERE t1.id = 2;

--创建同义词 v1 及其关联视图 ot.v_t1。
vastbase=# CREATE SYNONYM v1 FOR ot.v_t1;
vastbase=# CREATE VIEW ot.v_t1 AS SELECT * FROM ot.t1;

--使用同义词 v1。
vastbase=# SELECT * FROM v1;

--创建重载函数 ot.add 及其同义词 add。
vastbase=# CREATE OR REPLACE FUNCTION ot.add(a integer, b integer) RETURNS integer AS
$$
SELECT $1 + $2
$$
LANGUAGE sql;

vastbase=# CREATE OR REPLACE FUNCTION ot.add(a decimal(5,2), b decimal(5,2)) RETURNS decimal(5,2) AS
$$
SELECT $1 + $2
$$
```

```
LANGUAGE sql;

vastbase=# CREATE OR REPLACE SYNONYM add FOR ot.add;

--使用同义词 add。
vastbase=# SELECT add(1,2);
vastbase=# SELECT add(1.2,2.3);

--创建存储过程 ot.register 及其同义词 register。
vastbase=# CREATE PROCEDURE ot.register(n_id integer, n_name varchar2(10))
SECURITY INVOKER
AS
BEGIN
   INSERT INTO ot.t1 VALUES(n_id, n_name);
END;
/

vastbase=# CREATE OR REPLACE SYNONYM register FOR ot.register;

--使用同义词 register，调用存储过程。
vastbase=# CALL register(3,'mia');

--删除同义词。
vastbase=# DROP SYNONYM t1;
vastbase=# DROP SYNONYM IF EXISTS v1;
vastbase=# DROP SYNONYM IF EXISTS add;
vastbase=# DROP SYNONYM register;
vastbase=# DROP SCHEMA ot CASCADE;
```

相关链接

11.16.15ALTER SYNONYM，11.16.75DROP SYNONYM

11.16.48. CREATE TABLE

功能描述

在当前数据库中创建一个新的空白表，该表由命令执行者所有。

注意事项

- 列存表支持的数据类型请参考 11.3.16 列存表支持的数据类型。
- 创建列存表的数量建议不超过 1000 个。
- 表中的主键约束和唯一约束必须包含分布列。
- 如果在建表过程中数据库系统发生故障，系统恢复后可能无法自动清除之前已创建的、大小为 0 的磁盘文件。此种情况出现概率小，不影响数据库系统的正常运行。
- 列存表的表级约束只支持 PARTIAL CLUSTER KEY，不支持主外键等表级约束。
- 列存表的字段约束只支持 NULL、NOT NULL 和 DEFAULT 常量值。

❖ 列存表支持 delta 表，受参数 enable_delta_store 控制是否开启，受参数 deltarow_threshold 控制进入 delta 表的阀值。

❖ 使用 JDBC 时，支持通过 PrepareStatement 对 DEFAUTL 值进行参数化设置。

语法格式

❖ 创建表。

```
CREATE [ [ GLOBAL | LOCAL ] { TEMPORARY | TEMP } | UNLOGGED ] TABLE [ IF NOT EXISTS ] table_name
    ({ column_name data_type [ compress_mode ] [ COLLATE collation ] [ column_constraint [ ... ] ]
       | table_constraint
       | LIKE source_table [ like_option [...] ] }
       [, ... ])
    [ WITH ( {storage_parameter = value} [, ... ] ) ]
    [ ON COMMIT { PRESERVE ROWS | DELETE ROWS | DROP } ]
    [ COMPRESS | NOCOMPRESS ]
    [ TABLESPACE tablespace_name ];
```

– 其中列约束 column_constraint 为：

```
[ CONSTRAINT constraint_name ]
{ NOT NULL |
  NULL |
  CHECK ( expression ) |
  DEFAULT default_expr |
  UNIQUE index_parameters |
  PRIMARY KEY index_parameters }
[ DEFERRABLE | NOT DEFERRABLE | INITIALLY DEFERRED | INITIALLY IMMEDIATE ]
```

– 其中列的压缩可选项 compress_mode 为：

```
{ DELTA | PREFIX | DICTIONARY | NUMSTR | NOCOMPRESS }
```

– 其中表约束 table_constraint 为：

```
[ CONSTRAINT constraint_name ]
{ CHECK ( expression ) |
  UNIQUE ( column_name [, ... ] ) index_parameters |
  PRIMARY KEY ( column_name [, ... ] ) index_parameters |
  PARTIAL CLUSTER KEY ( column_name [, ... ] ) }
[ DEFERRABLE | NOT DEFERRABLE | INITIALLY DEFERRED | INITIALLY IMMEDIATE ]
```

– 其中 like 选项 like_option 为：

```
{ INCLUDING | EXCLUDING } { DEFAULTS | CONSTRAINTS | INDEXES | STORAGE | COMMENTS | PARTITION
| RELOPTIONS | DISTRIBUTION | ALL }
```

其中索引参数 index_parameters 为：

```
[ WITH ( {storage_parameter = value} [, ... ] ) ]
[ USING INDEX TABLESPACE tablespace_name ]
```

参数说明

❖ UNLOGGED

如果指定此关键字，则创建的表为非日志表。在非日志表中写入的数据不会被写入到预写日志中，这样就会比普通表快很多。但是非日志表在冲突、执行操作系统重启、强制重启、切断电

源操作或异常关机后会被自动截断，会造成数据丢失的风险。非日志表中的内容也不会被复制到备服务器中。在非日志表中创建的索引也不会被自动记录。

使用场景：非日志表不能保证数据的安全性，用户应该在确保数据已经做好备份的前提下使用，例如系统升级时进行数据的备份。

故障处理：当异常关机等操作导致非日志表上的索引发生数据丢失时，用户应该对发生错误的索引进行重建。

- GLOBAL | LOCAL

  创建临时表时可以在 TEMP 或 TEMPORARY 前指定 GLOBAL 或 LOCAL 关键字。目前这两个关键字的设立，仅是为了兼容 SQL 标准，实际上无论指定 GLOBAL 还是 LOCAL，Vastbase 都会创建本地临时表。

- TEMPORARY | TEMP

  如果指定 TEMP 或 TEMPORARY 关键字，则创建的表为临时表。临时表只在当前会话可见，本会话结束后会自动删除。因此，在除当前会话连接的数据库节点故障时，仍然可以在当前会话上创建和使用临时表。由于临时表只在当前会话创建，对于涉及对临时表操作的 DDL 语句，会产生 DDL 失败的报错。因此，建议 DDL 语句中不要对临时表进行操作。TEMP 和 TEMPORARY 等价。

须知

- 临时表通过每个会话独立的以 pg_temp 开头的 schema 来保证只对当前会话可见，因此，不建议用户在日常操作中手动删除以 pg_temp，pg_toast_temp 开头的 schema。
- 如果建表时不指定 TEMPORARY/TEMP 关键字，而指定表的 schema 为当前会话的 pg_temp_开头的 schema，则此表会被创建为临时表。

- IF NOT EXISTS

  如果已经存在相同名称的表，不会报出错误，而会发出通知，告知通知此表已存在。

- table_name

  要创建的表名。

- column_name

  新表中要创建的字段名。

- data_type

  字段的数据类型。

- compress_mode

  表字段的压缩选项，当前仅对行存表有效。该选项指定表字段优先使用的压缩算法。

  取值范围：DELTA、PREFIX、DICTIONARY、NUMSTR、NOCOMPRESS

❖ COLLATE collation

COLLATE 子句指定列的排序规则（该列必须是可排列的数据类型）。如果没有指定，则使用默认的排序规则。

❖ LIKE source_table [like_option ...]

LIKE 子句声明一个表，新表自动从这个表中继承所有字段名及其数据类型和非空约束。

新表与源表之间在创建动作完毕之后是完全无关的。在源表做的任何修改都不会传播到新表中，并且也不可能在扫描源表的时候包含新表的数据。

被复制的列和约束并不使用相同的名称进行融合。如果明确的指定了相同的名称或者在另外一个 LIKE 子句中，将会报错。

- 源表上的字段缺省表达式只有在指定 INCLUDING DEFAULTS 时，才会复制到新表中。缺省是不包含缺省表达式的，即新表中的所有字段的缺省值都是 NULL。
- 源表上的 CHECK 约束仅在指定 INCLUDING CONSTRAINTS 时，会复制到新表中，而其他类型的约束永远不会复制到新表中。非空约束总是复制到新表中。此规则同时适用于表约束和列约束。
- 如果指定了 INCLUDING INDEXES，则源表上的索引也将在新表上创建，默认不建立索引。
- 如果指定了 INCLUDING STORAGE，则复制列的 STORAGE 设置会复制到新表中，默认情况下不包含 STORAGE 设置。
- 如果指定了 INCLUDING COMMENTS，则源表列、约束和索引的注释会复制到新表中。默认情况下，不复制源表的注释。
- 如果指定了 INCLUDING PARTITION，则源表的分区定义会复制到新表中，同时新表将不能再使用 PARTITION BY 子句。默认情况下，不拷贝源表的分区定义。
- 如果指定了 INCLUDING RELOPTIONS，则源表的存储参数（即源表的 WITH 子句）会复制到新表中。默认情况下，不复制源表的存储参数。
- 如果指定了 INCLUDING DISTRIBUTION，则源表的分布信息会复制到新表中，包括分布类型和分布列。默认情况下，不拷贝源表的分布信息。
- INCLUDING ALL 包含了 INCLUDING DEFAULTS、INCLUDING CONSTRAINTS、INCLUDING INDEXES、INCLUDING STORAGE、INCLUDING COMMENTS、INCLUDING PARTITION、INCLUDING RELOPTIONS 和 INCLUDING DISTRIBUTION 的内容。

须知

- 如果源表包含 serial、bigserial、smallseriral 类型，或者源表字段的默认值是 sequence，且 sequence 属于源表（通过 CREATE SEQUENCE ... OWNED BY 创建），这些 Sequence 不会关联到新表中，新表中会重新创建属于自己的 sequence。这和之前版本的处理逻辑不同。如果用户希望源表和新表共享 Sequence，需要首先创建一个共享的 Sequence（避免使用 OWNED BY），并配置为源表字段默认值，这样创建的新表会和源表共享该 Sequence。

- 不建议将其他表私有的 Sequence 配置为源表字段的默认值，尤其是其他表只分布在特定的 NodeGroup 上，这可能导致 CREATE TABLE ... LIKE 执行失败。另外，如果源表配置其他表私有的 Sequence，当该表删除时 Sequence 也会连带删除，这样源表的 Sequence 将不可用。如果用户希望多个表共享 Sequence，建议创建共享的 Sequence。

❖ WITH ({ storage_parameter = value } [, ...])

这个子句为表或索引指定一个可选的存储参数。

说明

使用任意精度类型 Numeric 定义列时，建议指定精度 p 以及刻度 s。在不指定精度和刻度时，会按输入的显示出来。

参数的详细描述如下所示。

– FILLFACTOR

一个表的填充因子（fillfactor）是一个介于 10 和 100 之间的百分数。100（完全填充）是默认值。如果指定了较小的填充因子，INSERT 操作仅按照填充因子指定的百分率填充表页。每个页上的剩余空间将用于在该页上更新行，这就使得 UPDATE 有机会在同一页上放置同一条记录的新版本，这比把新版本放置在其他页上更有效。对于一个从不更新的表将填充因子设为 100 是最佳选择，但是对于频繁更新的表，选择较小的填充因子则更加合适。该参数对于列存表没有意义。

取值范围：10~100

– ORIENTATION

指定表数据的存储方式，即行存方式、列存方式、ORC 格式的方式，该参数设置成功后就不再支持修改。

取值范围：

■ ROW，表示表的数据将以行式存储。

行存储适合于 OLTP 业务，此类型的表上交互事务比较多，一次交互会涉及表中的多个列，用行存查询效率较高。

■ COLUMN，表示表的数据将以列式存储。

列存储适合于数据仓库业务，此类型的表上会做大量的汇聚计算，且涉及的列操作较少。

默认值：

若指定表空间为普通表空间，默认值为 ROW。

– COMPRESSION

指定表数据的压缩级别，它决定了表数据的压缩比以及压缩时间。一般来讲，压缩级别越高，压缩比也越大，压缩时间也越长；反之亦然。实际压缩比取决于加载的表数据的分布特征。

取值范围：

列存表的有效值为 YES/NO/LOW/MIDDLE/HIGH，默认值为 LOW。

- COMPRESSLEVEL

  指定表数据同一压缩级别下的不同压缩水平，它决定了同一压缩级别下表数据的压缩比以及压缩时间。对同一压缩级别进行了更加详细的划分，为用户选择压缩比和压缩时间提供了更多的空间。总体来讲，此值越大，表示同一压缩级别下压缩比越大，压缩时间越长；反之亦然。

  取值范围：0~3，默认值为 0。

- MAX_BATCHROW

  指定了在数据加载过程中一个存储单元可以容纳记录的最大数目。该参数只对列存表有效。

  取值范围：10000~60000

- PARTIAL_CLUSTER_ROWS

  指定了在数据加载过程中进行将局部聚簇存储的记录数目。该参数只对列存表有效。

  取值范围：600000~2147483647

- DELTAROW_THRESHOLD

  指定列存表导入时小于多少行的数据进入 delta 表，只在 GUC 参数 enable_delta_store 开启时生效。该参数只对列存表有效。

  取值范围：0～9999，默认值为 100

- VERSION

  指定 ORC 存储格式的版本。

  取值范围：0.12，目前支持 ORC 0.12 格式，后续会随着 ORC 格式的发展，支持更多格式。

  默认值：0.12

❖ ON COMMIT { PRESERVE ROWS | DELETE ROWS | DROP }

ON COMMIT 选项决定在事务中执行创建临时表操作，当事务提交时，此临时表的后续操作。有以下三个选项，当前支持 PRESERVE ROWS 和 DELETE ROWS 选项。

- PRESERVE ROWS（缺省值）：提交时不对临时表做任何操作，临时表及其表数据保持不变。
- DELETE ROWS：提交时删除临时表中数据。
- DROP：提交时删除此临时表。

❖ COMPRESS | NOCOMPRESS

创建新表时，需要在 CREATE TABLE 语句中指定关键字 COMPRESS，这样，当对该表进行批量插入时就会触发压缩特性。该特性会在页范围内扫描所有元组数据，生成字典、压缩元组数据并进行存储。指定关键字 NOCOMPRESS 则不对表进行压缩。

缺省值：NOCOMPRESS，即不对元组数据进行压缩。

- TABLESPACE tablespace_name

  创建新表时指定此关键字，表示新表将要在指定表空间内创建。如果没有声明，将使用默认表空间。

- CONSTRAINT constraint_name

  列约束或表约束的名称。可选的约束子句用于声明约束，新行或者更新的行必须满足这些约束才能成功插入或更新。

  定义约束有两种方法：

  - 列约束：作为一个列定义的一部分，仅影响该列。
  - 表约束：不和某个列绑在一起，可以作用于多个列。

- NOT NULL

  字段值不允许为 NULL。

- NULL

  字段值允许为 NULL ，这是缺省值。

  这个子句只是为和非标准 SQL 数据库兼容。不建议使用。

- CHECK (expression)

  CHECK 约束声明一个布尔表达式，每次要插入的新行或者要更新的行的新值必须使表达式结果为真或未知才能成功，否则会抛出一个异常并且不会修改数据库。

  声明为字段约束的检查约束应该只引用该字段的数值，而在表约束里出现的表达式可以引用多个字段。

说明

expression 表达式中，如果存在“<>NULL”或“！=NULL”，这种写法是无效的，需要写成“is NOT NULL”。

- DEFAULT default_expr

  DEFAULT 子句给字段指定缺省值。该数值可以是任何不含变量的表达式(不允许使用子查询和对本表中的其他字段的交叉引用)。缺省表达式的数据类型必须和字段类型匹配。

  缺省表达式将被用于任何未声明该字段数值的插入操作。如果没有指定缺省值则缺省值为 NULL 。

- UNIQUE index_parameters

  UNIQUE (column_name [, ...]) index_parameters

  UNIQUE 约束表示表里的一个字段或多个字段的组合必须在全表范围内唯一。

  对于唯一约束，NULL 被认为是互不相等的。

- PRIMARY KEY index_parameters

PRIMARY KEY (column_name [, ...]) index_parameters

主键约束声明表中的一个或者多个字段只能包含唯一的非 NULL 值。

一个表只能声明一个主键。

- DEFERRABLE | NOT DEFERRABLE

  这两个关键字设置该约束是否可推迟。一个不可推迟的约束将在每条命令之后马上检查。可推迟约束可以推迟到事务结尾使用 SET CONSTRAINTS 命令检查。缺省是 NOT DEFERRABLE。目前，UNIQUE 约束和主键约束可以接受这个子句。所有其他约束类型都是不可推迟的。

- PARTIAL CLUSTER KEY

  局部聚簇存储，列存表导入数据时按照指定的列(单列或多列)，进行局部排序。

- INITIALLY IMMEDIATE | INITIALLY DEFERRED

  如果约束是可推迟的，则这个子句声明检查约束的缺省时间。

  - 如果约束是 INITIALLY IMMEDIATE（缺省），则在每条语句执行之后就立即检查它；
  - 如果约束是 INITIALLY DEFERRED ，则只有在事务结尾才检查它。

  约束检查的时间可以用 SET CONSTRAINTS 命令修改。

- USING INDEX TABLESPACE tablespace_name

  为 UNIQUE 或 PRIMARY KEY 约束相关的索引声明一个表空间。如果没有提供这个子句，这个索引将在 default_tablespace 中创建，如果 default_tablespace 为空，将使用数据库的缺省表空间。

示例

```
--创建简单的表。
vastbase=# CREATE TABLE tpcds.warehouse_t1
(
    W_WAREHOUSE_SK          INTEGER             NOT NULL,
    W_WAREHOUSE_ID          CHAR(16)            NOT NULL,
    W_WAREHOUSE_NAME         VARCHAR(20)                 ,
    W_WAREHOUSE_SQ_FT        INTEGER                    ,
    W_STREET_NUMBER         CHAR(10)                    ,
    W_STREET_NAME           VARCHAR(60)                 ,
    W STREET TYPE           CHAR(15)                   ,
    W SUITE NUMBER          CHAR(10)                   ,
    W CITY                 VARCHAR(60)                ,
    W COUNTY                VARCHAR(30)                ,
    W STATE                CHAR(2)                    ,
    W_ZIP                  CHAR(10)                   ,
    W_COUNTRY               VARCHAR(20)                ,
    W_GMT_OFFSET            DECIMAL(5,2)
);

vastbase=# CREATE TABLE tpcds.warehouse_t2
(
```

```
    W_WAREHOUSE_SK          INTEGER             NOT NULL,
    W_WAREHOUSE_ID          CHAR(16)            NOT NULL,
    W_WAREHOUSE_NAME        VARCHAR(20)                 ,
    W_WAREHOUSE_SQ_FT       INTEGER                     ,
    W_STREET_NUMBER         CHAR(10)                    ,
    W_STREET_NAME           VARCHAR(60)        DICTIONARY,
    W_STREET_TYPE           CHAR(15)                    ,
    W_SUITE_NUMBER          CHAR(10)                    ,
    W_CITY                  VARCHAR(60)                 ,
    W_COUNTY                VARCHAR(30)                 ,
    W_STATE                 CHAR(2)                     ,
    W_ZIP                   CHAR(10)                    ,
    W_COUNTRY               VARCHAR(20)                 ,
    W_GMT_OFFSET            DECIMAL(5,2)
);
--创建表，并指定 W_STATE 字段的缺省值为 GA。
vastbase=# CREATE TABLE tpcds.warehouse_t3
(
    W_WAREHOUSE_SK          INTEGER             NOT NULL,
    W_WAREHOUSE_ID          CHAR(16)            NOT NULL,
    W_WAREHOUSE_NAME        VARCHAR(20)                 ,
    W_WAREHOUSE_SQ_FT       INTEGER                     ,
    W_STREET_NUMBER         CHAR(10)                    ,
    W_STREET_NAME           VARCHAR(60)                 ,
    W_STREET_TYPE           CHAR(15)                    ,
    W_SUITE_NUMBER          CHAR(10)                    ,
    W_CITY                  VARCHAR(60)                 ,
    W_COUNTY                VARCHAR(30)                 ,
    W_STATE                 CHAR(2)         DEFAULT 'GA',
    W_ZIP                   CHAR(10)                    ,
    W_COUNTRY               VARCHAR(20)                 ,
    W_GMT_OFFSET            DECIMAL(5,2)
);

--创建表，并在事务结束时检查 W_WAREHOUSE_NAME 字段是否有重复。
vastbase=# CREATE TABLE tpcds.warehouse_t4
(
    W_WAREHOUSE_SK          INTEGER             NOT NULL,
    W_WAREHOUSE_ID          CHAR(16)            NOT NULL,
    W_WAREHOUSE_NAME        VARCHAR(20)  UNIQUE DEFERRABLE,
    W_WAREHOUSE_SQ_FT       INTEGER                     ,
    W_STREET_NUMBER         CHAR(10)                    ,
    W_STREET_NAME           VARCHAR(60)                 ,
    W_STREET_TYPE           CHAR(15)                    ,
    W_SUITE_NUMBER          CHAR(10)                    ,
    W_CITY                  VARCHAR(60)                 ,
    W_COUNTY                VARCHAR(30)                 ,
    W_STATE                 CHAR(2)                     ,
    W_ZIP                   CHAR(10)                    ,
    W_COUNTRY               VARCHAR(20)                 ,
    W_GMT_OFFSET            DECIMAL(5,2)
);
--创建一个带有 70%填充因子的表。
vastbase=# CREATE TABLE tpcds.warehouse_t5
(
```

```
    W_WAREHOUSE_SK          INTEGER             NOT NULL,
    W_WAREHOUSE_ID          CHAR(16)            NOT NULL,
    W_WAREHOUSE_NAME        VARCHAR(20)                 ,
    W_WAREHOUSE_SQ_FT       INTEGER                     ,
    W_STREET_NUMBER         CHAR(10)                    ,
    W_STREET_NAME           VARCHAR(60)                 ,
    W_STREET_TYPE           CHAR(15)                    ,
    W_SUITE_NUMBER          CHAR(10)                    ,
    W_CITY                  VARCHAR(60)                 ,
    W_COUNTY                VARCHAR(30)                 ,
    W_STATE                 CHAR(2)                     ,
    W_ZIP                   CHAR(10)                    ,
    W_COUNTRY               VARCHAR(20)                 ,
    W_GMT_OFFSET            DECIMAL(5,2),
    UNIQUE(W_WAREHOUSE_NAME) WITH(fillfactor=70)
);

--或者用下面的语法。
vastbase=# CREATE TABLE tpcds.warehouse_t6
(
    W_WAREHOUSE_SK          INTEGER             NOT NULL,
    W_WAREHOUSE_ID          CHAR(16)            NOT NULL,
    W_WAREHOUSE_NAME        VARCHAR(20)            UNIQUE,
    W_WAREHOUSE_SQ_FT       INTEGER                     ,
    W_STREET_NUMBER         CHAR(10)                    ,
    W_STREET_NAME           VARCHAR(60)                 ,
    W_STREET_TYPE           CHAR(15)                    ,
    W_SUITE_NUMBER          CHAR(10)                    ,
    W_CITY                  VARCHAR(60)                 ,
    W_COUNTY                VARCHAR(30)                 ,
    W_STATE                 CHAR(2)                     ,
    W_ZIP                   CHAR(10)                    ,
    W_COUNTRY               VARCHAR(20)                 ,
    W_GMT_OFFSET            DECIMAL(5,2)
) WITH(fillfactor=70);

--创建表，并指定该表数据不写入预写日志。
vastbase=# CREATE UNLOGGED TABLE tpcds.warehouse_t7
(
    W_WAREHOUSE_SK          INTEGER            NOT NULL,
    W_WAREHOUSE_ID          CHAR(16)           NOT NULL,
    W_WAREHOUSE_NAME        VARCHAR(20)                ,
    W_WAREHOUSE_SQ_FT       INTEGER                    ,
    W_STREET_NUMBER         CHAR(10)                   ,
    W_STREET_NAME           VARCHAR(60)                ,
    W_STREET_TYPE           CHAR(15)                   ,
    W_SUITE_NUMBER          CHAR(10)                   ,
    W_CITY                  VARCHAR(60)                ,
    W_COUNTY                VARCHAR(30)                ,
    W_STATE                 CHAR(2)                    ,
    W_ZIP                   CHAR(10)                   ,
    W_COUNTRY               VARCHAR(20)                ,
    W_GMT_OFFSET            DECIMAL(5,2)
);
```

```
--创建表临时表。
vastbase=# CREATE TEMPORARY TABLE warehouse_t24
(
    W_WAREHOUSE_SK          INTEGER             NOT NULL,
    W_WAREHOUSE_ID          CHAR(16)            NOT NULL,
    W_WAREHOUSE_NAME        VARCHAR(20)                 ,
    W_WAREHOUSE_SQ_FT       INTEGER                     ,
    W_STREET_NUMBER         CHAR(10)                    ,
    W_STREET_NAME           VARCHAR(60)                 ,
    W_STREET_TYPE           CHAR(15)                    ,
    W_SUITE_NUMBER          CHAR(10)                    ,
    W_CITY                  VARCHAR(60)                 ,
    W_COUNTY                VARCHAR(30)                 ,
    W_STATE                 CHAR(2)                     ,
    W_ZIP                   CHAR(10)                    ,
    W_COUNTRY               VARCHAR(20)                 ,
    W_GMT_OFFSET            DECIMAL(5,2)
);

--事务中创建表临时表，并指定提交事务时删除该临时表数据。
vastbase=# CREATE TEMPORARY TABLE warehouse_t25
(
    W_WAREHOUSE_SK          INTEGER             NOT NULL,
    W_WAREHOUSE_ID          CHAR(16)            NOT NULL,
    W_WAREHOUSE_NAME        VARCHAR(20)                 ,
    W_WAREHOUSE_SQ_FT       INTEGER                     ,
    W_STREET_NUMBER         CHAR(10)                    ,
    W_STREET_NAME           VARCHAR(60)                 ,
    W_STREET_TYPE           CHAR(15)                    ,
    W_SUITE_NUMBER          CHAR(10)                    ,
    W_CITY                  VARCHAR(60)                 ,
    W_COUNTY                VARCHAR(30)                 ,
    W_STATE                 CHAR(2)                     ,
    W_ZIP                   CHAR(10)                    ,
    W_COUNTRY               VARCHAR(20)                 ,
    W_GMT_OFFSET            DECIMAL(5,2)
) ON COMMIT DELETE ROWS;

--创建表时，不希望因为表已存在而报错。
vastbase=# CREATE TABLE IF NOT EXISTS tpcds.warehouse t8
(
    W_WAREHOUSE_SK          INTEGER             NOT NULL,
    W_WAREHOUSE_ID          CHAR(16)            NOT NULL,
    W_WAREHOUSE_NAME        VARCHAR(20)                 ,
    W_WAREHOUSE_SQ_FT       INTEGER                     ,
    W_STREET_NUMBER         CHAR(10)                    ,
    W_STREET_NAME           VARCHAR(60)                 ,
    W_STREET_TYPE           CHAR(15)                    ,
    W_SUITE_NUMBER          CHAR(10)                    ,
    W_CITY                  VARCHAR(60)                 ,
    W_COUNTY                VARCHAR(30)                 ,
    W_STATE                 CHAR(2)                     ,
    W_ZIP                   CHAR(10)                    ,
    W_COUNTRY               VARCHAR(20)                 ,
    W_GMT_OFFSET            DECIMAL(5,2)
```

```
);
 --创建带有自增列的表。
vastbase=# create table test_b
(
  id serial PRIMARY KEY,
  name character varying(64)
  );
--创建普通表空间。
vastbase=# CREATE TABLESPACE DS_TABLESPACE1 RELATIVE LOCATION 'tablespace/tablespace_1';

--创建表时，指定表空间。
vastbase=# CREATE TABLE tpcds.warehouse_t9
(
    W_WAREHOUSE_SK          INTEGER             NOT NULL,
    W_WAREHOUSE_ID          CHAR(16)            NOT NULL,
    W_WAREHOUSE_NAME         VARCHAR(20)                 ,
    W_WAREHOUSE_SQ_FT        INTEGER                    ,
    W_STREET_NUMBER         CHAR(10)                    ,
    W_STREET_NAME           VARCHAR(60)                 ,
    W_STREET_TYPE           CHAR(15)                   ,
    W_SUITE_NUMBER          CHAR(10)                    ,
    W_CITY                 VARCHAR(60)                 ,
    W_COUNTY                VARCHAR(30)                 ,
    W_STATE                CHAR(2)                     ,
    W_ZIP                  CHAR(10)                    ,
    W_COUNTRY               VARCHAR(20)                 ,
    W_GMT_OFFSET            DECIMAL(5,2)
) TABLESPACE DS_TABLESPACE1;

--创建表时，单独指定W_WAREHOUSE_NAME的索引表空间。
vastbase=# CREATE TABLE tpcds.warehouse_t10
(
    W_WAREHOUSE_SK          INTEGER             NOT NULL,
    W_WAREHOUSE_ID          CHAR(16)            NOT NULL,
    W_WAREHOUSE_NAME         VARCHAR(20)         UNIQUE USING INDEX TABLESPACE DS_TABLESPACE1,
    W_WAREHOUSE_SQ_FT        INTEGER                    ,
    W_STREET_NUMBER         CHAR(10)                    ,
    W_STREET_NAME           VARCHAR(60)                 ,
    W_STREET_TYPE           CHAR(15)                   ,
    W_SUITE_NUMBER          CHAR(10)                    ,
    W_CITY                 VARCHAR(60)                 ,
    W_COUNTY                VARCHAR(30)                 ,
    W_STATE                CHAR(2)                     ,
    W_ZIP                  CHAR(10)                    ,
    W_COUNTRY               VARCHAR(20)                 ,
    W_GMT_OFFSET            DECIMAL(5,2)
);
--创建一个有主键约束的表。
vastbase=# CREATE TABLE tpcds.warehouse_t11
(
    W_WAREHOUSE_SK          INTEGER          PRIMARY KEY,
    W_WAREHOUSE_ID          CHAR(16)           NOT NULL,
    W_WAREHOUSE_NAME         VARCHAR(20)                 ,
    W_WAREHOUSE_SQ_FT        INTEGER                    ,
    W_STREET_NUMBER         CHAR(10)                    ,
```

```
    W_STREET_NAME           VARCHAR(60)                ,
    W_STREET_TYPE           CHAR(15)                   ,
    W_SUITE_NUMBER          CHAR(10)                   ,
    W_CITY                  VARCHAR(60)                ,
    W_COUNTY                VARCHAR(30)                ,
    W_STATE                 CHAR(2)                    ,
    W_ZIP                   CHAR(10)                   ,
    W_COUNTRY               VARCHAR(20)                ,
    W_GMT_OFFSET            DECIMAL(5,2)
);

---或是用下面的语法，效果完全一样。
vastbase=# CREATE TABLE tpcds.warehouse_t12
(
    W_WAREHOUSE_SK          INTEGER             NOT NULL,
    W_WAREHOUSE_ID          CHAR(16)            NOT NULL,
    W_WAREHOUSE_NAME        VARCHAR(20)                ,
    W_WAREHOUSE_SQ_FT       INTEGER                    ,
    W_STREET_NUMBER         CHAR(10)                   ,
    W_STREET_NAME           VARCHAR(60)                ,
    W_STREET_TYPE           CHAR(15)                   ,
    W_SUITE_NUMBER          CHAR(10)                   ,
    W_CITY                  VARCHAR(60)                ,
    W_COUNTY                VARCHAR(30)                ,
    W_STATE                 CHAR(2)                    ,
    W_ZIP                   CHAR(10)                   ,
    W_COUNTRY               VARCHAR(20)                ,
    W_GMT_OFFSET            DECIMAL(5,2),
    PRIMARY KEY(W_WAREHOUSE_SK)
);

--或是用下面的语法，指定约束的名称。
vastbase=# CREATE TABLE tpcds.warehouse_t13
(
    W_WAREHOUSE_SK          INTEGER             NOT NULL,
    W_WAREHOUSE_ID          CHAR(16)            NOT NULL,
    W_WAREHOUSE_NAME        VARCHAR(20)                ,
    W_WAREHOUSE_SQ_FT       INTEGER                    ,
    W_STREET_NUMBER         CHAR(10)                   ,
    W_STREET_NAME           VARCHAR(60)                ,
    W_STREET_TYPE           CHAR(15)                   ,
    W_SUITE_NUMBER          CHAR(10)                   ,
    W_CITY                  VARCHAR(60)                ,
    W_COUNTY                VARCHAR(30)                ,
    W_STATE                 CHAR(2)                    ,
    W_ZIP                   CHAR(10)                   ,
    W_COUNTRY               VARCHAR(20)                ,
    W_GMT_OFFSET            DECIMAL(5,2),
    CONSTRAINT W_CSTR_KEY1 PRIMARY KEY(W_WAREHOUSE_SK)
);

--创建一个有复合主键约束的表。
vastbase=# CREATE TABLE tpcds.warehouse_t14
(
    W_WAREHOUSE_SK          INTEGER             NOT NULL,
```

```
    W_WAREHOUSE_ID          CHAR(16)            NOT NULL,
    W_WAREHOUSE_NAME        VARCHAR(20)                 ,
    W_WAREHOUSE_SQ_FT       INTEGER                     ,
    W_STREET_NUMBER         CHAR(10)                    ,
    W_STREET_NAME           VARCHAR(60)                 ,
    W_STREET_TYPE           CHAR(15)                    ,
    W_SUITE_NUMBER          CHAR(10)                    ,
    W_CITY                  VARCHAR(60)                 ,
    W_COUNTY                VARCHAR(30)                 ,
    W_STATE                 CHAR(2)                     ,
    W_ZIP                   CHAR(10)                    ,
    W_COUNTRY               VARCHAR(20)                 ,
    W_GMT_OFFSET            DECIMAL(5,2),
    CONSTRAINT W_CSTR_KEY2 PRIMARY KEY(W_WAREHOUSE_SK, W_WAREHOUSE_ID)
);

--创建列存表。
vastbase=# CREATE TABLE tpcds.warehouse_t15
(
    W_WAREHOUSE_SK          INTEGER             NOT NULL,
    W_WAREHOUSE_ID          CHAR(16)            NOT NULL,
    W_WAREHOUSE_NAME        VARCHAR(20)                 ,
    W_WAREHOUSE_SQ_FT       INTEGER                     ,
    W_STREET_NUMBER         CHAR(10)                    ,
    W_STREET_NAME           VARCHAR(60)                 ,
    W_STREET_TYPE           CHAR(15)                    ,
    W_SUITE_NUMBER          CHAR(10)                    ,
    W_CITY                  VARCHAR(60)                 ,
    W_COUNTY                VARCHAR(30)                 ,
    W_STATE                 CHAR(2)                     ,
    W_ZIP                   CHAR(10)                    ,
    W_COUNTRY               VARCHAR(20)                 ,
    W_GMT_OFFSET            DECIMAL(5,2)
) WITH (ORIENTATION = COLUMN);

--创建局部聚簇存储的列存表。
vastbase=# CREATE TABLE tpcds.warehouse_t16
(
    W_WAREHOUSE_SK          INTEGER             NOT NULL,
    W_WAREHOUSE_ID          CHAR(16)            NOT NULL,
    W_WAREHOUSE_NAME        VARCHAR(20)                 ,
    W_WAREHOUSE_SQ_FT       INTEGER                     ,
    W_STREET_NUMBER         CHAR(10)                    ,
    W_STREET_NAME           VARCHAR(60)                 ,
    W_STREET_TYPE           CHAR(15)                    ,
    W_SUITE_NUMBER          CHAR(10)                    ,
    W_CITY                  VARCHAR(60)                 ,
    W_COUNTY                VARCHAR(30)                 ,
    W_STATE                 CHAR(2)                     ,
    W_ZIP                   CHAR(10)                    ,
    W_COUNTRY               VARCHAR(20)                 ,
    W_GMT_OFFSET            DECIMAL(5,2),
    PARTIAL CLUSTER KEY(W_WAREHOUSE_SK, W_WAREHOUSE_ID)
) WITH (ORIENTATION = COLUMN);
```

```
--定义一个带压缩的列存表。
vastbase=# CREATE TABLE tpcds.warehouse_t17
(
    W_WAREHOUSE_SK          INTEGER             NOT NULL,
    W_WAREHOUSE_ID          CHAR(16)            NOT NULL,
    W_WAREHOUSE_NAME        VARCHAR(20)                 ,
    W_WAREHOUSE_SQ_FT       INTEGER                     ,
    W_STREET_NUMBER         CHAR(10)                    ,
    W_STREET_NAME           VARCHAR(60)                 ,
    W_STREET_TYPE           CHAR(15)                    ,
    W_SUITE_NUMBER          CHAR(10)                    ,
    W_CITY                  VARCHAR(60)                 ,
    W_COUNTY                VARCHAR(30)                 ,
    W_STATE                 CHAR(2)                     ,
    W_ZIP                   CHAR(10)                    ,
    W_COUNTRY               VARCHAR(20)                 ,
    W_GMT_OFFSET            DECIMAL(5,2)
) WITH (ORIENTATION = COLUMN, COMPRESSION=HIGH);

--定义一个带压缩的表。
vastbase=# CREATE TABLE tpcds.warehouse_t18
(
    W_WAREHOUSE_SK          INTEGER             NOT NULL,
    W_WAREHOUSE_ID          CHAR(16)            NOT NULL,
    W_WAREHOUSE_NAME        VARCHAR(20)                 ,
    W_WAREHOUSE_SQ_FT       INTEGER                     ,
    W_STREET_NUMBER         CHAR(10)                    ,
    W_STREET_NAME           VARCHAR(60)                 ,
    W_STREET_TYPE           CHAR(15)                    ,
    W_SUITE_NUMBER          CHAR(10)                    ,
    W_CITY                  VARCHAR(60)                 ,
    W_COUNTY                VARCHAR(30)                 ,
    W_STATE                 CHAR(2)                     ,
    W_ZIP                   CHAR(10)                    ,
    W_COUNTRY               VARCHAR(20)                 ,
    W_GMT_OFFSET            DECIMAL(5,2)
) COMPRESS;

--定义一个检查列约束。
vastbase=# CREATE TABLE tpcds.warehouse t19
(
    W_WAREHOUSE_SK          INTEGER             PRIMARY KEY CHECK (W_WAREHOUSE_SK > 0),
    W_WAREHOUSE_ID          CHAR(16)            NOT NULL,
    W_WAREHOUSE_NAME        VARCHAR(20)          CHECK (W_WAREHOUSE_NAME IS NOT NULL),
    W_WAREHOUSE_SQ_FT       INTEGER                     ,
    W_STREET_NUMBER         CHAR(10)                    ,
    W_STREET_NAME           VARCHAR(60)                 ,
    W_STREET_TYPE           CHAR(15)                    ,
    W_SUITE_NUMBER          CHAR(10)                    ,
    W_CITY                  VARCHAR(60)                 ,
    W_COUNTY                VARCHAR(30)                 ,
    W_STATE                 CHAR(2)                     ,
    W_ZIP                   CHAR(10)                    ,
    W_COUNTRY               VARCHAR(20)                 ,
    W_GMT_OFFSET            DECIMAL(5,2)
```

```
);

vastbase=# CREATE TABLE tpcds.warehouse_t20
(
    W_WAREHOUSE_SK          INTEGER             PRIMARY KEY,
    W_WAREHOUSE_ID          CHAR(16)            NOT NULL,
    W_WAREHOUSE_NAME        VARCHAR(20)          CHECK (W_WAREHOUSE_NAME IS NOT NULL),
    W_WAREHOUSE_SQ_FT       INTEGER                    ,
    W_STREET_NUMBER         CHAR(10)                   ,
    W_STREET_NAME           VARCHAR(60)                ,
    W_STREET_TYPE           CHAR(15)                   ,
    W_SUITE_NUMBER          CHAR(10)                   ,
    W_CITY                  VARCHAR(60)                ,
    W_COUNTY                VARCHAR(30)                ,
    W_STATE                 CHAR(2)                    ,
    W_ZIP                   CHAR(10)                   ,
    W_COUNTRY               VARCHAR(20)                ,
    W_GMT_OFFSET            DECIMAL(5,2),
    CONSTRAINT W_CONSTR_KEY2 CHECK(W_WAREHOUSE_SK > 0 AND W_WAREHOUSE_NAME IS NOT NULL)
);

--定义一个表，表中每一个行存在数据库节点中。
vastbase=# CREATE TABLE tpcds.warehouse_t21
(
    W_WAREHOUSE_SK          INTEGER             NOT NULL,
    W_WAREHOUSE_ID          CHAR(16)            NOT NULL,
    W_WAREHOUSE_NAME        VARCHAR(20)                ,
    W_WAREHOUSE_SQ_FT       INTEGER                    ,
    W_STREET_NUMBER         CHAR(10)                   ,
    W_STREET_NAME           VARCHAR(60)                ,
    W_STREET_TYPE           CHAR(15)                   ,
    W_SUITE_NUMBER          CHAR(10)                   ,
    W_CITY                  VARCHAR(60)                ,
    W_COUNTY                VARCHAR(30)                ,
    W_STATE                 CHAR(2)                    ,
    W_ZIP                   CHAR(10)                   ,
    W_COUNTRY               VARCHAR(20)                ,
    W_GMT_OFFSET            DECIMAL(5,2)
);

--定义一个表，使用 HASH 分布。
vastbase=# CREATE TABLE tpcds.warehouse_t22
(
    W_WAREHOUSE_SK          INTEGER             NOT NULL,
    W_WAREHOUSE_ID          CHAR(16)            NOT NULL,
    W_WAREHOUSE_NAME        VARCHAR(20)                ,
    W_WAREHOUSE_SQ_FT       INTEGER                    ,
    W_STREET_NUMBER         CHAR(10)                   ,
    W_STREET_NAME           VARCHAR(60)                ,
    W_STREET_TYPE           CHAR(15)                   ,
    W_SUITE_NUMBER          CHAR(10)                   ,
    W_CITY                  VARCHAR(60)                ,
    W_COUNTY                VARCHAR(30)                ,
    W_STATE                 CHAR(2)                    ,
    W_ZIP                   CHAR(10)                   ,
```

```
   W_COUNTRY              VARCHAR(20)                ,
   W_GMT_OFFSET            DECIMAL(5,2),
   CONSTRAINT W_CONSTR_KEY3 UNIQUE(W_WAREHOUSE_SK)
);

--向 tpcds.warehouse_t19 表中增加一个 varchar 列。
vastbase=# ALTER TABLE tpcds.warehouse_t19 ADD W_GOODS_CATEGORY varchar(30);

--给 tpcds.warehouse_t19 表增加一个检查约束。
vastbase=# ALTER TABLE tpcds.warehouse_t19 ADD CONSTRAINT W_CONSTR_KEY4 CHECK (W_STATE IS NOT NULL);

--在一个操作中改变两个现存字段的类型。
vastbase=# ALTER TABLE tpcds.warehouse_t19
   ALTER COLUMN W_GOODS_CATEGORY TYPE varchar(80),
   ALTER COLUMN W_STREET_NAME TYPE varchar(100);

--此语句与上面语句等效。
vastbase=# ALTER TABLE tpcds.warehouse_t19 MODIFY (W_GOODS_CATEGORY varchar(30), W_STREET_NAME
varchar(60));

--给一个已存在字段添加非空约束。
vastbase=# ALTER TABLE tpcds.warehouse_t19 ALTER COLUMN W_GOODS_CATEGORY SET NOT NULL;

--移除已存在字段的非空约束。
vastbase=# ALTER TABLE tpcds.warehouse_t19 ALTER COLUMN W_GOODS_CATEGORY DROP NOT NULL;

--如果列存表中还未指定局部聚簇，向在一个列存表中添加局部聚簇列。
vastbase=# ALTER TABLE tpcds.warehouse_t17 ADD PARTIAL CLUSTER KEY(W_WAREHOUSE_SK);

--查看约束的名称，并删除一个列存表中的局部聚簇列。
vastbase=# \d+ tpcds.warehouse_t17
                      Table "tpcds.warehouse_t17"
     Column      |         Type          | Modifiers | Storage  | Stats target | Description
-----------------+-----------------------+-----------+----------+--------------+-------------
 w_warehouse_sk    | integer             | not null  | plain    |              |
 w_warehouse_id    | character(16)       | not null  | extended |              |
 w_warehouse_name  | character varying(20) |           | extended |              |
 w_warehouse_sq_ft | integer             |           | plain    |              |
 w_street_number   | character(10)       |           | extended |              |
 w_street_name     | character varying(60) |         | extended |              |
 w_street_type     | character(15)       |           | extended |              |
 w_suite_number    | character(10)       |           | extended |              |
 w_city            | character varying(60) |         | extended |              |
 w_county          | character varying(30) |         | extended |              |
 w_state           | character(2)        |           | extended |              |
 w_zip             | character(10)       |           | extended |              |
 w_country         | character varying(20) |         | extended |              |
 w_gmt_offset      | numeric(5,2)        |           | main     |              |
Partial Cluster :
   "warehouse_t17_cluster" PARTIAL CLUSTER KEY (w_warehouse_sk)
Has OIDs: no
Location Nodes: ALL DATANODES
Options: compression=no, version=0.12
vastbase=# ALTER TABLE tpcds.warehouse_t17 DROP CONSTRAINT warehouse_t17_cluster;
```

```
--将表移动到另一个表空间。
vastbase=# ALTER TABLE tpcds.warehouse_t19 SET TABLESPACE PG_DEFAULT;
--创建模式 joe。
vastbase=# CREATE SCHEMA joe;

--将表移动到另一个模式中。
vastbase=# ALTER TABLE tpcds.warehouse_t19 SET SCHEMA joe;

--重命名已存在的表。
vastbase=# ALTER TABLE joe.warehouse_t19 RENAME TO warehouse_t23;

--从 warehouse_t23 表中删除一个字段。
vastbase=# ALTER TABLE joe.warehouse t23 DROP COLUMN W STREET NAME;

--删除表空间、模式 joe 和模式表 warehouse。
vastbase=# DROP TABLE tpcds.warehouse_t1;
vastbase=# DROP TABLE tpcds.warehouse_t2;
vastbase=# DROP TABLE tpcds.warehouse_t3;
vastbase=# DROP TABLE tpcds.warehouse_t4;
vastbase=# DROP TABLE tpcds.warehouse_t5;
vastbase=# DROP TABLE tpcds.warehouse_t6;
vastbase=# DROP TABLE tpcds.warehouse_t7;
vastbase=# DROP TABLE tpcds.warehouse_t8;
vastbase=# DROP TABLE tpcds.warehouse_t9;
vastbase=# DROP TABLE tpcds.warehouse_t10;
vastbase=# DROP TABLE tpcds.warehouse_t11;
vastbase=# DROP TABLE tpcds.warehouse_t12;
vastbase=# DROP TABLE tpcds.warehouse_t13;
vastbase=# DROP TABLE tpcds.warehouse_t14;
vastbase=# DROP TABLE tpcds.warehouse_t15;
vastbase=# DROP TABLE tpcds.warehouse_t16;
vastbase=# DROP TABLE tpcds.warehouse_t17;
vastbase=# DROP TABLE tpcds.warehouse_t18;
vastbase=# DROP TABLE tpcds.warehouse_t20;
vastbase=# DROP TABLE tpcds.warehouse_t21;
vastbase=# DROP TABLE tpcds.warehouse_t22;
vastbase=# DROP TABLE joe.warehouse_t23;
vastbase=# DROP TABLE warehouse_t24;
vastbase=# DROP TABLE warehouse_t25;
vastbase=# DROP TABLESPACE DS_TABLESPACE1;
vastbase=# DROP SCHEMA IF EXISTS joe CASCADE;
```

相关链接

11.16.17ALTER TABLE，11.16.76DROP TABLE，11.16.51CREATE TABLESPACE

优化建议

- UNLOGGED
  - UNLOGGED 表和表上的索引因为数据写入时不通过 WAL 日志机制，写入速度远高于普通表。因此，可以用于缓冲存储复杂查询的中间结果集，增强复杂查询的性能。

- UNLOGGED 表无主备机制，在系统故障或异常断点等情况下，会有数据丢失风险，因此，不可用来存储基础数据。

❖ TEMPORARY | TEMP
- 临时表只在当前会话可见，会话结束后会自动删除。

❖ LIKE
- 新表自动从这个表中继承所有字段名及其数据类型和非空约束，新表与源表之间在创建动作完毕之后是完全无关的。

❖ LIKE INCLUDING DEFAULTS
- 源表上的字段缺省表达式只有在指定 INCLUDING DEFAULTS 时，才会复制到新表中。缺省是不包含缺省表达式的，即新表中的所有字段的缺省值都是 NULL。

❖ LIKE INCLUDING CONSTRAINTS
- 源表上的 CHECK 约束仅在指定 INCLUDING CONSTRAINTS 时，会复制到新表中，而其他类型的约束永远不会复制到新表中。非空约束总是复制到新表中。此规则同时适用于表约束和列约束。

❖ LIKE INCLUDING INDEXES
- 如果指定了 INCLUDING INDEXES，则源表上的索引也将在新表上创建，默认不建立索引。

❖ LIKE INCLUDING STORAGE
- 如果指定了 INCLUDING STORAGE，则复制列的 STORAGE 设置会复制到新表中，默认情况下不包含 STORAGE 设置。

❖ LIKE INCLUDING COMMENTS
- 如果指定了 INCLUDING COMMENTS，则源表列、约束和索引的注释会复制到新表中。默认情况下，不复制源表的注释。

❖ LIKE INCLUDING PARTITION
- 如果指定了 INCLUDING PARTITION，则源表的分区定义会复制到新表中，同时新表将不能再使用 PARTITION BY 子句。默认情况下，不拷贝源表的分区定义。

❖ LIKE INCLUDING RELOPTIONS
- 如果指定了 INCLUDING RELOPTIONS，则源表的存储参数（即源表的 WITH 子句）会复制到新表中。默认情况下，不复制源表的存储参数。

❖ LIKE INCLUDING DISTRIBUTION
- 如果指定了 INCLUDING DISTRIBUTION，则源表的分布信息会复制到新表中，包括分布类型和分布列。默认情况下，不拷贝源表的分布信息。

❖ LIKE INCLUDING ALL
- INCLUDING ALL 包含了 INCLUDING DEFAULTS、INCLUDING CONSTRAINTS、INCLUDING INDEXES、INCLUDING STORAGE、INCLUDING COMMENTS、INCLUDING PARTITION、INCLUDING RELOPTIONS 和 INCLUDING DISTRIBUTION 的内容。

❖ ORIENTATION ROW
- 创建行存表，行存储适合于 OLTP 业务，此类型的表上交互事务比较多，一次交互会涉及

表中的多个列，用行存查询效率较高。

- ORIENTATION COLUMN
  - 创建列存表，列存储适合于数据仓库业务，此类型的表上会做大量的汇聚计算，且涉及的列操作较少。

11.16.49. CREATE TABLE AS

功能描述

根据查询结果创建表。

CREATE TABLE AS 创建一个表并且用来自 SELECT 命令的结果填充该表。该表的字段和 SELECT 输出字段的名称及数据类型相关。不过用户可以通过明确地给出一个字段名称列表来覆盖 SELECT 输出字段的名称。

CREATE TABLE AS 对源表进行一次查询，然后将数据写入新表中，而查询视图结果会根据源表的变化而有所改变。相比之下，每次做查询的时候，视图都重新计算定义它的 SELECT 语句。

注意事项

- 分区表不能采用此方式进行创建。
- 如果在建表过程中数据库系统发生故障，系统恢复后可能无法自动清除之前已创建的、大小非 0 的磁盘文件。此种情况出现概率小，不影响数据库系统的正常运行。

语法格式

```
CREATE [ UNLOGGED ] TABLE table_name
   [ (column_name [, ...] ) ]
   [ WITH ( {storage_parameter = value} [, ... ] ) ]
   [ COMPRESS | NOCOMPRESS ]
   [ TABLESPACE tablespace_name ]
   AS query
   [ WITH [ NO ] DATA ];
```

参数说明

- UNLOGGED

  指定表为非日志表。在非日志表中写入的数据不会被写入到预写日志中，这样就会比普通表快很多。但是，它也是不安全的，非日志表在冲突或异常关机后会被自动删截。非日志表中的内容也不会被复制到备用服务器中。在该类表中创建的索引也不会被自动记录。

  - 使用场景：非日志表不能保证数据的安全性，用户应该在确保数据已经做好备份的前提下使用，例如系统升级时进行数据的备份。
  - 故障处理：当异常关机等操作导致非日志表上的索引发生数据丢失时，用户应该对发生错误的索引进行重建。

- table_name

  要创建的表名。

  取值范围：字符串，要符合标识符的命名规范。

- column_name

  新表中要创建的字段名。

  取值范围：字符串，要符合标识符的命名规范。

- WITH (storage_parameter [= value] [, ...])

  这个子句为表或索引指定一个可选的存储参数。参数的详细说明如下所示。

  - FILLFACTOR

    一个表的填充因子（fillfactor）是一个介于 10 和 100 之间的百分数。100（完全填充）是默认值。如果指定了较小的填充因子，INSERT 操作仅按照填充因子指定的百分率填充表页。每个页上的剩余空间将用于在该页上更新行，这就使得 UPDATE 有机会在同一页上放置同一条记录的新版本，这比把新版本放置在其他页上更有效。对于一个从不更新的表将填充因子设为 100 是最佳选择，但是对于频繁更新的表，选择较小的填充因子则更加合适。该参数只对行存表有效。

    取值范围：10~100

  - ORIENTATION

    取值范围：

    COLUMN：表的数据将以列式存储。

    ROW（缺省值）：表的数据将以行式存储。

  - COMPRESSION

    指定表数据的压缩级别，它决定了表数据的压缩比以及压缩时间。一般来讲，压缩级别越高，压缩比也越大，压缩时间也越长；反之亦然。实际压缩比取决于加载的表数据的分布特征。

    取值范围：

    列存表的有效值为 YES/NO/LOW/MIDDLE/HIGH，默认值为 LOW。

    行存表的有效值为 YES/NO，默认值为 NO。

  - MAX_BATCHROW

    指定了在数据加载过程中一个存储单元可以容纳记录的最大数目。该参数只对列存表有效。

    取值范围：10000~60000

- COMPRESS / NOCOMPRESS

创建一个新表时，需要在创建表语句中指定关键字 COMPRESS，这样，当对该表进行批量插入时就会触发压缩特性。该特性会在页范围内扫描所有元组数据，生成字典、压缩元组数据并进行存储。指定关键字 NOCOMPRESS 则不对表进行压缩。

缺省值：NOCOMPRESS，即不对元组数据进行压缩。

- TABLESPACE tablespace_name

  指定新表将要在 tablespace_name 表空间内创建。如果没有声明，将使用默认表空间。

- AS query

  一个 SELECT VALUES 命令或者一个运行预备好的 SELECT 或 VALUES 查询的 EXECUTE 命令。

- [WITH [NO] DATA]

  创建表时，是否也插入查询到的数据。默认是要数据，选择“NO”参数时，则不要数据。

示例

```
--创建一个表 tpcds.store_returns 表。
vastbase=# CREATE TABLE tpcds.store_returns
(
    W_WAREHOUSE_SK          INTEGER            NOT NULL,
    W_WAREHOUSE_ID          CHAR(16)           NOT NULL,
    sr_item_sk             VARCHAR(20)                ,
    W_WAREHOUSE_SQ_FT        INTEGER
);
--创建一个表 tpcds.store_returns_t1 并插入 tpcds.store_returns 表中 sr_item_sk 字段中大于 16 的数值。
vastbase=# CREATE TABLE tpcds.store_returns_t1 AS SELECT * FROM tpcds.store_returns WHERE sr_item_sk >
'4795';

--使用 tpcds.store_returns 拷贝一个新表 tpcds.store_returns_t2。
vastbase=# CREATE TABLE tpcds.store_returns_t2 AS table tpcds.store_returns;

--删除表。
vastbase=# DROP TABLE tpcds.store_returns_t1 ;
vastbase=# DROP TABLE tpcds.store_returns_t2 ;
vastbase=# DROP TABLE tpcds.store_returns;
```

相关链接

11.16.48CREATE TABLE， SELECT

11.16.50. CREATE TABLE PARTITION

功能描述

创建分区表。分区表是把逻辑上的一张表根据某种方案分成几张物理块进行存储，这张逻辑上的表称之为分区表，物理块称之为分区。分区表是一张逻辑表，不存储数据，数据实际是存储在分区上的。

常见的分区方案有范围分区（Range Partitioning）、哈希分区（Hash Partitioning）、列表分区（List Partitioning）、数值分区（Value Partition）等。目前行存表、列存表仅支持范围分区。

范围分区是根据表的一列或者多列，将要插入表的记录分为若干个范围，这些范围在不同的分区里没有重叠。为每个范围创建一个分区，用来存储相应的数据。

范围分区的分区策略是指记录插入分区的方式。目前范围分区仅支持范围分区策略。

范围分区策略：根据分区键值将记录映射到已创建的某个分区上，如果可以映射到已创建的某一分区上，则把记录插入到对应的分区上，否则给出报错和提示信息。这是最常用的分区策略。

分区可以提供若干好处：

- 某些类型的查询性能可以得到极大提升。特别是表中访问率较高的行位于一个单独分区或少数几个分区上的情况下。分区可以减少数据的搜索空间，提高数据访问效率。
- 当查询或更新一个分区的大部分记录时，连续扫描那个分区而不是访问整个表可以获得巨大的性能提升。
- 如果需要大量加载或者删除的记录位于单独的分区上，则可以通过直接读取或删除那个分区以获得巨大的性能提升，同时还可以避免由于大量 DELETE 导致的 VACUUM 超载（仅范围分区）。

注意事项

有限地支持唯一约束和主键约束，即唯一约束和主键约束的约束键必须包含所有分区键。

语法格式

```
CREATE TABLE [ IF NOT EXISTS ] partition_table_name
( [
   { column_name data_type [ COLLATE collation ] [ column_constraint [ ... ] ]
   | table_constraint
   | LIKE source_table [ like_option [...] ] }[, ... ]
] )
   [ WITH ( {storage_parameter = value} [, ... ] ) ]
   [ COMPRESS | NOCOMPRESS ]
   [ TABLESPACE tablespace_name ]
    PARTITION BY {
       {RANGE (partition_key) ( partition_less_than_item [, ... ] )} |
       {RANGE (partition_key) ( partition_start_end_item [, ... ] )}
   } [ { ENABLE | DISABLE } ROW MOVEMENT ];
```

- 列约束 column_constraint：

```
[ CONSTRAINT constraint_name ]
{ NOT NULL |
  NULL |
  CHECK ( expression ) |
  DEFAULT default_expr |
  UNIQUE index_parameters |
  PRIMARY KEY index_parameters }
[ DEFERRABLE | NOT DEFERRABLE | INITIALLY DEFERRED | INITIALLY IMMEDIATE ]
```

- 表约束 table_constraint：

```
[ CONSTRAINT constraint_name ]
{ CHECK ( expression ) |
```

```
 UNIQUE ( column_name [, ... ] ) index_parameters |
 PRIMARY KEY ( column_name [, ... ] ) index_parameters}
[ DEFERRABLE | NOT DEFERRABLE | INITIALLY DEFERRED | INITIALLY IMMEDIATE ]
```

- like 选项 like_option：

```
{ INCLUDING | EXCLUDING } { DEFAULTS | CONSTRAINTS | INDEXES | STORAGE | COMMENTS | RELOPTIONS
| DISTRIBUTION | ALL }
```

- 索引存储参数 index_parameters：

```
[ WITH ( {storage_parameter = value} [, ... ] ) ]
[ USING INDEX TABLESPACE tablespace_name ]
```

- partition_less_than_item：

```
PARTITION partition_name VALUES LESS THAN ( { partition_value | MAXVALUE } ) [TABLESPACE
tablespace_name]
```

- partition_start_end_item：

```
PARTITION partition_name {
      {START(partition_value) END (partition_value) EVERY (interval_value)} |
      {START(partition_value) END ({partition_value | MAXVALUE})} |
      {START(partition_value)} |
      {END({partition_value | MAXVALUE})}
} [TABLESPACE tablespace_name]
```

参数说明

- IF NOT EXISTS

  如果已经存在相同名称的表，不会抛出一个错误，而会发出一个通知，告知表关系已存在。

- partition_table_name

  分区表的名称。

  取值范围：字符串，要符合标识符的命名规范。

- column_name

  新表中要创建的字段名。

  取值范围：字符串，要符合标识符的命名规范。

- data_type

  字段的数据类型。

- COLLATE collation

  COLLATE 子句指定列的排序规则（该列必须是可排列的数据类型）。如果没有指定，则使用默认的排序规则。

- CONSTRAINT constraint_name

  列约束或表约束的名称。可选的约束子句用于声明约束，新行或者更新的行必须满足这些约束才能成功插入或更新。

  定义约束有两种方法：

- 列约束：作为一个列定义的一部分，仅影响该列。
- 表约束：不和某个列绑在一起，可以作用于多个列。

❖ LIKE source_table [like_option ...]

LIKE 子句声明一个表，新表自动从这个表里面继承所有字段名及其数据类型和非空约束。

和 INHERITS 不同，新表与原来的表之间在创建动作完毕之后是完全无关的。在源表做的任何修改都不会传播到新表中，并且也不可能在扫描源表的时候包含新表的数据。

字段缺省表达式只有在声明了 INCLUDING DEFAULTS 之后才会包含进来。缺省是不包含缺省表达式的，即新表中所有字段的缺省值都是 NULL。

非空约束将总是复制到新表中，CHECK 约束则仅在指定了 INCLUDING CONSTRAINTS 的时候才复制，而其他类型的约束则永远也不会被复制。此规则同时适用于表约束和列约束。

和 INHERITS 不同，被复制的列和约束并不使用相同的名称进行融合。如果明确的指定了相同的名称或者在另外一个 LIKE 子句中，将会报错。

- 如果指定了 INCLUDING INDEXES，则源表上的索引也将在新表上创建，默认不建立索引。
- 如果指定了 INCLUDING STORAGE，则拷贝列的 STORAGE 设置也将被拷贝，默认情况下不包含 STORAGE 设置。
- 如果指定了 INCLUDING COMMENTS，则源表列、约束和索引的注释也会被拷贝过来。默认情况下，不拷贝源表的注释。
- 如果指定了 INCLUDING RELOPTIONS，则源表的存储参数（即源表的 WITH 子句）也将拷贝至新表。默认情况下，不拷贝源表的存储参数。
- 如果指定了 INCLUDING DISTRIBUTION，则新表将拷贝源表的分布信息，包括分布类型和分布列。默认情况下，不拷贝源表的分布信息。
- INCLUDING ALL 是 INCLUDING DEFAULTS INCLUDING CONSTRAINTS INCLUDING INDEXES INCLUDING STORAGE INCLUDING COMMENTS INCLUDING RELOPTIONS INCLUDING DISTRIBUTION 的简写形式。

❖ WITH (storage_parameter [= value] [, ...])

这个子句为表或索引指定一个可选的存储参数。参数的详细描述如下所示：

- FILLFACTOR

  一个表的填充因子（fillfactor）是一个介于 10 和 100 之间的百分数。100（完全填充）是默认值。如果指定了较小的填充因子，INSERT 操作仅按照填充因子指定的百分率填充表页。每个页上的剩余空间将用于在该页上更新行，这就使得 UPDATE 有机会在同一页上放置同一条记录的新版本，这比把新版本放置在其他页上更有效。对于一个从不更新的表将填充因子设为 100 是最佳选择，但是对于频繁更新的表，选择较小的填充因子则更加合适。该参数对于列存表没有意义。

  取值范围：10~100

– ORIENTATION

决定了表的数据的存储方式。

取值范围：

■ COLUMN：表的数据将以列式存储。

■ ROW（缺省值）：表的数据将以行式存储。

须知

orientation 不支持修改。

– COMPRESSION

■ 列存表的有效值为 LOW/MIDDLE/HIGH/YES/NO，压缩级别依次升高，默认值为 LOW。

■ 行存表的有效值为 YES/NO，默认值为 NO。

– MAX_BATCHROW

指定了在数据加载过程中一个存储单元可以容纳记录的最大数目。该参数只对列存表有效。

取值范围：10000~60000

– PARTIAL_CLUSTER_ROWS

指定了在数据加载过程中进行将局部聚簇存储的记录数目。该参数只对列存表有效。

取值范围：其有效值为大于等于 10 万。此值是 MAX_BATCHROW 的倍数。

– DELTAROW_THRESHOLD

预留参数。该参数只对列存表有效。

取值范围：0～9999

❖ COMPRESS / NOCOMPRESS

创建一个新表时，需要在创建表语句中指定关键字 COMPRESS，这样，当对该表进行批量插入时就会触发压缩特性。该特性会在页范围内扫描所有元组数据，生成字典、压缩元组数据并进行存储。指定关键字 NOCOMPRESS 则不对表进行压缩。

缺省值为 NOCOMPRESS，即不对元组数据进行压缩。

❖ TABLESPACE tablespace_name

指定新表将要在 tablespace_name 表空间内创建。如果没有声明，将使用默认表空间。

❖ PARTITION BY RANGE(partition_key)

创建范围分区。partition_key 为分区键的名称。

（1）对于从句是 VALUES LESS THAN 的语法格式：

须知

对于从句是 VALUE LESS THAN 的语法格式，范围分区策略的分区键最多支持 4 列。

该情形下，分区键支持的数据类型为：SMALLINT、INTEGER、BIGINT、DECIMAL、NUMERIC、REAL、DOUBLE PRECISION、CHARACTER VARYING(n)、VARCHAR(n)、CHARACTER(n)、CHAR(n)、CHARACTER、CHAR、TEXT、NVARCHAR2、NAME、TIMESTAMP[(p)] [WITHOUT TIME ZONE]、TIMESTAMP[(p)] [WITH TIME ZONE]、DATE。

（2）对于从句是 START END 的语法格式：

须知

对于从句是 START END 的语法格式，范围分区策略的分区键仅支持 1 列。

该情形下，分区键支持的数据类型为：SMALLINT、INTEGER、BIGINT、DECIMAL、NUMERIC、REAL、DOUBLE PRECISION、TIMESTAMP[(p)] [WITHOUT TIME ZONE]、TIMESTAMP[(p)] [WITH TIME ZONE]、DATE。

- PARTITION partition_name VALUES LESS THAN ({ partition_value | MAXVALUE })

  指定各分区的信息。partition_name 为范围分区的名称。partition_value 为范围分区的上边界，取值依赖于 partition_key 的类型。MAXVALUE 表示分区的上边界，它通常用于设置最后一个范围分区的上边界。

须知

- 每个分区都需要指定一个上边界。
- 分区上边界的类型应当和分区键的类型一致。
- 分区列表是按照分区上边界升序排列的，值较小的分区位于值较大的分区之前。

- PARTITION partition_name {START (partition_value) END (partition_value) EVERY (interval_value)} | {START (partition_value) END (partition_value|MAXVALUE)} | {START(partition_value)} | {END (partition_value | MAXVALUE)}

  指定各分区的信息，各参数意义如下：

  - partition_name：范围分区的名称或名称前缀，除以下情形外（假定其中的 partition_name 是 p1），均为分区的名称。
    - 若该定义是 START+END+EVERY 从句，则语义上定义的分区的名称依次为 p1_1, p1_2, ...。例如对于定义“PARTITION p1 START(1) END(4)

EVERY(1)”，则生成的分区是：[1, 2), [2, 3) 和 [3, 4)，名称依次为 p1_1, p1_2 和 p1_3，即此处的 p1 是名称前缀。

■ 若该定义是第一个分区定义，且该定义有 START 值，则范围（MINVALUE, START）将自动作为第一个实际分区，其名称为 p1_0，然后该定义语义描述的分区名称依次为 p1_1, p1_2, ...。例如对于完整定义“PARTITION p1 START(1), PARTITION p2 START(2)”，则生成的分区是：(MINVALUE, 1), [1, 2) 和 [2, MAXVALUE)，其名称依次为 p1_0, p1_1 和 p2，即此处 p1 是名称前缀，p2 是分区名称。这里 MINVALUE 表示最小值。

– partition_value：范围分区的端点值（起始或终点），取值依赖于 partition_key 的类型，不可是 MAXVALUE。

– interval_value：对[START，END）表示的范围进行切分，interval_value 是指定切分后每个分区的宽度，不可是 MAXVALUE；如果（END-START）值不能整除以 EVERY 值，则仅最后一个分区的宽度小于 EVERY 值。

– MAXVALUE：表示最大值，它通常用于设置最后一个范围分区的上边界。

须知

1. 在创建分区表若第一个分区定义含 START 值，则范围（MINVALUE，START）将自动作为实际的第一个分区。
2. START END 语法需要遵循以下限制：

- 每个 partition_start_end_item 中的 START 值（如果有的话，下同）必须小于其 END 值；
- 相邻的两个 partition_start_end_item，第一个的 END 值必须等于第二个的 START 值；
- 每个 partition_start_end_item 中的 EVERY 值必须是正向递增的，且必须小于（END-START）值；
- 每个分区包含起始值，不包含终点值，即形如：[起始值，终点值)，起始值是 MINVALUE 时则不包含；
- 一个 partition_start_end_item 创建的每个分区所属的 TABLESPACE 一样；
- partition_name 作为分区名称前缀时，其长度不要超过 57 字节，超过时自动截断；
- 在创建、修改分区表时请注意分区表的分区总数不可超过最大限制（32767）；

3. 在创建分区表时 START END 与 LESS THAN 语法不可混合使用。
4. 即使创建分区表时使用 START END 语法，备份（vb_dump）出的 SQL 语句也是 VALUES LESS THAN 语法格式。

❖ { ENABLE | DISABLE } ROW MOVEMENT

行迁移开关。

如果进行 UPDATE 操作时，更新了元组在分区键上的值，造成了该元组所在分区发生变化，就会根据该开关给出报错信息，或者进行元组在分区间的转移。

取值范围：

– ENABLE（缺省值）：行迁移开关打开。

– DISABLE：行迁移开关关闭。

❖ NOT NULL

字段值不允许为 NULL。ENABLE 用于语法兼容，可省略。

- NULL

  字段值允许 NULL ，这是缺省。

  这个子句只是为和非标准 SQL 数据库兼容。不建议使用。

- CHECK (condition) [NO INHERIT]

  CHECK 约束声明一个布尔表达式，每次要插入的新行或者要更新的行的新值必须使表达式结果为真或未知才能成功，否则会抛出一个异常并且不会修改数据库。

  声明为字段约束的检查约束应该只引用该字段的数值，而在表约束里出现的表达式可以引用多个字段。

  用 NO INHERIT 标记的约束将不会传递到子表中去。

  ENABLE 用于语法兼容，可省略。

- DEFAULT default_expr

  DEFAULT 子句给字段指定缺省值。该数值可以是任何不含变量的表达式(不允许使用子查询和对本表中的其他字段的交叉引用)。缺省表达式的数据类型必须和字段类型匹配。

  缺省表达式将被用于任何未声明该字段数值的插入操作。如果没有指定缺省值则缺省值为 NULL 。

- UNIQUE index_parameters

  UNIQUE (column_name [, ...]) index_parameters

  UNIQUE 约束表示表里的一个字段或多个字段的组合必须在全表范围内唯一。

  对于唯一约束，NULL 被认为是互不相等的。

- PRIMARY KEY index_parameters

  PRIMARY KEY (column_name [, ...]) index_parameters

  主键约束声明表中的一个或者多个字段只能包含唯一的非 NULL 值。

  一个表只能声明一个主键。

- DEFERRABLE | NOT DEFERRABLE

  这两个关键字设置该约束是否可推迟。一个不可推迟的约束将在每条命令之后马上检查。可推迟约束可以推迟到事务结尾使用 SET CONSTRAINTS 命令检查。缺省是 NOT DEFERRABLE。目前，UNIQUE 约束和主键约束可以接受这个子句。所有其他约束类型都是不可推迟的。

- INITIALLY IMMEDIATE | INITIALLY DEFERRED

  如果约束是可推迟的，则这个子句声明检查约束的缺省时间。

  - 如果约束是 INITIALLY IMMEDIATE（缺省），则在每条语句执行之后就立即检查它；

– 如果约束是 INITIALLY DEFERRED ，则只有在事务结尾才检查它。

约束检查的时间可以用 SET CONSTRAINTS 命令修改。

❖ USING INDEX TABLESPACE tablespace_name

为 UNIQUE 或 PRIMARY KEY 约束相关的索引声明一个表空间。如果没有提供这个子句，这个索引将在 default_tablespace 中创建，如果 default_tablespace 为空，将使用数据库的缺省表空间。

示例

❖ 示例 1：创建范围分区表 tpcds.web_returns_p1，含有 8 个分区，分区键为 integer 类型。 分区的范围分别为：wr_returned_date_sk< 2450815，2450815<= wr_returned_date_sk< 2451179，2451179<=wr_returned_date_sk< 2451544，2451544 <= wr_returned_date_sk< 2451910，2451910 <= wr_returned_date_sk< 2452275，2452275 <= wr_returned_date_sk< 2452640，2452640 <= wr_returned_date_sk< 2453005，wr_returned_date_sk>=2453005。

```
--创建表 tpcds.web_returns。
vastbase=# CREATE TABLE tpcds.web_returns
(
    W_WAREHOUSE_SK          INTEGER             NOT NULL,
    W_WAREHOUSE_ID          CHAR(16)            NOT NULL,
    W_WAREHOUSE_NAME        VARCHAR(20)                 ,
    W_WAREHOUSE_SQ_FT       INTEGER                     ,
    W_STREET_NUMBER         CHAR(10)                    ,
    W_STREET_NAME           VARCHAR(60)                 ,
    W_STREET_TYPE           CHAR(15)                    ,
    W_SUITE_NUMBER          CHAR(10)                    ,
    W_CITY                  VARCHAR(60)                 ,
    W_COUNTY                VARCHAR(30)                 ,
    W_STATE                 CHAR(2)                     ,
    W_ZIP                   CHAR(10)                    ,
    W_COUNTRY               VARCHAR(20)                 ,
    W_GMT_OFFSET            DECIMAL(5,2)
);
--创建分区表 tpcds.web_returns_p1。
vastbase=# CREATE TABLE tpcds.web_returns_p1
(
    WR_RETURNED_DATE_SK      INTEGER                    ,
    WR_RETURNED_TIME_SK      INTEGER                    ,
    WR_ITEM_SK               INTEGER            NOT NULL,
    WR_REFUNDED_CUSTOMER_SK   INTEGER                   ,
    WR_REFUNDED_CDEMO_SK     INTEGER                    ,
    WR_REFUNDED_HDEMO_SK     INTEGER                    ,
    WR_REFUNDED_ADDR_SK      INTEGER                    ,
    WR_RETURNING_CUSTOMER_SK  INTEGER                   ,
    WR_RETURNING_CDEMO_SK    INTEGER                    ,
    WR_RETURNING_HDEMO_SK    INTEGER                    ,
    WR_RETURNING_ADDR_SK     INTEGER                    ,
    WR_WEB_PAGE_SK           INTEGER                    ,
```

```
    WR_REASON_SK            INTEGER                   ,
    WR_ORDER_NUMBER         BIGINT              NOT NULL,
    WR_RETURN_QUANTITY       INTEGER                   ,
    WR_RETURN_AMT           DECIMAL(7,2)               ,
    WR_RETURN_TAX           DECIMAL(7,2)               ,
    WR_RETURN_AMT_INC_TAX    DECIMAL(7,2)               ,
    WR_FEE                 DECIMAL(7,2)               ,
    WR_RETURN_SHIP_COST      DECIMAL(7,2)               ,
    WR_REFUNDED_CASH        DECIMAL(7,2)               ,
    WR_REVERSED_CHARGE       DECIMAL(7,2)               ,
    WR_ACCOUNT_CREDIT        DECIMAL(7,2)               ,
    WR_NET_LOSS             DECIMAL(7,2)
)
WITH (ORIENTATION = COLUMN,COMPRESSION=MIDDLE)
PARTITION BY RANGE(WR_RETURNED_DATE_SK)
(
       PARTITION P1 VALUES LESS THAN(2450815),
       PARTITION P2 VALUES LESS THAN(2451179),
       PARTITION P3 VALUES LESS THAN(2451544),
       PARTITION P4 VALUES LESS THAN(2451910),
       PARTITION P5 VALUES LESS THAN(2452275),
       PARTITION P6 VALUES LESS THAN(2452640),
       PARTITION P7 VALUES LESS THAN(2453005),
       PARTITION P8 VALUES LESS THAN(MAXVALUE)
);

--从示例数据表导入数据。
vastbase=# INSERT INTO tpcds.web_returns_p1 SELECT * FROM tpcds.web_returns;

--删除分区 P8。
vastbase=# ALTER TABLE tpcds.web_returns_p1 DROP PARTITION P8;

--增加分区 WR_RETURNED_DATE_SK 介于 2453005 和 2453105 之间。
vastbase=# ALTER TABLE tpcds.web_returns_p1 ADD PARTITION P8 VALUES LESS THAN (2453105);

--增加分区 WR_RETURNED_DATE_SK 介于 2453105 和 MAXVALUE 之间。
vastbase=# ALTER TABLE tpcds.web_returns_p1 ADD PARTITION P9 VALUES LESS THAN (MAXVALUE);

--删除分区 P8。
vastbase=# ALTER TABLE tpcds.web_returns_p1 DROP PARTITION FOR (2453005);

--分区 P7 重命名为 P10。
vastbase=# ALTER TABLE tpcds.web_returns_p1 RENAME PARTITION P7 TO P10;

--分区 P6 重命名为 P11。
vastbase=# ALTER TABLE tpcds.web_returns_p1 RENAME PARTITION FOR (2452639) TO P11;

--查询分区 P10 的行数。
vastbase=# SELECT count(*) FROM tpcds.web_returns_p1 PARTITION (P10);
 count
--------
 0
(1 row)
```

```
--查询分区 P1 的行数。
vastbase=# SELECT COUNT(*) FROM tpcds.web_returns_p1 PARTITION FOR (2450815);
 count
--------
 0
(1 row)
```

❖ 示例 2：创建范围分区表 tpcds.web_returns_p2，含有 8 个分区，分区键类型为 integer 类型，其中第 8 个分区上边界为 MAXVALUE。

八个分区的范围分别为： wr_returned_date_sk< 2450815，2450815<= wr_returned_date_sk< 2451179，2451179<=wr_returned_date_sk< 2451544，2451544 <= wr_returned_date_sk< 2451910，2451910 <= wr_returned_date_sk< 2452275，2452275 <= wr_returned_date_sk< 2452640，2452640 <= wr_returned_date_sk< 2453005，wr_returned_date_sk>=2453005。

分区表 tpcds.web_returns_p2 的表空间为 example1；分区 P1 到 P7 没有声明表空间，使用采用分区表 tpcds.web_returns_p2 的表空间 example1；指定分区 P8 的表空间为 example2。

假定数据库节点的数据目录/pg_location/mount1/path1，数据库节点的数据目录/pg_location/mount2/path2，数据库节点的数据目录/pg_location/mount3/path3，数据库节点的数据目录/pg_location/mount4/path4 是 dwsadmin 用户拥有读写权限的空目录。

```
vastbase=# CREATE TABLESPACE example1 RELATIVE LOCATION 'tablespace1/tablespace_1';
vastbase=# CREATE TABLESPACE example2 RELATIVE LOCATION 'tablespace2/tablespace_2';
vastbase=# CREATE TABLESPACE example3 RELATIVE LOCATION 'tablespace3/tablespace_3';
vastbase=# CREATE TABLESPACE example4 RELATIVE LOCATION 'tablespace4/tablespace_4';

vastbase=# CREATE TABLE tpcds.web_returns_p2
(
    WR_RETURNED_DATE_SK      INTEGER                   ,
    WR_RETURNED_TIME_SK      INTEGER                   ,
    WR_ITEM_SK               INTEGER            NOT NULL,
    WR_REFUNDED_CUSTOMER_SK   INTEGER                   ,
    WR_REFUNDED_CDEMO_SK     INTEGER                   ,
    WR_REFUNDED_HDEMO_SK     INTEGER                   ,
    WR_REFUNDED_ADDR_SK      INTEGER                   ,
    WR_RETURNING_CUSTOMER_SK  INTEGER                   ,
    WR_RETURNING_CDEMO_SK    INTEGER                   ,
    WR_RETURNING_HDEMO_SK    INTEGER                   ,
    WR_RETURNING_ADDR_SK     INTEGER                   ,
    WR_WEB_PAGE_SK           INTEGER                  ,
    WR_REASON_SK             INTEGER                  ,
    WR_ORDER_NUMBER          BIGINT              NOT NULL,
    WR_RETURN_QUANTITY       INTEGER                   ,
    WR_RETURN_AMT            DECIMAL(7,2)              ,
    WR RETURN TAX            DECIMAL(7,2)              ,
    WR RETURN AMT INC TAX     DECIMAL(7,2)              ,
    WR FEE                  DECIMAL(7,2)              ,
    WR RETURN SHIP COST       DECIMAL(7,2)              ,
    WR REFUNDED CASH         DECIMAL(7,2)              ,
    WR REVERSED CHARGE       DECIMAL(7,2)              ,
    WR ACCOUNT CREDIT        DECIMAL(7,2)              ,
```

```
   WR_NET_LOSS               DECIMAL(7,2)
)
TABLESPACE example1
PARTITION BY RANGE(WR_RETURNED_DATE_SK)
(
      PARTITION P1 VALUES LESS THAN(2450815),
      PARTITION P2 VALUES LESS THAN(2451179),
      PARTITION P3 VALUES LESS THAN(2451544),
      PARTITION P4 VALUES LESS THAN(2451910),
      PARTITION P5 VALUES LESS THAN(2452275),
      PARTITION P6 VALUES LESS THAN(2452640),
      PARTITION P7 VALUES LESS THAN(2453005),
      PARTITION P8 VALUES LESS THAN(MAXVALUE) TABLESPACE example2
)
ENABLE ROW MOVEMENT;

--以 like 方式创建一个分区表。
vastbase=# CREATE TABLE tpcds.web_returns_p3 (LIKE tpcds.web_returns_p2 INCLUDING PARTITION);

--修改分区 P1 的表空间为 example2。
vastbase=# ALTER TABLE tpcds.web_returns_p2 MOVE PARTITION P1 TABLESPACE example2;

--修改分区 P2 的表空间为 example3。
vastbase=# ALTER TABLE tpcds.web_returns_p2 MOVE PARTITION P2 TABLESPACE example3;

--以 2453010 为分割点切分 P8。
vastbase=# ALTER TABLE tpcds.web_returns_p2 SPLIT PARTITION P8 AT (2453010) INTO
(
      PARTITION P9,
      PARTITION P10
);

--将 P6, P7 合并为一个分区。
vastbase=# ALTER TABLE tpcds.web_returns_p2 MERGE PARTITIONS P6, P7 INTO PARTITION P8;

--修改分区表迁移属性。
vastbase=# ALTER TABLE tpcds.web_returns_p2 DISABLE ROW MOVEMENT;
--删除表和表空间。
vastbase=# DROP TABLE tpcds.web_returns_p1;
vastbase=# DROP TABLE tpcds.web_returns_p2;
vastbase=# DROP TABLE tpcds.web_returns_p3;
vastbase=# DROP TABLESPACE example1;
vastbase=# DROP TABLESPACE example2;
vastbase=# DROP TABLESPACE example3;
vastbase=# DROP TABLESPACE example4;
```

❖ 示例 3：START END 语法创建、修改 Range 分区表。

假定/home/vastbase/startend_tbs1，/home/vastbase/startend_tbs2，/home/vastbase/startend_tbs3，/home/vastbase/startend_tbs4 是 vastbase 用户拥有读写权限的空目录。

```
-- 创建表空间
vastbase=# CREATE TABLESPACE startend_tbs1 LOCATION '/home/vastbase/startend_tbs1';
vastbase=# CREATE TABLESPACE startend_tbs2 LOCATION '/home/vastbase/startend_tbs2';
```

```
vastbase=# CREATE TABLESPACE startend_tbs3 LOCATION '/home/vastbase/startend_tbs3';
vastbase=# CREATE TABLESPACE startend_tbs4 LOCATION '/home/vastbase/startend_tbs4';

-- 创建临时 schema
vastbase=# CREATE SCHEMA tpcds;
vastbase=# SET CURRENT_SCHEMA TO tpcds;

-- 创建分区表，分区键是 integer 类型
vastbase=# CREATE TABLE tpcds.startend_pt (c1 INT, c2 INT)
TABLESPACE startend_tbs1
PARTITION BY RANGE (c2) (
   PARTITION p1 START(1) END(1000) EVERY(200) TABLESPACE startend_tbs2,
   PARTITION p2 END(2000),
   PARTITION p3 START(2000) END(2500) TABLESPACE startend_tbs3,
   PARTITION p4 START(2500),
   PARTITION p5 START(3000) END(5000) EVERY(1000) TABLESPACE startend_tbs4
)
ENABLE ROW MOVEMENT;

-- 查看分区表信息
vastbase=# SELECT relname, boundaries, spcname FROM pg_partition p JOIN pg_tablespace t ON
p.reltablespace=t.oid and p.parentid='tpcds.startend_pt'::regclass ORDER BY 1;
  relname   | boundaries |    spcname
------------+------------+---------------
 p1_0       | {1}        | startend_tbs2
 p1_1       | {201}      | startend_tbs2
 p1_2       | {401}      | startend_tbs2
 p1_3       | {601}      | startend_tbs2
 p1_4       | {801}      | startend_tbs2
 p1_5       | {1000}     | startend_tbs2
 p2         | {2000}     | startend_tbs1
 p3         | {2500}     | startend_tbs3
 p4         | {3000}     | startend_tbs1
 p5_1       | {4000}     | startend_tbs4
 p5_2       | {5000}     | startend_tbs4
 startend_pt |           | startend_tbs1
(12 rows)

-- 导入数据，查看分区数据量
vastbase=# INSERT INTO tpcds.startend_pt VALUES (GENERATE_SERIES(0, 4999), GENERATE_SERIES(0,
4999));
vastbase=# SELECT COUNT(*) FROM tpcds.startend_pt PARTITION FOR (0);
 count
-------
     1
(1 row)

vastbase=# SELECT COUNT(*) FROM tpcds.startend_pt PARTITION (p3);
 count
-------
   500
(1 row)

-- 增加分区：[5000, 5300), [5300, 5600), [5600, 5900), [5900, 6000)
vastbase=# ALTER TABLE tpcds.startend_pt ADD PARTITION p6 START(5000) END(6000) EVERY(300)
```

```
TABLESPACE startend_tbs4;

-- 增加 MAXVALUE 分区：p7
vastbase=# ALTER TABLE tpcds.startend_pt ADD PARTITION p7 END(MAXVALUE);

-- 重命名分区 p7 为 p8
vastbase=# ALTER TABLE tpcds.startend_pt RENAME PARTITION p7 TO p8;

-- 删除分区 p8
vastbase=# ALTER TABLE tpcds.startend_pt DROP PARTITION p8;

-- 重命名 5950 所在的分区为：p71
vastbase=# ALTER TABLE tpcds.startend_pt RENAME PARTITION FOR(5950) TO p71;

-- 分裂 4500 所在的分区[4000, 5000)
vastbase=# ALTER TABLE tpcds.startend pt SPLIT PARTITION FOR(4500) INTO(PARTITION q1 START(4000)
END(5000) EVERY(250) TABLESPACE startend tbs3);

-- 修改分区 p2 的表空间为 startend_tbs4
vastbase=# ALTER TABLE tpcds.startend_pt MOVE PARTITION p2 TABLESPACE startend_tbs4;

-- 查看分区情形
vastbase=# SELECT relname, boundaries, spcname FROM pg_partition p JOIN pg_tablespace t ON
p.reltablespace=t.oid and p.parentid='tpcds.startend_pt'::regclass ORDER BY 1;
   relname   | boundaries |    spcname
-------------+------------+---------------
 p1_0        | {1}        | startend_tbs2
 p1_1        | {201}      | startend_tbs2
 p1_2        | {401}      | startend_tbs2
 p1_3        | {601}      | startend_tbs2
 p1_4        | {801}      | startend_tbs2
 p1_5        | {1000}     | startend_tbs2
 p2          | {2000}     | startend_tbs4
 p3          | {2500}     | startend_tbs3
 p4          | {3000}     | startend_tbs1
 p5_1        | {4000}     | startend_tbs4
 p6_1        | {5300}     | startend_tbs4
 p6_2        | {5600}     | startend_tbs4
 p6_3        | {5900}     | startend_tbs4
 p71         | {6000}     | startend_tbs4
 q1_1        | {4250}     | startend_tbs3
 q1_2        | {4500}     | startend_tbs3
 q1_3        | {4750}     | startend_tbs3
 q1_4        | {5000}     | startend_tbs3
 startend_pt |            | startend_tbs1
(19 rows)

-- 删除表和表空间
vastbase=# DROP SCHEMA tpcds CASCADE;
vastbase=# DROP TABLESPACE startend_tbs1;
vastbase=# DROP TABLESPACE startend_tbs2;
vastbase=# DROP TABLESPACE startend_tbs3;
vastbase=# DROP TABLESPACE startend_tbs4;
```

相关链接

11.16.18ALTER TABLE PARTITION，11.16.76DROP TABLE

11.16.51. CREATE TABLESPACE

功能描述

在数据库中创建一个新的表空间。

注意事项

- 只有系统管理员可以创建表空间。
- 不允许在一个事务块内部执行 CREATE TABLESPACE。
- 执行 CREATE TABLESPACE 失败，如果内部创建目录（文件）操作成功了就会产生残留的目录（文件），重新创建时需要用户手动清理表空间指定的目录下残留的内容。如果在创建过程中涉及到数据目录下的表空间软连接残留，需要先将软连接的残留文件删除，再重新执行 OM 相关操作。
- CREATE TABLESPACE 不支持两阶段事务，如果部分节点执行失败，不支持回滚。
- 创建表空间前的准备工作参考下述参数说明。

语法格式

```
CREATE TABLESPACE tablespace_name
   [ OWNER user_name ] RELATIVE LOCATION 'directory' [ MAXSIZE 'space_size' ]
   [with_option_clause];
```

其中普通表空间的 with_option_clause 为：

```
WITH ( {filesystem= { 'general'| "general" | general} |
   random_page_cost = { 'value ' | value } |
   seq_page_cost = { 'value ' | value }}[,...])
```

参数说明

- tablespace_name

  要创建的表空间名称。

  表空间名称不能和数据 Vastbase 中的其他表空间重名，且名称不能以"pg"开头，这样的名称留给系统表空间使用。

  取值范围：字符串，要符合标识符的命名规范。

- OWNER user_name

  指定该表空间的所有者。缺省时，新表空间的所有者是当前用户。

  只有系统管理员可以创建表空间，但是可以通过 OWNER 子句把表空间的所有权赋给其他非系统管理员。

取值范围：字符串，已存在的用户。

- RELATIVE

  使用相对路径，LOCATION 目录是相对于各个数据库节点数据目录下的。

  目录层次：数据库节点的数据目录/pg_location/相对路径

  相对路径最多指定两层。

- LOCATION directory

  用于表空间的目录，对于目录有如下要求：

  - Vastbase 系统用户必须对该目录拥有读写权限，并且目录为空。如果该目录不存在，将由系统自动创建。
  - 目录必须是绝对路径，目录中不得含有特殊字符（如$,> 等）。
  - 目录不允许指定在数据库数据目录下。
  - 目录需为本地路径。

  取值范围：字符串，有效的目录。

- MAXSIZE 'space_size'

  指定表空间在单个数据库节点上的最大值。

  取值范围：字符串格式为正整数+单位，单位当前支持 K/M/G/T/P。解析后的数值以 K 为单位，且范围不能够超过 64 比特表示的有符号整数，即 1KB~9007199254740991KB。

- random_page_cost

  指定随机读取 page 的开销。

  取值范围：0~1.79769e+308。

  默认值：使用 GUC 参数 random_page_cost 的值。

- seq_page_cost

  指定顺序读取 page 的开销。

  取值范围：0~1.79769e+308。

  默认值：使用 GUC 参数 seq_page_cost 的值。

示例

```
--创建表空间。
vastbase=# CREATE TABLESPACE ds_location1 RELATIVE LOCATION 'tablespace/tablespace_1';

--创建用户 joe。
vastbase=# CREATE ROLE joe IDENTIFIED BY 'Bigdata@123';

--创建用户 jay。
vastbase=# CREATE ROLE jay IDENTIFIED BY 'Bigdata@123';
```

```
--创建表空间，且所有者指定为用户 joe。
vastbase=# CREATE TABLESPACE ds_location2 OWNER joe RELATIVE LOCATION 'tablespace/tablespace_2';

--把表空间 ds_location1 重命名为 ds_location3。
vastbase=# ALTER TABLESPACE ds_location1 RENAME TO ds_location3;

--改变表空间 ds_location2 的所有者。
vastbase=# ALTER TABLESPACE ds_location2 OWNER TO jay;

--删除表空间。
vastbase=# DROP TABLESPACE ds_location2;
vastbase=# DROP TABLESPACE ds_location3;

--删除用户。
vastbase=# DROP ROLE joe;
vastbase=# DROP ROLE jay;
```

相关链接

11.16.36CREATE DATABASE，11.16.48CREATE TABLE，11.16.41CREATE INDEX，11.16.77DROP TABLESPACE，11.16.19ALTER TABLESPACE

优化建议

- create tablespace

  不建议在事务内部创建表空间。

11.16.52. CREATE TEXT SEARCH CONFIGURATION

功能描述

创建新的文本搜索配置。一个文本搜索配置声明一个能将一个字符串划分成符号的文本搜索解析器，加上可以用于确定搜索对哪些标记感兴趣的字典。

注意事项

- 若仅声明分析器，那么新的文本搜索配置初始没有从符号类型到词典的映射， 因此会忽略所有的单词。后面必须调用 ALTER TEXT SEARCH CONFIGURATION 命令创建映射使配置生效。如果声明了 COPY 选项，那么会自动拷贝指定的文本搜索配置的解析器、映射、配置选项等信息。
- 若模式名称已给出，那么文本搜索配置会在声明的模式中创建。否则会在当前模式创建。
- 定义文本搜索配置的用户成为其所有者。
- PARSER 和 COPY 选项是互相排斥的，因为当一个现有配置被复制，其分析器配置也被复制了。
- 若仅声明分析器，那么新的文本搜索配置初始没有从符号类型到词典的映射， 因此会忽略所有的单词。

语法格式

```
CREATE TEXT SEARCH CONFIGURATION name
    ( PARSER = parser_name | COPY = source_config )
    [ WITH ( {configuration_option = value} [, ...] )];
```

参数说明

- name

  要创建的文本搜索配置的名称。该名称可以有模式修饰。

- parser_name

  用于该配置的文本搜索分析器的名称。

- source_config

  要复制的现有文本搜索配置的名称。

- configuration_option

  文本搜索配置的配置参数，主要是针对 parser_name 执行的解析器，或者 source_config 隐含的解析器而言的。

  取值范围：目前共支持 default、ngram 两种类型的解析器，其中 default 类型的解析器没有对应的 configuration_option，ngram 类型解析器对应的 configuration_option 如表 11-53 所示。

表 11-53. ngram 类型解析器对应的配置参数

解析器	配置参数	参数描述	取值范围
ngram	gram_size	分词长度。	正整数，1~4 默认值：2
	punctuation_ignore	是否忽略标点符号。	• true（默认值）：忽略标点符号。 • false：不忽略标点符号。
	grapsymbol_ignore	是否忽略图形化字符。	• true：忽略图形化字符。 • false（默认值）：不忽略图形化字符。

示例

```
--创建文本搜索配置。
vastbase=# CREATE TEXT SEARCH CONFIGURATION ngram2 (parser=ngram) WITH (gram size = 2,
grapsymbol_ignore = false);
```

```
--创建文本搜索配置。
vastbase=# CREATE TEXT SEARCH CONFIGURATION ngram3 (copy=ngram2) WITH (gram_size = 2, grapsymbol_ignore
= false);

--添加类型映射。
vastbase=# ALTER TEXT SEARCH CONFIGURATION ngram2 ADD MAPPING FOR multisymbol WITH simple;

--创建用户 joe。
vastbase=# CREATE USER joe IDENTIFIED BY 'Bigdata@123';

--修改文本搜索配置的所有者。
vastbase=# ALTER TEXT SEARCH CONFIGURATION ngram2 OWNER TO joe;

--修改文本搜索配置的 schema。
vastbase=# ALTER TEXT SEARCH CONFIGURATION ngram2 SET SCHEMA joe;

--重命名文本搜索配置。
vastbase=# ALTER TEXT SEARCH CONFIGURATION joe.ngram2 RENAME TO ngram_2;

--删除类型映射。
vastbase=# ALTER TEXT SEARCH CONFIGURATION joe.ngram_2 DROP MAPPING IF EXISTS FOR multisymbol;

--删除文本搜索配置。
vastbase=# DROP TEXT SEARCH CONFIGURATION joe.ngram_2;
vastbase=# DROP TEXT SEARCH CONFIGURATION ngram3;

--删除 Schema 及用户 joe。
vastbase=# DROP SCHEMA IF EXISTS joe CASCADE;
vastbase=# DROP ROLE IF EXISTS joe;
```

相关链接

11.16.20ALTER TEXT SEARCH CONFIGURATION， 11.16.78DROP TEXT SEARCH CONFIGURATION

11.16.53. CREATE TEXT SEARCH DICTIONARY

功能描述

创建一个新的全文检索词典。词典是一种指定在全文检索时识别特定词并处理的方法。

词典的创建依赖于预定义模板（在系统表 14.2.65PG_TS_TEMPLATE 中定义），支持创建五种类型的词典，分别是 Simple、Ispell、Synonym、Thesaurus、以及 Snowball，每种类型的词典可以完成不同的任务。

注意事项

- 具有 SYSADMIN 权限的用户可以执行创建词典操作，创建该词典的用户自动成为其所有者。

- 临时模式（pg_temp）下不允许创建词典。
- 创建或修改词典之后，任何对于用户自定义的词典定义文件的修改，将不会影响到数据库中的词典。如果需要在数据库中使用这些修改，需使用 ALTER 语句更新对应词典的定义文件。

语法格式

```
CREATE TEXT SEARCH DICTIONARY name (
    TEMPLATE = template
    [, option = value [, ... ]]
);
```

参数说明

- name

  要创建的词典的名称（可指定模式名，否则在当前模式下创建）。

  取值范围：符合标识符命名规范的字符串，且最大长度不超过 63 个字符。

- template

  模板名。

  取值范围：系统表 14.2.65PG_TS_TEMPLATE 中定义的模板：Simple/Synonym/Thesaurus/Ispell/Snowball。

- option

  参数名。与 template 值对应，不同的词典模板具有不同的参数列表，且与指定顺序无关。

  - Simple 词典对应的 option
    - STOPWORDS

      停用词表文件名，默认后缀名为 stop。停用词文件格式为一组 word 列表，每行定义一个停用词。词典处理时，文件中的空行和空格会被忽略，并将 stopword 词组转换为小写形式。

    - ACCEPT

      是否将非停用词设置为已识别。默认值为 true。

      当 Simple 词典设置参数 ACCEPT=true 时，将不会传递任何 token 给后继词典，此时建议将其放置在词典列表的最后。反之，当 ACCEPT=false 时，建议将该 Simple 词典放置在列表中的至少一个词典之前。

    - FILEPATH

      词典文件所在目录。目录可以指定为本地目录和 OBS 目录。其中，本地目录格式为"file://absolute_path"，OBS 目录格式为"obs://bucket_name/path accesskey=ak secretkey=sk region=rg"。默认值为预定义词典文件所在目录。FILEPATH 参数必须和 STOPWORDS 参数同时指定，不允许单独指定。

- Synonym 词典对应的 option
  - SYNONYM

    同义词词典的定义文件名，默认后缀名为 syn。

    文件格式为一组同义词列表，每行格式为"token synonym"，即 token 和其对应的 synonym，中间以空格相连。
  - CASESENSITIVE

    设置是否大小写敏感，默认值为 false，此时词典文件中的 token 和 synonym 均会转为小写形式处理。如果设置为 true，则不会进行小写转换。
  - FILEPATH

    同义词词典文件所在目录。目录可以指定为本地目录和 OBS 目录两种形式。其中，本地目录格式为"file://absolute_path"，OBS 目录格式为"obs://bucket_name/path accesskey=ak secretkey=sk region=rg"。默认值为预定义词典文件所在目录。
- Thesaurus 词典对应的 option
  - DICTFILE

    词典定义文件名，默认后缀名为 ths。

    文件格式为一组同义词列表，每行格式为"sample words : indexed words"，中间冒号（:）作为短语和其替换词间的分隔符。TZ 词典处理时，如果有多个匹配的 sample words，将选择最长匹配输出。
  - DICTIONARY

    用于词规范化的子词典名，必须且仅能定义一个。该词典必须是已经存在的，在检查短语匹配之前使用，用于识别和规范输入文本。

    如果子词典无法识别输入词，将会报错。此时，需要移除该词或者更新子词典使其识别。此外，可在 indexed words 的开头放上一个星号（*）来跳过在其上应用子词典，但是所有 sample words 必须可以被子词典识别。

    如果词典文件定义的 sample words 中，含有子词典中定义的停用词，需要用问号（？）替代停用词。假设 a 和 the 是子词典中所定义的停用词，如下：

    ```
    ? one ? two : swsw
    ```

    上述同义词组定义会匹配"a one the two"以及"the one a two"，这两个短语均会被 swsw 替代输出。
  - FILEPATH

词典定义文件所在目录。目录可以指定为本地目录和 OBS 目录两种形式。其中，本地目录格式为"file://absolute_path"，OBS 目录格式为"obs://bucket_name/path accesskey=ak secretkey=sk region=rg"。默认值为预定义词典文件所在目录。

- Ispell 词典
  - DICTFILE

    词典定义文件名，默认后缀名为 dict。
  - AFFFILE

    词缀文件名，默认后缀名为 affix。
  - STOPWORDS

    停用词文件名，默认后缀名为 stop，文件格式要求与 Simple 类型词典的停用词文件相同。
  - FILEPATH

    词典文件所在目录。可以指定为本地目录和 OBS 目录两种形式。其中，本地目录格式为"file://absolute_path"，OBS 目录格式为"obs://bucket_name/path accesskey=ak secretkey=sk region=rg"。默认值为预定义词典文件所在目录。
- Snowball 词典
  - LANGUAGE

    语言名，标识使用哪种语言的词干分析算法。算法按照对应语言中的拼写规则，缩减输入词的常见变体形式为一个基础词或词干。
  - STOPWORDS

    停用词表文件名，默认后缀名为 stop，文件格式要求与 Simple 类型词典的停用词文件相同。
  - FILEPATH

    词典定义文件所在目录。可以指定为本地目录或者 OBS 目录。其中，本地目录格式为"file://absolute_path"，OBS 目录格式为"obs://bucket_name/path accesskey=ak secretkey=sk region=rg"。默认值为预定义词典文件所在目录。FILEPATH 参数必须和 STOPWORDS 参数同时指定，不允许单独指定。

说明

词典定义文件的文件名仅支持小写字母、数据、下划线混合。

- value

  参数值。如果不是简单的标识符或数字，则参数值必须加单引号（标示符和数字同样可以加上单引号）。

示例

请参见 11.8.7 配置示例一节的示例。

相关链接

11.16.21ALTER TEXT SEARCH DICTIONARY，11.16.53CREATE TEXT SEARCH DICTIONARY

11.16.54. CREATE TRIGGER

功能描述

创建一个触发器。 触发器将与指定的表或视图关联，并在特定条件下执行指定的函数。

注意事项

- 当前仅支持在普通行存表上创建触发器，不支持在列存表、临时表、unlogged 表等类型表上创建触发器。
- 如果为同一事件定义了多个相同类型的触发器，则按触发器的名称字母顺序触发它们。
- 触发器常用于多表间数据关联同步场景，对 SQL 执行性能影响较大，不建议在大数据量同步及对性能要求高的场景中使用。

语法格式

```
CREATE [ CONSTRAINT ] TRIGGER trigger_name { BEFORE | AFTER | INSTEAD OF } { event [ OR ... ] }
   ON table_name
   [ FROM referenced_table_name ]
   { NOT DEFERRABLE | [ DEFERRABLE ] { INITIALLY IMMEDIATE | INITIALLY DEFERRED } }
   [ FOR [ EACH ] { ROW | STATEMENT } ]
   [ WHEN ( condition ) ]
   EXECUTE PROCEDURE function_name ( arguments );
```

其中 event 包含以下几种：

```
   INSERT
   UPDATE [ OF column_name [, ... ] ]
   DELETE
   TRUNCATE
```

参数说明

- CONSTRAINT

  可选项，指定此参数将创建约束触发器，即触发器作为约束来使用。除了可以使用 SET CONSTRAINTS 调整触发器触发的时间之外，这与常规触发器相同。约束触发器必须是 AFTER ROW 触发器。

- trigger_name

触发器名称，该名称不能限定模式，因为触发器自动继承其所在表的模式，且同一个表的触发器不能重名。 对于约束触发器，使用 11.16.106SET CONSTRAINTS 修改触发器行为时也使用此名称。

取值范围：符合标识符命名规范的字符串，且最大长度不超过 63 个字符。

- BEFORE

  触发器函数是在触发事件发生前执行。

- AFTER

  触发器函数是在触发事件发生后执行，约束触发器只能指定为 AFTER。

- INSTEAD OF

  触发器函数直接替代触发事件。

- event

  启动触发器的事件，取值范围包括：INSERT、UPDATE、DELETE 或 TRUNCATE，也可以通过 OR 同时指定多个触发事件。

  对于 UPDATE 事件类型，可以使用下面语法指定列：

  ```
  UPDATE OF column_name1 [, column_name2 ... ]
  ```

  表示只有这些列作为 UPDATE 语句的目标列时，才会启动触发器，但是 INSTEAD OF UPDATE 类型不支持指定列信息。

- table_name

  需要创建触发器的表名称。

  取值范围：数据库中已经存在的表名称。

- referenced_table_name

  约束引用的另一个表的名称。 只能为约束触发器指定，常见于外键约束。由于当前不支持外键，因此不建议使用。

  取值范围：数据库中已经存在的表名称。

- DEFERRABLE | NOT DEFERRABLE

  约束触发器的启动时机，仅作用于约束触发器。这两个关键字设置该约束是否可推迟。

  详细介绍请参见 11.16.48CREATE TABLE。

- INITIALLY IMMEDIATE | INITIALLY DEFERRED

  如果约束是可推迟的，则这个子句声明检查约束的缺省时间，仅作用于约束触发器。

  详细介绍请参见 11.16.48CREATE TABLE。

- FOR EACH ROW | FOR EACH STATEMENT

触发器的触发频率。

- FOR EACH ROW 是指该触发器是受触发事件影响的每一行触发一次。
- FOR EACH STATEMENT 是指该触发器是每个 SQL 语句只触发一次。

未指定时默认值为 FOR EACH STATEMENT。约束触发器只能指定为 FOR EACH ROW。

❖ condition

决定是否实际执行触发器函数的条件表达式。当指定 WHEN 时，只有在条件返回 true 时才会调用该函数。

在 FOR EACH ROW 触发器中，WHEN 条件可以通过分别写入 OLD.column_name 或 NEW.column_name 来引用旧行或新行值的列。 当然，INSERT 触发器不能引用 OLD 和 DELETE 触发器不能引用 NEW。

INSTEAD OF 触发器不支持 WHEN 条件。

WHEN 表达式不能包含子查询。

对于约束触发器，WHEN 条件的评估不会延迟，而是在执行更新操作后立即发生。 如果条件返回值不为 true，则触发器不会排队等待延迟执行。

❖ function_name

用户定义的函数，必须声明为不带参数并返回类型为触发器，在触发器触发时执行。

❖ arguments

执行触发器时要提供给函数的可选的以逗号分隔的参数列表。参数是文字字符串常量，简单的名称和数字常量也可以写在这里，但它们都将被转换为字符串。 请检查触发器函数的实现语言的描述，以了解如何在函数内访问这些参数。

说明

关于触发器种类：

- INSTEAD OF 的触发器必须标记为 FOR EACH ROW，并且只能在视图上定义。
- BEFORE 和 AFTER 触发器作用在视图上时，只能标记为 FOR EACH STATEMENT。
- TRUNCATE 类型触发器仅限 FOR EACH STATEMENT。

表 11-54. 表和视图上支持的触发器种类：

触发时机	触发事件	行级	语句级
BEFORE	INSERT/UPDATE/DELETE	表	表和视图
	TRUNCATE	不支持	表
AFTER	INSERT/UPDATE/DELETE	表	表和视图
	TRUNCATE	不支持	表

触发时机	触发事件	行级	语句级
INSTEAD OF	INSERT/UPDATE/DELETE	视图	不支持
	TRUNCATE	不支持	不支持

表 11-55. PLPgsql 类型触发器函数特殊变量:

变量名	变量含义
NEW	INSERT 及 UPDATE 操作涉及 tuple 信息中的新值，对 DELETE 为空。
OLD	UPDATE 及 DELETE 操作涉及 tuple 信息中的旧值，对 INSERT 为空。
TG_NAME	触发器名称。
TG_WHEN	触发器触发时机(BEFORE/AFTER/INSTEAD OF)。
TG_LEVEL	触发频率（ROW/STATEMENT）。
TG_OP	触发操作（INSERT/UPDATE/DELETE/TRUNCATE）。
TG_RELID	触发器所在表 OID。
TG_RELNAME	触发器所在表名（已废弃，现用 TG_TABLE_NAME 替代）。
TG_TABLE_NAME	触发器所在表名。
TG_TABLE_SCHEMA	触发器所在表的 SCHEMA 信息。
TG_NARGS	触发器函数参数个数。
TG_ARGV[]	触发器函数参数列表。

示例

```
--创建源表及触发表
vastbase=# CREATE TABLE test_trigger_src_tbl(id1 INT, id2 INT, id3 INT);
vastbase=# CREATE TABLE test_trigger_des_tbl(id1 INT, id2 INT, id3 INT);

--创建触发器函数
```

```
vastbase=# CREATE OR REPLACE FUNCTION tri_insert_func() RETURNS TRIGGER AS
         $$
         DECLARE
         BEGIN
                INSERT INTO test_trigger_des_tbl VALUES(NEW.id1, NEW.id2, NEW.id3);
                RETURN NEW;
         END
         $$ LANGUAGE PLPgsql;

vastbase=# CREATE OR REPLACE FUNCTION tri_update_func() RETURNS TRIGGER AS
         $$
         DECLARE
         BEGIN
                UPDATE test_trigger_des_tbl SET id3 = NEW.id3 WHERE id1=OLD.id1;
                RETURN OLD;
         END
         $$ LANGUAGE PLPgsql;

vastbase=# CREATE OR REPLACE FUNCTION TRI_DELETE_FUNC() RETURNS TRIGGER AS
         $$
         DECLARE
         BEGIN
                DELETE FROM test_trigger_des_tbl WHERE id1=OLD.id1;
                RETURN OLD;
         END
         $$ LANGUAGE PLPgsql;

--创建 INSERT 触发器
vastbase=# CREATE TRIGGER insert_trigger
         BEFORE INSERT ON test_trigger_src_tbl
         FOR EACH ROW
         EXECUTE PROCEDURE tri_insert_func();

--创建 UPDATE 触发器
vastbase=# CREATE TRIGGER update_trigger
         AFTER UPDATE ON test_trigger_src_tbl
         FOR EACH ROW
         EXECUTE PROCEDURE tri_update_func();

--创建 DELETE 触发器
vastbase=# CREATE TRIGGER delete trigger
         BEFORE DELETE ON test_trigger_src_tbl
         FOR EACH ROW
         EXECUTE PROCEDURE tri_delete_func();

--执行 INSERT 触发事件并检查触发结果
vastbase=# INSERT INTO test_trigger_src_tbl VALUES(100,200,300);
vastbase=# SELECT * FROM test_trigger_src_tbl;
vastbase=# SELECT * FROM test_trigger_des_tbl;  //查看触发操作是否生效。

--执行 UPDATE 触发事件并检查触发结果
vastbase=# UPDATE test_trigger_src_tbl SET id3=400 WHERE id1=100;
vastbase=# SELECT * FROM test_trigger_src_tbl;
vastbase=# SELECT * FROM test_trigger_des_tbl;  //查看触发操作是否生效
```

```
--执行 DELETE 触发事件并检查触发结果
vastbase=# DELETE FROM test_trigger_src_tbl WHERE id1=100;
vastbase=# SELECT * FROM test_trigger_src_tbl;
vastbase=# SELECT * FROM test_trigger_des_tbl;  //查看触发操作是否生效

--修改触发器
vastbase=# ALTER TRIGGER delete_trigger ON test_trigger_src_tbl RENAME TO delete_trigger_renamed;

--禁用 insert_trigger 触发器
vastbase=# ALTER TABLE test_trigger_src_tbl DISABLE TRIGGER insert_trigger;

--禁用当前表上所有触发器
vastbase=# ALTER TABLE test trigger src tbl DISABLE TRIGGER ALL;

--删除触发器
vastbase=# DROP TRIGGER insert_trigger ON test_trigger_src_tbl;
vastbase=# DROP TRIGGER update_trigger ON test_trigger_src_tbl;
vastbase=# DROP TRIGGER delete_trigger_renamed ON test_trigger_src_tbl;
```

相关链接

11.16.22ALTER TRIGGER，11.16.80DROP TRIGGER，11.16.17ALTER TABLE

11.16.55. CREATE TYPE

功能描述

在当前数据库中定义一种新的数据类型。定义数据类型的用户将成为该数据类型的拥有者。类型只适用于行存表

有四种形式的 CREATE TYPE，分别为：复合类型、基本类型、shell 类型和枚举类型。

- 复合类型

  复合类型由一个属性名和数据类型的列表指定。如果属性的数据类型是可排序的，也可以指定该属性的排序规则。复合类型本质上和表的行类型相同，但是如果只想定义一种类型，使用 CREATE TYPE 避免了创建一个实际的表。单独的复合类型也是很有用的，例如可以作为函数的参数或者返回类型。

  为了能够创建复合类型，必须拥有在其所有属性类型上的 USAGE 特权。

- 基本类型

  用户可以自定义一种新的基本类型（标量类型）。通常来说这些函数必须是底层语言所编写。

- shell 类型

  shell 类型是一种用于后面要定义的类型的占位符，通过发出一个不带除类型名之外其他参数的 CREATE TYPE 命令可以创建这种类型。在创建基本类型时，需要 shell 类型作为一种向前引用。

- 枚举类型

由若干个标签构成的列表，每一个标签值都是一个非空字符串，且字符串长度必须不超过 64 个字节。

注意事项

如果给定一个模式名，那么该类型将被创建在指定的模式中。否则它会被创建在当前模式中。类型名称必须与同一个模式中任何现有的类型或者域相区别（因为表具有相关的数据类型，类型名称也必须与同一个模式中任何现有表的名称不同）。

语法格式

```
CREATE TYPE name AS
    ( [ attribute_name data_type [ COLLATE collation ] [, ... ] ] )

CREATE TYPE name (
    INPUT = input_function,
    OUTPUT = output_function
    [ , RECEIVE = receive_function ]
    [ , SEND = send_function ]
    [ , TYPMOD_IN =
type_modifier_input_function ]
    [ , TYPMOD_OUT =
type_modifier_output_function ]
    [ , ANALYZE = analyze_function ]
    [ , INTERNALLENGTH = { internallength |
VARIABLE } ]
    [ , PASSEDBYVALUE ]
    [ , ALIGNMENT = alignment ]
  [ , STORAGE = storage ]
    [ , LIKE = like_type ]
    [ , CATEGORY = category ]
    [ , PREFERRED = preferred ]
    [ , DEFAULT = default ]
    [ , ELEMENT = element ]
    [ , DELIMITER = delimiter ]
    [ , COLLATABLE = collatable ]
)

CREATE TYPE name

CREATE TYPE name AS ENUM
    ( [ 'label' [, ... ] ] )
```

参数说明

复合类型

- name

  要创建的类型的名称（可以被模式限定）。

- attribute_name

复合类型的一个属性（列）的名称。

- data_type

  要成为复合类型的一个列的现有数据类型的名称。

- collation

  要关联到复合类型的一列的现有排序规则的名称。

基本类型

自定义基本类型时，参数可以以任意顺序出现，input_function 和 output_function 为必选参数，其它为可选参数。

- input_function

  将数据从类型的外部文本形式转换为内部形式的函数名。

  输入函数可以被声明为有一个 cstring 类型的参数，或者有三个类型分别为 cstring、 oid、integer 的参数。

  - cstring 参数是以 C 字符串存在的输入文本。
  - oid 参数是该类型自身的 OID（对于数组类型则是其元素类型的 OID）。
  - integer 参数是目标列的 typmod（如果知道，不知道则将传递 -1）。

  输入函数必须返回一个该数据类型本身的值。通常，一个输入函数应该被声明为 STRICT。 如果不是这样，在读到一个 NULL 输入值时，调用输入函数时第一个参数会是 NULL。在这种情况下，该函数必须仍然返回 NULL，除非调用函数发生了错误（这种情况主要是想支持域输入函数，域输入函数可能需要拒绝 NULL 输入）。

说明

输入和输出函数能被声明为具有新类型的结果或参数是因为：必须在创建新类型之前创建这两个函数。而新类型应该首先被定义为一种 shell type，它是一种占位符类型，除了名称和拥有者之外它没有其他属性。这可以通过不带额外参数的命令 CREATE TYPE name 做到。然后用 C 写的 I/O 函数可以被定义为引用这种 shell type。最后，用带有完整定义的 CREATE TYPE 把该 shell type 替换为一个完全的、合法的类型定义，之后新类型就可以正常使用了。

- output_function

  将数据从类型的内部形式转换为外部文本形式的函数名。

  输出函数必须被声明为有一个新数据类型的参数。输出函数必须返回类型 cstring。对于 NULL 值不会调用输出函数。

- receive_function

  可选参数。将数据从类型的外部二进制形式转换成内部形式的函数名。

  如果没有该函数，该类型不能参与到二进制输入中。二进制表达转换成内部形式代价更低，然而却更容易移植（例如，标准的整数数据类型使用网络字节序作为外部二进制表达，而内部表达是机器本地的字节序）。receive_function 应该执行足够的检查以确保该值是有效的。

接收函数可以被声明为有一个 internal 类型的参数，或者有三个类型分别为 internal、oid、integer 的参数。

- internal 参数是一个指向 StringInfo 缓冲区的指针，其中保存着接收到的字节串。
- oid 和 integer 参数和文本输入函数的相同。

接收函数必须返回一个该数据类型本身的值。通常，一个接收函数应该被声明为 STRICT。如果不是这样，在读到一个 NULL 输入值时调用接收函数时第一个参数会是 NULL。在这种情况下，该函数必须仍然返回 NULL，除非接收函数发生了错误（这种情况主要是想支持域接收函数，域接收函数可能需要拒绝 NULL 输入）。

- send_function

  可选参数。将数据从类型的内部形式转换为外部二进制形式的函数名。

  如果没有该函数，该类型将不能参与到二进制输出中。发送函数必须被声明为有一个新数据类型的参数。发送函数必须返回类型 bytea。对于 NULL 值不会调用发送函数。

- type_modifier_input_function

  可选参数。将类型的修饰符数组转换为内部形式的函数名。

- type_modifier_output_function

  可选参数。将类型的修饰符的内部形式转换为外部文本形式的函数名。

说明

如果该类型支持修饰符（附加在类型声明上的可选约束，例如，char(5)或 numeric(30,2)），则需要可选的 type_modifier_input_function 以及 type_modifier_output_function。Vastbase 允许用户定义的类型有一个或者多个简单常量或者标识符作为修饰符。不过，为了存储在系统目录中，该信息必须能被打包到一个非负整数值中。所声明的修饰符会被以 cstring 数组的形式传递给 type_modifier_input_function。 type_modifier_input_function 必须检查该值的合法性（如果值错误就抛出一个错误），如果值正确，要返回一个非负 integer 值，该值将被存储在“typmod”列中。如果类型没有 type_modifier_input_function 则类型修饰符将被拒绝。type_modifier_output_function 把内部的整数 typmod 值转换回正确的形式用于用户显示。type_modifier_output_function 必须返回一个 cstring 值，该值就是追加到类型名称后的字符串。例如，numeric 的函数可能会返回(30,2)。如果默认的显示格式就是只把存储的 typmod 整数值放在圆括号内，则允许省略 type_modifier_output_function。

- analyze_function

  可选参数。为该数据类型执行统计分析的函数名的可选参数。

  默认情况下，如果该类型有一个默认的 B-tree 操作符类，ANALYZE 将尝试用类型的“equals”和“less-than”操作符来收集统计信息。这种行为对于非标量类型并不合适，因此可以通过指定一个自定义分析函数来覆盖这种行为。分析函数必须被声明为有一个类型为 internal 的参数，并且返回一个 boolean 结果。

- internallength

  可选参数。一个数字常量，用于指定新类型的内部表达的字节长度。默认为变长。

虽然只有 I/O 函数和其他为该类型创建的函数才知道新类型的内部表达的细节，但是内部表达的一些属性必须被向 Vastbase 声明。其中最重要的是 internallength。基本数据类型可以是定长的（这种情况下 internallength 是一个正整数）或者是变长的（把 internallength 设置为 VARIABLE，在内部通过把 typlen 设置为-1 表示）。所有变长类型的内部表达都必须以一个 4 字节整数开始，internallength 定义了总长度。

- PASSEDBYVALUE

  可选参数。表示这种数据类型的值需要被传值而不是传引用。传值的类型必须是定长的，并且它们的内部表达不能超过 Datum 类型（某些机器上是 4 字节，其他机器上是 8 字节）的尺寸。

- alignment

  可选参数。该参数指定数据类型的存储对齐需求。如果被指定，必须是 char、int2、int4 或者 double。默认是 int4。

  允许的值等同于以 1、2、4 或 8 字节边界对齐。要注意变长类型的 alignment 参数必须至少为 4，因为它们需要包含一个 int4 作为它们的第一个组成部分。

- storage

  可选参数。该数据类型的存储策略。

  如果被指定，必须是 plain、external、extended 或者 main。 默认是 plain。

  - plain 指定该类型的数据将总是被存储在线内并且不会被压缩。(对定长类型只允许 plain)
  - extended 指定系统将首先尝试压缩一个长的数据值，并且将在数据仍然太长的情况下把值移出主表行。
  - external 允许值被移出主表， 但是系统将不会尝试对它进行压缩。
  - main 允许压缩，但是不鼓励把值移出主表（如果没有其他办法让行的大小变得合适，具有这种存储策略的数据项仍将被移出主表，但比起 extended 以及 external 项来，这种存储策略的数据项会被优先考虑保留在主表中）。

    除 plain 之外所有的 storage 值都暗示该数据类型的函数能处理被 TOAST 过的值。指定的值仅仅是决定一种可 TOAST 数据类型的列的默认 TOAST 存储策略，用户可以使用 ALTER TABLE SET STORAGE 为列选取其他策略。

- like_type

  可选参数。与新类型具有相同表达的现有数据类型的名称。会从这个类型中复制 internallength、passedbyvalue、alignment 以及 storage 的值（除非在这个 CREATE TYPE 命令的其他地方用显式说明覆盖）。

  当新类型的低层实现是以一种现有的类型为参考时，用这种方式指定表达特别有用。

- category

可选参数。这种类型的分类码（一个 ASCII 字符）。 默认是“用户定义类型”的'U'。为了创建自定义分类， 也可以选择其他 ASCII 字符。

- preferred

  可选参数。如果这种类型是其类型分类中的优先类型则为 TRUE，否则为 FALSE。默认为假。在一个现有类型分类中创建一种新的优先类型要非常谨慎， 因为这可能会导致很大的改变。

说明

category 和 preferred 参数可以被用来帮助控制在混淆的情况下应用哪一种隐式造型。每一种数据类型都属于一个用单个 ASCII 字符命名的分类，并且每一种类型可以是其所属分类中的“首选”。当有助于解决重载函数或操作符时，解析器将优先造型到首选类型（但是只能从同类的其他类型造型）。对于没有隐式转换到或来自任意其他类型的类型，让这些设置保持默认即可。不过，对于有隐式转换的相关类型的组，把它们都标记为属于同一个类别并且选择一种或两种“最常用”的类型作为该类别的首选通常是很有用的。在把一种用户定义的类型增加到一个现有的内建类别（例如，数字或者字符串类型）中时，category 参数特别有用。不过，也可以创建新的全部是用户定义类型的类别。对这样的类别，可选择除大写字母之外的任何 ASCII 字符。

- default

  可选参数。数据类型的默认值。如果被省略，默认值是空。

  如果用户希望该数据类型的列被默认为某种非空值，可以指定一个默认值。默认值可以用 DEFAULT 关键词指定（这样一个默认值可以被附加到一个特定列的显式 DEFAULT 子句覆盖）。

- element

  可选参数。被创建的类型是一个数组，element 指定了数组元素的类型。例如，要定义一个 4 字节整数的数组（int4）， 应指定 ELEMENT = int4。

- delimiter

  可选参数。指定这种类型组成的数组中分隔值的定界符。

  可以把 delimiter 设置为一个特定字符，默认的定界符是逗号（,）。注意定界符是与数组元素类型相关的，而不是数组类型本身相关。

- collatable

  可选参数。如果这个类型的操作可以使用排序规则信息，则为 TRUE。默认为 FALSE。

  如果 collatable 为 TRUE，这种类型的列定义和表达式可能通过使用 COLLATE 子句携带有排序规则信息。在该类型上操作的函数的实现负责真正利用这些信息，仅把类型标记为可排序的并不会让它们自动地去使用这类信息。

- lable

  可选参数。与枚举类型的一个值相关的文本标签，其值为长度不超过 64 个字符的非空字符串。

说明

在创建用户定义类型的时候， Vastbase 会自动创建一个与之关联的数组类型，其名称由该元素类型的名称前缀一个下划线组成。

示例

```
--创建一种复合类型，建表并插入数据以及查询:
vastbase=# CREATE TYPE compfoo AS (f1 int, f2 text);
vastbase=# CREATE TABLE t1_compfoo(a int, b compfoo);
vastbase=# CREATE TABLE t2_compfoo(a int, b compfoo);
vastbase=# INSERT INTO t1_compfoo values(1,(1,'demo'));
vastbase=# INSERT INTO t2_compfoo select * from t1_compfoo;
vastbase=# SELECT (b).f1 FROM t1_compfoo;
vastbase=# SELECT * FROM t1_compfoo t1 join t2_compfoo t2 on (t1.b).f1=(t1.b).f1;

--重命名数据类型:
vastbase=# ALTER TYPE compfoo RENAME TO compfoo1;

--要改变一个用户定义类型 compfoo1 的所有者为 usr1:
CREATE USER usr1 PASSWORD 'Bigdata@123';
vastbase=# ALTER TYPE compfoo1 OWNER TO usr1;

--把用户定义类型 compfoo1 的模式改变为 usr1:
vastbase=# ALTER TYPE compfoo1 SET SCHEMA usr1;

--给一个数据类型增加一个新的属性:
vastbase=# ALTER TYPE usr1.compfoo1 ADD ATTRIBUTE f3 int;

--删除 compfoo1 类型:
vastbase=# DROP TYPE usr1.compfoo1 cascade;

--删除相关表和用户:
vastbase=# DROP TABLE t1_compfoo;
vastbase=# DROP TABLE t2_compfoo;
vastbase=# DROP SCHEMA usr1;
vastbase=# DROP USER usr1;

--创建一个枚举类型
vastbase=# CREATE TYPE bugstatus AS ENUM ('create', 'modify', 'closed');

--添加一个标签值
vastbase=# ALTER TYPE bugstatus ADD VALUE IF NOT EXISTS 'regress' BEFORE 'closed';

--重命名一个标签值
vastbase=# ALTER TYPE bugstatus RENAME VALUE 'create' TO 'new';
```

相关链接

11.16.23ALTER TYPE，11.16.81DROP TYPE

11.16.56. CREATE USER

功能描述

创建一个用户。

注意事项

- 通过 CREATE USER 创建的用户，默认具有 LOGIN 权限；
- 通过 CREATE USER 创建用户的同时系统会在执行该命令的数据库中，为该用户创建一个同名的 SCHEMA；其他数据库中，则不自动创建同名的 SCHEMA；用户可使用 CREATE SCHEMA 命令，分别在其他数据库中，为该用户创建同名 SCHEMA。
- 系统管理员在普通用户同名 schema 下创建的对象，所有者为 schema 的同名用户（非系统管理员）。

语法格式

```
CREATE USER user_name [ [ WITH ] option [ ... ] ] [ ENCRYPTED | UNENCRYPTED ] { PASSWORD | IDENTIFIED BY } { 'password' | DISABLE };
```

其中 option 子句用于设置权限及属性等信息。

```
{SYSADMIN | NOSYSADMIN}
   | {MONADMIN | NOMONADMIN}
   | {OPRADMIN | NOOPRADMIN}
   | {POLADMIN | NOPOLADMIN}
   | {AUDITADMIN | NOAUDITADMIN}
   | {CREATEDB | NOCREATEDB}
   | {USEFT | NOUSEFT}
   | {CREATEROLE | NOCREATEROLE}
   | {INHERIT | NOINHERIT}
   | {LOGIN | NOLOGIN}
   | {REPLICATION | NOREPLICATION}
   | {INDEPENDENT | NOINDEPENDENT}
   | {VCADMIN | NOVCADMIN}
   | CONNECTION LIMIT connlimit
   | VALID BEGIN 'timestamp'
   | VALID UNTIL 'timestamp'
   | RESOURCE POOL 'respool'
   | PERM SPACE 'spacelimit'
   | TEMP SPACE 'tmpspacelimit'
   | SPILL SPACE 'spillspacelimit'
   | IN ROLE role_name [, ...]
   | IN GROUP role_name [, ...]
   | ROLE role_name [, ...]
   | ADMIN role_name [, ...]
   | USER role_name [, ...]
   | SYSID uid
   | DEFAULT TABLESPACE tablespace_name
   | PROFILE DEFAULT
   | PROFILE profile_name
   | PGUSER
```

参数说明

- user_name

  用户名称。

取值范围：字符串，要符合标识符的命名规范。且最大长度不超过 63 个字符。

- password

  登录密码。

  密码规则如下：

  - 密码默认不少于 8 个字符。
  - 不能与用户名及用户名倒序相同。
  - 至少包含大写字母（A-Z），小写字母（a-z），数字（0-9），非字母数字字符（限定为~!@#$%^&*()-_=+\|[{}];:,<.>/?）四类字符中的三类字符。
  - 创建用户时，应当使用双引号或单引号将用户密码括起来。

  取值范围：字符串。

CREATE USER 的其他参数值请参考 11.16.44CREATE ROLE。

示例

```
--创建用户 jim，登录密码为 Bigdata@123。
vastbase=# CREATE USER jim PASSWORD 'Bigdata@123';

--下面语句与上面的等价。
vastbase=# CREATE USER kim IDENTIFIED BY 'Bigdata@123';

--如果创建有“创建数据库”权限的用户，则需要加 CREATEDB 关键字。
vastbase=# CREATE USER dim CREATEDB PASSWORD 'Bigdata@123';

--将用户 jim 的登录密码由 Bigdata@123 修改为 Abcd@123。
vastbase=# ALTER USER jim IDENTIFIED BY 'Abcd@123' REPLACE 'Bigdata@123';

--为用户 jim 追加 CREATEROLE 权限。
vastbase=# ALTER USER jim CREATEROLE;

--将 enable_seqscan 的值设置为 on， 设置成功后，在下一会话中生效。
vastbase=# ALTER USER jim SET enable_seqscan TO on;

--重置 jim 的 enable_seqscan 参数。
vastbase=# ALTER USER jim RESET enable_seqscan;

--锁定 jim 帐户。
vastbase=# ALTER USER jim ACCOUNT LOCK;

--删除用户。
vastbase=# DROP USER kim CASCADE;
vastbase=# DROP USER jim CASCADE;
vastbase=# DROP USER dim CASCADE;
```

相关链接

11.16.24ALTER USER，11.16.44CREATE ROLE，11.16.82DROP USER

11.16.57. CREATE VIEW

功能描述

创建一个视图。视图与基本表不同，是一个虚拟的表。数据库中仅存放视图的定义，而不存放视图对应的数据，这些数据仍存放在原来的基本表中。若基本表中的数据发生变化，从视图中查询出的数据也随之改变。从这个意义上讲，视图就像一个窗口，透过它可以看到数据库中用户感兴趣的数据及变化。

注意事项

无。

语法格式

```
CREATE [ OR REPLACE ] [ TEMP | TEMPORARY ] VIEW view_name [ ( column_name [, ...] ) ]
    [ WITH ( {view_option_name [= view_option_value]} [, ... ] ) ]
    AS query;
```

说明

创建视图时使用 WITH(security_barriers)可以创建一个相对安全的视图，避免攻击者利用低成本函数的 RAISE 语句打印出隐藏的基表数据。

参数说明

- OR REPLACE

  如果视图已存在，则重新定义。

- TEMP | TEMPORARY

  创建临时视图。

- view_name

  要创建的视图名称。可以用模式修饰。

  取值范围：字符串，符合标识符命名规范。

- column_name

  可选的名称列表，用作视图的字段名。如果没有给出，字段名取自查询中的字段名。

  取值范围：字符串，符合标识符命名规范。

- view_option_name [= view_option_value]

  该子句为视图指定一个可选的参数。

  目前 view_option_name 支持的参数仅有 security_barrier，当 VIEW 试图提供行级安全时，应使用该参数。

  取值范围：Boolean 类型，TRUE、FALSE

- query

  为视图提供行和列的 SELECT 或 VALUES 语句。

示例

```
--创建字段 spcname 为 pg_default 组成的视图。
vastbase=# CREATE VIEW myView AS
   SELECT * FROM pg_tablespace WHERE spcname = 'pg_default';

--查看视图。
vastbase=# SELECT * FROM myView ;

--删除视图 myView。
vastbase=# DROP VIEW myView;
--创建物化视图。
--创建表 student
vastbase=# create  table student(
vastbase(# student_no int4 primary key,
vastbase(# student_name varchar(30),
vastbase(# age int2 );
CREATE TABLE
vastbase=# insert into student values(1,'xiaoming',12);
INSERT 0 1
vastbase=# insert into student values(2,'xiaohong',11);
INSERT 0 1
-- 创建视图
vastbase=# create view v_student as select * from student;
CREATE VIEW
-- 创建物化视图
vastbase=# create materialized view mv_student as select * from student;
SELECT 2
Vastbase=# select * from mv_student ;
Student_no | student_name   |  age
-----------+----------------+-------
     1     | xiaoming       | 12
     2     | xiaohong       | 11
(2 rows)
```

相关链接

11.16.25ALTER VIEW，11.16.83DROP VIEW

11.16.58. CURSOR

功能描述

CURSOR 命令定义一个游标，用于在一个大的查询里面检索少数几行数据。

为了处理 SQL 语句，存储过程进程分配一段内存区域来保存上下文联系。游标是指向上下文区域的句柄或指针。借助游标，存储过程可以控制上下文区域的变化。

注意事项

- 游标命令只能在事务块里使用。
- 通常游标和 SELECT 一样返回文本格式。因为数据在系统内部是用二进制格式存储的，系统必须对数据做一定转换以生成文本格式。一旦数据是以文本形式返回，客户端应用需要把它们转换成二进制进行操作。使用 FETCH 语句，游标可以返回文本或二进制格式。
- 应该小心使用二进制游标。文本格式一般都比对应的二进制格式占用的存储空间大。二进制游标返回内部二进制形态的数据，可能更易于操作。如果想以文本方式显示数据，则以文本方式检索会为用户节约很多客户端的工作。比如，如果查询从某个整数列返回 1，在缺省的游标里将获得一个字符串 1，但在二进制游标里将得到一个 4 字节的包含该数值内部形式的数值（大端顺序）。

语法格式

```
CURSOR cursor_name
    [ BINARY ]  [ NO SCROLL ]  [ { WITH | WITHOUT } HOLD ]
    FOR query ;
```

参数说明

- cursor_name

  将要创建的游标名。

  取值范围：遵循数据库对象命名规范。

- BINARY

  指明游标以二进制而不是文本格式返回数据。

- NO SCROLL

  声明游标检索数据行的方式。

  - NO SCROLL：声明该游标不能用于以倒序的方式检索数据行。
  - 未声明：根据执行计划的不同，自动判断该游标是否可以用于以倒序的方式检索数据行。

- WITH HOLD | WITHOUT HOLD

  声明当创建游标的事务结束后，游标是否能继续使用。

  - WITH HOLD：声明该游标在创建它的事务结束后仍可继续使用。
  - WITHOUT HOLD：声明该游标在创建它的事务之外不能再继续使用，此游标将在事务结束时被自动关闭。
  - 如果不指定 WITH HOLD 或 WITHOUT HOLD，默认行为是 WITHOUT HOLD。
  - 跨节点事务不支持 WITH HOLD（例如在多 DBnode 部署 Vastbase 中所创建的含有 DDL 的事务属于跨节点事务）。

- query

  使用 SELECT 或 VALUES 子句指定游标返回的行。

取值范围：SELECT 或 VALUES 子句。

示例

请参考 FETCH 的示例。

相关链接

11.16.87FETCH

11.16.59. DEALLOCATE

功能描述

DEALLOCATE 用于删除前面编写的预备语句。如果用户没有明确删除一个预备语句，那么它将在会话结束的时候被删除。

PREPARE 关键字总被忽略。

注意事项

无。

语法格式

```
DEALLOCATE [ PREPARE ] { name | ALL };
```

参数说明

- name

  将要删除的预备语句。

- ALL

  删除所有预备语句。

示例

无。

11.16.60. DECLARE

功能描述

DECLARE 命令既可以定义一个游标，用于在一个大的查询里面检索少数几行数据，也可以作为一个匿名块的开始。

本节主要描述定义为游标的用法，开启匿名块的用法见 11.16.27BEGIN。

为了处理 SQL 语句，存储过程进程分配一段内存区域来保存上下文联系。游标是指向上下文区域的句柄或指针。借助游标，存储过程可以控制上下文区域的变化。

通常游标和 SELECT 一样返回文本格式。因为数据在系统内部是用二进制格式存储的，系统必须对数据做一定转换以生成文本格式。一旦数据是以文本形式返回，客户端应用需要把它们转换成二进制进行操作。使用 FETCH 语句，游标可以返回文本或二进制格式。

注意事项

- 游标命令只能在事务块里使用。
- 应该小心使用二进制游标。文本格式一般都比对应的二进制格式占用的存储空间大。二进制游标返回内部二进制形态的数据，可能更易于操作。如果想以文本方式显示数据，则以文本方式检索会为用户节约很多客户端的工作。比如，如果查询从某个整数列返回 1，在缺省的游标里将获得一个字符串 1，但在二进制游标里将得到一个 4 字节的包含该数值内部形式的数值（大端顺序）。

语法格式

- 定义游标

```
DECLARE cursor_name [ BINARY ] [ NO SCROLL ]
    CURSOR [ { WITH | WITHOUT } HOLD ] FOR query ;
```

- 开启匿名块

```
[DECLARE [declare_statements]]
BEGIN
execution_statements
END;
/
```

参数说明

- cursor_name

  将要创建的游标名。

  取值范围：遵循数据库对象命名规范。

- BINARY

  指明游标以二进制而不是文本格式返回数据。

- NO SCROLL

  声明游标检索数据行的方式。

  - NO SCROLL：声明该游标不能用于以倒序的方式检索数据行。
  - 未声明：根据执行计划的不同，自动判断该游标是否可以用于以倒序的方式检索数据行。

- WITH HOLD

  WITHOUT HOLD

声明当创建游标的事务结束后，游标是否能继续使用。

- WITH HOLD：声明该游标在创建它的事务结束后仍可继续使用。
- WITHOUT HOLD：声明该游标在创建它的事务之外不能再继续使用，此游标将在事务结束时被自动关闭。
- 如果不指定 WITH HOLD 或 WITHOUT HOLD，默认行为是 WITHOUT HOLD。

❖ query

使用 SELECT 或 VALUES 子句指定游标返回的行。

取值范围：SELECT 或 VALUES 子句。

❖ declare_statements

声明变量，包括变量名和变量类型，如“sales_cnt int”。

❖ execution_statements

匿名块中要执行的语句。

取值范围：已存在的函数名称。

示例

定义游标示例请参考 FETCH 的示例。

相关链接

11.16.27BEGIN，11.16.87FETCH

11.16.61. DELETE

功能描述

DELETE 从指定的表里删除满足 WHERE 子句的行。如果 WHERE 子句不存在，将删除表中所有行，结果只保留表结构。

注意事项

❖ 要删除表中的数据，用户必须对它有 DELETE 权限。同样也必须有 USING 子句引用的表以及 condition 上读取的表的 SELECT 权限。

❖ 对于行存复制表，仅支持有主键约束场景下的 delete 操作。

❖ 对于列存表，暂时不支持 RETURNING 子句。

语法格式

```
[ WITH [ RECURSIVE ] with_query [, ...] ]
DELETE FROM [ ONLY ] table_name [ * ] [ [ AS ] alias ]
```

```
[ USING using_list ]
[ WHERE condition | WHERE CURRENT OF cursor_name ]
[ RETURNING { * | { output_expr [ [ AS ] output_name ] } [, ...] } ];
```

参数说明

- WITH [RECURSIVE] with_query [, ...]

  用于声明一个或多个可以在主查询中通过名称引用的子查询，相当于临时表。

  如果声明了 RECURSIVE，那么允许 SELECT 子查询通过名称引用它自己。

  其中 with_query 的详细格式为：

  with_query_name [(column_name [, ...])] AS

  ({select | values | insert | update | delete})

  – with_query_name 指定子查询生成的结果集名称，在查询中可使用该名称访问子查询的结果集。

  – column_name 指定子查询结果集中显示的列名。

  – 每个子查询可以是 SELECT，VALUES，INSERT，UPDATE 或 DELETE 语句。

- ONLY

  如果指定 ONLY 则只有该表被删除；如果没有声明，则该表和它的所有子表将都被删除。

- table_name

  目标表的名称（可以有模式修饰）。

  取值范围：已存在的表名。

- alias

  目标表的别名。

  取值范围：字符串，符合标识符命名规范。

- using_list

  using 子句。

- condition

  一个返回 Boolean 值的表达式，用于判断哪些行需要被删除。

- WHERE CURRENT OF cursor_name

  当前不支持，仅保留语法接口。

- output_expr

  DELETE 命令删除行之后计算输出结果的表达式。该表达式可以使用表的任意字段。可以使用* 返回被删除行的所有字段。

- output_name

  一个字段的输出名称。

  取值范围：字符串，符合标识符命名规范。

示例

```
--创建表 tpcds.customer_address。
vastbase=# CREATE TABLE tpcds.customer_address
(
   ca_address_sk      integer               NOT NULL   ,
   ca_address_id      character(16)          NOT NULL
);

--在 tpcds.customer_address 表插入数据
vastbase=# INSERT INTO tpcds.customer_address VALUES (3769, 'hello');
vastbase=# INSERT INTO tpcds.customer_address VALUES (14889, 'hi');

--创建表 tpcds.customer_address_bak。
vastbase=# CREATE TABLE tpcds.customer_address_bak AS TABLE tpcds.customer_address;

--删除 tpcds.customer_address_bak 中 ca_address_sk 小于 14888 的职员。
vastbase=# DELETE FROM tpcds.customer_address_bak WHERE ca_address_sk < 14888;

--删除 tpcds.customer_address_bak 中所有数据。
vastbase=# DELETE FROM tpcds.customer_address_bak;

--删除 tpcds.customer_address_bak 表。
vastbase=# DROP TABLE tpcds.customer_address_bak;

--删除 tpcds.customer_address 表。
vastbase=# DROP TABLE tpcds.customer_address;
```

优化建议

- delete

  如果要删除表中的所有记录，建议使用 truncate 语法。

11.16.62. DO

功能描述

执行匿名代码块。

代码块被看做是没有参数的一段函数体，返回值类型是 void。它的解析和执行是同一时刻发生的。

注意事项

- 程序语言在使用之前，必须通过命令 CREATE LANGUAGE 安装到当前的数据库中。 plpgsql 是默认的安装语言，其它语言安装时必须指定。
- 如果语言是不受信任的，用户必须有使用程序语言的 USAGE 权限，或者是系统管理员。

语法格式

```
DO [ LANGUAGE lang_name ] code;
```

参数说明

- lang_name

  用来解析代码的程序语言的名称，如果缺省，默认的语言是 plpgsql。

- code

  程序语言代码可以被执行的。程序语言必须指定为字符串才行。

示例

```
--创建用户 webuser。
vastbase=# CREATE USER webuser PASSWORD 'Bigdata@123';

--授予用户 webuser 对模式 tpcds 下视图的所有操作权限。
vastbase=# DO $$DECLARE r record;
BEGIN
   FOR r IN SELECT c.relname table_name,n.nspname table_schema FROM pg_class c,pg_namespace n
          WHERE c.relnamespace = n.oid AND n.nspname = 'tpcds' AND relkind IN ('r','v')
   LOOP
      EXECUTE 'GRANT ALL ON ' || quote_ident(r.table_schema) || '.' || quote_ident(r.table_name) ||
' TO webuser';
   END LOOP;
END$$;


--删除用户 webuser。
vastbase=# DROP USER webuser CASCADE;
```

11.16.63. DROP DATABASE

功能描述

删除一个数据库。

注意事项

- 只有数据库所有者有权限执行 DROP DATABASE 命令，系统管理员默认拥有此权限。

- 不能对系统默认安装的三个数据库（POSTGRES、TEMPLATE0 和 TEMPLATE1）执行删除操作，系统做了保护。如果想查看当前服务中有哪几个数据库，可以用 vsql 的\l 命令查看。
- 如果有用户正在与要删除的数据库连接，则删除操作失败。
- 不能在事务块中执行 DROP DATABASE 命令。
- 如果执行 DROP DATABASE 失败，事务回滚，需要再次执行一次 DROP DATABASE IF EXISTS。

须知

DROP DATABASE 一旦执行将无法撤销，请谨慎使用。

语法格式

```
DROP DATABASE [ IF EXISTS ] database_name ;
```

参数说明

- IF EXISTS

  如果指定的数据库不存在，则发出一个 notice 而不是抛出一个错误。

- database_name

  要删除的数据库名称。

  取值范围：字符串，已存在的数据库名称。

示例

请参见 CREATE DATABASE 的示例。

相关链接

11.16.36CREATE DATABASE

优化建议

- drop database

  不支持在事务中删除 database。

11.16.64. DROP DATA SOURCE

功能描述

删除一个 Data Source 对象。

注意事项

只有属主/系统管理员/初始用户才可以删除一个 Data Source 对象。

语法格式

```
DROP DATA SOURCE [IF EXISTS] src_name [CASCADE | RESTRICT];
```

参数说明

- src_name

  待删除的 Data Source 对象名称。

  取值范围：字符串，符合标识符命名规范。

- IF EXISTS

  如果指定的 Data Source 不存在，则发出一个 notice 而不是报错。

- CASCADE | RESTRICT
  - CASCADE：表示允许级联删除依赖于 Data Source 的对象
  - RESTRICT（缺省值）：表示有依赖于该 Data Source 的对象存在，则该 Data Source 无法删除。

    目前 Data Source 对象没有被依赖的对象，CASCADE 和 RESTRICT 效果一样，保留此选项是为了向后兼容性。

示例

```
--创建 Data Source 对象。
vastbase=# CREATE DATA SOURCE ds_tst1;

--删除 Data Source 对象。
vastbase=# DROP DATA SOURCE ds_tst1 CASCADE;
vastbase=# DROP DATA SOURCE IF EXISTS ds_tst1 RESTRICT;
```

相关链接

11.16.37CREATE DATA SOURCE，11.16.3ALTER DATA SOURCE

11.16.65. DROP DIRECTORY

功能描述

删除指定的 directory 对象。

注意事项

默认只有初始化用户可以执行 drop 操作，当 enable_access_server_directory 开启时（可参考 enable_access_server_directory），sysadmin 权限的用户也可以执行 drop 操作。

语法格式

```
DROP DIRECTORY [ IF EXISTS ] directory_name;
```

参数说明

- directory_name

  目录名称。

  取值范围：已经存在的目录名。

示例

```
--创建目录。
vastbase=# CREATE OR REPLACE DIRECTORY  dir  as '/tmp/';

--删除目录。
vastbase=# DROP DIRECTORY dir;
```

相关链接

11.16.38CREATE DIRECTORY，11.16.5ALTER DIRECTORY

11.16.66. DROP FUNCTION

功能描述

删除一个已存在的函数。

注意事项

如果函数中涉及对临时表相关操作，则无法使用 DROP FUNCTION 删除函数。

语法格式

```
DROP FUNCTION [ IF EXISTS ] function_name
[ ( [ {[ argmode ] [ argname ] argtype} [, ...] ] ) [ CASCADE | RESTRICT ] ];
```

参数说明

- IF EXISTS

  IF EXISTS 表示，如果函数存在则执行删除操作，函数不存在也不会报错，只是发出一个 notice。

- function_name

  要删除的函数名称。

  取值范围：已存在的函数名。

- argmode

  函数参数的模式。

- argname

  函数参数的名称。

- argtype

  函数参数的类型

示例

请参见的示例。

相关链接

11.16.6ALTER FUNCTION，11.16.39CREATE FUNCTION

11.16.67. DROP GROUP

功能描述

删除用户组。

DROP GROUP 是 DROP ROLE 的别名。

注意事项

DROP GROUP 是 Vastbase 管理工具封装的内部接口，用来实现 Vastbase 管理。该接口不建议用户直接使用，以免对 Vastbase 状态造成影响。

语法格式

```
DROP GROUP [ IF EXISTS ] group_name [, ...];
```

参数说明

请参见 DROP ROLE 的参数说明。

相关链接

11.16.40CREATE GROUP，11.16.7ALTER GROUP，11.16.72DROP ROLE

11.16.68. DROP INDEX

功能描述

删除索引。

注意事项

只有索引的所有者有权限执行 DROP INDEX 命令，系统管理员默认拥有此权限。

语法格式

```
DROP INDEX [ CONCURRENTLY ] [ IF EXISTS ]
   index_name [, ...] [ CASCADE | RESTRICT ];
```

参数说明

- CONCURRENTLY

  以不加锁的方式删除索引。删除索引时，一般会阻塞其他语句对该索引所依赖表的访问。加此关键字，可实现删除过程中不做阻塞。

  此选项只能指定一个索引的名称， 并且 CASCADE 选项不支持。

  普通 DROP INDEX 命令可以在事务内执行，但是 DROP INDEX CONCURRENTLY 不可以在事务内执行。

- IF EXISTS

  如果指定的索引不存在，则发出一个 notice 而不是抛出一个错误。

- index_name

  要删除的索引名。

  取值范围：已存在的索引。

- CASCADE | RESTRICT
  - CASCADE：表示允许级联删除依赖于该索引的对象。
  - RESTRICT（缺省值）：表示有依赖与此索引的对象存在，则该索引无法被删除。

示例

请参见 CREATE INDEX 的示例。

相关链接

11.16.8ALTER INDEX，11.16.41CREATE INDEX

11.16.69. DROP OWNED

功能描述

删除一个数据库角色所拥有的数据库对象。

注意事项

- 所有该角色在当前数据库里和共享对象（数据库，表空间） 上的所有对象上的权限都将被撤销。
- DROP OWNED 常常被用来为移除一个或者多个角色做准备。因为 DROP OWNED 只影响当前数据库中的对象，通常需要在包含将被移除角色所拥有的对象的每一个数据库中都执行这个命令。
- 使用 CASCADE 选项可能导致这个命令递归去删除由其他用户所拥有的对象。
- 角色所拥有的数据库、表空间将不会被移除。

语法格式

```
DROP OWNED BY name [, ...] [ CASCADE | RESTRICT ];
```

参数说明

- name

  角色名。

- CASCADE | RESTRICT
  - CASCADE：级联删除所有依赖于被删除对象的对象。
  - RESTRICT（缺省值）：拒绝删除那些有任何依赖对象存在的对象。

相关链接

11.16.95REASSIGN OWNED , 11.16.72DROP ROLE

11.16.70. DROP ROW LEVEL SECURITY POLICY

功能描述

删除表上某个行访问控制策略。

注意事项

仅表的所有者或者管理员用户才能删除表的行访问控制策略。

语法格式

```
DROP [ ROW LEVEL SECURITY ] POLICY [ IF EXISTS ] policy_name ON table_name [ CASCADE | RESTRICT ]
```

参数说明

- IF EXISTS

  如果指定的行访问控制策略不存在，发出一个 notice 而不是抛出一个错误。

- policy_name

  要删除的行访问控制策略的名称。

  - table_name

    行访问控制策略所在的数据表名。

  - CASCADE/RESTRICT

    仅适配此语法，无对象依赖于该行访问控制策略，CASCADE 和 RESTRICT 效果相同。

示例

```
--创建数据表 all_data
vastbase=# CREATE TABLE all_data(id int, role varchar(100), data varchar(100));

--创建行访问控制策略
vastbase=# CREATE ROW LEVEL SECURITY POLICY all_data_rls ON all_data USING(role = CURRENT_USER);

--删除行访问控制策略
vastbase=# DROP ROW LEVEL SECURITY POLICY all_data_rls ON all_data;
```

相关链接

11.16.11ALTER ROW LEVEL SECURITY POLICY，11.16.42CREATE ROW LEVLEL SECURITY POLICY

11.16.71. DROP PROCEDURE

功能描述

删除已存在的存储过程。

注意事项

无。

语法格式

```
DROP PROCEDURE [ IF EXISTS ] procedure_name ;
```

参数说明

- IF EXISTS

  如果指定的存储过程不存在，发出一个 notice 而不是抛出一个错误。

- procedure_name

  要删除的存储过程名称。

  取值范围：已存在的存储过程名。

相关链接

11.16.43CREATE PROCEDURE

11.16.72. DROP ROLE

功能描述

删除指定的角色。

注意事项

无。

语法格式

```
DROP ROLE [ IF EXISTS ] role_name [, ...];
```

参数说明

- IF EXISTS

  如果指定的角色不存在，则发出一个 notice 而不是抛出一个错误。

- role_name

  要删除的角色名称。

  取值范围：已存在的角色。

示例

请参见 CREATE ROLE 的示例。

相关链接

11.16.44CREATE ROLE，11.16.10ALTER ROLE，11.16.107SET ROLE

11.16.73. DROP SCHEMA

功能描述

从数据库中删除模式。

注意事项

只有模式的所有者有权限执行 DROP SCHEMA 命令，系统管理员默认拥有此权限。

语法格式

```
DROP SCHEMA [ IF EXISTS ] schema_name [, ...] [ CASCADE | RESTRICT ];
```

参数说明

- IF EXISTS

  如果指定的模式不存在，发出一个 notice 而不是抛出一个错误。

- schema_name

  模式的名称。

  取值范围：已存在模式名。

- CASCADE | RESTRICT
  - CASCADE：自动删除包含在模式中的对象。
  - RESTRICT：如果模式包含任何对象，则删除失败（缺省行为）。

须知

不要随意删除 pg_temp 或 pg_toast_temp 开头的模式，这些模式是系统内部使用的，如果删除，可能导致无法预知的结果。

说明

无法删除当前模式。如果要删除当前模式，须切换到其他模式下。

示例

请参见 CREATE SCHEMA 的示例。

相关链接

11.16.12ALTER SCHEMA，11.16.45CREATE SCHEMA。

11.16.74. DROP SEQUENCE

功能描述

从当前数据库里删除序列。

注意事项

只有序列的所有者或者系统管理员才能删除。

语法格式

```
DROP SEQUENCE [ IF EXISTS ] {[schema.]sequence_name} [ , ... ] [ CASCADE | RESTRICT ];
```

参数说明

- IF EXISTS

  如果指定的序列不存在，则发出一个 notice 而不是抛出一个错误。

- name

  序列名称。

- CASCADE

  级联删除依赖序列的对象。

- RESTRICT

  如果存在任何依赖的对象，则拒绝删除序列。此项是缺省值。

示例

```
--创建一个名为 serial 的递增序列，从 101 开始。
vastbase=# CREATE SEQUENCE serial START 101;

--删除序列。
vastbase=# DROP SEQUENCE serial;
```

相关链接

11.16.13ALTER SEQUENCE， 11.16.74DROP SEQUENCE

11.16.75. DROP SYNONYM

功能描述

删除指定的 SYNONYM 对象。

注意事项

只有 SYNONYM 的所有者有权限执行 DROP SYNONYM 命令，系统管理员默认拥有此权限。

语法格式

```
DROP SYNONYM [ IF EXISTS ] synonym_name [ CASCADE | RESTRICT ];
```

参数描述

- IF EXISTS

  如果指定的同义词不存在，则发出一个 notice 而不是抛出一个错误。

- synonym_name

  同义词名字，可以带模式名。

- CASCADE | RESTRICT
  - CASCADE：级联删除依赖同义词的对象（比如视图）。
  - RESTRICT：如果有依赖对象存在，则拒绝删除同义词。此选项为缺省值。

示例

请参考 CREATE SYNONYM 的示例。

相关链接

11.16.15ALTER SYNONYM，11.16.47CREATE SYNONYM

11.16.76. DROP TABLE

功能描述

删除指定的表。

注意事项

DROP TABLE 会强制删除指定的表，删除表后，依赖该表的索引会被删除，而使用到该表的函数和存储过程将无法执行。删除分区表，会同时删除分区表中的所有分区。

语法格式

```
DROP TABLE [ IF EXISTS ]
   { [schema.]table_name } [, ...] [ CASCADE | RESTRICT ];
```

参数说明

- IF EXISTS

  如果指定的表不存在，则发出一个 notice 而不是抛出一个错误。

- schema

  模式名称。

- table_name

  表名称。

- CASCADE | RESTRICT
  - CASCADE：级联删除依赖于表的对象（比如视图）。
  - RESTRICT（缺省项）：如果存在依赖对象，则拒绝删除该表。这个是缺省。

示例

请参考 CREATE TABLE 的示例。

相关链接

11.16.17ALTER TABLE，11.16.48CREATE TABLE

11.16.77. DROP TABLESPACE

功能描述

删除一个表空间。

注意事项

- 只有表空间所有者有权限执行 DROP TABLESPACE 命令，系统管理员默认拥有此权限。
- 在删除一个表空间之前，表空间里面不能有任何数据库对象，否则会报错。
- DROP TABLESPACE 不支持回滚，因此，不能出现在事务块内部。
- 执行 DROP TABLESPACE 操作时，如果有另外的会话执行\db 查询操作，可能会由于 tablespace 事务的原因导致查询失败，请重新执行\db 查询操作。
- 如果执行 DROP TABLESPACE 失败，需要再次执行一次 DROP TABLESPACE IF EXISTS。

语法格式

```
DROP TABLESPACE [ IF EXISTS ] tablespace_name;
```

参数说明

- IF EXISTS

  如果指定的表空间不存在，则发出一个 notice 而不是抛出一个错误。

- tablespace_name

  表空间的名称。

  取值范围：已存在的表空间的名称。

示例

请参见 CREATE TABLESPACE 的示例。

相关链接

11.16.19ALTER TABLESPACE， 11.16.51CREATE TABLESPACE

优化建议

- drop database

  不支持在事务中删除 database。

11.16.78. DROP TEXT SEARCH CONFIGURATION

功能描述

删除已有文本搜索配置。

注意事项

要执行这个命令，用户必须是该配置的所有者。

语法格式

```
DROP TEXT SEARCH CONFIGURATION [ IF EXISTS ] name [ CASCADE | RESTRICT ];
```

参数说明

- IF EXISTS

  如果指定的文本搜索配置不存在，那么发出一个 notice 而不是抛出一个错误。

- name

  要删除的文本搜索配置名称（可有模式修饰）。

- CASCADE

  级联删除依赖文本搜索配置的对象。

- RESTRICT

  若有任何对象依赖文本搜索配置则拒绝删除它。这是默认情况。

示例

请参见 CREATE TEXT SEARCH CONFIGURATION 的示例。

相关链接

11.16.20ALTER TEXT SEARCH CONFIGURATION， 11.16.52CREATE TEXT SEARCH CONFIGURATION

11.16.79. DROP TEXT SEARCH DICTIONARY

功能描述

删除全文检索词典。

注意事项

- 预定义词典不支持 DROP 操作。
- 只有词典的所有者可以执行 DROP 操作，系统管理员默认拥有此权限。
- 谨慎执行 DROP...CASCADE 操作，该操作将级联删除使用该词典的文本搜索配置（TEXT SEARCH CONFIGURATION）。

语法格式

```
DROP TEXT SEARCH DICTIONARY [ IF EXISTS ] name [ CASCADE | RESTRICT ]
```

参数说明

- IF EXISTS

  如果指定的全文检索词典不存在，那么发出一个 Notice 而不是报错。

- name

  要删除的词典名称（可指定模式名，否则默认在当前模式下）。

  取值范围：已存在的词典名。

- CASCADE

  自动删除依赖于该词典的对象，并依次删除依赖于这些对象的所有对象。

如果存在任何一个使用该词典的文本搜索配置，此 DROP 命令将不会成功。可添加 CASCADE 以删除引用该词典的所有文本搜索配置以及词典。

- RESTRICT

  如果任何对象依赖词典，则拒绝删除该词典。这是缺省值。

示例

```
--删除词典 english
DROP TEXT SEARCH DICTIONARY english;
```

相关链接

11.16.21ALTER TEXT SEARCH DICTIONARY，11.16.53CREATE TEXT SEARCH DICTIONARY

11.16.80. DROP TRIGGER

功能描述

删除触发器。

注意事项

只有触发器的所有者可以执行 DROP TRIGGER 操作，系统管理员默认拥有此权限。

语法格式

```
DROP TRIGGER [ IF EXISTS ] trigger_name ON table_name [ CASCADE | RESTRICT ];
```

参数说明

- IF EXISTS

  如果指定的触发器不存在，则发出一个 notice 而不是抛出一个错误。

- trigger_name

  要删除的触发器名称。

  取值范围：已存在的触发器。

- table_name

  要删除的触发器所在的表名称。

  取值范围：已存在的含触发器的表。

- CASCADE | RESTRICT
  - CASCADE：级联删除依赖此触发器的对象。

- RESTRICT：如果有依赖对象存在，则拒绝删除此触发器。此选项为缺省值。

示例

请参见 11.16.54CREATE TRIGGER 的示例。

相关链接

11.16.54CREATE TRIGGER，11.16.22ALTER TRIGGER，11.16.17ALTER TABLE

11.16.81. DROP TYPE

功能描述

删除一个用户定义的数据类型。只有类型所有者才有删除权限。

语法格式

```
DROP TYPE [ IF EXISTS ] name [, ...] [ CASCADE | RESTRICT ]
```

参数说明

- IF EXISTS

  如果指定的类型不存在，那么发出一个 notice 而不是抛出一个错误。

- name

  要删除的类型名(可以有模式修饰)。

- CASCADE

  级联删除依赖该类型的对象(比如字段、函数、操作符等)

- RESTRICT

  如果有依赖对象，则拒绝删除该类型（缺省行为）。

示例

请参考 CREATE TYPE 的示例。

相关链接

11.16.55CREATE TYPE，11.16.23ALTER TYPE

11.16.82. DROP USER

功能描述

删除用户，同时会删除同名的 schema。

注意事项

- 须使用 CASCADE 级联删除依赖用户的对象（除数据库外）。当删除用户的级联对象时，如果级联对象处于锁定状态，则此级联对象无法被删除，直到对象被解锁或锁定级联对象的进程被杀死。
- 在数据库中删除用户时，如果依赖用户的对象在其他数据库中或者依赖用户的对象是其他数据库，请用户先手动删除其他数据库中的依赖对象或直接删除依赖数据库，再删除用户。即 drop user 不支持跨数据库进行级联删除。
- 如果该用户被 DATA SOURCE 对象依赖时，无法直接级联删除该用户，需要手动删除对应的 DATA SOURCE 对象之后再删除该用户。

语法格式

```
DROP USER [ IF EXISTS ] user_name [, ...] [ CASCADE | RESTRICT ];
```

参数说明

- IF EXISTS

  如果指定的用户不存在，发出一个 notice 而不是抛出一个错误。

- user_name

  待删除的用户名。

  取值范围：已存在的用户名。

- CASCADE | RESTRICT
  - CASCADE：级联删除依赖用户的对象。
  - RESTRICT：如果用户还有任何依赖的对象，则拒绝删除该用户（缺省行为）。

说明

在 Vastbase 中，存在一个配置参数 enable_kill_query，此参数在配置文件 postgresql.conf 中。此参数影响级联删除用户对象的行为：

- 当参数 enable_kill_query 为 on ，且使用 CASCADE 模式删除用户时，会自动 kill 锁定用户级联对象的进程，并删除用户。
- 当参数 enable_kill_query 为 off，且使用 CASCADE 模式删除用户时，会等待锁定级联对象的进程结束之后再删除用户。

示例

请参考 CREATE USER 的示例。

相关链接

11.16.24ALTER USER，11.16.56CREATE USER

11.16.83. DROP VIEW

功能描述

数据库中强制删除已有的视图。

注意事项

只有视图的所有者有权限执行 DROP VIEW 的命令，系统管理员默认拥有此权限。

语法格式

```
DROP VIEW [ IF EXISTS ] view_name [, ...] [ CASCADE | RESTRICT ];
```

参数说明

- IF EXISTS

  如果指定的视图不存在，则发出一个 notice 而不是抛出一个错误。

- view_name

  要删除的视图名称。

  取值范围：已存在的视图。

- CASCADE | RESTRICT
  - CASCADE：级联删除依赖此视图的对象（比如其他视图）。
  - RESTRICT：如果有依赖对象存在，则拒绝删除此视图。此选项为缺省值。

示例

请参见 CREATE VIEW 的示例。

相关链接

11.16.25ALTER VIEW，11.16.57CREATE VIEW

11.16.84. EXECUTE

功能描述

执行一个预先准备好的预备语句。一个预备语句只在会话的生命期里存在，因此预备语句必须是在当前会话的早期用 PREPARE 语句创建的。

注意事项

如果创建预备语句的 PREPARE 语句声明了一些参数，那么传递给 EXECUTE 语句的必须是一个兼容的参数集，否则就会生成一个错误。

语法格式

```
EXECUTE name [ ( parameter [, ...] ) ];
```

参数说明

- name

  要执行的预备语句的名称。

- parameter

  给预备语句的一个参数的具体数值。它必须是一个和生成与创建这个预备语句时指定参数的数据类型相兼容的值的表达式。

示例

```
--创建表 reason。
vastbase=# CREATE TABLE tpcds.reason (
   CD_DEMO_SK        INTEGER        NOT NULL,
   CD_GENDER         character(16)          ,
   CD_MARITAL_STATUS   character(100)
)
;

--插入数据。
vastbase=# INSERT INTO tpcds.reason VALUES(51, 'AAAAAAAADDAAAAAA', 'reason 51');

--创建表 reason_t1。
vastbase=# CREATE TABLE tpcds.reason_t1 AS TABLE tpcds.reason;

--为一个 INSERT 语句创建一个预备语句然后执行它。
vastbase=# PREPARE insert_reason(integer,character(16),character(100)) AS INSERT INTO
tpcds.reason_t1 VALUES($1,$2,$3);

vastbase=# EXECUTE insert_reason(52, 'AAAAAAAADDAAAAAA', 'reason 52');

--删除表 reason 和 reason_t1。
```

```
vastbase=# DROP TABLE tpcds.reason;
vastbase=# DROP TABLE tpcds.reason_t1;
```

11.16.85. EXPLAIN

功能描述

显示 SQL 语句的执行计划。

执行计划将显示 SQL 语句所引用的表会采用什么样的扫描方式，如：简单的顺序扫描、索引扫描等。如果引用了多个表，执行计划还会显示用到的 JOIN 算法。

执行计划的最关键的部分是语句的预计执行开销，这是计划生成器估算执行该语句将花费多长的时间。

若指定了 ANALYZE 选项，则该语句会被执行，然后根据实际的运行结果显示统计数据，包括每个计划节点内时间总开销（毫秒为单位）和实际返回的总行数。这对于判断计划生成器的估计是否接近现实非常有用。

注意事项

在指定 ANALYZE 选项时，语句会被执行。如果用户想使用 EXPLAIN 分析 INSERT，UPDATE，DELETE，CREATE TABLE AS 或 EXECUTE 语句，而不想改动数据（执行这些语句会影响数据），请使用这种方法：

```
START TRANSACTION;
EXPLAIN ANALYZE ...;
ROLLBACK;
```

语法格式

- 显示 SQL 语句的执行计划，支持多种选项，对选项顺序无要求。

```
EXPLAIN [ ( option [, ...] ) ] statement;
```

其中选项 option 子句的语法为。

```
ANALYZE [ boolean ] |
   ANALYSE [ boolean ] |
   VERBOSE [ boolean ] |
   COSTS [ boolean ] |
   CPU [ boolean ] |
   DETAIL [ boolean ] |
   NODES [ boolean ] |
   NUM_NODES [ boolean ] |
   BUFFERS [ boolean ] |
   TIMING [ boolean ] |
   PLAN [ boolean ] |
   FORMAT { TEXT | XML | JSON | YAML }
```

- 显示 SQL 语句的执行计划，且要按顺序给出选项。

```
EXPLAIN { [ { ANALYZE | ANALYSE } ] [ VERBOSE ] | PERFORMANCE } statement;
```

参数说明

- statement

  指定要分析的 SQL 语句。

- ANALYZE boolean | ANALYSE boolean

  显示实际运行时间和其他统计数据。

  取值范围：

  - TRUE（缺省值）：显示实际运行时间和其他统计数据。
  - FALSE：不显示。

- VERBOSE boolean

  显示有关计划的额外信息。

  取值范围：

  - TRUE（缺省值）：显示额外信息。
  - FALSE：不显示。

- COSTS boolean

  包括每个规划节点的估计总成本，以及估计的行数和每行的宽度。

  取值范围：

  - TRUE（缺省值）：显示估计总成本和宽度。
  - FALSE：不显示。

- CPU boolean

  打印 CPU 的使用情况的信息。

  取值范围：

  - TRUE（缺省值）：显示 CPU 的使用情况。
  - FALSE：不显示。

- DETAIL boolean

  打印数据库节点上的信息。

  取值范围：

  - TRUE（缺省值）：打印数据库节点的信息。
  - FALSE：不打印。

- NODES boolean

  打印 query 执行的节点信息。

  取值范围：

  - TRUE（缺省值）：打印执行的节点的信息。

- FALSE：不打印。

❖ NUM_NODES boolean

打印执行中的节点的个数信息。

取值范围：

- TRUE（缺省值）：打印数据库节点个数的信息。
- FALSE：不打印。

❖ BUFFERS boolean

包括缓冲区的使用情况的信息。

取值范围：

- TRUE：显示缓冲区的使用情况。
- FALSE（缺省值）：不显示。

❖ TIMING boolean

包括实际的启动时间和花费在输出节点上的时间信息。

取值范围：

- TRUE（缺省值）：显示启动时间和花费在输出节点上的时间信息。
- FALSE：不显示。

❖ PLAN

是否将执行计划存储在 plan_table 中。当该选项开启时，会将执行计划存储在 PLAN_TABLE 中，不打印到当前屏幕，因此该选项为 on 时，不能与其他选项同时使用。

取值范围：

- ON（缺省值）：将执行计划存储在 plan_table 中，不打印到当前屏幕。执行成功返回 EXPLAIN SUCCESS。
- OFF：不存储执行计划，将执行计划打印到当前屏幕。

❖ FORMAT

指定输出格式。

取值范围：TEXT，XML，JSON 和 YAML。

默认值：TEXT。

❖ PERFORMANCE

使用此选项时，即打印执行中的所有相关信息。

示例

```
--创建一个表 tpcds.customer_address_p1。
vastbase=# CREATE TABLE tpcds.customer_address_p1 AS TABLE tpcds.customer_address;
```

```
--修改 explain_perf_mode 为 normal
vastbase=# SET explain_perf_mode=normal;

--显示表简单查询的执行计划。
vastbase=# EXPLAIN SELECT * FROM tpcds.customer_address_p1;
QUERY PLAN
------------------------------------------------
Data Node Scan  (cost=0.00..0.00 rows=0 width=0)
Node/s: All dbnodes
(2 rows)

--以 JSON 格式输出的执行计划（explain_perf_mode 为 normal 时）。
vastbase=# EXPLAIN(FORMAT JSON) SELECT * FROM tpcds.customer_address_p1;
           QUERY PLAN
-------------------------------------
 [                            +
   {                          +
     "Plan": {                  +
       "Node Type": "Data Node Scan",+
       "Startup Cost": 0.00,       +
       "Total Cost": 0.00,         +
       "Plan Rows": 0,             +
       "Plan Width": 0,            +
       "Node/s": "All dbnodes"     +
     }                          +
   }                          +
 ]
(1 row)

--如果有一个索引，当使用一个带索引 WHERE 条件的查询，可能会显示一个不同的计划。
vastbase=# EXPLAIN SELECT * FROM tpcds.customer_address_p1 WHERE ca_address_sk=10000;
QUERY PLAN
------------------------------------------------
Data Node Scan  (cost=0.00..0.00 rows=0 width=0)
Node/s: dn_6005_6006
(2 rows)

--以 YAML 格式输出的执行计划（explain_perf_mode 为 normal 时）。
vastbase=# EXPLAIN(FORMAT YAML) SELECT * FROM tpcds.customer_address_p1 WHERE ca_address_sk=10000;
         QUERY PLAN
---------------------------------
 - Plan:                   +
     Node Type: "Data Node Scan"+
     Startup Cost: 0.00        +
     Total Cost: 0.00          +
     Plan Rows: 0             +
     Plan Width: 0            +
     Node/s: "dn_6005_6006"
(1 row)

--禁止开销估计的执行计划。
vastbase=# EXPLAIN(COSTS FALSE)SELECT * FROM tpcds.customer_address_p1 WHERE ca_address_sk=10000;
      QUERY PLAN
------------------------
 Data Node Scan
```

```
  Node/s: dn_6005_6006
(2 rows)

--带有聚集函数查询的执行计划。
vastbase=# EXPLAIN SELECT SUM(ca_address_sk) FROM tpcds.customer_address_p1 WHERE ca_address_sk<10000;
                              QUERY PLAN
-----------------------------------------------------------------------------------
 Aggregate  (cost=18.19..14.32 rows=1 width=4)
   ->  Streaming (type: GATHER)  (cost=18.19..14.32 rows=3 width=4)
         Node/s: All dbnodes
         ->  Aggregate  (cost=14.19..14.20 rows=3 width=4)
               ->  Seq Scan on customer_address_p1  (cost=0.00..14.18 rows=10 width=4)
                     Filter: (ca_address_sk < 10000)
(6 rows)

--删除表tpcds.customer_address_p1。
vastbase=# DROP TABLE tpcds.customer_address_p1;
```

相关链接

11.16.26ANALYZE | ANALYSE

11.16.86. EXPLAIN PLAN

功能描述

通过 EXPLAIN PLAN 命令可以将查询执行的计划信息存储于 PLAN_TABLE 表中。与 EXPLAIN 命令不同的是，EXPLAIN PLAN 仅将计划信息进行存储，而不会打印到屏幕。

语法格式

```
EXPLAIN PLAN
[ SET STATEMENT_ID = string ]
FOR statement ;
```

参数说明

- EXPLAIN 中的 PLAN 选项表示需要将计划信息存储于 PLAN_TABLE 中，存储成功将返回“EXPLAIN SUCCESS”。
- STATEMENT_ID 用户可以对查询设置标签，输入的标签信息也将存储于 PLAN_TABLE 中。

说明

用户在执行 EXPLAIN PLAN 时，如果没有进行 SET　STATEMENT_ID，则默认为空值。同时，用户可输入的 STATEMENT_ID 最大长度为 30 个字节，超过长度将会产生报错。

注意事项

- EXPLAIN PLAN 不支持在数据库节点上执行。

- 对于执行错误的 SQL 无法进行计划信息的收集。
- PLAN_TABLE 中的数据是 session 级生命周期并且 session 隔离和用户隔离，用户只能看到当前 session、当前用户的数据。

示例 1

使用 EXPLAIN　PLAN 收集 SQL 语句的执行计划，通常包括以下步骤：

步骤 1　执行 EXPLAN PLAN。

说明

执行 EXPLAIN PLAN 后会将计划信息自动存储于 PLAN_TABLE 中，不支持对 PLAN_TABLE 进行 INSERT、UPDATE、ANALYZE 等操作。

PLAN_TABLE 详细介绍见 14.3.75PLAN_TABLE。

```
explain plan set statement_id='TPCH-Q4' for
select
o_orderpriority,
count(*) as order_count
from
orders
where
o_orderdate >= '1993-07-01'::date
and o_orderdate < '1993-07-01'::date + interval '3 month'
and exists (
select
*
from
lineitem
where
l_orderkey = o_orderkey
and l_commitdate < l_receiptdate
)
group by
o_orderpriority
order by
o_orderpriority;
```

步骤 2　查询 PLAN_TABLE。

```
SELECT * FROM PLAN_TABLE;
```

```
 statement_id | plan_id  | id |      operation       |   options   | object_name | object_type | object_owner |                projection
--------------+----------+----+----------------------+-------------+-------------+-------------+--------------+-------------------------------------------
 TPCH-Q4      | 16781167 |  1 | ROW ADAPTER          |             |             |             | ORDERS.O_ORDERPRIORITY, (PG_CATALOG.COUNT(*))
 TPCH-Q4      | 16781167 |  2 | VECTOR SORT          |             |             |             | ORDERS.O_ORDERPRIORITY, (PG_CATALOG.COUNT(*))
 TPCH-Q4      | 16781167 |  3 | VECTOR AGGREGATE     | HASHED      |             |             | ORDERS.O_ORDERPRIORITY, PG_CATALOG.COUNT(*)
 TPCH-Q4      | 16781167 |  4 | VECTOR STREAMING     | GATHER      |             |             | ORDERS.O_ORDERPRIORITY, (COUNT(*))
 TPCH-Q4      | 16781167 |  5 | VECTOR AGGREGATE     | HASHED      |             |             | ORDERS.O_ORDERPRIORITY, COUNT(*)
 TPCH-Q4      | 16781167 |  6 | VECTOR NESTED LOOPS  | SEMI        |             |             | ORDERS.O_ORDERPRIORITY
 TPCH-Q4      | 16781167 |  7 | TABLE ACCESS         | CSTORE SCAN | ORDERS      | TABLE       | TPCH         | ORDERS.O_ORDERPRIORITY, ORDERS.O_ORDERKEY
 TPCH-Q4      | 16781167 |  8 | VECTOR MATERIALIZE   |             |             |             |              | LINEITEM.L_ORDERKEY
 TPCH-Q4      | 16781167 |  9 | TABLE ACCESS         | CSTORE SCAN | LINEITEM    | TABLE       | TPCH         | LINEITEM.L_ORDERKEY
```

步骤 3　清理 PLAN_TABLE 表中的数据。

```
DELETE FROM PLAN_TABLE WHERE xxx;
```

11.16.87. FETCH

功能描述

FETCH 通过已创建的游标来检索数据。

每个游标都有一个供 FETCH 使用的关联位置。游标的关联位置可以在查询结果的第一行之前，或者在结果中的任意行，或者在结果的最后一行之后：

- 游标刚创建完之后，关联位置在第一行之前的。
- 在抓取了一些移动行之后，关联位置在检索到的最后一行上。
- 如果 FETCH 抓取完了所有可用行，它就停在最后一行后面，或者在反向抓取的情况下是停在第一行前面。
- FETCH ALL 或 FETCH BACKWARD ALL 将总是把游标的关联位置放在最后一行或者在第一行前面。

注意事项

- 如果游标定义了 NO SCROLL，则不允许使用例如 FETCH BACKWARD 之类的反向抓取。
- NEXT，PRIOR，FIRST，LAST，ABSOLUTE，RELATIVE 形式在恰当地移动游标之后抓取一条记录。如果后面没有数据行，就返回一个空的结果，此时游标就会停在查询结果的最后一行之后（向后查询时）或者第一行之前（向前查询时）。
- FORWARD 和 BACKWARD 形式在向前或者向后移动的过程中抓取指定的行数，然后把游标定位在最后返回的行上；当 count 大于可用的行数，将游标定位到所有行之后（向后查询时）或者之前（向前查询时）。
- RELATIVE 0，FORWARD 0，BACKWARD 0 都要求在不移动游标的前提下抓取当前行，也就是重新抓取最近刚抓取过的行。除非游标定位在第一行之前或者最后一行之后，这个动作都应该成功，而在那两种情况下，不返回任何行。
- 当 FETCH 的游标上涉及列存表时，不支持 BACKWARD、PRIOR、FIRST 等涉及反向获取操作。

语法格式

```
FETCH [ direction { FROM | IN } ] cursor_name;
```

其中 direction 子句为可选参数。

```
NEXT
  | PRIOR
  | FIRST
  | LAST
  | ABSOLUTE count
  | RELATIVE count
  | count
  | ALL
  | FORWARD
  | FORWARD count
```

```
| FORWARD ALL
| BACKWARD
| BACKWARD count
| BACKWARD ALL
```

参数说明

- direction_clause

  定义抓取数据的方向。

  取值范围：

  - NEXT（缺省值）

    从当前关联位置开始，抓取下一行。

  - PRIOR

    从当前关联位置开始，抓取上一行。

  - FIRST

    抓取查询的第一行（和 ABSOLUTE 1 相同）。

  - LAST

    抓取查询的最后一行（和 ABSOLUTE -1 相同）。

  - ABSOLUTE count

    抓取查询中第 count 行。

    ABSOLUTE 抓取不会比用相对位移移动到需要的数据行更快，因为下层的实现必须遍历所有中间的行。

    count 取值范围：有符号的整数

    - count 为正数，就从查询结果的第一行开始，抓取第 count 行。当 count 小于当前游标位置时，涉及到 rewind 操作，暂不支持。
    - count 为负数或 0，涉及到反向扫描操作，暂不支持。

  - RELATIVE count

    从当前关联位置开始，抓取随后或前面的第 count 行。

    取值范围：有符号的整数

    - count 为正数就抓取当前关联位置之后的第 count 行。
    - count 为负数或 0，涉及到反向扫描操作，暂不支持。
    - 如果当前行没有数据的话，RELATIVE 0 返回空。

  - count

    抓取随后的 count 行（和 FORWARD count 一样）。

  - ALL

从当前关联位置开始，抓取所有剩余的行（和 FORWARD ALL 一样）。

- FORWARD

  抓取下一行（和 NEXT 一样）。

- FORWARD count

  与 RELATIVE count 的效果相同，从当前关联位置开始，抓取随后或前面的第 count 行。

- FORWARD ALL

  从当前关联位置开始，抓取所有剩余行。

- BACKWARD

  从当前关联位置开始，抓取前面一行(和 PRIOR 一样）。

- BACKWARD count

  从当前关联位置开始，抓取前面的 count 行（向后扫描）。

  取值范围：有符号的整数

  - count 为正数就抓取当前关联位置之前的第 count 行。
  - count 为负数就抓取当前关联位置之后的第 abs（count）行。
  - 如果有数据的话，BACKWARD 0 重新抓取当前行。

- BACKWARD ALL

  从当前关联位置开始，抓取所有前面的行（向后扫描） 。

❖ { FROM | IN } cursor_name

使用关键字 FROM 或 IN 指定游标名称。

取值范围：已创建的游标的名称。

示例

```
--SELECT 语句，用一个游标读取一个表。开始一个事务。
vastbase=# START TRANSACTION;

--建立一个名为 cursor1 的游标。
vastbase=# CURSOR cursor1 FOR SELECT * FROM tpcds.customer_address ORDER BY 1;

--抓取头 3 行到游标 cursor1 里。
vastbase=# FETCH FORWARD 3 FROM cursor1;
 ca_address_sk |  ca_address_id   | ca_street_number |   ca_street_name   | ca_street_type |
ca_suite_number |     ca_city     |    ca_county    | ca_state |  ca_zip   |  ca_country   |
ca_gmt_offset |   ca_location_type
---------------+------------------+------------------+--------------------+-----------------+----
------------+-----------------+-----------------+----------+------------+---------------+-------
--------+---------------------
           1 | AAAAAAAABAAAAAAA | 18               | Jackson          | Parkway        | Suite 280      |
Fairfield      | Maricopa County | AZ       | 86192     | United States |        -7.00 | condo
```

```
          2 | AAAAAAAACAAAAAAA | 362            | Washington 6th     | RD            | Suite 80       |
Fairview       | Taos County     | NM      | 85709      | United States |        -7.00 | condo
          3 | AAAAAAAADAAAAAAA | 585            | Dogwood Washington | Circle        | Suite Q        |
Pleasant Valley | York County     | PA      | 12477      | United States |        -5.00 | single family
(3 rows)

--关闭游标并提交事务。
vastbase=# CLOSE cursor1;

--结束一个事务。
vastbase=# END;

--VALUES 子句，用一个游标读取 VALUES 子句中的内容。开始一个事务。
vastbase=# START TRANSACTION;

--建立一个名为 cursor2 的游标。
vastbase=# CURSOR cursor2 FOR VALUES(1,2),(0,3) ORDER BY 1;

--抓取头 2 行到游标 cursor2 里。
vastbase=# FETCH FORWARD 2 FROM cursor2;
column1 | column2
---------+---------
0 |       3
1 |       2
(2 rows)

--关闭游标并提交事务。
vastbase=# CLOSE cursor2;

--结束一个事务。
vastbase=# END;

--WITH HOLD 游标的使用，开启事务。
vastbase=# START TRANSACTION;

--创建一个 with hold 游标。
vastbase=# DECLARE cursor1 CURSOR WITH HOLD FOR SELECT * FROM tpcds.customer_address ORDER BY 1;

--抓取头 2 行到游标 cursor1 里。
vastbase=# FETCH FORWARD 2 FROM cursor1;
 ca_address_sk |  ca_address_id   | ca_street_number |   ca_street_name   | ca_street_type  |
ca_suite_number |    ca_city     |    ca_county    | ca_state |   ca_zip   |  ca_country   |
ca_gmt_offset |   ca_location_type
---------------+------------------+------------------+--------------------+-----------------+----
-------------+-----------------+-----------------+----------+------------+---------------+-------
--------+----------------------
          1 | AAAAAAAABAAAAAAA | 18               | Jackson            | Parkway         | Suite 280      |
Fairfield      | Maricopa County | AZ      | 86192      | United States |        -7.00 | condo
          2 | AAAAAAAACAAAAAAA | 362              | Washington 6th     | RD              | Suite 80       |
Fairview       | Taos County     | NM      | 85709      | United States |        -7.00 | condo
(2 rows)

--结束事务。
vastbase=# END;
```

```
--抓取下一行到游标 cursor1 里。
vastbase=# FETCH FORWARD 1 FROM cursor1;
 ca_address_sk |  ca_address_id   | ca_street_number |   ca_street_name   | ca_street_type  |
ca_suite_number |     ca_city     |    ca_county    | ca_state |   ca_zip   |  ca_country   |
ca_gmt_offset |   ca_location_type
---------------+------------------+------------------+--------------------+-----------------+----
-------------+-----------------+-----------------+----------+------------+---------------+-------
--------+----------------------
           3 | AAAAAAAADAAAAAAA | 585              | Dogwood Washington | Circle          | Suite Q       |
Pleasant Valley | York County     | PA       | 12477      | United States |        -5.00 | single family
(1 row)

--关闭游标。
vastbase=# CLOSE cursor1;
```

相关链接

11.16.30CLOSE，11.16.91MOVE

11.16.88. GRANT

功能描述

对角色和用户进行授权操作。

使用 GRANT 命令进行用户授权包括以下三种场景：

- 将系统权限授权给角色或用户

  系统权限又称为用户属性，包括 SYSADMIN、CREATEDB、CREATEROLE、AUDITADMIN 和 LOGIN。

  系统权限一般通过 CREATE/ALTER ROLE 语法来指定。其中，SYSADMIN 权限可以通过 GRANT/REVOKE ALL PRIVILEGE 授予或撤销。但系统权限无法通过 ROLE 和 USER 的权限被继承，也无法授予 PUBLIC。

- 将数据库对象授权给角色或用户

  将数据库对象（表和视图、指定字段、数据库、函数、模式、表空间等）的相关权限授予特定角色或用户；

  GRANT 命令将数据库对象的特定权限授予一个或多个角色。这些权限会追加到已有的权限上。

  关键字 PUBLIC 表示该权限要赋予所有角色，包括以后创建的用户。PUBLIC 可以看做是一个隐含定义好的组，它总是包括所有角色。任何角色或用户都将拥有通过 GRANT 直接赋予的权限和所属的权限，再加上 PUBLIC 的权限。

  如果声明了 WITH GRANT OPTION，则被授权的用户也可以将此权限赋予他人，否则就不能授权给他人。这个选项不能赋予 PUBLIC，这是 Vastbase 特有的属性。

Vastbase 会将某些类型的对象上的权限授予 PUBLIC。默认情况下，对表、表字段、序列、外部数据源、外部服务器、模式或表空间对象的权限不会授予 PUBLIC，而以下这些对象的权限会授予 PUBLIC：数据库的 CONNECT 权限和 CREATE TEMP TABLE 权限、函数的 EXECUTE 特权、语言和数据类型（包括域）的 USAGE 特权。当然，对象拥有者可以撤销默认授予 PUBLIC 的权限并专门授予权限给其他用户。为了更安全，建议在同一个事务中创建对象并设置权限，这样其他用户就没有时间窗口使用该对象。另外，这些初始的默认权限可以使用 ALTER DEFAULT PRIVILEGES 命令修改。

- 将角色或用户的权限授权给其他角色或用户

  将一个角色或用户的权限授予一个或多个其他角色或用户。在这种情况下，每个角色或用户都可视为拥有一个或多个数据库权限的集合。

  当声明了 WITH ADMIN OPTION，被授权的用户可以将该权限再次授予其他角色或用户，以及撤销所有由该角色或用户继承到的权限。当授权的角色或用户发生变更或被撤销时，所有继承该角色或用户权限的用户拥有的权限都会随之发生变更。

  数据库系统管理员可以给任何角色或用户授予/撤销任何权限。拥有 CREATEROLE 权限的角色可以赋予或者撤销任何非系统管理员角色的权限。

注意事项

无。

语法格式

- 将表或视图的访问权限赋予指定的用户或角色。

```
GRANT { { SELECT | INSERT | UPDATE | DELETE | TRUNCATE | REFERENCES } [, ...]
    | ALL [ PRIVILEGES ] }
   ON { [ TABLE ] table_name [, ...]
     | ALL TABLES IN SCHEMA schema_name [, ...] }
   TO { [ GROUP ] role_name | PUBLIC } [, ...]
   [ WITH GRANT OPTION ];
```

- 将表中字段的访问权限赋予指定的用户或角色。

```
GRANT { {{ SELECT | INSERT | UPDATE | REFERENCES } ( column_name [, ...] )} [, ...]
    | ALL [ PRIVILEGES ] ( column_name [, ...] ) }
   ON [ TABLE ] table_name [, ...]
   TO { [ GROUP ] role_name | PUBLIC } [, ...]
   [ WITH GRANT OPTION ];
```

- 将数据库的访问权限赋予指定的用户或角色。

```
GRANT { { CREATE | CONNECT | TEMPORARY | TEMP } [, ...]
    | ALL [ PRIVILEGES ] }
   ON DATABASE database_name [, ...]
   TO { [ GROUP ] role_name | PUBLIC } [, ...]
   [ WITH GRANT OPTION ];
```

- 将域的访问权限赋予指定的用户或角色。

```
GRANT { USAGE | ALL [ PRIVILEGES ] }
    ON DOMAIN domain_name [, ...]
    TO { [ GROUP ] role_name | PUBLIC } [, ...]
    [ WITH GRANT OPTION ];
```

说明

本版本暂时不支持赋予域的访问权限。

- 将外部数据源的访问权限赋予给指定的用户或角色。

```
GRANT { USAGE | ALL [ PRIVILEGES ] }
    ON FOREIGN DATA WRAPPER fdw_name [, ...]
    TO { [ GROUP ] role_name | PUBLIC } [, ...]
    [ WITH GRANT OPTION ];
```

- 将外部服务器的访问权限赋予给指定的用户或角色。

```
GRANT { USAGE | ALL [ PRIVILEGES ] }
    ON FOREIGN SERVER server_name [, ...]
    TO { [ GROUP ] role_name | PUBLIC } [, ...]
    [ WITH GRANT OPTION ];
```

- 将函数的访问权限赋予给指定的用户或角色。

```
GRANT { EXECUTE | ALL [ PRIVILEGES ] }
    ON { FUNCTION {function_name ( [ {[ argmode ] [ arg_name ] arg_type} [, ...] ] )} [, ...]
       | ALL FUNCTIONS IN SCHEMA schema_name [, ...] }
    TO { [ GROUP ] role_name | PUBLIC } [, ...]
    [ WITH GRANT OPTION ];
```

- 将过程语言的访问权限赋予给指定的用户或角色。

```
GRANT { USAGE | ALL [ PRIVILEGES ] }
    ON LANGUAGE lang_name [, ...]
    TO { [ GROUP ] role_name | PUBLIC } [, ...]
    [ WITH GRANT OPTION ];
```

- 将大对象的访问权限赋予指定的用户或角色。

```
GRANT { { SELECT | UPDATE } [, ...] | ALL [ PRIVILEGES ] }
    ON LARGE OBJECT loid [, ...]
    TO { [ GROUP ] role_name | PUBLIC } [, ...]
    [ WITH GRANT OPTION ];
```

说明

本版本暂时不支持大对象。

- 将模式的访问权限赋予指定的用户或角色。

```
GRANT { { CREATE | USAGE } [, ...] | ALL [ PRIVILEGES ] }
    ON SCHEMA schema_name [, ...]
    TO { [ GROUP ] role_name | PUBLIC } [, ...]
    [ WITH GRANT OPTION ];
```

说明

将模式中的表或者视图对象授权给其他用户时，需要将表或视图所属的模式的 USAGE 权限同时授予该用户，若没有该权限，则只能看到这些对象的名称，并不能实际进行对象访问。

- 将表空间的访问权限赋予指定的用户或角色。

```
GRANT { CREATE | ALL [ PRIVILEGES ] }
    ON TABLESPACE tablespace_name [, ...]
    TO { [ GROUP ] role_name | PUBLIC } [, ...]
    [ WITH GRANT OPTION ];
```

- 将类型的访问权限赋予指定的用户或角色。

```
GRANT { USAGE | ALL [ PRIVILEGES ] }
   ON TYPE type_name [, ...]
   TO { [ GROUP ] role_name | PUBLIC } [, ...]
   [ WITH GRANT OPTION ];
```

说明

本版本暂时不支持赋予类型的访问权限。

- 将角色的权限赋予其他用户或角色的语法。

```
GRANT role name [, ...]
  TO role name [, ...]
  [ WITH ADMIN OPTION ];
```

- 将 sysadmin 权限赋予指定的角色。

```
GRANT ALL { PRIVILEGES | PRIVILEGE }
  TO role_name;
```

- 将 Data Source 对象的权限赋予指定的角色。

```
GRANT {USAGE | ALL [PRIVILEGES]}
  ON DATA SOURCE src_name [, ...]
  TO {[GROUP] role_name | PUBLIC} [, ...] [WITH GRANT OPTION];
```

- 将 directory 对象的权限赋予指定的角色。

```
GRANT {READ|WRITE| ALL [PRIVILEGES]}
  ON DIRECTORY directory_name [, ...]
  TO {[GROUP] role_name | PUBLIC} [, ...] [WITH GRANT OPTION];
```

参数说明

GRANT 的权限分类如下所示。

- SELECT

  允许对指定的表、视图、序列执行 SELECT 命令，UPDATE 或 DELETE 时也需要对应字段上的 SELECT 权限。

- INSERT

  允许对指定的表执行 INSERT 命令。

- UPDATE

  允许对声明的表中任意字段执行 UPDATE 命令。通常，UPDATE 命令也需要 SELECT 权限来查询出哪些行需要更新。SELECT... FOR UPDATE 和 SELECT... FOR SHARE 除了需要 SELECT 权限外，还需要 UPDATE 权限。

- DELETE

  允许执行 DELETE 命令删除指定表中的数据。通常，DELETE 命令也需要 SELECT 权限来查询出哪些行需要删除。

- TRUNCATE

  允许执行 TRUNCATE 语句删除指定表中的所有记录。

- REFERENCES

  创建一个外键约束，必须拥有参考表和被参考表的 REFERENCES 权限。由于当前不支持外键，所以不建议使用该参数。

- CREATE
  - 对于数据库，允许在数据库里创建新的模式。
  - 对于模式，允许在模式中创建新的对象。如果要重命名一个对象，用户除了必须是该对象的所有者外，还必须拥有该对象所在模式的 CREATE 权限。
  - 对于表空间，允许在表空间中创建表，允许在创建数据库和模式的时候把该表空间指定为缺省表空间。
- CONNECT

  允许用户连接到指定的数据库。

- EXECUTE

  允许使用指定的函数，以及利用这些函数实现的操作符。

- USAGE
  - 对于过程语言，允许用户在创建函数的时候指定过程语言。
  - 对于模式，USAGE 允许访问包含在指定模式中的对象，若没有该权限，则只能看到这些对象的名称。
  - 对于序列，USAGE 允许使用 nextval 函数。
  - 对于 Data Source 对象，USAGE 是指访问权限，也是可赋予的所有权限，即 USAGE 与 ALL PRIVILEGES 等价。
- ALL PRIVILEGES

  一次性给指定用户/角色赋予所有可赋予的权限。只有系统管理员有权执行 GRANT ALL PRIVILEGES。

GRANT 的参数说明如下所示。

- role_name

  已存在用户名称。

- table_name

  已存在表名称。

- column_name

  已存在字段名称。

- schema_name

  已存在模式名称。

- database_name

已存在数据库名称。

- funcation_name

  已存在函数名称。

- sequence_name

  已存在序列名称。

- domain_name

  已存在域类型名称。

- fdw_name

  已存在外部数据包名称。

- lang_name

  已存在语言名称。

- type_name

  已存在类型名称。

- src_name

  已存在的 Data Source 对象名称。

- argmode

  参数模式。

  取值范围：字符串，要符合标识符命名规范。

- arg_name

  参数名称。

  取值范围：字符串，要符合标识符命名规范。

- arg_type

  参数类型。

  取值范围：字符串，要符合标识符命名规范。

- loid

  包含本页的大对象的标识符。

  取值范围：字符串，要符合标识符命名规范。

- tablespace_name

  表空间名称。

- directory_name

目录名称。

取值范围：字符串，要符合标识符命名规范。

- WITH GRANT OPTION

  如果声明了 WITH GRANT OPTION，则被授权的用户也可以将此权限赋予他人，否则就不能授权给他人。这个选项不能赋予 PUBLIC。

非对象所有者给其他用户授予对象权限时，命令按照以下规则执行：

- 如果用户没有该对象上指定的权限，命令立即失败。
- 如果用户有该对象上的部分权限，则 GRANT 命令只授予他有授权选项的权限。
- 如果用户没有可用的授权选项，GRANT ALL PRIVILEGES 形式将发出一个警告信息，其他命令形式将发出在命令中提到的且没有授权选项的相关警告信息。

说明

数据库系统管理员可以访问所有对象，而不会受对象的权限设置影响。这个特点类似 Unix 系统的 root 的权限。和 root 一样，除了必要的情况外，建议不要总是以系统管理员身份进行操作。

不允许对表分区进行 GRANT 操作，对分区表进行 GRANT 操作会引起告警。

示例

示例：将系统权限授权给用户或者角色。

创建名为 joe 的用户，并将 sysadmin 权限授权给他。

```
vastbase=# CREATE USER joe PASSWORD 'Bigdata@123';
vastbase=# GRANT ALL PRIVILEGES TO joe;
```

授权成功后，用户 joe 会拥有 sysadmin 的所有权限。

示例：将对象权限授权给用户或者角色。

1. 撤销 joe 用户的 sysadmin 权限，然后将模式 tpcds 的使用权限和表 tpcds.reason 的所有权限授权给用户 joe。

```
vastbase=# REVOKE ALL PRIVILEGES FROM joe;
vastbase=# GRANT USAGE ON SCHEMA tpcds TO joe;
vastbase=# GRANT ALL PRIVILEGES ON tpcds.reason TO joe;
```

授权成功后，joe 用户就拥有了 tpcds.reason 表的所有权限，包括增删改查等权限。

2. 将 tpcds.reason 表中 r_reason_sk、r_reason_id、r_reason_desc 列的查询权限，r_reason_desc 的更新权限授权给 joe。

```
vastbase=# GRANT select (r_reason_sk,r_reason_id,r_reason_desc),update (r_reason_desc) ON
tpcds.reason TO joe;
```

授权成功后，用户 joe 对 tpcds.reason 表中 r_reason_sk，r_reason_id 的查询权限会立即生效。如果 joe 用户需要拥有将这些权限授权给其他用户的权限，可以通过以下语法对 joe 用户进行授权。

```
vastbase=# GRANT select (r_reason_sk, r_reason_id) ON tpcds.reason TO joe WITH GRANT OPTION;
```

将数据库 vastbase 的连接权限授权给用户 joe，并给予其在 vastbase 中创建 schema 的权限，而且允许 joe 将此权限授权给其他用户。

```
vastbase=# GRANT create,connect on database vastbase TO joe WITH GRANT OPTION;
```

创建角色 tpcds_manager，将模式 tpcds 的访问权限授权给角色 tpcds_manager，并授予该角色在 tpcds 下创建对象的权限，不允许该角色中的用户将权限授权给其他人。

```
vastbase=# CREATE ROLE tpcds_manager PASSWORD 'Bigdata@123';
vastbase=# GRANT USAGE,CREATE ON SCHEMA tpcds TO tpcds_manager;
```

将表空间 tpcds_tbspc 的所有权限授权给用户 joe，但用户 joe 无法将权限继续授予其他用户。

```
vastbase=# CREATE TABLESPACE tpcds_tbspc RELATIVE LOCATION 'tablespace/tablespace_1';
vastbase=# GRANT ALL ON TABLESPACE tpcds_tbspc TO joe;
```

示例：将用户或者角色的权限授权给其他用户或角色。

1. 创建角色 manager，将 joe 的权限授权给 manager，并允许该角色将权限授权给其他人。

```
vastbase=# CREATE ROLE manager PASSWORD 'Bigdata@123';
vastbase=# GRANT joe TO manager WITH ADMIN OPTION;
```

2. 创建用户 senior_manager，将用户 manager 的权限授权给该用户。

```
vastbase=# CREATE ROLE senior_manager PASSWORD 'Bigdata@123';
vastbase=# GRANT manager TO senior_manager;
```

3. 撤销权限，并清理用户。

```
vastbase=# REVOKE manager FROM joe;
vastbase=# REVOKE senior_manager FROM manager;
vastbase=# DROP USER manager;
```

示例：撤销上述授予的权限，并清理角色和用户。

```
vastbase=# REVOKE ALL PRIVILEGES ON tpcds.reason FROM joe;
vastbase=# REVOKE ALL PRIVILEGES ON SCHEMA tpcds FROM joe;
vastbase=# REVOKE ALL ON TABLESPACE tpcds_tbspc FROM joe;
vastbase=# DROP TABLESPACE tpcds_tbspc;
vastbase=# REVOKE USAGE,CREATE ON SCHEMA tpcds FROM tpcds_manager;
vastbase=# DROP ROLE tpcds_manager;
vastbase=# DROP ROLE senior_manager;
vastbase=# DROP USER joe CASCADE;
```

相关链接

11.16.99REVOKE，11.16.4ALTER DEFAULT PRIVILEGES

11.16.89. INSERT

功能描述

向表中添加一行或多行数据。

注意事项

- ❖ 只有拥有表 INSERT 权限的用户，才可以向表中插入数据。
- ❖ 如果使用 RETURNING 子句，用户必须要有该表的 SELECT 权限。
- ❖ 如果使用 ON DUPLICATE KEY UPDATE，用户必须要有该表的 SELECT、UPDATE 权限，唯一约束（主键或唯一索引）的 SELECT 权限。
- ❖ 如果使用 query 子句插入来自查询里的数据行，用户还需要拥有在查询里使用的表的 SELECT 权限。
- ❖ 当连接到 TD 兼容的数据库时，td_compatible_truncation 参数设置为 on 时，将启用超长字符串自动截断功能，在后续的 insert 语句中（不包含外表的场景下），对目标表中 char 和 varchar 类型的列上插入超长字符串时，系统会自动按照目标表中相应列定义的最大长度对超长字符串进行截断。

说明

如果向字符集为字节类型编码（SQL_ASCII，LATIN1 等）的数据库中插入多字节字符数据（如汉字等），且字符数据跨越截断位置，这种情况下，按照字节长度自动截断，自动截断后会在尾部产生非预期结果。如果用户有对于截断结果正确性的要求，建议用户采用 UTF8 等能够按照字符截断的输入字符集作为数据库的编码集。

语法格式

```
[ WITH [ RECURSIVE ] with_query [, ...] ]
INSERT INTO table_name [ ( column_name [, ...] ) ]
   { DEFAULT VALUES
   | VALUES {( { expression | DEFAULT } [, ...] ) }[, ...]
   | query }
   [ ON DUPLICATE KEY UPDATE { column_name = { expression | DEFAULT } } [, ...] ]
   [ RETURNING {* | {output_expression [ [ AS ] output_name ] }[, ...]} ];
```

参数说明

- ❖ WITH [RECURSIVE] with_query [, ...]

  用于声明一个或多个可以在主查询中通过名称引用的子查询，相当于临时表。

  如果声明了 RECURSIVE，那么允许 SELECT 子查询通过名称引用它自己。

  其中 with_query 的详细格式为：with_query_name [(column_name [, ...])] AS ({select | values | insert | update | delete})

  – with_query_name 指定子查询生成的结果集名称，在查询中可使用该名称访问子查询的结果集。

  – column_name 指定子查询结果集中显示的列名。

  – 每个子查询可以是 SELECT，VALUES，INSERT，UPDATE 或 DELETE 语句。

说明

INSERT ON DUPLICATE KEY UPDATE 不支持 WITH 及 WITH RECURSIVE 子句。

- table_name

  要插入数据的目标表名。

  取值范围：已存在的表名。

- column_name

  目标表中的字段名：

  - 字段名可以有子字段名或者数组下标修饰。
  - 没有在字段列表中出现的每个字段，将由系统默认值，或者声明时的默认值填充，若都没有则用 NULL 填充。例如，向一个复合类型中的某些字段插入数据的话，其他字段将是 NULL。
  - 目标字段（column_name）可以按顺序排列。如果没有列出任何字段，则默认全部字段，且顺序为表声明时的顺序。
  - 如果 value 子句和 query 中只提供了 N 个字段，则目标字段为前 N 个字段。
  - value 子句和 query 提供的值在表中从左到右关联到对应列。

  取值范围：已存在的字段名。

- expression

  赋予对应 column 的一个有效表达式或值：

  - 如果是 INSERT ON DUPLICATE KEY UPDATE 语句下，expression 可以为 VALUES(column_name) 或 EXCLUDED.column_name 用来表示引用冲突行对应的 column_name 字段的值。需注意，其中 VALUES(column_name)不支持嵌套在表达式中（例如 VALUES(column_name)+1），但 EXCLUDED 不受此限制。
  - 向表中字段插入单引号 " ' "时需要使用单引号自身进行转义。
  - 如果插入行的表达式不是正确的数据类型，系统试图进行类型转换，若转换不成功，则插入数据失败，系统返回错误信息。

- DEFAULT

  对应字段名的缺省值。如果没有缺省值，则为 NULL。

- query

  一个查询语句（SELECT 语句），将查询结果作为插入的数据。

- RETURNING

  返回实际插入的行，RETURNING 列表的语法与 SELECT 的输出列表一致。注意：INSERT ON DUPLICATE KEY UPDATE 不支持 RETURNING 子句。

- output_expression

  INSERT 命令在每一行都被插入之后用于计算输出结果的表达式。

取值范围：该表达式可以使用 table 的任意字段。可以使用*返回被插入行的所有字段。

- output_name

  字段的输出名称。

  取值范围：字符串，符合标识符命名规范。

- ON DUPLICATE KEY UPDATE

  对于带有唯一约束（UNIQUE INDEX 或 PRIMARY KEY）的表，如果插入数据违反唯一约束，则对冲突行执行 UPDATE 子句完成更新。

  对于不带唯一约束的表，则仅执行插入。

  - 不支持触发器，不会触发目标表的 INSERT 或 UPDATE 触发器。
  - 不支持延迟生效（DEFERRABLE）的唯一约束或主键。
  - 如果表中存在多个唯一约束，如果所插入数据违反多个唯一约束，对于检测到冲突的第一行进行更新，其他冲突行不更新（检查顺序与索引维护具有强相关性，一般先创建的索引先进行冲突检查）。
  - 分布列、唯一索引列不允许 UPDATE。
  - 不支持列存。

示例

```
--创建表 tpcds.reason_t2。
vastbase=# CREATE TABLE tpcds.reason_t2
(
  r_reason_sk    integer,
  r_reason_id    character(16),
  r_reason_desc  character(100)
);

--向表中插入一条记录。
vastbase=# INSERT INTO tpcds.reason_t2(r_reason_sk, r_reason_id, r_reason_desc) VALUES (1,
'AAAAAAAABAAAAAAA', 'reason1');

--向表中插入一条记录，和上一条语法等效。
vastbase=# INSERT INTO tpcds.reason_t2 VALUES (2, 'AAAAAAAABAAAAAAA', 'reason2');

--向表中插入多条记录。
vastbase=# INSERT INTO tpcds.reason_t2 VALUES (3, 'AAAAAAAACAAAAAAA','reason3'),(4,
'AAAAAAAADAAAAAAA', 'reason4'),(5, 'AAAAAAAAEAAAAAAA','reason5');

--向表中插入 tpcds.reason 中 r_reason_sk 小于 5 的记录。
vastbase=# INSERT INTO tpcds.reason_t2 SELECT * FROM tpcds.reason WHERE r_reason_sk <5;

--对表创建唯一索引
vastbase=# CREATE UNIQUE INDEX reason_t2_u_index ON tpcds.reason_t2(r_reason_sk);

--向表中插入多条记录，如果冲突则更新冲突数据行中 r_reason_id 字段为'BBBBBBBBCAAAAAAA'。
```

```
vastbase=# INSERT INTO tpcds.reason_t2 VALUES (5, 'BBBBBBBBCAAAAAAA','reason5'),(6,
'AAAAAAAADAAAAAAA', 'reason6') ON DUPLICATE KEY UPDATE r_reason_id = 'BBBBBBBBCAAAAAAA';

--删除表tpcds.reason_t2。
vastbase=# DROP TABLE tpcds.reason_t2;
```

优化建议

- VALUES

  通过 insert 语句批量插入数据时，建议将多条记录合并入一条语句中执行插入，以提高数据加载性能。例如，INSERT INTO sections VALUES (30, 'Administration', 31, 1900),(40, 'Development', 35, 2000), (50, 'Development' , 60 , 2001);

11.16.90. LOCK

功能描述

LOCK TABLE 获取表级锁。

Vastbase 在为一个引用了表的命令自动请求锁时，尽可能选择最小限制的锁模式。如果用户需要一种更为严格的锁模式，可以使用 LOCK 命令。例如，一个应用是在 Read Committed 隔离级别上运行事务，并且它需要保证表中的数据在事务的运行过程中不被修改。为实现这个目的，则可以在查询之前对表使用 SHARE 锁模式进行锁定。这样将防止数据不被并发修改，从而保证后续的查询可以读到已提交的持久化的数据。因为 SHARE 锁模式与任何写操作需要的 ROW EXCLUSIVE 模式冲突，并且 LOCK TABLE name IN SHARE MODE 语句将等到所有当前持有 ROW EXCLUSIVE 模式锁的事务提交或回滚后才能执行。因此，一旦获得该锁，就不会存在未提交的写操作，并且其他操作也只能等到该锁释放之后才能开始。

注意事项

- LOCK TABLE 只能在一个事务块的内部有用，因为锁在事务结束时就会被释放。出现在任意事务块外面的 LOCK TABLE 都会报错。
- 如果没有声明锁模式，缺省为最严格的模式 ACCESS EXCLUSIVE。
- LOCK TABLE ... IN ACCESS SHARE MODE 需要在目标表上有 SELECT 权限。所有其他形式的 LOCK 需要 UPDATE 和/或 DELETE 权限。
- 没有 UNLOCK TABLE 命令，锁总是在事务结束时释放。
- LOCK TABLE 只处理表级的锁，因此那些带“ROW”字样的锁模式都是有歧义的。这些模式名称通常可理解为用户试图在一个被锁定的表中获取行级的锁。同样，ROW EXCLUSIVE 模式也是一个可共享的表级锁。注意，只要是涉及到 LOCK TABLE ，所有锁模式都有相同的语意，区别仅在于规则中锁与锁之间是否冲突，规则请参见表 11-56。

语法格式

```
LOCK [ TABLE ] {[ ONLY ] name [, ...]| {name [ * ]} [, ...]}
    [ IN {ACCESS SHARE | ROW SHARE | ROW EXCLUSIVE | SHARE UPDATE EXCLUSIVE | SHARE | SHARE ROW EXCLUSIVE
| EXCLUSIVE | ACCESS EXCLUSIVE} MODE ]
    [ NOWAIT ];
```

参数说明

表 11-56. 冲突的锁模式

请求的锁模式/当前锁模式	ACCESS SHARE	ROW SHARE	ROW EXCLUSIVE	SHARE UPDATE EXCLUSIVE	SHARE	SHARE ROW EXCLUSIVE	EXCLUSIVE	ACCESS EXCLUSIVE
ACCESS SHARE	-	-	-	-	-	-	-	X
ROW SHARE	-	-	-	-	-	-	X	X
ROW EXCLUSIVE	-	-	-	-	X	X	X	X
SHARE UPDATE EXCLUSIVE	-	-	-	X	X	X	X	X
SHARE	-	-	X	X	-	X	X	X
SHARE ROW EXCLUSIVE	-	-	X	X	X	X	X	X
EXCLUSIVE	-	X	X	X	X	X	X	X
ACCESS EXCLUSIVE	X	X	X	X	X	X	X	X

LOCK 的参数说明如下所示：

- name

  要锁定的表的名称，可以有模式修饰。

  LOCK TABLE 命令中声明的表的顺序就是上锁的顺序。

  取值范围：已存在的表名。

- ONLY

如果指定 ONLY，只有该表被锁定。如果没有声明，该表和他的所有子表将都被锁定。

- ACCESS SHARE

  ACCESS 锁只允许对表进行读取，而禁止对表进行修改。所有对表进行读取而不修改的 SQL 语句都会自动请求这种锁。例如，SELECT 命令会自动在被引用的表上请求一个这种锁。

- ROW SHARE

  ROW SHARE 锁允许对表进行并发读取，禁止对表进行其他操作。

  SELECT FOR UPDATE 和 SELECT FOR SHARE 命令会自动在目标表上请求 ROW SHARE 锁（且所有被引用但不是 FOR SHARE/FOR UPDATE 的其他表上，还会自动加上 ACCESS SHARE 锁）。

- ROW EXCLUSIVE

  与 ROW SHARE 锁相同，ROW EXCLUSIVE 允许并发读取表，但是禁止修改表中数据。UPDATE，DELETE，INSERT 命令会自动在目标表上请求这个锁（且所有被引用的其他表上还会自动加上的 ACCESS SHARE 锁）。通常情况下，所有会修改表数据的命令都会请求表的 ROW EXCLUSIVE 锁。

- SHARE UPDATE EXCLUSIVE

  这个模式保护一个表的模式不被并发修改，以及禁止在目标表上执行垃圾回收命令（VACUUM ）。

  VACUUM（不带 FULL 选项），ANALYZE，CREATE INDEX CONCURRENTLY 命令会自动请求这样的锁。

- SHARE

  SHARE 锁允许并发的查询，但是禁止对表进行修改。

  CREATE INDEX（不带 CONCURRENTLY 选项）语句会自动请求这种锁。

- SHARE ROW EXCLUSIVE

  SHARE ROW EXCLUSIVE 锁禁止对表进行任何的并发修改，而且是独占锁，因此一个会话中只能获取一次。

  任何 SQL 语句都不会自动请求这个锁模式。

- EXCLUSIVE

  EXCLUSIVE 锁允许对目标表进行并发查询，但是禁止任何其他操作。

  这个模式只允许并发加 ACCESS SHARE 锁，也就是说，只有对表的读动作可以和持有这个锁模式的事务并发执行。

  任何 SQL 语句都不会在用户表上自动请求这个锁模式。然而在某些操作的时候，会在某些系统表上请求它。

- ACCESS EXCLUSIVE

  这个模式保证其所有者（事务）是可以访问该表的唯一事务。

  ALTER TABLE，DROP TABLE，TRUNCATE，REINDEX，CLUSTER，VACUUM FULL 命令会自动请求这种锁。

  在 LOCK TABLE 命令没有明确声明需要的锁模式时，它是缺省锁模式。

- NOWAIT

  声明 LOCK TABLE 不去等待任何冲突的锁释放，如果无法立即获取该锁，该命令退出并且发出一个错误信息。

  在不指定 NOWAIT 的情况下获取表级锁时，如果有其他互斥锁存在的话，则等待其他锁的释放。

示例

```
--在执行删除操作时对一个有主键的表进行 SHARE ROW EXCLUSIVE 锁。
vastbase=# CREATE TABLE tpcds.reason_t1 AS TABLE tpcds.reason;

vastbase=# START TRANSACTION;

vastbase=# LOCK TABLE tpcds.reason_t1 IN SHARE ROW EXCLUSIVE MODE;

vastbase=# DELETE FROM tpcds.reason_t1 WHERE r_reason_desc IN(SELECT r_reason_desc FROM
tpcds.reason_t1 WHERE r_reason_sk < 6 );

vastbase=# DELETE FROM tpcds.reason_t1 WHERE r_reason_sk = 7;

vastbase=# COMMIT;

--删除表tpcds.reason_t1。
vastbase=# DROP TABLE tpcds.reason_t1;
```

11.16.91. MOVE

功能描述

MOVE 在不检索数据的情况下重新定位一个游标。MOVE 的作用类似于 11.16.87FETCH 命令，但只是重定位游标而不返回行。

注意事项

无。

语法格式

```
MOVE [ direction [ FROM | IN ] ] cursor_name;
```

其中 direction 子句为可选参数。

```
NEXT
   | PRIOR
   | FIRST
   | LAST
   | ABSOLUTE count
   | RELATIVE count
   | count
   | ALL
   | FORWARD
   | FORWARD count
   | FORWARD ALL
   | BACKWARD
   | BACKWARD count
   | BACKWARD ALL
```

参数说明

MOVE 命令的参数与 FETCH 的相同，详细请参见 FETCH 的参数说明。

说明

成功完成时，MOVE 命令将返回一个“MOVE count”的标签，count 是一个使用相同参数的 FETCH 命令会返回的行数（可能为零）。

示例

```
--开始一个事务。
vastbase=# START TRANSACTION;

--定义一个名为 cursor1 的游标。
vastbase=# CURSOR cursor1 FOR SELECT * FROM tpcds.reason;

--忽略游标 cursor1 的前 3 行。
vastbase=# MOVE FORWARD 3 FROM cursor1;

--抓取游标 cursor1 的前 4 行。
vastbase=# FETCH 4 FROM cursor1;
 r_reason_sk |   r_reason_id    |                                  r_reason_desc
-------------+------------------+--------------------------------------------------------------------------------
-------------------------------------
           4 | AAAAAAAAEAAAAAAA | Not the product that was ordred
           5 | AAAAAAAAFAAAAAAA | Parts missing
           6 | AAAAAAAAGAAAAAAA | Does not work with a product that I have
           7 | AAAAAAAAHAAAAAAA | Gift exchange
(4 rows)

--关闭游标。
vastbase=# CLOSE cursor1;

--结束一个事务。
vastbase=# END;
```

相关链接

11.16.30CLOSE，11.16.87FETCH

11.16.92. MERGE INTO

功能描述

通过 MERGE INTO 语句，将目标表和源表中数据针对关联条件进行匹配，若关联条件匹配时对目标表进行 UPDATE，无法匹配时对目标表执行 INSERT。此语法可以很方便地用来合并执行 UPDATE 和 INSERT，避免多次执行。

注意事项

- 进行 MERGE INTO 操作的用户需要同时拥有目标表的 UPDATE 和 INSERT 权限，以及源表的 SELECT 权限。
- 不支持重分布过程中 MERGE INTO。

语法格式

```
MERGE INTO table_name [ [ AS ] alias ]
USING { { table_name | view_name } | subquery } [ [ AS ] alias ]
ON ( condition )
[
  WHEN MATCHED THEN
  UPDATE SET { column_name = { expression | DEFAULT } |
         ( column_name [, ...] ) = ( { expression | DEFAULT } [, ...] ) } [, ...]
  [ WHERE condition ]
]
[
  WHEN NOT MATCHED THEN
  INSERT { DEFAULT VALUES |
  [ ( column_name [, ...] ) ] VALUES ( { expression | DEFAULT } [, ...] ) [, ...] [ WHERE condition ] }
];
```

参数说明

- INTO 子句

  指定正在更新或插入的目标表。目标表为复制表时，暂不支持目标表中某列默认值为 volatile 函数（如自增列）。

  - talbe_name

    目标表的表名。

  - alias

    目标表的别名。

取值范围：字符串，符合标识符命名规范。

- USING 子句

  指定源表，源表可以为表、视图或子查询。目标表为复制表时，暂不支持 USING 子句中包含非复制表。

- ON 子句

  关联条件，用于指定目标表和源表的关联条件。不支持更新关联条件中的字段。

- WHEN MATCHED 子句

  当源表和目标表中数据针对关联条件可以匹配上时，选择 WHEN MATCHED 子句进行 UPDATE 操作。

  不支持更新分布列。不支持更新系统表、系统列。

- WHEN NOT MATCHED 子句

  当源表和目标表中数据针对关联条件无法匹配时，选择 WHEN NOT MATCHED 子句进行 INSERT 操作。

  不支持 INSERT 子句中包含多个 VALUES。

  WHEN MATCHED 和 WHEN NOT MATCHED 子句顺序可以交换，可以缺省其中一个，但不能同时缺省，不支持同时指定两个 WHEN MATCHED 或 WHEN NOT MATCHED 子句。

- DEFAULT

  用对应字段的缺省值填充该字段。

  如果没有缺省值，则为 NULL。

- WHERE condition

  UPDATE 子句和 INSERT 子句的条件，只有在条件满足时才进行更新操作，可缺省。不支持 WHERE 条件中引用系统列。

示例

```
-- 创建目标表 products 和源表 newproducts，并插入数据
vastbase=# CREATE TABLE products
(
product_id INTEGER,
product_name VARCHAR2(60),
category VARCHAR2(60)
);

vastbase=# INSERT INTO products VALUES (1501, 'vivitar 35mm', 'electrncs');
vastbase=# INSERT INTO products VALUES (1502, 'olympus is50', 'electrncs');
vastbase=# INSERT INTO products VALUES (1600, 'play gym', 'toys');
vastbase=# INSERT INTO products VALUES (1601, 'lamaze', 'toys');
vastbase=# INSERT INTO products VALUES (1666, 'harry potter', 'dvd');
```

```
vastbase=# CREATE TABLE newproducts
(
product_id INTEGER,
product_name VARCHAR2(60),
category VARCHAR2(60)
);

vastbase=# INSERT INTO newproducts VALUES (1502, 'olympus camera', 'electrncs');
vastbase=# INSERT INTO newproducts VALUES (1601, 'lamaze', 'toys');
vastbase=# INSERT INTO newproducts VALUES (1666, 'harry potter', 'toys');
vastbase=# INSERT INTO newproducts VALUES (1700, 'wait interface', 'books');

-- 进行MERGE INTO操作
vastbase=# MERGE INTO products p
USING newproducts np
ON (p.product_id = np.product_id)
WHEN MATCHED THEN
  UPDATE SET p.product_name = np.product_name, p.category = np.category WHERE p.product_name != 'play
gym'
WHEN NOT MATCHED THEN
  INSERT VALUES (np.product_id, np.product_name, np.category) WHERE np.category = 'books';
MERGE 4

-- 查询更新后的结果
vastbase=# SELECT * FROM products ORDER BY product_id;
 product_id |  product_name  | category
------------+----------------+-----------
      1501 | vivitar 35mm   | electrncs
      1502 | olympus camera | electrncs
      1600 | play gym       | toys
      1601 | lamaze         | toys
      1666 | harry potter   | toys
      1700 | wait interface | books
(6 rows)

-- 删除表
vastbase=# DROP TABLE products;
vastbase=# DROP TABLE newproducts;
```

11.16.93. PREPARE

功能描述

创建一个预备语句。

预备语句是服务端的对象，可以用于优化性能。在执行 PREPARE 语句的时候，指定的查询被解析、分析、重写。当随后发出 EXECUTE 语句的时候，预备语句被规划和执行。这种设计避免了重复解析、分析工作。PREPARE 语句创建后在整个数据库会话期间一直存在，一旦创建成功，即便是在事务块中创建，事务回滚，PREPARE 也不会删除。只能通过显式调用 11.16.59DEALLOCATE 进行删除，会话结束时，PREPARE 也会自动删除。

注意事项

无。

语法格式

```
PREPARE name [ ( data_type [, ...] ) ] AS statement;
```

参数说明

- name

  指定预备语句的名称。它必须在该会话中是唯一的。

- data_type

  参数的数据类型。

- statement

  是 SELECT INSERT、UPDATE、DELETE、MERGE INTO 或 VALUES 语句之一。

示例

请参见 EXECUTE 的示例。

相关链接

11.16.59DEALLOCATE

11.16.94. PREPARE TRANSACTION

功能描述

为当前事务做两阶段提交的准备。

在命令之后，事务就不再和当前会话关联了；它的状态完全保存在磁盘上，它被提交成功的可能性非常高，即使是在请求提交之前数据库发生了崩溃也如此。

一旦准备好了，一个事务就可以在稍后用 11.16.34COMMIT PREPARED 或 11.16.101ROLLBACK PREPARED 命令分别进行提交或者回滚。这些命令可以从任何会话中发出，而不光是最初执行事务的那个会话。

从发出命令的会话的角度来看，PREPARE TRANSACTION 不同于 ROLLBACK：在执行它之后，就不再有活跃的当前事务了，并且预备事务的效果无法见到（在事务提交的时候其效果会再次可见）。

如果 PREPARE TRANSACTION 因为某些原因失败，那么它就会变成一个 ROLLBACK，当前事务被取消。

注意事项

- 事务功能由数据库自动维护，不应显式使用事务功能。
- 在运行 PREPARE TRANSACTION 命令时，必须在 postgresql.conf 配置文件中增大 max_prepared_transactions 的数值。建议至少将其设置为等于 max_connections，这样每个会话都可以有一个等待中的预备事务。

语法格式

```
PREPARE TRANSACTION transaction_id;
```

参数说明

transaction_id

待提交事务的标识符，用于后面在 COMMIT PREPARED 或 ROLLBACK PREPARED 的时候标识这个事务。它不能和任何当前预备事务已经使用了的标识符同名。

取值范围：标识符必须以字符串文本的方式书写，并且必须小于 200 字节长。

相关链接

11.16.34COMMIT PREPARED，11.16.101ROLLBACK PREPARED

11.16.95. REASSIGN OWNED

功能描述

修改数据库对象的属主。

REASSIGN OWNED 要求系统将所有 old_roles 拥有的数据库对象的属主更改为 new_role。

注意事项

- REASSIGN OWNED 常用于在删除角色之前的准备工作。
- 执行 REASSIGN OWNED 需要有原角色和目标角色上的权限。

语法格式

```
REASSIGN OWNED BY old_role [, ...] TO new_role;
```

参数说明

- old_role

  旧属主的角色名。

- new_role

将要成为这些对象属主的新角色的名称。

示例

无。

11.16.96. REINDEX

功能描述

为表中的数据重建索引。

在以下几种情况下需要使用 REINDEX 重建索引：

- 索引崩溃，并且不再包含有效的数据。
- 索引变得“臃肿”，包含大量的空页或接近空页。
- 为索引更改了存储参数（例如填充因子），并且希望这个更改完全生效。

  使用 CONCURRENTLY 选项创建索引失败，留下了一个“非法”索引。

注意事项

REINDEX DATABASE 和 SYSTEM 这种形式的重建索引不能在事务块中执行。

语法格式

- 重建普通索引。

```
REINDEX { INDEX | [INTERNAL] TABLE | DATABASE | SYSTEM } name [ FORCE ];
```

- 重建索引分区。

```
REINDEX  { INDEX|  [INTERNAL] TABLE} name
   PARTITION partition_name [ FORCE  ];
```

参数说明

- INDEX

  重新建立指定的索引。

- INTERNAL TABLE

  重建列存表或 Hadoop 内表的 Desc 表的索引，如果表有从属的"TOAST"表，则这个表也会重建索引。

- TABLE

  重新建立指定表的所有索引，如果表有从属的"TOAST"表，则这个表也会重建索引。

- DATABASE

  重建当前数据库里的所有索引。

- SYSTEM

  在当前数据库上重建所有系统表上的索引。不会处理在用户表上的索引。

- name

  需要重建索引的索引、表、数据库的名称。表和索引可以有模式修饰。

说明

REINDEX DATABASE 和 SYSTEM 只能重建当前数据库的索引，所以 name 必须和当前数据库名称相同。

- FORCE

  无效选项，会被忽略。

- partition_name

  需要重建索引的分区的名称或者索引分区的名称。

  取值范围：

  - 如果前面是 REINDEX INDEX，则这里应该指定索引分区的名称；
  - 如果前面是 REINDEX TABLE，则这里应该指定分区的名称；
  - 如果前面是 REINDEX INTERNAL TABLE，则这里应该指定列存分区表的分区的名称。

须知

REINDEX DATABASE 和 SYSTEM 这种形式的重建索引不能在事务块中执行。

示例

```
--创建一个行存表 tpcds.customer_t1，并在 tpcds.customer_t1 表上的 c_customer_sk 字段创建索引。
vastbase=# CREATE TABLE tpcds.customer_t1
(
    c_customer_sk             integer               not null,
    c_customer_id             char(16)              not null,
    c_current_cdemo_sk        integer                       ,
    c_current_hdemo_sk        integer                       ,
    c_current_addr_sk         integer                       ,
    c_first_shipto_date_sk    integer                       ,
    c_first_sales_date_sk     integer                       ,
    c_salutation              char(10)                      ,
    c_first_name              char(20)                      ,
    c_last_name               char(30)                      ,
    c_preferred_cust_flag     char(1)                       ,
    c_birth_day               integer                       ,
    c_birth_month             integer                       ,
    c_birth_year              integer                       ,
    c_birth_country           varchar(20)                   ,
    c_login                   char(13)                      ,
    c_email_address           char(50)                      ,
    c_last_review_date        char(10)
)
```

```
WITH (orientation = row)

vastbase=# CREATE INDEX tpcds_customer_index1 ON tpcds.customer_t1 (c_customer_sk);

vastbase=# INSERT INTO tpcds.customer_t1 SELECT * FROM tpcds.customer WHERE c_customer_sk < 10;

--重建一个单独索引。
vastbase=# REINDEX INDEX tpcds.tpcds_customer_index1;

--重建表 tpcds.customer_t1 上的所有索引。
vastbase=# REINDEX TABLE tpcds.customer_t1;

--删除 tpcds.customer_t1 表。
vastbase=# DROP TABLE tpcds.customer_t1;
```

优化建议

- INTERNAL TABLE

  此种情况大多用于故障恢复，不建议进行并发操作。

- DATABASE

  不建议在事务中 reindex database。

- SYSTEM

  不建议在事务中 reindex 系统表。

11.16.97. RELEASE SAVEPOINT

功能描述

RELEASE SAVEPOINT 删除一个当前事务先前定义的保存点。

把一个保存点删除就令其无法作为回滚点使用，除此之外它没有其它用户可见的行为。它并不能撤销在保存点建立起来之后执行的命令的影响。要撤销那些命令可以使用 ROLLBACK TO SAVEPOINT 。在不再需要的时候删除一个保存点可以令系统在事务结束之前提前回收一些资源。

RELEASE SAVEPOINT 也删除所有在指定的保存点建立之后的所有保存点。

注意事项

- 不能 RELEASE 一个没有定义的保存点，语法上会报错。
- 如果事务在回滚状态，则不能释放保存点。
- 如果多个保存点拥有同样的名称，只有最近定义的那个才被释放。

语法格式

```
RELEASE [ SAVEPOINT ] savepoint_name;
```

参数说明

- savepoint_name
  要删除的保存点的名称

示例

```
--创建一个新表。
vastbase=# CREATE TABLE tpcds.table1(a int);

--开启事务。
vastbase=# START TRANSACTION;

--插入数据。
vastbase=# INSERT INTO tpcds.table1 VALUES (3);

--建立保存点。
vastbase=# SAVEPOINT my_savepoint;

--插入数据。
vastbase=# INSERT INTO tpcds.table1 VALUES (4);

--删除保存点。
vastbase=# RELEASE SAVEPOINT my_savepoint;

--提交事务。
vastbase=# COMMIT;

--查询表的内容，会同时看到 3 和 4。
vastbase=# SELECT * FROM tpcds.table1;

--删除表。
vastbase=# DROP TABLE tpcds.table1;
```

相关链接

11.16.103SAVEPOINT，11.16.102ROLLBACK TO SAVEPOINT

11.16.98. RESET

功能描述

RESET 将指定的运行时参数恢复为缺省值。这些参数的缺省值是指 postgresql.conf 配置文件中所描述的参数缺省值。

RESET 命令与如下命令的作用相同：

SET configuration_parameter TO DEFAULT

注意事项

RESET 的事务性行为和 SET 相同：它的影响将会被事务回滚撤销。

语法格式

```
RESET {configuration_parameter | CURRENT_SCHEMA | TIME ZONE | TRANSACTION ISOLATION LEVEL | SESSION
AUTHORIZATION | ALL };
```

参数说明

- configuration_parameter

  运行时参数的名称。

  取值范围：可以使用 SHOW ALL 命令查看运行时参数。

- CURRENT_SCHEMA

  当前模式

- TIME ZONE

  时区。

- TRANSACTION ISOLATION LEVEL

  事务的隔离级别。

- SESSION AUTHORIZATION

  当前会话的用户标识符。

- ALL

  所有运行时参数。

示例

```
--把 timezone 设为缺省值。
vastbase=# RESET timezone;

--把所有参数设置为缺省值。
vastbase=# RESET ALL;
```

相关链接

11.16.105SET，11.16.110SHOW

11.16.99. REVOKE

功能描述

REVOKE 用于撤销一个或多个角色的权限。

注意事项

非对象所有者试图在对象上 REVOKE 权限，命令按照以下规则执行：

- 如果授权用户没有该对象上的权限，则命令立即失败。
- 如果授权用户有部分权限，则只撤销那些有授权选项的权限。
- 如果授权用户没有授权选项，REVOKE ALL PRIVILEGES 形式将发出一个错误信息，而对于其他形式的命令而言，如果是命令中指定名称的权限没有相应的授权选项，该命令将发出一个警告。
- 不允许对表分区进行 REVOKE 操作，对分区表进行 REVOKE 操作会引起告警。

语法格式

- 回收指定表和视图上权限。

```
REVOKE [ GRANT OPTION FOR ]
    { { SELECT | INSERT | UPDATE | DELETE | TRUNCATE | REFERENCES }[, ...]
    | ALL [ PRIVILEGES ] }
    ON { [ TABLE ] table_name [, ...]
       | ALL TABLES IN SCHEMA schema_name [, ...] }
    FROM { [ GROUP ] role_name | PUBLIC } [, ...]
    [ CASCADE | RESTRICT ];
```

- 回收表上指定字段权限。

```
REVOKE [ GRANT OPTION FOR ]
    { {{ SELECT | INSERT | UPDATE | REFERENCES } ( column_name [, ...] )}[, ...]
    | ALL [ PRIVILEGES ] ( column_name [, ...] ) }
    ON [ TABLE ] table_name [, ...]
    FROM { [ GROUP ] role_name | PUBLIC } [, ...]
    [ CASCADE | RESTRICT ];
```

- 回收指定数据库上权限。

```
REVOKE [ GRANT OPTION FOR ]
    { { CREATE | CONNECT | TEMPORARY | TEMP } [, ...]
    | ALL [ PRIVILEGES ] }
    ON DATABASE database_name [, ...]
    FROM { [ GROUP ] role_name | PUBLIC } [, ...]
    [ CASCADE | RESTRICT ];
```

- 回收指定函数上权限。

```
REVOKE [ GRANT OPTION FOR ]
    { EXECUTE | ALL [ PRIVILEGES ] }
    ON { FUNCTION {function_name ( [ {[ argmode ] [ arg_name ] arg_type} [, ...] ] )} [, ...]
       | ALL FUNCTIONS IN SCHEMA schema_name [, ...] }
    FROM { [ GROUP ] role_name | PUBLIC } [, ...]
    [ CASCADE | RESTRICT ];
```

- 回收指定大对象上权限。

```
REVOKE [ GRANT OPTION FOR ]
    { { SELECT | UPDATE } [, ...] | ALL [ PRIVILEGES ] }
    ON LARGE OBJECT loid [, ...]
    FROM { [ GROUP ] role_name | PUBLIC } [, ...]
    [ CASCADE | RESTRICT ];
```

- 回收指定模式上权限。

```
REVOKE [ GRANT OPTION FOR ]
    { { CREATE | USAGE } [, ...] | ALL [ PRIVILEGES ] }
    ON SCHEMA schema_name [, ...]
    FROM { [ GROUP ] role_name | PUBLIC } [, ...]
    [ CASCADE | RESTRICT ];
```

- 回收指定表空间上权限。

```
REVOKE [ GRANT OPTION FOR ]
    { CREATE | ALL [ PRIVILEGES ] }
    ON TABLESPACE tablespace_name [, ...]
    FROM { [ GROUP ] role_name | PUBLIC } [, ...]
    [ CASCADE | RESTRICT ];
```

- 按角色回收角色上的权限。

```
REVOKE [ ADMIN OPTION FOR ]
    role_name [, ...] FROM role_name [, ...]
    [ CASCADE | RESTRICT ];
```

- 回收角色上的 sysadmin 权限。

```
REVOKE ALL { PRIVILEGES | PRIVILEGE } FROM role_name;
```

- 回收 Data Source 对象上的权限。

```
REVOKE {USAGE | ALL [PRIVILEGES]}
    ON DATA SOURCE src_name [, ...]
    FROM {[GROUP] role_name | PUBLIC} [, ...];
```

- 回收 directory 对象的权限。

```
REVOKE {READ|WRITE| ALL [PRIVILEGES]}
   ON DIRECTORY src_name [, ...]
   FROM {[GROUP] role_name | PUBLIC} [, ...] [WITH GRANT OPTION];
```

参数说明

关键字 PUBLIC 表示一个隐式定义的拥有所有角色的组。

权限类别和参数说明，请参见 GRANT 的参数说明。

任何特定角色拥有的特权包括直接授予该角色的特权、从该角色作为其成员的角色中得到的权限以及授予给 PUBLIC 的权限。因此，从 PUBLIC 收回 SELECT 特权并不一定会意味着所有角色都会失去在该对象上的 SELECT 特权，那些直接被授予的或者通过另一个角色被授予的角色仍然会拥有它。类似地，从一个用户收回 SELECT 后，如果 PUBLIC 仍有 SELECT 权限，该用户还是可以使用 SELECT。

指定 GRANT OPTION FOR 时，只撤销对该权限授权的权力，而不撤销该权限本身。

如用户 A 拥有某个表的 UPDATE 权限，及 WITH GRANT OPTION 选项，同时 A 把这个权限赋予了用户 B，则用户 B 持有的权限称为依赖性权限。当用户 A 持有的权限或者授权选项被撤销时，依赖性权限仍然存在，但如果声明了 CASCADE，则所有依赖性权限都被撤销。

一个用户只能撤销由它自己直接赋予的权限。例如，如果用户 A 被指定授权（WITH ADMIN OPTION）选项，且把一个权限赋予了用户 B，然后用户 B 又赋予了用户 C，则用户 A 不能直接将 C 的权限撤销。但

是，用户 A 可以撤销用户 B 的授权选项，并且使用 CASCADE。这样，用户 C 的权限就会自动被撤销。另外一个例子：如果 A 和 B 都赋予了 C 同样的权限，则 A 可以撤销他自己的授权选项，但是不能撤销 B 的，因此 C 仍然拥有该权限。

如果执行 REVOKE 的角色持有的权限是通过多层成员关系获得的，则具体是哪个包含的角色执行的该命令是不确定的。在这种场合下，最好的方法是使用 SET ROLE 成为特定角色，然后执行 REVOKE，否则可能导致删除了不想删除的权限，或者是任何权限都没有删除。

示例

请参考 GRANT 的示例。

相关链接

11.16.88GRANT

11.16.100. ROLLBACK

功能描述

回滚当前事务并取消当前事务中的所有更新。

在事务运行的过程中发生了某种故障，事务不能继续执行，系统将事务中对数据库的所有已完成的操作全部撤销，数据库状态回到事务开始时。

注意事项

如果不在一个事务内部发出 ROLLBACK 不会有问题，但是将抛出一个 NOTICE 信息。

语法格式

```
ROLLBACK [ WORK | TRANSACTION ];
```

参数说明

- WORK | TRANSACTION

  可选关键字。除了增加可读性，没有任何其他作用。

示例

```
--开启一个事务
vastbase=# START TRANSACTION;

--取消所有更改
vastbase=# ROLLBACK;
```

相关链接

11.16.33COMMIT | END

11.16.101. ROLLBACK PREPARED

功能描述

取消一个先前为两阶段提交准备好的事务。

注意事项

- 该功能仅在维护模式(GUC 参数 xc_maintenance_mode 为 on 时)下可用。该模式谨慎打开，一般供维护人员排查问题使用，一般用户不应使用该模式。
- 要想回滚一个预备事务，必须是最初发起事务的用户，或者是系统管理员。
- 事务功能由数据库自动维护，不应显式使用事务功能。

语法格式

```
ROLLBACK PREPARED transaction_id ;
```

参数说明

- transaction_id
  待提交事务的标识符。它不能和任何当前预备事务已经使用了的标识符同名。

相关链接

11.16.34COMMIT PREPARED，11.16.94PREPARE TRANSACTION。

11.16.102. ROLLBACK TO SAVEPOINT

功能描述

ROLLBACK TO SAVEPOINT 用于回滚到一个保存点，隐含地删除所有在该保存点之后建立的保存点。回滚所有指定保存点建立之后执行的命令。保存点仍然有效，并且需要时可以再次回滚到该点。

注意事项

- 不能回滚到一个未定义的保存点，语法上会报错。
- 在保存点方面，游标有一些非事务性的行为。任何在保存点里打开的游标都会在回滚掉这个保存点之后关闭。如果一个前面打开了的游标在保存点里面，并且游标被一个 FETCH 命令影响，而这个保存点稍后回滚了，那么这个游标的位置仍然在 FETCH 让它指向的位置(也就是 FETCH 不会

被回滚)。关闭一个游标的行为也不会被回滚给撤消掉。如果一个游标的操作导致事务回滚，那么这个游标就会置于不可执行状态，所以，尽管一个事务可以用 ROLLBACK TO SAVEPOINT 重新恢复，但是游标不能再使用了。

- 使用 ROLLBACK TO SAVEPOINT 回滚到一个保存点。使用 RELEASE SAVEPOINT 删除一个保存点，但是保留该保存点建立后执行的命令的效果。

语法格式

```
ROLLBACK [ WORK | TRANSACTION ] TO [ SAVEPOINT ] savepoint_name;
```

参数说明

- savepoint_name
  回滚截至的保存点

示例

```
--撤销 my_savepoint 建立之后执行的命令的影响。
vastbase=# START TRANSACTION;
vastbase=# SAVEPOINT my_savepoint;
vastbase=# ROLLBACK TO SAVEPOINT my_savepoint;
--游标位置不受保存点回滚的影响。
vastbase=# DECLARE foo CURSOR FOR SELECT 1 UNION SELECT 2;
vastbase=# SAVEPOINT foo;
vastbase=# FETCH 1 FROM foo;
 ?column?
----------
        1
vastbase=# ROLLBACK TO SAVEPOINT foo;
vastbase=# FETCH 1 FROM foo;
 ?column?
----------
        2
vastbase=# RELEASE SAVEPOINT my_savepoint;
vastbase=# COMMIT;
```

相关链接

11.16.103SAVEPOINT，11.16.97RELEASE SAVEPOINT

11.16.103. SAVEPOINT

功能描述

SAVEPOINT 用于在当前事务里建立一个新的保存点。

保存点是事务中的一个特殊记号，它允许将那些在它建立后执行的命令全部回滚，把事务的状态恢复到保存点所在的时刻。

注意事项

- 使用 ROLLBACK TO SAVEPOINT 回滚到一个保存点。使用 RELEASE SAVEPOINT 删除一个保存点，但是保留该保存点建立后执行的命令的效果。
- 保存点只能在一个事务块里面建立。在一个事务里面可以定义多个保存点。
- 函数、匿名块和存储过程中不支持使用 SAVEPOINT 语法。
- 由于节点故障或者通信故障引起的分布式节点线程或进程退出导致的报错，以及由于 COPY FROM 操作中源数据与目标表的表结构不一致导致的报错，均不能正常回滚到保存点之前，而是整个事务回滚。
- SQL 标准要求，使用 savepoint 建立一个同名保存点时，需要自动删除前面那个同名保存点。在 Vastbase 数据库里，我们将保留旧的保存点，但是在回滚或者释放的时候，只使用最近的那个。释放了新的保存点将导致旧的再次成为 ROLLBACK TO SAVEPOINT 和 RELEASE SAVEPOINT 可以访问的保存点。除此之外，SAVEPOINT 是完全符合 SQL 标准的。

语法格式

```
SAVEPOINT savepoint_name;
```

参数说明

- savepoint_name
  新建保存点的名称。

示例

```
--创建一个新表。
vastbase=# CREATE TABLE table1(a int);

--开启事务。
vastbase=# START TRANSACTION;

--插入数据。
vastbase=# INSERT INTO table1 VALUES (1);

--建立保存点。
vastbase=# SAVEPOINT my_savepoint;

--插入数据。
vastbase=# INSERT INTO table1 VALUES (2);

--回滚保存点。
vastbase=# ROLLBACK TO SAVEPOINT my_savepoint;

--插入数据。
vastbase=# INSERT INTO table1 VALUES (3);

--提交事务。
```

```
vastbase=# COMMIT;

--查询表的内容，会同时看到 1 和 3,不能看到 2，因为 2 被回滚。
vastbase=# SELECT * FROM table1;

--删除表。
vastbase=# DROP TABLE table1;

--创建一个新表。
vastbase=# CREATE TABLE table2(a int);

--开启事务。
vastbase=# START TRANSACTION;

--插入数据。
vastbase=# INSERT INTO table2 VALUES (3);

--建立保存点。
vastbase=# SAVEPOINT my_savepoint;

--插入数据。
vastbase=# INSERT INTO table2 VALUES (4);

--回滚保存点。
vastbase=# RELEASE SAVEPOINT my_savepoint;

--提交事务。
vastbase=# COMMIT;

--查询表的内容，会同时看到 3 和 4。
vastbase=# SELECT * FROM table2;

--删除表。
vastbase=# DROP TABLE table2;
```

相关链接

11.16.97RELEASE SAVEPOINT，11.16.102ROLLBACK TO SAVEPOINT

- SELECT

功能描述

SELECT 用于从表或视图中取出数据。

SELECT 语句就像叠加在数据库表上的过滤器，利用 SQL 关键字从数据表中过滤出用户需要的数据。

注意事项

- 必须对每个在 SELECT 命令中使用的字段有 SELECT 权限。
- 使用 FOR UPDATE 或 FOR SHARE 还要求 UPDATE 权限。

语法格式

- 查询数据

```
[ WITH [ RECURSIVE ] with_query [, ...] ]
SELECT [/*+ plan_hint */] [ ALL | DISTINCT [ ON ( expression [, ...] ) ] ]
{ * | {expression [ [ AS ] output_name ]} [, ...] }
[ FROM from_item [, ...] ]
[ WHERE condition ]
[ GROUP BY grouping_element [, ...] ]
[ HAVING condition [, ...] ]
[ WINDOW {window_name AS ( window_definition )} [, ...] ]
[ { UNION | INTERSECT | EXCEPT | MINUS } [ ALL | DISTINCT ] select ]
[ ORDER BY {expression [ [ ASC | DESC | USING operator ] | nlssort_expression_clause ] [ NULLS { FIRST
| LAST } ]} [, ...] ]
[ LIMIT { [offset,] count | ALL } ]
[ OFFSET start [ ROW | ROWS ] ]
[ FETCH { FIRST | NEXT } [ count ] { ROW | ROWS } ONLY ]
[ {FOR { UPDATE | SHARE } [ OF table_name [, ...] ] [ NOWAIT ]} [...] ];
```

说明

condition 和 expression 中可以使用 targetlist 中表达式的别名。

- 只能同一层引用。
- 只能引用 targetlist 中的别名。
- 只能是后面的表达式引用前面的表达式。
- 不能包含 volatile 函数。
- 不能包含 Window function 函数。
- 不支持在 join on 条件中引用别名。
- targetlist 中有多个要应用的别名则报错。

- 其中子查询 with_query 为：

```
with_query_name [ ( column_name [, ...] ) ]
    AS ( {select | values | insert | update | delete} )
```

- 其中指定查询源 from_item 为：

```
{[ ONLY ] table_name [ * ] [ partition_clause ] [ [ AS ] alias [ ( column_alias [, ...] ) ] ]
[ TABLESAMPLE sampling_method ( argument [, ...] ) [ REPEATABLE ( seed ) ] ]
|( select ) [ AS ] alias [ ( column_alias [, ...] ) ]
|with_query_name [ [ AS ] alias [ ( column_alias [, ...] ) ] ]
|function_name ( [ argument [, ...] ] ) [ AS ] alias [ ( column_alias [, ...] | column_definition
[, ...] ) ]
|function_name ( [ argument [, ...] ] ) AS ( column_definition [, ...] )
|from_item [ NATURAL ] join_type from_item [ ON join_condition | USING ( join_column [, ...] ) ]}
```

- 其中 group 子句为：

```
( )
| expression
| ( expression [, ...] )
| ROLLUP ( { expression | ( expression [, ...] ) } [, ...] )
| CUBE ( { expression | ( expression [, ...] ) } [, ...] )
| GROUPING SETS ( grouping_element [, ...] )
```

- 其中指定分区 partition_clause 为：

```
PARTITION { ( partition_name ) |
      FOR ( partition_value [, ...] ) }
```

说明

指定分区只适合普通表。

- 其中设置排序方式 nlssort_expression_clause 为：

```
NLSSORT ( column_name, ' NLS_SORT = { SCHINESE_PINYIN_M | generic_m_ci } ' )
```

- 简化版查询语法，功能相当于 select * from table_name。

```
TABLE { ONLY {(table_name)| table_name} | table_name [ * ]};
```

参数说明

- WITH [RECURSIVE] with_query [, ...]

  用于声明一个或多个可以在主查询中通过名称引用的子查询，相当于临时表。

  如果声明了 RECURSIVE，那么允许 SELECT 子查询通过名称引用它自己。

  其中 with_query 的详细格式为：with_query_name [(column_name [, ...])] AS ({select | values | insert | update | delete})

  - with_query_name 指定子查询生成的结果集名称，在查询中可使用该名称访问子查询的结果集。
  - column_name 指定子查询结果集中显示的列名。
  - 每个子查询可以是 SELECT，VALUES，INSERT，UPDATE 或 DELETE 语句。

- plan_hint 子句

  以/*+ */的形式在 SELECT 关键字后，用于对 SELECT 对应的语句块生成的计划进行 hint 调优。

- ALL

  声明返回所有符合条件的行，是默认行为，可以省略该关键字。

- DISTINCT [ON (expression [, ...])]

  从 SELECT 的结果集中删除所有重复的行，使结果集中的每行都是唯一的。

  ON (expression [, ...]) 只保留那些在给出的表达式上运算出相同结果的行集合中的第一行。

须知

DISTINCT ON 表达式是使用与 ORDER BY 相同的规则进行解释的。除非使用了 ORDER BY 来保证需要的行首先出现，否则，"第一行" 是不可预测的。

- SELECT 列表

  指定查询表中列名，可以是部分列或者是全部（使用通配符*表示）。

  通过使用子句 AS output_name 可以为输出字段取个别名，这个别名通常用于输出字段的显示。

  列名可以用下面几种形式表达：

- 手动输入列名，多个列之间用英文逗号（,）分隔。
- 可以是 FROM 子句里面计算出来的字段。

❖ FROM 子句

为 SELECT 声明一个或者多个源表。

FROM 子句涉及的元素如下所示。

- table_name

  表名或视图名，名称前可加上模式名，如：schema_name.table_name。

- alias

  给表或复杂的表引用起一个临时的表别名，以便被其余的查询引用。

  别名用于缩写或者在自连接中消除歧义。如果提供了别名，它就会完全隐藏表的实际名称。

- TABLESAMPLE sampling_method (argument [, ...]) [REPEATABLE (seed)]

  table_name 之后的 TABLESAMPLE 子句表示应该用指定的 *sampling_method* 来检索表中行的子集。

  可选的 REPEATABLE 子句指定一个用于产生采样方法中随机数的*种子*数。种子值可以是任何非空常量值。如果查询时表没有被更改，指定相同种子和 *argument* 值的两个查询将会选择该表相同的采样。但是不同的种子值通常将会产生不同的采样。如果没有给出 REPEATABLE，则会基于一个系统产生的种子为每一个查询选择一个新的随机采样。

- column_alias

  列别名

- PARTITION

  查询分区表的某个分区的数据。

- partition_name

  分区名。

- partition_value

  指定的分区键值。在创建分区表时，如果指定了多个分区键，可以通过 PARTITION FOR 子句指定的这一组分区键的值，唯一确定一个分区。

- subquery

  FROM 子句中可以出现子查询，创建一个临时表保存子查询的输出。

- with_query_name

  WITH 子句同样可以作为 FROM 子句的源，可以通过 WITH 查询的名称对其进行引用。

- function_name

函数名称。函数调用也可以出现在 FROM 子句中。

- join_type

  有 5 种类型，如下所示。

  - [INNER] JOIN

    一个 JOIN 子句组合两个 FROM 项。可使用圆括弧以决定嵌套的顺序。如果没有圆括弧，JOIN 从左向右嵌套。

    在任何情况下，JOIN 都比逗号分隔的 FROM 项绑定得更紧。

  - LEFT [OUTER] JOIN

    返回笛卡尔积中所有符合连接条件的行，再加上左表中通过连接条件没有匹配到右表行的那些行。这样，左边的行将扩展为生成表的全长，方法是在那些右表对应的字段位置填上 NULL。请注意，只在计算匹配的时候，才使用 JOIN 子句的条件，外层的条件是在计算完毕之后施加的。

  - RIGHT [OUTER] JOIN

    返回所有内连接的结果行，加上每个不匹配的右边行（左边用 NULL 扩展）。

    这只是一个符号上的方便，因为总是可以把它转换成一个 LEFT OUTER JOIN，只要把左边和右边的输入互换位置即可。

  - FULL [OUTER] JOIN

    返回所有内连接的结果行，加上每个不匹配的左边行（右边用 NULL 扩展），再加上每个不匹配的右边行（左边用 NULL 扩展）。

  - CROSS JOIN

    CROSS JOIN 等效于 INNER JOIN ON（TRUE），即没有被条件删除的行。这种连接类型只是符号上的方便，因为它们与简单的 FROM 和 WHERE 的效果相同。

说明

必须为 INNER 和 OUTER 连接类型声明一个连接条件，即 NATURAL ON，join_condition，USING (join_column [，...]) 之一。但是它们不能出现在 CROSS JOIN 中。

其中 CROSS JOIN 和 INNER JOIN 生成一个简单的笛卡尔积，和在 FROM 的顶层列出两个项的结果相同。

- ON join_condition

  连接条件，用于限定连接中的哪些行是匹配的。如：ON left_table.a = right_table.a。

- USING(join_column[，...])

  ON left_table.a = right_table.a AND left_table.b = right_table.b ... 的简写。要求对应的列必须同名。

- NATURAL

  NATURAL 是具有相同名称的两个表的所有列的 USING 列表的简写。

- from item

  用于连接的查询源对象的名称。

❖ WHERE 子句

WHERE 子句构成一个行选择表达式，用来缩小 SELECT 查询的范围。condition 是返回值为布尔型的任意表达式，任何不满足该条件的行都不会被检索。

WHERE 子句中可以通过指定"(+)"操作符的方法将表的连接关系转换为外连接。但是不建议用户使用这种用法，因为这并不是 SQL 的标准语法，在做平台迁移的时候可能面临语法兼容性的问题。同时，使用"(+)"有很多限制：

a. "(+)"只能出现在 where 子句中。
b. 如果 from 子句中已经有指定表连接关系，那么不能再在 where 子句中使用"(+)"。
c. "(+)"只能作用在表或者视图的列上，不能作用在表达式上。
d. 如果表 A 和表 B 有多个连接条件，那么必须在所有的连接条件中指定"(+)"，否则"(+)"将不会生效，表连接会转化成内连接，并且不给出任何提示信息。
e. "(+)"作用的连接条件中的表不能跨查询或者子查询。如果"(+)"作用的表，不在当前查询或者子查询的 from 子句中，则会报错。如果"(+)"作用的对端的表不存在，则不报错，同时连接关系会转化为内连接。
f. "(+)"作用的表达式不能直接通过"OR"连接。
g. 如果"(+)"作用的列是和一个常量的比较关系，那么这个表达式会成为 join 条件的一部分。
h. 同一个表不能对应多个外表。
i. "(+)"只能出现"比较表达式"，"NOT 表达式"，“ANY 表达式”，“ALL 表达式”，“IN 表达式”，“NULLIF 表达式”，“IS DISTINCT FROM 表达式”，“IS OF”表达式。"(+)"不能出现在其他类型表达式中，并且这些表达式中不允许出现通过“AND”和“OR”连接的表达式。
j. "(+)"只能转化为左外连接或者右外连接，不能转化为全连接，即不能在一个表达式的两个表上同时指定"(+)"

须知

对于 WHERE 子句的 LIKE 操作符，当 LIKE 中要查询特殊字符“%”、“_”、“\”的时候需要使用反斜杠“\”来进行转义。

- GROUP BY 子句

  将查询结果按某一列或多列的值分组，值相等的为一组。

  - CUBE ({ expression | (expression [, ...]) } [, ...])

    CUBE 是自动对 group by 子句中列出的字段进行分组汇总，结果集将包含维度列中各值的所有可能组合，以及与这些维度值组合相匹配的基础行中的聚合值。它会为每个分组返回一行汇总信息， 用户可以使用 CUBE 来产生交叉表值。比如，在 CUBE 子句中给出三个表达式（n = 3），运算结果为 $2^n = 2^3 = 8$ 组。 以 n 个表达式的值分组的行称为常规行，其余的行称为超级聚集行。

  - GROUPING SETS (grouping_element [, ...])

    GROUPING SETS 子句是 GROUP BY 子句的进一步扩展，它可以使用户指定多个 GROUP BY 选项。 这样做可以通过裁剪用户不需要的数据组来提高效率。 当用户指定了所需的数据组时，数据库不需要执行完整 CUBE 或 ROLLUP 生成的聚合集合。

须知

如果 SELECT 列表的表达式中引用了那些没有分组的字段，则会报错，除非使用了聚集函数，因为对于未分组的字段，可能返回多个数值。

- HAVING 子句

  与 GROUP BY 子句配合用来选择特殊的组。HAVING 子句将组的一些属性与一个常数值比较，只有满足 HAVING 子句中的逻辑表达式的组才会被提取出来。

- WINDOW 子句

  一般形式为 WINDOW window_name AS (window_definition) [， ...]，window_name 是可以被随后的窗口定义所引用的名称，window_definition 可以是以下的形式：

  [existing_window_name]

  [PARTITION BY expression [, ...]]

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

  [frame_clause]

  frame_clause 为窗函数定义一个窗口框架 window frame，窗函数（并非所有）依赖于框架，window frame 是当前查询行的一组相关行。frame_clause 可以是以下的形式：

  [RANGE | ROWS] frame_start

[RANGE | ROWS] BETWEEN frame_start AND frame_end

frame_start 和 frame_end 可以是：

UNBOUNDED PRECEDING

value PRECEDING

CURRENT ROW

value FOLLOWING

UNBOUNDED FOLLOWING

须知

对列存表的查询目前只支持 row_number 窗口函数，不支持 frame_clause。

- UNION 子句

  UNION 计算多个 SELECT 语句返回行集合的并集。

  UNION 子句有如下约束条件：

  - 除非声明了 ALL 子句，否则缺省的 UNION 结果不包含重复的行。
  - 同一个 SELECT 语句中的多个 UNION 操作符是从左向右计算的，除非用圆括弧进行了标识。
  - FOR UPDATE 不能在 UNION 的结果或输入中声明。

  一般表达式：

  select_statement UNION [ALL] select_statement

  - select_statement 可以是任何没有 ORDER BY、LIMIT、FOR UPDATE 子句的 SELECT 语句。
  - 如果用圆括弧包围，ORDER BY 和 LIMIT 可以附着在子表达式里。

- INTERSECT 子句

  INTERSECT 计算多个 SELECT 语句返回行集合的交集，不含重复的记录。

  INTERSECT 子句有如下约束条件：

  - 同一个 SELECT 语句中的多个 INTERSECT 操作符是从左向右计算的，除非用圆括弧进行了标识。
  - 当对多个 SELECT 语句的执行结果进行 UNION 和 INTERSECT 操作的时候，会优先处理 INTERSECT。

  一般形式：

  select_statement INTERSECT select_statement

select_statement 可以是任何没有 FOR UPDATE 子句的 SELECT 语句。

- EXCEPT 子句

  EXCEPT 子句有如下的通用形式：

  select_statement EXCEPT [ALL] select_statement

  select_statement 是任何没有 FOR UPDATE 子句的 SELECT 表达式。

  EXCEPT 操作符计算存在于左边 SELECT 语句的输出而不存在于右边 SELECT 语句输出的行。

  EXCEPT 的结果不包含任何重复的行，除非声明了 ALL 选项。使用 ALL 时，一个在左边表中有 m 个重复而在右边表中有 n 个重复的行将在结果中出现 max(m-n,0) 次。

  除非用圆括弧指明顺序，否则同一个 SELECT 语句中的多个 EXCEPT 操作符是从左向右计算的。EXCEPT 和 UNION 的绑定级别相同。

  目前，不能给 EXCEPT 的结果或者任何 EXCEPT 的输入声明 FOR UPDATE 子句。

- MINUS 子句

  与 EXCEPT 子句具有相同的功能和用法。

- ORDER BY 子句

  对 SELECT 语句检索得到的数据进行升序或降序排序。对于 ORDER BY 表达式中包含多列的情况：

  - 首先根据最左边的列进行排序，如果这一列的值相同，则根据下一个表达式进行比较，依此类推。
  - 如果对于所有声明的表达式都相同，则按随机顺序返回。
  - ORDER BY 中排序的列必须包括在 SELECT 语句所检索的结果集的列中。

须知

如果要支持中文拼音排序和不区分大小写排序，需要在初始化数据库时指定编码格式为 UTF-8 或 GBK。 命令如下: initdb –E UTF8 –D ../data –locale=zh_CN.UTF-8 或 initdb –E GBK –D ../data –locale=zh_CN.GBK。

- LIMIT 子句

  LIMIT 子句由两个独立的子句组成：

  LIMIT { count | ALL }

  OFFSET start count 声明返回的最大行数，而 start 声明开始返回行之前忽略的行数。如果两个都指定了，会在开始计算 count 个返回行之前先跳过 start 行。

- OFFSET 子句

  SQL：2008 开始提出一种不同的语法：

OFFSET start { ROW | ROWS }

start 声明开始返回行之前忽略的行数。

- FETCH { FIRST | NEXT } [count] { ROW | ROWS } ONLY

  如果不指定 count，默认值为 1，FETCH 子句限定返回查询结果从第一行开始的总行数。

- FOR UPDATE 子句

  FOR UPDATE 子句将对 SELECT 检索出来的行进行加锁。这样避免它们在当前事务结束前被其他事务修改或者删除，即其他企图 UPDATE、 DELETE、 SELECT FOR UPDATE 这些行的事务将被阻塞，直到当前事务结束。

  为了避免操作等待其他事务提交，可使用 NOWAIT 选项，如果被选择的行不能立即被锁住，执行 SELECT FOR UPDATE NOWAIT 将会立即汇报一个错误，而不是等待。

  FOR SHARE 的行为类似，只是它在每个检索出来的行上要求一个共享锁，而不是一个排他锁。一个共享锁阻塞其它事务执行 UPDATE、DELETE、SELECT，不阻塞 SELECT FOR SHARE。

  如果在 FOR UPDATE 或 FOR SHARE 中明确指定了表名称，则只有这些指定的表被锁定，其他在 SELECT 中使用的表将不会被锁定。否则，将锁定该命令中所有使用的表。

  如果 FOR UPDATE 或 FOR SHARE 应用于一个视图或者子查询，它同样将锁定所有该视图或子查询中使用到的表。

  多个 FOR UPDATE 和 FOR SHARE 子句可以用于为不同的表指定不同的锁定模式。

  如果一个表中同时出现（或隐含同时出现）在 FOR UPDATE 和 FOR SHARE 子句中，则按照 FOR UPDATE 处理。类似的，如果影响一个表的任意子句中出现了 NOWAIT，该表将按照 NOWAIT 处理。

须知

对列存表的查询不支持 for update/share。

- NLS_SORT

  指定某字段按照特殊方式排序。目前仅支持中文拼音格式排序和不区分大小写排序。

  取值范围：

  - SCHINESE_PINYIN_M，按照中文拼音排序。如果要支持此排序方式，在创建数据库时需要指定编码格式为“GBK”，否则排序无效。
  - generic_m_ci，不区分大小写排序。

- PARTITION 子句

  查询某个分区表中相应分区的数据。

示例

```
--先通过子查询得到一张临时表 temp_t，然后查询表 temp_t 中的所有数据。
vastbase=# WITH temp_t(name,isdba) AS (SELECT usename,usesuper FROM pg_user) SELECT * FROM temp_t;

--查询 tpcds.reason 表的所有 r_reason_sk 记录，且去除重复。
vastbase=# SELECT DISTINCT(r_reason_sk) FROM tpcds.reason;

--LIMIT 子句示例：获取表中一条记录。
vastbase=# SELECT * FROM tpcds.reason LIMIT 1;

--查询所有记录，且按字母升序排列。
vastbase=# SELECT r_reason_desc FROM tpcds.reason ORDER BY r_reason_desc;

--通过表别名，从 pg_user 和 pg_user_status 这两张表中获取数据。
vastbase=# SELECT a.usename,b.locktime FROM pg_user a,pg_user_status b WHERE a.usesysid=b.roloid;

--FULL JOIN 子句示例：将 pg_user 和 pg_user_status 这两张表的数据进行全连接显示，即数据的合集。
vastbase=# SELECT a.usename,b.locktime,a.usesuper FROM pg_user a FULL JOIN pg_user_status b on
a.usesysid=b.roloid;

--GROUP BY 子句示例：根据查询条件过滤，并对结果进行分组。
vastbase=# SELECT r_reason_id, AVG(r_reason_sk) FROM tpcds.reason GROUP BY r_reason_id HAVING
AVG(r_reason_sk) > 25;

--GROUP BY CUBE 子句示例：根据查询条件过滤，并对结果进行分组汇总。
vastbase=# SELECT r_reason_id,AVG(r_reason_sk) FROM tpcds.reason GROUP BY
CUBE(r_reason_id,r_reason_sk);

--GROUP BY GROUPING SETS 子句示例:根据查询条件过滤，并对结果进行分组汇总。
vastbase=# SELECT r reason id,AVG(r reason sk) FROM tpcds.reason GROUP BY GROUPING
SETS((r reason id,r reason sk),r reason sk);

--UNION 子句示例：将表 tpcds.reason 里 r_reason_desc 字段中的内容以 W 开头和以 N 开头的进行合并。
vastbase=# SELECT r_reason_sk, tpcds.reason.r_reason_desc
   FROM tpcds.reason
   WHERE tpcds.reason.r_reason_desc LIKE 'W%'
UNION
SELECT r_reason_sk, tpcds.reason.r_reason_desc
   FROM tpcds.reason
   WHERE tpcds.reason.r_reason_desc LIKE 'N%';

--NLS_SORT 子句示例：中文拼音排序。
vastbase=# SELECT * FROM tpcds.reason ORDER BY NLSSORT( r_reason_desc, 'NLS_SORT = SCHINESE_PINYIN_M');


--不区分大小写排序：
vastbase=# SELECT * FROM tpcds.reason ORDER BY NLSSORT( r_reason_desc, 'NLS_SORT = generic_m_ci');

--创建分区表 tpcds.reason_p
vastbase=# CREATE TABLE tpcds.reason_p
(
  r_reason_sk integer,
  r_reason_id character(16),
```

```
 r_reason_desc character(100)
)
PARTITION BY RANGE (r_reason_sk)
(
 partition P_05_BEFORE values less than (05),
 partition P_15 values less than (15),
 partition P_25 values less than (25),
 partition P_35 values less than (35),
 partition P_45_AFTER values less than (MAXVALUE)
)
;

--插入数据。
vastbase=# INSERT INTO tpcds.reason_p values(3,'AAAAAAAABAAAAAAA','reason
1'),(10,'AAAAAAAABAAAAAAA','reason 2'),(4,'AAAAAAAABAAAAAAA','reason
3'),(10,'AAAAAAAABAAAAAAA','reason 4'),(10,'AAAAAAAABAAAAAAA','reason
5'),(20,'AAAAAAAACAAAAAAA','reason 6'),(30,'AAAAAAAACAAAAAAA','reason 7');

--PARTITION 子句示例：从 tpcds.reason_p 的表分区 P_05_BEFORE 中获取数据。
vastbase=#  SELECT * FROM tpcds.reason_p PARTITION (P_05_BEFORE);
 r_reason_sk |   r_reason_id    |   r_reason_desc
-------------+------------------+----------------------------------
           4 | AAAAAAAABAAAAAAA | reason 3
           3 | AAAAAAAABAAAAAAA | reason 1
(2 rows)

--GROUP BY 子句示例：按 r_reason_id 分组统计 tpcds.reason_p 表中的记录数。
vastbase=# SELECT COUNT(*),r_reason_id FROM tpcds.reason_p GROUP BY r_reason_id;
 count |   r_reason_id
-------+------------------
     2 | AAAAAAAACAAAAAAA
     5 | AAAAAAAABAAAAAAA
(2 rows)

--GROUP BY CUBE 子句示例：根据查询条件过滤，并对查询结果分组汇总。
vastbase=# SELECT * FROM tpcds.reason GROUP BY  CUBE (r_reason_id,r_reason_sk,r_reason_desc);

--GROUP BY GROUPING SETS 子句示例：根据查询条件过滤，并对查询结果分组汇总。
vastbase=# SELECT * FROM tpcds.reason GROUP BY  GROUPING SETS
((r_reason_id,r_reason_sk),r_reason_desc);

--HAVING 子句示例：按 r_reason_id 分组统计 tpcds.reason_p 表中的记录，并只显示 r_reason_id 个数大于 2 的信息。
vastbase=# SELECT COUNT(*) c,r_reason_id FROM tpcds.reason_p GROUP BY r_reason_id HAVING c>2;
 c |   r_reason_id
---+------------------
 5 | AAAAAAAABAAAAAAA
(1 row)

--IN 子句示例：按 r_reason_id 分组统计 tpcds.reason_p 表中的 r_reason_id 个数，并只显示 r_reason_id 值为
AAAAAAAABAAAAAAA 或 AAAAAAAADAAAAAAA 的个数。
vastbase=# SELECT COUNT(*),r_reason_id FROM tpcds.reason_p GROUP BY r_reason_id HAVING r_reason_id
IN('AAAAAAAABAAAAAAA','AAAAAAAADAAAAAAA');
count |   r_reason_id
-------+------------------
```

```
    5 | AAAAAAAABAAAAAAA
(1 row)

--INTERSECT 子句示例：查询 r_reason_id 等于 AAAAAAAABAAAAAAA，并且 r_reason_sk 小于 5 的信息。
vastbase=# SELECT * FROM tpcds.reason_p WHERE r_reason_id='AAAAAAAABAAAAAAA' INTERSECT SELECT * FROM
tpcds.reason_p WHERE r_reason_sk<5;
 r_reason_sk |   r_reason_id    |     r_reason_desc
-------------+------------------+-----------------------------------
         4 | AAAAAAAABAAAAAAA | reason 3
         3 | AAAAAAAABAAAAAAA | reason 1
(2 rows)

--EXCEPT 子句示例：查询 r_reason_id 等于 AAAAAAAABAAAAAAA，并且去除 r_reason_sk 小于 4 的信息。
vastbase=# SELECT * FROM tpcds.reason_p WHERE r_reason_id='AAAAAAAABAAAAAAA' EXCEPT SELECT * FROM
tpcds.reason_p WHERE r_reason_sk<4;
r_reason_sk |   r_reason_id    |     r_reason_desc
-------------+------------------+-----------------------------------
        10 | AAAAAAAABAAAAAAA | reason 2
        10 | AAAAAAAABAAAAAAA | reason 5
        10 | AAAAAAAABAAAAAAA | reason 4
         4 | AAAAAAAABAAAAAAA | reason 3
(4 rows)

--通过在 where 子句中指定"(+)"来实现左连接。
vastbase=# select t1.sr_item_sk ,t2.c_customer_id from store_returns t1, customer t2 where
t1.sr_customer_sk  = t2.c_customer_sk(+)
order by 1 desc limit 1;
 sr_item_sk | c_customer_id
------------+---------------
     18000 |
(1 row)

--通过在 where 子句中指定"(+)"来实现右连接。
vastbase=# select t1.sr_item_sk ,t2.c_customer_id from store_returns t1, customer t2 where
t1.sr_customer_sk(+)  = t2.c_customer_sk
order by 1 desc limit 1;
 sr_item_sk |  c_customer_id
------------+------------------
          | AAAAAAAAJNGEBAAA
(1 row)

--通过在 where 子句中指定"(+)"来实现左连接，并且增加连接条件。
vastbase=# select t1.sr_item_sk ,t2.c_customer_id from store_returns t1, customer t2 where
t1.sr_customer_sk  = t2.c_customer_sk(+) and t2.c_customer_sk(+) < 1 order by 1  limit 1;
 sr_item_sk | c_customer_id
------------+---------------
        1 |
(1 row)

--不支持在 where 子句中指定"(+)"的同时使用内层嵌套 AND/OR 的表达式。
vastbase=# select t1.sr_item_sk ,t2.c_customer_id from store_returns t1, customer t2 where
not(t1.sr_customer_sk  = t2.c_customer_sk(+) and t2.c_customer_sk(+) < 1);
ERROR:  Operator "(+)" can not be used in nesting expression.
LINE 1: ...tomer_id from store_returns t1, customer t2 where not(t1.sr_...
                                                             ^
```

```
--where 子句在不支持表达式宏指定"(+)"会报错。
vastbase=# select t1.sr_item_sk ,t2.c_customer_id from store_returns t1, customer t2 where
(t1.sr_customer_sk  = t2.c_customer_sk(+))::bool;
ERROR:  Operator "(+)" can only be used in common expression.

--where 子句在表达式的两边都指定"(+)"会报错。
vastbase=# select t1.sr_item_sk ,t2.c_customer_id from store_returns t1, customer t2 where
t1.sr_customer_sk(+)  = t2.c_customer_sk(+);
ERROR:  Operator "(+)" can't be specified on more than one relation in one join condition
HINT:  "t1", "t2"...are specified Operator "(+)" in one condition.

--删除表。
vastbase=# DROP TABLE tpcds.reason_p;
```

11.16.104. SELECT INTO

功能描述

SELECT INTO 用于根据查询结果创建一个新表，并且将查询到的数据插入到新表中。

数据并不返回给客户端，这一点和普通的 SELECT 不同。新表的字段具有和 SELECT 的输出字段相同的名称和数据类型。

注意事项

CREATE TABLE AS 的作用和 SELECT INTO 类似，且提供了 SELECT INTO 所提供功能的超集。建议使用 CREATE TABLE AS 语法替代 SELECT INTO，因为 SELECT INTO 不能在存储过程中使用。

语法格式

```
[ WITH [ RECURSIVE ] with_query [, ...] ]
SELECT [ ALL | DISTINCT [ ON ( expression [, ...] ) ] ]
   { * | {expression [ [ AS ] output_name ]} [, ...] }
   INTO [ UNLOGGED ] [ TABLE ] new_table
   [ FROM from_item [, ...] ]
   [ WHERE condition ]
   [ GROUP BY expression [, ...] ]
   [ HAVING condition [, ...] ]
   [ WINDOW {window_name AS ( window_definition )} [, ...] ]
   [ { UNION | INTERSECT | EXCEPT | MINUS } [ ALL | DISTINCT ] select ]
   [ ORDER BY {expression [ [ ASC | DESC | USING operator ] | nlssort_expression_clause ] [ NULLS
{ FIRST | LAST } ]} [, ...] ]
   [ LIMIT { count | ALL } ]
   [ OFFSET start [ ROW | ROWS ] ]
   [ FETCH { FIRST | NEXT } [ count ] { ROW | ROWS } ONLY ]
   [ {FOR { UPDATE | SHARE } [ OF table_name [, ...] ] [ NOWAIT ]} [...] ];
```

参数说明

INTO [UNLOGGED] [TABLE] new_table

UNLOGGED 指定表为非日志表。在非日志表中写入的数据不会被写入到预写日志中，这样就会比普通表快很多。但是，它也是不安全的，非日志表在冲突或异常关机后会被自动删截。非日志表中的内容也不会被复制到备用服务器中。在该类表中创建的索引也不会被自动记录。

new_table 指定新建表的名称。

说明

SELECT INTO 的其它参数可参考 SELECT 的参数说明。

示例

```
--将 tpcds.reason 表中 r_reason_sk 小于 5 的值加入到新建表中。
vastbase=# SELECT * INTO tpcds.reason_t1 FROM tpcds.reason WHERE r_reason_sk < 5;
INSERT 0 6

--删除 tpcds.reason_t1 表。
vastbase=# DROP TABLE tpcds.reason_t1;
```

相关链接

SELECT

优化建议

- DATABASE

  不建议在事务中 reindex database。

- SYSTEM

  不建议在事务中 reindex 系统表。

11.16.105. SET

功能描述

用于修改运行时配置参数。

注意事项

大多数运行时参数都可以用 SET 在运行时设置，但有些则在服务运行过程中或会话开始之后不能修改。

语法格式

- 设置所处的时区。

```
SET [ SESSION | LOCAL ] TIME ZONE { timezone | LOCAL | DEFAULT };
```

- 设置所属的模式。

```
SET [ SESSION | LOCAL ]
   {CURRENT_SCHEMA { TO | = } { schema | DEFAULT }
   | SCHEMA 'schema'};
```

- 设置客户端编码集。

```
SET [ SESSION | LOCAL ] NAMES encoding_name;
```

- 设置 XML 的解析方式。

```
SET [ SESSION | LOCAL ] XML OPTION { DOCUMENT | CONTENT };
```

- 设置其他运行时参数。

```
SET [ LOCAL | SESSION ]
   { {config_parameter { { TO | = } { value | DEFAULT }
                    | FROM CURRENT }}};
```

参数说明

- SESSION

  声明的参数只对当前会话起作用。如果 SESSION 和 LOCAL 都没出现，则 SESSION 为缺省值。

  如果在事务中执行了此命令，命令的产生影响将在事务回滚之后消失。如果该事务已提交，影响将持续到会话的结束，除非被另外一个 SET 命令重置参数。

- LOCAL

  声明的参数只在当前事务中有效。在 COMMIT 或 ROLLBACK 之后，会话级别的设置将再次生效。

  不论事务是否提交，此命令的影响只持续到当前事务结束。一个特例是：在一个事务里面，即有 SET 命令，又有 SET LOCAL 命令，且 SET LOCAL 在 SET 后面，则在事务结束之前，SET LOCAL 命令会起作用，但事务提交之后，则是 SET 命令会生效。

- TIME ZONE timezone

  用于指定当前会话的本地时区。

  取值范围：有效的本地时区。该选项对应的运行时参数名称为 TimeZone，DEFAULT 缺省值为 PRC。

- CURRENT_SCHEMA

  CURRENT_SCHEMA 用于指定当前的模式。

  取值范围：已存在模式名称。

- SCHEMA schema

  同 CURRENT_SCHEMA。此处的 schema 是个字符串。

  例如：set schema 'public';

- NAMES encoding_name

  用于设置客户端的字符编码。等价于 set client_encoding to encoding_name。

取值范围：有效的字符编码。该选项对应的运行时参数名称为 client_encoding，默认编码为 SQL_ASCII。

- XML OPTION option

  用于设置 XML 的解析方式。

  取值范围：CONTENT（缺省）、DOCUMENT

- config_parameter

  可设置的运行时参数的名称。可用的运行时参数可以使用 SHOW ALL 命令查看。

- value

  config_parameter 的新值。可以声明为字符串常量、标识符、数字，或者逗号分隔的列表。DEFAULT 用于把这些参数设置为它们的缺省值。

示例

```
--设置模式搜索路径。
vastbase=# SET search_path TO tpcds, public;

--把日期时间风格设置为传统的 POSTGRES 风格(日在月前)。
vastbase=# SET datestyle TO postgres;
```

相关链接

11.16.98RESET，11.16.110SHOW

11.16.106. SET CONSTRAINTS

功能描述

SET CONSTRAINTS 设置当前事务检查行为的约束条件。

IMMEDIATE 约束是在每条语句后面进行检查。DEFERRED 约束一直到事务提交时才检查。每个约束都有自己的模式。

从创建约束条件开始，一个约束总是设定为 DEFERRABLE INITIALLY DEFERRED，DEFERRABLE INITIALLY IMMEDIATE，NOT DEFERRABLE 三个特性之一。第三种总是 IMMEDIATE，并且不会受 SET CONSTRAINTS 影响。前两种以指定的方式启动每个事务，但是其行为可以在事务里用 SET CONSTRAINTS 改变。

带着一个约束名列表的 SET CONSTRAINTS 改变这些约束的模式（都必须是可推迟的）。如果有多个约束匹配某个名称，则所有都会被影响。SET CONSTRAINTS ALL 改变所有可推迟约束的模式。

当 SET CONSTRAINTS 把一个约束从 DEFERRED 改成 IMMEDIATE 的时候，新模式反作用式地起作用：任何将在事务结束准备进行的数据修改都将在 SET CONSTRAINTS 的时候执行检查。如果违反了任

何约束，SET CONSTRAINTS 都会失败（并且不会修改约束模式）。因此，SET CONSTRAINTS 可以用于强制在事务中某一点进行约束检查。

检查和唯一约束总是不可推迟的。

注意事项

SET CONSTRAINTS 只在当前事务里设置约束的行为。因此，如果用户在事务块之外（START TRANSACTION/COMMIT 对）执行这个命令，它将没有任何作用。

语法格式

```
SET CONSTRAINTS { ALL | { name } [, ...] } { DEFERRED | IMMEDIATE } ;
```

参数说明

- name

  约束名。

  取值范围：已存在的约束名。可以在系统表 pg_constraint 中查到。

- ALL

  所有约束。

- DEFERRED

  约束一直到事务提交时才检查。

- IMMEDIATE

  约束在每条语句后进行检查。

示例

```
--设置所有约束在事务提交时检查。
vastbase=# SET CONSTRAINTS ALL DEFERRED;
```

11.16.107. SET ROLE

功能描述

设置当前会话的当前用户标识符。

注意事项

- 当前会话的用户必须是指定的 rolename 角色的成员，但系统管理员可以选择任何角色。
- 使用这条命令，它可能会增加一个用户的权限，也可能会限制一个用户的权限。如果会话用户的角色有 INHERITS 属性，则它自动拥有它能 SET ROLE 变成的角色的所有权限；在这种情况下，

SET ROLE 实际上是删除了所有直接赋予会话用户的权限，以及它的所属角色的权限，只剩下指定角色的权限。另一方面，如果会话用户的角色有 NOINHERITS 属性，SET ROLE 删除直接赋予会话用户的权限，而获取指定角色的权限。

语法格式

- 设置当前会话的当前用户标识符。

```
SET [ SESSION | LOCAL ] ROLE role_name PASSWORD 'password';
```

- 重置当前用户标识为当前会话用户标识符。

```
RESET ROLE;
```

参数说明

- SESSION

  声明这个命令只对当前会话起作用，此参数为缺省值。

  取值范围：字符串，要符合标识符的命名规范。

- LOCALE

  声明该命令只在当前事务中有效。

- role_name

  角色名。

  取值范围：字符串，要符合标识符的命名规范。

- password

  角色的密码。要求符合密码的命名规则。

- RESET ROLE

  用于重置当前用户标识。

示例

```
--创建角色 paul。
vastbase=# CREATE ROLE paul IDENTIFIED BY 'Bigdata@123';

--设置当前用户为 paul。
vastbase=# SET ROLE paul PASSWORD 'Bigdata@123';

--查看当前会话用户，当前用户。
vastbase=# SELECT SESSION_USER, CURRENT_USER;

--重置当前用户。
vastbase=# RESET role;

--删除用户。
vastbase=# DROP USER paul;
```

11.16.108. SET SESSION AUTHORIZATION

功能描述

把当前会话里的会话用户标识和当前用户标识都设置为指定的用户。

注意事项

只有在初始会话用户有系统管理员权限的时候，会话用户标识符才能改变。否则，只有在指定了被认证的用户名的情况下，系统才接受该命令。

语法格式

- 为当前会话设置会话用户标识符和当前用户标识符。

```
SET [ SESSION | LOCAL ] SESSION AUTHORIZATION role_name PASSWORD 'password';
```

- 重置会话和当前用户标识符为初始认证的用户名。

```
{SET [ SESSION | LOCAL ] SESSION AUTHORIZATION DEFAULT
    | RESET SESSION AUTHORIZATION};
```

参数说明

- SESSION

  声明这个命令只对当前会话起作用。

  取值范围：字符串，要符合标识符的命名规范。

- LOCALE

  声明该命令只在当前事务中有效。

- role_name

  用户名。

  取值范围：字符串，要符合标识符的命名规范。

- password

  角色的密码。要求符合密码的命名规则。

- DEFAULT

  重置会话和当前用户标识符为初始认证的用户名。

示例

```
--创建角色 paul。
vastbase=# CREATE ROLE paul IDENTIFIED BY 'Bigdata@123';

--设置当前用户为 paul。
vastbase=# SET SESSION AUTHORIZATION paul password 'Bigdata@123';
```

```
--查看当前会话用户，当前用户。
vastbase=# SELECT SESSION_USER, CURRENT_USER;

--重置当前用户。
vastbase=# RESET SESSION AUTHORIZATION;

--删除用户。
vastbase=# DROP USER paul;
```

相关参考

11.16.107SET ROLE

11.16.109. SET TRANSACTION

功能描述

为当前事务设置特性。它对后面的事务没有影响。事务特性包括事务隔离级别、事务访问模式(读/写或者只读)。

注意事项

无。

语法格式

设置事务的隔离级别、读写模式。

```
{ SET [ LOCAL ] TRANSACTION|SET SESSION CHARACTERISTICS AS TRANSACTION }
 { ISOLATION LEVEL { READ COMMITTED | SERIALIZABLE | REPEATABLE READ }
 | { READ WRITE | READ ONLY } } [, ...]
```

参数说明

- LOCAL

  声明该命令只在当前事务中有效。

- SESSION

  声明这个命令只对当前会话起作用。

  取值范围：字符串，要符合标识符的命名规范。

- ISOLATION_LEVEL

  指定事务隔离级别，该参数决定当一个事务中存在其他并发运行事务时能够看到什么数据。

说明

在事务中第一个数据修改语句（SELECT，INSERT，DELETE，UPDATE，FETCH，COPY）执行之后，事务隔离级别就不能再次设置。

取值范围：

- READ COMMITTED：读已提交隔离级别，只能读到已经提交的数据，而不会读到未提交的数据。这是缺省值。
- REPEATABLE READ：可重复读隔离级别，仅仅能看到事务开始之前提交的数据，不能看到未提交的数据，以及在事务执行期间由其它并发事务提交的修改。
- SERIALIZABLE：Vastbase 目前功能上不支持此隔离级别，等价于 REPEATABLE READ。

❖ READ WRITE | READ ONLY

指定事务访问模式（读/写或者只读）。

示例

```
--开启一个事务，设置事务的隔离级别为 READ COMMITTED，访问模式为 READ ONLY。
vastbase=# START TRANSACTION;
vastbase=# SET LOCAL TRANSACTION ISOLATION LEVEL READ COMMITTED READ ONLY;
vastbase=# COMMIT;
```

11.16.110. SHOW

功能描述

SHOW 将显示当前运行时参数的数值。

注意事项

无。

语法格式

```
SHOW
 {
   configuration_parameter |
   CURRENT_SCHEMA |
   TIME ZONE |
   TRANSACTION ISOLATION LEVEL |
   SESSION AUTHORIZATION |
   ALL
 };
```

参数说明

显示变量的参数请参见 RESET 的参数说明。

示例

```
--显示 timezone 参数值。
vastbase=# SHOW timezone;

--显示所有参数。
vastbase=# SHOW ALL;
```

相关链接

11.16.105SET，11.16.98RESET

11.16.111. START TRANSACTION

功能描述

通过 START TRANSACTION 启动事务。如果声明了隔离级别、读写模式，那么新事务就使用这些特性，类似执行了 11.16.109SET TRANSACTION。

注意事项

无。

语法格式

格式一：START TRANSACTION 格式

```
START TRANSACTION
  [
    {
      ISOLATION LEVEL { READ COMMITTED | SERIALIZABLE | REPEATABLE READ }
      | { READ WRITE | READ ONLY }
     } [, ...]
  ];
```

格式二：BEGIN 格式

```
BEGIN [ WORK | TRANSACTION ]
  [
    {
      ISOLATION LEVEL { READ COMMITTED | SERIALIZABLE | REPEATABLE READ }
      | { READ WRITE | READ ONLY }
     } [, ...]
  ];
```

参数说明

- WORK | TRANSACTION

  BEGIN 格式中的可选关键字，没有实际作用。

- ISOLATION LEVEL

指定事务隔离级别，它决定当一个事务中存在其他并发运行事务时它能够看到什么数据。

说明

在事务中第一个数据修改语句（SELECT, INSERT，DELETE，UPDATE，FETCH，COPY）执行之后，事务隔离级别就不能再次设置。

取值范围：

- READ COMMITTED：读已提交隔离级别，只能读到已经提交的数据，而不会读到未提交的数据。这是缺省值。
- REPEATABLE READ： 可重复读隔离级别，仅仅看到事务开始之前提交的数据，它不能看到未提交的数据，以及在事务执行期间由其它并发事务提交的修改。
- SERIALIZABLE：Vastbase 目前功能上不支持此隔离级别，等价于 REPEATABLE READ。

❖ READ WRITE | READ ONLY

指定事务访问模式（读/写或者只读）。

示例

```
--以默认方式启动事务。
vastbase=# START TRANSACTION;
vastbase=# SELECT * FROM tpcds.reason;
vastbase=# END;

--以默认方式启动事务。
vastbase=# BEGIN;
vastbase=# SELECT * FROM tpcds.reason;
vastbase=# END;

--以隔离级别为 READ COMMITTED，读/写方式启动事务。
vastbase=# START TRANSACTION ISOLATION LEVEL READ COMMITTED READ WRITE;
vastbase=# SELECT * FROM tpcds.reason;
vastbase=# COMMIT;
```

相关链接

11.16.33COMMIT | END，11.16.100ROLLBACK，11.16.109SET TRANSACTION

11.16.112. TRUNCATE

功能描述

清理表数据，TRUNCATE 快速地从表中删除所有行。

它和在目标表上进行无条件的 DELETE 有同样的效果，但由于 TRUNCATE 不做表扫描，因而快得多。在大表上操作效果更明显。

注意事项

- TRUNCATE TABLE 在功能上与不带 WHERE 子句 DELETE 语句相同：二者均删除表中的全部行。
- TRUNCATE TABLE 比 DELETE 速度快且使用系统和事务日志资源少：
  - DELETE 语句每次删除一行，并在事务日志中为所删除每行记录一项。
  - TRUNCATE TABLE 通过释放存储表数据所用数据页来删除数据，并且只在事务日志中记录页的释放。
- TRUNCATE，DELETE，DROP 三者的差异如下：
  - TRUNCATE TABLE，删除内容，释放空间，但不删除定义。
  - DELETE TABLE，删除内容，不删除定义，不释放空间。
  - DROP TABLE，删除内容和定义，释放空间。

语法格式

- 清理表数据。

```
TRUNCATE [ TABLE ] [ ONLY ] {table_name [ * ]} [, ... ]
   [ CONTINUE IDENTITY ] [ CASCADE | RESTRICT ];
```

- 清理表分区的数据。

```
ALTER TABLE [ IF EXISTS  ] { [ ONLY  ] table_name
                     | table_name *
                     | ONLY ( table_name )  }
   TRUNCATE PARTITION { partition_name
                 | FOR (  partition_value  [, ...] )  } ;
```

参数说明

- ONLY

  如果声明 ONLY，只有指定的表会被清空。如果没有声明 ONLY，这个表以及其所有子表（若有）会被清空。

- table_name

  目标表的名称（可以有模式修饰）。

  取值范围：已存在的表名。

- CONTINUE IDENTITY

  不改变序列的值。这是缺省值。

- CASCADE | RESTRICT
  - CASCADE：级联清空所有由于 CASCADE 而被添加到组中的表。
  - RESTRICT（缺省值）：完全清空。
- partition_name

  目标分区表的分区名。

取值范围：已存在的分区名。

- partition_value

  指定的分区键值。

  通过 PARTITION FOR 子句指定的这一组值，可以唯一确定一个分区。

  取值范围：需要进行删除数据分区的分区键的取值范围。

须知

使用 PARTITION FOR 子句时，partition_value 所在的整个分区会被清空。

示例

```
--创建表。
vastbase=# CREATE TABLE tpcds.reason_t1 AS TABLE tpcds.reason;

--清空表 tpcds.reason_t1。
vastbase=# TRUNCATE TABLE tpcds.reason_t1;

--删除表。
vastbase=# DROP TABLE tpcds.reason_t1;
--创建分区表。
vastbase=# CREATE TABLE tpcds.reason_p
(
  r_reason_sk integer,
  r_reason_id character(16),
  r_reason_desc character(100)
)PARTITION BY RANGE (r_reason_sk)
(
  partition p_05_before values less than (05),
  partition p_15 values less than (15),
  partition p_25 values less than (25),
  partition p_35 values less than (35),
  partition p_45_after values less than (MAXVALUE)
);

--插入数据。
vastbase=# INSERT INTO tpcds.reason_p SELECT * FROM tpcds.reason;

--清空分区 p_05_before。
vastbase=# ALTER TABLE tpcds.reason_p TRUNCATE PARTITION p_05_before;

--清空分区 p_15。
vastbase=# ALTER TABLE tpcds.reason_p TRUNCATE PARTITION for (13);

--清空分区表。
vastbase=# TRUNCATE TABLE tpcds.reason_p;
```

```
--删除表。
vastbase=# DROP TABLE tpcds.reason_p;
```

11.16.113. UPDATE

功能描述

更新表中的数据。UPDATE 修改满足条件的所有行中指定的字段值，WHERE 子句声明条件，SET 子句指定的字段会被修改，没有出现的字段则保持它们的原值。

注意事项

- 要修改表，用户必须对该表有 UPDATE 权限。
- 对 expression 或 condition 条件里涉及到的任何表要有 SELECT 权限。
- 对于列存表，暂时不支持 RETURNING 子句。
- 列存表不支持结果不确定的更新(non-deterministic update)。试图对列存表用多行数据更新一行时会报错。
- 列存表的更新操作，旧记录空间不会回收，需要执行 VACUUM FULL table_name 进行清理。
- 对于列存复制表，暂不支持 UPDATE 操作。

语法格式

```
UPDATE [ ONLY ] table_name [ * ] [ [ AS ] alias ]
SET {column_name = { expression | DEFAULT }
   |( column_name [, ...] ) = {( { expression | DEFAULT } [, ...] ) |sub_query }}[, ...]
   [ FROM from_list] [ WHERE condition ]
   [ RETURNING {*
              | {output_expression [ [ AS ] output_name ]} [, ...] }];

where sub_query can be:
SELECT [ ALL | DISTINCT [ ON ( expression [, ...] ) ] ]
{ * | {expression [ [ AS ] output_name ]} [, ...] }
[ FROM from_item [, ...] ]
[ WHERE condition ]
[ GROUP BY grouping_element [, ...] ]
[ HAVING condition [, ...] ]
```

参数说明

- table_name

  要更新的表名，可以使用模式修饰。

  取值范围：已存在的表名称。

- alias

  目标表的别名。

取值范围：字符串，符合标识符命名规范。

- column_name

  要修改的字段名。

  支持使用目标表的别名加字段名来引用这个字段。例如：

  UPDATE foo AS f SET f.col_name = 'vastbase';

  取值范围：已存在的字段名。

- expression

  赋给字段的值或表达式。

- DEFAULT

  用对应字段的缺省值填充该字段。

  如果没有缺省值，则为 NULL。

- sub_query

  子查询。

  使用同一数据库里其他表的信息来更新一个表可以使用子查询的方法。其中 SELECT 子句具体介绍请参考 SELECT。

- from_list

  一个表的表达式列表，允许在 WHERE 条件里使用其他表的字段。与在一个 SELECT 语句的 FROM 子句里声明表列表类似。

须知

目标表绝对不能出现在 from_list 里，除非在使用一个自连接（此时它必须以 from_list 的别名出现）。

- condition

  一个返回 Boolean 类型结果的表达式。只有这个表达式返回 true 的行才会被更新。

- output_expression

  在所有需要更新的行都被更新之后，UPDATE 命令用于计算返回值的表达式。

  取值范围：使用任何 table 以及 FROM 中列出的表的字段。*表示返回所有字段。

- output_name

  字段的返回名称。

示例

```
--创建表 student1。
vastbase=# CREATE TABLE student1
(
  stuno    int,
  classno   int
)

--插入数据。
vastbase=# INSERT INTO student1 VALUES(1,1);
vastbase=# INSERT INTO student1 VALUES(2,2);
vastbase=# INSERT INTO student1 VALUES(3,3);

--查看数据。
vastbase=# SELECT * FROM student1;

--直接更新所有记录的值。
vastbase=# UPDATE student1 SET classno = classno*2;

--查看数据。
vastbase=# SELECT * FROM student1;

--删除表。
vastbase=# DROP TABLE student1;
```

11.16.114. VACUUM

功能描述

VACUUM 回收表或 B-Tree 索引中已经删除的行所占据的存储空间。在一般的数据库操作里，那些已经 DELETE 的行并没有从它们所属的表中物理删除；在完成 VACUUM 之前它们仍然存在。因此有必要周期地运行 VACUUM，特别是在经常更新的表上。

注意事项

- 如果没有参数，VACUUM 处理当前数据库里用户拥有相应权限的每个表。如果参数指定了一个表，VACUUM 只处理指定的那个表。
- 要对一个表进行 VACUUM 操作，通常用户必须是表的所有者或系统管理员。数据库的所有者允许对数据库中除了共享目录以外的所有表进行 VACUUM 操作（该限制意味着只有系统管理员才能真正对一个数据库进行 VACUUM 操作）。VACUUM 命令会跳过那些用户没有权限的表进行垃圾回收操作。
- VACUUM 不能在事务块内执行。
- 建议生产数据库经常清理（至少每晚一次），以保证不断地删除失效的行。尤其是在增删了大量记录之后，对受影响的表执行 VACUUM ANALYZE 命令是一个很好的习惯。这样将更新系统目录为最近的更改，并且允许查询优化器在规划用户查询时有更好的选择。

❖ 不建议日常使用 FULL 选项，但是可以在特殊情况下使用。例如在用户删除了一个表的大部分行之后，希望从物理上缩小该表以减少磁盘空间占用。VACUUM FULL 通常要比单纯的 VACUUM 收缩更多的表尺寸。FULL 选项并不清理索引，所以推荐周期性的运行 11.16.96REINDEX 命令。实际上，首先删除所有索引，再运行 VACUUM FULL 命令，最后重建索引通常是更快的选择。如果执行此命令后所占用物理空间无变化（未减少），请确认是否有其他活跃事务（删除数据事务开始之前开始的事务，并在 VACUUM FULL 执行前未结束）存在，如果有等其他活跃事务退出进行重试。

❖ VACUUM 会导致 I/O 流量的大幅增加，这可能会影响其他活动会话的性能。因此，有时候会建议使用基于开销的 VACUUM 延迟特性。

❖ 如果指定了 VERBOSE 选项，VACUUM 将打印处理过程中的信息，以表明当前正在处理的表。各种有关当前表的统计信息也会打印出来。但是对于列存表执行 VACUUM 操作，指定了 VERBOSE 选项，无信息输出。

❖ 当含有带括号的选项列表时，选项可以以任何顺序写入。如果没有括号，则选项必须按语法显示的顺序给出。

❖ VACUUM 和 VACUUM FULL 时，会根据参数 vacuum_defer_cleanup_age 延迟清理行存表记录，即不会立即清理刚刚删除的元组。

❖ VACUUM ANALYZE 先执行一个 VACUUM 操作，然后给每个选定的表执行一个 ANALYZE。对于日常维护脚本而言，这是一个很方便的组合。

❖ 简单的 VACUUM（不带 FULL 选项）只是简单地回收空间并且令其可以再次使用。这种形式的命令可以和对表的普通读写并发操作，因为没有请求排他锁。VACUUM FULL 执行更广泛的处理，包括跨块移动行，以便把表压缩到最少的磁盘块数目里。这种形式要慢许多并且在处理的时候需要在表上施加一个排他锁。

❖ VACUUM 列存表内部执行的操作包括三个：迁移 delta 表中的数据到主表、VACUUM 主表的 delta 表、VACUUM 主表的 desc 表。该操作不会回收 delta 表的存储空间，如果要回收 delta 表的冗余存储空间，需要对该列存表执行 VACUUM DELTAMERGE。

❖ 同时执行多个 VACUUM FULL 可能出现死锁。

语法格式

❖ 回收空间并更新统计信息，对关键字顺序无要求。

```
VACUUM [ ( { FULL | FREEZE | VERBOSE | {ANALYZE | ANALYSE }} [,...] ) ]
    [ table_name [ (column_name [, ...] ) ] ] [ PARTITION ( partition_name ) ];
```

❖ 仅回收空间，不更新统计信息。

```
VACUUM [ FULL [COMPACT] ] [ FREEZE ] [ VERBOSE ] [ table_name ] [ PARTITION ( partition_name ) ];
```

❖ 回收空间并更新统计信息，且对关键字顺序有要求。

```
VACUUM [ FULL ] [ FREEZE ] [ VERBOSE ] { ANALYZE | ANALYSE } [ VERBOSE ]
    [ table_name [ (column_name [, ...] ) ] ] [ PARTITION ( partition_name ) ];
```

参数说明

- FULL

  选择“FULL”清理，这样可以恢复更多的空间，但是需要耗时更多，并且在表上施加了排他锁。

说明

使用 FULL 参数会导致统计信息丢失，如果需要收集统计信息，请在 VACUUM FULL 语句中加上 analyze 关键字。

- FREEZE

  指定 FREEZE 相当于执行 VACUUM 时将 vacuum_freeze_min_age 参数设为 0。

- VERBOSE

  为每个表打印一份详细的清理工作报告。

- ANALYZE | ANALYSE

  更新用于优化器的统计信息，以决定执行查询的最有效方法。

- table_name

  要清理的表的名称（可以有模式修饰）。

  取值范围：要清理的表的名称。缺省时为当前数据库中的所有表。

- column_name

  要分析的具体的字段名称。

  取值范围：要分析的具体的字段名称。缺省时为所有字段。

- PARTITION

  COMPACT 和 PARTITION 参数不能同时使用。

- partition_name

  要清理的表的分区名称。缺省时为所有分区。

- DELTAMERGE

  只针对列存表，将列存表的 delta table 中的数据转移到主表存储上。对列存表而言，此操作受 enable_delta_store 和参数说明中的 deltarow_threshold 控制。

说明

为了检查列存 delta 表中的信息，提供下述 DFX 函数，用于获取某个列存表的 delta 表中数据存储情况：

- pgxc_get_delta_info(TEXT)，传入参数为列存表名，搜集并显示各个节点上的对应 delta 表信息，包括当前存活 tuple 数量、表大小、使用的最大 block ID。
- get_delta_info(TEXT)，传入参数为列存表名，汇总 pgxc_get_delta_info 得到的结果，返回其 delta 表整体的当前存活 tuple 数量、表大小、使用的最大 block ID。

示例

```
--在表 tpcds.reason 上创建索引
CREATE UNIQUE INDEX ds_reason_index1 ON tpcds.reason(r_reason_sk);

--对带索引的表 tpcds.reason 执行 VACUUM 操作。
vastbase=# VACUUM (VERBOSE, ANALYZE) tpcds.reason;

--删除索引
vastbase=# DROP INDEX ds_reason_index1 CASCADE;
vastbase=# DROP TABLE tpcds.reason;
```

优化建议

- vacuum
  - VACUUM 不能在事务块内执行。
  - 建议生产数据库经常清理（至少每晚一次），以保证不断地删除失效的行。尤其是在增删了大量记录后，对相关表执行 VACUUM ANALYZE 命令。
  - 不建议日常使用 FULL 选项，但是可以在特殊情况下使用。例如，一个例子就是在用户删除了一个表的大部分行之后，希望从物理上缩小该表以减少磁盘空间占用。
  - 执行 VACUUM FULL 操作时，建议首先删除相关表上的所有索引，再运行 VACUUM FULL 命令，最后重建索引。

11.16.115. VALUES

功能描述

根据给定的值表达式计算一个或一组行的值。它通常用于在一个较大的命令内生成一个“常数表”。

注意事项

- 应当避免使用 VALUES 返回数量非常大的结果行，否则可能会遭遇内存耗尽或者性能低下。出现在 INSERT 中的 VALUES 是一个特殊情况，因为目标字段类型可以从 INSERT 的目标表获知，并不需要通过扫描 VALUES 列表来推测，所以在此情况下可以处理非常大的结果行。
- 如果指定了多行，那么每一行都必须拥有相同的元素个数。

语法格式

```
VALUES {( expression [, ...] )} [, ...]
    [ ORDER BY { sort_expression [ ASC | DESC | USING operator ] } [, ...] ]
    [ LIMIT { count | ALL } ]
    [ OFFSET start [ ROW | ROWS ] ]
    [ FETCH { FIRST | NEXT } [ count ] { ROW | ROWS } ONLY ];
```

参数说明

- expression

  用于计算或插入结果表指定地点的常量或者表达式。

  在一个出现在 INSERT 顶层的 VALUES 列表中，expression 可以被 DEFAULT 替换以表示插入目的字段的缺省值。除此以外，当 VALUES 出现在其他场合的时候是不能使用 DEFAULT 的。

- sort_expression

  一个表示如何排序结果行的表达式或者整数常量。

- ASC

  指定按照升序排列。

- DESC

  指定按照降序排列。

- operator

  一个排序操作符。

- count

  返回的最大行数。

- OFFSET start { ROW | ROWS }

  声明返回的最大行数，而 start 声明开始返回行之前忽略的行数。

- FETCH { FIRST | NEXT } [count] { ROW | ROWS } ONLY

  FETCH 子句限定返回查询结果从第一行开始的总行数，count 的缺省值为 1。

示例

请参见 INSERT 的示例。

11.17. Oracle 语法兼容

11.17.1. 兼容 Insert All&First 子句语法

功能描述

Insert All&First 语句允许用户在一条 Insert 语句中往不同的表同时插入数据，也可以实现有条件的插入。Insert All 表示所有满足条件的插入语句都会被执行。Insert First 在执行第一个满足条件的插入语句后就会结束。

语法格式

```
Insert [ First | All ]
[ WHEN { Condition } THEN INTO table [ [ AS ] alias ] Values (v1, ...), ...]
{ Subquery }
```

参数说明

- when {Condition}

  就是一个任意判断表达式，判断表达式中可以使用常量或引用查询中的表字段

示例

```
vastbase=# CREATE TABLE small_orders
vastbase-#    (order_id      NUMERIC   NOT NULL,
vastbase(#     customer_id    NUMERIC    NOT NULL,
vastbase(#     order_total    NUMERIC
vastbase(#    );
CREATE TABLE
vastbase=# CREATE TABLE medium_orders AS SELECT * FROM small_orders;
INSERT 0 0
vastbase=# CREATE temp TABLE large_orders AS SELECT * FROM small_orders;
INSERT 0 0
vastbase=# CREATE TABLE special_orders
vastbase-#    (order_id      NUMERIC  NOT NULL,
vastbase(#     customer_id    NUMERIC  NOT NULL,
vastbase(#     order_total    NUMERIC,
vastbase(#     other         NUMERIC
vastbase(#    );
CREATE TABLE
vastbase=# CREATE TABLE orders
vastbase-#    (order_id      NUMERIC NOT NULL,
vastbase(#     customer_id    NUMERIC NOT NULL,
vastbase(#     order_total    NUMERIC ,
vastbase(#     other         NUMERIC
vastbase(#    );
CREATE TABLE
vastbase=# INSERT INTO orders SELECT 1, 1, 10000, 1 ;
INSERT 0 1
vastbase=# INSERT INTO orders SELECT 1, 1, 20000, 1 ;
INSERT 0 1
vastbase=# INSERT INTO orders SELECT 1, 1, 30000, 1 ;
INSERT 0 1
vastbase=# INSERT INTO orders SELECT 1, 1, 210000, 1 ;
INSERT 0 1
vastbase=# INSERT INTO orders SELECT 1, 1, 220000, 1 ;
INSERT 0 1
vastbase=# INSERT INTO orders SELECT 1, 1, 110000, 1 ;
INSERT 0 1
vastbase=# INSERT INTO orders SELECT 1, 1, 120000, 1 ;
INSERT 0 1
vastbase=# INSERT INTO orders SELECT 1, 1, 130000, 1 ;
INSERT 0 1
vastbase=# INSERT INTO orders SELECT 1, 1, 140000, 1 ;
```

```
INSERT 0 1
vastbase=# INSERT INTO orders SELECT 1, 1, 310000, 1 ;
INSERT 0 1
vastbase=# INSERT INTO orders SELECT 1, 1, 340000, 1 ;
INSERT 0 1
vastbase=# INSERT ALL
vastbase-#   WHEN ottl < 100000 THEN
vastbase-#     INTO small_orders
vastbase-#       VALUES(oid, ottl, cid)
vastbase-#   WHEN ottl > 100000 and ottl < 200000 THEN
vastbase-#     INTO medium_orders
vastbase-#       VALUES(oid, ottl, cid)
vastbase-#   WHEN ottl > 200000 THEN
vastbase-#     into large_orders
vastbase-#       VALUES(oid, ottl, cid)
vastbase-#   -- WHEN ottl > 290000 THEN
vastbase-#   --   INTO special_orders
vastbase-#   SELECT o.order_id oid, o.customer_id cid, o.order_total ottl,
vastbase-#     o.other
vastbase-#     FROM orders o;

vastbase=# INSERT First
Vastbase-#   INTO small_orders
Vastbase-#     VALUES(oid, ottl, cid)
Vastbase-#   WHEN ottl < 100000 THEN
Vastbase-#     INTO small_orders
Vastbase-#       VALUES(oid, ottl, cid)
Vastbase-#   WHEN ottl > 100000 and ottl < 200000 THEN
Vastbase-#     INTO medium_orders
Vastbase-#       VALUES(oid, ottl, cid)
Vastbase-#   WHEN ottl > 200000 THEN
Vastbase-#     into large_orders
Vastbase-#       VALUES(oid, ottl, cid)
Vastbase-#   SELECT o.order_id oid, o.customer_id cid, o.order_total ottl,
Vastbase-#     o.other
Vastbase-#     FROM orders o;
vastbase=# select * from medium_orders;
 order_id | customer_id | order_total
----------+-------------+-------------
        1 |      110000 |           1
        1 |      120000 |           1
        1 |      130000 |           1
        1 |      140000 |           1
(4 rows)

vastbase=# select * from small_orders;
 order_id | customer_id | order_total
----------+-------------+-------------
        1 |       10000 |           1
        1 |       20000 |           1
        1 |       30000 |           1
(3 rows)

vastbase=# select * from large_orders;
 order_id | customer_id | order_total
```

```
----------+-------------+-------------
      1 |     210000 |         1
      1 |     220000 |         1
      1 |     310000 |         1
      1 |     340000 |         1
(4 rows)

vastbase=# select * from special_orders;
 order_id | customer_id | order_total | other
----------+-------------+-------------+-------
(0 rows)
```

11.17.2.兼容 Merge Into 子句语法

功能描述

Merge Into 功能实现基于源表对目标表有条件的进行 Update, Delete, Insert 语句。

语法格式

```
MERGE INTO [ schema. ] { table | view } [ t_alias ]
  USING { [ schema. ] { table | view } | subquery } [ t_alias ]
  ON ( condition )
  [ merge_update_clause ]
  [ merge_insert_clause ];
```

参数说明

- subquery

  子查询。

- merge_update_clause

  WHEN MATCHED THEN UPDATE SET 子句。

- merge_insert_clause

  WHEN NOT MATCHED THEN INSERT 子句。

示例

```
vastbase=# create table A_MERGE
vastbase-# (
vastbase(# id NUMBER not null,
vastbase(# name VARCHAR2(12) not null,
vastbase(# year NUMBER
vastbase(# );
CREATE TABLE
vastbase=# create table B_MERGE
```

```
vastbase-# (
vastbase(# id NUMBER not null,
vastbase(# aid NUMBER not null,
vastbase(# name VARCHAR2(12) not null,
vastbase(# year NUMBER,
vastbase(# city VARCHAR2(12)
vastbase(# );
CREATE TABLE
vastbase=# insert into A_MERGE values(1,'liuwei',20);
INSERT 0 1
vastbase=# insert into A_MERGE values(2,'zhangbin',21);
INSERT 0 1
vastbase=# insert into A_MERGE values(3,'fuguo',20);
INSERT 0 1
vastbase=# insert into B_MERGE values(1,2,'zhangbin',30,'');
INSERT 0 1
vastbase=# insert into B_MERGE values(2,4,'yihe',33,'');
INSERT 0 1
vastbase=# insert into B_MERGE values(3,3,'fuguo','','山');
INSERT 0 1
vastbase=# MERGE INTO A_MERGE A USING (select B.AID,B.NAME,B.YEAR from B_MERGE B) C ON (A.id=C.AID)
vastbase-# WHEN MATCHED THEN
vastbase-# UPDATE SET A.YEAR=C.YEAR
vastbase-# WHEN NOT MATCHED THEN
vastbase-# INSERT(A.ID,A.NAME,A.YEAR) VALUES(C.AID,C.NAME,C.YEAR);
MERGE 3
vastbase=#
vastbase=# select * from A_MERGE;
 id |   name   | year
----+----------+------
  1 | liuwei   |   20
  2 | zhangbin |   30
  3 | fuguo    |
  4 | yihe     |   33
(4 rows)
vastbase=# create table A_MERGE1
vastbase-# (
vastbase(# id NUMBER not null,
vastbase(# name VARCHAR2(12) not null,
vastbase(# year NUMBER
vastbase(# );
CREATE TABLE
vastbase=# create table B_MERGE1
vastbase-# (
vastbase(# id NUMBER not null,
vastbase(# aid NUMBER not null,
vastbase(# name VARCHAR2(12) not null,
vastbase(# year NUMBER,
vastbase(# city VARCHAR2(12)
vastbase(# );
CREATE TABLE
vastbase=#
vastbase=# insert into A_MERGE values(1,'liuwei',20);
INSERT 0 1
vastbase=# insert into A_MERGE values(2,'zhangbin',21);
```

```
INSERT 0 1
vastbase=# insert into A_MERGE values(3,'fuguo',20);
INSERT 0 1
vastbase=# insert into B_MERGE values(1,2,'zhangbin',30,'');
INSERT 0 1
vastbase=# insert into B_MERGE values(2,4,'yihe',33,'');
INSERT 0 1
vastbase=# insert into B_MERGE values(3,3,'fuguo','','山');
INSERT 0 1
vastbase=#
vastbase=# MERGE INTO A_MERGE1 A USING (select B.AID,B.NAME,B.YEAR from B_MERGE B) C ON (A.id=C.AID)
vastbase-# WHEN MATCHED THEN
vastbase-# UPDATE SET A.YEAR=C.YEAR
vastbase-# WHEN NOT MATCHED THEN
vastbase-# INSERT(A.ID,A.NAME,A.YEAR) VALUES(C.AID,C.NAME,C.YEAR);
MERGE 6
```

11.17.3.兼容 with function 子句语法

功能描述

With Function 功能实现在 WITH 语句中定义临时函数，在后续的子语句中可以重复使用该函数。

语法格式

```
WITH { FUNCTION func name ( [ [ argmode ] [ argname ] argtype [ { DEFAULT | = } default expr ] [, ...] ] )
    [ RETURNS rettype
      | RETURNS TABLE ( column name column type [, ...] ) ] AS 'definition' } [, …]
{ SelectStmt }
```

参数说明

- 参考 create function 的语法

示例

```
vastbase=# WITH FUNCTION withfunc(x INTEGER) RETURNS INTEGER AS $$
vastbase$# BEGIN
vastbase$#     RETURN x+1;
vastbase$# END
vastbase$# $$,
vastbase-# FUNCTION withfunc2(x INTEGER, y INTEGER) RETURNS INTEGER AS $$
vastbase$# BEGIN
vastbase$#     RETURN x+y;
vastbase$# END;
vastbase$# $$,
vastbase-# FUNCTION withfunc3(x TEXT) RETURNS TEXT AS $$
vastbase$# BEGIN
```

```
vastbase$#     RETURN x || '-test';
vastbase$# END;
vastbase$# $$
vastbase-# SELECT withfunc(1),withfunc2(2,3),withfunc3('4');
 withfunc | withfunc2 | withfunc3
----------+-----------+-----------
        2 |         5 | 4-test
(1 row)
```

11.17.4.兼容 sample 抽样采集语法

功能描述

sample 子句允许从表中的随机数据样本中进行选择，而不是从整个表中进行选择。

语法格式

```
TABLESAMPLE sampling_method ( argument [, ...] ) [ REPEATABLE ( seed ) ];
```

参数说明

- sampling_method

  采样方法

  - BERNOULLI：抽样方式为随机抽取表的数据行数据，抽样级别为数据行级别。使用全表扫描的采样方法，按采样参数百分比返回。比 system 具有更好的随机性，但是性能要差很多。
  - SYSTEM：使用块级采样方法，随机抽取表的数据块上的数据，按采样参数百分比返回（被采样到的数据块内的所有记录都将被返回）。因此离散度不如 BERNOULLI，但是效率高很多。
  - SYSTEM_ROWS：tsm_system_rows 模块提供了表采样方法 SYSTEM_ROWS，它可以用在 SELECT 命令的 TABLESAMPLE 子句中。这种表采样方法接受一个整数参数，它是要读取的最大行数。得到的采样将总是包 含正好这么多行，除非该表中没有足够的行，在那种情况下整个表都会被选择出来。和内建的 SYSTEM 采样方法一样， SYSTEM_ROWS 执行块级别的采样，所以采样不是完全随机的， 而是服从于聚簇效果，特别是只要求少量行时。SYSTEM_ROWS 不支持 REPEATABLE 子句。

- argument

  要采样的表的分数，表示为一个 0 到 100 之间的百分数。

  取这个参数可以是任意的实数值表达式（除了 BERNOULLI 以及 SYSTEM 采样，其他的采样方法可能接受更多或者不同的参数）

- REPEATABLE(seed)

采样随机种子，如果种子一样，那么多次采样请求得到的结果是一样的。如果忽略 REPEATABLE 则每次都是使用新的 seed 值，得到不同的结果。seed 取值为任何非空浮点值。

示例

```
vastbase=#  create table test_sample (id int4,message text);
CREATE TABLE
vastbase=#   insert into test_sample(id,message) select n,md5(random()::text) from
generate_series(1,1000000) n;
INSERT 0 1000000
vastbase=# select * from test_sample tablesample system(0.1);
  id   |          message
-------+----------------------------------
 23325 | e6ed799d1e646ed54924bad9650a10cb
 23326 | 0720940e7a95525794c2a03854fa4c47
 23327 | 121d8d190bd6d9edc06e09903c59e67b
 23328 | 3b1d342a4a2c9fd092c7f7bcdbfbafec
 23329 | 6d435ec5f3da453dbcd3bbd9776f7490
 23330 | 00278ec96f0a29729dcac0c8c6153f20
 23331 | 6a44ffb66beaceb5b4664ba930139209
 23332 | ed8e7e3bbea75609a94e7da72ef43774
 23333 | 0398893fbc3fc2c01ac6d6757c0623de
 23334 | 710b351b5a80bcb06c033c33e89ea39c
 23335 | 9b80997923a919c6d76b9ef390e5b2ce
 23336 | f13fa551eec03efa3c570493bc180663
 23337 | 7794ad832701cacced2276ea53cc482c
 23338 | 2954b1035fbf737aec7dfadd2e8370c6
 23339 | 9e681fbdc63f98a493b6205e19e558cd
 23340 | 7f94ccb23055c1fa2088301607fe40f7
 23341 | c000704e30e01e29f8f0882620465684
 23342 | 2db48906a2499fc6f28209ceef13a27a
 23343 | 43e4345b0cd677f08368d600de424d43
 23344 | 296f5aea85bc91bf001ad545dbec53c3
 23345 | 36bd42be95bdc469c0daebf248b54e09
 23346 | c290053d296534e2e36310a0d7ace2e8
 23347 | 3c1aab1ffe99c46bba459e80182e7a56
 23348 | 570e53a8c175c8169ddbb52d44c5dc92
 23349 | d65fe827979cb5dea97b4bed92c956e3
 23350 | 6365a6754b693defc6f3e64542d1c4e3
Cancel request sent
vastbase=#  select ctid,* from test_sample tablesample BERNOULLI(0.05);
   ctid    |   id   |          message
-----------+--------+----------------------------------
 (64,8)    |   7624 | e75af1c6c3e4c740d5735c0a28dc59c4
 (80,21)   |   9541 | a02617490ae98892ee140015e7962fd1
 (83,21)   |   9898 | 49ada89a84ec3293fb4a5bf5320f2de7
 (85,38)   |  10153 | 4d0e904e7d45a14b9d28b5f45b4c5841
 (91,16)   |  10845 | a451f0c41807addbcc3d611cba3567df
 (101,114) |  12133 | 98fcc8b38c9b90eef41a2c0c209ec504
 (112,12)  |  13340 | 96b46cad304016a9225f29550a99be0f
 (113,77)  |  13524 | cb39d46575108c9ee405f6b09c6d954f
 (122,98)  |  14616 | 120d3feacbaf1c7adc63a1be61a76f65
 (179,64)  |  21365 | d6156bd13abaa98b265b682e8fd1de75
```

```
(182,48)   |  21706 | 5c3972cfd72875501897b65e87adca76
(194,68)   |  23154 | 6d1840a8cb269d0786fc3d0b744fccd3
(205,16)   |  24411 | 74663f25f633a49f3b0ffb222fbaed8b
(207,104)  |  24737 | 49e49f926b0072e60d67c51daf3c27dc
(219,3)    |  26064 | df054bdca100691003b56eafe730487d
(225,98)   |  26873 | c1ce06b26411f33aa014a3b3770f5c32
(254,10)   |  30236 | 91a70bb471ceb380a7c43522f83b67bd
(262,86)   |  31264 | 54d98c8a53c06ce3441718caef28be72
(263,88)   |  31385 | ef25817e60853b6c7dcf661a6e618268
(313,5)    |  37252 | cb603f6ef2d935a00868058d8f3ac3bd
(346,90)   |  41264 | e145df0228a738cd06f73d7c635addd9
(381,118)  |  45457 | 77c72d0857f3e42af621bd7ce5cc1eb5
(382,61)   |  45519 | 36f6c89506e39d0f2438752185d5c3ae
(418,15)   |  49757 | 7f9d3ca40cc146e7e3af2815c6c8680c
(425,33)   |  50608 | 770983968bef5794054fd30185632301
(432,101)  |  51509 | 69c48b91a7978adb9e75af2e5a7fd8cc
Cancel request sent
```

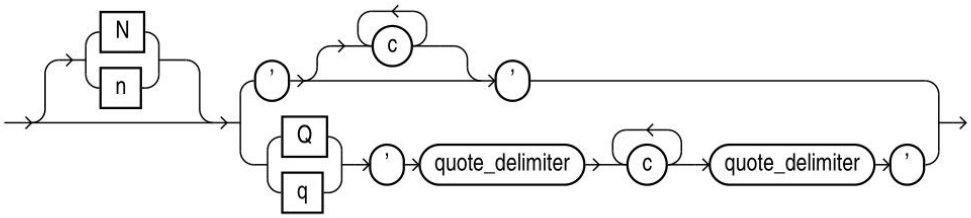
11.17.5.兼容 Q’ 转义字符语法

功能描述

实现 Oracle 中的 Q’ 转义字符，该转义字符开启 Oracle 中的 alternative quoting（引号替换）机制。

语法说明

***string*::=**



开启 alternative quoting（引号替换）机制后，内部 c 的所有字符，包括单引号（'）、双引号（"）等均被作为字符处理。

注意事项

- Q or q 表示将使用引号替代机制。这种机制允许为文本字符串使用广泛的分隔符。
- 最外面的’ ’ 是两个单引号，分别位于左 quote_delimiter 和右 quote_delimiter 的前面和后面。

- c 是用户字符集的任何成员。可以在由 c 字符组成的文本中包含引号（"）。还可以包括 quote_delimiter，只要它不是紧跟在单引号之后。
- quote_delimiter 是除空格、制表符和回车之外的任何单字节或多字节字符。quote_delimiter 可以是单引号。但是，如果 quote_delimiter 出现在文本本身中，请确保它后面不紧跟单引号。
- 如果左 quote_delimiter 是 [, {, <，或者（, 则右 quote_delimiter 必须是相应的], }, >，或者 ）。在所有其他情况下，左和右 quote_delimiter 必须是相同的字符。

示例

```
vastbase=# select q'asdffwwda';
 ?column?
----------
 sdffwwd
(1 row)

vastbase=#  select q'df'er'ew'd';
 ?column?
----------
 f'er'ew'
(1 row)

vastbase=# select q'1dsfwwe1';
 ?column?
----------
 dsfwwe
(1 row)

vastbase=# select q'!ffe'see!';
 ?column?
----------
 ffe'see
(1 row)

vastbase=# select q'[sfeew[]we]';
 ?column?
-----------
 sfeew[]we
(1 row)

vastbase=#  select q'(dsf"sdfe)';
 ?column?
----------
 dsf"sdfe
(1 row)

vastbase=#  select q'{sdfewww  wwfew}';
   ?column?
----------------
 sdfewww  wwfew
(1 row)
```

```
vastbase=#  create table qq_test(name char(20));
CREATE TABLE
vastbase=# insert into qq_test values(q'd er'd 'wwd');
INSERT 0 1
vastbase=# select * from qq_test;
       name
----------------------
  er'd 'ww
(1 row)

vastbase=#  truncate qq_test;
TRUNCATE TABLE
vastbase=#  CREATE OR REPLACE PROCEDURE qq_proc(str text)
vastbase-#  AS
vastbase$# BEGIN
vastbase$#  insert into qq_test values(str);
vastbase$# end;
vastbase$#  /
CREATE PROCEDURE
vastbase=# call qq_proc(q'dde ffedfg eed');
 qq_proc
---------

(1 row)

vastbase=#  select * from qq_test;
       name
----------------------
 de ffedfg ee
(1 row)

vastbase=#

vastbase=# CREATE OR REPLACE function qq_func1() RETURNS text
vastbase-#  AS $$
vastbase$#  DECLARE
vastbase$#  ret text;
vastbase$# BEGIN
vastbase$# ret := q'(dddsef)';
vastbase$# return ret;
vastbase$# end $$ LANGUAGE PLPGSQL;
CREATE FUNCTION
vastbase=# select qq_func1();
 qq_func1
----------
 dddsef
(1 row)
```

11.17.6.兼容虚拟列语法

功能描述

创建数据表时（CREATE TABLE）可指定字段为虚拟列，在修改数据表（ALTER TABLE）可添加虚拟列字段。

语法格式

```
GENERATED ALWAYS AS ( generation_expr ) STORED;
```

参数说明

- generation_expr
  表达式

示例

```
vastbase=# create table t_virtual(c1 int,c2 int,v_c3 int GENERATED ALWAYS AS(c1 + c2) stored);
CREATE TABLE
vastbase=#  insert into t_virtual values(1, 2);
INSERT 0 1
vastbase=#  select * from t_virtual;
 c1 | c2 | v_c3
----+----+------
  1 |  2 |    3
(1 row)

vastbase=# update t_virtual set c1=2;
UPDATE 1
vastbase=# select * from t_virtual;
 c1 | c2 | v_c3
----+----+------
  2 |  2 |    4
(1 row)

vastbase=# ALTER TABLE t_virtual drop c1;
ALTER TABLE
vastbase=#  select * from t_virtual;
 c2
----
  2
(1 row)

vastbase=# drop table t_virtual;
DROP TABLE
vastbase=#  create table t_virtual(c1 int,c2 int,v_c3 int GENERATED ALWAYS AS(c1 + c2) stored);
CREATE TABLE
vastbase=#  ALTER TABLE t_virtual drop v_c3;
ALTER TABLE
```

```
vastbase=# select * from t_virtual;
 c1 | c2
----+----
(0 rows)

vastbase=# drop table t_virtual;
DROP TABLE
vastbase=#  create table t_virtual(c1 int,c2 int,v_c3 int GENERATED ALWAYS AS(c1 + c2) stored);
CREATE TABLE
vastbase=#  insert into t_virtual values(100000,2000000000);
INSERT 0 1
vastbase=# ALTER TABLE t_virtual ALTER v_c3 TYPE float;
ALTER TABLE
vastbase=# ALTER TABLE t_virtual rename v_c3 to v_c4;
ALTER TABLE
vastbase=# insert into t_virtual values(1,2);
INSERT 0 1
vastbase=#  select * from t_virtual;
   c1   |    c2     |    v_c4
--------+------------+------------
 100000 | 2000000000 | 2000100000
      1 |          2 |          3
(2 rows)

vastbase=#  drop table t_virtual;
DROP TABLE
vastbase=# create table t_virtual(c1 int,c2 int,v_c3 int GENERATED ALWAYS AS(c1 + c2) stored);
CREATE TABLE
vastbase=#  ALTER TABLE t_virtual add column c4 int;
ALTER TABLE
vastbase=#  select * from t_virtual;
 c1 | c2 | v_c3 | c4
----+----+------+----
(0 rows)

vastbase=#  drop table t_virtual;
DROP TABLE
```

11.17.7.兼容 VPD 功能

功能描述

通过添加管理策略功能可针对某个模式 schema 中的某个对象 object（包括 table 或视图 view）添加管理策略。比如对模式 TEST 中的表 T1 添加管理策略。

语法格式

➢ 添加 VPD 策略

```
dbms_rls.add_policy(
      object_name text, --对象名称
      policy_name text, --策略名称
      policy_function text, --策略函数名称
```

```
       object_schema text DEFAULT NULL::text, --对象模式，默认为空
       function_schema text DEFAULT NULL::text, --函数模式，默认为空
       statement_types text DEFAULT 'insert,update,delete,select'::text, --策略作用的语句类型，语句类型使用英文逗号分隔，默认为 insert,update,delete,select
       update_check boolean DEFAULT false, --是否启动更新检查，默认为 false
       enable boolean DEFAULT true, --是否启用策略
       sec_relevant_cols text DEFAULT NULL:text --敏感列，敏感列使用英文逗号分隔
) RETURN VOID   --返回值为空
```

➢ 删除 VPD 策略

```
dbms_rls.drop_policy(
        object_name text, --对象名称
        policy_name text, --策略名称
        object_schema text DEFAULT NULL:text --对象模式，默认为空
) RETURN VOID    --返回值为空
```

➢ 启动/禁用 VPD 策略

```
dbms_rls.enable_policy(
        object_name text, --对象名称
        policy_name text, --策略名称
        enable boolean, --是否启用策略
        object_schema text DEFAULT NULL:text --对象模式，默认为空
) RETURN VOID    --返回值为空
```

示例

```
vastbase=# create schema vpd;
CREATE SCHEMA
vastbase=# set search_path to vpd;
SET
vastbase=# create table t1(id1 int, id2 int);
CREATE TABLE
vastbase=# create or replace function f1(schema text, object text)
vastbase-# returns text as $$
vastbase$# declare
vastbase$# begin
vastbase$#   return 'id1 between 1 and 2 and id2 between 3 and 4';
vastbase$# end;
vastbase$# $$
vastbase-# language plpgsql;
CREATE FUNCTION
vastbase=# select dbms_rls.add_policy(
vastbase(#     object_name => 't1',
vastbase(#     policy_name => 'mypolicy1',
vastbase(#     policy_function => 'f1',
vastbase(#     object_schema => 'vpd',
vastbase(#     function_schema => 'vpd',
vastbase(#     enable => true
vastbase(# );
 add_policy
------------

(1 row)
```

```
vastbase=# explain (COSTS false) select id1 from t1;
                       QUERY PLAN
------------------------------------------------------------------
 Seq Scan on t1
   Filter: ((id1 >= 1) AND (id1 <= 2) AND (id2 >= 3) AND (id2 <= 4))
(2 rows)

vastbase=# explain (COSTS false) select id2 from t1;
                       QUERY PLAN
------------------------------------------------------------------
 Seq Scan on t1
   Filter: ((id1 >= 1) AND (id1 <= 2) AND (id2 >= 3) AND (id2 <= 4))
(2 rows)

vastbase=# explain (COSTS false) select * from t1 where id2=10;
                            QUERY PLAN
-------------------------------------------------------------------------------------
 Seq Scan on t1
   Filter: ((id1 >= 1) AND (id1 <= 2) AND (id2 >= 3) AND (id2 <= 4) AND (id2 = 10))
(2 rows)

vastbase=# explain (COSTS false) delete from t1;
                         QUERY PLAN
-----------------------------------------------------------------------
 Delete on t1
   ->  Seq Scan on t1
         Filter: ((id1 >= 1) AND (id1 <= 2) AND (id2 >= 3) AND (id2 <= 4))
(3 rows)

vastbase=# explain (COSTS false) delete from t1 where id1 = 1;
                              QUERY PLAN
-----------------------------------------------------------------------------------------
 Delete on t1
   ->  Seq Scan on t1
         Filter: ((id1 >= 1) AND (id1 <= 2) AND (id2 >= 3) AND (id2 <= 4) AND (id1 = 1))
(3 rows)

vastbase=# explain (COSTS false) update t1 set id1=1;
                         QUERY PLAN
-----------------------------------------------------------------------
 Update on t1
   ->  Seq Scan on t1
         Filter: ((id1 >= 1) AND (id1 <= 2) AND (id2 >= 3) AND (id2 <= 4))
(3 rows)
```

11.17.8.兼容 Oracle 宏变量

功能描述

系统提供宏变量包括：dbtimezone,sessiontimezone 和 systimestamp。

示例

```
select dbtimezone;
select sessiontimezone;
select systimestamp;
```

11.17.9.兼容 ROWNUM

功能描述

支持 Oracle 的 ROWNUM 伪列。

示例

```
CREATE TEMPORARY TABLE empsalary (
depname varchar,
empno bigint,
salary int,
enroll_date date
);
INSERT INTO empsalary VALUES
('develop', 10, 5200, '2007-08-01'),
('sales', 1, 5000, '2006-10-01'),
('personnel', 5, 3500, '2007-12-10'),
('sales', 4, 4800, '2007-08-08'),
('personnel', 2, 3900, '2006-12-23'),
('develop', 7, 4200, '2008-01-01'),
('develop', 9, 4500, '2008-01-01'),
('sales', 3, 4800, '2007-08-01'),
('develop', 8, 6000, '2006-10-01'),
('develop', 11, 5200, '2007-08-15');

select rownum,depname,empno,salary from empsalary where rownum <3;
```

11.17.10. 兼容^=运算符

功能描述

实现类似 Oracle 的^=运算符，即不等于运算符，类似于!=和<>。

示例

```
create table t_ysf4(
  id int,
  col_varchar varchar(30),
  col_int int,
  col_date date
  );
  insert into t_ysf4 values(1,'1',1,'1997-02-20');
  insert into t_ysf4 values(2,'2',2,'2007-02-20');
select * from t_ysf4 where col_int ^='1';
```

```
select 2 ^= 10;
select '1' ^= '2';
select '2010-06-18'::date ^= '2010-06-18'::date;
```

11.17.11. 兼容序列支持 restart 语法

功能描述

支持 ALTER SEQUENCE RESTART 语法，通过该语法重新设置序列的起始值。

语法格式

```
ALTER SEQUENCE name RESTART [WITH] number;
```

参数说明

- number
  设置的序列起始值

示例

```
CREATE SEQUENCE sequence_test20 START WITH 32;
ALTER SEQUENCE sequence_test20 RESTART WITH 22;
SELECT nextval('sequence_test20');
```

11.17.12. 兼容 ROWID

功能描述

支持 Oracle 的 ROWID 伪列特性。

示例

```
select amname,rowid from pg_am where rowid in ('KQoAAA==AAAAAA==AQA=','KQoAAA==AAAAAA==AwA=') order
by oid;
```

11.17.13. 兼容内置包-DBMS_UTILITY

功能描述

支持 Oracle 的 DBMS_UTILITY 包使用，提供了各种实用子程序。

函数说明

- format_call_stack

格式化调用堆栈。该内置函数返回一个格式化的字符串，它显示了执行调用堆栈：直至当前函数调用处的所有过程或者函数的调用顺序。该函数可在存储过程、函数或包中调用以可读格式返回当前调用堆栈。

- exec_ddl_statement

  执行 DDL 语句。可以在过程或函数中调用，用其来执行 DDL 命令。

示例

```
CREATE OR REPLACE FUNCTION checkHexCallStack() returns text as
$$
       DECLARE
          stack text;
       BEGIN
          select * INTO stack from dbms_utility.format_call_stack('o');
          return stack;
       END;
$$ LANGUAGE plpgsql;
```

11.17.14. 兼容内置函数-MONTHS_BETWEEN 函数

功能描述

实现 Oracle 的 MONTHS_BETWEEN 函数。

语法格式

```
MONTHS_BETWEEN(date1, date2)
```

参数说明

- date1

  指定时间值。

  取值范围：date 或 TIMESTAMP WITH TIME ZONE

- date2

  指定时间值。

  取值范围：date 或 TIMESTAMP WITH TIME ZONE

注意事项

如果 date1 比 date2 更晚，则返回结果为正数，反正则为负数，即返回结果为 date1 减去 date2 的差值。

如果 date1 和 date2 都为其所在月份的第几天，或者都为某个月份的最后一天，那么返回值为一个整数。否则，返回的是一个浮点数，整数部分为两个时间差中完整月份的差数，小数部分则为剩余时间差与一个月（31 天）的比例值。

示例

```
select months_between('2020-05-20'::date,'2020-03-20'::date);
```

11.17.15. 兼容内置函数-table 函数

功能描述

实现 Oracle 的 table 函数。

语法格式

```
select * from table(arg);
```

参数说明

- arg

  输入结果集。

示例

```
create table testtable(id int);
insert into testtable values(123),(1234);
select * from table(testtable);
```

11.17.16. 兼容 PL/pgSQL-cursor for select into

功能描述

支持 cursor xxx for/is select ... into ...用法,类似 Oracle 的 cursor xxx is select ... into ...。

语法格式

```
DECLARE
   CURSOR name FOR SELECT columns INTO ... FROM table_name;
```

参数说明

- INTO ...

  可以是声明的变量，数量与 SELECT ...一致。

示例

```
CREATE TABLE stu_info
(
id INT NOT NULL DEFAULT 0,
name TEXT
);

INSERT INTO stu_info (id,name) VALUES(1,'name01');
INSERT INTO stu_info (id,name) VALUES(2,'name02');
INSERT INTO stu_info (id,name) VALUES(3,'name03');
do language plpgsql $$
declare
v_id int;
v_name TEXT;
cursor cur1 for
select id,name into v_id,v_name from stu_info where id>1;
begin
open cur1;
loop
fetch cur1 into v_id,v_name;
if found then
update stu_info set name='test' where id = v_id;
else
exit;
end if;
end loop;
end;
$$;
```

11.17.17. 兼容 PL/pgSQL-PIPELINED 与 PIPE ROW

功能描述

CREATE FUNCTION 中支持 PIPELINED 和 PIPE ROW 语法，实现管道函数（PIPELINE FUNCTION）的功能。

语法格式

```
CREATE [ OR REPLACE ] FUNCTION func_name ( ... )
RETURNS rettype PIPELINED
...
```

参数说明

- rettype
  函数返回值的数据类型。

示例

```
CREATE OR REPLACE FUNCTION func_pipe1(int) RETURNS int PIPELINED AS $$
DECLARE
i int;
BEGIN
```

```
FOR i IN 1 .. $1 LOOP
PIPE ROW(i);
END LOOP;
RETURN;
END;
$$
LANGUAGE plpgsql;
```

11.17.18. 兼容 INSERT 支持别名

功能描述

支持 INSERT 语句中表名使用别名的用法。

语法格式

```
INSERT INTO table_name t1(t1.col1,t1.col2);
```

参数说明

- table_name
  表名
- t1
  表的别名
- col1
  表的列 1
- col2
  表的列 2

示例

```
create table table1 (col1 int,col2 int);
create table table2 (col1 int,col2 int);
insert into table2 values(1,2);
insert into table2 values(2,3);
insert into table1  t1(t1.col1,t1.col2) select  t2.col1,t2.col2 from table2 t2;
select * from table1;
```

11.17.19. 兼容内置包-DBMS_SQL

功能描述

支持 Oracle 的 DBMS_SQL 包使用，提供了各种实用子程序。

函数说明

- OPEN_CURSOR

  打开一个 cursor，并返回该 cursor 的 id。

  - 输入

    无

  - 输出：

    - int：cursor 的 id 号

- PARSER

  解析当前语句的语法。

  - 输入

    - int：cursor 的 id 号
    - text：待解析的 SQL 语句
    - int：标志位，用于兼容 Oracle 语句，无实际意义，仅支持传入 dbms_sql.native

  - 输出：

    无

- DEFINE_COLUMN

  用于定义返回的结果中的列结构。

  - 输入

    - int：cursor 的 id 号
    - nt：列的位置索引号
    - datatype：第三个参数为结果的类型
    - int：当第三个参数为 text 时，该参数表示字符串长度

  - 输出：

    无

- EXECUTE

  用于执行 PARSE 函数处理的 SQL 语句。

  - 输入

    - int：cursor 的 id 号

  - 输出：

    - int：SQL 语句影响的函数

- FETCH_ROWS

  用于载入结果集中的一行结果。

  - 输入
    - int：cursor 的 id 号
  - 输出：
    - int：实际 fetch 的行数，小于 0 表示执行过程中出错
- COLUMN_VALUE

  用于将 fetch 结果中的指定列保存到返回参数中。

  - 输入
    - int：cursor 的 id 号
    - int：返回的列的索引号
    - datatype：参数类型为 inout，类型与获取的数据的类型一致
  - 输出：

    无
- CLOSE_CURSOR

  用于关闭一个 cursor。

  - 输入
    - int：cursor 的 id 号
  - 输出：

    无

示例

```
create table dbms_t1(id number,con varchar(20));
insert into dbms_t1 values(1.5,'aaa'),(2.0,'bbb'),(3,'ccc');
CREATE OR REPLACE function dbms_demo1() return text AS
      cursor_name INT;
      rows_processed INT;
            ret INT;
            rec text;
BEGIN
      cursor_name := DBMS_SQL.open_cursor();
            DBMS_SQL.PARSE(cursor_name,'select con from public.dbms_t1;',0);
            ret := DBMS_SQL.EXECUTE(cursor_name);
            loop
            if DBMS_SQL.FETCH_ROWS(cursor_name) > 0 then
                  select DBMS_SQL.COLUMN_VALUE(cursor_name,1,rec) into rec;
            else
                  exit;
```

```
            end if;
            end loop;
            DBMS_SQL.CLOSE_CURSOR(cursor_name);
            return rec;
END;
/
select dbms_demo1();
```

11.17.20. 兼容序列支持 no cache、nocache、order、no order、noorder

功能描述

提供类似 Oracle 的序列功能、支持创建（create）或修改（alter）时使用：no cache、nocache、order、no order、noorder。

语法格式

➢ 增加

```
CREATE [ TEMPORARY | TEMP ] SEQUENCE [ IF NOT EXISTS ] name [ INCREMENT [ BY ] increment ]
    [ MINVALUE minvalue | NO MINVALUE ] [ MAXVALUE maxvalue | NO MAXVALUE ]
    [ START [ WITH ] start ] [ CACHE cache ] [ NOCACHE ] [ NO CACHE ] [ [ NO ] CYCLE ]
       [ NOORDER ] [ [ NO ] ORDER ] [ OWNED BY { table_name.column_name | NONE } ]
```

➢ 修改

```
ALTER SEQUENCE [ IF EXISTS ] name
    [ AS data_type ]
    [ INCREMENT [ BY ] increment ]
    [ MINVALUE minvalue | NO MINVALUE ] [ MAXVALUE maxvalue | NO MAXVALUE ]
    [ START [ WITH ] start ]
    [ RESTART [ [ WITH ] restart ] ]
    [ CACHE cache ] [ NOCACHE ] [ NO CACHE ] [ [ NO ] CYCLE ]
```

关键字说明

- no cache、nocache

  相当于 cache 1（cache：设置缓存的序列值个数）。

- Order

  保证序列号按请求顺序产生。如果想以序列号作为 timestamp(时间戳)类型的话，可以采用该选项。

- noorder、no order

  与 order 相对，不能保证序列号按请求顺序产生。

示例

```
create sequence SEQ_ID1
```

```
 minvalue 1
 maxvalue 99999999
 start with 1
 increment by 1
 nocache
 order;

create sequence SEQ_ID2
 minvalue 1
 maxvalue 99999999
 start with 1
 increment by 1
 nocache
 order;

create sequence SEQ_ID3
 minvalue 1
 maxvalue 99999999
 start with 1
 increment by 1
 nocache
 noorder;

create sequence SEQ_ID4
 minvalue 1
 maxvalue 99999999
 start with 1
 increment by 1
 nocache
 no order;

create sequence SEQ_ID5
 minvalue 1
 maxvalue 99999999
 start with 1
 increment by 1
 no cache
 order;

create sequence SEQ_ID
minvalue 1
maxvalue 40
start with 1
increment by 1
no cache
noorder;

alter sequence SEQ_ID no cache;
alter sequence SEQ_ID nocache;
alter sequence SEQ_ID order;
alter sequence SEQ_ID noorder;
alter sequence SEQ_ID no order;
```

11.17.21. 兼容内置包-UTL_FILE

功能描述

UTL_FILE 包提供了在存储过程或函数中对系统文件进行读写的功能合集。

函数说明

- fopen

  打开一个文件用于输入或输出。

- is_open

  检查一个文件句柄是否指向一个打开的文件。

- get_line

  从一个打开的文件中读取指定的一行文本。

- get_nextline

  获得下一行。

- put

  将一个字符串写入到文件。

- put_line

  在文件中写入一行，在行的末尾写入一个行终结符。

- new_line

  在文件末尾写入一个行终结符。

- putf

  格式化输出过程。

- fflush

  将文件缓存刷到物理写入。

- fclose

  关闭一个文件。

- fclose_all

  关闭所有打开的文件句柄。

- fremove

  删除磁盘上的文件。

- frename

重命名一个已经存在的文件。

- fcopy

  将一个文件中的内容拷贝到另一个文件中。

- fgetattr

  获取文件属性。

- tmpdir

  获得临时目录路径。

示例

```
1.创建主备文件
shell 执行:
cat > /tmp/test.txt
111aaa
222bbb
333ccc
444ddd

数据库下执行:
insert into utl_file.utl_file_dir values ('/tmp');

2.文件复制:
数据库下执行:
select utl_file.fcopy('/tmp', 'test.txt', '/tmp', 'test_cp.txt',3,6);

3.查看复制的文件
shell 执行:
cat test_cp.txt
```

11.17.22. 兼容内置包-DBMS_ALERT

功能描述

DBMS_ALERT 用于生成并传递数据库警报信息，是一种在数据库内部和应用程序间通信的一种方式。DBMS_ALERT 产生的警报是基于事务的，这意味着如果产生警报的事务未被提交，则等待该警报的 session 会话将不会收到该警报。一个正在等到警报的 session 将在数据库中被阻塞。

函数说明

- REGISTER

  注册当前会话接受指定的警报。

  - 输入

- name：text 类型，指定的警报的名字。

➢ 输出：

无

❖ REMOVE

去除对指定警报的注册。

➢ 输入

- name：text 类型，将要被移除的警报的名字（大小写不敏感）。

➢ 输出

无

❖ REMOVEALL

去除对所有警报的注册。

➢ 输入

无

➢ 输出：

无

❖ SIGNAL

为指定警报发出信号。

➢ 输入

- name：text 类型，指定的警报的名字。
- message：text 类型，本次警报携带的消息字符串，将被传递给执行 wait 的会话。

➢ 输出：

无

❖ WAITANY

等待任何已注册的警报出现。

➢ 输入

- name：text 类型，OUT 参数，收到的警报的名字。
- message：text 类型，OUT 参数，收到的警报携带的消息字符串。如果收到的警报在 WAITANY 之前被 SIGNAL 产生了多次，那么收到的 message 消息字符串是最近的一次 SIGNAL 产生的，之前的 SIGNAL 产生的 message 消息将被忽略。。
- status：integer 类型，OUT 参数，0 表示由收到警报返回，1 表示由超时返回。
- timeout：float8 类型，等待的最长时间。缺省默认值为 86400000（1000 天）。

➢ 输出：

● record 类型，由 out 参数产生。

❖ WAITONE

等待指定的警报出现。

➢ 输入

● name：text 类型，等待的警报的名字。

● message：text 类型，OUT 参数，收到的警报携带的消息字符串。如果收到的警报在 WAITONE 之前被 SIGNAL 产生了多次，那么收到的 message 消息字符串是最近的一次 SIGNAL 产生的，之前的 SIGNAL 产生的 message 消息将被忽略。

● status：integer 类型，OUT 参数，0 表示由收到警报返回，1 表示由超时返回。

● timeout：float8 类型，等待的最长时间。缺省默认值为 86400000（1000 天）。

➢ 输出：

● record 类型，由 out 参数产生。

示例

```
会话 1:接受进程(客户端接收):
select DBMS_ALERT.REGISTER('alert1');
select DBMS_ALERT.WAITONE('alert1',100);

会话 2 发送进程(服务器端发出):
BEGIN;
select DBMS_ALERT.SIGNAL('alert1', 'hello,this is sending process!');
commit;
```

11.17.23. 数组支持 record 类型

功能描述

可以用数组定义一组 record 类型。

语法格式

```
TYPE array_type IS VARRAY(size) OF datatype;
```

参数说明

❖ array_type

要定义的数组类型名。

❖ VARRAY

表示要定义的数组类型。

- size

  取值为正整数，表示可以容纳的成员的最大数量。

- data_type

  要创建的数组中成员的类型，现支持 record 类型。

示例

```
declare
TYPE sum_type IS RECORD
(
sum_id varchar2(20),
sum_sum number
);

sum_rec sum_type;
TYPE sum_type_array IS VARRAY(2) of sum_type;
sum_rec_array sum_type_array;

begin
sum_rec.sum_id :='GLG001_CBB_DAY';
sum_rec.sum_sum := 1;
sum_rec_array[1] := sum_rec;

sum_rec.sum_id :='GLG001_CBC_SUM';
sum_rec.sum_sum := 2;
sum_rec_array[2] := sum_rec;

sum_rec := sum_rec_array[1];
raise notice 'sum_rec_array[1]: % %', sum_rec.sum_id, sum_rec.sum_sum;

sum_rec := sum_rec_array[2];
raise notice 'sum_rec_array[2]: % %', sum_rec.sum_id, sum_rec.sum_sum;
end;
/
```

11.17.24. 视图管理

功能描述

vastbase 支持对单表视图增删改。

语法格式

- 增加

```
INSERT INTO <view_name> (column,...)
VALUES(value,...);
```

- 删除

```
DELETE FROM <view_name>
WHERE condition_clause;
```

➢ 修改

```
UPDATE <view_name> (column,...)
SET column_name = value,...
WHERE condition_clause;
```

示例

```
创建表并插入数据
vastbase2=# create table t_view(id int,name text);
CREATE TABLE
vastbase2=# insert into t_view values(1,'test1');
INSERT 0 1
vastbase2=# insert into t_view values(2,'test2');
INSERT 0 1
vastbase2=# insert into t_view values(3,'test3');
INSERT 0 1
vastbase2=# insert into t_view values(4,'test4');
INSERT 0 1
vastbase2=# insert into t_view values(5,'test5');
INSERT 0 1
vastbase2=# CREATE VIEW t_view_v AS
vastbase2-#SELECT * FROM t_view
vastbase2-#WHERE id > 2;
CREATE VIEW
向视图中插入数据
vastbase2=# insert into t_view values(5,'test5');
INSERT 0 1

修改视图
vastbase2=# update t_view_v set name='test66' where id =3;
UPDATE 1
通过视图删除数据
vastbase2=# delete from t_view_v  where id =3;
DELETE 1
```

对于支持增删改的视图有一些约束与依赖如下：

1)只支持基于单表的简单视图，即形如：select * from table_name where condition 这样的 SQL；

2)视图不能有 INSTEAD 规则和 INSTEAD 触发器；

3)视图中的字段不能是系统列或虚拟列；

4)视图的查询语句不能有 DISRINCT、GROUP BY、HAVING、UNION、INSTERSECT、EXCEPT、WITH AS、LIMIT、OFFSET 关键字；

5)视图不能为 Security-barrier 视图。

11.17.25. 兼容正则表达式

功能描述

兼容正则表达式 REGEXP_INSTR、REGEXP_SUBSTR、REGEXP_REPLACE。

函数说明

- REGEXP_INSTR
  - 语法：

```
REGEXP_INSTR ( source_char, pattern
               [, position
                 [, occurrence
                   [, return_opt
                     [, match_param
                       [, subexpr ]
                     ]
                   ]
                 ]
               ]
             )
```

  - 参数说明：

- source_char：作为搜索值的字符表达式。通常是一个字符列，并且可以是任何字符数据类型。
- pattern：正则表达式。可以是任何字符数据类型。
- positin：一个正整数，表示在 source_char 中开始搜索的位置。默认值为 1。
- occurrence：一个正整数，表示在 source_char 中第 n 次 pattern 出现的位置。
- return_opt：可以指定相对事件应返回的内容。取值有：
  - 如果指定为 0，则返回匹配字符串的第一个字符的位置。
  - 如果指定 1，则返回匹配字符串后第一个字符的位置。
- match_param：一个文本，可以更改函数的默认匹配行为。取值有：
  - 'i'：指定不区分大小写的匹配。
  - 'c'：指定区分大小写的匹配。
  - 'n'：允许句点'.'与换行符匹配。
  - 'm'：将源字符串视为多行。
  - 'x'：忽略空格字符。
- subexpr：对于一个带有子表达式的 pattern，subexpr 是一个 0 到 9 的非负整数，表示 pattern 中指定的子表达式位置。

- REGEXP_SUBSTR
  - 语法：

```
REGEXP_SUBSTR ( source_char, pattern
                [, position
                  [, occurrence
                    [, match_param
```

```
                        [, subexpr ]
                    ]
                ]
            ]
        )
```

➢ 参数说明：

- source_char：作为搜索值的字符表达式。通常是一个字符列，并且可以是任何字符数据类型。
- pattern：正则表达式。可以是任何字符数据类型。
- position：一个正整数，表示在 source_char 中开始搜索的位置。默认值为 1。
- occurrence：一个正整数，表示在 source_char 中第 n 次 pattern 出现的位置。
- match_param：一个文本，可以更改函数的默认匹配行为。取值有
  - 'i'：指定不区分大小写的匹配。
  - 'c'：指定区分大小写的匹配。
  - 'n'：允许句点'.'与换行符匹配。
  - 'm'：将源字符串视为多行。
  - 'x'：忽略空格字符。
- subexpr：对于一个带有子表达式的 patter，subexpr 是一个 0 到 9 的非负整数，表示 pattern 中指定的子表达式位置。

❖ REGEXP_REPLACE

➢ 语法：

```
REGEXP_REPLACE ( source_char, pattern
                 [, replace_string
                   [, position
                     [, occurrence
                       [, match_param ]
                     ]
                   ]
                 ]
               )
```

➢ 参数说明：

- source_char：作为搜索值的字符表达式。通常是一个字符列，并且可以是任何字符数据类型。
- pattern：正则表达式。可以是任何字符数据类型。
- replace_string：用于替换的字符串，可以是任务字符数据类型。
- position：一个正整数，表示在 source_char 中开始搜索的位置。默认值为 1。
- occurrence：一个正整数，表示在 source_char 中第 n 次 pattern 出现的位置。
- match_param：字符类型，可以更改函数的默认匹配行为。取值有：
  - 'i'：指定不区分大小写的匹配。
  - 'c'：指定区分大小写的匹配。
  - 'n'：允许句点'.'与换行符匹配。
  - 'm'：将源字符串视为多行。

■ 'x'：忽略空格字符。

示例

```
select regexp_instr ('hello itmyhome', 'e') from dual;
select regexp_instr ('hello itmyhome', 'h',2) from dual;
select regexp_substr('17,20,23','[^,]+',1,1,'i') as str from dual;
select regexp_replace('0123456789','01234','0abc') from dual;
select regexp_replace('张三      李四','( ){2,}',', ') from dual;
```

11.17.26. 兼容分析函数

功能描述

兼容分析函数 FIRST_VALUE、LAST_VALUE、VAR_POP、VAR_SAMP、VARIANCE、STDDEV_POP、STDDEV_SAMP、STDDEV、COVAR_POP、COVAR_SAMP、CORR。

函数说明

- FIRST_VALUE
  - 语法：FIRST_VALUE (expr) OVER (analytic_clause)
  - 参数说明
    - expr：用于分析的字段或表达式。
    - analytic_clause：分析子句。
- LAST_VALUE
  - 语法：LAST_VALUE (expr) OVER (analytic_clause)
  - 参数说明：
    - expr：用于分析的字段或表达式。
    - analytic_clause：分析子句。
- VAR_POP
  - 语法：VAR_POP (expr) OVER (analytic_clause)
  - 参数说明：
    - expr：用于分析的字段或表达式。
    - analytic_clause：分析子句。
- VAR_SAMP
  - 语法：VAR_SAMP (expr) OVER (analytic_clause)
  - 参数说明：

- expr：用于分析的字段或表达式。
- analytic_clause：分析子句。

❖ STDDEV_SAMP

➢ 语法：STDDEV_SAMP(expr) OVER (analytic_clause)

➢ 参数说明：

- expr：用于分析的字段或表达式。
- analytic_clause：分析子句。

❖ STDDEV

➢ 语法：STDDEV(expr) OVER (analytic_clause)

➢ 参数说明：

- expr：用于分析的字段或表达式。
- analytic_clause：分析子句。

❖ COVAR_POP

➢ 语法：COVAR_POP(expr) OVER (analytic_clause)

➢ 参数说明：

- expr：用于分析的字段或表达式。
- analytic_clause：分析子句。

❖ COVAR_SAMP

➢ 语法：COVAR_SAMP(expr) OVER (analytic_clause)

➢ 参数说明：

- expr：用于分析的字段或表达式。
- analytic_clause：分析子句。

❖ CORR

➢ 语法：CORR(expr) OVER (analytic_clause)

➢ 参数说明：

- expr：用于分析的字段或表达式。
- analytic_clause：分析子句。

示例

```
--创建测试表
create table t_salary (id number(2), name varchar2(10), salary number(6,2));
insert into t_salary values (1,'张三',120);
insert into t_salary values (2,'李四',240);
```

```
insert into t_salary values (2,'王五',80);
insert into t_salary values (3,'李飞',300);
insert into t_salary values (3,'李明',500);
insert into t_salary values (3,'张强',1300);
insert into t_salary values (3,'吴迪',40);
insert into t_salary values (3,'刘飞',200);
insert into t_salary values (3,'侯明',800);

select ID,name, salary,FIRST_VALUE(salary) OVER( partition by ID ORDER BY salary ) min_salary from
t_salary;

select ID, name, salary, LAST_VALUE(salary) OVER (partition by ID order by
 salary rows between  unbounded preceding and unbounded following)  as highest_salary from t_salary
order by ID, name;

SELECT id,
      name,
      salary,
      VAR_POP (salary) OVER (PARTITION BY id ORDER BY id)  AS "VAR_POP",    --方差
      VAR_SAMP (salary) OVER (PARTITION BY id ORDER BY id)  AS "VAR_SAMP", --样本方差
      STDDEV (salary) OVER (PARTITION BY id ORDER BY id)  AS "STDDEV", --标准差
      STDDEV_SAMP (salary) OVER (PARTITION BY id ORDER BY id)   AS "STDDEV_SAMP"   --样本标准差
        from   t_salary;

--创建测试表
create table window_test(name int, value int);
insert into window_test values(1,600);
insert into window_test values(2,470);
insert into window_test values(3,170);
insert into window_test values(4,430);
insert into window_test values(5,300);

select name, VARIANCE(value) OVER( ORDER BY name ) max_value from window_test;
select name, STDDEV_POP(value) OVER( ORDER BY name ) max_value from window_test;
select name, COVAR_POP(value, value) OVER( ORDER BY name ) max_value from window_test;
select name, COVAR_SAMP(value, value) OVER( ORDER BY name ) max_value from window_test;
select name, CORR(value, value) OVER( ORDER BY name ) max_value from window_test;
```

11.17.27. 兼容内置函数-数值函数

功能描述

实现 Oracle 的部分数值函数：

- sinh 函数：计算输入参数的双曲正弦值，并返回结果。
- cosh 函数:计算输入参数的双曲余弦值，并返回结果。
- tanh 函数:计算输入参数的双曲正切值，并返回结果。

语法格式

➢ sinh 函数

```
select sinh(arg);
```

➢ cosh 函数

```
select cosh(arg);
```

➢ tanh 函数

```
select tanh(arg);
```

参数说明

➢ sinh 函数、cosh 函数、tanh 函数

❖ arg

输入 float 类型数值。

示例

```
select cosh(1);
select sinh(33);
select tanh(9);
```

11.17.28. 兼容内置函数-空值比较函数

功能描述

实现 Oracle 的空值比较函数：

- lnnvl 函数：bool 类型取反输出，如果输入参数是 NULL 则返回 true。
- nanvl 函数：如果输入值（参数一）是 NaN(不是数字)，它指示数据库返回一个替代值（参数二）。
- nvl2 函数：nvl2 允许根据指定的表达式是否为 null 来确定查询返回的值。

语法格式

➢ lnnvl 函数

```
select lnnvl(arg);
```

➢ nanvl 函数

```
select nanvl(arg1, arg2);
```

➢ nvl2 函数

```
select nvl2(arg1, arg2, arg3);
```

参数说明

➢ lnnvl 函数

❖ arg

输入 bool 类型，可以是返回 bool 类型的函数或表达式，可以是 NULL。

➢ nanvl 函数

- arg1、arg2

  输入数值类型，还可以是可隐式转换为数值类型的其他类型。

➢ nvl2 函数

- arg1、arg2、arg3

  输入任意类型。3 个参数需要是同一个类型，除了 NULL。NULL 可以和其他任意类型混用。

示例

```
select lnnvl(1=1);
select nanvl(1,2);
select nvl2(1,2,3);
```

11.17.29. 兼容内置包-DBMS_OUTPUT

功能描述

DBMS_OUTPUT 包提供了在存储过程或包内对外发送信息的功能。

函数说明

- serveroutput

  消息打印开关。
- enable

  在 serveroutput 设置为 true 的情况下，用来使 dbms_output 生效。
- disable

  在 serveroutput 设置为 true 的情况下，用来使 dbms_output 失效。
- put

  将数据写入到缓存中，等到 put_line/new_line 时一起输出。
- put_line

  将数据写入到缓存中，并将缓存中的数据一起输出。
- new_line

  向 put 写入的行末尾添加终结符，如果缓存中有数据将其输出。
- get_line

  从缓存中读取一行数据。
- get_lines

  从缓存中读取多行数据。

示例

```
CREATE TABLE dbms_output_test (buff VARCHAR(20),status INTEGER);
CREATE FUNCTION dbms_output_test() RETURNS VOID AS $$
```

```
DECLARE
    buff VARCHAR(20);
    buff1 VARCHAR(20);
    buff2 VARCHAR(20);
    buff3 VARCHAR(20);
    stts INTEGER := 10;
BEGIN
    PERFORM DBMS_OUTPUT.DISABLE();
    PERFORM DBMS_OUTPUT.ENABLE();
    PERFORM DBMS_OUTPUT.SERVEROUTPUT (true);
    PERFORM DBMS_OUTPUT.PUT('ORAFCE TEST 1');
    PERFORM DBMS_OUTPUT.PUT_LINE('ORAFCE TEST 2');
    PERFORM DBMS_OUTPUT.PUT_LINE('ORAFCE TEST 3');
    SELECT INTO buff1,buff2.buff3,stts lines[1],lines[2],lines[3],numlines FROM
DBMS_OUTPUT.GET_LINES(stts);
    INSERT INTO dbms_output_test VALUES (buff1,stts);
    PERFORM DBMS_OUTPUT.NEW_LINE();
    INSERT INTO dbms_output_test VALUES (buff2,stts);
    INSERT INTO dbms_output_test VALUES (buff3,stts);
    SELECT INTO buff,stts lines[1],numlines FROM DBMS_OUTPUT.GET_LINES();
    INSERT INTO dbms_output_test VALUES(buff,stts);
    PERFORM DBMS_OUTPUT.DISABLE();
    PERFORM DBMS_OUTPUT.ENABLE();
END;
$$ LANGUAGE plpgsql;
SELECT dbms_output_test();
SELECT * FROM dbms_output_test;
```

11.17.30. 兼容 WITH AS 子语句

功能描述

With 查询语句以 with 开头，相当于在查询之前先构建一个临时表，被指定的查询结果存于临时表中，之后便可多次使用它进一步的分析和处理。

语法格式

```
WITH
  name_for_summary_data AS (
    SELECT Statement)
  SELECT columns
  FROM name_for_summary_data
  WHERE conditions <=> (
                SELECT column
                FROM name_for_summary_data)
  [ORDER BY columns]
```

参数说明

- name_for_summary_data

  with 子句的名称。

- Statement

  查询表达式。

- columns

  查询的列。

- conditions

  where 条件表达式。

示例

```
WITH CTE1(aa) AS (SELECT 1 UNION ALL SELECT 2 UNION ALL SELECT 3),
CTE2(bb) AS (SELECT 'a' UNION ALL SELECT 'b'),
CTE3(cc) AS (SELECT aa FROM CTE1 WHERE aa < 3)
SELECT * FROM CTE2, CTE3;
```

11.17.31. 兼容创建 Package

功能描述

Package 是用于组织过程、函数和变量的一种方式，由包声明和包体组成。

语法格式

- 包声明

```
CREATE [OR REPLACE] PACKAGE [schema_name.]<package_name> IS | AS
   declarations;
END;
```

- 包体

```
CREATE [OR REPLACE] PACKAGE BODY [schema_name.]<package_name> IS | AS
   declarations
   implementations;
[BEGIN
EXCEPTION]
END;
```

参数说明

- 包声明
  - declarations

    可以定义公有数据类型、声明公有游标、声明公有变量和常量、声明公有子程序。

- 包体
  - declarations

定义私有数据类型、声明私有变量和常量、声明私有子程序。

- implementations

  定义私有子程序、定义游标、定义公有子程序。

示例

```
1.创建包
CREATE PACKAGE emp_bonus AS
PROCEDURE calc_bonus (d date);
END;
/

2.创建包体
CREATE PACKAGE BODY emp_bonus AS
PROCEDURE calc_bonus (date_hired DATE) IS
BEGIN
DBMS_OUTPUT.PUT_LINE
('Employees hired on ' || date_hired || ' get bonus.');
END;
END;
/

DROP PACKAGE BODY emp_bonus;
```

11.17.32. 兼容（+）操作符的左外连接和右外连接查询

功能描述

(+)外连接为左右外连接（left [outer] join、right [outer] join）的另一种表示法。当(+) 在“=”左侧时，代表以右侧为全量输出，即为右外连接；当(+) 在“=”右侧时，代表以左侧为全量输出，即为左外连接。语法与左右外连接一致。

示例

```
1.创建测试表
CREATE TABLE t_plus_outer_join_1(id INTEGER, name TEXT);
CREATE TABLE t_plus_outer_join_2(id INTEGER, addr TEXT);
2.创建测试数据
INSERT INTO t_plus_outer_join_1 VALUES(1, '张三');
INSERT INTO t_plus_outer_join_1 VALUES(2, '李四');
INSERT INTO t_plus_outer_join_1 VALUES(3, '赵五');
INSERT INTO t_plus_outer_join_2 VALUES(1, '广州');
INSERT INTO t_plus_outer_join_2 VALUES(2, '深圳');
INSERT INTO t_plus_outer_join_2 VALUES(4, '北京');
3.执行(+)左外连接
SELECT a.id, name, addr FROM t_plus_outer_join_1 a, t_plus_outer_join_2 b WHERE a.id = b.id(+);
4.执行(+)右外连接
```

```
SELECT a.id, name, addr FROM t_plus_outer_join_1 a, t_plus_outer_join_2 b WHERE a.id(+) = b.id;
5.删除测试表
DROP TABLE t_plus_outer_join_1, t_plus_outer_join_2;
```

11.17.33. 兼容 PL/pgSQL-存储过程自治事务

功能描述

在 plpgsql 定义存储过程中，在存储过程的声明部分增加指令：PRAGMA AUTONOMOUS_TRANSACTION，来控制对当前的存储过程中的内部事务，能够独立提交，而不影响其他事务。该执行方式与普通方式执行存储过程的结果相同。

示例

```
create table at_tb2(id int, val varchar(20));
create or replace procedure at_test3(i int)
AS
DECLARE
  PRAGMA AUTONOMOUS_TRANSACTION;
BEGIN
  START TRANSACTION;
  insert into at_tb2 values(1, 'before s1');
  insert into at_tb2 values(2, 'after s1');
  if i > 10 then
  rollback;
  else
  commit;
  end if;
end;
/
call at_test3(6);
 at_test3
----------

(1 row)

select * from at_tb2;
 id |   val
----+-----------
  1 | before s1
  2 | after s1
(2 rows)
```

11.18. OCI/OCCI 接口

支持 Oracle Call Interface 和 Oracle C++ Call Interface 的以下功能函数：

OCILogon	创建一个简单的登录会话
OCILogoff	释放使用 OCILogon 登录的会话

SQ	执行模块实现函数
OCIStmtPrepar	函数实现准备要执行的 SQL 或 PL / SQL 语句
OCIBindByPos	在 SQL 语句或 PL / SQL 块中的程序变量和占位符之间创建关联
OCIBindByName	函数实现在 SQL 语句或 PL / SQL 块中的程序变量和占位符之间创建关联
OCIBindArrayOfStruct	函数实现为静态数组绑定设置跳过参数
OCIDefineByPos	函数实现将选择列表中的项目与类型和输出数据缓冲区关联
OCIDefineArrayOfStruct	指定在多行多列结构读取数组中使用的静态数组定义所必需的其他属性
OCIStmtExecute	函数实现将应用程序请求与服务器关联
OCIStmtFetch	函数实现从查询中获取行
OCITransStart	函数实现设置事务开始
OCITransCommit	函数实现事务提交
OCITransRollback	函数实现事务回滚
OCIHandleAlloc	函数返回指向已分配和初始化的句柄的指针
OCIHandleFree	函数实现该调用显式取消分配句柄
OCIErrorGet	在提供的缓冲区中返回错误消息和 Oracle 数据库错误代码
OCIDateToText	函数实现将日期类型转换为字符串
OCINumberFromText	函数实现将 number 类型转换为字符类型
OCINumberFromReal	函数实现将浮点类型转化为 number 类型
OCINumberfromInt	函数实现将 int 类型转换为 number 类型
OCIDateSysDate	函数实现获取客户端的当前系统日期和时间
OCIDateCompare	函数实现两个日期进行比较
OCINumberCmp	函数实现将两个 number 类型进行比较
OCINumberAdd	函数实现将两个 number 类型值进行相加
OCINumberSub	函数实现将两个 number 类型值进行相减
OCINumberMul	函数实现将两个 number 类型值进行相乘
OCINumberDiv	函数实现将两个 number 类型值进行相除
OCIParamGet	函数返回由描述句柄或语句句柄中的位置指定的参数的描述符

11.19. 附录

11.19.1.GIN 索引

11.19.1.1. 介绍

GIN（Generalized Inverted Index）通用倒排索引。设计为处理索引项为组合值的情况，查询时需要通过索引搜索出出现在组合值中的特定元素值。例如，文档是由多个单词组成，需要查询出文档中包含的特定单词。

使用 item 表示索引的组合值，key 表示一个元素值。GIN 用来存储和搜索 key，而不是 item。

GIN 索引存储一系列（key，posting list）键值对，这里的 posting list 是一组出现 key 的行 ID。由于每个 item 都可能包含多个 key，同一个行 ID 可能会出现在多个 posting list 中，而每个 key 值只被存储一次，所以在相同的 key 在 item 中出现多次的情况下，GIN 索引是非常简洁的。

因为 GIN 索引的访问方式不需要了解他的运行方式，所以 GIN 索引是通用的。GIN 索引使用为特殊数据类型定义的策略。策略定义了如何从索引选项和查询条件中抽出 key，以及如何确定在查询中包含某些 key 值的行是否实际满足查询条件。

11.19.1.2. 扩展性

GIN 索引的接口实现了一个高层次的抽象，要求访问用户仅需要实现被访问数据类型的语义。GIN 层自身可以处理并发操作、记录日志、搜索树结构的任务。

定义 GIN 索引的访问方式所要做的事情就是实现多个用户定义的方法，这些方法定义了键在树中的行为、键与键之间的关系、需要索引的 item、能够使用索引的查询。简而言之，GIN 索引将扩展性与普遍性、代码重用、清晰的接口结合在了一起。

实现 GIN 索引的操作符类有如下四个方法：

- int compare(Datum a, Datum b)

  比较两个 key（不是索引的 item）然后返回一个小于零、零或大于零的值，分别表示第一个 key 小于、等于或大于第二个 key。NULL 不会被传入这个函数。

- Datum *extractValue(Datum itemValue, int32 *nkeys, bool **nullFlags)

  给定一个要被索引的 item，返回一个对应 key 的数组。返回 key 的数目必须存储在*nkeys 中。如果任何 key 都可能为 NULL，还要分配包含*nkeys 个布尔元素的数组，将地址存储到*nullFlags，并且根据需要设置 NULL 值。 如果所有 key 都是非 NULL，可以让*nullFlags 保持为 NULL（他的初始值）。如果 item 不包含任何 key，返回值可以为 NULL。

❖ Datum *extractQuery(Datum query, int32 *nkeys, StrategyNumber n, bool **pmatch, Pointer **extra_data, bool **nullFlags, int32 *searchMode)

给定一个被查询的值，返回一个对应的 key 的数组。也就是说，query 是可索引操作符右侧的值，而该操作符左侧是被索引的字段。 n 是操作符类中操作符的策略号。通常，extractQuery 需要参考 n 来决定 query 的数据类型以及抽取键值的方法。返回 key 的个数必须存放在*nkeys 中。如果任何 key 都可能为 NULL，还要分配包含*nkeys 个布尔元素的数组，将地址存储到 *nullFlags，并且根据需要设置 NULL 值。 如果所有 key 都是非 NULL 的，可以让*nullFlags 保持为 NULL（他的初始值）。如果 query 不包含任何 key，返回值可以为 NULL。

searchMode 是一个输出参数，他允许 extractQuery 指定一些关于如何执行搜索的细节。如果设置*searchMode 为 GIN_SEARCH_MODE_DEFAULT (这也是调用函数前此参数的初始化值)，只有那些至少返回一个 key 的 item 才能被考虑作为候选匹配项。如果设置*searchMode 为 GIN_SEARCH_MODE_INCLUDE_EMPTY，除了包含至少一个匹配 key 的 item 之外，根本不包含任何 key 的 item 也被考虑作为候选匹配项。（这个模式对于实现像“是否是子集”这样的操作是有用的）如果设置*searchMode 为 GIN_SEARCH_MODE_ALL，索引中所有非 NULL 的 item 都被考虑作为候选匹配项，不管他们是否匹配返回 key 中的任何一个。

pmatch 是一个允许支持部分匹配的输出参数。如果使用此参数，extractQuery 必须分配有 *nkeys 个布尔元素的数组，并把数组地址保存到*pmatch。如果需要部分匹配相应的 key，则数组的每个元素应该设置为 TRUE；如果不需要匹配，则设置为 FALSE。如果设置*pmatch 为 NULL，则假设 GIN 不需要部分匹配。在函数调用前这个值被初始化为 NULL，因此，对于不支持部分匹配的操作符类，可以忽略这个参数。

extra_data 是一个允许 extractQuery 以 consistent 和 comparePartial 的方式传递额外数据的输出参数。如果使用他，extractQuery 必须分配一个包含*nkeys 个 Pointer 元素的数组，并把数组地址保存到*extra_data，然后把他想附加的东西存储到各个独立的指针中。在函数调用前这个值初始化为 NULL，因此，对于不需要附加数据的操作符类，可以忽略这个参数。如果设置了*extra_data，那么以 consistent 方式传递整个数组，使用 comparePartial 方式传递适当的元素。

❖ bool consistent(bool check[], StrategyNumber n, Datum query, int32 nkeys, Pointer extra_data[], bool *recheck, Datum queryKeys[], bool nullFlags[])

如果被索引项满足 StrategyNumber 为 n 的查询操作符则返回 TRUE。这个函数并不直接访问被索引项的值，因为 GIN 并没有精确的把项目保存下来，但是需要知道从查询中提取的哪些键值出现在给定的被索引项中。 check 数组的长度是 nkeys，这个与 query 调用 extractQuery 函数返回的键值的数目相同。如果索引项包含了相应的查询键，check 数组中对应的元素值就是 TRUE。比如，如果(check[i] == TRUE)，那么意味着 extractQuery 的结果数组的第 i 个键出现在索引项中。考虑可能会用到 consistent 方式，原始的 query 也被作为参数传入进来。

与此相同的还有 extractQuery 函数返回的 queryKeys[]和 nullFlags[]。 extra_data 是 extractQuery 函数返回的额外数据数组，如果没有的话就是 NULL。

当 extractQuery 在 queryKeys[]中返回一个 NULL 的键值，如果被索引项包含 NULL 键值，相应的 check[]中的元素是 TRUE。也就是说，check[]的语义很像 IS NOT DISTINCT FROM。如果需要知道是通常值匹配还是 NULL 匹配，consistent 函数可以检查相应的 nullFlags[]元素。

成功执行后，如果堆元组需要针对查询运算符进行重新检查，*recheck 需要设置为 TRUE，如果索引测试已经是精确的了，则设为 FALSE。也就是说，FALSE 的返回值确保堆元组不匹配这个查询；设置*recheck 为 FALSE 的 TRUE 的返回值确保堆元组匹配这个查询；设置*recheck 为 TRUE 的 TRUE 的返回值意味着堆元组可能匹配这个查询，因此需要通过直接对照原始索引项对查询运算符进行获取和重新检查。

GIN 操作符类可以可选地提供第五个函数。

- int comparePartial(Datum partial_key, Datum key, StrategyNumber n, Pointer extra_data)

  比较一个部分匹配查询键和一个索引键。返回一个整型值，它的符号代表了不同的含义：小于 0 意味着索引键不匹配查询，但是索引扫描应该继续； 0 意味着索引键匹配查询；大于 0 指示应该终止索引扫描，因为不可能再有更多的匹配。在需要确定何时结束扫描的语义的情况下，这里提供了生成部分一致查询的操作符的策略号 n。同样的，extra_data 是 extractQuery 生成的额外数据数组中的相应元素，如果没有对应的元素，则为 NULL。 NULL 的键永远不会被传入这个函数。

为了支持"部分匹配"查询，一个操作符类必须提供 comparePartial 方法，并且当遇到部分匹配查询时，他的 extractQuery 方法必须设置 pmatch 参数。详细信息请参考部分匹配算法。

上面的各种 Datum 值的实际数据类型根据操作符类的不同而不同。传入到 extractValue 中的项目值总是操作符类的输入类型，所有的键值类型必须是这个类的 STORAGE 类型。传入到 extractQuery 和 consistent 的 query 参数的类型是由策略号识别的类成员操作符的右操作数的输入类型。他不需要和项目类型相同，只要可以从中抽取出正确类型的键值。

11.19.1.3. 实现

在内部，GIN 索引包含一个在键上构造的 B-tree 索引，每个键是一个或多个被索引项的一个元素（比如，一个数组的一个成员）。并且页面上每个元组包含了堆指针的 B-tree 的一个指针（一个 posting tree），当列表小到足以和键值一起存储到一个索引元组中时，则是堆指针的一个简单列表（一个 posting list）。

多列 GIN 索引通过在组合值（列号，键值）上建立一个单个的 B-tree 实现。不同列的键值可以有不同的类型。

GIN 快速更新技术

由于倒排索引的本身特性影响，更新一个 GIN 索引可能会比较慢。插入或更新一个堆行可能导致许多往索引的插入。当对表执行 VACUUM 后，或者如果待处理实体的列表太大了（大于 work_mem），这些实体被使用和初始索引创建时用到的相同的 bulk 插入方法，移动到主要的 GIN 数据结构。即使把额外的 VACUUM 开销算进去，这也大大提升了 GIN 索引更新的速度。而且，这种额外开销的工作可以通过后台进程而不是前端查询来处理。

这种方法的主要缺点在于搜索时除了常规的索引还必须要扫描待处理实体的列表。因此，大的待处理实体的列表会显著的拖慢搜索。另一个缺点是，虽然大多数更新很快，但是一个导致待处理列表（pending list）变得“太大”的更新将引发一个立即清理，并因此比起其它更新会非常慢。恰当的使用 autovacuum 可以弱化这两个问题。

如果一致的响应时间（清理实体速度和更新速度的响应时间）比更新速度更重要，可以通过把 GIN 索引的存储参数 FASTUPDATE 设置为 off 而不使用待处理实体。详细请参考 11.16.41CREATE INDEX。

部分匹配算法

GIN 可以支持“部分匹配”查询。即：查询并不决定单个或多个键的一个精确的匹配，而是，可能的匹配落在一个合理的狭窄键值范围内(根据 compare 支持函数决定的键值排序顺序)。此时，extractQuery 方法并不返回一个用于精确匹配的键值，取而代之的是，返回一个要被搜索的键值范围的下边界，并且设置 pmatch 为 true。然后，使用 comparePartial 方式扫描这个键值范围。comparePartial 必须为一个相匹配的索引键返回 0，如果不匹配但依然在被搜索范围内时返回小于 0 的值，对超过可以匹配的范围的索引键则返回大于 0 的值。

11.19.1.4. GIN 提示与技巧

创建 vs 插入

由于可能要为每个项目插入很多键，所以 GIN 索引的插入可能比较慢。对于向表中大量插入的操作，我们建议先删除 GIN 索引，在完成插入之后再重建索引。与 GIN 索引创建、查询性能相关的 GUC 参数如下：

- maintenance_work_mem

  GIN 索引的构建时间对 maintenance_work_mem 的设置非常敏感。

- work_mem

  往已有的启用了 FASTUPDATE 的 GIN 索引的插入操作期间，只要待处理实体列表的大小超过了 work_mem，系统就会清理这个列表。为了避免可观察到的响应时间的大起大落，让待处理实体列表在后台被清理是比较合适的（比如通过 autovacuum）。前端清理操作可以通过增加 work_mem 或者执行 autovacuum 来避免。然而，扩大 work_mem 意味着如果发生了前端清理，那么他的执行时间将更长。

- gin_fuzzy_search_limit

  开发 GIN 索引的主要目的是为了让 Vastbase 支持高度可伸缩的全文索引，并且常常会遇见全文索引返回海量结果的情形。而且，这经常发生在查询高频词的时候，因而这样的结果集没什么用处。因为从磁盘读取大量记录并对其进行排序会消耗大量资源，这在产品环境下是不能接受的。

  为了控制这种情况，GIN 索引有一个可配置的返回结果行数上限的配置参数 gin_fuzzy_search_limit。缺省值 0 表示没有限制。如果设置了非零值，那么返回结果就是从完整结果集中随机选择的一部分。

11.19.2.扩展函数

下表列举了 Vastbase 中支持的扩展函数，不作为商用特性交付，仅供参考。

分类	函数名称	描述
访问权限查询函数	has_sequence_privilege(user, sequence, privilege)	指定用户是否有访问序列的权限
	has_sequence_privilege(sequence, privilege)	当前用户是否有访问序列的权限
触发器函数	pg_get_triggerdef(trigger_oid)	为触发器获取 CREATE [CONSTRAINT] TRIGGER 命令
	pg_get_triggerdef(trigger_oid, pretty_bool)	为触发器获取 CREATE [CONSTRAINT] TRIGGER 命令

11.19.3.扩展语法

Vastbase 提供了一些扩展语法，目前处于 Beta 阶段，仅供内部使用，外部不可用。
下表列举了 Vastbase 中支持的扩展语法，不作为商用特性交付，仅供参考。

表 11-57. 扩展 SQL 语法

类别	语法关键字	描述
创建表（CREATE TABLE）	INHERITS (parent_table [, ...])	是否支持继承表。
	table_constraint: EXCLUDE [USING index_method] (exclude_element WITH operator [, ...]) index_parameters [WHERE (predicate)] | FOREIGN KEY (column_name [, ...]) REFERENCES	不支持用 EXCLUDE [USING index_method] (exclude_element WITH operator [, ...])为表创建排除约束。

类别	语法关键字	描述
	reftable [(refcolumn [, ...])] [MATCH FULL | MATCH PARTIAL | MATCH SIMPLE] [ON DELETE action] [ON UPDATE action]	
修改表（ALTER TABLE）	TO { GROUP groupname | NODE (nodename [, ...]) }	修改表数据分布的数据节点列表。
	DELETE NODE (nodename [, ...])	删除表数据分布的数据节点。
加载模块	CREATE EXTENSION	把一个新的模块（例如 DBLINK）加载进当前数据库中。
	ALTER EXTENSION	修改已加载的模块。
	DROP EXTENSION	删除已加载的模块。
聚集函数	CREATE AGGREGATE	定义一个新的聚集函数。
	ALTER AGGREGATE	修改一个聚集函数的定义。
	DROP AGGREGATE	删除一个现存的聚集函数。
操作符	CREATE OPERATOR	定义一个新的操作符。
	ALTER OPERATOR	修改一个操作符的定义。
	DROP OPERATOR	从数据库中删除一个现存的操作符。
操作符类	CREATE OPERATOR CLASS	定义一个新的操作符类。
	ALTER OPERATOR CLASS	修改一个操作符类的定义。
	DROP OPERATOR CLASS	删除一个现有操作符类。
操作符族	CREATE OPERATOR FAMILY	定义一个新的操作符族。
	ALTER OPERATOR FAMILY	修改一个操作符族的定义。
	DROP OPERATOR FAMILY	删除一个已有的操作符族。
文本检索解析器	CREATE TEXT SEARCH PARSER	创建一个新的文本检索解析器。
	ALTER TEXT SEARCH PARSER	修改文本检索解析器。
	DROP TEXT SEARCH PARSER	删除现有的文本检索解析器。
文本检索模	CREATE TEXT SEARCH TEMPLATE	创建一个新的文本检索模板。

类别	语法关键字	描述
板	ALTER TEXT SEARCH TEMPLATE	修改文本检索模板。
	DROP TEXT SEARCH TEMPLATE	删除现有的文本检索模板。
排序规则	CREATE COLLATION	创建排序规则。 排序规则使得用户可以定义数据在列级，甚至是操作级的排序规则和字符分类行为。
	ALTER COLLATION	修改排序规则。
	DROP COLLATION	删除排序规则。
生成一个通知	NOTIFY	NOTIFY 命令发送带有可选的“有效载荷”字符串的通知给先前执行监听数据库中指定的同一渠道的每一个客户端。
监听一个通知	LISTEN	给当前会话注册一个监听器。
停止监听通知信息	UNLISTEN	清除本次会话注册的所有监听器。
加载或重新加载一个共享库文件	LOAD	加载一个共享库文件到数据库服务器的地址空间。
释放一个数据库的会话资源	DISCARD	释放与此数据库会话的相关资源。
过程语言	CREATE LANGUAGE	注册一个新的语言。
	ALTER LANGUAGE	修改一个过程语言的定义。
	DROP LANGUAGE	删除一个过程语言。
域	CREATE DOMAIN	创建一个新的域。
	ALTER DOMAIN	修改现有域的定义。
	DROP DOMAIN	删除一个域。
编码转换	CREATE CONVERSION	定义字符集之间的转换。

类别	语法关键字	描述
	ALTER CONVERSION	修改一个编码转换的定义。
	DROP CONVERSION	删除一个先前定义过的编码转换。
类型转换	CREATE CAST	定义一个新的转换。一个转换说明如何在两个类型之间进行转换。
	DROP CAST	删除一个先前定义过的类型转换。
创建游标	CURSOR name [BINARY] [INSENSITIVE] [[NO] SCROLL] [WITH HOLD] FOR query	INSENSITIVE： 这个关键字没有什么作用，只在为了兼容 SQL 标准。 SCROLL：声明该游标可以用于反向检索。 WITH HOLD:声明该游标可以在创建它的事务成功提交后继续使用。
移动游标	MOVE BACKWARD	反向移动游标，需与 SCROLL 结合才能使用。

12. 用户自定义函数

12.1. PL/pgsql 语言函数

PL/pgsql 是一种可载入的过程语言。

用 PL/pgsql 创建的函数可以被用在任何可以使用内建函数的地方。例如，可以创建复杂条件的计算函数并且后面用它们来定义操作符或把它们用于索引表达式。

SQL 被大多数数据库用作查询语言。它是可移植的并且容易学习。但是每一个 SQL 语句必须由数据库服务器单独执行。

这意味着客户端应用必须发送每一个查询到数据库服务器、等待它被处理、接收并处理结果、做一些计算，然后发送更多查询给服务器。如果客户端和数据库服务器不在同一台机器上，则会引起进程间通信并且将带来网络负担。

通过 PL/pgsql，可以将一整块计算和一系列查询分组在数据库服务器内部，这样就有了一种过程语言的能力并且使 SQL 更易用，同时能节省客户端/服务器通信开销。

- 客户端和服务器之间的额外往返通信被消除。
- 客户端不需要的中间结果不必被整理或者在服务器和客户端之间传送。
- 多轮的查询解析可以被避免。

PL/pgsql 可以使用 SQL 中所有的数据类型、操作符和函数。

应用 PL/pgsql 创建函数的语法为 11.16.39CREATE FUNCTION。PL/pgsql 是一种可载入的过程语言。其应用方法与 13 存储过程相似，只是存储过程无返回值，函数有返回值。

13. 存储过程

13.1. 存储过程

商业规则和业务逻辑可以通过程序存储在 Vastbase 中，这个程序就是存储过程。

存储过程是SQL和PL/SQL的组合。存储过程使执行商业规则的代码可以从应用程序中移动到数据库。从而，代码存储一次能够被多个程序使用。

存储过程的创建及调用办法请参考 11.16.43CREATE PROCEDURE。

12.1PL/pgsql 语言函数节所提到的 PL/pgsql 语言创建的函数与存储过程的应用方法相通。下面各节中，除非特别声明，否则内容通用于存储过程和 PL/pgsql 语言函数。

13.2. 数据类型

数据类型是一组值的集合以及定义在这个值集上的一组操作。Vastbase 数据库是由表的集合组成的，而各表中的列定义了该表，每一列都属于一种数据类型，Vastbase 根据数据类型有相应函数对其内容进行操作，例如 Vastbase 可对数值型数据进行加、减、乘、除操作。

13.3. 数据类型转换

数据库中允许有些数据类型进行隐式类型转换（赋值、函数调用的参数等），有些数据类型间不允许进行隐式数据类型转换，可尝试使用 Vastbase 提供的类型转换函数，例如 CAST 进行数据类型强转。

Vastbase 数据库常见的隐式类型转换，请参见表 13-1。

须知

Vastbase 支持的 DATE 的效限范围是：公元前 4713 年到公元 294276 年。

表 13-1. 隐式类型转换表

原始数据类型	目标数据类型	备注
CHAR	VARCHAR2	-
CHAR	NUMBER	原数据必须由数字组成。
CHAR	DATE	原数据不能超出合法日期范围。
CHAR	RAW	-

原始数据类型	目标数据类型	备注
CHAR	CLOB	-
VARCHAR2	CHAR	-
VARCHAR2	NUMBER	原数据必须由数字组成。
VARCHAR2	DATE	原数据不能超出合法日期范围。
VARCHAR2	CLOB	-
NUMBER	CHAR	-
NUMBER	VARCHAR2	-
DATE	CHAR	-
DATE	VARCHAR2	-
RAW	CHAR	-
RAW	VARCHAR2	-
CLOB	CHAR	-
CLOB	VARCHAR2	-
CLOB	NUMBER	原数据必须由数字组成。
INT4	CHAR	-

13.4. 数组和 record

13.4.1. 数组

数组类型的使用

在使用数组之前，需要自定义一个数组类型。

在存储过程中紧跟 AS 关键字后面定义数组类型。定义方法为：

```
TYPE array_type IS VARRAY(size) OF data_type;
```

其中：

- array_type：要定义的数组类型名。
- VARRAY：表示要定义的数组类型。
- size：取值为正整数，表示可以容纳的成员的最大数量。
- data_type：要创建的数组中成员的类型。

说明

- 在 Vastbase 中，数组会自动增长，访问越界会返回一个 NULL，不会报错。

- 在存储过程中定义的数组类型，其作用域仅在该存储过程中。
- 建议选择上述定义方法的一种来自定义数组类型，当同时使用两种方法定义同名的数组类型时，Vastbase 会优先选择存储过程中定义的数组类型来声明数组变量。

Vastbase 支持使用圆括号来访问数组元素，且还支持一些特有的函数，如 extend，count，first，last 来访问数组的内容。

说明

存储过程中如果有 DML 语句（SELECT、UPDATE、INSERT、DELETE），DML 语句只能使用中括号来访问数组元素，从而和函数表达式区分开。

13.4.2. record

record 类型的变量

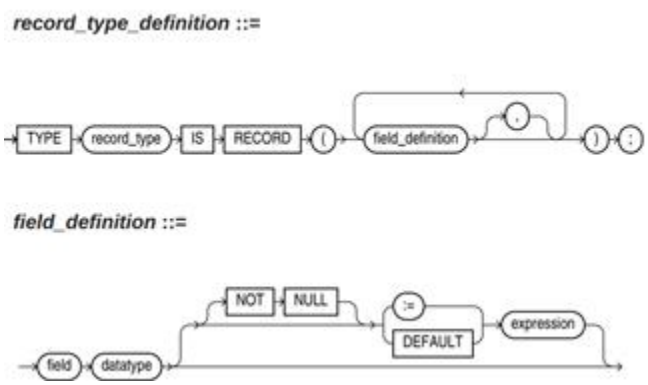
创建一个 record 变量的方式：

定义一个 record 类型 ，然后使用该类型来声明一个变量。

语法

record 类型的语法参见图 13-1。

图 13-2. record 类型的语法



对以上语法格式的解释如下：

- record_type：声明的类型名称。
- field：record 类型中的成员名称。
- datatype：record 类型中成员的类型。
- expression：设置默认值的表达式。

说明

在 Vastbase 中：

- record 类型变量的赋值支持：

- 在函数或存储过程的声明阶段，声明一个 record 类型，并且可以在该类型中定义成员变量。
- 一个 record 变量到另一个 record 变量的赋值。
- SELECT INTO 和 FETCH 向一个 record 类型的变量中赋值。
- 将一个 NULL 值赋值给一个 record 变量。
- 不支持 INSERT 和 UPDATE 语句使用 record 变量进行插入数据和更新数据。
- 如果成员有复合类型，在声明阶段不支持指定默认值，该行为同声明阶段的变量一样。

示例

```
下面存储过程中用到的表定义如下:
vastbase=# \d emp_rec
             Table "public.emp_rec"
  Column  |            Type             | Modifiers
----------+-----------------------------+-----------
 empno    | numeric(4,0)                | not null
 ename    | character varying(10)       |
 job      | character varying(9)        |
 mgr      | numeric(4,0)                |
 hiredate | timestamp(0) without time zone |
 sal      | numeric(7,2)                |
 comm     | numeric(7,2)                |
 deptno   | numeric(2,0)                |

--演示在存储过程中对数组进行操作。
vastbase=# CREATE OR REPLACE FUNCTION regress_record(p_w VARCHAR2)
RETURNS
VARCHAR2  AS $$
DECLARE

  --声明一个 record 类型.
  type rec_type is record (name  varchar2(100), epno int);
  employer rec_type;

  --使用%type 声明 record 类型
  type rec_type1 is record (name  emp_rec.ename%type, epno int not null :=10);
  employer1 rec_type1;

  --声明带有默认值的 record 类型
  type rec_type2 is record (
       name varchar2 not null := 'SCOTT',
       epno int not null :=10);
   employer2 rec_type2;
   CURSOR C1 IS  select ename,empno from emp_rec order by 1 limit 1;

BEGIN
     --对一个 record 类型的变量的成员赋值。
    employer.name := 'WARD';
    employer.epno = 18;
    raise info 'employer name: % , epno:%', employer.name, employer.epno;

    --将一个 record 类型的变量赋值给另一个变量。
    employer1 := employer;
```

```
    raise info 'employer1 name: % , epno: %',employer1.name, employer1.epno;

    --将一个 record 类型变量赋值为 NULL。
    employer1 := NULL;
    raise info 'employer1 name: % , epno: %',employer1.name, employer1.epno;

    --获取 record 变量的默认值。
    raise info 'employer2 name: % ,epno: %', employer2.name, employer2.epno;

    --在 for 循环中使用 record 变量
    for employer in select ename,empno from emp_rec order by 1  limit 1
       loop
           raise info 'employer name: % , epno: %', employer.name, employer.epno;
       end loop;

    --在 select into 中使用 record 变量。
    select ename,empno  into employer2 from emp_rec order by 1 limit 1;
    raise info 'employer name: % , epno: %', employer2.name, employer2.epno;

    --在 cursor 中使用 record 变量。
    OPEN C1;
    FETCH C1 INTO employer2;
    raise info 'employer name: % , epno: %', employer2.name, employer2.epno;
    CLOSE C1;
    RETURN employer.name;
END;
$$
LANGUAGE plpgsql;

--调用该存储过程。
vastbase=# CALL regress_record('abc');

--删除存储过程。
vastbase=# DROP PROCEDURE regress_record;
```

13.5. 声明语法

13.5.1. 基本结构

结构

PL/SQL 块中可以包含子块，子块可以位于 PL/SQL 中任何部分。PL/SQL 块的结构如下：

- 声明部分：声明 PL/SQL 用到的变量、类型、游标、局部的存储过程和函数。

```
DECLARE
```

说明

不涉及变量声明时声明部分可以没有。

- 对匿名块来说，没有变量声明部分时，可以省去 DECLARE 关键字。
- 对存储过程来说，没有 DECLARE， AS 相当于 DECLARE。即便没有变量声明的部分，关键字 AS 也必须保留。

- 执行部分：过程及 SQL 语句，程序的主要部分。必选。

```
BEGIN
```

- 执行异常部分：错误处理。可选。

```
EXCEPTION
```

- 结束

```
END;
/
```

须知

禁止在 PL/SQL 块中使用连续的 Tab，连续的 Tab 可能会造成在使用 vsql 工具带“-r”参数执行 PL/SQL 块时出现异常。

分类

PL/SQL 块可以分为以下几类：

- 匿名块：动态构造，只能执行一次。语法请参考图 13-2。
- 子程序：存储在数据库中的存储过程、函数、操作符和高级包等。当在数据库上建立好后，可以在其他程序中调用它们。

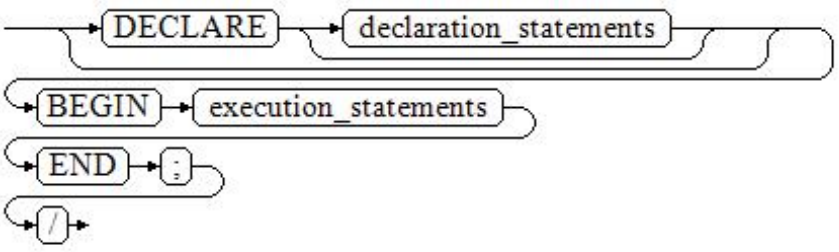
13.5.2. 匿名块

匿名块（Anonymous Block）一般用于不频繁执行的脚本或不重复进行的活动。它们在一个会话中执行，并不被存储。

语法

匿名块的语法参见图 13-2。

图 13-3. anonymous_block::=



对以上语法图的解释如下：

- 匿名块程序实施部分，以 BEGIN 语句开始，以 END 语句停顿，以一个分号结束。输入“/”按回车执行它。

须知

最后的结束符“/”必须独占一行，不能直接跟在 END 后面。

- 声明部分包括变量定义、类型、游标定义等。
- 最简单的匿名块不执行任何命令。但一定要在任意实施块里至少有一个语句，甚至是一个 NULL 语句。

13.5.3. 子程序

存储在数据库中的存储过程、函数、操作符和高级包等。当在数据库上建立好后，可以在其他程序中调用它们。

13.6. 基本语句

在编写 PL/SQL 过程中，会定义一些变量，给变量赋值，调用其他存储过程等。介绍 PL/SQL 中的基本语句，包括定义变量、赋值语句、调用语句以及返回语句。

说明

尽量不要在存储过程中调用包含密码的 SQL 语句，因为存储在数据库中的存储过程文本可能被其他有权限的用户看到导致密码信息被泄漏。如果存储过程中包含其他敏感信息也需要配置存储过程的访问权限，保证敏感信息不会泄漏。

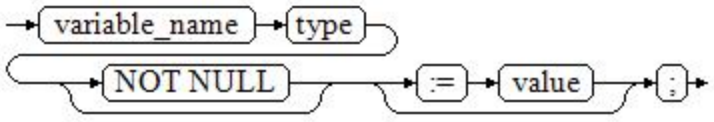
13.6.1. 定义变量

介绍 PL/SQL 中变量的声明，以及该变量在代码中的作用域。

变量声明

变量声明语法请参见图 13-3。

图 13-4. declare_variable::=



对以上语法格式的解释如下：

- variable_name：变量名。
- type：变量类型。
- value：该变量的初始值（如果不给定初始值，则初始为 NULL）。value 也可以是表达式。

示例

```
vastbase=# DECLARE
   emp_id  INTEGER := 7788;  --定义变量并赋值
BEGIN
   emp_id := 5*7784;  --变量赋值
END;
/
```

变量类型除了支持基本类型，还可以是使用%TYPE 和%ROWTYPE 去声明一些与其他表字段或表结构本身相关的变量。

%TYPE 属性

%TYPE 主要用于声明某个与其他变量类型（例如，表中某列的类型）相同的变量。假如我们想定义一个 my_name 变量，它的变量类型与 employee 的 firstname 类型相同，我们可以通过如下定义：

```
my_name employee.firstname%TYPE
```

这样定义可以带来两个好处，首先，我们不用预先知道 employee 表的 firstname 类型具体是什么。其次，即使之后 firstname 类型有了变化，我们也不需要再次修改 my_name 的类型。

%ROWTYPE 属性

%ROWTYPE 属性主要用于对一组数据的类型声明，用于存储表中的一行数据或从游标匹配的结果。假如，我们需要一组数据，该组数据的字段名称与字段类型都与 employee 表相同。我们可以通过如下定义：

```
my_employee employee%ROWTYPE
```

变量作用域

变量的作用域表示变量在代码块中的可访问性和可用性。只有在它的作用域内，变量才有效。

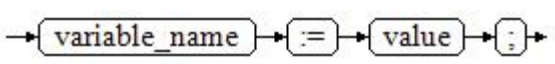
- 变量必须在 declare 部分声明，即必须建立 BEGIN-END 块。块结构也强制变量必须先声明后使用，即变量在过程内有不同作用域、不同的生存期。
- 同一变量可以在不同的作用域内定义多次，内层的定义会覆盖外层的定义。
- 在外部块定义的变量，可以在嵌套块中使用。但外部块不能访问嵌套块中的变量。

13.6.2. 赋值语句

语法

给变量赋值的语法请参见图 13-4。

图 13-5. assignment_value::=



对以上语法格式的解释如下：

- variable_name：变量名。

- value：可以是值或表达式。值 value 的类型需要和变量 variable_name 的类型兼容才能正确赋值。

示例

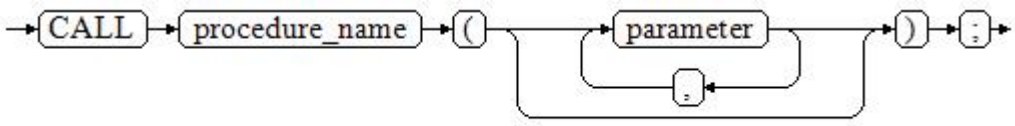
```
vastbase=# DECLARE
   emp_id  INTEGER := 7788;--赋值
BEGIN
   emp_id := 5;--赋值
   emp_id := 5*7784;
END;
/
```

13.6.3. 调用语句

语法

调用一个语句的语法请参见图 13-5。

图 13-6. call_clause::=



对以上语法格式的解释如下：

- procedure_name：存储过程名。
- parameter：存储过程的参数，可以没有或者有多个参数。

示例

```
--创建表
create table staffs(
section_id int,
salary int
);
insert into staffs values(1,1000);
insert into staffs values(2,1000);
insert into staffs values(3,1000);
insert into staffs values(4,1000);
insert into staffs values(5,1000);
insert into staffs values(6,1000);
insert into staffs values(7,1000);
insert into staffs values(8,1000);

--创建存储过程 proc_staffs
vastbase=# CREATE OR REPLACE PROCEDURE proc_staffs
(
```

```
section    NUMBER(6),
salary_sum out NUMBER(8,2),
staffs_count out INTEGER
)
IS
BEGIN
SELECT sum(salary), count(*) INTO salary_sum, staffs_count FROM staffs where section_id = section;
END;
/

--调用存储过程proc_return.
vastbase=# CALL proc_staffs(2,8,6);

--清除存储过程
vastbase=# DROP PROCEDURE proc_staffs;
```

13.7. 动态语句

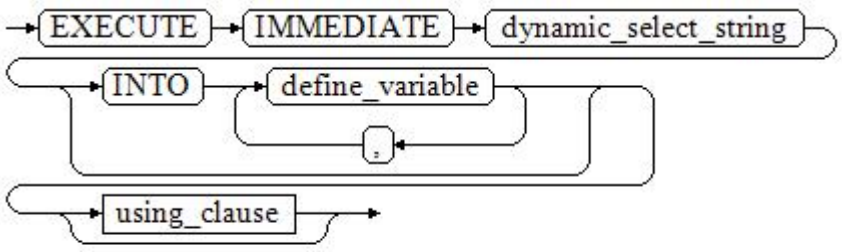
13.7.1. 执行动态查询语句

介绍执行动态查询语句。Vastbase 提供两种方式：使用 EXECUTE IMMEDIATE、OPEN FOR 实现动态查询。前者通过动态执行 SELECT 语句，后者结合了游标的使用。当需要将查询的结果保存在一个数据集用于提取时，可使用 OPEN FOR 实现动态查询。

EXECUTE IMMEDIATE

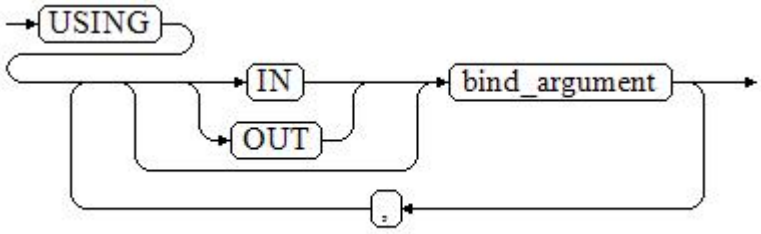
语法图请参见图 13-6。

图 13-7. EXECUTE IMMEDIATE dynamic_select_clause::=



using_clause 子句的语法图参见图 13-7。

图 13-8.　using_clause::=



对以上语法格式的解释如下：

- define_variable：用于指定存放单行查询结果的变量。
- USING IN bind_argument：用于指定存放传递给动态 SQL 值的变量，即在 dynamic_select_string 中存在占位符时使用。
- USING OUT bind_argument：用于指定存放动态 SQL 返回值的变量。

须知

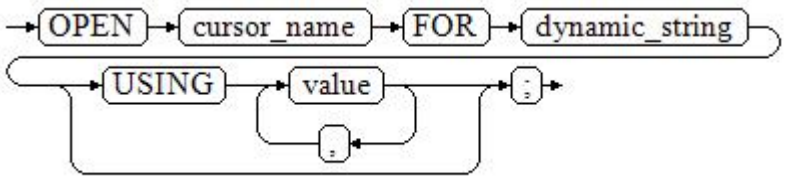
- 查询语句中，into 和 out 不能同时存在；
- 占位符命名以“:”开始，后面可跟数字、字符或字符串，与 USING 子句的 bind_argument 一一对应；
- bind_argument 只能是值、变量或表达式，不能是表名、列名、数据类型等数据库对象，即不支持使用 bind_argument 为动态 SQL 语句传递模式对象。如果存储过程需要通过声明参数传递数据库对象来构造动态 SQL 语句（常见于执行 DDL 语句时），建议采用连接运算符“||”拼接 dynamic_select_clause；
- 动态 PL/SQL 块允许出现重复的占位符，即相同占位符只能与 USING 子句的一个 bind_argument 按位置对应。

OPEN FOR

动态查询语句还可以使用 OPEN FOR 打开动态游标来执行。

语法参见图 13-8。

图 13-9.　open_for::=



参数说明：

- cursor_name：要打开的游标名。
- dynamic_string：动态查询语句。
- USING value：在 dynamic_string 中存在占位符时使用。

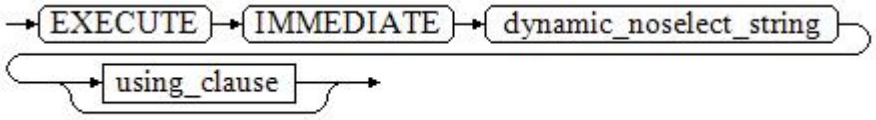
游标的使用请参考 13.11 游标。

13.7.2. 执行动态非查询语句

语法

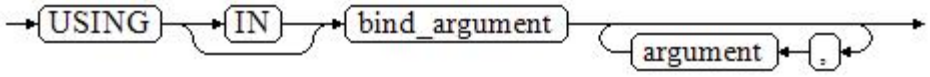
语法请参见图 13-9。

图 13-10. noselect::=



using_clause 子句的语法参见图 13-10。

图 13-11. using_clause::=



对以上语法格式的解释如下：

USING IN bind_argument 用于指定存放传递给动态 SQL 值的变量，在 dynamic_noselect_string 中存在占位符时使用，即动态 SQL 语句执行时，bind_argument 将替换相对应的占位符。要注意的是，bind_argument 只能是值、变量或表达式，不能是表名、列名、数据类型等数据库对象。如果存储过程需要通过声明参数传递数据库对象来构造动态 SQL 语句（常见于执行 DDL 语句时），建议采用连接运算符“||”拼接 dynamic_select_clause。另外，动态语句允许出现重复的占位符，相同占位符只能与唯一一个 bind_argument 按位置一一对应。

示例

```
--创建表
vastbase=# CREATE TABLE sections_t1
(
  section      NUMBER(4) ,
  section_name  VARCHAR2(30),
  manager_id    NUMBER(6),
  place_id      NUMBER(4)
);

--声明变量
vastbase=# DECLARE
  section      NUMBER(4) := 280;
  section_name  VARCHAR2(30) := 'Info support';
  manager_id    NUMBER(6) := 103;
  place_id      NUMBER(4) := 1400;
  new_colname   VARCHAR2(10) := 'sec_name';
```

```
BEGIN
--执行查询
   EXECUTE IMMEDIATE 'insert into sections_t1 values(:1, :2, :3, :4)'
      USING section, section_name, manager_id,place_id;
--执行查询（重复占位符）
   EXECUTE IMMEDIATE 'insert into sections_t1 values(:1, :2, :3, :1)'
      USING section, section_name, manager_id;
--执行 ALTER 语句（建议采用“||”拼接数据库对象构造 DDL 语句）
   EXECUTE IMMEDIATE 'alter table sections_t1 rename section_name to ' || new_colname;
END;
/

--查询数据
vastbase=# SELECT * FROM sections_t1;

--删除表
vastbase=# DROP TABLE sections_t1;
```

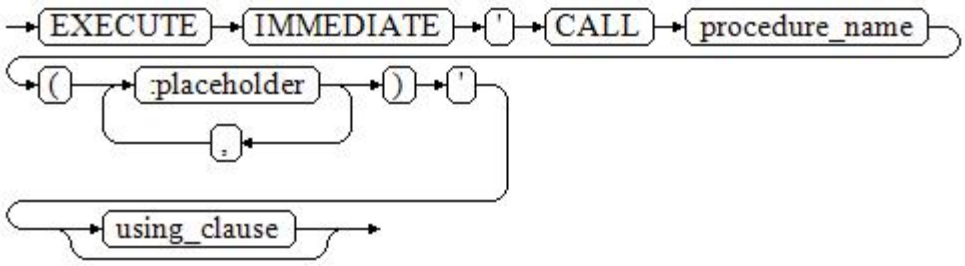
13.7.3. 动态调用存储过程

动态调用存储过程必须使用匿名的语句块将存储过程或语句块包在里面，使用 EXECUTE IMMEDIATE...USING 语句后面带 IN、OUT 来输入、输出参数。

语法

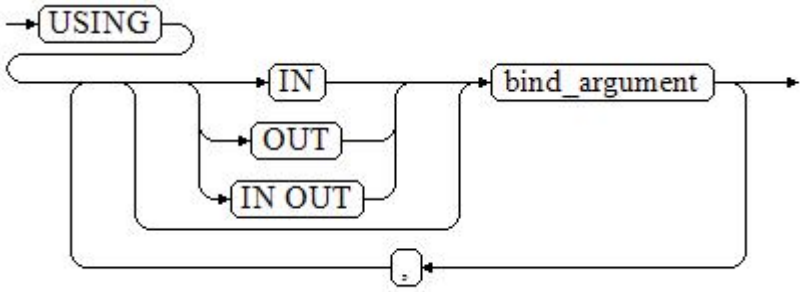
语法请参见图 13-11。

图 13-12. call_procedure::=



using_clause 子句的语法参见图 13-12。

图 13-13. using_clause::=



对以上语法格式的解释如下：

- CALL procedure_name: 调用存储过程。
- [:placeholder1，:placeholder2，...]: 存储过程参数占位符列表。占位符个数与参数个数相同。
- USING [IN|OUT|IN OUT] bind_argument: 用于指定存放传递给存储过程参数值的变量。bind_argument 前的修饰符与对应参数的修饰符一致。

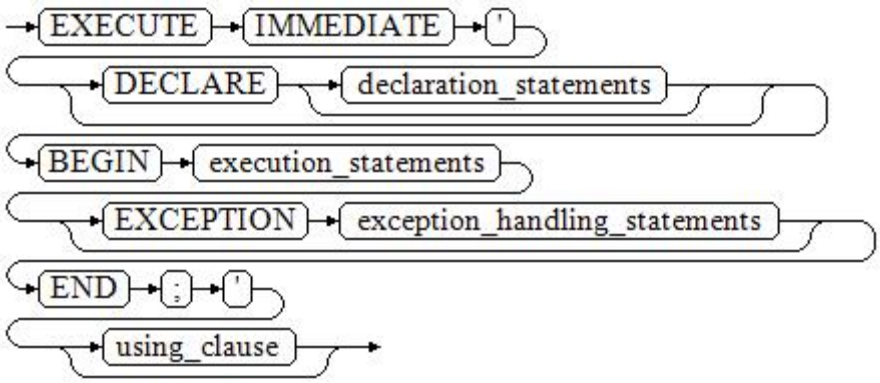
13.7.4. 动态调用匿名块

动态调用匿名块是指在动态语句中执行匿名块，使用 EXECUTE IMMEDIATE...USING 语句后面带 IN、OUT 来输入、输出参数。

语法

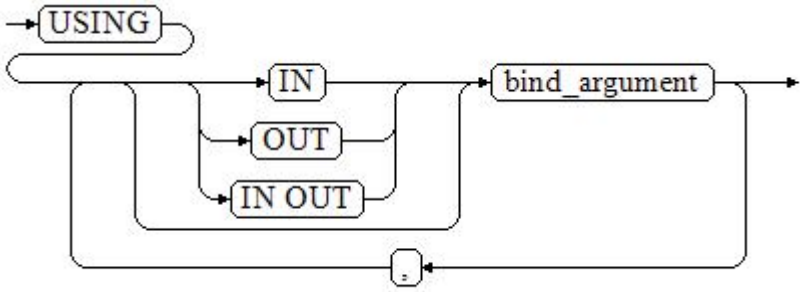
语法请参见图 13-13。

图 13-14. call_anonymous_block::=



using_clause 子句的语法参见图 13-14。

图 13-15. using_clause::=



对以上语法格式的解释如下：

- 匿名块程序实施部分，以 BEGIN 语句开始，以 END 语句停顿，以一个分号结束。
- USING [IN|OUT|IN OUT] bind_argument，用于指定存放传递给存储过程参数值的变量。bind_argument 前的修饰符与对应参数的修饰符一致。
- 匿名块中间的输入输出参数使用占位符来指明，要求占位符个数与参数个数相同，并且占位符所对应参数的顺序和 USING 中参数的顺序一致。
- 目前 Vastbase 在动态语句调用匿名块时，EXCEPTION 语句中暂不支持使用占位符进行输入输出参数的传递。

13.8. commit/rollback

在 plpgsql 定义存储过程中，增加两条指令：commit 和 rollback。来控制对当前子事务的提交或是回滚。

示例

```
vastbase=# create table test_pro(id int,a text,c char(20));
CREATE TABLE
vastbase=# create or replace procedure test_pro1 is
vastbase$# begin
vastbase$# insert into test_pro values(1,'b1','c1');
vastbase$# commit;
vastbase$# insert into test_pro values(2,'b2','b3');
vastbase$# insert into test_pro values(3,'b3','b3');
vastbase$# insert into test_pro values(4,'b4','b4');
vastbase$# insert into test_pro values(5,'b5','b3');
vastbase$# insert into test_pro values(6,'b6','b6');
vastbase$# commit;
vastbase$# delete from test_pro where id =2;
vastbase$# rollback;
vastbase$# delete from test_pro where id =2;
vastbase$# commit;
vastbase$# end;
vastbase$# /
```

```
CREATE PROCEDURE
vastbase=# call test_pro1();
 test_pro1
-------------
(一行记录)
vastbase=# select * from test_pro;
```

13.9. 控制语句

13.9.1. 返回语句

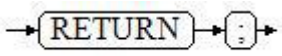
Vastbase 提供两种方式返回数据：RETURN 或 RETURN NEXT 及 RETURN QUERY。其中，RETURN NEXT 和 RETURN QUERY 只适用于函数，不适用存储过程。

13.9.1.1. RETURN

语法

返回语句的语法请参见图 13-15。

图 13-16. return_clause::=



对以上语法的解释如下：

用于将控制从存储过程或函数返回给调用者。

示例

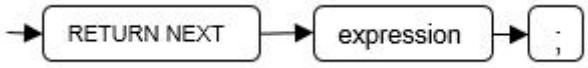
请参见调用语句的示例。

13.9.1.2. RETURN NEXT 及 RETURN QUERY

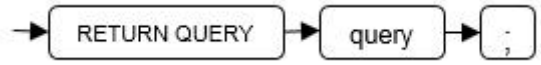
语法

创建函数时需要指定返回值 SETOF datatype。

return_next_clause::=



return_query_clause::=



对以上语法的解释如下：

当需要函数返回一个集合时，使用 RETURN NEXT 或者 RETURN QUERY 向结果集追加结果，然后继续执行函数的下一条语句。随着后续的 RETURN NEXT 或 RETURN QUERY 命令的执行，结果集中会有多个结果。函数执行完成后会一起返回所有结果。

RETURN NEXT 可用于标量和复合数据类型。

RETURN QUERY 有一种变体 RETURN QUERY EXECUTE，后面还可以增加动态查询，通过 USING 向查询插入参数。

示例

```
vastbase=# CREATE TABLE t1(a int);
vastbase=# INSERT INTO t1 VALUES(1),(10);

--RETURN NEXT
vastbase=# CREATE OR REPLACE FUNCTION fun_for_return_next() RETURNS SETOF t1 AS $$
DECLARE
  r t1%ROWTYPE;
BEGIN
  FOR r IN select * from t1
  LOOP
    RETURN NEXT r;
  END LOOP;
  RETURN;
END;
$$ LANGUAGE PLPgsql;
vastbase=# call fun_for_return_next();
 a
---
 1
 10
(2 rows)

-- RETURN QUERY
vastbase=# CREATE OR REPLACE FUNCTION fun_for_return_query() RETURNS SETOF t1 AS $$
DECLARE
  r t1%ROWTYPE;
BEGIN
  RETURN QUERY select * from t1;
END;
$$
language plpgsql;
vastbase=# call fun_for_return_query();
 a
---
 1
 10
(2 rows)
```

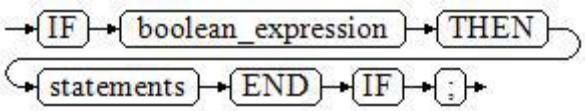
13.9.2. 条件语句

条件语句的主要作用判断参数或者语句是否满足已给定的条件，根据判定结果执行相应的操作。

Vastbase 有五种形式的 IF：

- IF_THEN

图 13-17. IF_THEN::=



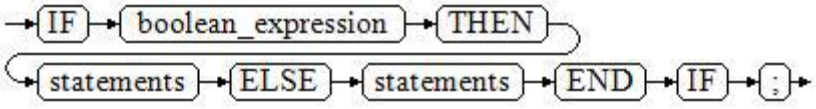
IF_THEN 语句是 IF 的最简单形式。如果条件为真，statements 将被执行。否则，将忽略它们的结果使该 IF_THEN 语句执行结束。

示例

```
vastbase=# IF v_user_id <> 0 THEN
   UPDATE users SET email = v_email WHERE user_id = v_user_id;
END IF;
```

❖ IF_THEN_ELSE

图 13-18. IF_THEN_ELSE::=



IF_THEN_ELSE 语句增加了 ELSE 的分支，可以声明在条件为假的时候执行的语句。

示例

```
vastbase=# IF parentid IS NULL OR parentid = ''
THEN
   RETURN;
ELSE
   hp_true_filename(parentid);--表示调用存储过程
END IF;
```

❖ IF_THEN_ELSE IF

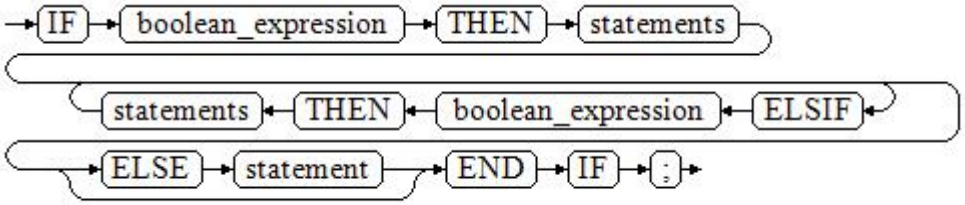
IF 语句可以嵌套，嵌套方式如下：

```
vastbase=# IF sex = 'm' THEN
   pretty_sex := 'man';
ELSE
   IF sex = 'f' THEN
      pretty_sex := 'woman';
   END IF;
END IF;
```

这种形式实际上就是在一个 IF 语句的 ELSE 部分嵌套了另一个 IF 语句。因此需要一个 END IF 语句给每个嵌套的 IF，另外还需要一个 END IF 语句结束父 IF-ELSE。如果有多个选项，可使用下面的形式。

❖ IF_THEN_ELSIF_ELSE

图 13-19. IF_THEN_ELSIF_ELSE::=



示例

```
IF number_tmp = 0 THEN
   result := 'zero';
ELSIF number_tmp > 0 THEN
   result := 'positive';
ELSIF number_tmp < 0 THEN
   result := 'negative';
ELSE
   result := 'NULL';
END IF;
```

- IF_THEN_ELSEIF_ELSE

ELSEIF 是 ELSIF 的别名。

综合示例

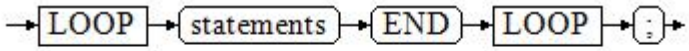
```
CREATE OR REPLACE PROCEDURE proc_control_structure(i in integer)
AS
   BEGIN
      IF i > 0 THEN
         raise info 'i:% is greater than 0. ',i;
      ELSIF i < 0 THEN
         raise info 'i:% is smaller than 0. ',i;
      ELSE
         raise info 'i:% is equal to 0. ',i;
      END IF;
      RETURN;
   END;
/

CALL proc_control_structure(3);

--删除存储过程
DROP PROCEDURE proc_control_structure;
```

13.9.3. 循环语句

简单 LOOP 语句

语法图

图 13-20. loop::=



示例

```
CREATE OR REPLACE PROCEDURE proc_loop(i in integer, count out integer)
AS
   BEGIN
      count:=0;
      LOOP
      IF count > i THEN
         raise info 'count is %. ', count;
         EXIT;
      ELSE
         count:=count+1;
      END IF;
      END LOOP;
   END;
/

CALL proc_loop(10,5);
```

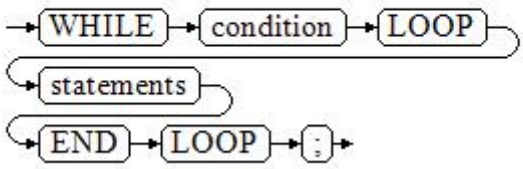
须知

该循环必须要结合 EXIT 使用，否则将陷入死循环。

WHILE_LOOP 语句

语法图

图 13-21. while_loop::=



只要条件表达式为真，WHILE 语句就会不停的在一系列语句上进行循环，在每次进入循环体的时候进行条件判断。

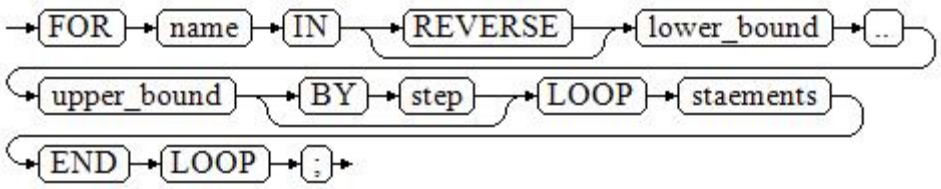
示例

```
CREATE TABLE integertable(c1 integer) ;
CREATE OR REPLACE PROCEDURE proc_while_loop(maxval in integer)
AS
   DECLARE
   i int :=1;
   BEGIN
      WHILE i < maxval LOOP
         INSERT INTO integertable VALUES(i);
         i:=i+1;
      END LOOP;
   END;
/

--调用函数
CALL proc_while_loop(10);

--删除存储过程和表
DROP PROCEDURE proc_while_loop;
DROP TABLE integertable;
```

FOR_LOOP（integer 变量）语句

语法图

图 13-22. for_loop::=



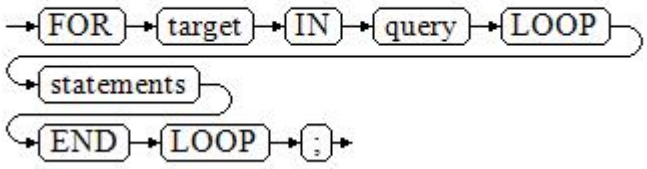
说明

- 变量 name 会自动定义为 integer 类型并且只在此循环里存在。变量 name 介于 lower_bound 和 upper_bound 之间。
- 当使用 REVERSE 关键字时，lower_bound 必须大于等于 upper_bound，否则循环体不会被执行。

FOR_LOOP 查询语句

语法图

图 13-23. for_loop_query::=



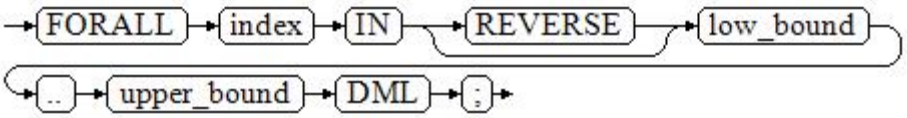
说明

变量 target 会自动定义，类型和 query 的查询结果的类型一致，并且只在此循环中有效。target 的取值就是 query 的查询结果。

FORALL 批量查询语句

语法图

图 13-24. forall::=



说明

变量 index 会自动定义为 integer 类型并且只在此循环里存在。index 的取值介于 low_bound 和 upper_bound 之间。

示例

```
CREATE TABLE hdfs_t1 (
  title NUMBER(6),
  did VARCHAR2(20),
  data_peroid VARCHAR2(25),
  kind VARCHAR2(25),
  interval VARCHAR2(20),
  time DATE,
  isModified VARCHAR2(10)
);

INSERT INTO hdfs_t1 VALUES( 8, 'Donald', 'OConnell', 'DOCONNEL', '650.507.9833', to_date('21-06-1999',
'dd-mm-yyyy'), 'SH_CLERK' );

CREATE OR REPLACE PROCEDURE proc_forall()
AS
BEGIN
   FORALL i IN 100..120
      update hdfs_t1 set title = title + 100*i;
END;
/

--调用函数
CALL proc_forall();

--查询存储过程调用结果
SELECT * FROM hdfs_t1 WHERE title BETWEEN 100 AND 120;
```

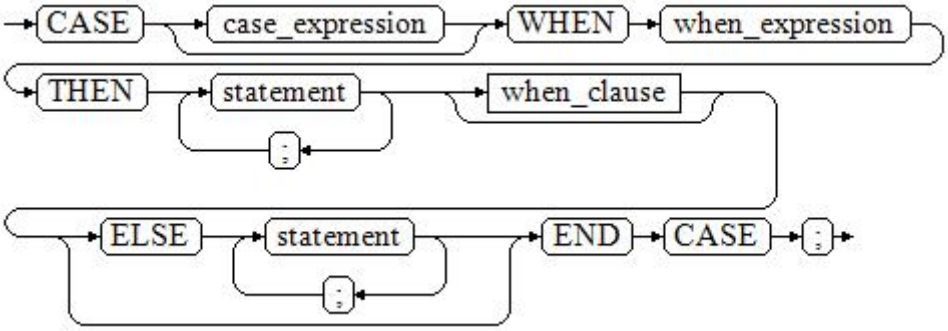
```
--删除存储过程和表
DROP PROCEDURE proc_forall;
DROP TABLE hdfs_t1;
```

13.9.4. 分支语句

语法

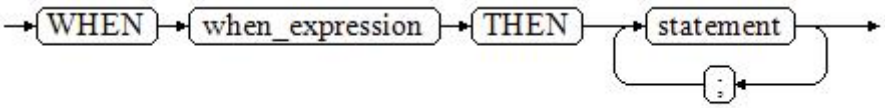
分支语句的语法请参见图 13-24。

图 13-25. case_when::=



when_clause 子句的语法图参见图 13-25。

图 13-26. when_clause::=



参数说明：

- case_expression：变量或表达式。
- when_expression：常量或者条件表达式。
- statement：执行语句。

示例

```
CREATE OR REPLACE PROCEDURE proc_case_branch(pi_result in integer, pi_return out integer)
AS
   BEGIN
      CASE pi_result
         WHEN 1 THEN
            pi_return := 111;
         WHEN 2 THEN
            pi_return := 222;
         WHEN 3 THEN
```

```
            pi_return := 333;
         WHEN 6 THEN
            pi_return := 444;
         WHEN 7 THEN
            pi_return := 555;
         WHEN 8 THEN
            pi_return := 666;
         WHEN 9 THEN
            pi_return := 777;
         WHEN 10 THEN
            pi_return := 888;
         ELSE
            pi_return := 999;
      END CASE;
      raise info 'pi_return : %',pi_return ;
END;
/

CALL proc_case_branch(3,0);

--删除存储过程
DROP PROCEDURE proc_case_branch;
```

13.9.5. 空语句

在 PL/SQL 程序中，可以用 NULL 语句来说明“不用做任何事情”，相当于一个占位符，可以使某些语句变得有意义，提高程序的可读性。

语法

空语句的用法如下：

```
DECLARE
   …
BEGIN
   …
   IF v_num IS NULL THEN
      NULL; -- 不需要处理任何数据。
   END IF;
END;
/
```

13.9.6. 错误捕获语句

缺省时，当 PL/SQL 函数执行过程中发生错误时退出函数执行，并且周围的事务也会回滚。可以用一个带有 EXCEPTION 子句的 BEGIN 块捕获错误并且从中恢复。其语法是正常的 BEGIN 块语法的一个扩展：

```
[<<label>>]
[DECLARE
   declarations]
BEGIN
   statements
```

```
EXCEPTION
   WHEN condition [OR condition ...] THEN
      handler_statements
   [WHEN condition [OR condition ...] THEN
      handler_statements
   ...]
END;
```

如果没有发生错误，这种形式的块儿只是简单地执行所有语句，然后转到 END 之后的下一个语句。但是如果在执行的语句内部发生了一个错误，则这个语句将会回滚，然后转到 EXCEPTION 列表。寻找匹配错误的第一个条件。若找到匹配，则执行对应的 handler_statements，然后转到 END 之后的下一个语句。如果没有找到匹配，则会向事务的外层报告错误，和没有 EXCEPTION 子句一样。

也就是说该错误可以被一个包围块用 EXCEPTION 捕获，如果没有包围块，则进行退出函数处理。

condition 的名称可以是 SQL 标准错误码编号说明的任意值。特殊的条件名 OTHERS 匹配除了 QUERY_CANCELED 之外的所有错误类型。

如果在选中的 handler_statements 里发生了新错误，则不能被这个 EXCEPTION 子句捕获，而是向事务的外层报告错误。一个外层的 EXCEPTION 子句可以捕获它。

如果一个错误被 EXCEPTION 捕获，PL/SQL 函数的局部变量保持错误发生时的原值，但是所有该块中想写入数据库中的状态都回滚。

示例：

```
CREATE TABLE mytab(id INT,firstname VARCHAR(20),lastname VARCHAR(20)) ;

INSERT INTO mytab(firstname, lastname) VALUES('Tom', 'Jones');

CREATE FUNCTION fun_exp() RETURNS INT
AS $$
DECLARE
   x INT :=0;
   y INT;
BEGIN
   UPDATE mytab SET firstname = 'Joe' WHERE lastname = 'Jones';
   x := x + 1;
   y := x / 0;
EXCEPTION
   WHEN division_by_zero THEN
      RAISE NOTICE 'caught division_by_zero';
      RETURN x;
END;$$
LANGUAGE plpgsql;

call fun_exp();
NOTICE:  caught division_by_zero
 fun_exp
---------
       1
(1 row)

select * from mytab;
 id | firstname | lastname
----+-----------+----------
    | Tom       | Jones
(1 row)

DROP FUNCTION fun_exp();
DROP TABLE mytab;
```

当控制到达给 y 赋值的地方时，会有一个 division_by_zero 错误失败。这个错误将被 EXCEPTION 子句捕获。而在 RETURN 语句里返回的数值将是 x 的增量值。

说明

进入和退出一个包含 EXCEPTION 子句的块要比不包含的块开销大的多。因此，不必要的时候不要使用 EXCEPTION。

在下列场景中，无法捕获处理异常，整个存储过程回滚：节点故障、网络故障引起的存储过程参与节点线程退出以及 COPY FROM 操作中源数据与目标表的表结构不一致造成的异常。

示例 ： UPDATE/INSERT 异常

这个例子根据使用异常处理器执行恰当的 UPDATE 或 INSERT 。

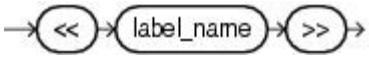
```
CREATE TABLE db (a INT, b TEXT);

CREATE FUNCTION merge_db(key INT, data TEXT) RETURNS VOID AS
$$
BEGIN
    LOOP

--第一次尝试更新 key
       UPDATE db SET b = data WHERE a = key;
       IF found THEN
          RETURN;
       END IF;
--不存在，所以尝试插入 key，如果其他人同时插入相同的 key，我们可能得到唯一 key 失败。
       BEGIN
          INSERT INTO db(a,b) VALUES (key, data);
          RETURN;
       EXCEPTION WHEN unique_violation THEN
       --什么也不做，并且循环尝试再次更新。
       END;
    END LOOP;
END;
$$
LANGUAGE plpgsql;

SELECT merge_db(1, 'david');
SELECT merge_db(1, 'dennis');

--删除 FUNCTION 和 TABLE
DROP FUNCTION merge_db;
DROP TABLE db ;
```

13.9.7. GOTO 语句

GOTO 语句可以实现从 GOTO 位置到目标语句的无条件跳转。GOTO 语句会改变原本的执行逻辑，因此应该慎重使用，或者也可以使用 EXCEPTION 处理特殊场景。当执行 GOTO 语句时，目标 Label 必须是唯一的。

语法

label declaration ::=



goto statement ::=

示例

```
vastbase=# CREATE OR REPLACE PROCEDURE GOTO_test()
AS
DECLARE
   v1  int;
BEGIN
   v1  := 0;
      LOOP
      EXIT WHEN v1 > 100;
             v1 := v1 + 2;
             if v1 > 25 THEN
                    GOTO pos1;
             END IF;
      END LOOP;
<<pos1>>
v1 := v1 + 10;
raise info 'v1 is %. ', v1;
END;
/

call GOTO_test();
```

限制场景

GOTO 使用有以下限制场景

- 不支持有多个相同的 GOTO labels 目标场景，无论是否在同一个 block 中。

```
BEGIN
 GOTO pos1;
 <<pos1>>
 SELECT * FROM ...
 <<pos1>>
 UPDATE t1 SET ...
END;
```

- 不支持 GOTO 跳转到 IF 语句，CASE 语句，LOOP 语句中。

```
BEGIN
  GOTO pos1;
  IF valid THEN
    <<pos1>>
    SELECT * FROM ...
  END IF;
 END;
```

- 不支持 GOTO 语句从一个 IF 子句跳转到另一个 IF 子句，或从一个 CASE 语句的 WHEN 子句跳转到另一个 WHEN 子句。

```
BEGIN
  IF valid THEN
    GOTO pos1;
    SELECT * FROM ...
  ELSE
    <<pos1>>
    UPDATE t1 SET ...
  END IF;
 END;
```

- 不支持从外部块跳转到内部的 BEGIN-END 块。

```
BEGIN
  GOTO pos1;
  BEGIN
    <<pos1>>
    UPDATE t1 SET ...
  END;
END;
```

- 不支持从异常处理部分跳转到当前的 BEGIN-END 块。但可以跳转到上层 BEGIN-END 块。

```
BEGIN
  <<pos1>>
  UPDATE t1 SET ...
  EXCEPTION
    WHEN condition THEN
      GOTO pos1;
END;
```

- 如果从 GOTO 到一个不包含执行语句的位置，需要添加 NULL 语句。

```
DECLARE
  done  BOOLEAN;
BEGIN
  FOR i IN 1..50 LOOP
    IF done THEN
      GOTO end_loop;
    END IF;
    <<end_loop>>  -- not allowed unless an executable statement follows
    NULL; -- add NULL statement to avoid error
  END LOOP;  -- raises an error without the previous NULL
END;
/
```

13.10. 其他语句

13.10.1.锁操作

Vastbase 提供了多种锁模式用于控制对表中数据的并发访问。这些模式可以用在 MVCC（多版本并发控制）无法给出期望行为的场合。同样，大多数 Vastbase 命令自动施加恰当的锁，以保证被引用的表在命令的执行过程中不会以一种不兼容的方式被删除或者修改。比如，在存在其他并发操作的时候，ALTER TABLE 是不能在同一个表上执行的。

13.10.2.游标操作

Vastbase 中游标（cursor）是系统为用户开设的一个数据缓冲区，存放着 SQL 语句的执行结果。每个游标区都有一个名称。用户可以用 SQL 语句逐一从游标中获取记录，并赋给主变量，交由主语言进一步处理。

游标的操作主要有游标的定义、打开、获取和关闭。

完整的游标操作示例可参考 13.11.2 显式游标。

13.11. 游标

13.11.1.游标概述

为了处理 SQL 语句，存储过程进程分配一段内存区域来保存上下文联系。游标是指向上下文区域的句柄或指针。借助游标，存储过程可以控制上下文区域的变化。

须知

当游标作为存储过程的返回值时，如果使用 JDBC 调用该存储过程，返回的游标将不可用。

游标的使用分为显式游标和隐式游标。对于不同的 SQL 语句，游标的使用情况不同，详细信息请参见表 13-2。

表 13-2. 游标使用情况

SQL 语句	游标
非查询语句	隐式的
结果是单行的查询语句	隐式的或显式的
结果是多行的查询语句	显式的

13.11.2.显式游标

显式游标主要用于对查询语句的处理，尤其是在查询结果为多条记录的情况下。

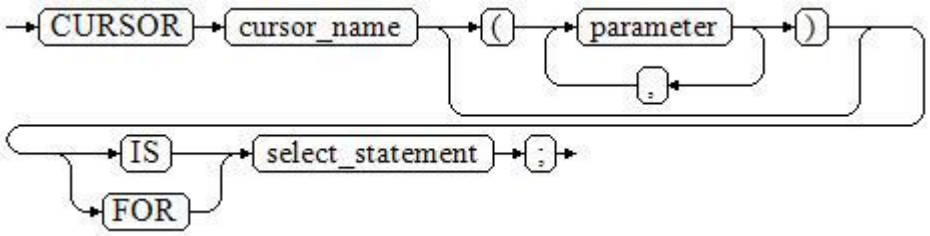
处理步骤

显式游标处理需六个 PL/SQL 步骤：

步骤 1 定义静态游标：就是定义一个游标名，以及与其相对应的 SELECT 语句。

定义静态游标的语法图，请参见图 13-26。

图 13-27. static_cursor_define::=



参数说明：

- cursor_name：定义的游标名。
- parameter：游标参数，只能为输入参数，其格式为：

```
parameter_name datatype
```

- select_statement：查询语句。

说明

根据执行计划的不同，系统会自动判断该游标是否可以用于以倒序的方式检索数据行。

定义动态游标：指 ref 游标，可以通过一组静态的 SQL 语句动态的打开游标。首先定义 ref 游标类型，然后定义该游标类型的游标变量，在打开游标时通过 OPEN FOR 动态绑定 SELECT 语句。

定义动态游标的语法图，请参见图 13-27 和图 13-28。

图 13-28. cursor_typename::=

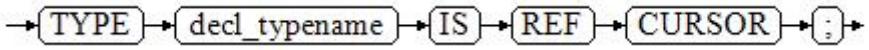
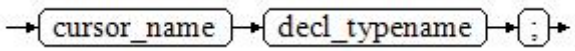


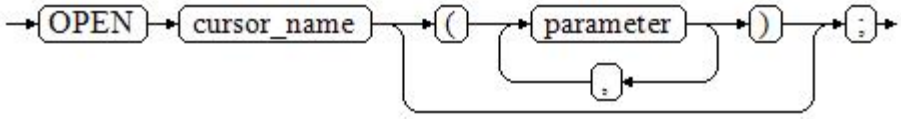
图 13-29. dynamic_cursor_define::=



步骤 2 打开静态游标：就是执行游标所对应的 SELECT 语句，将其查询结果放入工作区，并且指针指向工作区的首部，标识游标结果集合。如果游标查询语句中带有 FOR UPDATE 选项，OPEN 语句还将锁定数据库表中游标结果集合对应的数据行。

打开静态游标的语法图，请参见图 13-29。

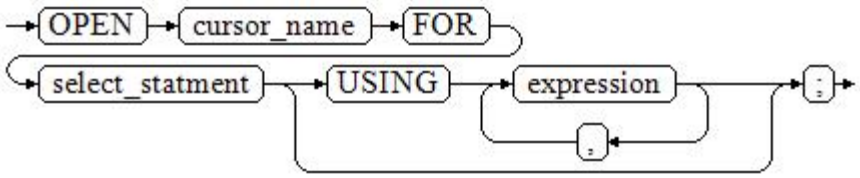
图 13-30. open_static_cursor::=



打开动态游标：可以通过 OPEN FOR 语句打开动态游标，动态绑定 SQL 语句。

打开动态游标的语法图，请参见图 13-30。

图 13-31. open_dynamic_cursor::=

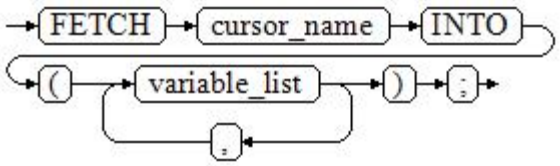


PL/SQL 程序不能用 OPEN 语句重复打开一个游标。

步骤 3 提取游标数据：检索结果集合中的数据行，放入指定的输出变量中。

提取游标数据的语法图，请参见图 13-31。

图 13-32. fetch_cursor::=



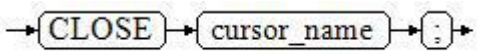
步骤 4 对该记录进行处理。

步骤 5 继续处理，直到活动集合中没有记录。

步骤 6 关闭游标：当提取和处理完游标结果集合数据后，应及时关闭游标，以释放该游标所占用的系统资源，并使该游标的工作区变成无效，不能再使用 FETCH 语句获取其中数据。关闭后的游标可以使用 OPEN 语句重新打开。

关闭游标的语法图，请参见图 13-32。

图 13-33. close_cursor::=



属性

游标的属性用于控制程序流程或者了解程序的状态。当运行 DML 语句时，PL/SQL 打开一个内建游标并处理结果，游标是维护查询结果的内存中的一个区域，游标在运行 DML 语句时打开，完成后关闭。显式游标的属性为：

- %FOUND 布尔型属性：当最近一次读记录时成功返回，则值为 TRUE。
- %NOTFOUND 布尔型属性：与%FOUND 相反。
- %ISOPEN 布尔型属性：当游标已打开时返回 TRUE。
- %ROWCOUNT 数值型属性：返回已从游标中读取的记录数。

13.11.3.隐式游标

对于非查询语句，如修改、删除操作，则由系统自动地为这些操作设置游标并创建其工作区，这些由系统隐含创建的游标称为隐式游标，隐式游标的名称为 SQL，这是由系统定义的。

简介

对于隐式游标的操作，如定义、打开、取值及关闭操作，都由系统自动地完成，无需用户进行处理。用户只能通过隐式游标的相关属性，来完成相应的操作。在隐式游标的工作区中，所存放的数据是最新处理的一条 SQL 语句所包含的数据，与用户自定义的显式游标无关。

格式调用为： SQL%

说明

INSERT，UPDATE，DELETE，SELECT 语句中不必明确定义游标。

属性

隐式游标属性为：

- SQL%FOUND 布尔型属性：当最近一次读记录时成功返回，则值为 TRUE。
- SQL%NOTFOUND 布尔型属性：与%FOUND 相反。
- SQL%ROWCOUNT 数值型属性：返回已从游标中读取得记录数。
- SQL%ISOPEN 布尔型属性：取值总是 FALSE。SQL 语句执行完毕立即关闭隐式游标。

示例

```
--创建模式 hr
CREATE SCHEMA hr;

--创建表 hr.staffs
create table hr.staffs(
section id int,
salary int
);
insert into hr.staffs values(1,1000);
```

```
insert into hr.staffs values(2,1000);
insert into hr.staffs values(3,1000);
insert into hr.staffs values(4,1000);
insert into hr.staffs values(5,1000);
insert into hr.staffs values(6,1000);
insert into hr.staffs values(7,1000);
insert into hr.staffs values(8,1000);

--创建表hr.sections
create table hr.sections(
section_id int,
salary int
);
insert into hr.sections values(1,1000);
insert into hr.sections values(2,1000);
insert into hr.sections values(3,1000);
insert into hr.sections values(4,1000);
insert into hr.sections values(5,1000);
insert into hr.sections values(6,1000);
insert into hr.sections values(7,1000);

--删除EMP表中某部门的所有员工，如果该部门中已没有员工，则在DEPT表中删除该部门。
CREATE TABLE hr.staffs_t1 AS TABLE hr.staffs;
CREATE TABLE hr.sections_t1 AS TABLE hr.sections;

CREATE OR REPLACE PROCEDURE proc_cursor3()
AS
   DECLARE
   V_DEPTNO NUMBER(4) := 100;
   BEGIN
      DELETE FROM hr.staffs WHERE section_ID = V_DEPTNO;
      --根据游标状态做进一步处理
      IF SQL%NOTFOUND THEN
      DELETE FROM hr.sections_t1 WHERE section_ID = V_DEPTNO;
      END IF;
   END;
/

CALL proc_cursor3();

--删除存储过程和表
DROP PROCEDURE proc_cursor3;
DROP TABLE hr.staffs_t1;
DROP TABLE hr.sections_t1;
DROP TABLE hr.staffs;
DROP TABLE hr.sections;
```

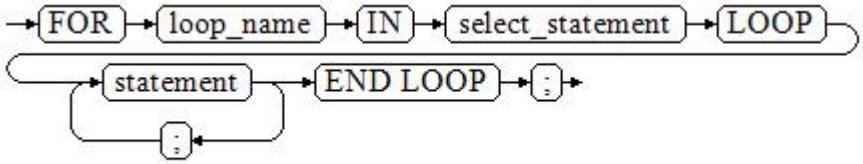
13.11.4.游标循环

游标在 WHILE 语句、LOOP 语句中的使用称为游标循环，一般这种循环都需要使用 OPEN、FETCH 和 CLOSE 语句。下面要介绍的一种循环不需要这些操作，可以简化游标循环的操作，这种循环方式适用于静态游标的循环，不用执行静态游标的四个步骤。

语法

FOR AS 循环的语法请参见图 13-33。

图 13-34. FOR_AS_loop::=



注意事项

- 不能在该循环语句中对查询的表进行更新操作。
- 变量 loop_name 会自动定义且只在此循环中有效，类型和 select_statement 的查询结果类型一致。loop_name 的取值就是 select_statement 的查询结果。
- 游标的属性中%FOUND、%NOTFOUND、%ROWCOUNT 在 Vastbase 数据库中都是访问同一个内部变量，事务和匿名块不支持多个游标同时访问。

13.12. Retry 管理

Retry 是数据库在 SQL 或存储过程（包含匿名块）执行失败时，在数据库内部进行重新执行的过程，以提高执行成功率和用户体验。数据库内部通过检查发生错误时的错误码及 Retry 相关配置，决定是否进行重试。

- 失败时回滚之前执行的语句，并重新执行存储过程进行 Retry。

  示例：

```
vastbase=# CREATE OR REPLACE PROCEDURE retry_basic ( IN  x INT)
AS
BEGIN
    INSERT INTO t1 (a) VALUES (x);
    INSERT INTO t1 (a) VALUES (x+1);
END;
/

vastbase=# CALL retry_basic(1);
```

13.13. 调试

语法

RAISE 有以下五种语法格式：

图 13-35. raise_format::=

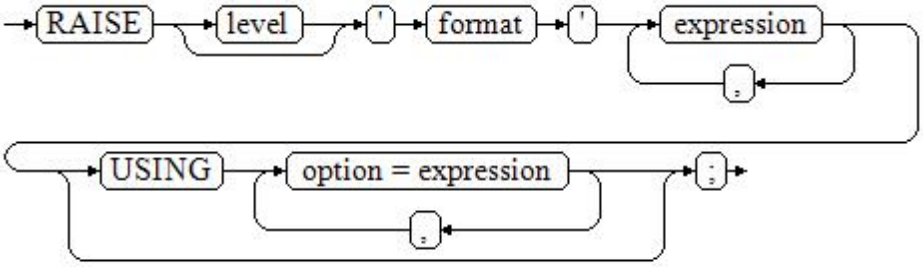


图 13-36. raise_condition::=

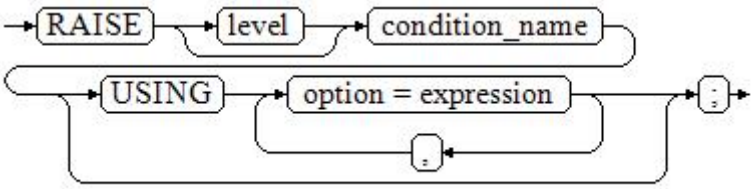


图 13-37. raise_sqlstate::=

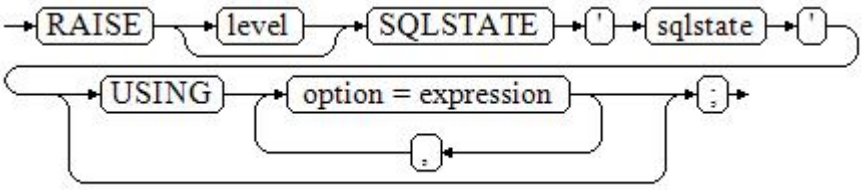


图 13-38. raise_option::=

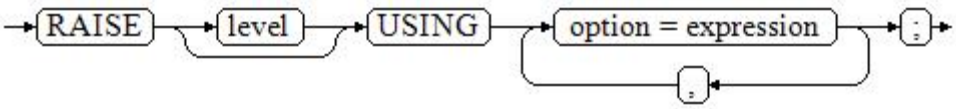
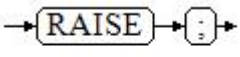


图 13-39. raise::=



参数说明：

- level 选项用于指定错误级别，有 DEBUG，LOG，INFO，NOTICE，WARNING 以及 EXCEPTION（默认值）。EXCEPTION 抛出一个正常终止当前事务的异常，其他的仅产生不同异常级别的信息。

特殊级别的错误信息是否报告到客户端、写到服务器日志由 log_min_messages 和 client_min_messages 这两个配置参数控制。

- format：格式字符串，指定要报告的错误消息文本。格式字符串后可跟表达式，用于向消息文本中插入。在格式字符串中，%由 format 后面跟着的参数的值替换，%%用于打印出%。例如：

```
--v_job_id 将替换字符串中的 %:
RAISE NOTICE 'Calling cs_create_job(%)',v_job_id;
```

- option = expression：向错误报告中添加另外的信息。关键字 option 可以是 MESSAGE、DETAIL、HINT 以及 ERRCODE，并且每一个 expression 可以是任意的字符串。
  - MESSAGE，指定错误消息文本，这个选项不能用于在 USING 前包含一个格式字符串的 RAISE 语句中。
  - DETAIL，说明错误的详细信息。
  - HINT，用于打印出提示信息。
  - ERRCODE，向报告中指定错误码（SQLSTATE）。可以使用条件名称或者直接用五位字符的 SQLSTATE 错误码。
- condition_name：错误码对应的条件名。
- sqlstate：错误码。

如果在 RAISE EXCEPTION 命令中既没有指定条件名也没有指定 SQLSTATE，默认用 RAISE EXCEPTION (P0001)。如果没有指定消息文本，默认用条件名或者 SQLSTATE 作为消息文本。

须知

当由 SQLSTATE 指定了错误码，则不局限于已定义的错误码，可以选择任意包含五个数字或者大写的 ASCII 字母的错误码，而不是 00000。建议避免使用以三个 0 结尾的错误码，因为这种错误码是类别码，会被整个种类捕获。

说明

图 13-38 所示的语法不接任何参数。这种形式仅用于一个 BEGIN 块中的 EXCEPTION 语句，它使得错误重新被处理。

示例

终止事务时，给出错误和提示信息：

```
CREATE OR REPLACE PROCEDURE proc_raise1(user_id in integer)
AS
BEGIN
RAISE EXCEPTION 'Noexistence ID --> %',user_id USING HINT = 'Please check your user ID';
END;
/

call proc_raise1(300011);

--执行结果
ERROR:  Noexistence ID --> 300011
HINT:  Please check your user ID
```

两种设置 SQLSTATE 的方式：

```
CREATE OR REPLACE PROCEDURE proc_raise2(user_id in integer)
AS
BEGIN
RAISE 'Duplicate user ID: %',user_id USING ERRCODE = 'unique_violation';
END;
/

\set VERBOSITY verbose
call proc_raise2(300011);

--执行结果
ERROR:  Duplicate user ID: 300011
SQLSTATE: 23505
LOCATION:  exec_stmt_raise, pl_exec.cpp:3482
```

如果主要的参数是条件名或者是 SQLSTATE，可以使用：

RAISE division_by_zero;

RAISE SQLSTATE '22012';

例如：

```
CREATE OR REPLACE PROCEDURE division(div in integer, dividend in integer)
AS
DECLARE
res int;
    BEGIN
    IF dividend=0 THEN
       RAISE division_by_zero;
       RETURN;
    ELSE
       res := div/dividend;
       RAISE INFO 'division result: %', res;
       RETURN;
    END IF;
    END;
/
call division(3,0);

--执行结果
ERROR:  division_by_zero
```

或者另一种方式：

```
RAISE unique_violation USING MESSAGE = 'Duplicate user ID: ' || user_id;
```

14. 系统表和系统视图

14.1. 系统表和系统视图概述

系统表是 Vastbase 存放结构元数据的地方，它是 Vastbase 数据库系统运行控制信息的来源，是数据库系统的核心组成部分。

系统视图提供了查询系统表和访问数据库内部状态的方法。

系统表和系统视图要么只对管理员可见，要么对所有用户可见。下面的系统表和视图有些标识了需要管理员权限，这些系统表和视图只有管理员可以查询。

用户可以删除后重新创建这些表、增加列、插入和更新数值，但是用户修改系统表会导致系统信息的不一致，从而导致系统控制紊乱。正常情况下不应该由用户手工修改系统表或系统视图，或者手工重命名系统表或系统视图所在的模式，而是由 SQL 语句关联的系统表操作自动维护系统表信息。

须知

用户应该禁止对系统表进行增删改等操作，人为对系统表的修改或破坏可能会导致系统各种异常情况甚至 Vastbase 不可用

14.2. 系统表

14.2.1. GS_OPT_MODEL

GS_OPT_MODEL 是启用 AiEngine 执行计划时间预测功能时的数据表，记录机器学习模型的配置、训练结果、功能、对应系统函数、训练历史等相关信息。

表 14-1. GS_OPT_MODEL 字段

名称	类型	描述
template_name	name	机器学习模型的模板名，决定训练和预测调用的函数接口，目前只实现了 rlstm，方便后续扩展。
model_name	name	模型的实例名，每个模型对应 aiEngine 在线学习进程中的一套参数、训练日志、模型系数。此列需为 unique。

名称	类型	描述
datname	name	该模型所服务的 database 名，每个模型只针对单个 database。此参数决定训练时所使用的数据。
ip	name	AiEngine 端所部署的 host ip 地址。
port	integer	AiEngine 端所监听的端口号。
max_epoch	integer	模型每次训练的迭代次数上限。
learning_rate	real	模型训练的学习速率，推荐缺省值 1。
dim_red	real	模型特征维度降维系数。
hidden_units	integer	模型隐藏层神经元个数。如果训练发现模型长期无法收敛，可以适量提升本参数。
batch_size	integer	模型每次迭代时一个 batch 的大小，尽量设为大于等于训练数据总量的值，加快模型的收敛速度。
feature_size	integer	[不需设置] 模型特征的长度，用于触发重新训练，模型训练后该参数自动更新。
available	boolean	[不需设置]标识模型是否收敛。
Is_training	boolean	[不需设置]标识模型是否正在训练。
label	"char"[]	模型的目标任务： • S: startup time • T: total time • R: rows • M: peak memory 目前受模型性能限制，推荐{S, T}或{R}。

名称	类型	描述
max	bigint[]	[不需设置]标识模型各任务标签的最大值，用于触发重新训练。
acc	real[]	[不需设置]标识模型个任务的准确率。
description	text	模型注释。

14.2.2. GS_WLM_INSTANCE_HISTORY

GS_WLM_INSTANCE_HISTORY 系统表存储与实例（数据库主节点或数据库节点）相关的资源使用相关信息。该系统表里每条记录都是对应时间点某实例资源使用情况，包括：内存、CPU 核数、磁盘 IO、进程物理 IO 和进程逻辑 IO 信息。

表 14-2. GS_WLM_INSTANCE_HISTORY 字段

名称	类型	描述
instancename	text	实例名称。
timestamp	timestamp with time zone	时间戳。
used_cpu	int	实例使用 CPU 所占用的百分比。
free_mem	int	实例未使用的内存大小，单位 MB。
used_mem	int	实例已使用的内存大小，单位 MB。
io_await	real	实例所使用磁盘的 io_wait 值（10 秒均值）。
io_util	real	实例所使用磁盘的 io_util 值（10 秒均值）。
disk_read	real	实例所使用磁盘的读速率（10 秒均值），单位 KB/s。
disk_write	real	实例所使用磁盘的写速率（10 秒均值），单位 KB/s。
process_read	bigint	实例对应进程从磁盘读数据的读速率(不包括从磁盘 pagecache 中读取的字节数，10 秒均值)，单位 KB/s。
process_write	bigint	实例对应进程向磁盘写数据的写速率(不包括向磁盘 pagecache 中写入的字节数，10 秒均值)，单位 KB/s。

名称	类型	描述
logical_read	bigint	数据库主节点实例：不统计。 数据库节点实例：该实例在本次统计间隙（10 秒）内逻辑读字节速率，单位 KB/s。
logical_write	bigint	数据库主节点实例：不统计。 数据库节点实例：该实例在本次统计间隙（10 秒）内逻辑写字节速率，单位 KB/s。
read_counts	bigint	数据库主节点实例：不统计。 数据库节点实例：该实例在本次统计间隙（10 秒）内逻辑读操作次数之和，单位次。
write_counts	bigint	数据库主节点实例：不统计。 数据库节点实例：该实例在本次统计间隙（10 秒）内逻辑写操作次数之和，单位次。

14.2.3. GS_WLM_OPERATOR_INFO

GS_WLM_OPERATOR_INFO 系统表显示执行作业结束后的算子相关的记录。此数据是从内核中转储到系统表中的数据。开启此功能会占用系统存储空间并对性能有一定影响，不建议用户使用。

表 14-3. GS_WLM_OPERATOR_INFO 的字段

名称	类型	描述
queryid	bigint	语句执行使用的内部 query_id。
pid	bigint	后端线程 id。
plan_node_id	integer	查询对应的执行计划的 plan node id。
plan_node_name	text	对应于 plan_node_id 的算子的名称。
start_time	timestamp with time zone	该算子处理第一条数据的开始时间。
duration	bigint	该算子到结束时候总的执行时间(ms)。
query_dop	integer	当前算子执行时的并行度。
estimated_rows	bigint	优化器估算的行数信息。

名称	类型	描述
tuple_processed	bigint	当前算子返回的元素个数。
min_peak_memory	integer	当前算子在数据库节点上的最小内存峰值(MB)。
max_peak_memory	integer	当前算子在数据库节点上的最大内存峰值(MB)。
average_peak_memory	integer	当前算子在数据库节点平均内存峰值(MB)。
memory_skew_percent	integer	当前算子在数据库节点间的内存使用倾斜率。
min_spill_size	integer	若发生下盘，数据库节点盘的最小数据量(MB)，默认为 0。
max_spill_size	integer	若发生下盘，数据库节点上下盘的最大数据量(MB)，默认为 0。
average_spill_size	integer	若发生下盘，数据库节点盘的平均数据量(MB)，默认为 0。
spill_skew_percent	integer	若发生下盘，数据库节点间下盘倾斜率。
min_cpu_time	bigint	该算子在数据库节点最小执行时间(ms)。
max_cpu_time	bigint	该算子在数据库节点上的最大执行时间(ms)。
total_cpu_time	bigint	该算子在数据库节点上的总执行时间(ms)。
cpu_skew_percent	integer	数据库节点间执行时间的倾斜率。
warning	text	主要显示如下几类告警信息： • Sort/SetOp/HashAgg/HashJoin spill • Spill file size large than 256MB • Broadcast size large than 100MB • Early spill • Spill times is greater than 3 • Spill on memory adaptive • Hash table conflict

14.2.4. GS_WLM_PLAN_ENCODING_TABLE

GS_WLM_PLAN_ENCODING_TABLE 系统表显示计划算子级的编码信息，为机器学习模型的提供包括 startup time, total time, peak memory, rows 等标签值的训练、预测集。

表 14-4. GS_WLM_PLAN_ENCODING_TABLE 的字段

名称	类型	描述
queryid	bigint	语句执行使用的内部 query_id。
plan_node_id	integer	查询对应的执行计划的 plan node id。
startup_time	bignit	该算子处理第一条数据的开始时间。
total_time	bigint	该算子到结束时候总的执行时间(ms)。
rows	bigint	当前算子执行的行数信息。
peak_memory	integer	当前算子在数据库节点上的最大内存峰值(MB)。
encode	text	当前计划算子的编码信息。

14.2.5. GS_WLM_PLAN_OPERATOR_INFO

GS_WLM_PLAN_OPERATOR_INFO 系统表显示执行作业结束后计划算子级的相关的记录。此数据是从内核中转储到系统表中的数据。当设置 GUC 参数 enable_resource_record 为 on 时，系统会定时（周期为 3 分钟）将 14.3.6GS_WLM_PLAN_OPERATOR_HISTORY 中的记录导入此系统表，开启此功能会占用系统存储空间并对性能有一定影响，不建议用户使用。

表 14-5. GS_WLM_PLAN_OPERATOR_INFO 的字段

名称	类型	描述
datname	name	收集计划信息所在的 database 名。
queryid	bigint	语句执行使用的内部 query_id。
plan_node_id	integer	查询对应的执行计划的 plan node id。
startup_time	bigint	该算子处理第一条数据的开始时间。
total_time	bigint	该算子到结束时候总的执行时间(ms)。
actual_rows	bigint	实际执行的行数信息。
max_peak_memory	integer	当前算子在数据库节点上的最大内存峰值(MB)。
query_dop	integer	当前算子执行时的并行度。
parent_node_id	integer	当前算子的父节点 node id。

名称	类型	描述
left_child_id	integer	当前算子的左孩子节点 node id。
right_child_id	integer	当前算子的右孩子节点 node id。
operation	text	当前算子进行的操作名称。
orientation	text	当前算子的对齐方式。
strategy	text	当前算子操作的实现方法。
options	text	当前算子操作的选择方式。
condition	text	当前算子操作的过滤条件。
projection	text	当前算子的映射关系。

14.2.6. GS_WLM_USER_RESOURCE_HISTORY

GS_WLM_USER_RESOURCE_HISTORY 系统表存储与用户使用资源相关的信息，仅在数据库主节点上有效。该系统表的每条记录都是对应时间点某用户的资源使用情况，包括：内存、CPU 核数、存储空间、临时空间、算子落盘空间、逻辑 IO 流量、逻辑 IO 次数和逻辑 IO 速率信息。其中，内存、CPU、IO 相关监控项仅记录用户复杂作业的资源使用情况。对于 IO 相关监控项，当参数 enable_logical_io_statistics 为 on 时才有效；当参数 enable_user_metric_persistent 为 on 时，才会开启用户监控数据保存功能。GS_WLM_USER_RESOURCE_HISTORY 系统表的数据来源于 PG_TOTAL_USER_RESOURCE_INFO 视图。

表 14-6. GS_WLM_USER_RESOURCE_HISTORY

名称	类型	描述
username	text	用户名
timestamp	timestamp with time zone	时间戳
used_memory	integer	正在使用的内存大小，单位 MB。
total_memory	integer	可以使用的内存大小，单位为 MB。值为 0 表示未限制最大可用内存，其限制取决于数据库最大可用内存。
used_cpu	real	正在使用的 CPU 核数。
total_cpu	integer	该机器节点上，用户关联控制组的 CPU 核数总和。
used_space	bigint	已使用的存储空间大小，单位 KB。

名称	类型	描述
total_space	bigint	可使用的存储空间大小，单位 KB，值为-1 表示未限制最大存储空间。
used_temp_space	bigint	已使用的临时存储空间大小，单位 KB。
total_temp_space	bigint	可使用的临存储空间大小，单位 KB，值为-1 表示未限制最大临时存储空间。
used_spill_space	bigint	已使用的算子落盘存储空间大小，单位 KB。
total_spill_space	bigint	可使用的算子落盘存储空间大小，单位 KB，值为-1 表示未限制最大算子落盘存储空间。
read_kbytes	bigint	监控周期内，读操作的字节流量，单位 KB。
write_kbytes	bigint	监控周期内，写操作的字节流量，单位 KB。
read_counts	bigint	监控周期内，读操作的次数，单位次。
write_counts	bigint	监控周期内，写操作的次数，单位次。
read_speed	real	监控周期内，读操作的字节速率，单位 KB/s。
write_speed	real	监控周期内，写操作的字节速率，单位 KB/s。

14.2.7. PG_AGGREGATE

PG_AGGREGATE 系统表存储与聚集函数有关的信息。PG_AGGREGATE 里的每条记录都是一条 pg_proc 里面的记录的扩展。PG_PROC 记录承载该聚集的名称、输入和输出数据类型，以及其它一些和普通函数类似的信息。

表 14-7. PG_AGGREGATE 字段

名称	类型	引用	描述
aggfnoid	regproc	14.2.48PG_PROC.proname	此聚集函数的 14.2.48PG_PROC proname。
aggtransfn	regproc	14.2.48PG_PROC.proname	转换函数。
aggcollectfn	regproc	14.2.48PG_PROC.proname	收集函数。

名称	类型	引用	描述
aggfinalfn	regproc	14.2.48PG_PROC.proname	最终处理函数（如果没有则为零）。
aggsortop	oid	14.2.44PG_OPERATOR.oid	关联排序操作符（如果没有则为零）。
aggtranstype	oid	14.2.66PG_TYPE.oid	此聚集函数的内部转换（状态）数据的数据类型。
agginitval	text	-	转换状态的初始值。这是一个文本数据域，它包含初始值的外部字符串表现形式。如果数据域是 null，则转换状态值从 null 开始。
agginitcollect	text	-	收集状态的初始值。这是一个文本数据域，它包含初始值的外部字符串表现形式。如果数据域是 null，则收集状态值从 null 开始。
aggkind	"char"	-	此聚集函数类型： • 'n'：表示 Normal Agg • 'o'：表示 Ordered Set Agg
aggnumdirectargs	smallint	-	Ordered Set Agg 类型聚集函数的直接参数 (非聚集相关参数) 数量。对 Normal Agg 类型聚集函数，该值为 0。

14.2.8. PG_AM

PG_AM 系统表存储有关索引访问方法的信息。系统支持的每种索引访问方法都有一行。

表 14-8. PG_AM 字段

名称	类型	引用	描述
amname	name	-	访问方法的名称。
amstrategies	smallint	-	访问方法的操作符策略个数，或者如果访问方法没有一个固定的操作符策略集则为 0。

名称	类型	引用	描述
amsupport	smallint	-	访问方法的支持过程个数。
amcanorder	Boolean	-	这种访问方式是否支持通过索引字段值的命令扫描排序。
amcanorderbyop	Boolean	-	这种访问方式是否支持通过索引字段上操作符的结果的命令扫描排序。
amcanbackward	Boolean	-	访问方式是否支持向后扫描。
amcanunique	Boolean	-	访问方式是否支持唯一索引。
amcanmulticol	Boolean	-	访问方式是否支持多字段索引。
amoptionalkey	Boolean	-	访问方式是否支持第一个索引字段上没有任何约束的扫描。
amsearcharray	Boolean	-	访问方式是否支持 ScalarArrayOpExpr 搜索。
amsearchnulls	Boolean	-	访问方式是否支持 IS NULL/NOT NULL 搜索。
amstorage	Boolean	-	允许索引存储的数据类型与列的数据类型是否不同。
amclusterable	Boolean	-	是否允许在一个这种类型的索引上 Vastbase。
ampredlocks	Boolean	-	是否允许这种类型的一个索引管理细粒度的谓词锁定。
amkeytype	oid	14.2.66PG_TYPE.oid	存储在索引里数据的类型，如果不是一个固定的类型则为 0。
aminsert	regproc	14.2.48PG_PROC.proname	“插入这个行”函数。
ambeginscan	regproc	14.2.48PG_PROC.proname	“准备索引扫描” 函数。
amgettuple	regproc	14.2.48PG_PROC.proname	“下一个有效行”函数，如果没有则为 0。
amgetbitmap	regproc	14.2.48PG_PROC.proname	“抓取所有的有效行” 函数，如果没有则为 0。
amrescan	regproc	14.2.48PG_PROC.proname	“（重新）开始索引扫描”函数。

名称	类型	引用	描述
amendscan	regproc	14.2.48PG_PROC.proname	“索引扫描后清理” 函数。
ammarkpos	regproc	14.2.48PG_PROC.proname	“标记当前扫描位置”函数。
amrestrpos	regproc	14.2.48PG_PROC.proname	“恢复已标记的扫描位置”函数。
ammerge	regproc	14.2.48PG_PROC.proname	“归并多个索引对象”函数。
ambuild	regproc	14.2.48PG_PROC.proname	“建立新索引”函数。
ambuildempty	regproc	14.2.48PG_PROC.proname	“建立空索引”函数。
ambulkdelete	regproc	14.2.48PG_PROC.proname	批量删除函数。
amvacuumcleanup	regproc	14.2.48PG_PROC.proname	VACUUM 后的清理函数。
amcanreturn	regproc	14.2.48PG_PROC.proname	检查是否索引支持唯一索引扫描的函数，如果没有则为 0。
amcostestimate	regproc	14.2.48PG_PROC.proname	估计一个索引扫描开销的函数。
amoptions	regproc	14.2.48PG_PROC.proname	为一个索引分析和确认 reloptions 的函数。

14.2.9. PG_AMOP

PG_AMOP 系统表存储有关和访问方法操作符族关联的信息。如果一个操作符是一个操作符族中的成员，则在这个表中会占据一行。一个族成员是一个 search 操作符或一个 ordering 操作符。一个操作符可以在多个族中出现，但是不能在一个族中的多个搜索位置或多个排序位置中出现。

表 14-9. PG_AMOP 字段

名称	类型	引用	描述
oid	oid	-	行标识符（隐藏属性，必须明确选

名称	类型	引用	描述
			择）。
amopfamily	oid	14.2.45PG_OPFAMILY.oid	这个项的操作符族。
amoplefttype	oid	14.2.66PG_TYPE.oid	操作符的左输入类型。
amoprighttype	oid	14.2.66PG_TYPE.oid	操作符的右输入类型。
amopstrategy	smallint	-	操作符策略数。
amoppurpose	"char"	-	操作符目的，s 为搜索或 o 为排序。
amopopr	oid	14.2.44PG_OPERATOR.oid	该操作符的 OID。
amopmethod	oid	14.2.8PG_AM.oid	索引访问方式操作符族。
amopsortfamily	oid	14.2.45PG_OPFAMILY.oid	如果是一个排序操作符，则为这个项排序所依据的 btree 操作符族；如果是一个搜索操作符，则为 0。

search 操作符表明这个操作符族的一个索引可以被搜索，找到所有满足 WHERE indexed_column operator constant 的行。显然，这样的操作符必须返回布尔值，并且它的左输入类型必须匹配索引的字段数据类型。

ordering 操作符表明这个操作符族的一个索引可以被扫描，返回以 ORDER BY indexed_column operator constant 顺序表示的行。这样的操作符可以返回任意可排序的数据类型，它的左输入类型也必须匹配索引的字段数据类型。 ORDER BY 的确切的语义是由 amopsortfamily 字段指定的，该字段必须为操作符的返回类型引用一个 btree 操作符族。

14.2.10. PG_AMPROC

PG_AMPROC 系统表存储有关与访问方法操作符族相关联的支持过程的信息。每个属于某个操作符族的支持过程都占有一行。

表 14-10. PG_AMPROC 字段

名称	类型	引用	描述
oid	oid	-	行标识符（隐藏属性，必须明确选择）。
amprocfamily	oid	14.2.45PG_OPFAMILY.oid	该项的操作符族。

名称	类型	引用	描述
amproclefttype	oid	14.2.66PG_TYPE.oid	相关操作符的左输入数据类型。
amprocrighttype	oid	14.2.66PG_TYPE.oid	相关操作符的右输入数据类型。
amprocnum	smallint	-	支持过程编号。
amproc	regproc	14.2.48PG_PROC .proname	过程的 OID。

amproclefttype 和 amprocrighttype 字段的习惯解释，标识一个特定支持过程支持的操作符的左和右输入类型。对于某些访问方式，匹配支持过程本身的输入数据类型，对其他的则不这样。有一个对索引的“缺省”支持过程的概念，amproclefttype 和 amprocrighttype 都等于索引操作符类的 opcintype。

14.2.11. PG_APP_WORKLOADGROUP_MAPPING

PG_APP_WORKLOADGROUP_MAPPING 系统表提供了数据库负载映射组的信息。

表 14-11. PG_APP_WORKLOADGROUP_MAPPING 字段

名称	类型	描述
oid	oid	行标识符（隐藏属性，必须明确选择）。
appname	name	应用名称。
workload_gpname	name	映射到的负载组名称。

14.2.12. PG_ATTRDEF

PG_ATTRDEF 系统表存储列的默认值。

表 14-12. PG_ATTRDEF 字段

名称	类型	描述
oid	oid	行标识符（隐藏属性，必须明确选择）。
adrelid	oid	该列的所属表。
adnum	smallint	该列的数目。
adbin	pg_node_tree	字段缺省值的内部表现形式。

名称	类型	描述
adsrc	text	人类可读的缺省值的内部表现形式。

14.2.13. PG_ATTRIBUTE

PG_ATTRIBUTE 系统表存储关于表字段的信息。

表 14-13. PG_ATTRIBUTE 字段

名称	类型	描述
attrelid	oid	此字段所属表。
attname	name	字段名。
atttypid	oid	字段类型。
attstattarget	integer	控制 ANALYZE 为这个字段积累的统计细节的级别。 • 零值表示不收集统计信息。 • 负数表示使用系统缺省的统计对象。 • 正数值的确切信息是和数据类型相关的。 对于标量数据类型，ATTSTATTARGET 既是要收集的"最常用数值"的目标数目，也是要创建的柱状图的目标数量。
attlen	smallint	是本字段类型的 pg_type.typlen 的拷贝。
attnum	smallint	字段编号。
attndims	integer	如果该字段是数组，则是维数，否则是 0 。
attcacheoff	integer	在磁盘上的时候总是-1 ，但是如果加载入内存中的行描述器中，它可能会被更新以缓冲在行中字段的偏移量。
atttypmod	integer	记录创建新表时支持的类型特定的数据（比如一个 varchar 字段的最大长度）。它传递给类型相关的输入和长度转换函数当做第三个参数。其值对那些不需要 ATTTYPMOD 的类型通常为-1。
attbyval	Boolean	这个字段类型的 pg_type.typbyval 的拷贝。
attstorage	"char"	这个字段类型的 pg_type.typstorage 的拷贝。
attalign	"char"	这个字段类型的 pg_type.typalign 的拷贝。

名称	类型	描述
attnotnull	Boolean	这代表一个非空约束。可以改变这个字段以打开或者关闭这个约束。
atthasdef	Boolean	这个字段有一个缺省值，此时它对应 pg_attrdef 表里实际定义此值的记录。
attgenerate	"char"	
attisdropped	Boolean	这个字段已经被删除了，不再有效。一个已经删除的字段物理上仍然存在表中，但会被分析器忽略，因此不能再通过 SQL 访问。
attislocal	Boolean	这个字段是局部定义在关系中的。请注意一个字段可以同时是局部定义和继承的。
attcmprmode	tinyint	对某一列指定压缩方式。压缩方式包括： • ATT_CMPR_NOCOMPRESS • ATT_CMPR_DELTA • ATT_CMPR_DICTIONARY • ATT_CMPR_PREFIX • ATT_CMPR_NUMSTR
attinhcount	integer	这个字段所拥有的直接父表的个数。如果一个字段的父表个数非零，则它就不能被删除或重命名。
attcollation	oid	对此列定义的校对列。
attacl	aclitem[]	列级访问权限控制。
attoptions	text[]	属性级可选项。
attfdwoptions	text[]	属性级外数据选项。
attinitdefval	bytea	存储了此列默认的值表达式。行存表的 ADD COLUMN 需要使用此字段。
attkvtype	tinyint	对某一列指定 key value 类型。类型包括： 0. ATT_KV_UNDEFINED：默认 1. ATT_KV_TAG： 维度 2. ATT_KV_FIELD：指标 3. ATT_KV_TIMETAG :时间列

14.2.14. PG_AUTHID

PG_AUTHID 系统表存储有关数据库认证标识符（角色）的信息。角色把“用户”的概念包含在内。一个用户实际上就是一个 rolcanlogin 标志被设置的角色。任何角色（不管 rolcanlogin 设置与否）都能够把其他角色作为成员。

Vastbase 中只有一份 pg_authid，不是每个数据库有一份。需要有系统管理员权限才可以访问此系统表。

表 14-14. PG_AUTHID 字段

名称	类型	描述
oid	oid	行标识符（隐藏属性，必须明确选择）。
rolname	name	角色名称。
rolsuper	Boolean	角色是否是拥有最高权限的初始系统管理员。
rolinherit	Boolean	角色是否自动继承其所属角色的权限。
rolcreaterole	Boolean	角色是否可以创建更多角色。
rolcreatedb	Boolean	角色是否可以创建数据库。
rolcatupdate	Boolean	角色是否可以直接更新系统表。只有 usesysid=10 的初始系统管理员拥有此权限。其他用户无法获得此权限。
rolcanlogin	Boolean	角色是否可以登录，也就是说，这个角色可以给予会话认证标识符。
rolssoadmin	Boolean	
rolreplication	Boolean	角色是一个复制的角色（适配作用，没有实际的功能）。
rolauditadmin	Boolean	审计用户。
rolsystemadmin	Boolean	管理员用户。
rolconnlimit	integer	对于可以登录的角色，限制其最大并发连接数量。 -1 表示没有限制。
rolpassword	text	口令(可能是加密的)，如果没有口令，则为 NULL。
rolvalidbegin	timestamp with time	帐户的有效开始时间，如果没有开始时间，则为

名称	类型	描述
	zone	NULL。
rolvaliduntil	timestamp with time zone	帐户的有效结束时间，如果没有结束时间，则为 NULL。
rolrespool	name	用户所能够使用的 resource pool。
roluseft	Boolean	角色是否可以操作外表。
rolparentid	oid	用户所在组用户的 OID。
roltabspace	text	用户数据表的最大空间限额。
rolkind	char	特殊用户种类，包括私有用户、普通用户。
rolnodegroup	oid	该字段不支持。
roltempspace	text	用户临时表的最大空间限额。
rolspillspace	text	用户执行作业时下盘数据的最大空间限额。
rolexcpdata	text	用户可以设置的查询规则(当前未使用)。

14.2.15.PG_AUTH_HISTORY

PG_AUTH_HISTORY 系统表记录了角色的认证历史。需要有系统管理员权限才可以访问此系统表。

表 14-15. PG_AUTH_HISTORY 字段

名称	类型	描述
oid	oid	行标识符（隐藏属性，必须明确选择）。
roloid	oid	角色标识。
passwordtime	timestamp with time zone	创建和修改密码的时间。
rolpassword	text	角色密码，使用 MD5、SHA256 加密或者不加密。

14.2.16.PG_AUTH_MEMBERS

PG_AUTH_MEMBERS 系统表存储显示角色之间的成员关系。

表 14-16. PG_AUTH_MEMBERS 字段

名称	类型	描述
roleid	oid	拥有成员的角色 ID。
member	oid	属于 ROLEID 角色的一个成员的角色 ID。
grantor	oid	赋予此成员关系的角色 ID。
admin_option	Boolean	如果 MEMBER 可以把 ROLEID 角色的成员关系赋予其他角色，则为真。

14.2.17. PG_CAST

PG_CAST 系统表存储数据类型之间的转化关系。

表 14-17. PG_CAST 字段

名称	类型	描述
oid	oid	行标识符（隐藏属性，必须明确选择）。
castsource	oid	源数据类型的 OID。
casttarget	oid	目标数据类型的 OID。
castfunc	oid	转化函数的 OID。如果为零表明不需要转化函数。
castcontext	"char"	源数据类型和目标数据类型间的转化方式： • 'e'：表示只能进行显式转化（使用 CAST 或::语法）。 • 'i'：表示只能进行隐式转化。 • 'a'：表示类型间同时支持隐式和显式转化。
castmethod	"char"	转化方法： • 'f'：使用 castfunc 字段中指定的函数进行转化。 • 'b'：类型间是二进制强制转化，不使用 castfunc。

14.2.18. PG_CLASS

PG_CLASS 系统表存储数据库对象信息及其之间的关系。

表 14-18. PG_CLASS 字段

名称	类型	描述
oid	oid	行标识符（隐藏属性，必须明确选择）。
relname	name	表、索引、视图等对象的名称。
relnamespace	oid	包含这个关系的名称空间的 OID。
reltype	oid	对应这个表的行类型的数据类型（索引为零，因为索引没有 pg_type 记录）。
reloftype	oid	复合类型的 OID，0 表示其他类型。
relowner	oid	关系所有者。
relam	oid	如果行是索引，则就是所用的访问模式（B-tree，hash 等）。
relfilenode	oid	这个关系在磁盘上的文件的名称，如果没有则为 0。
reltablespace	oid	这个关系存储所在的表空间。如果为零，则意味着使用该数据库的缺省表空间。如果关系在磁盘上没有文件，则这个字段没有什么意义。
relpages	double precision	以页(大小为 BLCKSZ)为单位的此表在磁盘上的大小，它只是优化器用的一个近似值。
reltuples	double precision	表中行的数目，只是优化器使用的一个估计值。
relallvisible	integer	被标识为全可见的表中的页的数量。此字段是优化器用来做 SQL 执行优化使用的。VACUUM、ANALYZE 和一些 DDL 语句（例如，CREATE INDEX）会引起此字段更新。
reltoastrelid	oid	与此表关联的 TOAST 表的 OID ，如果没有则为 0。 TOAST 表在一个从属表里“离线”存储大字段。
reltoastidxid	oid	对于 TOAST 表是它的索引的 OID，如果不是 TOAST 表则为 0。
reldeltarelid	oid	Delta 表的 OID。 Delta 表附属于列存表。用于存储数据导入过程中的甩尾数据。
reldeltaidx	oid	Delta 表的索引表 OID。
relcudescrelid	oid	CU 描述表的 OID。 CU 描述表（Desc 表）附属于列存表。用于控制表目录中存储数据

名称	类型	描述
		的可见性。
relcudescidx	oid	CU 描述表的索引表 OID。
relhasindex	Boolean	如果它是一个表而且至少有（或者最近有过）一个索引，则为真。 它是由 CREATE INDEX 设置的，但 DROP INDEX 不会立即将它清除。如果 VACUUM 进程检测一个表没有索引，将会把它将清理 relhasindex 字段，将值设置为假。
relisshared	Boolean	如果该表在 Vastbase 中由所有数据库共享则为真。只有某些系统表（比如 pg_database）是共享的。
relpersistence	"char"	• p：表示永久表。 • u：表示非日志表。 • t：表示临时表。
relkind	"char"	• r：表示普通表。 • i：表示索引。 • S：表示序列。 • v：表示视图。 • c：表示复合类型。 • t：表示 TOAST 表。 • f：表示外表。
relnatts	smallint	关系中用户字段数目（除了系统字段以外）。在 pg_attribute 里肯定有相同数目对应行。
relchecks	smallint	表里的检查约束的数目；参阅 pg_constraint 表。
relhasoids	Boolean	如果为关系中每行都生成一个 OID 则为真。
relhassecids	Boolean	
relhaspkey	Boolean	如果这个表有一个（或者曾经有一个）主键，则为真。
relhasrules	Boolean	如表有规则就为真。是否有规则可参考系统表 PG_REWRITE。
relhastriggers	Boolean	True 表示表中有触发器，或者曾经有过触发器。系统表 pg_trigger 中记录了表和视图的触发器。

名称	类型	描述
relhassubclass	Boolean	如果有（或者曾经有）任何继承的子表，为真。
relcmprs	tinyint	表示是否启用表的启用压缩特性。需要特别注意，当且仅当批量插入才会触发压缩，普通的 CRUD 并不能够触发压缩。 • 0 表示其他不支持压缩的表（主要是指系统表，不支持压缩属性的修改操作）。 • 1 表示表数据的压缩特性为 NOCOMPRESS 或者无指定关键字。 • 2 表示表数据的压缩特性为 COMPRESS。
relhasclusterkey	Boolean	是否有局部聚簇存储。
relrowmovement	Boolean	针对分区表进行 update 操作时，是否允许行迁移。 • true：表示允许行迁移。 • false：表示不允许行迁移。
parttype	"char"	表或者索引是否具有分区表的性质。 • p：表示带有分区表性质。 • n：表示没有分区表特性。
relfrozenxid	xid32	该表中所有在这个之前的事务 ID 已经被一个固定的（"frozen"）事务 ID 替换。该字段用于跟踪此表是否需要为了防止事务 ID 重叠（或者允许收缩 pg_clog）而进行清理。如果该关系不是表则为零（InvalidTransactionId）。 为保持前向兼容，保留此字段，新增 relfrozenxid64 用于记录此信息。
relacl	aclitem[]	访问权限。 查询的回显结果为以下形式： rolename=xxxx/yyyy --赋予一个角色的权限 =xxxx/yyyy --赋予 public 的权限 xxxx 表示赋予的权限，yyyy 表示授予这个权限的角色。权限的参数说明请参见表 14-19。
reloptions	text[]	索引的访问方法，使用"keyword=value"格式的字符串。
relreplident	"char"	逻辑解码中解码列的标识： • d = 默认（主键，如果存在）。 • n = 无。

名称	类型	描述
		• f = 所有列。 • i = 索引的 indisreplident 被设置或者为默认。
relfrozenxid64	xid	该表中所有在这个之前的事务 ID 已经被一个固定的（"frozen"）事务 ID 替换。该字段用于跟踪此表是否需要为了防止事务 ID 重叠（或者允许收缩 pg_clog）而进行清理。如果该关系不是表则为零（InvalidTransactionId）。
relbucket	oid	
relbucketkey	int2vector	

表 14-19. 权限的参数说明

参数	参数说明
r	SELECT（读）
w	UPDATE（写）
a	INSERT（插入）
d	DELETE
D	TRUNCATE
x	REFERENCES
t	TRIGGER
X	EXECUTE
U	USAGE
C	CREATE
c	CONNECT
T	TEMPORARY
arwdDxt	ALL PRIVILEGES（用于表）
*	给前面权限的授权选项

14.2.19. PG_COLLATION

PG_COLLATION 系统表描述可用的排序规则，本质上从一个 SQL 名称映射到操作系统本地类别。

表 14-20. PG_COLLATION 字段

名称	类型	引用	描述
oid	oid	-	行标识符（隐藏属性，必须明确选择）。
collname	name	-	排序规则名（每个名称空间和编码唯一）。
collnamespace	oid	14.2.41PG_NAMESPACE.oid	包含这个排序规则的名称空间的 OID。
collowner	oid	14.2.14PG_AUTHID.oid	排序规则的所有者。
collencoding	integer	-	排序规则可用的编码，如果适用于任意编码为-1。
collcollate	name	-	这个排序规则对象的 LC_COLLATE。
collctype	name	-	这个排序规则对象的 LC_CTYPE。

14.2.20. PG_CONSTRAINT

PG_CONSTRAINT 系统表存储表上的检查约束、主键和唯一约束。

表 14-21. PG_CONSTRAINT 字段

名称	类型	描述
oid	oid	行标识符（隐藏属性，必须明确选择）。
conname	name	约束名称（不一定是唯一的）。
connamespace	oid	包含这个约束的名称空间的 OID。
contype	"char"	• c = 检查约束 • p = 主键约束 • u = 唯一约束 • t = 触发器约束
condeferrable	Boolean	这个约束是否可以推迟。
condeferred	Boolean	缺省时这个约束是否可以推迟。
convalidated	Boolean	约束是否有效。目前，只有外键和 CHECK 约束可将其设置

名称	类型	描述
		为 FALSE。
conrelid	oid	这个约束所在的表；如果不是表约束则为 0。
contypid	oid	这个约束所在的域；如果不是一个域约束则为 0。
conindid	oid	与约束关联的索引 ID。
confrelid	oid	如果是外键，则为参考的表；否则为 0。由于当前不支持外键，所以值恒为 0。
confupdtype	"char"	外键更新动作代码。 • a = 没动作 • r = 限制 • c = 级联 • n =设置为 null • d =设置为缺省 由于当前不支持外键，所以值为空。
confdeltype	"char"	外键删除动作代码。 • a = 没动作 • r = 限制 • c = 级联 • n =设置为 null • d =设置为缺省 由于当前不支持外键，所以值为空。
confmatchtype	"char"	外键匹配类型。 • f = 全部 • p = 部分 • u = 简单（未指定） 由于当前不支持外键，所以值为空。
conislocal	Boolean	是否是为关系创建的本地约束。
coninhcount	integer	约束直接继承父表的数目。继承父表数非零时，不能删除或重命名该约束。

名称	类型	描述
connoinherit	Boolean	是否可以被继承。
consoft	Boolean	是否为信息约束(Informational Constraint)。
conopt	Boolean	是否使用信息约束优化执行计划。
conenable	Boolean	
conkey	smallint[]	如果是表约束，则是约束控制的字段列表。
confkey	smallint[]	如果是一个外键，是参考的字段的列表。由于当前不支持外键，所以值为空。
conpfeqop	oid[]	如果是一个外键，是做 PK=FK 比较的相等操作符 ID 的列表。由于当前不支持外键，所以值为空。
conppeqop	oid[]	如果是一个外键，是做 PK=PK 比较的相等操作符 ID 的列表。由于当前不支持外键，所以值为空。
conffeqop	oid[]	如果是一个外键，是做 FK=FK 比较的相等操作符 ID 的列表。由于当前不支持外键，所以值为空。
conexclop	oid[]	如果是一个排他约束，是列的排他操作符 ID 列表。
conbin	pg_node_tree	如果是检查约束，那就是其表达式的内部形式。
consrc	text	如果是检查约束，则是表达式的人类可读形式。

须知

- consrc 在被引用的对象改变之后不会被更新，它不会跟踪字段的名称修改。与其依赖这个字段，最好还是使用 pg_get_constraintdef()来抽取一个检查约束的定义。
- pg_class.relchecks 需要和在此表上为给定关系找到的检查约束的数目一致。

14.2.21. PG_CONVERSION

PG_CONVERSION 系统表描述编码转换信息。

表 14-22. PG_CONVERSION 字段

名称	类型	引用	描述
oid	oid	-	行标识符（隐藏属性，必须明确选择）。

名称	类型	引用	描述
conname	name	-	转换名称（在一个名称空间里是唯一的）。
connamespace	oid	14.2.41PG_NAMESPACE.oid	包含这个转换的名称空间的 OID。
conowner	oid	14.2.14PG_AUTHID.oid	编码转换的属主。
conforencoding	integer	-	源编码 ID。
contoencoding	integer	-	目的编码 ID。
conproc	regproc	14.2.48PG_PROC.proname	转换过程。
condefault	Boolean	-	如果这是缺省转换则为真。

14.2.22. PG_DATABASE

PG_DATABASE 系统表存储关于可用数据库的信息。

表 14-23. PG_DATABASE 字段

名称	类型	描述
oid	oid	行标识符（隐藏属性，必须明确选择）。
datname	name	数据库名称。
datdba	oid	数据库所有人，通常为其创建者。
encoding	integer	数据库的字符编码方式。
datcollate	name	数据库使用的排序顺序。
datctype	name	数据库使用的字符分类。
datistemplate	Boolean	是否允许作为模板数据库。
datallowconn	Boolean	如果为假，则没有用户可以连接到这个数据库。这个字段用于保护 template0 数据库不被更改。
datconnlimit	integer	该数据库上允许的最大并发连接数，-1 表示无限制。

名称	类型	描述
datlastsysoid	oid	数据库里最后一个系统 OID 。
datfrozenxid	xid32	用于跟踪该数据库是否需要为了防止事务 ID 重叠而进行清理。 为保持前向兼容，保留此字段，新增 datfrozenxid64 用于记录此信息。
dattablespace	oid	数据库的缺省表空间。
datcompatibility	name	数据库兼容模式。
datacl	aclitem[]	访问权限。
datfrozenxid64	xid	用于跟踪该数据库是否需要为了防止事务 ID 重叠而进行清理。

14.2.23. PG_DB_ROLE_SETTING

PG_DB_ROLE_SETTING 系统表存储数据库运行时每个角色与数据绑定的配置项的默认值 。

表 14-24. PG_DB_ROLE_SETTING 字段

名称	类型	描述
setdatabase	oid	配置项所对应的数据库，如果未指定数据库，则为 0。
setrole	oid	配置项所对应的角色，如果未指定角色，则为 0。
setconfig	text[]	运行时配置项的默认值。

14.2.24. PG_DEFAULT_ACL

PG_DEFAULT_ACL 系统表存储为新建对象设置的初始权限。

表 14-25. PG_DEFAULT_ACL 字段

名称	类型	描述
oid	oid	行标识符（隐藏属性，必须明确选择）。

名称	类型	描述
defaclrole	oid	与此权限相关的角色 ID。
defaclnamespace	oid	与此权限相关的名称空间，如果没有，则为 0。
defaclobjtype	"char"	此权限的对象类型。
defaclacl	aclitem[]	创建该类型时所拥有的访问权限。

14.2.25. PG_DEPEND

PG_DEPEND 系统表记录数据库对象之间的依赖关系。这个信息允许 DROP 命令找出哪些其它对象必须由 DROP CASCADE 删除，或者是在 DROP RESTRICT 的情况下避免删除。

这个表的功能类似 14.2.54PG_SHDEPEND，用于记录那些在 Vastbase 之间共享的对象之间的依赖性关系。

表 14-26. PG_DEPEND 字段

名称	类型	引用	描述
classid	oid	14.2.18PG_CLASS.oid	有依赖对象所在系统表的 OID。
objid	oid	任意 OID 属性	指定的依赖对象的 OID。
objsubid	integer	-	对于表字段，这个是该属性的字段数（objid 和 classid 引用表本身）。对于所有其它对象类型，目前这个字段是零。
refclassid	oid	14.2.18PG_CLASS.oid	被引用对象所在的系统表的 OID。
refobjid	oid	任意 OID 属性	指定的被引用对象的 OID。
refobjsubid	integer	-	对于表字段，这个是该字段的字段号（refobjid 和 refclassid 引用表本身）。对于所有其它对象类型，目前这个字段是零。
deptype	"char"	-	一个定义这个依赖关系特定语义的代码。

在所有情况下，一个 PG_DEPEND 记录表示被引用的对象不能在有依赖的对象被删除前删除。不过，这里还有几种由 deptype 定义的情况：

❖ DEPENDENCY_NORMAL (n)：独立创建的对象之间的一般关系。有依赖的对象可以在不影响被引用对象的情况下删除。被引用对象只有在声明了 CASCADE 的情况下删除，这时有依赖的对象也被删除。例子：一个表字段对其数据类型有一般依赖关系。

❖ DEPENDENCY_AUTO (a)：有依赖对象可以和被引用对象分别删除，并且如果删除了被引用对象则应该被自动删除（不管是 RESTRICT 或 CASCADE 模式）。例子：一个表上面的命名约束是在该表上的自动依赖关系，因此如果删除了表，它也会被删除。

❖ DEPENDENCY_INTERNAL (i)：有依赖的对象是作为被引用对象的一部分创建的，实际上只是它的内部实现的一部分。 DROP 有依赖对象是不能直接允许的（将告诉用户发出一条删除被引用对象的 DROP）。一个对被引用对象的 DROP 将传播到有依赖对象，不管是否声明了 CASCADE。

❖ DEPENDENCY_EXTENSION (e)：依赖对象是被依赖对象 extension 的一个成员（请参见 14.2.29PG_EXTENSION）。依赖对象只可以通过在被依赖对象上 DROP EXTENSION 删除。函数上这个依赖类型和内部依赖一样动作，但是它为了清晰和简化 vb_dump 保持分开。

❖ DEPENDENCY_PIN (p)：没有依赖对象；这种类型的记录标志着系统本身依赖于被引用对象，因此这个对象决不能被删除。这种类型的记录只有在 initdb 的时候创建。有依赖对象的字段里是零。

14.2.26. PG_DESCRIPTION

PG_DESCRIPTION 系统表可以给每个数据库对象存储一个可选的描述（注释）。许多内置的系统对象的描述提供了 PG_DESCRIPTION 的初始内容。

这个表的功能类似 14.2.55PG_SHDESCRIPTION，用于记录 Vastbase 范围内共享对象的注释。

表 14-27. PG_DESCRIPTION 字段

名称	类型	引用	描述
objoid	oid	任意 OID 属性	这条描述所描述的对象的 OID。
classoid	oid	14.2.18PG_CLASS.oid	这个对象出现的系统表的 OID。
objsubid	integer	-	对于一个表字段的注释，它是字段号（objoid 和 classoid 指向表自身）。对于其它对象类型，它是零。
description	text	-	对该对象描述的任意文本。

14.2.27. PG_DIRECTORY

PG_DIRECTORY 系统表用于保存用户添加的 directory 对象可以通过 CREATE DIRECTORY 语句向该表中添加记录，目前只有系统管理员用户可以向该表中添加记录。

表 14-28. PG_DIRECTORY 字段

名称	类型	描述
oid	oid	行标识符（隐藏属性，必须明确选择）。
dirname	name	目录对象的名称。
owner	oid	目录对象的所有者。
dirpath	text	目录路径。
diracl	aclitem[]	访问权限。

14.2.28. PG_ENUM

PG_ENUM 系统表包含显示每个枚举类型值和标签的记录。给定枚举类型的内部表示实际上是 PG_ENUM 里面相关行的 OID。

表 14-29. PG_ENUM 字段

名称	类型	引用	描述
oid	oid	-	行标识符（隐藏属性，必须明确选择）。
enumtypid	oid	14.2.66PG_TYPE.oid	拥有这个枚举值的 pg_type 记录的 OID。
enumsortorder	real	-	这个枚举值在它的枚举类型中的排序位置。
enumlabel	name	-	这个枚举值的文本标签。

PG_ENUM 行的 OID 跟着一个特殊规则：偶数的 OID 保证用和它们的枚举类型一样的排序顺序排序。也就是，如果两个偶数 OID 属于相同的枚举类型，那么较小的 OID 必须有较小 enumsortorder 值。奇数 OID 需要毫无关系的排序顺序。这个规则允许枚举比较例程在许多常见情况下避开目录查找。创建和修改枚举类型的例程只要可能就尝试分配偶数 OID 给枚举值。

当创建了一个枚举类型时，它的成员赋予了排序顺序位置 1 到 n。但是随后添加的成员可能会分配 enumsortorder 的负值或分数值。对这些值的唯一要求是它们要正确的排序和在每个枚举类型中唯一。

14.2.29. PG_EXTENSION

PG_EXTENSION 系统表存储关于所安装扩展的信息。

表 14-30. PG_EXTENSION

名称	类型	描述
extname	name	扩展名。
extowner	oid	扩展的所有者。
extnamespace	oid	扩展导出对象的名称空间。
extrelocatable	Boolean	标识此扩展是否可迁移到其他名称空间，true 表示允许。
extversion	text	扩展的版本号。
extconfig	oid[]	扩展的配置信息。
extcondition	text[]	扩展配置信息的过滤条件。

14.2.30. PG_EXTENSION_DATA_SOURCE

PG_EXTENSION_DATA_SOURCE 系统表存储外部数据源对象的信息。一个外部数据源对象（Data Source）包含了外部数据库的一些口令编码等信息，主要配合 Extension Connector 使用。

表 14-31. PG_EXTENSION_DATA_SOURCE 字段

名称	类型	引用	描述
oid	oid	-	行标识符（隐藏属性，必须明确选择）。
srcname	name	-	外部数据源对象的名称。
srcowner	oid	PG_AUTHID.oid	外部数据源对象的所有者。
srctype	text	-	外部数据源对象的类型，缺省为空。
srcversion	text	-	外部数据源对象的版本，缺省为空。
srcacl	aclitem[]	-	访问权限。
srcoptions	text[]	-	外部数据源对象的指定选项，使用“keyword=value”格式的字符串。

14.2.31. PG_FOREIGN_DATA_WRAPPER

PG_FOREIGN_DATA_WRAPPER 系统表存储外部数据封装器定义。一个外部数据封装器是在外部服务器上驻留外部数据的机制，是可以访问的。

表 14-32. PG_FOREIGN_DATA_WRAPPER 字段

名称	类型	引用	描述
oid	oid	-	行标识符(隐藏属性，必须明确选择)。
fdwname	name	-	外部数据封装器名。
fdwowner	oid	14.2.14PG_AUTHID.oid	外部数据封装器的所有者。
fdwhandler	oid	14.2.48PG_PROC.oid	引用一个负责为外部数据封装器提供扩展例程的处理函数。如果没有提供处理函数则为零。
fdwvalidator	oid	14.2.48PG_PROC.oid	引用一个验证器函数，这个验证器函数负责验证给予外部数据封装器的选项、外部服务器选项和使用外部数据封装器的用户映射的有效性。如果没有提供验证器函数则为零。
fdwacl	aclitem[]	-	访问权限。
fdwoptions	text[]	-	外部数据封装器指定选项，使用“keyword=value”格式的字符串。

14.2.32. PG_FOREIGN_SERVER

PG_FOREIGN_SERVER 系统表存储外部服务器定义。一个外部服务器描述了一个外部数据源，例如一个远程服务器。外部服务器通过外部数据封装器访问。

表 14-33. PG_FOREIGN_SERVER 字段

名称	类型	引用	描述
oid	oid	-	行标识符（隐藏属性，必须明确选择）。
srvname	name	-	外部服务器名。
srvowner	oid	14.2.14PG_AUTHID.oid	外部服务器的所有者。

名称	类型	引用	描述
srvfdw	oid	14.2.31PG_FOREIGN_DATA_WRAPPER.oid	这个外部服务器的外部数据封装器的OID。
srvtype	text	-	服务器的类型（可选）。
srvversion	text	-	服务器的版本（可选）。
srvacl	aclitem[]	-	访问权限。
srvoptions	text[]	-	外部服务器指定选项，使用“keyword=value”格式的字符串。

14.2.33. PG_FOREIGN_TABLE

PG_FOREIGN_TABLE 系统表存储外部表的辅助信息。

表 14-34. PG_FOREIGN_TABLE 字段

名称	类型	描述
ftrelid	oid	外部表的 ID。
ftserver	oid	外部表的所在服务器。
ftwriteonly	Boolean	外部表是否可写。
ftoptions	text[]	外部表的可选项。

14.2.34. PG_INDEX

PG_INDEX 系统表存储索引的一部分信息，其他的信息大多数在 PG_CLASS 中。

表 14-35. PG_INDEX 字段

名称	类型	描述
indexrelid	oid	这个索引在 pg_class 里的记录的 OID。
indrelid	oid	使用这个索引的表在 pg_class 里的记录的 OID。
indnatts	smallint	索引中的字段数目。

名称	类型	描述
indisunique	Boolean	如果为真，这是个唯一索引。
indisprimary	Boolean	如果为真，该索引代表该表的主键。这个字段为真的时候 indisunique 总是为真。
indisexclusion	Boolean	如果为真，该索引支持排他约束。
indimmediate	Boolean	如果为真，在插入数据时会立即进行唯一性检查。
indisclustered	Boolean	如果为真，则该表最后在这个索引上建了簇。
indisusable	Boolean	如果为真，该索引对 insert/select 可用。
indisvalid	Boolean	如果为真，则该索引可以用于查询。如果为假，则该索引可能不完整，仍然必须在 INSERT/UPDATE 操作时进行更新，不过不能安全的用于查询。如果是唯一索引，则唯一属性也将不为真。
indcheckxmin	Boolean	如果为 true，查询不能使用索引，直到 pg_index 此行的 xmin 低于其快照的 TransactionXmin，因为该表可能包含它们能看到的不兼容行断开的 HOT 链。
indisready	Boolean	如果为真，表示此索引对插入数据是可用的，否则，在插入或修改数据时忽略此索引。
indkey	int2vector	这是一个包含 indnatts 值的数组，这些数组值表示这个索引所建立的表字段。比如一个值为 1 3 的意思是第一个字段和第三个字段组成这个索引键字。这个数组里的零表明对应的索引属性是在这个表字段上的一个表达式，而不是一个简单的字段引用。
indcollation	oidvector	索引用到的各列的 ID。
indclass	oidvector	对于索引键字里面的每个字段，这个字段都包含一个指向所使用的操作符类的 OID，参阅 pg_opclass 获取细节。
indoption	int2vector	存储列前标识位，该标识位是由索引的访问方法定义。
indexprs	pg_node_tree	表达式树（以 nodeToString()形式表现）用于那些非简单字段引用的索引属性。它是一个列表，个数与 INDKEY 中的零值个数相同。如果所有索引属性都是简单的引用，则为空。
indpred	pg_node_tree	部分索引断言的表达式树（以 nodeToString()的形式表现）。如果不是部分索引，则是空字符串。

名称	类型	描述
indisreplident	Boolean	如果为真，则此索引的列成为逻辑解码的解码列。

14.2.35. PG_INHERITS

PG_INHERITS 系统表记录关于表继承层次的信息。数据库里每个直接的子系表都有一条记录。间接的继承可以通过追溯记录链来判断。

表 14-36. PG_INHERITS 字段

名称	类型	引用	描述
inhrelid	oid	14.2.18PG_CLASS.oid	子表的 OID。
inhparent	oid	14.2.18PG_CLASS.oid	父表的 OID。
inhseqno	integer	-	如果一个子表存在多个直系父表（多重继承），这个数字表明此继承字段的排列顺序。计数从 1 开始。

14.2.36. PG_JOB

PG_JOB 系统表存储用户创建的定时任务的任务详细信息，定时任务线程定时轮询 pg_job 系统表中的时间，当任务到期会触发任务的执行，并更新 pg_job 表中的任务状态。该系统表属于 Shared Relation，所有创建的 job 记录对所有数据库可见。

表 14-37. PG_JOB 字段

名称	类型	描述
oid	oid	行标识符（隐藏属性，必须明确选择）。
job_id	bigint	作业 ID，主键，是唯一的（有唯一索引）
current_postgres_pid	bigint	如果当前任务已被执行，那么此处记录运行此任务的 postgres 线程 ID。默认为 -1，表示此任务未被执行过。
log_user	name	创建者的 UserName
priv_user	name	作业执行者的 UserName

名称	类型	描述
dbname	name	标识作业要在哪个数据库执行的数据库名称
node_name	name	标识当前作业是在哪个数据库主节点上创建和执行
job_status	"char"	当前任务的执行状态，取值范围：('r', 's', 'f', 'd')，默认为's'，取值含义： Status of job step: r=running, s=successfully finished, f=job failed, d=disable 当 job 连续执行失败 16 次，会将 job_status 自动设置为失效状态'd'，后续不再执行该 job。 注：当用户将定时任务关闭（即：guc 参数 job_queue_processes 为 0 时），由于监控 job 执行的线程不会启动，所以该状态不会根据 job 的实时状态进行设置，用户不需要关注此状态。只有当开启定时任务功能（即：guc 参数 job_queue_processes 为非 0 时），系统才会根据当前 job 的实时状态刷新该字段值。
start_date	timestamp without time zone	作业第一次开始执行时间，时间精确到毫秒。
next_run_date	timestamp without time zone	下次定时执行任务的时间，时间精确到毫秒。
failure_count	smallint	失败计数，作业连续执行失败 16 次，不再继续执行。
interval	text	作业执行的重复时间间隔。
listitem	text[]	
moditem	text[]	
remitem	text[]	
last_start_date	timestamp without time zone	上次运行开始时间，时间精确到毫秒。
last_end_date	timestamp without time zone	上次运行的结束时间，时间精确到毫秒。
last_suc_date	timestamp without time zone	上次成功运行的开始时间，时间精确到毫秒。
this_run_date	timestamp without time zone	正在运行任务的开始时间，时间精确到毫秒。

名称	类型	描述
nspname	name	标识作业执行时的 schema 的名称。

14.2.37. PG_JOB_PROC

PG_JOB_PROC 系统表对应 PG_JOB 表中每个任务的作业内容（包括：PL/SQL 代码块、匿名块）。将存储过程信息独立出来，如果放到 PG_JOB 中，被加载到共享内存的时候，会占用不必要的空间，所以在使用的时候再进行查询获取。

表 14-38. PG_JOB_PROC 字段

名称	类型	描述
oid	oid	行标识符（隐藏属性，必须明确选择）。
job_id	integer	关联 pg_job 表中的 job_id。
what	text	作业内容。

14.2.38. PG_LANGUAGE

PG_LANGUAGE 系统表登记编程语言，用户可以用这些语言或接口写函数或者存储过程。

表 14-39. PG_LANGUAGE 字段

名称	类型	引用	描述
oid	oid	-	行标识符（隐藏属性；必须明确选择）。
lanname	name	-	语言的名称。
lanowner	oid	14.2.14PG_AUTHID.oid	语言的所有者。
lanispl	Boolean	-	对于内部语言而言是假（比如 SQL），对于用户定义的语言则是真。目前，vb_dump 仍然使用这个东西判断哪种语言需要转储，但是这些可能在将来被其它机制取代。
lanpltrusted	Boolean	-	如果这是可信语言则为真，意味着系统相信它不会被授予任何正常 SQL 执行环境之外的权限。只

名称	类型	引用	描述
			有初始用户可以用不可信的语言创建函数。
lanplcallfoid	oid	14.2.48PG_PROC.oid	对于非内部语言，这是指向该语言处理器的引用，语言处理器是一个特殊函数， 负责执行以某种语言写的所有函数。
laninline	oid	14.2.48PG_PROC.oid	这个字段引用一个负责执行“inline”匿名代码块的函数（DO 块）。如果不支持内联块则为零。
lanvalidator	oid	14.2.48PG_PROC.oid	这个字段引用一个语言校验器函数，它负责检查新创建的函数的语法和有效性。如果没有提供校验器，则为零。
lanacl	aclitem[]	-	访问权限。

14.2.39. PG_LARGEOBJECT

PG_LARGEOBJECT 系统表保存那些标记着“大对象”的数据。一个大对象是使用其创建时分配的 OID 标识的。每个大对象都分解成足够小的小段或者“页面”以便以行的形式存储在 PG_LARGEOBJECT 里。每页的数据定义为 LOBLKSIZE。

需要有系统管理员权限才可以访问此系统表。

表 14-40. PG_LARGEOBJECT 字段

名称	类型	引用	描述
loid	oid	14.2.40PG_LARGEOBJECT_METADATA.oid	包含本页的大对象的标识符。
pageno	integer	-	本页在其大对象数据中的页码从零开始计算。
data	bytea	-	存储在大对象中的实际数据。这些数据绝不会超过 LOBLKSIZE 字节，而且可能更少。

PG_LARGEOBJECT 的每一行保存一个大对象的一个页面，从该对象内部的字节偏移（pageno * LOBLKSIZE）开始。这种实现允许松散的存储：页面可以丢失，而且可以比 LOBLKSIZE 字节少（即使它们不是对象的最后一页）。大对象内丢失的部分读做零。

14.2.40. PG_LARGEOBJECT_METADATA

PG_LARGEOBJECT_METADATA 系统表存储与大数据相关的元数据。实际的大对象数据存储在 PG_LARGEOBJECT 里。

表 14-41. PG_LARGEOBJECT_METADATA 字段

名称	类型	引用	描述
oid	oid	-	行标识符（隐藏属性，必须明确选择）。
lomowner	oid	14.2.14PG_AUTHID.oid	大对象的所有者。
lomacl	aclitem[]	-	访问权限。

14.2.41. PG_NAMESPACE

PG_NAMESPACE 系统表存储名称空间，即存储 schema 相关的信息。

表 14-42. PG_NAMESPACE 字段

名称	类型	描述
oid	oid	行标识符（隐藏属性，必须明确选择）。
nspname	name	名称空间的名称。
nspowner	oid	名称空间的所有者。
nsptimeline	bigint	在数据库节点上创建此命名空间时的时间线。此字段为内部使用，仅在数据库节点上有效。
nspacl	aclitem[]	访问权限。

14.2.42. PG_OBJECT

PG_OBJECT 系统表存储限定类型对象（普通表，索引，序列，视图，存储过程和函数）的创建用户、创建时间和最后修改时间。

表 14-43. PG_OBJECT 字段

名称	类型	描述

名称	类型	描述
object_oid	oid	对象标识符
object_type	"char"	对象类型： • r 表示普通表 • i 表示索引 • s 表示序列 • v 表示视图 • p 表示存储过程和函数
creator	oid	创建用户的标识符
ctime	timestamp with time zone	对象的创建时间
mtime	timestamp with time zone	对象的最后修改时间，修改行为包括 ALTER 操作和 GRANT、REVOKE 操作

须知

- 无法记录初始化数据库（initdb）过程中所创建或修改的对象，即 PG_OBJECT 无法查询到该对象记录。
- 对于上述两类对象再次修改时，会记录其修改时间（mtime），由于无法得知该对象创建时间，因此 ctime 为空。
- ctime 和 mtime 所记录的时间为用户当次操作所属事务的起始时间。
- 由扩容引起的对象修改时间也会被记录。

14.2.43. PG_OPCLASS

PG_OPCLASS 系统表定义索引访问方法操作符类。

每个操作符类为一种特定数据类型和一种特定索引访问方法定义索引字段的语义。一个操作符类本质上指定一个特定的操作符族适用于一个特定的可索引的字段数据类型。索引的字段实际可用的族中的操作符集是接受字段的数据类型作为它们的左边的输入的那个。

表 14-44. PG_OPCLASS 字段

名称	类型	引用	描述
oid	oid	-	行标识符（隐藏属性，必须明确选择）。
opcmethod	oid	14.2.8PG_AM.oid	操作符类所服务的索引访问方法。
opcname	name	-	这个操作符类的名称。

名称	类型	引用	描述
opcnamespace	oid	14.2.41PG_NAMESPACE.oid	这个操作符类的名称空间。
opcowner	oid	14.2.14PG_AUTHID.oid	操作符类属主。
opcfamily	oid	14.2.45PG_OPFAMILY.oid	包含该操作符类的操作符族。
opcintype	oid	14.2.66PG_TYPE.oid	操作符类索引的数据类型。
opcdefault	Boolean	-	如果操作符类是 opcintype 的缺省,则为真。
opckeytype	oid	14.2.66PG_TYPE.oid	索引数据的类型，如果和 opcintype 相同则为零。

一个操作符类的 opcmethod 必须匹配包含它的操作符族的 opfmethod。同样，对于任意给定的 opcmethod 和 opcintype 的组合，不能有超过一个 PG_OPCLASS 行有 opcdefault 为真。

14.2.44. PG_OPERATOR

PG_OPERATOR 系统表存储有关操作符的信息。

表 14-45. PG_OPERATOR 字段

名称	类型	引用	描述
oid	oid	-	行标识符 (隐藏属性，必须明确选择)。
oprname	name	-	操作符的名称。
oprnamespace	oid	14.2.41PG_NAMESPACE.oid	包含此操作符的名称空间的 OID。
oprowner	oid	14.2.14PG_AUTHID.oid	操作符所有者。
oprkind	"char"	-	• b=infix =中缀(“两边”) • l=前缀(“左边”) • r=后缀(“右边”)
oprcanmerge	Boolean	-	这个操作符支持合并连接。
oprcanhash	Boolean	-	这个操作符支持 Hash 连接。
oprleft	oid	14.2.66PG_TYPE.oid	左操作数的类型。

名称	类型	引用	描述
oprright	oid	14.2.66PG_TYPE.oid	右操作数的类型。
oprresult	oid	14.2.66PG_TYPE.oid	结果类型。
oprcom	oid	14.2.44PG_OPERATOR.oid	此操作符的交换符，如果存在的话。
oprnegate	oid	14.2.44PG_OPERATOR.oid	此操作符的反转器，如果存在的话。
oprcode	regproc	14.2.48PG_PROC.proname	实现这个操作符的函数。
oprrest	regproc	14.2.48PG_PROC.proname	此操作符的约束选择性计算函数。
oprjoin	regproc	14.2.48PG_PROC.proname	此操作符的连接选择性计算函数。

14.2.45. PG_OPFAMILY

PG_OPFAMILY 系统表定义操作符族。

每个操作符族是一个操作符和相关支持例程的集合，其中的例程实现为一个特定的索引访问方式指定的语义。另外，族中的操作符都是“兼容的”，通过由访问方式指定的方法。操作符族的概念允许交叉数据类型操作符和索引一起使用，并且合理的使用访问方式的语义的知识。

表 14-46. PG_OPFAMILY 字段

名称	类型	引用	描述
oid	oid	-	行标识符（隐藏属性，必须明确选择）。
opfmethod	oid	14.2.8PG_AM.oid	操作符族使用的索引方法。
opfname	name	-	这个操作符族的名称。
opfnamespace	oid	14.2.41PG_NAMESPACE.oid	这个操作符的名称空间。
opfowner	oid	14.2.14PG_AUTHID.oid	操作符族的所有者。

定义一个操作符族的大多数信息不在它的 PG_OPFAMILY 行里面，而是在相关的行 14.2.9PG_AMOP，14.2.10PG_AMPROC 和 14.2.43PG_OPCLASS 里。

14.2.46. PG_PARTITION

PG_PARTITION 系统表存储数据库内所有分区表(partitioned table)、分区(table partition)、分区上 toast 表和分区索引(index partition)四类对象的信息。分区表索引(partitioned index)的信息不在 PG_PARTITION 系统表中保存。

表 14-47. PG_PARTITION 字段

名称	类型	描述
oid	oid	行标识符（隐藏属性，必须明确选择）。
relname	name	分区表、分区、分区上 toast 表和分区索引的名称。
parttype	"char"	对象类型： • 'r'：partitioned table • 'p'：table partition • 'x'：index partition • 't'：toast table
parentid	oid	当对象为分区表或分区时，此字段表示分区表在 PG_CLASS 中的 OID。 当对象为 index partition 时，此字段表示所属分区表索引(partitioned index)的 OID。
rangenum	integer	保留字段。
intervalnum	integer	保留字段。
partstrategy	"char"	分区表分区策略，现在仅支持： • 'r'：范围分区。 • 'v'：数值分区。
relfilenode	oid	table partition、index partition、分区上 toast 表的物理存储位置。
reltablespace	oid	table partition、index partition、分区上 toast 表所属表空间的 OID。
relpages	double precision	统计信息：table partition、index partition 的数据页数量。

名称	类型	描述
reltuples	double precision	统计信息：table partition、index partition 的元组数。
relallvisible	integer	统计信息：table partition、index partition 的可见数据页数。
reltoastrelid	oid	table partition 所对应 toast 表的 OID。
reltoastidxid	oid	table partition 所对应 toast 表的索引的 OID。
indextblid	oid	index partition 对应 table partition 的 OID。
indisusable	Boolean	分区索引是否可用。
reldeltarelid	oid	Delta 表的 OID。
reldeltaidx	oid	Delta 表的索引表的 OID。
relcudescrelid	oid	CU 描述表的 OID。
relcudescidx	oid	CU 描述表的索引表的 OID。
relfrozenxid	xid32	冻结事务 ID 号。 为保持前向兼容，保留此字段，新增 relfrozenxid64 用于记录此信息。
intspnum	integer	间隔分区所属表空间的个数。
partkey	int2vector	分区键的列号。
intervaltablespace	oidvector	间隔分区所属的表空间，间隔分区以 round-robin 方式落在这些表空间内。
interval	text[]	间隔分区的间隔值。
boundaries	text[]	范围分区和间隔分区的上边界。
listitem	text[]	
moditem	text[]	
remitem	text[]	
transit	text[]	间隔分区的跳转点。
reloptions	text[]	设置 partition 的存储属性，与 pg_class.reloptions 的形态一样，用"keyword=value"格式的字符串来表示 ，目前用于在线扩容的信息搜集。

名称	类型	描述
relfrozenxid64	xid	冻结事务 ID 号。

14.2.47. PG_PLTEMPLATE

PG_PLTEMPLATE 系统表存储过程语言的“模板”信息。

表 14-48. PG_PLTEMPLATE 字段

名称	类型	描述
tmplname	name	这个模板所应用的语言的名称。
tmpltrusted	Boolean	如果语言被认为是可信的，则为真。
tmpldbacreate	Boolean	如果语言是由数据库所有者创建的，则为真。
tmplhandler	text	调用处理器函数的名称。
tmplinline	text	匿名块处理器的名称，若没有则为 NULL。
tmplvalidator	text	校验函数的名称，如果没有则为 NULL。
tmpllibrary	text	实现语言的共享库的路径。
tmplacl	aclitem[]	模板的访问权限（未使用）。

14.2.48. PG_PROC

PG_PROC 系统表存储函数或过程的信息。

表 14-49. PG_PROC 字段

名称	类型	描述
oid	oid	行标识符（隐藏属性，必须明确选择）。
proname	name	函数名称。
pronamespace	oid	包含该函数名称空间的 OID。
proowner	oid	函数的所有者。

名称	类型	描述
prolang	oid	这个函数的实现语言或调用接口。
procost	real	估算的执行成本。
prorows	real	估算的影响行的数目。
provariadic	oid	参数元素的数据类型。
protransform	regproc	此函数的简化调用方式。
prosecdef	Boolean	函数是一个安全定义器（也就是一个“setuid”函数）。
proleakproof	Boolean	函数没副作用。如果函数没有对参数进行防泄露处理，则会抛出错误。
proisstrict	Boolean	如果任何调用参数是空，则函数返回空。这时函数实际上连调用都不调用。不是“strict”的函数必须准备处理空输入。
proretset	Boolean	函数返回一个集合（也就是说，指定数据类型的多个数值）。
provolatile	"char"	告诉该函数的结果是否只倚赖于它的输入参数，或者还会被外接因素影响。 • 对于“不可变的”（immutable）它是 i，这样的函数对于相同的输入总是产生相同的结果。 • 对于“稳定的”（stable）函数它是 s，（对于固定输入）其结果在一次扫描里不变。 • 对于“易变”（volatile）函数它是 v，其结果可能在任何时候变化 v 也用于那些有副作用的函数，因此调用它们无法得到优化。
pronargs	smallint	参数数目。
pronargdefaults	smallint	有默认值的参数数目。
prorettype	oid	返回值的数据类型。
proargtypes	oidvector	一个存放函数参数的数据类型的数组。数组里只包括输入参数（包括 INOUT 参数）此代表该函数的调用签名（接口）。
proallargtypes	oid[]	一个包含函数参数的数据类型的数组。数组里包括所有参数的类型（包括 OUT 和 INOUT 参数），如果所有参数都是 IN 参数，则这个字段就会是空。请注意数组下标是以 1 为起点的，

名称	类型	描述
		而因为历史原因，proargtypes 的下标起点为 0。
proargmodes	"char"[]	一个保存函数参数模式的数组，编码如下：i 表示 IN 参数，o 表示 OUT 参数，b 表示 INOUT 参数。如果所有参数都是 IN 参数，则这个字段为空。请注意，下标对应的是 proallargtypes 的位置，而不是 proargtypes。
proargnames	text[]	一个保存函数参数的名称的数组。没有名称的参数在数组里设置为空字符串。如果没有一个参数有名称，这个字段将是空。请注意，此数组的下标对应 proallargtypes 而不是 proargtypes。
proargdefaults	pg_node_tree	默认值的表达式树。是 PRONARGDEFAULTS 元素的列表。
prosrc	text	描述函数或存储过程的定义。例如，对于解释型语言来说就是函数的源程序，或者一个链接符号，一个文件名，或者函数和存储过程创建时指定的其他任何函数体内容，具体取决于语言/调用习惯的实现。
probin	text	关于如何调用该函数的附加信息。同样，其含义也是和语言相关的。
proconfig	text[]	函数针对运行时配置变量的本地设置。
proacl	aclitem[]	访问权限。
prodefaultargpos	int2vector	函数默认值的位置，不局限于能最后几个参数才可有默认值。
fencedmode	Boolean	函数的执行模式，表示函数是在 fence 还是 not fence 模式下执行。如果是 fence 执行模式，函数的执行会在重新 fork 的进程中执行。默认值是 fence。
proshippable	Boolean	
propackage	Boolean	表示该函数是否支持重载，默认值是 false。
prokind	"char"	显示该函数或过程的类型

14.2.49. PG_RANGE

PG_RANGE 系统表存储关于范围类型的信息。除了 14.2.66PG_TYPE 里类型的记录。

表 14-50. PG_RANGE 字段

名称	类型	引用	描述
rngtypid	oid	14.2.66PG_TYPE.oid	范围类型的 OID。
rngsubtype	oid	14.2.66PG_TYPE.oid	这个范围类型的元素类型(子类型)的 OID。
rngcollation	oid	14.2.19PG_COLLATION.oid	用于范围比较的排序规则的 OID，如果没有则为零。
rngsubopc	oid	14.2.43PG_OPCLASS.oid	用于范围比较的子类型的操作符类的 OID。
rngcanonical	regproc	14.2.48PG_PROC.proname	转换范围类型为规范格式的函数名，如果没有则为 0。
rngsubdiff	regproc	14.2.48PG_PROC.proname	返回两个 double precision 元素值的不同的函数名，如果没有则为 0。

rngsubopc（如果元素类型是可排序的，则加上 rngcollation）决定用于范围类型的排序顺序。当元素类型是离散的时使用 rngcanonical。

14.2.50. PG_RESOURCE_POOL

PG_RESOURCE_POOL 系统表提供了数据库资源池的信息。

表 14-51. PG_RESOURCE_POOL 字段

名称	类型	描述
oid	oid	行标识符（隐藏属性，必须明确选择）。
respool_name	name	资源池名称。
mem_percent	integer	内存配置的百分比。
cpu_affinity	bigint	CPU 绑定 core 的数值。
control_group	name	资源池所在的 control group 名称。
active_statements	integer	资源池上最大的并发数。
max_dop	integer	最大并发度。
memory_limit	name	资源池最大的内存。
parentid	oid	父资源池 OID。

名称	类型	描述
io_limits	integer	每秒触发 IO 的次数上限。行存单位是万次/s，列存是次/s。
io_priority	name	IO 利用率高达 90%时，重消耗 IO 作业进行 IO 资源管控时关联的优先级等级。
nodegroup	name	表示资源池所在的逻辑 Vastbase 的名称。
is_foreign	boolean	表示资源池是否用于逻辑 Vastbase 之外的用户。如果为 true，表示资源池用来控制不属于当前资源池的普通用户的资源。

14.2.51.PG_REWRITE

PG_REWRITE 系统表存储为表和视图定义的重写规则。

表 14-52. PG_REWRITE 字段

名称	类型	描述
oid	oid	行标识符（隐藏属性，必须明确选择）。
rulename	name	规则名称。
ev_class	oid	使用这条规则的表名称。
ev_attr	smallint	这条规则适用的字段（目前总是为零，表示整个表）。
ev_type	"char"	规则适用的事件类型： • 1 = SELECT • 2 = UPDATE • 3 = INSERT • 4 = DELETE
ev_enabled	"char"	用于控制复制的触发。 • O = “origin”和“local”模式时触发。 • D =禁用触发。 • R = “replica”时触发。 • A = 任何模式是都会触发。
is_instead	Boolean	如果该规则是 INSTEAD 规则，则为真。

名称	类型	描述
ev_qual	pg_node_tree	规则的资格条件的表达式树（以 nodeToString() 形式存在）。
ev_action	pg_node_tree	规则动作的查询树（以 nodeToString() 形式存在）。

14.2.52. PG_RLSPOLICY

PG_RLSPOLICY 系统表存储行级访问控制策略。

表 14-53. PG_RLSPOLICY 字段

名称	类型	描述
oid	oid	行标识符（隐藏属性，必须明确选择）。
polname	name	行级访问控制策略的名称。
polrelid	oid	行级访问控制策略作用的表对象 oid。
polcmd	"char"	行级访问控制策略影响的 SQL 操作。
polpermissive	boolean	行级访问控制策略的属性，t 为表达式 OR 条件拼接，f 为表达式 AND 条件拼接。
polroles	oid[]	行级访问控制策略影响的用户 oid 列表，不指定表示影响所有的用户。
polqual	pg_node_tree	行级访问控制策略的表达式。
polwithcheck	pg_node_tree	

14.2.53. PG_SECLABEL

PG_SECLABEL 系统表存储数据对象上的安全标签。

14.2.56PG_SHSECLABEL 的作用类似，只是它是用于在一个 Vastbase 内共享的数据库对象的安全标签上的。

表 14-54. PG_SECLABEL 字段

名称	类型	引用	描述
objoid	oid	任意 OID 属性	这个安全标签所属的对象的 OID。

名称	类型	引用	描述
classoid	oid	14.2.18PG_CLASS.oid	出现这个对象的系统目录的 OID。
objsubid	integer	-	出现在这个对象中的列的序号。
provider	text	-	与这个标签相关的标签提供程序。
label	text	-	应用于这个对象的安全标签。

14.2.54. PG_SHDEPEND

PG_SHDEPEND 系统表记录数据库对象和共享对象（比如角色）之间的依赖性关系。这些信息允许 Vastbase 保证在企图删除这些对象之前，这些对象是没有被引用的。

14.2.25PG_DEPEND 的作用类似，只是它是用于在一个数据库内部的对象的依赖性关系的。

和其它大多数系统表不同，PG_SHDEPEND 是在 Vastbase 里面所有的数据库之间共享的：每个 Vastbase 只有一个 PG_SHDEPEND，而不是每个数据库一个。

表 14-55. PG_SHDEPEND 字段

名称	类型	引用	描述
dbid	oid	14.2.22PG_DATABASE.oid	依赖对象所在的数据库的 OID ，如果是共享对象，则为零。
classid	oid	14.2.18PG_CLASS.oid	依赖对象所在的系统表的 OID。
objid	oid	任意 OID 属性	指定的依赖对象的 OID。
objsubid	integer	-	对于一个表字段，这是字段号（objid 和 classid 参考表本身）。对于所有其他对象类型，这个字段为零。
refclassid	oid	14.2.18PG_CLASS.oid	被引用对象所在的系统表的 OID(必须是一个共享表)。
refobjid	oid	任意 OID 属性	指定的被引用对象的 OID。
deptype	"char"	-	一段代码，定义了这个依赖性关系的特定语义；参阅下文。
objfile	text	-	用户定义函数库文件路径。

在任何情况下，一条 PG_SHDEPEND 记录就表明这个被引用的对象不能在未删除依赖对象的前提下删除。不过，deptype 同时还标出了几种不同的子风格：

- SHARED_DEPENDENCY_OWNER (o)

  被引用的对象（必须是一个角色）是依赖对象的所有者。

- SHARED_DEPENDENCY_ACL (a)

  被引用的对象（必须是一个角色）在依赖对象的 ACL（访问控制列表，也就是权限列表）里提到。SHARED_DEPENDENCY_ACL 不会在对象的所有者头上添加的，因为所有者会有一个 SHARED_DEPENDENCY_OWNER 记录。

- SHARED_DEPENDENCY_PIN (p)

  没有依赖对象；这类记录标识系统自身依赖于该被依赖对象，因此这样的对象绝对不能被删除。这种类型的记录只是由 initdb 创建。这样的依赖对象的字段都是零。

14.2.55. PG_SHDESCRIPTION

PG_SHDESCRIPTION 系统表为共享数据库对象存储可选的注释。可以使用 COMMENT 命令操作注释的内容，使用 vsql 的\d 命令查看注释内容。

PG_DESCRIPTION 提供了类似的功能，它记录了单个数据库中对象的注释。

不同于大多数系统表，PG_SHDESCRIPTION 是在 Vastbase 里面所有的数据库之间共享的：每个 Vastbase 只有一个 PG_SHDESCRIPTION，而不是每个数据库一个。

表 14-56. PG_SHDESCRIPTION 字段

名称	类型	引用	描述
objoid	oid	任意 OID 属性	这条描述所描述的对象的 OID。
classoid	oid	14.2.18PG_CLASS.oid	这个对象出现的系统表的 OID。
description	text	-	作为对该对象的描述的任意文本。

14.2.56. PG_SHSECLABEL

PG_SHSECLABEL 系统表存储在共享数据库对象上的安全标签。安全标签可以用 SECURITY LABEL 命令操作。

查看安全标签的简单点的方法，请参阅 14.3.34PG_SECLABELS。

14.2.53PG_SECLABEL 的作用类似，只是它是用于在单个数据库内部的对象的安全标签的。

不同于大多数的系统表，PG_SHSECLABEL 在 Vastbase 中的所有数据库中共享：每个 Vastbase 只有一个 PG_SHSECLABEL，而不是每个数据库一个。

表 14-57. PG_SHSECLABEL 字段

名称	类型	引用	描述
objoid	oid	任意 OID 属性	这个安全标签所属的对象的 OID。
classoid	oid	14.2.18PG_CLASS.oid	出现这个对象的系统目录的 OID。
provider	text	-	与这个标签相关的标签提供程序。
label	text	-	应用于这个对象的安全标签。

14.2.57.PG_STATISTIC

PG_STATISTIC 系统表存储有关该数据库中表和索引列的统计数据。默认只有系统管理员权限才可以访问此系统表，普通用户需要授权才可以访问。

表 14-58. PG_STATISTIC 字段

名称	类型	描述
starelid	oid	所描述的字段所属的表或者索引。
starelkind	"char"	所属对象的类型。
staattnum	smallint	所描述的字段在表中的编号，从 1 开始。
stainherit	Boolean	是否统计有继承关系的对象。
stanullfrac	real	该字段中为 NULL 的记录的比率。
stawidth	integer	非 NULL 记录的平均存储宽度，以字节计。
stadistinct	real	标识全局统计信息中数据库节点上字段里唯一的非 NULL 数据值的数目。 • 一个大于零的数值是独立数值的实际数目。 • 一个小于零的数值是表中行数的分数的负数（比如，一个字段的数值平均出现概率为两次，则可以表示为 stadistinct=-0.5）。 • 零值表示独立数值的数目未知。
stakindN	smallint	一个编码，表示这种类型的统计存储在 pg_statistic 行的第 n 个“槽位”。 n 的取值范围：1 ~ 5
staopN	oid	一个用于生成这些存储在第 n 个“槽位”的统计信息的操作符。比如，

名称	类型	描述
		一个柱面图槽位会显示<操作符，该操作符定义了该数据的排序顺序。 n 的取值范围：1～5
stanumbersN	real[]	第 n 个“槽位”的相关类型的数值类型统计，如果该槽位和数值类型没有关系，则就是 NULL。 n 的取值范围：1～5
stavaluesN	anyarray	第 n 个“槽位”类型的字段数据值，如果该槽位类型不存储任何数据值，则就是 NULL。每个数组的元素值实际上都是指定字段的数据类型，因此，除了把这些字段的类型定义成 anyarray 之外，没有更好的办法。 n 的取值范围：1～5
stadndistinct	real	标识 dn1 上字段里唯一的非 NULL 数据值的数目。 • 一个大于零的数值是独立数值的实际数目。 • 一个小于零的数值是表中行数的分数的负数（比如，一个字段的数值平均出现概率为两次，则可以表示为 stadistinct=-0.5）。 • 零值表示独立数值的数目未知。
staextinfo	text	统计信息的扩展信息。预留字段。

14.2.58. PG_STATISTIC_EXT

PG_STATISTIC_EXT 系统表存储有关该数据库中表的扩展统计数据，包括多列统计数据和表达式统计数据（后续支持）。收集哪些扩展统计数据是由用户指定的。需要有系统管理员权限才可以访问此系统表。

表 14-59. PG_STATISTIC_EXT 字段

名称	类型	描述
starelid	oid	所描述的字段所属的表或者索引。
starelkind	"char"	所属对象的类型。
stainherit	Boolean	是否统计有继承关系的对象。
stanullfrac	real	该字段中为 NULL 的记录的比率。
stawidth	integer	非 NULL 记录的平均存储宽度，以字节计。
stadistinct	real	标识全局统计信息中数据库节点上字段里唯一的非 NULL 数据值的数

名称	类型	描述
		目。 • 一个大于零的数值是独立数值的实际数目。 • 一个小于零的数值是表中行数的分数的负数（比如，一个字段的数值平均出现概率为两次，则可以表示为 stadistinct=-0.5）。 • 零值表示独立数值的数目未知。
stadndistinct	real	标识 dn1 上字段里唯一的非 NULL 数据值的数目。 • 一个大于零的数值是独立数值的实际数目。 • 一个小于零的数值是表中行数的分数的负数（比如，一个字段的数值平均出现概率为两次，则可以表示为 stadistinct=-0.5）。 • 零值表示独立数值的数目未知。
stakindN	smallint	一个编码，表示这种类型的统计存储在 pg_statistic 行的第 n 个“槽位”。 n 的取值范围：1 ~ 5
staopN	oid	一个用于生成这些存储在第 n 个“槽位”的统计信息的操作符。比如，一个柱面图槽位会显示<操作符，该操作符定义了该数据的排序顺序。 n 的取值范围：1 ~ 5
stakey	int2vector	所描述的字段编号的数组。
stanumbersN	real[]	第 n 个“槽位”的相关类型的数值类型统计，如果该槽位和数值类型没有关系，则就是 NULL。 n 的取值范围：1 ~ 5
stavaluesN	anyarray	第 n 个“槽位”类型的字段数据值，如果该槽位类型不存储任何数据值，则就是 NULL。每个数组的元素值实际上都是指定字段的数据类型，因此，除了把这些字段的类型定义成 anyarray 之外，没有更好的办法。 n 的取值范围：1 ~ 5
staexprs	pg_node_tree	扩展统计信息对应的表达式。

14.2.59. PG_TABLESPACE

PG_TABLESPACE 系统表存储表空间信息。

表 14-60. PG_TABLESPACE 字段

名称	类型	描述
oid	oid	行标识符（隐藏属性，必须明确选择）。
spcname	name	表空间名称。
spcowner	oid	表空间的所有者，通常是创建它的人。
spcacl	aclitem[]	访问权限。具体请参见 11.16.88GRANT 和 11.16.99REVOKE。
spcoptions	text[]	表空间的选项。
spcmaxsize	text	可使用的最大磁盘空间大小，单位 Byte。
relative	boolean	标识表空间指定的存储路径是否为相对路径。

14.2.60. PG_TRIGGER

PG_TRIGGER 系统表存储触发器信息。

表 14-61. PG_TRIGGER 字段

名称	类型	描述
oid	oid	行标识符（隐藏属性，必须明确选择）。
tgrelid	oid	触发器所在表的 OID。
tgname	name	触发器名。
tgfoid	oid	触发器 OID。
tgtype	smallint	触发器类型。
tgenabled	"char"	O =触发器在“origin”和“local”模式下触发。 D =触发器被禁用。 R =触发器在“replica”模式下触发。 A =触发器始终触发。
tgisinternal	boolean	内部触发器标识，如果为 true 表示内部触发器。
tgconstrrelid	oid	完整性约束引用的表。

名称	类型	描述
tgconstrindid	oid	完整性约束的索引。
tgconstraint	oid	约束触发器在 pg_constraint 中的 OID。
tgdeferrable	boolean	约束触发器是为 DEFERRABLE 类型。
tginitdeferred	boolean	约束触发器是否为 INITIALLY DEFERRED 类型。
tgnargs	smallint	触发器函数入参个数。
tgattr	int2vector	当触发器指定列时的列号，未指定则为空数组。
tgargs	bytea	传递给触发器的参数。
tgqual	pg_node_tree	表示触发器的 WHEN 条件，如果没有则为 null。
tgowner	oid	

14.2.61.PG_TS_CONFIG

PG_TS_CONFIG 系统表包含表示文本搜索配置的记录。一个配置指定一个特定的文本搜索解析器和一个为了每个解析器的输出类型使用的字典的列表。

解析器在 PG_TS_CONFIG 记录中显示，但是字典映射的标记是由 14.2.62PG_TS_CONFIG_MAP 里面的辅助记录定义的。

表 14-62. PG_TS_CONFIG 字段

名称	类型	引用	描述
oid	oid	-	行标识符（隐藏属性，必须明确选择）。
cfgname	name	-	文本搜索配置名。
cfgnamespace	oid	14.2.41PG_NAMESPACE.oid	包含这个配置的名称空间的 OID。
cfgowner	oid	14.2.14PG_AUTHID.oid	配置的所有者。
cfgparser	oid	14.2.64PG_TS_PARSER.oid	这个配置的文本搜索解析器的 OID。
cfoptions	text[]	-	分词相关配置选项。

14.2.62. PG_TS_CONFIG_MAP

PG_TS_CONFIG_MAP 系统表包含为每个文本搜索配置的解析器的每个输出符号类型，显示哪个文本搜索字典应该被咨询、以什么顺序搜索的记录。

表 14-63. PG_TS_CONFIG_MAP 字段

名称	类型	引用	描述
mapcfg	oid	14.2.61PG_TS_CONFIG.oid	拥有这个映射记录的 14.2.61PG_TS_CONFIG 记录的 OID。
maptokentype	integer	-	由配置的解析器发出的一个符号类型。
mapseqno	integer	-	以什么顺序咨询这个记录。
mapdict	oid	14.2.63PG_TS_DICT.oid	要咨询的文本搜索字典的 OID。

14.2.63. PG_TS_DICT

PG_TS_DICT 系统表包含定义文本搜索字典的记录。字典取决于文本搜索模板，该模板声明所有需要的实现函数；字典本身提供模板支持的用户可设置的参数的值。

这种分工允许字典通过非权限用户创建。参数由文本字符串 dictinitoption 指定，参数的格式和意义取决于模板。

表 14-64. PG_TS_DICT 字段

名称	类型	引用	描述
oid	oid	-	行标识符（隐藏属性，必须明确选择）。
dictname	name	-	文本搜索字典名。
dictnamespace	oid	14.2.41PG_NAMESPACE.oid	包含这个字典的名称空间的 OID。
dictowner	oid	14.2.14PG_AUTHID.oid	字典的所有者。
dicttemplate	oid	14.2.65PG_TS_TEMPLATE.oid	这个字典的文本搜索模板的 OID。
dictinitoption	text	-	该模板的初始化选项字符串。

14.2.64. PG_TS_PARSER

PG_TS_PARSER 系统表包含定义文本解析器的记录。解析器负责分裂输入文本为词位，并且为每个词位分配标记类型。新解析器必须由数据库系统管理员创建。

表 14-65. PG_TS_PARSER 字段

名称	类型	引用	描述
oid	oid	-	行标识符（隐藏属性；必须明确选择）。
prsname	name	-	文本搜索解析器名。
prsnamespace	oid	14.2.41PG_NAMESPACE.oid	包含这个解析器的名称空间的 OID。
prsstart	regproc	14.2.48PG_PROC.proname	解析器的启动函数名。
prstoken	regproc	14.2.48PG_PROC.proname	解析器的下一个标记函数名。
prsend	regproc	14.2.48PG_PROC.proname	解析器的关闭函数名。
prsheadline	regproc	14.2.48PG_PROC.proname	解析器的标题函数名。
prslextype	regproc	14.2.48PG_PROC.proname	解析器的 lextype 函数名。

14.2.65. PG_TS_TEMPLATE

PG_TS_TEMPLATE 系统表包含定义文本搜索模板的记录。模板是文本搜索字典的类的实现框架。因为模板必须通过 C 语言级别的函数实现，索引新模板的创建必须由数据库系统管理员创建。

表 14-66. PG_TS_TEMPLATE 字段

名称	类型	引用	描述
oid	oid	-	行标识符（隐藏属性；必须明确选择）。
tmplname	name	-	文本搜索模板名。
tmplnamespace	oid	14.2.41PG_NAMESPACE.oid	包含这个模板的名称空间的 OID。

名称	类型	引用	描述
tmplinit	regproc	14.2.48PG_PROC.proname	模板的初始化函数名。
tmpllexize	regproc	14.2.48PG_PROC.proname	模板的 lexize 函数名。

14.2.66.PG_TYPE

PG_TYPE 系统表存储数据类型的相关信息。

表 14-67. PG_TYPE 字段

名称	类型	描述
oid	oid	行标识符（隐藏属性，必须明确选择）。
typname	name	数据类型名称。
typnamespace	oid	包含这个类型的名称空间的 OID。
typowner	oid	该类型的所有者。
typlen	smallint	对于定长类型是该类型内部表现形式的字节数目。对于变长类型是负数。 • -1 表示一种“变长”（有长度字属性的数据）。 • -2 表示这是一个 NULL 结尾的 C 字符串。
typbyval	Boolean	指定内部传递这个类型的数值时是传值还是传引用。如果该类型的 TYPLEN 不是 1，2，4，8， TYPBYVAL 最好为假。变长类型通常是传引用。即使 TYPLEN 允许传值，TYPBYVAL 也可以为假。
typtype	"char"	• 对于基础类型是 b。 • 对于复合类型是 c（比如，一个表的行类型）。 • 对于域类型是 d。 • 对于伪类型是 p。 参见 typrelid 和 typbasetype。
typcategory	"char"	是数据类型的模糊分类，可用于解析器做为数据转换的依据。
typispreferred	Boolean	如果为真，则数据符合 TYPCATEGORY 所指定的转换规则时进行转换。

名称	类型	描述
typisdefined	Boolean	如果定义了类型则为真，如果是一种尚未定义的类型的占位符则为假。如果为假，则除了该类型名称，名称空间和 OID 之外没有可靠的信息。
typdelim	"char"	当分析数组输入时，分隔两个此类型数值的字符请注意该分隔符是与数组元素数据类型相关联的，而不是和数组数据类型关联。
typrelid	oid	如果是复合类型（请参见 typtype），则这个字段指向 pg_class 中定义该表的行。对于自由存在的复合类型，pg_class 记录并不表示一个表，但是总需要它来查找该类型连接的 pg_attribute 记录。对于非复合类型为零。
typelem	oid	如果不为 0，则它标识 pg_type 里面的另外一行。当前类型可以当做一个产生类型为 typelem 的数组来描述。一个"真正的"数组类型是变长的（typlen= -1），但是一些定长的（typlen > 0）类型也拥有非零的 typelem（比如 name 和 point）。如果一个定长类型拥有一个 typelem ，则他的内部形式必须是 typelem 数据类型的某个数目的个数值，不能有其他数据。变长数组类型有一个该数组子过程定义的头（文件）。
typarray	oid	如果不为 0，则表示在 pg_type 中有对应的类型记录。
typinput	regproc	输入转换函数（文本格式）。
typoutput	regproc	输出转换函数（文本格式）。
typreceive	regproc	输入转换函数（二进制格式），如果没有则为 0。
typsend	regproc	输出转换函数（二进制格式），如果没有则为 0。
typmodin	regproc	输入类型修改符函数，如果为 0，则不支持。
typmodout	regproc	输出类型修改符函数，如果为 0，则不支持。
typanalyze	regproc	自定义的 ANALYZE 函数，如果使用标准函数，则为 0。
typalign	"char"	当存储此类型的数值时要求的对齐性质。它应用于磁盘存储以及该值在 PostgreSQL 内部的大多数形式。如果数值是连续存放的，比如在磁盘上以完全的裸数据的形式存放时，则先在此类型的数据前填充空白，这样它就可以按照要求的界限存储。对齐引用是该序列中第一个数据的开头。可能的值包含： • c = char 对齐，也就是不需要对齐。

名称	类型	描述
		• s = short 对齐（在大多数机器上是 2 字节） • i = int 对齐（在大多数机器上是 4 字节） • d = double 对齐（在大多数机器上是 8 字节，但不一定是全部） 须知 对于在系统表里使用的类型，在 pg_type 里定义的尺寸和对齐必须和编译器在一个表示表的一行的结构里的布局一样。
typstorage	"char"	指明一个变长类型（那些有 typlen = -1）是否准备好应付非常规值，以及对这种属性的类型的缺省策略是什么。可能的值包含： • p：数值总是以简单方式存储。 • e：数值可以存储在一个"次要"关系中（如果该关系有这么一个，请参见 pg_class.reltoastrelid）。 • m：数值可以以内联的压缩方式存储。 • x：数值可以以内联的压缩方式或者在"次要"表里存储。 须知 m 域也可以移到从属表里存储，但只是最后的解决方法（e 和 x 域先移走）。
typenotnull	Boolean	代表在某类型上的一个 NOTNULL 约束。目前只用于域。
typbasetype	oid	如果这是一个衍生类型（请参见 typtype），则该标识作为这个类型的基础的类型。如果不是衍生类型则为零。
typtypmod	integer	域使用 typtypmod 记录要作用到它们的基础类型上的 typmod（如果基础类型不使用 typmod 则为-1）。如果这种类型不是域，则为-1。
typndims	integer	如果一个域是数组，则 typndims 是数组维数的数值（也就是说，typbasetype 是一个数组类型；域的 typelem 将匹配基本类型的 typelem）。非域非数组域为零。
typcollation	oid	指定类型的排序规则。如果为 0，则表示不支持排序。
typdefaultbin	pg_node_tree	如果为非 NULL，则它是该类型缺省表达式的 nodeToString() 表现形式。目前这个字段只用于域。
typdefault	text	如果某类型没有相关缺省值，则取值是 NULL。如果 typdefaultbin 不是 NULL，则 typdefault 必须包含一个 typdefaultbin 代表的缺省表达式的人类可读的版本。如果 typdefaultbin 为 NULL 但 typdefault 不是，typdefault 则是该类型缺省值的外部表现形式，可以把它交给该类型的

名称	类型	描述
		输入转换器生成一个常量。
typacl	aclitem[]	访问权限。

14.2.67. PG_USER_MAPPING

PG_USER_MAPPING 系统表存储从本地用户到远程的映射。

需要有系统管理员权限才可以访问此系统表。普通用户可以使用视图 14.3.72PG_USER_MAPPINGS 进行查询。

表 14-68. PG_USER_MAPPING 字段

名称	类型	引用	描述
oid	oid	-	行标识符（隐藏属性，必须明确选择）。
umuser	oid	14.2.14PG_AUTHID.oid	被映射的本地用户的 OID，如果用户映射是公共的则为 0。
umserver	oid	14.2.32PG_FOREIGN_SERVER.oid	包含这个映射的外部服务器的 OID。
umoptions	text[]	-	用户映射指定选项，使用“keyword=value”格式的字符串。

14.2.68. PG_USER_STATUS

PG_USER_STATUS 系统表提供了访问数据库用户的状态。需要有系统管理员权限才可以访问此系统表

表 14-69. PG_USER_STATUS 字段

名称	类型	描述
roloid	oid	角色的标识。
failcount	integer	尝试失败次数。
locktime	timestamp with time zone	角色被锁定的时间点。

名称	类型	描述
rolstatus	smallint	角色的状态。 • 0：正常状态。 • 1：由于登录失败次数超过阈值被锁定了一定的时间。 • 2：被管理员锁定。
permspace	bigint	角色已经使用的永久表存储空间大小。
tempspace	bigint	角色已经使用的临时表存储空间大小。

14.2.69. PG_WORKLOAD_GROUP

PG_WORKLOAD_GROUP 系统表提供了数据库负载组的信息。

表 14-70. PG_WORKLOAD_GROUP 字段

名称	类型	描述
oid	oid	行标识符（隐藏属性，必须明确选择）。
workload_gpname	name	负载组名称。
respool_oid	oid	绑定到的资源池的 id。
act_statements	integer	负载组内最大的活跃语句数。

14.2.70. PLAN_TABLE_DATA

PLAN_TABLE_DATA 存储了用户通过执行 EXPLAIN PLAN 收集到的计划信息。与 PLAN_TABLE 视图不同的是 PLAN_TABLE_DATA 表存储了所有 session 和 user 执行 EXPLAIN PLAN 收集的计划信息。

表 14-71. PLAN_TABLE_DATA 字段

名称	类型	描述
session_id	text	表示插入该条数据的会话，由服务线程启动时间戳和服务线程 ID 组成。受非空约束限制。
user_id	oid	用户 ID，用于标识触发插入该条数据的用户。受非空约束限制。

名称	类型	描述
statement_id	varchar(30)	用户输入的查询标签。
plan_id	bigint	查询标识。
id	int	计划中的节点编号。
operation	varchar(30)	操作描述。
options	varchar(255)	操作选项。
object_name	name	操作对应的对象名，来自于用户定义。
object_type	varchar(30)	对象类型。
object_owner	name	对象所属 schema，来自于用户定义。
projection	varchar(4000)	操作输出的列信息。

说明

- PLAN_TABLE_DATA 中包含了当前节点所有用户、所有会话的数据，仅管理员有访问权限。普通用户可以通过 14.3.75PLAN_TABLE 视图查看属于自己的数据。
- PLAN_TABLE_DATA 中的数据是用户通过执行 EXPLAIN PLAN 命令后由系统自动插入表中，因此禁止用户手动对数据进行插入或更新，否则会引起表中的数据混乱。需要对表中数据删除时，建议通过 14.3.75PLAN_TABLE 视图。
- statement_id、object_name、object_owner 和 projection 字段内容遵循用户定义的大小写存储，其它字段内容采用大写存储。

14.3. 系统视图

14.3.1. GS_SESSION_CPU_STATISTICS

GS_SESSION_CPU_STATISTICS 视图显示和当前用户执行复杂作业正在运行时的负载管理 CPU 使用的信息。

表 14-72. GS_SESSION_CPU_STATISTICS 字段

名称	类型	描述
datid	oid	连接后端的数据库 OID。
usename	name	登录到该后端的用户名。

名称	类型	描述
pid	bigint	后端线程 ID。
start_time	timestamp with time zone	语句执行的开始时间。
min_cpu_time	bigint	语句在数据库节点上的最小 CPU 时间，单位为 ms。
max_cpu_time	bigint	语句在数据库节点上的最大 CPU 时间，单位为 ms。
total_cpu_time	bigint	语句在数据库节点上的 CPU 总时间，单位为 ms。
query	text	正在执行的语句
node_group	text	该字段不支持。
top_cpu_dn	text	cpu 使用量信息。

14.3.2. GS_SESSION_MEMORY_STATISTICS

GS_SESSION_MEMORY_STATISTICS 视图显示和当前用户执行复杂作业正在运行时的负载管理内存使用的信息。

表 14-73. GS_SESSION_MEMORY_STATISTICS 字段

名称	类型	描述
datid	oid	连接后端的数据库 OID。
usename	name	登录到该后端的用户名。
pid	bigint	后端线程 ID。
start_time	timestamp with time zone	语句执行的开始时间。
min_peak_memory	integer	语句在数据库节点上的最小内存峰值大小，单位 MB。
max_peak_memory	integer	语句在数据库节点上的最大内存峰值大小，单位 MB。
spill_info	text	语句在数据库节点上的下盘信息： None：数据库节点均未下盘。

名称	类型	描述
		All：数据库节点均下盘。 [a:b]：数量为 b 个数据库节点中有 a 个数据库节点下盘。
query	text	正在执行的语句。
node_group	text	该字段不支持。
top_mem_dn	text	mem 使用量信息。

14.3.3. GS_SQL_COUNT

GS_SQL_COUNT 视图显示数据库当前节点当前时刻执行的五类语句（SELECT、INSERT、UPDATE、DELETE、MERGE INTO）统计信息。

- 普通用户查询 GS_SQL_COUNT 视图仅能看到该用户当前节点的统计信息；管理员权限用户查询 GS_SQL_COUNT 视图则能看到所有用户当前节点的统计信息。
- 当 Vastbase 或该节点重启时，计数将清零，并重新开始计数 。
- 计数以节点收到的查询数为准，包括 Vastbase 内部进行的查询。

表 14-74. GS_SQL_COUNT 字段

名称	类型	描述
node_name	name	节点名称。
user_name	name	用户名。
select_count	bigint	select 语句统计结果。
update_count	bigint	update 语句统计结果。
insert_count	bigint	insert 语句统计结果。
delete_count	bigint	delete 语句统计结果。
mergeinto_count	bigint	MERGE INTO 语句统计结果。
ddl_count	bigint	DDL 语句的数量。
dml_count	bigint	DML 语句的数量。
dcl_count	bigint	DML 语句的数量。

名称	类型	描述
total_select_elapse	bigint	总 select 的时间花费（单位：微秒）。
avg_select_elapse	bigint	平均 select 的时间花费（单位：微秒）。
max_select_elapse	bigint	最大 select 的时间花费（单位：微秒）。
min_select_elapse	bigint	最小 select 的时间花费（单位：微秒）。
total_update_elapse	bigint	总 update 的时间花费（单位：微秒）。
avg_update_elapse	bigint	平均 update 的时间花费（单位：微秒）。
max_update_elapse	bigint	最大 update 的时间花费（单位：微秒）。
min_update_elapse	bigint	最小 update 的时间花费（单位：微秒）。
total_insert_elapse	bigint	总 insert 的时间花费（单位：微秒）。
avg_insert_elapse	bigint	平均 insert 的时间花费（单位：微秒）。
max_insert_elapse	bigint	最大 insert 的时间花费（单位：微秒）。
min_insert_elapse	bigint	最小 insert 的时间花费（单位：微秒）。
total_delete_elapse	bigint	总 delete 的时间花费（单位：微秒）。
avg_delete_elapse	bigint	平均 delete 的时间花费（单位：微秒）。
max_delete_elapse	bigint	最大 delete 的时间花费（单位：微秒）。
min_delete_elapse	bigint	最小 delete 的时间花费（单位：微秒）。

14.3.4. GS_WLM_OPERATOR_HISTORY

GS_WLM_OPERATOR_HISTORY 视图显示的是当前用户当前数据库主节点上执行作业结束后的算子的相关记录。

14.3.5. GS_WLM_OPERATOR_STATISTICS

GS_WLM_OPERATOR_STATISTICS 视图显示当前用户正在执行的作业的算子相关信息。

表 14-75. GS_WLM_OPERATOR_STATISTICS 的字段

名称	类型	描述
queryid	bigint	语句执行使用的内部 query_id。
pid	bigint	后端线程 id。
plan_node_id	integer	查询对应的执行计划的 plan node id。
plan_node_name	text	对应于 plan_node_id 的算子的名称。
start_time	timestamp with time zone	该算子处理第一条数据的开始时间。
duration	bigint	该算子到结束时候总的执行时间(ms)。
status	text	当前算子的执行状态，包括 finished 和 running。
query_dop	integer	当前算子执行时的并行度。
estimated_rows	bigint	优化器估算的行数信息。
tuple_processed	bigint	当前算子返回的元素个数。
min_peak_memory	integer	当前算子在 上的最小内存峰值(MB)。
max_peak_memory	integer	当前算子在数据库节点上的最大内存峰值(MB)。
average_peak_memory	integer	当前算子在数据库节点上的平均内存峰值(MB)。
memory_skew_percent	integer	当前算子在数据库节点间的内存使用倾斜率。
min_spill_size	integer	若发生下盘，数据库节点上下盘的最小数据量(MB)，默认为 0。
max_spill_size	integer	若发生下盘，数据库节点上下盘的最大数据量(MB)，默认为 0。
average_spill_size	integer	若发生下盘，数据库节点上下盘的平均数据量(MB)，默认为 0。
spill_skew_percent	integer	若发生下盘，数据库节点间下盘倾斜率。
min_cpu_time	bigint	该算子在数据库节点上的最小执行时间(ms)。
max_cpu_time	bigint	该算子在数据库节点上的最大执行时间(ms)。

名称	类型	描述
total_cpu_time	bigint	该算子在数据库节点上的总执行时间(ms)。
cpu_skew_percent	integer	数据库节点间执行时间的倾斜率。
warning	text	主要显示如下几类告警信息： • Sort/SetOp/HashAgg/HashJoin spill • Spill file size large than 256MB • Broadcast size large than 100MB • Early spill • Spill times is greater than 3 • Spill on memory adaptive • Hash table conflict

14.3.6. GS_WLM_PLAN_OPERATOR_HISTORY

GS_WLM_PLAN_OPERATOR_HISTORY视图显示的是当前用户数据库主节点上执行作业结束后的执行计划算子级的相关记录。

14.3.7. GS_WLM_REBUILD_USER_RESOURCE_POOL

该视图用于在当前连接节点上重建内存中用户的资源池信息，无输出。只是用于资源池信息缺失或者错乱时用作补救措施。

14.3.8. GS_WLM_RESOURCE_POOL

这是资源池上的一些统计信息。

表 14-76. GS_WLM_RESOURCE_POOL 的字段

名称	类型	描述
rpoid	oid	资源池的 OID。
respool	name	资源池的名称。
control_group	name	该字段不支持。

名称	类型	描述
parentid	oid	父资源池的 OID。
ref_count	integer	关联到该资源池上的作业数量。
active_points	integer	资源池上已经使用的点数。
running_count	integer	正在资源池上运行的作业数量。
waiting_count	integer	正在资源池上排队的作业数量。
io_limits	integer	资源池的 iops 上限。
io_priority	integer	资源池的 io 优先级。

14.3.9. GS_WLM_SESSION_HISTORY

GS_WLM_SESSION_HISTORY 视图显示当前用户在数据库主节点上执行作业结束后的负载管理记录。

表 14-77. GS_WLM_SESSION_HISTORY 的字段

名称	类型	描述
datid	oid	连接后端的数据库 OID。
dbname	text	连接后端的数据库名称。
schemaname	text	模式的名称。
nodename	text	语句执行的数据库节点名称。
username	text	连接到后端的用户名。
application_name	text	连接到后端的应用名。
client_addr	inet	连接到后端的客户端的 IP 地址。 如果此字段是 null，它表明通过服务器机器上 UNIX 套接字连接客户端或者这是内部进程，如 autovacuum。
client_hostname	text	客户端的主机名，这个字段是通过 client_addr 的反向 DNS 查找得到。这个字段只有在启动 log_hostname 且使用 IP 连接时才非空。

名称	类型	描述
client_port	integer	客户端用于与后端通讯的 TCP 端口号，如果使用 Unix 套接字，则为-1。
query_band	text	用于标示作业类型，可通过 GUC 参数 query_band 进行设置，默认为空字符串。
block_time	bigint	语句执行前的阻塞时间，包含语句解析和优化时间，单位 ms。
start_time	timestamp with time zone	语句执行的开始时间。
finish_time	timestamp with time zone	语句执行的结束时间。
duration	bigint	语句实际执行的时间，单位 ms。
estimate_total_time	bigint	语句预估执行时间，单位 ms。
status	text	语句执行结束状态：正常为 finished，异常为 aborted。
abort_info	text	语句执行结束状态为 aborted 时显示异常信息。
resource_pool	text	用户使用的资源池。
control_group	text	该字段不支持
estimate_memory	integer	语句估算内存大小。
min_peak_memory	integer	语句在数据库节点上的最小内存峰值，单位 MB。
max_peak_memory	integer	语句在数据库节点上的最大内存峰值，单位 MB。
average_peak_memory	integer	语句执行过程中的内存使用平均值，单位 MB。
memory_skew_percent	integer	语句数据库节点间的内存使用倾斜率。
spill_info	text	语句在数据库节点上的下盘信息： • None：数据库节点均未下盘。 • All: 数据库节点均下盘。 • [a:b]: 数量为 b 个数据库节点中有 a 个数据库节点下盘。
min_spill_size	integer	若发生下盘，数据库节点上下盘的最小数据量，单位 MB，默认为 0。
max_spill_size	integer	若发生下盘，数据库节点上下盘的最大数据量，单位 MB，

名称	类型	描述
		默认为 0。
average_spill_size	integer	若发生下盘，数据库节点上下盘的平均数据量，单位 MB，默认为 0。
spill_skew_percent	integer	若发生下盘，数据库节点间下盘倾斜率。
min_dn_time	bigint	语句在数据库节点上的最小执行时间，单位 ms。
max_dn_time	bigint	语句在数据库节点上的最大执行时间，单位 ms。
average_dn_time	bigint	语句在数据库节点上的平均执行时间，单位 ms。
dntime_skew_percent	integer	语句在数据库节点间的执行时间倾斜率。
min_cpu_time	bigint	语句在数据库节点上的最小 CPU 时间，单位 ms。
max_cpu_time	bigint	语句在数据库节点上的最大 CPU 时间，单位 ms。
total_cpu_time	bigint	语句在数据库节点上的 CPU 总时间，单位 ms。
cpu_skew_percent	integer	语句在数据库节点间的 CPU 时间倾斜率。
min_peak_iops	integer	语句在数据库节点上的每秒最小 IO 峰值（列存单位是次/s，行存单位是万次/s）。
max_peak_iops	integer	语句在数据库节点上的每秒最大 IO 峰值（列存单位是次/s，行存单位是万次/s）。
average_peak_iops	integer	语句在数据库节点上的每秒平均 IO 峰值（列存单位是次/s，行存单位是万次/s）。
iops_skew_percent	integer	语句在数据库节点间的 IO 倾斜率。
warning	text	主要显示如下几类告警信息： • Spill file size large than 256MB • Broadcast size large than 100MB • Early spill • Spill times is greater than 3 • Spill on memory adaptive • Hash table conflict

名称	类型	描述
queryid	bigint	语句执行使用的内部 query id。
query	text	执行的语句。
query_plan	text	语句的执行计划。
node_group	text	该字段不支持。
cpu_top1_node_name	text	当前数据库节点名称。
cpu_top2_node_name	text	该字段不支持。
cpu_top3_node_name	text	该字段不支持。
cpu_top4_node_name	text	该字段不支持。
cpu_top5_node_name	text	该字段不支持。
mem_top1_node_name	text	当前数据库节点名称。
mem_top2_node_name	text	该字段不支持。
mem_top3_node_name	text	该字段不支持。
mem_top4_node_name	text	该字段不支持。
mem_top5_node_name	text	该字段不支持。
cpu_top1_value	bigint	当前数据库节点 cpu 使用率。
cpu_top2_value	bigint	该字段不支持。
cpu_top3_value	bigint	该字段不支持。
cpu_top4_value	bigint	该字段不支持。
cpu_top5_value	bigint	该字段不支持。
mem_top1_value	bigint	当前数据库节点内存使用量。
mem_top2_value	bigint	该字段不支持。
mem_top3_value	bigint	该字段不支持。
mem_top4_value	bigint	该字段不支持。
mem_top5_value	bigint	该字段不支持。

名称	类型	描述
top_mem_dn	text	当前数据库节点内存使用量信息。
top_cpu_dn	text	当前数据库节点 cpu 使用量信息。

14.3.10.GS_WLM_SESSION_INFO_ALL

GS_WLM_SESSION_INFO_ALL 视图显示在数据库主节点上执行作业结束后的负载管理记录。

14.3.11.GS_WLM_USER_INFO

用户统计信息视图。

表 14-78. GS_WLM_USER_INFO 字段

名称	类型	描述
userid	oid	用户 OID。
username	name	用户名。
sysadmin	boolean	是否是管理员用户。
rpoid	oid	关联的资源池的 OID。
respool	name	关联的资源池的名称。
parentid	oid	用户组的 OID。
totalspace	bigint	用户的可用空间上限。
spacelimit	bigint	用户表空间限制。
childcount	interger	子用户的个数。
childlist	text	子用户列表。

14.3.12.GS_WLM_SESSION_STATISTICS

GS_WLM_SESSION_STATISTICS 视图显示当前用户在数据库主节点上正在执行的作业的负载管理记录。

表 14-79. GS_WLM_SESSION_STATISTICS 的字段

名称	类型	描述
datid	oid	连接后端的数据 OID。
dbname	name	连接后端的数据库名称。
schemaname	text	模式的名称。
nodename	text	语句执行的数据库主节点名称。
username	name	连接到后端的用户名。
application_name	text	连接到后端的应用名。
client_addr	inet	连接到后端的客户端的 IP 地址。 如果此字段是 null，它表明通过服务器机器上 UNIX 套接字连接客户端或者这是内部进程，如 autovacuum。
client_hostname	text	客户端的主机名，这个字段是通过 client_addr 的反向 DNS 查找得到。这个字段只有在启动 log_hostname 且使用 IP 连接时才非空。
client_port	integer	客户端用于与后端通讯的 TCP 端口号，如果使用 Unix 套接字，则为-1。
query_band	text	用于标示作业类型，可通过 GUC 参数 query_band 进行设置，默认为空字符串。
pid	bigint	后端线程 ID。
sessionid	bigint	会话 ID。
block_time	bigint	语句执行前的阻塞时间，单位 ms。
start_time	timestamp with time zone	语句执行的开始时间。
duration	bigint	语句已经执行的时间，单位 ms。
estimate_total_time	bigint	语句执行预估总时间，单位 ms。
estimate_left_time	bigint	语句执行预估剩余时间，单位 ms。
enqueue	text	该字段不支持。
resource_pool	name	用户使用的资源池。

名称	类型	描述
control_group	text	该字段不支持。
estimate_memory	integer	语句预估使用内存，单位 MB。
min_peak_memory	integer	语句在数据库节点上的最小内存峰值，单位 MB。
max_peak_memory	integer	语句在数据库节点上的最大内存峰值，单位 MB。
average_peak_memory	integer	语句执行过程中的内存使用平均值，单位 MB。
memory_skew_percent	integer	语句在数据库节点间的内存使用倾斜率。
spill_info	text	语句在数据库节点上的下盘信息： • None：数据库节点均未下盘。 • All：数据库节点均下盘。 • [a:b]：数量为 b 个数据库节点中有 a 个数据库节点下盘。
min_spill_size	integer	若发生下盘，数据库节点上下盘的最小数据量，单位 MB，默认为 0。
max_spill_size	integer	若发生下盘，数据库节点上下盘的最大数据量，单位 MB，默认为 0。
average_spill_size	integer	若发生下盘，数据库节点上下盘的平均数据量，单位 MB，默认为 0。
spill_skew_percent	integer	若发生下盘，数据库节点间下盘倾斜率。
min_dn_time	bigint	语句在数据库节点上的最小执行时间，单位 ms。
max_dn_time	bigint	语句在数据库节点上的最大执行时间，单位 ms。
average_dn_time	bigint	语句在数据库节点上的平均执行时间，单位 ms。
dntime_skew_percent	integer	语句在数据库节点间的执行时间倾斜率。
min_cpu_time	bigint	语句在数据库节点上的最小 CPU 时间，单位 ms。
max_cpu_time	bigint	语句在数据库节点上的最大 CPU 时间，单位 ms。
total_cpu_time	bigint	语句在数据库节点上的 CPU 总时间，单位 ms。
cpu_skew_percent	integer	语句在数据库节点间的 CPU 时间倾斜率。
min_peak_iops	integer	语句在数据库节点上的每秒最小 IO 峰值（列存单位是次/s，

名称	类型	描述
		行存单位是万次/s）。
max_peak_iops	integer	语句在数据库节点上的每秒最大 IO 峰值（列存单位是次/s，行存单位是万次/s）。
average_peak_iops	integer	语句在数据库节点上的每秒平均 IO 峰值（列存单位是次/s，行存单位是万次/s）。
iops_skew_percent	integer	语句在数据库节点间的 IO 倾斜率。
warning	text	主要显示如下几类告警信息： • Spill file size large than 256MB • Broadcast size large than 100MB • Early spill • Spill times is greater than 3 • Spill on memory adaptive • Hash table conflict
queryid	bigint	语句执行使用的内部 query id。
query	text	正在执行的语句。
query_plan	text	语句的执行计划。
node_group	text	该字段不支持。
top_mem_dn	text	内存使用量信息。
top_cpu_dn	text	cpu 使用量信息。

14.3.13.GS_STAT_DB_CU

GS_STAT_DB_CU视图查询Vastbase各个节点，每个数据库的CU命中情况。可以通过gs_stat_reset()进行清零。

表 14-80. GS_STAT_DB_CU 字段

名称	类型	描述
node_name1	text	节点名称。

名称	类型	描述
db_name	text	数据库名。
mem_hit	bigint	内存命中次数。
hdd_sync_read	bigint	硬盘同步读次数。
hdd_asyn_read	bigint	硬盘异步读次数。

14.3.14.GS_STAT_SESSION_CU

GS_STAT_SESSION_CU 视图查询 Vastbase 各个节点，当前运行 session 的 CU 命中情况。session 退出相应的统计数据会清零。Vastbase 重启后，统计数据也会清零。

表 14-81. GS_STAT_SESSION_CU 字段

名称	类型	描述
mem_hit	integer	内存命中次数。
hdd_sync_read	integer	硬盘同步读次数。
hdd_asyn_read	integer	硬盘异步读次数。

14.3.15.MPP_TABLES

MPP_TABLES 视图显示 PGXC_CLASS 中的表信息。

表 14-82. MPP_TABLES 字段

名称	类型	描述
schemaname	name	包含表的模式名。
tablename	name	表名。
tableowner	name	表的所有者。
tablespace	name	表所在的表空间。
pgroup	name	节点群的名称。
nodeoids	oidvector_extend	表分布的节点 OID 列表。

14.3.16. PG_AVAILABLE_EXTENSION_VERSIONS

PG_AVAILABLE_EXTENSION_VERSIONS 视图显示数据库中某些特性的扩展版本信息。

表 14-83. PG_AVAILABLE_EXTENSION_VERSIONS 字段

名称	类型	描述
name	name	扩展名。
version	text	版本名。
installed	Boolean	如果这个扩展的版本是当前已经安装了的则为真。
superuser	Boolean	如果只允许系统管理员安装这个扩展则为真。
relocatable	Boolean	如果扩展可以重新加载到另一个模式则为真。
schema	name	扩展必须安装到的模式名，如果部分或全部可重新定位则为 NULL。
requires	name[]	先决条件扩展的名称，如果没有则为 NULL。
comment	text	扩展的控制文件中的评论字符串。

14.3.17. PG_AVAILABLE_EXTENSIONS

PG_AVAILABLE_EXTENSIONS 视图显示数据库中某些特性的扩展信息。

表 14-84. PG_AVAILABLE_EXTENSIONS 字段

名称	类型	描述
name	name	扩展名。
default_version	text	缺省版本的名称，如果没有指定则为 NULL。
installed_version	text	扩展当前安装版本，如果没有安装任何版本则为 NULL。
comment	text	扩展的控制文件中的评论字符串。

14.3.18. PG_CURSORS

PG_CURSORS 视图列出了当前可用的游标。

表 14-85. PG_CURSORS 字段

名称	类型	描述
name	text	游标名。
statement	text	声明改游标时的查询语句。
is_holdable	Boolean	如果该游标是持久的（就是在声明该游标的事务结束后仍然可以访问该游标）则为 TRUE，否则为 FALSE。
is_binary	Boolean	如果该游标被声明为 BINARY 则为 TRUE，否则为 FALSE。
is_scrollable	Boolean	如果该游标可以滚动 (就是允许以不连续的方式检索) 则为 TRUE，否则为 FALSE。
creation_time	timestamp with time zone	声明该游标的时间戳。

14.3.19. PG_EXT_STATS

PG_EXT_STATS 视图提供对存储在 14.2.58PG_STATISTIC_EXT 表里面的扩展统计信息的访问。扩展统计信息目前包括多列统计信息。

表 14-86. PG_EXT_STATS 字段

名称	类型	引用	描述
schemaname	name	14.2.41PG_NAMESPACE.nspname	包含表的模式名。
tablename	name	14.2.18PG_CLASS.relname	表名。
attname	int2vector	14.2.58PG_STATISTIC_EXT.stakey	统计信息扩展的多列信息。
inherited	Boolean	-	如果为真，则包含继承的子列，否则只是指定表的字段。
null_frac	real	-	记录中字段组合为空的百分比。

名称	类型	引用	描述
avg_width	integer	-	字段组合记录以字节记的平均宽度。
n_distinct	real	-	• 如果大于零，表示字段组合中独立数值的估计数目。 • 如果小于零，表示独立数值的数目被行数除的负数。 1. 用负数形式是因为 ANALYZE 认为独立数值的数目是随着表增长而增长； 2. 正数的形式用于在字段看上去好像有固定的可能值数目的情况下。比如，-1 表示一个字段组合中独立数值的个数和行数相同。 • 如果等于零，表示独立数值的数目未知。
n_dndistinct	real	-	标识 dn1 上字段组合中非 NULL 数据的唯一值的数目。 • 如果大于零，表示独立数值的实际数目。 • 如果小于零，表示独立数值的数目被行数除的负数。（比如，一个字段组合的数值平均出现概率为两次，则可以表示为 n_dndistinct=-0.5）。 • 如果等于零，表示独立数值的数目未知。
most_common_vals	anyarray	-	一个字段组合里最常用数值的列表。如果该字段组合不存在最常用数值，则为 NULL。本列保存的多列常用数值均不为 NULL。
most_common_freqs	real[]	-	一个最常用数值组合的频率的列表，也就是说，每个出现的次数除以行数。如果 most_common_vals 是 NULL，则为 NULL。

名称	类型	引用	描述
most_common_vals_null	anyarray	-	一个字段组合里最常用数值的列表。如果该字段组合不存在最常用数值，则为 NULL。本列保存的多列常用数值中至少有一个值为 NULL。
most_common_freqs_null	real[]	-	一个最常用数值组合的频率的列表，也就是说，每个出现的次数除以行数。如果 most_common_vals_null 是 NULL，则为 NULL。
histogram_bounds	anyarray	-	直方图的边界值列表。

14.3.20. PG_GET_INVALID_BACKENDS

PG_GET_INVALID_BACKENDS 视图提供显示数据库主节点上连接到当前备机的后台线程信息。

表 14-87. PG_GET_INVALID_BACKENDS 字段

名称	类型	描述
pid	bigint	线程 ID。
node_name	text	后台线程中连接的节点信息。
dbname	name	当前连接的数据库。
backend_start	timestamp with time zone	后台线程启动的时间。
query	text	后台线程正在执行的查询语句。

14.3.21. PG_GET_SENDERS_CATCHUP_TIME

PG_GET_SENDERS_CATCHUP_TIME 视图显示数据库节点上当前活跃的主备发送线程的追赶信息。

表 14-88. PG_GET_SENDERS_CATCHUP_TIME 字段

名称	类型	描述
pid	bigint	当前 sender 的线程 ID。

名称	类型	描述
lwpid	integer	当前 sender 的 lwpid。
local_role	text	本地的角色。
peer_role	text	对端的角色。
state	text	当前 sender 的复制状态。
type	text	当前 sender 的类型。
catchup_start	timestamp with time zone	catchup 启动的时间。
catchup_end	timestamp with time zone	catchup 结束的时间。

14.3.22. PG_GROUP

PG_GROUP 视图查看数据库认证角色及角色之间的成员关系。

表 14-89. PG_GROUP 字段

名称	类型	描述
groname	name	组的名称。
grosysid	oid	组的 ID。
grolist	oid[]	一个数组，包含这个组里面所有角色的 ID。

14.3.23. PG_INDEXES

PG_INDEXES 视图提供对数据库中每个索引的有用信息的访问。

表 14-90. PG_INDEXES 字段

名称	类型	引用	描述
schemaname	name	14.2.41PG_NAMESPACE.nspname	包含表和索引的模式名称。
tablename	name	14.2.18PG_CLASS.relname	此索引所服务的表的名称。

名称	类型	引用	描述
indexname	name	14.2.18PG_CLASS.relname	索引的名称。
tablespace	name	14.2.59PG_TABLESPACE.nspname	包含索引的表空间名称。
indexdef	text	-	索引定义（一个重建的 CREATE INDEX 命令）。

14.3.24. PG_LOCKS

PG_LOCKS 视图存储各打开事务所持有的锁信息。

PG_LOCKS 字段

名称	类型	引用	描述
locktype	text	-	被锁定对象的类型：relation，extend，page，tuple，transactionid，virtualxid，object，userlock，advisory。
database	oid	14.2.22PG_DATABASE.oid	被锁定对象所在数据库的 OID。 • 如果被锁定的对象是共享对象，则 OID 为 0。 • 如果是一个事务 ID，则为 NULL。
relation	oid	14.2.18PG_CLASS.oid	关系的 OID，如果锁定的对象不是关系，也不是关系的一部分，则为 NULL。
page	integer	-	关系内部的页面编号，如果对象不是关系页或者不是行页，则为 NULL。
tuple	smallint	-	页面里边的行编号，如果对象不是行，则为 NULL。
bucket	integer		
virtualxid	text	-	事务的虚拟 ID，如果对象不是一个虚拟事务 ID，则为 NULL。
transactionid	xid	-	事务的 ID，如果对象不是一个事务 ID，则为 NULL。
classid	oid	14.2.18PG_CLASS.oid	包含该对象的系统表的 OID，如果对象不是普通

名称	类型	引用	描述
			的数据库对象，则为 NULL。
objid	oid	-	对象在其系统表内的 OID，如果对象不是普通数据库对象，则为 NULL。
objsubid	smallint	-	对于表的一个字段，这是字段编号；对于其他对象类型，这个字段是零；如果这个对象不是普通数据库对象，则为 NULL。
virtualtransaction	text	-	持有此锁或者在等待此锁的事务的虚拟 ID。
pid	bigint	-	持有或者等待这个锁的服务器线程的逻辑 ID。如果锁是被一个预备事务持有的，则为 NULL。
sessionid	bigint	-	持有或者等待这个锁的会话 ID。
mode	text	-	这个线程持有的或者是期望的锁模式。
granted	Boolean	-	• 如果锁是持有锁，则为 TRUE。 • 如果锁是等待锁，则为 FALSE。
fastpath	Boolean	-	如果通过 fast-path 获得锁，则为 TRUE；如果通过主要的锁表获得，则为 FALSE。

14.3.25. PG_NODE_ENV

PG_NODE_ENV 视图提供获取当前节点的环境变量信息。

表 14-91. PG_NODE_ENV 字段

名称	类型	描述
node_name	text	当前节点的名称。
host	text	当前节点的主机名称。
process	integer	当前节点的进程号。
port	integer	当前节点的端口号。
installpath	text	当前节点的安装目录。
datapath	text	当前节点的数据目录。

名称	类型	描述
log_directory	text	当前节点的日志目录。

14.3.26. PG_OS_THREADS

PG_OS_THREADS 视图提供当前节点下所有线程的状态信息。

表 14-92. PG_OS_THREADS 字段

名称	类型	描述
node_name	text	当前节点的名称。
pid	bigint	当前节点进程中正在运行的线程号。
lwpid	integer	与 pid 对应的轻量级线程号。
thread_name	text	与 pid 对应的线程名称。
creation_time	timestamp with time zone	与 pid 对应的线程创建的时间。

14.3.27. PG_PREPARED_STATEMENTS

PG_PREPARED_STATEMENTS 视图显示当前会话所有可用的预备语句。

表 14-93. PG_PREPARED_STATEMENTS 字段

名称	类型	描述
name	text	预备语句的标识符。
statement	text	创建该预备语句的查询字符串。对于从 SQL 创建的预备语句而言是客户端提交的 PREPARE 语句；对于通过前/后端协议创建的预备语句而言是预备语句自身的文本。
prepare_time	timestamp with time zone	创建该预备语句的时间戳。
parameter_types	regtype[]	该预备语句期望的参数类型，以 regtype 类型的数组格式出现。与该数组元素相对应的 OID 可以通过把 regtype 转换为 oid 值得到。

名称	类型	描述
from_sql	Boolean	• 如果该预备语句是通过 PREPARE 语句创建的则为 true。 • 如果是通过前/后端协议创建的则为 false。

14.3.28. PG_PREPARED_XACTS

PG_PREPARED_XACTS 视图显示当前准备好进行两阶段提交的事务的信息。

表 14-94. PG_PREPARED_XACTS 字段

名称	类型	引用	描述
transaction	xid	-	预备事务的数字事务标识。
gid	text	-	赋予该事务的全局事务标识。
prepared	timestamp with time zone	-	事务准备好提交的时间。
owner	name	14.2.14PG_AUTHID.rolname	执行该事务的用户的名称。
database	name	14.2.22PG_DATABASE.datname	执行该事务所在的数据库名。

14.3.29. PG_REPLICATION_SLOTS

PG_REPLICATION_SLOTS 视图查看复制节点的信息。

表 14-95. PG_REPLICATION_SLOTS 字段

名称	类型	描述
slot_name	text	复制节点的名称。
plugin	text	逻辑复制槽对应的输出插件名。
slot_type	text	复制节点的类型。
datoid	oid	复制节点的数据库 OID。
database	name	复制节点的数据库名称。

名称	类型	描述
active	Boolean	复制节点是否为激活状态。
xmin	xid	复制节点事务标识。
catalog_xmin	xid	逻辑复制槽对应的最早解码事务标识。
restart_lsn	text	复制节点的 Xlog 文件信息。
dummy_standby	Boolean	复制节点是否为假备。

14.3.30. PG_RLSPOLICIES

PG_RLSPOLICIES 视图提供查询行级访问控制策略。

表 14-96. PG_RLSPOLICIES 字段

名称	类型	描述
schemaname	name	行级访问控制策略作用的表对象所属模式名称。
tablename	name	行级访问控制策略作用的表对象名称。
policyname	name	行级访问控制策略名称。
policypermissive	text	行级访问控制策略属性。
policyroles	name[]	行级访问控制策略影响的用户列表，不指定表示影响所有的用户。
policycmd	text	行级访问控制策略影响的 SQL 操作。
policyqual	text	行级访问控制策略的表达式。

14.3.31. PG_ROLES

PG_ROLES 视图提供访问数据库角色的相关信息，默认只有初始化用户和具有 sysadmin 属性的用户可以查看，其他用户需要赋权后才可以查看。

表 14-97. PG_ROLES 字段

名称	类型	引用	描述
rolname	name	-	角色名称。

名称	类型	引用	描述
rolsuper	Boolean	-	该角色是否是拥有最高权限的初始系统管理员。
rolinherit	Boolean	-	该角色是否继承角色的权限。
rolcreaterole	Boolean	-	该角色是否可以创建其他的角色。
rolcreatedb	Boolean	-	该角色是否可以创建数据库。
rolcatupdate	Boolean	-	该角色是否可以直接更新系统表。只有 usesysid=10 的初始系统管理员拥有此权限。其他用户无法获得此权限。
rolcanlogin	Boolean	-	该角色是否可以登录数据库。
rolssoadmin	Boolean		该角色为安全管理员。
rolreplication	Boolean	-	该角色是否可以复制。
rolauditadmin	Boolean	-	该角色是否为审计管理员。
rolsystemadmin	Boolean	-	该角色是否为系统管理员。
rolconnlimit	integer	-	对于可以登录的角色，这里限制了该角色允许发起的最大并发连接数。 -1 表示无限制。
rolpassword	text	-	不是口令，总是********。
rolvalidbegin	timestamp with time zone	-	帐户的有效开始时间；如果没有设置有效开始时间，则为 NULL。
rolvaliduntil	timestamp with time zone	-	帐户的有效结束时间；如果没有设置有效结束时间，则为 NULL。
rolrespool	name	-	用户所能够使用的 resource pool。
rolparentid	oid	14.2.14PG_AUTHID.rolparentid	用户所在组用户的 OID。
roltabspace	text	-	用户永久表存储空间限额。
roltempspace	text	-	用户临时表存储空间限额。
rolspillspace	text	-	用户算子落盘空间限额。

名称	类型	引用	描述
rolconfig	text[]	-	运行时配置变量的会话缺省。
oid	oid	14.2.14PG_AUTHID.oid	角色的 ID。
roluseft	Boolean	14.2.14PG_AUTHID.roluseft	角色是否可以操作外表。
rolkind	"char"	-	角色类型。
nodegroup	name	-	该字段不支持。

14.3.32. PG_RULES

PG_RULES 视图提供对查询重写规则的有用信息访问的接口。

表 14-98. PG_RULES 字段

名称	类型	描述
schemaname	name	包含表的模式的名称。
tablename	name	规则作用的表的名称。
rulename	name	规则的名称。
definition	text	规则定义（一个重新构造的创建命令）。

14.3.33. PG_RUNNING_XACTS

PG_RUNNING_XACTS 视图主要功能是显示当前节点运行事务的信息。

表 14-99. PG_RUNNING_XACTS 字段

名称	类型	描述
handle	integer	事务在 GTM 对应的句柄。
gxid	xid	事务 id 号。
state	tinyint	事务状态（3：prepared 或者 0：starting）。

名称	类型	描述
node	text	节点名称。
xmin	xid	节点上当前数据涉及的最小事务号 xmin。
vacuum	Boolean	标志当前事务是否是 lazy vacuum 事务。
timeline	bigint	标志数据库重启次数。
prepare_xid	xid	处于 prepared 状态的事务的 id 号，若不在 prepared 状态，值为 0。
pid	bigint	事务对应的线程 id。
next_xid	xid	该字段不支持。

14.3.34. PG_SECLABELS

PG_SECLABELS 视图提供关于安全标签的信息。

表 14-100. PG_SECLABELS 字段

名称	类型	引用	描述
objoid	oid	任意 OID 属性	这个安全标签指向的对象的 OID。
classoid	oid	14.2.18PG_CLASS.oid	这个对象出现的系统表的 OID。
objsubid	integer	-	对于一个在表字段上的安全标签，是字段编号（引用表本身的 objoid 和 classoid）。对于所有其他对象类型，这个字段为零。
objtype	text	-	这个标签出现的对象的类型，文本格式。
objnamespace	oid	14.2.41PG_NAMESPACE.oid	这个对象的名称空间的 OID，如果适用；否则为 NULL。
objname	text	-	这个标签适用的对象的名称，文本格式。
provider	text	14.2.53PG_SECLABEL.provide	与这个标签相关的标签提供者。

名称	类型	引用	描述
		r	
label	text	14.2.53PG_SECLABEL.label	适用于这个对象的安全标签。

14.3.35.PG_SESSION_WLMSTAT

PG_SESSION_WLMSTAT 视图显示和当前用户执行作业正在运行时的负载管理相关信息。

表 14-101. PG_SESSION_WLMSTAT 字段

名称	类型	描述
datid	oid	连接后端的数据库 OID。
datname	name	连接后端的数据库名称。
threadid	bigint	后端线程 ID。
sessionid	bigint	会话 ID。
processid	integer	后端线程的 pid。
usesysid	oid	登录后端的用户 OID。
appname	text	连接到后端的应用名。
usename	name	登录到该后端的用户名。
priority	bigint	语句所在 Cgroups 的优先级。
attribute	text	语句的属性： • Ordinary：语句发送到数据库后被解析前的默认属性。 • Simple：简单语句。 • Complicated：复杂语句。 • Internal：数据库内部语句。
block_time	bigint	语句当前为止的 pending 的时间，单位 s。
elapsed_time	bigint	语句当前为止的实际执行时间，单位 s。
total_cpu_time	bigint	语句在上一时间周期内的数据库节点上 CPU 使用的总时间，单位 s。

名称	类型	描述
cpu_skew_percent	integer	语句在上一时间周期内的数据库节点上 CPU 使用的倾斜率。
statement_mem	integer	语句执行使用的 statement_mem，预留字段。
active_points	integer	语句占用的资源池并发点数。
dop_value	integer	语句的从资源池中获取的 dop 值。
control_group	text	该字段不支持。
status	text	语句当前的状态，包括： • pending：执行前状态。 • running：执行进行状态。 • finished：执行正常结束。 • aborted：执行异常终止。 • active：非以上四种状态外的正常状态。 • unknown：未知状态。
enqueue	text	该字段不支持。
resource_pool	name	语句当前所在的资源池。
query	text	该后端的最新查询。如果 state 状态是 active（活的），此字段显示当前正在执行的查询。所有其他情况表示上一个查询。
is_plana	boolean	该字段不支持。
node_group	text	该字段不支持。

14.3.36. PG_SESSION_IOSTAT

PG_SESSION_IOSTAT 视图显示当前用户执行作业正在运行时的 IO 负载管理相关信息。
以下涉及到 iops，对于行存，均以万次/s 为单位，对于列存，均以次/s 为单位。

表 14-102. PG_SESSION_IOSTAT 字段

名称	类型	描述
query_id	bigint	作业 id。

名称	类型	描述
mincurriops	integer	该作业当前 io 在数据库节点中的最小值。
maxcurriops	integer	该作业当前 io 在数据库节点中的最大值。
minpeakiops	integer	在作业运行时，作业 io 峰值中，数据库节点最小值。
maxpeakiops	integer	在作业运行时，作业 io 峰值中，数据库节点的最大值。
io_limits	integer	该作业所设 GUC 参数 io_limits。
io_priority	text	该作业所设 GUC 参数 io_priority。
query	text	作业。
node_group	text	该字段不支持。

14.3.37. PG_SETTINGS

PG_SETTINGS 视图显示数据库运行时参数的相关信息。

表 14-103. PG_SETTINGS 字段

名称	类型	描述
name	text	参数名称。
setting	text	参数当前值。
unit	text	参数的隐式结构。
category	text	参数的逻辑组。
short_desc	text	参数的简单描述。
extra_desc	text	参数的详细描述。
context	text	设置参数值的上下文，包括 internal，postmaster，sighup，backend，superuser，user。
vartype	text	参数类型，包括 bool，enum，integer，real，string。
source	text	参数的赋值方式。
min_val	text	参数最小值。如果参数类型不是数值型，那么该字段值为

名称	类型	描述
		null。
max_val	text	参数最大值。如果参数类型不是数值型，那么该字段值为 null。
enumvals	text[]	enum 类型参数合法值。如果参数类型不是 enum 型，那么该字段值为 null。
boot_val	text	数据库启动时参数默认值。
reset_val	text	数据库重置时参数默认值。
sourcefile	text	设置参数值的配置文件。如果参数不是通过配置文件赋值，那么该字段值为 null。
sourceline	integer	设置参数值的配置文件的行号。如果参数不是通过配置文件赋值，那么该字段值为 null。

14.3.38. PG_SHADOW

PG_SHADOW 视图显示了所有在 PG_AUTHID 中标记了 rolcanlogin 的角色的属性。

这个视图的名称来自于该视图是不可读的，因为它包含口令。14.3.71PG_USER 是一个在 PG_SHADOW 上公开可读的视图，只是把口令域填成了空白。

表 14-104. PG_SHADOW 字段

名称	类型	引用	描述
usename	name	14.2.14PG_AUTHID.rolname	用户名。
usesysid	oid	14.2.14PG_AUTHID.oid	用户的 ID。
usecreatedb	Boolean	-	用户可以创建数据库。
usesuper	Boolean	-	用户是系统管理员。
usecatupd	Boolean	-	用户可以更新视图。即使是系统管理员，如果这个字段不是真，也不能更新视图。
userepl	Boolean	-	用户可以初始化流复制和使系统处于或不

名称	类型	引用	描述
			处于备份模式。
passwd	text	-	口令（可能是加密的）；如果没有则为 null。参阅 14.2.14PG_AUTHID 获取加密的口令是如何存储的信息。
valbegin	timestamp with time zone	-	帐户的有效开始时间；如果没有设置有效开始时间，则为 NULL。
valuntil	timestamp with time zone	-	帐户的有效结束时间；如果没有设置有效结束时间，则为 NULL。
respool	name	-	用户使用的资源池。
parent	oid	-	父资源池。
spacelimit	text	-	永久表存储空间限额。
tempspacelimit	text	-	临时表存储空间限额。
spillspacelimit	text	-	算子落盘空间限额。
useconfig	text[ ]	-	运行时配置变量的会话缺省。

14.3.39. PG_STATS

PG_STATS 视图提供对存储在 pg_statistic 表里面的单列统计信息的访问。

表 14-105. PG_STATS 字段

名称	类型	引用	描述
schemaname	name	14.2.41PG_NAMESPACE.nspname	包含表的模式名。
tablename	name	14.2.18PG_CLASS.relname	表名。
attname	name	14.2.13PG_ATTRIBUTE.attname	字段的名称。
inherited	Boolean	-	如果为真，则包含继承的子列，否则只是指定表的字段。

名称	类型	引用	描述
null_frac	real	-	记录中字段为空的百分比。
avg_width	integer	-	字段记录以字节记的平均宽度。
n_distinct	real	-	• 如果大于零，表示字段中独立数值的估计数目。 • 如果小于零，表示独立数值的数目被行数除的负数。 1. 用负数形式是因为 ANALYZE 认为独立数值的数目是随着表增长而增长； 2. 正数的形式用于在字段看上去好像有固定的可能值数目的情况下。比如，-1 表示一个唯一字段，独立数值的个数和行数相同。
n_dndistinct	real	-	标识 dn1 上字段中非 NULL 数据的唯一值的数目。 • 如果大于零，表示独立数值的实际数目。 • 如果小于零，表示独立数值的数目被行数除的负数。（比如，一个字段的数值平均出现概率为两次，则可以表示为 n_dndistinct=-0.5）。 • 如果等于零，表示独立数值的数目未知。
most_common_vals	anyarray	-	一个字段里最常用数值的列表。如果里面的字段数值是最常见的，则为 NULL。
most_common_freqs	real[]	-	一个最常用数值的频率的列表，也就是说，每个出现的次数除以行数。如果 most_common_vals 是 NULL，则为 NULL。
histogram_bounds	anyarray	-	一个数值的列表，它把字段的数值分成几组大致相同热门的组。如果在

名称	类型	引用	描述
			most_common_vals 里有数值，则在这个饼图的计算中省略。如果字段数据类型没有<操作符或者 most_common_vals 列表代表了整个分布性，则这个字段为 NULL。
correlation	real	-	统计与字段值的物理行序和逻辑行序有关。它的范围从-1 到+1。在数值接近-1 或者+1 的时候，在字段上的索引扫描将被认为比它接近零的时候开销更少，因为减少了对磁盘的随机访问。如果字段数据类型没有<操作符，则这个字段为 NULL。
most_common_elems	anyarray	-	一个最常用的非空元素的列表。
most_common_elem_freqs	real[]	-	一个最常用元素的频率的列表。
elem_count_histogram	real[]	-	对于独立的非空元素的统计直方图。

14.3.40. PG_STAT_ACTIVITY

PG_STAT_ACTIVITY 视图显示和当前用户查询相关的信息。

表 14-106. PG_STAT_ACTIVITY 字段

名称	类型	描述
datid	oid	用户会话在后台连接到的数据库 OID。
datname	name	用户会话在后台连接到的数据库名称。
pid	bigint	后台线程 ID。
sessionid	bigint	会话 ID。
usesysid	oid	登录该后台的用户 OID。
usename	name	登录该后台的用户名。
application_name	text	连接到该后台的应用名。
client_addr	inet	连接到该后台的客户端的 IP 地址。 如果此字段是

名称	类型	描述
		null，它表明通过服务器机器上 UNIX 套接字连接客户端或者这是内部进程，如 autovacuum。
client_hostname	text	客户端的主机名，这个字段是通过 client_addr 的反向 DNS 查找得到。这个字段只有在启动 log_hostname 且使用 IP 连接时才非空。
client_port	integer	客户端用于与后台通讯的 TCP 端口号，如果使用 Unix 套接字，则为-1。
backend_start	timestamp with time zone	该过程开始的时间，即当客户端连接服务器时。
xact_start	timestamp with time zone	启动当前事务的时间，如果没有事务是活跃的，则为 null。如果当前查询是首个事务，则这列等同于 query_start 列。
query_start	timestamp with time zone	开始当前活跃查询的时间， 如果 state 的值不是 active，则这个值是上一个查询的开始时间。
state_change	timestamp with time zone	上次状态改变的时间。
waiting	Boolean	如果后台当前正等待锁则为 true。
enqueue	text	该字段不支持。
state	text	该后台当前总体状态。可能值是： • active：后台正在执行一个查询。 • idle：后台正在等待一个新的客户端命令。 • idle in transaction：后台在事务中，但事务中没有语句在执行。 • idle in transaction (aborted)：后台在事务中，但事务中有语句执行失败。 • fastpath function call：后台正在执行一个 fast-path 函数。 • disabled：如果后台禁用 track_activities，则报告这个状态。 说明 普通用户只能查看到自己帐户所对应的会话状态。即其他

名称	类型	描述
		帐户的 state 信息为空。例如以 judy 用户连接数据库后，在 pg_stat_activity 中查看到的普通用户 joe 及初始用户 vastbase 的 state 信息为空： SELECT datname, usename, usesysid, state,pid FROM pg_stat_activity; datname | usename | usesysid | state | pid ----------+---------+----------+--------+----------------- vastbase | vastbase | 10 | | 139968752121616 vastbase | vastbase | 10 | | 139968903116560 db_tpcc | judy | 16398 | active | 139968391403280 vastbase | vastbase | 10 | | 139968643069712 vastbase | vastbase | 10 | | 139968680818448 vastbase | joe | 16390 | | 139968563377936 (6 rows)
resource_pool	name	用户使用的资源池。
query_id	bigint	查询语句的 ID。
query	text	该后台的最新查询。如果 state 状态是 active（活跃的），此字段显示当前正在执行的查询。所有其他情况表示上一个查询。
connection_info	text	json 格式字符串，记录当前连接数据库的驱动类型、驱动版本号、当前驱动的部署路径、进程属主用户等信息（参见 connection_info）。

14.3.41.PG_STAT_ALL_INDEXES

PG_STAT_ALL_INDEXES 视图将包含当前数据库中的每个索引行，显示访问特定索引的统计。

索引可以通过简单的索引扫描或"位图"索引扫描进行使用。位图扫描中几个索引的输出可以通过 AND 或者 OR 规则进行组合， 因此当使用位图扫描的时候，很难将独立堆行抓取与特定索引进行组合， 因此，一个位图扫描增加 pg_stat_all_indexes.idx_tup_read 使用索引计数，并且增加 pg_stat_all_tables.idx_tup_fetch 表计数，但不影响 pg_stat_all_indexes.idx_tup_fetch。

表 14-107. PG_STAT_ALL_INDEXES 字段

名称	类型	描述
relid	oid	这个索引的表的 OID。
indexrelid	oid	索引的 OID。
schemaname	name	索引的模式名。
relname	name	索引的表名。
indexrelname	name	索引名。
idx_scan	bigint	索引上开始的索引扫描数。
idx_tup_read	bigint	通过索引上扫描返回的索引项数。
idx_tup_fetch	bigint	通过使用索引的简单索引扫描抓取的活表行数。

14.3.42. PG_STAT_ALL_TABLES

PG_STAT_ALL_TABLES 视图将包含当前数据库中每个表的一行（包括 TOAST 表），显示访问特定表的统计信息。

表 14-108. PG_STAT_ALL_TABLES 字段

名称	类型	描述
relid	oid	表的 OID。
schemaname	name	该表的模式名。
relname	name	表名。
seq_scan	bigint	该表发起的顺序扫描数。
seq_tup_read	bigint	顺序扫描抓取的活跃行数。
idx_scan	bigint	该表发起的索引扫描数。
idx_tup_fetch	bigint	索引扫描抓取的活跃行数。
n_tup_ins	bigint	插入行数。
n_tup_upd	bigint	更新行数。

名称	类型	描述
n_tup_del	bigint	删除行数。
n_tup_hot_upd	bigint	HOT 更新行数（比如没有更新所需的单独索引）。
n_live_tup	bigint	估计活跃行数。
n_dead_tup	bigint	估计死行数。
last_vacuum	timestamp with time zone	最后一次手动清理该表的时间（不计算 VACUUM FULL）。
last_autovacuum	timestamp with time zone	上次被 autovacuum 守护进程清理该表的时间。
last_analyze	timestamp with time zone	上次手动分析该表的时间。
last_autoanalyze	timestamp with time zone	上次被 autovacuum 守护进程分析该表的时间。
vacuum_count	bigint	这个表被手动清理的次数(不计算 VACUUM FULL）。
autovacuum_count	bigint	这个表被 autovacuum 清理的次数。
analyze_count	bigint	这个表被手动分析的次数。
autoanalyze_count	bigint	这个表被 autovacuum 守护进程分析的次数。
last_data_changed	timestamp with time zone	记录这个表上一次数据发生变化的时间（引起数据变化的操作包括 INSERT/UPDATE/DELETE、EXCHANGE/TRUNCATE/DROP partition)，该列数据仅在本地数据库主节点记录。

14.3.43. PG_STAT_BAD_BLOCK

PG_STAT_BAD_BLOCK 视图显示自节点启动后，读取数据时出现 Page/CU 校验失败的统计信息。

表 14-109. PG_STAT_BAD_BLOCK 字段

名称	类型	描述
nodename	text	节点名。
databaseid	integer	数据库 OID。
tablespaceid	integer	表空间 OID。
relfilenode	integer	文件对象 ID。
bucketid	smallint	一致性 hash bucket ID。
forknum	integer	文件类型。
error_count	integer	出现校验失败的次数。
first_time	timestamp with time zone	第一次出现时间。
last_time	timestamp with time zone	最近一次出现时间。

14.3.44. PG_STAT_BGWRITER

PG_STAT_BGWRITER 视图显示关于后端写进程活动的统计信息。

表 14-110. PG_STAT_BGWRITER 字段

名称	类型	描述
checkpoints_timed	bigint	执行的定期检查点数。
checkpoints_req	bigint	执行的需求检查点数。
checkpoint_write_time	double precision	花费在检查点处理部分的时间总量，其中文件被写入到磁盘，以毫秒为单位。
checkpoint_sync_time	double precision	花费在检查点处理部分的时间总量，其中文件被同步到磁盘，以毫秒为单位。
buffers_checkpoint	bigint	检查点写缓冲区数量。
buffers_clean	bigint	后端写进程写缓冲区数量。
maxwritten_clean	bigint	后端写进程停止清理扫描时间数，因为它写

名称	类型	描述
		了太多缓冲区。
buffers_backend	bigint	通过后端直接写缓冲区数。
buffers_backend_fsync	bigint	后端不得不执行自己的 fsync 调用的时间数（通常后端写进程处理这些即使后端确实自己写）。
buffers_alloc	bigint	分配的缓冲区数量。
stats_reset	timestamp with time zone	这些统计被重置的时间。

14.3.45. PG_STAT_DATABASE

PG_STAT_DATABASE 视图将包含 Vastbase 中每个数据库的数据库统计信息。

表 14-111. PG_STAT_DATABASE 字段

名称	类型	描述
datid	oid	数据库的 OID。
datname	name	这个数据库的名称。
numbackends	integer	当前连接到该数据库的后端数。 这是在返回一个反映目前状态值的视图中唯一的列；自上次重置所有其他列返回累积值。
xact_commit	bigint	此数据库中已经提交的事务数。
xact_rollback	bigint	此数据库中已经回滚的事务数。
blks_read	bigint	在这个数据库中读取的磁盘块的数量。
blks_hit	bigint	高速缓存中已经发现的磁盘块的次数， 这样读取是不必要的（这只包括 PostgreSQL 缓冲区高速缓存，没有操作系统的文件系统缓存）。
tup_returned	bigint	通过数据库查询返回的行数。
tup_fetched	bigint	通过数据库查询抓取的行数。
tup_inserted	bigint	通过数据库查询插入的行数。

名称	类型	描述
tup_updated	bigint	通过数据库查询更新的行数。
tup_deleted	bigint	通过数据库查询删除的行数。
conflicts	bigint	由于数据库恢复冲突取消的查询数量（只在备用服务器发生的冲突）。请参见 14.3.46PG_STAT_DATABASE_CONFLICTS 获取更多信息。
temp_files	bigint	通过数据库查询创建的临时文件数量。计算所有临时文件， 不论为什么创建临时文件（比如排序或者哈希）， 而且不管 log_temp_files 设置。
temp_bytes	bigint	通过数据库查询写入临时文件的数据总量。计算所有临时文件，不论为什么创建临时文件，而且不管 log_temp_files 设置。
deadlocks	bigint	在该数据库中检索的死锁数。
blk_read_time	double precision	通过数据库后端读取数据文件块花费的时间，以毫秒计算。
blk_write_time	double precision	通过数据库后端写入数据文件块花费的时间，以毫秒计算。
stats_reset	timestamp with time zone	重置当前状态统计的时间。

14.3.46. PG_STAT_DATABASE_CONFLICTS

PG_STAT_DATABASE_CONFLICTS 视图显示数据库冲突状态的统计信息。

表 14-112. PG_STAT_DATABASE_CONFLICTS 字段

名称	类型	描述
datid	oid	数据库标识。
datname	name	数据库名称。
confl_tablespace	bigint	冲突的表空间的数目。

名称	类型	描述
confl_lock	bigint	冲突的锁数目。
confl_snapshot	bigint	冲突的快照数目。
confl_bufferpin	bigint	冲突的缓冲区数目。
confl_deadlock	bigint	冲突的死锁数目。

14.3.47. PG_STAT_USER_FUNCTIONS

PG_STAT_USER_FUNCTIONS 视图显示命名空间中用户自定义函数（函数语言为非内部语言）的状态信息。

表 14-113. PG_STAT_USER_FUNCTIONS 字段

名称	类型	描述
funcid	oid	函数标识。
schemaname	name	模式的名称。
funcname	name	函数名称。
calls	bigint	函数被调用的次数。
total_time	double precision	函数的总执行时长。
self_time	double precision	当前线程调用函数的总的时长。

14.3.48. PG_STAT_USER_INDEXES

PG_STAT_USER_INDEXES 视图显示数据库中用户自定义普通表和 toast 表的索引状态信息。

表 14-114. PG_STAT_USER_INDEXES 字段

名称	类型	描述
relid	oid	这个索引的表的 OID。
indexrelid	oid	索引的 OID。

名称	类型	描述
schemaname	name	索引的模式名。
relname	name	索引的表名。
indexrelname	name	索引名。
idx_scan	bigint	索引上开始的索引扫描数。
idx_tup_read	bigint	通过索引上扫描返回的索引项数。
idx_tup_fetch	bigint	通过使用索引的简单索引扫描抓取的活表行数。

14.3.49. PG_STAT_USER_TABLES

PG_STAT_USER_TABLES 视图显示所有命名空间中用户自定义普通表和 toast 表的状态信息。

表 14-115. PG_STAT_USER_TABLES 字段

名称	类型	描述
relid	oid	表的 OID。
schemaname	name	该表的模式名。
relname	name	表名。
seq_scan	bigint	该表发起的顺序扫描数。
seq_tup_read	bigint	顺序扫描抓取的活跃行数。
idx_scan	bigint	该表发起的索引扫描数。
idx_tup_fetch	bigint	索引扫描抓取的活跃行数。
n_tup_ins	bigint	插入行数。
n_tup_upd	bigint	更新行数。
n_tup_del	bigint	删除行数。
n_tup_hot_upd	bigint	HOT 更新行数（即没有更新所需的单独索引）。
n_live_tup	bigint	估计活跃行数。

名称	类型	描述
n_dead_tup	bigint	估计死行数。
last_vacuum	timestamp with time zone	最后一次该表是手动清理的（不计算 VACUUM FULL）。
last_autovacuum	timestamp with time zone	上次被 autovacuum 守护进程清理的表。
last_analyze	timestamp with time zone	上次手动分析这个表。
last_autoanalyze	timestamp with time zone	上次被 autovacuum 守护进程分析的表。
vacuum_count	bigint	这个表被手动清理的次数(不计算 VACUUM FULL）。
autovacuum_count	bigint	这个表被 autovacuum 清理的次数。
analyze_count	bigint	这个表被手动分析的次数。
autoanalyze_count	bigint	这个表被 autovacuum 守护进程分析的次。
last_data_changed	timestamp with time zone	这个表数据最近修改时间。

14.3.50. PG_STAT_REPLICATION

PG_STAT_REPLICATION 视图用于描述日志同步状态信息，如发起端发送日志位置，收端接收日志位置等。

表 14-116. PG_STAT_REPLICATION 字段

名称	类型	描述
pid	bigint	线程的 PID。
usesysid	oid	用户系统 ID。
usename	name	用户名。
application_name	text	程序名称。
client_addr	inet	客户端地址。
client_hostname	text	客户端名。
client_port	integer	客户端端口。

名称	类型	描述
backend_start	timestamp with time zone	程序启动时间。
state	text	日志复制的状态（追赶状态，还是一致的流状态）。
sender_sent_location	text	发送端发送日志位置。
receiver_write_location	text	接收端 write 日志位置。
receiver_flush_location	text	接收端 flush 日志位置。
receiver_replay_location	text	接收端 replay 日志位置。
sync_priority	integer	同步复制的优先级（0 表示异步）。
sync_state	text	同步状态（异步复制，同步复制，还是潜在同步者）。

14.3.51. PG_STAT_SYS_INDEXES

PG_STAT_SYS_INDEXES 视图显示 pg_catalog、information_schema 模式中所有系统表的索引状态信息。

表 14-117. PG_STAT_SYS_INDEXES 字段

名称	类型	描述
relid	oid	这个索引的表的 OID。
indexrelid	oid	索引的 OID。
schemaname	name	索引的模式名。
relname	name	索引的表名。
indexrelname	name	索引名。
idx_scan	bigint	索引上开始的索引扫描数。
idx_tup_read	bigint	通过索引上扫描返回的索引项数。
idx_tup_fetch	bigint	通过使用索引的简单索引扫描抓取的活表行数。

14.3.52. PG_STAT_SYS_TABLES

PG_STAT_SYS_TABLES 视图显示 pg_catalog、information_schema 模式的所有命名空间中系统表的统计信息。

表 14-118. PG_STAT_SYS_TABLES 字段

名称	类型	描述
relid	oid	表的 OID。
schemaname	name	该表的模式名。
relname	name	表名。
seq_scan	bigint	该表发起的顺序扫描数。
seq_tup_read	bigint	顺序扫描抓取的活跃行数。
idx_scan	bigint	该表发起的索引扫描数。
idx_tup_fetch	bigint	索引扫描抓取的活跃行数。
n_tup_ins	bigint	插入行数。
n_tup_upd	bigint	更新行数。
n_tup_del	bigint	删除行数。
n_tup_hot_upd	bigint	HOT 更新行数（比如没有更新所需的单独索引）。
n_live_tup	bigint	估计活跃行数。
n_dead_tup	bigint	估计死行数。
last_vacuum	timestamp with time zone	最后一次该表是手动清理的（不计算 VACUUM FULL）。
last_autovacuum	timestamp with time zone	上次被 autovacuum 守护进程清理的。
last_analyze	timestamp with time zone	上次手动分析这个表。
last_autoanalyze	timestamp with time zone	上次被 autovacuum 守护进程分析的表。
vacuum_count	bigint	这个表被手动清理的次数(不计算 VACUUM FULL）。

名称	类型	描述
autovacuum_count	bigint	这个表被 autovacuum 清理的次数。
analyze_count	bigint	这个表被手动分析的次数。
autoanalyze_count	bigint	这个表被 autovacuum 守护进程分析的次数。
last_data_changed	timestamp with time zone	这个表数据最近修改时间。

14.3.53. PG_STAT_XACT_ALL_TABLES

PG_STAT_XACT_ALL_TABLES 视图显示命名空间中所有普通表和 toast 表的事务状态信息。

表 14-119. PG_STAT_XACT_ALL_TABLES 字段

名称	类型	描述
relid	oid	表的 OID。
schemaname	name	该表的模式名。
relname	name	表名。
seq_scan	bigint	该表发起的顺序扫描数。
seq_tup_read	bigint	顺序扫描抓取的活跃行数。
idx_scan	bigint	该表发起的索引扫描数。
idx_tup_fetch	bigint	索引扫描抓取的活跃行数。
n_tup_ins	bigint	插入行数。
n_tup_upd	bigint	更新行数。
n_tup_del	bigint	删除行数。
n_tup_hot_upd	bigint	HOT 更新行数（比如没有更新所需的单独索引）。

14.3.54. PG_STAT_XACT_SYS_TABLES

PG_STAT_XACT_SYS_TABLES 视图显示命名空间中系统表的事务状态信息。

表 14-120. PG_STAT_XACT_SYS_TABLES 字段

名称	类型	描述
relid	oid	表的 OID。
schemaname	name	该表的模式名。
relname	name	表名。
seq_scan	bigint	该表发起的顺序扫描数。
seq_tup_read	bigint	顺序扫描抓取的活跃行数。
idx_scan	bigint	该表发起的索引扫描数。
idx_tup_fetch	bigint	索引扫描抓取的活跃行数。
n_tup_ins	bigint	插入行数。
n_tup_upd	bigint	更新行数。
n_tup_del	bigint	删除行数。
n_tup_hot_upd	bigint	HOT 更新行数（比如没有更新所需的单独索引）。

14.3.55. PG_STAT_XACT_USER_FUNCTIONS

PG_STAT_XACT_USER_FUNCTIONS 视图包含每个函数的执行的统计信息。

表 14-121. PG_STAT_XACT_USER_FUNCTIONS 字段

名称	类型	描述
funcid	oid	函数标识。
schemaname	name	模式的名称。
funcname	name	函数名称。
calls	bigint	函数被调用的次数。

名称	类型	描述
total_time	double precision	函数的总执行时长。
self_time	double precision	当前线程调用函数的总的时长。

14.3.56. PG_STAT_XACT_USER_TABLES

PG_STAT_XACT_USER_TABLES 视图显示命名空间中用户表的事务状态信息。

表 14-122. PG_STAT_XACT_USER_TABLES 字段

名称	类型	描述
relid	oid	表的 OID。
schemaname	name	该表的模式名。
relname	name	表名。
seq_scan	bigint	该表发起的顺序扫描数。
seq_tup_read	bigint	顺序扫描抓取的活跃行数。
idx_scan	bigint	该表发起的索引扫描数。
idx_tup_fetch	bigint	索引扫描抓取的活跃行数。
n_tup_ins	bigint	插入行数。
n_tup_upd	bigint	更新行数。
n_tup_del	bigint	删除行数。
n_tup_hot_upd	bigint	HOT 更新行数（比如没有更新所需的单独索引）。

14.3.57. PG_STATIO_ALL_INDEXES

PG_STATIO_ALL_INDEXES 视图将包含当前数据库中的每个索引行， 显示特定索引的 I/O 的统计。

表 14-123. PG_STATIO_ALL_INDEXES 字段

名称	类型	描述

名称	类型	描述
relid	oid	索引的表的 OID。
indexrelid	oid	该索引的 OID。
schemaname	name	该索引的模式名。
relname	name	该索引的表名。
indexrelname	name	索引名称。
idx_blks_read	bigint	从索引中读取的磁盘块数。
idx_blks_hit	bigint	索引命中缓存数。

14.3.58. PG_STATIO_ALL_SEQUENCES

PG_STATIO_ALL_SEQUENCES 视图包含当前数据库中每个序列的 I/O 的统计信息。

表 14-124. PG_STATIO_ALL_SEQUENCES 字段

名称	类型	描述
relid	oid	序列 OID。
schemaname	name	序列中模式名。
relname	name	序列名。
blks_read	bigint	从序列中读取的磁盘块数。
blks_hit	bigint	序列中缓存命中数。

14.3.59. PG_STATIO_ALL_TABLES

PG_STATIO_ALL_TABLES 视图将包含当前数据库中每个表（包括 TOAST 表）的 I/O 统计信息。

表 14-125. PG_STATIO_ALL_TABLES 字段

名称	类型	描述
relid	oid	表 OID。

名称	类型	描述
schemaname	name	该表模式名。
relname	name	表名。
heap_blks_read	bigint	从该表中读取的磁盘块数。
heap_blks_hit	bigint	该表缓存命中数。
idx_blks_read	bigint	从表中所有索引读取的磁盘块数。
idx_blks_hit	bigint	表中所有索引命中缓存数。
toast_blks_read	bigint	该表的 TOAST 表读取的磁盘块数（如果存在）。
toast_blks_hit	bigint	该表的 TOAST 表命中缓冲区数（如果存在）。
tidx_blks_read	bigint	该表的 TOAST 表索引读取的磁盘块数（如果存在）。
tidx_blks_hit	bigint	该表的 TOAST 表索引命中缓冲区数（如果存在）。

14.3.60. PG_STATIO_SYS_INDEXES

PG_STATIO_SYS_INDEXES 视图显示命名空间中所有系统表索引的 IO 状态信息。

表 14-126. PG_STATIO_SYS_INDEXES 字段

名称	类型	描述
relid	oid	索引的表的 OID。
indexrelid	oid	该索引的 OID。
schemaname	name	该索引的模式名。
relname	name	该索引的表名。
indexrelname	name	索引名称。
idx_blks_read	bigint	从索引中读取的磁盘块数。
idx_blks_hit	bigint	索引命中缓存数。

14.3.61. PG_STATIO_SYS_SEQUENCES

PG_STATIO_SYS_SEQUENCES 视图显示命名空间中所有序列的 IO 状态信息。

表 14-127. PG_STATIO_SYS_SEQUENCES 字段

名称	类型	描述
relid	oid	序列 OID。
schemaname	name	序列中模式名。
relname	name	序列名。
blks_read	bigint	从序列中读取的磁盘块数。
blks_hit	bigint	序列中缓存命中数。

14.3.62. PG_STATIO_SYS_TABLES

PG_STATIO_SYS_TABLES 视图显示命名空间中所有系统表的 IO 状态信息。

表 14-128. PG_STATIO_SYS_TABLES 字段

名称	类型	描述
relid	oid	表 OID。
schemaname	name	该表模式名。
relname	name	表名。
heap_blks_read	bigint	从该表中读取的磁盘块数。
heap_blks_hit	bigint	该表缓存命中数。
idx_blks_read	bigint	从表中所有索引读取的磁盘块数。
idx_blks_hit	bigint	表中所有索引命中缓存数。
toast_blks_read	bigint	该表的 TOAST 表读取的磁盘块数（如果存在）。
toast_blks_hit	bigint	该表的 TOAST 表命中缓冲区数（如果存在）。
tidx_blks_read	bigint	该表的 TOAST 表索引读取的磁盘块数（如果存在）。

名称	类型	描述
tidx_blks_hit	bigint	该表的 TOAST 表索引命中缓冲区数（如果存在）。

14.3.63. PG_STATIO_USER_INDEXES

PG_STATIO_USER_INDEXES 视图显示命名空间中所有用户关系表索引的 IO 状态信息。

表 14-129. PG_STATIO_USER_INDEXES 字段

名称	类型	描述
relid	oid	索引的表的 OID。
indexrelid	oid	该索引的 OID。
schemaname	name	该索引的模式名。
relname	name	该索引的表名.
indexrelname	name	索引名称。
idx_blks_read	bigint	从索引中读取的磁盘块数。
idx_blks_hit	bigint	索引命中缓存数。

14.3.64. PG_STATIO_USER_SEQUENCES

PG_STATIO_USER_SEQUENCES 视图显示命名空间中所有用户关系表类型为序列的 IO 状态信息。

表 14-130. PG_STATIO_USER_SEQUENCES 字段

名称	类型	描述
relid	oid	序列 OID。
schemaname	name	序列中模式名。
relname	name	序列名。
blks_read	bigint	从序列中读取的磁盘块数。
blks_hit	bigint	序列中缓存命中数。

14.3.65. PG_STATIO_USER_TABLES

PG_STATIO_USER_TABLES 视图显示命名空间中所有用户关系表的 IO 状态信息。

表 14-131. PG_STATIO_USER_TABLES 字段

名称	类型	描述
relid	oid	表 OID。
schemaname	name	该表模式名。
relname	name	表名。
heap_blks_read	bigint	从该表中读取的磁盘块数。
heap_blks_hit	bigint	该表缓存命中数。
idx_blks_read	bigint	从表中所有索引读取的磁盘块数。
idx_blks_hit	bigint	表中所有索引命中缓存数。
toast_blks_read	bigint	该表的 TOAST 表读取的磁盘块数（如果存在）。
toast_blks_hit	bigint	该表的 TOAST 表命中缓冲区数（如果存在）。
tidx_blks_read	bigint	该表的 TOAST 表索引读取的磁盘块数（如果存在）。
tidx_blks_hit	bigint	该表的 TOAST 表索引命中缓冲区数（如果存在）。

14.3.66. PG_THREAD_WAIT_STATUS

通过 PG_THREAD_WAIT_STATUS 视图可以检测当前实例中工作线程（backend thread）以及辅助线程（auxiliary thread）的阻塞等待情况。

表 14-132. PG_THREAD_WAIT_STATUS 字段

名称	类型	描述
node_name	text	当前节点的名称。
db_name	text	数据库名称。
thread_name	text	线程名称。
query_id	bigint	查询 ID，对应 debug_query_id。

名称	类型	描述
tid	bigint	当前线程的线程号。
sessionid	bigint	当前会话 ID。
lwtid	integer	当前线程的轻量级线程号。
psessionid	bigint	父会话 ID。
tlevel	integer	streaming 线程的层级。
smpid	integer	并行线程的 ID。
wait_status	text	当前线程的等待状态。等待状态的详细信息请参见表 14-133。
wait_event	text	如果 wait_status 是 acquire lock、acquire lwlock、wait io 三种类型，此列描述具体的锁、轻量级锁、IO 的信息。否则是空。

wait_status 列的等待状态有以下状态。

表 14-133. 等待状态列表

wait_status 值	含义
none	没在等任意事件。
acquire lock	等待加锁，要么加锁成功，要么加锁等待超时。
acquire lwlock	等待获取轻量级锁。
wait io	等待 IO 完成。
wait cmd	等待完成读取网络通信包。
wait pooler get conn	等待 pooler 完成获取连接。
wait pooler abort conn	等待 pooler 完成终止连接。
wait pooler clean conn	等待 pooler 完成清理连接。
pooler create conn: [nodename], total N	等待 pooler 建立连接，当前正在与 nodename 指定节点建立连接，且仍有 N 个连接等待建立。
get conn	获取到其他节点的连接。
set cmd: [nodename]	在连接上执行 SET/RESET/TRANSACTION BLOCK LEVEL PARA SET/SESSION LEVEL PARA SET，当前正

wait_status 值	含义
	在 nodename 指定节点上执行。
cancel query	取消某连接上正在执行的 SQL 语句。
stop query	停止某连接上正在执行的查询。
wait node: [nodename](plevel), total N, [phase]	等待接收与某节点的连接上的数据，当前正在等待 nodename 节点 plevel 线程的数据，且仍有 N 个连接的数据待返回。如果状态包含 phase 信息，则可能的阶段状态有： • begin：表示处于事务开始阶段。 • commit：表示处于事务提交阶段。 • rollback：表示处于事务回滚阶段。
wait transaction sync: xid	等待 xid 指定事务同步。
wait wal sync	等待特定 LSN 的 wal log 完成到备机的同步。
wait data sync	等待完成数据页到备机的同步。
wait data sync queue	等待把行存的数据页或列存的 CU 放入同步队列。
flush data: [nodename](plevel), [phase]	等待向网络中 nodename 指定节点的 plevel 对应线程发送数据。如果状态包含 phase 信息，则可能的阶段状态为 wait quota，即当前通信流正在等待 quota 值。
stream get conn: [nodename], total N	初始化 stream flow 时，等待与 nodename 节点的 consumer 对象建立连接，且当前有 N 个待建连对象。
wait producer ready: [nodename](plevel), total N	初始化 stream flow 时，等待每个 producer 都准备好，当前正在等待 nodename 节点 plevel 对应线程的 producer 对象准备好，且仍有 N 个 producer 对象处于等待状态。
synchronize quit	stream plan 结束时，等待 stream 线程组内的线程统一退出。
wait stream nodegroup destroy	stream plan 结束时，等待销毁 stream node group。
wait active statement	等待作业执行，正在资源负载管控中。
gtm connect	等待与 GTM 建连。
gtm get gxid	等待从 GTM 获取事务 xid。

wait_status 值	含义
gtm get snapshot	等待从 GTM 获取事务快照 snapshot。
gtm begin trans	等待 GTM 开始事务。
gtm commit trans	等待 GTM 提交事务。
gtm rollback trans	等待 GTM 执行事务回滚。
gtm start prepare trans	等待 GTM 开始两阶段事务的 prepare 阶段。
gtm prepare trans	等待 GTM 完成两阶段事务的 prepare 阶段。
gtm open sequence	等待 GTM 打开 sequence。
gtm close sequence	等待 GTM 关闭 sequence。
gtm create sequence	等待 GTM 创建 sequence。
gtm alter sequence	等待 GTM 修改 sequence。
gtm get sequence val	等待从 GTM 获取 sequence 的下一个值。
gtm set sequence val	等待 GTM 设置 sequence 的值。
gtm drop sequence	等待 GTM 删除 sequence。
gtm rename sequece	等待 GTM 重命名 sequence。
gtm reset xmin	等待 GTM 重新设置 xmin 完成。
gtm get xmin	等待从 GTM 获取 xmin。
gtm get csn	事务提交时等待从 gtm 获取 csn。
analyze: [relname], [phase]	当前正在对表 relname 执行 analyze。如果状态包含 phase 信息，则为 autovacuum，表示是数据库自动开启 AutoVacuum 线程执行的 analyze 分析操作。
vacuum: [relname], [phase]	当前正在对表 relname 执行 vacuum。如果状态包含 phase 信息，则为 autovacuum，表示是数据库自动开启 AutoVacuum 线程执行的 vacuum 清理操作。
vacuum full: [relname]	当前正在对表 relname 执行 vacuum full 清理。
create index	当前正在创建索引。

wait_status 值	含义
HashJoin - [build hash | write file]	当前是 HashJoin 算子，主要关注耗时的执行阶段。 • build hash：表示当前 HashJoin 算子正在建立哈希表。 • write file：表示当前 HashJoin 算子正在将数据写入磁盘。
HashAgg - [build hash | write file]	当前是 HashAgg 算子，主要关注耗时的执行阶段。 • build hash：表示当前 HashAgg 算子正在建立哈希表。 • write file：表示当前 HashAgg 算子正在将数据写入磁盘。
HashSetop - [build hash | write file]	当前是 HashSetop 算子，主要关注耗时的执行阶段。 • build hash：表示当前 HashSetop 算子正在建立哈希表。 • write file：表示当前 HashSetop 算子正在将数据写入磁盘。
Sort | Sort - write file	当前是 Sort 算子做排序，write file 表示 Sort 算子正在将数据写入磁盘。
Material | Material - write file	当前是 Material 算子，write file 表示 Material 算子正在将数据写入磁盘。
NestLoop	当前是 NestLoop 算子。
wait memory	等待内存获取。
wait sync consumer next step	Stream 算子等待消费者执行。
wait sync producer next step	Stream 算子等待生产者执行。

当 wait_status 为 acquire lwlock、acquire lock 或者 wait io 时，表示有等待事件。正在等待获取 wait_event 列对应类型的轻量级锁、事务锁，或者正在进行 IO。

其中，wait_status 值为 acquire lwlock（轻量级锁）时对应的 wait_event 等待事件类型与描述信息如下。（wait_event 为 extension 时，表示此时的轻量级锁是动态分配的锁，未被监控。）

表 14-134. 轻量级锁等待事件列表

wait_event 类型	类型描述

wait_event 类型	类型描述
ShmemIndexLock	用于保护共享内存中的主索引哈希表。
OidGenLock	用于避免不同线程产生相同的 OID。
XidGenLock	用于避免两个事务获得相同的 xid。
ProcArrayLock	用于避免并发访问或修改 ProcArray 共享数组。
SInvalReadLock	用于避免与清理失效消息并发执行。
SInvalWriteLock	用于避免与其它写失效消息、清理失效消息并发执行。
WALInsertLock	用于避免与其它 WAL 插入操作并发执行。
WALWriteLock	用于避免并发 WAL 写盘。
ControlFileLock	用于避免 pg_control 文件的读写并发、写写并发。
CheckpointLock	用于避免多个 checkpoint 并发执行。
CLogControlLock	用于避免并发访问或者修改 Clog 控制数据结构。
SubtransControlLock	用于避免并发访问或者修改子事务控制数据结构。
MultiXactGenLock	用于串行分配唯一 MultiXact id。
MultiXactOffsetControlLock	用于避免对 pg_multixact/offset 的写写并发和读写并发。
MultiXactMemberControlLock	用于避免对 pg_multixact/members 的写写并发和读写并发。
RelCacheInitLock	用于失效消息场景对 init 文件进行操作时加锁。
CheckpointerCommLock	用于向 checkpointer 发起文件刷盘请求场景，需要串行的向请求队列插入请求结构。
TwoPhaseStateLock	用于避免并发访问或者修改两阶段信息共享数组。
TablespaceCreateLock	用于确定 tablespace 是否已经存在。
BtreeVacuumLock	用于防止 vacuum 清理 B-tree 中还在使用的页面。
AutovacuumLock	用于串行化访问 autovacuum worker 数组。
AutovacuumScheduleLock	用于串行化分配需要 vacuum 的 table。
AutoanalyzeLock	用于获取和释放允许执行 Autoanalyze 的任务资源。

wait_event 类型	类型描述
SyncScanLock	用于确定 heap 扫描时某个 relfilenode 的起始位置。
NodeTableLock	用于保护存放数据库节点信息的共享结构。
PoolerLock	用于保证两个线程不会同时从连接池里取到相同的连接。
RelationMappingLock	用于等待更新系统表到存储位置之间映射的文件。
AsyncCtlLock	用于避免并发访问或者修改共享通知状态。
AsyncQueueLock	用于避免并发访问或者修改共享通知信息队列。
SerializableXactHashLock	用于避免对于可串行事务共享结构的写写并发和读写并发。
SerializableFinishedListLock	用于避免对于已完成可串行事务共享链表的写写并发和读写并发。
SerializablePredicateLockListLock	用于保护对于可串行事务持有的锁链表。
OldSerXidLock	用于保护记录冲突可串行事务的结构。
FileStatLock	用于保护存储统计文件信息的数据结构。
SyncRepLock	用于在主备复制时保护 xlog 同步信息。
DataSyncRepLock	用于在主备复制时保护数据页同步信息。
CStoreColspaceCacheLock	用于保护列存表的 CU 空间分配。
CStoreCUCacheSweepLock	用于列存 CU Cache 循环淘汰。
MetaCacheSweepLock	用于元数据循环淘汰。
ExtensionConnectorLibLock	用于初始化 ODBC 连接场景，在加载与卸载特定动态库时进行加锁。
SearchServerLibLock	用于 GPU 加速场景初始化加载特定动态库时，对读文件操作进行加锁。
LsnXlogChkFileLock	用于串行更新特定结构中记录的主备机的 xlog flush 位置点。
GTMHostInfoLock	用于避免并发访问或者修改 GTM 主机信息。
ReplicationSlotAllocationLock	用于主备复制时保护主机端的流复制槽的分配。
ReplicationSlotControlLock	用于主备复制时避免并发更新流复制槽状态。

wait_event 类型	类型描述
ResourcePoolHashLock	用于避免并发访问或者修改资源池哈希表。
WorkloadStatHashLock	用于避免并发访问或者修改包含数据库主节点的 SQL 请求构成的哈希表。
WorkloadIoStatHashLock	用于避免并发访问或者修改用于统计当前数据库节点的IO信息的哈希表。
WorkloadCGroupHashLock	用于避免并发访问或者修改 Cgroup 信息构成的哈希表。
OBSGetPathLock	用于避免对 obs 路径的写写并发和读写并发。
WorkloadUserInfoLock	用于避免并发访问或修改负载管理的用户信息哈希表。
WorkloadRecordLock	用于避免并发访问或修改在内存自适应管理时对数据库主节点收到请求构成的哈希表。
WorkloadIOUtilLock	用于保护记录 iostat，CPU 等负载信息的结构。
WorkloadNodeGroupLock	用于避免并发访问或者修改内存中的 nodegroup 信息构成的哈希表。
JobShmemLock	用于定时任务功能中保护定时读取的全局变量。
OBSRuntimeLock	用于获取环境变量，如 GASSHOME。
LLVMDumpIRLock	用于导出动态生成函数所对应的汇编语言。
LLVMParseIRLock	用于在查询开始处从 IR 文件中编译并解析已写好的 IR 函数。
CriticalCacheBuildLock	用于从共享或者本地缓存初始化文件中加载 cache 的场景。
WaitCountHashLock	用于保护用户语句计数功能场景中的共享结构。
BufMappingLock	用于保护对共享缓冲映射表的操作。
LockMgrLock	用于保护常规锁结构信息。
PredicateLockMgrLock	用于保护可串行事务锁结构信息。
OperatorRealTLock	用于避免并发访问或者修改记录算子级实时数据的全局结构。
OperatorHistLock	用于避免并发访问或者修改记录算子级历史数据的全局结构。

wait_event 类型	类型描述
SessionRealTLock	用于避免并发访问或者修改记录 query 级实时数据的全局结构。
SessionHistLock	用于避免并发访问或者修改记录 query 级历史数据的全局结构。
CacheSlotMappingLock	用于保护 CU Cache 全局信息。
BarrierLock	用于保证当前只有一个线程在创建 Barrier。
dummyServerInfoCacheLock	用于保护缓存加速 Vastbase 连接信息的全局哈希表。
RPNumberLock	用于加速 Vastbase 的数据库节点对正在执行计划的任务线程的计数。
ClusterRPLock	用于加速 Vastbase 的 CCN 中维护的 Vastbase 负载数据的并发存取控制。
CBMParseXlogLock	Cbm 解析 xlog 时的保护锁
RelfilenodeReuseLock	避免错误地取消已重用的列属性文件的链接。
RcvWriteLock	防止并发调用 WalDataRcvWrite。
PercentileLock	用于保护全局 PercentileBuffer
CSNBufMappingLock	保护 csn 页面
UniqueSQLMappingLock	用于保护 uniquesql hash table
DelayDDLLock	防止并发 ddl。
CLOG Ctl	用于避免并发访问或者修改 Clog 控制数据结构
Async Ctl	保护 Async buffer
MultiXactOffset Ctl	保护 MultiXact offet 的 slru buffer
MultiXactMember Ctl	保护 MultiXact member 的 slrubuffer
OldSerXid SLRU Ctl	保护 old xids 的 slru buffer
ReplicationSlotLock	用于保护 ReplicationSlot
PGPROCLock	用于保护 pgproc

wait_event 类型	类型描述
MetaCacheLock	用于保护 MetaCache
DataCacheLock	用于保护 datacache
InstrUserLock	用于保护 InstrUserHTAB。
BadBlockStatHashLock	用于保护 global_bad_block_stat hash 表。
BufFreelistLock	用于保证共享缓冲区空闲列表操作的原子性。
CUSlotListLock	用于控制列存缓冲区槽位的并发操作。
AddinShmemInitLock	保护共享内存对象的初始化。
AlterPortLock	保护协调节点更改注册端口号的操作。
FdwPartitionCaheLock	HDFS 分区表缓冲区的管理锁。
DfsConnectorCacheLock	DFSConnector 缓冲区的管理锁。
DfsSpaceCacheLock	HDFS 表空间管理缓冲区的管理锁。
FullBuildXlogCopyStartPtrLock	用于保护全量 Build 中 Xlog 拷贝的操作。
DfsUserLoginLock	用于 HDFS 用户登录以及认证。
LogicalReplicationSlotPersistentDataLock	用于保护逻辑复制过程中复制槽位的数据。
WorkloadSessionInfoLock	保护负载管理 session info 内存 hash 表访问。
InstrWorkloadLock	保护负载管理统计信息的内存 hash 表访问。
PgfdwLock	用于管理实例向 Foreign server 建立连接。
InstanceTimeLock	用于获取实例中会话的时间信息。
XlogRemoveSegLock	保护 Xlog 段文件的回收操作。
DnUsedSpaceHashLock	用于更新会话对应的空间使用信息。
CsnMinLock	用于计算 CSNmin。
GPCCommitLock	用于保护全局 Plan Cache hash 表的添加操作。
GPCClearLock	用于保护全局 Plan Cache hash 表的清除操作。
GPCTimelineLock	用于保护全局 Plan Cache hash 表检查 Timeline 的操作。

wait_event 类型	类型描述
TsTagsCacheLock	用于时序 tag 缓存管理。
InstanceRealTLock	用于保护共享实例统计信息 hash 表的更新操作。
CLogBufMappingLock	用于提交日志缓存管理。
GPCMappingLock	用于全局 Plan Cache 缓存管理。
GPCPrepareMappingLock	用于全局 Plan Cache 缓存管理。
BufferIOLock	保护共享缓冲区页面的 IO 操作。
BufferContentLock	保护共享缓冲区页面内容的读取、修改。
CSNLOG Ctl	用于 CSN 日志管理。
DoubleWriteLock	用于双写的管理操作。
RowPageReplicationLock	用于管理行存储的数据页复制。
extension	其他轻量锁。

当 wait_status 值为 wait io 时对应的 wait_event 等待事件类型与描述信息如下。

表 14-135. IO 等待事件列表

wait_event 类型	类型描述
BufFileRead	从临时文件中读取数据到指定 buffer。
BufFileWrite	向临时文件中写入指定 buffer 中的内容。
ControlFileRead	读取 pg_control 文件。主要在数据库启动、执行 checkpoint 和主备校验过程中发生。
ControlFileSync	将 pg_control 文件持久化到磁盘。数据库初始化时发生。
ControlFileSyncUpdate	将 pg_control 文件持久化到磁盘。主要在数据库启动、执行 checkpoint 和主备校验过程中发生。
ControlFileWrite	写入 pg_control 文件。数据库初始化时发生。
ControlFileWriteUpdate	更新 pg_control 文件。主要在数据库启动、执行 checkpoint 和主备校验过程中发生。

wait_event 类型	类型描述
CopyFileRead	copy 文件时读取文件内容。
CopyFileWrite	copy 文件时写入文件内容。
DataFileExtend	扩展文件时向文件写入内容。
DataFileFlush	将表数据文件持久化到磁盘
DataFileImmediateSync	将表数据文件立即持久化到磁盘。
DataFilePrefetch	异步读取表数据文件。
DataFileRead	同步读取表数据文件。
DataFileSync	将表数据文件的修改持久化到磁盘。
DataFileTruncate	表数据文件 truncate。
DataFileWrite	向表数据文件写入内容。
LockFileAddToDataDirRead	读取"postmaster.pid"文件。
LockFileAddToDataDirSync	将"postmaster.pid"内容持久化到磁盘。
LockFileAddToDataDirWrite	将 pid 信息写到"postmaster.pid"文件。
LockFileCreateRead	读取 LockFile 文件"%s.lock"。
LockFileCreateSync	将 LockFile 文件"%s.lock"内容持久化到磁盘。
LockFileCreateWRITE	将 pid 信息写到 LockFile 文件"%s.lock"。
RelationMapRead	读取系统表到存储位置之间的映射文件
RelationMapSync	将系统表到存储位置之间的映射文件持久化到磁盘。
RelationMapWrite	写入系统表到存储位置之间的映射文件。
ReplicationSlotRead	读取流复制槽文件。重新启动时发生。
ReplicationSlotRestoreSync	将流复制槽文件持久化到磁盘。重新启动时发生。
ReplicationSlotSync	checkpoint 时将流复制槽临时文件持久化到磁盘。
ReplicationSlotWrite	checkpoint 时写流复制槽临时文件。
SLRUFlushSync	将 pg_clog、pg_subtrans 和 pg_multixact 文件持久化到磁盘。

wait_event 类型	类型描述
	主要在执行 checkpoint 和数据库停机时发生。
SLRURead	读取 pg_clog、pg_subtrans 和 pg_multixact 文件。
SLRUSync	将脏页写入文件 pg_clog、pg_subtrans 和 pg_multixact 并持久化到磁盘。主要在执行 checkpoint 和数据库停机时发生。
SLRUWrite	写入 pg_clog、pg_subtrans 和 pg_multixact 文件。
TimelineHistoryRead	读取 timeline history 文件。在数据库启动时发生。
TimelineHistorySync	将 timeline history 文件持久化到磁盘。在数据库启动时发生。
TimelineHistoryWrite	写入 timeline history 文件。在数据库启动时发生。
TwophaseFileRead	读取 pg_twophase 文件。在两阶段事务提交、两阶段事务恢复时发生。
TwophaseFileSync	将 pg_twophase 文件持久化到磁盘。在两阶段事务提交、两阶段事务恢复时发生。
TwophaseFileWrite	写入 pg_twophase 文件。在两阶段事务提交、两阶段事务恢复时发生。
WALBootstrapSync	将初始化的 WAL 文件持久化到磁盘。在数据库初始化发生。
WALBootstrapWrite	写入初始化的 WAL 文件。在数据库初始化发生。
WALCopyRead	读取已存在的 WAL 文件并进行复制时产生的读操作。在执行归档恢复完后发生。
WALCopySync	将复制的 WAL 文件持久化到磁盘。在执行归档恢复完后发生。
WALCopyWrite	读取已存在 WAL 文件并进行复制时产生的写操作。在执行归档恢复完后发生。
WALInitSync	将新初始化的 WAL 文件持久化磁盘。在日志回收或写日志时发生。
WALInitWrite	将新创建的 WAL 文件初始化为 0。在日志回收或写日志时发生。
WALRead	从 xlog 日志读取数据。两阶段文件 redo 相关的操作产生。
WALSyncMethodAssign	将当前打开的所有 WAL 文件持久化到磁盘。

wait_event 类型	类型描述
WALWrite	写入 WAL 文件。
DoubleWriteFileRead	双写 文件读取。
DoubleWriteFileSync	双写 文件强制刷盘。
DoubleWriteFileWrite	双写 文件写入。
PredoProcessPending	并行日志回放中当前记录回放等待其它记录回放完成。
PredoApply	并行日志回放中等待当前工作线程等待其他线程回放至本线程 LSN。
DisableConnectFileRead	HA 锁分片逻辑文件读取。
DisableConnectFileSync	HA 锁分片逻辑文件强制刷盘。
DisableConnectFileWrite	HA 锁分片逻辑文件写入。

当 wait_status 值为 acquire lock（事务锁）时对应的 wait_event 等待事件类型与描述信息如下。

表 14-136. 事务锁等待事件列表

wait_event 类型	类型描述
relation	对表加锁。
extend	对表扩展空间时加锁。
partition	对分区表加锁。
partition_seq	对分区表的分区加锁。
page	对表页面加锁。
tuple	对页面上的 tuple 加锁。
transactionid	对事务 ID 加锁。
virtualxid	对虚拟事务 ID 加锁。
object	加对象锁。
cstore_freespace	对列存空闲空间加锁。
userlock	加用户锁。

wait_event 类型	类型描述
advisory	加 advisory 锁。

14.3.67. PG_TABLES

PG_TABLES 视图提供了对数据库中每个表访问的有用信息。

表 14-137. PG_TABLES 字段

名称	类型	引用	描述
schemaname	name	14.2.41PG_NAMESPACE.nspname	包含表的模式名。
tablename	name	14.2.18PG_CLASS.relname	表名。
tableowner	name	pg_get_userbyid(14.2.18PG_CLASS.relowner)	表的所有者。
tablespace	name	14.2.59PG_TABLESPACE.spcname	包含表的表空间，默认为 NULL。
hasindexes	Boolean	14.2.18PG_CLASS.relhasindex	如果表上有索引（或者最近拥有）则为 TRUE，否则为 FALSE。
hasrules	Boolean	14.2.18PG_CLASS.relhasruls	如果表上有规则，则为 TRUE，否则为 FALSE。
hastriggers	Boolean	14.2.18PG_CLASS.RELHASTRIGGERS	如果表上有触发器，则为 TRUE，否则为 FALSE。
tablecreator	name		
created	Timestamp with time zone		
last_ddl_time	Timestamp with time zone		

14.3.68. PG_TDE_INFO

PG_TDE_INFO 视图提供了 Vastbase 加密信息。

表 14-138. PG_TDE_INFO 字段

名称	类型	描述
is_encrypt	Boolean	是否加密 Vastbase。 • f：非加密 Vastbase。 • t：加密 Vastbase。
g_tde_algo	text	加密算法。 • SM4-CTR-128 • AES-CTR-128
remain	text	保留字段。

14.3.69. PG_TIMEZONE_NAMES

PG_TIMEZONE_NAMES 视图提供了显示了所有能够被 SET TIMEZONE 识别的时区名及其缩写、UTC 偏移量、是否夏时制。

表 14-139. PG_TIMEZONE_NAMES 字段

名称	类型	描述
name	text	时区名。
abbrev	text	时区名缩写。
utc_offset	interval	相对于 UTC 的偏移量。
is_dst	Boolean	如果当前正处于夏令时范围则为 TRUE，否则为 FALSE。

14.3.70. PG_TOTAL_USER_RESOURCE_INFO

PG_TOTAL_USER_RESOURCE_INFO 视图显示所有用户资源使用情况，需要使用管理员用户进行查询。此视图在参数 use_workload_manager 为 on 时才有效。其中，IO 相关监控项在参数 enable_logical_io_statistics 为 on 时才有效。

表 14-140. PG_TOTAL_USER_RESOURCE_INFO 字段

名称	类型	描述

名称	类型	描述
username	name	用户名。
used_memory	integer	正在使用的内存大小，单位 MB。
total_memory	integer	可以使用的内存大小，单位 MB。值为 0 表示未限制最大可用内存，其限制取决于数据库最大可用内存。
used_cpu	double precision	正在使用的 CPU 核数（仅统计复杂作业 CPU 使用 情况，且该值为相关控制组的 CPU 使用统计值）。
total_cpu	integer	在该机器节点上，用户关联控制组的 CPU 核数总和。
used_space	bigint	已使用的永久表存储空间大小，单位 KB。
total_space	bigint	可使用的永久表存储空间大小，单位 KB，值为-1 表示未限制最大存储空间。
used_temp_space	bigint	已使用的临时空间大小，单位 KB。
total_temp_space	bigint	可使用的临时空间总大小，单位 KB，值为-1 表示未限制。
used_spill_space	bigint	已使用的算子落盘空间大小，单位 KB。
total_spill_space	bigint	可使用的算子落盘空间总大小，单位 KB，值为-1 表示未限制。
read_kbytes	bigint	数据库主节点：过去 5 秒内，该用户在数据库节点上复杂作业 read 的字节总数（单位 KB）。 数据库节点：实例启动至当前时间为止，该用户复杂作业 read 的字节总数（单位 KB）。
write_kbytes	bigint	数据库主节点：过去 5 秒内，该用户在数据库节点上复杂作业 write 的字节总数（单位 KB）。 数据库节点：实例启动至当前时间为止，该用户复杂作业 write 的字节总数（单位 KB）。
read_counts	bigint	数据库主节点：过去 5 秒内，该用户在数据库节点上复杂作业 read 的次数之和（单位次）。 数据库节点：实例启动至当前时间为止，该用户复杂作业 read 的次数之和（单位次）。
write_counts	bigint	数据库主节点：过去 5 秒内，该用户在数据库节点上复杂作业 write 的次数之和（单位次）。

名称	类型	描述
		数据库节点:实例启动至当前时间为止,该用户复杂作业 write 的次数之和（单位次）。
read_speed	double precision	数据库主节点：过去 5 秒内，该用户在单个数据库节点上复杂作业 read 平均速率（单位 KB/s）。 数据库节点：过去 5 秒内，该用户在该数据库节点上复杂作业 read 平均速率（单位 KB/s）。
write_speed	double precision	数据库主节点：过去 5 秒内，该用户在单个数据库节点上复杂作业 write 平均速率（单位 KB/s）。 数据库节点：过去 5 秒内，该用户在该数据库节点上复杂作业 write 平均速率（单位 KB/s）。

14.3.71.PG_USER

PG_USER 视图提供了访问数据库用户的信息，默认只有初始化用户和具有 sysadmin 属性的用户可以查看，其余用户需要赋权后才可以查看。

表 14-141. PG_USER 字段

名称	类型	描述
usename	name	用户名。
usesysid	oid	此用户的 ID。
usecreatedb	Boolean	用户是否可以创建数据库。
usesuper	Boolean	用户是否是拥有最高权限的初始系统管理员。
usecatupd	Boolean	用户是否可以直接更新系统表。只有 usesysid=10 的初始系统管理员拥有此权限。其他用户无法获得此权限。
userepl	Boolean	用户是否可以复制数据流。
passwd	text	密文存储后的用户口令，始终为********。
valbegin	timestamp with time zone	帐户的有效开始时间；如果没有设置有效开始时间，则为 NULL。

名称	类型	描述
valuntil	timestamp with time zone	帐户的有效结束时间；如果没有设置有效结束时间，则为 NULL。
respool	name	用户所在的资源池。
parent	oid	父用户 OID。
spacelimit	text	永久表存储空间限额。
tempspacelimit	text	临时表存储空间限额。
spillspacelimit	text	算子落盘空间限额。
useconfig	text[]	运行时配置参数的会话缺省。
nodegroup	name	用户关联的逻辑 Vastbase 名称,如果该用户没有管理 Vastbasess，则该字段为空。

14.3.72.PG_USER_MAPPINGS

PG_USER_MAPPINGS 视图提供访问关于用户映射的信息的接口。

这个视图只是一个 14.2.67PG_USER_MAPPING 的可读部分的视图化表现，如果用户无权使用它则查询该表时，有些选项字段会显示为空。普通用户需要授权才可以访问。

表 14-142. PG_USER_MAPPINGS 字段

名称	类型	引用	描述
umid	oid	14.2.67PG_USER_MAPPING.oid	用户映射的 OID。
srvid	oid	14.2.32PG_FOREIGN_SERVER.oid	包含这个映射的外部服务器的 OID。
srvname	name	14.2.32PG_FOREIGN_SERVER.srvname	外部服务器的名称。
umuser	oid	14.2.14PG_AUTHID.oid	被映射的本地角色的 OID，如果用户映射是公共的则为 0。
usename	name	-	被映射的本地用户的名称。
umoptions	text[ ]	-	如果当前用户是外部服务器的所有者，则为

名称	类型	引用	描述
			用户映射指定选项，使用 “keyword=value” 字符串，否则为 null。

14.3.73. PG_VIEWS

PG_VIEWS 视图提供访问数据库中每个视图的有用信息。

表 14-143. PG_VIEWS 字段

名称	类型	引用	描述
schemaname	name	14.2.41PG_NAMESPACE.nspname	包含视图的模式名。
viewname	name	14.2.18PG_CLASS.relname	视图名。
viewowner	name	14.2.14PG_AUTHID.Erolname	视图的所有者。
definition	text	-	视图的定义。

14.3.74. PG_WLM_STATISTICS

PG_WLM_STATISTICS 视图显示作业结束后或已被处理异常后的负载管理相关信息。

表 14-144. PG_WLM_STATISTICS 字段

名称	类型	描述
statement	text	执行了异常处理的语句。
block_time	bigint	语句执行前的阻塞时间。
elapsed_time	bigint	语句的实际执行时间。
total_cpu_time	bigint	语句执行异常处理时数据库节点上 CPU 使用的总时间。
qualification_time	bigint	语句检查倾斜率的时间周期。
cpu_skew_percent	integer	语句在执行异常处理时数据库节点上 CPU 使用的倾斜率。
control_group	text	该字段不支持。
status	text	语句执行异常处理后的状态，包括：

名称	类型	描述
		• pending：执行前预备状态。 • running：执行进行状态。 • finished：执行正常结束。 • abort：执行异常终止。
action	text	语句执行的异常处理动作，包括： • abort：执行终止操作。 • adjust：执行 Cgroups 调整操作，目前只有降级操作。 • finish：正常结束。

14.3.75. PLAN_TABLE

PLAN_TABLE 显示用户通过执行 EXPLAIN PLAN 收集到的计划信息。计划信息的生命周期是 session 级别，session 退出后相应的数据将被清除。同时不同 session 和不同 user 间的数据是相互隔离的。

表 14-145. PLAN_TABLE 字段

名称	类型	描述
statement_id	varchar(30)	用户输入的查询标签。
plan_id	bigint	查询标识。
id	int	查询生成的计划中的每一个执行算子的编号。
operation	varchar(30)	计划中算子的操作描述。
options	varchar(255)	操作选项。
object_name	name	操作对应的对象名，非查询中使用到的对象别名。来自于用户定义。
object_type	varchar(30)	对象类型。
object_owner	name	对象所属 schema，来自于用户定义。
projection	varchar(4000)	操作输出的列信息。

说明

- object_type 取值范围为 14.2.18PG_CLASS 中定义的 relkind 类型(TABLE 普通表，INDEX 索引，SEQUENCE 序列，VIEW 视图，COMPOSITE TYPE 复合类型，TOASTVALUE TOAST 表)和计划使用到的 rtekind(SUBQUERY, JOIN, FUNCTION, VALUES, CTE, REMOTE_QUERY)。
- object_owner 对于 RTE 来说是计划中使用的对象描述，非用户定义的类型不存在 object_owner。
- statement_id、object_name、object_owner、projection 字段内容遵循用户定义的大小写存储，其它字段内容采用大写存储。
- 支持用户对 PLAN_TABLE 进行 SELECT 和 DELETE 操作，不支持其它 DML 操作。

14.3.76.GS_FILE_STAT

GS_FILE_STAT 视图通过对数据文件 IO 的统计，反映数据的 IO 性能，用以发现 IO 操作异常等性能问题。

表 14-146. GS_FILE_STAT 字段

名称	类型	描述
filenum	oid	文件标识。
dbid	oid	数据库标识。
spcid	oid	表空间标识。
phyrds	bigint	读物理文件的数目。
phywrts	bigint	写物理文件的数目。
phyblkrd	bigint	读物理文件块的数目。
phyblkwrt	bigint	写物理文件块的数目。
readtim	bigint	读文件的总时长，单位微秒。
writetim	bigint	写文件的总时长，单位微秒。
avgiotim	bigint	读写文件的平均时长，单位微秒。
lstiotim	bigint	最后一次读文件时长，单位微秒。
miniotim	bigint	读写文件的最小时长，单位微秒。
maxiowtm	bigint	读写文件的最大时长，单位微秒。

14.3.77. GS_OS_RUN_INFO

GS_OS_RUN_INFO 视图显示当前操作系统运行的状态信息。

表 14-147. GS_OS_RUN_INFO 字段

名称	类型	描述
id	integer	编号。
name	text	操作系统运行状态名称。
value	numeric	操作系统运行状态值。
comments	text	操作系统运行状态注释。
cumulative	Boolean	操作系统运行状态的值是否为累加值。

14.3.78. GS_REDO_STAT

GS_REDO_STAT 视图用于统计回话线程日志回放情况。

表 14-148. GS_REDO_STAT 字段

名称	类型	描述
phywrts	bigint	日志回放过程中写数据的次数。
phyblkwrt	bigint	日志回放过程中写数据的块数。
writetim	bigint	日志回放过程中写数据所耗的总时间。
avgiotim	bigint	日志回放过程中写一次数据的平均消耗时间。
lstiotim	bigint	日志回放过程中最后一次写数据消耗的时间。
miniotim	bigint	日志回放过程中单次写数据消耗的最短时间。
maxiowtm	bigint	日志回放过程中单次写数据消耗的最长时间。

14.3.79.GS_SESSION_MEMORY

GS_SESSION_MEMORY 视图统计 Session 级别的内存使用情况，包含执行作业在数据节点上 Postgres 线程分配的所有内存。

表 14-149. GS_SESSION_MEMORY 字段

名称	类型	描述
sessid	text	线程启动时间+线程标识。
init_mem	integer	当前正在执行作业进入执行器前已分配的内存，单位 MB。
used_mem	integer	当前正在执行作业已分配的内存，单位 MB。
peak_mem	integer	当前正在执行作业已分配的内存峰值，单位 MB。

14.3.80.GS_SESSION_MEMORY_DETAIL

GS_SESSION_MEMORY_DETAIL 统计线程的内存使用情况，以 MemoryContext 节点来统计。当开启线程池（enable_thread_pool = on）时，该视图包含所有的线程和会话的内存使用情况。

其中内存上下文“TempSmallContextGroup”，记录当前线程中所有内存上下文字段“totalsize”小于 8192 字节的信息汇总，并且内存上下文统计计数记录到“usedsize”字段中。所以在视图中，“TempSmallContextGroup”内存上下文中的“totalsize”和“freesize”是该线程中所有内存上下文“totalsize”小于 8192 字节的汇总总和，usedsize 字段表示统计的内存上下文个数。

可通过"select * from gs_session_memctx_detail(threadid, '');"将某个线程所有内存上下文信息记录到“/tmp/dumpmem”目录下的“threadid_timestamp.log”文件中。其中 threadid 可通过下表 sessid 中获得。

表 14-150. GS_SESSION_MEMORY_DETAIL 字段

名称	类型	描述
sessid	text	线程启动时间+线程标识（字符串信息为 timestamp.threadid）。
sesstype	text	线程名称。
contextname	text	内存上下文名称。
level	smallint	当前上下文在整体内存上下文中的层级。

名称	类型	描述
parent	text	父内存上下文名称。
totalsize	bigint	当前内存上下文的内存总数，单位 Byte。
freesize	bigint	当前内存上下文中已释放的内存总数，单位 Byte。
usedsize	bigint	当前内存上下文中已使用的内存总数，单位 Byte；“TempSmallContextGroup”内存上下文中该字段含义为统计计数。

14.3.81.GS_SESSION_STAT

GS_SESSION_STAT 视图以会话线程或 AutoVacuum 线程为单位，统计会话状态信息。

表 14-151. GS_SESSION_STAT 字段

名称	类型	描述
sessid	text	线程标识+线程启动时间。
statid	integer	统计编号。
statname	text	统计会话名称。
statunit	text	统计会话单位。
value	bigint	统计会话值。

14.3.82.GS_SESSION_TIME

GS_SESSION_TIME 视图用于统计会话线程的运行时间信息，及各执行阶段所消耗时间。

表 14-152. GS_SESSION_TIME 字段

名称	类型	描述
sessid	text	线程标识+线程启动时间。
stat_id	integer	统计编号。
stat_name	text	会话类型名称。

名称	类型	描述
value	bigint	会话值。

14.3.83. GS_TOTAL_MEMORY_DETAIL

GS_TOTAL_MEMORY_DETAIL 视图统计当前数据库节点使用内存的信息，单位为 MB。

表 14-153. GS_TOTAL_MEMORY_DETAIL 字段

名称	类型	描述
nodename	text	节点名称。
memorytype	text	内存类型，包括以下几种： • max_process_memory：Vastbase 实例所占用的内存大小。 • process_used_memory：Vastbase 进程所使用的内存大小。 • max_dynamic_memory：最大动态内存。 • dynamic_used_memory：已使用的动态内存。 • dynamic_peak_memory：内存的动态峰值。 • dynamic_used_shrctx：最大动态共享内存上下文。 • dynamic_peak_shrctx：共享内存上下文的动态峰值。 • max_shared_memory：最大共享内存。 • shared_used_memory：已使用的共享内存。 • max_cstore_memory：列存所允许使用的最大内存。 • cstore_used_memory：列存已使用的内存大小。 • max_sctpcomm_memory：通信库所允许使用的最大内存。 • sctpcomm_used_memory：通信库已使用的内存大小。 • sctpcomm_peak_memory：通信库的内存峰值。 • other_used_memory：其他已使用的内存大小。
memorymbytes	integer	内存类型分配内存的大小。

14.3.84. PG_TIMEZONE_ABBREVS

PG_TIMEZONE_ABBREVS 视图提供了显示了所有可用的时区信息。

表 14-154. PG_TIMEZONE_ABBREVS 字段

名称	类型	描述
abbrev	text	时区名缩写。
utc_offset	interval	相对于 UTC 的偏移量。
is_dst	Boolean	如果当前正处于夏令时范围则为 TRUE，否则为 FALSE。

14.3.85. PG_TOTAL_USER_RESOURCE_INFO_OID

PG_TOTAL_USER_RESOURCE_INFO_OID 视图显示所有用户资源使用情况，需要使用管理员用户进行查询。此视图在参数 use_workload_manager 为 on 时才有效。

表 14-155. PG_TOTAL_USER_RESOURCE_INFO_OID 字段

名称	类型	描述
userid	oid	用户 ID。
used_memory	integer	正在使用的内存大小，单位 MB。
total_memory	integer	可以使用的内存大小，单位 MB。值为 0 表示未限制最大可用内存，其限制取决于数据库最大可用内存。
used_cpu	double precision	正在使用的 CPU 核数。
total_cpu	integer	在该机器节点上，用户关联控制组的 CPU 核数总和。
used_space	bigint	已使用的存储空间大小，单位 KB。
total_space	bigint	可使用的存储空间大小，单位 KB，值为-1 表示未限制最大存储空间。
used_temp_space	bigint	已使用的临时空间大小，单位 KB
total_temp_space	bigint	可使用的临时空间总大小，单位 KB，值为-1 表示未限制。
used_spill_space	bigint	已使用的下盘空间大小。单位 KB。

名称	类型	描述
total_spill_space	bigint	可使用的下盘空间总大小，单位 KB，值为-1 表示未限制。

14.3.86. PG_VARIABLE_INFO

PG_VARIABLE_INFO 视图用于查询 Vastbase 中当前节点的 xid、oid 的状态。

表 14-156. PG_VARIABLE_INFO 字段

名称	类型	描述
node_name	text	节点名称。
nextOid	oid	该节点下一次生成的 oid。
nextXid	xid	该节点下一次生成的事务号。
oldestXid	xid	该节点最老的事务号。
xidVacLimit	xid	强制 autovacuum 的临界点。
oldestXidDB	oid	该节点 datafrozenxid 最小的数据库 oid。
lastExtendCSNLogpage	xid	最后一次扩展 csnlog 的页面号。
startExtendCSNLogpage	xid	csnlog 扩展的起始页面号。
nextCommitSeqNo	xid	该节点下次生成的 csn 号。
latestCompletedXid	xid	该节点提交或者回滚后节点上的最新事务号。
startupMaxXid	xid	该节点关机前的最后一个事务号。

14.3.87. GS_INSTANCE_TIME

提供当前集节点下的各种时间消耗信息，主要分为以下类型:

- DB_TIME: 作业在多核下的有效时间花费。
- CPU_TIME：CPU 时间的消耗。
- EXECUTION_TIME：执行器内花费的时间。
- PARSE_TIME：SQL 解析的时间花费。
- PLAN_TIME：生成 Plan 的时间花费。

- REWRITE_TIME：SQL 重写的时间消耗。
- PL_EXECUTION_TIME ：plpgsql（存储过程）的执行时间.
- PL_COMPILATION_TIME：plpgsql（存储过程）编译时间。
- NET_SEND_TIME：网络上的时间花销。
- DATA_IO_TIME：IO 时间上的花销。

表 14-157. GS_INSTANCE_TIME 字段

名称	类型	描述
stat_id	integer	统计编号。
stat_name	text	类型名称。
value	bigint	时间值(单位：微秒)。

14.3.88. USER_ALL_TABLES

USER_ALL_TABLES 视图提供当前用户的表的信息

列名	类型	描述
schema_name	varchar	表所属模式
table_name	name	表名
tablespace_name	name	包含表的表空间的名称；分区表，临时表和索引组织表的值为 NULL
status	varchar	如果前一个 DROP TABLE 操作失败，则指示该表是不可用的（UNUSABLE）还是有效的（VALID）
temporary	text	指示表是临时表（Y）还是非临时表（N）

14.3.89. USER_CONS_COLUMNS

USER_CONS_COLUMNS 视图提供当前用户的表上约束指定列信息的信息

列名	类型	描述
schema_name	varchar	约束所属的模式
constraint_name	varchar	约束名
table_name	varchar	约束所属的表的名称
column_name	varchar	约束中引用的列的名称
position	smallint	列在对象定义中的位置

14.3.90. USER_CONSTRAINTS

USER_CONSTRAINTS 视图提供当前用户的表上的约束信息

列名	类型	描述
schema_name	varchar	约束所属的模式
constraint_name	varchar	约束名称
constraint_type	text	约束类型
table_name	varchar	表名
index_owner	varchar	索引所有者
index_name	varchar	索引的名称

14.3.91. USER_IND_COLUMNS

USER_IND_COLUMNS 视图提供当前用户的索引中包含的列信息

列名	类型	描述
schema_name	varchar	索引所属模式的名称
index_name	varchar	索引的名称
index_owner	varchar	索引所有者
table_owner	varchar	表所有者
table_name	varchar	索引所属表的名称
column_name	name	列的名称
column_position	smallint	列在索引中的位置
column_length	smallint	列的长度（字节）
char_length	numeric	列的长度（以字符为单位）
descend	char	该索引是否为降序索引

14.3.92. USER_INDEXES

USER_INDEXES 视图提供当前用户的表上的索引信息

列名	类型	描述
owner	varchar	所有者
schema_name	varchar	索引所属模式
index_name	varchar	索引名称

index_type	text	索引类型
table_owner	text	索引表所有者
table_name	varchar	索引表名称
table_type	text	仅为了实现兼容性而提供。始终设置为 TABLE
uniqueness	text	指示索引是 UNIQUE 还是 NONUNIQUE
compression	char	仅为了实现兼容性而提供。始终设置为 N（不压缩
tablespace_name	text	表所在的表空间（如果不是默认表空间）的名称
logging	text	仅为了实现兼容性而提供。始终设置为 LOGGING
status	text	对象的状态（VALID 或 INVALID）
partitioned	char	仅为了实现兼容性而提供。始终设置为 NO。
temporary	char	仅为了实现兼容性而提供。始终设置为 N。
secondary	char	仅为了实现兼容性而提供。始终设置为 N。
join_index	char	仅为了实现兼容性而提供。始终设置为 NO。
dropped	char	仅为了实现兼容性而提供。始终设置为 NO。

14.3.93. USER_OBJECTS

USER_OBJECTS 视图提供当前用户的数据库中所有对象的信息

列名	类型	描述
object_name	name	对象名称
object_id	oid	对象 id
object_type	name	对象类型
namespace	oid	命名空间 id
created	timestamp with time zone	对象创建时间
last_ddl_time	timestamp with time zone	DDL 最后修改时间

14.3.94. USER_PART_KEY_COLUMNS

USER_PART_KEY_COLUMNS 视图提供当前用户的数据库中的分区表的分区键列的信息

列名	类型	描述

schema_name	text	分区表或索引所属模式
name	text	分区表或索引名称
object_type	char	对象类型（TABLE，INDEX）
column_name	text	分区键名称
column_position	integer	分区键所在列位置

14.3.95. USER_PART_TABLES

USER_PART_TABLES 视图提供当前用户的数据库中的分区表信息

列名	类型	描述
schema_name	text	分区表所在的模式名称
table_name	text	表名称
partitioning_type	text	分区方法的类型
subpartitioning_type	varchar	子分区类型
partition_count	bigint	复合分区方法的类型
def_subpartition_count	integer	对于复合分区的默认分区数
partitioning_key_count	integer	分区键值总数
subpartitioning_key_count	integer	复合分区键值总数
status	varchar	该分区状态
def_tablespace_name	varchar	添加分区默认的所在表空间
def_pct_free	numeric	添加分区时 pctfree 默认值
def_pct_used	numeric	添加分区时 Pctused 默认值
def_ini_trans	numeric	添加分区时 Initrans 默认值
def_max_trans	numeric	添加分区 Maxtrans 最大值
def_initial_extent	varchar	添加分区时默认 extent 大小
def_min_extents	varchar	添加分区默认 minextent 数量
def_max_extents	varchar	添加分区默认 maxextents 数量
def_pct_increase	varchar	添加分区默认 pctincrease 值
def_freelists	numeric	添加分区默认 freelists 值
def_freelist_groups	numeric	添加分区默认 freelistgroups 值
def_logging	varchar	添加分区默认是否记录日志
def_compression	varchar	添加分区时是否默认压缩
def_buffer_pool	varchar	添加分区使用的默认 bufferpool 大小

ref_ptn_constraint_name	varchar	引用分区表的约束名称
interval	varchar	间隔值默认字符串

14.3.96.USER_PROCEDURES

USER_PROCEDURES:提供有关当前用户拥有的存储过程的信息

列名	类型	描述
owner	varchar	所有者
object_name	varchar	存储过程名
argument_number	smallint	参数数量

14.3.97.USER_ROLE_PRIVS

USER_ROLE_PRIVS：显示授予当前用户的角色

列名	类型	描述
username	text	用户名，或 PUBLIC
granted_role	text	授予用户的角色名称
admin_option	text	指示是否同时将所授权限的操作权限(即再授权权限)同时赋予用户(YES，NO)
default_role	text	指示角色是否为当前用户默认角色（YES，NO）
os_granted	varchar	指示角色是否由操作系统授予（YES，NO）

14.3.98.USER_SEQUENCES

USER_SEQUENCES 视图提供当前用户的数据库用户定义的序列信息

列名	类型	描述
sequence_owner	varchar	序列所有者
schema_name	varchar	序列所属模式
sequence_name	varchar	序列名称

14.3.99.USER_SOURCE

USER_SOURCE:提供有关当前用户拥有的所有程序的信息

列名	类型	描述

owner	varchar	相对于给定程序的源代码行号
text	text	定义程序的语句
name	varchar	程序名

14.3.100. USER_SUBPART_KEY_COLUMNS

USER_SUBPART_KEY_COLUMNS 视图提供当前用户的数据库分区表的子分区的关键列的信息。

列名	类型	描述
schema_name	text	表所在的模式的名称
name	text	列所在表的名称
object_type	char	对象类型
column_name	text	定义键的列的名称
column_position	integer	列在定义键的位置

14.3.101. USER_SYNONYMS

USER_SYNONYMS 视图提供当前用户同义词信息

列名	类型	描述
schema_name	text	名字空间名
synonym_name	text	同义词名
table_owner	text	引用对象的所有者
table_schema_name	text	引用对象所在的名字空间名
table_name	text	引用对象名
db_link	text	引用数据库连接的名字

14.3.102. USER_TAB_COLUMNS

USER_TAB_COLUMNS 视图提供当前用户的表和视图中所有列的信息

列名	类型	描述
owner	varchar	所有者
schema_name	varchar	表或者视图所在的模式名
table_name	varchar	列所在的表或视图的名称
column_name	varchar	列名

data_type	varchar	列的类型
data_length	integer	文本列的长度
data_precision	integer	数字列的精度（位数）
data_scale	integer	数值列的比例
nullable	char	列是否可为空
column_id	integer	表中列的相对位置
comments	text	描述
data_default	text	指定给列的默认值

14.3.103. USER_TAB_PARTITIONS

USER_TAB_PARTITIONS 视图提供当前用户的数据库中的所有分区的信息

列名	类型	描述
table_owner	varchar	表的所有者
schema	varchar	表所在的模式的名称
table_name	varchar	表的名称
partition_name	varchar	分区的名称
high_value	text	建表语句中指定的高分区值
tablespace_name	name	子分区所在表空间的名称

14.3.104. USER_TAB_SUBPARTITIONS

USER_TAB_SUBPARTITIONS 视图提供当前用户的数据库中的所有子分区的信息

列名	类型	描述
schema_name	text	表所在的模式的名称
table_name	text	表的名称
partition_name	text	分区的名称
subpartition_name	text	子分区名称
high_value	text	建表语句中指定的高子分区值
high_value_length	integer	子分区值的长度
subpartition_position	numeric	子分区的位置
tablespace_name	text	子分区所在表空间的名称
pct_free	numeric	块中可用空间的最小百分比

pct_used	numeric	块中已用空间的最小百分比
ini_trans	numeric	起始允许事务数
max_trans	numeric	最大允许事务数
initial_extent	numeric	以字节为单位的初始数据块大小（对于范围分区）；以块为单位的初始数据块大小（对于复合分区）
next_extent	numeric	以字节为单位的辅助数据块大小（对于范围分区）；以块为单位的辅助数据块大小（对于复合分区）
min_extent	numeric	段中允许的最小范围数
max_extent	numeric	段中允许的最大范围数
pct_increase	numeric	扩展数据块大小增加百分比
freelists	numeric	此段中分配的空闲列表组数
freelist_groups	numeric	此段中分配的空闲列表组数
logging	varchar	子分区的日志属性
compression	varchar	压缩
num_rows	numeric	行数
blocks	doubleprecision	总块数
empty_blocks	numeric	空块数
avg_space	numeric	平均占用大小
chain_cnt	numeric	链的数量
avg_row_len	numeric	平均行长
sample_size	numeric	样本大小
last_analyzed	timestamp without time zone	最近分析此表的日期
buffer_pool	varchar	此子分区的默认缓冲池
global_stats	varchar	指示子分区的列统计信息是通过分析整个表收集的（是）还是根据为分区和子分区收集的统计信息估计的（否）
user_stats	varchar	指示是否由用户直接输入统计信息（是）或否（否）
backing_table	regclass	分区备份表的名称

14.3.105. USER_TABLES

USER_TABLES 视图提供当前用户的数据库用户定义表的信息

列名	类型	描述

schema_name	varchar	表所属模式的名称
table_name	varchar	表名
tablespace_name	varchar	表所在的表空间
status	varchar	表的状态
temporary	char	是否为临时表

14.3.106. USER_TRIGGERS

USER_TRIGGERS 视图提供当前用户的数据库表上的触发器信息

列名	类型	描述
trigger_name	varchar	触发器的名称
table_owner	varchar	定义触发器的表的所有者的用户名
table_name	varchar	定义触发器的表的名称

14.3.107. USER_TYPES

USER_TYPES 视图提供当前用户的数据库的对象类型信息

列名	类型	描述
schema_name	text	定义类型的模式的名称
type_name	text	类型的名称
type_oid	oid	类型的对象标识符
typecode	text	类型的类型代码
attributes	integer	类型中的属性数

14.3.108. USER_USERS

USER_USERS 视图提供当前用户的数据库的用户信息

列名	类型	描述
username	varchar	用户名
user_id	oid	用户编号
account_status	varchar	账号的当前状态
lock_date	timestamp without time zone	显示账号被锁定的日期和时间。
expiry_date	timestamp without time zone	账号的到期日

default_tablespace	varchar	与账号关联的默认表空间
temporary_tablespace	varchar	该账号使用的临时表空间
created	timestamp without time zone	账号创建时间
initial_rsrc_consumer_group	varchar	用户的初始资源使用者组
external_name	varchar	用户外部名称

14.3.109. USER_VIEWS

USER_VIEWS 视图提供当前用户的数据库的视图信息

列名	类型	描述
owner	varchar	所有者
schema_name	text	视图所在模式的名称
view_name	varchar	视图名
text	text	定义视图的查询语句

14.3.110. DBA_ALL_TABLES

DBA_ALL_TABLES 视图提供所有用户的表的信息

列名	类型	描述
owner	varchar	表的所有者
schema_name	varchar	表所在的模式名称
table_name	varchar	表的名称
tablespace_name	varchar	表所在的表空间名称
status	varchar	表所在集群名称
temporary	char	该表是否为临时表空间
dropped	varchar	是否已删除
num_rows	numeric	行数

14.3.111. DBA_SEGMENTS

DBA_SEGMENTS 描述数据库中的所有段分配的存储。

列名	类型	描述
owner	name	段的所有者

segment_name	name	段的名称
segment_type	varchar	段的类型
tablespace_name	name	段所在表空间
header_file	oid	包含段头的文件编号
header_block	oid	包含段头的块的编号
bytes	bigint	段的大小
blocks	double precision	段占用多少块

14.3.112. DBA_CONS_COLUMNS

DBA_CONS_COLUMNS 视图提供在数据库中所有表上指定的约束中包括的所有列的相关信息。

列名	类型	描述
owner	varchar	约束所有者
schema_name	varchar	约束所属模式名
constraint_name	varchar	约束的名称
table_name	varchar	约束所属表的名称
column_name	varchar	约束中引用的列的名称
position	smallint	列在对象定义中的位置
constraint_def	text	约束的定义

14.3.113. DBA_CONSTRAINTS

DBA_CONSTRAINTS 视图提供所有用户的表上的约束信息。

列名	类型	描述
owner	varchar	约束的所有者
schema_name	varchar	约束所在模式的名称
constraint_name	varchar	约束名称
constraint_type	text	约束类型
table_name	varchar	约束所属表名称
search_condition	text	检查约束的条件文本
r_owner	text	引用约束表的所有者
r_constraint_name	text	引用表的唯一约束名称
delete_rule	text	删除规则

deferrable	boolean	是否可以延迟检查
deferred	boolean	约束是否延迟检查
index_owner	varchar	索引的所有者
index_name	varchar	索引的名称
constraint_def	text	约束延迟定义

14.3.114. DBA_IND_COLUMNS

DBA_IND_COLUMNS 视图提供所有用户的索引中包含的列信息。

列名	类型	描述
index_owner	varchar	索引的所有者
schema_name	varchar	索引所在的模式名称
index_name	varchar	索引名称
table_owner	varchar	索引所在表的所有者
table_name	varchar	索引所在表名称
column_name	name	索引所在字段名称
column_position	smallint	列所在索引的位置
column_length	smallint	索引所在的列长
char_length	numeric	列的最大长度
descend	char	该列降序还是升序

14.3.115. DBA_INDEXES

DBA_INDEXES 视图提供所有用户的表上的索引信息。

列名	类型	描述
owner	varchar	索引的所有者
schema_name	varchar	索引所在的模式名称
index_name	varchar	索引名称
index_type	text	索引类型
table_owner	text	索引所在表的所有者
table_name	varchar	索引所在表的名称
table_type	text	索引所在表的类型
uniqueness	text	是否为唯一索引
compression	char	索引是否被压缩

tablespace_name	text	索引所在表空间名称
logging	text	是否记录对索引的修改
status	text	索引的状态
partitioned	char	索引是否为分区索引
temporary	char	是否为临时索引
secondary	char	是否为辅助索引
join_index	char	是否为关联索引
dropped	char	索引是否已删除且在回收站中

14.3.116. DBA_OBJECTS

DBA_OBJECTS 视图提供所有用户的数据库中所有对象的信息。

列名	类型	描述
owner	name	对象的所有者
schema_name	varchar	对象所属的模式名
object_name	name	对象名
object_id	oid	对象 oid
object_type	name	对象类型：index、function、procedure、sequence、synonym、table、trigger、view
namespace	oid	命名空间 oid
created	timestamp with time zone	对象创建时间
last_ddl_time	timestamp with time zone	DDL 最后修改时间

14.3.117. DBA_PART_KEY_COLUMNS

DBA_PART_KEY_COLUMNS 视图提供所有用户的数据库中的分区表的分区键列的信息。

列名	类型	描述
owner	text	分区表或索引的所有者
schema_name	text	分区表或索引所属模式
name	text	分区表或索引的名称
object_type	char	对象类型（TABLE，INDEX）

column_name	text	分区键名称
column_position	integer	分区键所在列位置

14.3.118. DBA_PART_TABLES

DBA_PART_TABLES 视图提供所有用户的数据库中的分区表信息。

列名	类型	描述
table_owner	varchar	分区表的所有者
schema_name	varchar	分区表所在的模式名称
table_name	varchar	表名称
partitioning_type	text	分区方法的类型
partition_count	bigint	分区的总数
partitioning_key_count	integer	分区键值总数
subpartitioning_key_count	integer	复合分区键值总数
status	varchar	该分区状态
def_tablespace_name	name	添加分区默认的所在表空间
def_pct_free	numeric	添加分区时 pctfree 默认值
def_pct_used	numeric	添加分区时 Pctused 默认值
def_ini_trans	numeric	添加分区时 Initrans 默认值
def_max_trans	numeric	添加分区 Maxtrans 最大值
def_initial_extent	varchar	添加分区时默认 extent 大小
def_next_extent	varchar	添加分区时默认下一个 extent 大小
def_min_extents	varchar	添加分区默认 minextent 数量
def_max_extents	varchar	添加分区默认 maxextents 数量
def_pct_increase	varchar	添加分区默认 pctincrease 值
def_freelists	numeric	添加分区默认 freelists 值
def_freelist_groups	numeric	添加分区默认 freelistgroups 值
def_logging	varchar	添加分区默认是否记录日志
def_compression	varchar	添加分区时是否默认压缩
def_buffer_pool	varchar	添加分区使用的默认 bufferpool 大小
ref_ptn_constraint_name	varchar	引用分区表的约束名称
interval	varchar	间隔值默认字符串

14.3.119.DBA_ROLE_PRIVS

DBA_ROLE_PRIVS：包含当前数据库所有角色的权限信息

列名	类型	描述
grantee	text	接受授权的用户或角色名称
granted_role	text	被授权角色名称
admin_option	text	权限是否与 amin option 一起使用
default_role	text	是否被定义为默认

14.3.120.DBA_ROLES

DBA_ROLES：包含当前系统的角色信息

列名	类型	描述
role	text	角色名称
password_required	text	指定角色使用密码认证

14.3.121.DBA_SEQUENCES

DBA_SEQUENCES 视图提供有关所有用户定义序列的信息。

列名	类型	描述
sequence_owner	varchar	序列的所有者
schema_name	varchar	序列所在模式的名称
sequence_name	varchar	序列名称

14.3.122.DBA_SUBPART_KEY_COLUMNS

数据库分区表的子分区键列信息

列名	类型	描述
owner	text	所有者
schema_name	text	所属名字空间
name	text	对象名
object_type	char	对象类型
column_name	text	字段名
column_position	integer	字段位置

14.3.123.DBA_SYNONYMS

数据库同义词信息

列名	类型	描述
owner	text	所有者
schema_name	text	名字空间名
synonym_name	text	同义词名
table_owner	text	引用对象的所有者
table_schema_name	text	引用对象所在的名字空间名
table_name	text	引用对象名
db_link	text	引用数据库连接的名字

14.3.124.DBA_TAB_COLUMNS

DBA_TAB_COLUMNS 视图提供所有用户的表和视图中所有列的信息。

列名	类型	描述
owner	varchar	所有者
schema_name	varchar	名字空间名
table_name	varchar	表、视图名
column_name	varchar	字段名
data_type	varchar	字段类型
data_length	integer	数据长度
data_precision	integer	数字数据精度
data_scale	integer	小数点右边的数字
nullable	char	指定列是否允许空值。
column_id	integer	字段序列号
comments	text	描述
data_default	text	默认值

14.3.125.DBA_TAB_PARTITIONS

DBA_TAB_PARTITIONS 视图提供所有用户的数据库中的所有分区的信息。

列名	类型	描述
table_owner	varchar	表所有者
schema	varchar	名字空间名

table_name	varchar	表名
partition_name	varchar	分区名
high_value	text	分区界限值表达式
tablespace_name	name	表空间名

14.3.126.DBA_TAB_SUBPARTITIONS

DBA_TAB_SUBPARTITIONS 视图提供所有用户的数据库中的所有子分区的信息。

列名	类型	描述
table_owner	text	所有者
schema_name	text	名字空间名
table_name	text	表名
partition_name	text	分区名
subpartition_name	text	子分区名
high_value	text	分区界限值表达式
high_value_length	integer	分区边界值表达式的长度
subpartition_position	numeric	子分区在分区中的位置
tablespace_name	text	表空间名
pct_free	numeric	仅为支持兼容性。该值始终为 0
pct_used	numeric	仅为支持兼容性。该值始终为 0
ini_trans	numeric	仅为支持兼容性。该值始终为 0
max_trans	numeric	仅为支持兼容性。该值始终为 0
initial_extent	numeric	仅为支持兼容性。该值始终为 NULL
next_extent	numeric	仅为支持兼容性。该值始终为 NUll
min_extent	numeric	仅为支持兼容性。该值始终为 0
max_extent	numeric	仅为支持兼容性。该值始终为 0
pct_increase	numeric	仅为支持兼容性。该值始终为 0
freelists	numeric	仅为支持兼容性。该值始终为 NULL
freelist_groups	numeric	仅为支持兼容性。该值始终为 NULL
logging	varchar	仅为支持兼容性。该值始终为 YES
compression	varchar	仅为支持兼容性。该值始终为 NONE
num_rows	numeric	分区中的行数
blocks	double precision	分区中使用的块的数量

empty_blocks	numeric	仅为支持兼容性。该值始终为 NULL
avg_space	numeric	仅为支持兼容性。该值始终为 NULL
chain_cnt	numeric	仅为支持兼容性。该值始终为 NULL
avg_row_len	numeric	仅为支持兼容性。该值始终为 NULL
sample_size	numeric	仅为支持兼容性。该值始终为 NULL
last_analyzed	timestamp without time zone	仅为支持兼容性。该值始终为 NULL
buffer_pool	varchar	仅为支持兼容性。该值始终为 NULL
global_stats	varchar	仅为支持兼容性。该值始终为 YES
user_stats	varchar	仅为支持兼容性。该值始终为 NO
backing_table	regclass	分区表名

14.3.127.DBA_TABLES

DBA_TABLES 视图提供所有用户的数据库用户定义表的信息。

列名	类型	描述
owner	varchar	表所有者
schema_name	varchar	名字空间名
table_name	varchar	表名
tablespace_name	varchar	表空间名
status	varchar	状态
temporary	char	是否为临时表，(Y）是，(N)否

14.3.128.DBA_TRIGGERS

DBA_TRIGGERS 视图提供所有用户的数据库表上的触发器信息。

列名	类型	描述
trigger_name	varchar	触发器名
table_name	varchar	显示触发对象名
table_owner	varchar	所有者名

14.3.129.DBA_TYPES

DBA_TYPES 视图提供所有用户的数据库的对象类型信息。

列名	类型	描述
owner	text	所有者

schema_name	text	名字空间名
type_name	text	类型名
type_oid	oid	类型 id
typecode	text	类型码
attributes	integer	属性号

14.3.130.DBA_USERS

DBA_USERS 视图提供所有用户的数据库的用户信息。

列名	类型	描述
username	varchar	用户名

14.3.131.DBA_VIEWS

DBA_VIEWS 视图提供所有用户的数据库的视图信息。

列名	类型	描述
owner	varchar	所有者名
schema_name	text	名字空间名
view_name	varchar	视图名
text	text	视图定义

14.3.132.ALL_ALL_TABLES

ALL_ALL_TABLES 视图提供有关当前用户可访问的表的信息。

列名	类型	描述
owner	name	表的所有者
schema_name	varchar	表所属的模式名
table_name	name	表名
tablespace_name	name	表所在的表空间
status	varchar	状态
temporary	text	是否为临时表，(Y）是，(N)否

14.3.133.ALL_CONS_COLUMNS

ALL_CONS_COLUMNS 视图提供有关当前用户可访问的表上约束指定列信息的信息。

列名	类型	描述

owner	text	约束列的所有者
schema_name	varchar	约束所属的模式名
constraint_name	varchar	约束名
table_name	varchar	定义约束的表名
column_name	varchar	定义约束的字段名
position	smallint	列在表中定义的位置
constraint_def	text	约束定义

14.3.134.ALL_CONSTRAINTS

ALL_CONSTRAINTS 视图提供当前用户可访问表上的约束信息。

列名	类型	描述
owner	text	约束定义的所有者
schema_name	varchar	约束定义的的模式名
constraint_name	varchar	约束名字（不需要唯一！）
constraint_type	text	c=检查约束，f=外键约束，p=主键约束，u=唯一约束，t=约束触发器，x=排他约束
table_name	varchar	表的名字
search_condition	text	检查约束条件
r_owner	text	约束中引用表的所有者
r_constraint_name	text	被引用表的唯一约束定义的名称
delete_rule	text	外键删除动作代码：a=无动作，r=限制，c=级联，n=置空，d=置为默认值
deferrable	bool	该约束是否能被延迟
deferred	bool	该约束是否默认被延迟
index_owner	varchar	索引的所有者
index_name	varchar	索引的名字
constraint_def	text	约束的定义

14.3.135.ALL_IND_COLUMNS

ALL_IND_COLUMNS 视图提供当前用户可访问索引中包含的列信息。

列名	类型	描述
index_owner	varchar	索引列的所有者

schema_name	varchar	索引列所属的模式名
index_name	varchar	索引名称
table_owner	varchar	表的所有者
table_name	varchar	表名
column_name	name	列名
column_position	smallint	列的编号。一般列从 1 开始向上编号。系统列则拥有（任意）负值编号
column_length	smallint	对于一个固定尺寸的类型，typlen 是该类型内部表示的字节数。对于一个变长类型，typlen 为负值。-1 表示一个“varlena”类型（具有长度字），-2 表示一个以空值结尾的 C 字符串
char_length	numeric	列的数据类型如果为 char，表示字节数。否则为 0
descend	char	默认 DESC。指示列是按降序（DESC）还是按升序（ASC）排序

14.3.136.ALL_INDEXES

ALL_INDEXES 视图提供当前用户可访问表上的索引信息。

列名	类型	描述
owner	varchar	索引的所有者
index_name	varchar	索引名字
table_name	varchar	索引对象的表名
uniqueness	text	索引是唯一的（UNIQUE）还是非唯一的（NONUNIQUE）
generate	varchar	
partitioned	char	索引是否已分区

14.3.137.ALL_OBJECTS

ALL_OBJECTS 视图提供当前用户可访问数据库中所有对象的信息。

列名	类型	描述
owner	name	对象的所有者
object_name	name	对象名称

object_id	oid	对象 oid
object_type	name	对象的类型（例如 INDEX，FUNCTION，PROCEDURE，TABLE，INDEX）
namespace	oid	命名空间 oid
created	timestamp with time zone	对象创建时间
last_ddl_time	timestamp with time zone	DDL 最后修改时间

14.3.138.ALL_PART_KEY_COLUMNS

ALL_PART_KEY_COLUMNS 视图提供当前用户可访问数据库中的分区表的分区键列的信息。

列名	类型	描述
owner	text	分区表的所有者
schema_name	text	分区表所属的模式名
name	text	分区名称
object_type	char	仅为支持兼容性。该值始终为 table
column_name	text	字段名
column_position	integer	分区键中列的位置

14.3.139.ALL_PART_TABLES

ALL_PART_TABLES 视图提供当前用户可访问数据库中的分区表信息。

列名	类型	描述
owner	text	分区表的所有者
schema_name	text	分区表所属的模式名
table_name	text	分区表名
partitioning_type	text	分区类型：range、list、hash
subpartitioning_type	varchar	子分区类型：range、list、hash
partition_count	bigint	分区表中的分区数
def_subpartition_count	integer	复合分区表中，子分区数
partitioning_key_count	integer	分区键的列数

subpartitioning_key_count	integer	复合分区表中，子分区中的列数
status	varchar	仅为支持兼容性。该值始终为 valid
def_tablespace_name	varchar	仅为支持兼容性。该值始终为 null
def_pct_free	numeric	仅为支持兼容性。该值始终为 null
def_pct_used	numeric	仅为支持兼容性。该值始终为 null
def_ini_trans	numeric	仅为支持兼容性。该值始终为 null
def_max_trans	numeric	仅为支持兼容性。该值始终为 null
def_initial_extent	varchar	仅为支持兼容性。该值始终为 null
def_next_extent	varchar	仅为支持兼容性。该值始终为 null
def_min_extents	varchar	仅为支持兼容性。该值始终为 null
def_max_extents	varchar	仅为支持兼容性。该值始终为 null
def_pct_increase	varchar	仅为支持兼容性。该值始终为 null
def_freelists	numeric	仅为支持兼容性。该值始终为 null
def_freelist_groups	numeric	仅为支持兼容性。该值始终为 null
def_logging	varchar	仅为支持兼容性。该值始终为 yes
def_compression	varchar	仅为支持兼容性。该值始终为 none
def_buffer_pool	varchar	仅为支持兼容性。该值始终为 default
ref_ptn_constraint_name	varchar	仅为支持兼容性。该值始终为 null
interval	varchar	仅为支持兼容性。该值始终为 null

14.3.140.ALL_SEQUENCES

ALL_SEQUENCES 视图提供当前用户可访问数据库用户定义的序列信息。

列名	类型	描述
sequence_owner	name	序列的所有者
schema_name	varchar	序列所属的模式名
sequence_name	name	序列名字
min_value	bigint	序列的最小值
max_value	bigint	序列的最大值

increment_by	bigint	序列递增值
cycle_flag	char	序列达到最大值是否循环

14.3.141.ALL_SUBPART_KEY_COLUMNS

ALL_SUBPART_KEY_COLUMNS 视图提供当前用户可访问数据库分区表的子分区的关键列的信息。

列名	类型	描述
owner	text	分区表的所有者
schema_name	text	分区表所属模式名
name	text	字段所在的表名
object_type	char	仅为支持兼容性。该值始终为 table
column_name	text	分区键的列名
column_position	integer	列所在分区键的位置

14.3.142.ALL_TAB_COLUMNS

ALL_TAB_COLUMNS 视图提供当前用户可访问表和视图中所有列的信息。

列名	类型	描述
owner	varchar	字段所在表或视图的所有者
table_name	varchar	表名
column_name	varchar	列名
data_type	varchar	列的数据类型
column_id	integer	列在表中的顺序编号
data_length	integer	列的字节长度
comments	text	描述
avg_col_len	numeric	列的平均长度
nullable	char	列的值是否允许为空
data_precision	integer	数据类型的精度
data_scale	integer	数字小数点右边精度
char_length	numeric	列的长度（以字符为单位）

14.3.143.ALL_TAB_PARTITIONS

ALL_TAB_PARTITIONS 视图提供当前用户可访问数据库中的所有分区的信息。

列名	类型	描述
table_owner	text	分区表的所有者
schema_name	text	分区表所属的模式名
table_name	text	表名称
partition_name	text	分区名
subpartition_count	bigint	子分区数
high_value	text	对应分区范围最大值
high_value_length	integer	对应分区范围最大值的字节长度
partition_position	numeric	分区在表中的位置
tablespace_name	text	分区所在表空间名
pct_free	numeric	仅为支持兼容性
pct_used	numeric	仅为支持兼容性
ini_trans	numeric	仅为支持兼容性
max_trans	numeric	仅为支持兼容性
initial_extent	numeric	仅为支持兼容性
next_extent	numeric	仅为支持兼容性
min_extent	numeric	仅为支持兼容性
max_extent	numeric	仅为支持兼容性
pct_increase	numeric	仅为支持兼容性
freelists	numeric	仅为支持兼容性
freelist_groups	numeric	仅为支持兼容性
logging	varchar	是否开启表的 logging
compression	varchar	分区是否开启压缩
num_rows	numeric	分区中的数据行数
blocks	integer	分区中数据块数

empty_blocks	numeric	仅为支持兼容性
avg_space	numeric	仅为支持兼容性
chain_cnt	numeric	仅为支持兼容性
avg_row_len	numeric	仅为支持兼容性
sample_size	numeric	仅为支持兼容性
last_analyzed	timestamp	仅为支持兼容性
buffer_pool	varchar	仅为支持兼容性
global_stats	varchar	仅为支持兼容性
user_stats	varchar	仅为支持兼容性
backing_table	regclass	仅为支持兼容性

14.3.144.ALL_TAB_SUBPARTITIONS

ALL_TAB_SUBPARTITIONS 视图提供当前用户可访问数据库中的所有子分区的信息。

列名	类型	描述
table_owner	text	子分区表的所有者
schema_name	text	子分区表所属的模式名
table_name	text	表名
partition_name	text	分区名
subpartition_name	text	子分区名字
high_value	text	子分区范围最大值
high_value_length	integer	子分区范围最大值的字节长度
subpartition_position	numeric	子分区字段所在表中的位置
tablespace_name	text	子分区所在表空间名
pct_free	numeric	仅为支持兼容性，改值始终为 0
pct_used	numeric	仅为支持兼容性，改值始终为 0
ini_trans	numeric	仅为支持兼容性，改值始终为 0
max_trans	numeric	仅为支持兼容性，改值始终为 0
initial_extent	numeric	仅为支持兼容性，改值始终为 null

next_extent	numeric	仅为支持兼容性，改值始终为 null
min_extent	numeric	仅为支持兼容性，改值始终为 0
max_extent	numeric	仅为支持兼容性，改值始终为 0
pct_increase	numeric	仅为支持兼容性，改值始终为 0
freelists	numeric	仅为支持兼容性，改值始终为 null
freelist_groups	numeric	仅为支持兼容性，改值始终为 null
logging	varchar	仅为支持兼容性，改值始终为 YES
compression	varchar	仅为支持兼容性，改值始终为 None
num_rows	numeric	子分区的数据量
blocks	integer	子分区的数据块数
empty_blocks	numeric	仅为支持兼容性，改值始终为 null
avg_space	numeric	仅为支持兼容性，改值始终为 null
chain_cnt	numeric	仅为支持兼容性，改值始终为 null
avg_row_len	numeric	仅为支持兼容性，改值始终为 null
sample_size	numeric	仅为支持兼容性，改值始终为 null
last_analyzed	timestamp	仅为支持兼容性，改值始终为 null
buffer_pool	varchar	仅为支持兼容性，改值始终为 null
global_stats	varchar	仅为支持兼容性，改值始终为 YES
user_stats	varchar	仅为支持兼容性，改值始终为 NO
backing_table	regclass	子分区所属表名

14.3.145. ALL_TAB_COLS

ALL_TAB_COLS 记录了用户所能访问的所有表字段相关的信息。

列名	类型	描述
owner	name	所有者
table_name	information_schema.sql_identifier	列所在的表或视图的名称
column_name	information_schema.sql	列名

	_identifier	
data_type	information_schema.character_data	列的类型
data_type_mod	text	列的类型
data_type_owner	name	
data_length	integer	文本列的长度
data_precision	information_schema.cardinal_number	数字列的精度（位数）
data_scale	information_schema.cardinal_number	数值列的比例
nullable	information_schema.yes_or_no	列是否可为空
column_id	information_schema.cardinal_number	表中列的相对位置
default_length	text	
data_default	information_schema.character_data	指定给列的默认值
num_distinct	real	
low_value	text	
high_value	text	
density	double precision	
num_nulls	text	
num_buckets	text	
last_analyzed	text	
sample_size	text	
character_set_name	text	
char_col_decl_length	information_schema.cardinal_number	
global_stats	text	
user_stats	text	
avg_col_len	integer	
char_length	information_schema.cardinal_number	
char_used	text	
v80_fmt_image	text	
data_upgraded	text	

hidden_column	text	
virtual_column	text	
segment_column_id	text	
internal_column_id	information_schema.cardinal_number	
histogram	text	
qualified_col_name	information_schema.sql_identifier	

14.3.146.ALL_TABLES

ALL_TABLES 视图提供当前用户可访问数据库用户定义表的信息。

列名	类型	描述
owner	text	表的所有者
schema_name	text	表所属的模式名
table_name	text	表名
tablespace_name	text	表所在的表空间
status	varchar	表是否有效
temporary	char	是否为临时表:是(Y)、否(N)

14.3.147.ALL_TRIGGERS

ALL_TRIGGERS 视图提供当前用户可访问数据库表上的触发器信息。

列名	类型	描述
owner	text	触发器的所有者
schema_name	text	触发器所属的模式名
trigger_name	text	触发器名称
trigger_type	text	触发类型: before/after
triggering_event	text	触发事件: insert/update/delete/truncate
table_owner	text	表的所有者
base_object_type	text	触发对象：表

table_name	text	表名
referencing_names	text	仅为支持兼容性
status	text	触发器是否生效
description	text	仅为支持兼容性
trigger_body	text	触发器定义
action_statement	text	执行命令

14.3.148.ALL_TYPES

ALL_TYPES 视图提供当前用户可访问数据库的对象类型信息。

列名	类型	描述
owner	text	类型的所有者
schema_name	text	类型所属的模式名
type_name	text	类型的名称
type_oid	oid	类型的编号
typecode	text	类型码
attributes	integer	属性码

14.3.149. ALL_USERS

ALL_USERS 视图提供当前用户可访问数据库的用户信息。

列名	类型	描述
username	name	用户名
user_id	oid	用户名所属的编号

14.3.150.ALL_VIEWS

ALL_VIEWS 视图提供当前用户可访问数据库的视图信息。

列名	类型	描述
owner	name	视图的所有者
view_name	name	视图名称
text_length	integer	
text	text	视图定义

14.3.151.V$DATABASE

V$DATABASE 包含当前数据库的基本信息

列名	类型	描述
dbid	oid	数据库唯一标识符
name	name	数据库名称
created	timestamp with time zone	数据库的创建时间
resetlogs_change#	integer	打开的重置日志中的系统更改号
resetlogs_time	timestamp with time zone	打开的重置日志的时间戳
prior_resetlogs_change#	integer	先前的重置日志中的版本号
prior_resetlogs_time	text	先前重置日志的时间
log_mode	text	归档日志模式
checkpoint_change#	integer	上一个 scn 检查点
archive_change#	integer	数据库强制归档 scn
controlfile_type	text	控制文件类型
controlfile_created	text	控制文件的创建日期
controlfile_sequence#	integer	控制文件序列号由控制文件事务增加
controlfile_change#	integer	备份控制文件中的最后一个 scn；如果控制文件不是备份，则为 null
controlfile_time	text	备份控制文件中的最后一个时间戳；如果控制文件不是备份，则为 null
open_resetlogs	integer	表明下一个打开的数据库是允许还是需要 resetlogs 选项
version_time	text	版本时间
open_mode	text	数据库打开模式
protection_mode	text	当前数据库的保护模式
protection_level	text	当前数据库的保护级别
remote_archive	text	remote_archive_enable 初始化参数的值
activation#	integer	分配给数据库实例的编号
switchover#	integer	分配给数据库切换的编号
database_role	text	数据库的当前角色
archivelog_change#	integer	归档日志的最高编号
archivelog_compression	text	归档日志的压缩状态

switchover_status	text	表明允许切换
dataguard_broker	text	dataguard 代理信息
guard_status	text	数据保护状态
supplemental_log_data_min	text	确保 logminer 有足够的信息进行工作
supplemental_log_data_pk	text	将所有具有主键的表的键值放入重做日志中
supplemental_log_data_ui	text	对于所有拥有唯一键的表，指示修改时放入重做日志中
force_logging	text	指示数据库当前是否为强制日志模式
platform_id	integer	数据库的平台标示号
platform_name	text	数据库的平台名称
recovery_target_incarnation#	integer	该数据库恢复的所有文件的编号
last_open_incarnation#	integer	v$database_incarnation 成功打开的记录号
current_scn	integer	当前的 scn 号
flashback_on	text	指示闪回功能是否开启
supplemental_log_data_fk	text	对于拥有外键的表，修改时是否将键值放入重做日志中
supplemental_log_data_all	text	对于所有列，修改时是否全部放入重做日志中
db_unique_name	name	数据库的唯一标示名称
standby_became_primary_scn	integer	备机成为主机的 scn 号
fs_failover_status	text	快速故障转移的状态
fs_failover_current_target	text	db_unique_name 备用数据库
fs_failover_threshold	integer	尝试使用备用机前的观察时间（单位为秒）
fs_failover_observer_present	text	指示监控者是否连接到数据库
fs_failover_observer_host	text	当前监控器的主机名
controlfile_converted	text	指示还原期间是否隐式转换控制文件
primary_db_unique_name	text	对于任何节点数据库，此参数标明主节点名称
supplemental_log_data_pl	text	在支持复制工作时，是否额外在重做日志中记录内部的调用信息
min_required_capture_change#	text	required_checkpoint_scn 数据库上所有

		本地捕获过程的最小值

14.3.152.V$INSTANCE

V$INSTANCE 描述当前系统实例信息

列名	类型	描述
instance_number	integer	实例编号
instance_name	name	实例名称
host_name	text	当前实例所在主机名称
version	text	数据库版本信息
startup_time	timestamp withtime zone	数据库启动时间
status	text	数据库状态
parallel	text	指明数据库是否在集群中
thread#	integer	数据库打开的重做线程数
archiver	text	自动归档信息
log_switch_wait	text	记录切换日志的等待事件
logins	text	指明数据库是否处于受限状态
shutdown_pending	text	是否正在关闭
database_status	text	当前数据库状态
instance_role	text	当前实例角色信息
active_state	text	当前实例的活动状态
blocked	text	当前实例是否阻塞
edition	text	版本信息

14.3.153.V$LICENSE

V$LICENSE 包含当前系统 license 信息

列名	类型	描述
sessions_max	integer	允许的最大会话数
sessions_warning	integer	并发用户的警告阈值
sessions_current	integer	当前并发的会话数量
sessions_highwater	integer	会话高水位线
users_max	integer	数据库允许的最大用户数

cpu_count_current	text	系统上当前逻辑 cpu 数
cpu_core_count_current	text	系统上当前物理 cpu 数
cpu_socket_count_current	text	系统上当前的 cpu 插槽数
cpu_count_highwater	text	自实例启动以来，系统上逻辑 cpu 的最大值
cpu_core_count_highwater	text	自实例启动以来，系统上的最大 cpu 物理核
cpu_socket_count_highwater	text	自实例启动以来，系统上的最大 cpu 插槽数

14.3.154.V$VERSION

V$VERSUIB 包含当前软件的版本信息

列名	类型	描述
banner	text	当前的组件名称与版本号

14.3.155.V$OBSOLETE_PARAMETER

V$OBSOLETE_PARAMETER 包含当前实例的初始化参数信息

列名	类型	描述
name	text	参数名称
isspecified	boolean	指明参数是否在参数文件中指定

14.3.156.V$OPTION

V$OPTION 包含当前数据库扩展功能信息

列名	类型	描述
parameter	name	选项功能的名称
value	boolean	该功能是否开启

14.3.157.V$PARAMETER

V$PARAMETER 包含当前实例参数的详细信息

列名	类型	描述
num	bigint	参数编号
name	text	参数名称
type	integer	参数类型

value	text	当前会话的参数值
display_value	text	参数值的智能显示
isdefault	boolean	是否为默认值
isses_modifiable	boolean	是否可以通过 altersession 修改
issys_modifiable	boolean	是否可以通过 altersystem 修改
isinstance_modifiable	boolean	集群中每个实例参数是否可以不同
ismodified	boolean	该参数是否已修改
isadjusted	boolean	数据库实例的建议值
isdeprecated	boolean	该参数是否已弃用
isbasic	boolean	该参数是否为基本参数
description	text	参数描述
update_comment	text	参数更新的相关注释
hash	text	参数名的哈希值

14.3.158.V$TABLESPACE

V$TABLESPACE 包含当前实例表空间的信息

列名	类型	描述
ts#	bigint	表空间编号
name	name	表空间名称
included_in_database_backup	text	该表空间是否可以包含在备份集
bigfile	text	是否开启大文件
flashback_on	text	是否参与到 flashbackdatabase 操作中
encrypt_in_backup	text	该表空间是否为加密

14.3.159.V$OBJECT_USAGE

V$OBJECT_USAGE 包含数据库收集的用户索引使用情况的统计信息

列名	类型	描述
index_name	name	索引名称
table_name	name	索引所在的表名
monitoring	text	该索引是否被监控
used	text	该索引是否被使用
start_monitoring	text	开始监控的时间

end_monitoring	text	停止监控的时间

14.3.160.V$DBLINK

V$DBLINK 包含当前会话在使用 dblink 时所有链接信息，在事务提交或回滚时关闭

列名	类型	描述
db_link	text	数据库链接的名称
owner_id	text	数据库链接 uid 的所有者
logged_on	text	数据库链接当前是否已登录
heterogeneous	text	数据库链接是否异构
protocol	text	数据库链接的通信协议
open_cursors	text	数据库链接是否有打开的游标
in_transaction	text	数据库链接当前是否在事务中
update_sent	text	数据库链接上是否有更新
commit_point_strength	text	数据库链接上事务的提交点强度

15. DBLINK

功能描述

当用户需要跨越本地数据库，访问远程数据库的数据时，可以通过 DBLINK 像访问本地数据库一样访问远程数据库表中的数据。

示例

```
1.创建扩展
create extension dblink;
2.先执行 dblink_connect 保持连接
SELECT dblink_connect('mycoon','hostaddr=172.16.103.92 port=6036 dbname=vastbase user=lst
password=Bigdata@123');
3.执行 BEGIN 命令
SELECT dblink_exec('mycoon', 'BEGIN');
4.执行数据操作
SELECT dblink_exec('mycoon', 'create table people(id int,info varchar(10))');
SELECT dblink_exec('mycoon', 'insert into people values(1,''foo'')');
SELECT dblink_exec('mycoon', 'insert into people values(2,''foo'')');
SELECT dblink_exec('mycoon', 'update people set info=''bar'' where id=1');
5.执行事务提交
SELECT dblink_exec('mycoon', 'COMMIT');
6.执行查询
select * from dblink('mycoon','select * from people') as testTable (id int,info varchar(10));
7.解除连接
SELECT dblink_disconnect('mycoon');
```

16. 外部数据封装器

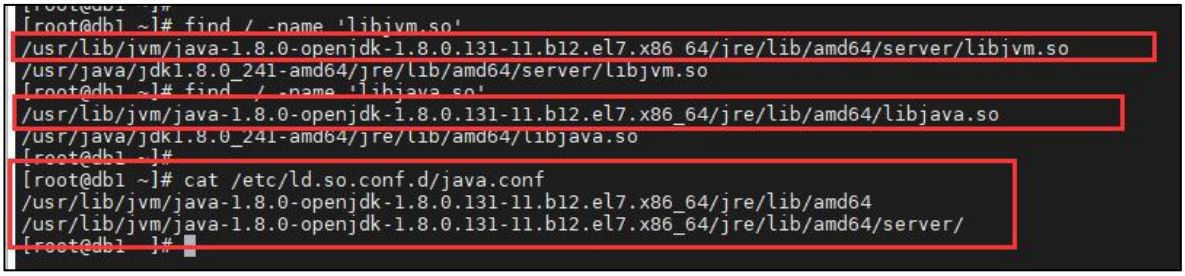
16.1. JDBC FDW

功能描述

支持创建外部数据封装器，通过 JDBC 连接 Oracle、MySQL 等数据库，并能在外部表上进行查询操作。

前置条件

1．安装 JDK1.8，并将 jdk 的库目录加到系统库加载目录。先找到 JDK 的库目录，然后将 libjvm.so 和 libjava.so 所在目录写到文件/etc/ld.so.conf.d/java.conf 中。数据库操作系统用户需要有文件 /etc/ld.so.conf.d/java.conf 的读权限。



2．配置环境变量 GAUSSHOME，并修改数据库配置参数：在 shared_preload_libraries 参数配置中加上 jdbc_fdw，然后重启数据库。

示例

```
1.加载 jdbc_fdww 扩展
create extension jdbc_fdw;

2.在 oracle 创建表
create table emp_fdw(empno int,ename varchar(30));
insert into emp_fdw values(1,'foo');
insert into emp_fdw values(2,'bar');

3.创建连接
create server ora_jdbc foreign data wrapper jdbc_fdw options(
drivername 'oracle.jdbc.driver.OracleDriver',
url 'jdbc:oracle:thin:@//172.16.103.104:1521/orcl',
querytimeout '100',
jarfile '/home/vastbase/bin/ojdbc7.jar',
maxheapsize '200'
);

4.新建用户并授权
create user use_ora_jdbc password 'Bigdata@123';
grant usage on foreign server ora_jdbc to use_ora_jdbc;
```

```
5.创建到oracle的映射
create user mapping for use_ora_jdbc server ora_jdbc options(username 'system',password 'root');

6.创建需要访问的oracle中对应表的结构
create foreign table emp_fdw_ora(empno int,ename varchar(30))
server ora_jdbc options(table 'EMP_FDW');
grant all on emp_fdw_ora to use_ora_jdbc;

7.查看外部表
\c - use_ora_jdbc
select * from emp_fdw_ora;
```

17. DBE_PERF Schema

DBE_PERF Schema 内视图主要用来诊断性能问题，也是 WDR Snapshot 的数据来源。数据库安装后，默认只有初始用户和监控管理员具有模式 dbe_perf 的权限。若是由老版本升级而来，为保持权限的前向兼容，模式 dbe_perf 的权限与老版本保持一致。从 OS、Instance、Memory 等多个维度划分组织视图，并且符合如下命名规范：

- GLOBAL_开头的视图，代表从数据库节点请求数据，并将数据追加对外返回，不会处理数据。
- SUMMARY_开头的视图，代表是将 Vastbase 内的数据概述，多数情况下是返回数据库节点(有时只有数据库主节点的)的数据，会对数据进行加工和汇聚。
- 非这两者开头的视图，一般代表本地视图，不会向其它数据库节点请求数据。

17.1. OS

17.1.1. OS_RUNTIME

显示当前操作系统运行的状态信息。

表 17-1. OS_RUNTIME 字段

名称	类型	描述
id	integer	编号。
name	text	操作系统运行状态名称。
value	numeric	操作系统运行状态值。
comments	text	操作系统运行状态注释。
cumulative	boolean	操作系统运行状态的值是否为累加值。

17.1.2. GLOBAL_OS_RUNTIME

提供 Vastbase 中所有正常节点下的操作系统运行状态信息。查询视图必须具有 monadmin 权限。

表 17-2. GLOBAL_OS_RUNTIME 字段

名称	类型	描述
node_name	name	数据库进程名称。
id	integer	编号。
name	text	操作系统运行状态名称。
value	numeric	操作系统运行状态值。
comments	text	操作系统运行状态注释。
cumulative	boolean	操作系统运行状态的值是否为累加值。

17.1.3. OS_THREADS

提供当前节点下所有线程的状态信息。

表 17-3. OS_THREADS 字段

名称	类型	描述
node_name	text	数据库进程名称。
pid	bigint	数据库进程中正在运行的线程号。
lwpid	integer	与 pid 对应的轻量级线程号。
thread_name	text	与 pid 对应的线程名称。
creation_time	timestamp with time zone	与 pid 对应的线程创建的时间。

17.1.4. GLOBAL_OS_THREADS

提供 Vastbase 中所有正常节点下的线程状态信息。查询视图必须具有 monadmin 权限。

表 17-4. GLOBAL_OS_THREADS 字段

名称	类型	描述
node_name	text	数据库进程名称。
pid	bigint	当前节点进程中正在运行的线程号。
lwpid	integer	与 pid 对应的轻量级线程号。
thread_name	text	与 pid 对应的线程名称。
creation_time	timestamp with time zone	与 pid 对应的线程创建的时间。

17.2. Instance

17.2.1. INSTANCE_TIME

提供当前集群节点下的各种时间消耗信息，主要分为以下类型：

- DB_TIME：作业在多核下的有效时间花费。
- CPU_TIME：CPU 时间的消耗。
- EXECUTION_TIME：执行器内花费的时间。
- PARSE_TIME：SQL 解析的时间花费。
- PLAN_TIME：生成 Plan 的时间花费。
- REWRITE_TIME：SQL 重写的时间消耗。
- PL_EXECUTION_TIME ：plpgsql（存储过程）的执行时间。
- PL_COMPILATION_TIME：plpgsql（存储过程）编译时间。
- NET_SEND_TIME：网络上的时间花销。
- DATA_IO_TIME：IO 时间上的花销。

表 17-5. INSTANCE_TIME 字段

名称	类型	描述
stat_id	integer	统计编号。
stat_name	text	类型名称。
value	bigint	时间值（单位：微秒）。

17.2.2. GLOBAL_INSTANCE_TIME

提供 Vastbase 中所有正常节点下的各种时间消耗信息(时间类型见 instance_time 视图)。查询视图必须具有 monadmin 权限。

表 17-6. GLOBAL_INSTANCE_TIME 字段

名称	类型	描述
node_name	name	数据库进程的名称。
stat_id	integer	统计编号。
stat_name	text	类型名称。
value	bigint	时间值（单位：微秒）。

17.3. Memory

17.3.1. MEMORY_NODE_DETAIL

显示某个数据库节点内存使用情况。

表 17-7. MEMORY_NODE_DETAIL 字段

名称	类型	描述
nodename	text	节点名称。
memorytype	text	内存的名称。 • max_process_memory：Vastbase 实例所占用的内存大小。 • process_used_memory：进程所使用的内存大小。 • max_dynamic_memory：最大动态内存。 • dynamic_used_memory：已使用的动态内存。 • dynamic_peak_memory：内存的动态峰值。 • dynamic_used_shrctx：最大动态共享内存上下文。 • dynamic_peak_shrctx：共享内存上下文的动态峰值。 • max_shared_memory：最大共享内存。 • shared_used_memory：已使用的共享内存。

名称	类型	描述
		• max_cstore_memory：列存所允许使用的最大内存。 • cstore_used_memory：列存已使用的内存大小。 • max_sctpcomm_memory：sctp 通信所允许使用的最大内存。 • sctpcomm_used_memory：sctp 通信已使用的内存大小。 • sctpcomm_peak_memory：sctp 通信的内存峰值。 • other_used_memory：其他已使用的内存大小。 • gpu_max_dynamic_memory：GPU 最大动态内存。 • gpu_dynamic_used_memory：GPU 已使用的动态内存。 • gpu_dynamic_peak_memory：GPU 内存的动态峰值。 • pooler_conn_memory：链接池申请内存计数。 • pooler_freeconn_memory：链接池空闲连接的内存计数。 • storage_compress_memory：存储模块压缩使用的内存大小。 • udf_reserved_memory：UDF 预留的内存大小。
memorymbytes	integer	内存使用的大小，单位为 MB。

17.3.2. GLOBAL_MEMORY_NODE_DETAIL

显示前 Vastbase 中所有正常节点下的内存使用情况。查询视图必须具有 monadmin 权限。

表 17-8. GLOBAL_MEMORY_NODE_DETAIL 字段

名称	类型	描述
nodename	text	数据库进程名称。
memorytype	text	内存使用的名称。 • max_process_memory：Vastbase 实例所占用的内存大小。 • process_used_memory：进程所使用的内存大小。 • max_dynamic_memory：最大动态内存。 • dynamic_used_memory：已使用的动态内存。 • dynamic_peak_memory：内存的动态峰值。 • dynamic_used_shrctx：最大动态共享内存上下文。

名称	类型	描述
		• dynamic_peak_shrctx：共享内存上下文的动态峰值。 • max_shared_memory：最大共享内存。 • shared_used_memory：已使用的共享内存。 • max_cstore_memory：列存所允许使用的最大内存。 • cstore_used_memory：列存已使用的内存大小。 • max_sctpcomm_memory：sctp 通信所允许使用的最大内存。 • sctpcomm_used_memory：sctp 通信已使用的内存大小。 • sctpcomm_peak_memory：sctp 通信的内存峰值。 • other_used_memory：其他已使用的内存大小。 • gpu_max_dynamic_memory：GPU 最大动态内存。 • gpu_dynamic_used_memory：GPU 已使用的动态内存。 • gpu_dynamic_peak_memory：GPU 内存的动态峰值。 • pooler_conn_memory：链接池申请内存计数。 • pooler_freeconn_memory：链接池空闲连接的内存计数。 • storage_compress_memory：存储模块压缩使用的内存大小。 • udf_reserved_memory：UDF 预留的内存大小。
memorymbytes	integer	内存使用的大小，单位为 MB。

17.3.3. SHARED_MEMORY_DETAIL

查询当前节点所有已产生的共享内存上下文的使用信息。

表 17-9. SHARED_MEMORY_DETAIL 字段

名称	类型	描述
contextname	text	内存上下文的名称。
level	smallint	内存上下文的级别。
parent	text	上级内存上下文。
totalsize	bigint	共享内存总大小(单位：字节)。

名称	类型	描述
freesize	bigint	共享内存剩余大小(单位：字节)。
usedsize	bigint	共享内存使用大小(单位：字节)。

17.3.4. GLOBAL_SHARED_MEMORY_DETAIL

查询Vastbase中所有正常节点下的共享内存上下文的使用信息。查询视图必须具有monadmin权限。

表 17-10. GLOBAL_SHARED_MEMORY_DETAIL 字段

名称	类型	描述
node_name	name	数据库进程名称。
contextname	text	内存上下文的名称。
level	smallint	内存上下文的级别。
parent	text	上级内存上下文。
totalsize	bigint	共享内存总大小(单位：字节)。
freesize	bigint	共享内存剩余大小(单位：字节)。
usedsize	bigint	共享内存使用大小(单位：字节)。

17.4. File

17.4.1. FILE_IOSTAT

通过对数据文件 IO 的统计，反映数据的 IO 性能，用以发现 IO 操作异常等性能问题。

表 17-11. FILE_IOSTAT 字段

名称	类型	描述
filenum	oid	文件标识。
dbid	oid	数据库标识。

名称	类型	描述
spcid	oid	表空间标识。
phyrds	bigint	读物理文件的数目。
phywrts	bigint	写物理文件的数目。
phyblkrd	bigint	读物理文件块的数目。
phyblkwrt	bigint	写物理文件块的数目。
readtim	bigint	读文件的总时长（单位：微秒）。
writetim	bigint	写文件的总时长（单位：微秒）。
avgiotim	bigint	读写文件的平均时长（单位：微秒）。
lstiotim	bigint	最后一次读文件时长（单位：微秒）。
miniotim	bigint	读写文件的最小时长（单位：微秒）。
maxiowtm	bigint	读写文件的最大时长（单位：微秒）。

17.4.2. SUMMARY_FILE_IOSTAT

通过 Vastbase 内对数据文件汇聚 IO 的统计，反映数据的 IO 性能，用以发现 IO 操作异常等性能问题。查询视图必须具有 monadmin 权限。

表 17-12. SUMMARY_FILE_IOSTAT 字段

名称	类型	描述
filenum	oid	文件标识。
dbid	oid	数据库标识。
spcid	oid	表空间标识。
phyrds	numeric	读物理文件的数目。
phywrts	numeric	写物理文件的数目。
phyblkrd	numeric	读物理文件块的数目。
phyblkwrt	numeric	写物理文件块的数目。

名称	类型	描述
readtim	numeric	读文件的总时长（单位：微秒）。
writetim	numeric	写文件的总时长（单位：微秒）。
avgiotim	bigint	读写文件的平均时长（单位：微秒）。
lstiotim	bigint	最后一次读文件时长（单位：微秒）。
miniotim	bigint	读写文件的最小时长（单位：微秒）。
maxiowtm	bigint	读写文件的最大时长（单位：微秒）。

17.4.3. GLOBAL_FILE_IOSTAT

得到所有节点上的数据文件 IO 统计信息。查询视图必须具有 monadmin 权限。

表 17-13. GLOBAL_FILE_IOSTAT 字段

名称	类型	描述
node_name	name	数据库进程名称
filenum	oid	文件标识。
dbid	oid	数据库标识。
spcid	oid	表空间标识。
phyrds	bigint	读物理文件的数目。
phywrts	bigint	写物理文件的数目。
phyblkrd	bigint	读物理文件块的数目。
phyblkwrt	bigint	写物理文件块的数目。
readtim	bigint	读文件的总时长（单位：微秒）。
writetim	bigint	写文件的总时长（单位：微秒）。
avgiotim	bigint	读写文件的平均时长（单位：微秒）。
lstiotim	bigint	最后一次读文件时长（单位：微秒）。
miniotim	bigint	读写文件的最小时长（单位：微秒）。

名称	类型	描述
maxiowtm	bigint	读写文件的最大时长（单位：微秒）。

17.4.4. FILE_REDO_IOSTAT

本节点 Redo(WAL)相关的统计信息。

表 17-14. FILE_REDO_IOSTAT 字段

名称	类型	描述
phywrts	bigint	向 wal buffer 中写的次数。
phyblkwrt	bigint	向 wal buffer 中写的 block 的块数。
writetim	bigint	向 xlog 文件中写操作的时间（单位：微秒）。
avgiotim	bigint	平均写 xlog 的时间(writetim/phywrts)（单位：微秒）。
lstiotim	bigint	最后一次写 xlog 的时间（单位：微秒）。
miniotim	bigint	最小的写 xlog 时间（单位：微秒）。
maxiowtm	bigint	最大的写 xlog 时间（单位：微秒）。

17.4.5. SUMMARY_FILE_REDO_IOSTAT

Vastbase 内汇聚所有的 Redo(WAL)相关的统计信息。查询视图必须具有 monadmin 权限。

表 17-15. SUMMARY_FILE_REDO_IOSTAT 字段

名称	类型	描述
phywrts	numeric	向 wal buffer 中写的次数。
phyblkwrt	numeric	向 wal buffer 中写的 block 的块数。
writetim	numeric	向 xlog 文件中写操作的时间（单位：微秒）。
avgiotim	bigint	平均写 xlog 的时间(writetim/phywrts)（单位：微秒）。
lstiotim	bigint	最后一次写 xlog 的时间（单位：微秒）。

名称	类型	描述
miniotim	bigint	最小的写 xlog 时间（单位：微秒）。
maxiowtm	bigint	最大的写 xlog 时间（单位：微秒）。

17.4.6. GLOBAL_FILE_REDO_IOSTAT

得到 Vastbase 内各节点的 Redo(WAL)相关统计信息。查询视图必须具有 monadmin 权限。

表 17-16. GLOBALXC_FILE_REDO_IOSTAT 字段

名称	类型	描述
node_name	name	数据库进程名称。
phywrts	bigint	向 wal buffer 中写的次数。
phyblkwrt	bigint	向 wal buffer 中写的 block 的块数。
writetim	bigint	向 xlog 文件中写操作的时间（单位：微秒）。
avgiotim	bigint	平均写 xlog 的时间(writetim/phywrts)（单位：微秒）。
lstiotim	bigint	最后一次写 xlog 的时间（单位：微秒）。
miniotim	bigint	最小的写 xlog 时间（单位：微秒）。
maxiowtm	bigint	最大的写 xlog 时间（单位：微秒）。

17.4.7. LOCAL_REL_IOSTAT

获取当前节点中数据文件 IO 状态的累计值，显示为所有数据文件 IO 状态的总和。

表 17-17. LOCAL_REL_IOSTAT 字段

名称	类型	描述
phyrds	bigint	读物理文件的数目。
phywrts	bigint	写物理文件的数目。
phyblkrd	bigint	读物理文件的块的数目。

名称	类型	描述
phyblkwrt	bigint	写物理文件的块的数目。

17.4.8. GLOBAL_REL_IOSTAT

获取所有节点上的数据文件 IO 统计信息，查询视图必须具有 monadmin 权限。

表 17-18. GLOBAL_REL_IOSTAT 字段

名称	类型	描述
node_name	name	数据库进程名称
phyrds	bigint	读物理文件的数目。
phywrts	bigint	写物理文件的数目。
phyblkrd	bigint	读物理文件块的数目。
phyblkwrt	bigint	写物理文件块的数目。

17.4.9. SUMMARY_REL_IOSTAT

获取所有节点上的数据文件 IO 统计信息，查询视图必须具有 monadmin 权限。

表 17-19. SUMMARY_REL_IOSTAT 字段

名称	类型	描述
phyrds	numeric	读物理文件的数目。
phywrts	numeric	写物理文件的数目。
phyblkrd	numeric	读物理文件的块的数目。
phyblkwrt	numeric	写物理文件的块的数目。

17.5. Object

17.5.1. STAT_USER_TABLES

显示当前节点所有命名空间中用户自定义普通表的状态信息。

表 17-20. STAT_USER_TABLES 字段

名称	类型	描述
relid	oid	表的 OID。
schemaname	name	该表的模式名。
relname	name	表名。
seq_scan	bigint	该表发起的顺序扫描数。
seq_tup_read	bigint	顺序扫描抓取的活跃行数。
idx_scan	bigint	该表发起的索引扫描数。
idx_tup_fetch	bigint	索引扫描抓取的活跃行数。
n_tup_ins	bigint	插入行数。
n_tup_upd	bigint	更新行数。
n_tup_del	bigint	删除行数。
n_tup_hot_upd	bigint	HOT 更新行数（即没有更新所需的单独索引）。
n_live_tup	bigint	估计活跃行数。
n_dead_tup	bigint	估计死行数。
last_vacuum	timestamp with time zone	最后一次该表是手动清理的（不计算 VACUUM FULL）时间。
last_autovacuum	timestamp with time zone	上次被 autovacuum 守护进程清理的时间。
last_analyze	timestamp with time zone	上次手动分析该表的时间。
last_autoanalyze	timestamp with time zone	上次被 autovacuum 守护进程分析的时间。
vacuum_count	bigint	该表被手动清理的次数（不计算 VACUUM FULL）。

名称	类型	描述
autovacuum_count	bigint	该表被 autovacuum 清理的次数。
analyze_count	bigint	该表被手动分析的次数。
autoanalyze_count	bigint	该表被 autovacuum 守护进程分析的次数。

17.5.2. SUMMARY_STAT_USER_TABLES

Vastbase 内汇聚所有命名空间中用户自定义普通表的状态信息。查询视图必须具有 monadmin 权限。

表 17-21. SUMMARY_STAT_USER_TABLES

名称	类型	描述
schemaname	name	此表的模式名。
relname	name	表名。
seq_scan	numeric	此表发起的顺序扫描数。
seq_tup_read	numeric	顺序扫描抓取的活跃行数。
idx_scan	numeric	此表发起的索引扫描数。
idx_tup_fetch	numeric	索引扫描抓取的活跃行数。
n_tup_ins	numeric	插入行数。
n_tup_upd	numeric	更新行数。
n_tup_del	numeric	删除行数。
n_tup_hot_upd	numeric	HOT 更新行数（即没有更新所需的单独索引）。
n_live_tup	numeric	估计活跃行数。
n_dead_tup	numeric	估计死行数。
last_vacuum	timestamp with time zone	最后一次此表是手动清理的（不计算 VACUUM FULL）时间。
last_autovacuum	timestamp with time zone	上次被 autovacuum 守护进程清理的时间。
last_analyze	timestamp with time zone	上次手动分析这个表的时间。

名称	类型	描述
last_autoanalyze	timestamp with time zone	上次被 autovacuum 守护进程分析的时间。
vacuum_count	numeric	这个表被手动清理的次数（不计算 VACUUM FULL）。
autovacuum_count	numeric	这个表被 autovacuum 清理的次数。
analyze_count	numeric	这个表被手动分析的次数。
autoanalyze_count	numeric	这个表被 autovacuum 守护进程分析的次数。

17.5.3. GLOBAL_STAT_USER_TABLES

得到各节点所有命名空间中用户自定义普通表的状态信息。查询视图必须具有 monadmin 权限。

表 17-22. GLOBAL_STAT_USER_TABLES 字段

名称	类型	描述
node_name	name	数据库进程名称。
relid	oid	表的 OID。
schemaname	name	此表的模式名。
relname	name	表名。
seq_scan	bigint	此表发起的顺序扫描数。
seq_tup_read	bigint	顺序扫描抓取的活跃行数。
idx_scan	bigint	此表发起的索引扫描数。
idx_tup_fetch	bigint	索引扫描抓取的活跃行数。
n_tup_ins	bigint	插入行数。
n_tup_upd	bigint	更新行数。
n_tup_del	bigint	删除行数。
n_tup_hot_upd	bigint	HOT 更新行数（即没有更新所需的单独索引）。
n_live_tup	bigint	估计活跃行数。

名称	类型	描述
n_dead_tup	bigint	估计死行数。
last_vacuum	timestamp with time zone	最后一次此表是手动清理的（不计算 VACUUM FULL）时间。
last_autovacuum	timestamp with time zone	上次被 autovacuum 守护进程清理的时间。
last_analyze	timestamp with time zone	上次手动分析这个表的时间。
last_autoanalyze	timestamp with time zone	上次被 autovacuum 守护进程分析的时间。
vacuum_count	bigint	这个表被手动清理的次数（不计算 VACUUM FULL）。
autovacuum_count	bigint	这个表被 autovacuum 清理的次数。
analyze_count	bigint	这个表被手动分析的次数。
autoanalyze_count	bigint	这个表被 autovacuum 守护进程分析的次数。

17.5.4. STAT_USER_INDEXES

显示数据库中用户自定义普通表的索引状态信息。

表 17-23. STAT_USER_INDEXES 字段

名称	类型	描述
relid	oid	此索引的表的 OID。
indexrelid	oid	索引的 OID。
schemaname	name	索引的模式名。
relname	name	索引的表名。
indexrelname	name	索引名。
idx_scan	bigint	索引上开始的索引扫描数。
idx_tup_read	bigint	通过索引上扫描返回的索引项数。
idx_tup_fetch	bigint	通过使用索引的简单索引扫描抓取的活表行数。

17.5.5. SUMMARY_STAT_USER_INDEXES

Vastbase 内汇聚所有数据库中用户自定义普通表的索引状态信息。查询视图必须具有 monadmin 权限。

表 17-24. SUMMARY_STAT_USER_INDEXES 字段

名称	类型	描述
schemaname	name	索引中模式名。
relname	name	索引的表名。
indexrelname	name	索引名。
idx_scan	numeric	索引上开始的索引扫描数。
idx_tup_read	numeric	通过索引上扫描返回的索引项数。
idx_tup_fetch	numeric	通过使用索引的简单索引扫描抓取的活表行数。

17.5.6. GLOBAL_STAT_USER_INDEXES

得到各节点数据库中用户自定义普通表的索引状态信息。查询视图必须具有 monadmin 权限。

表 17-25. GLOBAL_STAT_USER_INDEXES 字段

名称	类型	描述
node_name	name	数据库进程名称。
relid	oid	这个索引的表的 OID。
indexrelid	oid	索引的 OID。
schemaname	name	索引中模式名。
relname	name	索引的表名。
indexrelname	name	索引名。
idx_scan	bigint	索引上开始的索引扫描数。
idx_tup_read	bigint	通过索引上扫描返回的索引项数。
idx_tup_fetch	bigint	通过使用索引的简单索引扫描抓取的活表行数。

17.5.7. STAT_SYS_TABLES

显示 pg_catalog、information_schema 模式的所有命名空间中系统表的统计信息。

表 17-26. STAT_SYS_TABLES 字段

名称	类型	描述
relid	oid	表的 OID。
schemaname	name	该表的模式名。
relname	name	表名。
seq_scan	bigint	该表发起的顺序扫描数。
seq_tup_read	bigint	顺序扫描抓取的活跃行数。
idx_scan	bigint	该表发起的索引扫描数。
idx_tup_fetch	bigint	索引扫描抓取的活跃行数。
n_tup_ins	bigint	插入行数。
n_tup_upd	bigint	更新行数。
n_tup_del	bigint	删除行数。
n_tup_hot_upd	bigint	HOT 更新行数（比如没有更新所需的单独索引）。
n_live_tup	bigint	估计活跃行数。
n_dead_tup	bigint	估计死行数。
last_vacuum	timestamp with time zone	最后一次该表是手动清理的（不计算 VACUUM FULL）时间。
last_autovacuum	timestamp with time zone	上次被 autovacuum 守护进程清理的时间。
last_analyze	timestamp with time zone	上次手动分析该表的时间。
last_autoanalyze	timestamp with time zone	上次被 autovacuum 守护进程分析的时间。
vacuum_count	bigint	这个表被手动清理的次数（不计算 VACUUM FULL）。

名称	类型	描述
autovacuum_count	bigint	该表被 autovacuum 清理的次数。
analyze_count	bigint	该表被手动分析的次数。
autoanalyze_count	bigint	该表被 autovacuum 守护进程分析的次数。

17.5.8. SUMMARY_STAT_SYS_TABLES

Vastbase 内汇聚 pg_catalog、information_schema 模式的所有命名空间中系统表的统计信息。查询视图必须具有 monadmin 权限。

表 17-27. SUMMARY_STAT_SYS_TABLES 字段

名称	类型	描述
schemaname	name	此表的模式名。
relname	name	表名。
seq_scan	numeric	此表发起的顺序扫描数。
seq_tup_read	numeric	顺序扫描抓取的活跃行数。
idx_scan	numeric	此表发起的索引扫描数。
idx_tup_fetch	numeric	索引扫描抓取的活跃行数。
n_tup_ins	numeric	插入行数。
n_tup_upd	numeric	更新行数。
n_tup_del	numeric	删除行数。
n_tup_hot_upd	numeric	HOT 更新行数（比如没有更新所需的单独索引）。
n_live_tup	numeric	估计活跃行数。
n_dead_tup	numeric	估计死行数。
last_vacuum	timestamp with time zone	最后一次此表是手动清理的（不计算 VACUUM FULL）时间。
last_autovacuum	timestamp with time zone	上次被 autovacuum 守护进程清理的时间。

名称	类型	描述
last_analyze	timestamp with time zone	上次手动分析这个表的时间。
last_autoanalyze	timestamp with time zone	上次被 autovacuum 守护进程分析的时间。
vacuum_count	numeric	这个表被手动清理的次数（不计算 VACUUM FULL）。
autovacuum_count	numeric	这个表被 autovacuum 清理的次数。
analyze_count	numeric	这个表被手动分析的次数。
autoanalyze_count	numeric	这个表被 autovacuum 守护进程分析的次数。

17.5.9. GLOBAL_STAT_SYS_TABLES

得到各节点 pg_catalog、information_schema 模式的所有命名空间中系统表的统计信息。查询视图必须具有 monadmin 权限。

表 17-28. GLOBAL_STAT_SYS_TABLES 字段

名称	类型	描述
node_name	name	节点名称。
relid	oid	表的 OID。
schemaname	name	此表的模式名。
relname	name	表名。
seq_scan	bigint	此表发起的顺序扫描数。
seq_tup_read	bigint	顺序扫描抓取的活跃行数。
idx_scan	bigint	此表发起的索引扫描数。
idx_tup_fetch	bigint	索引扫描抓取的活跃行数。
n_tup_ins	bigint	插入行数。
n_tup_upd	bigint	更新行数。
n_tup_del	bigint	删除行数。

名称	类型	描述
n_tup_hot_upd	bigint	HOT 更新行数（比如没有更新所需的单独索引）。
n_live_tup	bigint	估计活跃行数。
n_dead_tup	bigint	估计死行数。
last_vacuum	timestamp with time zone	最后一次此表是手动清理的（不计算 VACUUM FULL）时间。
last_autovacuum	timestamp with time zone	上次被 autovacuum 守护进程清理的时间。
last_analyze	timestamp with time zone	上次手动分析这个表的时间。
last_autoanalyze	timestamp with time zone	上次被 autovacuum 守护进程分析的时间。
vacuum_count	bigint	这个表被手动清理的次数（不计算 VACUUM FULL）。
autovacuum_count	bigint	这个表被 autovacuum 清理的次数。
analyze_count	bigint	这个表被手动分析的次数。
autoanalyze_count	bigint	这个表被 autovacuum 守护进程分析的次数。

17.5.10.STAT_SYS_INDEXES

显示 pg_catalog、information_schema 以及 pg_toast 模式中所有系统表的索引状态信息。

表 17-29. STAT_SYS_INDEXES 字段

名称	类型	描述
relid	oid	此索引的表的 OID。
indexrelid	oid	索引的 OID。
schemaname	name	索引的模式名。
relname	name	索引的表名。
indexrelname	name	索引名。
idx_scan	bigint	索引上开始的索引扫描数。

名称	类型	描述
idx_tup_read	bigint	通过索引上扫描返回的索引项数。
idx_tup_fetch	bigint	通过使用索引的简单索引扫描抓取的活表行数。

17.5.11.SUMMARY_STAT_SYS_INDEXES

Vastbase 内汇聚 pg_catalog、information_schema 以及 pg_toast 模式中所有系统表的索引状态信息。查询视图必须具有 monadmin 权限。

表 17-30. SUMMARY_STAT_SYS_INDEXES 字段

名称	类型	描述
schemaname	name	索引中模式名。
relname	name	索引的表名。
indexrelname	name	索引名。
idx_scan	numeric	索引上开始的索引扫描数。
idx_tup_read	numeric	通过索引上扫描返回的索引项数。
idx_tup_fetch	numeric	通过使用索引的简单索引扫描抓取的活表行数。

17.5.12.GLOBAL_STAT_SYS_INDEXES

得到各节点 pg_catalog、information_schema 以及 pg_toast 模式中所有系统表的索引状态信息。查询视图必须具有 monadmin 权限。

表 17-31. GLOBAL_STAT_SYS_INDEXES 字段

名称	类型	描述
node_name	name	数据库进程名称
relid	oid	这个索引的表的 OID。
indexrelid	oid	索引的 OID。
schemaname	name	索引中模式名。

名称	类型	描述
relname	name	索引的表名。
indexrelname	name	索引名。
idx_scan	bigint	索引上开始的索引扫描数。
idx_tup_read	bigint	通过索引上扫描返回的索引项数。
idx_tup_fetch	bigint	通过使用索引的简单索引扫描抓取的活表行数。

17.5.13.STAT_ALL_TABLES

本节点内数据库中每个表（包括 TOAST 表）的一行的统计信息。

表 17-32. STAT_ALL_TABLES 字段

名称	类型	描述
relid	oid	表的 OID。
schemaname	name	该表的模式名。
relname	name	表名。
seq_scan	bigint	该表发起的顺序扫描数。
seq_tup_read	bigint	顺序扫描抓取的活跃行数。
idx_scan	bigint	该表发起的索引扫描数。
idx_tup_fetch	bigint	索引扫描抓取的活跃行数。
n_tup_ins	bigint	插入行数。
n_tup_upd	bigint	更新行数。
n_tup_del	bigint	删除行数。
n_tup_hot_upd	bigint	HOT 更新行数（比如没有更新所需的单独索引）。
n_live_tup	bigint	估计活跃行数。
n_dead_tup	bigint	估计死行数。
last_vacuum	timestamp with time zone	最后一次该表是手动清理的（不计算 VACUUM FULL）

名称	类型	描述
		的时间。
last_autovacuum	timestamp with time zone	上次被 autovacuum 守护进程清理的时间。
last_analyze	timestamp with time zone	上次手动分析该表的时间。
last_autoanalyze	timestamp with time zone	上次被 autovacuum 守护进程分析时间。
vacuum_count	bigint	该表被手动清理的次数（不计算 VACUUM FULL）。
autovacuum_count	bigint	该表被 autovacuum 清理的次数。
analyze_count	bigint	该表被手动分析的次数。
autoanalyze_count	bigint	该表被 autovacuum 守护进程分析的次数。

17.5.14.SUMMARY_STAT_ALL_TABLES

Vastbase 内汇聚数据库中每个表的一行(包括 TOAST 表)的统计信息。查询视图必须具有 monadmin 权限。

表 17-33. SUMMARY_STAT_ALL_TABLES 字段

名称	类型	描述
schemaname	name	此表的模式名。
relname	name	表名。
seq_scan	numeric	此表发起的顺序扫描数。
seq_tup_read	numeric	顺序扫描抓取的活跃行数。
idx_scan	numeric	此表发起的索引扫描数。
idx_tup_fetch	numeric	索引扫描抓取的活跃行数。
n_tup_ins	numeric	插入行数。
n_tup_upd	numeric	更新行数。
n_tup_del	numeric	删除行数。
n_tup_hot_upd	numeric	HOT 更新行数（比如没有更新所需的单独索引）。

名称	类型	描述
n_live_tup	numeric	估计活跃行数。
n_dead_tup	numeric	估计死行数。
last_vacuum	timestamp with time zone	最后一次此表是手动清理的（不计算 VACUUM FULL）的时间。
last_autovacuum	timestamp with time zone	上次被 autovacuum 守护进程清理的时间。
last_analyze	timestamp with time zone	上次手动分析这个表的时间。
last_autoanalyze	timestamp with time zone	上次被 autovacuum 守护进程分析时间。
vacuum_count	numeric	这个表被手动清理的次数（不计算 VACUUM FULL）。
autovacuum_count	numeric	这个表被 autovacuum 清理的次数。
analyze_count	numeric	这个表被手动分析的次数。
autoanalyze_count	numeric	这个表被 autovacuum 守护进程分析的次数。

17.5.15. GLOBAL_STAT_ALL_TABLES

得到各节点数据中每个表的一行 (包括 TOAST 表) 的统计信息。查询视图必须具有 monadmin 权限。

表 17-34. GLOBAL_STAT_ALL_TABLES 字段

名称	类型	描述
node_name	name	数据库进程名称。
relid	oid	表的 OID。
schemaname	name	此表的模式名。
relname	name	表名。
seq_scan	bigint	此表发起的顺序扫描数。
seq_tup_read	bigint	顺序扫描抓取的活跃行数。
idx_scan	bigint	此表发起的索引扫描数。

名称	类型	描述
idx_tup_fetch	bigint	索引扫描抓取的活跃行数。
n_tup_ins	bigint	插入行数。
n_tup_upd	bigint	更新行数。
n_tup_del	bigint	删除行数。
n_tup_hot_upd	bigint	HOT 更新行数（比如没有更新所需的单独索引）。
n_live_tup	bigint	估计活跃行数。
n_dead_tup	bigint	估计死行数。
last_vacuum	timestamp with time zone	最后一次此表是手动清理的（不计算 VACUUM FULL）的时间。
last_autovacuum	timestamp with time zone	上次被 autovacuum 守护进程清理的时间。
last_analyze	timestamp with time zone	上次手动分析这个表的时间。
last_autoanalyze	timestamp with time zone	上次被 autovacuum 守护进程分析时间。
vacuum_count	bigint	这个表被手动清理的次数（不计算 VACUUM FULL）。
autovacuum_count	bigint	这个表被 autovacuum 清理的次数。
analyze_count	bigint	这个表被手动分析的次数。
autoanalyze_count	bigint	这个表被 autovacuum 守护进程分析的次数。

17.5.16.STAT_ALL_INDEXES

将包含本节点数据库中的每个索引行，显示访问特定索引的统计。

表 17-35. STAT_ALL_INDEXES 字段

名称	类型	描述
relid	oid	这个索引的表的 OID。
indexrelid	oid	索引的 OID。
schemaname	name	索引中模式名。

名称	类型	描述
relname	name	索引的表名。
indexrelname	name	索引名。
idx_scan	bigint	索引上开始的索引扫描数。
idx_tup_read	bigint	通过索引上扫描返回的索引项数。
idx_tup_fetch	bigint	通过使用索引的简单索引扫描抓取的活表行数。

17.5.17.SUMMARY_STAT_ALL_INDEXES

将包含 Vastbase 内汇聚数据库中的每个索引行，显示访问特定索引的统计。查询视图必须具有 monadmin 权限。

表 17-36. SUMMARY_STAT_ALL_INDEXES 字段

名称	类型	描述
schemaname	name	索引中模式名。
relname	name	索引的表名。
indexrelname	name	索引名。
idx_scan	numeric	索引上开始的索引扫描数。
idx_tup_read	numeric	通过索引上扫描返回的索引项数。
idx_tup_fetch	numeric	通过使用索引的简单索引扫描抓取的活表行数。

17.5.18.GLOBAL_STAT_ALL_INDEXES

将包含各节点数据库中的每个索引行，显示访问特定索引的统计。查询视图必须具有 monadmin 权限。

表 17-37. GLOBAL_STAT_ALL_INDEXES 字段

名称	类型	描述
node_name	name	数据库进程名称。

名称	类型	描述
relid	oid	这个索引的表的 OID。
indexrelid	oid	索引的 OID。
schemaname	name	索引中模式名。
relname	name	索引的表名。
indexrelname	name	索引名。
idx_scan	bigint	索引上开始的索引扫描数。
idx_tup_read	bigint	通过索引上扫描返回的索引项数。
idx_tup_fetch	bigint	通过使用索引的简单索引扫描抓取的活表行数。

17.5.19.STAT_DATABASE

视图将包含本节点中每个数据库的统计信息。

表 17-38. STAT_DATABASE 字段

名称	类型	描述
datid	oid	数据库的 OID。
datname	name	此数据库的名称。
numbackends	integer	当前连接到该数据库的后端数。这是在返回一个反映目前状态值的视图中唯一的列；自上次重置所有其他列返回累积值。
xact_commit	bigint	此数据库中已经提交的事务数。
xact_rollback	bigint	此数据库中已经回滚的事务数。
blks_read	bigint	在这个数据库中读取的磁盘块的数量。
blks_hit	bigint	高速缓存中已经发现的磁盘块的次数，这样读取是不必要的（这只包括 PostgreSQL 缓冲区高速缓存，没有操作系统的文件系统缓存）。
tup_returned	bigint	通过数据库查询返回的行数。

名称	类型	描述
tup_fetched	bigint	通过数据库查询抓取的行数。
tup_inserted	bigint	通过数据库查询插入的行数。
tup_updated	bigint	通过数据库查询更新的行数。
tup_deleted	bigint	通过数据库查询删除的行数。
conflicts	bigint	由于数据库恢复冲突取消的查询数量（只在备用服务器发生的冲突）。请参见 17.5.22STAT_DATABASE_CONFLICTS 获取更多信息。
temp_files	bigint	通过数据库查询创建的临时文件数量。计算所有临时文件，不论为什么创建临时文件（比如排序或者哈希），而且不管 log_temp_files 设置。
temp_bytes	bigint	通过数据库查询写入临时文件的数据总量。计算所有临时文件，不论为什么创建临时文件，而且不管 log_temp_files 设置。
deadlocks	bigint	在该数据库中检索的死锁数。
blk_read_time	double precision	通过数据库后端读取数据文件块花费的时间，以毫秒计算。
blk_write_time	double precision	通过数据库后端写入数据文件块花费的时间，以毫秒计算。
stats_reset	timestamp with time zone	重置当前状态统计的时间。

17.5.20.SUMMARY_STAT_DATABASE

视图将包含集群内汇聚的每个数据库的每一行，显示数据库统计。查询视图必须具有 monadmin 权限。

表 17-39. SUMMARY_STAT_DATABASE

名称	类型	描述
datname	name	这个数据库的名称。
numbackends	bigint	当前连接到该数据库的后端数。这是在返回一个反映目前状态值的视图中唯一的列；自上次重置所有其他列返

名称	类型	描述
		回累积值。
xact_commit	numeric	此数据库中已经提交的事务数。
xact_rollback	numeric	此数据库中已经回滚的事务数。
blks_read	numeric	在这个数据库中读取的磁盘块的数量。
blks_hit	numeric	高速缓存中已经发现的磁盘块的次数，这样读取是不必要的（这只包括 PostgreSQL 缓冲区高速缓存，没有操作系统的文件系统缓存）。
tup_returned	numeric	通过数据库查询返回的行数。
tup_fetched	numeric	通过数据库查询抓取的行数。
tup_inserted	bigint	通过数据库查询插入的行数。
tup_updated	bigint	通过数据库查询更新的行数。
tup_deleted	bigint	通过数据库查询删除的行数。
conflicts	bigint	由于数据库恢复冲突取消的查询数量（只在备用服务器发生的冲突）。请参见 17.5.22STAT_DATABASE_CONFLICTS 获取更多信息。
temp_files	numeric	通过数据库查询创建的临时文件数量。计算所有临时文件，不论为什么创建临时文件（比如排序或者哈希），而且不管 log_temp_files 设置。
temp_bytes	numeric	通过数据库查询写入临时文件的数据总量。计算所有临时文件，不论为什么创建临时文件，而且不管 log_temp_files 设置。
deadlocks	bigint	在该数据库中检索的死锁数。
blk_read_time	double precision	通过数据库后端读取数据文件块花费的时间，以毫秒计算。
blk_write_time	double precision	通过数据库后端写入数据文件块花费的时间，以毫秒计算。
stats_reset	timestamp with time zone	重置当前状态统计的时间。

17.5.21.GLOBAL_STAT_DATABASE

视图将包含 Vastbase 中各节点的每个数据库的每一行，显示数据库统计。查询视图必须具有 monadmin 权限。

表 17-40. GLOBAL_STAT_DATABASE 字段

名称	类型	描述
node_name	name	数据库进程名称。
datid	oid	数据库的 OID。
datname	name	这个数据库的名称。
numbackends	integer	当前连接到该数据库的后端数。这是在返回一个反映目前状态值的视图中唯一的列；自上次重置所有其他列返回累积值。
xact_commit	bigint	此数据库中已经提交的事务数。
xact_rollback	bigint	此数据库中已经回滚的事务数。
blks_read	bigint	在这个数据库中读取的磁盘块的数量。
blks_hit	bigint	高速缓存中已经发现的磁盘块的次数，这样读取是不必要的（这只包括数据库内核缓冲区高速缓存，没有操作系统的文件系统缓存）。
tup_returned	bigint	通过数据库查询返回的行数。
tup_fetched	bigint	通过数据库查询抓取的行数。
tup_inserted	bigint	通过数据库查询插入的行数。
tup_updated	bigint	通过数据库查询更新的行数。
tup_deleted	bigint	通过数据库查询删除的行数。
conflicts	bigint	由于数据库恢复冲突取消的查询数量（只在备用服务器发生的冲突）。请参见 17.5.22STAT_DATABASE_CONFLICTS 获取更多信息。
temp_files	bigint	通过数据库查询创建的临时文件数量。计算所有临时文件，不论为什么创建临时文件（比如排序或者哈希），

名称	类型	描述
		而且不管 log_temp_files 设置。
temp_bytes	bigint	通过数据库查询写入临时文件的数据总量。计算所有临时文件，不论为什么创建临时文件，而且不管 log_temp_files 设置。
deadlocks	bigint	在该数据库中检索的死锁数。
blk_read_time	double precision	通过数据库后端读取数据文件块花费的时间，以毫秒计算。
blk_write_time	double precision	通过数据库后端写入数据文件块花费的时间，以毫秒计算。
stats_reset	timestamp with time zone	重置当前状态统计的时间。

17.5.22.STAT_DATABASE_CONFLICTS

显示当前节点数据库冲突状态的统计信息。

表 17-41. STAT_DATABASE_CONFLICTS 字段

名称	类型	描述
datid	oid	数据库标识。
datname	name	数据库名称。
confl_tablespace	bigint	冲突的表空间的数目。
confl_lock	bigint	冲突的锁数目。
confl_snapshot	bigint	冲突的快照数目。
confl_bufferpin	bigint	冲突的缓冲区数目。
confl_deadlock	bigint	冲突的死锁数目。

17.5.23.SUMMARY_STAT_DATABASE_CONFLICTS

显示 Vastbase 内汇聚的数据库冲突状态的统计信息。查询视图必须具有 monadmin 权限。

表 17-42. SUMMARY_STAT_DATABASE_CONFLICTS 字段

名称	类型	描述
datname	name	数据库名称。
confl_tablespace	bigint	冲突的表空间的数目。
confl_lock	bigint	冲突的锁数目。
confl_snapshot	bigint	冲突的快照数目。
confl_bufferpin	bigint	冲突的缓冲区数目。
confl_deadlock	bigint	冲突的死锁数目。

17.5.24. GLOBAL_STAT_DATABASE_CONFLICTS

显示每个节点的数据库冲突状态的统计信息。查询视图必须具有 monadmin 权限。

表 17-43. GLOBAL_STAT_DATABASE_CONFLICTS 字段

名称	类型	描述
node_name	name	数据库进程名称
datid	oid	数据库标识。
datname	name	数据库名称。
confl_tablespace	bigint	冲突的表空间的数目。
confl_lock	bigint	冲突的锁数目。
confl_snapshot	bigint	冲突的快照数目。
confl_bufferpin	bigint	冲突的缓冲区数目。
confl_deadlock	bigint	冲突的死锁数目。

17.5.25. STAT_XACT_ALL_TABLES

显示命名空间中所有普通表和 toast 表的事务状态信息。

表 17-44. STAT_XACT_ALL_TABLES 字段

名称	类型	描述
relid	oid	表的 OID。
schemaname	name	该表的模式名。
relname	name	表名。
seq_scan	bigint	该表发起的顺序扫描数。
seq_tup_read	bigint	顺序扫描抓取的活跃行数。
idx_scan	bigint	该表发起的索引扫描数。
idx_tup_fetch	bigint	索引扫描抓取的活跃行数。
n_tup_ins	bigint	插入行数。
n_tup_upd	bigint	更新行数。
n_tup_del	bigint	删除行数。
n_tup_hot_upd	bigint	HOT 更新行数（比如没有更新所需的单独索引）。

17.5.26. SUMMARY_STAT_XACT_ALL_TABLES

显示 Vastbase 内汇聚的命名空间中所有普通表和 toast 表的事务状态信息。查询视图必须具有 monadmin 权限。

表 17-45. SUMMARY_STAT_XACT_ALL_TABLES 字段

名称	类型	描述
schemaname	name	此表的模式名。
relname	name	表名。
seq_scan	numeric	此表发起的顺序扫描数。
seq_tup_read	numeric	顺序扫描抓取的活跃行数。
idx_scan	numeric	此表发起的索引扫描数。
idx_tup_fetch	numeric	索引扫描抓取的活跃行数。

名称	类型	描述
n_tup_ins	numeric	插入行数。
n_tup_upd	numeric	更新行数。
n_tup_del	numeric	删除行数。
n_tup_hot_upd	numeric	HOT 更新行数（比如没有更新所需的单独索引）。

17.5.27. GLOBAL_STAT_XACT_ALL_TABLES

显示各节点的命名空间中所有普通表和 toast 表的事务状态信息。查询视图必须具有 monadmin 权限。

表 17-46. GLOBAL_STAT_XACT_ALL_TABLES 字段

名称	类型	描述
node_name	name	数据库进程名称。
relid	oid	表的 OID。
schemaname	name	此表的模式名。
relname	name	表名。
seq_scan	bigint	此表发起的顺序扫描数。
seq_tup_read	bigint	顺序扫描抓取的活跃行数。
idx_scan	bigint	此表发起的索引扫描数。
idx_tup_fetch	bigint	索引扫描抓取的活跃行数。
n_tup_ins	bigint	插入行数。
n_tup_upd	bigint	更新行数。
n_tup_del	bigint	删除行数。
n_tup_hot_upd	bigint	HOT 更新行数（比如没有更新所需的单独索引）。

17.5.28.STAT_XACT_SYS_TABLES

显示当前节点命名空间中系统表的事务状态信息。

表 17-47. STAT_XACT_SYS_TABLES 字段

名称	类型	描述
relid	oid	表的 OID。
schemaname	name	该表的模式名。
relname	name	表名。
seq_scan	bigint	该表发起的顺序扫描数。
seq_tup_read	bigint	顺序扫描抓取的活跃行数。
idx_scan	bigint	该表发起的索引扫描数。
idx_tup_fetch	bigint	索引扫描抓取的活跃行数。
n_tup_ins	bigint	插入行数。
n_tup_upd	bigint	更新行数。
n_tup_del	bigint	删除行数。
n_tup_hot_upd	bigint	HOT 更新行数（比如没有更新所需的单独索引）。

17.5.29.SUMMARY_STAT_XACT_SYS_TABLES

显示 Vastbase 内汇聚的命名空间中系统表的事务状态信息。查询视图必须具有 monadmin 权限。

表 17-48. SUMMARY_STAT_XACT_SYS_TABLES 字段

名称	类型	描述
schemaname	name	此表的模式名。
relname	name	表名。
seq_scan	numeric	此表发起的顺序扫描数。
seq_tup_read	numeric	顺序扫描抓取的活跃行数。

名称	类型	描述
idx_scan	numeric	此表发起的索引扫描数。
idx_tup_fetch	numeric	索引扫描抓取的活跃行数。
n_tup_ins	numeric	插入行数。
n_tup_upd	numeric	更新行数。
n_tup_del	numeric	删除行数。
n_tup_hot_upd	numeric	HOT 更新行数（比如没有更新所需的单独索引）。

17.5.30. GLOBAL_STAT_XACT_SYS_TABLES

显示各节点命名空间中系统表的事务状态信息。查询视图必须具有 monadmin 权限。

表 17-49. GLOBAL_STAT_XACT_SYS_TABLES 字段

名称	类型	描述
node_name	name	节点名称。
relid	oid	表的 OID。
schemaname	name	此表的模式名。
relname	name	表名。
seq_scan	bigint	此表发起的顺序扫描数。
seq_tup_read	bigint	顺序扫描抓取的活跃行数。
idx_scan	bigint	此表发起的索引扫描数。
idx_tup_fetch	bigint	索引扫描抓取的活跃行数。
n_tup_ins	bigint	插入行数。
n_tup_upd	bigint	更新行数。
n_tup_del	bigint	删除行数。
n_tup_hot_upd	bigint	HOT 更新行数（比如没有更新所需的单独索引）。

17.5.31.STAT_XACT_USER_TABLES

显示当前节点命名空间中用户表的事务状态信息。

表 17-50. STAT_XACT_USER_TABLES 字段

名称	类型	描述
relid	oid	表的 OID。
schemaname	name	该表的模式名。
relname	name	表名。
seq_scan	bigint	该表发起的顺序扫描数。
seq_tup_read	bigint	顺序扫描抓取的活跃行数。
idx_scan	bigint	该表发起的索引扫描数。
idx_tup_fetch	bigint	索引扫描抓取的活跃行数。
n_tup_ins	bigint	插入行数。
n_tup_upd	bigint	更新行数。
n_tup_del	bigint	删除行数。
n_tup_hot_upd	bigint	HOT 更新行数（比如没有更新所需的单独索引）。

17.5.32.SUMMARY_STAT_XACT_USER_TABLES

显示集群内汇聚的命名空间中用户表的事务状态信息。查询视图必须具有 monadmin 权限。

表 17-51. SUMMARY_STAT_XACT_USER_TABLES 字段

名称	类型	描述
schemaname	name	此表的模式名。
relname	name	表名。
seq_scan	numeric	此表发起的顺序扫描数。
seq_tup_read	numeric	顺序扫描抓取的活跃行数。

名称	类型	描述
idx_scan	numeric	此表发起的索引扫描数。
idx_tup_fetch	numeric	索引扫描抓取的活跃行数。
n_tup_ins	numeric	插入行数。
n_tup_upd	numeric	更新行数。
n_tup_del	numeric	删除行数。
n_tup_hot_upd	numeric	HOT 更新行数（比如没有更新所需的单独索引）。

17.5.33. GLOBAL_STAT_XACT_USER_TABLES

显示各节点命名空间中用户表的事务状态信息。查询视图必须具有 monadmin 权限。

表 17-52. GLOBAL_STAT_XACT_USER_TABLES 字段

名称	类型	描述
node_name	name	数据库进程名称。
relid	oid	表的 OID。
schemaname	name	此表的模式名。
relname	name	表名。
seq_scan	bigint	此表发起的顺序扫描数。
seq_tup_read	bigint	顺序扫描抓取的活跃行数。
idx_scan	bigint	此表发起的索引扫描数。
idx_tup_fetch	bigint	索引扫描抓取的活跃行数。
n_tup_ins	bigint	插入行数。
n_tup_upd	bigint	更新行数。
n_tup_del	bigint	删除行数。
n_tup_hot_upd	bigint	HOT 更新行数（比如没有更新所需的单独索引）。

17.5.34. STAT_XACT_USER_FUNCTIONS

视图包含当前节点本事务内函数执行的统信息。

表 17-53. STAT_XACT_USER_FUNCTIONS 字段

名称	类型	描述
funcid	oid	函数标识。
schemaname	name	模式的名称。
funcname	name	函数名称。
calls	bigint	函数被调用的次数。
total_time	double precision	函数的总执行时长。
self_time	double precision	当前线程调用函数的总的时长。

17.5.35. SUMMARY_STAT_XACT_USER_FUNCTIONS

视图包含 Vastbase 内汇聚的本事务内函数执行的统信息。查询视图必须具有 monadmin 权限。

表 17-54. SUMMARY_STAT_XACT_USER_FUNCTIONS 字段

名称	类型	描述
schemaname	name	模式的名称。
funcname	name	函数名称。
calls	numeric	函数被调用的次数。
total_time	double precision	函数的总执行时长。
self_time	double precision	当前线程调用函数的总的时长。

17.5.36. GLOBAL_STAT_XACT_USER_FUNCTIONS

视图包含各节点本事务内函数执行的统信息。查询视图必须具有 monadmin 权限。

表 17-55. GLOBAL_STAT_XACT_USER_FUNCTIONS 字段

名称	类型	描述
node_name	name	节点名称。
funcid	oid	函数标识。
schemaname	name	模式的名称。
funcname	name	函数名称。
calls	bigint	函数被调用的次数。
total_time	double precision	此函数及其调用的所有其他函数所花费的总时间。
self_time	double precision	在此函数本身中花费的总时间（不包括它调用的其他函数）。

17.5.37.STAT_BAD_BLOCK

获得当前节点表、索引等文件的读取失败信息。

表 17-56. STAT_BAD_BLOCK 字段

名称	类型	描述
nodename	text	数据库进程名称。
databaseid	integer	database 的 oid。
tablespaceid	integer	tablespace 的 oid。
relfilenode	integer	relation 的 file node。
bucketid	smallint	一致性 hash bucket ID。
forknum	integer	fork 编号。
error_count	integer	error 的数量。
first_time	timestamp with time zone	坏块第一次出现的时间。
last_time	timestamp with time zone	坏块最后出现的时间。

17.5.38.SUMMARY_STAT_BAD_BLOCK

获得 Vastbase 内汇聚的表、索引等文件的读取失败信息。查询视图必须具有 monadmin 权限。

表 17-57. SUMMARY_STAT_BAD_BLOCK 字段

名称	类型	描述
databaseid	integer	database 的 oid。
tablespaceid	integer	tablespace 的 oid。
relfilenode	integer	relation 的 file node。
forknum	bigint	fork 编号。
error_count	bigint	error 的数量。
first_time	timestamp with time zone	坏块第一次出现的时间。
last_time	timestamp with time zone	坏块最后出现的时间。

17.5.39.GLOBAL_STAT_BAD_BLOCK

获得各节点的表、索引等文件的读取失败信息。查询视图必须具有 monadmin 权限。

表 17-58. GLOBAL_STAT_BAD_BLOCK 字段

名称	类型	描述
node_name	text	数据库进程名称。
databaseid	integer	database 的 oid。
tablespaceid	integer	tablespace 的 oid。
relfilenode	integer	relation 的 file node。
forknum	integer	fork 编号。
error_count	integer	error 的数量。
first_time	timestamp with time zone	坏块第一次出现的时间。
last_time	timestamp with time zone	坏块最后出现的时间。

17.5.40.STAT_USER_FUNCTIONS

STAT_USER_FUNCTIONS 视图显示命名空间中用户自定义函数（函数语言为非内部语言）的状态信息。

表 17-59. STAT_USER_FUNCTIONS 字段

名称	类型	描述
funcid	oid	函数标识。
schemaname	name	schema 的名称。
funcname	name	用户自定义函数的名称。
calls	bigint	函数被调用的次数。
total_time	double precision	调用此函数花费的总时间，包含调用其它函数的时间(单位：毫秒)。
self_time	double precision	调用此函数自己花费的时间，不包含调用其它函数的时间(单位：毫秒)。

17.5.41.SUMMARY_STAT_USER_FUNCTIONS

SUMMARY_STAT_USER_FUNCTIONS 用来统计所数据库节点用户自定义视图的相关统计信息，查询视图必须具有 monadmin 权限。

表 17-60. SUMMARY_STAT_USER_FUNCTIONS 字段

名称	类型	描述
schemaname	name	schema 的名称。
funcname	name	用户 function 的名称。
calls	numeric	总调用次数。
total_time	double precision	调用此 function 的总时间花费，包含调用其它 function 的时间(单位：毫秒)。
self_time	double precision	调用此 function 自己时间的花费，不包含调用其它 function 的时间(单位：毫秒)。

17.5.42. GLOBAL_STAT_USER_FUNCTIONS

提供 Vastbase 中各个节点的用户所创建的函数的状态的统计信息，查询视图必须具有 monadmin 权限。

表 17-61. GLOBAL_STAT_USER_FUNCTIONS 字段

名称	类型	描述
node_name	name	数据库进程名称。
funcid	oid	函数的 id。
schemaname	name	此函数所在模式的名称。
funcname	name	函数名称。
calls	bigint	该函数被调用的次数。
total_time	double precision	此函数及其调用的所有其他函数所花费的总时间（以毫秒为单位）。
self_time	double precision	在此函数本身中花费的总时间（不包括它调用的其他函数），以毫秒为单位。

17.6. Workload

17.6.1. WORKLOAD_SQL_COUNT

显示当前节点 workload 上的 SQL 数量分布。普通用户只可以看到自己在 workload 上的 SQL 分布；monadmin 可以看到总的 workload 的负载情况。

表 17-62. WORKLOAD_SQL_COUNT 字段

名称	类型	描述
workload	name	负载名称。
select_count	bigint	select 数量。
update_count	bigint	update 数量。
insert_count	bigint	insert 数量。

名称	类型	描述
delete_count	bigint	delete 数量。
ddl_count	bigint	ddl 数量。
dml_count	bigint	dml 数量。
dcl_count	bigint	dcl 数量。

17.6.2. SUMMARY_WORKLOAD_SQL_COUNT

显示 Vastbase 内各数据库主节点的 workload 上的 SQL 数量分布。查询视图必须具有 monadmin 权限。

表 17-63. SUMMARY_WORKLOAD_SQL_COUNT 字段

名称	类型	描述
node_name	name	数据库进程名称。
workload	name	负载名称。
select_count	bigint	select 数量。
update_count	bigint	update 数量。
insert_count	bigint	insert 数量。
delete_count	bigint	delete 数量。
ddl_count	bigint	ddl 数量。
dml_count	bigint	dml 数量。
dcl_count	bigint	dcl 数量。

17.6.3. WORKLOAD_TRANSACTION

当前节点上负载的事务信息。

表 17-64. WORKLOAD_TRANSACTION 字段

名称	类型	描述

名称	类型	描述
workload	name	负载的名称。
commit_counter	bigint	用户事务 commit 数量。
rollback_counter	bigint	用户事务 rollback 数量。
resp_min	bigint	用户事务最小响应时间（单位：微秒）。
resp_max	bigint	用户事务最大响应时间（单位：微秒）。
resp_avg	bigint	用户事务平均响应时间（单位：微秒）。
resp_total	bigint	用户事务总响应时间（单位：微秒）。
bg_commit_counter	bigint	后台事务 commit 数量。
bg_rollback_counter	bigint	后台事务 rollback 数量。
bg_resp_min	bigint	后台事务最小响应时间（单位：微秒）。
bg_resp_max	bigint	后台事务最大响应时间（单位：微秒）。
bg_resp_avg	bigint	后台事务平均响应时间（单位：微秒）。
bg_resp_total	bigint	后台事务总响应时间（单位：微秒）。

17.6.4. SUMMARY_WORKLOAD_TRANSACTION

显示 Vastbase 内汇聚的负载事务信息。查询视图必须具有 monadmin 权限。

表 17-65. SUMMARY_WORKLOAD_TRANSACTION 字段

名称	类型	描述
workload	name	负载的名称。
commit_counter	numeric	用户事务 commit 数量。
rollback_counter	numeric	用户事务 rollback 数量。
resp_min	bigint	用户事务最小响应时间（单位：微秒）。
resp_max	bigint	用户事务最大响应时间（单位：微秒）。
resp_avg	bigint	用户事务平均响应时间（单位：微秒）。

名称	类型	描述
resp_total	numeric	用户事务总响应时间（单位：微秒）。
bg_commit_counter	numeric	后台事务 commit 数量。
bg_rollback_counter	numeric	后台事务 rollback 数量。
bg_resp_min	bigint	后台事务最小响应时间（单位：微秒）。
bg_resp_max	bigint	后台事务最大响应时间（单位：微秒）。
bg_resp_avg	bigint	后台事务平均响应时间（单位：微秒）。
bg_resp_total	numeric	后台事务总响应时间（单位：微秒）。

17.6.5. GLOBAL_WORKLOAD_TRANSACTION

显示各节点上的 workload 的负载信息。查询视图必须具有 monadmin 权限。

表 17-66. GLOBAL_WORKLOAD_TRANSACTION 字段

名称	类型	描述
node_name	name	数据库进程名称。
workload	name	负载的名称。
commit_counter	bigint	用户事务 commit 数量。
rollback_counter	bigint	用户事务 rollback 数量。
resp_min	bigint	用户事务最小响应时间（单位：微秒）。
resp_max	bigint	用户事务最大响应时间（单位：微秒）。
resp_avg	bigint	用户事务平均响应时间（单位：微秒）。
resp_total	bigint	用户事务总响应时间（单位：微秒）。
bg_commit_counter	bigint	后台事务 commit 数量。
bg_rollback_counter	bigint	后台事务 rollback 数量。
bg_resp_min	bigint	后台事务最小响应时间（单位：微秒）。
bg_resp_max	bigint	后台事务最大响应时间（单位：微秒）。

名称	类型	描述
bg_resp_avg	bigint	后台事务平均响应时间（单位：微秒）。
bg_resp_total	bigint	后台事务总响应时间（单位：微秒）。

17.6.6. WORKLOAD_SQL_ELAPSE_TIME

WORKLOAD_SQL_ELAPSE_TIME 用来统计 workload（业务负载）上的 SUID 信息。

表 17-67. WORKLOAD_SQL_ELAPSE_TIME 字段

名称	类型	描述
workload	name	workload（业务负载）名称。
total_select_elapse	bigint	总 select 的时间花费（单位：微秒）。
max_select_elapse	bigint	最大 select 的时间花费（单位：微秒）。
min_select_elapse	bigint	最小 select 的时间花费（单位：微秒）。
avg_select_elapse	bigint	平均 select 的时间花费（单位：微秒）。
total_update_elapse	bigint	总 update 的时间花费（单位：微秒）。
max_update_elapse	bigint	最大 update 的时间花费（单位：微秒）。
min_update_elapse	bigint	最小 update 的时间花费（单位：微秒）。
avg_update_elapse	bigint	平均 update 的时间花费（单位：微秒）。
total_insert_elapse	bigint	总 insert 的时间花费（单位：微秒）。
max_insert_elapse	bigint	最大 insert 的时间花费（单位：微秒）。
min_insert_elapse	bigint	最小 insert 的时间花费（单位：微秒）。
avg_insert_elapse	bigint	平均 insert 的时间花费（单位：微秒）。
total_delete_elapse	bigint	总 delete 的时间花费（单位：微秒）。
max_delete_elapse	bigint	最大 delete 的时间花费（单位：微秒）。
min_delete_elapse	bigint	最小 delete 的时间花费（单位：微秒）。
avg_delete_elapse	bigint	平均 delete 的时间花费（单位：微秒）。

17.6.7. SUMMARY_WORKLOAD_SQL_ELAPSE_TIME

SUMMARY_WORKLOAD_SQL_ELAPSE_TIME 用来统计数据库主节点上 workload（业务）负载的 SUID 信息，查询视图必须具有 monadmin 权限。

表 17-68. SUMMARY_WORKLOAD_SQL_ELAPSE_TIM 字段

名称	类型	描述
node_name	name	数据库进程名称。
workload	name	workload（业务负载）名称。
total_select_elapse	bigint	总 select 的时间花费（单位：微秒）。
max_select_elapse	bigint	最大 select 的时间花费（单位：微秒）。
min_select_elapse	bigint	最小 select 的时间花费（单位：微秒）。
avg_select_elapse	bigint	平均 select 的时间花费（单位：微秒）。
total_update_elapse	bigint	总 update 的时间花费（单位：微秒）。
max_update_elapse	bigint	最大 update 的时间花费（单位：微秒）。
min_update_elapse	bigint	最小 update 的时间花费（单位：微秒）。
avg_update_elapse	bigint	平均 update 的时间花费（单位：微秒）。
total_insert_elapse	bigint	总 insert 的时间花费（单位：微秒）。
max_insert_elapse	bigint	最大 insert 的时间花费（单位：微秒）。
min_insert_elapse	bigint	最小 insert 的时间花费（单位：微秒）。
avg_insert_elapse	bigint	平均 insert 的时间花费（单位：微秒）。
total_delete_elapse	bigint	总 delete 的时间花费（单位：微秒）。
max_delete_elapse	bigint	最大 delete 的时间花费（单位：微秒）。
min_delete_elapse	bigint	最小 delete 的时间花费（单位：微秒）。
avg_delete_elapse	bigint	平均 delete 的时间花费（单位：微秒）。

17.7. Session/Thread

17.7.1. SESSION_STAT

当前节点以会话线程或 AutoVacuum 线程为单位，统计会话状态信息。

表 17-69. SESSION_STAT 字段

名称	类型	描述
sessid	text	线程启动时间+线程标识。
statid	integer	统计编号。
statname	text	统计会话名称。
statunit	text	统计会话单位。
value	bigint	统计会话值。

17.7.2. GLOBAL_SESSION_STAT

各节点上以会话线程或 AutoVacuum 线程为单位，统计会话状态信息。查询视图必须具有 monadmin 权限。

表 17-70. GLOBAL_SESSION_STAT 字段

名称	类型	描述
node_name	name	数据库进程名称。
sessid	text	线程启动时间+线程标识。
statid	integer	统计编号。
statname	text	统计会话名称。
statunit	text	统计会话单位。
value	bigint	统计会话值。

17.7.3. SESSION_TIME

用于统计当前节点会话线程的运行时间信息，及各执行阶段所消耗时间。

表 17-71. SESSION_TIME 字段

名称	类型	描述
sessid	text	线程启动时间+线程标识。
stat_id	integer	统计编号。
stat_name	text	会话类型名称。
value	bigint	会话值。

17.7.4. GLOBAL_SESSION_TIME

用于统计各节点会话线程的运行时间信息，及各执行阶段所消耗时间。查询视图必须具有 monadmin 权限。

表 17-72. GLOBAL_SESSION_TIME 字段

名称	类型	描述
node_name	name	数据库进程名称。
sessid	text	线程启动时间+线程标识。
stat_id	integer	统计编号。
stat_name	text	会话类型名称。
value	bigint	会话值。

17.7.5. SESSION_MEMORY

统计 Session 级别的内存使用情况，包含执行作业在数据节点上 Postgres 线程和 Stream 线程分配的所有内存，单位为 MB。

表 17-73. SESSION_MEMORY 字段

名称	类型	描述

名称	类型	描述
sessid	text	线程启动时间+线程标识。
init_mem	integer	当前正在执行作业进入执行器前已分配的内存。
used_mem	integer	当前正在执行作业已分配的内存。
peak_mem	integer	当前正在执行作业已分配的内存峰值。

17.7.6. GLOBAL_SESSION_MEMORY

统计各节点的 Session 级别的内存使用情况，包含执行作业在数据节点上 vastbase 线程和 Stream 线程分配的所有内存，单位为 MB。查询视图必须具有 monadmin 权限。

表 17-74. GLOBAL_SESSION_MEMORY 字段

名称	类型	描述
node_name	name	数据库进程名称。
sessid	text	线程启动时间+线程标识。
init_mem	integer	当前正在执行作业进入执行器前已分配的内存。
used_mem	integer	当前正在执行作业已分配的内存。
peak_mem	integer	当前正在执行作业已分配的内存峰值。

17.7.7. SESSION_MEMORY_DETAIL

统计线程的内存使用情况，以 MemoryContext 节点来统计。

表 17-75. SESSION_MEMORY_DETAIL 字段

名称	类型	描述
sessid	text	线程启动时间+线程标识。
sesstype	text	线程名称。
contextname	text	内存上下文名称。
level	smallint	内存上下文的重要级别。

名称	类型	描述
parent	text	父级内存上下文名称。
totalsize	bigint	总申请内存大小(单位：字节)。
freesize	bigint	空闲内存大小(单位：字节)。
usedsize	bigint	使用内存大小(单位：字节)。

17.7.8. GLOBAL_SESSION_MEMORY_DETAIL

统计各节点的线程的内存使用情况，以 MemoryContext 节点来统计。查询视图必须具有 monadmin 权限。

表 17-76. GLOBAL_SESSION_MEMORY_DETAIL 字段

名称	类型	描述
node_name	name	数据库进程名称。
sessid	text	线程启动时间+线程标识。
sesstype	text	线程名称。
contextname	text	内存上下文名称。
level	smallint	内存上下文的重要级别。
parent	text	父级内存上下文名称。
totalsize	bigint	总申请内存大小(单位：字节)。
freesize	bigint	空闲内存大小(单位：字节)。
usedsize	bigint	使用内存大小(单位：字节)。

17.7.9. SESSION_STAT_ACTIVITY

显示当前节点上正在运行的线程相关的信息。

表 17-77. SESSION_STAT_ACTIVITY 字段

名称	类型	描述

名称	类型	描述
datid	oid	用户会话在后台连接到的数据库 OID。
datname	name	用户会话在后台连接到的数据库名称。
pid	bigint	后台线程 ID。
usesysid	oid	登录该后台的用户 OID。
usename	name	登录该后台的用户名。
application_name	text	连接到该后台的应用名。
client_addr	inet	连接到该后台的客户端的 IP 地址。如果此字段是 null，它表明通过服务器机器上 UNIX 套接字连接客户端或者这是内部进程，如 autovacuum。
client_hostname	text	客户端的主机名，这个字段是通过 client_addr 的反向 DNS 查找得到。这个字段只有在启动 log_hostname 且使用 IP 连接时才非空。
client_port	integer	客户端用于与后台通讯的 TCP 端口号，如果使用 Unix 套接字，则为-1。
backend_start	timestampwith time zone	该过程开始的时间，即当客户端连接服务器时间。
xact_start	timestampwith time zone	启动当前事务的时间，如果没有事务是活跃的，则为 null。如果当前查询是首个事务，则这列等同于 query_start 列。
query_start	timestampwith time zone	开始当前活跃查询的时间，如果 state 的值不是 active，则这个值是上一个查询的开始时间。
state_change	timestampwith time zone	上次状态改变的时间。
waiting	boolean	如果后台当前正等待锁则为 true。
enqueue	text	该字段不支持 。
state	text	该后台当前总体状态。可能值是： • active：后台正在执行一个查询。 • idle：后台正在等待一个新的客户端命令。 • idle in transaction：后台在事务中，但是目前无法执行查询。

名称	类型	描述
		• idle in transaction (aborted)：这个状态除说明事务中有某个语句导致了错误外，类似于 idle in transaction • fastpath function call：后台正在执行一个 fast-path 函数。 • disabled：如果后台禁用 track_activities，则报告这个状态。 说明 普通用户只能查看到自己帐户所对应的会话状态。即其他帐户的 state 信息为空。例如以 judy 用户连接数据库后，在 pg_stat_activity 中查看到的普通用户 joe 及初始用户 vastbase 的 state 信息为空： vastbase=# SELECT datname, usename, usesysid,state,pid FROM pg_stat_activity; datname | usename | usesysid | state | pid ----------+---------+----------+--------+----------------- vastbase | vastbase | 10 | |139968752121616 vastbase | vastbase | 10 | |139968903116560 db_tpcds | judy | 16398 | active |139968391403280 vastbase | vastbase | 10 | |139968643069712 vastbase | vastbase | 10 | |139968680818448 vastbase | joe | 16390 | |139968563377936 (6 rows)
resource_pool	name	用户使用的资源池。
query_id	bigint	查询语句的 ID。
query	text	该后台的最新查询。如果 state 状态是 active（活跃的），此字段显示当前正在执行的查询。所有其他情况表示上一个查询。

17.7.10.GLOBAL_SESSION_STAT_ACTIVITY

显示 Vastbase 内各节点上正在运行的线程相关的信息。查询视图必须具有 monadmin 权限。

表 17-78. GLOBAL_SESSION_STAT_ACTIVITY 字段

名称	类型	描述
coorname	text	数据库进程名称。

名称	类型	描述
datid	oid	用户会话在后台连接到的数据库 OID。
datname	text	用户会话在后台连接到的数据库名称。
pid	bigint	后台线程 ID。
usesysid	oid	登录该后台的用户 OID。
usename	text	登录该后台的用户名。
application_name	text	连接到该后台的应用名。
client_addr	inet	连接到该后台的客户端的 IP 地址。如果此字段是 null，它表明通过服务器机器上 UNIX 套接字连接客户端或者这是内部进程，如 autovacuum。
client_hostname	text	客户端的主机名，这个字段是通过 client_addr 的反向 DNS 查找得到。这个字段只有在启动 log_hostname 且使用 IP 连接时才非空。
client_port	integer	客户端用于与后台通讯的 TCP 端口号，如果使用 Unix 套接字，则为-1。
backend_start	timestampwith time zone	该过程开始的时间，即当客户端连接服务器时间。
xact_start	timestampwith time zone	启动当前事务的时间，如果没有事务是活跃的，则为 null。如果当前查询是首个事务，则这列等同于 query_start 列。
query_start	timestampwith time zone	开始当前活跃查询的时间，如果 state 的值不是 active，则这个值是上一个查询的开始时间。
state_change	timestampwith time zone	上次状态改变的时间。
waiting	boolean	如果后台当前正等待锁则为 true。
enqueue	text	该字段不支持 。
state	text	该后台当前总体状态。可能值是： • active：后台正在执行一个查询。 • idle：后台正在等待一个新的客户端命令。 • idle in transaction：后台在事务中，但是目前无法执行查询。

名称	类型	描述
		• idle in transaction (aborted)：这个状态除说明事务中有某个语句导致了错误外，类似于 idle in transaction • fastpath function call：后台正在执行一个 fast-path 函数。 • disabled：如果后台禁用 track_activities，则报告这个状态。 说明 普通用户只能查看到自己帐户所对应的会话状态。即其他帐户的 state 信息为空。例如以 judy 用户连接数据库后，在 pg_stat_activity 中查看到的普通用户 joe 及初始用户 vastbase 的 state 信息为空： vastbase=# SELECT datname, usename, usesysid,state,pid FROM pg_stat_activity; datname | usename | usesysid | state | pid ----------+---------+----------+--------+----------------- vastbase | vastbase | 10 | |139968752121616 vastbase | vastbase | 10 | |139968903116560 db_tpcds | judy | 16398 | active |139968391403280 vastbase | vastbase | 10 | |139968643069712 vastbase | vastbase | 10 | |139968680818448 vastbase | joe | 16390 | |139968563377936 (6 rows)
resource_pool	name	用户使用的资源池。
query_id	bigint	查询语句的 ID。
query	text	该后台的最新查询。如果 state 状态是 active（活跃的），此字段显示当前正在执行的查询。所有其他情况表示上一个查询。

17.7.11.THREAD_WAIT_STATUS

通过该视图可以检测当前实例中工作线程（backend thread）以及辅助线程（auxiliary thread）的阻塞等待情况，具体事件信息请参见表 14-133、表 14-134、表 14-135 和表 14-136。

表 17-79. THREAD_WAIT_STATUS 字段

名称	类型	描述
node_name	text	数据库进程名称。

名称	类型	描述
db_name	text	数据库名称。
thread_name	text	线程名称。
query_id	bigint	查询 ID，对应 debug_query_id。
tid	bigint	当前线程的线程号。
sessionid	bigint	session 的 ID。
lwtid	integer	当前线程的轻量级线程号。
psessionid	bigint	streaming 线程的父线程。
tlevel	integer	streaming 线程的层级。
smpid	integer	并行线程的 ID。
wait_status	text	当前线程的等待状态。等待状态的详细信息请参见表 14-133。
wait_event	text	如果 wait_status 是 acquire lock、acquire lwlock、wait io 三种类型，此列描述具体的锁、轻量级锁、IO 的信息；否则为空。

17.7.12. GLOBAL_THREAD_WAIT_STATUS

通过该视图可以检测所有节点上工作线程（backend thread）以及辅助线程（auxiliary thread）的阻塞等待情况，查询视图必须具有 monadmin 权限。具体事件信息请参见表 14-133、表 14-134、表 14-135 和表 14-136。

通过 GLOBAL_THREAD_WAIT_STATUS 视图，可以查看 Vastbase 全局各个节点上所有 SQL 语句产生的线程之间的调用层次关系，以及各个线程的阻塞等待状态，从而更容易定位 hang 以及类似现象的原因。

GLOBAL_THREAD_WAIT_STATUS 视图和 THREAD_WAIT_STATUS 视图列定义完全相同，这是由于 GLOBAL_THREAD_WAIT_STATUS 视图本质是到 Vastbase 中各个节点上查询 THREAD_WAIT_STATUS 视图汇总的结果。

表 17-80. GLOBAL_THREAD_WAIT_STATUS 字段

名称	类型	描述
node_name	text	数据库进程名称。
db_name	text	数据库名称。

名称	类型	描述
thread_name	text	线程名称。
query_id	bigint	查询 ID，对应 debug_query_id。
tid	bigint	当前线程的线程号。
sessionid	bigint	session 的 ID
lwtid	integer	当前线程的轻量级线程号。
psessionid	bigint	streaming 线程的父线程。
tlevel	integer	streaming 线程的层级。
smpid	integer	并行线程的 ID。
wait_status	text	当前线程的等待状态。等待状态的详细信息请参见表 14-133。
wait_event	text	如果 wait_status 是 acquire lock、acquire lwlock、wait io 三种类型，此列描述具体的锁、轻量级锁、IO 的信息。否则是空。

17.7.13.LOCAL_THREADPOOL_STATUS

LOCAL_THREADPOOL_STATUS 视图显示线程池下工作线程及会话的状态信息。该视图仅在线程池开启（enable_thread_pool = on）时生效。

表 17-81. LOCAL_THREADPOOL_STATUS 字段

名称	类型	描述
node_name	text	数据库进程名称。
group_id	integer	线程池组 ID。
bind_numa_id	integer	该线程池组绑定的 NUMA ID。
bind_cpu_number	integer	该线程池组绑定的 CPU 信息。如果未绑定 CPU，该值为 NULL。
listener	integer	该线程池组的 Listener 线程数量。
worker_info	text	线程池中线程相关信息，包括以下信息： • default：该线程池组中的初始线程数量。

名称	类型	描述
		• new：该线程池组中新增线程的数量。 • expect：该线程池组中预期线程的数量。 • actual：该线程池组中实际线程的数量。 • idle：该线程池组中空闲线程的数量。 • pending：该线程池组中等待线程的数量。
session_info	text	线程池中会话相关信息，包括以下信息： • total：该线程池组中所有的会话数量。 • waiting：该线程池组中等待调度的会话数量。 • running：该线程池中正在执行的会话数量。 • idle：该线程池组中空闲的会话数量。

17.7.14.GLOBAL_THREADPOOL_STATUS

GLOBAL_THREADPOOL_STATUS 视图显示在所有节点上的线程池中工作线程及会话的状态信息。具体的字段表 17-81。

17.7.15.SESSION_CPU_RUNTIME

SESSION_CPU_RUNTIME 视图显示和当前用户执行复杂作业（正在运行）时的负载管理 CPU 使用的信息。

表 17-82. SESSION_CPU_RUNTIME 字段

名称	类型	描述
datid	oid	连接后端的数据库 OID。
usename	name	登录到该后端的用户名。
pid	bigint	后端线程 ID。
start_time	timestamp with time zone	语句执行的开始时间。
min_cpu_time	bigint	语句在数据库节点上的最小 CPU 时间，单位为 ms。
max_cpu_time	bigint	语句在数据库节点上的最大 CPU 时间，单位为 ms。

名称	类型	描述
total_cpu_time	bigint	语句在数据库节点上的 CPU 总时间，单位为 ms。
query	text	正在执行的语句。
node_group	text	语句所属用户对应的逻辑 Vastbase。
top_cpu_dn	text	cpu 使用量 topN 信息。

17.7.16.SESSION_MEMORY_RUNTIME

SESSION_MEMORY_RUNTIME 视图显示和当前用户执行复杂作业正在运行时的负载管理内存使用的信息。

表 17-83. SESSION_MEMORY_RUNTIME 字段

名称	类型	描述
datid	oid	连接后端的数据库 OID。
username	name	登录到该后端的用户名。
pid	bigint	后端线程 ID。
start_time	timestamp with time zone	语句执行的开始时间。
min_peak_memory	integer	语句在数据库节点上的最小内存峰值大小，单位 MB。
max_peak_memory	integer	语句在数据库节点上的最大内存峰值大小，单位 MB。
spill_info	text	语句在数据库节点上的下盘信息： • None：数据库节点均未下盘 • All：数据库节点均下盘 • [a:b]：数量为 b 个数据库节点中有 a 个数据库节点下盘
query	text	正在执行的语句。
node_group	text	语句所属用户对应的逻辑 Vastbase。
top_mem_dn	text	mem 使用量 topN 信息。

17.7.17.STATEMENT_IOSTAT_COMPLEX_RUNTIME

STATEMENT_IOSTAT_COMPLEX_RUNTIME 视图显示当前用户执行作业正在运行时的 IO 负载管理相关信息。以下涉及到 iops，对于行存，均以万次/s 为单位，对于列存，均以次/s 为单位。

表 17-84. STATEMENT_IOSTAT_COMPLEX_RUNTIME 字段

名称	类型	描述
query_id	bigint	作业 id。
mincurriops	integer	该作业当前 io 在各数据库节点中的最小值。
maxcurriops	integer	该作业当前 io 在各数据库节点中的最大值。
minpeakiops	integer	在作业运行时，作业 io 峰值中，各数据库节点的最小值。
maxpeakiops	integer	在作业运行时，作业 io 峰值中，各数据库节点的最大值。
io_limits	integer	该作业所设 GUC 参数 io_limits。
io_priority	text	该作业所设 GUC 参数 io_priority。
query	text	作业。
node_group	text	作业所属用户对应的逻辑 Vastbase。

17.8. Transaction

17.8.1. TRANSACTIONS_RUNNING_XACTS

显示当前节点运行事务的信息。

表 17-85. TRANSACTIONS_RUNNING_XACTS 字段

名称	类型	描述
handle	integer	事务在 GTM 对应的句柄。
gxid	xid	事务 id 号。
state	tinyint	事务状态（3：prepared 或者 0：starting）。
node	text	节点名称。

名称	类型	描述
xmin	xid	节点上当前数据涉及的最小事务号 xmin。
vacuum	boolean	标志当前事务是否是 lazy vacuum 事务。
timeline	bigint	标志数据库重启次数。
prepare_xid	xid	处于 prepared 状态的事务的 id 号，若不在 prepared 状态，值为 0。
pid	bigint	事务对应的线程 id。
next_xid	xid	数据库主节点传给数据库节点的事务 id 号。

17.8.2. SUMMARY_TRANSACTIONS_RUNNING_XACTS

显示 Vastbase 中各个数据库主节点运行事务的信息，字段内容和 transactions_running_xacts 一致。查询视图必须具有 monadmin 权限。

表 17-86. SUMMARY_TRANSACTIONS_RUNNING_XACTS 字段

名称	类型	描述
handle	integer	事务在 GTM 对应的句柄。
gxid	xid	事务 id 号。
state	tinyint	事务状态（3：prepared 或者 0：starting）。
node	text	节点名称。
xmin	xid	节点上当前数据涉及的最小事务号 xmin。
vacuum	boolean	标志当前事务是否是 lazy vacuum 事务。
timeline	bigint	标志数据库重启次数。
prepare_xid	xid	处于 prepared 状态的事务的 id 号，若不在 prepared 状态，值为 0。
pid	bigint	事务对应的线程 id。
next_xid	xid	数据库主节点传给数据库节点的事务 id 号。

17.8.3. GLOBAL_TRANSACTIONS_RUNNING_XACTS

显示 Vastbase 中各个节点运行事务的信息。查询视图必须具有 monadmin 权限。

表 17-87. GLOBAL_TRANSACTIONS_RUNNING_XACTS 字段

名称	类型	描述
handle	integer	事务在 GTM 对应的句柄。
gxid	xid	事务 id 号。
state	tinyint	事务状态（3：prepared 或者 0：starting）。
node	text	节点名称。
xmin	xid	节点上当前数据涉及的最小事务号 xmin。
vacuum	boolean	标志当前事务是否是 lazy vacuum 事务。
timeline	bigint	标志数据库重启次数。
prepare_xid	xid	处于 prepared 状态的事务的 id 号，若不在 prepared 状态，值为 0。
pid	bigint	事务对应的线程 id。
next_xid	xid	数据库主节点传给数据库节点的事务 id 号。

17.8.4. TRANSACTIONS_PREPARED_XACTS

显示当前准备好进行两阶段提交的事务的信息。

表 17-88. TRANSACTIONS_PREPARED_XACTS 字段

名称	类型	描述
transaction	xid	预备事务的数字事务标识。
gid	text	赋予该事务的全局事务标识。
prepared	timestamp with time zone	事务准备好提交的时间。
owner	name	执行该事务的用户的名称。
database	name	执行该事务所在的数据库名。

17.8.5. SUMMARY_TRANSACTIONS_PREPARED_XACTS

显示 Vastbase 中数据库主节点当前准备好进行两阶段提交的事务的信息。查询视图必须具有 monadmin 权限。

表 17-89. SUMMARY_TRANSACTIONS_PREPARED_XACTS 字段

名称	类型	描述
transaction	xid	预备事务的数字事务标识。
gid	text	赋予该事务的全局事务标识。
prepared	timestamp with time zone	事务准备好提交的时间。
owner	name	执行该事务的用户的名称。
database	name	执行该事务所在的数据库名。

17.8.6. GLOBAL_TRANSACTIONS_PREPARED_XACTS

显示各节点当前准备好进行两阶段提交的事务的信息。查询视图必须具有 monadmin 权限。

表 17-90. GLOBAL_TRANSACTIONS_PREPARED_XACTS 字段

名称	类型	描述
transaction	xid	预备事务的数字事务标识。
gid	text	赋予该事务的全局事务标识。
prepared	timestamp with time zone	事务准备好提交的时间。
owner	name	执行该事务的用户的名称。
database	name	执行该事务所在的数据库名。

17.9. Query

17.9.1. STATEMENT

获得当前节点的执行语句(归一化 SQL)的信息。查询视图必须具有 sysadmin 权限。数据库主节点上可以看到此数据库主节点接收到的归一化的 SQL 的全量统计信息（包含数据库节点）；数据库节点上仅可看到归一化的 SQL 的此节点执行的统计信息。

表 17-91. STATEMENT 字段

名称	类型	描述
node_name	name	数据库进程名称。
node_id	integer	节点的 ID(pgxc_node 中的 node_id)。
user_name	name	用户名称。
user_id	oid	用户 OID。
unique_sql_id	bigint	归一化的 SQL ID。
query	text	归一化的 SQL。
n_calls	bigint	调用次数。
min_elapse_time	bigint	SQL 在内核内的最小运行时间（单位：微秒）。
max_elapse_time	bigint	SQL 在内核内的最大运行时间（单位：微秒）。
total_elapse_time	bigint	SQL 在内核内的总运行时间（单位：微秒）。
n_returned_rows	bigint	SELECT 返回的结果集行数。
n_tuples_fetched	bigint	随机扫描行。
n_tuples_returned	bigint	顺序扫描行。
n_tuples_inserted	bigint	插入行。
n_tuples_updated	bigint	更新行。
n_tuples_deleted	bigint	删除行。
n_blocks_fetched	bigint	buffer 的块访问次数。
n_blocks_hit	bigint	buffer 的块命中次数。

名称	类型	描述
n_soft_parse	bigint	软解析次数, n_soft_parse + n_bard_parse 可能大于 n_calls，因为子查询未计入 n_calls。
n_hard_parse	bigint	硬解析次数, n_soft_parse + n_bard_parse 可能大于 n_calls，因为子查询未计入 n_calls。
db_time	bigint	有效的 DB 时间花费，多线程将累加（单位：微秒）。
cpu_time	bigint	CPU 时间（单位：微秒）。
execution_time	bigint	执行器内执行时间（单位：微秒）。
parse_time	bigint	SQL 解析时间（单位：微秒）。
plan_time	bigint	SQL 生成计划时间（单位：微秒）。
rewrite_time	bigint	SQL 重写时间（单位：微秒）。
pl_execution_time	bigint	plpgsql 上的执行时间（单位：微秒）。
pl_compilation_time	bigint	plpgsql 上的编译时间（单位：微秒）。
net_send_time	bigint	网络上的时间花费（单位：微秒）。
data_io_time	bigint	IO 上的时间花费（单位：微秒）。

17.9.2. SUMMARY_STATEMENT

获得各数据库主节点的执行语句(归一化 SQL)的全量信息(包含数据库节点)。查询视图必须具有 monadmin 权限。

表 17-92. SUMMARY_STATEMENT 字段

名称	类型	描述
node_name	name	数据库进程名称。
node_id	integer	节点的 ID(pgxc_node 中的 node_id)。
user_name	name	用户名称。
user_id	oid	用户 OID。
unique_sql_id	bigint	归一化的 SQL ID。

名称	类型	描述
query	text	归一化的 SQL。
n_calls	bigint	调用次数。
min_elapse_time	bigint	SQL 在内核内的最小运行时间（单位：微秒）。
max_elapse_time	bigint	SQL 在内核内的最大运行时间（单位：微秒）。
total_elapse_time	bigint	SQL 在内核内的总运行时间（单位：微秒）。
n_returned_rows	bigint	SELECT 返回的结果集行数。
n_tuples_fetched	bigint	随机扫描行。
n_tuples_returned	bigint	顺序扫描行。
n_tuples_inserted	bigint	插入行。
n_tuples_updated	bigint	更新行。
n_tuples_deleted	bigint	删除行。
n_blocks_fetched	bigint	buffer 的块访问次数。
n_blocks_hit	bigint	buffer 的块命中次数。
n_soft_parse	bigint	软解析次数。
n_hard_parse	bigint	硬解析次数。
db_time	bigint	有效的 DB 时间花费，多线程将累加（单位：微秒）。
cpu_time	bigint	CPU 时间（单位：微秒）。
execution_time	bigint	执行器内执行时间（单位：微秒）。
parse_time	bigint	SQL 解析时间（单位：微秒）。
plan_time	bigint	SQL 生成计划时间（单位：微秒）。
rewrite_time	bigint	SQL 重写时间（单位：微秒）。
pl_execution_time	bigint	plpgsql 上的执行时间（单位：微秒）。
pl_compilation_time	bigint	plpgsql 上的编译时间（单位：微秒）。
net_send_time	bigint	网络上的时间花费（单位：微秒）。

名称	类型	描述
data_io_time	bigint	IO 上的时间花费（单位：微秒）。

17.9.3. STATEMENT_COUNT

显示数据库当前节点当前时刻执行的五类语句 (SELECT、INSERT、UPDATE、DELETE、MERGE INTO)和(DDL、DML、DCL)统计信息。

说明

普通用户查询 STATEMENT_COUNT 视图仅能看到该用户当前节点的统计信息；管理员权限用户查询 STATEMENT_COUNT 视图则能看到所有用户当前节点的统计信息。当 Vastbase 或该节点重启时，计数将清零，并重新开始计数。计数以节点收到的查询数为准，Vastbasess 内部进行的查询。例如，数据库主节点收到一条查询，若下发多条查询数据库节点，那将在数据库节点上进行相应次数的计数。

表 17-93. STATEMENT_COUNT 字段

名称	类型	描述
node_name	text	数据库进程名称。
user_name	text	用户名。
select_count	bigint	select 语句统计结果。
update_count	bigint	update 语句统计结果。
insert_count	bigint	insert 语句统计结果。
delete_count	bigint	delete 语句统计结果。
mergeinto_count	bigint	merge into 语句统计结果。
ddl_count	bigint	DDL 语句的数量。
dml_count	bigint	DML 语句的数量。
dcl_count	bigint	DCL 语句的数量。
total_select_elapse	bigint	总 select 的时间花费（单位：微秒）。
avg_select_elapse	bigint	平均 select 的时间花费（单位：微秒）。
max_select_elapse	bigint	最大 select 的时间花费(单位：微秒)。
min_select_elapse	bigint	最小 select 的时间花费（单位：微秒）。

名称	类型	描述
total_update_elapse	bigint	总 update 的时间花费（单位：微秒）。
avg_update_elapse	bigint	平均 update 的时间花费(单位：微秒)。
max_update_elapse	bigint	最大 update 的时间花费（单位：微秒）。
min_update_elapse	bigint	最小 update 的时间花费（单位：微秒）。
total_insert_elapse	bigint	总 insert 的时间花费（单位：微秒）。
avg_insert_elapse	bigint	平均 insert 的时间花费（单位：微秒）。
max_insert_elapse	bigint	最大 insert 的时间花费（单位：微秒）。
min_insert_elapse	bigint	最小 insert 的时间花费（单位：微秒）。
total_delete_elapse	bigint	总 delete 的时间花费（单位：微秒）。
avg_delete_elapse	bigint	平均 delete 的时间花费（单位：微秒）。
max_delete_elapse	bigint	最大 delete 的时间花费（单位：微秒）。
min_delete_elapse	bigint	最小 delete 的时间花费（单位：微秒）。

17.9.4. GLOBAL_STATEMENT_COUNT

显示数据库各节点当前时刻执行的五类语句（SELECT、INSERT、UPDATE、DELETE、MERGE INTO）和(DDL、DML、DCL)统计信息。查询视图必须具有 monadmin 权限。

表 17-94. GLOBAL_STATEMENT_COUNT 字段

名称	类型	描述
node_name	text	数据库进程名称。
user_name	text	用户名。
select_count	bigint	select 语句统计结果。
update_count	bigint	update 语句统计结果。
insert_count	bigint	insert 语句统计结果。
delete_count	bigint	delete 语句统计结果。

名称	类型	描述
mergeinto_count	bigint	merge into 语句统计结果。
ddl_count	bigint	DDL 语句的数量。
dml_count	bigint	DML 语句的数量。
dcl_count	bigint	DCL 语句的数量。
total_select_elapse	bigint	总 select 的时间花费（单位：微秒）。
avg_select_elapse	bigint	平均 select 的时间花费（单位：微秒）。
max_select_elapse	bigint	最大 select 的时间花费(单位：微秒)。
min_select_elapse	bigint	最小 select 的时间花费（单位：微秒）。
total_update_elapse	bigint	总 update 的时间花费(单位：微秒)。
avg_update_elapse	bigint	平均 update 的时间花费（单位：微秒）。
max_update_elapse	bigint	最大 update 的时间花费（单位：微秒）。
min_update_elapse	bigint	最小 update 的时间花费（单位：微秒）。
total_insert_elapse	bigint	总 insert 的时间花费（单位：微秒）。
avg_insert_elapse	bigint	平均 insert 的时间花费（单位：微秒）。
max_insert_elapse	bigint	最大 insert 的时间花费(单位：微秒)。
min_insert_elapse	bigint	最小 insert 的时间花费（单位：微秒）。
total_delete_elapse	bigint	总 delete 的时间花费（单位：微秒）。
avg_delete_elapse	bigint	平均 delete 的时间花费(单位：微秒)。
max_delete_elapse	bigint	最大 delete 的时间花费（单位：微秒）。
min_delete_elapse	bigint	最小 delete 的时间花费（单位：微秒）。

17.9.5. SUMMARY_STATEMENT_COUNT

显示数据库汇聚各节点(数据库节点)当前时刻执行的五类语句 (SELECT、INSERT、UPDATE、DELETE、MERGE INTO）和(DDL、DML、DCL)统计信息。查询视图必须具有 monadmin 权限。

表 17-95. SUMMARY_STATEMENT_COUNT 字段

名称	类型	描述
user_name	text	用户名。
select_count	numeric	select 语句统计结果。
update_count	numeric	update 语句统计结果。
insert_count	numeric	insert 语句统计结果。
delete_count	numeric	delete 语句统计结果。
mergeinto_count	numeric	merge into 语句统计结果。
ddl_count	numeric	DDL 语句的数量。
dml_count	numeric	DML 语句的数量。
dcl_count	numeric	DCL 语句的数量。
total_select_elapse	numeric	总 select 的时间花费（单位：微秒）。
avg_select_elapse	bigint	平均 select 的时间花费（单位：微秒）。
max_select_elapse	bigint	最大 select 的时间花费（单位：微秒）。
min_select_elapse	bigint	最小 select 的时间花费（单位：微秒）。
total_update_elapse	numeric	总 update 的时间花费（单位：微秒）。
avg_update_elapse	bigint	平均 update 的时间花费（单位：微秒）。
max_update_elapse	bigint	最大 update 的时间花费（单位：微秒）。
min_update_elapse	bigint	最小 update 的时间花费（单位：微秒）。
total_insert_elapse	numeric	总 insert 的时间花费(单位：微秒)。
avg_insert_elapse	bigint	平均 insert 的时间花费（单位：微秒）。
max_insert_elapse	bigint	最大 insert 的时间花费（单位：微秒）。
min_insert_elapse	bigint	最小 insert 的时间花费（单位：微秒）。
total_delete_elapse	numeric	总 delete 的时间花费（单位：微秒）。
avg_delete_elapse	bigint	平均 delete 的时间花费（单位：微秒）。

名称	类型	描述
max_delete_elapse	bigint	最大 delete 的时间花费（单位：微秒）。
min_delete_elapse	bigint	最小 delete 的时间花费（单位：微秒）。

17.9.6. GLOBAL_STATEMENT_COMPLEX_HISTORY

显示各个节点执行作业结束后的负载管理记录，查询该函数必须具有 monadmin 权限。

表 17-96. GLOBAL_STATEMENT_COMPLEX_HISTORY 的字段

名称	类型	描述
datid	oid	连接后端的数据库 OID。
dbname	text	连接后端的数据库名称。
schemaname	text	模式的名称。
nodename	text	数据库进程名称
username	text	连接到后端的用户名。
application_name	text	连接到后端的应用名。
client_addr	inet	连接到后端的客户端的 IP 地址。 如果此字段是 null，它表明通过服务器机器上 UNIX 套接字连接客户端或者这是内部进程，如 autovacuum。
client_hostname	text	客户端的主机名，这个字段是通过 client_addr 的反向 DNS 查找得到。这个字段只有在启动 log_hostname 且使用 IP 连接时才非空。
client_port	integer	客户端用于与后端通讯的 TCP 端口号，如果使用 Unix 套接字，则为-1。
query_band	text	用于标示作业类型，可通过 GUC 参数 query_band 进行设置，默认为空字符串。
block_time	bigint	语句执行前的阻塞时间，包含语句解析和优化时间，单位 ms。
start_time	timestamp with time zone	语句执行的开始时间。

名称	类型	描述
finish_time	timestamp with time zone	语句执行的结束时间。
duration	bigint	语句实际执行的时间，单位 ms。
estimate_total_time	bigint	语句预估执行时间，单位 ms。
status	text	语句执行结束状态：正常为 finished，异常为 aborted。
abort_info	text	语句执行结束状态为 aborted 时显示异常信息。
resource_pool	text	用户使用的资源池。
control_group	text	语句所使用的 Cgroup。
estimate_memory	integer	语句预估使用内存。
min_peak_memory	integer	语句在数据库节点上的最小内存峰值，单位 MB。
max_peak_memory	integer	语句在数据库节点上的最大内存峰值，单位 MB。
average_peak_memory	integer	语句执行过程中的内存使用平均值，单位 MB。
memory_skew_percent	integer	语句数据库节点间的内存使用倾斜率。
spill_info	text	语句在数据库节点上的下盘信息： • None：数据库节点均未下盘。 • All：数据库节点均下盘。 • [a:b]：数量为 b 个数据库节点中有 a 个数据库节点下盘。
min_spill_size	integer	若发生下盘，数据库节点上下盘的最小数据量，单位 MB，默认为 0。
max_spill_size	integer	若发生下盘，数据库节点上下盘的最大数据量，单位 MB，默认为 0。
average_spill_size	integer	若发生下盘，数据库节点上下盘的平均数据量，单位 MB，默认为 0。
spill_skew_percent	integer	若发生下盘，数据库节点间下盘倾斜率。
min_dn_time	bigint	语句在数据库节点上的最小执行时间，单位 ms。

名称	类型	描述
max_dn_time	bigint	语句在数据库节点上的最大执行时间，单位 ms。
average_dn_time	bigint	语句在数据库节点上的平均执行时间，单位 ms。
dntime_skew_percent	integer	语句在数据库节点的执行时间倾斜率。
min_cpu_time	bigint	语句在数据库节点上的最小 CPU 时间，单位 ms。
max_cpu_time	bigint	语句在数据库节点上的最大 CPU 时间，单位 ms。
total_cpu_time	bigint	语句在数据库节点上的 CPU 总时间，单位 ms。
cpu_skew_percent	integer	语句在数据库节点间的 CPU 时间倾斜率。
min_peak_iops	integer	语句在数据库节点上的每秒最小 IO 峰值 (列存单位是次/s，行存单位是万次/s）。
max_peak_iops	integer	语句在数据库节点上的每秒最大 IO 峰值 (列存单位是次/s，行存单位是万次/s）。
average_peak_iops	integer	语句在数据库节点上的每秒平均 IO 峰值 (列存单位是次/s，行存单位是万次/s）。
iops_skew_percent	integer	语句在数据库节点间的 IO 倾斜率。
warning	text	主要显示如下几类告警信息： • Spill file size large than 256MB • Broadcast size large than 100MB • Early spill • Spill times is greater than 3 • Spill on memory adaptive • Hash table conflict
queryid	bigint	语句执行使用的内部 query id。
query	text	执行的语句。
query_plan	text	语句的执行计划。
node_group	text	语句所属用户对应的逻辑 Vastbase。
cpu_top1_node_name	text	cpu 使用率第 1 的节点名称。

名称	类型	描述
cpu_top2_node_name	text	cpu 使用率第 2 的节点名称。
cpu_top3_node_name	text	cpu 使用率第 3 的节点名称。
cpu_top4_node_name	text	cpu 使用率第 4 的节点名称。
cpu_top5_node_name	text	cpu 使用率第 5 的节点名称。
mem_top1_node_name	text	内存使用量第 1 的节点名称。
mem_top2_node_name	text	内存使用量第 2 的节点名称。
mem_top3_node_name	text	内存使用量第 3 的节点名称。
mem_top4_node_name	text	内存使用量第 4 的节点名称。
mem_top5_node_name	text	内存使用量第 5 的节点名称。
cpu_top1_value	bigint	cpu 使用率。
cpu_top2_value	bigint	cpu 使用率。
cpu_top3_value	bigint	cpu 使用率。
cpu_top4_value	bigint	cpu 使用率。
cpu_top5_value	bigint	cpu 使用率。
mem_top1_value	bigint	内存使用量。
mem_top2_value	bigint	内存使用量。
mem_top3_value	bigint	内存使用量。
mem_top4_value	bigint	内存使用量。
mem_top5_value	bigint	内存使用量。
top_mem_dn	text	内存使用量 topN 信息。
top_cpu_dn	text	cpu 使用量 topN 信息。

17.9.7. GLOBAL_STATEMENT_COMPLEX_HISTORY_TABLE

显示各个节点执行作业结束后的负载管理记录。此数据是从内核中转储到系统表中的数据。具体的字段请参考 17.9.6GLOBAL_STATEMENT_COMPLEX_HISTORY 中的字段。

17.9.8. GLOBAL_STATEMENT_COMPLEX_RUNTIME

显示当前用户在各个节点上正在执行的作业的负载管理记录，查询该函数必须具有 monadmin 权限。

表 17-97. GLOBAL_STATEMENT_COMPLEX_RUNTIME 的字段

名称	类型	描述
datid	oid	连接后端的数据 OID。
dbname	name	连接后端的数据库名称。
schemaname	text	模式的名称。
nodename	text	数据库进程名称
username	name	连接到后端的用户名。
application_name	text	连接到后端的应用名。
client_addr	inet	连接到后端的客户端的 IP 地址。 如果此字段是 null，它表明通过服务器机器上 UNIX 套接字连接客户端或者这是内部进程，如 autovacuum。
client_hostname	text	客户端的主机名，这个字段是通过 client_addr 的反向 DNS 查找得到。这个字段只有在启动 log_hostname 且使用 IP 连接时才非空。
client_port	integer	客户端用于与后端通讯的 TCP 端口号，如果使用 Unix 套接字，则为-1。
query_band	text	用于标示作业类型，可通过 GUC 参数 query_band 进行设置，默认为空字符串。
pid	bigint	后端线程 ID。
block_time	bigint	语句执行前的阻塞时间，单位 ms。
start_time	timestamp with time zone	语句执行的开始时间。

名称	类型	描述
duration	bigint	语句已经执行的时间，单位 ms。
estimate_total_time	bigint	语句执行预估总时间，单位 ms。
estimate_left_time	bigint	语句执行预估剩余时间，单位 ms。
enqueue	text	工作负载管理资源状态。
resource_pool	name	用户使用的资源池。
control_group	text	语句所使用的 Cgroup。
estimate_memory	integer	语句预估使用内存，单位 MB。
min_peak_memory	integer	语句在数据库节点上的最小内存峰值，单位 MB。
max_peak_memory	integer	语句在数据库节点上的最大内存峰值，单位 MB。
average_peak_memory	integer	语句执行过程中的内存使用平均值，单位 MB。
memory_skew_percent	integer	语句在数据库节点间的内存使用倾斜率。
spill_info	text	语句在数据库节点上的下盘信息： • None：数据库节点均未下盘。 • All：数据库节点均下盘。 • [a:b]：数量为 b 个数据库节点中有 a 个数据库节点下盘。
min_spill_size	integer	若发生下盘，数据库节点上下盘的最小数据量，单位 MB，默认为 0。
max_spill_size	integer	若发生下盘，数据库节点上下盘的最大数据量，单位 MB，默认为 0。
average_spill_size	integer	若发生下盘，数据库节点上下盘的平均数据量，单位 MB，默认为 0。
spill_skew_percent	integer	若发生下盘，数据库节点间下盘倾斜率。
min_dn_time	bigint	语句在数据库节点上的最小执行时间，单位 ms。
max_dn_time	bigint	语句在数据库节点上的最大执行时间，单位 ms。
average_dn_time	bigint	语句在数据库节点上的平均执行时间，单位 ms。

名称	类型	描述
dntime_skew_percent	integer	语句在数据库节点的执行时间倾斜率。
min_cpu_time	bigint	语句在数据库节点上的最小 CPU 时间，单位 ms。
max_cpu_time	bigint	语句在数据库节点上的最大 CPU 时间，单位 ms。
total_cpu_time	bigint	语句在数据库节点上的 CPU 总时间，单位 ms。
cpu_skew_percent	integer	语句在数据库节点间的 CPU 时间倾斜率。
min_peak_iops	integer	语句在数据库节点上的每秒最小 IO 峰值 (列存单位是次/s，行存单位是万次/s）。
max_peak_iops	integer	语句在数据库节点上的每秒最大 IO 峰值 (列存单位是次/s，行存单位是万次/s）。
average_peak_iops	integer	语句在数据库节点上的每秒平均 IO 峰值 (列存单位是次/s，行存单位是万次/s）。
iops_skew_percent	integer	语句在数据库节点间的 IO 倾斜率。
warning	text	主要显示如下几类告警信息： • Spill file size large than 256MB • Broadcast size large than 100MB • Early spill • Spill times is greater than 3 • Spill on memory adaptive • Hash table conflict
queryid	bigint	语句执行使用的内部 query id。
query	text	正在执行的语句。
query_plan	text	语句的执行计划。
node_group	text	语句所属用户对应的逻辑 Vastbase。
top_cpu_dn	text	cpu 使用量 topN 信息。
top_mem_dn	text	内存使用量 topN 信息。

17.9.9. STATEMENT_RESPONSETIME_PERCENTILE

获取 VastbaseSQL 响应时间 P80，P95 分布信息。

表 17-98. STATEMENT_RESPONSETIME_PERCENTILE 的字段

名称	类型	描述
p80	bigint	Vastbase80%的 SQL 的响应时间（单位：微秒）。
p95	bigint	Vastbase95%的 SQL 的响应时间（单位：微秒）。

17.9.10.STATEMENT_USER_COMPLEX_HISTORY

STATEMENT_USER_COMPLEX_HISTORY 系统表显示数据库主节点执行作业结束后的负载管理记录。此数据是从内核中转储到系统表中的数据。具体的字段请参考表 14-77。

17.9.11.STATEMENT_COMPLEX_RUNTIME

STATEMENT_COMPLEX_RUNTIME 视图显示当前用户在数据库主节点上正在执行的作业的负载管理记录。

表 17-99. STATEMENT_COMPLEX_RUNTIME 的字段

名称	类型	描述
datid	oid	连接后端的数据 OID。
dbname	name	连接后端的数据库名称。
schemaname	text	模式的名称。
nodename	text	数据库进程名称
username	name	连接到后端的用户名。
application_name	text	连接到后端的应用名。
client_addr	inet	连接到后端的客户端的 IP 地址。 如果此字段是 null，它表明通过服务器机器上 UNIX 套接字连接客户端或者这是内部进程，如 autovacuum。
client_hostname	text	客户端的主机名，这个字段是通过 client_addr 的反向

名称	类型	描述
		DNS 查找得到。这个字段只有在启动 log_hostname 且使用 IP 连接时才非空。
client_port	integer	客户端用于与后端通讯的 TCP 端口号，如果使用 Unix 套接字，则为-1。
query_band	text	用于标示作业类型，可通过 GUC 参数 query_band 进行设置，默认为空字符串。
pid	bigint	后端线程 ID。
block_time	bigint	语句执行前的阻塞时间，单位 ms。
start_time	timestamp with time zone	语句执行的开始时间。
duration	bigint	语句已经执行的时间，单位 ms。
estimate_total_time	bigint	语句执行预估总时间，单位 ms。
estimate_left_time	bigint	语句执行预估剩余时间，单位 ms。
enqueue	text	工作负载管理资源状态。
resource_pool	name	用户使用的资源池。
control_group	text	语句所使用的 Cgroup。
estimate_memory	integer	语句预估使用内存，单位 MB。
min_peak_memory	integer	语句在数据库节点上的最小内存峰值，单位 MB。
max_peak_memory	integer	语句在数据库节点上的最大内存峰值，单位 MB。
average_peak_memory	integer	语句执行过程中的内存使用平均值，单位 MB。
memory_skew_percent	integer	语句在数据库节点间的内存使用倾斜率。
spill_info	text	语句在数据库节点上的下盘信息： • None：数据库节点均未下盘。 • All：数据库节点均下盘。 • [a:b]：数量为 b 个数据库节点中有 a 个数据库节点下盘。
min_spill_size	integer	若发生下盘，数据库节点上下盘的最小数据量，单位

名称	类型	描述
		MB，默认为 0。
max_spill_size	integer	若发生下盘，数据库节点上下盘的最大数据量，单位 MB，默认为 0。
average_spill_size	integer	若发生下盘，数据库节点上下盘的平均数据量，单位 MB，默认为 0。
spill_skew_percent	integer	若发生下盘，数据库节点间下盘倾斜率。
min_dn_time	bigint	语句在数据库节点上的最小执行时间，单位 ms。
max_dn_time	bigint	语句在数据库节点上的最大执行时间，单位 ms。
average_dn_time	bigint	语句在数据库节点上的平均执行时间，单位 ms。
dntime_skew_percent	integer	语句在数据库节点的执行时间倾斜率。
min_cpu_time	bigint	语句在数据库节点上的最小 CPU 时间，单位 ms。
max_cpu_time	bigint	语句在数据库节点上的最大 CPU 时间，单位 ms。
total_cpu_time	bigint	语句在数据库节点上的 CPU 总时间，单位 ms。
cpu_skew_percent	integer	语句在数据库节点间的 CPU 时间倾斜率。
min_peak_iops	integer	语句在数据库节点上的每秒最小 IO 峰值(列存单位是次/s，行存单位是万次/s）。
max_peak_iops	integer	语句在数据库节点上的每秒最大 IO 峰值(列存单位是次/s，行存单位是万次/s）。
average_peak_iops	integer	语句在数据库节点上的每秒平均 IO 峰值(列存单位是次/s，行存单位是万次/s）。
iops_skew_percent	integer	语句在数据库节点间的 IO 倾斜率。
warning	text	主要显示如下几类告警信息： • Spill file size large than 256MB • Broadcast size large than 100MB • Early spill • Spill times is greater than 3 • Spill on memory adaptive

名称	类型	描述
		• Hash table conflict
queryid	bigint	语句执行使用的内部 query id。
query	text	正在执行的语句。
query_plan	text	语句的执行计划。
node_group	text	语句所属用户对应的逻辑 Vastbase。
top_cpu_dn	text	cpu 使用量 topN 信息。
top_mem_dn	text	内存使用量 topN 信息。

17.9.12.STATEMENT_COMPLEX_HISTORY_TABLE

STATEMENT_COMPLEX_HISTORY_TABLE 系统表显示数据库主节点执行作业结束后的负载管理记录。此数据是从内核中转储到系统表中的数据。具体的字段请参考表 14-77。

17.9.13.STATEMENT_COMPLEX_HISTORY

STATEMENT_COMPLEX_HISTORY 视图显示在数据库主节点上执行作业结束后的负载管理记录。

17.9.14.STATEMENT_WLMSTAT_COMPLEX_RUNTIME

STATEMENT_WLMSTAT_COMPLEX_RUNTIME 视图显示和当前用户执行作业正在运行时的负载管理相关信息。

表 17-100. STATEMENT_WLMSTAT_COMPLEX_RUNTIME 字段

名称	类型	描述
datid	oid	连接后端的数据库 OID。
datname	name	连接后端的数据库名称。
threadid	bigint	后端线程 ID。
processid	integer	后端线程的 pid。
usesysid	oid	登录后端的用户 OID。

名称	类型	描述
appname	text	连接到后端的应用名。
username	name	登录到该后端的用户名。
priority	bigint	语句所在 Cgroups 的优先级。
attribute	text	语句的属性： • Ordinary：语句发送到数据库后被解析前的默认属性。 • Simple：简单语句。 • Complicated：复杂语句。 • Internal：数据库内部语句。
block_time	bigint	语句当前为止的 pending 的时间，单位 s。
elapsed_time	bigint	语句当前为止的实际执行时间，单位 s。
total_cpu_time	bigint	语句在上一时间周期内的数据库节点上 CPU 使用的总时间，单位 s。
cpu_skew_percent	integer	语句在上一时间周期内的数据库节点上 CPU 使用的倾斜率。
statement_mem	integer	语句执行使用的 statement_mem，预留字段。
active_points	integer	语句占用的资源池并发点数。
dop_value	integer	语句的从资源池中获取的 dop 值。
control_group	text	该字段不支持。
status	text	该字段不支持
enqueue	text	语句当前的排队情况，包括： • Global：在全局队列中排队。 • Respool：在资源池队列中排队。 • CentralQueue：在中心协调节点(CCN)中排队。 • Transaction：语句处于一个事务块中。 • StoredProc：句处于一个存储过程中。 • None：未在排队。

名称	类型	描述
		• Forced None：事务块语句或存储过程语句由于超出设定的等待时间而强制执行。
resource_pool	name	语句当前所在的资源池。
query	text	该后端的最新查询。如果 state 状态是 active (活的)，此字段显示当前正在执行的查询。所有其他情况表示上一个查询。
is_plana	boolean	逻辑 Vastbase 模式下，语句当前是否占用其他逻辑 Vastbase 的资源执行。该值默认为 f（否）。
node_group	text	语句所属用户对应的逻辑 Vastbase。

17.10. Cache/IO

17.10.1.STATIO_USER_TABLES

STATIO_USER_TABLES 视图显示命名空间中所有用户关系表的 IO 状态信息。

表 17-101. STATIO_USER_TABLES 字段

名称	类型	描述
relid	oid	表 OID。
schemaname	name	该表模式名。
relname	name	表名。
heap_blks_read	bigint	从该表中读取的磁盘块数。
heap_blks_hit	bigint	该表缓存命中数。
idx_blks_read	bigint	从表中所有索引读取的磁盘块数。
idx_blks_hit	bigint	表中所有索引命中缓存数。
toast_blks_read	bigint	该表的 TOAST 表读取的磁盘块数（如果存在）。
toast_blks_hit	bigint	该表的 TOAST 表命中缓冲区数（如果存在）。

名称	类型	描述
tidx_blks_read	bigint	该表的 TOAST 表索引读取的磁盘块数（如果存在）。
tidx_blks_hit	bigint	该表的 TOAST 表索引命中缓冲区数（如果存在）。

17.10.2. SUMMARY_STATIO_USER_TABLES

SUMMARY_STATIO_USER_TABLES 视图显示 Vastbase 内汇聚的命名空间中所有用户关系表的 IO 状态信息。查询视图必须具有 monadmin 权限。

表 17-102. SUMMARY_STATIO_USER_TABLES 字段

名称	类型	描述
schemaname	name	该表模式名。
relname	name	表名。
heap_blks_read	numeric	从该表中读取的磁盘块数。
heap_blks_hit	numeric	此表缓存命中数。
idx_blks_read	numeric	从表中所有索引读取的磁盘块数。
idx_blks_hit	numeric	表中所有索引命中缓存数。
toast_blks_read	numeric	此表的 TOAST 表读取的磁盘块数（如果存在）。
toast_blks_hit	numeric	此表的 TOAST 表命中缓冲区数（如果存在）。
tidx_blks_read	numeric	此表的 TOAST 表索引读取的磁盘块数（如果存在）。
tidx_blks_hit	numeric	此表的 TOAST 表索引命中缓冲区数（如果存在）。

17.10.3. GLOBAL_STATIO_USER_TABLES

GLOBAL_STATIO_USER_TABLES 视图显示各节点的命名空间中所有用户关系表的 IO 状态信息。查询视图必须具有 monadmin 权限。

表 17-103. GLOBAL_STATIO_USER_TABLES 字段

名称	类型	描述

名称	类型	描述
node_name	name	节点名称。
relid	oid	表 OID。
schemaname	name	该表模式名。
relname	name	表名。
heap_blks_read	bigint	从该表中读取的磁盘块数。
heap_blks_hit	bigint	此表缓存命中数。
idx_blks_read	bigint	从表中所有索引读取的磁盘块数。
idx_blks_hit	bigint	表中所有索引命中缓存数。
toast_blks_read	bigint	此表的 TOAST 表读取的磁盘块数（如果存在）。
toast_blks_hit	bigint	此表的 TOAST 表命中缓冲区数（如果存在）。
tidx_blks_read	bigint	此表的 TOAST 表索引读取的磁盘块数（如果存在）。
tidx_blks_hit	bigint	此表的 TOAST 表索引命中缓冲区数（如果存在）。

17.10.4.STATIO_USER_INDEXES

STATIO_USER_INDEXES 视图显示当前节点命名空间中所有用户关系表索引的 IO 状态信息。

表 17-104. STATIO_USER_INDEXES 字段

名称	类型	描述
relid	oid	索引的表的 OID。
indexrelid	oid	该索引的 OID。
schemaname	name	该索引的模式名。
relname	name	该索引的表名。
indexrelname	name	索引名称。
idx_blks_read	bigint	从索引中读取的磁盘块数。
idx_blks_hit	bigint	索引命中缓存数。

17.10.5.SUMMARY_STATIO_USER_INDEXES

SUMMARY_STATIO_USER_INDEXES 视图显示 Vastbase 内汇聚的命名空间中所有用户关系表索引的 IO 状态信息。查询视图必须具有 monadmin 权限。

表 17-105. SUMMARY_STATIO_USER_INDEXES 字段

名称	类型	描述
schemaname	name	该索引的模式名。
relname	name	该索引的表名。
indexrelname	name	索引名称。
idx_blks_read	numeric	从索引中读取的磁盘块数。
idx_blks_hit	numeric	索引命中缓存数。

17.10.6.GLOBAL_STATIO_USER_INDEXES

GLOBAL_STATIO_USER_INDEXES 视图显示各节点的命名空间中所有用户关系表索引的 IO 状态信息。查询视图必须具有 monadmin 权限。

表 17-106. GLOBAL_STATIO_USER_INDEXES 字段

名称	类型	描述
node_name	name	数据库进程名称。
relid	oid	索引的表的 OID。
indexrelid	oid	该索引的 OID。
schemaname	name	该索引的模式名。
relname	name	该索引的表名。
indexrelname	name	索引名称。
idx_blks_read	numeric	从索引中读取的磁盘块数。
idx_blks_hit	numeric	索引命中缓存数。

17.10.7.STATIO_USER_SEQUENCES

STATIO_USER_SEQUENCE 视图显示当前节点的命名空间中所有用户关系表类型为序列的 IO 状态信息。

表 17-107. STATIO_USER_SEQUENCE 字段

名称	类型	描述
relid	oid	序列 OID。
schemaname	name	序列中模式名。
relname	name	序列名。
blks_read	bigint	从序列中读取的磁盘块数。
blks_hit	bigint	序列中缓存命中数。

17.10.8.SUMMARY_STATIO_USER_SEQUENCES

SUMMARY_STATIO_USER_SEQUENCES视图显示Vastbase内汇聚的命名空间中所有用户关系表类型为序列的 IO 状态信息。查询视图必须具有 monadmin 权限。

表 17-108. SUMMARY_STATIO_USER_SEQUENCES 字段

名称	类型	描述
schemaname	name	序列中模式名。
relname	name	序列名。
blks_read	numeric	从序列中读取的磁盘块数。
blks_hit	numeric	序列中缓存命中数。

17.10.9.GLOBAL_STATIO_USER_SEQUENCES

GLOBAL_STATIO_USER_SEQUENCES 视图显示各节点的命名空间中所有用户关系表类型为序列的 IO 状态信息。查询视图必须具有 monadmin 权限。

表 17-109. GLOBAL_STATIO_USER_SEQUENCES 字段

名称	类型	描述
node_name	name	数据库进程名称。
relid	oid	序列 OID。
schemaname	name	序列中模式名。
relname	name	序列名。
blks_read	bigint	从序列中读取的磁盘块数。
blks_hit	bigint	序列中缓存命中数。

17.10.10. STATIO_SYS_TABLES

STATIO_SYS_TABLES 视图显示命名空间中所有系统表的 IO 状态信息。

表 17-110. STATIO_SYS_TABLES 字段

名称	类型	描述
relid	oid	表 OID。
schemaname	name	该表模式名。
relname	name	表名。
heap_blks_read	bigint	从该表中读取的磁盘块数。
heap_blks_hit	bigint	该表缓存命中数。
idx_blks_read	bigint	从表中所有索引读取的磁盘块数。
idx_blks_hit	bigint	表中所有索引命中缓存数。
toast_blks_read	bigint	该表的 TOAST 表读取的磁盘块数（如果存在）。
toast_blks_hit	bigint	该表的 TOAST 表命中缓冲区数（如果存在）。
tidx_blks_read	bigint	该表的 TOAST 表索引读取的磁盘块数（如果存在）。
tidx_blks_hit	bigint	该表的 TOAST 表索引命中缓冲区数（如果存在）。

17.10.11. SUMMARY_STATIO_SYS_TABLES

SUMMARY_STATIO_SYS_TABLES 视图显示 Vastbase 内汇聚的命名空间中所有系统表的 IO 状态信息。查询视图必须具有 monadmin 权限。

表 17-111. SUMMARY_STATIO_SYS_TABLES 字段

名称	类型	描述
schemaname	name	该表模式名。
relname	name	表名。
heap_blks_read	numeric	从该表中读取的磁盘块数。
heap_blks_hit	numeric	此表缓存命中数。
idx_blks_read	numeric	从表中所有索引读取的磁盘块数。
idx_blks_hit	numeric	表中所有索引命中缓存数。
toast_blks_read	numeric	此表的 TOAST 表读取的磁盘块数（如果存在）。
toast_blks_hit	numeric	此表的 TOAST 表命中缓冲区数（如果存在）。
tidx_blks_read	numeric	此表的 TOAST 表索引读取的磁盘块数（如果存在）。
tidx_blks_hit	numeric	此表的 TOAST 表索引命中缓冲区数（如果存在）。

17.10.12. GLOBAL_STATIO_SYS_TABLES

GLOBAL_STATIO_SYS_TABLES 视图显示各节点的命名空间中所有系统表的 IO 状态信息。查询视图必须具有 monadmin 权限。

表 17-112. GLOBAL_STATIO_SYS_TABLES 字段

名称	类型	描述
node_name	name	数据库进程名称。
relid	oid	表 OID。
schemaname	name	该表模式名。
relname	name	表名。

名称	类型	描述
heap_blks_read	bigint	从该表中读取的磁盘块数。
heap_blks_hit	bigint	此表缓存命中数。
idx_blks_read	bigint	从表中所有索引读取的磁盘块数。
idx_blks_hit	bigint	表中所有索引命中缓存数。
toast_blks_read	bigint	此表的 TOAST 表读取的磁盘块数（如果存在）。
toast_blks_hit	bigint	此表的 TOAST 表命中缓冲区数（如果存在）。
tidx_blks_read	bigint	此表的 TOAST 表索引读取的磁盘块数（如果存在）。
tidx_blks_hit	bigint	此表的 TOAST 表索引命中缓冲区数（如果存在）。

17.10.13. STATIO_SYS_INDEXES

STATIO_SYS_INDEXES 显示命名空间中所有系统表索引的 IO 状态信息。

表 17-113. STATIO_SYS_INDEXES 字段

名称	类型	描述
relid	oid	索引的表的 OID。
indexrelid	oid	该索引的 OID。
schemaname	name	该索引的模式名。
relname	name	该索引的表名。
indexrelname	name	索引名称。
idx_blks_read	bigint	从索引中读取的磁盘块数。
idx_blks_hit	bigint	索引命中缓存数。

17.10.14. SUMMARY_STATIO_SYS_INDEXES

SUMMARY_STATIO_SYS_INDEXES视图显示Vastbase内汇聚的命名空间中所有系统表索引的IO状态信息。查询视图必须具有 monadmin 权限。

表 17-114. SUMMARY_STATIO_SYS_INDEXES 字段

名称	类型	描述
schemaname	name	该索引的模式名。
relname	name	该索引的表名。
indexrelname	name	索引名称。
idx_blks_read	numeric	从索引中读取的磁盘块数。
idx_blks_hit	numeric	索引命中缓存数。

17.10.15. GLOBAL_STATIO_SYS_INDEXES

GLOBAL_STATIO_SYS_INDEXES 视图显示各节点的命名空间中所有系统表索引的 IO 状态信息。查询视图必须具有 monadmin 权限。

表 17-115. GLOBAL_STATIO_SYS_INDEXES 字段

名称	类型	描述
node_name	name	数据库进程名称。
relid	oid	索引的表的 OID。
indexrelid	oid	该索引的 OID。
schemaname	name	该索引的模式名。
relname	name	该索引的表名。
indexrelname	name	索引名称。
idx_blks_read	numeric	从索引中读取的磁盘块数。
idx_blks_hit	numeric	索引命中缓存数。

17.10.16. STATIO_SYS_SEQUENCES

STATIO_SYS_SEQUENCES 显示命名空间中所有系统表为序列的 IO 状态信息。

表 17-116. STATIO_SYS_SEQUENCES 字段

名称	类型	描述
relid	oid	序列 OID。
schemaname	name	序列中模式名。
relname	name	序列名。
blks_read	bigint	从序列中读取的磁盘块数。
blks_hit	bigint	序列中缓存命中数。

17.10.17. SUMMARY_STATIO_SYS_SEQUENCES

SUMMARY_STATIO_SYS_SEQUENCES 视图显示 Vastbase 内汇聚的命名空间中所有系统表为序列的 IO 状态信息。查询视图必须具有 monadmin 权限。

表 17-117. SUMMARY_STATIO_SYS_SEQUENCES 字段

名称	类型	描述
schemaname	name	序列中模式名。
relname	name	序列名。
blks_read	numeric	从序列中读取的磁盘块数。
blks_hit	numeric	序列中缓存命中数。

17.10.18. GLOBAL_STATIO_SYS_SEQUENCES

GLOBAL_STATIO_SYS_SEQUENCES 视图显示各节点的命名空间中所有系统表为序列的 IO 状态信息。查询视图必须具有 monadmin 权限。

表 17-118. GLOBAL_STATIO_SYS_SEQUENCES 字段

名称	类型	描述
node_name	name	数据库进程名称。
relid	oid	序列 OID。

名称	类型	描述
schemaname	name	序列中模式名。
relname	name	序列名。
blks_read	bigint	从序列中读取的磁盘块数。
blks_hit	bigint	序列中缓存命中数。

17.10.19. STATIO_ALL_TABLES

STATIO_ALL_TABLES 视图将包含数据库中每个表（包括 TOAST 表）的一行，显示出特定表 I/O 的统计。

表 17-119. STATIO_ALL_TABLES 字段

名称	类型	描述
relid	oid	表 OID。
schemaname	name	该表模式名。
relname	name	表名。
heap_blks_read	bigint	从该表中读取的磁盘块数。
heap_blks_hit	bigint	该表缓存命中数。
idx_blks_read	bigint	从表中所有索引读取的磁盘块数。
idx_blks_hit	bigint	表中所有索引命中缓存数。
toast_blks_read	bigint	该表的 TOAST 表读取的磁盘块数（如果存在）。
toast_blks_hit	bigint	该表的 TOAST 表命中缓冲区数（如果存在）。
tidx_blks_read	bigint	该表的 TOAST 表索引读取的磁盘块数（如果存在）。
tidx_blks_hit	bigint	该表的 TOAST 表索引命中缓冲区数（如果存在）。

17.10.20. SUMMARY_STATIO_ALL_TABLES

SUMMARY_STATIO_ALL_TABLES 视图将包含 Vastbase 内汇聚的数据库中每个表(包括 TOAST 表)的 I/O 的统计。查询视图必须具有 monadmin 权限。

表 17-120. SUMMARY_STATIO_ALL_TABLES 字段

名称	类型	描述
schemaname	name	该表模式名。
relname	name	表名。
heap_blks_read	numeric	从该表中读取的磁盘块数。
heap_blks_hit	numeric	此表缓存命中数。
idx_blks_read	numeric	从表中所有索引读取的磁盘块数。
idx_blks_hit	numeric	表中所有索引命中缓存数。
toast_blks_read	numeric	此表的 TOAST 表读取的磁盘块数（如果存在）。
toast_blks_hit	numeric	此表的 TOAST 表命中缓冲区数（如果存在）。
tidx_blks_read	numeric	此表的 TOAST 表索引读取的磁盘块数（如果存在）。
tidx_blks_hit	numeric	此表的 TOAST 表索引命中缓冲区数（如果存在）。

17.10.21. GLOBAL_STATIO_ALL_TABLES

GLOBAL_STATIO_ALL_TABLES 视图将包含各节点的数据库中每个表（包括 TOAST 表）的 I/O 的统计。查询视图必须具有 monadmin 权限。

表 17-121. GLOBAL_STATIO_ALL_TABLES 字段

名称	类型	描述
node_name	name	数据库进程名称。
relid	oid	表 OID。
schemaname	name	该表模式名。
relname	name	表名。

名称	类型	描述
heap_blks_read	bigint	从该表中读取的磁盘块数。
heap_blks_hit	bigint	此表缓存命中数。
idx_blks_read	bigint	从表中所有索引读取的磁盘块数。
idx_blks_hit	bigint	表中所有索引命中缓存数。
toast_blks_read	bigint	此表的 TOAST 表读取的磁盘块数（如果存在）。
toast_blks_hit	bigint	此表的 TOAST 表命中缓冲区数（如果存在）。
tidx_blks_read	bigint	此表的 TOAST 表索引读取的磁盘块数（如果存在）。
tidx_blks_hit	bigint	此表的 TOAST 表索引命中缓冲区数（如果存在）。

17.10.22. STATIO_ALL_INDEXES

STATIO_ALL_INDEXES 视图包含数据库中的每个索引行，显示特定索引的 I/O 的统计。

表 17-122. STATIO_ALL_INDEXES 字段

名称	类型	描述
relid	oid	索引的表的 OID。
indexrelid	oid	该索引的 OID。
schemaname	name	该索引的模式名。
relname	name	该索引的表名。
indexrelname	name	索引名称。
idx_blks_read	bigint	从索引中读取的磁盘块数。
idx_blks_hit	bigint	索引命中缓存数。

17.10.23. SUMMARY_STATIO_ALL_INDEXES

SUMMARY_STATIO_ALL_INDEXES 视图包含含 Vastbase 内汇聚的数据库中的每个索引行，显示特定索引的 I/O 的统计。查询视图必须具有 monadmin 权限。

表 17-123. SUMMARY_STATIO_ALL_INDEXES 字段

名称	类型	描述
schemaname	name	该索引的模式名。
relname	name	该索引的表名。
indexrelname	name	索引名称。
idx_blks_read	numeric	从索引中读取的磁盘块数。
idx_blks_hit	numeric	索引命中缓存数。

17.10.24. GLOBAL_STATIO_ALL_INDEXES

GLOBAL_STATIO_ALL_INDEXES 视图包含各节点的数据库中的每个索引行，显示特定索引的 I/O 的统计。查询视图必须具有 monadmin 权限。

表 17-124. GLOBAL_STATIO_ALL_INDEXES 字段

名称	类型	描述
node_name	name	数据库进程名称。
relid	oid	索引的表的 OID。
indexrelid	oid	该索引的 OID。
schemaname	name	该索引的模式名。
relname	name	该索引的表名。
indexrelname	name	索引名称。
idx_blks_read	numeric	从索引中读取的磁盘块数。
idx_blks_hit	numeric	索引命中缓存数。

17.10.25. STATIO_ALL_SEQUENCES

STATIO_ALL_SEQUENCES 视图包含数据库中每个序列的每一行，显示特定序列关于 I/O 的统计。

表 17-125. STATIO_ALL_SEQUENCES 字段

名称	类型	描述
relid	oid	序列 OID。
schemaname	name	序列中模式名。
relname	name	序列名。
blks_read	bigint	从序列中读取的磁盘块数。
blks_hit	bigint	序列中缓存命中数。

17.10.26. SUMMARY_STATIO_ALL_SEQUENCES

SUMMARY_STATIO_ALL_SEQUENCES 视图包含 Vastbase 内汇聚的数据库中每个序列的每一行,显示特定序列关于 I/O 的统计。查询视图必须具有 monadmin 权限。

表 17-126. SUMMARY_STATIO_ALL_SEQUENCES 字段

名称	类型	描述
schemaname	name	序列中模式名。
relname	name	序列名。
blks_read	numeric	从序列中读取的磁盘块数。
blks_hit	numeric	序列中缓存命中数。

17.10.27. GLOBAL_STATIO_ALL_SEQUENCES

GLOBAL_STATIO_ALL_SEQUENCES 包含各节点的数据库中每个序列的每一行，显示特定序列关于 I/O 的统计。查询视图必须具有 monadmin 权限。

表 17-127. GLOBAL_STATIO_ALL_SEQUENCES 字段

名称	类型	描述
node_name	name	数据库进程名称。
relid	oid	序列 OID。

名称	类型	描述
schemaname	name	序列中模式名。
relname	name	序列名。
blks_read	bigint	从序列中读取的磁盘块数。
blks_hit	bigint	序列中缓存命中数。

17.10.28. GLOBAL_STAT_DB_CU

GLOBAL_STAT_DB_CU 视图用于查询 Vastbase，每个数据库的 CU 命中情况。可以通过 gs_stat_reset()进行清零。查询视图必须具有 monadmin 权限。

表 17-128. GLOBAL_STAT_DB_CU 字段

名称	类型	描述
node_name1	text	数据库进程名称。
db_name	text	数据库名。
mem_hit	bigint	内存命中次数。
hdd_sync_read	bigint	硬盘同步读次数。
hdd_asyn_read	bigint	硬盘异步读次数。

17.10.29. GLOBAL_STAT_SESSION_CU

GLOBAL_STAT_SESSION_CU 用于查询 Vastbase 各个节点，当前运行 session 的 CU 命中情况。session 退出相应的统计数据会清零。Vastbase 重启后，统计数据也会清零。查询视图必须具有 monadmin 权限。

表 17-129. GLOBAL_STAT_SESSION_CU 字段

名称	类型	描述
mem_hit	integer	内存命中次数。
hdd_sync_read	integer	硬盘同步读次数。

名称	类型	描述
hdd_asyn_read	integer	硬盘异步读次数。

17.11. Utility

17.11.1.REPLICATION_STAT

REPLICATION_STAT 用于描述日志同步状态信息，如发起端发送日志位置，收端接收日志位置等。

表 17-130. REPLICATION_STAT 字段

名称	类型	描述
pid	bigint	线程的 PID。
usesysid	oid	用户系统 ID。
usename	name	用户名。
application_name	text	程序名称。
client_addr	inet	客户端地址。
client_hostname	text	客户端名。
client_port	integer	客户端端口。
backend_start	timestamp with time zone	程序启动时间。
state	text	日志复制的状态（追赶状态，还是一致的流状态）。
sender_sent_location	text	发送端发送日志位置。
receiver_write_location	text	接收端 write 日志位置。
receiver_flush_location	text	接收端 flush 日志位置。
receiver_replay_location	text	接收端 replay 日志位置。
sync_priority	integer	同步复制的优先级（0 表示异步）。
sync_state	text	同步状态（异步复制，同步复制，还是潜在同步者）。

17.11.2. GLOBAL_REPLICATION_STAT

GLOBAL_REPLICATION_STAT 视图用于获得各节点描述日志同步状态信息，如发起端发送日志位置，收端接收日志位置等。查询视图必须具有 monadmin 权限。

表 17-131. GLOBAL_REPLICATION_STAT 字段

名称	类型	描述
node_name	name	数据库进程名称。
pid	bigint	线程的 PID。
usesysid	oid	用户系统 ID。
usename	name	用户名。
application_name	text	程序名称。
client_addr	inet	客户端地址。
client_hostname	text	客户端名。
client_port	integer	客户端端口。
backend_start	timestamp with time zone	程序启动时间。
state	text	日志复制的状态（追赶状态，还是一致的流状态）。
sender_sent_location	text	发送端发送日志位置。
receiver_write_location	text	接收端 write 日志位置。
receiver_flush_location	text	接收端 flush 日志位置。
receiver_replay_location	text	接收端 replay 日志位置。
sync_priority	integer	同步复制的优先级（0 表示异步）。
sync_state	text	同步状态： • 异步复制 • 同步复制 • 潜在同步者

17.11.3. REPLICATION_SLOTS

REPLICATION_SLOTS 视图用于查看复制节点的信息。

表 17-132. REPLICATION_SLOTS 字段

名称	类型	描述
slot_name	text	复制节点的名称。
plugin	text	插件名称。
slot_type	text	复制节点的类型。
datoid	oid	复制节点的数据库 OID。
database	name	复制节点的数据库名称。
active	boolean	复制节点是否为激活状态。
xmin	xid	复制节点事务标识。
catalog_xmin	xid	逻辑复制槽对应的最早解码事务标识。
restart_lsn	text	复制节点的 Xlog 文件信息。
dummy_standby	boolean	复制节点是否为假备。

17.11.4. GLOBAL_REPLICATION_SLOTS

GLOBAL_REPLICATION_SLOTS 视图用于查看 Vastbase 各节点的复制节点的信息。查询视图必须具有 monadmin 权限。

表 17-133. GLOBAL_REPLICATION_SLOTS 字段

名称	类型	描述
node_name	name	数据库进程名称。
slot_name	text	复制节点的名称。
plugin	text	插件名称。
slot_type	text	复制节点的类型。
datoid	oid	复制节点的数据库 OID。

名称	类型	描述
database	name	复制节点的数据库名称。
active	boolean	复制节点是否为激活状态。
x_min	xid	复制节点事务标识。
catalog_xmin	xid	逻辑复制槽对应的最早解码事务标识.
restart_lsn	text	复制节点的 Xlog 文件信息。
dummy_standby	boolean	复制节点是否为假备。

17.11.5. BGWRITER_STAT

BGWRITER_STAT 视图显示关于后端写进程活动的统计信息。

表 17-134. BGWRITER_STAT 字段

名称	类型	描述
checkpoints_timed	bigint	执行的定期检查点数。
checkpoints_req	bigint	执行的需求检查点数。
checkpoint_write_time	double precision	花费在检查点处理部分的时间总量，其中文件被写入到磁盘，以毫秒为单位。
checkpoint_sync_time	double precision	花费在检查点处理部分的时间总量，其中文件被同步到磁盘，以毫秒为单位。
buffers_checkpoint	bigint	检查点写缓冲区数量。
buffers_clean	bigint	后端写进程写缓冲区数量。
maxwritten_clean	bigint	后端写进程停止清理扫描时间数，因为它写了太多缓冲区。
buffers_backend	bigint	通过后端直接写缓冲区数。
buffers_backend_fsync	bigint	后端不得不执行自己的 fsync 调用的时间数（通常后端写进程处理这些即使后端确实自己写）。
buffers_alloc	bigint	分配的缓冲区数量。

名称	类型	描述
stats_reset	timestamp with time zone	这些统计被重置的时间。

17.11.6.GLOBAL_BGWRITER_STAT

GLOBAL_BGWRITER_STAT 视图显示各节点关于后端写进程活动的统计信息。查询视图必须具有 monadmin 权限。

表 17-135. GLOBAL_BGWRITER_STAT 字段

名称	类型	描述
node_name	name	数据库进程名称。
checkpoints_timed	bigint	执行的定期检查点数。
checkpoints_req	bigint	执行的需求检查点数。
checkpoint_write_time	double precision	花费在检查点处理部分的时间总量,其中文件被写入到磁盘，以毫秒为单位。
checkpoint_sync_time	double precision	花费在检查点处理部分的时间总量,其中文件被同步到磁盘，以毫秒为单位。
buffers_checkpoint	bigint	检查点写缓冲区数量。
buffers_clean	bigint	后端写进程写缓冲区数量。
maxwritten_clean	bigint	后端写进程停止清理扫描时间数,因为它写了太多缓冲区。
buffers_backend	bigint	通过后端直接写缓冲区数。
buffers_backend_fsync	bigint	后端不得不执行自己的 fsync 调用的时间数（通常后端写进程处理这些即使后端确实自己写）。
buffers_alloc	bigint	分配的缓冲区数量。
stats_reset	timestamp with time zone	这些统计被重置的时间。

17.11.7.GLOBAL_CKPT_STATUS

GLOBAL_CKPT_STATUS 视图用于显示 Vastbase 所有实例的检查点信息和各类日志刷页情况。查询视图必须具有 monadmin 权限。

表 17-136. GLOBAL_CKPT_STATUS 字段

名称	类型	描述
node_name	text	数据库进程名称。
ckpt_redo_point	test	当前实例的检查点。
ckpt_clog_flush_num	bigint	从启动到当前时间 clog 刷盘页面数。
ckpt_csnlog_flush_num	bigint	从启动到当前时间 csnlog 刷盘页面数。
ckpt_multixact_flush_num	bigint	从启动到当前时间 multixact 刷盘页面数。
ckpt_predicate_flush_num	bigint	从启动到当前时间 predicate 刷盘页面数。
ckpt_twophase_flush_num	bigint	从启动到当前时间 twophase 刷盘页面数。

17.11.8.GLOBAL_DOUBLE_WRITE_STATUS

GLOBAL_DOUBLE_WRITE_STATUS 视图显示 Vastbase 所有实例的双写文件的情况。查询视图必须具有 monadmin 权限。

表 17-137. GLOBAL_DOUBLEWRITE_STATUS 字段

名称	类型	描述
node_name	text	数据库进程名称。
curr_dwn	bigint	当前双写文件的序列号。
curr_start_page	bigint	当前双写文件恢复起始页面。
file_trunc_num	bigint	当前双写文件复用的次数。
file_reset_num	bigint	当前双写文件写满后发生重置的次数。
total_writes	bigint	当前双写文件总的 I/O 次数。
low_threshold_writes	bigint	低效率写双写文件的 I/O 次数（一次 I/O 刷页数量少于 16

名称	类型	描述
		页面）。
high_threshold_writes	bigint	高效率写双写文件的 I/O 次数(一次 I/O 刷页数量多于一批，421 个页面）。
total_pages	bigint	当前刷页到双写文件区的总的页面个数。
low_threshold_pages	bigint	低效率刷页的页面个数。
high_threshold_pages	bigint	高效率刷页的页面个数。

17.11.9. GLOBAL_PAGEWRITER_STATUS

GLOBAL_PAGEWRITER_STATUS 视图显示 Vastbase 实例的刷页信息和检查点信息。查询视图必须具有 monadmin 权限。

表 17-138. GLOBAL_PAGEWRITER_STATUS 字段

名称	类型	描述
node_name	text	数据库进程名称。
pgwr_actual_flush_total_num	bigint	从启动到当前时间总计刷脏页数量。
pgwr_last_flush_num	integer	上一批刷脏页数量。
remain_dirty_page_num	bigint	当前预计还剩余多少脏页。
queue_head_page_rec_lsn	text	当前实例的脏页队列第一个脏页的 recovery_lsn。
queue_rec_lsn	text	当前实例的脏页队列的 recovery_lsn。
current_xlog_insert_lsn	text	当前实例 XLog 写入的位置。
ckpt_redo_point	text	当前实例的检查点。

17.11.10. GLOBAL_RECORD_RESET_TIME

GLOBAL_RECORD_RESET_TIME 用于重置（重启，主备倒换，数据库删除）汇聚 Vastbase 统计信息时间。查询视图必须具有 monadmin 权限。

表 17-139. GLOBAL_RECORD_RESET_TIME 字段

名称	类型	描述
node_name	text	数据库进程名称。
reset_time	timestamp with time zone	重置时间点。

17.11.11. GLOBAL_REDO_STATUS

GLOBAL_REDO_STATUS 视图显示 Vastbase 实例的日志回放情况。查询视图必须具有 monadmin 权限。

表 17-140. GLOBAL_REDO_STATUS 字段

名称	类型	描述
node_name	text	数据库进程名称。
redo_start_ptr	bigint	当前实例日志回放的起始点。
redo_start_time	bigint	当前实例日志回放的起始 UTC 时间。
redo_done_time	bigint	当前实例日志回放的结束 UTC 时间。
curr_time	bigint	当前实例的当前 UTC 时间。
min_recovery_point	bigint	当前实例日志的最小一致性点位置。
read_ptr	bigint	当前实例日志的读取位置。
last_replayed_read_ptr	bigint	当前实例的日志回放位置。
recovery_done_ptr	bigint	当前实例启动完成时的回放位置。
read_xlog_io_counter	bigint	当前实例读取回放日志的 io 次数计数。
read_xlog_io_total_dur	bigint	当前实例读取回放日志的 io 总时延。
read_data_io_counter	bigint	当前实例回放过程中读取数据页面的 io 次数计数。
read_data_io_total_dur	bigint	当前实例回放过程中读取数据页面的 io 总时延。
write_data_io_counter	bigint	当前实例回放过程中写数据页面的 io 次数计数。
write_data_io_total_dur	bigint	当前实例回放过程中写数据页面的 io 总时延。

名称	类型	描述
process_pending_counter	bigint	当前实例回放过程中日志分发线程的同步次数计数。
process_pending_total_dur	bigint	当前实例回放过程中日志分发线程的同步总时延。
apply_counter	bigint	当前实例回放过程中回放线程的同步次数计数。
apply_total_dur	bigint	当前实例回放过程中回放线程的同步总时延。
speed	bigint	当前实例日志回放速率。
local_max_ptr	bigint	当前实例启动成功后本地收到的回放日志的最大值。
primary_flush_ptr	bigint	主机落盘日志的位置。
worker_info	text	当前实例回放线程信息，若没有开并行回放则该值为空。

17.11.12. GLOBAL_RECOVERY_STATUS

GLOBAL_RECOVERY_STATUS 视图显示关于主机和备机的日志流控信息。查询视图必须具有 monadmin 权限。

表 17-141. GLOBAL_RECOVERY_STATUS 字段

名称	类型	描述
node_name	text	主机进程名称，包含主机和备机。
standby_node_name	text	备机进程名称。
source_ip	text	主机的 IP 地址。
source_port	integer	主机的端口号。
dest_ip	text	备机的 IP 地址。
dest_port	integer	备机的端口号。
current_rto	bigint	备机当前的日志流控时间，单位秒。
target_rto	bigint	备机通过 GUC 参数设置的预期流控时间，单位秒。
current_sleep_time	bigint	为了达到这个预期主机所需要的睡眠时间，单位微妙。

17.11.13. CLASS_VITAL_INFO

CLASS_VITAL_INFO 视图用于做 WDR 时校验相同的表或者索引的 Oid 是否一致。

表 17-142. CLASS_VITAL_INFO 字段

名称	类型	描述
relid	oid	表的 oid。
schemaname	name	schema 名称。
relname	name	表名。
relkind	"char"	表示对象类型，取值范围如下： • r：表示普通表。 • t：表示 toast 表。 • i：表示索引。

17.11.14. USER_LOGIN

USER_LOGIN 用来记录用户登录和退出次数的相关信息。

表 17-143. USER_LOGIN 字段

名称	类型	描述
node_name	text	数据库进程名称。
user_name	text	用户名称。
user_id	integer	用户 oid(同 pg_authid 中的 oid 字段)。
login_counter	bigint	登录次数。
logout_counter	bigint	退出次数。

17.11.15. SUMMARY_USER_LOGIN

SUMMARY_USER_LOGIN 用来记录数据库主节点上用户登录和退出次数的相关信息，查询视图必须具有 monadmin 权限。

表 17-144. SUMMARY_USER_LOGIN 字段

名称	类型	描述
node_name	text	数据库进程名称。
user_name	text	用户名称。
user_id	integer	用户 oid(同 pg_authid 中的 oid 字段)。
login_counter	bigint	登录次数。
logout_counter	bigint	退出次数。

17.12. Lock

17.12.1.LOCKS

LOCKS 视图用于查看各打开事务所持有的锁信息。

表 17-145. LOCKS 字段

名称	类型	描述
locktype	text	被锁定对象的类型：relation，extend，page，tuple，transactionid，virtualxid，object，userlock，advisory。
database	oid	被锁定对象所在数据库的 OID： • 如果被锁定的对象是共享对象，则 OID 为 0。 • 如果是一个事物 ID，则为 NULL。
relation	oid	关系的 OID，如果锁定的对象不是关系，也不是关系的一部分，则为 NULL。
page	integer	关系内部的页面编号，如果对象不是关系页或者不是行页，则为 NULL。
tuple	smallint	页面里边的行编号，如果对象不是行，则为 NULL。
bucket	integer	哈希桶号。
virtualxid	text	事务的虚拟 ID，如果对象不是一个虚拟事务 ID，则为 NULL。
transactionid	xid	事务的 ID，如果对象不是一个事务 ID，则为 NULL。

名称	类型	描述
classid	oid	包含该对象的系统表的 OID，如果对象不是普通的数据库对象，则为 NULL。
objid	oid	对象在其系统表内的 OID，如果对象不是普通数据库对象，则为 NULL。
objsubid	smallint	对于表的一个字段，这是字段编号；对于其他对象类型，这个字段是零；如果这个对象不是普通数据库对象，则为 NULL。
virtualtransaction	text	持有此锁或者在等待此锁的事务的虚拟 ID。
pid	bigint	持有或者等待这个锁的服务器线程的逻辑 ID。如果锁是被一个预备事务持有的，则为 NULL。
sessionid	bigint	持有或者等待这个锁的会话 ID。如果锁是被一个预备事务持有的，则为 NULL。
mode	text	这个线程持有的或者是期望的锁模式。
granted	boolean	• 如果锁是持有锁，则为 TRUE。 • 如果锁是等待锁，则为 FALSE。
fastpath	boolean	如果通过 fast-path 获得锁，则为 TRUE；如果通过主要的锁表获得，则为 FALSE。

17.12.2. GLOBAL_LOCKS

GLOBAL_LOCKS 视图用于查看各节点各打开事务所持有的锁信息。查询视图必须具有 monadmin 权限。

表 17-146. GLOBAL_LOCKS 字段

名称	类型	描述
node_name	name	数据库进程名称。
locktype	text	被锁定对象的类型：relation，extend，page，tuple，transactionid，virtualxid，object，userlock，advisory。
database	oid	被锁定对象所在数据库的 OID： • 如果被锁定的对象是共享对象，则 OID 为 0。

名称	类型	描述
		• 如果是一个事物 ID，则为 NULL。
relation	oid	关系的 OID，如果锁定的对象不是关系，也不是关系的一部分，则为 NULL。
page	integer	关系内部的页面编号，如果对象不是关系页或者不是行页，则为 NULL。
tuple	smallint	页面里边的行编号，如果对象不是行，则为 NULL。
virtualxid	text	事务的虚拟 ID，如果对象不是一个虚拟事务 ID，则为 NULL。
transactionid	xid	事务的 ID，如果对象不是一个事务 ID，则为 NULL。
classid	oid	包含该对象的系统表的 OID，如果对象不是普通的数据库对象，则为 NULL。
objid	oid	对象在其系统表内的 OID，如果对象不是普通数据库对象，则为 NULL。
objsubid	smallint	对于表的一个字段，这是字段编号；对于其他对象类型，这个字段是零；如果这个对象不是普通数据库对象，则为 NULL。
virtualtransaction	text	持有此锁或者在等待此锁的事务的虚拟 ID。
pid	bigint	持有或者等待这个锁的服务器线程的逻辑 ID。如果锁是被一个预备事务持有的，则为 NULL。
mode	text	这个线程持有的或者是期望的锁模式。
granted	boolean	• 如果锁是持有锁，则为 TRUE。 • 如果锁是等待锁，则为 FALSE。
fastpath	boolean	如果通过 fast-path 获得锁，则为 TRUE；如果通过主要的锁表获得，则为 FALSE。

17.13. Wait Events

17.13.1. WAIT_EVENTS

WAIT_EVENTS 显示当前节点的 event 的等待相关的统计信息。具体事件信息见表 14-133、表 14-134、表 14-135 和表 14-136。

表 17-147. WAIT_EVENTS 字段

名称	类型	描述
nodename	text	数据库进程名称。
type	text	event 类型。
event	text	event 名称。
wait	bigint	等待次数。
failed_wait	bigint	失败的等待次数。
total_wait_time	bigint	总等待时间（单位：微秒）。
avg_wait_time	bigint	平均等待时间（单位：微秒）。
max_wait_time	bigint	最大等待时间（单位：微秒）。
min_wait_time	bigint	最小等待时间（单位：微秒）。

17.13.2. GLOBAL_WAIT_EVENTS

GLOBAL_WAIT_EVENTS 视图显示各节点的 event 的等待相关的统计信息。查询视图必须具有 monadmin 权限。

表 17-148. GLOBAL_WAIT_EVENTS 字段

名称	类型	描述
nodename	text	数据库进程名称。
type	text	event 类型。
event	text	event 名称。

名称	类型	描述
wait	bigint	等待次数。
failed_wait	bigint	失败的等待次数。
total_wait_time	bigint	总等待时间（单位：微秒）。
avg_wait_time	bigint	平均等待时间（单位：微秒）。
max_wait_time	bigint	最大等待时间（单位：微秒）。
min_wait_time	bigint	最小等待时间（单位：微秒）。

17.14. Configuration

17.14.1.CONFIG_SETTINGS

CONFIG_SETTINGS 视图显示数据库运行时参数的相关信息。

表 17-149. CONFIG_SETTINGS 字段

名称	类型	描述
name	text	参数名称。
setting	text	参数当前值。
unit	text	参数的隐式结构。
category	text	参数的逻辑组。
short_desc	text	参数的简单描述。
extra_desc	text	参数的详细描述。
context	text	设置参数值的上下文，包括 internal，postmaster，sighup，backend，superuser，user。
vartype	text	参数类型，包括 bool，enum，integer，real，string。
source	text	参数的赋值方式。
min_val	text	参数最大值。如果参数类型不是数值型，那么该字段值为 null。

名称	类型	描述
max_val	text	参数最小值。如果参数类型不是数值型，那么该字段值为 null。
enumvals	text[]	enum 类型参数合法值。如果参数类型不是 enum 型，那么该字段值为 null。
boot_val	text	数据库启动时参数默认值。
reset_val	text	数据库重置时参数默认值。
sourcefile	text	设置参数值的配置文件。如果参数不是通过配置文件赋值，那么该字段值为 null。
sourceline	integer	设置参数值的配置文件的行号。如果参数不是通过配置文件赋值，那么该字段值为 null。

17.14.2. GLOBAL_CONFIG_SETTINGS

GLOBAL_CONFIG_SETTINGS 显示各节点数据库运行时参数的相关信息。查询视图必须具有 monadmin 权限。

表 17-150. GLOBAL_CONFIG_SETTINGS 的字段

名称	类型	描述
node_name	text	数据库进程名称。
name	text	参数名称。
setting	text	参数当前值。
unit	text	参数的隐式结构。
category	text	参数的逻辑组。
short_desc	text	参数的简单描述。
extra_desc	text	参数的详细描述。
context	text	设置参数值的上下文，包括 internal，postmaster，sighup，backend，superuser，user。
vartype	text	参数类型，包括 bool，enum，integer，real，string。

名称	类型	描述
source	text	参数的赋值方式。
min_val	text	参数最大值。如果参数类型不是数值型，那么该字段值为 null。
max_val	text	参数最小值。如果参数类型不是数值型，那么该字段值为 null。
enumvals	text[]	enum 类型参数合法值。如果参数类型不是 enum 型，那么该字段值为 null。
boot_val	text	数据库启动时参数默认值。
reset_val	text	数据库重置时参数默认值。
sourcefile	text	设置参数值的配置文件。如果参数不是通过配置文件赋值，那么该字段值为 null。
sourceline	integer	设置参数值的配置文件的行号。如果参数不是通过配置文件赋值，那么该字段值为 null。

17.15. Operator

17.15.1.OPERATOR_HISTORY_TABLE

OPERATOR_HISTORY_TABLE 系统表显示执行作业结束后的算子相关的记录。此数据是从内核中转储到系统表中的数据。

表 17-151. OPERATOR_HISTORY_TABLE 的字段

名称	类型	描述
queryid	bigint	语句执行使用的内部 query_id。
pid	bigint	后端线程 id。
plan_node_id	integer	查询对应的执行计划的 plan node id。
plan_node_name	text	对应于 plan_node_id 的算子的名称。
start_time	timestamp with time zone	该算子处理第一条数据的开始时间。
duration	bigint	该算子到结束时候总的执行时间(ms)。

名称	类型	描述
query_dop	integer	当前算子执行时的并行度。
estimated_rows	bigint	优化器估算的行数信息。
tuple_processed	bigint	当前算子返回的元素个数。
min_peak_memory	integer	当前算子在数据库节点上的最小内存峰值(MB)。
max_peak_memory	integer	当前算子在数据库节点上的最大内存峰值(MB)。
average_peak_memory	integer	当前算子在数据库节点上的平均内存峰值(MB)。
memory_skew_percent	integer	当前算子在数据库节点间的内存使用倾斜率。
min_spill_size	integer	若发生下盘，数据库节点上下盘的最小数据量(MB)，默认为0。
max_spill_size	integer	若发生下盘，数据库节点上下盘的最大数据量(MB)，默认为0。
average_spill_size	integer	若发生下盘，数据库节点上下盘的平均数据量(MB)，默认为0。
spill_skew_percent	integer	若发生下盘，数据库节点间下盘倾斜率。
min_cpu_time	bigint	该算子在数据库节点上的最小执行时间(ms)。
max_cpu_time	bigint	该算子在数据库节点上的最大执行时间(ms)。
total_cpu_time	bigint	该算子在数据库节点上的总执行时间(ms)。
cpu_skew_percent	integer	数据库节点间执行时间的倾斜率。
warning	text	主要显示如下几类告警信息： • Sort/SetOp/HashAgg/HashJoin spill • Spill file size large than 256MB • Broadcast size large than 100MB • Early spill • Spill times is greater than 3 • Spill on memory adaptive • Hash table conflict

17.15.2. OPERATOR_HISTORY

OPERATOR_HISTORY 视图显示的是当前用户数据库主节点上执行作业结束后的算子的相关记录。记录的数据同表 14-3。

17.15.3. OPERATOR_RUNTIME

OPERATOR_RUNTIME 视图显示当前用户正在执行的作业的算子相关信息。

表 17-152. OPERATOR_RUNTIME 的字段

名称	类型	描述
queryid	bigint	语句执行使用的内部 query_id。
pid	bigint	后端线程 id。
plan_node_id	integer	查询对应的执行计划的 plan node id。
plan_node_name	text	对应于 plan_node_id 的算子的名称。
start_time	timestamp with time zone	该算子处理第一条数据的开始时间。
duration	bigint	该算子到结束时候总的执行时间(ms)。
status	text	当前算子的执行状态，包括 finished 和 running。
query_dop	integer	当前算子执行时的并行度。
estimated_rows	bigint	优化器估算的行数信息。
tuple_processed	bigint	当前算子返回的元素个数。
min_peak_memory	integer	当前算子在数据库节点上的最小内存峰值(MB)。
max_peak_memory	integer	当前算子在数据库节点上的最大内存峰值(MB)。
average_peak_memory	integer	当前算子在数据库节点上的平均内存峰值(MB)。
memory_skew_percent	integer	当前算子在数据库节点的内存使用倾斜率。
min_spill_size	integer	若发生下盘，数据库节点上下盘的最小数据量(MB)，默认为 0。

名称	类型	描述
max_spill_size	integer	若发生下盘，数据库节点上下盘的最大数据量(MB)，默认为0。
average_spill_size	integer	若发生下盘，数据库节点上下盘的平均数据量(MB)，默认为0。
spill_skew_percent	integer	若发生下盘，数据库节点间下盘倾斜率。
min_cpu_time	bigint	该算子在数据库节点上的最小执行时间(ms)。
max_cpu_time	bigint	该算子在数据库节点上的最大执行时间(ms)。
total_cpu_time	bigint	该算子在数据库节点上的总执行时间(ms)。
cpu_skew_percent	integer	数据库节点间执行时间的倾斜率。
warning	text	主要显示如下几类告警信息： • Sort/SetOp/HashAgg/HashJoin spill • Spill file size large than 256MB • Broadcast size large than 100MB • Early spill • Spill times is greater than 3 • Spill on memory adaptive • Hash table conflict

17.15.4. GLOBAL_OPERATOR_HISTORY

GLOBAL_OPERATOR_HISTORY 系统视图显示的是当前用户在数据库主节点上执行作业结束后的算子的相关记录。查询视图必须具有 monadmin 权限。

表 17-153. GLOBAL_OPERATOR_HISTORY 的字段

名称	类型	描述
queryid	bigint	语句执行使用的内部 query_id。
pid	bigint	后端线程 id。
plan_node_id	integer	查询对应的执行计划的 plan node id。

名称	类型	描述
plan_node_name	text	对应于 plan_node_id 的算子的名称。
start_time	timestamp with time zone	该算子处理第一条数据的开始时间。
duration	bigint	该算子到结束时候总的执行时间(ms)。
query_dop	integer	当前算子执行时的并行度。
estimated_rows	bigint	优化器估算的行数信息。
tuple_processed	bigint	当前算子返回的元素个数。
min_peak_memory	integer	当前算子在数据库节点上的最小内存峰值(MB)。
max_peak_memory	integer	当前算子在数据库节点上的最大内存峰值(MB)。
average_peak_memory	integer	当前算子在数据库节点上的平均内存峰值(MB)。
memory_skew_percent	integer	当前算子在数据库节点间的内存使用倾斜率。
min_spill_size	integer	若发生下盘，数据库节点上下盘的最小数据量(MB)，默认为 0。
max_spill_size	integer	若发生下盘，数据库节点上下盘的最大数据量(MB)，默认为 0。
average_spill_size	integer	若发生下盘，数据库节点上下盘的平均数据量(MB)，默认为 0。
spill_skew_percent	integer	若发生下盘，数据库节点间下盘倾斜率。
min_cpu_time	bigint	该算子在数据库节点上的最小执行时间(ms)。
max_cpu_time	bigint	该算子在数据库节点上的最大执行时间(ms)。
total_cpu_time	bigint	该算子在数据库节点上的总执行时间(ms)。
cpu_skew_percent	integer	数据库节点间执行时间的倾斜率。
warning	text	主要显示如下几类告警信息： 1. Sort/SetOp/HashAgg/HashJoin spill 2. Spill file size large than 256MB 3. Broadcast size large than 100MB 4. Early spill

名称	类型	描述
		5. Spill times is greater than 3 6. Spill on memory adaptive 7. Hash table conflict

17.15.5. GLOBAL_OPERATOR_HISTORY_TABLE

GLOBAL_OPERATOR_HISTORY_TABLE 视图显示数据库主节点执行作业结束后的算子相关的记录。此数据是从内核中转储到系统表 GS_WLM_OPERATOR_INFO 中的数据。该视图是查询数据库主节点系统表 GS_WLM_OPERATOR_INFO 的汇聚视图。表字段同表 17-153。查询视图必须具有 monadmin 权限。

17.15.6. GLOBAL_OPERATOR_RUNTIME

GLOBAL_OPERATOR_RUNTIME 视图显示当前用户在数据库主节点上正在执行的作业的算子相关信息。查询视图必须具有 monadmin 权限。

表 17-154. GLOBAL_OPERATOR_RUNTIME 的字段

名称	类型	描述
queryid	bigint	语句执行使用的内部 query_id。
pid	bigint	后端线程 id。
plan_node_id	integer	查询对应的执行计划的 plan node id。
plan_node_name	text	对应于 plan_node_id 的算子的名称。
start_time	timestamp with time zone	该算子处理第一条数据的开始时间。
duration	bigint	该算子到结束时候总的执行时间(ms)。
status	text	当前算子的执行状态，包括 finished 和 running。
query_dop	integer	当前算子执行时的并行度。
estimated_rows	bigint	优化器估算的行数信息。
tuple_processed	bigint	当前算子返回的元素个数。
min_peak_memory	integer	当前算子在数据库节点上的最小内存峰值(MB)。
max_peak_memory	integer	当前算子在数据库节点上的最大内存峰值(MB)。

名称	类型	描述
average_peak_memory	integer	当前算子在数据库节点上的平均内存峰值(MB)。
memory_skew_percent	integer	当前算子在数据库节点的内存使用倾斜率。
min_spill_size	integer	若发生下盘，数据库节点上下盘的最小数据量(MB)，默认为0。
max_spill_size	integer	若发生下盘，数据库节点上下盘的最大数据量(MB)，默认为0。
average_spill_size	integer	若发生下盘，数据库节点上下盘的平均数据量(MB)，默认为0。
spill_skew_percent	integer	若发生下盘，数据库节点间下盘倾斜率。
min_cpu_time	bigint	该算子在数据库节点上的最小执行时间(ms)。
max_cpu_time	bigint	该算子在数据库节点上的最大执行时间(ms)。
total_cpu_time	bigint	该算子在数据库节点上的总执行时间(ms)。
cpu_skew_percent	integer	数据库节点间执行时间的倾斜率。
warning	text	主要显示如下几类告警信息： • Sort/SetOp/HashAgg/HashJoin spill • Spill file size large than 256MB • Broadcast size large than 100MB • Early spill • Spill times is greater than 3 • Spill on memory adaptive • Hash table conflict

17.16. Workload Manager

17.16.1.WLM_USER_RESOURCE_CONFIG

WLM_USER_RESOURCE_CONFIG 视图显示用户的资源配置信息。

表 17-155. WLM_USER_RESOURCE_CONFIG 字段

名称	类型	描述
userid	oid	用户 oid。
username	name	用户名称。
sysadmin	boolean	是否是 sysadmin。
rpoid	oid	资源池的 oid。
respool	name	资源池的名称。
parentid	oid	父用户的 oid。
totalspace	bigint	占用总空间大小。
spacelimit	bigint	空间大上限。
childcount	integer	子用户数量。
childlist	text	子用户的列表。

17.16.2. WLM_USER_RESOURCE_RUNTIME

WLM_USER_RESOURCE_RUNTIME 视图显示所有用户资源使用情况，需要使用管理员用户进行查询。此视图在 GUC 参数“use_workload_manager”为“on”时才有效。

表 17-156. WLM_USER_RESOURCE_RUNTIME 字段

名称	类型	描述
username	name	用户名。
used_memory	integer	正在使用的内存大小，单位 MB。
total_memory	integer	可以使用的内存大小，单位 MB。值为 0 表示未限制最大可用内存，其限制取决于数据库最大可用内存。
used_cpu	integer	正在使用的 CPU 核数。
total_cpu	integer	在该机器节点上，用户关联控制组的 CPU 核数总和。
used_space	bigint	已使用的存储空间大小，单位 KB。
total_space	bigint	可使用的存储空间大小，单位 KB，值为-1 表示未限制最

名称	类型	描述
		大存储空间。
used_temp_space	bigint	已使用的临时空间大小(预留字段，暂未使用)，单位 KB。
total_temp_space	bigint	可使用的临时空间大小(预留字段，暂未使用)，单位 KB，值为-1 表示未限制最大临时存储空间。
used_spill_space	bigint	已使用的下盘空间大小(预留字段，暂未使用)，单位 KB。
total_spill_space	bigint	可使用的下盘空间大小(预留字段，暂未使用)，单位 KB，值为-1 表示未限制最大下盘空间。

18. WDR Snapshot Schema

WDR 是基于快照的工作负载诊断资料库，通过生成快照的方式定时收集系统负载，生成数 WDR 性能报告。WDR Snasphot 在启动后会在用户表空间"pg_default"，数据库" vastbase"下新建 schema "snapshot"，用于持久化 WDR 快照数据。

18.1. WDR Snapshot 原信息表

18.1.1. SNAPSHOT

SNAPSHOT 表记录当前系统中存储的 WDR 快照数据的索引信息，开始，结束时间。

表 18-1. SNAPSHOT 表属性

名称	类型	描述	示例
snapshot_id	bigint	WDR 快照序号。	1
start_ts	timestamp	WDR 快照的开始时间。	2019-12-28 17:11:27.423742+08
end_ts	timestamp	WDR 快照的结束时间。	2019-12-28 17:11:43.67726+08

18.1.2. TABLES_SNAP_TIMESTAMP

TABLES_SNAP_TIMESTAMP 表记录所有存储的 WDR snapshot 中数据库，表对象，数据采集的开始，结束时间。

表 18-2. TABLES_SNAP_TIMESTAMP 表属性

名称	类型	描述	示例
snapshot_id	bigint	WDR 快照序号。	1
db_name	text	WDR snapshot 对应的 database。	tpcc1000
tablename	text	WDR snasphot 对应的 table。	snap_xc_statio_all_indexes
start_ts	timestamp	WDR 快照的开始时间。	2019-12-28 17:11:27.425849+08
end_ts	timestamp	WDR 快照的结束时间。	2019-12-28 17:11:27.707398+08

18.2. WDR Snapshot 数据表

WDR Snapshot 数据表命名原则： snap_{源数据表}。
WDR Snapshot 数据表来源为 DBE_PERF Schema 一章下所有的视图。

18.3. 开启 WDR Snapshot

18.3.1. 相关参数

与 WDR Snapshot 相关的参数有四个：

- enable_wdr_snapshot，设为 on 时，开启 WDR Snapshot 功能，默认为 off。
- wdr_snapshot_retention_days，快照保留周期，默认为 8 天。
- wdr_snapshot_query_timeout，设置快照操作相关的 sql 语句的执行超时时间。
- wdr_snapshot_interval，设置快照生成的时间间隔，默认为 1 小时。

18.3.2. 开启 WDR Snapshot 示例

使用表 10-2 参数设置方式中的方法开启 WDR Snapshot：

```
查询 enable_wdr_snapshot，结果为 off，表示未开启 WDR Snapshot：
vastbase=# show enable_wdr_snapshot;
 enable_wdr_snapshot
---------------------
 Off
(1 row)
修改参数：
postgres=# alter system set enable_wdr_snapshot to on;
ALTER SYSTEM SET
vastbase=#
重启实例生效。
过一段时间，检查是否生成快照：
postgres=# select * from snapshot.snapshot;
 snapshot_id |         start_ts          |          end_ts
-------------+---------------------------+---------------------------
           1 | 2020-09-07 10:20:36.763244+08 | 2020-09-07 10:20:42.166511+08
           2 | 2020-09-07 10:21:13.416352+08 | 2020-09-07 10:21:19.470911+08
```

18. 4. WDR Snapshot 生成性能报告

基于 WDR Snapshot 数据表汇总、统计，生成性能报告。

18.4.1. 前提条件

WDR Snasphot 在启动后（即参数 enable_wdr_snapshot 为 on 时）， 且快照数量大于等于 2 时可以生成报告。

18.4.2. 操作步骤

1. 执行如下命令查询已经生成的快照，以获取快照的 snapshot_id。

```
select * from snapshot.snapshot;
```

2. （可选）执行如下命令手动创建快照。数据库中只有一个快照或者需要查询在当前时段数据库监控的监控数据，可以选择手动执行快照操作。

```
select create_wdr_snapshot();
```

3. 执行如下步骤生成性能报告。

a. 执行如下命令生成格式化性能报告文件。

```
\a \t \o 服务器文件路径
```

上述命令涉及参数说明如下：

- \a：切换非对齐模式。
- \t：切换输出的字段名的信息和行计数脚注。
- \o：把所有的查询结果发送至服务器文件里。
- 服务器文件路径：生成性能报告文件存放路径。用户需要拥有此路径的读写权限。

b. 执行如下命令将查询到的信息写入性能报告中。

```
select generate_wdr_report(begin_snap_id Oid, end_snap_id Oid, varchar report_type, varchar report_scope, int node_name );
```

命令中涉及的参数说明如下。

表 18-3. generate_wdr_report 函数参数说明

参数	说明	取值范围
begin_snap_id	查询时间段开始的 snapshot 的 id(表 snapshot.snaoshot 中的 snapshot_id）。	

参数	说明	取值范围
end_snap_id	查询时间段结束 snapshot 的 id。默认 end_snap_id 大于 begin_snap_id (表 snapshot.snaoshot 中的 snapshot_id）。	
report_type	指定生成 report 的类型。例如，summary/detail/all。	summary：汇总数据。 detail：明细数据。 all：包含 summary 和 detail。
report_scope	指定生成 report 的范围，可以为 cluster 或者 node。	cluster：数据库级别的信息。 node：节点级别的信息。
node_name	在 report_scope 指定为 node 时，需要把该参数指定为对应节点的名称。(节点名称可以执行 select * from pg_node_env;查询)。 在 report_scope 为 cluster 时，该值可以指定为省略、空或者为 NULL。	node：openGauss 中的节点名称。 cluster：省略、空或者 NULL。

c.执行如下命令关闭输出选项及格式化输出命令。

```
\o \a \t
```

18.4.3. 示例

```
查询 enable_wdr_snapshot，此参数需要为 on。
vastbase=# show enable_wdr_snapshot;
 enable_wdr_snapshot
---------------------
 on
(1 row)

--查询已经生成的快照。
vastbase=# select * from snapshot.snapshot;
 snapshot_id |         start_ts          |          end_ts
-------------+---------------------------+---------------------------
           1 | 2020-09-07 10:20:36.763244+08 | 2020-09-07 10:20:42.166511+08
           2 | 2020-09-07 10:21:13.416352+08 | 2020-09-07 10:21:19.470911+08
(2 rows)

--生成格式化性能报告 wdrTestNode.html。
postgres=# \a \t \o /home/om/wdrTestNode.html
Output format is unaligned.
```

```
Showing only tuples.

--向性能报告 wdrTestNode.html 中写入数据。
vastbase=# select generate_wdr_report(1, 2, 'all', 'node', 'dn');

--关闭性能报告 wdrTestNode.html。
postgres=# \o

--生成格式化性能报告 wdrTestCluster.html。
vastbase=# \o /home/om/wdrTestCluster.html

--向格式化性能报告 wdrTestCluster.html 中写入数据。
postgres=# select generate_wdr_report(1, 2, 'all', 'cluster', '');

--关闭性能报告 wdrTestCluster.html。
vastbase=# \o \a \t
Output format is aligned.
Tuples only is off.
```

19. GUC 参数说明

19.1. GUC 使用说明

数据库提供了许多运行参数，配置这些参数可以影响数据库系统的行为。在修改这些参数时请确保用户理解了这些参数对数据库的影响，否则可能会导致无法预料的结果。

注意事项

- 参数中如果取值范围为字符串，此字符串应遵循操作系统的路径和文件名命名规则。
- 取值范围最大值为 INT_MAX 的参数，此选项最大值跟所在的操作系统有关。
- 取值范围最大值为 DBL_MAX 的参数，此选项最大值跟所在的操作系统有关。

19.2. 文件位置

数据库安装后会自动生成三个配置文件（postgresql.conf、pg_hba.conf 和 pg_ident.conf），并统一存放在数据目录（data）下。用户可以使用本节介绍的方法修改配置文件的名称和存放路径。

修改任意一个配置文件的存放目录时，postgresql.conf 里的 data_directory 参数必须设置为实际数据目录（data）。

须知

考虑到配置文件修改一旦出错对数据库的影响很大，不建议安装后再修改本节的配置文件。

data_directory

参数说明：设置 Vastbase 的数据目录（data 目录）。此参数可以通过如下方式指定。

- 在安装 Vastbase 时指定。
- 该参数属于 POSTMASTER 类型参数，请参考表 10-1 中对应设置方法进行设置。

取值范围：字符串，长度大于 0

默认值：安装时指定，如果在安装时不指定，则默认不初始化数据库。

config_file

参数说明：设置主服务器配置文件名称（postgresql.conf）。

该参数属于 POSTMASTER 类型参数，请参考表 10-1 中对应设置方法进行设置。

取值范围：字符串，长度大于 0

默认值：postgresql.conf(实际安装可能带有绝对目录)

hba_file

参数说明：设置基于主机认证（HBA）的配置文件（pg_hba.conf）。此参数只能在配置文件 postgresql.conf 中指定。

该参数属于 POSTMASTER 类型参数，请参考表 10-1 中对应设置方法进行设置。

取值范围：字符串

默认值：pg_hba.conf(实际安装可能带有绝对目录)

ident_file

参数说明：设置用于客户端认证的配置文件的名称（pg_ident.conf）。

该参数属于 POSTMASTER 类型参数，请参考表 10-1 中对应设置方法进行设置。

取值范围：字符串

默认值：pg_ident.conf(实际安装可能带有绝对目录)

external_pid_file

参数说明：声明可被服务器管理程序使用的额外 PID 文件。

该参数属于 POSTMASTER 类型参数，请参考表 10-1 中对应设置方法进行设置。

须知

这个参数只能在数据库服务重新启动后生效。

取值范围：字符串

默认值：空

19.3. 连接和认证

19.3.1. 连接设置

介绍设置客户端和服务器连接方式相关的参数。

listen_addresses

参数说明：声明服务器监听客户端的 TCP/IP 地址。

该参数属于 POSTMASTER 类型参数，请参考表 10-1 中对应设置方法进行设置。

取值范围：

- 主机名或 IP 地址，多个值之间用英文逗号分隔。
- 星号“*”或“0.0.0.0”表示监听所有 IP 地址。配置监听所有 IP 地址存在安全风险，不推荐用户使用。必须与有效地址结合使用（比如本地 IP 等），否则，可能造成 Build 失败的问题。
- 置空则服务器不会监听任何 IP 地址，这种情况下，只有 Unix 域套接字可以用于连接数据库。

默认值：localhost

说明

localhost 表示只允许进行本地“回环”连接。

local_bind_address

参数说明：声明当前节点连接 Vastbase 其他节点绑定的本地 IP 地址。

该参数属于 POSTMASTER 类型参数，请参考表 10-1 中对应设置方法进行设置。

默认值：0.0.0.0（实际值由安装时的配置文件指定）

port

参数说明：Vastbase 服务监听的 TCP 端口号。

该参数属于 POSTMASTER 类型参数，请参考表 10-1 中对应设置方法进行设置。

取值范围：整型，1～65535

默认值：5432（实际值有安装时的配置文件指定）

max_connections

参数说明：允许和数据库连接的最大并发连接数。此参数会影响 Vastbase 的并发能力。

该参数属于 POSTMASTER 类型参数，请参考表 10-1 中对应设置方法进行设置。

取值范围：整型。最小值为 1，最大值为 262143。

默认值：数据库节点：5000。如果该默认值超过内核支持的最大值（在执行 vb_initdb 的时候判断），系统会提示错误。

设置建议：

数据库主节点中此参数建议保持默认值。数据库节点中此参数建议设置为数据库主节点的个数乘以数据库主节点中此参数的值。

增大这个参数可能导致 Vastbase 要求更多的 SystemV 共享内存或者信号量，可能超过操作系统缺省配置的最大值。这种情况下，请酌情对数值加以调整。

sysadmin_reserved_connections

参数说明：为管理员用户预留的最少连接数，不建议设置过大。

该参数属于 POSTMASTER 类型参数，请参考表 10-1 中对应设置方法进行设置。

取值范围：整型，最小值为 0，最大值为 MIN(262143, max_connections)。

默认值：3

unix_socket_directory

参数说明：设置 Vastbase 服务器监听客户端连接的 Unix 域套接字目录。

该参数属于 POSTMASTER 类型参数，请参考表 10-1 中对应设置方法进行设置。

该参数的长度限制于操作系统的长度，超过该限制将会导致 Unix-domain socket path "xxx" is too long 的问题。

取值范围：字符串

默认值：空字符串（实际值由安装时配置文件指定）

unix_socket_group

参数说明：设置 Unix 域套接字的所属组（套接字的所属用户总是启动服务器的用户）。可以与选项 unix_socket_permissions 一起用于对套接字进行访问控制。

该参数属于 POSTMASTER 类型参数，请参考表 10-1 中对应设置方法进行设置。

取值范围：字符串，其中空字符串表示当前用户的缺省组。

默认值：空字符串

unix_socket_permissions

参数说明：设置 Unix 域套接字的访问权限。

Unix 域套接字使用普通的 Unix 文件系统权限集。这个参数的值应该是数值的格式(chmod 和 umask 命令可接受的格式）。如果使用自定义的八进制格式，数字必须以 0 开头。

建议设置为 0770（只有当前连接数据库的用户和同组的人可以访问）或者 0700（只有当前连接数据库的用户自己可以访问，同组或者其他人都没有权限）。

该参数属于 POSTMASTER 类型参数，请参考表 10-1 中对应设置方法进行设置。

取值范围：0000-0777

默认值：0700

说明

在 Linux 中，文档具有十个属性，其中第一个属性为文档类型，后面九个为权限属性，分别为 Owner，Group 及 Others 这三个组别的 read、write、execute 属性。

文档的权限属性分别简写为 r，w，x，这九个属性三个为一组，也可以使用数字来表示文档的权限，对照表如下：

r：4

w: 2

x：1

-：0

同一组（owner/group/others）的三个属性是累加的。

例如，-rwxrwx---表示这个文档的权限为：

owner = rwx = 4+2+1 = 7

group = rwx = 4+2+1 = 7

others = --- = 0+0+0 = 0

所以其权限为 0770。

application_name

参数说明：当前连接请求当中，所使用的客户端名称。

该参数属于 USERSET 类型参数，请参考表 10-1 中对应设置方法进行设置。

取值范围：字符串。

默认值：空字符串(连接到后端的应用名，以实际安装为准)

connection_info

参数说明：连接数据库的驱动类型、驱动版本号、当前驱动的部署路径和进程属主用户。

该参数属于 USERSET 类型参数，属于运维类参数，不建议用户设置。

取值范围：字符串。

默认值：空字符串。

说明

- 空字符串，表示当前连接数据库的驱动不支持自动设置 connection_info 参数或应用程序未设置。

- 驱动连接数 enable_resource_record 据库的时候自行拼接的 connection_info 参数格式如下：

  {"driver_name":"ODBC","driver_version": "(Vastbase x.x build 62e7353e) compiled at 2019-06-26 14:56:09 commit 5361 last mr 9168
  debug","driver_path":"/usr/local/lib/psqlodbcw.so","os_user":"vastbase"}

  默认显示 driver_name 和 driver_version，driver_path 和 os_user 的显示由用户控制（参见 3.2 连接数据库）。

19.3.2. 安全和认证（postgresql.conf）

介绍设置客户端和服务器的安全认证方式的相关参数。

authentication_timeout

参数说明：完成客户端认证的最长时间。如果一个客户端没有在这段时间里完成与服务器端的认证，则服务器自动中断与客户端的连接，这样就避免了出问题的客户端无限制地占用连接数。

该参数属于 SIGHUP 类型参数，请参考表 10-1 中对应设置方法进行设置。

取值范围：整型，最小值为 1，最大值为 600，最小单位为 s。

默认值：1min

auth_iteration_count

参数说明：认证加密信息生成过程中使用的迭代次数。

该参数属于 SIGHUP 类型参数，请参考表 10-1 中对应设置方法进行设置。

取值范围：整型，2048-134217728。

默认值：10000

须知

迭代次数设置过小会降低口令存储的安全性，设置过大会导致认证、用户创建等涉及口令加密的场景性能劣化，请根据实际硬件条件合理设置迭代次数，推荐采用默认迭代次数。

---

session_timeout

参数说明：表明与服务器建立链接后，不进行任何操作的最长时间。

该参数属于 USERSET 类型参数，请参考表 10-1 中对应设置方法进行设置。

取值范围：整型，0-86400，最小单位为 s，0 表示关闭超时设置。

默认值：10min

须知

Vastbase vsql 客户端中有自动重连机制，所以针对初始化用户本地连接，超时后 vsql 表现的现象为断开后重连。

---

ssl

参数说明：启用 SSL 连接。请在使用这个选项之前阅读 3.2.2 使用 vsql 连接。

该参数属于 POSTMASTER 类型参数，请参考表 10-1 中对应设置方法进行设置。

取值范围：布尔型

- on 表示启用 SSL 连接。
- off 表示不启用 SSL 连接。

须知

Vastbase 目前支持 SSL 的场景为客户端连接数据库主节点场景，该参数目前建议只在数据库主节点中开启，数据库节点默认值为 off。开启此参数需要同时配置 ssl_cert_file、ssl_key_file 和 ssl_ca_file 等参数及对应文件，不正确的配置可能会导致 Vastbase 无法正常启动。

默认值：on

require_ssl

参数说明：设置服务器端是否强制要求 SSL 连接，该参数只有当参数 ssl 为 on 时才有效。请在使用这个选项之前阅读 3.2.2 使用 vsql 连接。

该参数属于 SIGHUP 类型参数，请参考表 10-1 中对应设置方法进行设置。

取值范围：布尔型

- on 表示服务器端强制要求 SSL 连接。
- off 表示服务器端对是否通过 SSL 连接不作强制要求。

须知

Vastbase 目前支持 SSL 的场景为客户端连接数据库主节点场景，该参数目前建议只在数据库主节点中开启。

默认值：off

ssl_ciphers

参数说明：指定 SSL 支持的加密算法列表。

该参数属于 POSTMASTER 类型参数，请参考表 10-1 中对应设置方法进行设置。

取值范围：字符串，如果指定多个加密算法，加密算法之间需要以分号分割。详细请参见 5.1.4 用 SSL 进行安全的 TCP/IP 连接获取支持的加密算法。

默认值：ALL

ssl_renegotiation_limit

参数说明：指定在会话密钥重新协商之前，通过 SSL 加密通道可以传输的流量。这个重新协商流量限制机制可以减少攻击者针对大量数据使用密码分析法破解密钥的几率，但是也带来较大的性能损失。流量是指发送和接受的流量总和。使用 SSL 重协商机制可能引入其他风险，因此已禁用 SSL 重协商机制，为保持版本兼容保留此参数，修改参数配置不再起作用。

该参数属于 USERSET 类型参数，请参考表 10-1 中对应设置方法进行设置。

取值范围：整型，最小值为 0，最大值为 2147483647。单位为 KB。其中 0 表示禁用重新协商机制。

默认值：0

ssl_cert_file

参数说明：指定包含 SSL 服务器证书的文件的名称。必须使用相对路径，相对路径是相对于数据目录的。

该参数属于 POSTMASTER 类型参数，请参考表 10-1 中对应设置方法进行设置。

取值范围：字符串

默认值：server.crt

ssl_key_file

参数说明：指定包含 SSL 私钥的文件名称。必须使用相对路径，相对路径是相对于数据目录的。

该参数属于 POSTMASTER 类型参数，请参考表 10-1 中对应设置方法进行设置。

取值范围：字符串

默认值：server.key

ssl_ca_file

参数说明：指定包含 CA 信息的文件的名称。必须使用相对路径，相对路径是相对于数据目录的。

该参数属于 POSTMASTER 类型参数，请参考表 10-1 中对应设置方法进行设置。

取值范围：字符串

默认值：cacert.pem

ssl_crl_file

参数说明：指定包含 CRL 信息的文件的名称。必须使用相对路径，相对路径是相对于数据目录的。

该参数属于 POSTMASTER 类型参数，请参考表 10-1 中对应设置方法进行设置。

取值范围：字符串，其中空表示没有 CA 文件被加载，不进行客户端证书验证。

默认值：空

krb_server_keyfile

参数说明：指定 Kerberos 服务主配置文件的位置，详细请参见 5.1.1 配置客户端接入认证。

该参数属于 SIGHUP 类型参数，请参考表 10-1 中对应设置方法进行设置。

取值范围：字符串

默认值：空

krb_srvname

参数说明：设置 Kerberos 服务名，详细请参见 5.1.1 配置客户端接入认证。

该参数属于 SIGHUP 类型参数，请参考表 10-1 中对应设置方法进行设置。

取值范围：字符串

默认值：postgres

krb_caseins_users

参数说明：设置 Kerberos 用户名是否大小写敏感。

该参数属于 SIGHUP 类型参数，请参考表 10-1 中对应设置方法进行设置。

取值范围：布尔型

- on 表示大小写不敏感
- off 表示大小写敏感

默认值：off

modify_initial_password

参数说明：当 Vastbase 安装成功后，数据库中仅存在一个初始用户（UID 为 10 的用户）。客户通过该帐户初次登录数据库进行操作时，该参数决定是否要对该初始帐户的密码进行修改。

该参数属于 SIGHUP 类型参数，请参考表 10-1 中对应设置方法进行设置。

取值范围：布尔型

- on 表示 Vastbase 安装成功后初始用户首次登录操作前需要修改初始密码。
- off 表示 Vastbase 安装成功后初始用户无需修改初始密码即可进行操作。

默认值：off

password_policy

参数说明：在使用 CREATE ROLE/USER 或者 ALTER ROLE/USER 命令创建或者修改 Vastbase 帐户时，该参数决定是否进行密码复杂度检查。关于密码复杂度检查策略请参见 5.2.9.3 设置密码安全策略。

该参数属于 SIGHUP 类型参数，请参考表 10-1 中对应设置方法进行设置。

须知

从安全性考虑，请勿关闭密码复杂度策略。

取值范围：0、1

- 0 表示不采用任何密码复杂度策略。
- 1 表示采用默认密码复杂度校验策略。

默认值：1

password_reuse_time

参数说明：在使用 ALTER USER 或者 ALTER ROLE 修改用户密码时，该参数指定是否对新密码进行可重用天数检查。关于密码可重用策略请参见 5.2.9.3 设置密码安全策略。

该参数属于 SIGHUP 类型参数，请参考表 10-1 中对应设置方法进行设置。

须知

修改密码时会检查配置参数 password_reuse_time 和 password_reuse_max。

- 当 password_reuse_time 和 password_reuse_max 都为正数时，只要满足其中任一个，即可认为密码可以重用。
- 当 password_reuse_time 为 0 时，表示不限制密码重用天数，仅限制密码重用次数。
- 当 password_reuse_max 为 0 时，表示不限制密码重用次数，仅限制密码重用天数。
- 当 password_reuse_time 和 password_reuse_max 都为 0 时，表示不对密码重用进行限制。

取值范围：浮点型（天），最小值为 0，最大值为 3650。

- 0 表示不检查密码可重用的天数。
- 正数表示新密码不能为该值指定的天数内使用过的密码。

默认值：60

password_reuse_max

参数说明：在使用 ALTER USER 或者 ALTER ROLE 修改用户密码时，该参数指定是否对新密码进行可重用次数检查。关于密码可重用策略请参见 5.2.9.3 设置密码安全策略。

该参数属于 SIGHUP 类型参数，请参考表 10-1 中对应设置方法进行设置。

须知

修改密码时会检查配置参数 password_reuse_time 和 password_reuse_max。

- 当 password_reuse_time 和 password_reuse_max 都为正数时，只要满足其中任一个，即可认为密码可以重用。
- 当 password_reuse_time 为 0 时，表示不限制密码重用天数，仅限制密码重用次数。
- 当 password_reuse_max 为 0 时，表示不限制密码重用次数，仅限制密码重用天数。
- 当 password_reuse_time 和 password_reuse_max 都为 0 时，表示不对密码重用进行限制。

取值范围：整型，最小值为 0，最大值为 1000。

- 0 表示不检查密码可重用次数。
- 正整数表示新密码不能为该值指定的次数内使用过的密码。

默认值：0

password_lock_time

参数说明：该参数指定帐户被锁定后自动解锁的时间。关于帐户自动锁定策略请参见。

该参数属于 SIGHUP 类型参数，请参考表 10-1 中对应设置方法进行设置。

须知

password_lock_time 和 failed_login_attempts 必须都为正数时锁定和解锁功能才能生效。

取值范围：浮点型，最小值为 0，最大值为 365，单位为天。

- 0 表示密码验证失败时，自动锁定功能不生效。
- 正数表示帐户被锁定后，当锁定时间超过 password_lock_time 设定的值时，帐户将会被自行解锁。

默认值：1

failed_login_attempts

参数说明：在任意时候，如果输入密码错误的次数达到 failed_login_attempts 则当前帐户被锁定，password_lock_time 秒后被自动解锁。 例如，登录时输入密码失败，ALTER USER 时修改密码失败等。关于帐户自动锁定策略请参见 5.2.9.3 设置密码安全策略。

该参数属于 SIGHUP 类型参数，请参考表 10-1 中对应设置方法进行设置。

须知

failed_login_attempts 和 password_lock_time 必须都为正数时锁定和解锁功能才能生效。

取值范围：整型，最小值为 0，最大值为 1000。

- 0 表示自动锁定功能不生效。
- 正整数表示当错误密码次数达到 failed_login_attempts 设定的值时，当前帐户将被锁定。

默认值：10

password_encryption_type

参数说明：该字段决定采用何种加密方式对用户密码进行加密存储。修改此参数的配置不会自动触发已有用户密码加密方式的修改，只会影响新创建用户或修改用户密码操作。

该参数属于 SIGHUP 类型参数，请参考表 10-1 中对应设置方法进行设置。

取值范围：0、1、2

- 0 表示采用 md5 方式对密码加密。
- 1 表示采用 sha256 和 md5 两种方式分别对密码加密。
- 2 表示采用 sha256 方式对密码加密。

须知

md5 为不安全的加密算法，不建议用户使用。

默认值：2

password_min_length

参数说明：该字段决定帐户密码的最小长度。

该参数属于 SIGHUP 类型参数，请参考表 10-1 中对应设置方法进行设置。

取值范围：整型，6~999 个字符。

默认值：8

password_max_length

参数说明：该字段决定帐户密码的最大长度。

该参数属于 SIGHUP 类型参数，请参考表 10-1 中对应设置方法进行设置。

取值范围：整型，6~999 个字符。

默认值：32

password_min_uppercase

参数说明：该字段决定帐户密码中至少需要包含大写字母个数。

该参数属于 SIGHUP 类型参数，请参考表 10-1 中对应设置方法进行设置。

取值范围：整型，0~999

- 0 表示没有限制。
- 1~999 表示创建账户所指定的密码中至少需要包含大写字母个数。

默认值：0

password_min_lowercase

参数说明：该字段决定帐户密码中至少需要包含小写字母的个数。

该参数属于 SIGHUP 类型参数，请参考表 10-1 中对应设置方法进行设置。

取值范围：整型，0~999

- 0 表示没有限制。
- 1~999 表示创建帐户所指定的密码中至少需要包含小写字母个数。

默认值：0

password_min_digital

参数说明：该字段决定帐户密码中至少需要包含数字的个数。

该参数属于 SIGHUP 类型参数，请参考表 10-1 中对应设置方法进行设置。

取值范围：整型，0~999

- 0 表示没有限制。
- 1~999 表示创建帐户所指定的密码中至少需要包含数字个数。

默认值：0

password_min_special

参数说明：该字段决定帐户密码中至少需要包含特殊字符个数。

该参数属于 SIGHUP 类型参数，请参考表 10-1 中对应设置方法进行设置。

取值范围：整型，0~999

- 0 表示没有限制。
- 1~999 表示创建帐户所指定的密码中至少需要包含特殊字符个数。

默认值：0

password_effect_time

参数说明：该字段决定帐户密码的有效时间。

该参数属于 SIGHUP 类型参数，请参考表 10-1 中对应设置方法进行设置。

取值范围：浮点型，最小值为 0，最大值为 999，单位为天。

- 0 表示不开启有效期限制功能。
- 1~999 表示创建帐户所指定的密码有效期，临近或超过有效期系统会提示用户修改密码。

默认值：90

password_notify_time

参数说明：该字段决定帐户密码到期前提醒的天数。

该参数属于 SIGHUP 类型参数，请参考表 10-1 中对应设置方法进行设置。

取值范围：整型，最小值为 0，最大值为 999，单位为天。

- 0 表示不开启提醒功能。
- 1~999 表示帐户密码到期前提醒的天数。

默认值：7

19.3.3. 通信库参数

本节介绍通信库相关的参数设置及取值范围等内容。

tcp_keepalives_idle

参数说明：在支持 TCP_KEEPIDLE 套接字选项的系统上，设置发送活跃信号的间隔秒数。不设置发送保持活跃信号，连接就会处于闲置状态。

该参数属于 USERSET 类型参数，请参考表 10-1 中对应设置方法进行设置。

须知

- 如果操作系统不支持 TCP_KEEPIDLE 选项 ，这个参数的值必须为 0。
- 在通过 Unix 域套接字进行的连接的操作系统上，这个参数将被忽略。

取值范围：0-3600，单位为 s。

默认值：60

tcp_keepalives_interval

参数说明：在支持 TCP_KEEPINTVL 套接字选项的操作系统上，以秒数声明在重新传输之间等待响应的时间。

该参数属于 USERSET 类型参数，请参考表 10-1 中对应设置方法进行设置。

取值范围：0-180，单位为 s。

默认值：30

须知

- 如果操作系统不支持 TCP_KEEPINTVL 选项，这个参数的值必须为 0。
- 在通过 Unix 域套接字进行的连接的操作系统上，这个参数将被忽略。

tcp_keepalives_count

参数说明：在支持 TCP_KEEPCNT 套接字选项的操作系统上，设置 Vastbase 服务端在断开与客户端连接之前可以等待的保持活跃信号个数。

该参数属于 USERSET 类型参数，请参考表 10-1 中对应设置方法进行设置。

须知

- 如果操作系统不支持 TCP_KEEPCNT 选项，这个参数的值必须为 0。
- 在通过 Unix 域套接字进行连接的操作系统上，这个参数将被忽略。

取值范围：0-100，其中 0 表示 Vastbase 未收到客户端反馈的保持活跃信号则立即断开连接。

默认值：20

comm_tcp_mode

参数说明：通信库使用 TCP 或 SCTP 协议建立数据通道的切换开关，重启 Vastbase 生效。

该参数属于 POSTMASTER 类型参数，请参考表 10-1 中对应设置方法进行设置。

须知

SCTP 协议的连接不再提供支持，为了保持兼容，提供此参数的接口，但此参数会在设置过程中强制改为 on。

取值范围：布尔型，数据库主节点设置为 on 表示使用 TCP 模式连接数据库节点。

默认值：on

comm_sctp_port

参数说明：TCP 代理通信库或 SCTP 通信库使用的 TCP 或 SCTP 协议监听端口，负责监听数据报文通道。

该参数属于 POSTMASTER 类型参数，请参考表 10-1 中对应设置方法进行设置。

须知

Vastbase 部署时会自动分配此端口号，请不要轻易修改此参数，如端口号配置不正确会导致数据库通信失败。

取值范围：整型，最小值为 0，最大值为 65535。

默认值：7000

comm_control_port

参数说明：TCP 代理通信库或 SCTP 通信库使用的 TCP 协议监听端口。

该参数属于 INTERNAL 类型参数，为固定参数，用户无法修改此参数，只能查看。

须知

Vastbase 部署时会自动分配此端口号，请不要轻易修改此参数，如端口号配置不正确会导致数据库通信失败。

取值范围：整型，最小值为 0，最大值为 65535。

默认值：7001

comm_max_receiver

参数说明：TCP 代理通信库或 SCTP 通信库内部接收线程数量。

该参数属于 POSTMASTER 类型参数，请参考表 10-1 中对应设置方法进行设置。

取值范围：整型，最小值为 1，最大值为 50。

默认值：4

comm_quota_size

参数说明：TCP 代理通信库或 SCTP 通信库最大可连续发送包总大小。使用 1GE 网卡时，建议取较小值，推荐设置为 20KB~40KB。

该参数属于 POSTMASTER 类型参数，请参考表 10-1 中对应设置方法进行设置。

取值范围：整型，最小值为 0，最大值为 2048000，默认单位为 KB。

默认值：1MB

comm_usable_memory

参数说明：单个数据库节点内 TCP 代理通信库或 SCTP 通信库缓存最大可使用内存。

须知

此参数需根据环境内存及部署方式具体配置，过大会造成 OOM，过小会降低 TCP 代理通信库或 SCTP 通信库性能。

该参数属于 POSTMASTER 类型参数，请参考表 10-1 中对应设置方法进行设置。

取值范围：整型，最小值为 100*1024，最大值为 INT_MAX/2，默认单位为 KB。

默认值：4000MB

comm_memory_pool

参数说明：单个数据库节点内 TCP 代理通信库或 SCTP 通信库可使用内存池资源的容量大小。

须知

此参数需根据实际业务情况做调整，若通信库使用内存小，可设置该参数数值较小，反之设置数值较大。

该参数属于 POSTMASTER 类型参数，请参考表 10-1 中对应设置方法进行设置。

取值范围：整型，最小值为 100*1024，最大值为 INT_MAX/2，默认单位为 KB。

默认值：2000MB

comm_memory_pool_percent

参数说明：单个数据库节点内 TCP 代理通信库或 SCTP 通信库可使用内存池资源的百分比，用于自适应负载预留通信库通信消耗的内存大小。

须知

此参数需根据实际业务情况做调整，若通信库使用内存小，可设置该参数数值较小，反之设置数值较大。

该参数属于 POSTMASTER 类型参数，请参考表 10-1 中对应设置方法进行设置。

取值范围：整型，最小值为 0，最大值为 100。

默认值：0

comm_no_delay

参数说明：是否使用通信库连接的 NO_DELAY 属性，重启 Vastbase 生效。

该参数属于 POSTMASTER 类型参数，请参考表 10-1 中对应设置方法进行设置。

取值范围：布尔型

须知

如果 Vastbase 出现因每秒接收数据包过多导致的丢包时，需设置为 off，以便小包合并成大包发送，减少数据包总数。

默认值：off

comm_debug_mode

参数说明：TCP 代理通信库或 SCTP 通信库 debug 模式开关，该参数设置是否打印通信层详细日志。

须知

设置为 on 时，打印日志量较大，会增加额外的 overhead 并降低数据库性能，仅在调试时打开。

该参数属于 USERSET 类型参数，请参考表 10-1 中对应设置方法进行设置。

取值范围：布尔型

- on 表示打印通信库详细 debug 日志。
- off 表示不打印通信库详细 debug 日志。

默认值：off

comm_ackchk_time

参数说明：无数据包接收情况下，该参数设置通信库服务端主动 ACK 触发时长。

该参数属于 USERSET 类型参数，请参考表 10-1 中对应设置方法进行设置。

取值范围：整型，最小值为 0，最大值为 20000，单位为毫秒。取值为 0 表示关闭此功能。

默认值：2000

comm_timer_mode

参数说明：TCP 代理通信库或 SCTP 通信库 timer 模式开关，该参数设置是否打印通信层各阶段时间桩。

须知

设置为 on 时，打印日志量较大，会增加额外的 overhead 并降低数据库性能，仅在调试时打开。

该参数属于 USERSET 类型参数，请参考表 10-1 中对应设置方法进行设置。

取值范围：布尔型

- on 表示打印通信库详细时间桩日志。
- off 表示不打印通信库详细时间桩日志。

默认值：off

comm_stat_mode

参数说明：TCP 代理通信库或 SCTP 通信库 stat 模式开关，该参数设置是否打印通信层的统计信息。

须知

设置为 on 时，打印日志量较大，会增加额外的 overhead 并降低数据库性能，仅在调试时打开。

该参数属于 USERSET 类型参数，请参考表 10-1 中对应设置方法进行设置。

取值范围：布尔型

- on 表示打印通信库统计信息日志。
- off 表示不打印通信库统计信息日志。

默认值：off

19.4. 资源消耗

19.4.1. 内存

介绍与内存相关的参数设置。

须知

这些参数只能在数据库服务重新启动后生效。

memorypool_enable

参数说明：设置是否允许使用内存池。

该参数属于 POSTMASTER 类型参数，请参考表 10-1 中对应设置方法进行设置。

取值范围：布尔型

- on 表示允许使用内存池。
- off 表示不允许使用内存池。

默认值：off

memorypool_size

参数说明：设置内存池大小。

该参数属于 POSTMASTER 类型参数，请参考表 10-1 中对应设置方法进行设置。

取值范围：整型，128*1024～INT_MAX/2，单位为 KB。

默认值：512MB

enable_memory_limit

参数说明：启用逻辑内存管理模块。

该参数属于 POSTMASTER 类型参数，请参考表 10-1 中对应设置方法进行设置。

取值范围：布尔型

- on 表示启用逻辑内存管理模块。
- off 表示不启用逻辑内存管理模块。

默认值：on

须知

若 max_process_memory-shared buffer-cstore buffers 少于 2G，Vastbase 强制把 enable_memory_limit 设置为 off。

---

max_process_memory

参数说明：设置一个数据库节点可用的最大物理内存。

该参数属于 POSTMASTER 类型参数，请参考表 10-1 中对应设置方法进行设置。

取值范围：整型，2*1024*1024～INT_MAX，单位为 KB。

默认值：非从备数据库节点自动适配，公式为（物理内存大小）*0.6/(1+主节点个数)，当结果不足 2GB 时，默认取 2GB。从备节点默认为 12GB。

设置建议：

数据库节点上该数值需要根据系统物理内存及单节点部署主数据库节点个数决定。建议计算公式如下：物理内存大小* 系数，系数通常建议为 0.8。该系数的目的在于预留内存供操作系统内核使用，尽可

能保证系统的可靠性，防止因数据库内存膨胀导致物理节点 OOM。节点的物理内存越小，系数应越小。例如, 128GB 物理内存的节点可以使用 0.9 作为系数，而 32GB 的节点可能只适合 0.7 作为系数。

enable_memory_context_control

参数说明：启用检查内存上下文是否超过给定限制的功能。仅适用于 DEBUG 版本。

该参数属于 SIGHUP 类型参数，请参考表 10-1 中对应设置方法进行设置。

取值范围：布尔型

- on 表示启用最大内存上下文限制检查功能。
- off 表示关闭最大内存上下文限制检查功能。

默认值：off

uncontrolled_memory_context

参数说明：启用检查内存上下文是否超过给定限制的功能时，设置不受此功能约束。仅适用于 DEBUG 版本。

该参数属于 USERSET 类型参数，请参考表 10-1 中对应设置方法进行设置。

查询时会在参数值的最前面添加标题含义字符串“MemoryContext white list:”"。

取值范围：字符串

默认值：空

shared_buffers

参数说明：设置 Vastbase 使用的共享内存大小。增加此参数的值会使 Vastbase 比系统默认设置需要更多的 System V 共享内存。

该参数属于 POSTMASTER 类型参数，请参考表 10-1 中对应设置方法进行设置。

取值范围：整型，16 ~ 1073741823 ，单位为 8KB。

改变 BLCKSZ 的值会改变最小值。

默认值：数据库节点为 1GB。如果操作系统支持的共享内存小于 32MB，则在初始化数据存储区时会自动调整为操作系统支持的最大值。

设置建议：

建议设置 shared_buffers 值为内存的 40%以内。行存列存分开对待。行存设大，列存设小。列存：(单服务器内存/单服务器数据库节点个数)*0.4*0.25。

如果设置较大的 shared_buffers 需要同时增加 checkpoint_segments 的值，因为写入大量新增、修改数据需要消耗更多的时间周期。

bulk_write_ring_size

参数说明：数据并行导入使用的环形缓冲区大小。

该参数属于 USERSET 类型参数，请参考表 10-1 中对应设置方法进行设置。

取值范围：整型，16384 ~ 2147483647，单位为 KB。

默认值：2GB

设置建议：建议导入压力大的场景中增加数据库节点中此参数配置。

standby_shared_buffers_fraction

参数说明：备实例所在服务器使用 shared_buffers 内存缓冲区大小的比例。

该参数属于 SIGHUP 类型参数，请参考表 10-1 中对应设置方法进行设置。

取值范围：双精度类型，0.1~1.0

默认值：0.3

temp_buffers

参数说明：设置每个数据库会话使用的 LOCAL 临时缓冲区的大小。

该参数属于 USERSET 类型参数，请参考表 10-1 中对应设置方法进行设置。

在每个会话的第一次使用临时表之前可以改变 temp_buffers 的值，之后的设置将是无效的。

一个会话将按照 temp_buffers 给出的限制，根据需要分配临时缓冲区。如果在一个并不需要大量临时缓冲区的会话里设置一个大的数值，其开销只是一个缓冲区描述符的大小。当缓冲区被使用，就会额外消耗 8192 字节。

取值范围：整型，100~1073741823，单位为 8KB。

默认值：8MB

max_prepared_transactions

参数说明：设置可以同时处于"预备"状态的事务的最大数目。增加此参数的值会使 Vastbase 比系统默认设置需要更多的 System V 共享内存。

当 Vastbase 部署为主备双机时，在备机上此参数的设置必须要高于或等于主机上的，否则无法在备机上进行查询操作。

该参数属于 POSTMASTER 类型参数，请参考表 10-1 中对应设置方法进行设置。

取值范围：整型。0~536870911。

默认值：800

说明

为避免在准备步骤失败，此参数的值不能小于 max_connections。

work_mem

参数说明：设置内部排序操作和 Hash 表在开始写入临时磁盘文件之前使用的内存大小。ORDER BY，DISTINCT 和 merge joins 都要用到排序操作。Hash 表在散列连接、散列为基础的聚集、散列为基础的 IN 子查询处理中都要用到。

对于复杂的查询，可能会同时并发运行好几个排序或者散列操作，每个都可以使用此参数所声明的内存量，不足时会使用临时文件。同样，好几个正在运行的会话可能会同时进行排序操作。因此使用的总内存可能是 work_mem 的好几倍。

该参数属于 USERSET 类型参数，请参考表 10-1 中对应设置方法进行设置。

取值范围：整型，64~2147483647，单位为 KB。

默认值：64MB

设置建议：

依据查询特点和并发来确定，一旦 work_mem 限定的物理内存不够，算子运算数据将写入临时表空间，带来 5-10 倍的性能下降，查询响应时间从秒级下降到分钟级。

- 对于串行无并发的复杂查询场景，平均每个查询有 5-10 关联操作，建议 work_mem=50%内存/10。
- 对于串行无并发的简单查询场景，平均每个查询有 2-5 个关联操作，建议 work_mem=50%内存/5。
- 对于并发场景，建议 work_mem=串行下的 work_mem/物理并发数。

query_mem

参数说明：设置执行作业所使用的内存。如果设置的 query_mem 值大于 0，在生成执行计划时，优化器会将作业的估算内存调整为该值。

该参数属于 USERSET 类型参数，请参考表 10-1 中对应设置方法进行设置。

取值范围：0，或大于 32M 的整型，默认单位为 KB。如果设置值为负数或小于 32M，将设置为默认值 0，此时优化器不会根据该值调整作业的估算内存。

默认值：0

query_max_mem

参数说明：设置执行作业所能够使用的最大内存。如果设置的 query_max_mem 值大于 0，当作业执行时所使用内存超过该值时，将报错退出。

该参数属于 USERSET 类型参数，请参考表 10-1 中对应设置方法进行设置。

取值范围：0，或大于 32M 的整型，默认单位为 KB。如果设置值为负数或小于 32M，将设置为默认值 0，此时不会根据该值限制作业的内存使用。

默认值：0

maintenance_work_mem

参数说明：设置在维护性操作 (比如 VACUUM、CREATE INDEX、ALTER TABLE ADD FOREIGN KEY 等) 中可使用的最大的内存。该参数的设置会影响 VACUUM、VACUUM FULL、CLUSTER、CREATE INDEX 的执行效率。

该参数属于 USERSET 类型参数，请参考表 10-1 中对应设置方法进行设置。

取值范围：整型，1024~INT_MAX，单位为 KB。

默认值：128MB

设置建议：

- 建议设置此参数的值大于 work_mem，可以改进清理和恢复数据库转储的速度。因为在一个数据库会话里，任意时刻只有一个维护性操作可以执行，并且在执行维护性操作时不会有太多的会话。
- 当 21.13 自动清理进程运行时，autovacuum_max_workers 倍数的内存将会被分配，所以此时设置 maintenance_work_mem 的值应该不小于 work_mem。
- 如果进行大数据量的 cluster 等，可以在 session 中调大该值。

psort_work_mem

参数说明：设置列存表在进行局部排序中在开始写入临时磁盘文件之前使用的内存大小。带 partial cluster key 的表、带索引的表插入，创建表索引，删除表和更新表都会用到。

该参数属于 USERSET 类型参数，请参考表 10-1 中对应设置方法进行设置。

须知

同样，好几个正在运行的会话可能会同时进行表的局部排序操作。因此使用的总内存可能是 psort_work_mem 的好几倍。

取值范围：整型 64~2147483647，单位为 KB。

默认值：512MB

max_loaded_cudesc

参数说明：设置列存表在做扫描时，每列缓存 cudesc 信息的个数。增大设置会提高查询性能，但也会增加内存占用，特别是当列存表的列非常多时。

该参数属于 USERSET 类型参数，请参考表 10-1 中对应设置方法进行设置。

须知

max_loaded_cudesc 设置过高时，有可能引起内存分配不足。

取值范围：100~1073741823。

默认值：1024

max_stack_depth

参数说明：设置 Vastbase 执行堆栈的最大安全深度。需要这个安全界限是因为在服务器里，并非所有程序都检查了堆栈深度，只是在可能递规的过程，比如表达式计算这样的过程里面才进行检查。

该参数属于 SUSET 类型参数，请参考表 10-1 中对应设置方法进行设置。

设置原则：

- 数据库需要预留 640KB 堆栈深度，因此此参数的最佳设置是等于操作系统内核允许的最大值（就是 ulimit -s 的设置）- 640KB。
- 如果设置此参数的值大于实际的内核限制，则一个正在运行的递归函数可能会导致一个独立的服务器进程崩溃。在 Vastbase 能够检测内核限制的操作系统上（SLES 上），将自动限制设置为一个不安全的值。
- 因为并非所有的操作都能够检测，所以建议用户在此设置一个明确的值。

取值范围：整型，100~INT_MAX，单位为 KB。

默认值：2MB

说明

默认值 2MB，这个值相对比较小，不容易导致系统崩溃。但是可能会因为该值较小，导致无法执行复杂的函数。

cstore_buffers

参数说明：设置列存所使用的共享缓冲区的大小。

该参数属于 POSTMASTER 类型参数，请参考表 10-1 中对应设置方法进行设置。

取值范围：整型，16384～1073741823，单位为 KB。

默认值：32768KB

设置建议：

列存表使用 cstore_buffers 设置的共享缓冲区，几乎不用 shared_buffers。因此在列存表为主的场景中，应减少 shared_buffers，增加 cstore_buffers。

bulk_read_ring_size

参数说明：并行导出，使用的环形缓冲区大小。

该参数属于 USERSET 类型参数，请参考表 10-1 中对应设置方法进行设置。

取值范围：整型，256~2147483647，单位为 KB。

默认值：16MB

19.4.2. 磁盘空间

介绍与磁盘空间相关的参数，用于限制临时文件所占用的磁盘空间。

sql_use_spacelimit

参数说明：限制单个 SQL 在单个数据库节点上，触发落盘操作时，落盘文件的空间大小，管控的空间包扩普通表、临时表以及中间结果集落盘占用的空间。

该参数属于 USERSET 类型参数，请参考表 10-1 中对应设置方法进行设置

取值范围：整型，-1~2147483647，单位为 KB。其中-1 表示没有限制。

默认值：**-1**

temp_file_limit

参数说明：限制一个会话中，触发下盘操作时，单个下盘文件的空间大小。例如一次会话中，排序和哈希表使用的临时文件，或者游标占用的临时文件。

此设置为会话级别的下盘文件控制。

该参数属于 SUSET 类型参数，请参考表 10-1 中对应设置方法进行设置。

须知

SQL 查询执行时使用的临时表空间不在此限制。

取值范围：整型，-1~2147483647，单位为 KB。其中-1 表示没有限制。

默认值：-1

19.4.3. 内核资源使用

介绍与操作系统内核相关的参数，这些参数是否生效依赖于操作系统的设置。

max_files_per_process

参数说明：设置每个服务器进程允许同时打开的最大文件数目。如果操作系统内核强制一个合理的数目，则不需要设置。

但是在一些平台上（特别是大多数 BSD 系统），内核允许独立进程打开比系统真正可以支持的数目大得多得文件数。如果用户发现有的“Too many open files”这样的失败现象，请尝试缩小这个设置。通常情况下需要满足，系统 FD（file descriptor）数量>=最大并发数*数据库节点个数*max_files_per_process*3。

该参数属于 POSTMASTER 类型参数，请参考表 10-1 中对应设置方法进行设置。

取值范围：整型，25~2147483647。

默认值：1000

shared_preload_libraries

参数说明：此参数用于声明一个或者多个在服务器启动的时候预先装载的共享库，多个库名称之间用逗号分隔。比如 '$libdir/mylib' 会在加载标准库目录中的库文件之前预先加载 mylib.so（某些平台上可能是 mylib.sl）库文件。

可以用这个方法预先装载 Vastbase 的存储过程库，通常是使用 '$libdir/plXXX' 语法。XXX 只能是 pgsql，perl，tcl，python 之一。

通过预先装载一个共享库并在需要的时候初始化它，可以避免第一次使用这个库的加载时间。但是启动每个服务器进程的时间可能会增加，即使进程从来没有使用过这些库。因此建议对那些将被大多数会话使用的库才使用这个选项。

该参数属于 POSTMASTER 类型参数，请参考表 10-1 中对应设置方法进行设置。

须知

- 如果被声明的库不存在，Vastbase 服务将会启动失败。
- 每一个支持 Vastbase 的库都有一个特殊的标记用于保证兼容性。因此，不支持 Vastbase 的库不能用这种方法加载。

取值范围：字符串

默认值：空

19.4.4. 基于开销的清理延迟

这个特性的目的是允许管理员减少 VACUUM 和 ANALYZE 语句在并发活动的数据库上的 I/O 影响。比如，像 VACUUM 和 ANALYZE 这样的维护语句并不需要迅速完成，并且不希望他们严重干扰系统执行其他的数据库操作。基于开销的清理延迟为管理员提供了一个实现这个目的手段。

须知

有些清理操作会持有关键的锁，这些操作应该尽快结束并释放锁。所以 Vastbase 的机制是，在这类操作过程中，基于开销的清理延迟不会发生作用。为了避免在这种情况下的长延时，实际的开销限制取下面两者之间的较大值：

- vacuum_cost_delay*accumulated_balance/vacuum_cost_limit
- vacuum_cost_delay*4

背景信息

在 11.16.26ANALYZE | ANALYSE 和 11.16.114VACUUM 语句执行过程中，系统维护一个内部的记数器，跟踪所执行的各种 I/O 操作的近似开销。如果积累的开销达到了 vacuum_cost_limit 声明的限制，则执行这个操作的进程将睡眠 vacuum_cost_delay 指定的时间。然后它会重置记数器然后继续执行。

这个特性是缺省关闭的。要想打开它，把 vacuum_cost_delay 变量设置为一个非零值。

vacuum_cost_delay

参数说明：指定开销超过 vacuum_cost_limit 的值时，进程睡眠的时间。

要注意在许多系统上，睡眠的有效分辨率是 10 毫秒。因此把 vacuum_cost_delay 设置为一个不是 10 的整数倍的数值与将它设置为下一个 10 的整数倍作用相同。

此参数一般设置较小，常见的设置是 10 或 20 毫秒。调整此特性资源占用率时，最好是调整其他参数，而不是此参数。

该参数属于 USERSET 类型参数，请参考表 10-1 中对应设置方法进行设置。

取值范围：integer（毫秒），0~100，正数值表示打开基于开销的清理延迟特性；0 表示关闭基于开销的清理延迟特性。

默认值：0

vacuum_cost_page_hit

参数说明：清理一个在共享缓存里找到的缓冲区的预计开销。他代表锁住缓冲池、查找共享的 Hash 表、扫描页面内容的开销。

该参数属于 USERSET 类型参数，请参考表 10-1 中对应设置方法进行设置。

取值范围：integer，0~10000。

默认值：1

vacuum_cost_page_miss

参数说明：清理一个要从磁盘上读取的缓冲区的预计开销。他代表锁住缓冲池、查找共享 Hash 表、从磁盘读取需要的数据块、扫描它的内容的开销。

该参数属于 USERSET 类型参数，请参考表 10-1 中对应设置方法进行设置。

取值范围：integer，0~10000。

默认值：10

vacuum_cost_page_dirty

参数说明：清理修改一个原先是干净的块的预计开销。他代表把一个脏的磁盘块再次刷新到磁盘上的额外开销。

该参数属于 USERSET 类型参数，请参考表 10-1 中对应设置方法进行设置。

取值范围：integer，0~1000

默认值：20

vacuum_cost_limit

参数说明：导致清理进程休眠的开销限制。

该参数属于 USERSET 类型参数，请参考表 10-1 中对应设置方法进行设置。

取值范围：integer，1~10000。

默认值：200

19.4.5. 后端写进程

介绍后端写（background writer）进程的参数配置。后端写进程的功能就是把共享缓冲区中的脏数据（指共享缓冲区中新增或者修改的内容）写入到磁盘。目的是让数据库进程在进行用户查询时可以很少或者几乎不等待写动作的发生（写动作由后端写进程完成）。

此机制同样也减少了检查点造成的性能下降。后端写进程将持续的把脏页面刷新到磁盘上，所以在检查点到来的时候，只有几个页面需要刷新到磁盘上。但是这样还是增加了 I/O 的总净负荷，因为以前的检查点间隔里，一个重复弄脏的页面可能只会冲刷一次，而同一个间隔里，后端写进程可能会写好几次。在大多数情况下，连续的低负荷要比周期性的尖峰负荷好，但是在本节讨论的参数可以用于按实际需要调节其行为。

bgwriter_delay

参数说明：设置后端写进程写"脏"共享缓冲区之间的时间间隔。每一次，后端写进程都会为一些脏的缓冲区发出写操作（用 bgwriter_lru_maxpages 参数控制每次写的量），然后休眠 bgwriter_delay 毫秒后才再次启动。

在许多系统上，休眠延时的有效分辨率是 10 毫秒。因此，设置一个不是 10 的倍数的数值与把它设置为下一个 10 的倍数是一样的效果。

该参数属于 SIGHUP 类型参数，请参考表 10-1 中对应设置方法进行设置。

取值范围：整型，10~10000，单位为毫秒。

默认值：10s

设置建议：在数据写压力比较大的场景中可以尝试减小该值以降低 checkpoint 的压力。

bgwriter_lru_maxpages

参数说明：设置后端写进程每次可写入磁盘的“脏”缓存区的个数。

该参数属于 SIGHUP 类型参数，请参考表 10-1 中对应设置方法进行设置。

取值范围：整型，0 ~ 1000

说明

此参数设置为 0 表示禁用后端写功能，禁用后端写功能不会对 checkpoints 产生影响。

默认值：100

bgwriter_lru_multiplier

参数说明：通过与已使用缓存区数目的乘积评估下次服务器需要的缓存区数目。

写“脏”缓存区到磁盘的数目取决于服务器最近几次使用的缓存区数目。最近的 buffers 数目的平均值乘以 bgwriter_lru_multiplier 是为了评估下次服务器进程需要的 buffers 数目。在有足够多的干净的、可用的缓存区之前，后端写进程会一直写“脏”缓存区的（每次写的缓存区数目不会超过 bgwriter_lru_maxpages 的值）。

设置 bgwriter_lru_multiplier 的值为 1.0 表示一种“实时”策略，其作用是精准预测下次写“脏”缓冲区的数目。设置为较大的值可以应对突然的需求高峰，而较小的值则可以让服务器进程执行更多的写操作。

设置较小的 bgwriter_lru_maxpages 和 bgwriter_lru_multiplier 会减小后端写进程导致的额外 I/O 开销，但是服务器进程必须自己发出写操作，增加了对查询的响应时间。

该参数属于 SIGHUP 类型参数，请参考表 10-1 中对应设置方法进行设置。

取值范围：浮点型，0~10。

默认值：2

19.4.6. 异步 IO

enable_adio_debug

参数说明：允许维护人员输出一些与 ADIO 相关的日志，便于定位 ADIO 相关问题。开发人员专用，不建议普通用户使用。

该参数属于 SUSET 类型参数，请参考表 10-1 中对应设置方法进行设置。

取值范围：布尔型

- on/true 表示开启此日志开关。
- off/false 表示关闭此日志开关。

默认值：off

enable_adio_function

参数说明：是否开起 ADIO 功能。

该参数属于 POSTMASTER 类型参数，请参考表 10-1 中对应设置方法进行设置。

取值范围：布尔型

- on/true 表示开启此功能。
- off/false 表示关闭此功能。

默认值：off

enable_fast_allocate

参数说明：磁盘空间快速分配开关。

该参数属于 SUSET 类型参数，请参考表 10-1 中对应设置方法进行设置。只有在 XFS 文件系统上才能开启该开关。

取值范围：布尔型

- on/true 表示开启此功能。
- off/false 表示关闭此功能。

默认值：off

prefetch_quantity

参数说明：描述行存储使用 ADIO 预读取 IO 量的大小。

该参数属于 USERSET 类型参数，请参考表 10-1 中对应设置方法进行设置。

取值范围：整型，128～131072，单位为 8KB。

默认值：32MB (4096 * 8KB)

backwrite_quantity

参数说明：描述行存储使用 ADIO 写入 IO 量的大小。

该参数属于 USERSET 类型参数，请参考表 10-1 中对应设置方法进行设置。

取值范围：整型，128～131072，单位为 8KB。

默认值：8MB (1024 * 8KB)

cstore_prefetch_quantity

参数说明：描述列存储使用 ADIO 预取 IO 量的大小。

该参数属于 USERSET 类型参数，请参考表 10-1 中对应设置方法进行设置。

取值范围：整型，1024～1048576，单位为 KB。

默认值：32MB

cstore_backwrite_quantity

参数说明：描述列存储使用 ADIO 写入 IO 量的大小。

该参数属于 USERSET 类型参数，请参考表 10-1 中对应设置方法进行设置。

取值范围：整型，1024～1048576，单位为 KB。

默认值：8MB

cstore_backwrite_max_threshold

参数说明：描述列存储使用 ADIO 写入数据库可缓存最大的 IO 量。

该参数属于 USERSET 类型参数，请参考表 10-1 中对应设置方法进行设置。

取值范围：整型，4096～INT_MAX/2，单位为 KB。

默认值：2GB

fast_extend_file_size

参数说明：描述列存储使用 ADIO 预扩展磁盘的大小。

该参数属于 SUSET 类型参数，请参考表 10-1 中对应设置方法进行设置。

取值范围：整型，1024～1048576，单位为 KB。

默认值：8MB

effective_io_concurrency

参数说明：磁盘子系统可以同时有效处理的请求数。对于 RAID 阵列，此参数应该是阵列中驱动器主轴的数量。

该参数属于 USERSET 类型参数，请参考表 10-1 中对应设置方法进行设置。

取值范围：整型 ，0~1000

默认值：1

19.5. 并行导入

Vastbase 提供了并行导入功能，以快速、高效地完成大量数据导入。介绍 Vastbase 并行导入的相关参数。

raise_errors_if_no_files

参数说明：导入时是否区分“导入文件记录数为空”和“导入文件不存在”。raise_errors_if_no_files=TRUE，则“导入文件不存在”的时候，Vastbase 将抛出“文件不存在的”错误。

该参数属于 SUSET 类型参数，请参考表 10-1 中对应设置方法进行设置。

取值范围：布尔型

- on 表示导入时区分“导入文件记录数为空”和“导入文件不存在”。
- off 表示导入时不区分“导入文件记录数为空”和“导入文件不存在”。

默认值：off

partition_mem_batch

参数说明：为了优化对列存分区表的批量插入，在批量插入过程中会对数据进行缓存后再批量写盘。通过 partition_mem_batch 可指定缓存个数。该值设置过大，将消耗较多系统内存资源；设置过小，将降低系统列存分区表批量插入性能。

该参数属于 USERSET 类型参数，请参考表 10-1 中对应设置方法进行设置。

取值范围：1~ 65535

默认值：256

partition_max_cache_size

参数说明：为了优化对列存分区表的批量插入，在批量插入过程中会对数据进行缓存后再批量写盘。通过 partition_max_cache_size 可指定数据缓存区大小。该值设置过大，将消耗较多系统内存资源；设置过小，将降低列存分区表批量插入性能。

该参数属于 USERSET 类型参数，请参考表 10-1 中对应设置方法进行设置。

取值范围：

列存分区表：4096~ INT_MAX / 2，最小单位为 KB。

默认值：2GB

enable_delta_store

参数说明：为了增强列存单条数据导入的性能和解决磁盘冗余问题，可通过此参数选择是否开启支持列存 delta 表功能。该参数开启时，数据导入列存表，会根据表定义时指定的 DELTAROW_THRESHOLD 决定数据进入 delta 表存储还是主表 CU 存储，当数据量小于 DELTAROW_THRESHOLD 时，数据进入 delta 表。该参数影响的操作包括 insert，copy，vacuum，vacuum full，vacuum deltamerge 重分布等所有涉及列存表数据移动的操作。

该参数属于 POSTMASTER 类型参数，请参考表 10-1 中对应设置方法进行设置。

取值范围：

- on 表示开启列存 delta 表功能。
- off 表示不开启列存 delta 表功能。

默认值：off

19.6. 预写式日志

19.6.1. 设置

wal_level

参数说明：设置写入 WAL 信息量的级别。

该参数属于 POSTMASTER 类型参数，请参考表 10-1 中对应设置方法进行设置。

须知

- 如果需要启用 WAL 日志归档和主备机的数据流复制，必须将此参数设置为 archive 或者 hot_standby。
- 如果此参数设置为 minimal, archive_mode 必须设置为 off, hot_standby 必须设置为 off，max_wal_senders 参数设置为 0，且需为单机环境，否则将导致数据库无法启动。
- 如果此参数设置为 archive，hot_standby 必须设置为 off，否则将导致数据库无法启动。但是，hot_standby 在双机环境中不能设置为 off，具体参见 hot_standby 参数说明。

取值范围：枚举类型

- minimal

  优点：一些重要操作（包括创建表、创建索引、簇操作和表的复制）都能安全的跳过，这样就可以使操作变得更快。

  缺点：WAL 仅提供从数据库服务器崩溃或者紧急关闭状态恢复时所需要的基本信息，无法用 WAL 归档日志恢复数据。

- archive

这个参数增加了 WAL 归档需要的日志信息，从而可以支持数据库的归档恢复。

- hot_standby
  - 这个参数进一步增加了在备机上运行的 SQL 查询的信息，这个参数只能在数据库服务重新启动后生效。
  - 为了在备机上开启只读查询，wal_level 必须在主机上设置成 hot_standby ，并且备机必须打开 hot_standby 参数。hot_standby 和 archive 级别之间的性能只有微小的差异，如果它们的设置对产品的性能影响有明显差异，欢迎反馈。

默认值：hot_standby

fsync

参数说明：设置 Vastbase 服务器是否使用 fsync()系统函数（请参见 wal_sync_method）确保数据的更新及时写入物理磁盘中。

该参数属于 SIGHUP 类型参数，请参考表 10-1 中对应设置方法进行设置。

须知

- 使用 fsync()系统函数可以保证在操作系统或者硬件崩溃的情况下将数据恢复到一个已知的状态。
- 如果将此参数关闭，可能会在系统崩溃时无法恢复原来的数据，导致数据库不可用。

取值范围：布尔型

- on 表示使用 fsync()系统函数。
- off 表示不使用 fsync()系统函数。

默认值：on

synchronous_commit

参数说明：设置当前事务的同步方式。

该参数属于 USERSET 类型参数，请参考表 10-1 中对应设置方法进行设置。

取值范围：枚举类型

- on 表示将备机的同步日志刷新到磁盘。
- off 表示异步提交。
- local 表示为本地提交。
- remote_write 表示要备机的同步日志写到磁盘。
- remote_receive 表示要备机同步日志接收数据。

默认值：on

wal_sync_method

参数说明：设置向磁盘强制更新 WAL 数据的方法。

该参数属于 SIGHUP 类型参数，请参考表 10-1 中对应设置方法进行设置。

须知

如果将 fsync 关闭，这个参数的设置就没有意义，因为所有数据更新都不会强制写入磁盘。

取值范围：枚举类型

- open_datasync 表示用带 O_DSYNC 选项的 open()打开“WAL”文件。
- fdatasync 表示每次提交的时候都调用 fdatasync()。（支持 suse10 和 suse11）。
- fsync_writethrough 表示每次提交的时候调用 fsync()强制把缓冲区任何数据写入磁盘。

说明

由于历史原因，我们允许在 Windows 平台上将 wal_sync_method 设置为 fsync_writethrough，尽管它和 fsync 等效。

- fsync 表示每次提交的时候调用 fsync()。（支持 suse10 和 suse11）
- open_sync 表示用带 O_SYNC 选项的 open()写“WAL”文件。（支持 suse10 和 suse11）

默认值：fdatasync

full_page_writes

参数说明：设置 Vastbase 服务器在检查点之后对页面的第一次修改时，是否将每个磁盘页面的全部内容写到 WAL 日志中。

该参数属于 SIGHUP 类型参数，请参考表 10-1 中对应设置方法进行设置。

须知

- 设置这个参数是因为在操作系统崩溃过程中可能磁盘页面只写入了一部分内容，从而导致在同一个页面中包含新旧数据的混合。在崩溃后的恢复期间，由于在 WAL 日志中存储的行变化信息不够完整，因此无法完全恢复该页。把完整的页面影像保存下来就可以保证页面被正确还原，代价是增加了写入 WAL 日志的数据量。
- 关闭此参数，在系统崩溃的时候，可能无法恢复原来的数据。如果服务器硬件的特质（比如电池供电的磁盘控制器）可以减小部分页面的写入风险，或者文件系统特性支持（比如 ReiserFS 4），并且清楚知道写入风险在一个可以接受的范畴，可以关闭这个参数。

取值范围：布尔型

- on 表示启用此特性。
- off 表示关闭此特性。

默认值：on

wal_log_hints

参数说明：设置在检查点之后对页面的第一次修改为页面上元组 hint bits 的修改时，是否将整个页面的全部内容写到 WAL 日志中。不推荐用户修改此设置。

该参数属于 POSTMASTER 类型参数，请参考表 10-1 中对应设置方法进行设置。

取值范围：布尔型

- on 表示整个页面全部内容写到 WAL 日志中。
- off 表示整个页面内容不会写到 WAL 日志中。

默认值：on

wal_buffers

参数说明：设置用于存放 WAL 数据的共享内存空间的 XLOG_BLCKSZ 数，XLOG_BLCKSZ 的大小默认为 8KB。

该参数属于 POSTMASTER 类型参数，请参考表 10-1 中对应设置方法进行设置。

取值范围：-1~2^{18}

- 如果设置为-1，表示 wal_buffers 的大小随着参数 shared_buffers 自动调整，为 shared_buffers 的 1/32，最小值为 8 个 XLOG_BLCKSZ，最大值为 2048 个 XLOG_BLCKSZ。
- 如果设置为其他值，当小于 4 时，会被默认设置为 4。

默认值：16MB

设置建议：每次事务提交时，WAL 缓冲区的内容都写入到磁盘中，因此设置为很大的值不会带来明显的性能提升。如果将它设置成几百兆，就可以在有很多即时事务提交的服务器上提高写入磁盘的性能。根据经验来说，默认值可以满足大多数的情况。

wal_writer_delay

参数说明：WalWriter 进程的写间隔时间。

该参数属于 SIGHUP 类型参数，请参考表 10-1 中对应设置方法进行设置。

须知

如果时间过长可能造成 WAL 缓冲区的内存不足，时间过短会引起 WAL 不断写入，增加磁盘 I/O 负担。

取值范围：整型， 1～10000（毫秒）

默认值：200ms

commit_delay

参数说明：表示一个已经提交的数据在 WAL 缓冲区中存放的时间。

该参数属于 USERSET 类型参数，请参考表 10-1 中对应设置方法进行设置。

须知

- 设置为非 0 值时事务执行 commit 后不会立即写入 WAL 中，而仍存放在 WAL 缓冲区中，等待 WalWriter 进程周期性写入磁盘。
- 如果系统负载很高，在延迟时间内，其他事务可能已经准备好提交。但如果没有事务准备提交，这个延迟就是在浪费时间。

取值范围：整型， 0～100000（微秒），其中 0 表示无延迟。

默认值：0

commit_siblings

参数说明：当一个事务发出提交请求时，如果数据库中正在执行的事务数量大于此参数的值，则该事务将等待一段时间（commit_delay 的值），否则该事务则直接写入 WAL。

该参数属于 USERSET 类型参数，请参考表 10-1 中对应设置方法进行设置。

取值范围：整型， 0～1000

默认值：5

wal_block_size

参数说明：说明 WAL 日志段文件中日志页面的大小。

该参数属于 INTERNAL 类型参数，为固定参数，用户无法修改此参数，只能查看。

取值范围：整型，单位为 Byte。

默认值：8192

wal_segment_size

参数说明：说明 WAL 日志段文件的大小。

该参数属于 INTERNAL 类型参数，为固定参数，用户无法修改此参数，只能查看。

取值范围：整型，单位为 8KB。

默认值：16MB (2048 * 8KB)

19.6.2. 检查点

checkpoint_segments

参数说明：设置 checkpoint_timeout 周期内所保留的最少 WAL 日志段文件数量。每个日志文件大小为 16MB。

该参数属于 SIGHUP 类型参数，请参考表 10-1 中对应设置方法进行设置。

取值范围：整型，最小值 1

提升此参数可加快大数据的导入速度，但需要结合 checkpoint_timeout、shared_buffers 这两个参数统一考虑。这个参数同时影响 WAL 日志段文件复用数量，通常情况下 pg_xlog 文件夹下最大的复用文

件个数为 2 倍的 checkpoint_segments 个，复用的文件被改名为后续即将使用的 WAL 日志段文件，不会被真正删除。

默认值：64

checkpoint_timeout

参数说明：设置自动 WAL 检查点之间的最长时间。

该参数属于 SIGHUP 类型参数，请参考表 10-1 中对应设置方法进行设置。

取值范围：整型， 30～3600（秒）

在提升 checkpoint_segments 以加快大数据导入的场景也需将此参数调大，同时这两个参数提升会加大 shared_buffers 的负担，需要综合考虑。

默认值：15min

checkpoint_completion_target

参数说明：指定检查点完成的目标。

该参数属于 SIGHUP 类型参数，请参考表 10-1 中对应设置方法进行设置。

取值范围：0.0～1.0

默认值：0.5

说明

默认值 0.5 表示：每个 checkpoint 需要在 checkpoints 间隔时间的 50%内完成。

checkpoint_warning

参数说明：如果由于填充检查点段文件导致检查点发生的时间间隔接近这个参数表示的秒数，就向服务器日志发送一个建议增加 checkpoint_segments 值的消息。

该参数属于 SIGHUP 类型参数，请参考表 10-1 中对应设置方法进行设置。

取值范围：整型，0~INT_MAX（秒），其中 0 表示关闭警告。

默认值：5min

推荐值：5min

checkpoint_wait_timeout

参数说明：设置请求检查点等待 checkpointer 线程启动的最长时间。

该参数属于 SIGHUP 类型参数，请参考表 10-1 中对应设置方法进行设置。

取值范围：整型，2～3600（秒）

默认值：1min

enable_incremental_checkpoint

参数说明：增量检查点开关。

该参数属于 POSTMASTER 类型参数，请参考表 10-1 中对应设置方法进行设置。

取值范围：布尔型

默认值：on

enable_double_write

参数说明：双写开关，增量检查点开关打开时，不再使用 full_page_writes 防止半页写问题，而是依赖双写特性保护。

该参数属于 POSTMASTER 类型参数，请参考表 10-1 中对应设置方法进行设置。

取值范围：布尔型

默认值：on

incremental_checkpoint_timeout

参数说明：增量检查点开关打开之后，设置自动 WAL 检查点之间的最长时间。

该参数属于 SIGHUP 类型参数，请参考表 10-1 中对应设置方法进行设置。

取值范围：整型，1~3600（秒）

默认值：1min

enable_xlog_prune

参数说明：设置在任一备机断联时，主机是否根据 xlog 日志的大小超过参数 max_size_for_xlog_prune 的值而回收日志。

该参数属于 SIGHUP 类型参数，请参考表 10-1 中对应设置方法进行设置。

取值范围：布尔型

默认值：on

max_size_for_xlog_prune

参数说明：在 enable_xlog_prune 打开时生效，如果有备机断连且 xlog 日志大小大于此阈值，则回收日志。

该参数属于 SIGHUP 类型参数，请参考表 10-1 中对应设置方法进行设置。

取值范围：整型，0 ~ 576 460 752 303 423 487，单位为 MB

默认值：100000，单位 MB

19.6.3. 日志回放

recovery_time_target

参数说明：设置 recovery_time_target 秒能够让备机完成日志写入和回放。

该参数属于 SIGHUP 类型参数，请参考表 10-1 中对应设置方法进行设置。

取值范围：整型，0~3600 （秒）

0 是指不开启日志流控，1~3600 是指备机能够在 recovery_time_target 时间内完成日志的写入和回放，可以保证主机与备机切换时能够在 recovery_time_target 秒完成日志写入和回放，保证备机能够快速升主机。recovery_time_target 设置时间过小会影响主机的性能，设置过大会失去流控效果。

默认值：0

recovery_max_workers

参数说明：设置最大并行回放线程个数。

该参数属于 POSTMASTER 类型参数，请参考表 10-1 中对应设置方法进行设置。

取值范围：整型，0~20

默认值：1

recovery_parse_workers

参数说明：是极致 RTO 特性中 ParseRedoRecord 线程的数量。

该参数属于 POSTMASTER 类型参数，请参考表 10-1 中对应设置方法进行设置。

取值范围：整型，1~16

仅在开启极致 RTO 情况下可以设置 recovery_parse_workers 为>1。需要配合 recovery_redo_workers 使用。若同时开启 recovery_parse_workers 和 recovery_max_workers，以开启极致 RTO 的 recovery_parse_workers 为准，并行回放特性失效。因极致 RTO 不支持 hot standby 模式，仅在关闭 hot standby 时可以设置 recovery_parse_workers 为>1.

默认值：1

recovery_redo_workers

参数说明：是极致 RTO 特性中每个 ParseRedoRecord 线程对应的 PageRedoWorker 数量。

该参数属于 POSTMASTER 类型参数，请参考表 10-1 中对应设置方法进行设置。

取值范围：整型，1~8

需要配合 recovery_parse_workers 使用。recovery_redo_workers 是极致 RTO 特性中每个 ParseRedoRecord 线程对应的 PageRedoWorker 线程数量。在配合 recovery_parse_workers 使用时，只有 recovery_parse_workers 大于 0

recovery_redo_workers 参数才生效。

默认值：1

recovery_parallelism

参数说明：查询实际回放线程个数，该参数为只读参数。

该参数属于 POSTMASTER 类型参数，请参考表 10-1 中对应设置方法进行设置。

取值范围：整型，1~2147483647

默认值：1

enable_page_lsn_check

参数说明：数据页 lsn 检查开关。回放时，检查数据页当前的 lsn 是否是期望的 lsn。

该参数属于 POSTMASTER 类型参数，请参考表 10-1 中对应设置方法进行设置。

取值范围：布尔型

默认值：true

19.6.4. 归档

archive_mode

参数说明：表示是否进行归档操作。

该参数属于 SIGHUP 类型参数，请参考表 10-1 中对应设置方法进行设置。

须知

当 wal_level 设置成 minimal 时，archive_mode 参数无法使用。

取值范围：布尔型

- on 表示进行归档。
- off 表示不进行归档。

默认值：off

archive_command

参数说明：由管理员设置的用于归档 WAL 日志的命令，建议归档路径为绝对路径。

该参数属于 SIGHUP 类型参数，请参考表 10-1 中对应设置方法进行设置。

须知

- 字符串中任何%p 都被要归档的文件的绝对路径代替，而任何%f 都只被该文件名代替（相对路径都相对于数据目录的）。如果需要在命令里嵌入%字符就必须双写%。
- 这个命令当且仅当成功的时候才返回零。示例如下：

```
archive_command = 'cp --remove-destination %p /mnt/server/archivedir/%f'
```

- --remove-destination 选项作用为：拷贝前如果目标文件已存在，会先删除已存在的目标文件，然后执行拷贝操作。

取值范围：字符串

默认值：(disabled)

archive_timeout

参数说明：表示归档周期。

该参数属于 SIGHUP 类型参数，请参考表 10-1 中对应设置方法进行设置。

须知

- 超过该参数设定的时间时强制切换 WAL 段。
- 由于强制切换而提早关闭的归档文件仍然与完整的归档文件长度相同。因此，将 archive_timeout 设为很小的值将导致占用巨大的归档存储空间，建议将 archive_timeout 设置为 60 秒。

取值范围：整型，0 ~ INT_MAX，单位为秒。其中 0 表示禁用该功能。

默认值：0

19.7. 双机复制

19.7.1. 发送端服务器

max_wal_senders

参数说明：指定事务日志发送进程的并发连接最大数量。不可大于等于 max_connections。

该参数属于 POSTMASTER 类型参数，请参考表 10-1 中对应设置方法进行设置。

须知

wal_level 必须设置为 archive 或者 hot_standby 以允许备机的连接。

取值范围：整型，0 ~ 262143

默认值：单机默认为 4，主备环境默认为 8

wal_keep_segments

参数说明：Xlog 日志文件段数量。设置“pg_xlog”目录下保留事务日志文件的最小数目，备机通过获取主机的日志进行流复制。

该参数属于 SIGHUP 类型参数，请参考表 10-1 中对应设置方法进行设置。

取值范围：整型，2 ~ INT_MAX

默认值：65

设置建议：

- 当服务器开启日志归档或者从检查点恢复时，保留的日志文件数量可能大于 wal_keep_segments 设定的值。

❖ 如果此参数设置过小，则在备机请求事务日志时，此事务日志可能已经被产生的新事务日志覆盖，导致请求失败，主备关系断开。

❖ 当双机为异步传输时，以 COPY 方式连续导入 4G 以上数据需要增大 wal_keep_segments 配置。以 T6000 单板为例，如果导入数据量为 50G，建议调整参数为 1000。您可以在导入完成并且日志同步正常后，动态恢复此参数设置。

wal_sender_timeout

参数说明：设置本端等待事务日志接收端接收日志的最大等待时间。

该参数属于 SIGHUP 类型参数，请参考表 10-1 中对应设置方法进行设置。

须知

- 如果主机数据较大，重建操作需要增大此参数的值，主机数据在 500G 时，此参数的参考值为 600s。
- 此值不能大于 wal_receiver_timeout 或数据库重建时的超时参数。

取值范围：整型， 0 ~ INT_MAX，单位为毫秒（ms）。

默认值：6s

replconninfo1

参数说明：设置本端监听和鉴权的第一个节点信息。

该参数属于 SIGHUP 类型参数，请参考表 10-1 中对应设置方法进行设置。

取值范围：字符串。其中空字符串表示没有配置第一个节点信息。

默认值：空字符串

replconninfo2

参数说明：设置本端监听和鉴权的第二个节点信息。

该参数属于 SIGHUP 类型参数，请参考表 10-1 中对应设置方法进行设置。

取值范围：字符串。其中空字符串表示没有配置第二个节点信息。

默认值：空字符串

replconninfo3

参数说明：设置本端监听和鉴权的第三个节点信息。

该参数属于 SIGHUP 类型参数，请参考表 10-1 中对应设置方法进行设置。

取值范围：字符串。其中空字符串表示没有配置第三个节点信息。

默认值：空字符串

replconninfo4

参数说明：设置本端监听和鉴权的第四个节点信息。

该参数属于 SIGHUP 类型参数，请参考表 10-1 中对应设置方法进行设置。

取值范围：字符串。其中空字符串表示没有配置第四个节点信息。

默认值：空字符串

replconninfo5

参数说明：设置本端监听和鉴权的第五个节点信息。

该参数属于 SIGHUP 类型参数，请参考表 10-1 中对应设置方法进行设置。

取值范围：字符串。其中空字符串表示没有配置第五个节点信息。

默认值：空字符串

replconninfo6

参数说明：设置本端监听和鉴权的第六个节点信息。

该参数属于 SIGHUP 类型参数，请参考表 10-1 中对应设置方法进行设置。

取值范围：字符串。其中空字符串表示没有配置第六个节点信息。

默认值：空字符串

replconninfo7

参数说明：设置本端监听和鉴权的第七个节点信息。

该参数属于 SIGHUP 类型参数，请参考表 10-1 中对应设置方法进行设置。

取值范围：字符串。其中空字符串表示没有配置第七个节点信息。

默认值：空字符串

19.7.2. 主服务器

synchronous_standby_names

参数说明：潜在同步复制的备机名称列表，每个名称用逗号分隔。

该参数属于 SIGHUP 类型参数，请参考表 10-1 中对应设置方法进行设置。

须知

- 当前连接的同步备机是列表中的第一个名称。如果当前同步备机失去连接，则它会立即更换下一个优先级更高的备机，并将此备机的名称放入列表中。
- 备机名称可以通过设置环境变量 PGAPPNAME 指定。

取值范围：字符串。当取值为*，表示不启用同步复制。

默认值：*

most_available_sync

参数说明：指定在备机同步失败时，是否阻塞主机。

该参数属于 SIGHUP 类型参数，请参考表 10-1 中对应设置方法进行设置。

取值范围：布尔型

- on 表示在备机同步失败时，不阻塞主机。
- off 表示在备机同步失败时，阻塞主机。

默认值：off

enable_stream_replication

参数说明：控制主备、主从是否进行数据和日志同步。

该参数属于 SIGHUP 类型参数，请参考表 10-1 中对应设置方法进行设置。

须知

- 此参数属于性能测试参数，用于测试带有备机和不带备机的性能参数。关闭参数后，不能进行切换、故障等异常场景测试，否则会出现主备从不一致的情况。
- 此参数属于受控参数，不建议正常业务场景下关闭此参数。

取值范围：布尔型

- on 表示打开主备、主从同步。
- off 表示关闭主备、主从同步。

默认值：on

enable_mix_replication

参数说明：控制主备、主从之间 WAL 日志及数据复制的方式。

该参数属于 INTERNAL 类型参数，默认值为 off，不允许外部修改。

须知

此参数属于内部参数，目前不允许正常业务场景下改变其值，即关闭 WAL 日志、数据页混合复制模式。

取值范围：布尔型

- on 表示打开 WAL 日志、数据页混合复制模式。
- off 表示关闭 WAL 日志、数据页混合复制模式。

默认值：off

vacuum_defer_cleanup_age

参数说明：指定 VACUUM 使用的事务数，VACUUM 会延迟清除无效的行存表记录，延迟的事务个数通过 vacuum_defer_cleanup_age 进行设置。即 VACUUM 和 VACUUM FULL 操作不会立即清理刚刚被删除元组。

该参数属于 SIGHUP 类型参数，请参考表 10-1 中对应设置方法进行设置。

取值范围：整型，0～1000000，值为 0 表示不延迟。

默认值：0

data_replicate_buffer_size

参数说明：发送端与接收端传递数据页时，队列占用内存的大小。此参数会影响主备之间复制的缓冲大小。

该参数属于 POSTMASTER 类型参数，请参考表 10-1 中对应设置方法进行设置。

取值范围：整型，4096~1047552，单位为 KB。

默认值：128MB（即 131072KB）

walsender_max_send_size

参数说明：设置主机端日志或数据发送缓冲区的大小。

该参数属于 POSTMASTER 类型参数，请参考表 10-1 中对应设置方法进行设置。

取值范围：整型，8~INT_MAX，单位为 KB。

默认值：8M（即 8192KB）

enable_data_replicate

参数说明：当数据库在数据导入行存表时，主机与备机的数据同步方式可以进行选择。

该参数属于 USERSET 类型参数，请参考表 10-1 中对应设置方法进行设置。

取值范围：布尔型

- on 表示导入数据行存表时主备数据采用数据页的方式进行同步。当 replication_type 参数为 1 时，不允许设置为 on，如果此时用 guc 工具设置成 on，会强制改为 off。
- off 表示导入数据行存表时主备数据采用日志（Xlog）方式进行同步。

默认值：off

ha_module_debug

参数说明：用于查看数据复制时具体数据块的复制状态日志。

该参数属于 USERSET 类型参数，请参考表 10-1 中对应设置方法进行设置。

取值范围：布尔型

- on 表示日志中将打印数据复制时每个数据块的状态。
- off 表示日志中不打印数据复制时每个数据块的状态。

默认值：off

enable_incremental_catchup

参数说明：控制主备之间数据追赶（catchup）的方式。

该参数属于 SIGHUP 类型参数，请参考表 10-1 中对应设置方法进行设置。

取值范围：布尔型

- on 表示备机 catchup 时用增量 catchup 方式，即从从备本地数据文件扫描获得主备差异数据文件列表，进行主备之间的 catchup。
- off 表示备机 catchup 时用全量 catchup 方式，即从主机本地所有数据文件扫描获得主备差异数据文件列表，进行主备之间的 catchup。

默认值：on

wait_dummy_time

参数说明：同时控制增量数据追赶（catchup）时，Vastbase 主备从按顺序启动时等待从备启动的最长时间以及等待从备发回扫描列表的最长时间。

该参数属于 SIGHUP 类型参数，请参考表 10-1 中对应设置方法进行设置。

取值范围：整型，范围 1~INT_MAX，单位为秒

默认值：300

说明

单位只能设置为秒。

19.7.3. 备服务器

hot_standby

参数说明：设置是否允许备机在恢复过程中连接和查询。

该参数属于 POSTMASTER 类型参数，请参考表 10-1 中对应设置方法进行设置。

须知

- 如果此参数设置为 on，wal_level 必须设置为 hot_standby，否则将导致数据库无法启动。
- 在双机环境中，因为会对双机其他一些功能产生影响，hot_standby 参数不能设置成 off。

取值范围：布尔型

- on 表示允许备机在恢复过程中连接和查询。
- off 表示不允许备机在恢复过程中连接和查询。

默认值：on

max_standby_archive_delay

参数说明：当开启双机热备模式时，如果备机正处理归档 WAL 日志数据，这时进行查询就会产生冲突，此参数就是设置备机取消查询之前所等待的时间。

该参数属于 SIGHUP 类型参数，请参考表 10-1 中对应设置方法进行设置。

须知

-1 表示允许备机一直等待冲突的查询完成。

取值范围：整型，范围：-1~INT_MAX，单位为毫秒。

默认值：3s（**即** 3000ms）

max_standby_streaming_delay

参数说明：当开启双机热备模式时，如果备机正通过流复制接收 WAL 日志数据，这时进行查询就会产生冲突，这个参数就是设置备机取消查询之前所等待的时间。

该参数属于 SIGHUP 类型参数，请参考表 10-1 中对应设置方法进行设置。

须知

-1 表示允许备机一直等待冲突的查询完成。

取值范围：整型（毫秒），范围：-1~INT_MAX

默认值：**3s（即** 3000ms）

wal_receiver_status_interval

参数说明：设置 WAL 日志接收进程的状态通知给主机的最大时间间隔。

该参数属于 SIGHUP 类型参数，请参考表 10-1 中对应设置方法进行设置。

取值范围：整型，范围：0 ~ INT_MAX ，单位为毫秒。

默认值：5s（**即** 5000ms）

hot_standby_feedback

参数说明：设置是否允许将备机上执行查询的结果反馈给主机，这可以避免查询冲突。

该参数属于 SIGHUP 类型参数，请参考表 10-1 中对应设置方法进行设置。

取值范围：布尔型

- on 表示允许将备机上执行查询的最小事务号反馈给主机。
- off 表示不允许将备机上执行查询的最小事务号反馈给主机。

默认值：off

须知

当该参数为 on 时，主机的旧版本数据的清理会受限于备机正在读的事务，即主机只允许清理小于备机反馈回来的事务所作的更改。

所以，若该参数开启时，会影响主机的性能。

wal_receiver_timeout

参数说明：设置从主机接收数据的最大等待时间。

该参数属于 SIGHUP 类型参数，请参考表 10-1 中对应设置方法进行设置。

取值范围：整型， 0 ~ INT_MAX，单位为毫秒。

默认值：6s（即 6000ms）

wal_receiver_connect_timeout

参数说明：设置连接主机的最大等待超时时间。

该参数属于 SIGHUP 类型参数，请参考表 10-1 中对应设置方法进行设置。

取值范围：整型， 0 ~ INT_MAX / 1000，单位为秒。

默认值：2s

wal_receiver_connect_retries

参数说明：设置连接主机的最大尝试次数。

该参数属于 SIGHUP 类型参数，请参考表 10-1 中对应设置方法进行设置。

取值范围：整型， 1~ INT_MAX。

默认值：1

wal_receiver_buffer_size

参数说明：备机与从备接收 Xlog 存放到内存缓冲区的大小。

该参数属于 POSTMASTER 类型参数，请参考表 10-1 中对应设置方法进行设置。

取值范围：整型，4096~1047552，单位为 KB。

默认值：64MB（即 65536KB）

primary_slotname

参数说明：设置备机对应主机的 slot name，用于主备校验，与 wal 日志删除机制。

该参数属于 SIGHUP 类型参数，请参考表 10-1 中对应设置方法进行设置。

取值范围：字符型

默认值：空字符串

19.8. 查询规划

介绍查询优化器方法配置、开销常量、规划算法以及一些配置参数。

说明

优化器中涉及的两个参数：

- INT_MAX 数据类型 INT 的最大值，其值为 2147483647。
- DBL_MAX 数据类型 FLOAT 的最大值。

19.8.1. 优化器方法配置

这些配置参数提供了影响查询优化器选择查询规划的原始方法。如果优化器为特定的查询选择的缺省规划并不是最优的，可以通过使用这些配置参数强制优化器选择一个不同的规划来临时解决这个问题。更好的方法包括调节优化器开销常量、手动运行 ANALYZE、增加配置参数 default_statistics_target 的值、增加使用 ALTER TABLE SET STATISTICS 为指定列增加收集的统计信息。

enable_bitmapscan

参数说明：控制优化器对位图扫描规划类型的使用。

该参数属于 USERSET 类型参数，请参考表 10-1 中对应设置方法进行设置。

取值范围：布尔型

- on 表示使用。
- off 表示不使用。

默认值：on

force_bitmapand

参数说明：控制优化器强制使用 bitmapand 规划类型的使用。

该参数属于 USERSET 类型参数，请参考表 10-1 中对应设置方法进行设置。

取值范围：布尔型

- on 表示使用。
- off 表示不使用。

默认值：off

enable_hashagg

参数说明：控制优化器对 Hash 聚集规划类型的使用。

该参数属于 USERSET 类型参数，请参考表 10-1 中对应设置方法进行设置。

取值范围：布尔型

- on 表示使用。

- off 表示不使用。

默认值：on

enable_hashjoin

参数说明：控制优化器对 Hash 连接规划类型的使用。

该参数属于 USERSET 类型参数，请参考表 10-1 中对应设置方法进行设置。

取值范围：布尔型

- on 表示使用。
- off 表示不使用。

默认值：on

enable_indexscan

参数说明：控制优化器对索引扫描规划类型的使用。

该参数属于 USERSET 类型参数，请参考表 10-1 中对应设置方法进行设置。

取值范围：布尔型

- on 表示使用。
- off 表示不使用。

默认值：on

enable_indexonlyscan

参数说明：控制优化器对仅索引扫描规划类型的使用。

该参数属于 USERSET 类型参数，请参考表 10-1 中对应设置方法进行设置。

取值范围：布尔型

- on 表示使用。
- off 表示不使用。

默认值：on

enable_material

参数说明：控制优化器对实体化的使用。消除整个实体化是不可能的，但是可以关闭这个变量以防止优化器插入实体节点。

该参数属于 USERSET 类型参数，请参考表 10-1 中对应设置方法进行设置。

取值范围：布尔型

- on 表示使用。
- off 表示不使用。

默认值：on

enable_mergejoin

参数说明：控制优化器对融合连接规划类型的使用。

该参数属于 USERSET 类型参数，请参考表 10-1 中对应设置方法进行设置。

取值范围：布尔型

- on 表示使用。
- off 表示不使用。

默认值：off

enable_nestloop

参数说明：控制优化器对内表全表扫描嵌套循环连接规划类型的使用。完全消除嵌套循环连接是不可能的，但是关闭这个变量就会让优化器在存在其他方法的时候优先选择其他方法。

该参数属于 USERSET 类型参数，请参考表 10-1 中对应设置方法进行设置。

取值范围：布尔型

- on 表示使用。
- off 表示不使用。

默认值：off

enable_index_nestloop

参数说明：控制优化器对内表参数化索引扫描嵌套循环连接规划类型的使用。

该参数属于 USERSET 类型参数，请参考表 10-1 中对应设置方法进行设置。

取值范围：布尔型

- on 表示使用。
- off 表示不使用。

默认值：on

enable_seqscan

参数说明：控制优化器对顺序扫描规划类型的使用。完全消除顺序扫描是不可能的，但是关闭这个变量会让优化器在存在其他方法的时候优先选择其他方法。

该参数属于 USERSET 类型参数，请参考表 10-1 中对应设置方法进行设置。

取值范围：布尔型

- on 表示使用。
- off 表示不使用。

默认值：on

enable_sort

参数说明：控制优化器使用的排序步骤。完全消除明确的排序是不可能的，但是关闭这个变量可以让优化器在存在其他方法的时候优先选择其他方法。

该参数属于 USERSET 类型参数，请参考表 10-1 中对应设置方法进行设置。

取值范围：布尔型

- on 表示使用。
- off 表示不使用。

默认值：on

enable_tidscan

参数说明：控制优化器对 TID 扫描规划类型的使用。

该参数属于 USERSET 类型参数，请参考表 10-1 中对应设置方法进行设置。

取值范围：布尔型

- on 表示使用。
- off 表示不使用。

默认值：on

enable_kill_query

参数说明：CASCADE 模式删除用户时，会删除此用户拥有的所有对象。此参数标识是否允许在删除用户的时候，取消锁定此用户所属对象的 query。

该参数属于 SUSET 类型参数，请参考表 10-1 中对应设置方法进行设置。

取值范围：布尔型

- on 表示允许取消锁定。
- off 表示不允许取消锁定。

默认值：off

enforce_a_behavior

参数说明：控制正则表达式的规则匹配模式。

该参数属于 USERSET 类型参数，请参考表 10-1 中对应设置方法进行设置。

取值范围：布尔型

- on 表示正则表达式采用 A 格式的匹配规则。
- off 表示正则表达式采用 POSIX 格式的匹配规则。

默认值：on

max_recursive_times

参数说明：控制 with recursive 的最大迭代次数。

该参数属于 USERSET 类型参数，请参考表 10-1 中对应设置方法进行设置。

取值范围：整型，0～INT_MAX。

默认值：200

enable_vector_engine

参数说明：控制优化器对向量化执行引擎的使用。

该参数属于 USERSET 类型参数，请参考表 10-1 中对应设置方法进行设置。

取值范围：布尔型

- on 表示使用。
- off 表示不使用。

默认值：on

enable_broadcast

参数说明：控制优化器对 stream 代价估算时对 broadcast 分布方式的使用。

该参数属于 USERSET 类型参数，请参考表 10-1 中对应设置方法进行设置。

取值范围：布尔型

- on 表示使用。
- off 表示不使用。

默认值：on

enable_change_hjcost

参数说明：控制优化器在 Hash Join 代价估算路径选择时，是否使用将内表运行时代价排除在 Hash Join 节点运行时代价外的估算方式。如果使用，则有利于选择条数少，但运行代价大的表做内表。

该参数属于 SUSET 类型参数，请参考表 10-1 中对应设置方法进行设置。

取值范围：布尔型

- on 表示使用。
- off 表示不使用。

默认值：off

enable_absolute_tablespace

参数说明：控制表空间是否可以使用绝对路径。

该参数属于 USERSET 类型参数，请参考表 10-1 中对应设置方法进行设置。

取值范围：布尔型

- on 表示可以使用绝对路径。
- off 表示不可以使用绝对路径。

默认值：on

enable_valuepartition_pruning

参数说明：是否对 DFS 分区表进行静态/动态优化。

该参数属于 USERSET 类型参数，请参考表 10-1 中对应设置方法进行设置。

取值范围：布尔型

- on 表示对 DFS 分区表进行静态/动态优化。
- off 表示不对 DFS 分区表进行静态/动态优化。

默认值：on

expected_computing_nodegroup

参数说明：标识选定的计算 Node Group 模式或目标计算 Node Group。Node Group 目前为内部用机制，用户无需设置。

共 4 种计算 Node Group 模式，用于关联操作和聚集操作时选定计算 Node Group。在每一种模式中，优化器有针对性地选定几个候选计算 Node Group，然后根据代价，从中为当前算子挑选最佳计算 Node Group。

该参数属于 USERSET 类型参数，请参考表 10-1 中对应设置方法进行设置。

取值范围：字符串

- optimal：候选计算 Node Group 列表包含算子操作对象所在的 Node Group 和由当前用户具有 COMPUTE 权限的所有 Node Group 包含的数据库节点构成的 Node Group
- query：候选计算 Node Group 列表包含算子操作对象所在的 Node Group 和由当前查询涉及的所有基表所在 Node Group 包含的数据库节点构成的 Node Group
- Node Group 名（enable_nodegroup_debug 被设置为 off）：候选计算 Node Group 列表包含算子操作对象所在的 Node Group 和该指定的 Node Group
- Node Group 名（enable_nodegroup_debug 被设置为 on）：候选计算 Node Group 为指定的 Node Group

默认值：query

enable_nodegroup_debug

参数说明：控制优化器在多 Node Group 环境下，是否使用强制弹性计算。Node Group 目前为内部用机制，用户无需设置。

该参数只在 expected_computing_nodegroup 被设置为具体 Node Group 时生效。

该参数属于 USERSET 类型参数，请参考表 10-1 中对应设置方法进行设置。

取值范围：布尔型

- on 表示强制将计算弹性到 expected_computing_nodegroup 所指定的 Node Group 进行计算。
- off 表示不强制使用某个 Node Group 进行计算。

默认值：off

qrw_inlist2join_optmode

参数说明：控制是否使用 inlist-to-join 查询重写。

该参数属于 USERSET 类型参数，请参考表 10-1 中对应设置方法进行设置。

取值范围：字符串

- disable：关闭 inlist2join 查询重写。
- cost_base：基于代价的 inlist2join 查询重写。
- rule_base：基于规则的 inlist2join 查询重写，即强制使用 inlist2join 查询重写。
- 任意正整数：inlist2join 查询重写阈值，即 list 内元素个数大于该阈值，进行 inlist2join 查询重写。

默认值：cost_base

skew_option

参数说明：控制是否使用优化策略。

该参数属于 USERSET 类型参数，请参考表 10-1 中对应设置方法进行设置。

取值范围：字符串

- off：关闭策略。
- normal：采用激进策略。对于不确定是否出现倾斜的场景，认为存在倾斜，并进行相应优化。
- lazy：采用保守策略。对于不确定是否出现倾斜场景，认为不存在倾斜，不进行优化。

默认值：normal

19.8.2. 优化器开销常量

介绍优化器开销常量。这里描述的开销可以按照任意标准度量。只关心其相对值，因此以相同的系数缩放它们将不会对优化器的选择产生任何影响。缺省时，它们以抓取顺序页的开销为基本单位。也就是说将 seq_page_cost 设为 1.0，同时其他开销参数以它为基准设置。也可以使用其他基准，比如以毫秒计的实际执行时间。

seq_page_cost

参数说明：设置优化器计算一次顺序磁盘页面抓取的开销。

该参数属于 USERSET 类型参数，请参考表 10-1 中对应设置方法进行设置。

取值范围：浮点型，0 ~ DBL_MAX。

默认值：1

random_page_cost

参数说明：设置优化器计算一次非顺序抓取磁盘页面的开销。

该参数属于 USERSET 类型参数，请参考表 10-1 中对应设置方法进行设置。

须知

虽然服务器允许将 random_page_cost 设置的比 seq_page_cost 小，但是物理上实际不受影响。如果所有数据库都位于随机访问内存中时，两者设置为相等很合理。因为在此种情况下，非顺序抓取页并没有副作用。同样，在缓冲率很高的数据库上，应该相对于 CPU 参数同时降低这两个值，因为获取内存中的页要比通常情况下开销小很多。

取值范围：浮点型，0 ~ DBL_MAX。

默认值：4

说明

- 对于特别表空间中的表和索引，可以通过设置同名的表空间的参数来覆盖这个值。
- 相对于 seq_page_cost，减少这个值将导致系统更倾向于使用索引扫描，而增加这个值使得索引扫描开销比较高。可以通过同时增加或减少这两个值来调整磁盘 I/O 相对于 CPU 的开销。

cpu_tuple_cost

参数说明：设置优化器计算在一次查询中处理每一行数据的开销。

该参数属于 USERSET 类型参数，请参考表 10-1 中对应设置方法进行设置。

取值范围：浮点型，0 ~ DBL_MAX。

默认值：0.01

cpu_index_tuple_cost

参数说明：设置优化器计算在一次索引扫描中处理每条索引的开销。

该参数属于 USERSET 类型参数，请参考表 10-1 中对应设置方法进行设置。

取值范围：浮点型，0 ~ DBL_MAX。

默认值：0.005

cpu_operator_cost

参数说明：设置优化器计算一次查询中执行一个操作符或函数的开销。

该参数属于 USERSET 类型参数，请参考表 10-1 中对应设置方法进行设置。

取值范围：浮点型，0 ~ DBL_MAX。

默认值：0.0025

effective_cache_size

参数说明：设置优化器在一次单一的查询中可用的磁盘缓冲区的有效大小。

设置这个参数，还要考虑 Vastbase 的共享缓冲区以及内核的磁盘缓冲区。另外，还要考虑预计的在不同表之间的并发查询数目，因为它们将共享可用的空间。

这个参数对 Vastbase 分配的共享内存大小没有影响，它也不会使用内核磁盘缓冲，它只用于估算。数值是用磁盘页来计算的，通常每个页面是 8192 字节。

该参数属于 USERSET 类型参数，请参考表 10-1 中对应设置方法进行设置。

取值范围：整型，1～INT_MAX，单位为 8KB。

比默认值高的数值可能会导致使用索引扫描，更低的数值可能会导致选择顺序扫描。

默认值：128MB

allocate_mem_cost

参数说明：设置优化器计算 Hash Join 创建 Hash 表开辟内存空间所需的开销，供 Hash join 估算不准时调优使用。

该参数属于 USERSET 类型参数，请参考表 10-1 中对应设置方法进行设置。

取值范围：浮点型，0～DBL_MAX。

默认值：0

19.8.3. 基因查询优化器

介绍基因查询优化器相关的参数。基因查询优化器（GEQO）是一种启发式的查询规划算法。这个算法减少了对复杂查询规划的时间，而且生成规划的开销有时也小于正常的详尽的查询算法。

geqo

参数说明：控制基因查询优化的使用。

该参数属于 USERSET 类型参数，请参考表 10-1 中对应设置方法进行设置。

须知

通常情况下在执行过程中不要关闭，geqo_threshold 变量提供了更精细的控制 GEQO 的方法。

取值范围：布尔型

- on 表示使用。
- off 表示不使用。

默认值：on

geqo_threshold

参数说明：如果执行语句的数量超过设计的 FROM 的项数，则会使用基因查询优化来执行查询。

该参数属于 USERSET 类型参数，请参考表 10-1 中对应设置方法进行设置。

须知

- 对于简单的查询，通常用详尽搜索方法，当涉及多个表的查询的时候，用 GEQO 可以更好的管理查询。
- 一个 FULL OUTER JOIN 构造仅作为一个 FROM 项。

取值范围：整型，2～INT_MAX。

默认值：12

geqo_effort

参数说明：控制 GEQO 在规划时间和规划质量之间的平衡。

该参数属于 USERSET 类型参数，请参考表 10-1 中对应设置方法进行设置。

须知

geqo_effort 实际上并没有直接做任何事情，只是用于计算其他影响 GEQO 的变量的默认值。如果愿意，可以手工设置其他参数。

取值范围：整型，1～10。

须知

比默认值大的数值增加了查询规划的时间，但是也增加了选中有效查询的几率。

默认值：5

geqo_pool_size

参数说明：控制 GEQO 使用池的大小，也就是基因全体中的个体数量。

该参数属于 USERSET 类型参数，请参考表 10-1 中对应设置方法进行设置。

取值范围：整型，0～INT_MAX。

须知

至少是 2，且有用的值一般在 100 到 1000 之间。设置为 0，表示使用系统自适应方式，Vastbase 会基于 geqo_effort 和表的个数选取合适的值。

默认值：0

geqo_generations

参数说明：控制 GEQO 使用的算法的迭代次数。

该参数属于 USERSET 类型参数，请参考表 10-1 中对应设置方法进行设置。

取值范围：整型，0～INT_MAX。

须知

必须至少是 1，且有用的值介于 100 和 1000 之间。如果设置为 0，则基于 geqo_pool_size 选取合适的值。

默认值：0

geqo_selection_bias

参数说明：控制 GEQO 的选择性偏好，即就是一个种群中的选择性压力。

该参数属于 USERSET 类型参数，请参考表 10-1 中对应设置方法进行设置。

取值范围：浮点型，1.5 ~ 2.0。

默认值：2

geqo_seed

参数说明：控制 GEQO 使用的随机数生产器的初始化值，用来从顺序连接在一起的查询空间中查找随机路径。

该参数属于 USERSET 类型参数，请参考表 10-1 中对应设置方法进行设置。

取值范围：浮点型，0.0 ~ 1.0。

须知

不同的值会改变搜索的连接路径，从而影响了所找路径的优劣。

默认值：0

19.8.4. 其他优化器选项

default_statistics_target

参数说明：为没有用 ALTER TABLE SET STATISTICS 设置字段目标的表设置缺省统计目标。此参数设置为正数是代表统计信息的样本数量，为负数时，代表使用百分比的形式设置统计目标，负数转换为对应的百分比，即-5 代表 5%。

该参数属于 USERSET 类型参数，请参考表 10-1 中对应设置方法进行设置。

取值范围：整型，-100 ~ 10000。

须知

- 比默认值大的正数数值增加了 ANALYZE 所需的时间，但是可能会改善优化器的估计质量。
- 调整此参数可能存在性能劣化的风险，如果某个查询劣化，可以考虑

- 恢复默认的统计信息。

5. 使用 plan hint 来调整到之前的查询计划。

- 当此 guc 参数设置为负数时，如果计算的采样样本数大于等于总数据量的 2%，且用户表的数据量小于 1600000 时，ANALYZE 所需时间相比 guc 参数为默认值的时间会有所增加。
- 当此 guc 参数设置为负数时，则 autoanalyze 不生效。

默认值：100

constraint_exclusion

参数说明：控制查询优化器使用表约束查询的优化。

该参数属于 USERSET 类型参数，请参考表 10-1 中对应设置方法进行设置。

取值范围：枚举类型

- on 表示检查所有表的约束。
- off 表示不检查约束。
- partition 表示只检查继承的子表和 UNION ALL 子查询。

须知

当 constraint_exclusion 为 on，优化器用查询条件和表的 CHECK 约束比较，并且在查询条件和约束冲突的时候忽略对表的扫描。

默认值：partition

说明

目前， constraint_exclusion 缺省被打开，通常用来实现表分区。为所有的表打开它时，对于简单的查询强加了额外的规划，并且对简单查询没有什么好处。如果不用分区表，可以关掉它。

cursor_tuple_fraction

参数说明：优化器估计游标获取行数在总行数中的占比。

该参数属于 USERSET 类型参数，请参考表 10-1 中对应设置方法进行设置。

取值范围：浮点型，0.0 ~ 1.0。

须知

比默认值小的值与使用 “fast start” 为游标规划的值相偏离，从而使得前几行恢复的很快而抓取全部的行需要很长的时间。比默认值大的值加大了总的估计的时间。在最大的值 1.0 处，像正常的查询一样规划游标，只考虑总的估计时间和传送第一行的时间。

默认值：0.1

from_collapse_limit

参数说明：根据生成的 FROM 列表的项数来判断优化器是否将把子查询合并到上层查询，如果 FROM 列表项个数小于等于该参数值，优化器会将子查询合并到上层查询。

该参数属于 USERSET 类型参数，请参考表 10-1 中对应设置方法进行设置。

取值范围：整型，1～INT_MAX。

须知

比默认值小的数值将降低规划时间，但是可能生成差的执行计划。

默认值：8

join_collapse_limit

参数说明：根据得出的列表项数来判断优化器是否执行把除 FULL JOINS 之外的 JOIN 构造重写到 FROM 列表中。

该参数属于 USERSET 类型参数，请参考表 10-1 中对应设置方法进行设置。

取值范围：整型，1～INT_MAX。

须知

- 设置为 1 会避免任何 JOIN 重排。这样就使得查询中指定的连接顺序就是实际的连接顺序。查询优化器并不是总能选取最优的连接顺序，高级用户可以选择暂时把这个变量设置为 1，然后指定它们需要的连接顺序。
- 比默认值小的数值减少规划时间但也降低了执行计划的质量。

默认值：8

plan_mode_seed

参数说明：该参数为调测参数，目前仅支持 OPTIMIZE_PLAN 和 RANDOM_PLAN 两种。其中：OPTIMIZE_PLAN 表示通过动态规划算法进行代价估算的最优 plan，参数值设置为 0；RANDOM_PLAN 表示随机生成的 plan；如果设置为-1，表示用户不指定随机数的种子标识符 seed 值，由优化器随机生成[1, 2147483647]范围整型值的随机数，并根据随机数生成随机的执行计划；如果用户指定 guc 参数值为[1, 2147483647]范围的整型值，表示指定的生成随机数的种子标识符 seed，优化器需要根据 seed 值生成随机的执行计划。

该参数属于 USERSET 类型参数，请参考表 10-1 中对应设置方法进行设置。

取值范围：整型，-1～ 2147483647

默认值：0

须知

- 当该参数设置为随机执行计划模式时，优化器会生成不同的随机执行计划，该执行计划可能不是最优计划。因此在随机计划模式下，会对查询性能产生影响，所以建议在升级、扩容、缩容等正常业务操作或运维过程中将该参数保持为默认值 0。
- 当该参数不为 0 时，查询指定的 plan hint 不会生效。

hashagg_table_size

参数说明：用于设置执行 HASH JOIN 操作时 HASH 表的大小。

该参数属于 USERSET 类型参数，请参考表 10-1 中对应设置方法进行设置。

取值范围：整型，0～INT_MAX/2。

默认值：0

enable_codegen

参数说明：标识是否允许开启代码生成优化，目前代码生成使用的是 LLVM 优化。

该参数属于 USERSET 类型参数，请参考表 10-1 中对应设置方法进行设置。

取值范围：布尔型

- on 表示允许开启代码生成优化。
- off 表示不允许开启代码生成优化。

须知

目前 LLVM 优化仅支持向量化执行引擎特性和 SQL on Hadoop 特性，在其他场景下建议关闭此参数。

默认值：on

codegen_strategy

参数说明：标识在表达式 codegen 化过程中所使用的代码生成优化策略。

该参数属于 USERSET 类型参数，请参考表 10-1 中对应设置方法进行设置。

取值范围：枚举类型

- partial 表示当所计算表达式中即使包含部分未被 codegen 化的函数时，仍可借助表达式全 codegen 框架调用 LLVM 动态编译优化策略。
- pure 表示当所计算表达式整体可被 codegen 化时，才考虑调用 LLVM 动态编译优化策略。

须知

在开启代码生成优化会导致查询性能下降的场景下可以设置此参数为 pure，其他场景下建议不改变此参数的默认值 partial。

默认值：partial

enable_codegen_print

参数说明：标识是否允许在 log 日志中打印所生成的 LLVM IR 函数。

该参数属于 USERSET 类型参数，请参考表 10-1 中对应设置方法进行设置。

取值范围：布尔型

- on 表示允许在 log 日志中打印 IR 函数。
- off 表示不允许在 log 日志中打印 IR 函数。

默认值：off

codegen_cost_threshold

参数说明：由于 LLVM 编译生成最终的可执行机器码需要一定时间，因此只有当实际执行的代价大于编译生成机器码所需要的代码和优化后的执行代价之和时，利用代码生成才有收益。

codegen_cost_threshold 标识代价的阈值，当执行估算代价大于该代价时，使用 LLVM 优化。

该参数属于 USERSET 类型参数，请参考表 10-1 中对应设置方法进行设置。

取值范围：整型，0 ~ 2147483647。

默认值：10000

enable_bloom_filter

参数说明: 标识是否允许使用 BloomFilter 优化。该参数属于 USERSET 类型参数，请参考表 10-1 中对应设置方法进行设置。

取值范围：布尔型

- on 表示允许使用 BloomFilter 优化。
- off 表示不允许使用 BloomFilter 优化。

默认值：on

enable_extrapolation_stats

参数说明: 标识对于日期类型是否允许基于历史统计信息使用推理估算的逻辑。使用该逻辑对于未及时收集统计信息的表可以增大估算准确的可能性，但也存在错误推理导致估算过大的可能性，需要对于日期类型数据定期插入的场景开启此开关。该参数属于 SUSET 类型参数，请参考表 10-1 中对应设置方法进行设置。

取值范围：布尔型

- on 表示允许基于历史统计信息使用推理估算的逻辑。
- off 表示不允许基于历史统计信息使用推理估算的逻辑。

默认值：off

autoanalyze

参数说明：标识是否允许在生成计划的时候，对于没有统计信息的表进行统计信息自动收集。对于外表和临时表，不支持 autoanalyze，如果需要收集统计信息，用户需手动执行 analyze 操作。如果在 auto analyze 某个表的过程中数据库发生异常，当数据库正常运行之后再执行语句有可能仍提示需要收集此表的统计信息。此时需要用户对该表手动执行一次 analyze 操作，以同步统计信息数据。该参数属于 SUSET 类型参数，请参考表 10-1 中对应设置方法进行设置。

取值范围：布尔型

- on 表示允许自动进行统计信息收集。
- off 表示不允许自动进行统计信息收集。

默认值：off

enable_analyze_check

参数说明：标识是否允许在生成计划的时候，对于在 pg_class 中显示 reltuples 和 relpages 均为 0 的表，检查该表是否曾进行过统计信息收集。

该参数属于 SUSET 类型参数，请参考表 10-1 中对应设置方法进行设置。

取值范围：布尔型

- on 表示允许检查。
- off 表示不允许检查。

默认值：on

enable_sonic_hashagg

参数说明：标识是否依据规则约束使用基于面向列的 hash 表设计的 Hash Agg 算子。

该参数属于 USERSET 类型参数，请参考表 10-1 中对应设置方法进行设置。

取值范围：布尔型

- on 表示在满足约束条件时使用基于面向列的 hash 表设计的 Hash Agg 算子。
- off 表示不使用面向列的 hash 表设计的 Hash Agg 算子。

说明

- 在开启 enable_sonic_hashagg，且查询达到约束条件使用基于面向列的 hash 表设计的 Hash Agg 算子时，查询对应的 Hash Agg 算子内存使用通常可获得精简。但对于代码生成技术可获得显著性能提升的场景(enable_codegen 打开后获得较大性能提升)，对应的算子查询性能可能会出现劣化。
- 开启 enable_sonic_hashagg，且查询达到约束条件使用基于面向列的 hash 表设计的 Hash Agg 算子时，在 Explain Analyze/Performance 的执行计划和执行信息中，算子显示为“Sonic Hash Aggregation”，而未达到该约束条件时，算子名称将显示为“Hash Aggregation”。

默认值：on

enable_sonic_hashjoin

参数说明：标识是否依据规则约束使用基于面向列的 hash 表设计的 Hash Join 算子。

该参数属于 USERSET 类型参数，请参考表 10-1 中对应设置方法进行设置。

取值范围：布尔型

- on 表示在满足约束条件时使用基于面向列的 hash 表设计的 Hash Join 算子。
- off 表示不使用面向列的 hash 表设计的 Hash Join 算子。

说明

- 当前开关仅适用于 Inner Join 的场景。
- 在开启 enable_sonic_hashjoin，查询对应的 Hash Inner 算子内存使用通常可获得精简。但对于代码生成技术可获得显著性能提升的场景，对应的算子查询性能可能会出现劣化。
- 开启 enable_sonic_hashjoin，且查询达到约束条件使用基于面向列的 hash 表设计的 Hash Join 算子时，在 Explain Analyze/Performance 的执行计划和执行信息中，算子显示为“Sonic Hash Join”，而未达到该约束条件时，算子名称将显示为“Hash Join”。

默认值：on

enable_sonic_optspill

参数说明：标识是否对面向列的 hash 表设计的 Hash Join 算子进行下盘文件数优化。该参数打开时，在 Hash Join 算子下盘文件较多的时候，下盘文件数不会显著增加。

该参数属于 USERSET 类型参数，请参考表 10-1 中对应设置方法进行设置。

取值范围：布尔型

- on 表示优化面向列的 hash 表设计的 Hash Join 算子的下盘文件数。
- off 表示不优化面向列的 hash 表设计的 Hash Join 算子的下盘文件数。

默认值：on

log_parser_stats

参数说明：控制优化器输出 parser 模块的性能日志。

该参数属于 SUSET 类型参数，请参考表 10-1 中对应设置方法进行设置。

取值范围：布尔型

- on 表示使用。
- off 表示不使用。

默认值：off

log_planner_stats

参数说明：控制优化器输出 planner 模块的性能日志。

该参数属于 SUSET 类型参数，请参考表 10-1 中对应设置方法进行设置。

取值范围：布尔型

- on 表示使用。
- off 表示不使用。

默认值：off

log_executor_stats

参数说明：控制优化器输出 executor 模块的性能日志。

该参数属于 SUSET 类型参数，请参考表 10-1 中对应设置方法进行设置。

取值范围：布尔型

- on 表示使用。
- off 表示不使用。

默认值：off

log_statement_stats

参数说明：控制优化器输出该语句的性能日志。

该参数属于 SUSET 类型参数，请参考表 10-1 中对应设置方法进行设置。

取值范围：布尔型

- on 表示使用。
- off 表示不使用。

默认值：off

plan_cache_mode

参数说明：标识在 prepare 语句中，选择生成执行计划的策略。

该参数属于 USERSET 类型参数，请参考表 10-1 中对应设置方法进行设置。

取值范围：枚举类型

- auto 表示按照默认的方式选择 custom plan 或者 generic plan。
- force_generic_plan 表示强制走 generic plan。
- force_custom_plan 表示强制走 custom plan。

说明

- 此参数只对 prepare 语句生效，一般用在 prepare 语句中参数化字段存在比较严重的数据倾斜的场景下。
- custom plan 是指对于 preapre 语句，在执行 execute 的时候，把 execute 语句中的参数嵌套到语句之后生成的计划。custom plan 会根据 execute 语句中具体的参数生成计划，这种方案的优点是每次都按照具体的参数生成优选计划，执行性能比较好；缺点是每次执行前都需要重新生成计划，存在大量的重复的优化器开销。
- generic plan 是指对于 preapre 语句生成计划,该计划策略会在执行 execute 语句的时候把参数 bind 到 plan 中，然后执行计划。这种方案的优点是每次执行可以省去重复的优化器开销；缺点是当 bind 参数字段上数据存在倾斜时该计划可能不是最优的，部分 bind 参数场景下执行性能较差。

默认值：auto

19.9. 错误报告和日志

19.9.1. 记录日志的位置

log_destination

参数说明：Vastbase 支持多种方法记录服务器日志，log_destination 的取值为一个逗号分隔开的列表（如 log_destination="stderr,csvlog"）。

该参数属于 SIGHUP 类型参数，请参考表 10-1 中对应设置方法进行设置。

取值范围：字符串

有效值为 stderr、csvlog、syslog。

- 取值为 stderr，表示日志打印到屏幕。
- 取值为 csvlog，表示日志的输出格式为“逗号分隔值”即 CSV （Comma Separated Value）格式。使用 csvlog 记录日志的前提是将 logging_collector 设置为 on，请参见 21.9.4 使用 CSV 格式写日志。
- 取值为 syslog，表示通过操作系统的 syslog 记录日志。Vastbase 使用 syslog 的 LOCAL0 ~ LOCAL7 记录日志，请参见 syslog_facility。使用 syslog 记录日志需在操作系统后台服务配置文件中添加代码：

```
local0.*  /var/log/postgresql
```

默认值：stderr

logging_collector

参数说明：控制开启后端日志收集进程 logger 进行日志收集。该进程捕获发送到 stderr 或 csvlog 的日志消息并写入日志文件。

这种记录日志的方法比将日志记录到 syslog 更加有效，因为某些类型的消息在 syslog 的输出中无法显示。例如动态链接库加载失败消息和脚本（例如 archive_command）产生的错误消息。

该参数属于 POSTMASTER 类型参数，请参考表 10-1 中对应设置方法进行设置。

须知

将服务器日志发送到 stderr 时可以不使用 logging_collector 参数，此时日志消息会被发送到服务器的 stderr 指向的空间。这种方法的缺点是日志回滚困难，只适用于较小的日志容量。

取值范围：布尔型

- on 表示开启日志收集功能。
- off 表示关闭日志收集功能。

默认值：on

log_directory

参数说明：logging_collector 设置为 on 时，log_directory 决定存放服务器日志文件的目录。 它可以是绝对路径，或者是相对路径（相对于数据目录的路径）。在配置文件 postgresql.conf 中修改 log_directory 后，可以通过 vb_ctl reload 使参数生效。

该参数属于 SIGHUP 类型参数，请参考表 10-1 中对应设置方法进行设置。

须知

- 当配置文件中 log_directory 的值为非法路径时，会导致 Vastbase 无法重新启动。
- 通过 vb_ctl reload 加载 log_directory 配置时，当指定路径为合法路径时，日志输出到新的路径下。当指定路径为非法路径时，日志输出到上一次合法的日志输出路径下而不影响数据库正常运行。此时即使指定的 log_directory 的值非法，也会写入到配置文件中。

说明

合法路径：用户对此路径有读写权限

非法路径：用户对此路径无读写权限

取值范围：字符串

默认值：安装时指定。

log_filename

参数说明：logging_collector 设置为 on 时，log_filename 决定服务器运行日志文件的名称。通常日志文件名是按照 strftime 模式生成，因此可以用系统时间定义日志文件名，用%转义字符实现。

该参数属于 SIGHUP 类型参数，请参考表 10-1 中对应设置方法进行设置。

须知

- 建议使用%转义字符定义日志文件名称，否则难以对日志文件进行有效的管理。
- 当 log_destination 设为 csvlog 时，系统会生成附加了时间戳的日志文件名，文件格式为 csv 格式，例如“server_log.1093827753.csv”。

取值范围：字符串

默认值：postgresql-%Y-%m-%d_%H%M%S.log

log_file_mode

参数说明：logging_collector 设置为 on 时，log_file_mode 设置服务器日志文件的权限。通常 log_file_mode 的取值是能够被 chmod 和 umask 系统调用接受的数字。

该参数属于 SIGHUP 类型参数，请参考表 10-1 中对应设置方法进行设置。

须知

- 使用此选项前请设置 log_directory，将日志存储到数据目录之外的地方。
- 因日志文件可能含有敏感数据，故不能将其设为对外可读。

取值范围：整型，0000～0777 （8 进制计数，转化为十进制 0 ~ 511）。

说明

- 0600 表示只允许服务器管理员读写日志文件。
- 0640 表示允许管理员所在用户组成员只能读日志文件。

默认值：0600

log_truncate_on_rotation

参数说明：logging_collector 设置为 on 时，log_truncate_on_rotation 设置日志消息的写入方式。该参数属于 SIGHUP 类型参数，请参考表 10-1 中对应设置方法进行设置。

示例如下：

假设日志需要保留 7 天，每天生成一个日志文件，日志文件名设置为 server_log.Mon、server_log.Tue 等。第二周的周二生成的日志消息会覆盖写入到 server_log.Tue。设置方法：将 log_filename 设置为 server_log.%a ，log_truncate_on_rotation 设置为 on，log_rotation_age 设置为 1440，即日志有效时间为 1 天。

取值范围： 布尔型

- on 表示 Vastbase 以覆盖写入的方式写服务器日志消息。
- off 表示 Vastbase 将日志消息附加到同名的现有日志文件上。

默认值：off

log_rotation_age

参数说明：logging_collector 设置为 on 时，log_rotation_age 决定创建一个新日志文件的时间间隔。当现在的时间减去上次创建一个服务器日志的时间超过了 log_rotation_age 的值时，将生成一个新的日志文件。

该参数属于 SIGHUP 类型参数，请参考表 10-1 中对应设置方法进行设置。

取值范围：整型，0 ~ 35791394，单位为 min。其中 0 表示关闭基于时间的新日志文件的创建。

默认值：1440(min)

log_rotation_size

参数说明：logging_collector 设置为 on 时，log_rotation_size 决定服务器日志文件的最大容量。当日志消息的总量超过日志文件容量时，服务器将生成一个新的日志文件。

该参数属于 SIGHUP 类型参数，请参考表 10-1 中对应设置方法进行设置。

取值范围：整型，0 ~ INT_MAX / 1024，单位为 KB。

0 表示关闭基于容量的新日志文件的创建。

建议该值大小设置级别至少为 MB 级,利于日志文件的及时划分。

默认值：20MB

syslog_facility

参数说明：log_destination 设置为 syslog 时，syslog_facility 配置使用 syslog 记录日志的“设备”。

该参数属于 SIGHUP 类型参数，请参考表 10-1 中对应设置方法进行设置。

取值范围：枚举类型，有效值有 local0、local1、local2、local3、local4、local5、local6、local7。

默认值：local0

syslog_ident

参数说明：log_destination 设置为 syslog 时，syslog_ident 设置在 syslog 日志中 Vastbase 日志消息的标识。

该参数属于 SIGHUP 类型参数，请参考表 10-1 中对应设置方法进行设置。

取值范围：字符串

默认值：postgres

event_source

参数说明：该参数仅在 windows 环境下生效, Vastbase 暂不支持。log_destination 设置为 eventlog 时，event_source 设置在日志中 Vastbase 日志消息的标识。

该参数属于 POSTMASTER 类型参数，请参考表 10-1 中对应设置方法进行设置。

取值范围：字符串

默认值：PostgreSQL

19.9.2. 记录日志的时间

client_min_messages

参数说明：控制发送到客户端的消息级别。每个级别都包含排在它后面的所有级别中的信息。级别越低，发送给客户端的消息就越少。

该参数属于 USERSET 类型参数，请参考表 10-1 中对应设置方法进行设置。

须知

当 client_min_messages 和 log_min_messages 取相同值时，其值所代表的级别不同。

取值范围：枚举类型，有效值有 debug、debug5、debug4、debug3、debug2、debug1、info、log、notice、warning、error、fatal、panic。参数的详细信息请参见表 19-1。在实际设置过程中，如果设置的级别大于 error，为 fatal 或 panic，系统会默认将级别转为 error。

默认值：notice

log_min_messages

参数说明：控制写到服务器日志文件中的消息级别。每个级别都包含排在它后面的所有级别中的信息。级别越低，服务器运行日志中记录的消息就越少。

该参数属于 SUSET 类型参数，请参考表 10-1 中对应设置方法进行设置。

须知

当 client_min_messages 和 log_min_messages 取相同值 log 时所代表的消息级别不同。

取值范围：枚举类型，有效值有 debug、debug5、debug4、debug3、debug2、debug1、info、log、notice、warning、error、fatal、panic。参数的详细信息请参见表 19-1。

默认值：warning

log_min_error_statement

参数说明：控制在服务器日志中记录错误的 SQL 语句。

该参数属于 SUSET 类型参数，请参考表 10-1 中对应设置方法进行设置。

取值范围：枚举类型，有效值有 debug、debug5、debug4、debug3、debug2、debug1、info、log、notice、warning、error、fatal、panic。参数的详细信息请参见表 19-1。

说明

- 设置为 error ，表示导致错误、日志消息、致命错误、panic 的语句都将被记录。
- 设置为 panic，表示关闭此特性。

默认值：error

log_min_duration_statement

参数说明：当某条语句的持续时间大于或者等于特定的毫秒数时，log_min_duration_statement 参数用于控制记录每条完成语句的持续时间。

设置 log_min_duration_statement 可以很方便地跟踪需要优化的查询语句。对于使用扩展查询协议的客户端，语法分析、绑定、执行每一步所花时间被独立记录。

该参数属于 SUSET 类型参数，请参考表 10-1 中对应设置方法进行设置。

须知

当此选项与 log_statement 同时使用时，已经被 log_statement 记录的语句文本不会被重复记录。在没有使用 syslog 情况下，推荐使用 log_line_prefix 记录 PID 或会话 ID，方便将当前语句消息连接到最后的持续时间消息。

取值范围：整型，-1 ~ INT_MAX，单位为毫秒。

- 设置为 250，所有运行时间不短于 250ms 的 SQL 语句都会被记录。
- 设置为 0，输出所有语句的持续时间。
- 设置为-1，关闭此功能。

默认值：30min

backtrace_min_messages

参数说明：控制当产生该设置参数级别相等或更高级别的信息时，会打印函数的堆栈信息到服务器日志文件中。

该参数属于 SUSET 类型参数，请参考表 10-1 中对应设置方法进行设置。

须知

该参数作为客户现场问题定位手段使用，且由于频繁的打印函数栈会对系统的开销及稳定性有一定的影响，因此如果需要进行问题定位时，建议避免将 backtrace_min_messages 的值设置为 fatal 及 panic 以外的级别。

取值范围：枚举类型

有效值有 debug、debug5、debug4、debug3、debug2、debug1、info、log、notice、warning、error、fatal、panic。参数的详细信息请参见表 19-1。

默认值：panic

表 19-1 解释 Vastbase 中使用的消息安全级别。当日志输出到 syslog 或者 eventlog(仅 windows 环境下, Vastbase 版本不涉及该参数)时，Vastbase 进行如表中的转换。

表 19-1. 信息严重程度分类

信息严重程度类型	详细说明	系统日志	事件日志
debug[1-5]	报告详细调试信息。	DEBUG	INFORMATION
log	报告对数据库管理员有用的信息， 比如检查点操作统计信息。	INFO	INFORMATION
info	报告用户可能需求的信息， 比如在 VACUUM VERBOSE 过程中的信息。	INFO	INFORMATION
notice	报告可能对用户有帮助的信息， 比如，长标识符的截断， 作为主键一部分创建的索引等。	NOTICE	INFORMATION
warning	报告警告信息， 比如在事务块范围之外的 COMMIT。	NOTICE	WARNING
error	报告导致当前命令退出的错误。	WARNING	ERROR

信息严重程度类型	详细说明	系统日志	事件日志
fatal	报告导致当前会话终止的原因。	ERR	ERROR
panic	报告导致整个数据库被关闭的原因。	CRIT	ERROR

plog_merge_age

参数说明：该参数用于控制性能日志数据输出的周期。

该参数属于 USERSET 类型参数，请参考表 10-1 中对应设置方法进行设置。

须知

该参数以毫秒为单位的，建议在使用过程中设置值为 1000 的整数倍，即设置值以秒为最小单位。该参数所控制的性能日志文件以 prf 为扩展名，文件放置在$GAUSSLOG/gs_profile/<node_name> 目录下面，其中 node_name 是由 postgres.conf 文件中的 pgxc_node_name 的值，不建议外部使用该参数。

取值范围：0~2147483647，单位为毫秒（ms）。

当设置为 0 时，当前会话不再输出性能日志数据。当设置为非 0 时，当前会话按照指定的时间周期进行输出性能日志数据。

该参数设置得越小，输出的日志数据越多，对性能的负面影响越大。

默认值：3s

19.9.3. 记录日志的内容

debug_print_parse

参数说明：用于控制打印解析树结果。

该参数属于 SIGHUP 类型参数，请参考表 10-1 中对应设置方法进行设置。

取值范围：布尔型

- on 表示开启打印结果的功能。
- off 表示关闭打印结果的功能。

默认值：off

debug_print_rewritten

参数说明：用于控制打印查询重写结果。

该参数属于 SIGHUP 类型参数，请参考表 10-1 中对应设置方法进行设置。

取值范围：布尔型

- on 表示开启打印结果的功能。

- off 表示关闭打印结果的功能。

默认值：off

debug_print_plan

参数说明：用于控制打印查询执行结果。

该参数属于 SIGHUP 类型参数，请参考表 10-1 中对应设置方法进行设置。

取值范围：布尔型

- on 表示开启打印结果的功能。
- off 表示关闭打印结果的功能。

默认值：off

须知

- 只有当日志的级别为 log 及以上时，debug_print_parse、debug_print_rewritten 和 debug_print_plan 的调试信息才会输出。当这些选项打开时，调试信息只会记录在服务器的日志中，而不会输出到客户端的日志中。通过设置 client_min_messages 和 log_min_messages 参数可以改变日志级别。
- 在打开 debug_print_plan 开关的情况下需尽量避免调用 gs_encrypt_aes128 及 gs_decrypt_aes128 函数，避免敏感参数信息在日志中泄露的风险。同时建议用户在打开 debug_print_plan 开关生成的日志中对 gs_encrypt_aes128 及 gs_decrypt_aes128 函数的参数信息进行过滤后再提供给外部维护人员定位，日志使用完成后请及时删除。

debug_pretty_print

参数说明：设置此选项对 debug_print_parse、debug_print_rewritten 和 debug_print_plan 产生的日志进行缩进，会生成易读但比设置为 off 时更长的输出格式。

该参数属于 USERSET 类型参数，请参考表 10-1 中对应设置方法进行设置。

取值范围：布尔型

- on 表示进行缩进。
- off 表示不进行缩进。

默认值：on

log_checkpoints

参数说明：控制在服务器日志中记录检查点和重启点的信息。打开此参数时，服务器日志消息包含涉及检查点和重启点的统计量，其中包含需要写的缓存区的数量及写入所花费的时间等。

该参数属于 SIGHUP 类型参数，请参考表 10-1 中对应设置方法进行设置。

取值范围：布尔型

- on 表示打开此参数时，服务器日志消息包含涉及检查点和重启点的统计量。
- off 表示关闭此参数时，服务器日志消息包含不涉及检查点和重启点的统计量。

默认值：off

log_connections

参数说明：控制记录客户端的连接请求信息。

该参数属于 BACKEND 类型参数，请参考表 10-1 中对应设置方法进行设置。

须知

有些客户端程序（例如 vsql），在判断是否需要口令的时候会尝试连接两次，因此日志消息中重复的“connection receive”（收到连接请求）并不意味着一定是问题。

取值范围：布尔型

- on 表示记录信息。
- off 表示不记录信息。

默认值：off

log_disconnections

参数说明：控制记录客户端结束连接信息。

该参数属于 BACKEND 类型参数，请参考表 10-1 中对应设置方法进行设置。

取值范围：布尔型

- on 表示记录信息。
- off 表示不记录信息。

默认值：off

log_duration

参数说明：控制记录每个已完成 SQL 语句的执行时间。对使用扩展查询协议的客户端、会记录语法分析、绑定和执行每一步所花费的时间。

该参数属于 SUSET 类型参数，请参考表 10-1 中对应设置方法进行设置。

取值范围：布尔型

- 设置为 off ，该选项与 log_min_duration_statement 的不同之处在于 log_min_duration_statement 强制记录查询文本。
- 设置为 on 并且 log_min_duration_statement 大于零，记录所有持续时间，但是仅记录超过阈值的语句。这可用于在高负载情况下搜集统计信息。

默认值：on

log_error_verbosity

参数说明：控制服务器日志中每条记录的消息写入的详细度。

该参数属于 SUSET 类型参数，请参考表 10-1 中对应设置方法进行设置。

取值范围：枚举类型

- terse 输出不包括 DETAIL、HINT、QUERY 及 CONTEXT 错误信息的记录。
- verbose 输出包括 SQLSTATE 错误代码 、源代码文件名、函数名及产生错误所在的行号。
- default 输出包括 DETAIL、HINT、QUERY 及 CONTEXT 错误信息的记录，不包括 SQLSTATE 错误代码 、源代码文件名、函数名及产生错误所在的行号。

默认值：default

log_hostname

参数说明：选项关闭状态下，连接消息日志只显示正在连接主机的 IP 地址。打开此选项同时可以记录主机名。由于解析主机名可能需要一定的时间，可能影响数据库的性能。

该参数属于 SIGHUP 类型参数，请参考表 10-1 中对应设置方法进行设置。

取值范围：布尔型

- on 表示可以同时记录主机名。
- off 表示不可以同时记录主机名。

默认值：on

log_line_prefix

参数说明：控制每条日志信息的前缀格式。日志前缀类似于 printf 风格的字符串，在日志的每行开头输出。用以%为开头的“转义字符”代替表 19-2 中的状态信息。

该参数属于 SIGHUP 类型参数，请参考表 10-1 中对应设置方法进行设置。

表 19-2. 转义字符表

转义字符	效果
%a	应用程序名称。
%u	用户名。
%d	数据库名。
%r	远端主机名或者 IP 地址以及远端端口，在不启动 log_hostname 时显示 IP 地址及远端端口。
%h	远端主机名或者 IP 地址，在不启动 log_hostname 时只显示 IP 地址。
%p	线程 ID。
%t	时间戳（没有毫秒）。
%m	带毫秒的时间戳。

转义字符	效果
%n	表示指定错误从哪个节点上报的。
%i	命令标签：会话当前执行的命令类型。
%e	SQLSTATE 错误码。
%c	会话 ID，详见说明。
%l	每个会话或线程的日志编号，从 1 开始。
%s	进程启动时间。
%v	虚拟事务 ID（backendID/ localXID）
%x	事务 ID（0 表示没有分配事务 ID）。
%q	不产生任何输出。如果当前线程是后端线程，忽略这个转义序列，继续处理后面的转义序列；如果当前线程不是后端线程，忽略这个转义序列和它后面的所有转义序列。
%%	字符% 。

说明

转义字符%c 打印一个独一无二的会话 ID，由两个 4 字节的十六进制数组成，通过字符“.”分开。这两个十六进制数分别表示进程的启动时间及进程编号，所以%c 也可以看作是保存打印这些名目的途径的空间。比如，从 pg_stat_activity 中产生会话 ID，可以用下面的查询：

```
SELECT to_hex(EXTRACT(EPOCH FROM backend_start)::integer) || '.' ||
       to_hex(pid)
FROM pg_stat_activity;
```

- 当 log_line_prefix 设置为空值时，请将其最后一个字符作为一个独立的段，以此来直观地与后续的日志进行区分，也可以使用一个标点符号。
- Syslog 生成自己的时间戳及进程 ID 信息，所以当登录日志时，不需要包含这些转义字符。

取值范围：字符串

默认值：%m %c %d %p %a %x %n %e

说明

%m %c %d %p %a %x %n %e 表示在日志开头附加会话开始时间戳，会话 ID，数据库名，线程 ID，应用程序名，事务 ID，报错节点，SQLSTATE 错误码。

log_lock_waits

参数说明：当一个会话的等待获得一个锁的时间超过 deadlock_timeout 的值时，此选项控制在数据库日志中记录此消息。这对于决定锁等待是否会产生一个坏的行为是非常有用的。

该参数属于 SUSET 类型参数，请参考表 10-1 中对应设置方法进行设置。

取值范围：布尔型

- on 表示记录此信息。
- off 表示不记录此信息。

默认值：off

log_statement

参数说明：控制记录 SQL 语句。对于使用扩展查询协议的客户端，记录接收到执行消息的事件和绑定参数的值（内置单引号要双写）。

该参数属于 SUSET 类型参数，请参考表 10-1 中对应设置方法进行设置。

须知

即使 log_statement 设置为 all，包含简单语法错误的语句也不会被记录，因为仅在完成基本的语法分析并确定了语句类型之后才记录日志。在使用扩展查询协议的情况下，在执行阶段之前（语法分析或规划阶段）同样不会记录。将 log_min_error_statement 设为 ERROR 或更低才能记录这些语句。

取值范围：枚举类型

- none 表示不记录语句。
- ddl 表示记录所有的数据定义语句，比如 CREATE、ALTER 和 DROP 语句。
- mod 表示记录所有 DDL 语句，还包括数据修改语句 INSERT、UPDATE、DELETE、TRUNCATE 和 COPY FROM 。
- all 表示记录所有语句，PREPARE、EXECUTE 和 EXPLAIN ANALYZE 语句也同样被记录。

默认值：none

log_temp_files

参数说明：控制记录临时文件的删除信息。临时文件可以用来排序、哈希及临时查询结果。当一个临时文件被删除时，将会产生一条日志消息。

该参数属于 SUSET 类型参数，请参考表 10-1 中对应设置方法进行设置。

取值范围：整型，最小值为-1，最大值 2147483647，单位 KB。

- 正整数表示只记录比 log_temp_files 设定值大的临时文件的删除信息。
- 值 0 表示记录所有的临时文件的删除信息。
- 值-1 表示不记录任何临时文件的删除信息。

默认值：-1

log_timezone

参数说明：设置服务器写日志文件时使用的时区。与 TimeZone 不同，这个值是数据库范围的，针对所有连接到本数据库的会话生效。

该参数属于 SIGHUP 类型参数，请参考表 10-1 中对应设置方法进行设置。

取值范围：字符串

默认值：PRC

说明

vb_initdb 进行相应系统环境设置时会对默认值进行修改。

logging_module

参数说明：用于设置或者显示模块日志在服务端的可输出性。该参数属于会话级参数。

该参数属于 USERSET 类型参数，设置请参考表 10-1 中对应设置的方法进行设置。

取值范围：字符串

默认值：所有模块日志在服务端是不输出的，可由 SHOW logging_module 查看。为 ALL,on(),off(DFS,GUC,ORC,SLRU,MEM_CTL,AUTOVAC,CACHE,ADIO,SSL,TBLSPC,WLM,EXECUTOR,OPFUSION,VEC_EXECUTOR,LLVM,OPT,OPT_REWRITE,OPT_JOIN,OPT_AGG,OPT_SUBPLAN,OPT_SETOP,OPT_SKEW,UDF,COOP_ANALYZE,WLMCP,ACCELERATE,,PLANHINT,SNAPSHOT,XACT,HANDLE,CLOG,EC,REMOTE,CN_RETRY,PLSQL,TEXTSEARCH,SEQ,REDO,FUNCTION,PARSER,INSTR,INCRE_CKPT,DBL_WRT,RTO,HEARTBEAT)。

设置方法：首先，可以通过 SHOW logging_module 来查看哪些模块是支持可控制的。例如，查询输出结果为：

```
vastbase=# show logging_module;
logging_module
------------------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------------------
-----------
ALL,on(),off(DFS,GUC,ORC,SLRU,MEM_CTL,AUTOVAC,CACHE,ADIO,SSL,TBLSPC,WLM,EXECUTOR,VEC_EXECUTOR,LLV
M,OPT,OPT_REWRITE,OPT_JOIN,OPT_AGG,OPT_SUBPLAN,OPT_SETOP,OPT_SKEW,UDF,COOP_ANALYZE,WLMCP,ACCELERA
TE,T,PLANHINT,SNAPSHOT,XACT,HANDLE,CLOG,EC,REMOTE,CN_RETRY,PLSQL,TEXTSEARCH,SEQ,REDO,FUNCTION,PAR
SER,INSTR,INCRE_CKPT,DBL_WRT,RTO,HEARTBEAT)
(1 row)
```

支持可控制的模块使用大写来标识，特殊标识 ALL 用于对所有模块日志进行设置。可以使用 on/off 来控制模块日志的输出。设置 SSL 模块日志为可输出，使用如下命令：

```
vastbase=# set logging_module='on(SSL)';
SET
vastbase=# show logging_module;
logging_module
------------------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------------------
----------
ALL,on(SSL),off(DFS,GUC,ORC,SLRU,MEM_CTL,AUTOVAC,CACHE,ADIO,TBLSPC,WLM,EXECUTOR,VEC_EXECUTOR,LLVM,
OPT,OPT_REWRITE,OPT_JOIN,OPT_AGG,OPT_SUBPLAN,OPT_SETOP,OPT_SKEW,UDF,COOP_ANALYZE,WLMCP,ACCELERATE,
,PLANHINT,SNAPSHOT,XACT,HANDLE,CLOG,EC,REMOTE,CN_RETRY,PLSQL,TEXTSEARCH,SEQ,REDO,FUNCTION,PARSER,
INSTR,INCRE_CKPT,DBL_WRT,RTO,HEARTBEAT,COMM_IPC,COMM_PARAM)
(1 row)
```

可以看到模块 SSL 的日志输出被打开。

ALL 标识是相当于一个快捷操作，即对所有模块的日志可输出进行开启或关闭。

```
vastbase=# set logging_module='off(ALL)';
SET
vastbase=# show logging_module;
logging_module
------------------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------------------
-----------
ALL,on(),off(DFS,GUC,ORC,SLRU,MEM_CTL,AUTOVAC,CACHE,ADIO,SSL,TBLSPC,WLM,EXECUTOR,VEC_EXECUTOR,LLV
M,OPT,OPT_REWRITE,OPT_JOIN,OPT_AGG,OPT_SUBPLAN,OPT_SETOP,OPT_SKEW,UDF,COOP_ANALYZE,WLMCP,ACCELERA
TE,PLANHINT,SNAPSHOT,XACT,HANDLE,CLOG,EC,REMOTE,CN_RETRY,PLSQL,TEXTSEARCH,SEQ,REDO,FUNCTION,PARSE
R,INSTR,INCRE_CKPT,DBL_WRT,RTO,HEARTBEAT)
(1 row)

vastbase=# set logging_module='on(ALL)';
SET
vastbase=# show logging_module;
logging_module
------------------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------------------
-----------
ALL,on(DFS,GUC,ORC,SLRU,MEM_CTL,AUTOVAC,CACHE,ADIO,SSL,TBLSPC,WLM,EXECUTOR,VEC_EXECUTOR,LLVM,OPT,
OPT_REWRITE,OPT_JOIN,OPT_AGG,OPT_SUBPLAN,OPT_SETOP,OPT_SKEW,UDF,COOP_ANALYZE,WLMCP,ACCELERATE,PLA
NHINT,SNAPSHOT,XACT,HANDLE,CLOG,EC,REMOTE,CN_RETRY,PLSQL,TEXTSEARCH,SEQ,REDO,FUNCTION,PARSER,INST
R,INCRE_CKPT,DBL_WRT,RTO,HEARTBEAT),off()
(1 row)
```

依赖关系：该参数依赖于 log_min_level 参数的设置

opfusion_debug_mode

参数说明：用于调试简单查询是否进行查询优化。设置成 log 级别可以在数据库节点的执行计划中看到没有查询优化的具体原因。

该参数属于 USERSET 类型参数，请参考表 10-1 中对应设置方法进行设置。

取值范围：枚举类型

- off 表示不打开该功能。
- log 表示打开该功能，可以在数据库节点的执行计划中看到没有查询优化的具体原因。

须知

提供在 log 中显示语句没有查询优化的具体原因，需要将参数设置成 log 级别，log_min_messages 设置成 debug4 级别，logging_module 设置'on(OPFUSION)'，注意 log 内容可能会比较多，尽可能在调优期间执行少量作业使用。

默认值：off

19.9.4. 使用 CSV 格式写日志

前提条件

- log_destination 的值设置为 csvlog。
- logging_collector 的值设置为 on。

csvlog 定义

以“逗号分隔值” 即 CSV（Comma Separated Value）的形式发出日志。

以下是简单的用来存储 CSV 形式日志输出的表定义：

```
CREATE TABLE postgres_log
(
log_time timestamp(3) with time zone,
node_name text,
user_name text,
database_name text,
process_id bigint,
connection_from text,
"session_id" text,
session_line_num bigint,
command_tag text,
session_start_time timestamp with time zone,
virtual_transaction_id text,
transaction_id bigint,
query_id bigint,
module text,
error_severity text,
sql_state_code text,
message text,
detail text,
hint text,
internal_query text,
internal_query_pos integer,
context text,
query text,
query_pos integer,
location text,
application_name text
);
```

详细说明请参见表 19-3。

表 19-3. csvlog 字段含义表

字段名	字段含义	字段名	字段含义
log_time	毫秒级的时间戳	module	日志所属模块
node_name	节点名称	error_severity	ERRORSTATE 代码

字段名	字段含义	字段名	字段含义
user_name	用户名	sql_state_code	SQLSTATE 代码
database_name	数据库名	message	错误消息
process_id	进程 ID	detail	详细错误消息
connection_from	客户主机：端口号	hint	提示
session_id	会话 ID	internal_query	内部查询（查询那些导致错误的信息，如果有的话）
session_line_num	每个会话的行数	internal_query_pos	内部查询指针
command_tag	命令标签	context	环境
session_start_time	会话开始时间	query	错误发生位置的字符统计
virtual_transaction_id	常规事务	query_pos	错误发生位置指针
transaction_id	事务 ID	location	在 Vastbase 源代码中错误的位置（如果 log_error_verbosity 的值设为 verbose ）
query_id	查询 ID	application_name	应用名称

使用 COPY FROM 命令将日志文件导入这个表：

```
COPY postgres_log FROM '/opt/data/pg_log/logfile.csv' WITH csv;
```

说明

此处的日志名“logfile.csv”要换成实际生成的日志的名称。

简化输入

简化输入到 CSV 日志文件，可以通过如下操作：

- 设置 log_filename 和 log_rotation_age，为日志文件提供一个一致的、可预测的命名方案。通过日志文件名，预测一个独立的日志文件完成并进入准备导入状态的时间。
- 将 log_rotation_size 设为 0 来终止基于尺寸的日志回滚，因为基于尺寸的日志回滚让预测日志文件名变得非常的困难。

- 将log_truncate_on_rotation设为 on 以便区分在同一日志文件中旧的日志数据和新的日志数据。

19.10. 告警检测

在 Vastbase 运行的过程中，会对数据库中的错误场景进行检测，便于用户及早感知到 Vastbase 的错误。

enable_alarm

参数说明：允许打开告警检测线程，检测数据库中可能的错误场景。

该参数属于 POSTMASTER 类型参数，请参考表 10-1 中对应设置方法进行设置。

取值范围：布尔型

- on 表示允许打开告警检测线程。
- off 表示不允许打开告警检测线程。

默认值：on

connection_alarm_rate

参数说明：允许和数据库连接的最大并发连接数的比率限制。数据库连接的最大并发连接数为max_connections* connection_alarm_rate。

该参数属于 SIGHUP 类型参数，请参考表 10-1 中对应设置方法进行设置。

取值范围：浮点型，0.0~1.0

默认值：0.9

alarm_report_interval

参数说明：指定告警上报的时间间隔。

该参数属于 SIGHUP 类型参数，请参考表 10-1 中对应设置方法进行设置。

取值范围：整型，单位为秒。

默认值：10

alarm_component

参数说明：在对告警做上报时，会进行告警抑制，即同一个实例的同一个告警项在alarm_report_interval（默认值为 10s）内不做重复上报。在这种情况下设置用于处理告警内容的告警组件的位置。

该参数属于 POSTMASTER 类型参数，请参考表 10-1 中对应设置方法进行设置。

取值范围：字符串

默认值：/opt/huawei/snas/bin/snas_cm_cmd

19.11. 运行时统计

19.11.1.查询和索引统计收集器

查询和索引统计收集器负责收集数据库系统运行中的统计数据，如在一个表和索引上进行了多少次插入与更新操作、磁盘块的数量和元组的数量、每个表上最近一次执行清理和分析操作的时间等。可以通过查询系统视图 pg_stats 和 pg_statistic 查看统计数据。下面的参数设置服务器范围内的统计收集特性。

track_activities

参数说明：控制收集每个会话中当前正在执行命令的统计数据。

该参数属于 SUSET 类型参数，请参考表 10-1 中对应设置方法进行设置。

取值范围：布尔型

- on 表示开启收集功能。
- off 表示关闭收集功能。

默认值：on

track_counts

参数说明：控制收集数据库活动的统计数据。

该参数属于 SUSET 类型参数，请参考表 10-1 中对应设置方法进行设置。

取值范围：布尔型

- on 表示开启收集功能。
- off 表示关闭收集功能。

说明

在 AutoVacuum 自动清理进程中选择清理的数据库时，需要数据库的统计数据，故默认值设为 on。

默认值：on

track_io_timing

参数说明：控制收集数据库 I/O 调用时序的统计数据。I/O 时序统计数据可以在 pg_stat_database 中查询。

该参数属于 SUSET 类型参数，请参考表 10-1 中对应设置方法进行设置。

取值范围：布尔型

- on 表示开启收集功能，开启时，收集器会在重复地去查询当前时间的操作系统，这可能会引起某些平台的重大开销，故默认值设置为 off。
- off 表示关闭收集功能。

默认值：off

track_functions

参数说明：控制收集函数的调用次数和调用耗时的统计数据。

该参数属于 SUSET 类型参数，请参考表 10-1 中对应设置方法进行设置。

须知

当 SQL 语言函数设置为调用查询的“内联”函数时，不管是否设置此选项，这些 SQL 语言函数无法被追踪到。

取值范围：枚举类型

- pl 表示只追踪过程语言函数。
- all 表示追踪 SQL 语言函数。
- none 表示关闭函数追踪功能。

默认值：none

track_activity_query_size

参数说明：设置用于跟踪每一个活动会话的当前正在执行命令的字节数。

该参数属于 POSTMASTER 类型参数，请参考表 10-1 中对应设置方法进行设置。

取值范围：整型，100～102400

默认值：1024

update_process_title

参数说明：控制收集因每次服务器接收到一个新的 SQL 语句时而产生的进程名称更新的统计数据。进程名称可以通过 ps 命令进行查看。

该参数属于 SUSET 类型参数，请参考表 10-1 中对应设置方法进行设置。

取值范围：布尔型

- on 表示开启收集功能。
- off 表示关闭收集功能。

默认值：off

stats_temp_directory

参数说明：设置存储临时统计数据的目录。

该参数属于 SIGHUP 类型参数，请参考表 10-1 中对应设置方法进行设置。

须知

将其设置为一个基于 RAM 的文件系统目录会减少实际的 I/O 开销并可以提升其性能。

取值范围：字符串

默认值：pg_stat_tmp

track_thread_wait_status_interval

参数说明：用来定期收集 thread 状态信息的时间间隔。

该参数属于 SUSET 类型参数，请参考表 10-1 中对应设置方法进行设置。

取值范围：0～1 天，单位为 min。

默认值：30min

enable_save_datachanged_timestamp

参数说明：确定是否收集 insert/update/delete, exchange/truncate/drop partition 操作对表数据改动的时间。

该参数属于 USERSET 类型参数，请参考表 10-1 中对应设置方法进行设置。

取值范围：布尔型

- on 表示允许收集相关操作对表数据改动的时间。
- off 表示禁止收集相关操作对表数据改动的时间。

默认值：on

track_sql_count

参数说明：控制对每个会话中当前正在执行的 SELECT、INSERT、UPDATE、DELETE、MERGE INTO 语句进行计数的统计数据。

该参数属于 SUSET 类型参数，请参考表 10-1 中对应设置方法进行设置。

取值范围：布尔型

- on 表示开启计数功能。
- off 表示关闭计数功能。

默认值：**on**

说明

- track_sql_count 参数受 track_activities 约束：
- track_activities 开启而 track_sql_count 关闭时，如果查询了 gs_sql_count 或 pgxc_sql_count 视图，日志中将会有 WARNING 提示 track_sql_count 是关闭的；
- track_activities 和 track_sql_count 同时关闭，那么此时日志中将会有两条 WARNING，分别提示 track_activities 是关闭的和 track_sql_count 是关闭的；
- track_activities 关闭而 track_sql_count 开启，此时日志中将仅有 WARNING 提示 track_activities 是关闭。
- 当参数关闭时，查询视图的结果为 0 行。

19.11.2.性能统计

在数据库运行过程中，会涉及到锁的访问、磁盘 IO 操作、无效消息的处理，这些操作都可能是数据库的性能瓶颈，通过 Vastbase 提供的性能统计方法，可以方便定位性能问题。

输出性能统计日志

参数说明：对每条查询，以下 4 个选项控制在服务器日志里记录相应模块的性能统计数据，具体含义如下：

- log_parser_stats 控制在服务器日志里记录解析器的性能统计数据。
- log_planner_stats 控制在服务器日志里记录查询优化器的性能统计数据。
- log_executor_stats 控制在服务器日志里记录执行器的性能统计数据。
- log_statement_stats 控制在服务器日志里记录整个语句的性能统计数据。

这些参数只能辅助管理员进行粗略分析，类似 Linux 中的操作系统工具 getrusage() 。

这些参数属于 SUSET 类型参数，请参考表 10-1 中对应设置方法进行设置。

须知

- log_statement_stats 记录总的语句统计数据，而其他的只记录针对每个模块的统计数据。
- log_statement_stats 不能和其他任何针对每个模块统计的选项一起打开。

取值范围：布尔型

- on 表示开启记录性能统计数据的功能。
- off 表示关闭记录性能统计数据的功能。

默认值：off

19.12. 负载管理

未对数据库资源做控制时，容易出现并发任务抢占资源导致操作系统过载甚至最终崩溃。操作系统过载时，其响应用户任务的速度会变慢甚至无响应；操作系统崩溃时，整个系统将无法对用户提供任何服务。Vastbase 的负载管理功能能够基于可用资源的多少均衡数据库的负载，以避免数据库系统过载。

use_workload_manager

参数说明：是否开启资源管理功能。

该参数属于 SIGHUP 类型参数，请参考表 10-1 中对应设置方法进行设置。

取值范围：布尔型

- on 表示打开资源管理。
- off 表示关闭资源管理。

说明

- 当使用表 10-1 中的方式二来修改参数值时，新参数值只能对更改操作执行后启动的线程生效。此外，对于后台线程以及线程复用执行的新作业，该参数值的改动不会生效。如果希望这类线程即时识别参数变化，可以使用 kill session 或重启节点的方式来实现。
- use_workload_manager 参数由 off 变为 on 状态后，不会统计 off 时的存储资源。如果需要统计 off 时用户使用的存储资源，请在数据库中执行以下命令：

```
select gs_wlm_readjust_user_space(0);
```

默认值：on

cgroup_name

参数说明：设置当前使用的 Cgroups 的名称或者调整当前 group 下排队的优先级。

即如果先设置 cgroup_name，再设置 session_respool，那么 session_respool 关联的控制组起作用，如果再切换 cgroup_name，那么新切换的 cgroup_name 起作用。

切换 cgroup_name 的过程中如果指定到 Workload 控制组级别，数据库不对级别进行验证。级别的范围只要在 1-10 范围内都可以。

该参数属于 USERSET 类型参数，请参考表 10-1 中方式三的方法进行设置。

建议尽量不要混合使用 cgroup_name 和 session_respool。

取值范围：字符串

默认值：InvalidGroup

cpu_collect_timer

参数说明：设置语句执行时在数据库节点上收集 CPU 时间的周期。

数据库管理员需根据系统资源（如 CPU 资源、IO 资源和内存资源）情况，调整此数值大小，使得系统支持较合适的收集周期，太小会影响执行效率，太大会影响异常处理的精确度。

该参数属于 SIGHUP 类型参数，请参考表 10-1 中对应设置方法进行设置。

取值范围：整型，1 ~ INT_MAX， 单位为秒。

默认值：30

memory_tracking_mode

参数说明：设置记录内存信息的模式。

该参数属于 USERSET 类型参数，请参考表 10-1 中对应设置方法进行设置。

取值范围：

- none，不启动内存统计功能。
- normal，仅做内存实时统计，不生成文件。
- executor，生成统计文件，包含执行层使用过的所有已分配内存的上下文信息。
- fullexec，生成文件包含执行层申请过的所有内存上下文信息。

默认值：none

memory_detail_tracking

参数说明：设置需要的线程内分配内存上下文的顺序号以及当前线程所在 query 的 plannodeid。

该参数属于 USERSET 类型参数，请参考表 10-1 中对应设置方法进行设置。

取值范围：字符型

默认值：空

须知

该参数不允许用户进行设置，建议保持默认值。

---

enable_resource_track

参数说明：是否开启资源实时监控功能。

该参数属于 SIGHUP 类型参数，请参考表 10-1 中对应设置方法进行设置。

取值范围：布尔型

- on 表示打开资源监控。
- off 表示关闭资源监控。

默认值：on

enable_resource_record

参数说明：是否开启资源监控记录归档功能。

该参数属于 SIGHUP 类型参数，请参考表 10-1 中对应设置方法进行设置。

取值范围：布尔型

- on 表示开启资源监控记录归档功能。
- off 表示关闭资源监控记录归档功能。

默认值：off

enable_logical_io_statistics

参数说明：设置是否开启资源监控逻辑 IO 统计功能。开启时，对于 PG_TOTAL_USER_RESOURCE_INFO 视图中的 read_kbytes、write_kbytes、read_counts、write_counts、read_speed 和 write_speed 字段，会统计对应用户的逻辑读写字节数、次数以及速率。

该参数属于 SIGHUP 类型参数，请参考表 10-1 中对应设置方法进行设置。

取值范围：布尔型

- on 表示开启资源监控逻辑 IO 统计功能。
- off 表示关闭资源监控逻辑 IO 统计功能。

默认值：on

enable_user_metric_persistent

参数说明：设置是否开启用户历史资源监控转存功能。开启时，对于PG_TOTAL_USER_RESOURCE_INFO 视图中数据，会定期采样保存到14.2.6GS_WLM_USER_RESOURCE_HISTORY 系统表中。

该参数属于 SIGHUP 类型参数，请参考表 10-1 中对应设置方法进行设置。

取值范围：布尔型

- on 表示开启用户历史资源监控转存功能。
- off 表示关闭用户历史资源监控转存功能。

默认值：on

user_metric_retention_time

参数说明：设置用户历史资源监控数据的保存天数。该参数仅在 enable_user_metric_persistent 为 on 时有效。

该参数属于 USERSET 类型参数，请参考表 10-2 中的方法一和方法二进行设置。

取值范围：整型，0 ~ 730，单位为天。

- 值等于 0 时，用户历史资源监控数据将永久保存。
- 值大于 0 时，用户历史资源监控数据将保存对应天数。

默认值：7

enable_instance_metric_persistent

参数说明：设置是否开启实例资源监控转存功能。开启时，对实例的监控数据会保存到14.2.2GS_WLM_INSTANCE_HISTORY 系统表中。

该参数属于 SIGHUP 类型参数，请参考表 10-1 中对应设置方法进行设置。

取值范围：布尔型

- on 表示开启实例资源监控转存功能。
- off 表示关闭实例资源监控转存功能。

默认值：on

instance_metric_retention_time

参数说明：设置实例历史资源监控数据的保存天数。该参数仅在 enable_instance_metric_persistent 为 on 时有效。

该参数属于 USERSET 类型参数，请参考表 10-2 中的方法一和方法二进行设置。

取值范围：整型，0 ~ 3650，单位为天。

- 值等于 0 时，实例历史资源监控数据将永久保存。
- 值大于 0 时，实例历史资源监控数据将保存对应设置天数。

默认值：7

resource_track_level

参数说明：设置当前会话的资源监控的等级。该参数只有当参数 enable_resource_track 为 on 时才有效。

该参数属于 USERSET 类型参数，请参考表 10-1 中对应设置方法进行设置。

取值范围：枚举型

- none，不开启资源监控功能。
- query，开启 query 级别资源监控功能。
- operator，开启 query 级别和算子级别资源监控功能。

默认值：query

resource_track_cost

参数说明：设置对当前会话的语句进行资源监控的最小执行代价。该参数只有当参数 enable_resource_track 为 on 时才有效。

该参数属于 USERSET 类型参数，请参考表 10-1 中对应设置方法进行设置。

取值范围：整型，-1 ~ INT_MAX

- 值为-1 时，不进行资源监控。
- 值大于或等于 0 时，值大于或等于 0 且小于等于 9 时，对执行代价大于等于 10 的语句进行资源监控。
- 值大于或等于 10 时，对执行代价超过该参数值的语句进行资源监控。

默认值：100000

resource_track_duration

参数说明：设置资源监控实时视图中记录的语句执行结束后进行历史信息转存的最小执行时间。当执行完成的作业，其执行时间不小于此参数值时，作业信息会从实时视图（以 statistics 为后缀的视图）转存到相应的历史视图（以 history 为后缀的视图）中。该参数只有当 enable_resource_track 为 on 时才有效。

该参数属于 USERSET 类型参数，请参考表 10-1 中对应设置方法进行设置。

取值范围：整型，0 ~ INT_MAX，单位为秒。

- 值为 0 时，资源监控实时视图中记录的所有语句都进行历史信息归档。
- 值大于 0 时，资源监控实时视图中记录的语句的执行时间超过这个值就会进行历史信息归档。

默认值：1min

disable_memory_protect

参数说明：禁止内存保护功能。当系统内存不足时如果需要查询系统视图，可以先将此参数置为 on，禁止内存保护功能，保证视图可以正常查询。该参数只适用于在系统内存不足时进行系统诊断和调试，正常运行时请保持该参数配置为 off。

该参数属于 USERSET 类型参数，且只对当前会话有效。请参考表 10-1 中对应设置方法进行设置。

取值范围：布尔型

- on 表示禁止内存保护功能。
- off 表示启动内存保护功能。

默认值：off

query_band

参数说明：用于标示当前会话的作业类型，由用户自定义。

该参数属于 USERSET 类型参数，请参考表 10-1 中对应设置方法进行设置。

取值范围：字符型

默认值：空

enable_bbox_dump

参数说明：是否开启黑匣子功能，在系统不配置 core 机制的时候仍可产生 core 文件。

该参数属于 SIGHUP 类型参数，请参考表 10-1 中对应设置方法进行设置。

取值范围：布尔型

- on 表示打开黑匣子功能。
- off 表示关闭黑匣子功能。

默认值：off

bbox_dump_count

参数说明：在 bbox_dump_path 定义的路径下，允许存储的 Vastbase 所产生 core 文件最大数。超过此数量，旧的 core 文件会被删除。此参数只有当 enable_bbox_dump 为 on 时才生效。

该参数属于 USERSET 类型参数，请参考表 10-1 中对应设置方法进行设置。

取值范围：整型，1～20

默认值：8

说明

在并发产生 core 文件时，core 文件的产生个数可能大于 bbox_dump_count。

bbox_dump_path

参数说明：黑匣子 core 文件的生成路径。此参数只有当 enable_bbox_dump 为 on 时才生效。

该参数属于 SIGHUP 类型参数，请参考表 10-1 中对应设置方法进行设置。

取值范围：字符型

默认值：空。默认生成黑匣子 core 文件的路径为读取/proc/sys/Vastbase/core_pattern 下的路径，如果这个路径不是一个目录，或者用户对此目录没有写权限，黑匣子 core 文件将生成在数据库的 data 目录下。或者以安装时指定的目录为准。

io_limits

参数说明：每秒触发 IO 的上限。

该参数属于 USERSET 类型参数，请参考表 10-1 中对应类型的设置的方法进行设置。

取值范围：整型，0～1073741823

默认值：0

io_priority

参数说明：IO 利用率高达 50%时，重消耗 IO 作业进行 IO 资源管控时关联的优先级等级。

该参数属于 USERSET 类型参数，请参考表 10-1 中对应类型的设置的方法进行设置。

取值范围：枚举型

- None: 表示不受控。
- Low: 表示限制 iops 为该作业原始触发数值的 10%。
- Medium: 表示限制 iops 为该作业原始触发数值的 20%。
- High: 表示限制 iops 为该作业原始触发数值的 50%。

默认值：None

io_control_unit

参数说明：行存场景下，io 管控时用来对 io 次数进行计数的单位。

该参数属于 SIGHUP 类型参数，请参考表 10-1 中对应类型的设置方法进行设置。

记多少次 io 触发为一计数单位，通过此计数单位所记录的次数进行 io 管控。

取值范围：整型，1000~1000000

默认值：6000

session_respool

参数说明：当前的 session 关联的 resource pool。

该参数属于 USERSET 类型参数，请参考表 10-1 中对应类型的设置方法进行设置。

即如果先设置 cgroup_name，再设置 session_respool，那么 session_respool 关联的控制组起作用，如果再切换 cgroup_name，那么新切换的 cgroup_name 起作用。

切换 cgroup_name 的过程中如果指定到 Workload 控制组级别，数据库不对级别进行验证。级别的范围只要在 1-10 范围内都可以。

建议尽量不要混合使用 cgroup_name 和 session_respool。

取值范围：string 类型，通过 create resource pool 所设置的资源池。

默认值：invalid_pool

session_statistics_memory

参数说明：设置实时查询视图的内存大小。

该参数属于 SIGHUP 类型参数，请参考表 10-1 中对应设置方法进行设置。

取值范围：整型，5 * 1024 ~ max_process_memory 的 50%，单位 KB。

默认值：5MB

topsql_retention_time

参数说明：设置历史 TopSQL 中 gs_wlm_operator_info 表中数据 的保存时间。

该参数属于 SIGHUP 类型参数，请参考表 10-1 中对应设置方法进行设置。

取值范围：整型，0 ~ 730，单位为天。

- 值为 0 时，表示数据永久保存。
- 值大于 0 时，表示数据能够保存的对应天数。

默认值：0

session_history_memory

参数说明：设置历史查询视图的内存大小。

该参数属于 SIGHUP 类型参数，请参考表 10-1 中对应设置方法进行设置。

取值范围：整型，10 * 1024 ~ max_process_memory 的 50%，单位 KB。

默认值：10MB

transaction_pending_time

参数说明：事务块语句和存储过程语句排队的最大时间。

该参数属于 USERSET 类型参数，请参考表 10-1 中对应设置方法进行设置。

取值范围：整型，-1 ~ INT_MAX/2，单位为秒。

- 值为-1 或 0：事务块语句和存储过程语句无超时判断，排队至资源满足可执行条件。
- 值大于 0：事务块语句和存储过程语句排队超过所设数值的时间后，无视当前资源情况强制执行。

默认值：0

19.13. 自动清理

系统自动清理进程（autovacuum）自动执行 VACUUM 和 ANALYZE 命令，回收被标识为删除状态的记录空间，并更新表的统计数据。

autovacuum

参数说明：控制数据库自动清理进程（autovacuum）的启动。自动清理进程运行的前提是将 track_counts 设置为 on。

该参数属于 SIGHUP 类型参数，请参考表 10-2 中对应设置方法进行设置。

说明

- 如果希望系统在故障恢复后，具备自动清理两阶段事务的功能，请将 autovacuum 设置为 on;
- 当设置 autovacuum 为 on，autovacuum_max_workers 为 0 时，表示系统不会自动进行 autovacuum，只会在故障恢复后，自动清理两阶段事务；
- 当设置 autovacuum 为 on，autovacuum_max_workers 大于 0 时，表示系统不仅在故障恢复后，自动清理两阶段事务，并且还可以自动清理进程。

须知

即使此参数设置为 off，当事务 ID 回绕即将发生时，数据库也会自动启动自动清理进程。对于 create/drop database 发生异常时，可能有的节点提交或回滚，有的节点未提交（prepared 状态），此时系统不能自动修复，需要手动修复，修复步骤：

6. 使用 gs_clean 工具（-N 参数）查询出异常两阶段事务的 xid 以及处于 prepared 的节点；
7. 登录事务处于 prepared 状态的节点，系统管理员连接一个可用的数据库（如 postgres），执行语句 set xc_maintenance_mode = on；
8. 根据事务全局状态提交或者回滚此两阶段事务（如提交语句；回滚语句）。

取值范围：布尔型

- on 表示开启数据库自动清理进程。
- off 表示关闭数据库自动清理进程。

默认值：on

autovacuum_mode

参数说明：该参数仅在 autovacuum 设置为 on 的场景下生效，它控制 autoanalyze 或 autovacuum 的打开情况。

该参数属于 SIGHUP 类型参数，请参考表 10-2 中对应设置方法进行设置。

取值范围：枚举类型

- analyze 表示只做 autoanalyze。
- vacuum 表示只做 autovacuum。
- mix 表示 autoanalyze 和 autovacuum 都做。
- none 表示二者都不做。

默认值：mix

autoanalyze_timeout

参数说明：设置 autoanalyze 的超时时间。在对某张表做 autoanalyze 时，如果该表的 analyze 时长超过了 autoanalyze_timeout，则自动取消该表此次 analyze。

该参数属于 SIGHUP 类型参数，请参考表 10-2 中对应设置方法进行设置。

取值范围：int 类型，单位是 s，0~2147483。

默认值：5min（即 300s）

autovacuum_io_limits

参数说明：控制 autovacuum 进程每秒触发 IO 的上限。

该参数属于 SIGHUP 类型参数，请参考表 10-2 中对应设置方法进行设置。

取值范围：整型，0～1073741823 和-1。其中-1 表示不控制，而是使用系统默认控制组。

默认值：-1

log_autovacuum_min_duration

参数说明：当自动清理的执行时间大于或者等于某个特定的值时，向服务器日志中记录自动清理执行的每一步操作。设置此选项有助于追踪自动清理的行为。

该参数属于 SIGHUP 类型参数，请参考表 10-2 中对应设置方法进行设置。

举例如下：

将 log_autovacuum_min_duration 设置为 250ms，记录所有运行大于或者等于 250ms 的自动清理命令的相关信息。

取值范围：整型，最小值为-1，最大值为 2147483647，单位为毫秒。

- 当参数设置为 0 时，表示所有的自动清理操作都记录到日志中。
- 当参数设置为-1 时，表示所有的自动清理操作都不记录到日志中。
- 当参数设置为非-1 时，当由于锁冲突的存在导致一个自动清理操作被跳过，记录一条消息。

默认值：-1

autovacuum_max_workers

参数说明：设置能同时运行的自动清理线程的最大数量，该参数的取值上限与 GUC 参数 max_connections 和 job_queue_processes 大小有关。

该参数属于 POSTMASTER 类型参数，请参考表 10-2 中对应设置方法进行设置。

取值范围：整型，最小值为 0（表示不会自动进行 autovacuum），理论最大值为 262143，实际最大值为动态值，计算公式为“262143 - max_connections - job_queue_processes - 辅助线程数 – autovacuum 的 lancher 线程数 - 1”，其中辅助线程数和 autovacuum 的 lancher 线程数由两个宏来指定，当前版本的默认值分别为 20 和 2。

默认值：3

autovacuum_naptime

参数说明：设置两次自动清理操作的时间间隔。

该参数属于 SIGHUP 类型参数，请参考表 10-2 中对应设置方法进行设置。

取值范围：整型，单位为 s，最小值为 1，最大值为 2147483。

默认值：10min（即 600s）

autovacuum_vacuum_threshold

参数说明：设置触发 VACUUM 的阈值。当表上被删除或更新的记录数超过设定的阈值时才会对这个表执行 VACUUM 操作。

该参数属于 SIGHUP 类型参数，请参考表 10-2 中对应设置方法进行设置。

取值范围：整型，最小值为 0，最大值为 2147483647。

默认值：50

autovacuum_analyze_threshold

参数说明：设置触发 ANALYZE 操作的阈值。当表上被删除、插入或更新的记录数超过设定的阈值时才会对这个表执行 ANALYZE 操作。

该参数属于 SIGHUP 类型参数，请参考表 10-2 中对应设置方法进行设置。

取值范围：整型，最小值为 0，最大值为 2147483647。

默认值：50

autovacuum_vacuum_scale_factor

参数说明：设置触发一个 VACUUM 时增加到 autovacuum_vacuum_threshold 的表大小的缩放系数。

该参数属于 SIGHUP 类型参数，请参考表 10-2 中对应设置方法进行设置。

取值范围：浮点型，0.0 ~ 100.0

默认值：0.2

autovacuum_analyze_scale_factor

参数说明：设置触发一个 ANALYZE 时增加到 autovacuum_analyze_threshold 的表大小的缩放系数。

该参数属于 SIGHUP 类型参数，请参考表 10-2 中对应设置方法进行设置。

取值范围：浮点型，0.0 ~ 100.0

默认值：0.1

autovacuum_freeze_max_age

参数说明：设置事务内的最大时间，使得表的 pg_class.relfrozenxid 字段在 VACUUM 操作执行之前被写入。

- VACUUM 也可以删除 pg_clog/子目录中的旧文件。
- 即使自动清理进程被禁止，系统也会调用自动清理进程来防止循环重复。

该参数属于 POSTMASTER 类型参数，请参考表 10-2 中对应设置方法进行设置。

取值范围：**长**整型，100 000 ~ 576 460 752 303 423 487

默认值：20000000000

autovacuum_vacuum_cost_delay

参数说明：设置在自动 VACUUM 操作里使用的开销延迟数值。

该参数属于 SIGHUP 类型参数，请参考表 10-2 中对应设置方法进行设置。

取值范围：整型，-1～100，单位为毫秒（ms）。其中-1 表示使用常规的 vacuum_cost_delay。

默认值：20ms

autovacuum_vacuum_cost_limit

参数说明：设置在自动 VACUUM 操作里使用的开销限制数值。

该参数属于 SIGHUP 类型参数，请参考表 10-2 中对应设置方法进行设置。

取值范围：整型，-1～10000。其中-1 表示使用常规的 vacuum_cost_limit。

默认值：-1

twophase_clean_workers

参数说明：该参数用来控制内核调度 gs_clean 工具的并发清理数。

该参数属于 SIGHUP 类型参数，请参考表 10-2 中对应设置方法进行设置。

取值范围：整型，1～10

默认值：3

defer_csn_cleanup_time

参数说明：用来指定本地回收时间间隔，单位为毫秒（ms）。

取值范围：整型，0~INT_MAX。

默认值：5s（即 5000ms）

19.14. 客户端连接缺省设置

19.14.1.语句行为

介绍 SQL 语句执行过程的相关默认参数。

search_path

参数说明：当一个被引用对象没有指定模式时，此参数设置模式搜索顺序。它的值由一个或多个模式名构成，不同的模式名用逗号隔开。

该参数属于 USERSET 类型参数，请参考表 10-2 中对应设置方法进行设置。

- 当前会话如果存放临时表的模式时，可以使用别名 pg_temp 将它列在搜索路径中，如 'pg_temp，public' 。 存放临时表的模式始终会作为第一个被搜索的对象，排在 pg_catalog 和

search_path 中所有模式的前面，即具有第一搜索优先级。建议用户不要在 search_path 中显示设置 pg_temp。如果在 search_path 中指定了 pg_temp，但不是在最前面，系统会提示设置无效，pg_temp 仍被优先搜索。通过使用别名 pg_temp，系统只会在存放临时表的模式中搜索表、视图和数据类型这样的数据库对象，不会在里面搜索函数或运算符这样的数据库对象。

❖ 系统表所在的模式 pg_catalog，总是排在 search_path 中指定的所有模式前面被搜索，即具有第二搜索优先级（pg_temp 具有第一搜索优先级）。建议用户不要在 search_path 中显式设置 pg_catalog。如果在 search_path 中指定了 pg_catalog，但不是在最前面，系统会提示设置无效，pg_catalog 仍被第二优先搜索。

❖ 当没有指定一个特定模式而创建一个对象时，它们被放置到以 search_path 为命名的第一个有效模式中。当搜索路径为空时，会报错误。

❖ 通过 SQL 函数 current_schema 可以检测当前搜索路径的有效值。这和检测 search_path 的值不尽相同，因为 current_schema 显示 search_path 中首位有效的模式名称。

取值范围：字符串

说明

- 设置为"$user"，public 时，支持共享数据库（没有用户具有私有模式和所有共享使用 public），用户私有模式和这些功能的组合使用。可以通过改变默认搜索路径来获得其他效果，无论是全局化的还是私有化的。
- 设置为空串（''）的时候，系统会自动转换成一对双引号。
- 设置的内容中包含双引号，系统会认为是不安全字符，会将每个双引号转换成一对双引号。

默认值："$user",public

说明

$user 表示与当前会话用户名同名的模式名，如果这样的模式不存在，$user 将被忽略。

current_schema

参数说明：此参数设置当前的模式。

该参数属于 USERSET 类型参数，请参考表 10-2 中对应设置方法进行设置。

取值范围：字符串

默认值："$user",public

说明

$user 表示与当前会话用户名同名的模式名，如果这样的模式不存在，$user 将被忽略。

default_tablespace

参数说明：当 CREATE 命令没有明确声明表空间时，所创建对象(表和索引等)的缺省表空间。

❖ 值是一个表空间的名称或者一个表示使用当前数据库缺省表空间的空字符串。若指定的是一个非默认表空间，用户必须具有它的 CREATE 权限，否则尝试创建会失败。

❖ 临时表不使用此参数，可以用 temp_tablespaces 代替。

❖ 创建数据库时不使用此参数。默认情况下，一个新的数据库从模板数据库继承表空间配置。

该参数属于 USERSET 类型参数，请参考表 10-2 中对应设置方法进行设置。

取值范围：字符串，其中空表示使用默认表空间。

默认值：空

default_storage_nodegroup

参数说明：此参数设置当前的默认建表所在的 Node Group，目前只适用普通表。

该参数属于 USERSET 类型参数，请参考表 10-2 中对应设置方法进行设置。

- 值为“installation”表示建表会默认建在安装的 Node Group 上。
- 值为其他字符串表示建表会默认建在设置的 Node Group 上。

取值范围：字符串

默认值：installation

temp_tablespaces

参数说明：当一个 CREATE 命令没有明确指定一个表空间时，temp_tablespaces 指定了创建临时对象（临时表和临时表的索引）所在的表空间。在这些表空间中创建临时文件用来做大型数据的排序工作。

其值是一系列表空间名的列表。如果列表中有多个表空间时，每次临时对象的创建，Vastbase 会在列表中随机选择一个表空间；如果在事务中，连续创建的临时对象被放置在列表里连续的表空间中。如果选择的列表中的元素是一个空串，Vastbase 将自动将当前的数据库设为默认的表空间。

该参数属于 USERSET 类型参数，请参考表 10-2 中对应设置方法进行设置。

取值范围：字符串。空字符串表示所有的临时对象仅在当前数据库默认的表空间中创建，请参见 default_tablespace。

默认值：空

check_function_bodies

参数说明：设置是否在 CREATE FUNCTION 执行过程中进行函数体字符串的合法性验证。为了避免产生问题（比如避免从转储中恢复函数定义时向前引用的问题），偶尔会禁用验证。

该参数属于 USERSET 类型参数，请参考表 10-2 中对应设置方法进行设置。

取值范围：布尔型

- on 表示在 CREATE FUNCTION 执行过程中进行函数体字符串的合法性验证。
- off 表示在 CREATE FUNCTION 执行过程中不进行函数体字符串的合法性验证。

默认值：on

default_transaction_isolation

参数说明：设置默认的事务隔离级别。

该参数属于 USERSET 类型参数，请参考表 10-2 中对应设置方法进行设置。

取值范围：枚举类型

- read committed 表示事务读已提交。
- repeatable read 表示事务可重复读。
- serializable，Vastbase 目前功能上不支持此隔离级别，等价于 repeatable read。

默认值：read committed

default_transaction_read_only

参数说明：设置每个新创建事务是否是只读状态。

该参数属于 SIGHUP 类型参数，请参考表 10-2 中对应设置方法进行设置。

取值范围：布尔型

- on 表示只读状态。
- off 表示非只读状态。

默认值：off

default_transaction_deferrable

参数说明：控制每个新事务的默认延迟状态。只读事务或者那些比序列化更加低的隔离级别的事务除外。

Vastbase 不支持可串行化的隔离级别，因此，该参数无实际意义。

该参数属于 USERSET 类型参数，请参考表 10-2 中对应设置方法进行设置。

取值范围：布尔型

- on 表示默认延迟。
- off 表示默认不延迟。

默认值：off

session_replication_role

参数说明：控制当前会话与复制相关的触发器和规则的行为。

该参数属于 SUSET 类型参数，请参考表 10-2 中对应设置方法进行设置。

须知

设置此参数会丢弃之前任何缓存的执行计划。

取值范围：枚举类型

- origin 表示从当前会话中复制插入、删除、更新等操作。
- replica 表示从其他地方复制插入、删除、更新等操作到当前会话。
- local 表示函数执行复制时会检测当前登录数据库的角色并采取相应的操作。

默认值：origin

statement_timeout

参数说明：当语句执行时间超过该参数设置的时间（从服务器收到命令时开始计时）时，该语句将会报错并退出执行。

该参数属于 USERSET 类型参数，请参考表 10-2 中对应设置方法进行设置。

取值范围：整型，最小值为 0，最大值为 2147483647，单位为毫秒。

默认值：0

vacuum_freeze_min_age

参数说明：指定 VACUUM 在扫描一个表时用于判断是否用 FrozenXID 替换事务 ID 的中断寿命(在同一个事务中)。

该参数属于 USERSET 类型参数，请参考表 10-2 中对应设置方法进行设置。

取值范围：整型，0～576 460 752 303 423 487

说明

尽管随时可以将此参数设为 0 到 10 亿之间的任意值，但是，VACUUM 将默认其有效值范围限制在 autovacuum_freeze_max_age 的 50%以内。

默认值：5000000000

vacuum_freeze_table_age

参数说明：指定 VACUUM 对全表的扫描冻结元组的时间。如果表的 pg_class.relfrozenxid 字段的值已经达到了参数指定的时间，VACUUM 对全表进行扫描。

该参数属于 USERSET 类型参数，请参考表 10-2 中对应设置方法进行设置。

取值范围：整型，0～576 460 752 303 423 487

说明

尽管随时可以将此参数设为零到 20 亿之间的值，但是，VACUUM 将默认其有效值范围限制在 autovacuum_freeze_max_age 的 95%以内。定期的手动 VACUUM 可以在对此表的反重叠自动清理启动之前运行。

默认值：15000000000

bytea_output

参数说明：设置 bytea 类型值的输出格式。

该参数属于 USERSET 类型参数，请参考表 10-2 中对应设置方法进行设置。

取值范围：枚举类型

- hex：将二进制数据编码为每字节 2 位十六进制数字。
- escape：传统化的 PostgreSQL 格式。采用以 ASCII 字符序列表示二进制串的方法，同时将那些无法表示成 ASCII 字符的二进制串转换成特殊的转义序列。

默认值：hex

xmlbinary

参数说明：设置二进制值是如何在 XML 中进行编码的。

该参数属于 USERSET 类型参数，请参考表 10-2 中对应设置方法进行设置。

取值范围：枚举类型

- base64
- hex

默认值：base64

xmloption

参数说明：当 XML 和字符串值之间进行转换时，设置 document 或 content 是否是隐含的。

该参数属于 USERSET 类型参数，请参考表 10-2 中对应设置方法进行设置。

取值范围：枚举类型

- document：表示 HTML 格式的文档。
- content：普通的字符串。

默认值：content

max_compile_functions

参数说明：设置服务器存储的函数编译结果的最大数量。存储过多的函数和存储过程的编译结果可能占用很大内存。将此参数设置为一个合理的值，有助于减少内存占用，提升系统性能。

该参数属于 POSTMASTER 类型参数，请参考表 10-2 中对应设置方法进行设置。

取值范围：整型，取值必须大于等于 1。

默认值：1000

gin_pending_list_limit

参数说明：设置当 GIN 索引启用 fastupdate 时，pending list 容量的最大值。当 pending list 的容量大于设置值时，会把 pending list 中数据批量移动到 GIN 索引数据结构中以进行清理。单个 GIN 索引可通过更改索引存储参数覆盖此设置值。

该参数属于 USERSET 类型参数，请参考表 10-2 中对应设置方法进行设置。

取值范围：整型，最小值为 64，最大值为 INT_MAX，单位为 KB。

默认值：4MB

19.14.2. 区域和格式化

介绍时间格式设置的相关参数。

DateStyle

参数说明：设置日期和时间值的显示格式，以及有歧义的输入值的解析规则。

这个变量包含两个独立的加载部分：输出格式声明（ISO、Postgres、SQL、German）和输入输出的年/月/日顺序（DMY、MDY、YMD）。这两个可以独立设置或者一起设置。关键字 Euro 和 European 等价于 DMY；关键字 US、NonEuro、NonEuropean 等价于 MDY 。

该参数属于 USERSET 类型参数，请参考表 10-2 中对应设置方法进行设置。

取值范围：字符串

默认值：ISO, MDY

说明

vb_initdb 会将这个参数初始化成与 lc_time 一致的值。

设置建议：优先推荐使用 ISO 格式。Postgres、SQL 和 German 均采用字母缩写的形式来表示时区，例如 “EST、WST、CST” 等。这些缩写可同时指代不同的时区，比如 CST 可同时代表美国中部时间(Central Standard Time (USA) UT-6:00)、澳大利亚中部时间(Central Standard Time (Australia) UT+9:30)、中国标准时间(China Standard Time UT+8:00)、古巴标准时间(Cuba Standard Time UT-4:00)。这种情况下在时区转化时可能会得不到正确的结果，从而引发其他问题。

IntervalStyle

参数说明：设置区间值的显示格式。

该参数属于 USERSET 类型参数，请参考表 10-2 中对应设置方法进行设置。

取值范围：枚举类型

- sql_standard 表示产生与 SQL 标准规定匹配的输出。
- postgres 表示产生与 PostgreSQL 8.4 版本相匹配的输出，当 DateStyle 参数被设为 ISO 时。
- postgres_verbose 表示产生与 PostgreSQL 8.4 版本相匹配的输出，当 DateStyle 参数被设为 non_ISO 时。
- iso_8601 表示产生与在 ISO 8601 中定义的 “格式与代号” 相匹配的输出。
- a 表示与 numtodsinterval 函数相匹配的输出结果，详细请参考 numtodsinterval。

须知

IntervalStyle 参数也会影响不明确的间隔输入的说明。

默认值：postgres

TimeZone

参数说明：设置显示和解释时间类型数值时使用的时区。

该参数属于 USERSET 类型参数，请参考表 10-2 中对应设置方法进行设置。

取值范围：字符串，可查询视图 14.3.69PG_TIMEZONE_NAMES 获得。

默认值：PRC

说明

vb_initdb 将设置一个与其系统环境一致的时区值。

timezone_abbreviations

参数说明：设置服务器接受的时区缩写值。

该参数属于 USERSET 类型参数，请参考表 10-2 中对应设置方法进行设置。

取值范围：字符串，可查询视图 pg_timezone_names 获得。

默认值：Default

说明

Default 表示通用时区的缩写，适合绝大部分情况。但也可设置其他诸如 'Australia' 和 'India' 等用来定义特定的安装。而设置除此之外的时区缩写，需要在建数据库之前通过相应的配置文件进行设置。

extra_float_digits

参数说明：这个参数为浮点数值调整显示的数据位数，浮点类型包括 float4、float8 以及几何数据类型。参数值加在标准的数据位数上（FLT_DIG 或 DBL_DIG 中合适的）。

该参数属于 USERSET 类型参数，请参考表 10-2 中对应设置方法进行设置。

取值范围：整型，-15 ~ 3

说明

- 设置为 3，表示包括部分关键的数据位。这个功能对转储那些需要精确恢复的浮点数据特别有用。
- 设置为负数，表示消除不需要的数据位。

默认值：0

client_encoding

参数说明：设置客户端的字符编码类型。

请根据前端业务的情况确定。尽量客户端编码和服务器端编码一致，提高效率。

该参数属于 USERSET 类型参数，请参考表 10-2 中对应设置方法进行设置。

取值范围：兼容 PostgreSQL 所有的字符编码类型。其中 UTF8 表示使用数据库的字符编码类型。

说明

- 使用命令 locale -a 查看当前系统支持的区域和相应的编码格式，并可以选择进行设置。
- 默认情况下，vb_initdb 会根据当前的系统环境初始化此参数，通过 locale 命令可以查看当前的配置环境。
- 参数建议保持默认值，不建议直接在 postgresql.conf 文件中设置 client_encoding 参数，即使设置也不会生效，以保证 Vastbase 内部通信编码格式一致。

默认值：UTF8

推荐值：SQL_ASCII/UTF8

lc_messages

参数说明：设置信息显示的语言。

- ❖ 可接受的值是与系统相关的。
- ❖ 在一些系统上，这个区域范畴并不存在，不过仍然允许设置这个变量，只是不会有任何效果。同样，也有可能是所期望的语言的翻译信息不存在。在这种情况下，用户仍然能看到英文信息。

该参数属于 SUSET 类型参数，请参考表 10-2 中对应设置方法进行设置。

取值范围：字符串

说明

- 使用命令 locale -a 查看当前系统支持的区域和相应的编码格式，并可以选择进行设置。
- 默认情况下，vb_initdb 会根据当前的系统环境初始化此参数，通过 locale 命令可以查看当前的配置环境。
- 默认情况下，数据库会话提示符为英语。调整参数值为“zh_CN.UTF8”，可将会话提示符转为中文，且操作系统支持中文输出。

默认值：C

lc_monetary

参数说明：设置货币值的显示格式，影响 to_char 之类的函数的输出。可接受的值是系统相关的。该参数属于 USERSET 类型参数，请参考表 10-2 中对应设置方法进行设置。

取值范围：字符串

说明

- 使用命令 locale -a 查看当前系统支持的区域和相应的编码格式，并可以选择进行设置。
- 默认情况下，vb_initdb 会根据当前的系统环境初始化此参数，通过 locale 命令可以查看当前的配置环境。

默认值：C

lc_numeric

参数说明：设置数值的显示格式，影响 to_char 之类的函数的输出。可接受的值是系统相关的。该参数属于 USERSET 类型参数，请参考表 10-2 中对应设置方法进行设置。

取值范围：字符串

说明

- 使用命令 locale -a 查看当前系统支持的区域和相应的编码格式，并可以选择进行设置。
- 默认情况下，vb_initdb 会根据当前的系统环境初始化此参数，通过 locale 命令可以查看当前的配置环境。

默认值：C

lc_time

参数说明：设置时间和区域的显示格式，影响 to_char 之类的函数的输出。可接受的值是系统相关的。该参数属于 USERSET 类型参数，请参考表 10-2 中对应设置方法进行设置。

取值范围：字符串

说明

- 使用命令 locale -a 查看当前系统支持的区域和相应的编码格式，并可以选择进行设置。
- 默认情况下，vb_initdb 会根据当前的系统环境初始化此参数，通过 locale 命令可以查看当前的配置环境。

默认值：C

default_text_search_config

参数说明：设置全文检索的配置信息。

如果设置为不存在的文本搜索配置时将会报错。如果 default_text_search_config 对应的文本搜索配置被删除，需要重新设置 default_text_search_config，否则会报设置错误。

- 其被文本搜索函数使用，这些函数并没有一个明确指定的配置。
- 当与环境相匹配的配置文件确定时，vb_initdb 会选择一个与环境相对应的设置来初始化配置文件。

该参数属于 USERSET 类型参数，请参考表 10-2 中对应设置方法进行设置。

取值范围：字符串

说明

Vastbase 支持 pg_catalog.english，pg_catalog.simple 两种配置。

默认值：pg_catalog.english

19.14.3.其他缺省

主要介绍数据库系统默认的库加载参数。

dynamic_library_path

参数说明：设置数据查找动态加载的共享库文件的路径。当需要打开一个可以动态装载的模块并且在 CREATE FUNCTION 或 LOAD 命令里面声明的名称没有目录部分时，系统将搜索这个目录以查找声明的文件。

用于 dynamic_library_path 的数值必须是一个冒号分隔的绝对路径列表。当一个路径名称以特殊变量$libdir 为开头时，会替换为 Vastbase 发布提供的模块安装路径。例如：

```
dynamic_library_path = '/usr/local/lib/postgresql:/opt/testgs/lib:$libdir'
```

该参数属于 SUSET 类型参数，请参考表 10-2 中对应设置方法进行设置。

取值范围：字符串

说明

设置为空字符串，表示关闭自动路径搜索。

默认值：$libdir

gin_fuzzy_search_limit

参数说明：设置 GIN 索引返回的集合大小的上限。

该参数属于 USERSET 类型参数，请参考表 10-2 中对应设置方法进行设置。

取值范围：整型，0~2147483647

默认值：0

local_preload_libraries

参数说明：指定一个或多个共享库，它们在开始连接前预先加载。多个加载库之间用逗号分隔，除了双引号，所有的库名都转换为小写。

- 并非只有系统管理员才能更改此选项，因此只能加载安装的标准库目录下 plugins 子目录中的库文件，数据库管理员有责任确保该目录中的库都是安全的。local_preload_libraries 中指定的项可以明确含有该目录，例如$libdir/plugins/mylib；也可以仅指定库的名称，例如 mylib（等价于$libdir/plugins/mylib）。
- 与 shared_preload_libraries 不同，在会话开始之前加载模块与在会话中使用到该模块的时候临时加载相比并不具有性能优势。相反，这个特性的目的是为了调试或者测量在特定会话中不明确使用 LOAD 加载的库。例如针对某个用户将该参数设为 ALTER USER SET 来进行调试。
- 当指定的库未找到时，连接会失败。
- 每一个支持 Vastbase 的库都有一个“magic block”用于确保兼容性，因此不支持 Vastbase 的库不能通过这个方法加载。

该参数属于 BACKEND 类型参数，请参考表 10-2 中对应设置方法进行设置。

取值范围：字符串

默认值：空

19.15. 锁管理

在 Vastbase 中，并发执行的事务由于竞争资源会导致死锁。本节介绍的参数主要管理事务锁的机制。

deadlock_timeout

参数说明：设置死锁超时检测时间，以毫秒为单位。当申请的锁超过设定值时，系统会检查是否产生了死锁。

- 死锁的检查代价是比较高的，服务器不会在每次等待锁的时候都运行这个过程。在系统运行过程中死锁是不经常出现的，因此在检查死锁前只需等待一个相对较短的时间。增加这个值就减少了无用的死锁检查浪费的时间，但是会减慢真正的死锁错误报告的速度。在一个负载过重的服务器上，用户可能需要增大它。这个值的设置应该超过事务持续时间，这样就可以减少在锁释放之前就开始死锁检查的问题。

- ❖ 如果要通过设置 log_lock_waits 来将查询执行过程中的锁等待耗时信息写入日志，请确保 log_lock_waits 的设置值小于 deadlock_timeout 的设置值（或默认值）。

该参数属于 SUSET 类型参数，请参考表 10-2 中对应设置方法进行设置。

取值范围：整型，1~2147483647，单位为毫秒（ms）。

默认值：1s

lockwait_timeout

参数说明：控制单个锁的最长等待时间。当申请的锁等待时间超过设定值时，系统会报错。

该参数属于 SUSET 类型参数，请参考表 10-2 中对应设置方法进行设置。

取值范围：整型，0 ~ INT_MAX，单位为毫秒（ms）。

默认值：20min

update_lockwait_timeout

参数说明：允许并发更新参数开启情况下，该参数控制并发更新同一行时单个锁的最长等待时间。当申请的锁等待时间超过设定值时，系统会报错。

该参数属于 SUSET 类型参数，请参考表 10-2 中对应设置方法进行设置。

取值范围：整型，0 ~ INT_MAX，单位为毫秒（ms）。

默认值：2min

max_locks_per_transaction

参数说明：控制每个事务能够得到的平均的对象锁的数量。

- ❖ 共享的锁表的大小是以假设任意时刻最多只有 max_locks_per_transaction*(max_connections+max_prepared_transactions) 个独立的对象需要被锁住为基础进行计算的。不超过设定数量的多个对象可以在任一时刻同时被锁定。当在一个事务里面修改很多不同的表时，可能需要提高这个默认数值。只能在数据库启动的时候设置。
- ❖ 增大这个参数可能导致 Vastbase 请求更多的 System V 共享内存，有可能超过操作系统的缺省配置。
- ❖ 当运行备机时，请将此参数设置不小于主机上的值，否则，在备机上查询操作不会被允许。

该参数属于 POSTMASTER 类型参数，请参考表 10-2 中对应设置方法进行设置。

取值范围：整型，10 ~ INT_MAX

默认值：256

max_pred_locks_per_transaction

参数说明：控制每个事务允许断定锁的最大数量，是一个平均值。

- ❖ 共享的断定锁表的大小是以假设任意时刻最多只有 max_pred_locks_per_transaction*(max_connections+max_prepared_transactions) 个独立

的对象需要被锁住为基础进行计算的。不超过设定数量的多个对象可以在任一时刻同时被锁定。当在一个事务里面修改很多不同的表时，可能需要提高这个默认数值。只能在服务器启动的时候设置。

- 增大这个参数可能导致 Vastbase 请求更多的 System V 共享内存，有可能超过操作系统的缺省配置。

该参数属于 POSTMASTER 类型参数，请参考表 10-2 中对应设置方法进行设置。

取值范围：整型，10 ~ INT_MAX

默认值：64

gs_clean_timeout

参数说明：控制 DBnode 周期性调用 gs_clean 工具的时间，是一个平均值。

- Vastbase 数据库中事务处理使用的是两阶段提交的方法，当有两阶段事务残留时，该事务通常会拿着表级锁，导致其它连接无法加锁，此时需要调用 gs_clean 工具对 Vastbase 中两阶段事务进行清理，gs_clean_timeout 是控制 DBnode 周期性调用 gs_clean 的时间。
- 增大这个参数可能导致 Vastbase 周期性调用 gs_clean 工具的时间延长，导致两阶段事务清理时间延长。

该参数属于 SIGHUP 类型参数，请参考表 10-2 中对应设置方法进行设置。

取值范围：整型，0 ~ INT_MAX / 1000，单位为秒（s）。

默认值：5min

partition_lock_upgrade_timeout

参数说明：在执行某些查询语句的过程中，会需要将分区表上的锁级别由允许读的 ExclusiveLock 级别升级到读写阻塞的 AccessExclusiveLock 级别。如果此时已经存在并发的读事务，那么该锁升级操作将阻塞等待。partition_lock_upgrade_timeout 为尝试锁升级的等待超时时间。

- 在分区表上进行 MERGE PARTITION 和 CLUSTER PARTITION 操作时，都利用了临时表进行数据重排和文件交换，为了最大程度提高分区上的操作并发度，在数据重排阶段给相关分区加锁 ExclusiveLock，在文件交换阶段加锁 AccessExclusiveLock。
- 常规加锁方式是等待加锁，直到加锁成功，或者等待时间超过 lockwait_timeout 发生超时失败。
- 在分区表上进行 MERGE PARTITION 或 CLUSTER PARTITION 操作时，进入文件交换阶段需要申请加锁 AccessExclusiveLock，加锁方式是尝试性加锁，加锁成功了则立即返回，不成功则等待 50ms 后继续下次尝试，加锁超时时间使用会话级设置参数 partition_lock_upgrade_timeout。
- 特殊值：若 partition_lock_upgrade_timeout 取值-1，表示无限等待，即不停的尝试锁升级，直到加锁成功。

该参数属于 USERSET 类型参数，请参考表 10-2 中对应设置方法进行设置。

取值范围：整型，最小值-1，最大值 3000，单位为秒（s）。

默认值：1800

fault_mon_timeout

参数说明：轻量级死锁检测周期。该参数属于 SIGHUP 类型参数，请参考表 10-2 中对应设置方法进行设置。

取值范围：整型，最小值 0，最大值 1440，单位为分钟（min）

默认值：5min

enable_online_ddl_waitlock

参数说明：控制 DDL 是否会阻塞等待 pg_advisory_lock/pgxc_lock_for_backup 等 Vastbase 锁。主要用于 OM 在线操作场景，不建议用户设置。

该参数属于 SIGHUP 类型参数，参考表 10-2 中对应设置方法进行设置。

取值范围：布尔型

- on 表示开启。
- off 表示关闭。

默认值：off

xloginsert_locks

参数说明：控制用于并发写预写式日志锁的个数。主要用于提高写预写式日志的效率。

该参数属于 POSTMASTER 类型参数，参考表 10-2 中对应设置方法进行设置。

取值范围：整型，最小值 1，最大值 1000

默认值：8

19.16. 版本和平台兼容性

19.16.1.历史版本兼容性

Vastbase 介绍数据库的向下兼容性和对外兼容性特性的参数控制。数据库系统的向后兼容性能够为对旧版本的数据库应用提供支持。本节介绍的参数主要控制数据库的向后兼容性。

array_nulls

参数说明：控制数组输入解析器是否将未用引用的 NULL 识别为数组的一个 NULL 元素。

该参数属于 USERSET 类型参数，请参考表 10-2 中对应设置方法进行设置。

取值范围：布尔型

- on 表示允许向数组中输入空元素。
- off 表示向下兼容旧式模式。仍然能够创建包含 NULL 值的数组。

默认值：on

backslash_quote

参数说明：控制字符串文本中的单引号是否能够用\'表示。

该参数属于 USERSET 类型参数，请参考表 10-2 中对应设置方法进行设置。

须知

在字符串文本符合 SQL 标准的情况下，\没有任何其他含义。这个参数影响的是如何处理不符合标准的字符串文本，包括明确的字符串转义语法是（E'...'）。

取值范围：枚举类型

- on 表示一直允许使用\'表示。
- off 表示拒绝使用\'表示。
- safe_encoding 表示仅在客户端字符集编码不会在多字节字符末尾包含\的 ASCII 值时允许。

默认值：safe_encoding

escape_string_warning

参数说明：警告在普通字符串中直接使用反斜杠转义。

- 如果需要使用反斜杠作为转义，可以调整为使用转义字符串语法(E'...')来做转义，因为在每个 SQL 标准中，普通字符串的默认行为现在将反斜杠作为一个普通字符。
- 这个变量可以帮助定位需要改变的代码。

该参数属于 USERSET 类型参数，请参考表 10-2 中对应设置方法进行设置。

取值范围：布尔型

默认值：on

lo_compat_privileges

参数说明：控制是否启动对大对象权限检查的向后兼容模式。

该参数属于 SUSET 类型参数，请参考表 10-2 中对应设置方法进行设置。

取值范围：布尔型

on 表示当读取或修改大对象时禁用权限检查，与 PostgreSQL 9.0 以前的版本兼容。

默认值：off

quote_all_identifiers

参数说明：当数据库生成 SQL 时，此选项强制引用所有的标识符（包括非关键字）。这将影响到 EXPLAIN 的输出及函数的结果，例如 pg_get_viewdef。详细说明请参见 vb_dump 的 --quote-all-identifiers 选项。

该参数属于 USERSET 类型参数，请参考表 10-2 中对应设置方法进行设置。

取值范围：布尔型

- on 表示打开强制引用。
- off 表示关闭强制引用。

默认值：off

sql_inheritance

参数说明：控制继承语义。用来控制继承表的访问策略，off 表示各种命令不能访问子表，即默认使用 ONLY 关键字。这是为了兼容 7.1 之前版本而设置的。

该参数属于 USERSET 类型参数，请参考表 10-2 中对应设置方法进行设置。

取值范围：布尔型

- on 表示可以访问子表。
- off 表示不访问子表。

默认值：on

standard_conforming_strings

参数说明：控制普通字符串文本（'...'）中是否按照 SQL 标准把反斜扛当普通文本。

- 应用程序通过检查这个参数可以判断字符串文本的处理方式。
- 建议明确使用转义字符串语法（E'...'）来转义字符。

该参数属于 USERSET 类型参数，请参考表 10-2 中对应设置方法进行设置。

取值范围：布尔型

- on 表示打开控制功能。
- off 表示关闭控制功能。

默认值：on

synchronize_seqscans

参数说明：控制启动同步的顺序扫描。在大约相同的时间内并行扫描读取相同的数据块，共享 I/O 负载。

该参数属于 USERSET 类型参数，请参考表 10-2 中对应设置方法进行设置。

取值范围：布尔型

- on 表示扫描可能从表的中间开始，然后选择"环绕"方式来覆盖所有的行，为了与已经在进行中的扫描活动同步。这可能会造成没有用 ORDER BY 子句的查询得到行排序造成不可预测的后果。
- off 表示确保顺序扫描是从表头开始的。

默认值：on

enable_beta_features

参数说明：控制开启某些非正式发布的特性，仅用于 POC 验证。这些特性属于延伸特性，建议客户谨慎开启，在某些功能场景下可能存在问题。

该参数属于 USERSET 类型参数，请参考表 10-2 中对应设置方法进行设置。

取值范围：布尔型

- on 表示开启这些功能受限的特性，保持前向兼容。但某些场景可能存在功能上的问题。
- off 表示禁止使用这些特性。

默认值：off

19.16.2.平台和客户端兼容性

很多平台都使用数据库系统，数据库系统的对外兼容性给平台提供了很大的方便。

transform_null_equals

参数说明：控制表达式 expr = NULL（或 NULL = expr）当做 expr IS NULL 处理。如果 expr 得出 NULL 值则返回真，否则返回假。

- 正确的 SQL 标准兼容的 expr = NULL 总是返回 NULL（未知）。
- Microsoft Access 里的过滤表单生成的查询使用 expr = NULL 来测试空值。打开这个选项，可以使用该接口来访问数据库。

该参数属于 USERSET 类型参数，请参考表 10-2 中对应设置方法进行设置。

取值范围：布尔型

- on 表示控制表达式 expr = NULL（或 NULL = expr）当做 expr IS NULL 处理。
- off 表示不控制，即 expr = NULL 总是返回 NULL（未知）。

默认值：off

说明

新用户经常在涉及 NULL 的表达式上语义混淆，故默认值设为 off。

support_extended_features

参数说明：控制是否支持数据库的扩展特性。

该参数属于 POSTMASTER 类型参数，请参考表 10-2 中对应设置方法进行设置。

取值范围：布尔型

- on 表示支持数据库的扩展特性。
- off 表示不支持数据库的扩展特性。

默认值：off

td_compatible_truncation

参数说明：控制是否开启与 Teradata 数据库相应兼容的特征。该参数在用户连接上与 TD 兼容的数据库时，可以将参数设置成为 on（即超长字符串自动截断功能启用），该功能启用后，在后续的 insert

语句中，对目标表中 char 和 varchar 类型的列插入超长字符串时，会按照目标表中相应列定义的最大长度对超长字符串进行自动截断。保证数据都能插入目标表中，而不是报错。

说明

超长字符串自动截断功能不适用于 insert 语句包含外表的场景。

如果向字符集为字节类型编码（SQL_ASCII，LATIN1 等）的数据库中插入多字节字符数据（如汉字等），且字符数据跨越截断位置，这种情况下，按照字节长度自动截断，自动截断后会在尾部产生非预期结果。如果用户有对于截断结果正确性的要求，建议用户采用 UTF8 等能够按照字符截断的输入字符集作为数据库的编码集。

该参数属于 USERSET 类型参数，请参考表 10-2 中对应设置方法进行设置。

取值范围：布尔型

- on 表示启动超长字符串自动截断功能。
- off 表示停止超长字符串自动截断功能。

默认值：off

19.17. 容错性

当数据库系统发生错误时，以下参数控制服务器处理错误的方式。

exit_on_error

参数说明：控制终止会话。

该参数属于 USERSET 类型参数，请参考表 10-2 中对应设置方法进行设置。

取值范围：布尔型

- on 表示任何错误都会终止当前的会话。
- off 表示只有 FATAL 级别的错误才会终止会话。

默认值：off

restart_after_crash

参数说明：设置为 on，后端进程崩溃时，Vastbase 将自动重新初始化此后端进程。

该参数属于 SIGHUP 类型参数，请参考表 10-2 中对应设置方法进行设置。

取值范围：布尔型

- on 表示能够最大限度地提高数据库的可用性。

  在某些情况（比如当采用管理工具（例如 xCAT）管理 Vastbase 时），能够最大限度地提高数据库的可用性。

- off 表示能够使得管理工具在后端进程崩溃时获取控制权并采取适当的措施进行处理。

默认值：on

omit_encoding_error

参数说明：设置为 on，数据库的客户端字符集编码为 UTF-8 时，出现的字符编码转换错误将打印在日志中，有转换错误的被转换字符会被忽略，以"?"代替。

该参数属于 USERSET 类型参数，请参考表 10-2 中对应设置方法进行设置。

取值范围：布尔型

- on 表示有转换错误的字符将被忽略，以"?"代替，打印错误信息到日志中。
- off 表示有转换错误的字符不能被转换，打印错误信息到终端。

默认值：off

max_query_retry_times

参数说明：指定 SQL 语句出错自动重试功能的最大重跑次数（目前支持重跑的错误类型为“Connection reset by peer”、“Lock wait timeout”和“Connection timed out”等），设定为 0 时关闭重跑功能。

该参数属于 USERSET 类型参数，请参考表 10-2 中对应设置方法进行设置。

取值范围：整型，0~20。

默认值：0

cn_send_buffer_size

参数说明：指定数据库主节点发送数据缓存区的大小。

该参数属于 POSTMASTER 类型参数，请参考表 10-2 中对应设置方法进行设置。

取值范围：整型，8~128， 单位为 KB。

默认值：8KB

max_cn_temp_file_size

参数说明：指定 SQL 语句出错自动重试功能中数据库主节点端使用临时文件的最大值，设定为 0 表示不使用临时文件。

该参数属于 SIGHUP 类型参数，请参考表 10-2 中对应设置方法进行设置。

取值范围：整型，0~10485760， 单位为 KB。

默认值：5GB

retry_ecode_list

参数说明：指定 SQL 语句出错自动重试功能支持的错误类型列表。

该参数属于 USERSET 类型参数，请参考表 10-2 中对应设置方法进行设置。

取值范围：字符串。

默认值：YY001 YY002 YY003 YY004 YY005 YY006 YY007 YY008 YY009 YY010 YY011 YY012 YY013 YY014 YY015 53200 08006 08000 57P01 XX003 XX009 YY016

data_sync_retry

参数说明：控制当 fsync 到磁盘失败后是否继续运行数据库。由于在某些操作系统的场景下，fsync 失败后重试阶段即使再次 fsync 失败也不会报错，从而导致数据丢失。

该参数属于 POSTMASTER 类型参数，请参考表 10-2 中对应设置方法进行设置。

取值范围：布尔型

- on 表示当 fsync 同步到磁盘失败后采取重试机制，数据库继续运行。
- off 表示当 fsync 同步到磁盘失败后直接报 panic，停止数据库。

默认值：off

19.18. 连接池参数

当使用连接池访问数据库时，在系统运行过程中，数据库连接是被当作对象存储在内存中的，当用户需要访问数据库时，并非建立一个新的连接，而是从连接池中取出一个已建立的空闲连接来使用。用户使用完毕后，数据库并非将连接关闭，而是将连接放回连接池中，以供下一个请求访问使用。

pooler_maximum_idle_time

参数说明：Pooler 链接自动清理功能使用，当链接池中链接空闲时间超过所设置值时，会触发自动清理机制，清理各节点的空闲链接数到 minimum_pool_size。

该参数属于 USERSET 类型参数，请参考表 10-2 中对应设置方法进行设置。

取值范围：整型，最小值为 0，最大值为 INT_MAX，最小单位为分钟

默认值：1h（即 60min）

minimum_pool_size

参数说明：Pooler 链接自动清理功能使用，自动清理后各 pooler 链接池对应节点的链接数最小剩余量，当参数设置为 0 时，可以关闭 pooler 链接自动清理功能。

该参数属于 USERSET 类型参数，请参考表 10-2 中对应设置方法进行设置。

取值范围：整型，最小值为 1，最大值为 65535

默认值：200

cache_connection

参数说明：是否回收连接池的连接。

该参数属于 SIGHUP 类型参数，请参考表 10-2 中对应设置方法进行设置。

取值范围：布尔型

- on 表示回收连接池的连接。
- off 表示不回收连接池的连接。

默认值：on

19.19. Vastbase 事务

介绍 Vastbase 事务隔离、事务只读、最大 prepared 事务数、维护模式目的参数设置及取值范围等内容。

transaction_isolation

参数说明：设置当前事务的隔离级别。

该参数属于 USERSET 类型参数，请参考表 10-1 中对应设置方法进行设置。

取值范围：字符串，只识别以下字符串，大小写空格敏感：

- serializable：Vastbase 中等价于 REPEATABLE READ。
- read committed：只能读取已提交的事务的数据（缺省），不能读取到未提交的数据。
- repeatable read：仅能读取事务开始之前提交的数据，不能读取未提交的数据以及在事务执行期间由其它并发事务提交的修改。
- default：设置为 defualt_transaction_isolation 所设隔离级别。

默认值：read committed

transaction_read_only

参数说明：设置当前事务是只读事务。

该参数属于 USERSET 类型参数，请参考表 10-1 中对应设置方法进行设置。

取值范围：布尔型

- on 表示设置当前事务为只读事务。
- off 表示该事务可以是非只读事务。

默认值：off

xc_maintenance_mode

参数说明：设置系统进入维护模式。

该参数属于 SUSET 类型参数，仅支持表 10-1 中的方式三进行设置。

取值范围：布尔型

- on 表示该功能启用。
- off 表示该功能被禁用。

须知

谨慎打开这个开关，避免引起 Vastbase 数据不一致。

默认值：off

allow_concurrent_tuple_update

参数说明：设置是否允许并发更新。

该参数属于 USERSET 类型参数，请参考表 10-1 中对应设置方法进行设置。

取值范围：布尔型

- on 表示该功能启用。
- off 表示该功能被禁用。

默认值：on

pgxc_node_name

参数说明：指定节点名称。

该参数属于 POSTMASTER 类型参数，请参考表 10-1 中对应设置方法进行设置。

取值范围：字符串。

默认值：当前节点名称。

transaction_deferrable

参数说明：指定是否允许一个只读串行事务延迟执行，使其不会执行失败。该参数设置为 on 时，当一个只读事务发现读取的元组正在被其他事务修改，则延迟该只读事务直到其他事务修改完成。目前，Vastbase 暂时未用到这个参数。与该参数类似的还有一个 default_transaction_deferrable，设置它来指定一个事务是否允许延迟。

该参数属于 USERSET 类型参数，请参考表 10-1 中对应设置方法进行设置。

取值范围：布尔型

- on 表示允许执行。
- off 表示不允许执行。

默认值：off

enforce_two_phase_commit

参数说明：强制使用两阶段提交，为了兼容历史版本功能保留该参数，当前版本设置无效。

该参数属于 SUSET 类型参数，请参考表 10-1 中对应设置方法进行设置。

取值范围：布尔型

- on 表示强制使用两阶段提交。
- off 表示不强制使用两阶段提交。

默认值：on

enable_show_any_tuples

参数说明：该参数只有在只读事务中可用，用于分析。当这个参数被置为 on/true 时，表中元组的所有版本都会可见。

该参数属于 USERSET 类型参数，请参考表 10-1 中对应设置方法进行设置。

取值范围：布尔型

- on/true 表示表中元组的所有版本都会可见。
- off/false 表示表中元组的所有版本都不可见。

默认值：off

replication_type

参数说明：标记当前 HA 模式是主备从模式还是一主多备模式。

该参数属于 POSTMASTER 类型参数，请参考表 10-1 中对应设置方法进行设置。

该参数是内部参数，用户不能自己去设置参数值。

取值范围：0~2

- 1 表示使用一主多备模式。
- 0 表示主备从模式。

默认值：1

19.20. 开发人员选项

allow_system_table_mods

参数说明：设置是否允许修改系统表的结构。

该参数属于 POSTMASTER 类型参数，请参考表 10-1 中对应设置方法进行设置。

取值范围：布尔型

- on 表示允许修改系统表的结构。
- off 表示不允许修改系统表的结构。

默认值：off

debug_assertions

参数说明：控制打开各种断言检查。能够协助调试，当遇到奇怪的问题或者崩溃，请把此参数打开，因为它能暴露编程的错误。要使用这个参数，必须在编译 Vastbase 的时候定义宏 USE_ASSERT_CHECKING（通过 configure 选项 --enable-cassert 完成）。

该参数属于 USERSET 类型参数，请参考表 10-1 中对应设置方法进行设置。

取值范围：布尔型

- on 表示打开断言检查。
- off 表示不打开断言检查。

说明

当启用断言选项编译 Vastbase 时，debug_assertions 缺省值为 on 。

默认值：off

ignore_checksum_failure

参数说明： 设置读取数据时是否忽略校验信息检查失败（但仍然会告警），继续执行可能导致崩溃，传播或隐藏损坏数据，无法从远程节点恢复数据及其他严重问题。不建议用户修改设置。

该参数属于 SUSET 类型参数，请参考表 10-1 中对应设置方法进行设置。

取值范围：布尔型

- on 表示忽略数据校验错误。
- off 表示数据校验错误正常报错。

默认值：off

enable_force_vector_engine

参数说明：对于支持向量化的执行器算子，如果其子节点是非向量化的算子，通过设置此参数为 on，强制生成向量化的执行计划。

该参数属于 USERSET 类型参数，请参考表 10-1 中对应设置方法进行设置。

取值范围：布尔型

默认值：off

explain_dna_file

参数说明：指定 explain_perf_mode 为 run，导出的 csv 信息的目标文件。

该参数属于 USERSET 类型参数，请参考表 10-1 中对应设置方法进行设置。

须知

这个参数的取值必须是绝对路径加上.csv 格式的文件名。

取值范围：字符串

默认值：空

explain_perf_mode

参数说明：此参数用来指定 explain 的显示格式。

该参数属于 USERSET 类型参数，请参考表 10-1 中对应设置方法进行设置。

取值范围：normal、pretty、summary、run

- normal：代表使用默认的打印格式。
- pretty：代表使用 Vastbase 改进后的新显示格式。新的格式层次清晰，计划包含了 plan node id，性能分析简单直接。
- summary：是在 pretty 的基础上增加了对打印信息的分析。

- run：在 summary 的基础上，将统计的信息输出到 csv 格式的文件中，以便于进一步分析。

默认值：pretty

ignore_system_indexes

参数说明：读取系统表时忽略系统索引（但是修改系统表时依然同时修改索引）。

该参数属于 BACKEND 类型参数，请参考表 10-1 中对应设置方法进行设置。

须知

这个参数在从系统索引被破坏的表中恢复数据的时候非常有用。

取值范围：布尔型

- on 表示忽略系统索引。
- off 表示不忽略系统索引。

默认值：off

post_auth_delay

参数说明：在认证成功后，延迟指定时间，启动服务器连接。允许调试器附加到启动进程上。

该参数属于 BACKEND 类型参数，请参考表 10-1 中对应设置方法进行设置。

取值范围：整型，最小值为 0，最大值为 2147，单位为秒。

默认值：0

pre_auth_delay

参数说明：启动服务器连接后，延迟指定时间，进行认证。允许调试器附加到认证过程上。

该参数属于 SIGHUP 类型参数，请参考表 10-1 中对应设置方法进行设置。

取值范围：整型，最小值为 0～60，单位为秒。

默认值：0

trace_notify

参数说明：为 LISTEN 和 NOTIFY 命令生成大量调试输出。client_min_messages 或 log_min_messages 级别必须是 DEBUG1 或者更低时，才能把这些输出分别发送到客户端或者服务器日志。

该参数属于 USERSET 类型参数，请参考表 10-1 中对应设置方法进行设置。

取值范围：布尔型

- on 表示打开输出功能。
- off 表示关闭输出功能。

默认值：off

trace_recovery_messages

参数说明：启用恢复相关调试输出的日志录，否则将不会被记录。该参数允许覆盖正常设置的log_min_messages，但是仅限于特定的消息，这是为了在调试备机中使用。

该参数属于 SIGHUP 类型参数，请参考表 10-1 中对应设置方法进行设置。

取值范围：枚举类型，有效值有 debug5、debug4、debug3、debug2、debug1、log，取值的详细信息请参见 log_min_messages。

默认值：log

说明

- 默认值 log 表示不影响记录决策。
- 除默认值外，其他值会导致优先级更高的恢复相关调试信息被记录，因为它们有 log 优先权。对于常见的 log_min_messages 设置，这会导致无条件地将它们记录到服务器日志上。

trace_sort

参数说明：控制是否在日志中打印排序操作中的资源使用相关信息。这个选项只有在编译 Vastbase 的时候定义了 TRACE_SORT 宏的时候才可用，不过目前 TRACE_SORT 是由缺省定义的。

该参数属于 USERSET 类型参数，请参考表 10-1 中对应设置方法进行设置。

取值范围：布尔型

- on 表示打开控制功能。
- off 表示关闭控制功能。

默认值：off

zero_damaged_pages

参数说明：控制检测导致 Vastbase 报告错误的损坏的页头，终止当前事务。

该参数属于 SUSET 类型参数，请参考表 10-1 中对应设置方法进行设置。

取值范围：布尔型

设置为 on 时，会导致系统报告一个警告，把损坏的页面填充为零然后继续处理。这种行为会破坏数据，也就是所有在已经损坏页面上的行记录。但是它允许绕开坏页面然后从表中尚存的未损坏页面上继续检索数据行。因此它在因为硬件或者软件错误导致的崩溃中进行恢复是很有用的。通常不应该把它设置为 on，除非不需要从崩溃的页面中恢复数据。

默认值：off

string_hash_compatible

参数说明：该参数用来说明 char 类型和 varchar/text 类型的 hash 值计算方式是否相同，以此来判断进行分布列从 char 类型到相同值的 varchar/text 类型转换，数据分布变化时，是否需要进行重分布。

该参数属于 POSTMASTER 类型参数，请参考表 10-1 中对应设置方法进行设置。

取值范围：布尔型

- on 表示计算方式相同，不需要进行重分布。
- off 表示计算方式不同，需要进行重分布。

说明

计算方式的不同主要体现在字符串计算 hash 值时传入的字节长度上。（如果为 char，则会忽略字符串后面空格的长度，如果为 text 或 varchar，则会保留字符串后面空格的长度。）hash 值的计算会影响到查询的计算结果，因此此参数一旦设置后，在整个数据库使用过程中不能再对其进行修改，以避免查询错误。

默认值：off

cost_param

参数说明：该参数用于控制在特定的客户场景中，使用不同的估算方法使得估算值与真实值更接近。此参数可以同时控制多种方法，与某一方法对应的位做与操作，不为 0 表示该方法被选择。

当 cost_param & 1 不为 0，表示对于求不等值连接选择率时选择一种改良机制，此方法在自连接（两个相同的表之间连接）的估算中更加准确，V300R002C00 版本开始，已弃用 cost_param & 1 不为 0 时的路径，默认选择更优的估算公式；

当 cost_param & 2 不为 0，表示求多个过滤条件（Filter）的选择率时，选择最小的作为总的选择率，而非两者乘积，此方法在过滤条件的列之间关联性较强时估算更加准确；

该参数属于 USERSET 类型参数，请参考表 10-1 中对应设置方法进行设置。

取值范围：整型，1～INT_MAX

默认值：0

convert_string_to_digit

参数说明：设置隐式转换优先级，是否优先将字符串转为数字。

该参数属于 USERSET 类型参数，请参考表 10-1 中对应设置方法进行设置。

取值范围：布尔型

- on 表示优先将字符串转为数字。
- off 表示不优先将字符串转为数字。

默认值：on

须知

请谨慎调整该参数，调整该参数会修改内部数据类型转换规则并可能导致不可预期的行为。

---

nls_timestamp_format

参数说明：设置时间戳默认格式。

该参数属于 USERSET 类型参数，请参考表 10-1 中对应设置方法进行设置。

取值范围：字符串

默认值：DD-Mon-YYYY HH:MI:SS.FF AM

remotetype

参数说明：设置远程连接类型。

该参数属于 BACKEND 类型参数，请参考表 10-1 中对应设置方法进行设置。

取值范围：枚举类型，有效值有 application，datanode, gtm，gtmproxy，internaltool，gtmtool。

默认值：application

enable_partitionwise

参数说明：分区表连接操作是否选择智能算法。

该参数属于 USERSET 类型参数，请参考表 10-1 中对应设置方法进行设置。

取值范围：布尔型

- on 表示选择智能算法。
- off 表示不选择智能算法。

默认值：off

max_function_args

参数说明：函数参数最大个数。

该参数属于 INTERNAL 类型参数，为固定参数，用户无法修改此参数，只能查看。

取值范围：整型

默认值：666

max_user_defined_exception

参数说明：异常最大个数，默认值不可更改。

该参数属于 USERSET 类型参数，请参考表 10-1 中对应设置方法进行设置。

取值范围：整型，当前只能取固定值 1000

默认值：1000

enable_debug_vacuum

参数说明：允许输出一些与 VACUUM 相关的日志，便于定位 VACUUM 相关问题。开发人员专用，不建议普通用户使用。

该参数属于 SIGHUP 类型参数，请参考表 10-1 中对应设置方法进行设置。

取值范围：布尔型

- on/true 表示开启此日志开关。
- off/false 表示关闭此日志开关。

默认值：off

enable_global_stats

参数说明：标识当前统计信息模式，区别采用全局统计信息收集模式还是单节点统计信息收集模式，默认创建为采用全局统计信息模式。当关闭该参数时，则默认收集 Vastbase 第一个节点的统计信息，此时可能会影响生成查询计划的质量，但信息收集性能较优，建议客户谨慎考虑。

该参数属于 SUSET 类型参数，请参考表 10-1 中对应设置方法进行设置。

取值范围：布尔型

- on/true 表示全局统计信息。
- off/false 表示数据库节点统计信息。

默认值：on

enable_fast_numeric

参数说明：标识是否开启 Numeric 类型数据运算优化。Numeric 数据运算是较为耗时的操作之一，通过将 Numeric 转化为 int64/int128 类型，提高 Numeric 运算的性能。

该参数属于 SUSET 类型参数，请参考表 10-1 中对应设置方法进行设置。

取值范围：布尔型

- on/true 表示开启 Numeric 优化。
- off/false 表示关闭 Numeric 优化。

默认值：on

rewrite_rule

参数说明：标识开启的可选查询重写规则。有部分查询重写规则是可选的，开启它们并不能总是对查询效率有提升效果。在特定的客户场景中，通过此 GUC 参数对查询重写规则进行设置，使得查询效率最优。

此参数可以控制查询重写规则的组合，比如有多个重写规则：rule1、rule2、rule3、rule4。可以设置：

```
    set rewrite_rule=rule1;                --启用查询重写规则rule1
set rewrite_rule=rule2,rule3;      --启用查询重写规则rule2和rule3
set rewrite_rule=none;                 --关闭所有可选查询重写规则
```

该参数属于 USERSET 类型参数，请参考表 10-1 中对应设置方法进行设置。

取值范围：字符串

- none：不使用任何可选查询重写规则
- lazyagg：使用 Lazy Agg 查询重写规则（消除子查询中的聚集运算）

默认值：magicset

enable_compress_spill

参数说明：标识是否开启下盘压缩功能。

该参数属于 USERSET 类型参数，请参考表 10-1 中对应设置方法进行设置。

取值范围：布尔型

- on/true 表示开启下盘优化。
- off/false 表示关闭下盘优化。

默认值：on

analysis_options

参数说明：通过开启对应选项中所对应的功能选项使用相应的定位功能，包括数据校验，性能统计等，参见取值范围中的选项说明。

该参数属于 USERSET 类型参数，请参考表 10-1 中对应设置方法进行设置。

取值范围：字符串

- LLVM_COMPILE 表示在 explain performance 显示界面中显示每个线程的 codegen 编译时间。
- HASH_CONFLICT 表示在数据库节点进程的 pg_log 目录中的 log 日志中显示 hash 表的统计信息，包括 hash 表大小，hash 链长，hash 冲突情况。
- STREAM_DATA_CHECK 表示对网络传输前后的数据进行 CRC 校验。

默认值：ALL,on(),off(LLVM_COMPILE,HASH_CONFLICT,STREAM_DATA_CHECK)，不开启任何定位功能。

须知

设置时，选择开启或者关闭的选项请使用'on()'或'off()'包括，未被显示指定的功能选项会维持原来的值。参考格式：

'on(option1, option2, ...)'

'off(ALL)'

---

resource_track_log

参数说明：控制自诊断的日志级别。目前仅对多列统计信息进行控制。

该参数属于 USERSET 类型参数，请参考表 10-1 中对应设置方法进行设置。

取值范围：字符串

- summary：显示简略的诊断信息。
- detail：显示详细的诊断信息。

目前这两个参数值只在显示多列统计信息未收集的告警的情况下有差别，summary 不显示未收集多列统计信息的告警，detail 会显示这类告警。

默认值：summary

udf_memory_limit

参数说明：控制每个数据库节点执行 UDF 时可用的最大物理内存量。

该参数属于 POSTMASTER 类型参数，请参考表 10-1 中对应设置方法进行设置。

取值范围：整型，200*1024 ~ max_process_memory，单位为 KB。

默认值：200MB

FencedUDFMemoryLimit

参数说明：控制每个 fenced udf worker 进程使用的虚拟内存。

该参数属于 USERSET 类型参数，请参考表 10-1 中对应设置方法进行设置。

设置建议：不建议设置此参数，可用 udf_memory_limit 代替。

取值范围：整数，0 ~ 2147483647，单位为 KB，设置可带单位（KB，MB，GB）。其中 0 表示不做内存控制。

默认值：0

UDFWorkerMemHardLimit

参数说明：控制 fencedUDFMemoryLimit 的最大值。

该参数属于 POSTMASTER 类型参数，请参考表 10-1 中对应设置方法进行设置。

设置建议：不建议设置此参数，可用 udf_memory_limit 代替。

取值范围：整数，0 ~ 2147483647，单位为 KB，设置时可带单位（KB，MB，GB）。

默认值：1GB

pljava_vmoptions

参数说明：用户自定义设置 PL/Java 函数所使用的 JVM 虚拟机的启动参数。

该参数属于 SUSET 类型参数，请参考表 10-1 中对应设置方法进行设置。

取值范围：字符串， 支持：

- JDK8 JVM 启动参数（可参见 JDK 官方说明）
- JDK8 JVM 系统属性参数（以–D 开头如–Djava.ext.dirs，可参见 JDK 官方说明）
- 用户自定义参数（以–D 开头，如–Duser.defined.option）

须知

如果用户在 pljava_vmoptions 中设置参数不满足上述取值范围，会在使用 PL/Java 语言函数时报错。此参数的详细说明参见 12.1PL/pgsql 语言函数。

默认值：空

enable_pbe_optimization

参数说明：设置优化器是否对以 PBE（Parse Bind Execute）形式执行的语句进行查询计划的优化。该参数属于 SUSET 类型参数，请参考表 10-1 中对应设置方法进行设置。

取值范围：布尔型。

- on 表示优化器将优化 PBE 语句的查询计划。
- off 表示不使用优化。

默认值：on

enable_light_proxy

参数说明：设置优化器是否对简单查询在数据库主节点上优化执行，应用端和内核端字符集不匹配时，该参数不生效，建议建库时将字符集设为 UTF8。

该参数属于 SUSET 类型参数，请参考表 10-1 中对应设置方法进行设置。

取值范围：布尔型。

- on 表示优化器将优化数据库主节点上简单查询的执行。
- off 表示不使用优化。

默认值：on

enable_global_plancache

参数说明：设置是否对 PBE 查询的执行计划进行缓存共享，开启该功能可以节省高并发下数据库节点的内存使用。

该参数属于 POSTMASTER 类型参数，请参考表 10-1 中对应设置方法进行设置。

取值范围：布尔型。

- on 表示对 PBE 查询的执行计划进行缓存共享。
- off 表示不共享。

默认值：off

checkpoint_flush_after

参数说明：设置 checkpointer 线程在连续写多少个磁盘页后会进行异步刷盘操作。Vastbase 中，磁盘页大小为 8KB。

该参数属于 SIGHUP 类型参数，请参考表 10-1 中对应设置方法进行设置。

取值范围：整型，0~256 （0 表示关闭异步刷盘功能）。例如，取值 32，表示 checkpointer 线程连续写 32 个磁盘页，即 32*8=256KB 磁盘空间后会进行异步刷盘。

默认值：32

bgwriter_flush_after

参数说明：设置 background writer 线程在连续写多少个磁盘页后会进行异步刷盘操作。Vastbase 中，磁盘页大小为 8KB。

该参数属于 SIGHUP 类型参数，请参考表 10-1 中对应设置方法进行设置。

取值范围：整型，0~256 （0 表示关闭异步刷盘功能），单位页面（8K）。例如，取值 64，表示 background writer 线程连续写 64 个磁盘页，即 64*8=512KB 磁盘空间后会进行异步刷盘。

默认值：256KB（即 32 个页面）

backend_flush_after

参数说明：设置 backend 线程在连续写多少个磁盘页后会产生异步刷盘操作。Vastbase 中，磁盘页大小为 8KB。

该参数属于 SIGHUP 类型参数，请参考表 10-1 中对应设置方法进行设置。

取值范围：整型，0~256 （0 表示关闭异步刷盘功能），单位页面（8K）。例如，取值 64，表示 backend 线程连续写 64 个磁盘页，即 64*8=512KB 磁盘空间后会进行异步刷盘。

默认值：0

enable_parallel_ddl

参数说明：控制多数据库节点对同一数据库对象是否能安全的并发执行 DDL 操作。

该参数属于 USERSET 类型参数，请参考表 10-1 中对应设置方法进行设置。

取值范围：布尔型

- on 表示可以安全的并发执行 DDL 操作，不会出现分布式死锁。
- off 表示不能安全的并发执行 DDL 操作，可能会出现分布式死锁。

默认值：on

show_acce_estimate_detail

参数说明：评估信息一般用于运维人员在维护工作中使用，因此该参数默认关闭，此外为了避免这些信息干扰正常的 explain 信息显示，只有在 explain 命令的 verbose 选项打开的情况下才显示评估信息

该参数属于 USERSET 类型参数，请参考表 10-1 中对应设置方法进行设置。

取值范围：布尔型

- on 表示可以在 explain 命令的输出中显示评估信息。
- off 表示不在 explain 命令的输出中显示评估信息。

默认值：off

enable_prevent_job_task_startup

参数说明：控制是否启动 job 线程。

该参数属于 SIGHUP 类型参数，请参考表 10-1 中对应设置方法进行设置。

取值范围：布尔型

- on 表示不能启动 job 线程。
- off 表示可以启动 job 线程。

默认值：off

enable_early_free

参数说明：控制是否可以实现算子内存的提前释放。

该参数属于 USERSET 类型参数，请参考表 10-1 中对应设置方法进行设置。

取值范围：布尔型

- on 表示支持算子内存提前释放。
- off 表示不支持算子内存提前释放。

默认值：on

support_batch_bind

参数说明：控制是否允许通过 JDBC、ODBC、Libpq 等接口批量绑定和执行 PBE 形式的语句。

该参数属于 SIGHUP 类型参数，请参考表 10-1 中对应设置方法进行设置。

取值范围：布尔型

- on 表示使用批量绑定和执行。
- off 表示不使用批量绑定和执行。

默认值：on

check_implicit_conversions

参数说明：控制是否对查询中有隐式类型转换的索引列是否会生成候选索引路径进行检查。

该参数属于 USERSET 类型参数，请参考表 10-1 中对应设置方法进行设置。

取值范围：布尔型

- on 表示对查询中有隐式类型转换的索引列是否会生成候选索引路径进行检查。
- off 表示不进行相关检查。

默认值：off

enable_thread_pool

参数说明：控制是否使用线程池功能。该参数属于 POSTMASTER 类型参数，请参考表 10-1 中对应设置方法进行设置。

取值范围：布尔型

- on 表示开启线程池功能。
- off 表示不开启线程池功能。

默认值：off

thread_pool_attr

参数说明：用于控制线程池功能的详细属性，该参数仅在 enable_thread_pool 打开后生效。该参数属于 POSTMASTER 类型参数，请参考表 10-1 中对应设置方法进行设置。

取值范围：字符串，长度大于 0

该参数分为 3 个部分，'thread_num, group_num, cpubind_info'，这 3 个部分的具体含义如下：

- thread_num：线程池中的线程总数，取值范围是 0~4096。其中 0 的含义是数据库根据系统 CPU core 的数量来自动配置线程池的线程数，如果参数值大于 0，线程池中的线程数等于 thread_num。
- group_num：线程池中的线程分组个数，取值范围是 0~64。其中 0 的含义是数据库根据系统 NUMA 组的个数来自动配置线程池的线程分组个数，如果参数值大于 0，线程池中的线程组个数等于 group_num。
- cpubind_info：线程池是否绑核的配置参数。可选择的配置方式有集中：1. '(nobind)' ，线程不做绑核；2. '(allbind)'，利用当前系统所有能查询到的 CPU core 做线程绑核；3. '(nodebind: 1, 2)'，利用 NUMA 组 1,2 中的 CPU core 进行绑核；4. '(cpubind: 0-30)'，利用 0-30 号 CPU core 进行绑核。该参数不区分大小写。

默认值：'16, 2, (nobind)'

numa_distribute_mode

参数说明：用于控制部分共享数据和线程在 NUMA 节点间分布的属性。用于大型多 NUMA 节点的 ARM 服务器性能调优，一般不用设置。

该参数属于 POSTMASTER 类型参数，请参考表 10-1 中对应设置方法进行设置。

取值范围：字符串，当前有效取值为'none', 'all'。

- none： 表示不启用本特性。
- all：表示将部分共享数据和线程分布到不同的 NUMA 节点下，减少远端访存次数，提高性能。目前仅适用于拥有多个 NUMA 节点的 ARM 服务器，并且要求全部 NUMA 节点都可用于数据库进程，不支持仅选择一部分 NUMA 节点。

默认值：'none'

log_pagewriter

参数说明：设置用于增量检查点打开后，显示线程的刷页信息以及增量检查点的详细信息，信息比较多，不建议设置为 ture。

该参数属于 SIGHUP 类型参数，请参考表 10-1 中对应设置方法进行设置。

取值范围：布尔型

默认值：off

enable_opfusion

参数说明：控制是否对简单查询进行查询优化。

该参数属于 USERSET 类型参数，请参考表 10-1 中对应设置方法进行设置。

取值范围：布尔型

- on 表示使用。
- off 表示不使用。

默认值：on

advance_xlog_file_num

参数说明：用于控制在后台周期性地提前初始化 xlog 文件的数目。该参数是为了避免事务提交时执行 xlog 文件初始化影响性能，但仅在超重负载时才可能出现，因此一般不用配置。

该参数属于 POSTMASTER 类型参数，请参考表 10-1 中对应设置方法进行设置。

取值范围：整型，0~100（0 表示不提前初始化）。例如，取值 10，表示后台线程会周期性地根据当前 xlog 写入位置提前初始化 10 个 xlog 文件。

默认值：0

enable_beta_opfusion

参数说明：在 enable_opfusion 参数打开的状态下，如果开启该参数，可以支持 TPCC 中出现的聚集函数，排序和 nestloop join 三类 SQL 语句的加速执行，提升 SQL 执行性能。需要注意的是 nestloop join 类型的支持，还需要打开下面的 enable_beta_nestloop_fusion 参数。

取值范围：布尔型

默认值：off

enable_beta_nestloop_fusion

参数说明：在 enable_opfusion 和 enable_beta_opfusion 两个参数都打开的状态下，如果开启该参数，可以支持 TPCC 中出现的 nestloop join 类 SQL 语句的加速执行，提升 SQL 执行性能。

取值范围：布尔型

默认值：off

19.21. 审计

19.21.1.审计开关

audit_enabled

参数说明：控制审计进程的开启和关闭。审计进程开启后，将从管道读取后台进程写入的审计信息，并写入审计文件。

该参数属于 SIGHUP 类型参数，请参考表 10-1 中对应设置方法进行设置。

取值范围：布尔型

- on 表示启动审计功能。
- off 表示关闭审计功能。

默认值：on

audit_directory

参数说明：审计文件的存储目录。一个相对于数据目录 data 的路径，可自行指定。

该参数属于 POSTMASTER 类型参数，请参考表 10-1 中对应设置方法进行设置。

取值范围：字符串

默认值：pg_audit。如果使用 om 工具部署 Vastbase，则审计日志路径为“$GAUSSLOG/pg_audit/实例名称”。

audit_data_format

参数说明：审计日志文件的格式。当前仅支持二进制格式。

该参数属于 POSTMASTER 类型参数，请参考表 10-1 中对应设置方法进行设置。

取值范围：字符串

默认值：binary

audit_rotation_interval

参数说明：指定创建一个新审计日志文件的时间间隔。当现在的时间减去上次创建一个审计日志的时间超过了此参数值时，服务器将生成一个新的审计日志文件。

该参数属于 SIGHUP 类型参数，请参考表 10-1 中对应设置方法进行设置。

取值范围：整型，1~INT_MAX/60，单位为 min。

默认值：1d

须知

请不要随意调整此参数，否侧可能会导致 audit_resource_policy 无法生效，如果需要控制审计日志的存储空间和时间，请使用 audit_resource_policy、audit_space_limit 和 audit_file_remain_time 参数进行控制。

audit_rotation_size

参数说明：指定审计日志文件的最大容量。当审计日志消息的总量超过此参数值时，服务器将生成一个新的审计日志文件。

该参数属于 SIGHUP 类型参数，请参考表 10-1 中对应设置方法进行设置。

取值范围：整型，1024~1048576，单位为 KB。

默认值：10MB

须知

请不要随意调整此参数，否侧可能会导致 audit_resource_policy 无法生效，如果需要控制审计日志的存储空间和时间，请使用 audit_resource_policy、audit_space_limit 和 audit_file_remain_time 参数进行控制。

audit_resource_policy

参数说明：控制审计日志的保存策略，以空间还是时间限制为优先策略。

该参数属于 SIGHUP 类型参数，请参考表 10-1 中对应设置方法进行设置。

取值范围：布尔型

- on 表示采用空间优先策略，最多存储 audit_space_limit 大小的日志。
- off 表示采用时间优先策略，最少存储 audit_file_remain_time 长度时间的日志。

默认值：on

audit_file_remain_time

参数说明：表示需记录审计日志的最短时间要求，该参数在 audit_resource_policy 为 off 时生效。

该参数属于 SIGHUP 类型参数，请参考表 10-1 中对应设置方法进行设置。

取值范围：整型，0~730，单位为 day，0 表示无时间限制。

默认值：90

audit_space_limit

参数说明：审计文件占用的磁盘空间总量。

该参数属于 SIGHUP 类型参数，请参考表 10-1 中对应设置方法进行设置。

取值范围：整型，1024KB~1024GB，单位为 KB。

默认值：1GB

audit_file_remain_threshold

参数说明：审计目录下审计文件个数的最大值。

该参数属于 SIGHUP 类型参数，请参考表 10-1 中对应设置方法进行设置。

取值范围：整型，1~1048576

默认值：1048576

须知

请尽量保证此参数为 1048576，并不要随意调整此参数，否则可能会导致 audit_resource_policy 无法生效，如果需要控制审计日志的存储空间和时间，请使用 audit_resource_policy、audit_space_limit 和 audit_file_remain_time 参数进行控制。

19.21.2.用户和权限审计

audit_login_logout

参数说明：这个参数决定是否审计 Vastbase 用户的登录（包括登录成功和登录失败）、注销。

该参数属于 SIGHUP 类型参数，请参考表 10-1 中对应设置方法进行设置。

取值范围：整型，0~7。

- 0 表示关闭用户登录、注销审计功能。
- 1 表示只审计用户登录成功。
- 2 表示只审计用户登录失败。
- 3 表示只审计用户登录成功和失败。
- 4 表示只审计用户注销。
- 5 表示只审计用户注销和登录成功。
- 6 表示只审计用户注销和登录失败。
- 7 表示审计用户登录成功、失败和注销。

默认值：7

audit_database_process

参数说明：该参数决定是否对 Vastbase 的启动、停止、切换和恢复进行审计。

该参数属于 SIGHUP 类型参数，请参考表 10-1 中对应设置方法进行设置。

取值范围：整型，0、1。

- 0 表示关闭 Vastbase 启动、停止、恢复和切换审计功能。
- 1 表示开启 Vastbase 启动、停止、恢复和切换审计功能。

默认值：1

audit_user_locked

参数说明：该参数决定是否审计 Vastbase 用户的锁定和解锁。

该参数属于 SIGHUP 类型参数，请参考表 10-1 中对应设置方法进行设置。

取值范围：整型，0、1。

- 0 表示关闭用户锁定和解锁审计功能。
- 1 表示开启审计用户锁定和解锁功能。

默认值：1

audit_user_violation

参数说明：该参数决定是否审计用户的越权访问操作。

该参数属于 SIGHUP 类型参数，请参考表 10-1 中对应设置方法进行设置。

取值范围：整型，0、1。

- 0 表示关闭用户越权操作审计功能。
- 1 表示开启用户越权操作审计功能。

默认值：0

audit_grant_revoke

参数说明：该参数决定是否审计 Vastbase 用户权限授予和回收的操作。

该参数属于 SIGHUP 类型参数，请参考表 10-1 中对应设置方法进行设置。

取值范围：整型，0、1。

- 0 表示关闭审计用户权限授予和回收功能。
- 1 表示开启审计用户权限授予和回收功能。

默认值：1

19.21.3.操作审计

audit_system_object

参数说明：该参数决定是否对 Vastbase 数据库对象的 CREATE、DROP、ALTER 操作进行审计。Vastbase 数据库对象包括 DATABASE、USER、schema、TABLE 等。通过修改该配置参数的值，可以只审计需要的数据库对象的操作。

该参数属于 SIGHUP 类型参数，请参考表 10-1 中对应设置方法进行设置。

取值范围：整型，0～524287

- 0 代表关闭 Vastbase 数据库对象的 CREATE、DROP、ALTER 操作审计功能。
- 非 0 代表只审计 Vastbase 的某类或者某些数据库对象的 CREATE、DROP、ALTER 操作。

取值说明：

该参数的值由 19 个二进制位的组合求出，这 19 个二进制位分别代表 Vastbase 的 19 类数据库对象。如果对应的二进制位取值为 0，表示不审计对应的数据库对象的 CREATE、DROP、ALTER 操作；取值为 1，表示审计对应的数据库对象的 CREATE、DROP、ALTER 操作。这 19 个二进制位代表的具体审计内容请参见表 19-4。

默认值：12295

表 19-4. audit_system_object 取值含义说明

二进制位	含义	取值说明
第 0 位	是否审计 DATABASE 对象的 CREATE、DROP、ALTER 操作。	• 0 表示不审计该对象的 CREATE、DROP、ALTER 操作； • 1 表示审计该对象的 CREATE、DROP、ALTER 操作。
第 1 位	是否审计 SCHEMA 对象的 CREATE、DROP、ALTER 操作。	• 0 表示不审计该对象的 CREATE、DROP、ALTER 操作； • 1 表示审计该对象的 CREATE、DROP、ALTER 操作。
第 2 位	是否审计 USER 对象的 CREATE、DROP、ALTER 操作。	• 0 表示不审计该对象的 CREATE、DROP、ALTER 操作； • 1 表示审计该对象的 CREATE、DROP、ALTER 操作。
第 3 位	是否审计 TABLE 对象的 CREATE、DROP、ALTER、TRUNCATE 操作。	• 0 表示不审计该对象的 CREATE、DROP、ALTER、TRUNCATE 操作； • 1 表示审计该对象的 CREATE、DROP、ALTER、TRUNCATE 操作。
第 4 位	是否审计 INDEX 对象的 CREATE、DROP、ALTER 操作。	• 0 表示不审计该对象的 CREATE、DROP、ALTER 操作； • 1 表示审计该对象的 CREATE、DROP、ALTER 操作。
第 5 位	是否审计 VIEW 对象的 CREATE、DROP 操作。	• 0 表示不审计该对象的 CREATE、DROP 操作； • 1 表示审计该对象的 CREATE、DROP 操作。
第 6 位	是否审计 TRIGGER 对象的 CREATE、DROP、ALTER 操作。	• 0 表示不审计该对象的 CREATE、DROP、ALTER 操作；

二进制位	含义	取值说明
		• 1 表示审计该对象的 CREATE、DROP、ALTER 操作。
第 7 位	是否审计 PROCEDURE/FUNCTION 对象的 CREATE、DROP、ALTER 操作。	• 0 表示不审计该对象的 CREATE、DROP、ALTER 操作； • 1 表示审计该对象的 CREATE、DROP、ALTER 操作。
第 8 位	是否审计 TABLESPACE 对象的 CREATE、DROP、ALTER 操作。	• 0 表示不审计该对象的 CREATE、DROP、ALTER 操作； • 1 表示审计该对象的 CREATE、DROP、ALTER 操作。
第 9 位	是否审计 RESOURCE POOL 对象的 CREATE、DROP、ALTER 操作。	• 0 表示不审计该对象的 CREATE、DROP、ALTER 操作； • 1 表示审计该对象的 CREATE、DROP、ALTER 操作
第 10 位	是否审计 WORKLOAD 对象的 CREATE、DROP、ALTER 操作。	• 0 表示不审计该对象的 CREATE、DROP、ALTER 操作； • 1 表示审计该对象的 CREATE、DROP、ALTER 操作
第 11 位	是否审计 SERVER FOR HADOOP 对象的 CREATE、DROP、ALTER 操作。	• 0 表示不审计该对象的 CREATE、DROP、ALTER 操作； • 1 表示审计该对象的 CREATE、DROP、ALTER 操作
第 12 位	是否审计 DATA SOURCE 对象的 CRAETE、DROP、ALTER 操作。	• 0 表示不审计该对象的 CREATE、DROP、ALTER 操作； • 1 表示审计该对象的 CREATE、DROP、ALTER 操作。
第 13 位	是否审计 NODE GROUP 对象的 CREATE、DROP 操作。	• 0 表示不审计该对象的 CREATE、DROP 操作； • 1 表示审计该对象的 CREATE、 DROP 操作。
第 14 位	是否审计 ROW LEVEL SECURITY 对象的 CREATE、	• 0 表示不审计该对象的 CREATE、DROP、ALTER 操作；

二进制位	含义	取值说明
	DROP、ALTER 操作。	• 1 表示审计该对象的 CREATE、DROP、ALTER 操作。
第 15 位	是否审计 TYPE 对象的 CREATE、DROP、ALTER 操作。	• 0 表示不审计 TYPE 对象的 CREATE、DROP、ALTER 操作； • 1 表示审计 TYPE 对象的 CREATE、DROP、ALTER 操作。
第 16 位	是否审计 TEXT SEARCH 对象（CONFIGURATION 和 DICTIONARY）的 CREATE、DROP、ALTER 操作。	• 0 表示不审计 TEXT SEARCH 对象的 CREATE、DROP、ALTER 操作； • 1 表示审计 TEXT SEARCH 对象的 CREATE、DROP、ALTER 操作。
第 17 位	是否审计 DIRECTORY 对象的 CREATE、DROP、ALTER 操作。	• 0 表示不审计 DIRECTORY 对象的 CREATE、DROP、ALTER 操作； • 1 表示审计 DIRECTORY 对象的 CREATE、DROP、ALTER 操作。
第 18 位	是否审计 SYNONYM 对象的 CREATE、DROP、ALTER 操作。	• 0 表示不审计 SYNONYM 对象的 CREATE、DROP、ALTER 操作； • 1 表示审计 SYNONYM 对象的 CREATE、DROP、ALTER 操作。

audit_dml_state

参数说明：这个参数决定是否对具体表的 INSERT、UPDATE、DELETE 操作进行审计。

该参数属于 SIGHUP 类型参数，请参考表 10-1 中对应设置方法进行设置。

取值范围：整型，0、1。

- ❖ 0 表示关闭具体表的 DML 操作（SELECT 除外）审计功能。
- ❖ 1 表示开启具体表的 DML 操作（SELECT 除外）审计功能。

默认值：0

audit_dml_state_select

参数说明：这个参数决定是否对 SELECT 操作进行审计。

该参数属于 SIGHUP 类型参数，请参考表 10-1 中对应设置方法进行设置。

取值范围：整型，0、1。

- 0 表示关闭 SELECT 操作审计功能。
- 1 表示开启 SELECT 审计操作功能。

默认值：0

audit_function_exec

参数说明：这个参数决定在执行存储过程、匿名块或自定义函数（不包括系统自带函数）时是否记录审计信息。

该参数属于 SIGHUP 类型参数，请参考表 10-1 中对应设置方法进行设置。

取值范围：整型，0、1。

- 0 表示关闭过程或函数执行的审计功能。
- 1 表示开启过程或函数执行的审计功能。

默认值：0

audit_copy_exec

参数说明：这个参数决定是否对 COPY 操作进行审计。

该参数属于 SIGHUP 类型参数，请参考表 10-1 中对应设置方法进行设置。

取值范围：整型，0、1。

- 0 表示关闭 COPY 审计功能。
- 1 表示开启 COPY 审计功能。

默认值：0

audit_set_parameter

参数说明：这个参数决定是否对 SET 操作进行审计。

该参数属于 SIGHUP 类型参数，请参考表 10-1 中对应设置方法进行设置。

取值范围：整型，0、1。

- 0 表示关闭 SET 审计功能。
- 1 表示开启 SET 审计功能。

默认值：1

sql_compatibility

参数说明：控制数据库的 SQL 语法和语句行为同哪一个主流数据库兼容。

该参数属于 INTERNAL 类型参数，为固定参数，用户无法修改此参数，只能查看。

取值范围：枚举型

- A 表示同 A 数据库兼容。
- B 表示同 B 数据库兼容。
- C 表示同 C 数据库兼容。

默认值：A

须知

在数据库中，该参数只能是确定的一个值，要么始终设置为 A，要么始终设置为 B，不能随便改动，否则会导致数据库行为不一致。

---

enableSeparationOfDuty

参数说明：是否开启三权分立选项。

该参数属于 POSTMASTER 类型参数，请参考表 10-1 中对应设置方法进行设置。

取值范围：布尔型

- on 表示开启三权分立。
- off 表示不开启三权分立。

默认值：off

enable_nonsysadmin_execute_direct

参数说明：是否允许非系统管理员执行 EXECUTE DIRECT ON 语句。

该参数属于 POSTMASTER 类型参数，请参考表 10-1 中对应设置方法进行设置。

取值范围：布尔型

- on 表示允许任意用户执行 EXECUTE DIRECT ON 语句。
- off 表示只允许系统管理员执行 EXECUTE DIRECT ON 语句。

默认值：off

enable_copy_server_files

参数说明：是否开启 copy 服务器端文件的权限。

该参数属于 SIGHUP 类型参数，请参考表 10-1 中对应设置方法进行设置。

取值范围：布尔型

- on 表示开启 copy 服务端文件的权限。
- off 表示不开启 copy 服务端文件的权限。

默认值：off

须知

copy from/to file 要求具有系统管理员权限的用户或初始用户才能使用，但是，在三权分立开启的状态下，系统管理员与初始用户的权限不同，可以通过使用 enable_copy_server_file 控制系统管理员的 copy 权限，当前默认不允许系统管理员权限用户进行对文件的 copy 操作，此参数打开后系统管理员方可执行该类型操作。

---

enable_access_server_directory

参数说明：是否开启系统管理员用户创建和删除 DIRECTORY 的权限。

该参数属于 SIGHUP 类型参数，请参考表 10-1 中对应设置方法进行设置。

取值范围：布尔型

- on 表示开启系统管理员用户创建和删除 DIRECTORY 的权限。
- off 表示不开启系统管理员用户创建和删除 DIRECTORY 的权限。

默认值：off

须知

- 出于安全考虑，默认情况下，只有初始用户才能够创建、删除 DIRECTORY 对象。
- 如果开启了 enable_access_server_directory，那么在三权分立关闭时，系统管理员（包括初始用户）可以创建、删除 DIRECTORY 对象；而在三权分立开启时，只有初始用户可以创建、删除 DIRECTORY 对象。

19.22. 事务监控

通过设置事务超时预警，可以监控自动回滚的事务并定位其中的语句问题，并且也可以监控执行时间过长的语句。

transaction_sync_naptime

参数说明：为保证数据一致性，当本地事务与 GTM 上 snapshot 中状态不一样时会阻塞其他事务的运行，需要等待本地节点上事务状态与 GTM 状态一致后再运行。该参数属于 USERSET 类型参数，请参考表 10-1 参数分类中对应设置方法进行设置。

取值范围：整型，0 ~ 2147483，单位为秒（s）。

默认值：30s

说明

若该值设为 0，则不会在阻塞达到时长时主动调用 gs_clean 进行清理，而是靠 gs_clean_timeout 间隔来调用 gs_clean，默认是 5 分钟。

transaction_sync_timeout

参数说明：为保证数据一致性，当本地事务与 GTM 上 snapshot 中状态不一样时会阻塞其他事务的运行，需要等待本地节点上事务状态与 GTM 状态一致后再运行。当数据库主节点上等待时长超过 transaction_sync_timeout 时会报错，回滚事务，避免由于 sync lock 等其他情况长时间进程停止响应造成对系统的阻塞。

该参数属于 USERSET 类型参数，请参考表 10-1 中对应设置方法进行设置。

取值范围：整型，0 ~ 2147483，单位为秒（s）。

默认值：10min

说明

- 若该值设为 0，则不会在阻塞超时时报错，回滚事务。
- 该值必须大于 gs_clean_timeout，避免数据库节点上由于还未被 gs_clean 清理的残留事务阻塞超时引起的不必要的事务回滚。

19.23. 升级参数

IsInplaceUpgrade

参数说明：标示是否在升级的过程中。该参数属于升级内部参数，用户无法修改。

该参数属于 SUSET 类型参数，请参考表 10-1 中对应设置方法进行设置。

取值范围：布尔型

- on 表示在升级过程中。
- off 表示不在升级过程中。

默认值：off

inplace_upgrade_next_system_object_oids

参数说明：标示就地升级过程中，新增系统对象的 OID。该参数属于升级内部参数，用户无法修改。

该参数属于 SUSET 类型参数，请参考表 10-1 中对应设置方法进行设置。

取值范围：字符串

默认值：空

upgrade_mode

参数说明：标示升级模式。该参数属于升级内部参数，不建议用户自己修改。

取值范围：整数，0~INT_MAX

- 0 表示不在升级过程中。
- 1 表示在就地升级过程中。
- 2 表示在灰度升级过程中。

默认值：0

19.24. 其它选项

server_version

参数说明：报告服务器版本号(字符串形式)。

该参数属于 INTERNAL 类型参数，为固定参数，用户无法修改此参数，只能查看。

取值范围：字符串

默认值：9.2.4

server_version_num

参数说明：报告服务器版本号(整数形式)。

该参数属于 INTERNAL 类型参数，为固定参数，用户无法修改此参数，只能查看。

取值范围：整数

默认值：90204

block_size

参数说明：报告当前数据库所使用的块大小。

该参数属于 INTERNAL 类型参数，为固定参数，用户无法修改此参数，只能查看。

取值范围：8192

默认值：8192

segment_size

参数说明：报告当前数据库所使用的段文件大小。

该参数属于 INTERNAL 类型参数，为固定参数，用户无法修改此参数，只能查看。

单位: 8KB

默认值：131072，即 1GB

max_index_keys

参数说明：报告当前数据库能够支持的索引键值的最大数目。

该参数属于 INTERNAL 类型参数，为固定参数，用户无法修改此参数，只能查看。

默认值：32

integer_datetimes

参数说明：报告是否支持 64 位整数形式的日期和时间格式。

该参数属于 INTERNAL 类型参数，为固定参数，用户无法修改此参数，只能查看。

取值范围：布尔型

- on 表示支持。
- off 表示不支持。

默认值：on

lc_collate

参数说明：报告当前数据库的字符串排序区域设置。

该参数属于 INTERNAL 类型参数，为固定参数，用户无法修改此参数，只能查看。

默认值：依赖于 Vastbase 安装部署时的配置

lc_ctype

参数说明：报告当前数据库的字母类别区域设置。如：哪些字符属于字母，它对应的大写形式是什么。

该参数属于 INTERNAL 类型参数，为固定参数，用户无法修改此参数，只能查看。

默认值：依赖于 Vastbase 安装部署时的配置

max_identifier_length

参数说明：报告当前系统允许的标识符最大长度。

该参数属于 INTERNAL 类型参数，为固定参数，用户无法修改此参数，只能查看。

取值范围：整型

默认值：63

server_encoding

参数说明：报告当前数据库的服务端编码字符集。

该参数属于 INTERNAL 类型参数，为固定参数，用户无法修改此参数，只能查看。

默认值：在创建数据库的时候决定的。

enable_upgrade_merge_lock_mode

参数说明：当该参数设置为 on 时，通过提升 deltamerge 内部实现的锁级别，避免和 update/delete 并发操作时的报错。

该参数属于 USERSET 类型参数，请参考表 10-1 中对应设置方法进行设置。

取值范围： 布尔型

- on，提升 deltamerge 内部实现的锁级别，并发执行 deltamerge 和 update/delete 操作时，一个操作先执行，另一个操作被阻塞，在前一个操作完成后，后一个操作再执行。
- off，在对表的 delta table 的同一行并发执行 deltamerge 和 update/delete 操作时，后一个对同一行数据更新的操作会报错退出。

默认值：off

job_queue_processes

参数说明：表示系统可以并发执行的 job 数目。该参数为 postmaster 级别，修改配置文件 postgres.conf 中“job_queue_processes”参数的值，该参数需要重启 vastbase 才能生效。

该参数属于 POSTMASTER 类型参数，请参考表 10-1 中对应设置方法进行设置。

取值范围：0～1000

功能：

- ❖ 当 job_queue_processes 设置为 0 值，表示不启用定时任务功能，任何 job 都不会被执行（因为开启定时任务的功能会对系统的性能有影响，有些局点可能不需要定时任务的功能，可以通过设置为 0 不启用定时任务功能）。
- ❖ 当 job_queue_processes 为大于 0 时，表示启用定时任务功能且系统能够并发处理的最大任务数。

启用定时任务功能后，job_scheduler 线程会在定时时间间隔轮询 pg_job 系统表，系统设置定时任务检查周期默认为 1s。

由于并行运行的任务数太多会消耗更多的系统资源，因此需要设置系统并发处理的任务数，当前并发的任务数达到 job_queue_processes 时，且此时又有任务到期，那么这些任务本次得不到执行而延期到下一轮询周期。因此，建议用户需要根据每个任务的执行时长合理的设置任务的时间间隔（即 submit 接口中的 interval 参数），来避免由于任务执行时间太长而导致下个轮询周期无法正常执行。

注：如果同一时间内并行的 job 数很多，过小的参数值会导致 job 等待。而过大的参数值则消耗更多的系统资源，建议设置此参数为 100，用户可以根据系统资源情况合理调整。

默认值：10

ngram_gram_size

参数说明：ngram 解析器分词的长度。

该参数属于 USERSET 类型参数，请参考表 10-1 中对应设置方法进行设置。

取值范围：整型 ，1～4

默认值：2

ngram_grapsymbol_ignore

参数说明：ngram 解析器是否忽略图形化字符。

该参数属于 USERSET 类型参数，请参考表 10-1 中对应设置方法进行设置。

取值范围：布尔型

- ❖ on 表示忽略图形化字符。
- ❖ off 表示不忽略图形化字符。

默认值：off

ngram_punctuation_ignore

参数说明：ngram 解析器是否忽略标点符号。

该参数属于 USERSET 类型参数，请参考表 10-1 中对应设置方法进行设置。

取值范围：布尔型

- ❖ on 表示忽略标点符号。
- ❖ off 表示不忽略标点符号。

默认值：on

transparent_encrypted_string

参数说明：它存储的是透明加密的一个样本串，使用数据库加密密钥加密固定串“TRANS_ENCRYPT_SAMPLE_STRING”后的密文，用来校验二次启动时获取的 DEK 是否正确。如果校验失败，那么数据库节点将拒绝启动。该参数属于 POSTMASTER 类型参数，请参考表 10-1 中对应设置方法进行设置。

取值范围：字符串，设置为空表示 Vastbase 非加密。

默认值：空

说明

请勿手动设置该参数，设置不当将导致 Vastbase 不可用。

transparent_encrypt_kms_url

参数说明：它存储的是透明加密的数据库密钥获取地址，内容要求不可出现 RFC3986 标准外的字符，最大长度 2047 字节。格式为“kms://协议@KMS 主机名 1;KMS 主机名 2:KMS 端口号/kms”，例如 kms://https@linux175:29800/。

该参数属于 POSTMASTER 类型参数，请参考表 10-1 中对应设置方法进行设置。

取值范围：字符串

默认值：空

transparent_encrypt_kms_region

参数说明：它存储的是 Vastbase 的部署区域，内容要求不可出现 RFC3986 标准外的字符，最大长度 2047 字节。

该参数属于 POSTMASTER 类型参数，请参考表 10-1 中对应设置方法进行设置。

取值范围：字符串

默认值：空

behavior_compat_options

参数说明：数据库兼容性行为配置项，该参数的值由若干个配置项用逗号隔开构成。

该参数属于 USERSET 类型参数，请参考表 10-1 中对应设置方法进行设置。

取值范围：字符串

默认值：空

说明

- 当前只支持表 19-5。
- 配置多个兼容性配置项时，相邻配置项用逗号隔开，例如：set behavior_compat_options='end_month_calculate,display_leading_zero';

表 19-5. 兼容性配置项

兼容性配置项	兼容性行为控制
display_leading_zero	浮点数显示配置项。 • 不设置此配置项时，对于-1~0 和 0~1 之间的小数，不显示小数点前的 0。比如，0.25 显示为.25。 • 设置此配置项时，对于-1~0 和 0~1 之间的小数，显示小数点前的 0。比如，0.25 显示为 0.25。

end_month_calculate	add_months 函数计算逻辑配置项。
	假定函数 add_months 的两个参数分别为 param1 和 param2，param1 的月份和 param2 的和为月份 result。
	• 不设置此配置项时，如果 param1 的日期（Day 字段）为月末，并且 param1 的日期（Day 字段）比 result 月份的月末日期小，计算结果中的日期字段（Day 字段）和 param1 的日期字段保持一致。比如，
	`vastbase=# select add_months('2018-02-28',3) from dual;`
	`add_months`
	`---------------------`
	`2018-05-28 00:00:00`
	`(1 row)`
	• 设置此配置项时，如果 param1 的日期(Day 字段)为月末，并且 param1 的日期（Day 字段）比 result 月份的月末日期比小，计算结果中的日期字段（Day 字段）和 result 的月末日期保持一致。比如，
	`vastbase=# select add_months('2018-02-28',3) from dual;`
	`add_months`
	`---------------------`
	`2018-05-31 00:00:00`
	`(1 row)`

compat_analyze_sample	analyze 采样行为配置项。 设置此配置项时，会优化 analyze 的采样行为，主要体现在 analyze 时全局采样会更精确的控制在 3 万条左右，更好的控制 analyze 时 DBnode 端的内存消耗，保证 analyze 性能的稳定性。
bind_schema_tablespace	绑定模式与同名表空间配置项。 如果存在与模式名 sche_name 相同的表空间名，那么如果设置 search_path 为 sche_name， default_tablespace 也会同步切换到 sche_name。
bind_procedure_searchpath	未指定模式名的数据库对象的搜索路径配置项。

兼容性配置项	兼容性行为控制
	在存储过程中如果不显示指定模式名，会优先在存储过程所属的模式下搜索。 如果找不到，则有两种情况： • 若不设置此参数，报错退出。 • 若设置此参数，按照 search_path 中指定的顺序继续搜索。如果还是找不到，报错退出。
correct_to_number	控制 to_number()结果兼容性的配置项。 若设置此配置项，则 to_number()函数结果与 pg11 保持一致，否则默认与 a db 保持一致。
unbind_dive_bound	控制对整数除法的结果进行范围校验。 若设置此配置项，则不需要对除法结果做范围校验，例如，INT_MIN/(-1)可以得到输出结果为 INT_MAX+1，反之，则会因为超过结果大于 INT_MAX 而报越界错误。
merge_update_multi	控制 merge into 匹配多行时是否进行 update 操作。 若设置此配置项，匹配多行时 update 不报错，否则默认与 a db 保持一致，报错。

return_null_string	控制函数 lpad()和 rpad()结果为空字符串''的显示配置项。
	• 不设置此配置项时，空字符串显示为 NULL。
	`vastbase=# select length(lpad('123',0,'*')) from dual;`
	`length`
	`--------`
	
	`(1 row)`
	• 设置此配置项时，空字符串显示为''。
	`vastbase=# select length(lpad('123',0,'*')) from dual;`
	`length`
	`--------`
	`0`
	`(1 row)`

compat_concat_variadic	控制函数 concat()和 concat_ws()对 variadic 类型结果兼容性的配置项。 若设置此配置项，当 concat 函数参数为 variadic 类型时，保留 a db 和 Teradata 兼容模式下不同的结果形式；否则默认 a db 和 Teradata 兼容模式下结果相同，且与 a db 保持一致。由于 mysql 无 variadic 类型，所以该

兼容性配置项	兼容性行为控制
	选项对 MySQL 无影响。
merge_update_multi	控制在使用 MERGE INTO ... WHEN MATCHED THEN UPDATE（参考 11.16.92MERGE INTO） 和 INSERT ... ON DUPLICATE KEY UPDATE（参考 11.16.89INSERT）时，当目标表中一条目标数据与多条源数据冲突时 UPDATE 行为。 若设置此配置项，当存在上述场景时，该冲突行将会多次执行 UPDATE；否则（默认）报错，即 MERGE 或 INSERT 操作失败。

table_skewness_warning_threshold

参数说明：设置用于表倾斜告警的阈值。

该参数属于 USERSET 类型参数，请参考表 10-1 中对应设置方法进行设置。

取值范围：浮点型，0～1

默认值：1

table_skewness_warning_rows

参数说明：设置用于表倾斜告警的行数。

该参数属于 USERSET 类型参数，请参考表 10-1 中对应设置方法进行设置。

取值范围：整型，0～INT_MAX

默认值：100000

datanode_heartbeat_interval

参数说明：设置心跳线程间心跳消息发送时间间隔，建议值不超过 wal_receiver_timeout / 2。

该参数属于 SIGHUP 类型参数，请参考表 10-1 中对应设置方法进行设置。

取值范围：整型，1000～60000（毫秒）

默认值：1s

pagewriter_thread_num

参数说明：设置用于增量检查点打开后后台刷页的线程数。

该参数属于 POSTMASTER 类型参数，请参考表 10-1 中对应设置方法进行设置。

取值范围：整型，1～8

默认值：2

pagewriter_threshold

参数说明：设置用于增量检查点打开后脏页数量达到这个数值时，后台刷页线程将一直刷脏页，不sleep。

该参数属于 SIGHUP 类型参数，请参考表 10-1 中对应设置方法进行设置。

取值范围：整型，1～INT_MAX

默认值：818

pagewriter_sleep

参数说明：设置用于增量检查点打开后，脏页数量不足 pagewriter_threshold 时，后台刷页线程将sleep 设置的时间继续刷页。

该参数属于 SIGHUP 类型参数，请参考表 10-1 中对应设置方法进行设置。

取值范围：整型，0～3600000（毫秒）

默认值：100ms

remote_read_mode

参数说明：远程读功能开关。读取主机上的页面失败时可以从备机上读取对应的页面。

取值范围：枚举类型

- off 表示关闭远程读功能
- non_authentication 表示开启远程读功能，但不进行证书认证
- authentication 表示开启远程读功能，但要进行证书认证

默认值：authentication

19.25. 等待事件

enable_instr_track_wait

参数说明：是否开启等待事件信息实时收集功能。

该参数属于 SIGHUP 类型参数，请参考表 10-1 中对应设置方法进行设置。

取值范围：布尔型

- on：表示打开等待事件信息收集功能。
- off：表示关闭等待事件信息收集功能。

默认值：on

19.26. Query

instr_unique_sql_count

参数说明：控制系统中 unique sql 信息实时收集功能。配置为 0 表示不启用 unique sql 信息收集功能。

该值由大变小将会清空系统中原有的数据重新统计；从小变大不受影响。

当系统中产生的 unique sql 信息大于 instr_unique_sql_count 时，系统产生的 unique sql 信息不被统计。

该参数属于 SIGHUP 类型参数，请参考表 10-1 中对应设置方法进行设置。

取值范围：整形 0~INT_MAX

默认值：100

instr_unique_sql_track_type

参数说明：unique sql 记录 SQL 方式。

该参数属于 INTERNAL 类型参数，请参考表 10-1 中对应设置方法进行设置。

取值范围：枚举类型

- top：只记录顶层 SQL。

默认值：top

enable_instr_rt_percentile

参数说明：是否开启计算系统中 80%和 95%的 SQL 响应时间的功能

该参数属于 SIGHUP 类型参数，请参考表 10-1 中对应设置方法进行设置。

取值范围：布尔型

- on：表示打开 sql 响应时间信息计算功能。
- off：表示关闭 sql 响应时间信息计算功能。

默认值：on

percentile

参数说明：sql 响应时间百分比信息，后台计算线程根据设置的值计算相应的百分比信息。

该参数属于 INTERNAL 类型参数，请参考表 10-1 中对应设置方法进行设置。

取值范围：字符串。

默认值："80，95"

instr_rt_percentile_interval

参数说明：sql 响应时间信息计算间隔，sql 响应时间信息计算功能打开后，后台计算线程每隔设置的时间进行一次计算。

该参数属于 SIGHUP 类型参数，请参考表 10-1 中对应设置方法进行设置。

取值范围：整型，0～3600（秒）。

默认值：10s

enable_instr_cpu_timer

参数说明：是否捕获 sql 执行的 cpu 时间消耗 。

该参数属于 SIGHUP 类型参数，请参考表 10-1 中对应设置方法进行设置。

取值范围：布尔型

- on：表示捕获 sql 执行的 cpu 时间消耗。
- off：表示不捕获 sql 执行的 cpu 时间消耗。

默认值：on

19.27. 系统性能快照

enable_wdr_snapshot

参数说明：是否开启数据库监控快照功能。

该参数属于 SIGHUP 类型参数，请参考表 10-1 中对应设置方法进行设置。

取值范围：布尔型

- on：打开数据库监控快照功能。
- off：关闭数据库监控快照功能。

默认值：off

wdr_snapshot_retention_days

参数说明：系统中数据库监控快照数据的保留天数，超过设置的值之后，系统每隔 wdr_snapshot_interval 时间间隔，清理 snapshot_id 最小的快照数据。

该参数属于 SIGHUP 类型参数，请参考表 10-1 中对应设置方法进行设置。

取值范围：整型，1～8。

默认值：8

wdr_snapshot_query_timeout

参数说明：系统执行数据库监控快照操作时，设置快照操作相关的 sql 语句的执行超时时间。如果语句超过设置的时间没有执行完并返回结果，则本次快照操作失败。

该参数属于 SIGHUP 类型参数，请参考表 10-1 中对应设置方法进行设置。

取值范围：整型，100～INT_MAX（秒）。

默认值：100s

wdr_snapshot_interval

参数说明：后台线程 Snapshot 自动对数据库监控数据执行快照操作的时间间隔。

该参数属于 SIGHUP 类型参数，请参考表 10-1 中对应设置方法进行设置。

取值范围：整型，10～60（分钟）。

默认值：1h

20. 常见故障定位指南

20.1. core 问题定位

20.1.1. 磁盘满故障引起的 core 问题

问题现象

TPCC 运行时，注入磁盘满故障，数据库进程 vastbase core 掉，如下图所示。

```
(gdb) bt
#0  0x00007fa32b359277 in raise () from /lib64/libc.so.6
#1  0x00007fa32b35aaa8 in abort () from /lib64/libc.so.6
#2  0x00005647da602df7 in errfinish(int, ...) ()
#3  0x00005647daddceb7 in XLogWrite(XLogwrtRqst const&, bool) ()
#4  0x00005647daddd2ec in XLogFlush(unsigned long, bool) ()
#5  0x00005647dadc3393 in dw_perform(unsigned int) ()
#6  0x00005647daa728eb in ckpt_pagewriter_main() ()
#7  0x00005647daa33988 in int GaussDbAuxiliaryThreadMain<(knl_thread_role)37>(knl_thread_arg*) ()
#8  0x00005647daa33ba6 in int GaussDbThreadMain<(knl_thread_role)37>(knl_thread_arg*) ()
#9  0x00005647daa17df1 in InternalThreadFunc(void*) ()
#10 0x00007fa32b6f7e25 in start_thread () from /lib64/libpthread.so.0
#11 0x00007fa32b421bad in clone () from /lib64/libc.so.6
(gdb) Quit
[root@xcw100 core]#
[root@xcw100 core]#
[root@xcw100 core]#
[root@xcw100 core]#
[root@xcw100 core]# ll
total 3024448
-rw-------. 1 single single 7938850816 May  7 11:31 core-gaussdb-1000-6-1588822317-xcw100.22075
```

原因分析

数据库本身机制，在磁盘满时，XLOG 日志无法进行写入，通过 panic 日志退出程序。

处理办法

外部监控磁盘使用状况，定时进行清理磁盘。

20.1.2. GUC 参数 log_directory 设置不正确引起的 core 问题

问题现象

数据库进程拉起后出现 coredump，日志无内容。

原因分析

GUC 参数 log_directory 设置的路径不可读取或无访问权限，数据库在启动过程中进行校验失败，通过 panic 日志退出程序。

处理办法

GUC 参数 log_directory 设置为合法路径，具体请参考 log_directory。

20.2. 备机处于 need repair(WAL)状态问题

问题现象

Vastbase 备机出现 Standby Need repair(WAL)故障。

原因分析

因网络故障、磁盘满等原因造成主备实例连接断开，主备日志不同步，导致数据库集群在启动时异常。

处理分析

通过 vb_ctl build -D 命令对故障节点进行重建，具体的操作方法请参见 Vastbase 工具参考中的 build 参数。

电话：010-82838118
地址：北京市海淀区学院路 30 号科大天工大厦 B 座 6 层
官网：www.vastdata.com.cn