GBase 8c

开发者手册

www.gbase.cn

GBase 8c 开发者手册，南大通用数据技术股份有限公司

GBase 版权所有©2022，保留所有权利

版权声明

本文档所涉及的软件著作权及其他知识产权已依法进行了相关注册、登记，由南大通用数据技术股份有限公司合法拥有，受《中华人民共和国著作权法》、《计算机软件保护条例》、《知识产权保护条例》和相关国际版权条约、法律、法规以及其它知识产权法律和条约的保护。未经授权许可，不得非法使用。

免责声明

本文档包含的南大通用数据技术股份有限公司的版权信息由南大通用数据技术股份有限公司合法拥有，受法律的保护，南大通用数据技术股份有限公司对本文档可能涉及到的非南大通用数据技术股份有限公司的信息不承担任何责任。在法律允许的范围内，您可以查阅，并仅能够在《中华人民共和国著作权法》规定的合法范围内复制和打印本文档。任何单位和个人未经南大通用数据技术股份有限公司书面授权许可，不得使用、修改、再发布本文档的任何部分和内容，否则将视为侵权，南大通用数据技术股份有限公司具有依法追究其责任的权利。

本文档中包含的信息如有更新，恕不另行通知。您对本文档的任何问题，可直接向南大通用数据技术股份有限公司告知或查询。

未经本公司明确授予的任何权利均予保留。

通讯方式

南大通用数据技术股份有限公司

天津市高新区华苑产业园区开华道22号普天创新园东塔20-23层(300384)

电话：400-013-9696　　邮箱：info@gbase.cn

商标声明

GBASE® 是南大通用数据技术股份有限公司向中华人民共和国国家商标局申请注册的注册商标，注册商标专用权由南大通用数据技术股份有限公司合法拥有，受法律保护。未经南大通用数据技术股份有限公司书面许可，任何单位及个人不得以任何方式或理由对该商标的任何部分进行使用、复制、修改、传播、抄录或与其它产品捆绑使用销售。凡侵犯南大通用数据技术股份有限公司商标权的，南大通用数据技术股份有限公司将依法追究其法律责任。

目　录

目　录 II
1 文档简介 1
1.1 概述 1
1.2 读者对象 1
1.3 文档阅读约定 1
2 概述 3
2.1 数据库逻辑结构图 3
2.2 数据查询请求处理过程 4
2.3 管理事务 4
2.3.1 事务控制 4
2.3.2 事务隔离级别 5
2.4 相关概念 6
2.4.1 数据库 6
2.4.2 表空间 6
2.4.3 模式 6
2.4.4 用户和角色 6
2.4.5 事务管理 6
2.4.5.1 2PC 6
2.4.5.2 MVCC 7
3 数据库使用 8
3.1 概述 8
3.2 连接数据库 11
3.2.1 确认连接信息 11
3.2.2 配置服务端远程连接 11

3.2.3 使用 gsql 连接 13
3.2.3.1 本地连接数据库 14
3.2.3.2 远程连接数据库 15
3.2.4 应用程序接口 16
3.3 字符集 16
3.4 创建和管理数据库 17
3.5 规划存储模型 19
3.5.1 行存表 20
3.5.2 列存表 20
3.5.3 行存表和列存表的选择 21
3.6 创建和管理表空间 21
3.7 创建和管理表 24
3.7.1 创建表 24
3.7.2 向表中插入数据 25
3.7.3 更新表中数据 28
3.7.4 查看数据 29
3.7.5 删除表中数据 30
3.8 查看系统表 31
3.8.1 查看数据库中包含的表 31
3.8.2 查看数据库用户 31
3.8.3 查看和停止正在运行的查询语句 32
3.9 其他操作 33
3.9.1 创建和管理 schema 33
3.9.2 创建和管理分区表 37
3.9.3 创建和管理索引 40
3.9.4 创建和管理视图 44

3.9.5 创建和管理序列 45
3.9.6 管理插件 47
3.9.6.1 dblink 47
3.9.6.2 pg_trgm 47
3.9.6.3 oracle_fdw 47
3.9.6.4 mysql_fdw 47
3.9.6.5 postgres_fdw 47
3.9.6.6 zhparser 48
3.9.6.7 sqldecoding 48
3.9.6.8 uuid-ossp 48
3.9.6.9 pgrouting 48
3.9.6.10 pgcrypto 48
3.9.6.11 logical to kafka 48
4 开发设计建议 48
4.1 开发设计建议概述 48
4.2 数据库对象命名 49
4.3 数据库对象设计 50
4.3.1 Database 和 Schema 设计 50
4.3.2 表设计 51
4.3.3 字段设计 53
4.3.4 约束设计 54
4.3.5 视图和关联表设计 55
4.4 工具对接 56
4.4.1 JDBC 配置 56
4.5 SQL 编写 57
5 最佳实践 61

5.1 表设计最佳实践 61
5.1.1 选择存储模型 61
5.1.2 使用局部聚簇 62
5.1.3 使用分区表 62
5.1.4 选择数据类型 63
5.2 导入最佳实践 63
5.3 SQL 查询最佳实践 65
6 应用程序开发教程 67
6.1 开发规范 67
6.2 基于 JDBC 开发 67
6.2.1 JDBC 包、驱动类和环境类 68
6.2.2 开发流程 69
6.2.3 加载驱动 70
6.2.4 连接数据库 70
6.2.5 连接数据库（以 SSL 方式） 77
6.2.6 执行 SQL 语句 80
6.2.6.1 执行普通 SQL 80
6.2.6.2 执行预编译 SQL 语句 81
6.2.6.3 调用存储过程 81
6.2.6.4 Oracle 兼容模式启用重载时，调用存储过程 83
6.2.6.5 执行批处理 84
6.2.7 处理结果集 85
6.2.8 关闭连接 87
6.2.9 日志管理 87
6.2.10 JDBC 接口参考 92
6.3 基于 ODBC 开发 92

6.3.1 ODBC 包及依赖的库和头文件 93
6.3.2 Linux 下配置数据源 93
6.3.3 开发流程 102
6.3.4 示例：常用功能和批量绑定 104
6.3.5 典型应用场景配置 111
6.3.6 ODBC 接口参考 122
6.4 基于 libpq 开发 122
6.4.1 libpq 使用依赖的头文件 122
6.4.2 开发流程 122
6.4.3 常用功能示例代码 123
6.4.4 libpq 接口参数 127
6.4.5 链接参数 127
6.5 基于 Psycopg 开发 132
6.5.1 Psycopg 包 132
6.5.2 开发流程 133
6.5.3 加载驱动 133
6.5.4 连接数据库 133
6.5.5 执行 SQL 语句 134
6.5.6 处理结果集 134
6.5.7 关闭连接 134
6.5.8 连接数据库（SSL 方式） 134
6.5.9 示例：常用操作 135
6.5.10 Psycopg 接口参考 136
6.6 调试 136
7 管理数据库安全 140
7.1 客户端接入认证 140

7.1.1 配置客户端接入认证 140
7.1.2 配置文件参考 143
7.1.3 用 SSL 进行安全的 TCP/IP 连接 147
7.1.4 用 SSH 隧道进行安全的 TCP/IP 连接 153
7.1.5 查看数据库连接数 154
7.1.6 SSL 证书管理 157
7.1.6.1 证书生成 157
7.1.6.2 证书替换 162
7.2 管理用户及权限 163
7.2.1 默认权限机制 163
7.2.2 管理员 164
7.2.3 三权分立 166
7.2.4 用户 167
7.2.5 角色 169
7.2.6 Schema 171
7.2.7 用户权限设置 173
7.2.8 行级访问控制 174
7.2.9 设置安全策略 175
7.2.9.1 设置账户安全策略 175
7.2.9.2 设置账号有效期 178
7.2.9.3 设置密码安全策略 179
7.3 身份鉴别 188
7.3.1 LDAP 认证 188
7.3.2 口令接入 189
7.4 传输加密 189
7.5 数据插入加密 191

7.6 设置数据库审计 194
7.6.1 审计概述 194
7.6.2 查看审计结果 199
7.6.3 维护审计日志 201
7.6.4 设置文件权限安全策略 204
7.6.5 统一审计策略概述 205
7.7 设置密态等值查询 206
7.7.1 密态等值查询概述 206
7.7.2 使用 gsql 操作密态数据库 206
7.7.3 使用 JDBC 操作密态数据库 208
7.7.4 密态支持函数/存储过程 211
7.8 设置账本数据库 213
7.8.1 账本数据库概述 213
7.8.2 查看账本历史操作记录 215
7.8.3 校验账本数据一致性 218
7.8.4 归档账本数据库 219
7.8.5 修复账本数据库 220
7.9 强制访问控制 222
8 接口参考 224
8.1 JDBC 224
8.1.1 java.sql.Connection 224
8.1.2 java.sql.CallableStatement 227
8.1.3 java.sql.DatabaseMetaData 229
8.1.4 java.sql.Driver 239
8.1.5 java.sql.PreparedStatement 240
8.1.6 java.sql.ResultSet 243

8.1.7 java.sql.ResultSetMetData 250
8.1.8 java.sql.Statement 252
8.1.9 javax.sql.ConnectionPoolDataSource 254
8.1.10 javax.sql.DataSource 255
8.1.11 javax.sql.PooledConnection 255
8.1.12 javax.naming.Context 256
8.1.13 javax.naming.spi.InitialContextFactory 256
8.1.14 CopyManager 256
8.1.15 PGReplicationConnection 258
8.2 ODBC 258
8.2.1 SQLAllocEnv 258
8.2.2 SQLAllocConnect 259
8.2.3 SQLAllocHandle 259
8.2.4 SQLAllocStmt 260
8.2.5 SQLBindCol 260
8.2.6 SQLBindParameter 261
8.2.7 SQLColAttribute 263
8.2.8 SQLConnect 264
8.2.9 SQLDisconnect 265
8.2.10 SQLExecDirect 266
8.2.11 SQLExecute 267
8.2.12 SQLFetch 269
8.2.13 SQLFreeStmt 269
8.2.14 SQLFreeConnect 270
8.2.15 SQLFreeHandle 270
8.2.16 SQLFreeEnv 271

8.2.17 SQLPrepare 271
8.2.18 SQLGetData 272
8.2.19 SQLGetDiagRec 273
8.2.20 SQLSetConnectAttr 275
8.2.21 SQLSetEnvAttr 276
8.2.22 SQLSetStmtAttr 277
8.2.23 示例 279
8.3 libpq 285
8.3.1 数据库连接控制函数 285
8.3.1.1 PQconnectdbParams 286
8.3.1.2 PQconnectdb 286
8.3.1.3 PQconninfoParse 287
8.3.1.4 PQconnectStart 288
8.3.1.5 PQerrorMessage 288
8.3.1.6 PQsetdbLogin 289
8.3.1.7 PQfinish 290
8.3.1.8 PQreset 290
8.3.1.9 PQstatus 291
8.3.2 数据库执行语句函数 292
8.3.2.1 PQclear 292
8.3.2.2 PQexec 293
8.3.2.3 PQexecParams 294
8.3.2.4 PQexecParamsBatch 294
8.3.2.5 PQexecPrepared 295
8.3.2.6 PQexecPreparedBatch 296
8.3.2.7 PQfname 297

8.3.2.8 PQgetvalue 298
8.3.2.9 PQnfields 299
8.3.2.10 PQntuples 299
8.3.2.11 PQprepare 300
8.3.2.12 PQresultStatus 301
8.3.3 异步命令处理 302
8.3.3.1 PQsendQuery 303
8.3.3.2 PQsendQueryParams 304
8.3.3.3 PQsendPrepare 305
8.3.3.4 PQsendQueryPrepared 306
8.3.3.5 PQflush 307
8.3.3.6 PQgetCancel 307
8.3.3.7 PQfreeCancel 308
8.3.3.8 PQcancel 309
8.4 Psycopg 309
8.4.1 psycopg2.connect() 310
8.4.2 connection.cursor() 310
8.4.3 cursor.execute(query,vars_list) 311
8.4.4 curosr.executemany(query,vars_list) 312
8.4.5 connection.commit() 312
8.4.6 connection.rollback() 313
8.4.7 cursor.fetchone() 313
8.4.8 cursor.fetchall() 313
8.4.9 cursor.close() 314
8.4.10 connection.close() 314
9 导入数据 314

9.1 通过 INSERT 语句直接写入数据 315
9.2 使用 COPY FROM STDIN 导入数据 315
9.2.1 关于 COPY FROM STDIN 导入数据 315
9.2.2 CopyManager 类简介 316
9.2.3 CopyManager 的继承关系 316
9.2.4 处理错误表 317
9.2.5 示例 1：通过本地文件导入导出数据 319
9.2.6 示例 2：从 MY 迁移数据 321
9.3 使用 gsql 元命令导入数据 323
9.4 用 gs_restore 命令导入数据 327
9.5 更新表中数据 333
9.5.1 使用 DML 命令更新表 333
9.5.2 使用合并方式更新和插入数据 334
9.6 深层复制 337
9.6.1 使用 CREATE TABLE 执行深层复制 337
9.6.2 使用 CREATE TABLE LIKE 执行深层复制 337
9.6.3 通过创建临时表并截断原始表来执行深层复制 338
9.7 分析表 339
9.8 对表执行 VACUUM 340
9.9 管理并发写入操作 340
9.9.1 写入和读写操作 341
9.9.2 并发写入事务的潜在死锁情况 341
9.9.3 并发写入示例 342
9.9.3.1 相同表的 INSERT 和 DELETE 并发 342
9.9.3.2 相同表的并发 INSERT 343
9.9.3.3 相同表的并发 UPDATE 343

9.9.3.4 数据导入和查询的并发 344
10 导出数据 345
10.1 使用 gs_dump 和 gs_dumpall 命令导出数据 345
10.1.1 概述 345
10.1.2 导出单个数据库 347
10.1.2.1 导出数据库 347
10.1.2.2 导出模式 350
10.1.2.3 导出表 353
10.1.3 导出所有数据库 357
10.1.3.1 导出所有数据库 357
10.1.3.2 导出全局对象 358
10.1.4 无权限角色导出数据 360
11 AI 特性 363
11.1 AI4DB: 数据库自治运维 364
11.1.1 DBMind 模式说明 365
11.1.1.1 service 子命令 366
11.1.1.2 component 子命令 370
11.1.1.3 set 子命令 371
11.1.2 DBMind 的支持组件 372
11.1.2.1 Prometheus Exporter 组件 372
11.1.2.1.1 概述 372
11.1.2.1.2 环境部署 372
11.1.2.1.3 使用指导 373
11.1.2.1.4 获取帮助 375
11.1.2.1.5 命令参考 375
11.1.2.1.6 常见问题处理 379

11.1.3 DBMind 的 AI 子功能 379
11.1.3.1 X-Tuner：参数调优与诊断 379
11.1.3.1.1 概述 379
11.1.3.1.2 使用准备 379
11.1.3.1.3 使用示例 385
11.1.3.1.4 获取帮助 389
11.1.3.1.5 命令参考 390
11.1.3.1.6 常见问题处理 392
11.1.3.2 Index-advisor：索引推荐 393
11.1.3.2.1 单 query 索引推荐 393
11.1.3.2.2 虚拟索引 394
11.1.3.2.3 workload 级别索引推荐 397
11.1.3.3 AI4DB：慢 SQL 根因分析 400
11.1.3.3.1 概述 400
11.1.3.3.2 环境部署 400
11.1.3.3.3 使用指导 400
11.1.3.3.4 获取帮助 400
11.1.3.3.5 命令参考 401
11.1.3.3.6 常见问题处理 401
11.1.3.4 AI4DB：趋势预测 402
11.1.3.4.1 概述 402
11.1.3.4.2 环境部署 402
11.1.3.4.3 使用指导 402
11.1.3.4.4 获取帮助 402
11.1.3.4.5 命令参考 403
11.1.3.4.6 常见问题处理 403

11.1.3.5 SQLdiag：慢 SQL 发现 404
11.1.3.5.1 概述 404
11.1.3.5.2 使用指导 404
11.1.3.5.3 获取帮助 406
11.1.3.5.4 命令参考 407
11.1.3.5.5 常见问题处理 407
11.2 DB4AI：数据库驱动 AI 407
11.2.1 原生 DB4AI 引擎 408
11.2.2 全流程 AI 419
11.2.2.1 PLPython Fenced 模式 419
11.2.2.2 DB4AI-Snapshots 数据版本管理 421
11.3 AI in DB：数据库内 AI 功能 427
11.3.1 概述 427
11.3.2 环境部署 427
11.3.3 使用指导 431
11.3.4 最佳实践 435
11.3.5 常见问题处理 436
12 内存表特性 436
12.1 MOT 介绍 436
12.1.1 MOT 简介 436
12.1.2 MOT 特性及价值 438
12.1.3 MOT 关键技术 438
12.1.4 MOT 应用场景 440
12.2 使用 MOT 441
12.2.1 MOT 使用概述 441
12.2.2 MOT 准备 442

12.2.3 MOT 部署 448
12.2.3.1 MOT 服务器优化 448
12.2.3.2 MOT 配置 451
12.2.4 MOT 使用 461
12.2.4.1 授予用户权限 461
12.2.4.2 创建/删除 MOT 462
12.2.4.3 为 MOT 创建索引 462
12.2.4.4 将磁盘表转换为 MOT 463
12.2.4.5 查询原生编译 465
12.2.4.6 重试中止事务 467
12.2.4.7 MOT 外部支持工具 468
12.2.4.8 MOT SQL 覆盖和限制 469
12.2.5 MOT 管理 473
12.2.5.1 MOT 持久性 473
12.2.5.2 MOT 恢复 478
12.2.5.3 MOT 复制和高可用 478
12.2.5.4 MOT 内存管理 479
12.2.5.5 MOT VACUUM 清理 479
12.2.5.6 MOT 统计 480
12.2.5.7 MOT 监控 480
12.2.5.8 MOT 错误消息 482
12.3 MOT 的概念 487
12.3.1 MOT 纵向扩容架构 488
12.3.2 MOT 并发控制机制 490
12.3.3 MOT 本地内存和全局内存 491
12.3.4 MOT SILO 增强特性 491

12.3.5 MOT 隔离级别 492
12.3.6 MOT 乐观并发控制 493
12.3.6.1 乐观 OCC 与悲观 2PL 493
12.3.6.2 OCC 与 2PL 的区别举例 495
12.3.7 扩展 FDW 与其他特性 496
12.3.8 NUMA-aware 分配和亲和性 499
12.3.9 MOT 索引 500
12.3.10 MOT 持久性概念 501
12.3.10.1 MOT 日志记录：WAL 重做日志概念 502
12.3.10.2 MOT 检查点概念 509
12.3.11 MOT 恢复概念 509
12.3.12 MOT 查询原生编译（JIT） 510
12.3.13 对比：磁盘与 MOT 512
12.4 附录 513
13 性能调优 516
13.1 总体调优思路 516
13.2 确定性能调优范围 517
13.2.1 硬件瓶颈点分析 519
13.2.1.1 CPU 519
13.2.1.2 内存 523
13.2.1.3 I/O 524
13.2.1.4 网络 526
13.2.2 查询最耗性能的 SQL 527
13.2.3 分析作业是否被阻塞 529
13.3 系统调优指南 530
13.3.1 操作系统参数调优 531

13.3.2 数据库系统参数调优 533
13.3.2.1 数据库内存参数调优 533
13.3.2.2 数据库并发队列参数调优 534
13.3.3 配置 SMP 535
13.3.3.1 SMP 适用场景与限制 535
13.3.3.2 资源对 SMP 性能的影响 537
13.3.3.3 其他因素对 SMP 性能的影响 537
13.3.3.4 SMP 使用建议 538
13.3.4 配置 LLVM 538
13.3.4.1 LLVM 适用场景与限制 539
13.3.4.2 其他因素对 LLVM 性能的影响 540
13.3.4.3 LLVM 使用建议 540
13.4 SQL 调优指南 541
13.4.1 Query 执行流程 541
13.4.2 SQL 执行计划介绍 544
13.4.2.1 SQL 执行计划概述 544
13.4.2.2 详解 545
13.4.2.2.1 执行计划 545
13.4.2.2.2 执行信息 548
13.4.3 调优流程 548
13.4.4 更新统计信息 549
13.4.5 审视和修改表定义 550
13.4.5.1 审视和修改表定义概述 550
13.4.5.2 选择存储模型 550
13.4.5.3 使用局部聚簇 550
13.4.5.4 使用分区表 551

13.4.5.5 选择数据类型 551
13.4.6 典型 SQL 调优点 552
13.4.6.1 SQL 自诊断 552
13.4.6.2 子查询调优 553
13.4.6.3 统计信息调优 564
13.4.6.4 算子级调优 566
13.4.7 经验总结：SQL 语句改写规则 568
13.4.8 SQL 调优关键参数调整 570
13.4.9 使用 Plan Hint 进行调优 571
13.4.9.1 Plan Hint 调优概述 571
13.4.9.2 Join 顺序的 Hint 577
13.4.9.3 Join 方式的 Hint 579
13.4.9.4 Join 行数的 Hint 580
13.4.9.5 Scan 方式的 Hint 581
13.4.9.6 子链接块名的 hint 582
13.4.9.7 Hint 的错误、冲突及告警 583
13.4.9.8 优化器 GUC 参数的 Hint 585
13.4.9.9 Custom Plan 和 Generic Plan 选择的 Hint 586
13.4.9.10 指定子查询不展开的 Hint 587
13.4.9.11 指定不使用全局计划缓存的 Hint 588
13.4.9.12 同层参数化路径的 Hint 588
13.5 使用向量化执行引擎进行调优 590
13.6 TPCC 性能调优测试指导 591
13.6.1 概述 591
13.6.2 搭建性能测试环境 592
13.6.2.1 硬件要求 592

13.6.2.2 软件要求 592
13.6.2.3 前置软件安装 592
13.6.2.4 BIOS 配置 593
13.6.2.5 操作系统配置 595
13.6.2.6 文件系统配置 595
13.6.2.7 网络配置 596
13.6.2.7.1 网卡多中断队列设置 596
13.6.2.7.2 中断调优 597
13.6.2.7.3 网卡固件确认与更新 597
13.6.2.8 数据库服务端及客户端绑核 598
13.6.3 测试 TPCC 性能 599
13.6.4 TPCC 测试用例 605
13.6.4.1 环境配置 605
13.6.4.1.1 硬件配置 605
13.6.4.1.2 软件配置 605
13.6.4.2 性能调优 606
13.6.4.2.1 系统参数调优 606
13.6.4.2.2 数据库参数调优 606
13.6.4.2.3 TPC-C 参数配置 608
13.6.4.2.4 建表语句 608
13.6.4.2.5 建索引语句 611
13.6.4.2.6 调优效果展示 612
13.7 实际调优案例 612
13.8 性能日志 616
14 配置运行参数 617
14.1 查看参数当前取值 617

14.2 重设参数 618
14.2.1 背景信息 618
14.2.2 GUC 参数设置 618
14.2.3 操作步骤 1
14.2.4 示例 1
15 用户自定义函数 1
15.1 PL/pgSQL 语言函数 1
16 存储过程 1
16.1 存储过程 1
16.2 数据类型 1
16.3 数据类型转换 1
16.4 数组和 record 1
16.4.1 数组 1
16.4.1.1 数组类型的使用 1
16.4.2 集合 1
16.4.2.1 集合类型的使用 1
16.4.3 record 1
16.4.3.1 record 类型的变量 1
16.4.3.2 语法 1
16.4.3.3 示例 1
16.5 声明语法 1
16.5.1 基本结构 1
16.5.1.1 结构 1
16.5.1.2 分类 1
16.5.2 匿名块 1
16.5.2.1 语法 1

16.5.3 子程序 1
16.6 基本语句 1
16.6.1 定义变量 1
16.6.1.1 变量声明 1
16.6.1.2 变量作用域 1
16.6.2 赋值语句 1
16.6.2.1 语法 1
16.6.2.2 嵌套赋值 1
16.6.2.3 INTO/BULK COLLECT INTO 1
16.6.3 调用语句 1
16.6.3.1 语法 1
16.6.3.2 示例 1
16.7 动态语句 1
16.7.1 执行动态查询语句 1
16.7.1.1 EXECUTE IMMEDIATE 1
16.7.1.2 OPEN FOR 1
16.7.2 执行动态非查询语句 1
16.7.2.1 语法 1
16.7.2.2 示例 1
16.7.3 动态调用存储过程 1
16.7.3.1 语法 1
16.7.4 动态调用匿名块 1
16.7.4.1 语法 1
16.8 控制语句 1
16.8.1 返回语句 1
16.8.1.1 RETURN 1

16.8.1.2 RETURN NEXT 及 RETURN QUERY 1
16.8.2 条件语句 1
16.8.3 循环语句 1
16.8.3.1 简单 LOOP 语句 1
16.8.3.2 WHILE_LOOP 语句 1
16.8.3.3 FOR_LOOP（integer 变量）语句 1
16.8.3.4 FOR_LOOP 查询语句 1
16.8.3.5 FORALL 批量查询语句 1
16.8.4 分支语句 1
16.8.5 空语句 1
16.8.6 错误捕获语句 1
16.8.7 GOTO 语句 1
16.9 事务管理 1
16.9.1 语法格式 1
16.9.2 使用场景 1
16.9.3 使用限制 1
16.9.4 示例 1
16.10 其他语句 1
16.10.1 锁操作 1
16.10.2 游标操作 1
16.11 游标 1
16.11.1 游标概述 1
16.11.2 显式游标 1
16.11.2.1 处理步骤 1
16.11.2.2 属性 1
16.11.3 隐式游标 1

16.11.3.1 简介 1
16.11.3.2 属性 1
16.11.3.3 示例 1
16.11.4 游标循环 1
16.11.4.1 语法 1
16.11.4.2 注意事项 1
16.12 高级包 1
16.12.1 基础接口 1
16.12.1.1 PKG_SERVICE 1
16.12.1.2 PKG_UTIL 1
16.13 Retry 管理 1
16.14 调试 1
16.14.1 语法 1
16.14.1.1 RAISE 语法 1
16.14.1.2 EXCEPTION_INIT 语法 1
16.14.2 示例 1
16.15 package 1
17 自治事务 1
17.1 存储过程支持自治事务 1
17.2 匿名块支持自治事务 1
17.3 用户自定义函数支持自治事务 1
17.4 规格约束 1
18 系统表和系统视图 1
18.1 系统表和系统视图概述 1
18.2 系统表 1
18.2.1 GS_ASP 1

18.2.2 GS_AUDITING_POLICY 1
18.2.3 GS_AUDITING_POLICY_ACCESS 1
18.2.4 GS_AUDITING_POLICY_FILTERS 1
18.2.5 GS_AUDITING_POLICY_PRIVILEGES 1
18.2.6 GS_CLIENT_GLOBAL_KEYS 1
18.2.7 GS_CLIENT_GLOBAL_KEYS_ARGS 1
18.2.8 GS_COLUMN_KEYS 1
18.2.9 GS_COLUMN_KEYS_ARGS 1
18.2.10 GS_DB_PRIVILEGE 1
18.2.11 GS_ENCRYPTED_COLUMNS 1
18.2.12 GS_ENCRYPTED_PROC 1
18.2.13 GS_GLOBAL_CHAIN 1
18.2.14 GS_GLOBAL_CONFIG 1
18.2.15 GS_MASKING_POLICY 1
18.2.16 GS_MASKING_POLICY_ACTIONS 1
18.2.17 GS_MASKING_POLICY_FILTERS 1
18.2.18 GS_MATVIEW 1
18.2.19 GS_MATVIEW_DEPENDENCY 1
18.2.20 GS_MODEL_WAREHOUSE 1
18.2.21 GS_OPT_MODEL 1
18.2.22 GS_PACKAGE 1
18.2.23 GS_POLICY_LABEL 1
18.2.24 GS_RECYCLEBIN 1
18.2.25 GS_TXN_SNAPSHOT 1
18.2.26 GS_UID 1
18.2.27 GS_WLM_EC_OPERATOR_INFO 1

18.2.28 GS_WLM_INSTANCE_HISTORY 1
18.2.29 GS_WLM_OPERATOR_INFO 1
18.2.30 GS_WLM_PLAN_ENCODING_TABLE 1
18.2.31 GS_WLM_PLAN_OPERATOR_INFO 1
18.2.32 GS_WLM_SESSION_QUERY_INFO_ALL 1
18.2.33 GS_WLM_USER_RESOURCE_HISTORY 1
18.2.34 PG_AGGREGATE 1
18.2.35 PG_AM 1
18.2.36 PG_AMOP 1
18.2.37 PG_AMPROC 1
18.2.38 PG_APP_WORKLOADGROUP_MAPPING 1
18.2.39 PG_ATTRDEF 1
18.2.40 PG_ATTRIBUTE 1
18.2.41 PG_AUTHID 1
18.2.42 PG_AUTH_HISTORY 1
18.2.43 PG_AUTH_MEMBERS 1
18.2.44 PG_CAST 1
18.2.45 PG_CLASS 1
18.2.46 PG_COLLATION 1
18.2.47 PG_CONSTRAINT 1
18.2.48 PG_CONVERSION 1
18.2.49 PG_DATABASE 1
18.2.50 PG_DB_ROLE_SETTING 1
18.2.51 PG_DEFAULT_ACL 1
18.2.52 PG_DEPEND 1
18.2.53 PG_DESCRIPTION 1

18.2.54 PG_DIRECTORY 1
18.2.55 PG_ENUM 1
18.2.56 PG_EXTENSION 1
18.2.57 PG_EXTENSION_DATA_SOURCE 1
18.2.58 PG_FOREIGN_DATA_WRAPPER 1
18.2.59 PG_FOREIGN_SERVER 1
18.2.60 PG_FOREIGN_TABLE 1
18.2.61 PG_HASHBUCKET 1
18.2.62 PG_INDEX 1
18.2.63 PG_INHERITS 1
18.2.64 PG_JOB 1
18.2.65 PG_JOB_PROC 1
18.2.66 PG_LANGUAGE 1
18.2.67 PG_LARGEOBJECT 1
18.2.68 PG_LARGEOBJECT_METADATA 1
18.2.69 PG_NAMESPACE 1
18.2.70 PG_OBJECT 1
18.2.71 PG_OPCLASS 1
18.2.72 PG_OPERATOR 1
18.2.73 PG_OPFAMILY 1
18.2.74 PG_PARTITION 1
18.2.75 PG_PLTEMPLATE 1
18.2.76 PG_PROC 1
18.2.77 PG_PUBLICATION 1
18.2.78 PG_PUBLICATION_REL 1
18.2.79 PG_RANGE 1

18.2.80 PG_REPLICATION_ORIGIN 1
18.2.81 PG_RESOURCE_POOL 1
18.2.82 PG_REWRITE 1
18.2.83 PG_RLSPOLICY 1
18.2.84 PG_SECLABEL 1
18.2.85 PG_SHDEPEND 1
18.2.86 PG_SHDESCRIPTION 1
18.2.87 PG_SHSECLABEL 1
18.2.88 PG_STATISTIC 1
18.2.89 PG_STATISTIC_EXT 1
18.2.90 PG_SUBSCRIPTION 1
18.2.91 PG_SYNONYM 1
18.2.92 PG_TABLESPACE 1
18.2.93 PG_TRIGGER 1
18.2.94 PG_TS_CONFIG 1
18.2.95 PG_TS_CONFIG_MAP 1
18.2.96 PG_TS_DICT 1
18.2.97 PG_TS_PARSER 1
18.2.98 PG_TS_TEMPLATE 1
18.2.99 PG_TYPE 1
18.2.100 PG_USER_MAPPING 1
18.2.101 PG_USER_STATUS 1
18.2.102 PG_WORKLOAD_GROUP 1
18.2.103 PGXC_CLASS 1
18.2.104 PGXC_GROUP 1
18.2.105 PGXC_NODE 1

18.2.106 PGXC_SLICE 1
18.2.107 PLAN_TABLE_DATA 1
18.2.108 STREAMING_CONT_QUERY 1
18.2.109 STATEMENT_HISTORY 1
18.2.110 STREAMING_REAPER_STATUS 1
18.2.111 STREAMING_STREAM 1
18.2.112 SYS_DUMMY 1
18.3 系统视图 1
18.3.1 DV_SESSIONS 1
18.3.2 DV_SESSION_LONGOPS 1
18.3.3 GS_AUDITING 1
18.3.4 GS_AUDITING_ACCESS 1
18.3.5 GS_AUDITING_PRIVILEGE 1
18.3.6 GS_DB_PRIVILEGES 1
18.3.7 GS_FILE_STAT 1
18.3.8 GS_INSTANCE_TIME 1
18.3.9 GS_LABELS 1
18.3.10 GS_MASKING 1
18.3.11 GS_MATVIEWS 1
18.3.12 GS_OS_RUN_INFO 1
18.3.13 GS_REDO_STAT 1
18.3.14 GS_SESSION_CPU_STATISTICS 1
18.3.15 GS_SESSION_MEMORY 1
18.3.16 GS_SESSION_MEMORY_CONTEXT 1
18.3.17 GS_SESSION_MEMORY_DETAIL 1
18.3.18 GS_SESSION_MEMORY_STATISTICS 1

18.3.19 GS_SESSION_STAT 1
18.3.20 GS_SESSION_TIME 1
18.3.21 GS_SQL_COUNT 1
18.3.22 GS_THREAD_MEMORY_CONTEXT 1
18.3.23 GS_TOTAL_MEMORY_DETAIL 1
18.3.24 GS_WLM_CGROUP_INFO 1
18.3.25 GS_WLM_EC_OPERATOR_STATISTICS 1
18.3.26 GS_WLM_EC_OPERATOR_HISTORY 1
18.3.27 GS_WLM_OPERATOR_HISTORY 1
18.3.28 GS_WLM_OPERATOR_STATISTICS 1
18.3.29 GS_WLM_PLAN_OPERATOR_HISTORY 1
18.3.30 GS_WLM_REBUILD_USER_RESOURCE_POOL 1
18.3.31 GS_WLM_RESOURCE_POOL 1
18.3.32 GS_WLM_SESSION_HISTORY 1
18.3.33 GS_WLM_SESSION_INFO 1
18.3.34 GS_WLM_SESSION_INFO_ALL 1
18.3.35 GS_WLM_SESSION_STATISTICS 1
18.3.36 GS_WLM_USER_INFO 1
18.3.37 GS_STAT_SESSION_CU 1
18.3.38 GS_WRITE_TERM_LOG 1
18.3.39 MPP_TABLES 1
18.3.40 PG_AVAILABLE_EXTENSION_VERSIONS 1
18.3.41 PG_AVAILABLE_EXTENSIONS 1
18.3.42 PG_CURSORS 1
18.3.43 PG_COMM_DELAY 1
18.3.44 PG_COMM_RECV_STREAM 1

18.3.45 PG_COMM_SEND_STREAM 1
18.3.46 PG_COMM_STATUS 1
18.3.47 PG_CONTROL_GROUP_CONFIG 1
18.3.48 PG_EXT_STATS 1
18.3.49 PG_GET_INVALID_BACKENDS 1
18.3.50 PG_GET_SENDERS_CATCHUP_TIME 1
18.3.51 PG_GROUP 1
18.3.52 PG_GTT_RELSTATS 1
18.3.53 PG_GTT_STATS 1
18.3.54 PG_GTT_ATTACHED_PIDS 1
18.3.55 PG_INDEXES 1
18.3.56 PG_LOCKS 1
18.3.57 PG_NODE_ENV 1
18.3.58 PG_OS_THREADS 1
18.3.59 PG_PREPARED_STATEMENTS 1
18.3.60 PG_PREPARED_XACTS 1
18.3.61 PG_PUBLICATION_TABLES 1
18.3.62 PG_REPLICATION_ORIGIN_STATUS 1
18.3.63 PG_REPLICATION_SLOTS 1
18.3.64 PG_RLSPOLICIES 1
18.3.65 PG_ROLES 1
18.3.66 PG_RULES 1
18.3.67 PG_RUNNING_XACTS 1
18.3.68 PG_SECLABELS 1
18.3.69 PG_TOTAL_MEMORY_DETAIL 1
18.3.70 PG_TOTAL_USER_RESOURCE_INFO_OID 1

18.3.71 PG_SETTINGS 1
18.3.72 PG_SHADOW 1
18.3.73 PG_STATS 1
18.3.74 PG_STAT_ACTIVITY 1
18.3.75 PG_STAT_ACTIVITY_NG 1
18.3.76 PG_STAT_ALL_INDEXES 1
18.3.77 PG_STAT_ALL_TABLES 1
18.3.78 PG_STAT_BAD_BLOCK 1
18.3.79 PG_STAT_BGWRITER 1
18.3.80 PG_STAT_DATABASE 1
18.3.81 PG_STAT_DATABASE_CONFLICTS 1
18.3.82 PG_STAT_USER_FUNCTIONS 1
18.3.83 PG_STAT_USER_INDEXES 1
18.3.84 PG_STAT_USER_TABLES 1
18.3.85 PG_STAT_REPLICATION 1
18.3.86 PG_STAT_SUBSCRIPTION 1
18.3.87 PG_STAT_SYS_INDEXES 1
18.3.88 PG_STAT_SYS_TABLES 1
18.3.89 PG_STAT_XACT_ALL_TABLES 1
18.3.90 PG_STAT_XACT_SYS_TABLES 1
18.3.91 PG_STAT_XACT_USER_FUNCTIONS 1
18.3.92 PG_STAT_XACT_USER_TABLES 1
18.3.93 PG_STATIO_ALL_INDEXES 1
18.3.94 PG_STATIO_ALL_SEQUENCES 1
18.3.95 PG_STATIO_ALL_TABLES 1
18.3.96 PG_STATIO_SYS_SEQUENCES 1

18.3.97 PG_STATIO_SYS_INDEXES 1
18.3.98 PG_STATIO_USER_INDEXES 1
18.3.99 PG_STATIO_USER_SEQUENCES 1
18.3.100 PG_STATIO_SYS_TABLES 1
18.3.101 PG_STATIO_USER_TABLES 1
18.3.102 PG_THREAD_WAIT_STATUS 1
18.3.103 PG_TABLES 1
18.3.104 PG_TDE_INFO 1
18.3.105 PG_TIMEZONE_NAMES 1
18.3.106 PG_TOTAL_USER_RESOURCE_INFO 1
18.3.107 PG_USER 1
18.3.108 PG_USER_MAPPINGS 1
18.3.109 PG_VIEWS 1
18.3.110 PG_TIMEZONE_ABBREVS 1
18.3.111 PG_VARIABLE_INFO 1
18.3.112 PG_SESSION_IOSTAT 1
18.3.113 PG_SESSION_WLMSTAT 1
18.3.114 PG_WLM_STATISTICS 1
18.3.115 PGXC_PREPARED_XACTS 1
18.3.116 PGXC_THREAD_WAIT_STATUS 1
18.3.117 PLAN_TABLE 1
18.3.118 GS_DB_PRIVILEGES 1
18.3.119 GS_GSC_MEMORY_DETAIL 1
18.3.120 GS_LSC_MEMORY_DETAIL 1
19 Schema 1
19.1 Information Schema 1

19.1.1 _PG_FOREIGN_DATA_WRAPPERS 1
19.1.2 _PG_FOREIGN_SERVERS 1
19.1.3 _PG_FOREIGN_TABLE_COLUMNS 1
19.1.4 _PG_FOREIGN_TABLES 1
19.1.5 _PG_USER_MAPPINGS 1
19.1.6 INFORMATION_SCHEMA_CATALOG_NAME 1
19.2 DBE_PERF Schema 1
19.2.1 OS 1
19.2.1.1 OS_RUNTIME 1
19.2.1.2 GLOBAL_OS_RUNTIME 1
19.2.1.3 OS_THREADS 1
19.2.1.4 GLOBAL_OS_THREADS 1
19.2.2 Instance 1
19.2.2.1 INSTANCE_TIME 1
19.2.2.2 GLOBAL_INSTANCE_TIME 1
19.2.3 Memory 1
19.2.3.1 MEMORY_NODE_DETAIL 1
19.2.3.2 GLOBAL_MEMORY_NODE_DETAIL 1
19.2.3.3 SHARED_MEMORY_DETAIL 1
19.2.3.4 GLOBAL_SHARED_MEMORY_DETAIL 1
19.2.4 File 1
19.2.4.1 FILE_IOSTAT 1
19.2.4.2 SUMMARY_FILE_IOSTAT 1
19.2.4.3 GLOBAL_FILE_IOSTAT 1
19.2.4.4 FILE_REDO_IOSTAT 1
19.2.4.5 SUMMARY_FILE_REDO_IOSTAT 1

19.2.4.6 GLOBAL_FILE_REDO_IOSTAT 1
19.2.4.7 LOCAL_REL_IOSTAT 1
19.2.4.8 GLOBAL_REL_IOSTAT 1
19.2.4.9 SUMMARY_REL_IOSTAT 1
19.2.5 Object 1
19.2.5.1 STAT_USER_TABLES 1
19.2.5.2 SUMMARY_STAT_USER_TABLES 1
19.2.5.3 GLOBAL_STAT_USER_TABLES 1
19.2.5.4 STAT_USER_INDEXES 1
19.2.5.5 SUMMARY_STAT_USER_INDEXES 1
19.2.5.6 GLOBAL_STAT_USER_INDEXES 1
19.2.5.7 STAT_SYS_TABLES 1
19.2.5.8 SUMMARY_STAT_SYS_TABLES 1
19.2.5.9 GLOBAL_STAT_SYS_TABLES 1
19.2.5.10 STAT_SYS_INDEXES 1
19.2.5.11 SUMMARY_STAT_SYS_INDEXES 1
19.2.5.12 GLOBAL_STAT_SYS_INDEXES 1
19.2.5.13 STAT_ALL_TABLES 1
19.2.5.14 SUMMARY_STAT_ALL_TABLES 1
19.2.5.15 GLOBAL_STAT_ALL_TABLES 1
19.2.5.16 STAT_ALL_INDEXES 1
19.2.5.17 SUMMARY_STAT_ALL_INDEXES 1
19.2.5.18 GLOBAL_STAT_ALL_INDEXES 1
19.2.5.19 STAT_DATABASE 1
19.2.5.20 SUMMARY_STAT_DATABASE 1
19.2.5.21 GLOBAL_STAT_DATABASE 1

19.2.5.22 STAT_DATABASE_CONFLICTS 1
19.2.5.23 SUMMARY_STAT_DATABASE_CONFLICTS 1
19.2.5.24 GLOBAL_STAT_DATABASE_CONFLICTS 1
19.2.5.25 STAT_XACT_ALL_TABLES 1
19.2.5.26 SUMMARY_STAT_XACT_ALL_TABLES 1
19.2.5.27 GLOBAL_STAT_XACT_ALL_TABLES 1
19.2.5.28 STAT_XACT_SYS_TABLES 1
19.2.5.29 SUMMARY_STAT_XACT_SYS_TABLES 1
19.2.5.30 GLOBAL_STAT_XACT_SYS_TABLES 1
19.2.5.31 STAT_XACT_USER_TABLES 1
19.2.5.32 SUMMARY_STAT_XACT_USER_TABLES 1
19.2.5.33 GLOBAL_STAT_XACT_USER_TABLES 1
19.2.5.34 STAT_XACT_USER_FUNCTIONS 1
19.2.5.35 SUMMARY_STAT_XACT_USER_FUNCTIONS 1
19.2.5.36 GLOBAL_STAT_XACT_USER_FUNCTIONS 1
19.2.5.37 STAT_BAD_BLOCK 1
19.2.5.38 SUMMARY_STAT_BAD_BLOCK 1
19.2.5.39 GLOBAL_STAT_BAD_BLOCK 1
19.2.5.40 STAT_USER_FUNCTIONS 1
19.2.5.41 SUMMARY_STAT_USER_FUNCTIONS 1
19.2.5.42 GLOBAL_STAT_USER_FUNCTIONS 1
19.2.6 Workload 1
19.2.6.1 WORKLOAD_SQL_COUNT 1
19.2.6.2 SUMMARY_WORKLOAD_SQL_COUNT 1
19.2.6.3 WORKLOAD_TRANSACTION 1
19.2.6.4 SUMMARY_WORKLOAD_TRANSACTION 1

19.2.6.5 GLOBAL_WORKLOAD_TRANSACTION 1
19.2.6.6 WORKLOAD_SQL_ELAPSE_TIME 1
19.2.6.7 SUMMARY_WORKLOAD_SQL_ELAPSE_TIME 1
19.2.6.8 USER_TRANSACTION 1
19.2.6.9 GLOBAL_USER_TRANSACTION 1
19.2.7 Session/Thread 1
19.2.7.1 SESSION_STAT 1
19.2.7.2 GLOBAL_SESSION_STAT 1
19.2.7.3 SESSION_TIME 1
19.2.7.4 GLOBAL_SESSION_TIME 1
19.2.7.5 SESSION_MEMORY 1
19.2.7.6 GLOBAL_SESSION_MEMORY 1
19.2.7.7 SESSION_MEMORY_DETAIL 1
19.2.7.8 GLOBAL_SESSION_MEMORY_DETAIL 1
19.2.7.9 SESSION_STAT_ACTIVITY 1
19.2.7.10 GLOBAL_SESSION_STAT_ACTIVITY 1
19.2.7.11 THREAD_WAIT_STATUS 1
19.2.7.12 GLOBAL_THREAD_WAIT_STATUS 1
19.2.7.13 LOCAL_THREADPOOL_STATUS 1
19.2.7.14 GLOBAL_THREADPOOL_STATUS 1
19.2.7.15 SESSION_CPU_RUNTIME 1
19.2.7.16 SESSION_MEMORY_RUNTIME 1
19.2.7.17 STATEMENT_IOSTAT_COMPLEX_RUNTIME 1
19.2.7.18 LOCAL_ACTIVE_SESSION 1
19.2.8 Transaction 1
19.2.8.1 TRANSACTIONS_PREPARED_XACTS 1

19.2.8.2 SUMMARY_TRANSACTIONS_PREPARED_XACTS 1
19.2.8.3 GLOBAL_TRANSACTIONS_PREPARED_XACTS 1
19.2.8.4 TRANSACTIONS_RUNNING_XACTS 1
19.2.8.5 SUMMARY_TRANSACTIONS_RUNNING_XACTS 1
19.2.8.6 GLOBAL_TRANSACTIONS_RUNNING_XACTS 1
19.2.9 Query 1
19.2.9.1 STATEMENT 1
19.2.9.2 SUMMARY_STATEMENT 1
19.2.9.3 STATEMENT_COUNT 1
19.2.9.4 GLOBAL_STATEMENT_COUNT 1
19.2.9.5 SUMMARY_STATEMENT_COUNT 1
19.2.9.6 GLOBAL_STATEMENT_COMPLEX_HISTORY 1
19.2.9.7 GLOBAL_STATEMENT_COMPLEX_HISTORY_TABLE 1
19.2.9.8 GLOBAL_STATEMENT_COMPLEX_RUNTIME 1
19.2.9.9 STATEMENT_RESPONSETIME_PERCENTILE 1
19.2.9.10 STATEMENT_USER_COMPLEX_HISTORY 1
19.2.9.11 STATEMENT_COMPLEX_HISTORY_TABLE 1
19.2.9.12 STATEMENT_COMPLEX_HISTORY 1
19.2.9.13 STATEMENT_COMPLEX_RUNTIME 1
19.2.9.14 STATEMENT_WLMSTAT_COMPLEX_RUNTIME 1
19.2.9.15 STATEMENT_HISTORY 1
19.2.10 Cache/IO 1
19.2.10.1 STATIO_USER_TABLES 1
19.2.10.2 SUMMARY_STATIO_USER_TABLES 1
19.2.10.3 GLOBAL_STATIO_USER_TABLES 1
19.2.10.4 STATIO_USER_INDEXES 1

19.2.10.5 SUMMARY_STATIO_USER_INDEXES 1
19.2.10.6 GLOBAL_STATIO_USER_INDEXES 1
19.2.10.7 STATIO_USER_SEQUENCES 1
19.2.10.8 SUMMARY_STATIO_USER_SEQUENCES 1
19.2.10.9 GLOBAL_STATIO_USER_SEQUENCES 1
19.2.10.10 STATIO_SYS_TABLES 1
19.2.10.11 SUMMARY_STATIO_SYS_TABLES 1
19.2.10.12 GLOBAL_STATIO_SYS_TABLES 1
19.2.10.13 STATIO_SYS_INDEXES 1
19.2.10.14 SUMMARY_STATIO_SYS_INDEXES 1
19.2.10.15 GLOBAL_STATIO_SYS_INDEXES 1
19.2.10.16 STATIO_SYS_SEQUENCES 1
19.2.10.17 SUMMARY_STATIO_SYS_SEQUENCES 1
19.2.10.18 GLOBAL_STATIO_SYS_SEQUENCES 1
19.2.10.19 STATIO_ALL_TABLES 1
19.2.10.20 SUMMARY_STATIO_ALL_TABLES 1
19.2.10.21 GLOBAL_STATIO_ALL_TABLES 1
19.2.10.22 STATIO_ALL_INDEXES 1
19.2.10.23 SUMMARY_STATIO_ALL_INDEXES 1
19.2.10.24 GLOBAL_STATIO_ALL_INDEXES 1
19.2.10.25 STATIO_ALL_SEQUENCES 1
19.2.10.26 SUMMARY_STATIO_ALL_SEQUENCES 1
19.2.10.27 GLOBAL_STATIO_ALL_SEQUENCES 1
19.2.10.28 GLOBAL_STAT_SESSION_CU 1
19.2.10.29 GLOBAL_STAT_DB_CU 1
19.2.11 Utility 1

19.2.11.1 REPLICATION_STAT 1
19.2.11.2 GLOBAL_REPLICATION_STAT 1
19.2.11.3 REPLICATION_SLOTS 1
19.2.11.4 GLOBAL_REPLICATION_SLOTS 1
19.2.11.5 BGWRITER_STAT 1
19.2.11.6 GLOBAL_BGWRITER_STAT 1
19.2.11.7 GLOBAL_CKPT_STATUS 1
19.2.11.8 GLOBAL_DOUBLE_WRITE_STATUS 1
19.2.11.9 GLOBAL_PAGEWRITER_STATUS 1
19.2.11.10 GLOBAL_RECORD_RESET_TIME 1
19.2.11.11 GLOBAL_REDO_STATUS 1
19.2.11.12 GLOBAL_RECOVERY_STATUS 1
19.2.11.13 CLASS_VITAL_INFO 1
19.2.11.14 USER_LOGIN 1
19.2.11.15 SUMMARY_USER_LOGIN 1
19.2.11.16 GLOBAL_GET_BGWRITER_STATUS 1
19.2.11.17 GLOBAL_SINGLE_FLUSH_DW_STATUS 1
19.2.11.18 GLOBAL_CANDIDATE_STATUS 1
19.2.12 Lock 1
19.2.12.1 LOCKS 1
19.2.12.2 GLOBAL_LOCKS 1
19.2.13 Wait Events 1
19.2.13.1 WAIT_EVENTS 1
19.2.13.2 GLOBAL_WAIT_EVENTS 1
19.2.14 Configuration 1
19.2.14.1 CONFIG_SETTINGS 1

19.2.14.2 GLOBAL_CONFIG_SETTINGS 1
19.2.15 Operator 1
19.2.15.1 OPERATOR_HISTORY_TABLE 1
19.2.15.2 OPERATOR_HISTORY 1
19.2.15.3 OPERATOR_RUNTIME 1
19.2.15.4 GLOBAL_OPERATOR_HISTORY 1
19.2.15.5 GLOBAL_OPERATOR_HISTORY_TABLE 1
19.2.15.6 GLOBAL_OPERATOR_RUNTIME 1
19.2.16 Workload Manager 1
19.2.16.1 WLM_USER_RESOURCE_CONFIG 1
19.2.16.2 WLM_USER_RESOURCE_RUNTIME 1
19.2.17 Global Plancache 1
19.2.17.1 GLOBAL_PLANCACHE_STATUS 1
19.2.17.2 GLOBAL_PLANCACHE_CLEAN 1
19.2.18 RTO & RPO 1
19.2.18.1 global_rto_status 1
19.3 WDR Snapshot Schema 1
19.3.1 WDR Snapshot 原信息表 1
19.3.1.1.1 SNAPSHOT.SNAPSHOT 1
19.3.1.1.2 SNAPSHOT.TABLES_SNAP_TIMESTAMP 1
19.3.1.1.3 SNAP_SEQ 1
19.3.2 WDR Snapshot 数据表 1
19.3.3 WDR Snapshot 生成性能报告 1
19.3.4 查看 WDR 报告 1
19.3.4.1 Database Stat 1
19.3.4.2 Load Profile 1

19.3.4.3 Instance Efficiency Percentages 1
19.3.4.4 Top 10 Events by Total Wait Time 1
19.3.4.5 Wait Classes by Total Wait Time 1
19.3.4.6 Host CPU 1
19.3.4.7 IO Profile 1
19.3.4.8 Memory Statistics 1
19.3.4.9 Time Model 1
19.3.4.10 SQL Statistics 1
19.3.4.11 Wait Events 1
19.3.4.12 Cache IO Stats 1
19.3.4.13 Utility status 1
19.3.4.14 Object stats 1
19.3.4.15 Configuration settings 1
19.3.4.16 SQL Detail 1
19.4 DBE_PLDEBUGGER Schema 1
19.4.1 DBE_PLDEBUGGER.turn_on 1
19.4.2 DBE_PLDEBUGGER.turn_off 1
19.4.3 DBE_PLDEBUGGER.local_debug_server_info 1
19.4.4 DBE_PLDEBUGGER.attach 1
19.4.5 DBE_PLDEBUGGER.next 1
19.4.6 DBE_PLDEBUGGER.continue 1
19.4.7 DBE_PLDEBUGGER.abort 1
19.4.8 DBE_PLDEBUGGER.print_var 1
19.4.9 DBE_PLDEBUGGER.info_code 1
19.4.10 DBE_PLDEBUGGER.step 1
19.4.11 DBE_PLDEBUGGER.delete_breakpoint 1

19.4.12 DBE_PLDEBUGGER.info_breakpoints 1
19.4.13 DBE_PLDEBUGGER.backtrace 1
19.4.14 DBE_PLDEBUGGER.enable_breakpoint 1
19.4.15 DBE_PLDEBUGGER.add_breakpoint 1
19.4.16 DBE_PLDEBUGGER.disable_breakpoint 1
19.4.17 DBE_PLDEBUGGER.finish 1
19.4.18 DBE_PLDEBUGGER.set_var 1
19.5 DB4AI Schema 1
19.5.1 DB4AI.SNAPSHOT 1
19.5.2 DB4AI.CREATE_SNAPSHOT 1
19.5.3 DB4AI.CREATE_SNAPSHOT_INTERNAL 1
19.5.4 DB4AI.PREPARE_SNAPSHOT 1
19.5.5 DB4AI.PREPARE_SNAPSHOT_INTERNAL 1
19.5.6 DB4AI.ARCHIVE_SNAPSHOT 1
19.5.7 DB4AI.PUBLISH_SNAPSHOT 1
19.5.8 DB4AI.MANAGE_SNAPSHOT_INTERNAL 1
19.5.9 DB4AI.SAMPLE_SNAPSHOT 1
19.5.10 DB4AI.PURGE_SNAPSHOT 1
19.5.11 DB4AI.PURGE_SNAPSHOT_INTERNAL 1
19.6 DBE_PLDEVELOPER 1
19.6.1 DBE_PLDEVELOPER.gs_source 1
19.6.2 DBE_PLDEVELOPER.gs_errors 1
20 逻辑复制 1
20.1 逻辑解码 1
20.1.1 逻辑解码概述 1
20.1.2 使用 SQL 函数接口进行逻辑解码 1

20.2 发布订阅 1
20.2.1 发布 1
20.2.2 订阅 1
20.2.3 冲突处理 1
20.2.4 限制 1
20.2.5 架构 1
20.2.6 监控 1
20.2.7 安全性 1
20.2.8 配置设置 1
20.2.9 快速设置 1
21 Foreign Data Wrapper 1
21.1 oracle_fdw 1
21.1.1 客户端配置 1
21.1.2 使用 oracle_fdw 1
21.1.3 常见问题 1
21.1.4 注意事项 1
21.2 mysql_fdw 1
21.2.1 安装 mysql_fdw 1
21.2.2 使用 mysql_fdw 1
21.2.3 常见问题 1
21.2.4 注意事项 1
21.3 postgres_fdw 1
21.3.1 使用 postgres_fdw 1
21.3.2 常见问题 1
21.3.3 注意事项 1
21.4 file_fdw 1

21.4.1 使用 file_fdw 1
21.4.2 注意事项 1
21.5 dblink 1
21.5.1 常用的 dblink 函数 1
21.5.2 注意事项 1
22 物化视图 1
22.1 全量物化视图 1
22.1.1 概述 1
22.1.2 使用 1
22.1.3 支持和约束 1
22.2 增量物化视图 1
22.2.1 概述 1
22.2.2 使用 1
22.2.3 支持和约束 1
23 GUC 参数说明 1
23.1 GUC 使用说明 1
23.2 文件位置 1
23.2.1 data_directory 1
23.2.2 config_file 1
23.2.3 hba_file 1
23.2.4 ident_file 1
23.2.5 external_pid_file 1
23.3 连接和认证 1
23.3.1 连接设置 1
23.3.1.1 listen_addresses 1
23.3.1.2 local_bind_address 1

23.3.1.3 port 1
23.3.1.4 max_connections 1
23.3.1.5 max_inner_tool_connections 1
23.3.1.6 sysadmin_reserved_connections 1
23.3.1.7 unix_socket_directory 1
23.3.1.8 unix_socket_group 1
23.3.1.9 unix_socket_permissions 1
23.3.1.10 application_name 1
23.3.1.11 connection_info 1
23.3.2 安全和认证（postgresql.conf） 1
23.3.2.1 authentication_timeout 1
23.3.2.2 auth_iteration_count 1
23.3.2.3 session_authorization 1
23.3.2.4 session_timeout 1
23.3.2.5 ssl 1
23.3.2.6 require_ssl 1
23.3.2.7 ssl_ciphers 1
23.3.2.8 ssl_renegotiation_limit 1
23.3.2.9 ssl_cert_file 1
23.3.2.10 ssl_key_file 1
23.3.2.11 ssl_ca_file 1
23.3.2.12 ssl_crl_file 1
23.3.2.13 krb_server_keyfile 1
23.3.2.14 krb_srvname 1
23.3.2.15 krb_caseins_users 1
23.3.2.16 password_policy 1

23.3.2.17 password_reuse_time 1
23.3.2.18 password_reuse_max 1
23.3.2.19 password_lock_time 1
23.3.2.20 failed_login_attempts 1
23.3.2.21 password_encryption_type 1
23.3.2.22 password_min_length 1
23.3.2.23 password_max_length 1
23.3.2.24 password_min_uppercase 1
23.3.2.25 password_min_lowercase 1
23.3.2.26 password_min_digital 1
23.3.2.27 password_min_special 1
23.3.2.28 password_effect_time 1
23.3.2.29 password_notify_time 1
23.3.3 通信库参数 1
23.3.3.1 tcp_keepalives_idle 1
23.3.3.2 tcp_keepalives_interval 1
23.3.3.3 tcp_keepalives_count 1
23.3.3.4 comm_proxy_attr 1
23.4 资源消耗 1
23.4.1 内存 1
23.4.1.1 memorypool_enable 1
23.4.1.2 memorypool_size 1
23.4.1.3 enable_memory_limit 1
23.4.1.4 max_process_memory 1
23.4.1.5 enable_memory_context_control 1
23.4.1.6 uncontrolled_memory_context 1

23.4.1.7 shared_buffers 1
23.4.1.8 segment_buffers 1
23.4.1.9 bulk_write_ring_size 1
23.4.1.10 standby_shared_buffers_fraction 1
23.4.1.11 temp_buffers 1
23.4.1.12 max_prepared_transactions 1
23.4.1.13 work_mem 1
23.4.1.14 query_mem 1
23.4.1.15 query_max_mem 1
23.4.1.16 maintenance_work_mem 1
23.4.1.17 psort_work_mem 1
23.4.1.18 max_loaded_cudesc 1
23.4.1.19 max_stack_depth 1
23.4.1.20 cstore_buffers 1
23.4.1.21 bulk_read_ring_size 1
23.4.1.22 enable_early_free 1
23.4.1.23 local_syscache_threshold 1
23.4.1.24 resilience_memory_reject_percent 1
23.4.2 磁盘空间 1
23.4.2.1 sql_use_spacelimit 1
23.4.2.2 temp_file_limit 1
23.4.3 内核资源使用 1
23.4.3.1 max_files_per_process 1
23.4.3.2 shared_preload_libraries 1
23.4.4 基于开销的清理延迟 1
23.4.4.1 背景信息 1

23.4.4.2 vacuum_cost_delay 1
23.4.4.3 vacuum_cost_page_hit 1
23.4.4.4 vacuum_cost_page_miss 1
23.4.4.5 vacuum_cost_page_dirty 1
23.4.4.6 vacuum_cost_limit 1
23.4.5 后端写进程 1
23.4.5.1 bgwriter_delay 1
23.4.5.2 candidate_buf_percent_target 1
23.4.5.3 bgwriter_lru_maxpages 1
23.4.5.4 bgwriter_lru_multiplier 1
23.4.5.5 pagewriter_thread_num 1
23.4.5.6 dirty_page_percent_max 1
23.4.5.7 pagewriter_sleep 1
23.4.5.8 max_io_capacity 1
23.4.5.9 enable_consider_usecount 1
23.4.5.10 dw_file_num 1
23.4.5.11 dw_file_size 1
23.4.6 异步 IO 1
23.4.6.1 enable_adio_debug 1
23.4.6.2 enable_adio_function 1
23.4.6.3 enable_fast_allocate 1
23.4.6.4 prefetch_quantity 1
23.4.6.5 backwrite_quantity 1
23.4.6.6 cstore_prefetch_quantity 1
23.4.6.7 cstore_backwrite_quantity 1
23.4.6.8 cstore_backwrite_max_threshold 1

23.4.6.9 fast_extend_file_size 1
23.4.6.10 effective_io_concurrency 1
23.4.6.11 checkpoint_flush_after 1
23.4.6.12 bgwriter_flush_after 1
23.4.6.13 backend_flush_after 1
23.5 预写式日志 1
23.5.1 设置 1
23.5.1.1 wal_level 1
23.5.1.2 fsync 1
23.5.1.3 synchronous_commit 1
23.5.1.4 wal_sync_method 1
23.5.1.5 full_page_writes 1
23.5.1.6 wal_log_hints 1
23.5.1.7 wal_buffers 1
23.5.1.8 wal_writer_delay 1
23.5.1.9 commit_delay 1
23.5.1.10 commit_siblings 1
23.5.1.11 wal_block_size 1
23.5.1.12 wal_segment_size 1
23.5.1.13 walwriter_cpu_bind 1
23.5.1.14 walwriter_sleep_threshold 1
23.5.1.15 wal_file_init_num 1
23.5.1.16 force_promote 1
23.5.1.17 wal_flush_timeout 1
23.5.1.18 wal_flush_delay 1
23.5.2 检查点 1

23.5.2.1 checkpoint_segments 1
23.5.2.2 checkpoint_timeout 1
23.5.2.3 checkpoint_completion_target 1
23.5.2.4 checkpoint_warning 1
23.5.2.5 checkpoint_wait_timeout 1
23.5.2.6 enable_incremental_checkpoint 1
23.5.2.7 enable_double_write 1
23.5.2.8 incremental_checkpoint_timeout 1
23.5.2.9 enable_xlog_prune 1
23.5.2.10 max_redo_log_size 1
23.5.2.11 max_size_for_xlog_prune 1
23.5.3 日志回放 1
23.5.3.1 recovery_time_target 1
23.5.3.2 recovery_max_workers 1
23.5.3.3 recovery_parse_workers 1
23.5.3.4 recovery_redo_workers 1
23.5.3.5 recovery_parallelism 1
23.5.3.6 enable_page_lsn_check 1
23.5.3.7 redo_bind_cpu_attr 1
23.5.4 归档 1
23.5.4.1 archive_mode 1
23.5.4.2 archive_command 1
23.5.4.3 archive_dest 1
23.5.4.4 archive_timeout 1
23.6 双机复制 1
23.6.1 发送端服务器 1

23.6.1.1 max_wal_senders 1
23.6.1.2 wal_keep_segments 1
23.6.1.3 wal_sender_timeout 1
23.6.1.4 max_replication_slots 1
23.6.1.5 enable_slot_log 1
23.6.1.6 max_changes_in_memory 1
23.6.1.7 max_cached_tuplebufs 1
23.6.1.8 enable_wal_shipping_compression 1
23.6.1.9 repl_auth_mode 1
23.6.1.10 repl_uuid 1
23.6.1.11 replconninfo1 1
23.6.1.12 replconninfo2 1
23.6.1.13 replconninfo3 1
23.6.1.14 replconninfo4 1
23.6.1.15 replconninfo5 1
23.6.1.16 replconninfo6 1
23.6.1.17 replconninfo7 1
23.6.1.18 replconninfo8 1
23.6.1.19 cross_cluster_replconninfo1 1
23.6.1.20 cross_cluster_replconninfo2 1
23.6.1.21 cross_cluster_replconninfo3 1
23.6.1.22 cross_cluster_replconninfo4 1
23.6.1.23 cross_cluster_replconninfo5 1
23.6.1.24 cross_cluster_replconninfo6 1
23.6.1.25 cross_cluster_replconninfo7 1
23.6.1.26 cross_cluster_replconninfo8 1

23.6.1.27 available_zone 1
23.6.2 主服务器 1
23.6.2.1 synchronous_standby_names 1
23.6.2.2 most_available_sync 1
23.6.2.3 keep_sync_window 1
23.6.2.4 enable_stream_replication 1
23.6.2.5 enable_mix_replication 1
23.6.2.6 vacuum_defer_cleanup_age 1
23.6.2.7 data_replicate_buffer_size 1
23.6.2.8 walsender_max_send_size 1
23.6.2.9 enable_data_replicate 1
23.6.2.10 ha_module_debug 1
23.6.2.11 enable_incremental_catchup 1
23.6.2.12 wait_dummy_time 1
23.6.2.13 catchup2normal_wait_time 1
23.6.2.14 sync_config_strategy 1
23.6.2.15 enable_save_confirmed_lsn 1
23.6.3 备服务器 1
23.6.3.1 hot_standby 1
23.6.3.2 max_standby_archive_delay 1
23.6.3.3 max_standby_streaming_delay 1
23.6.3.4 wal_receiver_status_interval 1
23.6.3.5 hot_standby_feedback 1
23.6.3.6 wal_receiver_timeout 1
23.6.3.7 wal_receiver_connect_timeout 1
23.6.3.8 wal_receiver_connect_retries 1

23.6.3.9 wal_receiver_buffer_size 1
23.6.3.10 primary_slotname 1
23.6.3.11 max_logical_replication_workers 1
23.7 内存表 1
23.7.1.1 enable_codegen_mot 1
23.7.1.2 force_pseudo_codegen_mot 1
23.7.1.3 enable_codegen_mot_print 1
23.7.1.4 codegen_mot_limit 1
23.7.1.5 mot_allow_index_on_nullable_column 1
23.7.1.6 mot_config_file 1
23.8 查询规划 1
23.8.1 优化器方法配置 1
23.8.1.1 enable_broadcast 1
23.8.1.2 enable_bitmapscan 1
23.8.1.3 force_bitmapand 1
23.8.1.4 enable_hashagg 1
23.8.1.5 enable_hashjoin 1
23.8.1.6 enable_indexscan 1
23.8.1.7 enable_indexonlyscan 1
23.8.1.8 enable_material 1
23.8.1.9 enable_mergejoin 1
23.8.1.10 enable_nestloop 1
23.8.1.11 enable_index_nestloop 1
23.8.1.12 enable_seqscan 1
23.8.1.13 enable_sort 1
23.8.1.14 enable_tidscan 1

23.8.1.15 enable_kill_query 1
23.8.1.16 enforce_a_behavior 1
23.8.1.17 max_recursive_times 1
23.8.1.18 enable_vector_engine 1
23.8.1.19 enable_change_hjcost 1
23.8.1.20 enable_absolute_tablespace 1
23.8.1.21 enable_valuepartition_pruning 1
23.8.1.22 qrw_inlist2join_optmode 1
23.8.1.23 skew_option 1
23.8.1.24 default_limit_rows 1
23.8.1.25 check_implicit_conversions 1
23.8.1.26 cost_weight_index 1
23.8.1.27 try_vector_engine_strategy 1
23.8.2 优化器开销常量 1
23.8.2.1 seq_page_cost 1
23.8.2.2 random_page_cost 1
23.8.2.3 cpu_tuple_cost 1
23.8.2.4 cpu_index_tuple_cost 1
23.8.2.5 cpu_operator_cost 1
23.8.2.6 effective_cache_size 1
23.8.2.7 allocate_mem_cost 1
23.8.3 基因查询优化器 1
23.8.3.1 geqo 1
23.8.3.2 geqo_threshold 1
23.8.3.3 geqo_effort 1
23.8.3.4 geqo_pool_size 1

23.8.3.5 geqo_generations 1
23.8.3.6 geqo_selection_bias 1
23.8.3.7 geqo_seed 1
23.8.4 其他优化器选项 1
23.8.4.1 enable_stream_recursive 1
23.8.4.2 explain_dna_file 1
23.8.4.3 explain_perf_mode 1
23.8.4.4 analysis_options 1
23.8.4.5 cost_param 1
23.8.4.6 enable_partitionwise 1
23.8.4.7 rewrite_rule 1
23.8.4.8 enable_pbe_optimization 1
23.8.4.9 enable_global_plancache 1
23.8.4.10 gpc_clean_timeout 1
23.8.4.11 enable_global_stats 1
23.8.4.12 enable_opfusion 1
23.8.4.13 enable_partition_opfusion 1
23.8.4.14 sql_beta_feature 1
23.8.4.15 ngram_gram_size 1
23.8.4.16 ngram_grapsymbol_ignore 1
23.8.4.17 ngram_punctuation_ignore 1
23.8.4.18 default_statistics_target 1
23.8.4.19 constraint_exclusion 1
23.8.4.20 cursor_tuple_fraction 1
23.8.4.21 from_collapse_limit 1
23.8.4.22 join_collapse_limit 1

23.8.4.23 plan_mode_seed 1
23.8.4.24 hashagg_table_size 1
23.8.4.25 enable_codegen 1
23.8.4.26 codegen_strategy 1
23.8.4.27 enable_codegen_print 1
23.8.4.28 codegen_cost_threshold 1
23.8.4.29 enable_bloom_filter 1
23.8.4.30 enable_extrapolation_stats 1
23.8.4.31 autoanalyze 1
23.8.4.32 enable_analyze_check 1
23.8.4.33 enable_sonic_hashagg 1
23.8.4.34 enable_sonic_hashjoin 1
23.8.4.35 enable_sonic_optspill 1
23.8.4.36 log_parser_stats 1
23.8.4.37 log_planner_stats 1
23.8.4.38 log_executor_stats 1
23.8.4.39 log_statement_stats 1
23.8.4.40 plan_cache_mode 1
23.8.4.41 enable_hypo_index 1
23.8.4.42 enable_force_vector_engine 1
23.8.4.43 enable_auto_explain 1
23.8.4.44 auto_explain_level 1
23.8.4.45 query_dop 1
23.8.4.46 enable_startwith_debug 1
23.9 错误报告和日志 1
23.9.1 记录日志的位置 1

23.9.1.1 log_destination 1
23.9.1.2 logging_collector 1
23.9.1.3 log_directory 1
23.9.1.4 log_filename 1
23.9.1.5 log_file_mode 1
23.9.1.6 log_truncate_on_rotation 1
23.9.1.7 log_rotation_age 1
23.9.1.8 log_rotation_size 1
23.9.1.9 syslog_facility 1
23.9.1.10 syslog_ident 1
23.9.1.11 event_source 1
23.9.2 记录日志的时间 1
23.9.2.1 client_min_messages 1
23.9.2.2 log_min_messages 1
23.9.2.3 log_min_error_statement 1
23.9.2.4 log_min_duration_statement 1
23.9.2.5 backtrace_min_messages 1
23.9.2.6 plog_merge_age 1
23.9.3 记录日志的内容 1
23.9.3.1 debug_print_parse 1
23.9.3.2 debug_print_rewritten 1
23.9.3.3 debug_print_plan 1
23.9.3.4 debug_pretty_print 1
23.9.3.5 log_checkpoints 1
23.9.3.6 log_connections 1
23.9.3.7 log_disconnections 1

23.9.3.8 log_duration 1
23.9.3.9 log_error_verbosity 1
23.9.3.10 log_hostname 1
23.9.3.11 log_line_prefix 1
23.9.3.12 log_lock_waits 1
23.9.3.13 log_statement 1
23.9.3.14 log_temp_files 1
23.9.3.15 log_timezone 1
23.9.3.16 logging_module 1
23.9.3.17 opfusion_debug_mode 1
23.9.3.18 enable_debug_vacuum 1
23.9.4 使用 CSV 格式写日志 1
23.10 告警检测 1
23.10.1 enable_alarm 1
23.10.2 connection_alarm_rate 1
23.10.3 alarm_report_interval 1
23.10.4 alarm_component 1
23.10.5 table_skewness_warning_threshold 1
23.10.6 table_skewness_warning_rows 1
23.11 运行时统计 1
23.11.1 查询和索引统计收集器 1
23.11.1.1 track_activities 1
23.11.1.2 track_counts 1
23.11.1.3 track_io_timing 1
23.11.1.4 track_functions 1
23.11.1.5 track_activity_query_size 1

23.11.1.6 stats_temp_directory 1
23.11.1.7 track_thread_wait_status_interval 1
23.11.1.8 enable_save_datachanged_timestamp 1
23.11.1.9 track_sql_count 1
23.11.2 性能统计 1
23.11.2.1 输出性能统计日志 1
23.12 负载管理 1
23.12.1 use_workload_manager 1
23.12.2 cgroup_name 1
23.12.3 cpu_collect_timer 1
23.12.4 memory_tracking_mode 1
23.12.5 memory_detail_tracking 1
23.12.6 enable_resource_track 1
23.12.7 enable_resource_record 1
23.12.8 enable_logical_io_statistics 1
23.12.9 enable_user_metric_persistent 1
23.12.10 user_metric_retention_time 1
23.12.11 enable_instance_metric_persistent 1
23.12.12 instance_metric_retention_time 1
23.12.13 resource_track_level 1
23.12.14 resource_track_cost 1
23.12.15 resource_track_duration 1
23.12.16 disable_memory_protect 1
23.12.17 query_band 1
23.12.18 memory_fault_percent 1
23.12.19 enable_bbox_dump 1

23.12.20 bbox_dump_count 1
23.12.21 bbox_dump_path 1
23.12.22 bbox_blanklist_items 1
23.12.23 enable_ffic_log 1
23.12.24 io_limits 1
23.12.25 io_priority 1
23.12.26 io_control_unit 1
23.12.27 session_respool 1
23.12.28 session_statistics_memory 1
23.12.29 topsql_retention_time 1
23.12.30 session_history_memory 1
23.12.31 transaction_pending_time 1
23.13 自动清理 1
23.13.1 autovacuum 1
23.13.2 autovacuum_mode 1
23.13.3 autoanalyze_timeout 1
23.13.4 autovacuum_io_limits 1
23.13.5 log_autovacuum_min_duration 1
23.13.6 autovacuum_max_workers 1
23.13.7 autovacuum_naptime 1
23.13.8 autovacuum_vacuum_threshold 1
23.13.9 autovacuum_analyze_threshold 1
23.13.10 autovacuum_vacuum_scale_factor 1
23.13.11 autovacuum_analyze_scale_factor 1
23.13.12 autovacuum_freeze_max_age 1
23.13.13 autovacuum_vacuum_cost_delay 1

23.13.14 autovacuum_vacuum_cost_limit 1
23.13.15 defer_csn_cleanup_time 1
23.14 客户端连接缺省设置 1
23.14.1 语句行为 1
23.14.1.1 exclude_reserved_words 1
23.14.1.2 search_path 1
23.14.1.3 current_schema 1
23.14.1.4 default_tablespace 1
23.14.1.5 temp_tablespaces 1
23.14.1.6 check_function_bodies 1
23.14.1.7 default_transaction_isolation 1
23.14.1.8 default_transaction_read_only 1
23.14.1.9 default_transaction_deferrable 1
23.14.1.10 session_replication_role 1
23.14.1.11 statement_timeout 1
23.14.1.12 vacuum_freeze_min_age 1
23.14.1.13 vacuum_freeze_table_age 1
23.14.1.14 bytea_output 1
23.14.1.15 xmlbinary 1
23.14.1.16 xmloption 1
23.14.1.17 max_compile_functions 1
23.14.1.18 gin_pending_list_limit 1
23.14.2 区域和格式化 1
23.14.2.1 DateStyle 1
23.14.2.2 IntervalStyle 1
23.14.2.3 TimeZone 1

23.14.2.4 timezone_abbreviations 1
23.14.2.5 extra_float_digits 1
23.14.2.6 client_encoding 1
23.14.2.7 lc_messages 1
23.14.2.8 lc_monetary 1
23.14.2.9 lc_numeric 1
23.14.2.10 lc_time 1
23.14.2.11 default_text_search_config 1
23.14.3 其他缺省 1
23.14.3.1 dynamic_library_path 1
23.14.3.2 gin_fuzzy_search_limit 1
23.14.3.3 local_preload_libraries 1
23.15 锁管理 1
23.15.1 deadlock_timeout 1
23.15.2 lockwait_timeout 1
23.15.3 update_lockwait_timeout 1
23.15.4 max_locks_per_transaction 1
23.15.5 max_pred_locks_per_transaction 1
23.15.6 gs_clean_timeout 1
23.15.7 partition_lock_upgrade_timeout 1
23.15.8 fault_mon_timeout 1
23.15.9 enable_online_ddl_waitlock 1
23.15.10 xloginsert_locks 1
23.15.11 num_internal_lock_partitions 1
23.16 版本和平台兼容性 1
23.16.1 历史版本兼容性 1

23.16.1.1 array_nulls 1
23.16.1.2 backslash_quote 1
23.16.1.3 escape_string_warning 1
23.16.1.4 lo_compat_privileges 1
23.16.1.5 quote_all_identifiers 1
23.16.1.6 sql_inheritance 1
23.16.1.7 standard_conforming_strings 1
23.16.1.8 synchronize_seqscans 1
23.16.1.9 enable_beta_features 1
23.16.2 平台和客户端兼容性 1
23.16.2.1 convert_string_to_digit 1
23.16.2.2 nls_timestamp_format 1
23.16.2.3 max_function_args 1
23.16.2.4 transform_null_equals 1
23.16.2.5 support_extended_features 1
23.16.2.6 sql_compatibility 1
23.16.2.7 behavior_compat_options 1
23.16.2.8 plpgsql.variable_conflict 1
23.16.2.9 td_compatible_truncation 1
23.16.2.10 uppercase_attribute_name 1
23.16.2.11 lastval_supported 1
23.17 容错性 1
23.17.1 exit_on_error 1
23.17.2 restart_after_crash 1
23.17.3 omit_encoding_error 1
23.17.4 max_query_retry_times 1

23.17.5 cn_send_buffer_size 1
23.17.6 max_cn_temp_file_size 1
23.17.7 retry_ecode_list 1
23.17.8 data_sync_retry 1
23.17.9 remote_read_mode 1
23.18 连接池参数 1
23.18.1 pooler_maximum_idle_time 1
23.18.2 minimum_pool_size 1
23.18.3 cache_connection 1
23.19 GBase 8c 事务 1
23.19.1 autocommit 1
23.19.2 enable_pgstat 1
23.19.3 transaction_isolation 1
23.19.4 transaction_read_only 1
23.19.5 xc_maintenance_mode 1
23.19.6 allow_concurrent_tuple_update 1
23.19.7 transaction_deferrable 1
23.19.8 enable_show_any_tuples 1
23.19.9 replication_type 1
23.19.10 pgxc_node_name 1
23.19.11 enable_defer_calculate_snapshot 1
23.20 双数据库实例复制参数 1
23.20.1 RepOriginId 1
23.21 开发人员选项 1
23.21.1 allow_system_table_mods 1
23.21.2 debug_assertions 1

23.21.3 ignore_checksum_failure 1
23.21.4 ignore_system_indexes 1
23.21.5 post_auth_delay 1
23.21.6 pre_auth_delay 1
23.21.7 trace_notify 1
23.21.8 trace_recovery_messages 1
23.21.9 trace_sort 1
23.21.10 zero_damaged_pages 1
23.21.11 remotetype 1
23.21.12 max_user_defined_exception 1
23.21.13 enable_fast_numeric 1
23.21.14 enable_compress_spill 1
23.21.15 resource_track_log 1
23.21.16 show_acce_estimate_detail 1
23.21.17 support_batch_bind 1
23.21.18 numa_distribute_mode 1
23.21.19 log_pagewriter 1
23.21.20 advance_xlog_file_num 1
23.21.21 enable_beta_opfusion 1
23.21.22 string_hash_compatible 1
23.21.23 pldebugger_timeout 1
23.21.24 plsql_show_all_error 1
23.21.25 ustore_attr 1
23.22 审计 1
23.22.1 审计开关 1
23.22.1.1 audit_enabled 1

23.22.1.2 audit_directory 1
23.22.1.3 audit_data_format 1
23.22.1.4 audit_rotation_interval 1
23.22.1.5 audit_rotation_size 1
23.22.1.6 audit_resource_policy 1
23.22.1.7 audit_file_remain_time 1
23.22.1.8 audit_space_limit 1
23.22.1.9 audit_file_remain_threshold 1
23.22.1.10 audit_thread_num 1
23.22.2 用户和权限审计 1
23.22.2.1 audit_login_logout 1
23.22.2.2 audit_database_process 1
23.22.2.3 audit_user_locked 1
23.22.2.4 audit_user_violation 1
23.22.2.5 audit_grant_revoke 1
23.22.3 操作审计 1
23.22.3.1 audit_system_object 1
23.22.3.2 audit_dml_state 1
23.22.3.3 audit_dml_state_select 1
23.22.3.4 audit_function_exec 1
23.22.3.5 audit_copy_exec 1
23.22.3.6 audit_set_parameter 1
23.22.3.7 audit_xid_info 1
23.22.3.8 enableSeparationOfDuty 1
23.22.3.9 enable_nonsysadmin_execute_direct 1
23.22.3.10 enable_access_server_directory 1

23.23 CM 相关参数 1
23.23.1 cm_agent 参数 1
23.23.1.1 log_dir 1
23.23.1.2 log_file_size 1
23.23.1.3 log_min_messages 1
23.23.1.4 incremental_build 1
23.23.1.5 alarm_component 1
23.23.1.6 alarm_report_interval 1
23.23.1.7 alarm_report_max_count 1
23.23.1.8 agent_report_interval 1
23.23.1.9 agent_phony_dead_check_interval 1
23.23.1.10 agent_check_interval 1
23.23.1.11 agent_heartbeat_timeout 1
23.23.1.12 agent_connect_timeout 1
23.23.1.13 agent_connect_retries 1
23.23.1.14 agent_kill_instance_timeout 1
23.23.1.15 security_mode 1
23.23.1.16 upgrade_from 1
23.23.1.17 process_cpu_affinity 1
23.23.1.18 log_threshold_check_interval 1
23.23.1.19 dilatation_shard_count_for_disk_capacity_alarm 1
23.23.1.20 log_max_size 1
23.23.1.21 log_max_count 1
23.23.1.22 log_saved_days 1
23.23.1.23 enable_log_compress 1
23.23.1.24 log_pattern_%s 1

23.23.1.25 agent_backup_open 1
23.23.1.26 enable_xc_maintenance_mode 1
23.23.1.27 unix_socket_directory 1
23.23.1.28 enable_dcf 1
23.23.1.29 disaster_recovery_type 1
23.23.2 cm_server 参数 1
23.23.2.1 log_dir 1
23.23.2.2 log_file_size 1
23.23.2.3 log_min_messages 1
23.23.2.4 thread_count 1
23.23.2.5 alarm_component 1
23.23.2.6 instance_failover_delay_timeout 1
23.23.2.7 instance_heartbeat_timeout 1
23.23.2.8 cmserver_ha_connect_timeout 1
23.23.2.9 cmserver_ha_heartbeat_timeout 1
23.23.2.10 phony_dead_effective_time 1
23.23.2.11 enable_transaction_read_only 1
23.23.2.12 datastorage_threshold_check_interval 1
23.23.2.13 datastorage_threshold_value_check 1
23.23.2.14 max_datastorage_threshold_check 1
23.23.2.15 cmserver_ha_status_interval 1
23.23.2.16 cmserver_self_vote_timeout 1
23.23.2.17 alarm_report_interval 1
23.23.2.18 alarm_report_max_count 1
23.23.2.19 enable_az_auto_switchover 1
23.23.2.20 instance_keep_heartbeat_timeout 1

23.23.2.21 az_switchover_threshold 1
23.23.2.22 az_check_and_arbitrate_interval 1
23.23.2.23 az_connect_check_interval 1
23.23.2.24 az_connect_check_delay_time 1
23.23.2.25 cmserver_demote_delay_on_etcd_fault 1
23.23.2.26 instance_phony_dead_restart_interval 1
23.23.2.27 cm_auth_method 1
23.23.2.28 cm_krb_server_keyfile 1
23.23.2.29 cm_server_arbitrate_delay_base_time_out 1
23.23.2.30 cm_server_arbitrate_delay_incrememtal_time_out 1
23.23.2.31 force_promote 1
23.23.2.32 switch_rto 1
23.23.2.33 backup_open 1
23.23.2.34 enable_dcf 1
23.23.2.35 install_type 1
23.23.2.36 enable_ssl 1
23.23.2.37 ssl_cert_expire_alert_threshold 1
23.23.2.38 ssl_cert_expire_check_interval 1
23.23.2.39 delay_arbitrate_timeout 1
23.23.2.40 ddb_type 1
23.23.2.41 ddb_log_level 1
23.23.2.42 ddb_log_backup_file_count 1
23.23.2.43 ddb_max_log_file_size 1
23.23.2.44 ddb_log_suppress_enable 1
23.23.2.45 ddb_election_timeout 1
23.24 升级参数 1

23.24.1 IsInplaceUpgrade 1
23.24.2 inplace_upgrade_next_system_object_oids 1
23.24.3 upgrade_mode 1
23.25 其它选项 1
23.25.1 enable_l2k 1
23.25.2 enable_slot_log 1
23.25.3 l2k_conf 1
23.25.4 enable_stream_smp 1
23.25.5 enable_force_smp 1
23.25.6 enable_ustore 1
23.25.7 enable_default_ustore_table 1
23.25.8 reserve_space_for_nullable_atts 1
23.25.9 ustore_attr 1
23.25.10 server_version 1
23.25.11 server_version_num 1
23.25.12 block_size 1
23.25.13 segment_size 1
23.25.14 max_index_keys 1
23.25.15 integer_datetimes 1
23.25.16 lc_collate 1
23.25.17 lc_ctype 1
23.25.18 max_identifier_length 1
23.25.19 server_encoding 1
23.25.20 enable_upgrade_merge_lock_mode 1
23.25.21 transparent_encrypted_string 1
23.25.22 transparent_encrypt_kms_url 1

23.25.23 transparent_encrypt_kms_region 1
23.25.24 basebackup_timeout 1
23.25.25 datanode_heartbeat_interval 1
23.25.26 max_concurrent_autonomous_transactions 1
23.26 等待事件 1
23.26.1 enable_instr_track_wait 1
23.27 Query 1
23.27.1 instr_unique_sql_count 1
23.27.2 instr_unique_sql_track_type 1
23.27.3 enable_instr_rt_percentile 1
23.27.4 percentile 1
23.27.5 instr_rt_percentile_interval 1
23.27.6 enable_instr_cpu_timer 1
23.27.7 enable_stmt_track 1
23.27.8 track_stmt_session_slot 1
23.27.9 track_stmt_details_size 1
23.27.10 track_stmt_retention_time 1
23.27.11 track_stmt_stat_level 1
23.27.12 enable_auto_clean_unique_sql 1
23.28 系统性能快照 1
23.28.1 enable_wdr_snapshot 1
23.28.2 wdr_snapshot_retention_days 1
23.28.3 wdr_snapshot_query_timeout 1
23.28.4 wdr_snapshot_interval 1
23.28.5 asp_flush_mode 1
23.28.6 asp_flush_rate 1

23.28.7 asp_log_filename 1
23.28.8 asp_retention_days 1
23.28.9 asp_sample_interval 1
23.28.10 asp_sample_num 1
23.28.11 enable_asp 1
23.29 安全配置 1
23.29.1 elastic_search_ip_addr 1
23.29.2 enable_security_policy 1
23.29.3 use_elastic_search 1
23.29.4 is_sysadmin 1
23.29.5 enable_tde 1
23.29.6 tde_cmk_id 1
23.30 全局临时表 1
23.30.1 max_active_global_temporary_table 1
23.30.2 vacuum_gtt_defer_check_age 1
23.30.3 enable_gtt_concurrent_truncate 1
23.31 HyperLogLog 1
23.31.1 hll_default_log2m 1
23.31.2 hll_default_log2explicit 1
23.31.3 hll_default_log2sparse 1
23.31.4 hll_duplicate_check 1
23.32 用户自定义函数 1
23.32.1 udf_memory_limit 1
23.32.2 FencedUDFMemoryLimit 1
23.32.3 UDFWorkerMemHardLimit 1
23.32.4 pljava_vmoptions 1

23.33 定时任务 1
23.33.1 job_queue_processes 1
23.33.2 enable_prevent_job_task_startup 1
23.34 线程池 1
23.34.1 enable_thread_pool 1
23.34.2 thread_pool_attr 1
23.34.3 resilience_threadpool_reject_cond 1
23.35 备份恢复 1
23.35.1 operation_mode 1
23.35.2 enable_cbm_tracking 1
23.35.3 hadr_max_size_for_xlog_receiver 1
23.36 Undo 1
23.36.1 undo_space_limit_size 1
23.36.2 undo_limit_size_per_transaction 1
23.37 DCF 参数设置 1
23.37.1 enable_dcf 1
23.37.2 dcf_ssl 1
23.37.3 dcf_config 1
23.37.4 dcf_data_path 1
23.37.5 dcf_log_path 1
23.37.6 dcf_node_id 1
23.37.7 dcf_max_workers 1
23.37.8 dcf_truncate_threshold 1
23.37.9 dcf_election_timeout 1
23.37.10 dcf_enable_auto_election_priority 1
23.37.11 dcf_election_switch_threshold 1

23.37.12 dcf_run_mode 1
23.37.13 dcf_log_level 1
23.37.14 dcf_log_backup_file_count 1
23.37.15 dcf_max_log_file_size 1
23.37.16 dcf_socket_timeout 1
23.37.17 dcf_connect_timeout 1
23.37.18 dcf_mec_fragment_size 1
23.37.19 dcf_stg_pool_max_size 1
23.37.20 dcf_stg_pool_init_size 1
23.37.21 dcf_mec_pool_max_size 1
23.37.22 dcf_flow_control_disk_rawait_threshold 1
23.37.23 dcf_flow_control_net_queue_message_num_threshold 1
23.37.24 dcf_flow_control_cpu_threshold 1
23.37.25 dcf_mec_batch_size 1
23.37.26 dcf_mem_pool_max_size 1
23.37.27 dcf_mem_pool_init_size 1
23.37.28 dcf_compress_algorithm 1
23.37.29 dcf_compress_level 1
23.37.30 dcf_mec_channel_num 1
23.37.31 dcf_rep_append_thread_num 1
23.37.32 dcf_mec_agent_thread_num 1
23.37.33 dcf_mec_reactor_thread_num 1
23.37.34 dcf_log_file_permission 1
23.37.35 dcf_log_path_permission 1
23.38 闪回相关参数 1
23.38.1 enable_recyclebin 1

23.38.2 recyclebin_retention_time 1
23.38.3 vacuum_defer_cleanup_age 1
23.38.4 undo_retention_time 1
23.39 回滚相关参数 1
23.39.1 max_undo_workers 1
23.40 预留参数 1
23.41 AI 特性 1
23.41.1 enable_hypo_index 1
23.41.2 db4ai_snapshot_mode 1
23.41.3 db4ai_snapshot_version_delimiter 1
23.41.4 db4ai_snapshot_version_separator 1
23.41.5 unix_socket_directory 1
23.42 Global SysCache 参数 1
23.42.1 enable_global_syscache 1
23.42.2 global_syscache_threshold 1

1 文档简介

1.1 概述

本文档介如何设计、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。

1.2 读者对象

本文档是为基于 GBase 8c 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。

1.3 文档阅读约定

通用格式约定

表 1-1 通用格式约定

格式	说明
幼圆	正文采用幼圆表示。
Times New Roman	正文中的英文字母、数字采用 Times New Roman 表示。
粗体	表格编号及标题、图片编号及标题、多级标题、正文小节标题、注意说明等，采用幼圆/Times New Roman 粗体。
“”	引用 GBase 8c 安装及使用过程中部分显示内容
……	表示省略部分显示内容

命令行格式约定

表 1-2 命令行格式约定

格式	说明
幼圆	命令行语句采用幼圆表示。
粗体	执行 bash 命令或 SQL 语句，采用幼圆加粗表示。
大写英文（如 CREATE、SELECT）	表示 GBase 8c 数据库中关键字
斜体	表示 GBase 8c 数据库语法中，需要用户自定义的变量
……	表示省略部分显示内容
[]	用“[]”括起来的部分，表示在语法命令中为必选参数
<>	用“<>”括起来的部分，表示在语法命令中为可选参数
# 描述	由“#”开头的注释行
#bash 命令	表示以 root 超级用户执行此命令
$bash 命令	表示以普通用户执行此命令

符号格式约定

在本文中可能出现下列标志，它们所代表的含义如下。

表 1-3 符号格式约定

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

GBase 8c 数据库的 DN（Datanode）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。

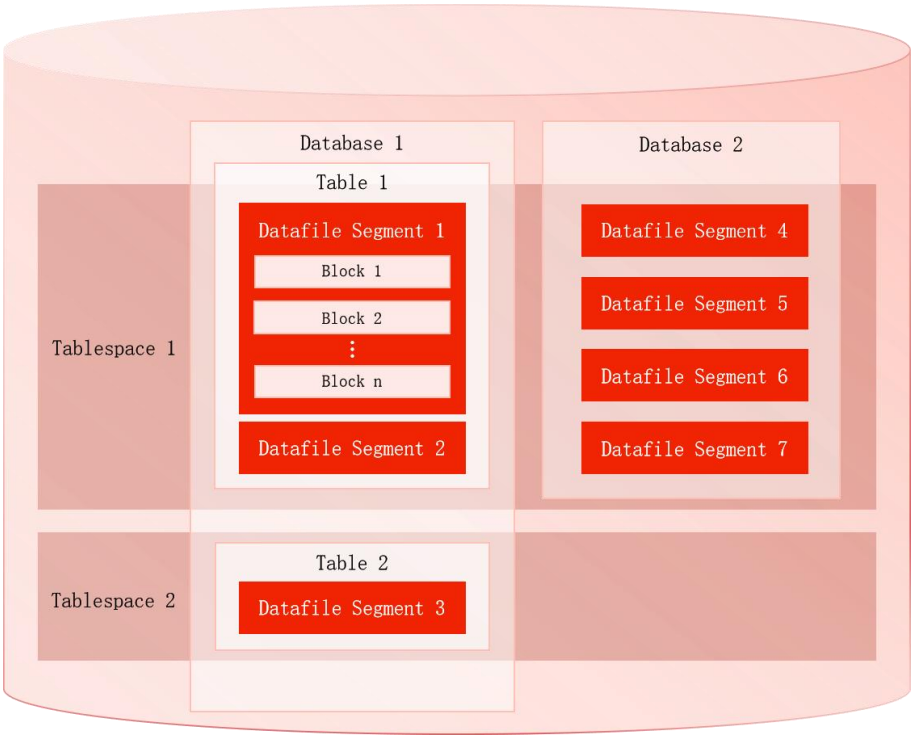


图 2-1 数据库逻辑结构图

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

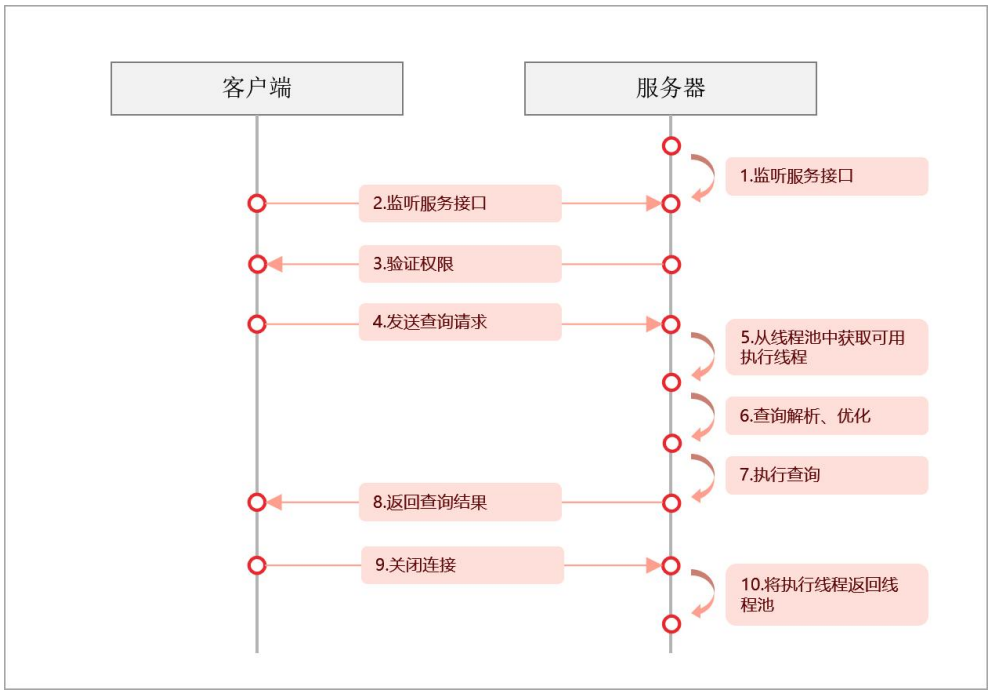


图 2-2 服务响应流程

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。GBase 8c 数据库支持的事务控制命令有启动、设置、提交、回滚事务。支持的事务隔离级别有读已提交和可重复读。

2.3.1 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 START TRANSACTION 和 BEGIN 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《GBase 8c V5_3.0.1_SQL 参考手册》SET TRANSACTION 章节。

- 提交事务

用户可以使用 COMMIT 或者 END 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《GBase 8c V5_3.0.1_SQL 参考手册》COMMIT | END 章节。

- 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《GBase 8c V5_3.0.1_SQL 参考手册》ROLLBACK 章节。

2.3.2 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

2.4.1 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。

2.4.2 表空间

在 GBase 8c 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。

2.4.3 模式

GBase 8c 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。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。

2.4.4 用户和角色

GBase 8c 通过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。其中，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 GBase 8c 数据库中，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。

2.4.5 事务管理

2.4.5.1 2PC

GBase 8c 采用两阶段提交（2PC）协议以保持分布式事务的原子性，防止出现部分 DN 提交部分 DN 回滚的“中间态”情况。两阶段提交协议将事务的提交操作分为两个阶段：

1、准备阶段：将所有提交操作所需要用到的信息和资源全部写入磁盘，完成持久化；

2、提交阶段：根据之前准备好的提交信息和资源，执行提交或回滚操作。

两阶段提交协议之所以能够保证分布式事务原子性的关键在于：一段准备阶段执行成功，那么提交需要的所有信息都完成持久化写入磁盘，即使后续提交阶段某个 DN 发生执行错误，该 DN 也可以再次从持久化的提交信息中尝试提交，直至提交成功。最终该分布式事务在所有 DN 上的状态一定是相同的，要么所有 DN 都提交，要么所有 DN 都回滚。因此，对外来讲，该事务的状态变化是原子性的。

增强特性：

1、内核功能：增强 2PC 处理流程，完善判决过程以及相关信息。

2、清理功能：增强对异常场景的覆盖，清理在异常场景下挂住的 2PC 事务，且需要保证事务在分布式数据库集群中 ACID 特性。

2.4.5.2 MVCC

在事务管理上，GBase 8c 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。MVCC 没有将历史版本数据统一存放，而是和当前元组的版本放在了一起。GBase 8c 数据库中没有回滚段的概念，但是为了定期清除历史版本数据，引入了一个 VACUUM 线程。一般情况下，除非用户要做性能调优，否则不用特别关注 vacuun 线程。此外，GBase 8c 单语句查询（不使用 begin 等语句显示事务块）是自动提交事务的。

3 数据库使用

3.1 概述

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。

GBase 8c 的 gsql 工具和关键字均支持自动补全功能，通过 tab 键自动补全命令。

前提条件

GBase 8c 数据库正常运行。

操作步骤

步骤 1 以操作系统用户 gbase 登录数据库主节点。

若不确定数据库主节点部署在哪台服务器，请参考 3.2.1 确认连接信息。

步骤 2 连接数据库。

```
[gbase@gbasehost ~]$ gsql -d postgres -p 5432
```

当结果显示为如下信息，则表示连接成功。

```
gsql ((GBase 8c V5 build a4c7bcc5) compiled at 2024-01-16 14:49:56 commit 0 last
mr 1632 )
Non-SSL connection (SSL connection is recommended when requiring high-security)
Last login: 2024-01-19 16:22:27+08 from [local] (user gbase)
Type "help" for help.

postgres=#
```

其中，postgres 为 GBase 8c 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，需根据数据库实际情况做替换，请参考 3.2.1 确认连接信息。

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。

gsql 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。

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

```
postgres=# CREATE USER joe WITH PASSWORD "xxx12GBase";
```

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 xxx12GBase 的用户。 如下命令为设置 joe 用户为系统管理员。

```
postgres=# GRANT ALL PRIVILEGES TO joe;
```

使用 GRANT 命令进行相关权限设置，具体操作请参考《GBase 8c V5_3.0.1_SQL 手册》GRANT 章节。引申信息：关于数据库用户的更多信息请参考 7.2 管理用户及权限。

步骤 4 创建数据库。

```
postgres=# 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 postgres 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 postgres 数据库下进行后续的体验。

```
postgres=# \q
[gbase@gbasehost ~]$ gsql -d postgres -p 5432 -U joe
Password for user joe:
gsql ((GBase 8c V5 build a4c7bcc5) compiled at 2024-01-16 14:49:56 commit 0 last
mr 1632 )
Non-SSL connection (SSL connection is recommended when requiring high-security)
Last login: 2024-01-19 16:34:22+08 from [local] (user joe)
Last failed login: 2024-01-19 16:34:17+08 from [local] (user joe)
There was 1 failed login attempt since the last successful login.
Type "help" for help.

postgres=>
```

步骤 5 创建 SCHEMA。

```
gbase=> 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

```
postgres=# 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 3.5 创建和管理表空间。

步骤 6 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。

```
postgres=# CREATE TABLE mytable (firstcol int);
CREATE TABLE
```

向表中插入数据：

```
postgres=# 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：

```
postgres=# 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 3.8.1 创建和管理 schema。

关于创建表的更多信息请参见 3.6 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 GBase 8c 安装信息以及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 3.7 查看系统表。

GBase 8c 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 3.4 规划存储模型。

----结束

3.2 连接数据库

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

- gsql 是数据库自带的客户端工具。可用于连接数据库，交互式地输入、编辑、执行 SQL 语句，参考 3.2.3 使用 gsql。
- 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 GBase 8c 的应用程序，参考 3.2.4 应用程序接口。

3.2.1 确认连接信息

客户端工具通过数据库主节点连接数据库。因此连接前，需获取数据库主节点所在服务器的 IP 地址及数据库主节点的端口号信息。

操作步骤

步骤 1 以操作系统用户 gbase，登录数据库主节点。

步骤 2 使用 gs_om -t status 命令，查询 GBase 8c 各实例情况。

返回信息中，“P”表示安装时指定的角色为 Primary，数据库安装后不在变化，从系统静态文件读取；“Primary”表示实例为主实例； “Normal”表示 GBase 8c 可用，且数据有冗余备份，所有进程都在运行，主备关系正常。

步骤 3 确认数据库主节点的端口号。

在步骤 2 查到的数据库主节点数据路径下的 postgresql.conf 文件中查看端口号信息。

```
[gbase@gbasehost ~]$ cat
/home/gbase/clusters/gbase8c/datanode/dn_1/postgresql.conf | grep port
```

----结束

3.2.2 配置服务端远程连接

进行远程连接前，需要在部署数据库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。

对于 GBase 8c 主备式场景，需要在数据主节点上进行；分布式场景下，需要在读写 CN 或主 DN 节点上进行，根据实际需要选择。

操作步骤

以下步骤需要在 GBase 8c 所在主机上执行。

步骤 1 以操作系统用户 gbase，登录数据库节点。

步骤 2 配置使用的客户端认证方式，允许客户端以某一用户连接到数据库。请参考 7.1.1 配置客户端接入认证。

步骤 3 配置 listen_addresses

listen_addresses 参数用于配置 TCP-IP 允许监听的地址，首先查看参数值，确认是否已经添加客户端机器的 IP。如果已经添加，则忽略此步骤；如果没有添加客户端 IP，则需要添加以允许其访问数据库节点。

(1) 使用 gs_guc 命令，查看数据库节点的 listen_addresses 配置。

```
gs_guc check <-Z coordinator/datanode/gtm> <-N NODE_NAME> {-I INSTANCE_NAME | -D DATADIR }  -c "listen_addresses"
```

注意

GBase 8c 主备式不涉及-Z 参数；分布式场景下必须指定-Z 参数。

例如，在主备式主 DN（节点 IP 为 192.168.142.146）上查询 listen_addresses，返回：

```
[gbase@gbasehost ~]$ gs_guc check -I all -c "listen_addresses"
expected guc information: gbase8c: listen_addresses=NULL:
[/home/gbase/clusters/gbase8c/datanode/dn1_1/postgresql.conf]
gs_guc check: gbase8c: listen_addresses='localhost,192.168.142.146':
[/home/gbase/clusters/gbase8c/datanode/dn1_1/postgresql.conf]

Total GUC values: 1. Failed GUC values: 0.
The value of parameter listen_addresses is same on all instances.
    listen_addresses='localhost, 192.168.142.146'
```

例中返回 listen_addresses='localhost, 192.168.142.146'，可以看出目前 listen_addresses 参数值并没有客户端 IP。

(2) 将要添加的客户端 IP 追加到 listen_addresses 后面，多个配置项之间用英文逗号分隔。

```
gs_guc set <-Z coordinator/datanode/gtm> <-N NODE_NAME> {-I INSTANCE_NAME | -D DATADIR } -c "listen_addresses=……"
```

注意

- GBase 8c 主备式不涉及-Z 参数；分布式场景下必须指定-Z 参数。
- 分布式场景下，需保证同 DN 组、同类型节点参数值一致，建议将 listen_addresses 参数值直接设置为*，表示允许任意 IP 访问数据库。

例如，在主备式场景下追加 IP 地址 10.11.12.13。

```
[gbase@gbasehost ~]$ gs_guc set -I all -c
"listen_addresses='localhost,192.168.142.146,10.11.12.13'"
```

步骤 4 配置 pg_hba.conf，添加数据库主节点 IP 和客户端 IP。详细说明，参见 7.1.1 配置客户端接入认证。

(1) 在 pg_hba.conf 配置文件中添加数据库主节点 IP，其中 initial_user 为数据库初始用户名称。

```
gs_guc reload <-Z coordinator/datanode/gtm> <-N NODE_NAME> {-I INSTANCE_NAME |
-D DATADIR } -h "host all initial_user hostip/32 trust"
```

例如，添加主 DN 节点 192.168.142.146。

```
[gbase@gbasehost ~]$ gs_guc reload -Z datanode -I all -h "host all gbase
192.168.142.146/32 trust"
```

(2) 在 pg_hba.conf 配置文件中，添加客户端 IP，认证方式为 sha256。

```
gs_guc reload <-Z coordinator/datanode/gtm> <-N NODE_NAME> {-I INSTANCE_NAME |
-D DATADIR } -h "host all all client_ip/32 sha256"
```

步骤 5 执行如下命令重启 GBase 8c。

```
[gbase@gbasehost ~]$ gha_ctl stop all -l dcslist [-c cluster]
[gbase@gbasehost ~]$ gha_ctl start all -l dcslist [-c cluster]
```

----结束

3.2.3 使用 gsql 连接

gsql 是 GBase 8c 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 gsql 连接数据库，关于 gsql 使用方法的更多信息请参考《GBase 8c V5_3.0.1_工具参考手册》中“客户端工具>gsql”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。

前提条件

已 3.2.1 确认连接信息。

3.2.3.1 本地连接数据库

步骤 1 以操作系统用户 gbase 登录数据库主节点。

步骤 2 连接数据库。

数据库安装完成后，默认生成名称为 postgres 的数据库。第一次连接数据库时可以连接到此数据库。

执行如下命令连接数据库。

```
[gbase@gbasehost ~]$ gsql -d postgres -p 5432
```

其中 postgres 为需要连接的数据库名称，5432 为数据库主节点的端口号。请根据实际情况替换。另外，也可以使用如下任一命令连接数据库。

```
[gbase@gbasehost ~]$ gsql postgres://gbase:gbase@host_ip:5432/postgres -r
gsql -d "host=127.0.0.1 port=5432 dbname=postgres user=gbase
password=gbase;123"
```

连接成功后，系统显示类似如下信息：

```
gsql ((GBase 8c V5 build a4c7bcc5) compiled at 2024-01-16 14:49:56 commit 0 last
mr 1632 )
Non-SSL connection (SSL connection is recommended when requiring high-security)
Last login: 2024-01-19 16:22:27+08 from [local] (user gbase)
Type "help" for help.

postgres=#
```

gbase 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，请使用 SSL 连接。

步骤 3 首次登录建议修改密码。命令如下。

```
postgres=# ALTER ROLE gbase IDENTIFIED BY 'XXXXXXXX' REPLACE 'XXXXXXXX';
```

步骤 4 退出数据库。

```
postgres=# \q
```

----结束

3.2.3.2 远程连接数据库

步骤 1 完成远程连接配置，操作步骤参见 3.2.2 配置服务端远程连接。

步骤 2 在客户端机器上，上传客户端工具包并配置 gsql 的执行环境变量。以客户端 IP 为 10.11.12.13 为例。

(1) 以 gbase 用户登录客户端机器。

(2) 创建安装目录。例如/tmp/tools。

```
[gbase@gbasehost ~]$ mkdir /tmp/tools
```

(3) 获取软件安装包中，并上传到安装目录下。以/tmp/tools 为例。

说明

- 软件包相对位置为安装时所放位置，根据实际情况填写。
- 不同的操作系统，工具包文件名称会有差异。请根据实际的操作系统类型选择对应的工具包。

(4) 解压文件。

```
cd /tmp/tools
tar -zxvf ******.tar.gz
```

(5) 登录数据库主节点所在的服务器，拷贝数据库安装目录下的 bin 目录到客户端主机的安装目录下。

```
scp -r /home/gbase/deploy/app/bin gbase@10.11.12.13:/tmp/tools
```

(6) 登录客户端所在主机，设置环境变量。打开“~/.bashrc”文件。

```
vi ~/.bashrc
```

在其中输入如下内容后，使用“:wq!”命令保存并退出

```
export PATH=/tmp/tools/bin:$PATH
export LD_LIBRARY_PATH=/tmp/tools/lib:$LD_LIBRARY_PATH
```

(7) 使环境变量配置生效。

```
source ~/.bashrc
```

步骤 3 连接数据库。

数据库安装完成后，默认生成名称为 postgres 的数据库。第一次连接数据库时可以连接

到此数据库。

```
gsql -d postgres -h host_ip -U jack -p 5432
Password for user jack:
```

postgres 为需要连接的数据库名称，host_ip 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。

说明

- 当连接机器与数据库所在主机不在同一网段时，-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 gbase 用户进行远程连接数据库。

----结束

3.2.4 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 GBase 8c 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。使用 JDBC 和 ODBC 接口连接数据库属远程连接，因此需要 GBase 8c 数据库做好支持远程连接的配置。

更多支持的应用程序接口详细信息请参考 6 应用程序开发教程。

3.3 字符集

GBase 8c 完全支持 GB18030 三级字符集，包括 GB18030-2022 所定义的所有字符和编码。这意味着数据库可以存储、处理和检索包含 GB18030 三级字符的数据，无论是存储中文文本、互联网内容还是其他语言字符。

GBase 8c 利用内部的字符集编码和转换机制，可以确保对 GB18030 三级字符的解析和处理的正确性和一致性。它能够正确地处理存储和查询操作，确保数据的完整性和可靠性，为用户提供了强大的中文字符处理能力，使得用户可以在数据库中方便地存储、查询和处理中文数据，满足复杂的语言处理需求。

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 7.2 管理用户及权限。

背景信息

- GBase 8c 数据库默认包含初始模板数据库 template0、template1、mydatabase，以及一个默认的用户数据库 postgres。postgres 默认的兼容数据库类型为 O（即 DBCOMPATIBILITY = A），该兼容类型下将空字符串作为 NULL 处理。
- CREATE DATABASE 实际上通过拷贝模板数据库来创建新数据库。可拷贝 template0 和 mydatabase。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。

  说明

  - 模板数据库中没有用户表，可通过系统表 PG_DATABASE 查看模板数据库属性。
  - 模板 template0 不允许用户连接；模板 template1 只允许数据库初始用户和系统管理员连接，普通用户无法连接。
- 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。当存在多个数据库时，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，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，可能造成出现半个字符的情况。

针对这种情况，请遵循以下条件：

- 保证数据对象的名称不超过限定长度。
- 修改数据库的默认存储编码集（server_encoding）为 utf-8 编码集。
- 不要使用多字节字符做为对象名。
- 创建的数据库总数目建议不超过 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 使用如下命令创建一个新的数据库 db_tpcc。

```
postgres=# CREATE DATABASE db_tpcc;
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。

```
postgres=# \l
```

- 使用如下命令通过系统表 pg_database 查询数据库列表。

```
postgres=# 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。

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

- 使用如下命令为数据库重新命名。

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

步骤 4 删除数据库

用户可以使用 DROP　DATABASE 命令删除数据库。这个命令删除了数据库中的系统目录，并且删除了磁盘上带有数据的数据库目录。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。删除数据库时请先连接到其他的数据库。

使用如下命令删除数据库：

```
postgres=# DROP DATABASE human_tpcds;
DROP DATABASE
```

----结束

3.5 规划存储模型

GBase 8c 数据库支持行列混合存储。行、列存储模型各有优劣，建议根据实际情况选择。通常默认使用行存储，适用于 TP 场景，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。

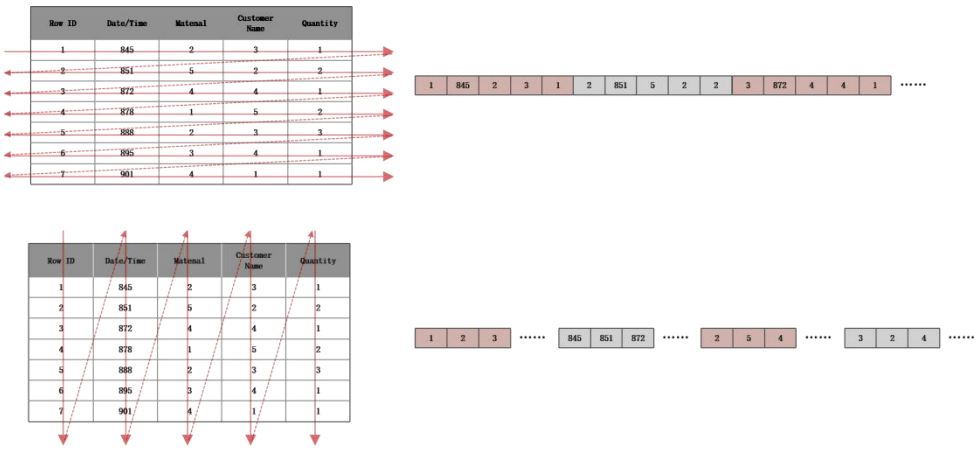


图 3-1 行存储和列存储的差异

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、列存储优缺点，如表 3-1 所示。

表 3-1 存储模型及优缺点

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。

表 3-2 存储模型及适用场景

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。
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）。

3.5.1 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。

```
postgres=# CREATE TABLE customer_t1
(
state_ID CHAR(2), state_NAME VARCHAR2(40),
area_ID NUMBER
);
--删除表
postgres=# DROP TABLE customer_t1;
```

3.5.2 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。

```
postgres=# CREATE TABLE customer_t2
(
state_ID CHAR(2), state_NAME VARCHAR2(40),
area_ID NUMBER
)
WITH (ORIENTATION = COLUMN);
--删除表
postgres=# DROP TABLE customer_t2;
```

3.5.3 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 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。根据实际情况，确认是否需要设置表空间的最大值。

  GBase 8c 自带了两个表空间：pg_default 和 pg_global。

- 默认表空间 pg_default：用来存储非共享系统表、用户表、用户表 index、临时 表、临时表 index、内部临时表的默认表空间。对应存储目录为实例数据目录下的 base 目录。

- 共享表空间 pg_global：用来存放共享系统表的表空间。对应存储目录为实例数据目录下的 global 目录。

注意事项

在 HCS 等场景下一般不建议用户使用自定义的表空间。用户自定义表空间通常配合主存（即默认表空间所在的存储设备，如磁盘）以外的其它存储介质使用，以隔离不同业务可以使用的 IO 资源，而在 HCS 等场景下，存储设备都是采用标准化的配置，无其它可用的存储介质，自定义表空间使用不当不利于系统长稳运行以及影响整体性能， 因此建议使用默认表空间即可。

操作步骤

步骤 1 创建表空间

- 执行如下命令创建用户 jack。

```
postgres=# CREATE USER jack IDENTIFIED BY 'xxxxxxxxx';
```

当结果显示为如下信息，则表示创建成功。

```
CREATE ROLE
```

- 执行如下命令创建表空间。

```
postgres=# 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，/pg_location/ tablespace/tablespace_1 是用户拥有读写权限的空目录。

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

```
postgres=# 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
```

步骤 2 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。

```
postgres=# 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。

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

假设设置 fastspace 为默认表空间，然后创建表 foo2。

步骤 3 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。

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

- 方式 2：使用 gsql 程序的元命令查询表空间。

```
postgres=# \db
```

步骤 4 查询表空间使用率

- 查询表空间的当前使用情况。

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

返回如下信息：

```
pg_tablespace_size
--------------------
6
(1 row)
```

其中 2146304 表示表空间的大小，单位为字节。

- 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。

步骤 5 修改表空间

执行如下命令对表空间 fastspace 重命名为 fspace。

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

步骤 6 删除表空间

- 执行如下命令删除用户 jack。

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

- 执行如下命令删除表 foo 和 foo2。

```
postgres=# DROP TABLE foo;
postgres=# 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。

```
postgres=# 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。

3.7 创建和管理表

3.7.1 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 3.4 规划存储模型。

创建表

执行如下命令创建表。

```
postgres=# 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)分别是这四个字段名称的类型。

3.7.2 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 INSERT 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 9 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 3-3 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。

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

表 3-4 客户端和服务端设置字符集的输出结果对比

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	postgres=# INSERT INTO table1 VALUES(1,reverse('12 3A A 78'),reverse('123A A 78'),reverse('123A 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	postgres=# INSERT INTO table1 VALUES(2,reverse('12 3A 78'),reverse('123A78'),reverse('123A 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_	UTF	是	postgres=# INSERT INTO table1	id | a | b| c --+--+--+---	字符串类型的字段长度是客户端

	ASCII	8		VALUES(3,'87A 123','87A 123','87A 123');	3 | 87A1 | 87A1 | 87A1	字符编码长度的整数倍，所以截断后产生结果正常。
4	SQL_ASCII	UTF8	否	postgres=# INSERT INTO table2 VALUES(1,reverse('123AA 78'),reverse('123AA 78'),reverse('123AA 78')); postgres=# INSERT INTO table2 VALUES(2,reverse('123A 78'),reverse('123A78'),reverse('123A 78'));	id |a|b|c ---+-+--+-- 1 |87|321| 87321 | 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 3.6 创建和管理表。

步骤 1 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。

```
postgres=# 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。例如以下命令上面的命令效果相同。

```
postgres=# 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：

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

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

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

步骤 2 如果需要在表中插入多行，请使用以下命令：

```
postgres=# 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。

步骤 3 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。

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

步骤 4 删除备份表。

```
postgres=# 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。

3.7.3 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 3.6.1 创建表和 3.6.2 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：

```
postgres=# 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：

```
postgres=# 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：

```
postgres=# 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。

3.7.4 查看数据

- 使用系统表 pg_tables 查询数据库所有表的信息。

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

- 使用 gsql 的\d+命令查询表的属性。

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

- 执行如下命令查询表 customer_t1 的数据量。

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

- 执行如下命令查询表 customer_t1 的所有数据。

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

- 执行如下命令只查询字段 c_customer_sk 的数据。

```
postgres=# 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。

```
postgres=# 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。

```
postgres=# 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。

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

3.7.5 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，用户可以指定准确行。用户可以删除匹配条件的一组行，或一次删除表中的所有行。

- 使用 DELETE 命令删除行。如删除表 customer_t1 中所有 c_customer_sk 为 3869 的记录：

```
postgres=# 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。

```
postgres=# DELETE  FROM customer_t1;
```

或

```
postgres=# 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：

```
postgres=# DROP TABLE customer_t1;
```

3.8 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 GBase 8c 安装信息以及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。

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。

GBase 8c 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。
- 新增的系统表和视图。这类系统表和视图具有 GS 前缀。

3.8.1 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。

```
postgres=#  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)
```

3.8.2 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。

```
postgres=# SELECT * FROM pg_user;
usename | usesysid | usecreatedb | usesuper | usecatupd | userepl |  passwd  |
valbegin | valuntil |   respool    | parent | spacelimit | useconfig | nodegroup
 | tempspacelimit | spillspacelimit | usemonitoradmin | useoperatoradmin | usepo
licyadmin
---------+----------+-------------+----------+-----------+---------+----------
-+-
```

```
---------+----------+--------------+--------+------------+-----------+-------
---
-+----------------+-----------------+-----------------+------------------+---
---
----------
 gbase   |       10 | t            | t         | t          | t        | ******** |
         |          | default_pool |       0 |             |            |
 |                |                 | t                | t                 | t
 jack    |    16549 | f            | f         | f          | f        | ******** |
         |          | default_pool |       0 |             |            |
 |                |                 | f                | f                 | f
(2 rows)
```

3.8.3 查看和停止正在运行的查询语句

通过视图 20.3.72 PG_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
----------+---------+--------+-----------------
postgres | Ruby | active | 140298793514752
postgres | Ruby | active | 140298718004992
postgres | Ruby | idle | 140298650908416
postgres | Ruby | idle | 140298625742592
postgres | gbase | 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

gsql 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.
```

----结束

3.9 其他操作

3.9.1 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

- GBase 8c 包含一个或多个已命名数据库。用户和用户组在 GBase 8c 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

步骤 1 创建管理用户及权限 schema

- 执行如下命令来创建一个 schema。

```
postgres=# 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。

```
postgres=# CREATE SCHEMA myschema AUTHORIZATION  gbase;
```

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

```
CREATE SCHEMA
```

步骤 2 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。

```
postgres=# 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。

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

步骤 3 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。

```
postgres=# 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。

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

步骤 4 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，则该 schema 的所有者应该赋予他对该 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。

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

- 使用以下命令查看现有的 schema：

```
postgres=# 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。

```
postgres=# CREATE USER jack IDENTIFIED BY 'xxxxxxxxx';
CREATE ROLE
postgres=# GRANT USAGE ON schema myschema TO jack;
GRANT
```

- 将用户 jack 对于 myschema 的 usage 权限收回。

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

步骤 5 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。

```
postgres=# 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。

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

- 执行如下命令删除用户 jack。

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

3.9.2 创建和管理分区表

背景信息

GBase 8c 数据库支持的分区表为范围分区表、间隔分区表、列表分区表、哈希分区表。

- 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。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。
- 间隔分区表：是一种特殊的范围分区表，相比范围分区表，新增间隔值定义，当插入记录找不到匹配的分区时，可以根据间隔值自动创建分区。
- 列表分区表：将数据中包含的键值分别存储再不同的分区中，依次将数据映射到每一个分区，分区中包含的键值由创建分区表时指定。
- 哈希分区表：将数据根据内部哈希算法依次映射到每一个分区中，包含的分区个数由创建分区表时指定。

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。

注意

支持自动分区创建的方式为每个 DN 节点本地进行，用以解决用户执行 DML 操作的时候，同时执行自动分区操作时会卡住集群问题。

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。

操作步骤

创建和管理表操作步骤如下，以使用默认表空间为例。

步骤 1 创建分区表（假设用户已创建 tpcds schema）

```
postgres=# CREATE TABLE  tpcds.customer_address (ca_address_sk integer NOT
NULL , ca_address_id character(16) NOT NULL , ca_country character varying(20),
ca_gmt_offset numeric(5,2), ca_location_type character(20), 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)) 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)) 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。

步骤 2 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。

```
postgres=# CREATE TABLE tpcds.web_returns_p2 (
ca_address_sk  integer NOT NULL, ca_address_id character(16),
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))
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)
)
ENABLE ROW MOVEMENT;
CREATE TABLE
```

```
postgres=# INSERT INTO tpcds.web_returns_p2 SELECT * FROM
tpcds.customer_address;
INSERT 0 0
```

步骤 3 修改分区表行迁移属性

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

步骤 4 删除分区

删除分区 P8。

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

步骤 5 增加分区

增加分区 P8，范围为 40000<= P8<=MAXVALUE。

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

步骤 6 重命名分区

重命名分区 P8 为 P_9。

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

重命名分区 P_9 为 P8。

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

步骤 7 查询分区

查询分区 P6。

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

步骤 8 删除分区表和表空间

```
postgres=# DROP TABLE tpcds.customer_address; DROP TABLE
postgres=# DROP TABLE tpcds.web_returns_p2; DROP TABLE
```

3.9.3 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。
  - 分区表索引分为 LOCAL 索引与 GLOBAL 索引，一个 LOCAL 索引对应一个具体分区，而 GLOBAL 索引则对应整个分区表。

操作步骤

创建分区表的步骤请参考 3.8.2 创建和管理表。

步骤 1 创建索引

创建分区表 LOCAL 索引 tpcds_web_returns_p2_index1，不指定索引分区的名称。

```
postgres=# 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
```

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

```
postgres=# 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
```

- 创建分区表 GLOBAL 索引 tpcds_web_returns_p2_global_index。

```
CREATE INDEX tpcds_web_returns_p2_global_index ON tpcds.web_returns_p2
(ca_street_number) GLOBAL;
```

步骤 2 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。

```
postgres=# 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。

```
postgres=# 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
```

步骤 3 重命名索引分区

执行如下命令对索引分区进行重命名操作。如将 web_returns_p2_P8_index 重命名为

web_returns_p2_P8_index_new。

```
postgres=# 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
```

步骤 4 查询索引

- 执行如下命令查询系统和用户定义的所有索引。

```
postgres=# SELECT RELNAME FROM PG_CLASS WHERE RELKIND='i' or RELKIND='I';
```

- 执行如下命令查询指定索引的信息。

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

步骤 5 删除索引

```
postgres=# DROP INDEX tpcds.tpcds_web_returns_p2_index1;
postgres=# 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
```

GBase 8c 支持 4 种创建索引的方式请参见表 3-4。

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。

表 3-5 索引方式

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 GBase 8c 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，GBase 8c 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，GBase 8c 中的 B-Tree 支持多字段索引，。

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。

步骤 6 创建一个普通表

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

步骤 7 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。

```
postgres=# 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。

```
postgres=# CREATE INDEX index_wr_returned_date_sk ON pcds.customer_address_bak
(ca_address_sk);
CREATE INDEX
```

使用以下命令创建索引。

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

CREATE INDEX

步骤 8 创建唯一索引

在表 tpcds.ship_mode_t1 上的 SM_SHIP_MODE_SK 字段上创建唯一索引。

```
postgres=# CREATE UNIQUE INDEX ds_ship_mode_t1_index1 ON
tpcds.ship_mode_t1(SM_SHIP_MODE_SK);
```

步骤 9 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。

```
postgres=# 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。

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

步骤 10 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。

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

步骤 11 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。

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

可以为上面的查询创建表达式索引：

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

步骤 12 删除 tpcds.customer_address_bak 表

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

3.9.4 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。

```
postgres=# CREATE OR REPLACE VIEW MyView AS SELECT * FROM tpcds.web_returns WHERE
trunc(wr_refunded_cash) > 10000;
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。

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

- 查看某视图的具体信息

  执行如下命令查询 MyView 视图的详细信息。

```
postgres=# \d+ MyView
View "PG_CATALOG.MyView"
Column |   Type    | Modifiers | Storage | Description
----------+----------------------+-----------+---------+-------------
USERNAME  | CHARACTER VARYING(64) |     | extended | View definition:
SELECT PG_AUTHID.ROLNAME::CHARACTER VARYING(64) AS USERNAME FROM PG_AUTHID;
```

- 删除视图

  执行如下命令删除 MyView 视图。

```
postgres=# DROP VIEW MyView;
DROP VIEW
```

3.9.5 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序列整型，由数据库在后台自动创建一个对应的 Sequence。
- 另一种是使用 CREATE 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：

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

当结果显示为如下信息，则表示创建成功。

```
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

```
postgres=# 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。

```
postgres=# 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。

```
postgres=# 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。

3.9.6 管理插件

3.9.6.1 dblink

通过 dblink 工具可以在 8c 数据库会话中连接其他数据库，如 opengauss、或者以 opengauss 为生态的其他数据库，从而达到访问其他数据库的目的。当前 dblink 的源码放在 contrib/dblink 目录，在安装完 8c 数据库主备式之后，如果使用 dblink，需要进入 contrib/dblink 目录编译并安装。主要包括加载 dblink 扩展、打开到远程数据库的持久连接、关闭到远程数据库的持久连接、在远程数据库执行 SQL 查询、返回所有打开的 dblink 连接名称、发送异步查询到远程数据库等功能。

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 GBase 8c 的应用程序。

3.9.6.2 pg_trgm

pg_trgm 模块提供函数和操作符测定字母数字文本基于三元模型匹配的相似性， 还有支持快速搜索相似字符串的索引操作符类。pg_trgm 模块需支持 similarity(text, text)、show_trgm(text)、show_limit()、set_limit(real)等函数；需支持 text % text、text <-> text 操作符；需支持 Gist 和 GIN 索引操作符类。

3.9.6.3 oracle_fdw

支持 Oracle 外部表功能，可通过 oracle_fdw 插件直接访问 Oracle 数据库，并使用 SQL 完成 8c 数据库与 Oracle 数据库之间的数据交互。

3.9.6.4 mysql_fdw

支持 MySQL 外部表功能，可通过 mysql_fdw 插件直接访问 MySQL 数据库，并使用 SQL 完成 8c 数据库与 MySQL 数据库之间的数据交互。

3.9.6.5 postgres_fdw

支持 PostgreSQL 外部表功能，可通过 postgres_fdw 插件直接访问 PostgreSQL 数据库，并使用 SQL 完成 8c 数据库与 PostgreSQL 数据库之间的数据交互。

3.9.6.6 zhparser

实现数据库 TEXT SEARCH PARSER 需要的接口，并调用 SCWS 中文分词引擎，从而支持中文分词，从而使 8c 数据库分布式集群模式支持中文分词和全文检索。

3.9.6.7 sqldecoding

实现 8c 的逻辑日志输出为 SQL 语句格式，便于异构数据库获取到增量数据后直接执行同步使用。

3.9.6.8 uuid-ossp

用于储存全局唯一标识符 UUID。

支持 uuid_generate_v1()、uuid_generate_v1mc()、uuid_generate_v4()等 uuid 生成函数。

支持使用 uuid_nil()、uuid_ns_dns()、uuid_ns_url()、uuid_ns_oid()、uuid_ns_x500()等产生 uuid 常量函数。

3.9.6.9 pgrouting

支持使用数据库中的路网数据构建拓扑，并基于拓扑路网数据进行最短路径查询或耗费成本计算，支持矩阵运算。

支持使用 pgrouting 函数做数据查询和最短路径分析。

3.9.6.10 pgcrypto

pgcrypto 插件提供的加密函数能够实现服务器端的数据加密，以便在 SQL 语句中调用这些函数来完成数据的加密。

3.9.6.11 logical to kafka

logical to kafka 插件工具支持分布式数据库各个 DN 主节点的逻辑日志文件流式写入 kafka 集群，以便把增量变化的数据操作实时同步到其他数据库。

4 开发设计建议

4.1 开发设计建议概述

本开发设计建议约定数据库建模和数据库应用程序开发过程中，应当遵守的设计规范。依据这些规范进行建模，能够更好地契合 GBase 8c 分布式处理架构，输出更高效的业务 SQL 代码。

本开发设计建议中所陈述的“建议”和“关注”含义如下：

- **建议**：用户应当遵守的设计规则。遵守这些规则，能够保证业务的高效运行；违反这些规则，将导致业务性能的大幅下降或某些业务逻辑错误。
- **关注**：在业务开发过程中客户需要注意的细则。用于标识容易导致客户理解错误的知识点（实际上遵守 SQL 标准的 SQL 行为），或者程序中潜在的客户不易感知的默认行为。

4.2 数据库对象命名

数据库对象命名需要满足约束：非时序表长度不超过 63 个字节，时序表长度不超过 53 个字符，以字母或下划线开头，中间字符可以是字母、数字、下划线、$、#。

- 【建议】避免使用保留或者非保留关键字命名数据库对象。

  说明

  可以使用 select * from pg_get_keywords()查询数据库关键字，或者在《GBase 8c V5_3.0.1_SQL 手册》关键字章节中查看。

- 【建议】避免使用双引号括起来的字符串来定义数据库对象名称，除非需要限制数据库对象名称的大小写。数据库对象名称大小写敏感会使定位问题难度增加。
- 【建议】数据库对象命名风格务必保持统一。
  - 增量开发的业务系统或进行业务迁移的系统，建议遵守历史的命名风格。
  - 建议使用多个单词组成，以下划线分割。
  - 数据库对象名称建议能够望文知意，尽量避免使用自定义缩写（可以使用通用的术语缩写进行命名）。例如，在命名中可以使用具有实际业务含义的英文词汇或汉语拼音，但规则应该在数据库实例范围内保持一致。
  - 变量名的关键是要具有描述性，即变量名称要有一定的意义，变量名要有前缀标明该变量的类型。
- 【建议】表对象的命名应该可以表征该表的重要特征。例如，在表对象命名时区分该表是普通表、临时表还是非日志表：
  - 普通表名按照数据集的业务含义命名。
  - 临时表以“tmp_+后缀”命名。
  - 非日志表以“ul_+后缀”命名。

- 外表以“f_+后缀”命名。
- 不创建以 redis_为前缀的数据库对象。
- 不创建以 mlog_和以 matviewmap_为前缀的数据库对象。

- 【建议】非时序表对象命名建议不要超过 63 字节。如果过该长度内核会对表名进行截断，从而造成和设置值不一致的现象。且在不同字符集下，可能造成字符被截断，出现预期外的字符。

4.3 数据库对象设计

4.3.1 Database 和 Schema 设计

GBase 8c 中可以使用 Database 和 Schema 实现业务的隔离，区别在于 Database 的隔离更加彻底，各个 Database 之间共享资源极少，可实现连接隔离、权限隔离等， Database 之间无法直接互访。Schema 隔离的方式共用资源较多，可以通过 grant 与 revoke 语法便捷地控制不同用户对各 Schema 及其下属对象的权限。

- 从便捷性和资源共享效率上考虑，推荐使用 Schema 进行业务隔离。
- 建议系统管理员创建 Schema 和 Database，再赋予相关用户对应的权限。

Database 设计建议

- 【规则】在实际业务中，根据需要创建新的 Database，不建议直接使用数据库实例默认的 postgres 数据库。
- 【建议】一个数据库实例内，用户自定义的 Database 数量建议不超过 3 个。
- 【建议】为了适应全球化的需求，使数据库编码能够存储与表示绝大多数的字符，建议创建 Database 的时候使用 UTF-8 编码。
- 【关注】创建 Database 时，需要重点关注字符集编码(ENCODING)和兼容性(DBCOMPATIBILITY)两个配置项。GBase 8c 支持 A、B、C 和 PG 四种兼容模式，分别表示兼容 O 语法、MY 语法、TD 语法和 POSTGRES 语法，不同兼容模式下的语法行为存在一定差异，默认为 A 兼容模式。
- 【关注】Database 的 owner 默认拥有该 Database 下所有对象的所有权限，包括删除权限。删除权限影响较大，请谨慎使用。

Schema 设计建议

- 【关注】如果该用户不具有 sysadmin 权限或者不是该 Schema 的 owner，要访问 Schema 下的对象，需要同时给用户赋予 Schema 的 usage 权限和对象的相应权 限。
- 【关注】如果要在 Schema 下创建对象，需要授予操作用户该 Schema 的 create 权限。
- 【关注】Schema 的 owner 默认拥有该 Schema 下对象的所有权限，包括删除权限。删除权限影响较大，请谨慎使用。

4.3.2 表设计

总体上讲，良好的表设计需要遵循以下原则：

- 【关注】减少需要扫描的数据量。通过分区表的剪枝机制可以大幅减少数据的扫描量。
- 【关注】尽量减少随机 I/O。通过聚簇/局部聚簇可以实现热数据的连续存储，将 随机 I/O 转换为连续 I/O，从而减少扫描的 I/O 代价。

选择存储方案

【建议】表的存储类型是表定义设计的第一步，客户业务类型是决定表的存储类型的主要因素，表存储类型的选择依据请参考表 4-1。

表 4-1 表的存储类型及场景

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。
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）。

选择分区方案

当表中的数据量很大时，应当对表进行分区，一般需要遵循以下原则：

- 【建议】使用具有明显区间性的字段进行分区，比如日期、区域等字段上建立分区。
- 【建议】分区名称应当体现分区的数据特征。例如，关键字+区间特征。
- 【建议】将分区上边界的分区值定义为 MAXVALUE，以防止可能出现的数据溢出。

表 4-2 表的分区方式及使用场景

分区方式	描述
Range	表数据通过范围进行分区。
Interval	表数据通过范围进行分区，超出范围的会自动根据间隔创建新的分区。
List	表数据通过指定列按照具体值进行分区。
Hash	表数据通过 Hash 散列方式进行分区。

典型的分区表定义如下：

```
--创建 Range 分区表
CREATE TABLE staffS_p1 (
staff_ID   NUMBER(6) not null, FIRST_NAME          VARCHAR2(20), LAST_NAME
           VARCHAR2(25), EMAIL     VARCHAR2(25), PHONE_NUMBER VARCHAR2(20),
HIRE_DATE          DATE,
employment_ID VARCHAR2(10), SALARY NUMBER(8,2), COMMISSION_PCT NUMBER(4,2),
MANAGER_ID NUMBER(6),
section_ID NUMBER(4)
)
PARTITION BY RANGE (HIRE_DATE) (
PARTITION HIRE_19950501 VALUES LESS THAN ('1995-05-01 00:00:00'), PARTITION
HIRE_19950502 VALUES LESS THAN ('1995-05-02 00:00:00'), PARTITION HIRE_maxvalue
VALUES LESS THAN (MAXVALUE)
);
--创建 Interval 分区表，初始两个分区，插入分区范围外的数据会自动新增分区 CREATE
TABLE sales
(prod_id NUMBER(6), cust_id NUMBER, time_id DATE, channel_id CHAR(1), promo_id
NUMBER(6),
quantity_sold NUMBER(3), amount_sold NUMBER(10,2)
)
PARTITION BY RANGE (time_id) INTERVAL('1 day')
( PARTITION p1 VALUES LESS THAN ('2019-02-01 00:00:00'), PARTITION p2 VALUES LESS
THAN ('2019-02-02 00:00:00')
);
--创建 List 分区表
CREATE TABLE test_list (col1 int, col2 int) partition by list(col1)
(
partition p1 values (2000), partition p2 values (3000), partition p3 values (4000),
partition p4 values (5000)
```

```
);
--创建 Hash 分区表
CREATE TABLE test_hash (col1 int, col2 int) partition by hash(col1)
(
partition p1, partition p2
);
```

更多的表分区语法信息，参见《GBase 8c V5_3.0.1_SQL 手册》中 CREATE TABLE PARTITION 章节。

4.3.3 字段设计

选择数据类型

在字段设计时，基于查询效率的考虑，一般遵循以下原则：

- 【建议】尽量使用高效数据类型。
- 选择数值类型时，在满足业务精度的情况下，选择数据类型的优先级从高到低依次为整数、浮点数、NUMERIC。
- 【建议】当多个表存在逻辑关系时，表示同一含义的字段应该使用相同的数据类型。
- 【建议】对于字符串数据，建议使用变长字符串数据类型，并指定最大长度。请务必确保指定的最大长度大于需要存储的最大字符数，避免超出最大长度时出现字符截断现象。除非明确知道数据类型为固定长度字符串，否则，不建议使用 CHAR(n)、BPCHAR(n)、NCHAR(n)、CHARACTER(n)。

常用字符串类型介绍

在进行字段设计时，需要根据数据特征选择相应的数据类型。字符串类型在使用时比较容易混淆，下表列出了支持的常见字符串类型：

表 4-3 常用字符串类型

名称	描述	最大存储空间
CHAR(n)	定长字符串，n 描述了存储的字节长度，如果输入的字符串字节格式小于 n，那么后面会自动用空字符补齐至 n 个字节。	10MB
CHARACTER(n)	定长字符串，n 描述了存储的字节长度，如果输入	10MB

	的字符串字节格式小于 n，那么后面会自动用空字符补齐至 n 个字节。	
NCHAR(n)	定长字符串，n 描述了存储的字节长度，如果输入的字符串字节格式小于 n，那么后面会自动用空字符补齐至 n 个字节。	10MB
BPCHAR(n)	定长字符串，n 描述了存储的字节长度，如果输入的字符串字节格式小于 n，那么后面会自动用空字符补齐至 n 个字节。	10MB
VARCHAR(n)	变长字符串，n 描述了可以存储的最大字节长度。	10MB
CHARACTER VARYING(n)	变长字符串，n 描述了可以存储的最大字节长度；此数据类型和 VARCHAR(n)是同一数据类型的不同表达形式。	10MB
VARCHAR2(n)	变长字符串，n 描述了可以存储的最大字节长度，此数据类型是为兼容 Oracle 类型新增的，行为和 VARCHAR(n)一致。	10MB
NVARCHAR2(n)	变长字符串，n 描述了可以存储的最大字节长度。	10MB
TEXT	不限长度(不超过 1GB-8203 字节)变长字符串。	1GB-8203 字节

4.3.4 约束设计

DEFAULT 和 NULL 约束

- 【建议】如果能够从业务层面补全字段值，那么，就不建议使用 DEFAULT 约束， 避免数据加载时产生不符合预期的结果。
- 【建议】给明确不存在 NULL 值的字段加上 NOT NULL 约束，优化器会在特定场景下对其进行自动优化。
- 【建议】给可以显式命名的约束显式命名。除了 NOT NULL 和 DEFAULT 约束外， 其他约束都可以显式命名。

局部聚簇

Partial Cluster Key（局部聚簇，简称 PCK）是列存表的一种局部聚簇技术，在 GBase 8c 中，使用 PCK 可以通过 min/max 稀疏索引实现事实表快速过滤扫描。PCK 的选取遵循以下原则：

- 【关注】一张表上只能建立一个 PCK，一个 PCK 可以包含多列，但是一般不建议超过 2 列。
- 【建议】在查询中的简单表达式过滤条件上创建 PCK。这种过滤条件一般形如 col op const，其中 col 为列名，op 为操作符=、>、>=、<=、<，const 为常量值。
- 【建议】在满足上面条件的前提下，选择 distinct 值比较多的列上建 PCK。

唯一约束

- 【关注】行存表、列存表均支持唯一约束。
- 【建议】从命名上明确标识唯一约束，例如，命名为“UNI+构成字段”。

主键约束

- 【关注】行存表、列存表均支持主键约束。
- 【建议】从命名上明确标识主键约束，例如，将主键约束命名为“PK+字段名”。

检查约束

- 【关注】行存表支持检查约束，而列存表不支持。
- 【建议】从命名上明确标识检查约束，例如，将检查约束命名为“CK+字段名”。

4.3.5 视图和关联表设计

视图设计

- 【建议】除非视图之间存在强依赖关系，否则不建议视图嵌套。
- 【建议】视图定义中尽量避免排序操作。

关联表设计

- 【建议】表之间的关联字段应该尽量少。
- 【建议】关联字段的数据类型应该保持一致。
- 【建议】关联字段在命名上，应该可以明显体现出关联关系。例如，采用同样名称来命名。

4.4 工具对接

4.4.1 JDBC 配置

目前，GBase 8c 相关的第三方工具都是通过 JDBC 进行连接的，此部分将介绍工具配置时的注意事项。

连接参数

- 【关注】第三方工具通过 JDBC 连接 GBase 8c 时，JDBC 向 GBase 8c 发起连接请求，会默认添加以下配置参数，详见 JDBC 代码 ConnectionFactoryImpl 类的实现。

```
params = {
{ "user", user },
{ "database", database },
{ "client_encoding", "UTF8" },
{ "DateStyle", "ISO" },
{ "extra_float_digits", "3" },
{ "TimeZone", createPostgresTimeZone() },
};
```

这些参数可能会导致 JDBC 客户端的行为与 gsql 客户端的行为不一致，例如，Date 数据显示方式、浮点数精度表示、timezone 显示。

如果实际期望和这些配置不符，建议在 java 连接设置代码中显式设定这些参数。

- 【建议】通过 JDBC 连接数据库时，应该保证下面三个时区设置一致：
  - JDBC 客户端所在主机的时区。
  - 数据库实例所在主机的时区。
  - 数据库实例配置过程中时区。

  说明

  时区设置相关的操作，请参考《安装指南》中“初始配置 > 同步系统时间”章节内容。

fetchsize

- 【关注】在应用程序中，如果需要使用 fetchsize，必须关闭 autocommit。开启 autocommit，会令 fetchsize 配置失效。

autocommit

- 【建议】在 JDBC 向 GBase 8c 申请连接的代码中，建议显式打开 autocommit 开关。如

果基于性能或者其它方面考虑，需要关闭 autocommit 时，需要应用程序自己来保证事务的提交。例如，在指定的业务 SQL 执行完之后做显式提交，特别是客户端退出之前 务必保证所有的事务已经提交。

释放连接

- 【建议】推荐使用连接池限制应用程序的连接数。每执行一条 SQL 就连接一次数据库，是一种不好 SQL 的编写习惯。
- 【建议】在应用程序完成作业任务之后，应当及时断开和 GBase 8c 的连接，释放资源。建议在任务中设置 session 超时时间参数。
- 【建议】使用 JDBC 连接池，在将连接释放给连接池前，需要执行以下操作，重置会话环境。否则，可能会因为历史会话信息导致的对象冲突。

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。

CopyManager

- 【建议】在不使用 ETL 工具，数据入库实时性要求又比较高的情况下，建议在开发应用程序时，使用 GBase 8c JDBC 驱动的 copyManger 接口进行微批导入。

4.5 SQL 编写

DDL

- 【建议】在 GBase 8c 中，建议 DDL（建表、comments 等）操作统一执行，在批处理作业中尽量避免 DDL 操作。避免大量并发事务对性能的影响。
- 【建议】在非日志表（unlogged table）使用完后，立即执行数据清理（truncate）操作。因为在异常场景下，GBase 8c 不保证非日志表(unlogged table)数据的安全性。
- 【建议】临时表和非日志表的存储方式建议和基表相同。当基表为行存（列存） 表时，临时表和非日志表也推荐创建为行存（列存）表，可以避免行列混合关联带来的高计算代价。
- 【建议】索引字段的总长度不超过 50 字节。否则，索引大小会膨胀比较严重，带来较大的存储开销，同时索引性能也会下降。
- 【建议】不要使用 DROP…CASCADE 方式删除对象，除非已经明确对象间的依赖关系，

以免误删。

数据加载和卸载

- 【建议】在 insert 语句中显式给出插入的字段列表。例如：

```
INSERT INTO task(name,id,comment) VALUES ('task1','100','第 100 个任务');
```

- 【建议】在批量数据入库之后，或者数据增量达到一定阈值后，建议对表进行 analyze 操作，防止统计信息不准确而导致的执行计划劣化。
- 【建议】如果要清理表中的所有数据，建议使用 truncate table 方式，不要使用 delete table 方式。delete table 方式删除性能差，且不会释放那些已经删除了的数据占用的磁盘空间。

类型转换

- 【建议】在需要数据类型转换（不同数据类型进行比较或转换）时，使用强制类型转换，以防隐式类型转换结果与预期不符。
- 【建议】在查询中，对常量要显式指定数据类型，不要试图依赖任何隐式的数据类型转换。
- 【关注】若 sql_compatibility 参数设置为 A，在导入数据时，空字符串会自动转化为 NULL。如果需要保留空字符串需要 sql_compatibility 参数设置为 C。

查询操作

- 【建议】除 ETL 程序外，应该尽量避免向客户端返回大量结果集的操作。如果结果集过大，应考虑业务设计是否合理。
- 【建议】使用事务方式执行 DDL 和 DML 操作。例如，truncate table、update table、delete table、drop table 等操作，一旦执行提交就无法恢复。对于这类操作，建议使用事务进行封装，必要时可以进行回滚。
- 【建议】在查询编写时，建议明确列出查询涉及的所有字段，不建议使用“SELECT *”这种写法。一方面基于性能考虑，尽量减少查询输出列；另一方面避免增删字段对前端业务兼容性的影响。
- 【建议】在访问表对象时带上 schema 前缀，可以避免因 schema 切换导致访问到非预期的表。
- 【建议】超过 3 张表或视图进行关联（特别是 full join）时，执行代价难以估算。建议使用 WITH TABLE AS 语句创建中间临时表的方式增加 SQL 语句的可读性。
- 【建议】尽量避免使用笛卡尔积和 Full join。这些操作会造成结果集的急剧膨胀， 同

时其执行性能也很低。

- 【关注】NULL 值的比较只能使用 IS NULL 或者 IS NOT NULL 的方式判断，其他任何形式的逻辑判断都返回 NULL。例如：NULL<>NULL、NULL=NULL 和 NULL<>1 返回结果都是 NULL，而不是期望的布尔值。
- 【关注】需要统计表中所有记录数时，不要使用 count(col)来替代 count(*)。count(*)会统计 NULL 值（真实行数），而 count(col)不会统计。
- 【关注】在执行 count(col)时，将“值为 NULL”的记录行计数为 0。在执行 sum(col)时，当所有记录都为 NULL 时，最终将返回 NULL；当不全为 NULL 时，“值为 NULL”的记录行将被计数为 0。
- 【关注】count(多个字段)时，多个字段名必须用圆括号括起来。例如，count((col1,col2,col3))。注意：通过多字段统计行数时，即使所选字段都为 NULL，该行也被计数，效果与 count(*)一致。
- 【关注】count(distinct col)用来计算该列不重复的非 NULL 的数量，NULL 将不被计数。
- 【关注】count(distinct (col1,col2,...))用来统计多列的唯一值数量，当所有统计字段都为 NULL 时，也会被计数，同时这些记录被认为是相同的。
- 【建议】使用连接操作符“||”替换 concat 函数进行字符串连接。因为 concat 函数生成的执行计划不能下推，导致查询性能严重劣化。
- 【建议】使用下面时间相关的宏替换 now 函数来获取当前时间。因为 now 函数生成的执行计划无法下推，导致查询性能严重劣化。

表 4-4 时间相关的宏

宏名称	描述	示例
CURRENT_DATE	获取当前日期，不包含时分秒。	postgres=# select CURRENT_DATE; date ------------ 2018-02-02 (1 row)
CURRENT_TIME	获取当前时间，不包含年月日。	postgres=# select CURRENT_TIME; timetz -------------------- 00:39:34.633938+08 (1 row)

CURRENT_TIMESTAMP(n)	获取当前日期和时间，包含年月日时分秒。说明 n 表示存储的毫秒位数。	postgres=# select CURRENT_TIMESTAMP(6); timestamptz ---------- 2018-02-02 00:39:55.231689+08 (1 row)

- 【建议】尽量避免标量子查询语句的出现。标量子查询是出现在 select 语句输出列 表中的子查询，在下面例子中，下划线部分即为一个标量子查询语句：

```
SELECT id, (SELECT COUNT(*) FROM films f WHERE f.did = s.id) FROM staffs_p1 s;
```

  标量子查询往往会导致查询性能急剧劣化，在应用开发过程中，应当根据业务逻辑，对标量子查询进行等价转换，将其写为表关联。

- 【建议】在 where 子句中，应当对过滤条件进行排序，把选择度较大（筛选出的记录数较少）的条件排在前面。

- 【建议】where 子句中的过滤条件，尽量符合单边规则。即把字段名放在比较条件的一边，优化器在某些场景下会自动进行剪枝优化。形如 col op expression， 其中 col 为表的一个列，op 为‘=’、‘>’的等比较操作符，expression 为不含列名的表达式。例如，

```
SELECT id, from_image_id, from_person_id, from_video_id FROM face_data WHERE current_timestamp(6) - time < '1 days'::interval;
```

  改写为：

```
SELECT id, from_image_id, from_person_id, from_video_id FROM face_data where time > current_timestamp(6) - '1 days'::interval;
```

- 【建议】尽量避免不必要的排序操作。排序需要耗费大量的内存及 CPU，如果业务逻辑许可，可以组合使用 order by 和 limit，减小资源开销。gbase 默认按照 ASC & NULL LAST 进行排序。

- 【建议】使用 ORDER BY 子句进行排序时，显式指定排序方式（ASC/DESC）， NULL 的排序方式（NULL FIRST/NULL LAST）。

- 【建议】不要单独依赖 limit 子句返回特定顺序的结果集。如果部分特定结果集， 可以将 ORDER BY 子句与 Limit 子句组合使用，必要时也可以使用 offset 跳过特定结果。

- 【建议】在保障业务逻辑准确的情况下，建议尽量使用 UNION ALL 来代替 UNION。

- 【建议】如果过滤条件只有 OR 表达式，可以将 OR 表达式转化为 UNION ALL 以提升性能。使用 OR 的 SQL 语句经常无法优化，导致执行速度慢。例如，将下面语句

```
SELECT * FROM scdc.pub_menu
WHERE (cdp= 300 AND inline=301) OR (cdp= 301 AND inline=302) OR (cdp= 302
ANDinline=301);
```

转换为：

```
SELECT * FROM scdc.pub_menu WHERE (cdp= 300 AND inline=301) union all
SELECT * FROM scdc.pub_menu WHERE (cdp= 301 AND inline=302) union all
SELECT * FROM tablename
WHERE (cdp= 302 AND inline=301)
```

- 【建议】当 in(val1, val2, val3…)表达式中字段较多时，建议使用 in (values (va11), (val2),(val3)…)语句进行替换。优化器会自动把 in 约束转换为非关联子查询，从而提升查询性能。
- 【建议】在关联字段不存在 NULL 值的情况下，使用(not) exist 代替(not) in。例如，在下面查询语句中，当 T1.C1 列不存在 NULL 值时，可以先为 T1.C1 字段添加 NOT NULL 约束，再进行如下改写。

```
SELECT * FROM T1 WHERE T1.C1 NOT IN (SELECT T2.C2 FROM T2);
```

可以改写为：

```
SELECT * FROM T1 WHERE NOT EXISTS (SELECT * FROM T1,T2 WHERE T1.C1=T2.C2);
```

说明

- 如果不能保证 T1.C1 列的值为 NOT NULL 的情况下，就不能进行上述改写。
- 如果 T1.C1 为子查询的输出，要根据业务逻辑确认其输出是否为 NOT NULL。

- 【建议】通过游标进行翻页查询，而不是使用 LIMIT OFFSET 语法，避免多次执行带来的资源开销。游标必须在事务中使用，执行完后务必关闭游标并提交事务。

5 最佳实践

5.1 表设计最佳实践

5.1.1 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）。

5.1.2 使用局部聚簇

局部聚簇（Partial Cluster Key）是列存下的一种技术。这种技术可以通过 min/max 稀疏索引较快的实现基表扫描的 filter 过滤。Partial Cluster Key 可以指定多列，但是一般不建议超过 2 列。Partial Cluster Key 的选取原则：

(1) 受基表中的简单表达式约束。这种约束一般形如 col op const，其中 col 为列名， op 为操作符 =、>、>=、<=、<，const 为常量值。

(2) 尽量采用选择度比较高(过滤掉更多数据)的简单表达式中的列。

(3) 尽量把选择度比较高的约束 col 放在 Partial Cluster Key 中的前面。

(4) 尽量把枚举类型的列放在 Partial Cluster Key 中的前面。

5.1.3 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：

(1) 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。

(2) 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。

(3) 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。

GBase 8c 数据库支持的分区表为范围分区表、间隔分区表、列表分区表、哈希分区表。

- 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。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。
- 间隔分区表：是一种特殊的范围分区表，相比范围分区表，新增间隔值定义，当插入记录找不到匹配的分区时，可以根据间隔值自动创建分区。

- 列表分区表：将数据中包含的键值分别存储再不同的分区中，依次将数据映射到每一个分区，分区中包含的键值由创建分区表时指定。
- 哈希分区表：将数据根据内部哈希算法依次映射到每一个分区中，包含的分区个数由创建分区表时指定。
- 二级分区表：二级分区表是在一级分区的基础上再进行分区，分区方案是由两个一级分区的分区方案组合而来的，目前二级分区表支持范围分区、列表分区、哈希分区交叉组合的 9 种分区策略。

5.1.4 选择数据类型

高效数据类型，主要包括以下三方面：

(1) 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。

(2) 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。

(3) 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。

5.2 导入最佳实践

使用 COPY 命令导入数据

COPY 命令从本地或其它数据库的多个数据源并行导入数据。COPY 导入大量数据的效率要比 INSERT 语句高很多，而且存储数据也更有效率。

有关如何使用 COPY 命令的更多信息，请参阅 9.2 使用 COPY FROM STDIN 导入数据。

使用 gsql 元命令导入数据

\copy 命令在任何 psql 客户端登录数据库成功后可以执行导入数据。与 COPY 命令相比

较，\copy 命令不是读取或写入指定文件的服务器，而是直接读取或写入文件。

这个操作不如 COPY 命令有效，因为所有的数据必须通过客户端/服务器的连接来传递。对于大量的数据来说 COPY 命令可能会更好。

有关如何使用\copy 命令的更多信息，请参阅 9.3 使用 gsql 元命令导入数据。

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。

使用 INSERT 多行插入

插入如果不能使用 copy 命令，而您需要进行 sql 插入，可以根据情况使用多行插入。如果您使用的是列存表，一次只插入一行或几行，则数据压缩效率低下。

多行插入是通过批量进行一系列插入而提高性能。下面的示例使用一条 insert 语句向一个三列表插入三行。这仍属于少量插入，只是用来说明多行插入的语法。创建表的步骤请参考 3.6-创建和管理表。

向表 customer_t1 中插入多行数据：

```
postgres=# insert into customer_t1 values (68, 'a1', 'zhou','wang'),
(43, 'b1', 'wu', 'zhao'),
(95, 'c1', 'zheng', 'qian');
```

有关更多详情和示例，请参阅《GBase 8c V5_3.0.1_SQL 手册》 INSERT。

使用 INSERT 批量插入

带 SELECT 子句使用批量插入操作来实现高性能数据插入。

如果需要将数据或数据子集从一个表移动到另一个表，可以使用《GBase 8c V5_3.0.1_SQL 手册》 INSERT 和《GBase 8c V5_3.0.1_SQL 手册》CREATE TABLE AS 命令。

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。

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

```
);
postgres=# INSERT INTO customer_t2 SELECT * FROM customer_t1;
```

上面的示例等价于：

```
postgres=# CREATE TABLE customer_t2 AS SELECT * FROM customer_t1;
```

5.3 SQL 查询最佳实践

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。

- 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。

- 合理选择执行计划

  GBase 8c 支持 LightProxy、FQS、Stream、RemoteQuery 4 种分布式执行计划，并且采用 CBO（Cost-Based Optimization）基于成本的优化器来对同一条 SQL 语句的不同执行计划所产生的多种执行路径进行成本评估，从中选择执行成本最低执行计划进行执行。

  RemoteQuery 分布式计划，CN 需要从 DN 拉取相关数据，节点间通信开销较大，RemoteQuery 计划的优先级最低。当 remote query、Stream 执行计划均能满足 SQL 执行出正确结果时，执行器应优先选择 stream 执行计划。

  Remote Query 算子用来从数据节点上拉取业务数据，并和其他算子一起完成整个的执行计划。如对于包含关联操作的 SQL 语句，Remote Query 算子将从数据节点上拉取数据，最终在协调节点上完成关联操作。

  stream 支持扩展数据类型，如 gist 和其它非缺省数据类型。

  stream 支持外部表和 disable trigger 场景 insert。

  stream 执行计划支持单表 for update，在进行多表 for update 时，需要支持 Remote Query 执行方式。

stream 支持 with recursive 功能。with recursive 是 SQL 中递归查询的功能，它允许用户在一个查询中多次引用同一查询并不断地生成临时表，实现递归查询的功能。

对于 one time filter 计划，不需要添加 local stream 节点。

对于列存表的查询、插入操作，支持使用 stream 执行计划。

支持列存表使用 stream 执行计划的 SMP 功能。

针对 INSERT INTO table_name VALUES(RANDOM()[,values...]); 、 INSERT INTO table_name SELECT RANDOM()[,values...];这两种 insert 场景，优化器支持默认使用 Remote Query 执行计划。

支持 with recursive 语句默认使用 Remote Query 执行计划。

在涉及复杂查询场景时，使用并行查询，将增加查询效率，提高性能。

- 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.c1);
```

  产生的计划如下：

```
postgres=# explain select * from t1 where not exists(select * from t2 where t1.c1 = t2.c1);
QUERY PLAN
------------------------------------------------------------------
Hash Anti Join (cost=58.35..107.44 rows=1074 width=8) Hash Cond: (t1.c1 = t2.c1)
-> Seq Scan on t1 (cost=0.00..31.49 rows=2149 width=8)
-> Hash (cost=31.49..31.49 rows=2149 width=4)
-> Seq Scan on t2 (cost=0.00..31.49 rows=2149 width=4) (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。

  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。
  - 多 DN 环境下对大表做 broadcast 的子查询。其他更多调优点，请参考 14.4.6 典型 SQL 调优点。

6 应用程序开发教程

6.1 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。

兼容性原则：

- 新驱动前向兼容数据库，若需使用驱动与数据库同步增加的新特性，必须升级数据库。
- behavior_compat_options='proc_outparam_override' 重载参数仅在 A 兼容模式可用。

6.2 基于 JDBC 开发

JDBC（Java Database Connectivity，Java 数据库连接）是一种用于执行 SQL 语句的 Java

API，可以为多种关系数据库提供统一访问接口，应用程序可基于它操作数据。GBase 8c 数据库提供了对 JDBC 4.0 特性的支持，需要使用 JDK1.8 版本编译程序代码，不支持 JDBC 桥接 ODBC 方式。

6.2.1 JDBC 包、驱动类和环境类

JDBC 包

在 linux 服务器端源代码目录下执行 build.sh，获得驱动 jar 包 postgresql.jar。。

驱动包与 PostgreSQL 保持兼容，其中类名、类结构与 PostgreSQL 驱动完全一致，曾经运行于 PostgreSQL 的应用程序可以直接移植到当前系统使用。

驱动类

在创建数据库连接之前，需要加载数据库驱动类“org.postgresql.Driver”。

由于 GBase 8c 在 JDBC 的使用上与 PG 的使用方法保持兼容，所以同时在同一进程内使用两个 JDBC 驱动的时候，可能会类名冲突。

相比于 PG 驱动，GBase 8c 的 JDBC 驱动主要做了以下特性的增强：

(1) 支持 SHA256 加密方式登录。

(2) 支持对接实现 sf4j 接口的第三方日志框架。

(3) 支持容灾切换。

环境类

客户端需配置 JDK1.8，配置方法如下：

步骤 1 DOS 窗口输入“java -version”，查看 JDK 版本，确认为 JDK1.8 版本。如果未安装 JDK，请从官方网站下载安装包并安装。

步骤 2 根据如下步骤配置系统环境变量。

(1) 右键单击“我的电脑”，选择“属性”。

(2) 在“系统”页面左侧导航栏单击“高级系统设置”。

(3) 在“系统属性”页面，“高级”页签上单击“环境变量”。

在“环境变量”页面上，“系统变量”区域单击“新建”或“编辑”配置系统变量。变量说明请参见表 6-1。

表 6-1 变量说明

变量名	操作	变量值
JAVA_HO ME	若存在，则单击“编辑”。 若不存在，则单击“新建”。	JAVA 的安装目录。 例如：C:\Program Files\Java\jdk1.8.0_131
Path	编辑	若配置了 JAVA_HOME，则在变量值的最前面加上： %JAVA_HOME%\bin; 若未配置 JAVA_HOME，则在变量值的最前面加上 JAVA 安装的全路径： C:\Program Files\Java\jdk1.8.0_131\bin;
CLASSPAT H	新建	.;%JAVA_HOME%\lib;%JAVA_HOME%\lib\tools.jar;

----结束

6.2.2 开发流程

图 6-1 采用 JDBC 开发应用程序的流程

6.2.3 加载驱动

在创建数据库连接之前，需要先加载数据库驱动程序。

加载驱动有两种方法：

- 在代码中创建连接之前任意位置隐含装载：Class.forName("org.postgresql.Driver");
- 在 JVM 启动时参数传递：java -Djdbc.drivers=org.postgresql.Driver jdbctest

说明

上述 jdbctest 为测试用例程序的名称。

6.2.4 连接数据库

在创建数据库连接之后，才能使用它来执行 SQL 语句操作数据。

函数原型

JDBC 提供了三个方法，用于创建数据库连接。

- DriverManager.getConnection(String url);
- DriverManager.getConnection(String url, Properties info);
- DriverManager.getConnection(String url, String user, String password);

参数

表 6-2 数据库连接参数

参数	描述
info	数据库连接属性（所有属性大小写敏感）。常用的属性如下： ● PGDBNAME：String 类型。表示数据库名称。（URL 中无需配置该参数，自动从 URL 中解析） ● PGHOST：String 类型。主机 IP 地址。详细示例见下。 ● PGPORT：Integer 类型。主机端口号。详细示例见下。 ● user：String 类型。表示创建连接的数据库用户。 ● password：String 类型。表示数据库用户的密码。 ● enable_ce: String 类型。其中 enable_ce=1 表示 JDBC 支持密态等值查询。 ● refreshClientEncryption:String 类型。其中 refreshClientEncryption=1 表示密态

数据库支持客户端缓存刷新（默认值为 1）。

- loggerLevel：String 类型。目前支持 3 种级别：OFF、DEBUG、 TRACE。设置为 OFF 关闭日志，设置为 DEBUG 和 TRACE 记录的日志信息详细程度不同。
- loggerFile：String 类型。Logger 输出的文件名。需要显示指定日志文件名，若未指定目录则生成在客户端运行程序目录。此参数已废弃， 不再生效，如需使用可通过 java.util.logging 属性文件或系统属性进行配置。
- allowEncodingChanges：Boolean 类型。设置该参数值为“true”进行字符集类型更改，配合 characterEncoding=CHARSET 设置字符集， 二者使用“&”分隔；characterEncoding 取值范围为 UTF8、GBK、LATIN1。
- currentSchema：String 类型。在 search-path 中指定要设置的 schema。
- hostRecheckSeconds：Integer 类型。JDBC 尝试连接主机后会保存主机状态：连接成功或连接失败。在 hostRecheckSeconds 时间内保持可信，超过则状态失效。缺省值是 10 秒。
- ssl：Boolean 类型。以 SSL 方式连接。
- ssl=true 可支持 NonValidatingFactory 通道和使用证书的方式：

  1、NonValidatingFactory 通道需要配置用户名和密码，同时将 SSL 设置为 true。

  2、配置客户端证书、密钥、根证书，将 SSL 设置为 true。
- sslmode：String 类型。SSL 认证方式。取值范围为：require、verify- ca、verify-full。
  - require 只尝试 SSL 连接，如果存在 CA 文件，则应设置成 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。
- sslcert：String 类型。提供证书文件的完整路径。客户端和服务端证书的类型为 End Entity。
- sslkey：String 类型。提供密钥文件的完整路径。使用时将客户端证书转换为 DER 格式：

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

- sslrootcert：String 类型。SSL 根证书的文件名。根证书的类型为 CA。
- sslpassword：String 类型。提供给 ConsoleCallbackHandler 使用。
- sslpasswordcallback：String 类型。SSL 密码提供者的类名。缺省值：

org.postgresql.ssl.jdbc4.LibPQFactory.ConsoleCallbackHandler。

- sslfactory：String 类型。提供的值是 SSLSocketFactory 在建立 SSL 连接时用的类名。
- sslfactoryarg：String 类型。此值是上面提供的 sslfactory 类的构造函数的可选参数（不推荐使用）。
- sslhostnameverifier：String 类型。主机名验证程序的类名。接口实现 javax.net.ssl.HostnameVerifier，默认使用 org.postgresql.ssl.PGjdbcHostnameVerifier。
- loginTimeout：Integer 类型。指建立数据库连接的等待时间。超时时间单位为秒。
- connectTimeout：Integer 类型。用于连接服务器操作的超时值。如果连接到服务器花费的时间超过此值，则连接断开。超时时间单位为秒，值为 0 时表示已禁用，timeout 不发生。
- socketTimeout：Integer 类型。用于 socket 读取操作的超时值。如果从服务器读取所花费的时间超过此值，则连接关闭。超时时间单位为秒，值为 0 时表示已禁用，timeout 不发生。
- cancelSignalTimeout：Integer 类型。发送取消消息本身可能会阻塞，此属性控制用于取消命令的“connect 超时”和“socket 超时”。 超时时间单位为秒，默认值为 10 秒。
- tcpKeepAlive：Boolean 类型。启用或禁用 TCP 保活探测功能。默认为 false。
- logUnclosedConnections：Boolean 类型。客户端可能由于未调用 Connection 对象的 close()方法而泄漏 Connection 对象。最终这些对象将被垃圾回收，并且调用 finalize()方法。如果调用者自己忽略了此操作，该方法将关闭 Connection。
- assumeMinServerVersion：String 类型。客户端会发送请求进行 float 精度设置。该参数设置要连接的服务器版本，如 assumeMinServerVersion=9.0，可以在建立时减少相关包的发送。
- ApplicationName：String 类型。设置正在使用连接的 JDBC 驱动的名称。通过在数据库主节点上查询 pg_stat_activity 表可以看到正在连接的客户端信息，JDBC 驱动名称显示在 application_name 列。缺省值为 PostgreSQL JDBC Driver。
- connectionExtraInfo：Boolean 类型。表示驱动是否上报当前驱动的部署路径、进程属主用户到数据库。

  取值范围：true 或 false，默认值为 false。设置 connectionExtraInfo 为 true，JDBC 驱动会将当前驱动的部署路径、进程属主用户、url 连接配置信息上报到数据库中，记录在 connection_info 参数里；同时可以在

PG_STAT_ACTIVITY 中查询到。

- autosave：String 类型。共有 3 种："always", "never", "conservative"。如果查询失败，指定驱动程序应该执行的操作。在 autosave=always 模式下，JDBC 驱动程序在每次查询之前设置一个保存点，并在失败时回滚到该保存点。在 autosave=never 模式（默认）下，无保存点。在 autosave=conservative 模式下，每次查询都会设置保存点，但是只会在“statement XXX 无效”等情况下回滚并重试。
- protocolVersion：Integer 类型。连接协议版本号，目前仅支持 1 和 3。注意：设置 1 时仅代表连接的是 V1 服务端。设置 3 时将采用 md5 加密方式，需要同步修改数据库的加密方式：gs_guc set -N all -I all -c "password_encryption_type=1"，重启数据库生效后需要创建用 md5 方式加密口令的用户。同时修改 pg_hba.conf，将客户端连接方 式修改为 md5。用新建用户进行登录（不推荐）。

  说明：

  MD5 加密算法安全性低，存在安全风险，建议使用更安全的加密算法。
- prepareThreshold：Integer 类型。控制 parse 语句何时发送。默认值是 5。第一次 parse 一个 SQL 比较慢，后面再 parse 就会比较快，因为有缓存了。如果一个会话连续多次执行同一个 SQL，在达到 prepareThreshold 次数以上时，JDBC 将不再对这个 SQL 发送 parse 命 令。
- preparedStatementCacheQueries：Integer 类型。确定每个连接中缓存的查询数，默认情况下是 256。若在 prepareStatement()调用中使用超过 256 个不同的查询，则最近最少使用的查询缓存将被丢弃。0 表示禁用缓存。
- preparedStatementCacheSizeMiB：Integer 类型。确定每个连接可缓存的最大值（以兆字节为单位），默认情况下是 5。若缓存了超过 5MB 的查询，则最近最少使用的查询缓存将被丢弃。0 表示禁用缓存。
- databaseMetadataCacheFields：Integer 类型。默认值是 65536。指定每个连接可缓存的最大值。“0”表示禁用缓存。
- databaseMetadataCacheFieldsMiB：Integer 类型。默认值是 5。每个连接可缓存的最大值，单位是 MB。“0”表示禁用缓存。
- stringtype：String 类型，可选字段为：false, "unspecified", "varchar"。设置通过 setString()方法使用的 PreparedStatement 参数的类型，如果 stringtype 设置为 VARCHAR（默认值），则这些参数将作为 varchar 参数发送给服务器。若 stringtype 设置为 unspecified，则参数将作为 untyped 值发送到服务器，服务器将尝试推断适当的类型。
- batchMode：String 类型。用于确定是否使用 batch 模式连接。默认值为 on，表示开启 batch 模式。
- fetchsize：Integer 类型。用于设置数据库连接所创建 statement 的默认

fetchsize。默认值为 0，表示一次获取所有结果。

- reWriteBatchedInserts：Boolean 类型。批量导入时，该参数设置为 true，可将 N 条插入语句合并为一条：insert into TABLE_NAME values(values1, ..., valuesN), ..., (values1, ..., valuesN);使用该参数时，需设置 batchMode=off。
- unknownLength：Integer 类型，默认为 Integer.MAX_VALUE。某些 postgresql 类型（例如 TEXT）没有明确定义的长度，当通过 ResultSetMetaData.getColumnDisplaySize 和 ResultSetMetaData.getPrecision 等函数返回关于这些类型的数据时，此参数指定未知长度类型的长度。
- uppercaseAttributeName：Boolean 类型，默认值为 false 不开启，为 true 时开启。该参数开启后会将获取元数据的接口的查询结果转为大写。适用场景为数据库中存储元数据全为小写，但要使用大写的元数据作为出参和入参。

  涉及到的接口：8.1.3 java.sql.DatabaseMetaData、8.1.7java.sql.ResultSetMetaData
- defaultRowFetchSize：Integer 类型。确定一次 fetch 在 ResultSet 中读取的行数。限制每次访问数据库时读取的行数可以避免不必要的内存消耗，从而避免 OutOfMemoryException。缺省值是 0，这意味着 ResultSet 中将一次获取所有行。没有负数。
- binaryTransfer：Boolean 类型。使用二进制格式发送和接收数据，默认值为“false”。
- binaryTransferEnable：String 类型。启用二进制传输的类型列表，以逗号分隔。OID 编号和名称二选一，例如 binaryTransferEnable=Integer4_ARRAY,Integer8_ARRAY。

  比如：OID 名称为 BLOB，编号为 88，可以如下配置：

  binaryTransferEnable=BLOB 或 binaryTransferEnable=88
- binaryTransferDisEnable：String 类型。禁用二进制传输的类型列表，以逗号分隔。OID 编号和名称二选一。覆盖 binaryTransferEnable 的设置。
- blobMode：String 类型。用于设置 setBinaryStream 方法为不同类型 的数据赋值，设置为 on 时表示为 blob 类型数据赋值，设置为 off 时表示为 bytea 类型数据赋值，默认为 on。
- socketFactory：String 类型。用于创建与服务器 socket 连接的类的名称。该类必须实现了接口“javax.net.SocketFactory”，并定义无参 或单 String 参数的构造函数。
- socketFactoryArg：String 类型。此值是上面提供的 socketFactory 类的构造函数的可选参数，不推荐使用。
- receiveBufferSize：Integer 类型。该值用于设置连接流上的 SO_RCVBUF。

- sendBufferSize：Integer 类型。该值用于设置连接流上的 SO_SNDBUF。
- preferQueryMode：String 类型。共有 4 种："extended", "extendedForPrepared", "extendedCacheEverything", "simple"。用于指定执行查询的模式，simple 模式会 excute，不 parse 和 bind； extended 模式会 bind 和 excute； extendedForPrepared 模式为 prepared statement 扩展使用；extendedCacheEverything 模式会缓存每个 statement。
- targetServerType： String 类型。该参数识别主备数据节点是通过查询 URL 连接串中，数据节点是否允许写操作来实现的，默认为"any"。共有四种："any"，"master"，"slave"，"preferSlave"：
  - master 则尝试连接到 URL 连接串中的主节点，如果找不到就抛出异常。
  - slave 则尝试连接到 URL 连接串中的备节点，如果找不到就抛出异常。
  - preferSlave 则尝试连接到 URL 连接串中的备数据节点（如果有可用的话），否则连接到主数据节点。
  - any 则尝试连接 URL 连接串中的任何一个数据节点。
- priorityServers：Integer 类型。此值用于指定 url 上配置的前 n 个节点作为主数据库实例被优先连接。默认值为 null。该值为数字，大于 0， 且小于 url 上配置的 DN 数量。

  例如：jdbc:postgresql://host1:port1,host2:port2,host3:port3,host4:port4,/database? priorityServers=2。即表示 host1 与 host2 为主数据库实例节点，host3 与 host4 为容灾数据库实例节点。
- forceTargetServerSlave：Boolean 类型。此值用于控制是否开启强制连接备机功能，并在数据库实例发生主备切换时，禁止已存在的连接在升主备机上继续使用。默认值为 false，表示不开启强制连接备机功能。true，表示开启强制连接备机功能。
- traceInterfaceClass：String 类型。默认值为 null，用于获取 traceId 的实现类。值是实现获取 traceId 方法的接口 org.postgresql.log.Tracer 的 实现类的完整限定类名。
- use_boolean：Boolean 类型。用于设置 extended 模式下 setBoolean 方法绑定的 oid 类型，默认为 false，绑定 int2 类型；设置为 true 则绑定 bool 类型。
- allowReadOnly：Boolean 类型。用于设置是否允许只读模式，默认为 true，允许设置只读模式；设置为 false 则禁用只读模式。
- TLSCiphersSupperted：String 类型。用于设置支持的 TLS 加密套件， 默认为 TLS_DHE_RSA_WITH_AES_128_GCM_SHA256,TLS_DHE_RSA_WITH_AES_256_GCM_SHA384,TLS_ECDHE_RSA_WITH_AES_128_GCM_S

	HA256,TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384,TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256,TLS_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384
user	数据库用户。
password	数据库用户的密码。

说明

uppercaseAttributeName 参数开启后，如果数据库中有小写、大写和大小写混合的元数据，只能查询出小写部分的元数据，并以大写的形式输出，使用前请务必确认元数据的存储是否全为小写以避免数据出错。

示例

```
//以下代码将获取数据库连接操作封装为一个接口，可通过给定用户名和密码来连接数据库。
public static Connection getConnect(String username, String passwd)
{
//驱动类。
String driver = "org.postgresql.Driver";
//数据库连接描述符。
String sourceURL = "jdbc:postgresql://10.10.0.13:5432/postgres"; Connection conn = null;
try
{
//加载驱动。Class.forName(driver);
}
catch( Exception e )
{
e.printStackTrace(); return null;
}
try
{
//创建连接。
conn = DriverManager.getConnection(sourceURL, username, passwd);
System.out.println("Connection succeed!");
}
catch(Exception e)
{
e.printStackTrace(); return null;
}
return conn;
```

```
};
// 以下代码将使用 Properties 对象作为参数建立连接
public static Connection getConnectUseProp(String username, String passwd)
{
//驱动类。
String driver = "org.postgresql.Driver";
//数据库连接描述符。
String sourceURL = "jdbc:postgresql://10.10.0.13:5432/postgres?"; Connection conn = null;
Properties info = new Properties();
try
{
//加载驱动。Class.forName(driver);
}
catch( Exception e )
{
e.printStackTrace(); return null;
}
try
{
info.setProperty("user", username); info.setProperty("password", passwd);
//创建连接。
conn = DriverManager.getConnection(sourceURL, info); System.out.println("Connection
succeed!");
}
catch(Exception e)
{
e.printStackTrace(); return null;
}
return conn;
};
```

6.2.5 连接数据库（以 SSL 方式）

用户通过 JDBC 连接数据库服务器时，可以通过开启 SSL 加密客户端和服务器之间 的通讯，为敏感数据在 Internet 上的传输提供了一种安全保障手段。本小节主要介绍应用程序通过 JDBC 如何采用 SSL 的方式连接 GBase 8c 数据库。在使用本小节所描述的方法前，默认用户已经获取了服务端和客户端所需要的证书和私钥文件，关于证书等文件的获取请参考 Openssl 相关文档和命令。

服务端配置

当开启 SSL 模式后，必须提供根证书、服务器证书和私钥。

配置步骤（假设用户的证书文件放在数据目录//home/gbase/data/dn1/dn1_1/下，且采用默认文件名）：

步骤 1 以操作系统用户 gbase 登录数据库主节点。

步骤 2 生成并配置证书。

生成 SSL 证书，具体请参见 7.1.6.1 证书生成。将生成出的文件 server.crt，server.key，cacert.pem 拷贝到服务端数据目录下。

使用如下命令可以查询数据库节点的数据目录，instance 列为数据目录。

```
gs_om -t status --detail
```

在 Unix 系统上，server.crt、server.key 的权限设置必须禁止任何外部或组的访问，请执行如下命令实现这一点。

```
chmod 0600 server.key
```

步骤 3 开启 SSL 认证模式。

```
gs_guc set -D /home/gbase/data/dn1/dn1_1/ -c "ssl=on"
```

步骤 4 配置客户端接入认证参数，IP 为所要连接的主机 IP。

```
gs_guc reload -D /home/gbase/data/dn1/dn1_1/ -h "hostssl all       all      127.0.0.1/32
cert"
```

gs_guc reload -D /home/gbase/data/dn1/dn1_1/ -h "hostssl all all IP/32 cert"

表示允许 127.0.0.1/32 网段的客户端以 ssl 认证方式连接到数据库服务器。

须知

- 如果服务端 pg_hba.conf 文件中 METHOD 配置为 cert，则只有客户端使用证书（client.crt）中所设置的用户名（common name）才能够成功连接数据库。如果设置为 md5、sm3 或 sha256 则对连接数据库的用户没有限制。
- MD5 加密算法安全性低，存在安全风险，建议使用更安全的加密算法。

步骤 5 配置 SSL 认证相关的数字证书参数。各命令后所附为设置成功的回显。

```
gs_guc set -D /home/gbase/data/dn1/dn1_1/ -c "ssl_cert_file='server.crt'"
gs_guc set: ssl_cert_file='server.crt'
gs_guc set -D /home/gbase/data/dn1/dn1_1/ -c "ssl_key_file='server.key'"
gs_guc set: ssl_key_file='server.key'
```

```
gs_guc set -D /home/gbase/data/dn1/dn1_1/ -c "ssl_ca_file='cacert.pem'"
gs_guc set: ssl_ca_file='cacert.pem'
```

步骤 6 重启数据库。

```
gha_ctl stop all -l http://<dcs_ip>:2379
gha_ctl start all -l http://<dcs_ip>:2379
```

----结束

客户端配置

配置步骤如下：

上传证书文件，将在服务端配置章节生成出的文件 client.key.pk8，client.crt，cacert.pem 放置在客户端。

示例

注：示例 1 和示例 2 选择其一。

```
public class SSL{
public static void main(String[] args) { Properties urlProps = new Properties();
String urls = "jdbc:postgresql://10.29.37.136:5432/postgres";
/**
* ================== 示例 1 使用 NonValidatingFactory 通道
*/ urlProps.setProperty("sslfactory","org.postgresql.ssl.NonValidatingFactory");
urlProps.setProperty("user", "world");
urlProps.setProperty("password", "test@123"); urlProps.setProperty("ssl", "true");
/**
* ================== 示例 2 使用证书
*/
urlProps.setProperty("sslcert", "client.crt"); urlProps.setProperty("sslkey", "client.key.pk8");
urlProps.setProperty("sslrootcert", "cacert.pem"); urlProps.setProperty("user", "world");
urlProps.setProperty("ssl", "true");
/* sslmode 可配置为：require、verify-ca、verify-full，以下三个示例选择其一*/
/* ================== 示例 2.1 设置 sslmode 为 require，使用证书 */
urlProps.setProperty("sslmode", "require");
/* ================== 示例 2.2 设置 sslmode 为 verify-ca，使用证书 */
urlProps.setProperty("sslmode", "verify-ca");
/* ================== 示例 2.3 设置 sslmode 为 verify-full，使用证书（Linux 下验证）
*/ urls = "jdbc:postgresql://world:5432/postgres";
urlProps.setProperty("sslmode", "verify-full"); try {
Class.forName("org.postgresql.Driver").newInstance();
} catch (Exception e) { e.printStackTrace();
```

```
}
try {
Connection conn;
conn = DriverManager.getConnection(urls,urlProps); conn.close();
} catch (Exception e) { e.printStackTrace();
}
}
}
/**
注：将客户端密钥转化为 DER 格式:
openssl pkcs8 -topk8 -outform DER -in client.key -out client.key.pk8 -nocrypt
openssl pkcs8 -topk8 -inform PEM -in client.key -outform DER -out client.key.der -v1
PBE-MD5-DES
openssl pkcs8 -topk8 -inform PEM -in client.key -outform DER -out client.key.der -v1
PBE-SHA1-3DES
以上算法由于安全级别较低，不推荐使用。
如果客户需要采用更高级别的私钥加密算法，启用 bouncycastle 或者其他第三方私钥解密
密码包后可以使用的私钥加密算法如下：
openssl pkcs8 -in client.key -topk8 -outform DER -out client.key.der -v2 AES128
openssl pkcs8 -in client.key -topk8 -outform DER -out client.key.der -v2 aes-256-cbc -iter
1000000
openssl pkcs8 -in client.key -topk8 -out client.key.der -outform Der -v2 aes-256-cbc -v2prf
hmacWithSHA512
启用 bouncycastle：使用 jdbc 的项目引入依赖：bcpkix-jdk15on.jar 包，版本建议：1.65 以
上。
*/
```

6.2.6 执行 SQL 语句

6.2.6.1 执行普通 SQL

应用程序通过执行 SQL 语句来操作数据库的数据（不用传递参数的语句），需要按以下步骤执行：

```
Connection conn = DriverManager.getConnection("url","user","password"); Statement stmt =
conn.createStatement();
```

步骤 1 调用 Connection 的 createStatement 方法创建语句对象。

```
int rc = stmt.executeUpdate("CREATE TABLE customer_t1(c_customer_sk INTEGER,
c_customer_name VARCHAR(32));");
```

步骤 2 调用 Statement 的 executeUpdate 方法执行 SQL 语句。

- 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，事务块中不支持 vacuum 操作。如果其中有一个语句失败，那么整个请求都将会被回滚。
- 使用 Statement 执行多语句时应以“;”作为各语句间的分隔符，存储过程、函数、匿名块不支持多语句执行。
- “/”可用作创建单个存储过程、函数、匿名块的结束符。

步骤 3 关闭语句对象。

```
stmt.close();
```

----结束

6.2.6.2 执行预编译 SQL 语句

预编译语句是只编译和优化一次，然后可以通过设置不同的参数值多次使用。由于已经预先编译好，后续使用会减少执行时间。因此，如果多次执行一条语句，请选择使用预编译语句。可以按以下步骤执行：

步骤 1 调用 Connection 的 prepareStatement 方法创建预编译语句对象。

步骤 2 调用 PreparedStatement 的 setShort 设置参数。

```
pstmt.setShort(1, (short)2);
```

步骤 3 调用 PreparedStatement 的 executeUpdate 方法执行预编译 SQL 语句。

```
int rowcount = pstmt.executeUpdate();
```

步骤 4 调用 PreparedStatement 的 close 方法关闭预编译语句对象。

```
pstmt.close();
```

----结束

6.2.6.3 调用存储过程

GBase 8c 支持通过 JDBC 直接调用事先创建的存储过程，步骤如下：

步骤 1 调用 Connection 的 prepareCall 方法创建调用语句对象。

```
Connection myConn = DriverManager.getConnection("url","user","password");
CallableStatement cstmt = myConn.prepareCall("{? = CALL TESTPROC(?,?,?)}");
```

步骤 2 调用 CallableStatement 的 setInt 方法设置参数。

```
cstmt.setInt(2, 50);
cstmt.setInt(1, 20);
cstmt.setInt(3, 90);
```

步骤 3 调用 CallableStatement 的 registerOutParameter 方法注册输出参数。

```
cstmt.registerOutParameter(4, Types.INTEGER); //注册 out 类型的参数，类型为整型。
```

步骤 4 调用 CallableStatement 的 execute 执行方法调用。

```
cstmt.execute();
```

步骤 5 调用 CallableStatement 的 getInt 方法获取输出参数。

```
int out = cstmt.getInt(4); //获取 out 参数
```

示例：

```
//在数据库中已创建了如下存储过程，它带有 out 参数。
create or replace procedure testproc (
psv_in1 in integer, psv_in2 in integer, psv_inout in out integer
)
as begin
psv_inout := psv_in1 + psv_in2 + psv_inout; end;
/
```

步骤 6 调用 CallableStatement 的 close 方法关闭调用语句。

```
cstmt.close();
```

说明

- 很多的数据库类如 Connection、Statement 和 ResultSet 都有 close()方法，在使用完对象后应把它们关闭。要注意的是，Connection 的关闭将间接关闭所有与它关联的 Statement， Statement 的关闭间接关闭了 ResultSet。
- 一些 JDBC 驱动程序还提供命名参数的方法来设置参数。命名参数的方法允许根据名称而不是顺序来设置参数，若参数有默认值，则可以不用指定参数值就可以使用此参数的默认值。即使存储过程中参数的顺序发生了变更，也不必修改应用程序。目前 GBase 8c 数据库的 JDBC 驱动程序不支持此方法。
- GBase 8c 数据库不支持带有输出参数的函数，也不支持存储过程和函数参数默认值。

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。

- 存储过程不能和普通 SQL 在同一条语句中执行。
- 存储过程中 inout 类型参数必需注册出参。

6.2.6.4 Oracle 兼容模式启用重载时，调用存储过程

打开参数 behavior_compat_options='proc_outparam_override'后，JDBC 调用事先创建的存储过程，步骤如下：

步骤 1 调用 Connection 的 prepareCall 方法创建调用语句对象。

```
Connection conn = DriverManager.getConnection("url","user","password"); CallableStatement cs = conn.prepareCall("{ CALL TEST_PROC(?,?,?) }");
```

步骤 2 调用 CallableStatement 的 setInt 方法设置参数。

```
PGobject pGobject = new PGobject();
pGobject.setType("public.compfoo"); // 设置复合类型名，格式为“schema.typename”。
pGobject.setValue("(1,demo)"); // 绑定复合类型值，格式为“(value1,value2)”。cs.setObject(1, pGobject);
```

步骤 3 调用 CallableStatement 的 registerOutParameter 方法注册输出参数。

```
// 注册 out 类型的参数，类型为复合类型,格式为“schema.typename”。
cs.registerOutParameter(2, Types.STRUCT, "public.compfoo");
```

步骤 4 调用 CallableStatement 的 execute 执行方法调用。

```
cs.execute();
```

步骤 5 调用 CallableStatement 的 getObject 方法获取输出参数。

```
PGobject result = (PGobject)cs.getObject(2);   // 获取 out 参数
result.getValue(); // 获取复合类型字符串形式值。
result.getArrayValue(); //获取复合类型数组形式值，以复合数据类型字段顺序排序。
result.getStruct(); //获取复合类型子类型名，按创建顺序排序。
```

步骤 6 调用 CallableStatement 的 close 方法关闭调用语句。

```
cs.close();
```

说明

- oracle 兼容模式开启参数后，调用存储过程必须使用{call proc_name(?,?,?)}形式调用，调用函数必须使用{? = call func_name(?,?)}形式调用（等号左侧的“？”为函数返回值的占位符，用于注册函数返回值）。

- 参数 behavior_compat_options='proc_outparam_override'行为变更后，业务需要重新建立连接，否则无法正确调用存储过程和函数。
- 函数和存储过程中包含复合类型时，参数的绑定与注册需要使用 schema.typename 形式。

```
// 在数据库创建复合数据类型。
CREATE TYPE compfoo AS (f1 int, f3 text);
// 在数据库中已创建了如下存储过程，它带有 out 参数。
create or replace procedure test_proc (
psv_in in compfoo, psv_out out compfoo
)
as begin
psv_out := psv_in; end;
/
```

----结束

6.2.6.5 执行批处理

用一条预处理语句处理多条相似的数据，数据库只创建一次执行计划，节省了语句的编译和优化时间。可以按如下步骤执行：

步骤 1 调用 Connection 的 prepareStatement 方法创建预编译语句对象。

```
Connection conn = DriverManager.getConnection("url","user","password"); PreparedStatement
pstmt = conn.prepareStatement("INSERT INTO customer_t1 VALUES (?)");
```

步骤 2 针对每条数据都要调用 setShort 设置参数，以及调用 addBatch 确认该条设置完毕。

```
pstmt.setShort(1, (short)2);
pstmt.addBatch();
```

步骤 3 调用 PreparedStatement 的 executeBatch 方法执行批处理。

```
int[] rowcount = pstmt.executeBatch();
```

步骤 4 调用 PreparedStatement 的 close 方法关闭预编译语句对象。

```
pstmt.close();
```

说明

在实际的批处理过程中，通常不终止批处理程序的执行，否则会降低数据库的性能。因此在批处理程序时，应该关闭自动提交功能，每几行提交一次。关闭自动提交功能的语句为：conn.setAutoCommit(false);

----结束

6.2.7 处理结果集

SQL 兼容性检查

GBase 8c 数据库支持接收外部发出的 SQL，并对其执行语法解析，无需重写或执行（因为数据库内当前不存在相应元数据）。若失败，返回失败原因。建议配合 GBase 8c 迁移工具使用，用于 SQL 兼容性检查。

设置结果集类型

不同类型的结果集有各自的应用场景，应用程序需要根据实际情况选择相应的结果集类型。在执行 SQL 语句过程中，都需要先创建相应的语句对象，而部分创建语句对象的方法提供了设置结果集类型的功能。具体的参数设置如表 6-3 所示。涉及的 Connection 的方法如下：

```
//创建一个 Statement 对象，该对象将生成具有给定类型和并发性的 ResultSet 对象。
createStatement(int resultSetType, int resultSetConcurrency);
//创建一个 PreparedStatement 对象，该对象将生成具有给定类型和并发性的 ResultSet 对象。
prepareStatement(String sql, int resultSetType, int resultSetConcurrency);
//创建一个 CallableStatement 对象，该对象将生成具有给定类型和并发性的 ResultSet 对象。
prepareCall(String sql, int resultSetType, int resultSetConcurrency);
```

表 6-3 结果集类型

参数	描述
resultSetType	表示结果集的类型，具体有三种类型： ● ResultSet.TYPE_FORWARD_ONLY：ResultSet 只能向前移动。是缺省值。 ● ResultSet.TYPE_SCROLL_SENSITIVE：在修改后重新滚动到修改所在行，可以看到修改后的结果。 ● ResultSet.TYPE_SCROLL_INSENSITIVE：对可修改例程所做的编辑不进行显示。 说明： 结果集从数据库中读取了数据之后，即使类型是 ResultSet.TYPE_SCROLL_SENSITIVE，也不会看到由其他事务在这之后引起的改变。调用 ResultSet 的 refreshRow()方法，可进入数据库并从其中取得当前游标所指记录的最新数据。

resultSetConcurrenc y	表示结果集的并发，具体有两种类型： ● ResultSet.CONCUR_READ_ONLY：如果不从结果集中的数据建立一个新的更新语句，不能对结果集中的数据进行更新。 ● ResultSet.CONCUR_UPDATEABLE：可改变的结果集。对于可滚动的结果集，可对结果集进行适当的改变。

在结果集中定位

ResultSet 对象具有指向其当前数据行的光标。最初，光标被置于第一行之前。next 方法将光标移动到下一行；因为该方法在 ResultSet 对象没有下一行时返回 false，所以可

以在 while 循环中使用它来迭代结果集。但对于可滚动的结果集，JDBC 驱动程序提供更多的定位方法，使 ResultSet 指向特定的行。定位方法如表 6-4 所示。

表 6-4 在结果集中定位的方法

方法	描述
next()	把 ResultSet 向下移动一行。
previous()	把 ResultSet 向上移动一行。
beforeFirst()	把 ResultSet 定位到第一行之前。
afterLast()	把 ResultSet 定位到最后一行之后。
first()	把 ResultSet 定位到第一行。
last()	把 ResultSet 定位到最后一行。
absolute(int)	把 ResultSet 移动到参数指定的行数。
relative(int)	通过设置为 1 向前（设置为 1，相当于 next()）或者向后（设置为-1，相当于 previous()）移动参数指定的行。

获取结果集中光标的位置

对于可滚动的结果集，可能会调用定位方法来改变光标的位置。JDBC 驱动程序提供了获取结果集中光标所处位置的方法。获取光标位置的方法如表 6-5 所示。

表 6-5 获取结果集光标的位置

方法	描述

isFirst()	是否在一行。
isLast()	是否在最后一行。
isBeforeFirst()	是否在第一行之前。
isAfterLast()	是否在最后一行之后。
getRow()	获取当前在第几行。

获取结果集中的数据

ResultSet 对象提供了丰富的方法，以获取结果集中的数据。获取数据常用的方法如表 6-6 所示，其他方法请参考 JDK 官方文档。

表 6-6 ResultSet 对象的常用方法

方法	描述
int getInt(int columnIndex)	按列标获取 int 型数据。
int getInt(String columnLabel)	按列名获取 int 型数据。
String getString(int columnIndex)	按列标获取 String 型数据。
String getString(String columnLabel)	按列名获取 String 型数据。
Date getDate(int columnIndex)	按列标获取 Date 型数据
Date getDate(String columnLabel)	按列名获取 Date 型数据。

6.2.8 关闭连接

在使用数据库连接完成相应的数据操作后，需要关闭数据库连接。

关闭数据库连接可以直接调用其 close 方法即可。如：Connection conn = DriverManager.getConnection("url","user","password") ; conn.close() ;

6.2.9 日志管理

GBase 8c JDBC 驱动程序支持使用日志记录，来帮助解决在应用程序中使用 JDBC 驱动程序时的问题。GBase 8c 的 JDBC 支持如下三种日志管理方式：

- 对接应用程序使用的 SLF4J 日志框架。

- 对接应用程序使用的 JdkLogger 日志框架。
- SLF4J 和 JdkLogger 是业界 Java 应用程序日志管理的主流框架，描述应用程序如何使用这些框架超出了本文范围，用户请参考对应的官方文档（SLF4J：http://www.slf4j.org/manual.html，JdkLogger：https://docs.oracle.com/javase/8/docs/technotes/guides/logging/overview.html）。

  方式一：对接应用程序的 SLF4J 日志框架。

  在建立连接时，url 配置 logger=Slf4JLogger。

  可采用 Log4j 或 Log4j2 来实现 SLF4J。当采用 Log4j 实现 SLF4J，需要添加如下 jar 包：

  log4j-*.jar、slf4j-api-*.jar、slf4j-log4*-*.jar，（*区分版本），和配置文件：

  log4j.properties。若采用 Log4j2 实现 SLF4J，需要添加如下 jar 包：log4j-api-*.jar、log4j-core-*.jar、log4j-slf4j18-impl-*.jar、slf4j-api-*-alpha1.jar（*区分版本），和配置文件：log4j2.xml。

  此方式支持日志管控。SLF4J 可通过文件中的相关配置实现强大的日志管控功能，建议使用此方式进行日志管理。

  ⚠ 注意

  此方式依赖 slf4j 的通用 API 接口，如 org.slf4j.LoggerFactory.getLogger(String name)、org.slf4j.Logger.debug(String var1)、org.slf4j.Logger.info(String var1)、org.slf4j.Logger.warn(String warn)、org.slf4j.Logger.warn(String warn)等，若以上接口发生变更，日志将无法打印。

  示例：

```
public static Connection GetConnection(String username, String passwd){
String sourceURL = "jdbc:postgresql://10.10.0.13:5432/postgres?logger=Slf4JLogger";
Connection conn = null;
try{
//创建连接
conn = DriverManager.getConnection(sourceURL,username,passwd);
System.out.println("Connection succeed!");
}catch (Exception e){ e.printStackTrace(); return null;
}
return conn;
}
```

log4j.properties 示例：

```
log4j.logger.org.postgresql=ALL, log_gsjdbc
# 默认文件输出配置 log4j.appender.log_gsjdbc=org.apache.log4j.RollingFileAppender
log4j.appender.log_gsjdbc.Append=true log4j.appender.log_gsjdbc.File=gsjdbc.log
log4j.appender.log_gsjdbc.Threshold=TRACE log4j.appender.log_gsjdbc.MaxFileSize=10MB
log4j.appender.log_gsjdbc.MaxBackupIndex=5
log4j.appender.log_gsjdbc.layout=org.apache.log4j.PatternLayout
log4j.appender.log_gsjdbc.layout.ConversionPattern=%d %p %t %c - %m%n
log4j.appender.log_gsjdbc.File.Encoding = UTF-8
```

log4j2.xml 示例：

```
<?xml version="1.0" encoding="UTF-8"?>
<configuration status="OFF">
<appenders>
<Console name="Console" target="SYSTEM_OUT">
<PatternLayout pattern="%d %p %t %c - %m%n"/>
</Console>
<File name="FileTest" fileName="test.log">
<PatternLayout pattern="%d %p %t %c - %m%n"/>
</File>
<!--JDBC Driver 日志文件输出配置，支持日志回卷，设定日志大小超过 10MB 时，创建新的文件，新文件的命名格式为：年-月-日-文件编号-->
<RollingFile name="RollingFileJdbc" fileName="gsjdbc.log"
filePattern="%d{yyyy-MM-dd}-%i.log">
<PatternLayout pattern="%d %p %t %c - %m%n"/>
<Policies>
<SizeBasedTriggeringPolicy size="10 MB"/>
</Policies>
</RollingFile>
</appenders>
<loggers>
<root level="all">
<appender-ref ref="Console"/>
<appender-ref ref="FileTest"/>
</root>
<!--指定 JDBC Driver 日志，级别为：all，可查看所有日志，输出到 gsjdbc.log 文件中-->
<logger name="org.postgresql" level="all" additivity="false">
<appender-ref ref="RollingFileJdbc"/>
</logger>
</loggers>
</configuration>
```

方式二：对接应用程序使用的 JdkLogger 日志框架。

默认的 Java 日志记录框架将其配置存储在名为 logging.properties 的文件中。Java 会在 Java 安装目录的文件夹中安装全局配置文件。logging.properties 文件也可以创建并与单个项目一起存储。

logging.properties 配置示例：

```
# 指定处理程序为文件。
handlers= java.util.logging.FileHandler
# 指定默认全局日志级别
.level= ALL
# 指定日志输出管控标准 java.util.logging.FileHandler.level=ALL
java.util.logging.FileHandler.pattern = gsjdbc.log java.util.logging.FileHandler.limit = 500000
java.util.logging.FileHandler.count = 30 java.util.logging.FileHandler.formatter =
java.util.logging.SimpleFormatter java.util.logging.FileHandler.append=false
```

代码中使用示例：

```
System.setProperty("java.util.logging.FileHandler.pattern","jdbc.log");
FileHandler fileHandler = new
FileHandler(System.getProperty("java.util.logging.FileHandler.pattern")); Formatter formatter
= new SimpleFormatter();
fileHandler.setFormatter(formatter);
Logger logger = Logger.getLogger("org.postgresql"); logger.addHandler(fileHandler);
logger.setLevel(Level.ALL); logger.setUseParentHandlers(false);
```

链路跟踪功能

JDBC 驱动程序提供了应用到数据库的链路跟踪功能，用于将数据库端离散的 SQL 和应用程序的请求关联起来。该功能需要应用开发者实现 org.postgresql.log.Tracer 接口类，并在 url 中指定接口实现类的全限定名。

url 示例：

```
String URL = "jdbc:postgresql://127.0.0.1:5432/postgres?
traceInterfaceClass=xxx.xxx.xxx.OpenGaussTraceImpl";
```

org.postgresql.log.Tracer 接口类定义如下：

```
public interface Tracer {
// Retrieves the value of traceId. String getTraceId();
}
```

org.postgresql.log.Tracer 接口实现类示例：

```
import org.postgresql.log.Tracer;
public class OpenGaussTraceImpl implements Tracer {
private static MDC mdc = new MDC(); private final String TRACE_ID_KEY = "traceId";
public void set(String traceId) { mdc.put(TRACE_ID_KEY, traceId);
}
public void reset() { mdc.clear();
}
@Override
public String getTraceId() {
return mdc.get(TRACE_ID_KEY);
}
}
```

上下文映射示例，用于存放不同请求的生成的 traceId。

```
import java.util.HashMap;
public class MDC {
static final private ThreadLocal<HashMap<String, String>> threadLocal = new
ThreadLocal<>();
public void put(String key, String val) { if (key == null || val == null) {
throw new IllegalArgumentException("key or val cannot be null");
} else {
if (threadLocal.get() == null) { threadLocal.set(new HashMap<>());
}
threadLocal.get().put(key, val);
}
}
public String get(String key) { if (key == null) {
throw new IllegalArgumentException("key cannot be null");
} else if (threadLocal.get() == null) { return null;
} else {
return threadLocal.get().get(key);
}
}
public void clear() {
if (threadLocal.get() == null) { return;
} else {
threadLocal.get().clear();
}
}
}
```

业务使用 traceId 示例。

```
String traceId = UUID.randomUUID().toString().replaceAll("-", "");
openGaussTrace.set(traceId);
pstm = con.prepareStatement("select * from test_trace_id where id = ?"); pstm.setInt(1, 1);
pstm.execute();
pstm = con.prepareStatement("insert into test_trace_id values(?,?)"); pstm.setInt(1, 2);
pstm.setString(2, "test"); pstm.execute(); openGaussTrace.reset();
```

6.2.10 JDBC 接口参考

请参见 8.1 JDBC。

6.3 基于 ODBC 开发

ODBC（Open Database Connectivity，开放数据库互连）是由 Microsoft 公司基于 X/ OPEN CLI 提出的用于访问数据库的应用程序编程接口。应用程序通过 ODBC 提供的 API 与数据库进行交互，增强了应用程序的可移植性、扩展性和可维护性。

ODBC 的系统结构参见图 6-2。

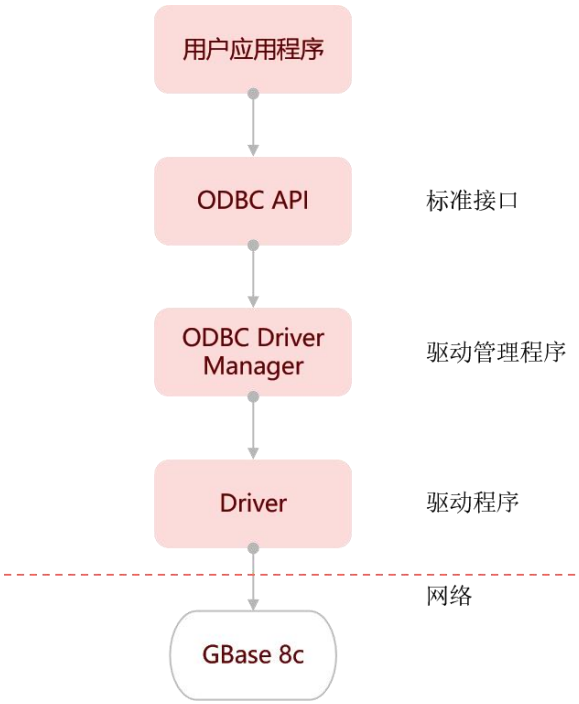


图 6-2 ODBC 系统机构

GBase 8c 目前在以下环境中提供对 ODBC 的支持。

表 6-7 ODBC 支持平台

操作系统	平台
CentOS 6.4/6.5/6.6/6.7/6.8/6.9/7.0/7.1/7.2/7.3/7.4	x86_64 位
CentOS 7.6	ARM64 位
EulerOS 2.0 SP2/SP3	x86_64 位
EulerOS 2.0 SP8	ARM64 位

UNIX/Linux 系统下的驱动程序管理器主要有 unixODBC 和 iODBC，在这选择驱动管理器 unixODBC-2.3.7 作为连接数据库的组件。

Windows 系统自带 ODBC 驱动程序管理器，在控制面板->管理工具中可以找到数据源（ODBC）选项。

说明

当前数据库 ODBC 驱动基于开源版本，对于 tinyint、smalldatetime、nvarchar、nvarchar2 类型，在获取数据类型的时候，可能会出现不兼容。

6.3.1 ODBC 包及依赖的库和头文件

Linux 下的 ODBC 包

获取发布包。Linux 环境下，开发应用程序要用到 unixODBC 提供的头文件（sql.h、sqlext.h 等）和库 libodbc.so。这些头文件和库可从 unixODBC-2.3.0 的安装包中获得。

6.3.2 Linux 下配置数据源

将 GBase 8c 提供的 ODBC DRIVER（psqlodbcw.so）配置到数据源中便可使用。配置数据源需要配置“odbc.ini”和“odbcinst.ini”两个文件（在编译安装 unixODBC 过程中生成且默认放在“/usr/local/etc”目录下），并在服务器端进行配置。

操作步骤

步骤 1 获取 unixODBC 源码包。

获取参考地址：https://sourceforge.net/projects/unixodbc/files/unixODBC/2.3.9/unixODBC-2.3.9pre.tar.gz/download

步骤 2 安装 unixODBC。如果机器上已经安装了其他版本的 unixODBC，可以直接覆盖安装。

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。

步骤 3 替换客户端驱动程序。

(1) 解压 tar 包。解压后会得到两个文件夹：lib 与 odbc，在 odbc 文件夹中还会有一个 lib 文件夹。/odbc/lib 中会有“psqlodbca.la”，“psqlodbca.so”，“psqlodbcw.la”和“psqlodbcw.so”四个文件，将这四个文件拷贝到“/usr/local/lib”目录下。

(2) 将 lib 目录中的库拷贝到“/usr/local/lib”目 录下。

步骤 4 配置数据源。

(1) 配置 ODBC 驱动文件。

在“/usr/local/etc/odbcinst.ini”文件中追加以下内容。

```
[GaussMPP] Driver64=/usr/local/lib/psqlodbcw.so setup=/usr/local/lib/psqlodbcw.so
```

odbcinst.ini 文件中的配置参数说明如表 6-8 所示。

表 6-8 odbcinst.ini 文件配置参数

参数	描述	示例
[DriverName]	驱动器名称，对应数据源 DSN 中的驱动名。	[DRIVER_N]
Driver64	驱动动态库的路径。	Driver64=/usr/local/lib/ psqlodbcw.so
setup	驱动安装路径，与 Driver64 中动态库的路径一致。	setup=/usr/local/lib/ psqlodbcw.so

(2) 配置数据源文件。

在“/usr/local/etc/odbc.ini”文件中追加以下内容。

```
[MPPODBC]
Driver=GaussMPP Servername=10.145.130.26（数据库 Server IP） Database=postgres （数据库名） Username=gbase （数据库用户名） Password= （数据库用户密码） Port=5432 （数据库侦听端口）
Sslmode=allow
```

odbc.ini 文件配置参数说明如表 6-9 所示。

表 6-9 odbc.ini 文件配置参数

参数	描述	示例
[DSN]	数据源的名称。	[MPPODBC]
Driver	驱动名，对应 odbcinst.ini 中的 DriverName。	Driver=DRIVER_N
Servername	服务器的 IP 地址。可配置多个 IP 地址。	Servername=10.145.130.2 6
Database	要连接的数据库的名称。	Database=postgres
Username	数据库用户名称。	Username=gbase
Password	数据库用户密码。	Password= 说明 ODBC 驱动本身已经对内存密码进行过清理，以保证用户密码在连接后不会再在内存中保留。 但是如果配置了此参数，由于 UnixODBC 对数据源文件等进行缓存，可能导致密码长期保留在内存中。 推荐在应用程序连接时，将密码传递给相应 API，而非写在数据源配置文件中。同时连接成功后，应当及时清理保存密码的内存段。
Port	服务器的端口号。	Port=5432
Sslmode	开启 SSL 模式	Sslmode=allow
Debug	设置为 1 时，将会打印 psqlodbc 驱动的 mylog，日志生成目录为 /tmp/。设置为 0 时则不会生成。	Debug=1
UseServerSidePrepare	是否开启数据库端扩展查询协议。 可选值 0 或 1，默认为 1， 表示	UseServerSidePrepare=1

	打开扩展查询协议。	
UseBatchProtocol	是否开启批量查询协议（打开可提高 DML 性 能）；可选值 0 或者 1，默认为 1。 当此值为 0 时，不使用批量查询协议（主要用于与早期数据库版本通信兼容）。 当此值为 1，并且数据库 support_batch_bind 参数存在且为 on 时，将打开批量查询协议。	UseBatchProtocol=1
ForExtensionConnect or	这个开关控制着 savepoint 是否发送，savepoint 相关问题可以注意这个开关。	ForExtensionConnector=1
UnamedPrepStmtThr eshold	每次调用 SQLFreeHandle 释放 Stmt 时，ODBC 都会向 server 端发送一个 Deallocate plan_name 语句，业务中存在大量这类语句。为了减少这类语句的发送，我们将 stmt->plan_name 置空，从而使得数据库识别这个为 unamed stmt。增加这个参数对 unamed stmt 的阈值进行控制。	UnamedPrepStmtThresho ld=100
ConnectionExtraInfo	GUC 参数 connection_info（参见 connection_info）中显示驱动部署路径和进程属主用户的开关。	ConnectionExtraInfo=1 说明 默认值为 0。当设置为 1 时，ODBC 驱动会将当前驱动的部署路径、进程属主用户上报到数据库中，记录在 connection_info 参数（参见 connection_info）里；同时可以在 20.3.72 PG_STAT_ACTIVITY 中查询到。
BoolAsChar	设置为 Yes 是，Bools 值将会映射为 SQL_CHAR。如不设置将会映射为 SQL_BIT。	BoolsAsChar = Yes
RowVersioning	当尝试更新一行数据时，设置为	RowVersioning=Yes

	Yes 会允许应用检测数据有没有被其他用户进行修改。	
ShowSystemTables	驱动将会默认系统表格为普通 SQL 表格。	ShowSystemTables=Yes

其中关于 Sslmode 的选项的允许值，具体信息见下表：

表 6-10 Sslmode 的可选项及其描述

Sslmode	是否会启用 SSL 加密	描述
disable	否	不使用 SSL 安全连接。
allow	可能	如果数据库服务器要求使用，则可以使用 SSL 安全加密连接，但不验证数据库服务器的真实性。
prefer	可能	如果数据库支持，那么建议使用 SSL 安全加密连接， 但不验证数据库服务器的真实性。
require	是	必须使用 SSL 安全连接，但是只做了数据加密，而并不验证数据库服务器的真实性。
verify-ca	是	必须使用 SSL 安全连接，并且验证数据库是否具有可信证书机构签发的证书。
verify- full	是	必须使用 SSL 安全连接，在 verify-ca 的验证范围之外，同时验证数据库所在主机的主机名是否与证书内容一致。GBase 8c 不支持此模式。

步骤 5 （可选）生成 SSL 证书，具体请参见 7.1.6.1 证书生成。在服务端与客户端通过 ssl 方式连接的情况下，需要执行步骤 5 或步骤 6。非 ssl 方式连接情况下可以跳过。

步骤 6 （可选）替换 SSL 证书，具体请参见 7.1.6.2 证书替换。

步骤 7 SSL 模式：

声明如下环境变量，同时保证 client.key*系列文件为 600 权限：

```
退回根目录，创建.postgresql 目录，并将 root.crt，client.crt，client.key，client.key.cipher，client.key.rand，
client.req，server.crt，server.key，server.key.cipher，server.key.rand，server.req 放在此路径下。
Unix 系统下，server.crt、server.key 的权限设置必须禁止任何外部或组的访问，请执行如下命令实现这一点。chmod 0600 server.key
```

将 root.crt 以及 server 开头的证书相关文件全部拷贝进数据库 install/data 目录下（与 postgresql.conf 文件在同一路径）。
修改 postgresql.conf 文件：
ssl = on
ssl_cert_file = 'server.crt' ssl_key_file = 'server.key' ssl_ca_file = 'root.crt'
修改完参数后需重启数据库。
修改配置文件 odbc.ini 中的 sslmode 参数（require 或 verify-ca）。

步骤 8 配置数据库服务器。

(1) 以操作系统用户 gbase 登录数据库主节点。

(2) 执行如下命令增加对外提供服务的网卡 IP 或者主机名（英文逗号分隔），NODETYPE 指定节点类型，NodeName 为当前节点名称：

```
gs_guc reload [-Z NODETYPE] -N NodeName -I all -c "listen_addresses='XXX,XX'"
```

其中-Z 参数仅在分布式情况下需要指定。

在 DR(Direct Routing, LVS 的直接路由 DR 模式)模式中需要将虚拟 IP 地址(10.11.12.13)加入到服务器的侦听地址列表中。

listen_addresses 也可以配置为“*”或“0.0.0.0”，此配置下将侦听所有网卡，但 存在安全风险，不推荐用户使用，推荐用户按照需要配置 IP 或者主机名，打开侦听。

(3) 执行如下命令在数据库主节点配置文件中增加一条认证规则。例如，假设客户端 IP 地址为 10.11.12.13，即远程连接的机器的 IP 地址，远程连接主备式集群为例。

```
gs_guc reload -N all -I all -h "host all jack 10.11.12.13/32 sha256"
```

- -N all 表示 GBase 8c 中的所有主机。
- -I all 表示主机中的所有实例。
- -h 表示指定需要在“pg_hba.conf”增加的语句。
- all 表示允许客户端连接到任意的数据库。
- jack 表示连接数据库的用户。
- 10.11.12.13/32 表示只允许 IP 地址为 10.11.12.13 的主机连接。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
- sha256 表示连接时 jack 用户的密码使用 sha256 算法加密。

如果将 ODBC 客户端配置在和要连接的数据库主节点在同一台机器上，则可使用 local trust 认证方式，如下：

```
local all all trust
```

如果将ODBC客户端配置在和要连接的数据库主节点在不同机器上，则需要使用sha256认证方式，如下：

```
host all all xxx.xxx.xxx.xxx/32 sha256
```

(4) 重启数据库。

```
gha_ctl stop all -l http://<dcs_ip>:2379
gha_ctl start all -l http://<dcs_ip>:2379
```

步骤 9 在客户端配置环境变量。

```
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
```

步骤 10 执行如下命令使设置生效。

```
source ~/.bashrc
```

----结束

测试数据源配置

执行./isql -v MPPODBC（数据源名称）命令。

- 如果显示如下信息，表明配置正确，连接成功。

```
+---------------------------------------+
| Connected!
|
| sql-statement
| help [tablename]
| quit
|
+---------------------------------------+ SQL>
```

- 若显示 ERROR 信息，则表明配置错误。请检查上述配置是否正确。

常见问题处理

- [UnixODBC][Driver Manager]Can't open lib 'xxx/xxx/psqlodbcw.so' : file not found.

此问题的可能原因：

- odbcinst.ini 文件中配置的路径不正确

确认的方法：'ls'一下错误信息中的路径，以确保该 psqlodbcw.so 文件存在， 同时具有执行权限。

- psqlodbcw.so 的依赖库不存在，或者不在系统环境变量中

确认的办法：ldd 一下错误信息中的路径，如果是缺少 libodbc.so.1 等 UnixODBC 的库，那么按照“操作步骤”中的方法重新配置 UnixODBC，并确保它的安装路径下的 lib 目录添加到了 LD_LIBRARY_PATH 中；如果是缺少其他库，请将 ODBC 驱动包中的 lib 目录添加到 LD_LIBRARY_PATH 中。

- [UnixODBC]connect to server failed: no such file or directory

此问题可能的原因：

- 配置了错误的/不可达的数据库地址，或者端口

请检查数据源配置中的 Servername 及 Port 配置项。

- 服务器侦听不正确

如果确认 Servername 及 Port 配置正确，请根据“操作步骤”中数据库服务器的相关配置，确保数据库侦听了合适的网卡及端口。

- 防火墙及网闸设备
- 请确认防火墙设置，将数据库的通信端口添加到可信端口中。如果有网闸设备，请确认一下相关的设置。

- [unixODBC]The password-stored method is not supported.

此问题可能原因：

数据源中未配置 sslmode 配置项。

解决办法：

请配置该选项至 allow 或以上选项。此配置的更多信息，见表 6-10。

- Server common name "xxxx" does not match host name "xxxxx"

此问题的原因：使用了 SSL 加密的“verify-full”选项，驱动程序会验证证书中的主机名与实际部署数据库的主机名是否一致。

解决办法：碰到此问题可以使用“verify-ca”选项，不再校验主机名；或者重新生成一套与数据库所在主机名相同的 CA 证书。

- Driver's SQLAllocHandle on SQL_HANDLE_DBC failed

此问题的可能原因：可执行文件（比如 UnixODBC 的 isql，以下都以 isql 为例）与数据库驱动（psqlodbcw.so）依赖于不同的 odbc 的库版本：libodbc.so.1 或者 libodbc.so.2。此 问题可以通过如下方式确认：

```
ldd `which isql` | grep odbc ldd psqlodbcw.so | grep odbc
```

这时，如果输出的 libodbc.so 最后的后缀数字不同或者指向不同的磁盘物理文件， 那么基本就可以断定是此问题。isql 与 psqlodbcw.so 都会要求加载 libodbc.so，这时如果它们加载的是不同的物理文件，便会导致两套完全同名的函数列表，同时出现在同一个可见域里（UnixODBC 的 libodbc.so.*的函数导出列表完全一致）， 产生冲突，无法加载数据库驱动。

解决办法：确定一个要使用的 UnixODBC，然后卸载另外一个（比如卸载库版本号为.so.2 的 UnixODBC），然后将剩下的.so.1 的库，新建一个同名但是后缀为.so.2 的软链接，便可解决此问题。

- FATAL: Forbid remote connection with trust method!

由于安全原因，数据库主节点禁止 GBase 8c 内部其他节点无认证接入。

如果要在 GBase 8c 内部访问数据库主节点，请将 ODBC 程序部署在数据库主节点所在机器，服务器地址使用"127.0.0.1"。建议业务系统单独部署在 GBase 8c 外部，否则可能会影响数据库运行性能。

- [unixODBC][Driver Manager]Invalid attribute value

有可能是 unixODBC 的版本并非推荐版本，建议通过“odbcinst --version”命令排查环境中的 unixODBC 版本。

- authentication method 10 not supported.

使用开源客户端碰到此问题，可能原因：

数据库中存储的口令校验只存储了 SHA256 格式哈希，而开源客户端只识别 MD5 校验，双方校验方法不匹配报错。

说明

- 数据库并不存储用户口令，只存储用户口令的哈希码。
- 数据库当用户更新用户口令或者新建用户时，会同时存储两种格式的哈希码，这时将兼容开源的认证协议。
- 但是当老版本升级到新版本时，由于哈希的不可逆性，所以数据库无法还原用户口

令， 进而生成新格式的哈希，所以仍然只保留了 SHA256 格式的哈希，导致仍然无法使用 MD5 做口令认证。

- MD5 加密算法安全性低，存在安全风险，建议使用更安全的加密算法。
- 要解决该问题，可以更新用户口令（参见《GBase 8c V5_3.0.1_SQL 手册》 ALTER USER 章节）；或者新建一个用户（参见《GBase 8c V5_3.0.1_SQL 手册》CREATE USER 章节），赋于同等权限，使用新用户连接数据库。

- unsupported frontend protocol 3.51: server supports 1.0 to 3.0

目标数据库版本过低，或者目标数据库为开源数据库。请使用对应版本的数据库驱动连接目标数据库。

- FATAL: GSS authentication method is not allowed because XXXX user password is not disabled.

目标数据库主节点的 pg_hba.conf 里配置了当前客户端 IP 使用"gss"方式来做认证， 该认证算法不支持用作客户端的身份认证，请修改到"sha256"后再试。配置方法见步骤 8。

6.3.3 开发流程

图 6-3 ODBC 开发应用程序的流程

开发流程中涉及的 API

表 6-11 相关 API 说明

功能	API
申请句柄资源	8.2.3 SQLAllocHandle：申请句柄资源，可替代如下函数： 8.2.1 SQLAllocEnv：申请环境句柄 8.2.2 SQLAllocConnect：申请连接句柄 8.2.4 SQLAllocStmt：申请语句句柄
设置环境属性	8.2.21 SQLSetEnvAttr
设置连接属性	8.2.20 SQLSetConnectAttr
设置语句属性	8.2.22 SQLSetStmtAttr
连接数据源	8.2.8 SQLConnect
绑定缓冲区到结果集的列中	8.2.5 SQLBindCol
绑定 SQL 语句的参数标志和缓冲区	8.2.6 SQLBindParameter
查看最近一次操作错误信息	8.2.19 SQLGetDiagRec
为执行 SQL 语句做准备	8.2.17 SQLPrepare
执行一条准备好的 SQL 语句	8.2.11 SQLExecute
直接执行 SQL 语句	8.2.10 SQLExecDirect
结果集中取行集	8.2.12 SQLFetch
返回结果集中某一列的数据	8.2.18 SQLGetData
获取结果集中列的描述信息	8.2.7 SQLColAttribute
断开与数据源的连接	8.2.9 SQLDisconnect

释放句柄资源	8.2.15 SQLFreeHandle：释放句柄资源，可替代如下函数： 8.2.16 SQLFreeEnv：释放环境句柄 8.2.14 SQLFreeConnect：释放连接句柄 8.2.13 SQLFreeStmt：释放语句句柄

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， 同时如果其中有一个语句失败，那么整个请求都将会被回滚。

警告

ODBC 为应用程序与数据库的中心层，负责把应用程序发出的 SQL 指令传到数据库当中，自身并不解析 SQL 语法。故在应用程序中写入带有保密信息的 SQL 语句时（如明文密码），保密信息会被暴露在驱动日志中。

6.3.4 示例：常用功能和批量绑定

常用功能示例代码

```
// 此示例演示如何通过 ODBC 方式获取 GBase 8c 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*) "gaussdb", 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>
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);
if (!SQL_SUCCEEDED(retcode)) { printf("SQLAllocHandle(SQL_HANDLE_STMT) failed");
return;
}
// Prepare Statement
retcode = SQLPrepare(hstmt, (SQLCHAR*) sql, SQL_NTS); sprintf((char*)loginfo,
"SQLPrepare log: %s", (char*)sql);
if (!SQL_SUCCEEDED(retcode)) {
```

```
printf("SQLPrepare(hstmt, (SQLCHAR*) sql, SQL_NTS) failed"); return;
}
// Execute Statement
retcode = SQLExecute(hstmt);
sprintf((char*)loginfo, "SQLExecute stmt log: %s", (char*)sql);
if (!SQL_SUCCEEDED(retcode)) { printf("SQLExecute(hstmt) failed"); return;
}
// Free Handle
retcode = SQLFreeHandle(SQL_HANDLE_STMT, hstmt); sprintf((char*)loginfo,
"SQLFreeHandle stmt log: %s", (char*)sql);
if (!SQL_SUCCEEDED(retcode)) { printf("SQLFreeHandle(SQL_HANDLE_STMT, hstmt)
failed"); return;
}
}
int main ()
{
SQLHENV henv = SQL_NULL_HENV; SQLHDBC hdbc = SQL_NULL_HDBC;
int  batchCount = 1000; //  批量绑定的数据量
SQLLEN rowsCount = 0;
int  ignoreCount = 0; //  批量绑定的数据中，不要入库的数据量
SQLRETURN retcode; SQLCHAR     dsn[1024] = {'\0'};
SQLCHAR    loginfo[2048];
do
{
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); if
(!SQL_SUCCEEDED(retcode)) {
printf("SQLAllocHandle failed"); goto exit;
}
// Set ODBC Verion
retcode = SQLSetEnvAttr(henv, SQL_ATTR_ODBC_VERSION,
(SQLPOINTER*)SQL_OV_ODBC3, 0);
if (!SQL_SUCCEEDED(retcode)) { printf("SQLSetEnvAttr failed"); goto exit;
}
// Allocate Connection
retcode = SQLAllocHandle(SQL_HANDLE_DBC, henv, &hdbc);
if (!SQL_SUCCEEDED(retcode)) { printf("SQLAllocHandle failed"); goto exit;
```

```
}
// Set Login Timeout
retcode = SQLSetConnectAttr(hdbc, SQL_LOGIN_TIMEOUT, (SQLPOINTER)5, 0);
if (!SQL_SUCCEEDED(retcode)) { printf("SQLSetConnectAttr failed"); goto exit;
}
// Set Auto Commit
retcode = SQLSetConnectAttr(hdbc, SQL_ATTR_AUTOCOMMIT,
(SQLPOINTER)(1), 0);
if (!SQL_SUCCEEDED(retcode)) { printf("SQLSetConnectAttr failed"); goto exit;
}
// Connect to DSN
// gaussdb 替换成用户所使用的数据源名称 sprintf(loginfo, "SQLConnect(DSN:%s)", dsn);
retcode = SQLConnect(hdbc, (SQLCHAR*) "gaussdb", SQL_NTS, (SQLCHAR*) NULL, 0,
NULL, 0);
if (!SQL_SUCCEEDED(retcode)) { printf("SQLConnect failed"); goto exit;
}
// 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);
if (!SQL_SUCCEEDED(retcode)) { printf("SQLAllocHandle failed"); goto exit;
}
// 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);
if (!SQL_SUCCEEDED(retcode)) { printf("SQLPrepare failed"); goto exit;
}
retcode = SQLSetStmtAttr(hstmtinesrt, SQL_ATTR_PARAMSET_SIZE,
(SQLPOINTER)batchCount, sizeof(batchCount));
if (!SQL_SUCCEEDED(retcode)) { printf("SQLSetStmtAttr failed"); goto exit;
}
retcode = SQLBindParameter(hstmtinesrt, 1, SQL_PARAM_INPUT, SQL_C_SLONG,
SQL_INTEGER, sizeof(ids[0]), 0,&(ids[0]), 0, bufLenIds);
if (!SQL_SUCCEEDED(retcode)) { printf("SQLBindParameter failed"); goto exit;
}
retcode = SQLBindParameter(hstmtinesrt, 2, SQL_PARAM_INPUT, SQL_C_CHAR,
SQL_CHAR, 50, 50,
cols, 50, bufLenCols);
if (!SQL_SUCCEEDED(retcode)) { printf("SQLBindParameter failed"); goto exit;
}
retcode = SQLSetStmtAttr(hstmtinesrt, SQL_ATTR_PARAMS_PROCESSED_PTR,
(SQLPOINTER)&process, sizeof(process));
if (!SQL_SUCCEEDED(retcode)) { printf("SQLSetStmtAttr failed"); goto exit;
}
retcode = SQLSetStmtAttr(hstmtinesrt, SQL_ATTR_PARAM_STATUS_PTR,
(SQLPOINTER)statusptr, sizeof(statusptr[0]) * batchCount);
if (!SQL_SUCCEEDED(retcode)) { printf("SQLSetStmtAttr failed"); goto exit;
}
```

```
retcode = SQLSetStmtAttr(hstmtinesrt, SQL_ATTR_PARAM_OPERATION_PTR,
(SQLPOINTER)operptr, sizeof(operptr[0]) * batchCount);
if (!SQL_SUCCEEDED(retcode)) { printf("SQLSetStmtAttr failed"); goto exit;
}
retcode = SQLExecute(hstmtinesrt);
sprintf((char*)loginfo, "SQLExecute stmt log: %s", (char*)sql);
if (!SQL_SUCCEEDED(retcode)) { printf("SQLExecute(hstmtinesrt) failed"); goto exit;
retcode = SQLRowCount(hstmtinesrt, &rowsCount); if (!SQL_SUCCEEDED(retcode)) {
printf("SQLRowCount failed");
goto exit;
}
if (rowsCount != (batchCount - ignoreCount))
{
sprintf(loginfo, "(batchCount - ignoreCount)(%d) != rowsCount(%d)", (batchCount -
ignoreCount), rowsCount);
if (!SQL_SUCCEEDED(retcode)) { printf("SQLExecute failed"); goto exit;
}
}
else
{
sprintf(loginfo, "(batchCount - ignoreCount)(%d) == rowsCount(%d)", (batchCount -
ignoreCount), rowsCount);
if (!SQL_SUCCEEDED(retcode)) { printf("SQLExecute failed"); goto exit;
}
}
// check row number returned if (rowsCount != process)
{
sprintf(loginfo, "process(%d) != rowsCount(%d)", process, rowsCount);
if (!SQL_SUCCEEDED(retcode)) { printf("SQLExecute failed"); goto exit;
}
}
else
{
sprintf(loginfo, "process(%d) == rowsCount(%d)", process, rowsCount);
if (!SQL_SUCCEEDED(retcode)) { printf("SQLExecute failed"); goto exit;
}
}
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]);
if (!SQL_SUCCEEDED(retcode)) { printf("SQLExecute failed"); goto exit;
}
}
}
else if (statusptr[i] != SQL_PARAM_SUCCESS)
{
sprintf(loginfo, "statusptr[%d](%d) != SQL_PARAM_SUCCESS", i, statusptr[i]);
if (!SQL_SUCCEEDED(retcode)) { printf("SQLExecute failed"); goto exit;
}
}
}
retcode = SQLFreeHandle(SQL_HANDLE_STMT, hstmtinesrt); sprintf((char*)loginfo,
"SQLFreeHandle hstmtinesrt");
if (!SQL_SUCCEEDED(retcode)) { printf("SQLFreeHandle failed"); goto exit;
}
}
exit:
(void) 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;
}
```

6.3.5 典型应用场景配置

日志诊断场景

ODBC 日志分为 unixODBC 驱动管理器日志和 psqlODBC 驱动端日志。前者可以用于追溯应用程序 API 的执行是否成功，后者是底层实现过程中的一些 DFX 日志，用来帮助定位问题。

```
unixODBC 日志需要在 odbcinst.ini 文件中配置：
[ODBC]
Trace=Yes TraceFile=/path/to/odbctrace.log
[GaussMPP] Driver64=/usr/local/lib/psqlodbcw.so setup=/usr/local/lib/psqlodbcw.so
```

psqlODBC 日志只需要在 odbc.ini 加上：

```
[gaussdb] Driver=GaussMPP
Servername=10.10.0.13（数据库 Server IP）
...
Debug=1（打开驱动端 debug 日志）
```

说明

unixODBC 日志将会生成在 TraceFile 配置的路径下，psqlODBC 会在系统/tmp/下生成 mylog_xxx.log。

高性能场景

进行大量数据插入时，建议如下：

- 需要设置批量绑定：odbc.ini 配置文件中设置 UseBatchProtocol=1、数据库设置
- support_batch_bind=on。
- ODBC 程序绑定类型要和数据库中类型一致。
- 客户端字符集和数据库字符集一致。
- 事务改成手动提交。

odbc.ini 配置文件：

```
[gaussdb] Driver=GaussMPP
Servername=10.10.0.13（数据库 Server IP）
...
UseBatchProtocol=1 （默认打开）
ConnSettings=set client_encoding=UTF8 （设置客户端字符编码，保证和 server 端一致）
```

绑定类型用例：

```
#include <stdio.h> #include <stdlib.h> #include <sql.h> #include <sqlext.h> #include
<string.h> #include <sys/time.h>
#define MESSAGE_BUFFER_LEN 128 SQLHANDLE h_env = NULL; SQLHANDLE
h_conn = NULL; SQLHANDLE h_stmt = NULL;
void print_error()
{
SQLCHAR  Sqlstate[SQL_SQLSTATE_SIZE+1];
SQLINTEGER NativeError;
SQLCHAR  MessageText[MESSAGE_BUFFER_LEN];
SQLSMALLINT TextLength; SQLRETURN ret = SQL_ERROR;
```

```
ret = SQLGetDiagRec(SQL_HANDLE_STMT, h_stmt, 1, Sqlstate, &NativeError, MessageText,
MESSAGE_BUFFER_LEN, &TextLength);
if ( SQL_SUCCESS == ret)
{
printf("\n STMT ERROR-%05d %s", NativeError, MessageText); return;
}
ret = SQLGetDiagRec(SQL_HANDLE_DBC, h_conn, 1, Sqlstate, &NativeError, MessageText,
MESSAGE_BUFFER_LEN, &TextLength);
if ( SQL_SUCCESS == ret)
{
printf("\n CONN ERROR-%05d %s", NativeError, MessageText); return;
}
ret = SQLGetDiagRec(SQL_HANDLE_ENV, h_env, 1, Sqlstate, &NativeError, MessageText,
MESSAGE_BUFFER_LEN, &TextLength);
if ( SQL_SUCCESS == ret)
{
printf("\n ENV ERROR-%05d %s", NativeError, MessageText); return;
}
return;
}
/* 期盼函数返回 SQL_SUCCESS */
#define RETURN_IF_NOT_SUCCESS(func) \
{\
SQLRETURN ret_value = (func);\ if (SQL_SUCCESS != ret_value)\
{\
print_error();\
printf("\n failed line = %u: expect SQL_SUCCESS, but ret = %d", LINE , ret_value);\ return
SQL_ERROR; \
}\
}
/* 期盼函数返回 SQL_SUCCESS */
#define RETURN_IF_NOT_SUCCESS_I(i, func) \
{\
SQLRETURN ret_value = (func);\ if (SQL_SUCCESS != ret_value)\
{\
print_error();\
printf("\n failed line = %u (i=%d): : expect SQL_SUCCESS, but ret = %d", LINE , (i),
ret_value);\ return SQL_ERROR; \
}\
}
```

```
/* 期盼函数返回 SQL_SUCCESS_WITH_INFO */ #define
RETURN_IF_NOT_SUCCESS_INFO(func) \
{\
SQLRETURN ret_value = (func);\
if (SQL_SUCCESS_WITH_INFO != ret_value)\
{\
print_error();\
printf("\n failed line = %u: expect SQL_SUCCESS_WITH_INFO, but ret = %d", LINE ,
ret_value);\ return SQL_ERROR; \
}\
}
/* 期盼数值相等 */
#define RETURN_IF_NOT(expect, value) \ if ((expect) != (value))\
{\
printf("\n failed line = %u: expect = %u, but value = %u", LINE , (expect), (value)); \ return
SQL_ERROR;\
}
/* 期盼字符串相同 */
#define RETURN_IF_NOT_STRCMP_I(i, expect, value) \ if (( NULL == (expect) ) || (NULL
== (value)))\
{\
printf("\n failed line = %u (i=%u): input NULL pointer !", LINE , (i)); \ return SQL_ERROR; \
}\
else if (0 != strcmp((expect), (value)))\
{\
printf("\n failed line = %u (i=%u): expect = %s, but value = %s", LINE , (i), (expect), (value)); \
return SQL_ERROR;\
}
// prepare + execute SQL 语句 int execute_cmd(SQLCHAR *sql)
{
if ( NULL == sql )
{
return SQL_ERROR;
}
if ( SQL_SUCCESS != SQLPrepare(h_stmt, sql, SQL_NTS))
{
return SQL_ERROR;
}
if ( SQL_SUCCESS != SQLExecute(h_stmt))
{
return SQL_ERROR;
```

```
}
return SQL_SUCCESS;
}
// execute + commit 句柄
int commit_exec()
{
if ( SQL_SUCCESS != SQLExecute(h_stmt))
{
return SQL_ERROR;
}
// 手动提交
if ( SQL_SUCCESS != SQLEndTran(SQL_HANDLE_DBC, h_conn, SQL_COMMIT))
{
return SQL_ERROR;
}
return SQL_SUCCESS;
}
int begin_unit_test()
{
SQLINTEGER ret;
/* 申请环境句柄 */
ret = SQLAllocHandle(SQL_HANDLE_ENV, SQL_NULL_HANDLE, &h_env); if
((SQL_SUCCESS != ret) && (SQL_SUCCESS_WITH_INFO != ret))
{
printf("\n begin_unit_test::SQLAllocHandle SQL_HANDLE_ENV failed ! ret = %d", ret);
return SQL_ERROR;
}
/* 进行连接前必须要先设置版本号 */
if (SQL_SUCCESS != SQLSetEnvAttr(h_env, SQL_ATTR_ODBC_VERSION,
(SQLPOINTER)SQL_OV_ODBC3, 0))
{
print_error();
printf("\n begin_unit_test::SQLSetEnvAttr SQL_ATTR_ODBC_VERSION failed ! ret = %d",
ret); SQLFreeHandle(SQL_HANDLE_ENV, h_env);
return SQL_ERROR;
}
/* 申请连接句柄 */
ret = SQLAllocHandle(SQL_HANDLE_DBC, h_env, &h_conn); if (SQL_SUCCESS != ret)
{
print_error();
```

```
printf("\n begin_unit_test::SQLAllocHandle SQL_HANDLE_DBC failed ! ret = %d", ret);
SQLFreeHandle(SQL_HANDLE_ENV, h_env);
return SQL_ERROR;
}
/* 建立连接 */
ret = SQLConnect(h_conn, (SQLCHAR*) "gaussdb", SQL_NTS,
(SQLCHAR*) NULL, 0, NULL, 0); if (SQL_SUCCESS != ret)
{
print_error();
printf("\n begin_unit_test::SQLConnect failed ! ret = %d", ret);
SQLFreeHandle(SQL_HANDLE_DBC, h_conn); SQLFreeHandle(SQL_HANDLE_ENV,
h_env);
return SQL_ERROR;
}
/* 申请语句句柄 */
ret = SQLAllocHandle(SQL_HANDLE_STMT, h_conn, &h_stmt); if (SQL_SUCCESS != ret)
{
print_error();
printf("\n begin_unit_test::SQLAllocHandle SQL_HANDLE_STMT failed ! ret = %d", ret);
SQLFreeHandle(SQL_HANDLE_DBC, h_conn);
SQLFreeHandle(SQL_HANDLE_ENV, h_env); return SQL_ERROR;
}
return SQL_SUCCESS;
}
void end_unit_test()
{
/* 释放语句句柄 */ if (NULL != h_stmt)
{
SQLFreeHandle(SQL_HANDLE_STMT, h_stmt);
}
/* 释放连接句柄 */ if (NULL != h_conn)
{
SQLDisconnect(h_conn); SQLFreeHandle(SQL_HANDLE_DBC, h_conn);
}
/* 释放环境句柄 */ if (NULL != h_env)
{
SQLFreeHandle(SQL_HANDLE_ENV, h_env);
}
return;
}
int main()
```

```
{
// begin test
if (begin_unit_test() != SQL_SUCCESS)
{
printf("\n begin_test_unit failed."); return SQL_ERROR;
}
// 句柄配置同前面用例
int  i = 0;
SQLCHAR* sql_drop = "drop table if exists test_bindnumber_001"; SQLCHAR* sql_create =
"create table test_bindnumber_001("
"f4 number, f5 number(10, 2)" ")";
SQLCHAR* sql_insert = "insert into test_bindnumber_001 values(?, ?)"; SQLCHAR*
sql_select = "select * from test_bindnumber_001";
SQLLEN      RowCount; SQL_NUMERIC_STRUCT st_number;
SQLCHAR   getValue[2][MESSAGE_BUFFER_LEN];
/* step 1. 建 表 */ RETURN_IF_NOT_SUCCESS(execute_cmd(sql_drop));
RETURN_IF_NOT_SUCCESS(execute_cmd(sql_create));
/* step 2.1 通 过 SQL_NUMERIC_STRUCT 结 构 绑 定 参 数 */
RETURN_IF_NOT_SUCCESS(SQLPrepare(h_stmt, sql_insert, SQL_NTS));
//第一行: 1234.5678
memset(st_number.val, 0, SQL_MAX_NUMERIC_LEN);
st_number.precision = 8;
st_number.scale = 4;
st_number.sign = 1; st_number.val[0] = 0x4E; st_number.val[1] = 0x61; st_number.val[2] =
0xBC;
RETURN_IF_NOT_SUCCESS(SQLBindParameter(h_stmt, 1, SQL_PARAM_INPUT,
SQL_C_NUMERIC, SQL_NUMERIC, sizeof(SQL_NUMERIC_STRUCT), 4, &st_number, 0,
NULL));
RETURN_IF_NOT_SUCCESS(SQLBindParameter(h_stmt, 2, SQL_PARAM_INPUT,
SQL_C_NUMERIC, SQL_NUMERIC, sizeof(SQL_NUMERIC_STRUCT), 4, &st_number, 0,
NULL));
// 关闭自动提交
SQLSetConnectAttr(h_conn, SQL_ATTR_AUTOCOMMIT,
(SQLPOINTER)SQL_AUTOCOMMIT_OFF, 0);
RETURN_IF_NOT_SUCCESS(commit_exec());
RETURN_IF_NOT_SUCCESS(SQLRowCount(h_stmt, &RowCount)); RETURN_IF_NOT(1,
RowCount);
//第二行: 12345678
memset(st_number.val, 0, SQL_MAX_NUMERIC_LEN);
st_number.precision = 8;
st_number.scale = 0;
```

```
st_number.sign = 1; st_number.val[0] = 0x4E; st_number.val[1] = 0x61; st_number.val[2] =
0xBC;
RETURN_IF_NOT_SUCCESS(SQLBindParameter(h_stmt, 1, SQL_PARAM_INPUT,
SQL_C_NUMERIC, SQL_NUMERIC, sizeof(SQL_NUMERIC_STRUCT), 0, &st_number, 0,
NULL));
RETURN_IF_NOT_SUCCESS(SQLBindParameter(h_stmt, 2, SQL_PARAM_INPUT,
SQL_C_NUMERIC, SQL_NUMERIC, sizeof(SQL_NUMERIC_STRUCT), 0, &st_number, 0,
NULL));
RETURN_IF_NOT_SUCCESS(commit_exec());
RETURN_IF_NOT_SUCCESS(SQLRowCount(h_stmt, &RowCount)); RETURN_IF_NOT(1,
RowCount);
//第三行: 12345678
memset(st_number.val, 0, SQL_MAX_NUMERIC_LEN);
st_number.precision = 0;
st_number.scale = 4;
st_number.sign = 1; st_number.val[0] = 0x4E; st_number.val[1] = 0x61; st_number.val[2] =
0xBC;
RETURN_IF_NOT_SUCCESS(SQLBindParameter(h_stmt, 1, SQL_PARAM_INPUT,
SQL_C_NUMERIC, SQL_NUMERIC, sizeof(SQL_NUMERIC_STRUCT), 4, &st_number, 0,
NULL));
RETURN_IF_NOT_SUCCESS(SQLBindParameter(h_stmt, 2, SQL_PARAM_INPUT,
SQL_C_NUMERIC, SQL_NUMERIC, sizeof(SQL_NUMERIC_STRUCT), 4, &st_number, 0,
NULL));
RETURN_IF_NOT_SUCCESS(commit_exec());
RETURN_IF_NOT_SUCCESS(SQLRowCount(h_stmt, &RowCount)); RETURN_IF_NOT(1,
RowCount);
/* step 2.2 第 四 行 通 过 SQL_C_CHAR 字 符 串 绑 定 参 数 */
RETURN_IF_NOT_SUCCESS(SQLPrepare(h_stmt, sql_insert, SQL_NTS)); SQLCHAR*
szNumber = "1234.5678";
RETURN_IF_NOT_SUCCESS(SQLBindParameter(h_stmt, 1, SQL_PARAM_INPUT,
SQL_C_CHAR, SQL_NUMERIC, strlen(szNumber), 0, szNumber, 0, NULL));
RETURN_IF_NOT_SUCCESS(SQLBindParameter(h_stmt, 2, SQL_PARAM_INPUT,
SQL_C_CHAR, SQL_NUMERIC, strlen(szNumber), 0, szNumber, 0, NULL));
RETURN_IF_NOT_SUCCESS(commit_exec());
RETURN_IF_NOT_SUCCESS(SQLRowCount(h_stmt, &RowCount)); RETURN_IF_NOT(1,
RowCount);
/* step 2.3 第 五 行 通 过 SQL_C_FLOAT 绑 定 参 数 */
RETURN_IF_NOT_SUCCESS(SQLPrepare(h_stmt, sql_insert, SQL_NTS)); SQLREAL
fNumber = 1234.5678;
RETURN_IF_NOT_SUCCESS(SQLBindParameter(h_stmt, 1, SQL_PARAM_INPUT,
SQL_C_FLOAT, SQL_NUMERIC, sizeof(fNumber), 4, &fNumber, 0, NULL));
```

```
RETURN_IF_NOT_SUCCESS(SQLBindParameter(h_stmt, 2, SQL_PARAM_INPUT,
SQL_C_FLOAT, SQL_NUMERIC, sizeof(fNumber), 4, &fNumber, 0, NULL));
RETURN_IF_NOT_SUCCESS(commit_exec());
RETURN_IF_NOT_SUCCESS(SQLRowCount(h_stmt, &RowCount)); RETURN_IF_NOT(1,
RowCount);
/* step 2.4 第 六 行 通 过 SQL_C_DOUBLE 绑 定 参 数 */
RETURN_IF_NOT_SUCCESS(SQLPrepare(h_stmt, sql_insert, SQL_NTS)); SQLDOUBLE
dNumber = 1234.5678;
RETURN_IF_NOT_SUCCESS(SQLBindParameter(h_stmt, 1, SQL_PARAM_INPUT,
SQL_C_DOUBLE, SQL_NUMERIC, sizeof(dNumber), 4, &dNumber, 0, NULL));
RETURN_IF_NOT_SUCCESS(SQLBindParameter(h_stmt, 2, SQL_PARAM_INPUT,
SQL_C_DOUBLE, SQL_NUMERIC, sizeof(dNumber), 4, &dNumber, 0, NULL));
RETURN_IF_NOT_SUCCESS(commit_exec());
RETURN_IF_NOT_SUCCESS(SQLRowCount(h_stmt, &RowCount)); RETURN_IF_NOT(1,
RowCount);
SQLBIGINT bNumber1 = 0xFFFFFFFFFFFFFFFF; SQLBIGINT bNumber2 = 12345;
/* step 2.5 第 七 行 通 过 SQL_C_SBIGINT 绑 定 参 数 */
RETURN_IF_NOT_SUCCESS(SQLPrepare(h_stmt, sql_insert, SQL_NTS));
RETURN_IF_NOT_SUCCESS(SQLBindParameter(h_stmt, 1, SQL_PARAM_INPUT,
SQL_C_SBIGINT, SQL_NUMERIC, sizeof(bNumber1), 4, &bNumber1, 0, NULL));
RETURN_IF_NOT_SUCCESS(SQLBindParameter(h_stmt, 2, SQL_PARAM_INPUT,
SQL_C_SBIGINT, SQL_NUMERIC, sizeof(bNumber2), 4, &bNumber2, 0, NULL));
RETURN_IF_NOT_SUCCESS(commit_exec());
RETURN_IF_NOT_SUCCESS(SQLRowCount(h_stmt, &RowCount)); RETURN_IF_NOT(1,
RowCount);
/* step 2.6 第 八 行 通 过 SQL_C_UBIGINT 绑 定 参 数 */
RETURN_IF_NOT_SUCCESS(SQLPrepare(h_stmt, sql_insert, SQL_NTS));
RETURN_IF_NOT_SUCCESS(SQLBindParameter(h_stmt,  1,  SQL_PARAM_INPUT,
SQL_C_UBIGINT,
SQL_NUMERIC, sizeof(bNumber1), 4, &bNumber1, 0, NULL));
RETURN_IF_NOT_SUCCESS(SQLBindParameter(h_stmt,  2,  SQL_PARAM_INPUT,
SQL_C_UBIGINT,
SQL_NUMERIC, sizeof(bNumber2), 4, &bNumber2, 0, NULL));
RETURN_IF_NOT_SUCCESS(commit_exec());
RETURN_IF_NO_UCCESS(SQLRowCount(h_stmt, &RowCount)); RETURN_IF_NOT(1,
RowCount);
SQLLEN lNumber1 = 0xFFFFFFFFFFFFFFFF; SQLLEN lNumber2 = 12345;
/* step 2.7 第 九 行 通 过 SQL_C_LONG 绑 定 参 数 */
RETURN_IF_NOT_SUCCESS(SQLPrepare(h_stmt, sql_insert, SQL_NTS));
RETURN_IF_NOT_SUCCESS(SQLBindParameter(h_stmt,  1,  SQL_PARAM_INPUT,
SQL_C_LONG,
```

```
SQL_NUMERIC, sizeof(lNumber1), 0, &lNumber1, 0, NULL));
RETURN_IF_NOT_SUCCESS(SQLBindParameter(h_stmt,  2,  SQL_PARAM_INPUT,
SQL_C_LONG,
SQL_NUMERIC, sizeof(lNumber2), 0, &lNumber2, 0, NULL));
RETURN_IF_NOT_SUCCESS(commit_exec());
RETURN_IF_NOT_SUCCESS(SQLRowCount(h_stmt, &RowCount)); RETURN_IF_NOT(1,
RowCount);
/* step 2.8 第 十 行 通 过 SQL_C_ULONG 绑 定 参 数 */
RETURN_IF_NOT_SUCCESS(SQLPrepare(h_stmt, sql_insert, SQL_NTS));
RETURN_IF_NOT_SUCCESS(SQLBindParameter(h_stmt,  1,  SQL_PARAM_INPUT,
SQL_C_ULONG,
SQL_NUMERIC, sizeof(lNumber1), 0, &lNumber1, 0, NULL));
RETURN_IF_NOT_SUCCESS(SQLBindParameter(h_stmt,  2,  SQL_PARAM_INPUT,
SQL_C_ULONG,
SQL_NUMERIC, sizeof(lNumber2), 0, &lNumber2, 0, NULL));
RETURN_IF_NOT_SUCCESS(commit_exec());
RETURN_IF_NOT_SUCCESS(SQLRowCount(h_stmt, &RowCount)); RETURN_IF_NOT(1,
RowCount);
SQLSMALLINT sNumber = 0xFFFF;
/* step 2.9 第 十 一 行 通 过 SQL_C_SHORT 绑 定 参 数 */
RETURN_IF_NOT_SUCCESS(SQLPrepare(h_stmt, sql_insert, SQL_NTS));
RETURN_IF_NOT_SUCCESS(SQLBindParameter(h_stmt,  1,  SQL_PARAM_INPUT,
SQL_C_SHORT,
SQL_NUMERIC, sizeof(sNumber), 0, &sNumber, 0, NULL));
RETURN_IF_NOT_SUCCESS(SQLBindParameter(h_stmt,  2,  SQL_PARAM_INPUT,
SQL_C_SHORT,
SQL_NUMERIC, sizeof(sNumber), 0, &sNumber, 0, NULL));
RETURN_IF_NOT_SUCCESS(commit_exec());
RETURN_IF_NOT_SUCCESS(SQLRowCount(h_stmt, &RowCount)); RETURN_IF_NOT(1,
RowCount);
/* step 2.10 第 十 二 行 通 过 SQL_C_USHORT 绑 定 参 数 */
RETURN_IF_NOT_SUCCESS(SQLPrepare(h_stmt, sql_insert, SQL_NTS));
RETURN_IF_NOT_SUCCESS(SQLBindParameter(h_stmt,  1,  SQL_PARAM_INPUT,
SQL_C_USHORT,
SQL_NUMERIC, sizeof(sNumber), 0, &sNumber, 0, NULL));
RETURN_IF_NOT_SUCCESS(SQLBindParameter(h_stmt,  2,  SQL_PARAM_INPUT,
SQL_C_USHORT,
SQL_NUMERIC, sizeof(sNumber), 0, &sNumber, 0, NULL));
RETURN_IF_NOT_SUCCESS(commit_exec());
RETURN_IF_NOT_SUCCESS(SQLRowCount(h_stmt, &RowCount)); RETURN_IF_NOT(1,
RowCount);
```

```
SQLCHAR cNumber = 0xFF;
/* step 2.11 第 十 三 行 通 过 SQL_C_TINYINT 绑 定 参 数 */
RETURN_IF_NOT_SUCCESS(SQLPrepare(h_stmt, sql_insert, SQL_NTS));
RETURN_IF_NOT_SUCCESS(SQLBindParameter(h_stmt, 1, SQL_PARAM_INPUT,
SQL_C_TINYINT,
SQL_NUMERIC, sizeof(cNumber), 0, &cNumber, 0, NULL));
RETURN_IF_NOT_SUCCESS(SQLBindParameter(h_stmt, 2, SQL_PARAM_INPUT,
SQL_C_TINYINT, SQL_NUMERIC, sizeof(cNumber), 0, &cNumber, 0, NULL));
RETURN_IF_NOT_SUCCESS(commit_exec());
RETURN_IF_NOT_SUCCESS(SQLRowCount(h_stmt, &RowCount)); RETURN_IF_NOT(1,
RowCount);
/* step 2.12 第 十 四 行 通 过 SQL_C_UTINYINT 绑 定 参 数 */
RETURN_IF_NOT_SUCCESS(SQLPrepare(h_stmt, sql_insert, SQL_NTS));
RETURN_IF_NOT_SUCCESS(SQLBindParameter(h_stmt,  1,  SQL_PARAM_INPUT,
SQL_C_UTINYINT,
SQL_NUMERIC, sizeof(cNumber), 0, &cNumber, 0, NULL));
RETURN_IF_NOT_SUCCESS(SQLBindParameter(h_stmt,  2,  SQL_PARAM_INPUT,
SQL_C_UTINYINT,
SQL_NUMERIC, sizeof(cNumber), 0, &cNumber, 0, NULL));
RETURN_IF_NOT_SUCCESS(commit_exec());
RETURN_IF_NOT_SUCCESS(SQLRowCount(h_stmt, &RowCount)); RETURN_IF_NOT(1,
RowCount);
/* 用字符串类型统一进行期盼 */
SQLCHAR*      expectValue[14][2] = {{"1234.5678",    "1234.57"},
{"12345678","12345678"},
{"0", "0"},
{"1234.5678","1234.57"},
{"1234.5677","1234.57"},
{"1234.5678","1234.57"},
{"-1","12345"},
{"-1","12345"},
{"4294967295","12345"},
{"18446744073709551615", "12345"},
RETURN_IF_NOT_SUCCESS(execute_cmd(sql_select)); while ( SQL_NO_DATA !=
SQLFetch(h_stmt))
{
RETURN_IF_NOT_SUCCESS_I(i, SQLGetData(h_stmt, 1, SQL_C_CHAR, &getValue[0],
MESSAGE_BUFFER_LEN, NULL));
RETURN_IF_NOT_SUCCESS_I(i, SQLGetData(h_stmt, 2, SQL_C_CHAR, &getValue[1],
MESSAGE_BUFFER_LEN, NULL));
//RETURN_IF_NOT_STRCMP_I(i, expectValue[i][0], getValue[0]);
```

```
//RETURN_IF_NOT_STRCMP_I(i, expectValue[i][1], getValue[1]); i++;
}
RETURN_IF_NOT_SUCCESS(SQLRowCount(h_stmt, &RowCount)); RETURN_IF_NOT(i,
RowCount);
SQLCloseCursor(h_stmt);
/* step final. 删 除 表 还 原 环 境 */
RETURN_IF_NOT_SUCCESS(execute_cmd(sql_drop));
end_unit_test();
}
```

说明

上述用例中定义了 number 列，调用 SQLBindParameter 接口时，绑定 SQL_NUMERIC 会比 SQL_LONG 性能高一些。因为如果是 char，在数据库服务端插入数据时需要进行数据类型转换， 从而引发性能瓶颈。

6.3.6 ODBC 接口参考

请参见 8.2 ODBC。

6.4 基于 libpq 开发

libpq 是 GBase 8c 的 C 应用程序接口。libpq 是一套允许客户程序向服务器服务进程发送查询并且获得查询返回的库函数。同时也是其他几个 GBase 8c 应用接口下面的引擎，如 ODBC 等依赖的库文件。本章给出两个示例，显示如何利用 libpq 编写代码。

6.4.1 libpq 使用依赖的头文件

使用 libpq 的前端程序必须包括头文件 libpq-fe.h，并且必须与 libpq 库链接。

6.4.2 开发流程

编译并且链接一个 libpq 的源程序，需要做下面的一些事情：

(1) 解压相应的发布包文件。其中 include 文件夹下的头文件为所需的头文件，lib 文件夹中为所需的 libpq 库文件。

说明

除 libpq-fe.h 外，include 文件夹下默认还存在头文件 postgres_ext.h，gs_thread.h，gs_threadlocal.h，这三个头文件是 libpq-fe.h 的依赖文件。

(2) 包含 libpq-fe.h 头文件：

```
#include <libpq-fe.h>
```

(3) 通过-I directory 选项，提供头文件的安装位置（有些时候编译器会查找缺省的目录，因此可以忽略这些选项）。如：

```
gcc -I (头文件所在目录) -L (libpq 库所在目录) testprog.c -lpq
```

(4) 如果要使用 makefile 命令，向 CPPFLAGS、LDFLAGS、LIBS 变量中增加如下选项：

```
CPPFLAGS += -I (头文件所在目录) LDFLAGS += -L (libpq 库所在目录) LIBS += -lpq
```

6.4.3 常用功能示例代码

示例 1

```
/*
* testlibpq.c
*/
#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;
/*
用户在命令行上提供了 conninfo 字符串的值时使用该值
否则环境变量或者所有其它连接参数
都使用缺省值。
*/
if (argc > 1)
conninfo = argv[1]; else
conninfo = "dbname=postgres port=42121 host='10.44.133.171' application_name=test
connect_timeout=5 sslmode=allow user='test' password='test_1234'";
/*  连接数据库  */
conn = PQconnectdb(conninfo);
/*  检查后端连接成功建立  */
if (PQstatus(conn) != CONNECTION_OK)
```

```
{
fprintf(stderr, "Connection to database failed: %s", PQerrorMessage(conn));
exit_nicely(conn);
}
/*
测试实例涉及游标的使用时候必须使用事务块
*把全部放在一个 "select * from pg_database"
PQexec() 里，过于简单，不推荐使用
*/
/* 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);
}
/*
在结果不需要的时候 PQclear PGresult，以避免内存泄漏
*/ PQclear(res);
/*
从系统表 pg_database（数据库的系统目录）里抓取数据
*/
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);
}
/* 打 印 属 性 名 称 */ nFields = PQnfields(res);
for (i = 0; i < nFields; i++)
printf("%-15s", PQfname(res, i)); printf("\n\n");
/* 打印行 */
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 ... 不用检查错误 ... */
res = PQexec(conn, "CLOSE myportal"); PQclear(res);
/* 结束事务 */
res = PQexec(conn, "END"); PQclear(res);
/* 关闭数据库连接并清理 */ PQfinish(conn);
return 0;
}
```

示例 2

```
/*
testlibpq2.c
测试外联参数和二进制 I/O。
*
在运行这个例子之前，用下面的命令填充一个数据库
*
*
CREATE TABLE test1 (i int4, t text);
*
INSERT INTO test1 values (2, 'ho there');
*
期望的输出如下
*
*
tuple 0: got
i = (4 bytes) 2
t = (8 bytes) 'ho there'
*
*/
#include <stdio.h> #include <stdlib.h> #include <string.h> #include <sys/types.h> #include
<libpq-fe.h>
/* for ntohl/htonl */ #include <netinet/in.h> #include <arpa/inet.h>
static void exit_nicely(PGconn *conn)
{
PQfinish(conn); exit(1);
}
/*
这个函数打印查询结果，这些结果是二进制格式，从上面的
```

```
注释里面创建的表中抓取出来的
*/
static void show_binary_results(PGresult *res)
{
int  i;
int         i_fnum, t_fnum;
/*  使用  PQfnumber  来避免对结果中的字段顺序进行假设  */ i_fnum = PQfnumber(res,
"i");
t_fnum = PQfnumber(res, "t");
for (i = 0; i < PQntuples(res); i++)
{
char       *iptr;
char       *tptr;
int  ival;
/*  获取字段值（忽略可能为空的可能）   */ iptr = PQgetvalue(res, i, i_fnum);
tptr = PQgetvalue(res, i, t_fnum);
/*
INT4  的二进制表现形式是网络字节序
建议转换成本地字节序
*/
ival = ntohl(*((uint32_t *) iptr));
/*
TEXT  的二进制表现形式是文本，因此 libpq 能够给它附加一个字节零
把它看做  C  字符串
*
*/
printf("tuple %d: got\n", i); printf(" i = (%d bytes) %d\n",
PQgetlength(res, i, i_fnum), ival); printf(" t = (%d bytes) '%s'\n",
PQgetlength(res, i, t_fnum), tptr); printf("\n\n");
}
}
int
main(int argc, char **argv)
{
const char *conninfo;
PGconn *conn;
PGresult *res;
const char *paramValues[1]; int   paramLengths[1];
int  paramFormats[1]; uint32_t binaryIntVal;
/*
如果用户在命令行上提供了参数，
```

```
那么使用该值为 conninfo 字符串；否则
使用环境变量或者缺省值。
*/
if (argc > 1)
conninfo = argv[1]; else
conninfo = "dbname=postgres port=42121 host='10.44.133.171' application_name=test
connect_timeout=5 sslmode=allow user='test' password='test_1234'";
/* 和数据库建立连接 */
conn = PQconnectdb(conninfo);
/* 检查与服务器的连接是否成功建立 */
if (PQstatus(conn) != CONNECTION_OK)
{
fprintf(stderr, "Connection to database failed: %s", PQerrorMessage(conn));
exit_nicely(conn);
}
/* 把整数值 "2" 转换成网络字节序 */ binaryIntVal = htonl((uint32_t) 2);
/* 为 PQexecParams 设置参数数组 */ paramValues[0] = (char *) &binaryIntVal;
paramLengths[0] = sizeof(binaryIntVal); paramFormats[0] = 1; /* 二进制 */
res = PQexecParams(conn,
"SELECT * FROM test1 WHERE i = $1::int4",
1, /* 一个参数 */
NULL, /* 让后端推导参数类型 */ paramValues,
paramLengths, paramFormats,
1); /* 要求二进制结果 */
if (PQresultStatus(res) != PGRES_TUPLES_OK)
{
fprintf(stderr, "SELECT failed: %s", PQerrorMessage(conn)); PQclear(res);
exit_nicely(conn);
}
show_binary_results(res);
PQclear(res);
/* 关闭与数据库的连接并清理 */ PQfinish(conn);
return 0;
}
```

6.4.4 libpq 接口参数

请参见 8.3 libpq。

6.4.5 链接参数

表 6-12 链接参数

参数	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（或者安装数据库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。 接受以‘,’分割的字符串来指定多个主机名，支持指定多个主机名。
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。 接受以‘,’分割的字符串来指定多个 IP 地址，支持指定多个 IP 地址。 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： ● 如果声明了不带 hostaddr 的 host 那么就强制进行主机名查找。 ● 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。 ● 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。 接受以‘,’分割的字符串来指定多个端口号，支持指定多个端口号。
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_tim eout	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_enco ding	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）。
tty	忽略（以前，该参数指定了发送服务器调试输出的位置）。
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。
fallback_ap plication_na me	为 application_name 配置参数指定一个后补值。如果通过一个连接参数或 PGAPPNAME 环境变量没有为 application_name 给定一个值， 将使用这个值。在希望设置一个默认应用名但不希望它被用户覆盖的一般工具程序中指定一个后补值很有用。
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_i dle	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_i nterval	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_c ount	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。
rw_timeout	设置客户端连接读写超时时间。
sslmode	启用 SSL 加密的方式： ● disable：不使用 SSL 安全连接。 ● allow：如果数据库服务器要求使用，则可以使用 SSL 安全加密连接，但不验证数据库服务器的真实性。 ● prefer：如果数据库支持，那么首选使用 SSL 安全加密连接，但不验证数据库服务器的真实性。

	● require：必须使用 SSL 安全连接，但是只做了数据加密，而并不验证数据库服务器的真实性。 ● verify-ca：必须使用 SSL 安全连接，当前 windows odbc 不支持 cert ● 方式认证。 ● verify-full：必须使用 SSL 安全连接，当前 windows odbc 不支持 cert 方式认证。
sslcompress ion	如果设置为 1（默认），SSL 连接之上传送的数据将被压缩（这要求 OpenSSL 版本为 0.9.8 或更高）。如果设置为 0，压缩将被禁用（这要求 OpenSSL 版本为 1.0.0 或更高）。如果建立的是一个没有 SSL 的连接，这个参数会被忽略。如果使用的 OpenSSL 版本不支持该参数，它也会被忽略。压缩会占用 CUP 时间，但是当瓶颈为网络时可以提高吞吐量。如果 CPU 性能是限制因素，禁用压缩能够改进响应时间和吞吐量。
sslcert	这个参数指定客户端 SSL 证书的文件名，它替换默认的~/.postgresql/postgresql.crt。如果没有建立 SSL 连接，这个参数会被忽略。
sslkey	这个参数指定用于客户端证书的密钥位置。它能指定一个会被用来替代默认的~/.postgresql/postgresql.key 的文件名，或者它能够指定一个从外部“引擎”（引擎是 OpenSSL 的可载入模块）得到的密钥。一个外部引擎说明应该由一个冒号分隔的引擎名称以及一个引擎相关的关键标识符组成。如果没有建立 SSL 连接，这个参数会被忽略。
sslrootcert	这个参数指定一个包含 SSL 证书机构（CA）证书的文件名称。如果该文件存在，服务器的证书将被验证是由这些机构之一签发。默认值是~/.postgresql/root.crt。
sslcrl	这个参数指定 SSL 证书撤销列表（CRL）的文件名。列在这个文件中的证书如果存在，在尝试认证该服务器证书时会被拒绝。默认值是~/.postgresql/root.crl。
requirepeer	这个参数指定服务器的操作系统用户，例如 requirepeer=postgres。当建立一个 Unix 域套接字连接时，如果设置了这个参数，客户端在连接开始时检查服务器进程是否运行在指定的用户名之下。如果发现不是，该连接会被一个错误中断。这个参数能被用来提供与 TCP/IP 连接上 SSL 证书相似的服务器认证（注意，如果 Unix 域套接字在/tmp 或另一个公共可写的位置，任何用户能启动一个在那里侦听的服务器。使用这个参数来保证你连接的是一个由可信用户运行的服务器）。这个选项只在实现了 peer 认证方法的平台上受支持。

krbsrvname	当用 GSSAPI 认证时，要使用的 Kerberos 服务名。为了让 Kerberos 认证成功，这必须匹配在服务器配置中指定的服务名。
gsslib	用于 GSSAPI 认证的 GSS 库。只用在 Windows 上。设置为 gssapi 可强制 libpq 用 GSSAPI 库来代替默认的 SSPI 进行认证。
service	用于附加参数的服务名。它指定保持附加连接参数的 pg_service.conf 中的一个服务名。这允许应用只指定一个服务名，这样连接参数能被集中维护。
authtype	不再使用“authtype”，因此将其标记为“不显示”。我们将其保留在数组中，以免拒绝旧应用程序中的 conninfo 字符串，这些应用程序可能仍在尝试设置它。
remote_nod ename	指定连接本地节点的远端节点名称。
localhost	指定在一个连接通道中的本地地址。
localport	指定在一个连接通道中的本地端口。
fencedUdfRPCMode	控制 fenced UDF RPC 协议是使用 unix 域套接字或特殊套接字文件名。缺省值是 0，意思为关闭，使用 unix domain socket 模式，文件类型为“.s.PGSQL.%d”，但是要使用 fenced udf，文件类型为.s.fencedMaster_unixdomain，可以更改为 1，意思为开启。
replication	这个选项决定是否该连接应该使用复制协议而不是普通协议。这是 PostgreSQL 的复制连接以及 pg_basebackup 之类的工具在内部使用的协议，但也可以被第三方应用使用。支持下列值，大小写无关： ● true、on、yes、1：连接进入到物理复制模式。 ● database：连接进入到逻辑复制模式，连接到 dbname 参数中指定的数据库。 ● false、off、no、0：该连接是一个常规连接，这是默认行为。在物理或者逻辑复制模式中，仅能使用简单查询协议。
backend_ve rsion	传递到远端的后端版本号。
prototype	设置当前协议级别，默认：PROTO_TCP。
enable_ce	控制是否允许客户端连接全密态数据库。默认 0，如果需要开启，则修改为 1。
connection_ info	Connection_info 是一个包含 driver_name、driver_version、driver_path

	和 os_user 的 json 字符串。 如果不为 NULL，使用 connection_info 忽略 connectionExtraInf 如果为 NULL，生成与 libpq 相关的连接信息字符串，当 connectionExtraInf 为 false 时 connection_info 只有 driver_name 和 driver_version。
connectionExtraInf	设置 connection_info 是否存在扩展信息，默认值为 0，如果包含其他信息，则需要设置为 1。
target_session_attrs	设定连接的主机的类型。主机的类型和设定的值一致时才能连接成功。target_session_attrs 的设置规则如下： ● any(默认值)：可以对所有类型的主机进行连接。 ● read-write：当连接的主机允许可读可写时，才进行连接。 ● read-only：仅对可读的主机进行连接。 ● primary：仅对主备系统中的主机能进行连接。 ● standby：仅对主备系统中的备机进行连接。 ● prefer-standby：首先尝试找到一个备机进行连接。如果对 hosts 列表的所有机器都连接失败，那么尝试“any”模式进行连接。

6.5 基于 Psycopg 开发

Psycopg 是一种用于执行 SQL 语句的 Python API，可以为数据库提供统一访问接口。基于此，应用程序可进行数据操作。Psycopg2 是对 libpq 的封装，主要使用 C 语言实现，既高效又安全。它具有客户端游标和服务器端游标、异步通信和通知、支持 COPY TO/COPY FROM 功能。支持多种类型 Python 开箱即用，适配 PostgreSQL 数据类型；可以通过灵活的对象适配系统进行扩展和定制适配。Psycopg2 兼容 Unicode 和 Python 3。

GBase 8c 数据库支持 Psycopg2 特性，并且支持通过 SSL 模式链接到 Psycopg2。

6.5.1 Psycopg 包

获取发布包，并解压。解压后有两个文件夹：

- psycopg2：psycopg2 库文件。
- lib：lib 库文件。

6.5.2 开发流程

图 6-4 采用 Psycopg2 开发应用程序的流程

6.5.3 加载驱动

- 在使用驱动之前，需要做如下操作：
  - 先解压版本对应驱动包，使用 root 用户将 psycopg2 拷贝到 python 安装目录下的 site-packages 文件夹下。
  - 修改 psycopg2 目录权限为 755。
  - 将 psycopg2 目录添加到环境变量$PYTHONPATH，并使之生效。
  - 对于非数据库用户，需要将解压后的 lib 目录，配置在 LD_LIBRARY_PATH 中。
- 在创建数据库连接之前，需要先加载如下数据库驱动程序：

```
import psycopg2
```

6.5.4 连接数据库

- 使用 psycopg2.connect 函数获得 connection 对象。

- 使用 connection 对象创建 cursor 对象。

6.5.5 执行 SQL 语句

- 构造操作语句，使用%s 作为占位符，执行时 psycopg2 会用参数值智能替换掉占位符。可以添加 RETURNING 子句，来得到自动生成的字段值。
- 使用 cursor.execute 方法来操作一行，使用 cursor.executemany 方法来操作多行。

6.5.6 处理结果集

- cursor.fetchone()：这种方法提取的查询结果集的下一行，返回一个序列，没有数 据可用时则返回空。
- cursor.fetchall()：这个例程获取所有查询结果(剩余)行，返回一个列表。空行时 则返回空列表。

6.5.7 关闭连接

- 在使用数据库连接完成相应的数据操作后，需要关闭数据库连接。关闭数据库连接可以直接调用其 close 方法，如 connection.close()。

  ⚠ 注意

  此方法关闭数据库连接，并不自动调用 commit()。如果只是关闭数据库连接而不调用 commit()方法，那么所有更改将会丢失。

6.5.8 连接数据库（SSL 方式）

用户通过 psycopy2 连接 Kernel 服务器时，可以通过开启 SSL 加密客户端和服务器之间的通讯。在使用 SSL 时，默认用户已经获取了服务端和客户端所需要的证书和私钥文件，关于证书等文件的获取请参考 Openssl 相关文档和命令。

- 使用*.ini 文件（python 的 configparser 包可以解析这种类型的配置文件）保存数据库连接的配置信息。
- 在连接选项中添加 SSL 连接相关参数：sslmode，sslcert，sslkey，sslrootcert。
  - sslmode：可选项见表 6-14。
  - sslcert：客户端证书路径。

- sslkey：客户端密钥路径。
- sslrootcert：根证书路径。

- 使用 psycopg2.connect 函数获得 connection 对象。
- 使用 connection 对象创建 cursor 对象。

表 6-14 sslmode 的可选项及其描述

sslmode	是否会启用 SSL 加密	描述
disable	否	不适用 SSL 安全连接。
allow	可能	如果数据库服务器要求使用，则可以使用 SSL 安全加密连接，但不验证数据库服务器的真实性。
prefer	可能	如果数据库支持，那么首选使用 SSL 连接， 但不验证数据库服务器的真实性。
require	是	必须使用 SSL 安全连接，但是只做了数据加密，而并不验证数据库服务器的真实性。
verify-ca	是	必须使用 SSL 安全连接。
verify-full	是	必须使用 SSL 安全连接，目前暂不支持。

6.5.9 示例：常用操作

```
import psycopg2
#创建连接对象
conn=psycopg2.connect(database="postgres",user="user",password="password",host="localho
st",port=port) cur=conn.cursor() #创建指针对象
#创建连接对象（SSl 连接）
conn = psycopg2.connect(dbname="postgres", user="user", password="password",
host="localhost", port=port,
sslmode="verify-ca", sslcert="client.crt",sslkey="client.key",sslrootcert="cacert.pem")
注意： 如果 sslcert, sslkey,sslrootcert 没有填写，默认取当前用户.postgresql 目录下对应的
client.crt， client.key， root.crt
# 创建表
cur.execute("CREATE TABLE student(id integer,name varchar,sex varchar);")
#插入数据
```

```
cur.execute("INSERT INTO student(id,name,sex) VALUES(%s,%s,%s)",(1,'Aspirin','M'))
cur.execute("INSERT INTO student(id,name,sex) VALUES(%s,%s,%s)",(2,'Taxol','F'))
cur.execute("INSERT INTO student(id,name,sex) VALUES(%s,%s,%s)",(3,'Dixheral','M'))
# 获取结果
cur.execute('SELECT * FROM student') results=cur.fetchall()
print (results)
# 关闭连接 conn.commit() cur.close() conn.close()
psycopg2 常用链接方式
conn = psycopg2.connect(dbname="postgres", user="user", password="password",
host="localhost", port=port)
conn = psycopg2.connect("dbname=postgres user=user password=password host=localhost
port=port")
使用日志 import logging import psycopg2
from psycopg2.extras import LoggingConnection
logging.basicConfig(level=logging.DEBUG) # 日志级别
logger = logging.getLogger( name )
db_settings = { "user": "user",
"password": "password", "host": "localhost", "database": "postgres", "port": port
}
conn = psycopg2.connect(connection_factory=LoggingConnection, **db_settings)
conn.initialize(logger)
```

6.5.10 Psycopg 接口参考

请参见 8.4 Psycopg。

6.6 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，从而更好地了解数据库的运行状态。

可调整的配置参数请参见表 6-15。

表 6-15 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 • DEBUG1 • LOG • NOTICE • WARNING • ERROR • FATAL • PANIC 默认值：NOTICE 。	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。
log_min_me ssages	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 ● DEBUG1 ● INFO ● NOTICE ● WARNING ● ERROR ● FATAL ● PANIC 缺省值：ERROR。	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。
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：30min。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_duratio n	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：不记录已完成语句的持续时间。 默认值：off。	只有系统管理员可以修改该参数。
log_stateme nt	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_hostna me	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 7.3.2 查看审计结果 、20.3.17 GS_SESSION_MEM ORY_DETAIL、20.3.72 PG_STAT_ACTIVITY 和 log_line_prefix 参数。

上表有关参数级别的说明请参见表 6-16。

表 6-16 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。
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。

7 管理数据库安全

7.1 客户端接入认证

7.1.1 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

例如在主备式集群中执行如下操作。分布式集群在执行 gs_guc 命令时需指定-Z 参数。

步骤 1 以操作系统用户 gbase 登录数据库主节点。

步骤 2 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 omm 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。

```
gs_guc reload -N all -I all -h "host all jack 10.10.0.30/32 sha256"
```

说明

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

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

- -N all 表示 GBase 8c 的所有主机。
- -I all 表示主机的所有实例。
- -h 表示指定需要在“pg_hba.conf”增加的语句。
- all 表示允许客户端连接到任意的数据库。
- 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 GBase 8c 内部 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。
- sha256 表示连接时 jack 用户的密码使用 sha256 算法加密。

这条命令在数据库主节点实例对应的 pg_hba.conf 文件中添加了一条规则，用于对连接数据库主节点的客户端进行鉴定。

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 7.1.2 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

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。GBase 8c 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 GBase 8c 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。
- pg_hba.conf 的配置通过 gs_guc reload 命令修改后，在新创建会话时会生效。如果通过 gs_guc set 命令设置或直接编辑 pg_hba.conf 文件修改配置参数，需要重启数据库生效或重新执行 gs_guc reload 命令生效。

-----结束

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 7-1。

表 7-1 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。

7.1.2 配置文件参考

表 7-2 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 gsql 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。	-
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。	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 7.1.3 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。	-
DATABAS E	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 7-3： ● trust ● reject ● md5（不推荐使用，默认不支持，可通过 password_encryption_type 参数配置） 说明：MD5 加密算法安全性低，存在安全风险，建议使用更安全的加密算法。 ● sha256 ● sm3 ● cert ● gss(仅用于 GBase 8c 内部节点间认证) ● peer（仅用于 local 模式）

表 7-3 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 gsql 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。 ● 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，GBase 8c 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： ➢ MD5 加密算法安全性低，存在安全风险，建议使用更安全的加密算

	法。 ➢ GBase 8c 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。
sm3	要求客户端提供一个 sm3 算法加密口令进行认证，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）的单项 sm3 的加密，增加了安全性。
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 GBase 8c 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 GBase 8c 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。

peer	获取客户端所在操作系统用户名，并检查与数据库初始用户名是否一致。此方式只支持数据库初始用户通过 local 模式本地连接，并支持通过配置 pg_ident.conf 建立操作系统用户与数据库初始用户映射关系。
	假设操作系统用户名为 omm，数据库初始用户为 dbAdmin，在 pg_hba.conf 中配置 local 模式为 peer 认证：
	`local all all peer map=mymap`
	其中 map=mymap 指定使用的用户名映射，并在 pg_ident.conf 中添加映射名称为 mymap 的用户名映射如下：
	`# MAPNAME SYSTEM-USERNAME PG-USERNAME`
	`mymap gbase dbAdmin`
	说明：
	通过 gs_guc reload 方式修改 pg_hba.conf 配置可以立即生效，无需重启数据库。直接编辑修改 pg_ident.conf 配置后下次连接时自动生效，无需重启数据库。

7.1.3 用 SSL 进行安全的 TCP/IP 连接

背景信息

GBase 8c 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

GBase 8c 在数据库部署完成后，默认已开启 SSL 认证模式。服务器端证书，私钥以及根证书已经默认配置完成。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 7-4。

步骤 1 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/gbase”目录。

双向认证需配置如下参数：

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

单向认证需要配置如下参数：

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

步骤 2 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 GBase 8c。

```
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 7-4 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 7-5。

表 7-5 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，其相对路径是相对于数据目录的。 **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，其相对路径是相对于数据目录的。 **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 7-7。 **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。
ssl_cert_notify_time	SSL 服务器证书到期前提醒的天数。	请按照需求配置证书过期前提醒天数。 **默认值：90**

在客户端配置 SSL 认证相关的环境变量，详细信息请参见表 7-6。

说明

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

表 7-6 客户端参数

环境变量	描述	取值范围
PGSSLC ERT	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/gbase/client.crt' **默认值：空**
PGSSLK EY	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/gbase/client.key' **默认值：空**
PGSSLM ODE	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**
PGSSLR OOTCER T	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/gbase/certca.pem'

		默认值：空
PGSSLC RL	指定证书吊销列表文件，用于验证服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。	必须包含文件的绝对路径，如： export PGSSLCRL='/home/gbase/sslcrl-file.crl' **默认值：空**
PGSSLC ERT	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/gbase/client.crt' **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。
	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 7-7 所示。

表 7-7 加密算法套件

OpenSSL 套件名	IANA 套件名	安全程度
ECDHE-RSA-AES128-GCM-SHA256	TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256	HIGH

ECDHE-RSA-AES256-GCM-SHA384	TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384	HIGH
ECDHE-ECDSA-AES128-GCM-SHA256	TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256	HIGH
ECDHE-ECDSA-AES256-GCM-SHA384	TLS_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384	HIGH
DHE-RSA-AES128-GCM-SHA256	TLS_DHE_RSA_WITH_AES_128_GCM_SHA256	HIGH
DHE-RSA-AES256-GCM-SHA384	TLS_DHE_RSA_WITH_AES_256_GCM_SHA384	HIGH

说明

- 目前只支持上表中的 6 种加密算法套件。
- 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。为保持前向兼容保留了 DHE 算法套件，非兼容场景不推荐使用。
- 如需指定以上加密算法套件，可以设置 ssl_ciphers 为上表中 OpenSSL 套件名称，加密算法套件之间需要使用分号分割，如在 postgresql.conf 设置：

```
ssl_ciphers='ECDHE-RSA-AES128-GCM-SHA256;ECDHE-ECDSA-AES128-GCM-SHA256'
```

- 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。

7.1.4 用 SSH 隧道进行安全的 TCP/IP 连接

背景信息

为了保证服务器和客户端之间的安全通讯，可以在服务器和客户端之间构建安全的 SSH 隧道。SSH 是目前较可靠，专为远程登录会话和其他网络服务提供安全性的协议。

从 SSH 客户端来看，SSH 提供了两种级别的安全验证：

- 基于口令的安全验证：使用帐号和口令登录到远程主机。所有传输的数据都会被加密，

但是不能保证正在连接的服务器就是需要连接的服务器。可能会有其他服务器冒充真正的服务器，也就是受到“中间人”方式的攻击。

- 基于密钥的安全验证：用户必须为自己创建一对密钥，并把公用密钥放在需要访问的服务器上。这种级别的认证不仅加密所有传送的数据，而且避免“中间人”攻击方式。但是整个登录的过程可能需要 10 秒。

前提条件

SSH 服务和数据库运行在同一台服务器上。

操作步骤

以 OpenSSH 为例介绍配置 SSH 隧道，对于如何配置基于密钥的安全验证不作赘述，OpenSSH 提供了多种配置适应网络的各种限制，更多详细信息请参考 OpenSSH 的相关文档。

从本地主机建立到服务器的 SSH 隧道。

```
ssh -L 63333:localhost:5432 username@hostIP
```

说明

- -L 参数的第一串数字（63333）是通道本端的端口号，可以自由选择。
- 第二串数字（5432）是通道远端的端口号，也就是服务器使用的端口号。
- localhost 是本机 IP 地址，username 是要连接的服务器上的用户名，hostIP 是要连接的主机 IP 地址。

7.1.5 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。
- 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 gbase 登录数据库主节点。

步骤 2 使用如下命令连接数据库。

```
gsql -d postgres -p 5432
```

postgres 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：

```
gsql ((GBase 8c V5 build a4c7bcc5) compiled at 2024-01-16 14:49:56 commit 0 last mr 1632 )
Non-SSL connection (SSL connection is recommended when requiring high-security)
Last login: 2024-01-19 16:22:27+08 from [local] (user gbase)
Type "help" for help.

postgres=#
```

步骤 3 查看全局会话连接数限制。

```
postgres=# 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 7-8。

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 7-8 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 gbase 的会话连接数上限。其中-1 表示没有对用户 gbase 设置连接数的限制。 postgres=# SELECT ROLNAME,ROLCONNLIMIT FROM PG_ROLES WHERE ROLNAME='gbase'; rolname | rolconnlimit

	----------+-------------- gbase | -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 gbase 已使用的会话连接数。其中，1 表示 omm 已使用的会话连接数。 postgres=# 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'; postgres=# SELECT COUNT(*) FROM DV_SESSIONS WHERE USERNAME='gbase'; count ------- 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 postgres 的会话连接数上限。其中-1 表示没有对数据库 postgres 设置连接数的限制。 postgres=# SELECT DATNAME,DATCONNLIMIT FROM PG_DATABASE WHERE DATNAME='postgres'; datname | datconnlimit ----------+-------------- postgres | -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 postgres 上已使用的会话连接数。其中，1 表示数据库 postgres 上已使用的会话连接数。 postgres=# SELECT COUNT(*) FROM PG_STAT_ACTIVITY WHERE DATNAME='postgres'; 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。 postgres=# 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'; postgres=# SELECT COUNT(*) FROM DV_SESSIONS; count ------- 10 (1 row)

----结束

7.1.6 SSL 证书管理

GBase 8c 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。

7.1.6.1 证书生成

背景信息

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 omm 已存在,搭建 CA 的路径为 test
--以 omm 用户身份登录 Linux 环境 mkdir test
cd test
--copy 配置文件 openssl.cnf 到 test 下
cp /etc/pki/tls/openssl.cnf ./
--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 700 ./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
RSA 证书： openssl genrsa -aes256 -out demoCA/private/cakey.pem 2048
ECDSA 证书： openssl ecparam -name prime256v1 -genkey -out demoCA/private/cakey.pem
Generating RSA private key, 2048 bit long modulus
.....+++
.....+++
e is 65537 (0x10001)
--设置根私钥的保护密码
Enter pass phrase for demoCA/private/cakey.pem:
--再次输入私钥密码
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 careq.pem
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
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
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
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 5 生成服务端证书私钥，RSA 和 ECDSA 加密方式可以根据需要选择其中一种。

```
--生成 RSA 服务端证书私钥文件 server.key
RSA 证书私钥：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
Verifying - Enter pass phrase for server.key:
--再次确认服务器私钥的保护密码
--生成 ECDSA 服务端证书私钥文件 server.key
ECDSA 证书私钥： openssl ecparam -name prime256v1 -genkey -out server.key
```

```
--对 ECDSA 证书私钥进行加密保护，根据提示输入加密密码：  openssl ec -in server.key -aes256 -out server.key
read EC key writing EC key
Enter PEM pass phrase:
Verifying - Enter PEM pass phrase:
--根据提示输入服务端私钥的密码，加密后会生成 server.key.cipher,server.key.rand 两个私钥密码保护文件。gs_guc encrypt -M server -D ./
```

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:
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 中属性为 no。  vi demoCA/index.txt.attr
--对生成的服务端证书请求文件进行签发，签发后将生成正式的服务端证书 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
```

步骤 8 客户端证书，私钥的生成。

生成客户端证书和客户端证书私钥的方法和要求与服务器相同。

```
--生成客户端证书私钥，RSA 和 ECDSA 加密方式可以根据需要选择其中一种。RSA 证书私钥：openssl genrsa -aes256 -out client.key 2048
ECDSA 证书私钥：openssl ecparam -name prime256v1 -genkey -out client.key  对于 ECDSA 证书私钥，需要执行如下命令进行加密保护，根据提示输入加密密码：  openssl ec -in server.key -aes256 -out server.key
--根据提示输入客户端私钥的密码，加密后会生成 client.key.cipher,client.key.rand 两个私钥密码保护文件 gs_guc encrypt -M client -D ./
--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
```

根据需要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
```

----结束

7.1.6.2 证书替换

操作场景

GBase 8c 默认配置了 SSL 连接所需要的安全的证书、私钥，用户如果需要替换为自己的证书、私钥则可按照此方法进行替换。

前提条件

用户需要从 CA 认证中心申请到正式的服务器、客户端的证书和密钥。

注意事项

GBase 8c 目前只支持 X509v3 的 PEM 格式证书。

操作步骤

步骤 1 准备证书、私钥。

服务端各个配置文件名称约定：

- 证书名称约定：server.crt。
- 私钥名称约定：server.key。
- 私钥密码加密文件约定：server.key.cipher、server.key.rand。

客户端各个配置文件名称约定：

- 证书名称约定：client.crt。
- 私钥名称约定：client.key。
- 私钥密码加密文件约定：client.key.cipher、client.key.rand。
- 根证书名称约定：cacert.pem。
- 吊销证书列表文件名称约定：sslcrl-file.crl。

步骤 2 制作压缩包。

压缩包名称约定：db-cert-replacement.zip。

压缩包格式约定：ZIP。

压缩包文件列表约定：server.crt、server.key、server.key.cipher、server.key.rand、client.crt、client.key、client.key.cipher、client.key.rand、cacert.pem。如果需要配置吊销证书列表，则列表中包含 sslcrl-file.crl。

步骤 3 调用接口，执行替换。

1. 将制作好的压缩包 db-cert-replacement.zip 上传到 GBase 8c 用户下的任意路径。

例如：/home/xxxx/db-cert-replacement.zip。

2. 调用如下命令进行替换。

```
gs_om    -t    cert --cert-file=/home/xxxx/db-cert-replacement.zip
```

步骤 4 重启 GBase 8c。

```
gs_om -t stop
gs_om -t start
```

说明

证书具有 rollback 功能，可以把上一次执行证书替换之前的证书，进行回退。可以使用 gs_om -t cert --rollback 进行远程调用该接口；使用 gs_om -t cert --rollback -L 进行本地调用该接口。以上一次成功执行证书替换后，被替换的证书版本为基础进行回退。

----结束

7.2 管理用户及权限

7.2.1 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。数据库安装后的默认情况下，未开启三权分立，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《GBase 8c V5_3.0.1_SQL 手册》 GRANT 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。

GBase 8c 支持以下的权限：SELECT、INSERT、UPDATE、DELETE、TRUNCATE、REFERENCES、CREATE、CONNECT、EXECUTE、USAGE、ALTER、DROP、COMMENT、INDEX 和 VACUUM。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《GBase 8c V5_3.0.1_SQL 手册》 GRANT。

要撤消已经授予的权限，可以使用《GBase 8c V5_3.0.1_SQL 手册》 REVOKE。对象所

有者的权限 (例如 ALTER、 DROP、COMMENT、INDEX、VACUUM、GRANT 和 REVOKE) 是隐式拥有的，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，系统管理员用户除外。

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 20 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《GBase 8c V5_3.0.1_SQL 手册》ALTER DATABASE。

7.2.2 管理员

初始用户

数据库安装过程中自动生成的帐户称为初始用户。初始用户拥有系统的最高权限，能够执行所有的操作。如果安装时不指定初始用户名称，则该帐户与进行数据库安装的操作系统用户同名。如果在安装时不指定初始用户的密码，安装完成后密码为空，在执行其他操作前需要通过 gsql 客户端修改初始用户的密码。如果初始用户密码为空，则除修改密码外无法执行其他 SQL 操作以及升级、扩容、节点替换等操作。

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《GBase 8c V5_3.0.1_SQL 手册》 CREATE USER 语句或《GBase 8c V5_3.0.1_SQL 手册》 ALTER USER 语句进行设置。

```
postgres=# CREATE USER sysadmin WITH SYSADMIN password "xxxxxxxxx";
```

或者

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

ALTER USER 时，要求用户已存在。

监控管理员

监控管理员是指具有 MONADMIN 属性的帐户，具有查看 dbe_perf 模式下视图和函数的权限，亦可以对 dbe_perf 模式的对象权限进行授予或收回。

要创建新的监控管理员，请以系统管理员身份连接数据库，并使用带 MONADMIN 选项的《GBase 8c V5_3.0.1_SQL 手册》CREATE USER 语句或《GBase 8c V5_3.0.1_SQL 手册》ALTER USER 语句进行设置。

```
postgres=# CREATE USER monadmin WITH MONADMIN password   "xxxxxxxxx";
```

或者

```
postgres=# ALTER   USER joe MONADMIN;
```

ALTER USER 时，要求用户已存在。

运维管理员

运维管理员是指具有 OPRADMIN 属性的帐户，具有使用 Roach 工具执行备份恢复的权限。

要创建新的运维管理员，请以初始用户身份连接数据库，并使用带 OPRADMIN 选项的 CREATE USER 语句或《GBase 8c V5_3.0.1_SQL 手册》ALTER USER 语句进行设置。

```
postgres=# CREATE USER opradmin WITH OPRADMIN password "xxxxxxxxx";
```

或者

```
postgres=# ALTER USER joe OPRADMIN;
```

ALTER USER 时，要求用户已存在。

安全策略管理员

安全策略管理员是指具有 POLADMIN 属性的帐户，具有创建资源标签，脱敏策略和统一审计策略的权限。

要创建新的安全策略管理员，请以系统管理员用户身份连接数据库，并使用带 POLADMIN 选项的《GBase 8c V5_3.0.1_SQL 手册》 CREATE USER 语句或《GBase 8c V5_3.0.1_SQL 手册》 ALTER USER 语句进行设置。

```
postgres=# CREATE USER poladmin WITH POLADMIN password "xxxxxxxxx";
```

或者

```
postgres=# ALTER USER joe POLADMIN;
```

ALTER USER 时，要求用户已存在。

7.2.3 三权分立

默认权限机制和管理员两节的描述基于的是数据库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 CREATEROLE 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 CREATEROLE 属性和 AUDITADMIN 属性 的更多信息请参考《GBase 8c V5_3.0.1_SQL 手册》CREATE 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。

三权分立的设置办法为：将参数 enableSeparationOfDuty 设置为 on。

三权分立前的权限详情及三权分立后的权限变化，请分别参见表 7-9 和表 7-10。

表 7-9 默认的用户权限

对象名称	系统管理员	安全管理员	审计管理员	普通用户
表空间	对表空间有创建、修改、删除、访问、分配操作的权限。	不具有对表空间进行创建、修改、删除、分配的权限，访问需要被赋权。		
表	对所有表有所有的权限。	仅对自己的表有所有的权限，对其他用户的表无权限。		
索引	可以在所有的表上建立索引。	仅可以在自己的表上建立索引。		
模式	对除 dbe_perf 以外的所有模式有所有的权限。	仅对自己的模式有所有的权限，对其他用户的模式无权限。		
函数	对除 dbe_perf 模式下的函数以外的所有的函数有所有的权限。	仅对自己的函数有所有的权限，对其他用户放在 public 这个公共模式下的函数有调用的权限，对其他用户放在其他模式下的函数无权限。		
自定义视图	对除 dbe_perf 模式下的视图以外的所有的视图有所有的权限。	仅对自己的视图有所有的权限，对其他用户的视图无权限。		

系统表和系统视图	可以查看所有系统表和视图。	只可以查看部分系统表和视图。详细请参见系统表和系统视图。

表 7-10 三权分立较非三权分立权限变化说明

对象名称	系统管理员	安全管理员	审计管理员	普通用户
表空间	无变化。	无变化。		
表	对所有表有所有的权限。	仅对自己的表有所有的权限，对其他用户的表无权限。		
索引	权限缩小。 只可以在自己的表及其他用户放在 public 模式下的表上建立索引。	无变化。		
模式	权限缩小。 只对自己的模式有所有的权限，对其他用户的模式无权限。	无变化。		
函数	权限缩小。 只对自己的函数及其他用户放在 public 模式下的函数有所有的权限，对其他用户放在属于各自模式下的函数无权限。	无变化。		
自定义视图	权限缩小。 只对自己的视图及其他用户放在 public 模式下的视图有所有的权限，对其他用户放在属于各自模式下的视图无权限。	无变化。		
系统表和系统视图	无变化。	无变化。		

7.2.4 用户

使用 CREATE USER 和 ALTER USER 可以创建和管理数据库用户。GBase 8c 包含一个或多个已命名数据库用户和角色在 GBase 8c 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。

非三权分立下，GBase 8c 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，用户帐户由安全管理员创建。

在用户登录 GBase 8c 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。

- 要创建用户，请使用 SQL 语句《GBase 8c V5_3.0.1_SQL 手册》 CREATE USER。

  例如：创建用户 joe，并设置用户拥有 CREATEDB 属性。

```
postgres=# CREATE USER joe WITH CREATEDB PASSWORD "xxxxxxxxx";
CREATE ROLE
```

- 要创建系统管理员，请使用带有 SYSADMIN 选项的《GBase 8c V5_3.0.1_SQL 手册》 CREATE USER 语句 。
- 要删除现有用户，请使用《GBase 8c V5_3.0.1_SQL 手册》 DROP USER。
- 要更改用户帐户（例如，重命名用户或更改密码），请使用《GBase 8c V5_3.0.1_SQL 手册》ALTER USER。
- 要查看用户列表，请查询视图 PG_USER：

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

- 要查看用户属性，请查询系统表 PG_AUTHID：

```
postgres=# 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。

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，GBase 8c 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。

```
postgres=# 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。

须知

PG_STATISTIC 系统表和 PG_STATISTIC_EXT 系统表存储了统计对象的一些敏感信息，如高频值 MCV。系统管理员仍然可以通过访问这两张系统表，得到私有用户所属表的统计信息里的这些信息。

永久用户

GBase 8c 提供永久用户方案，即创建具有 PERSISTENCE 属性的永久用户。

```
postgres=# CREATE USER user_persistence WITH PERSISTENCE IDENTIFIED BY
"1234@abc";
```

只允许初始用户创建、修改和删除具有 PERSISTENCE 属性的永久用户。

7.2.5 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。

GBase 8c 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《GBase 8c V5_3.0.1_SQL 手册》GRANT。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《GBase 8c V5_3.0.1_SQL 手册》 CREATE ROLE。
- 要在现有角色中添加或删除用户，请使用《GBase 8c V5_3.0.1_SQL 手册》ALTER ROLE。
- 要删除角色，请使用《GBase 8c V5_3.0.1_SQL 手册》DROP ROLE。DROP ROLE 只会删除角色，并不会删除角色中的成员用户帐户。

内置角色

GBase 8c 提供了一组默认角色，以 gs_role_开头命名。它们提供对特定的、通常需要高权限的操作的访问，可以将这些角色 GRANT 给数据库内的其他用户或角色，让这些用户能够使用特定的功能。在授予这些角色时应当非常小心，以确保它们被用在需要的地方。表 7-11 描述了内置角色允许的权限范围：

表 7-11 内置角色权限描述

角色	权限描述
gs_role_copy_f iles	具有执行 copy … to/from filename 的权限，但需要先打开 GUC 参数 enable_copy_server_files。
gs_role_signal _backend	具有调用函数 pg_cancel_backend、pg_terminate_backend 和 pg_terminate_session 来取消或终止其他会话的权限，但不能操作属于初始用户和 PERSISTENCE 用户的会话。
gs_role_tables pace	具有创建表空间（tablespace）的权限。
gs_role_replic ation	具有调用逻辑复制相关函数的权限，例如 kill_snapshot、pg_create_logical_replication_slot 、pg_create_physical_replication_slot、pg_drop_replication_slot、pg_replication_slot_advance 、pg_create_physical_replication_slot_extern 、pg_logical_slot_get_changes、pg_logical_slot_peek_changes、pg_logical_slot_get_binary_changes 、pg_logical_slot_peek_binary_changes。
gs_role_accou nt_lock	具有加解锁用户的权限，但不能加解锁初始用户和 PERSISTENCE 用户。
gs_role_pldeb ugger	具有执行 dbe_pldebugger 下调试函数的权限。
gs_role_direct ory_create	具有执行创建 directory 对象的权限，但需要先打开 GUC 参数 enable_access_server_directory。

gs_role_direct ory_drop	具有执行删除 directory 对象的权限，但需要先打开 GUC 参数 enable_access_server_directory。

关于内置角色的管理有如下约束：

- 以 gs_role_开头的角色名作为数据库的内置角色保留名，禁止新建以“gs_role_” 开头的用户/角色，也禁止将已有的用户/角色重命名为以“gs_role_”开头；
- 禁止对内置角色的 ALTER 和 DROP 操作；
- 内置角色默认没有 LOGIN 权限，不设预置密码；
- gsql 元命令\du 和\dg 不显示内置角色的相关信息，但若显示指定了 pattern 为特定内置角色则会显示。
- 三权分立关闭时，初始用户、具有 SYSADMIN 权限的用户和具有内置角色 ADMIN OPTION 权限的用户有权对内置角色执行 GRANT/REVOKE 管理。三权分立打开时，具有内置角色 ADMIN OPTION 权限的用户有权对内置角色执行 GRANT/REVOKE 管理。例如：

```
GRANT gs_role_signal_backend TO user1;
REVOKE gs_role_signal_backend FROM user1;
```

7.2.6 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 usage 权限，只有系统管理员和初始化用户可以在 public Schema 下创建函数、存储过程和同义词对象，其他用户即使赋予 create 权限后也不可以创建上述三种对 象。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。默认只有初始化用户可以在 pg_catalog 模式下创建对象。

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。

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。

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《GBase 8c V5_3.0.1_SQL 手册》 CREATE 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《GBase 8c V5_3.0.1_SQL 手册》 GRANT 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《GBase 8c V5_3.0.1_SQL 手册》 ALTER SCHEMA。Schema 所有者可以更改 Schema。
- 要删除 Schema 及其对象，请使用《GBase 8c V5_3.0.1_SQL 手册》 DROP 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。

```
postgres=# 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。

```
postgres=# 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。

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

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《GBase 8c V5_3.0.1_SQL 手册》 SHOW。

```
postgres=# 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。

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

7.2.7 用户权限设置

- 给用户直接授予某对象的权限，请使用《GBase 8c V5_3.0.1_SQL 手册》 GRANT。

  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。

```
postgres=# GRANT USAGE ON SCHEMA tpcds TO joe;
postgres=# 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
  - 创建角色。

  新建一个角色 lily，同时给角色指定系统权限 CREATEDB：

```
postgres=# CREATE ROLE lily WITH CREATEDB PASSWORD "xxxxxxxxx";
```

  - 给角色赋予对象权限，请使用《GBase 8c V5_3.0.1_SQL 手册》 GRANT。

  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。

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

  - 将角色的权限赋予用户。

```
postgres=# 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《GBase 8c V5_3.0.1_SQL 手册》 REVOKE。

7.2.8 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。

示例：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
postgres=# CREATE USER alice PASSWORD 'xxxxxxxxx';
postgres=# CREATE USER bob PASSWORD 'xxxxxxxxx';
postgres=# CREATE USER peter PASSWORD 'xxxxxxxxx';
--创建表 all_data，包含不同用户数据信息
postgres=# CREATE TABLE all_data(id int, role varchar(100), data varchar(100));
--向数据表插入数据
postgres=# INSERT INTO all_data VALUES(1, 'alice', 'alice data');
postgres=# INSERT INTO all_data VALUES(2, 'bob', 'bob data');
postgres=# 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
postgres=# GRANT SELECT ON all_data TO alice, bob, peter;
--打开行访问控制策略开关
postgres=# 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
postgres=# CREATE ROW LEVEL SECURITY POLICY all_data_rls ON all_data USING(role
= CURRENT_USER);
--查看表详细信息
```

```
postgres=# \d+ all_data
Table "public.all_data"
Column |       Type| Modifiers | Storage | Stats target | Description
--------+-----------------------+-----------+----------+--------------+-------------
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"
gbase=> SELECT * FROM public.all_data;
id | role | data
----+-------+------------
1 | alice | alice data (1 row)
gbase=> 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"
gbase=> SELECT * FROM public.all_data;
id | role | data
----+-------+------------
3 | peter | peter data (1 row)
gbase=> 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)
```

须知

PG_STATISTIC 系统表和 PG_STATISTIC_EXT 系统表存储了统计对象的一些敏感信息，如高频值 MCV。若创建行级访问控制后，将这两张系统表的查询权限授予普通用户，则普通用户仍然可以通过访问这两张系统表，得到统计对象里的这些信息。

7.2.9 设置安全策略

7.2.9.1 设置账户安全策略

背景信息

GBase 8c 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。

自动锁定和解锁账户

- 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，默认值为 10。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。

  例如在分布式集群中修改此参数，执行操作如下：

1. 以操作系统用户 gbase 登录数据库 CN 节点。
2. 使用如下命令连接数据库。

```
gsql -d postgres -p 5432
```

postgres 为需要连接的数据库名称，5432 为数据库 CN 节点的端口号。连接成功后，系统显示类似如下信息：

```
gsql ((GBase 8c V5 build a4c7bcc5) compiled at 2024-01-16 14:49:56 commit 0 last mr 1632 )
Non-SSL connection (SSL connection is recommended when requiring high-security)
Last login: 2024-01-19 16:22:27+08 from [local] (user gbase)
Type "help" for help.

postgres=#
```

3. 查看已配置的参数。

```
postgres=# SHOW failed_login_attempts;
failed_login_attempts
----------------------
10
(1 row)
```

如果显示结果不为 10，执行“\q”命令退出数据库。

4. 执行如下命令设置成默认值 10。

```
gs_guc reload -Z coordinator -D /home/gbase/data/dn1/dn1_1/ -c "failed_login_attempts=10"
```

其中-Z 参数仅在分布式情况下需要指定。

配置 password_lock_time 参数。

例如在分布式集群中修改此参数，执行操作如下：

1. 以操作系统用户 gbase 登录数据库 CN 节点。

2. 使用如下命令连接数据库。

```
gsql -d postgres -p 5432
```

postgres 为需要连接的数据库名称，5432 为数据库 CN 节点的端口号。连接成功后，系统显示类似如下信息：

```
gsql ((GBase 8c V5 build a4c7bcc5) compiled at 2024-01-16 14:49:56 commit 0 last mr 1632 )
Non-SSL connection (SSL connection is recommended when requiring high-security)
Last login: 2024-01-19 16:22:27+08 from [local] (user gbase)
Type "help" for help.

postgres=#
```

3. 查看已配置的参数。

```
postgres=# 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。

4. 执行如下命令设置成默认值 1。

```
gs_guc reload -Z coordinator -N all -I all -c    "password_lock_time=1"
```

其中-Z 参数仅在分布式情况下需要指定。

手动锁定和解锁账户

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 7.2.4 用户，命令格式如下：

- 手动锁定

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

- 手动解锁

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

删除不再使用的账户

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：

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

7.2.9.2 设置账号有效期

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 gbase 登录数据库 CN 节点。

步骤 2 使用如下命令连接数据库。

```
gsql -d postgres -p 5432
```

postgres 为需要连接的数据库名称，5432 为数据库 CN 节点的端口号。连接成功后，系统显示类似如下信息：

```
gsql ((GBase 8c V5 build a4c7bcc5) compiled at 2024-01-16 14:49:56 commit 0 last mr 1632 )
```

```
Non-SSL connection (SSL connection is recommended when requiring high-security)
Last login: 2024-01-19 16:22:27+08 from [local] (user gbase)
Type "help" for help.

postgres=#
```

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。

```
postgres=# CREATE USER joe WITH PASSWORD 'gbase;123' VALID BEGIN '2022-6-6 08:00:00' VALID UNTIL '2022-6-6 18:00:00';
```

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。

```
postgres=# ALTER USER joe WITH VALID BEGIN '2022-6-6 08:00:00' VALID UNTIL '2022-6-6 18:00:00';
```

显示如下信息表示重新设定成功。

```
ALTER ROLE
```

----结束

须知

- CREATE ROLE 语法中不指定“VALID BEGIN”和“VALID UNTIL”时，表示不对用户的开始操作时间和结束操作时间进行限定。
- ALTER ROLE 语法中不指定“VALID BEGIN”和“VALID UNTIL”时，表示不对用户的开始操作时间和结束操作时间进行修改，沿用之前设置。

7.2.9.3 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，GBase 8c 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 加密算法安全性低，存在安全风险，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。

MD5 加密算法安全性低，存在安全风险，不建议使用。

- 当参数 password_encryption_type 设置为 2 时，表示采用 sha256 方式对密码加密， 为默认配置。
- 当参数 password_encryption_type 设置为 3 时，表示采用 sm3 方式对密码加密。

例如在分布式集群中修改，执行操作如下：

步骤 1 以操作系统用户 gbase 登录数据库 CN 节点。

步骤 2 使用如下命令连接数据库。

```
gsql -d postgres -p 5432
```

postgres 为需要连接的数据库名称，5432 为数据库 CN 节点的端口号。连接成功后，系统显示类似如下信息：

```
gsql ((GBase 8c V5 build a4c7bcc5) compiled at 2024-01-16 14:49:56 commit 0 last mr 1632 )
Non-SSL connection (SSL connection is recommended when requiring high-security)
Last login: 2024-01-19 16:22:27+08 from [local] (user gbase)
Type "help" for help.

postgres=#
```

步骤 3 查看已配置的加密算法。

```
postgres=# SHOW password_encryption_type;
password_encryption_type
--------------------------
  2
(1 row)
```

如果显示结果为 0 或 1，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。

```
gs_guc reload -Z coordinator -N all -I all -c "password_encryption_type=2"
```

其中-Z 参数仅在分布式情况下需要指定。

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 时表示不采用密码复杂度校验，但需满足密码不能为空并且只包含有效字符，有效字符范围为大写字母（A-Z）、小写字母（a-z）、数字（0-9）及特殊字符详见表 7-12。设置为 0 会存在安全风险，不建议设置为 0。如需要设置，则需将所有数据库节点中的 password_policy 都设置为 0 才能生效。

配置 password_policy 参数。

例如在分布式集群中修改此参数，执行操作如下：

(1) 使用如下命令连接数据库。

```
gsql -d postgres -p 5432
```

postgres 为需要连接的数据库名称，5432 为数据库 CN 节点的端口号。连接成功后，系统显示类似如下信息：

```
gsql ((GBase 8c V5 build a4c7bcc5) compiled at 2024-01-16 14:49:56 commit 0 last mr 1632 )
Non-SSL connection (SSL connection is recommended when requiring high-security)
Last login: 2024-01-19 16:22:27+08 from [local] (user gbase)
Type "help" for help.

postgres=#
```

(2) 查看已配置的参数。

```
postgres=# 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。

(3) 执行如下命令设置成默认值 1。

```
gs_guc reload -Z coordinator -N all -I all -c "password_policy=1"
```

其中-Z 参数仅在分布式情况下需要指定。

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 7-12）
- 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。
- 不能是弱口令。
  - 弱口令指的是强度较低，容易被破解的密码，对于不同的用户或群体，弱口令的定义可能会有所区别，用户需自己添加定制化的弱口令。
  - 弱口令字典中的口令存放在 gs_global_config 系统表中，当创建用户、修改用户需要设置密码时，系统将会把用户设置口令和弱口令字典中存放的口令进行对比，如果符合，则会提示用户该口令为弱口令，设置密码失败。
  - 弱口令字典默认为空，用户通过以下语法可以对弱口令字典进行增加和删除，示例如下：

```
postgres=# CREATE WEAK PASSWORD DICTIONARY WITH VALUES ('password1'),
('password2');
postgres=# DROP WEAK PASSWORD DICTIONARY;
```

- 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 7-13 所示。

说明

不可重用天数默认值为 60 天，不可重用次数默认值是 0。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。

例如在分布式集群中修改此参数，执行操作如下：

(1) 使用如下命令连接数据库。

```
gsql -d postgres -p 5432
```

postgres 为需要连接的数据库名称，5432 为数据库 CN 节点的端口号。连接成功后，系统显示类似如下信息：

```
gsql ((GBase 8c V5 build a4c7bcc5) compiled at 2024-01-16 14:49:56 commit 0 last mr 1632 )
Non-SSL connection (SSL connection is recommended when requiring high-security)
Last login: 2024-01-19 16:22:27+08 from [local] (user gbase)
Type "help" for help.

postgres=#
```

(2) 查看已配置的参数。

```
postgres=# SHOW password_reuse_time;
password_reuse_time
--------------------
 60
(1 row)
```

如果显示结果不为 60，执行“\q”命令退出数据库。

(3) 执行如下命令设置成默认值 60。

说明

不建议设置为 0。如需要设置，则需将所有节点中的 password_reuse_time 都设置为 0，才能生效。

```
gs_guc reload -Z coordinator -N all -I all -c "password_reuse_time=60"
```

其中-Z 参数仅在分布式情况下需要指定。

配置 password_reuse_max 参数。

例如在分布式集群中修改此参数，执行操作如下：

(1) 使用如下命令连接数据库。

```
gsql -d postgres -p 5432
```

postgres 为需要连接的数据库名称，5432 为数据库 CN 节点的端口号。连接成功后，系统显示类似如下信息：

```
gsql ((GBase 8c V5 build a4c7bcc5) compiled at 2024-01-16 14:49:56 commit 0 last mr 1632 )
```

```
Non-SSL connection (SSL connection is recommended when requiring high-security)
Last login: 2024-01-19 16:22:27+08 from [local] (user gbase)
Type "help" for help.

postgres=#
```

(2) 查看已配置的参数。

```
postgres=# SHOW password_reuse_max;
password_reuse_max
--------------------
 0
(1 row)
```

如果显示结果不为 0，执行“\q”命令退出数据库。

(3) 执行如下命令设置成默认值 0。

```
gs_guc reload -Z coordinator -N all -I all -c "password_reuse_max = 0"
```

其中-Z 参数仅在分布式情况下需要指定。

- 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。

例如在分布式集群中修改此参数，执行操作如下：

(1) 使用如下命令连接数据库。

```
gsql -d postgres -p 5432
```

postgres 为需要连接的数据库名称，5432 为数据库 CN 节点的端口号。连接成功后，系统显示类似如下信息：

```
gsql ((GBase 8c V5 build a4c7bcc5) compiled at 2024-01-16 14:49:56 commit 0 last mr 1632 )
Non-SSL connection (SSL connection is recommended when requiring high-security)
Last login: 2024-01-19 16:22:27+08 from [local] (user gbase)
Type "help" for help.
```

```
postgres=#
```

(2) 查看已配置的参数。

```
postgres=# 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。

(3) 执行如下命令设置成默认值 90（不建议设置为 0）。

```
gs_guc reload -Z coordinator -N all -I all -c "password_effect_time = 90"
```

其中-Z 参数仅在分布式情况下需要指定。

配置 password_notify_time 参数。

例如在分布式集群中修改此参数，执行操作如下：

(1) 使用如下命令连接数据库。

```
gsql -d postgres -p 5432
```

postgres 为需要连接的数据库名称，5432 为数据库 CN 节点的端口号。连接成功后，系统显示类似如下信息：

```
gsql ((GBase 8c V5 build a4c7bcc5) compiled at 2024-01-16 14:49:56 commit 0 last mr 1632 )
Non-SSL connection (SSL connection is recommended when requiring high-security)
Last login: 2024-01-19 16:22:27+08 from [local] (user gbase)
Type "help" for help.

postgres=#
```

(2) 查看已配置的参数。

```
postgres=# SHOW password_notify_time;
password_notify_time
---------------------
 7
(1 row)
```

(3) 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）。

```
gs_guc reload -Z coordinator -N all -I all -c "password_notify_time = 7"
```

其中-Z 参数仅在分布式情况下需要指定。

- 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：

```
postgres=# 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：

```
postgres=# 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。

```
postgres=# SET ROLE joe PASSWORD "abc@1234";
ERROR: Invalid username/password,set role denied.
```

以设置用户 joe 为例，命令格式如下：

```
postgres=# SET ROLE joe PASSWORD "abc@1234";
ERROR: Invalid username/password,set role denied.
```

表 7-12 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	:	-	-

表 7-13 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 60。	● 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，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 0，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。

步骤 6 设置用户密码失效。

具有 CREATEROLE 权限的用户在创建用户时可以强制用户密码失效，新用户首次登陆

数据库后需要修改密码才允许执行其他查询操作，命令格式如下：

```
postgres=# CREATE USER joe PASSWORD "abc@1234" EXPIRED;
CREATE ROLE
```

具有 CREATEROLE 权限的用户可以强制用户密码失效或者强制修改密码且失效，命令格式如下：

```
postgres=# ALTER USER joe PASSWORD EXPIRED;
ALTER ROLE
postgres=# ALTER USER joe PASSWORD "abc@2345" EXPIRED;
ALTER ROLE
```

说明

- 密码失效的用户登录数据库后，当执行简单查询或者扩展查询时，会提示用户修改密码。修改密码后可以正常执行语句。
- 系统管理员（sysadmin）或拥有创建用户（CREATEROLE）权限的用户才可以设置用户密码失效，其中系统管理员也可以设置自己或其他系统管理员密码失效。

----结束

7.3 身份鉴别

7.3.1 LDAP 认证

GBase 8c 支持 LDAP 认证服务器进行身份鉴别。

LDAP（轻型目录访问协议）认证是一种常见的身份验证技术，被广泛应用关于企业内部网络中。在 GBase 8c 数据库中，使用 LDAP 认证可以提高数据库安全性，减少不合法访问的风险，并且 LDAP 目录服务可以在企业内部网络环境中进行部署，统一管理用户身份，减少管理员的工作量和管理成本。

LDAP 认证可以将用户身份验证过程与 LDAP 目录服务相结合。LDAP 目录服务会通过 LDAP 协议来提供身份认证服务。在 GBase 8c 系统中，需要支持以 LDAP 协议请求身份认证服务，将用户身份信息提交给 LDAP 服务器进行验证。如果验证成功，则用户可以进行 GBase 8c 数据库访问，否则不允许进行数据库访问。

LDAP 概念：

- 目录树：在一个目录服务系统中，整个目录信息集可以表示为一个目录信息树，树中每个节点是一个条目。

- 条目：每个条目是一条记录，每个条目有自己唯一可区分的名称。
- 对象类：与某个实体类型对应的一组属性，对象类可以继承。
- 属性：描述条目的某个方面的信息，一个属性由属性类型和属性值组成。属性有必须属性和非必须属性。

使用

服务器上安装依赖包

```
sudo yum install openldap-devel-y
```

搭建 ldap 服务器，修改 GBase 8c pg_hba.conf 配置文件，格式如下：

```
local test test ldap ldapserver=10.0.3.118 ldapprefix=”cn=” ldapsuffix=”,dc=gbase,dc=com”
host test test 127.0.0.0.1/32 ldap ldapserver=hostip ldapprefix=”cn=” ldapsuffix=”, dc=gbase,dc
```

7.3.2 口令接入

为增强密码口令存储的安全性，保护用户数据安全，GBase 8c 数据库支持使用专业的硬件密码卡设备提供的密钥存储管理、基于国密算法的加解密功能，以实现对数据库用户访问口令的加密存储、传输与身份认证。

实现原则如下：

- 数据库服务端

数据库服务端使用密码卡提供存储加密服务。

- 数据库客户端

客户端加密驱动模块调用密码卡使用国密算法实现用户数据库的加解密运算，运算的结果与存储的本用户的口令摘要值进行对比，如果对比成功，则授权用户登录；否则，用户登录失败。

7.4 传输加密

TLCP（即传输层密码协议）是一种网络安全协议，用于保护网络传输过程中的数据安全。该协议可以在传输层上提供端到端的加密和身份认证服务，可以有效的防止被动攻击、中间人攻击等网络安全威胁。其特点是采用加密证书/私钥和签名证书/私钥相分离的方式。TLCP

利用密码技术，为两个应用程序之间提供保密性和数据完整性。协议用到的密码算法包括非对称加密算法、分组密码算法、密码杂凑算法、数据扩展函数和伪随机函数，协议用到的密钥种类包含服务端密钥、客户端密钥、预主密钥、主密钥和工作密钥。

TLCP 使用的加密算法支持对称加密算法和非对称加密算法，并且支持 RSA、ECC 等公钥算法和 AES、DES 等对称加密算法。此外，TLCP 还支持多种数字签名算法、消息认证码算法和伪随机数生成算法。

GBase 8c 支持 TLCP 安全通信协议的支持，并符合 GB/T38636-2020 标准。

可参考以下认证方式与流程：

1. 生成自签名 CA 证书

2. 生成服务端签名证书和加密证书

3. 生成客户端签名证书和加密证书

4. 支持单向 tlcp 认证登陆（不需要客户端拥有 CA 证书）

5. 支持双向认证（需要服务器、客户端双方均拥有 CA 证书）

使用示例

(1) 生成自签名 CA 证书

```
openssl ecparam -name SM2 -out SM2.pem
openssl req -config ./openssl.cnf -nodes -subj '/C=AA/ST=BB/O=CC/OU=DD/CN=root ca' -keyout CA.key -newkey ec:SM2.pem -new -out CA.csr
openssl x509 -sm3 -req -days 30 -in CA.csr -extfile ./openssl.cnf -extensions v3_ca -signkey CA.key -out CA.crt
```

(2) 生成服务端签名证书和加密证书

```
openssl req -config ./openssl.cnf -nodes -subj '/C=AA/ST=BB/O=CC/OU=DD/CN=server sign' -keyout SS.key -newkey ec:SM2.pem -new -out SS.csr
openssl x509 -sm3 -req -days 30 -in SS.csr -CA CA.crt -CAkey CA.key -extfile ./openssl.cnf -extensions v3_req -out SS.crt -CAcreateserial
openssl req -config ./openssl.cnf -nodes -subj '/C=AA/ST=BB/O=CC/OU=DD/CN=server enc' -keyout SE.key -newkey ec:SM2.pem -new -out SE.csr
openssl x509 -sm3 -req -days 30 -in SE.csr -CA CA.crt -CAkey CA.key -extfile ./openssl.cnf -extensions v3enc_req -out SE.crt -CAcreateserial
```

(3) 生成客户端签名证书和加密证书

```
openssl req -config ./openssl.cnf -nodes -subj '/C=AA/ST=BB/O=CC/OU=DD/CN=client
sign' -keyout CS.key -newkey ec:SM2.pem -new -out CS.csr
openssl x509 -sm3 -req -days 30 -in CS.csr -CA CA.crt -CAkey CA.key
-extfile ./openssl.cnf -extensions v3_req -out CS.crt -CAcreateserial
openssl req -config ./openssl.cnf -nodes -subj '/C=AA/ST=BB/O=CC/OU=DD/CN=client
enc' -keyout CE.key -newkey ec:SM2.pem -new -out CE.csr
openssl x509 -sm3 -req -days 30 -in CE.csr -CA CA.crt -CAkey CA.key
-extfile ./openssl.cnf -extensions v3enc_req -out CE.crt -CAcreateserial
```

(4) 修改配置文件，证书和私钥为步骤 2 生成的对应文件

```
ssl = on
tlcp = true
ssl_cert_file = 'SS.crt'
ssl_key_file = 'SS.key'
ssl_ca_file = 'CA.crt'
ssl_enc_cert_file = 'SE.crt'
ssl_enc_key_file = 'SE.key'
```

(5) 修改 hba 配置文件

如果是单向认证的话，修改为

```
hostssl    all    all    0.0.0.0/0    md5
```

如果是双向认证的话，修改为

```
hostssl    all    all    0.0.0.0/0    cert clientcert=1
```

(6) 登陆

单向认证登陆：

```
gsql "user=user host=host port=port dbname=dbname tlcp=1"
```

双向认证登陆，服务端还需要验证客户端的证书和密钥。在运行 gsql 客户端的 /home/gbase 目录创建文件夹.postgresql,将生成的 CS.key 和 CS.crt 放入到上面文件，改名为 postgresql.key 和 postgresql.crt。

7.5 数据插入加密

GBase 8c 提供 gs_encrypt 工具，为输入的明文字符串进行加密操作。支持 AES128、SM4 算法。操作如下：

(1) 使用如下命令连接数据库。

```
gsql -d postgres -p 5432
```

postgres 为需要连接的数据库名称，5432 为数据库 CN 节点的端口号。连接成功后，系统显示类似如下信息：

```
gsql ((GBase 8c V5 build a4c7bcc5) compiled at 2024-01-16 14:49:56 commit 0 last mr 1632 )
Non-SSL connection (SSL connection is recommended when requiring high-security)
Last login: 2024-01-19 16:22:27+08 from [local] (user gbase)
Type "help" for help.

postgres=#
```

(2) 准备需要加密的表和数据。表 test0 中存放以 sm4 加密的字符串，test1 中存放以 aes128 加密的字符串。

```
postgres=# create table test0(name varchar(100));
postgres=# create table test1(name varchar(100));
```

(3) 插入加密数据

- 使用 SM4 算法：

```
postgres=# insert into test0 values (gs_encrypt('name1','2023Gbase','sm4'));
```

其中 gs_encrypt(encryptstr,keystr,encrypttype)函数用于对字符串 encryptstr 进行加密，keystr 为密钥关键字，长度范围为 8~16 字节，至少包含 3 种字符（大写字母、小写字母、数字、特殊字符）。encrypttype 为加密算法，可以是 aes128 或 sm4。

- 使用 aes128 算法：

```
postgres=# insert into test1 values (gs_encrypt('name2','2023Gbase','aes128'));
#或使用 gs_encryp_aes128 函数
postgres=# insert into test1 values (gs_encrypt_aes128('name3','2023Gbase'));
```

其中 gs_encrypt_aes128(encryptstr,keystr)函数以 keystr 为密钥对 encryptstr 字符串进行 aes128 算法加密，返回加密后的字符串。等效于 gs_encrypt(encryptstr,keystr,aes128)。

(4) 查询

- 直接查询表中数据，显示加密后的字符串。

```
postgres=# select name from test0;
                        name
------------------------------
 n9+qK12bhDdt57TJjqZMLoJnDUrX
```

```
(1 row)

postgres=# select name from test1;
                                                                name
-------------------------------------------------------------------------------------------
nI2FTtGNk4NUmetJv6cFzDEn7pBG7XYJ+Xv0jmB052VUN5sw1C5z4/hbUWi1hWXc6sut5Z
KzNfGRSFxNDqZeFoIjDJg=

nI2FTtGNk4NUmetJv6cFzBCKSd1KNugkJZF0OidENU3BVwHrSzc6rhQb7v3AqaC/n/S38++
6lo2NG0Bi8nqPr+z3dI4=
(2 rows)
```

- 通过指定密钥关键字，解密查询显示解密后的真实字符。

```
# 指定 sm4 算法解密，正确显示以 sm4 算法加密的字符串。
postgres=# select gs_decrypt(name,'2023Gbase','sm4') from test0;
                    gs_decrypt
---------------------------------------
 name1
(1 row)

# 指定 aes128 算法解密，正确显示以 aes128 算法加密的字符串。
postgres=# select gs_decrypt(name,'2023Gbase','aes128') from test1;
 gs_decrypt
------------
 name2
 name3
(2 rows)
#或
postgres=# select gs_decrypt_aes128(name,'2023Gbase') from test1;
 gs_decrypt_aes128
-------------------
 name2
 name3
(2 rows)
```

其中 gs_decrypt(decryptstr,keystr,decrypttype)函数根据 decrypttype，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 decrypttype 及 keystr 必须保证与加密时使用的 encrypttype 及 keystr 一致才能正常解密。keystr 不得为空。decrypttype 可以是 aes128 或 sm4。

gs_decrypt_aes128(decryptstr,keystr)以 keystr 为密钥对 decrypt 字符串进行 aes128 算法解密，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。

7.6 设置数据库审计

7.6.1 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。GBase 8c 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，GBase 8c 支持以下审计项如表 7-14 所示。

表 7-14 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 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 1，表示开启 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
事务 ID 记录	参数：audit_xid_info 默认值为 0，表示关闭审计日志记录事务 ID 功能。

安全相关参数及说明请参见表 7-15。

表 7-15 安全相关参数及说明

参数名	说明
ssl	指定是否启用 SSL 连接。
require_ssl	指定服务器端是否强制要求 SSL 连接。

ssl_ciphers	指定 SSL 支持的加密算法列表。
ssl_cert_file	指定包含 SSL 服务器证书的文件的名称。
ssl_key_file	指定包含 SSL 私钥的文件名称。
ssl_ca_file	指定包含 CA 信息的文件的名称。
ssl_crl_file	指定包含 CRL 信息的文件的名称。
password_policy	指定是否进行密码复杂度检查。
password_reuse_time	指定是否对新密码进行可重用天数检查。
password_reuse_max	指定是否对新密码进行可重用次数检查。
password_lock_time	指定帐户被锁定后自动解锁的时间。
failed_login_attempts	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	指定采用何种加密方式对用户密码进行加密存储。
password_min_uppercase	密码中至少需要包含大写字母的个数。
password_min_lowercase	密码中至少需要包含小写字母的个数。
password_min_digital	密码中至少需要包含数字的个数。
password_min_special	密码中至少需要包含特殊字符的个数。
password_min_length	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	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。
password_effect_time	密码的有效期限。
password_notify_time	密码到期提醒的天数。

audit_enabled	控制审计进程的开启和关闭。
audit_directory	审计文件的存储目录。
audit_data_format	审计日志文件的格式，当前仅支持二进制格式（binary）。
audit_rotation_interval	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	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	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	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	审计文件占用磁盘空间的最大值。
audit_file_remain_threshold	审计目录下审计文件的最大数量。
audit_login_logout	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	指定是否审计数据库启动、停止、切换和恢复的操作。
audit_user_locked	指定是否审计数据库用户的锁定和解锁。
audit_user_violation	指定是否审计数据库用户的越权访问操作。
audit_grant_revoke	指定是否审计数据库用户权限授予和回收的操作。
audit_system_object	指定是否审计数据库对象的 CREATE、DROP、ALTER 操作。
audit_dml_state	指定是否审计具体表的 INSERT、UPDATE、DELETE 操作。
audit_dml_state_select	指定是否审计 SELECT 操作。
audit_copy_exec	指定是否审计 COPY 操作。
audit_function_exec	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	指定是否审计 SET 操作。
enableSeparationOfDuty	指定是否开启三权分立。
session_timeout	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	认证加密信息生成过程中使用的迭代次数。

操作步骤

例如在分布式集群中执行如下操作：

步骤 1 以操作系统用户 gbase 登录数据库 CN 节点。

步骤 2 使用如下命令连接数据库。

```
gsql -d postgres -p 5432
```

postgres 为需要连接的数据库名称，5432 为数据库 CN 节点的端口号。连接成功后，系统显示类似如下信息：

```
gsql ((GBase 8c V5 build a4c7bcc5) compiled at 2024-01-16 14:49:56 commit 0 last mr 1632 )
Non-SSL connection (SSL connection is recommended when requiring high-security)
Last login: 2024-01-19 16:22:27+08 from [local] (user gbase)
Type "help" for help.

postgres=#
```

步骤 3 检查审计总开关状态。

- 用 show 命令显示审计总开关 audit_enabled 的值。

```
postgres=#    SHOW audit_enabled;
```

如果显示为 off，执行“\q”命令退出数据库。

- 执行如下命令开启审计功能，参数设置立即生效。

```
gs_guc set -Z coordinator -N all -I all    -c "audit_enabled=on"
```

其中-Z 参数仅在分布式情况下需要指定。

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，修改配置项的方法如下所示：

```
gs_guc reload -Z coordinator -N all -I all -c "audit_system_object=12295"
```

其中，-Z 参数仅在分布式情况下需要指定，audit_system_object 代表审计项开关，12295 为该审计开关的值。

----结束

7.6.2 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 7.2.4 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。

  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

例如在分布式集群中修改此参数，执行操作如下：

步骤 1 以操作系统用户 gbase 登录数据库 CN 节点。

步骤 2 使用如下命令连接数据库。

```
gsql -d postgres -p 5432
```

postgres 为需要连接的数据库名称，5432 为数据库 CN 节点的端口号。连接成功后，系统显示类似如下信息：

```
gsql ((GBase 8c V5 build a4c7bcc5) compiled at 2024-01-16 14:49:56 commit 0 last mr 1632 )
Non-SSL connection (SSL connection is recommended when requiring high-security)
Last login: 2024-01-19 16:22:27+08 from [local] (user gbase)
Type "help" for help.

postgres=#
```

步骤 3 查询审计记录。

```
postgres=# select * from pg_query_audit('2022-06-06 16:00:00','2022-06-06 17:00:00');
```

查询结果如下：

```
                time                 |        type          | result | userid | username | database
| client_conninfo |     object_name     |
                                        detail_info
                                            | node_name |                thread_id                |
local_port | remote_port
------------------------+---------------+--------+--------+----------+----------
+-----------------+-----------------+-------------------------------------------
-------------------------------------------------------------------------------
--------------------------------+-----------+---------------------------------+-
-----------+-------------
 2022-06-06 16:00:00+08 | login_success | ok         | 10         | gbase       | postgres
| gs_ctl@XX.X.X.X | postgres              | login db(postgres) success,the current use
r is:gbase, SSL=off
                                            | hg1           | 140125749114624@707817600787469 |
5601           | 40412
......
```

该条记录表明，用户 gbase 在 time 字段标识的时间点登录数据库 postgres。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。

说明

对于登录操作的记录，审计日志 detail_info 结尾会记录 SSL 信息，SSL=on 表示客户端通过 SSL 连接，SSL=off 表示客户端没有通过 SSL 连接。

----结束

7.6.3 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与其含义请参见表 7-16。

表 7-16 审计日志相关配置参数

配置项	含义	默认值
audit_directory	审计文件的存储目录。	/var/log/gaussdb/ 用 户 名 /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_thresh old	审计目录下审计文件的最大数量。	1048576

说明

如果使用 gs_om 工具部署数据库，则审计日志路径为“/var/log/gaussdb/用户名/pg_audit”。

- 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。

- 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 7-17 所示。

表 7-17 审计日志保存方式比较

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，GBase 8c 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

例如在分布式集群中修改此参数，执行操作如下：

步骤 1 以操作系统用户 gbase 登录数据库 CN 节点。

步骤 2 使用如下命令连接数据库。

```
gsql -d postgres -p 5432
```

postgres 为需要连接的数据库名称，5432 为数据库 CN 节点的端口号。连接成功后，系统显示类似如下信息：

```
gsql ((GBase 8c V5 build a4c7bcc5) compiled at 2024-01-16 14:49:56 commit 0 last mr 1632 )
Non-SSL connection (SSL connection is recommended when requiring high-security)
Last login: 2024-01-19 16:22:27+08 from [local] (user gbase)
Type "help" for help.

postgres=#
```

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。

```
postgres=# 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. 建议执行如下命令设置成默认值 1024MB。

```
gs_guc reload -Z coordinator -N all -I all -c "audit_space_limit=1024MB"
```

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。

```
postgres=# 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. 建议执行如下命令设置成默认值 1048576。

```
gs_guc reload -Z coordinator -N all -I all -c "audit_file_remain_threshold=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）。

```
postgres=# 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：

```
postgres=# SELECT pg_delete_audit('2012-09-20 00:00:00','2012-09-21 23:59:59');
```

----结束

7.6.4 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。

  数据库部署时通过创建 xml 配置文件中的 tmpMppdbPath 参数指定目录（若未指定，则默认创建/tmp/$USER_mppdb 目录）来存放“.s.PGSQL.*”文件，该目录 和文件权限设置为 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 7-18 所示。

表 7-18 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
gs_initdb	bin	0700
gs_dump	bin	0700
gs_ctl	bin	0700
gs_guc	bin	0700
gsql	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。

7.6.5 统一审计策略概述

背景信息

传统审计会产生大量的审计日志，且不支持定制化的访问对象和访问来源配置，不方便数据库安全管理员对审计日志的分析。而统一审计策略支持绑定资源标签、配置数据来源输出审计日志，可以提升安全管理员对数据库监控的效率。

操作步骤

请参见《安全加固指南》中“数据库配置 > 数据库安全管理策略 > 统一审计”章节。

7.7 设置密态等值查询

7.7.1 密态等值查询概述

随着企业数据上云，数据的安全隐私保护面临越来越严重的挑战。密态数据库将解决数据整个生命周期中的隐私保护问题，涵盖网络传输、数据存储以及数据运行态；更进一步，密态数据库可以实现云化场景下的数据隐私权限分离，即实现数据拥有者和实际数据管理者的数据读取能力分离。密态等值查询将优先解决密文数据的等值类查询问题。密态等值查询目前支持客户端工具 gsql 和 JDBC。接下来分别介绍如何使用客户端工具执行密态等值查询的相关操作。

7.7.2 使用 gsql 操作密态数据库

步骤 1 以操作系统用户 gbase 登录数据库主节点。

步骤 2 执行以下命令打开密态开关，连接密态数据库。

```
gsql -p PORT postgres -r -C
```

步骤 3 创建客户端主密钥 CMK 和列加密密钥 CEK。具体涉及到的新增创建 CMK 的语法参考《GBase 8c V5_3.0.1_SQL 手册》CREATE CLIENT MASTER KEY 章节，创建的 CEK 的语法参考《GBase 8c V5_3.0.1_SQL 手册》 CREATE COLUMN ENCRYPTION KEY

```
--创建客户端加密主密钥(CMK)
postgres=# CREATE CLIENT MASTER KEY ImgCMK1 WITH (KEY_STORE = localkms,
KEY_PATH ="key_path_value1", ALGORITHM = RSA_2048);
postgres=# CREATE CLIENT MASTER KEY ImgCMK WITH (KEY_STORE = localkms,
KEY_PATH ="key_path_value2", ALGORITHM = RSA_2048);
postgres=# CREATE COLUMN ENCRYPTION KEY ImgCEK1 WITH VALUES
(CLIENT_MASTER_KEY = ImgCMK1, ALGORITHM =
AEAD_AES_256_CBC_HMAC_SHA256);
CREATE COLUMN ENCRYPTION KEY
```

查询存储密钥信息的系统表结果如下。

```
postgres=# SELECT * FROM gs_client_global_keys;
global_key_name | key_namespace | key_owner |    key_acl |     create_date
----------------+---------------+-----------+---------+---------------------------
imgcmk1         | 2200          |     10    | 2022-06-06 11:04:00.656617
imgcmk          | 2200          |     10    | 2022-06-06 11:04:05.389746
```

```
(2 rows)
postgres=# SELECT column_key_name,column_key_distributed_id ,global_key_id,key_owner
FROM gs_column_keys;
column_key_name | column_key_distributed_id | global_key_id | key_owner
-----------------+---------------------------+---------------+-----------
imgcmk1        |                  |     10    |   2022-06-06 11:04:00.656617
imgcmk         |                  |     10    |   2022-06-06 11:04:05.389746
(2 rows)
postgres=# SELECT column_key_name,column_key_distributed_id ,global_key_id,key_owner
FROM gs_column_keys;
column_key_name | column_key_distributed_id | global_key_id | key_owner
-----------------+---------------------------+---------------+-----------
imgcek1      |     760411027    |              16392          |       10
imgcek       |     3618369306 |              16398          |       10
(2 rows)
```

步骤 4 创建加密表。

```
postgres=# CREATE TABLE creditcard_info (id_number       int, name   text encrypted with
(column_encryption_key = ImgCEK, encryption_type = DETERMINISTIC), credit_card
varchar(19) encrypted with (column_encryption_key = ImgCEK1, encryption_type =
DETERMINISTIC));
NOTICE: The 'DISTRIBUTE BY' clause is not specified. Using 'id_number' as the distribution
column by default.
HINT: Please use 'DISTRIBUTE BY' clause to specify suitable data distribution column.
CREATE TABLE
```

查询表的详细信息如下，Modifiers 值为 encrypted 则表示该列是加密列。

```
postgres=# \d creditcard_info
Table "public.creditcard_info" Column        |       Type| Modifiers
-------------+-------------------+------------
id_number | integer
name      | text
credit_card | character varying | encrypted
```

步骤 5 向加密表插入数据并进行等值查询。

```
postgres=# INSERT INTO creditcard_info VALUES (1,'joe','6217986500001288393');
INSERT 0 1
postgres=# INSERT INTO creditcard_info VALUES (2, 'joy','6219985678349543233');
INSERT 0 1
postgres=# select * from creditcard_info where name = 'joe';
id_number | name |       credit_card
```

```
-----------+------+--------------------
1 | joe | 6217986500001288393
(1 row)
```

注意：使用非密态客户端查看该加密表数据时是密文

```
postgres=# select id_number,name from creditcard_info;
id_number |   name
-----------+--------------------
1 |
\x011aefabd754ded0a536a96664790622487c4d366d313aecd5839e410a46d29cba96a60e48310
00000ee7905 6a114c9a6c041bb552b78052e912a8b730609142074c63791abebd0d38
2 |
\x011aefabd76853108eb406c0f90e7c773b71648fa6e2b8028cf634b49aec65b4fcfb376f3531000
000f7471c868 6682de215d09aa87113f6fb03884be2031ef4dd967afc6f7901646b
(2 rows)
```

步骤 6 （可选）对加密表进行 alter 和 update 操作。

```
postgres=# ALTER TABLE creditcard_info ADD COLUMN age int ENCRYPTED WITH
(COLUMN_ENCRYPTION_KEY = ImgCEK, ENCRYPTION_TYPE = DETERMINISTIC);
ALTER TABLE
postgres=# \d creditcard_info
Table "public.creditcard_info" Column      | Type| Modifiers
-------------+-------------------+------------
id_number | integer
name      | text
credit_card | character varying | encrypted
age       | integer       | encrypted
postgres=# ALTER TABLE creditcard_info DROP COLUMN age;
ALTER TABLE
postgres=# update creditcard_info set credit_card = '54320000011111111' where name = 'joy';
UPDATE 1
postgres=# select * from creditcard_info where name = 'joy';
id_number | name | credit_card
-----------+------+------------------
2 | joy | 54320000011111111
(1 row)
```

----结束

7.7.3 使用 JDBC 操作密态数据库

连接密态数据库

连接密态数据库需要使用驱动包 gsjdbc4.jar，具体 JDBC 连接参数参考基于 JDBC 开发章节介绍。JDBC 支持密态数据库相关操作，需要设置 enable_ce=1，示例如下。

```
public static Connection getConnect(String username, String passwd)
{
//驱动类。
String driver = "org.postgresql.Driver";
//数据库连接描述符。
String sourceURL = "jdbc:postgresql://10.10.0.13:5432/postgres?enable_ce=1"; Connection
conn = null;
try
{
//加载驱动。Class.forName(driver);
}
catch( Exception e )
{
e.printStackTrace(); return null;
}
try
{
//创建连接。
conn = DriverManager.getConnection(sourceURL, username, passwd);
System.out.println("Connection succeed!");
}
catch(Exception e)
{
e.printStackTrace(); return null;
}
return conn;
};
```

- 【建议】使用 JDBC 操作密态数据库时，一个数据库连接对象对应一个线程，否则，不同线程变更可能导致冲突。
- 【建议】使用 JDBC 操作密态数据库时，不同 connection 对密态配置数据有变更，由客户端调用 isvalid 方法保证 connection 能够持有变更后的密态配置数据，此时需要保证参数 refreshClientEncryption 为 1(默认值为 1)，在单客户端操作密态数据场景下，refreshClientEncryption 参数可以设置为 0。

使用 isValid 方法刷新缓存示例

```
// 创建客户端主密钥
```

```
Connection conn1 = DriverManager.getConnection("url","user","password");
// conn1 通过调用 isValid 刷新缓存 try {
if (!conn1.isValid(60)) {
System.out.println("isValid Failed for connection 1");
}
} catch (SQLException e) { e.printStackTrace();
return null;
}
```

执行密态等值查询相关的创建密钥语句

```
// 创建客户端主密钥
Connection conn = DriverManager.getConnection("url","user","password"); Statement stmt =
conn.createStatement();
int rc = stmt.executeUpdate("CREATE CLIENT MASTER KEY ImgCMK1 WITH
( KEY_STORE = localkms, KEY_PATH = \"key_path_value\" , ALGORITHM = RSA_2048);
```

说明

创建密钥之前需要使用 gs_ktool 工具提前生成密钥，才能创建 CMK 成功。

```
// 创建列加密密钥
int rc2 = stmt.executeUpdate("CREATE COLUMN ENCRYPTION KEY ImgCEK1 WITH
VALUES (CLIENT_MASTER_KEY = ImgCMK1, ALGORITHM =
AEAD_AES_256_CBC_HMAC_SHA256);");
```

执行密态等值查询相关的创建加密表的语句

```
int rc3 = stmt.executeUpdate("CREATE TABLE creditcard_info (id_number int, name
varchar(50) encrypted with (column_encryption_key = ImgCEK1, encryption_type =
DETERMINISTIC),credit_card varchar(19) encrypted with (column_encryption_key =
ImgCEK1, encryption_type = DETERMINISTIC));");
// 插入数据
int rc4 = stmt.executeUpdate("INSERT INTO creditcard_info VALUES
(1,'joe','6217986500001288393');");
// 查询加密表
ResultSet rs = null;
rs = stmt.executeQuery("select * from creditcard_info where name = 'joe';");
// 关闭语句对象
stmt.close();
```

执行加密表的预编译 SQL 语句

```
// 调用 Connection 的 prepareStatement 方法创建预编译语句对象。
PreparedStatement pstmt = con.prepareStatement("INSERT INTO creditcard_info VALUES
(?, ?, ?);");
```

```
// 调用 PreparedStatement 的 setShort 设置参数。
pstmt.setInt(1, 2); pstmt.setString(2, "joy");
pstmt.setString(3, "6219985678349543233");
// 调用 PreparedStatement 的 executeUpdate 方法执行预编译 SQL 语句。
int rowcount = pstmt.executeUpdate();
// 调用 PreparedStatement 的 close 方法关闭预编译语句对象。
pstmt.close();
```

执行加密表的批处理操作

```
// 调用 Connection 的 prepareStatement 方法创建预编译语句对象。
Connection conn = DriverManager.getConnection("url","user","password");
PreparedStatement pstmt = conn.prepareStatement("INSERT INTO batch_table (id, name,
address) VALUES (?,?,?)");
// 针对每条数据都要调用 setShort 设置参数，以及调用 addBatch 确认该条设置完毕。
int loopCount = 20;
for (int i = 1; i < loopCount + 1; ++i) { statemnet.setInt(1, i);
statemnet.setString(2, "Name " + i); statemnet.setString(3, "Address " + i);
// Add row to the batch. statemnet.addBatch();
}
// 调用 PreparedStatement 的 executeBatch 方法执行批处理。
int[] rowcount = pstmt.executeBatch();
// 调用 PreparedStatement 的 close 方法关闭预编译语句对象。
pstmt.close();
```

7.7.4 密态支持函数/存储过程

密态支持函数/存储过程当前版本只支持 sql 和 PL/pgsql 两种语言。 由于密态支持存储过程中创建和执行函数/存储过程对用户是无感知的，因此语法和非密态无区别。

密态等值查询支持函数存储过程新增系统表 gs_encrypted_proc，用于存储参数返回的原始数据类型。

创建并执行涉及加密列的函数/存储过程

步骤 1 创建密钥，详细步骤请参考 7.4.2 使用 gsql 操作密态数据库和 7.4.3 使用 JDBC 操作密态数据库。

步骤 2 创建加密表。

```
postgres=# CREATE TABLE creditcard_info (
gbase(# id_number int,
gbase(# name text,
```

```
gbase(# credit_card varchar(19) encrypted with (column_encryption_key = ImgCEK1,
encryption_type = DETERMINISTIC)
gbase(# ) with (orientation=row);
CREATE TABLE
```

步骤 3 插入数据。

```
postgres=# insert into creditcard_info values(1, 'Avi', '1234567890123456');
INSERT 0 1
postgres=# insert into creditcard_info values(2, 'Eli', '2345678901234567');
INSERT 0 1
```

步骤 4 创建函数支持密态等值查询。

```
postgres=# CREATE FUNCTION f_encrypt_in_sql(val1 text, val2 varchar(19)) RETURNS
text AS 'SELECT name from creditcard_info where name=$1 or credit_card=$2 LIMIT 1'
LANGUAGE SQL; CREATE FUNCTION
postgres=# CREATE FUNCTION f_encrypt_in_plpgsql (val1 text, val2 varchar(19))
gbase-# RETURNS text AS $$
gbase$# DECLARE
gbase$# c text;
gbase$# BEGIN
gbase$# SELECT into c name from creditcard_info where name=$1 or credit_card =$2 LIMIT
1;
gbase$# RETURN c;
gbase$# END; $$
gbase-# LANGUAGE plpgsql;
CREATE FUNCTION
```

步骤 5 执行函数。

```
postgres=# SELECT f_encrypt_in_sql('Avi','1234567890123456');
f_encrypt_in_sql
-----------------
Avi
(1 row)
postgres=# SELECT f_encrypt_in_plpgsql('Avi', val2=>'1234567890123456');
f_encrypt_in_plpgsql
---------------------
Avi
(1 row)
```

----结束

- 函数/存储过程中的“执行动态查询语句”中的查询是在执行过程中编译，因此函数/存储

过程中的表名、列名不能在创建阶段未知，输入参数不能用于表名、列名或以任何方式连接。

- 在 RETURNS、IN 和 OUT 的参数中，不支持混合使用加密和非加密类型参数。虽然参数类型都是原始数据类型，但实际类型不同。
- 当前版本函数/存储过程的 LANGUAGE 只支持 SQL 和 PL/pgSQL，不支持 C 和 JAVA 等其他过程语言。
- 不支持在函数/存储过程中执行其他查询加密列的函数/存储过程。
- 当前版本不支持 default、DECLARE 中为变量赋予默认值，且不支持对 DECLARE 中的返回值进行解密，用户可以用执行函数时用输入参数，输出参数来代替使用。
- 不支持 gs_dump 对涉及加密列的 function 进行备份。
- 不支持在函数/存储过程中创建密钥。
- 该版本密态函数/存储过程不支持触发器
- 密态等值查询函数/存储过程不支持对 plpgsql 语言对语法进行转义，对于语法主体带有引号的语法 CREATE FUNCTION AS ‘语法主体'，可以用 CREATE FUNCTION AS $$语法主体$$代替。
- 不支持在密态等值查询函数/存储过程中执行修改加密列定义的操作，包括对创建加密表，添　加加密列，由于执行函数是在服务端，客户端没法判断是否是需要刷新缓存，得断开连接后或触发刷新客户端加密列缓存才可以对该列做加密操作。

7.8 设置账本数据库

7.8.1 账本数据库概述

背景信息

账本数据库融合了区块链思想，将用户操作记录至两种历史表中：用户历史表和全局区块表。全局区块表存储在 CN 端，各个 CN 数据独立。用户历史表存储在 DN 端，历史表记录的数据为所在 DN 防篡改表的数据变化。

当用户创建防篡改用户表时，系统将自动为该表添加一个 hash 列来保存每行数据的 hash 摘要信息，同时在 blockchain 模式下会创建一张用户历史表来记录对应用户表中每条数据的变更行为；而用户对防篡改用户表的一次修改行为将记录至全局区块表中。由于历史表具有

只可追加不可修改的特点，因此历史表记录串联起来便形成了用户对防篡改用户表的修改历史。创建防篡改用户表的操作步骤如下：

步骤 1 创建防篡改模式。

例如，创建防篡改模式 ledgernsp。

```
postgres=# CREATE SCHEMA ledgernsp WITH BLOCKCHAIN;
```

步骤 2 在防篡改模式下创建防篡改用户表。

例如，创建防篡改用户表 ledgernsp.usertable。

```
postgres=# CREATE TABLE ledgernsp.usertable(id int, name text);
```

查看防篡改用户表结构及其对应的用户历史表结构。

```
postgres=# \d+ ledgernsp.usertable;
postgres=# \d+ blockchain.ledgernsp_usertable_hist;
```

执行结果如下：

```
postgres=# \d+ ledgernsp.usertable;
Table "ledgernsp.usertable"
Column | Type | Modifiers | Storage | Stats target | Description
--------+---------+-----------+----------+--------------+-------------
id | integer |  | plain |
name | text |  | extended |
hash | hash16| plain |
Has OIDs: no
Distribute By: HASH(id)
Location Nodes: ALL DATANODES Options: orientation=row, compression=no
History table name: ledgernsp_usertable_hist
postgres=# \d+ blockchain.ledgernsp_usertable_hist;
 Table "blockchain.ledgernsp_usertable_hist"
Column | Type | Modifiers | Storage | Stats target | Description
----------+--------+-----------+---------+--------------+-------------
rec_num | bigint |  | plain |
hash_ins | hash16 |  | plain  |
hash_del | hash16 |  | plain  |
pre_hash | hash32 |  | plain |
Indexes:
"gs_hist_16388_index" PRIMARY KEY, btree (rec_num int4_ops) TABLESPACE
pg_default Has OIDs: no
Options: internal_mask=263
```

- 防篡改表不支持非行存表、临时表、外表、unlog 表，非行存表均无防篡改属性。
- 防篡改表在创建时会自动增加一个名为 hash 的系统列，所以防篡改表单表最大列数为 1599。

步骤 3 修改防篡改用户表数据。

例如，对防篡改用户表执行 INSERT/UPDATE/DELETE。

```
postgres=# INSERT INTO ledgernsp.usertable VALUES(1, 'alex'), (2, 'bob'), (3,
'peter');
INSERT 0 3
postgres=# SELECT *, hash FROM ledgernsp.usertable ORDER BY id;
id | name |     hash
----+------+------------------
1 | alex  | 1f2e543c580cb8c5
2 | bob   | 8fcd74a8a6a4b484
3 | peter | f51b4b1b12d0354b
(3 rows)
postgres=# UPDATE ledgernsp.usertable SET name = 'bob2' WHERE id = 2;
UPDATE 1
postgres=# SELECT *, hash FROM ledgernsp.usertable ORDER BY id;
 id | name |    hash
----+------+------------------
1 | alex  | 1f2e543c580cb8c5
2 | bob2  | 437761affbb7c605
3 | peter | f51b4b1b12d0354b
(3 rows)
postgres=# DELETE FROM ledgernsp.usertable WHERE id = 3;
DELETE 1
postgres=# SELECT *, hash FROM ledgernsp.usertable ORDER BY id;
 id | name |    hash
----+------+------------------
1 | alex | 1f2e543c580cb8c5
2 | bob2 | 437761affbb7c605
(2 rows)
```

----结束

7.8.2 查看账本历史操作记录

前提条件

- 系统中需要有审计管理员或者具有审计管理员权限的角色。

- 数据库正常运行，并且对防篡改数据库执行了一系列增、删、改等操作，保证在查询时段内有账本操作记录结果产生。

背景信息

- 只有拥有 AUDITADMIN 属性的用户才可以查看账本历史操作记录。有关数据库用户及创建用户的办法请参见 7.2.4 用户。
- 查询全局区块表命令是直接查询 gs_global_chain 表，操作为：

```
SELECT * FROM gs_global_chain;
```

该表有 11 个字段，每个字段的含义见章节 20.2.13 GS_GLOBAL_CHAIN。

- 查询用户历史表的命令是直接查询 BLOCKCHAIN 模式下的用户历史表，操作为： 例如用户表所在的模式为 ledgernsp，表名为 usertable，则对应的用户历史表名为 blockchain.ledgernsp_usertable_hist;

```
SELECT * FROM blockchain.ledgernsp_usertable_hist;
```

用户历史表有 4 个字段，每个字段的含义见表 7-19。

说明

用户历史表的表名一般为 blockchain.<schemaname>_<tablename>_hist 形式。当防篡改用户表模式名或者表名过长导致前述方式生成的表名超出表名长度限制，则会采用 blockchain.<schema_oid>_<table_oid>_hist 的方式命名。

操作步骤

步骤 1 连接 CN 节点，登录数据库查询全局区块表记录。

```
postgres=# SELECT * FROM gs_global_chain;
```

例如，查询结果如下：

```
blocknum | dbname | username |  starttime | relid | relnsp  | relname |
   relhash | globalhash txcommand
---------+---------+---------+----------------------------+-------+----
------+----------+-----------------+----------------------------------+---
--------------
-------------------------------------------------------------
0 | postgres | gbase   | 2021-04-14 07:00:46.32757+08 | 16393 | ledgernsp |
usertable | a41714001181a294 | 6b5624e039e8aee36bff3e8295c75b40 | insert into
ledge
rnsp.usertable values(1, 'alex'), (2, 'bob'), (3, 'peter');
```

```
1 | postgres | gbase   | 2021-04-14 07:01:19.767799+08 | 16393 | ledgernsp |
usertable | b3a9ed0755131181 | 328b48c4370faed930937869783c23e0 | update
ledgernsp.
usertable set name = 'bob2' where id = 2;
2 | postgres | gbase   | 2021-04-14 07:01:29.896148+08 | 16393 | ledgernsp |
usertable | 0ae4b4e4ed2fcab5 | aa8f0a236357cac4e5bc1648a739f2ef | delete from
ledge
rnsp.usertable where id = 3;
```

该结果表明，用户 gbase 连续执行了三条 DML 命令，包括 INSERT、UPDATE 和 DELETE 操作。

步骤 2 查询历史表记录。

```
postgres=# SELECT * FROM blockchain.ledgernsp_usertable_hist;
```

查询结果如下：

```
rec_num |  hash_ins    |   hash_del    |   pre_hash
---------+------------------+-----------------+------------------------------
----
0  | 1f2e543c580cb8c5 |                |e1b664970d925d09caa295abd38d9b35
1  | 8fcd74a8a6a4b484 |                |dad3ed8939a141bf3682043891776b67
2  | f51b4b1b12d0354b |            | 53eb887fc7c4302402343c8914e43c69
3  | 437761affbb7c605 | 8fcd74a8a6a4b484 | c2868c5b49550801d0dbbbaa77a83a10
4  |                | f51b4b1b12d0354b | 9c512619f6ffef38c098477933499fe3
(5 rows)
```

查询结果显示，用户 gbase 对 ledgernsp.usertable 表插入了 3 条数据，更新了 1 条数据，随后删除了 1 行数据，最后剩余 2 行数据，hash 值分别为 1f2e543c580cb8c5 和 437761affbb7c605。

步骤 3 查询用户表数据及 hash 校验列。

```
postgres=# SELECT *, hash FROM ledgernsp.usertable;
```

查询结果如下：

```
id | name |    hash
----+------+------------------
1 | alex | 1f2e543c580cb8c5
2 | bob2 | 437761affbb7c605
(2 rows)
```

查询结果显示，用户表中剩余 2 条数据，与步骤 2 中的记录一致。

----结束

7.8.3 校验账本数据一致性

前提条件

数据库正常运行，并且对防篡改数据库执行了一系列增、删、改等操作，保证在查询时段内有账本操作记录结果产生。

背景信息

- 账本数据库校验功能目前提供两种校验接口，分别为： ledger_hist_check(text,...和 ledger_gchain_check(text...。普通用户调用校验接口，仅能校验自己有权限访问的表。
- 校验防篡改用户表和用户历史表的接口为 pg_catalog.ledger_hist_check，操作 为：

```
SELECT pg_catalog.ledger_hist_check(schema_name text,table_name text);
```

如果校验通过，函数返回 t，反之则返回 f。

- 校验防篡改用户表、用户历史表和全局区块表三者是否一致的接口为 pg_catalog.ledger_gchain_check，操作为：

```
SELECT pg_catalog.ledger_gchain_check(schema_name text, table_name text);
```

如果校验通过，函数返回 t，反之则返回 f。

操作步骤

步骤 1 校验防篡改用户表 ledgernsp.usertable 与其对应的历史表是否一致。

```
postgres=# SELECT pg_catalog.ledger_hist_check('ledgernsp', 'usertable');
```

查询结果如下：

```
ledger_hist_check
------------------
t
(1 row)
```

该结果表明：防篡改用户表和用户历史表中记录的结果能够一一对应，保持一致。

步骤 2 查询防篡改用户表 ledgernsp.usertable 与其对应的历史表以及全局区块表中关于该表的记录是否一致。

```
postgres=# SELECT pg_catalog.ledger_gchain_check('ledgernsp', 'usertable');
```

查询结果如下：

```
ledger_gchain_check
--------------------
t
(1 row)
```

查询结果显示，上述三表中关于 ledgernsp.usertable 的记录保持一致，未发生篡改行为。

----结束

7.8.4 归档账本数据库

前提条件

- 系统中需要有审计管理员或者具有审计管理员权限的角色。
- 数据库正常运行，并且对防篡改数据库执行了一系列增、删、改等操作，保证在查询时段内有账本操作记录结果产生。
- 数据库已经正确配置审计文件的存储路径 audit_directory。

背景信息

- 账本数据库归档功能目前提供两种校验接口，分别为：ledger_hist_archive(text, text)和 ledger_gchain_archive(void)，详见《GBase 8c V5_3.0.1_SQL 参考》中“5.21 账本数据库的函数”章节。账本数据库接口仅审计管理员可以调用。
- 归档用户历史表的接口为 pg_catalog.ledger_hist_archive，表示归档当前 DN 节点的用户历史表数据。操作为：

```
SELECT pg_catalog.ledger_hist_archive(schema_name text,table_name text);
```

如果归档成功，函数返回 t，反之则返回 f。

- 归档全局区块表的接口为 pg_catalog.ledger_gchain_archive，表示归档当前 CN 节点的全局历史表数据。操作为：

```
SELECT pg_catalog.ledger_gchain_archive();
```

如果归档成功，函数返回 t，反之则返回 f。

操作步骤

步骤 1 使用 EXECUTE DIRECT 对某个 DN 节点进行归档操作。例如：

```
postgres=# EXECUTE DIRECT ON (dn1) 'SELECT
pg_catalog.ledger_hist_archive("ledgernsp", "usertable");';
```

执行结果如下：

```
ledger_hist_archive
--------------------
t
(1 row)
```

用户历史表将归档为一条数据：

```
postgres=# EXECUTE DIRECT ON (datanode1) 'SELECT * FROM
blockchain.ledgernsp_usertable_hist;';
rec_num |      hash_ins |       hash_del |       pre_hash
---------+------------------+------------------+--------------------------------
3 | e78e75b00d396899 | 8fcd74a8a6a4b484 | fd61cb772033da297d10c4e658e898d7
(1 row)
```

该结果表明当前节点用户历史表导出成功。

步骤 2 连接登录 CN 节点，执行全局区块表导出操作。

```
postgres=# SELECT pg_catalog.ledger_gchain_archive();
```

执行结果如下：

```
ledger_gchain_archive
-----------------------
t
(1 row)
```

全局历史表将以用户表为单位归档为 N(用户表数量)条数据：

```
postgres=# SELECT * FROM gs_global_chain;
blocknum |      dbname     | username |   starttime | relid |    relnsp      |    relname |       relhash |
    globalhash       | txcommand
----------+----------+----------+-----------------------------+-------+-----------+-----------+-------------
-----+----------------------------------+-----------
1 | postgres | libc    | 2021-05-10 19:59:38.619472+08 | 16388 | ledgernsp | usertable |
57c101076694b415 | be82f98ee68b2bc4e375f69209345406 | Archived.
(1 row)
```

该结果表明，当前节点全局区块表导出成功。

----结束

7.8.5 修复账本数据库

前提条件

- 系统中需要有审计管理员或者具有审计管理员权限的角色。

- 数据库正常运行，并且对防篡改数据库执行了一系列增、删、改等操作，保证在查询时段内有账本操作记录结果产生。

背景信息

- 当前的账本数据库机制为：全局区块表存储在 CN 端，各个 CN 数据独立。用户历史表存储在 DN 端，历史表记录的数据为所在 DN 防篡改表的数据变化。当在异常情况或表被损坏时,需要使用 ledger_hist_repair(text, text)或 ledger_gchain_repair(text, text)接口对全局区块表或用户历史表进行修复，修复后调用全局区块表或用户历史表校验接口结果为 true。
- 在触发数据重分布时，可能导致防篡改表和用户历史表数据不一致，此时需要使用 ledger_hist_repair(text, text)接口对指定 DN 节点的用户历史表进行修复。

```
SELECT pg_catalog.ledger_hist_repair(schema_name text,table_name text);
```

修复后当前 DN 节点调用历史表校验接口结果为 true。

- 在 CN 剔除、修复的场景下，可能导致全局区块表数据丢失或者与用户历史表不一致，此时需要使用 ledger_gchain_repair(text, text)接口，对整个集群范围内的全局区块表进行修复。

```
SELECT pg_catalog.ledger_gchain_repair(schema_name text,table_name text);
```

修复后调用全局区块表校验接口结果为 true。

操作步骤

步骤 1 使用 EXECUTE DIRECT 对某个 DN 节点，执行历史表修复操作。

```
postgres=# SELECT pg_catalog.ledger_hist_repair('ledgernsp', 'usertable');
```

查询结果如下：

```
ledger_hist_repair
--------------------
84e8bfc3b974e9cf
 (1 row)
```

该结果表明当前节点用户历史表修复成功，修复造成的用户历史表 hash 增量为 84e8bfc3b974e9cf。

步骤 2 连接 CN 登录数据库，执行全局区块表修复操作。

```
postgres=# SELECT pg_catalog.ledger_gchain_repair('ledgernsp', 'usertable');
```

查询结果如下：

```
ledger_gchain_repair
---------------------
a41714001181a294
(1 row)
```

该结果表明，全局区块表修复成功，且插入一条修复数据，其 hash 值为 a41714001181a294。

----结束

7.9 强制访问控制

强制访问控制（Mandatory Access Control，缩写 MAC）是指一种由操作系统约束的访问控制，目标是限制主体或发起者访问或对对象或目标执行某种操作的能力。在实践中，主体通常是一个进程或线程，对象可能是文件、目录、TCP/UDP 端口、共享内存段、I/O 设备等。主体和对象各自具有一组安全属性。每当主体尝试访问对象时，都会由操作系统内核强制施行授权规则——检查安全属性并决定是否可进行访问。任何主体对任何对象的任何操作都将根据一组授权规则（也称策略）进行测试，决定操作是否允许。

为实现强制访问控制功能，GBase 8c 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等级。通过设定主客体间安全标记的比较规则来控制数据库用户对数据库对象的强制访问控制。

行级强制访问控制就是主体访问客体每一行数据时都会进行是否符合安全规则的判断，如果符合就返回读取结果或者修改数据，不符合就是返回空或者不修改。GBase8c 中的强制访问控制基于行级安全策略(RLS,Row-Level Security)和 Security Label 实现，Security Label 能够为一个数据库对象(object)指定一个 Security Label。

- 主客体

  主体对象包括数据库用户。

  客体对象包括数据表、视图、物化视图、存储过程、函数、序列、触发器等。

- 等级说明：

敏感标记由等级分类和非等级类别两部分组成，组成形式用冒号(**“:”**)分割，如：等级分类组：范畴组

其中:等级分类组由多个等级分类组成，范畴组由多个范畴组成。等级分类和范畴之间用逗号(“,”)分隔，用点号(“.”)表示区间。

- 等级分类

设置 16 个等级分类，以 L 开头，后面的数字[1-16]之间，如：L1。等级标记需满足偏序关系。

- 范畴

设置 32 个范畴，以 G 开头，后面的数字[1-32]之间，如：G1。范畴无法进行大小比较，但可以进行集合运算，例如范畴(G1,G3)包含 G1。

- 控制规则：

基于敏感标记实现的强制访问控制，对于数据库对象的读写规则需要满足：

强制访问控制策略	描述
表 insert 策略	主体等级标记小于表等级标记，且主体范畴是表范畴子集则允许插入。
表 update 策略	主体等级标记与元组的标记一致，则允许修改元组。
表 delete 策略	同 update 策略
表 select 策略	主体等级标记大于等于元组等级标记，且主体范畴是元组范畴超集，则允许查询。
存储过程、触发器	当主体等级等于函数/存储过程安全等级标记，且主体范畴是函数/存储过程范畴超集，则允许执行。

规则配置：

(1) 修改配置文件，将 ogmac_enable_mac 参数设置为 true。

(2) 通过 sql 创建安全标签

① 给主体打安全标签

② 给客体打安全标签

配置示例：

(1) 创建用户和表，执行命令：

```
create table t1 (id int, info text);create user user1 password 'user1@abcd';
```

(2) 创建安全标签，执行命令：

```
create security label l1 'L3:G1,G2';
```

(3) 为客体打安全标签，执行命令：

```
security label on table t1 is 'l1';
```

(4) 为主体打安全标签，执行命令：

```
security label on role user1 is 'l1';
```

注意

- 给表打标签的时候，表必须为空，否则报错。

8 接口参考

8.1 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。

8.1.1 java.sql.Connection

java.sql.Connection 是数据库连接接口。

表 8-1 对 java.sql.Connection 接口的支持情况

方法名	返回值类型	支持 JDBC 4
abort(Executor executor)	void	Yes
clearWarnings()	void	Yes
close()	void	Yes
commit()	void	Yes
createArrayOf(String typeName, Object[] elements)	Array	Yes
createBlob()	Blob	Yes
createClob()	Clob	Yes
createSQLXML()	SQLXML	Yes
createStatement()	Statement	Yes
createStatement(int resultSetType, int	Statement	Yes

resultSetConcurrency)		
createStatement(int resultSetType, int resultSetConcurrency, int resultSetHoldability)	Statement	Yes
getAutoCommit()	Boolean	Yes
getCatalog()	String	Yes
getClientInfo()	Properties	Yes
getClientInfo(String name)	String	Yes
getHoldability()	int	Yes
getMetaData()	DatabaseMetaData	Yes
getNetworkTimeout()	int	Yes
getSchema()	String	Yes
getTransactionIsolation()	int	Yes
getTypeMap()	Map<String,Class<?>>	Yes
getWarnings()	SQLWarning	Yes
isClosed()	Boolean	Yes
isReadOnly()	Boolean	Yes
isValid(int timeout)	boolean	Yes
nativeSQL(String sql)	String	Yes
prepareCall(String sql)	CallableStatement	Yes
prepareCall(String sql, int resultSetType, int resultSetConcurrency)	CallableStatement	Yes
prepareCall(String sql, int resultSetType, int resultSetConcurrency, int	CallableStatement	Yes

resultSetHoldability)		
prepareStatement(String sql)	PreparedStatement	Yes
prepareStatement(String sql, int autoGeneratedKeys)	PreparedStatement	Yes
prepareStatement(String sql, int[] columnIndexes)	PreparedStatement	Yes
prepareStatement(String sql, int resultSetType, int resultSetConcurrency)	PreparedStatement	Yes
prepareStatement(String sql, int resultSetType, int resultSetConcurrency, int resultSetHoldability)	PreparedStatement	Yes
prepareStatement(String sql, String[] columnNames)	PreparedStatement	Yes
releaseSavepoint(Savepoint savepoint)	void	Yes
rollback()	void	Yes
rollback(Savepoint savepoint)	void	Yes
setAutoCommit(boolean autoCommit)	void	Yes
setClientInfo(Properties properties)	void	Yes
setClientInfo(String name,String value)	void	Yes
setHoldability(int holdability)	void	Yes
setNetworkTimeout (Executor executor, int milliseconds)	void	Yes
setReadOnly(boolean readOnly)	void	Yes
setSavepoint()	Savepoint	Yes
setSavepoint(String name)	Savepoint	Yes

setSchema(String schema)	void	Yes
setTransactionIsolation(int level)	void	Yes
setTypeMap(Map<String, Class<?>> map)	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。

8.1.2 java.sql.CallableStatement

java.sql.CallableStatement 是存储过程执行接口。

表 8-2 对 java.sql.CallableStatement 的支持情况

方法名	返回值类型	支持 JDBC 4
getArray(int parameterIndex)	Array	Yes
getBigDecimal(int parameterIndex)	BigDecimal	Yes
getBlob(int parameterIndex)	Blob	Yes
getBoolean(int parameterIndex)	boolean	Yes
getByte(int parameterIndex)	byte	Yes
getBytes(int parameterIndex)	byte[]	Yes
getClob(int parameterIndex)	Clob	Yes
getDate(int parameterIndex)	Date	Yes
getDate(int parameterIndex, Calendar cal)	Date	Yes
getDouble(int parameterIndex)	double	Yes
getFloat(int parameterIndex)	float	Yes

getInt(int parameterIndex)	int	Yes
getLong(int parameterIndex)	long	Yes
getObject(int parameterIndex)	Object	Yes
getObject(int parameterIndex, Class<T> type)	Object	Yes
getShort(int parameterIndex)	short	Yes
getSQLXML(int parameterIndex)	SQLXML	Yes
getString(int parameterIndex)	String	Yes
getNString(int parameterIndex)	String	Yes
getTime(int parameterIndex)	Time	Yes
getTime(int parameterIndex, Calendar cal)	Time	Yes
getTimestamp(int parameterIndex)	Timestamp	Yes
getTimestamp(int parameterIndex, Calendar cal)	Timestamp	Yes
registerOutParameter(int parameterIndex, int type)	void	Yes
registerOutParameter(int parameterIndex, int sqlType, int type)	void	Yes
wasNull()	Boolean	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 。
- registerOutParameter(int parameterIndex, int sqlType, int type)方法仅用于注册复合数据类型，其它类型不支持。

8.1.3 java.sql.DatabaseMetaData

java.sql.DatabaseMetaData 是数据库对象定义接口。

表 8-3 对 java.sql.DatabaseMetaData 的支持情况

方法名	返回值类型	支持 JDBC 4
allProceduresAreCallable()	boolean	Yes
allTablesAreSelectable()	boolean	Yes
autoCommitFailureClosesAll ResultSets()	boolean	Yes
dataDefinitionCausesTransac tionCommit()	boolean	Yes
dataDefinitionIgnoredInTran sactions()	boolean	Yes
deletesAreDetected(int type)	boolean	Yes
doesMaxRowSizeIncludeBlob s()	boolean	Yes
generatedKeyAlwaysReturne d()	boolean	Yes
getBestRowIdentifier(String catalog, String schema, String table, int scope, boolean nullable)	ResultSet	Yes
getCatalogs()	ResultSet	Yes
getCatalogSeparator()	String	Yes
getCatalogTerm()	String	Yes
getClientInfoProperties()	ResultSet	Yes
getColumnPrivileges(String catalog, String schema, String table, String columnNamePattern)	ResultSet	Yes

getConnection()	Connection	Yes
getCrossReference(String parentCatalog, String parentSchema, String parentTable, String foreignCatalog, String foreignSchema, String foreignTable)	ResultSet	Yes
getDefaultTransactionIsolati on()	int	Yes
getExportedKeys(String catalog, String schema, String table)	ResultSet	Yes
getExtraNameCharacters()	String	Yes
getFunctionColumns(String catalog, String schemaPattern, String functionNamePattern, String columnNamePattern)	ResultSet	Yes
getFunctions(String catalog, String schemaPattern, String functionNamePattern)	ResultSet	Yes
getIdentifierQuoteString()	String	Yes
getImportedKeys(String catalog, String schema, String table)	ResultSet	Yes
getIndexInfo(String catalog, String schema, String table, boolean unique, boolean approximate)	ResultSet	Yes
getMaxBinaryLiteralLength()	int	Yes
getMaxCatalogNameLengt h()	int	Yes
getMaxCharLiteralLength()	int	Yes
getMaxColumnNameLengt h()	int	Yes
getMaxColumnsInGroupBy()	int	Yes
getMaxColumnsInIndex()	int	Yes
getMaxColumnsInOrderBy()	int	Yes
getMaxColumnsInSelect()	int	Yes

getMaxColumnsInTable()	int	Yes
getMaxConnections()	int	Yes
getMaxCursorNameLength()	int	Yes
getMaxIndexLength()	int	Yes
getMaxLogicalLobSize()	default long	Yes
getMaxProcedureNameLeng th()	int	Yes
getMaxRowSize()	int	Yes
getMaxSchemaNameLengt h()	int	Yes
getMaxStatementLength()	int	Yes
getMaxStatements()	int	Yes
getMaxTableNameLength()	int	Yes
getMaxTablesInSelect()	int	Yes
getMaxUserNameLength()	int	Yes
getNumericFunctions()	String	Yes
getPrimaryKeys(String catalog, String schema, String table)	ResultSet	Yes
getPartitionTablePrimaryKey s(String catalog, String schema, String table)	ResultSet	Yes
getProcedureColumns(String catalog, String schemaPattern, String procedureNamePattern, String columnNamePattern)	ResultSet	Yes
getProcedures(String catalog, String schemaPattern, String procedureNamePattern)	ResultSet	Yes
getProcedureTerm()	String	Yes
getSchemas()	ResultSet	Yes

getSchemas(String catalog, String schemaPattern)	ResultSet	Yes
getSchemaTerm()	String	Yes
getSearchStringEscape()	String	Yes
getSQLKeywords()	String	Yes
getSQLStateType()	int	Yes
getStringFunctions()	String	Yes
getSystemFunctions()	String	Yes
getTablePrivileges(String catalog, String schemaPattern, String tableNamePattern)	ResultSet	Yes
getTimeDateFunctions()	String	Yes
getTypeInfo()	ResultSet	Yes
getUDTs(String catalog, String schemaPattern, String typeNamePattern, int[] types)	ResultSet	Yes
getURL()	String	Yes
getVersionColumns(String catalog, String schema, String table)	ResultSet	Yes
insertsAreDetected(int type)	boolean	Yes
locatorsUpdateCopy()	boolean	Yes
othersDeletesAreVisible(int type)	boolean	Yes
othersInsertsAreVisible(int type)	boolean	Yes
othersUpdatesAreVisible(int type)	boolean	Yes
ownDeletesAreVisible(int type)	boolean	Yes
ownInsertsAreVisible(int type)	boolean	Yes
ownUpdatesAreVisible(int type)	boolean	Yes

storesLowerCaseIdentifiers()	boolean	Yes
storesMixedCaseIdentifiers()	boolean	Yes
storesUpperCaseIdentifiers()	boolean	Yes
supportsBatchUpdates()	boolean	Yes
supportsCatalogsInDataMani pulation()	boolean	Yes
supportsCatalogsInIndexDefi nitions()	boolean	Yes
supportsCatalogsInPrivilege Definitions()	boolean	Yes
supportsCatalogsInProcedure Calls()	boolean	Yes
supportsCatalogsInTableDefi nitions()	boolean	Yes
getTypeInfo()	ResultSet	Yes
getUDTs(String catalog, String schemaPattern, String typeNamePattern, int[] types)	ResultSet	Yes
getURL()	String	Yes
getVersionColumns(String catalog, String schema, String table)	ResultSet	Yes
insertsAreDetected(int type)	boolean	Yes
locatorsUpdateCopy()	boolean	Yes
othersDeletesAreVisible(int type)	boolean	Yes
othersInsertsAreVisible(int type)	boolean	Yes
othersUpdatesAreVisible(int type)	boolean	Yes
ownDeletesAreVisible(int type)	boolean	Yes
ownInsertsAreVisible(int type)	boolean	Yes
ownUpdatesAreVisible(int type)	boolean	Yes
storesLowerCaseIdentifiers()	boolean	Yes

storesMixedCaseIdentifiers()	boolean	Yes
storesUpperCaseIdentifiers()	boolean	Yes
supportsBatchUpdates()	boolean	Yes
supportsCatalogsInDataMani pulation()	boolean	Yes
supportsCatalogsInIndexDefi nitions()	boolean	Yes
supportsCatalogsInPrivilege Definitions()	boolean	Yes
supportsCatalogsInProcedure Calls()	boolean	Yes
supportsCatalogsInTableDefi nitions()	boolean	Yes
supportsCorrelatedSubquerie s()	boolean	Yes
supportsDataDefinitionAndD ataManipulationTransaction s()	boolean	Yes
supportsDataManipulationTr ansactionsOnly()	boolean	Yes
supportsGetGeneratedKeys()	boolean	Yes
supportsMixedCaseIdentifier s()	boolean	Yes
supportsMultipleOpenResult s()	boolean	Yes
supportsNamedParameters()	boolean	Yes
supportsOpenCursorsAcrossC ommit()	boolean	Yes
supportsOpenCursorsAcrossR ollback()	boolean	Yes
supportsOpenStatementsAcr ossCommit()	boolean	Yes
supportsOpenStatementsAcr ossRollback()	boolean	Yes
supportsPositionedDelete()	boolean	Yes
supportsPositionedUpdate()	boolean	Yes
supportsRefCursors()	boolean	Yes
supportsResultSetConcurrenc y(int type, int	boolean	Yes

concurrency)		
supportsResultSetType(int type)	boolean	Yes
supportsSchemasInIndexDefi nitions()	boolean	Yes
supportsSchemasInPrivilege Definitions()	boolean	Yes
supportsSchemasInProcedur eCalls()	boolean	Yes
supportsSchemasInTableDefi nitions()	boolean	Yes
supportsSelectForUpdate()	boolean	Yes
supportsStatementPooling()	boolean	Yes
supportsStoredFunctionsUsin gCallSyntax()	boolean	Yes
supportsStoredProcedures()	boolean	Yes
supportsSubqueriesInCompa risons()	boolean	Yes
supportsSubqueriesInExists()	boolean	Yes
supportsSubqueriesInIns()	boolean	Yes
supportsSubqueriesInQuantif ieds()	boolean	Yes
supportsTransactionIsolation Level(int level)	boolean	Yes
supportsTransactions()	boolean	Yes
supportsUnion()	boolean	Yes
supportsUnionAll()	boolean	Yes
updatesAreDetected(int type)	boolean	Yes
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
getDriverName()	String	Yes
getDriverVersion()	String	Yes
getDriverMajorVersion()	int	Yes
getDriverMinorVersion()	int	Yes
usesLocalFiles()	boolean	Yes
usesLocalFilePerTable()	boolean	Yes
supportsMixedCaseIdentifier s()	boolean	Yes
storesUpperCaseIdentifiers()	boolean	Yes
storesLowerCaseIdentifiers()	boolean	Yes
supportsMixedCaseQuotedId entifiers()	boolean	Yes
storesUpperCaseQuotedIden tifiers()	boolean	Yes
storesLowerCaseQuotedIden tifiers()	boolean	Yes
storesMixedCaseQuotedIden tifiers()	boolean	Yes
supportsAlterTableWithAddC olumn()	boolean	Yes

supportsAlterTableWithDrop Column()	boolean	Yes
supportsColumnAliasing()	boolean	Yes
nullPlusNonNullIsNull()	boolean	Yes
supportsConvert()	boolean	Yes
supportsConvert(int fromType, int toType)	boolean	Yes
supportsTableCorrelationNa mes()	boolean	Yes
supportsDifferentTableCorrel ationNames()	boolean	Yes
supportsExpressionsInOrderB y()	boolean	Yes
supportsOrderByUnrelated()	boolean	Yes
supportsGroupBy()	boolean	Yes
supportsGroupByUnrelated()	boolean	Yes
supportsGroupByBeyondSele ct()	boolean	Yes
supportsLikeEscapeClause()	boolean	Yes
supportsMultipleResultSets()	boolean	Yes
supportsMultipleTransaction s()	boolean	Yes
supportsNonNullableColumn s()	boolean	Yes
supportsMinimumSQLGram mar()	boolean	Yes
supportsCoreSQLGrammar()	boolean	Yes
supportsExtendedSQLGram mar()	boolean	Yes
supportsANSI92EntryLevelS QL()	boolean	Yes
supportsANSI92Intermediate SQL()	boolean	Yes
supportsANSI92FullSQL()	boolean	Yes
supportsIntegrityEnhanceme ntFacility()	boolean	Yes

supportsOuterJoins()	boolean	Yes
supportsFullOuterJoins()	boolean	Yes
supportsLimitedOuterJoins()	boolean	Yes
isCatalogAtStart()	boolean	Yes
supportsSchemasInDataMan ipulation()	boolean	Yes
supportsSavepoints()	boolean	Yes
supportsResultSetHoldabilit y(int holdability)	boolean	Yes
getResultSetHoldability()	int	Yes
getDatabaseMajorVersion()	int	Yes
getDatabaseMinorVersion()	int	Yes
getJDBCMajorVersion()	int	Yes
getJDBCMinorVersion()	int	Yes

说明

uppercaseAttributeName 为 true 时，以下接口会将查询结果转为大写，可转换范围与 java 中的 toUpperCase 方法一致。

- public ResultSet getProcedures(String catalog, String schemaPattern, String procedureNamePattern)
- public ResultSet getProcedureColumns(String catalog, String schemaPattern, String procedureNamePattern, String columnNamePattern)
- public ResultSet getTables(String catalog, String schemaPattern, String tableNamePattern, String[] types)
- public ResultSet getSchemas(String catalog, String schemaPattern)
- public ResultSet getColumns(String catalog, String schemaPattern, String tableNamePattern, String columnNamePattern)
- public ResultSet getColumnPrivileges(String catalog, String schema, String table, String columnNamePattern)

- public ResultSet getTablePrivileges(String catalog, String schemaPattern, String tableNamePattern)
- public ResultSet getBestRowIdentifier(String catalog, String schema, String table, int scope, boolean nullable)
- public ResultSet getPrimaryKeys(String catalog, String schema, String table)
- protected ResultSet getImportedExportedKeys(String primaryCatalog, String primarySchema, String primaryTable, String foreignCatalog, String foreignSchema, String foreignTable)
- public ResultSet getIndexInfo(String catalog, String schema, String tableName, boolean unique, boolean approximate)
- public ResultSet getUDTs(String catalog, String schemaPattern, String typeNamePattern, int[] types)
- public ResultSet getFunctions(String catalog, String schemaPattern, String functionNamePattern)

⚠ 注意

getPartitionTablePrimaryKeys(String catalog, String schema, String table)接口用于获取分区表含全局索引的主键列，使用示例如下：

```
PgDatabaseMetaData dbmd = (PgDatabaseMetaData)conn.getMetaData();
dbmd.getPartitionTablePrimaryKeys("catalogName", "schemaName", "tableName");
```

8.1.4 java.sql.Driver

java.sql.Driver 是数据库驱动接口。

表 8-4 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
getParentLogger()	Logger	Yes

8.1.5 java.sql.PreparedStatement

java.sql.PreparedStatement 是预处理数据接口。

表 8-5 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
executeLargeUpdate()	long	No
getMetaData()	ResultSetMetaData	Yes
getParameterMetaData()	ParameterMetaData	Yes
setArray(int parameterIndex, Array x)	void	Yes
setAsciiStream(int parameterIndex, InputStream x, int length)	void	Yes
setBinaryStream(int parameterIndex, InputStream x)	void	Yes
setBinaryStream(int parameterIndex, InputStream x, int length)	void	Yes
setBinaryStream(int parameterIndex, InputStream x, long length)	void	Yes
setBlob(int parameterIndex, InputStream inputStream)	void	Yes

setBlob(int parameterIndex, InputStream inputStream, long length)	void	Yes
setBlob(int parameterIndex, Blob x)	void	Yes
setCharacterStream(int parameterIndex, Reader reader)	void	Yes
setCharacterStream(int parameterIndex, Reader reader, int length)	void	Yes
setClob(int parameterIndex, Reader reader)	void	Yes
setClob(int parameterIndex, Reader reader, long length)	void	Yes
setClob(int parameterIndex, Clob x)	void	Yes
setDate(int parameterIndex, Date x, Calendar cal)	void	Yes
setNull(int parameterIndex, int sqlType)	void	Yes
setNull(int parameterIndex, int sqlType, String typeName)	void	Yes
setObject(int parameterIndex, Object x)	void	Yes
setObject(int parameterIndex, Object x, int targetSqlType)	void	Yes
setObject(int parameterIndex, Object x, int targetSqlType, int scaleOrLength)	void	Yes
setSQLXML(int parameterIndex, SQLXML xmlObject)	void	Yes
setTime(int parameterIndex, Time x)	void	Yes
setTime(int parameterIndex, Time x, Calendar cal)	void	Yes
setTimestamp(int parameterIndex, Timestamp x)	void	Yes

setTimestamp(int parameterIndex, Timestamp x, Calendar cal)	void	Yes
setUnicodeStream(int parameterIndex, InputStream x, int length)	void	Yes
setURL(int parameterIndex, URL x)	void	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
setNString(int parameterIndex, String x)	void	Yes
addBatch()	void	Yes
executeBatch()	int[]	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。
- executeLargeUpdate()方法必须在 JDBC4.2 及以上使用。

8.1.6 java.sql.ResultSet

java.sql.ResultSet 是执行结果集接口。

表 8-6 对 java.sql.ResultSet 的支持情况

方法名	返回值类型	支持 JDBC 4
absolute(int row)	Boolean	Yes
afterLast()	void	Yes
beforeFirst()	void	Yes
cancelRowUpdates()	void	Yes
clearWarnings()	void	Yes
close()	void	Yes
deleteRow()	void	Yes
findColumn(String columnLabel)	int	Yes
first()	Boolean	Yes
getArray(int columnIndex)	Array	Yes
getArray(String columnLabel)	Array	Yes
getAsciiStream(int columnIndex)	InputStream	Yes
getAsciiStream(String columnLabel)	InputStream	Yes
getBigDecimal(int columnIndex)	BigDecimal	Yes

getBigDecimal(String columnLabel)	BigDecimal	Yes
getBinaryStream(int columnIndex)	InputStream	Yes
getBinaryStream(String columnLabel)	InputStream	Yes
getBlob(int columnIndex)	Blob	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
getCharacterStream(int columnIndex)	Reader	Yes
getCharacterStream (String columnLabel)	Reader	Yes
getClob(int columnIndex)	Clob	Yes
getClob(String columnLabel)	Clob	Yes
getConcurrency()	int	Yes
getCursorName()	String	Yes
getDate(int columnIndex)	Date	Yes
getDate(int columnIndex, Calendar cal)	Date	Yes
getDate(String columnLabel)	Date	Yes
getDate(String columnLabel,	Date	Yes

Calendar cal)		
getDouble(int columnIndex)	double	Yes
getDouble(String columnLabel)	double	Yes
getFetchDirection()	int	Yes
getFetchSize()	int	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
getMetaData()	ResultSetMetaData	Yes
getObject(int columnIndex)	Object	Yes
getObject(int columnIndex, Class<T> type)	<T> T	Yes
getObject(int columnIndex, Map<String, Class<?>> map)	Object	Yes
getObject(String columnLabel)	Object	Yes
getObject(String columnLabel, Class<T> type)	<T> T	Yes
getObject(String columnLabel, Map<String, Class<?>> map)	Object	Yes
getRow()	int	Yes
getShort(int columnIndex)	short	Yes
getShort(String columnLabel)	short	Yes

getSQLXML(int columnIndex)	SQLXML	Yes
getSQLXML(String columnLabel)	SQLXML	Yes
getStatement()	Statement	Yes
getString(int columnIndex)	String	Yes
getString(String columnLabel)	String	Yes
getNString(int columnIndex)	String	Yes
getNString(String columnLabel)	String	Yes
getTime(int columnIndex)	Time	Yes
getTime(int columnIndex, Calendar cal)	Time	Yes
getTime(String columnLabel)	Time	Yes
getTime(String columnLabel, Calendar cal)	Time	Yes
getTimestamp(int columnIndex)	Timestamp	Yes
getTimestamp(int columnIndex, Calendar cal)	Timestamp	Yes
getTimestamp(String columnLabel)	Timestamp	Yes
getTimestamp(String columnLabel, Calendar cal)	Timestamp	Yes
getType()	int	Yes
getWarnings()	SQLWarning	Yes
insertRow()	void	Yes
isAfterLast()	Boolean	Yes
isBeforeFirst()	Boolean	Yes

isClosed()	Boolean	Yes
isFirst()	Boolean	Yes
isLast()	Boolean	Yes
last()	Boolean	Yes
moveToCurrentRow()	void	Yes
moveToInsertRow()	void	Yes
next()	Boolean	Yes
previous()	Boolean	Yes
refreshRow()	void	Yes
relative(int rows)	Boolean	Yes
rowDeleted()	Boolean	Yes
rowInserted()	Boolean	Yes
rowUpdated()	Boolean	Yes
setFetchDirection(int direction)	void	Yes
setFetchSize(int rows)	void	Yes
updateArray(int columnIndex, Array x)	void	Yes
updateArray(String columnLabel, Array x)	void	Yes
updateAsciiStream(int columnIndex, InputStream x, int length)	void	Yes
updateAsciiStream(String columnLabel, InputStream x, int length)	void	Yes
updateBigDecimal(int	void	Yes

columnIndex, BigDecimal x)		
updateBigDecimal(String columnLabel, BigDecimal x)	void	Yes
updateBinaryStream(int columnIndex, InputStream x, int length)	void	Yes
updateBinaryStream (String columnLabel, InputStream x, int length)	void	Yes
updateBoolean(int columnIndex, boolean x)	void	Yes
updateBoolean(String columnLabel, boolean x)	void	Yes
updateByte(int columnIndex, byte x)	void	Yes
updateByte(String columnLabel, byte x)	void	Yes
updateBytes(int columnIndex, byte[] x)	void	Yes
updateBytes(String columnLabel, byte[] x)	void	Yes
updateCharacterStream (int columnIndex, Reader x, int length)	void	Yes
updateCharacterStream (String columnLabel, Reader reader, int length)	void	Yes
updateDate(int columnIndex, Date x)	void	Yes
updateDate(String columnLabel, Date x)	void	Yes

updateDouble(int columnIndex, double x)	void	Yes
updateDouble(String columnLabel, double x)	void	Yes
updateFloat(int columnIndex, float x)	void	Yes
updateFloat(String columnLabel, float x)	void	Yes
updateInt(int columnIndex, int x)	void	Yes
updateInt(String columnLabel, int x)	void	Yes
updateLong(int columnIndex, long x)	void	Yes
updateLong(String columnLabel, long x)	void	Yes
updateNull(int columnIndex)	void	Yes
updateNull(String columnLabel)	void	Yes
updateObject(int columnIndex, Object x)	void	Yes
updateObject(int columnIndex, Object x, int scaleOrLength)	void	Yes
updateObject(String columnLabel, Object x)	void	Yes
updateObject(String columnLabel, Object x, int scaleOrLength)	void	Yes
updateRow()	void	Yes
updateShort(int columnIndex,	void	Yes

short x)		
updateShort(String columnLabel, short x)	void	Yes
updateSQLXML(int columnIndex, SQLXML xmlObject)	void	Yes
updateSQLXML(String columnLabel, SQLXML xmlObject)	void	Yes
updateString(int columnIndex, String x)	void	Yes
updateString(String columnLabel, String x)	void	Yes
updateTime(int columnIndex, Time x)	void	Yes
updateTime(String columnLabel, Time x)	void	Yes
updateTimestamp(int columnIndex, Timestamp x)	void	Yes
updateTimestamp(String columnLabel, Timestamp x)	void	Yes
wasNull()	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。

8.1.7 java.sql.ResultSetMetData

java.sql.ResultSetMetaData 是对 ResultSet 对象相关信息的具体描述。

表 8-7 对 java.sql.ResultSetMetData 的支持情况

方法名	返回值类型	支持 JDBC 4
getCatalogName(int column)	String	Yes
getColumnClassName(int column)	String	Yes
getColumnCount()	int	Yes
getColumnDisplaySize(int column)	int	Yes
getColumnLabel(int column)	String	Yes
getColumnName(int column)	String	Yes
getColumnType(int column)	int	Yes
getColumnTypeName(int column)	String	Yes
getPrecision(int column)	int	Yes
getScale(int column)	int	Yes
getSchemaName(int column)	String	Yes
getTableName(int column)	String	Yes
isAutoIncrement(int column)	boolean	Yes
isCaseSensitive(int column)	boolean	Yes
isCurrency(int column)	boolean	Yes
isDefinitelyWritable(int column)	boolean	Yes
isNullable(int column)	int	Yes
isReadOnly(int column)	boolean	Yes
isSearchable(int column)	boolean	Yes
isSigned(int column)	boolean	Yes
isWritable(int column)	boolean	Yes

说明

uppercaseAttributeName 为 true 时，下面接口会将查询结果转为大写，可转换范围为 26 个英文字母。

- public String getColumnName(int column)
- public String getColumnLabel(int column)

8.1.8 java.sql.Statement

java.sql.Statement 是 SQL 语句接口。

表 8-8 对 java.sql.Statement 的支持情况

方法名	返回值类型	支持 JDBC 4
addBatch(String sql)	void	Yes
clearBatch()	void	Yes
clearWarnings()	void	Yes
close()	void	Yes
closeOnCompletion()	void	Yes
execute(String sql)	Boolean	Yes
execute(String sql, int autoGeneratedKeys)	Boolean	Yes
execute(String sql, int[] columnIndexes)	Boolean	Yes
execute(String sql, String[] columnNames)	Boolean	Yes
executeBatch()	Boolean	Yes
executeQuery(String sql)	ResultSet	Yes
executeUpdate(String sql)	int	Yes
executeUpdate(String sql, int autoGeneratedKeys)	int	Yes
executeUpdate(String sql, int[] columnIndexes)	int	Yes

executeUpdate(String sql, String[] columnNames)	int	Yes
getConnection()	Connection	Yes
getFetchDirection()	int	Yes
getFetchSize()	int	Yes
getGeneratedKeys()	ResultSet	Yes
getMaxFieldSize()	int	Yes
getMaxRows()	int	Yes
getMoreResults()	boolean	Yes
getMoreResults(int current)	boolean	Yes
getResultSet()	ResultSet	Yes
getResultSetConcurrenc y()	int	Yes
getResultSetHoldability( )	int	Yes
getResultSetType()	int	Yes
getQueryTimeout()	int	Yes
getUpdateCount()	int	Yes
getWarnings()	SQLWarning	Yes
isClosed()	Boolean	Yes
isCloseOnCompletion()	Boolean	Yes
isPoolable()	Boolean	Yes
setCursorName(String name)	void	Yes
setEscapeProcessing (boolean enable)	void	Yes
setFetchDirection(int direction)	void	Yes

setMaxFieldSize(int max)	void	Yes
setMaxRows(int max)	void	Yes
setPoolable(boolean poolable)	void	Yes
setQueryTimeout(int seconds)	void	Yes
setFetchSize(int rows)	void	Yes
cancel()	void	Yes
executeLargeUpdate(Str ing sql)	long	No
getLargeUpdateCount()	long	No
executeLargeBatch()	long	No
executeLargeUpdate(Str ing sql, int autoGeneratedKeys)	long	No
executeLargeUpdate(Str ing sql, int[] columnIndexes)	long	No
executeLargeUpdate(Str ing sql, String[] columnNames)	long	No

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。
- LargeUpdate 相关方法必须在 JDBC4.2 及以上使用。

8.1.9 javax.sql.ConnectionPoolDataSource

javax.sql.ConnectionPoolDataSource 是数据源连接池接口。

表 8-9 对 javax.sql.ConnectionPoolDataSource 的支持情况

方法名	返回值类型	支持 JDBC 4
getPooledConnection()	PooledConnection	Yes
getPooledConnection(String user,String password)	PooledConnection	Yes

8.1.10 javax.sql.DataSource

java.sql.DataSource 是数据源接口。

表 8-10 对 java.sql.DataSource 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

8.1.11 javax.sql.PooledConnection

javax.sql.PooledConnection 是由连接池创建的连接接口。

表 8-11 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

8.1.12 javax.naming.Context

javax.naming.Context 是连接配置的上下文接口。

表 8-12 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

8.1.13 javax.naming.spi.InitialContextFactory

javax.naming.spi.InitialContextFactory 是初始连接上下文工厂接口。

表 8-13 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

8.1.14 CopyManager

CopyManager 是 GBase 8c 的 JDBC 驱动中提供的一个 API 接口类，用于批量向 GBase 8c

数据库中导入数据。

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 8-14 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
CopyOu t	copyOut(String sql)	——	SQLException
long	copyOut(String sql, OutputStream to)	将一个 COPY TO TDOUT 的结果集从数据库发送到 OutputStream 类中。	SQLException,IOException
long	copyOut(String sql,	将一个 COPY TO TDOUT	SQLException,IOE

	Writer to)	的结果集从数据库发送到 Writer 类中。	xception

8.1.15 PGReplicationConnection

PGReplicationConnection 是 GBase 8c 的 JDBC 驱动中提供的一个 API 接口类，用于执行逻辑复制相关的功能。

PGReplicationConnection 的继承关系

PGReplicationConnection 是逻辑复制的接口，实现类是 PGReplicationConnectionImpl，该类位于 org.postgresql.replication Package 中，该类的声明如下：

```
public class PGReplicationConnection implements PGReplicationConnection
```

构造方法

```
public PGReplicationConnection(BaseConnection connection)
```

常用方法

返回值	方法	描述	throws
ChainedCreate ReplicationSlot Builder	createReplicationSlot( )	用于创建逻辑复制槽	——
void	dropReplicationSlot(S tring slotName)	用于删除逻辑复制槽	SQLException,IOE xception
ChainedStream Builder	replicationStream()	用户开启逻辑复制	——

8.2 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。

8.2.1 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 8.2.3 SQLAllocHandle。

8.2.2 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 8.2.3 SQLAllocHandle。

8.2.3 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 8-16 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： ● 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。
OutputHandlePt r	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 SQLGetDiagRec，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 8.2.23 示例。

8.2.4 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 8.2.3 SQLAllocHandle。

8.2.5 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 8-17 SQLBindCol 参数

关键字	参数说明
StatementHandl e	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 SQLGetDiagRec 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 8.2.23 示例。

8.2.6 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 8-18 SQLBindParameter 参数

关键字	参数说明
StatementHandle	语句句柄。
ParameterNumbe r	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。
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。
ParameterValuePt r	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 SQLGetDiagRec 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 8.2.23 示例。

8.2.7 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 8-19 SQLColAttribute 参数

关键字	参数说明
SQLColAttribute	语句句柄。
SQLColAttribute	要检索字段的列号，起始为 1，依次递增。
SQLColAttribute	IRD 中 ColumnNumber 行的字段。
SQLColAttribute	输出参数：一个缓冲区指针，返回 FieldIdentifier 字段值。
SQLColAttribute	● 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。
SQLColAttribute	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。
NumericAttributePt r	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 SQLGetDiagRec 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 8.2.23 示例。

8.2.8 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 8-20 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。

注意事项

当调用 SQLConnect 函数返回 SQL_ERROR 或 SQL_SUCCESS_WITH_INFO 时，通过调用 SQLGetDiagRec 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 8.2.23 示例。

8.2.9 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 8-21 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 SQLGetDiagRec 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 8.2.23 示例。

8.2.10 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 8-22 SQLExecDirect 参数

关键字	参数说明
StatementHandl e	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_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 SQLGetDiagRec 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 8.2.23 示例。

8.2.11 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 8-23 SQLExecute 参数

关键字	参数说明
StatementHandle	要执行语句的语句句柄。
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。
- 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 SQLExecute 函数返回 SQL_ERROR 或 SQL_SUCCESS_WITH_INFO 时，可通过调用 SQLGetDiagRec 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 8.2.23 示例。

8.2.12 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 8-24 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_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 SQLFetch 函数返回 SQL_ERROR 或 SQL_SUCCESS_WITH_INFO 时，通过调用 SQLGetDiagRec 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 8.2.23 示例。

8.2.13 SQLFreeStmt

功能描述

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 8.2.15 SQLFreeHandle。

8.2.14 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 8.2.15 SQLFreeHandle。

8.2.15 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 8-25 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 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 8.2.23 示例。

8.2.16 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 8.2.15 SQLFreeHandle。

8.2.17 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 8-26 SQLPrepare 参数

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。

注意事项

当 SQLPrepare 返回的值为 SQL_ERROR 或 SQL_SUCCESS_WITH_INFO 时，通过调用 SQLGetDiagRec 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 8.2.23 示例。

8.2.18 SQLGetData

功能描述

SQLGetData 返回结果集中某一列的数据。可以多次调用该接口，以便检索部分不定长度的数据。

原型

```
SQLRETURN   SQLGetData(SQLHSTMT      StatementHandle,
                       SQLUSMALLINT Col_or_Param_Num,
                       SQLSMALLINT     TargetType,
                       SQLPOINTER      TargetValuePtr,
                       SQLLEN   BufferLength,
                       SQLLEN   *StrLen_or_IndPtr);
```

参数

表 8-27 SQLGetData 参数

关键字	参数说明
StatementHandle	语句句柄，通过 SQLAllocHandle 获得。
Col_or_Param_Nu m	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 SQLGetDiagRec 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 8.2.23 示例。

8.2.19 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 8-28 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。
NativeErrorPt r	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。
TextLengthPt r	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 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 不返回诊断记录，而是通过以上返回值，来报告执行结果。

当调用 ODBC 函数返回值为 SQL_ERROR 或 SQL_SUCCESS_WITH_INFO 时，可以通过调用 SQLGetDiagRec 返回的诊断信息值 SQLSTATE，查看失败或警告原因。SQLSTATE

值如下表所示。

表 8-29 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 8.2.23 示例。

8.2.20 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 8-30 SQLSetConnectAttr 参数

关键字	参数说明

EnvironmentHan dle	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 SQLGetDiagRec 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 8.2.23 示例。

8.2.21 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 8-31 SQLSetEnvAttr 参数

关键字	参数说明
EnvironmentHan dle	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 SQLGetDiagRec 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 8.2.23 示例。

8.2.22 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 8-32 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 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 SQLGetDiagRec 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 8.2.23 示例。

8.2.23 示例

常用功能示例代码

```
// 此示例演示如何通过 ODBC 方式获取 GBase 8c 数据库中的数据。
// 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*) "gaussdb", 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;
}
```

8.3 libpq

8.3.1 数据库连接控制函数

数据库连接控制函数控制与数据库服务器链接的事情。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。

8.3.1.1 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 8-33 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，不必更换函数名就可扩展参数集。因此，建议应用程序中使用这个函数或类似变种函数 PQconnectStartParams 和 PQconnectPoll。

示例

参见 6.4.3 示例。

8.3.1.2 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 8-34 PQconnectdb 参数

关键字	参数说明
conninfo	链接字符串，字符串中的字段见 6.4.5 链接参数章节。

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

参见 6.4.3 示例。

8.3.1.3 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 8-35 PQconninfoParse 参数

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。

8.3.1.4 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 8-36 PQconnectStart 参数

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 URL。

返回值

PGconn 类型指针。

8.3.1.5 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 8-37 PQerrorMessage 参数

关键字	参数说明
conn	连接句柄

返回值

char 类型指针。

示例

参见 6.4.3 示例。

8.3.1.6 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 8-38 PQsetdbLogin 参数

关键字	参数说明
pghost	要链接的主机名，详见 6.4.5 链接参数章节描述的 host 字段。
pgport	主机服务器的端口号，详见 6.4.5 链接参数描述的 port 字段。
pgoptions	添加命令行选项以在运行时发送到服务器，详见 6.4.5 链接参数描述的 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 6.4.5 链接参数描述的 dbname 字段。
login	要链接的用户名，详见 6.4.5 链接参数章节描述的 user 字段。
pwd	如果服务器要求口令认证，所用的口令，详见 6.4.5 链接参数描述的 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

参见 6.4.3 示例。

8.3.1.7 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 8-39 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 判断服务器链接失败，应用程序调用 PQfinish，释放被 PGconn 对象使用的存储器。当调用 PQfinish 后，PGconn 指针不可重复使用。

示例

参见 6.4.3 示例。

8.3.1.8 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 8-40 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

使用此函数，关闭与服务器的连接，并使用之前使用过的参数，重建与该服务器的连接。该函数适用于在链接异常后进行故障恢复的场景。

示例

参见 6.4.3 示例。

8.3.1.9 PQstatus

功能描述

返回链接的状态。

原型

```
ConnStatusType PQstatus(const PGconn *conn);
```

参数

表 8-41 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

参见 6.4.3 示例。

8.3.2 数据库执行语句函数

在成功建立与数据库服务器的连接后，可以使用数据库执行语句函数，用以执行 SQL 查询和命令。

8.3.2.1 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。

原型

```
void PQclear(PGresult *res);
```

参数

表 8-42 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

参见 6.4.3 示例。

8.3.2.2 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 8-43 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

参见 6.4.3 示例。

8.3.2.3 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 8-44 PQexecParams 参数

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）。
resultFormat	返回结果格式（文本或二进制）。

返回值

PGresult 类型指针。

8.3.2.4 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 8-45 PQexecParamsBatch 参数

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。
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。

8.3.2.5 PQexecPrepared

功能描述

发送一个请求，以用给定参数执行预备语句并等待结果。

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 **8-46** PQexecPrepared 参数

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。

8.3.2.6 PQexecPreparedBatch

功能描述

发送一个请求，以用给定的批量参数执行预备语句并等待结果。

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 8-47 PQexecPrepareBatch 参数

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。

8.3.2.7 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 8-48 PQfname 参数

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 6.4.3 示例。

8.3.2.8 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 8-49 PQgetvalue 参数

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 6.4.3 示例。

8.3.2.9 PQnfields

功能描述

返回查询结果中每一行的列数。

原型

```
int PQnfields(const PGresult *res);
```

参数

表 8-50 PQnfields 参数

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 6.4.3 示例。

8.3.2.10 PQntuples

功能描述

返回查询结果中的行（元组）数。由于返回的是整数结果，在 32 位操作系统上大数据量结果集，可能使返回值溢出。

原型

```
int PQntuples(const PGresult *res);
```

参数

表 8-51 PQntuples 参数

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 6.4.3 示例。

8.3.2.11 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 8-52 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

参见 6.4.3 示例。

8.3.2.12 PQresultStatus

功能描述

返回命令的结果状态。

原型

```
ExecStatusType PQresultStatus(const PGresult *res);
```

参数

表 8-53 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

- 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

参见 6.4.3 示例。

8.3.3 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。

8.3.3.1 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 8-54 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

参见 6.4.3 示例。

8.3.3.2 PQsendQueryParams

功能描述

向服务器提交与命令分隔的参数，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 8-55 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

参见 6.4.3 示例。

8.3.3.3 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 8-

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

参见 6.4.3 示例。

8.3.3.4 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 8-57 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

参见 6.4.3 示例。

8.3.3.5 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 8-58 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

参见 6.4.3 示例。

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。

8.3.3.6 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 8-59 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

参见 6.4.3 示例。

8.3.3.7 PQfreeCancel

功能描述

释放 PQgetCancel 创建的数据结构。

原型

```
void PQfreeCancel(PGcancel *cancel);
```

参数

表 8-60 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

参见 6.4.3 示例。

8.3.3.8 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 8-61 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

参见 6.4.3 示例。

8.4 Psycopg

Psycopg 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。

8.4.1 psycopg2.connect()

功能描述

此方法创建新的数据库会话并返回新的 connection 对象。

原型

```
conn=psycopg2.connect(dbname="test",user="postgres",password="secret",host="127.0.0.1",port="5432")
```

参数

表 8-62 psycopg2.connect 参数

关键字	参数说明
dbname	数据库名称
user	用户名
password	密码
host	数据库 IP 地址，默认为 UNIX socket 类型。
port	连接端口号，默认为 5432。
sslmode	ssl 模式，ssl 连接时用。
sslcert	客户端证书路径，ssl 连接时用。
sslkey	客户端密钥路径，ssl 连接时用。
sslrootcert	根证书路径，ssl 连接时用。

返回值

connection 对象（连接 GBase 8c 数据库实例的对象）。

示例

参见 6.5.9 示例：常用操作。

8.4.2 connection.cursor()

功能描述

此方法用于返回新的 cursor 对象。

原型

```
cursor(name=None,  cursor_factory=None,  scrollable=None, withhold=False)
```

参数

表 8-63 connection.cursor 参数

关键字	参数说明
name	cursor 名称，默认为 None。
cursor_factory	用于创造非标准 cursor，默认为 None。
scrollable	设置 SCROLL 选项，默认为 None。
withhold	设置 HOLD 选项，默认为 False。

返回值

cursor 对象（用于整个数据库使用 Python 编程的 cursor）。

示例

参见 6.5.9 示例：常用操作。

8.4.3 cursor.execute(query,vars_list)

功能描述

此方法执行被参数化的 SQL 语句（即占位符，而不是 SQL 文字）。psycopg2 模块支持用%s 标志的占位符。

原型

```
curosr.execute(query,vars_list)
```

参数

表 8-64 curosr.execute 参数

关键字	参数说明
query	待执行的 SQL 语句。
vars_list	变量列表，匹配 query 中%s 占位符。

返回值

无

示例

参见 6.5.9 示例：常用操作。

8.4.4 curosr.executemany(query,vars_list)

功能描述

此方法执行 SQL 命令所有参数序列或序列中的 SQL 映射。

原型

```
curosr.executemany(query,vars_list)
```

参数

表 8-65 curosr.executemany 参数

关键字	参数说明
query	待执行的 SQL 语句。
vars_list	变量列表，匹配 query 中%s 占位符。

返回值

无

示例

参见 6.5.9 示例：常用操作。

8.4.5 connection.commit()

功能描述

此方法将当前挂起的事务提交到数据库。

⚠ 注意

默认情况下，Psycopg 在执行第一个命令之前打开一个事务：如果不调用 commit()，任何数据操作的效果都将丢失。

原型

```
connection.commit()
```

示例

参见 6.5.9 示例：常用操作。

8.4.6 connection.rollback()

功能描述

此方法将当前挂起事务回滚。

注意

执行关闭连接“close()”而不先提交更改“commit()”将导致执行隐式回滚。

原型

```
connection.rollback()
```

示例

参见 6.5.9 示例：常用操作。

8.4.7 cursor.fetchone()

功能描述

此方法提取查询结果集的下一行，并返回一个元组。

原型

```
cursor.fetchone()
```

返回值

单个元组，为结果集的第一条结果，当没有更多数据可用时，返回为“None”。

示例

参见 6.5.9 示例：常用操作。

8.4.8 cursor.fetchall()

功能描述

此方法获取查询结果的所有（剩余）行，并将它们作为元组列表返回。

原型

```
cursor.fetchall()
```

返回值

元组列表，为结果集的所有结果。空行时则返回空列表。

示例

参见 6.5.9 示例：常用操作。

8.4.9 cursor.close()

功能描述

此方法关闭当前连接的游标。

原型

```
cursor.close()
```

示例

参见 6.5.9 示例：常用操作。

8.4.10 connection.close()

功能描述

此方法关闭数据库连接。

注意

此方法关闭数据库连接，并不自动调用 commit()。如果只是关闭数据库连接而不调用 commit()方法，那么所有更改将会丢失。

原型

```
connection.close()
```

示例

参见 6.5.9 示例：常用操作。

9 导入数据

GBase 8c 数据库提供了灵活的数据导入方式：INSERT、COPY 以及 gsql 元命令\copy。各方式具有不同的特点，具体请参见表 9-1。

表 9-1 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 GBase 8c 写入数据。 通过 JDBC 驱动的 CopyManager 接口，从其他数据库向 GBase 8c 数据库写入数据。这种方式具有业务数据无需落地成文件的优势。
gsql 工具的元命令 \copy	与直接使用 COPY 语句不同，该命令读取/写入的文件只能是 gsql 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 方式。

9.1 通过 INSERT 语句直接写入数据

用户可以通过以下方式执行 INSERT 语句(参见《GBase 8c V5_3.0.1_SQL 手册》INSERT 章节)，直接向 GBase 8c 数据库写入数据：

- 使用客户端工具向 GBase 8c 数据库写入数据。参见 3.7.2 向表中插入数据。
- 通过 JDBC/ODBC 驱动，连接数据库执行 INSERT 语句，向 GBase 8c 数据库写入数据。参见 3.2 连接数据库。

GBase 8c 数据库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。

9.2 使用 COPY FROM STDIN 导入数据

9.2.1 关于 COPY FROM STDIN 导入数据

用户可以通过一下方式执行 COPY FROM STDIN 语句，直接向 GBase 8c 数据库中写入数据。

- 通过键盘输入，向 GBase 8c 数据库写入数据。参见《GBase 8c V5_3.0.1_SQL 手册》COPY。
- 通过 JDBC 驱动的 CopyManager 接口，从文件或者数据库向 GBase 8c 写入数据。此方法支持 COPY 语法中 copy option 的所有参数。

9.2.2 CopyManager 类简介

CopyManager 是 GBase 8c 的 JDBC 驱动中一个 API 接口类，用于批量导入数据。

9.2.3 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 9-2 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 中快速向数据库中的表导入数据。	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导入数据。	SQLException,IOException
long	copyIn(String sql, Reader from, int bufferSize)	使用 COPY FROM STDIN 从 Reader 中	SQLException,IOException

		快速向数据库中的表导入数据。	
CopyOu t	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

9.2.4 处理错误表

操作场景

当数据导入发生错误时，请根据本文指引信息进行处理。

查询错误信息

数据导入过程中发生的错误，一般分为数据格式错误和非数据格式错误。

- 数据格式错误

在创建外表时，通过设置参数 LOG INTO error_table_name，将数据导入过程中出现的数据格式错误信息，写入指定的错误信息表 error_table_name 中。可以通过以下 SQL 命令，查询详细错误信息。

```
postgres=# SELECT * FROM    error_table_name;
```

错误信息表结构如表 9-3 所示。

表 9-3 错误信息表

列名称	类型	描述
nodeid	integer	报错节点编号。
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	numeric	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。

- 非数据格式错误

一旦发生非数据格式错误，将导致整个数据导入失败。用户可以根据执行数据导入过程中界面提示的错误信息，用以定位问题并处理错误表。

处理数据导入错误

根据获取的错误信息，对照表 9-4 以处理数据导入错误。

表 9-4 处理数据导入错误

错误信息	原因	解决办法
missing data for column "r_reason_desc"	数据源文件中的列数比外表定义的列数少。 对于 TEXT 格式的数据源文件，由于转义字符（\）导致 delimiter（分隔符）错位或者 quote（引号字符）错位造成的错误。 示例：目标表存在 3 列字段，导入的数据如下所示。由于存在转义字符“\”，分隔符“|”被转义为第二个字段的字段值，导致第三个字段值缺失。 BE|Belgium\\|1	1. 由于列数少导致的报错，选择下列办法解决： ● 在数据源文件中，增加列 r_reason_desc 的字段值。 ● 在创建外表时，将参数 fill_missing_fields 设置为 on。即当导入过程中，若数据源文件中一行数据的最后一个字段缺失，则把最后一个字段的值设置为 NULL，不报错。 2. 对由于转义字符导致的错误，需检查报错的行中是否含有转义字符（\）。若存在，建议在创建外表时，将参数“noescaping”（是否不对'\'和后面的字符进行转义）设置为 true。
extra data after last expected column	数据源文件中的列数比外表定义的列数多。	● 在数据源文件中，删除多余的字段值。 ● 在创建外表时，将参数 ignore_extra_data 设置为 on。即在导入过程中，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。
invalid input syntax for type numeric: "a"	数据类型错误。	在数据源文件中，修改输入字段的数据类型。根据此错误信息，需将输入的数据类型修改为 numeric。
null value in column "staff_id" violates not-null constraint	非空约束。	在数据源文件中，增加非空字段信息。根据此错误信息，需增加 staff_id 列的值。
duplicate key value violates unique constraint "reg_id_pk"	唯一约束。	删除数据源文件中重复的行。通过设置关键字 DISTINCT，从 SELECT 结果集中删除重复的行，保证导入的每一行都是唯一 的。 postgres=# INSERT INTO reasons SELECT DISTINCT * FROM foreign_tpcds_reasons;
value too long for type character varying(16)	字段值长度超过限制。	在数据源文件中，修改字段值长度。根据此错误信息，需将字段值长度限制为 VARCHAR2(16)。

9.2.5 示例 1：通过本地文件导入导出数据

在基于，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 GBase 8c 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();
            }
         }
      }
   }
}
```

9.2.6 示例 2：从 MY 迁移数据

下面示例演示如何通过 CopyManager 从 MY 向 GBase 8c 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");      //GBase 8c 数据库用户名
    String pass =    new String("passwd"); //GBase 8c 数据库密码 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.MY.jdbc.Driver").newInstance();
                Connection conn =
                DriverManager.getConnection("jdbc:MY://10.119.179.227:3306/jack?
                useSSL=false&allowPublicKeyRetrieval=true", "jack", "xxxxxxxxx");
                Statement stmt = conn.createStatement();
                rs = stmt.executeQuery("select * from migration_table");
            } catch (SQLException e) {
                    e.printStackTrace();
            } catch (Exception e) {
                    e.printStackTrace();
             }
            return rs;
        }
}
```

9.3 使用 gsql 元命令导入数据

gsql 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 9-5 \copy 元命令说明。

表 9-5 \copy 元命令说明

语法	说明
\copy { table [(column_list)] | (query) } { from | to } { filename | stdin | stdout | pstdin | pstdout }[with] [binary]	在 gsql 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/导出操作。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

[delimiter [as] 'character'] [null [as] 'string'] [csv [header] [quote [as] 'character'] [escape [as] 'character'] [force quote column_list | *] [force not null column_list]]	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场景，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 gsql 的标准输入。

- pstout

声明输出打印到 gsql 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。

```
postgres=# CREATE TABLE a(a int);
```

(2) 导入数据。

① 从 stdin 拷贝数据到目标表 a。

```
postgres=# \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。

```
postgres=# SELECT * FROM a;
 a
---
1
2
(2 rows)
```

② 从本地文件拷贝数据到目标表 a。假设存在本地文件/home/gbase/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。

```
postgres=# \copy a FROM '/home/gbase/2.csv' WITH (delimiter',',IGNORE_EXTRA_DATA
'on');
```

9.4 用 gs_restore 命令导入数据

操作场景

gs_restore 是 GBase 8c 数据库提供的与 gs_dump 配套的导入工具。通过该工具，可将 gs_dump 导出的文件导入至数据库。gs_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。

gs_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。导入时生成列会自动更新，并像普通列一样保存。

- 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 gs_dump 导出的纯文本格式文件。

gs_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

gs_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 gbase 登录数据库主节点。

步骤 2 使用 gs_restore 命令，从 postgres 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。

```
gs_restore -U jack /home/gbase/backup/MPPDB_backup.tar -p 5432 -d backupdb -s -e -c
Password:
```

表 9-6 常用参数说明

参数	参数说明	举例
-U	连接数据库的用户名。	-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 abcd@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	指定服务器所侦听的 TCP 端口或本地 Unix 域套接字后缀，以确保连接。	-p 5432
-e	当发送 SQL 语句到数据库时如果出现错误，则退出。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《GBase 8c V5_3.0.x_工具参考》中“服务端工具 > gs_restore”章节。

----结束

示例

示例一：执行 gs_restore，导入指定 MPPDB_backup.dmp 文件（自定义归档格式）中 postgres 数据库的数据和对象定义。

```
gs_restore backup/MPPDB_backup.dmp -p 5432 -d backupdb
Password:
gs_restore[2017-07-21 19:16:26]: restore operation successful
gs_restore: total time: 13053 ms
```

示例二：执行 gs_restore，导入指定 MPPDB_backup.tar 文件（tar 归档格式）中 postgres 数据库的数据和对象定义。

```
gs_restore backup/MPPDB_backup.tar -p 5432 -d backupdb
gs_restore[2017-07-21 19:21:32]: restore operation successful gs_restore[2017-07-21 19:21:32]:
total time: 21203 ms
```

示例三：执行 gs_restore，导入指定 MPPDB_backup 目录文件（目录归档格式）中 postgres 数据库的数据和对象定义。

```
gs_restore backup/MPPDB_backup -p 5432 -d backupdb
gs_restore[2017-07-21 19:26:46]: restore operation successful
gs_restore[2017-07-21 19:26:46]: total time: 21003 ms
```

示例四：执行 gs_restore，将 postgres 数据库的所有对象的定义导入至 backupdb 数据库。导入前，数据库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。

```
gs_restore /home/gbase/backup/MPPDB_backup.tar -p 5432 -d backupdb -s -e -c
Password:
gs_restore[2017-07-21 19:46:27]: restore operation successful
gs_restore[2017-07-21 19:46:27]: total time: 32993 ms
```

示例五：执行 gs_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。

```
gs_restore backup/MPPDB_backup.dmp -p 5432 -d backupdb -e -c -n PUBLIC
gs_restore: [archiver (db)] Error while PROCESSING TOC:
gs_restore: [archiver (db)] Error from TOC entry 313; 1259 337399 TABLE table1 gaussdba
gs_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。

```
gs_restore backup/MPPDB_backup.dmp -p 5432 -d backupdb -e -c -n PUBLIC
gs_restore[2017-07-21 19:52:26]: restore operation successful
gs_restore[2017-07-21 19:52:26]: total time: 2203 ms
```

示例六：执行 gs_restore，导入 MPPDB_backup.dmp 文件中 hr 模式下表 hr.staffs 的定义。在导入之前，hr.staffs 表不存在，需要确保存在 hr 的 schema。

```
gs_restore backup/MPPDB_backup.dmp -p 5432 -d backupdb -e -c -s -n hr -t staffs
gs_restore[2017-07-21 19:56:29]: restore operation successful
gs_restore[2017-07-21 19:56:29]: total time: 21000 ms
```

示例七：执行 gs_restore，导入 MPPDB_backup.dmp 文件中 hr 模式下表 hr.staffs 的数据。在导入之前，hr.staffs 表不存在数据，需要确保存在 hr 的 schema。

```
gs_restore backup/MPPDB_backup.dmp -p 5432 -d backupdb -e -a -n hr -t staffs
gs_restore[2017-07-21 20:12:32]: restore operation successful
gs_restore[2017-07-21 20:12:32]: total time: 20203 ms
```

示例八：执行 gs_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
gsql -d human_resource -p 5432
gsql ((GBase 8c V5 build a4c7bcc5) compiled at 2024-01-16 14:49:56 commit 0 last mr 1632 )
Non-SSL connection (SSL connection is recommended when requiring high-security)
Last login: 2024-01-19 16:22:27+08 from [local] (user gbase)
Type "help" for help.

human_resource=# drop table hr.staffs CASCADE; NOTICE: drop cascades to view
hr.staff_details_view DROP TABLE
gs_restore /home/gbase/backup/MPPDB_backup.tar -p 5432 -d human_resource -n hr -t staffs
-s -e Password:
restore operation successful
```

```
total time: 904 ms
```

示例九：执行 gs_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 | gbase | {orientation=row,compression=no}
hr | employments   | table | gbase | {orientation=row,compression=no}
hr | places | table | gbase | {orientation=row,compression=no}
hr | sections | table | gbase | {orientation=row,compression=no}
hr | states | table | gbase | {orientation=row,compression=no}
(5 rows)
gs_restore /home/gbase/backup/MPPDB_backup.tar -p 5432 -d human_resource -n hr -t staffs
-n hr -t areas
Password:
restore operation successful total time: 724 ms
human_resource=# \d
List of relations
Schema |       Name      | Type    | Owner    | Storage
--------+-------------------+-------+----------+--------------------------------
hr    | areas | table | gbase | {orientation=row,compression=no}
hr    | employment_history | table | gbase | {orientation=row,compression=no}
hr    | employments       | table | gbase | {orientation=row,compression=no}
hr | places | table | gbase | {orientation=row,compression=no}
hr | sections | table | gbase | {orientation=row,compression=no}
hr | staffs | table | gbase | {orientation=row,compression=no}
hr | states | table | gbase | {orientation=row,compression=no}
(7 rows)
human_resource=# select * from hr.areas; area_id |    area_name
---------+-----------------------
4 | Middle East and Africa 1 | Europe
| Americas
| Asia
```

(4 rows)

示例十：执行 gs_restore，导入 hr 的模式，包含模式下的所有对象定义和数据。

```
gs_restore /home/gbase/backup/MPPDB_backup1.dmp -p 5432 -d backupdb -n hr -e
Password:
restore operation successful total time: 702 ms
```

示例十一：执行 gs_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。

```
gs_restore /home/gbase/backup/MPPDB_backup2.dmp -p 5432 -d backupdb -n hr -n hr1 -s
Password:
restore operation successful total time: 665 ms
```

示例十二：执行 gs_restore，将 human_resource 数据库导出文件导入至 backupdb 数据库中。

```
postgres=# create database backupdb;
CREATE DATABASE

gs_restore /home/gbase/backup/MPPDB_backup.tar -p 5432 -d backupdb
restore operation successful total time: 23472 ms
gsql -d backupdb -p 5432 -r
gsql ((GBase 8c V5 build a4c7bcc5) compiled at 2024-01-16 14:49:56 commit 0 last mr 1632 )
Non-SSL connection (SSL connection is recommended when requiring high-security)
Last login: 2024-01-19 16:22:27+08 from [local] (user gbase)
Type "help" for help.

postgres=#

postgres=# select * from hr.areas;
area_id |area_name
---------+-------------------------
4 | Middle East and Africa 1 | Europe
| Americas
| Asia
(4 rows)
```

示例十三：用户 user1 不具备将导出文件中数据导入至数据库 backupdb 的权限，而角色 role1 具备该权限，要实现将文件数据导入数据库 backupdb，可以在导出命令中设置--role 角色为 role1，使用 role1 的权限，完成导出目的。

```
human_resource=# CREATE USER user1 IDENTIFIED BY "1234@abc"; CREATE ROLE
role1 with SYSADMIN IDENTIFIED BY "abc@1234";
gs_restore -U user1 /home/gbase/backup/MPPDB_backup.tar -p 5432 -d backupdb --role role1
-- rolepassword abc@1234
Password:
restore operation successful total time: 554 ms
gsql ((GBase 8c V5 build a4c7bcc5) compiled at 2024-01-16 14:49:56 commit 0 last mr 1632 )
Non-SSL connection (SSL connection is recommended when requiring high-security)
```

```
Last login: 2024-01-19 16:22:27+08 from [local] (user gbase)
Type "help" for help.

postgres=#
postgres=# select * from hr.areas;
 area_id |      area_name
---------+------------------------
4 | Middle East and Africa
1 | Europe
| Americas
| Asia
(4 rows)
```

9.5 更新表中数据

9.5.1 使用 DML 命令更新表

GBase 8c 支持标准 DML 命令（数据库操作语言），对表进行更新。

操作步骤

假设存在表 customer_t，表结构如下：

```
postgres=# 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。

```
postgres=# 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。

```
postgres=# 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，参见 3.7.2 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。

```
postgres=# UPDATE customer_t SET c_customer_id = 0;
```

更多关于 UPDATE 的使用方法，请参见《GBase 8c V5_3.0.1_SQL 手册》 UPDATE。

- 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。

```
postgres=# DELETE FROM customer_t WHERE c_last_name = 'Baker';
```

更多关于 DELETE 的使用方法，请参见《GBase 8c V5_3.0.1_SQL 手册》 DELETE。

- 使用 TRUNCATE 命令快速从表中删除所有的行。

```
postgres=# TRUNCATE TABLE customer_t;
```

更多关于 TRUNCATE 的使用方法，请参见《GBase 8c V5_3.0.1_SQL 手册》TRUNCATE。

删除表时，使用 DELETE 语句每次删除一行数据，仅删除数据而不释放存储空间；使用 TRUNCATE 语句则是通过释放表存储的数据页来删除数据，删除数据且释放存储空间。使用 TRUNCATE 语句进行表删除速度更快。

9.5.2 使用合并方式更新和插入数据

在需要将数据批量添加至现有表的场景下，GBase 8c 数据库提供了 MERGE INTO 语句，通过表合并的方式，高效地将新数据添加到现有表。

MERGE INTO 语句可将目标表和源表中的数据，通过关联条件进行匹配。若关联条件匹配，则对目标表进行更新操作（UPDATE）；若关联条件不匹配，则对目标表执行插入操作（INSERT）。这种方法能够将两个表合并执行 UPDATE 和 INSERT 操作，避免多次执行。

前提条件

用户如需进行 MERGE INTO 操作，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。

```
postgres=# CREATE TABLE products ( product_id INTEGER,product_name VARCHAR2(60),
category VARCHAR2(60));
postgres=# 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。

```
postgres=# CREATE TABLE newproducts ( product_id INTEGER, product_name
VARCHAR2(60), category VARCHAR2(60));
postgres=# 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。

```
postgres=# 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 9-7。更多信息，请参见《GBase 8c V5_3.0.1_SQL 手册》 MERGE INTO。

表 9-7 MERGE INTO 语句参数说明

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。	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。	取值：products p 说明：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。	取值：WHEN MATCHED THEN UPDATE SET 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 newproducts。

步骤 4 查询合并后的目标表 newproducts。

```
postgres=# 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)
```

----结束

9.6 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。

GBase 8c 数据库提供了三种深层复制的方式。

9.6.1 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，以完成原始表的复制。

在创建新表时，可以指定表以及列属性，比如主键。

操作步骤

执行以下步骤对表 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。

```
postgres=# 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。

```
postgres=# INSERT INTO customer_t_copy (SELECT * FROM customer_t);
```

步骤 3 删除原始表。

```
postgres=# 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。

```
postgres=# ALTER TABLE customer_t_copy RENAME TO customer_t;
```

----结束

9.6.2 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，可以使用 ALTER

TABLE 语句来添加主键属性。

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。

```
postgres=# 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。

```
postgres=# INSERT INTO customer_t_copy (SELECT * FROM customer_t);
```

步骤 3 删除原始表。

```
postgres=# 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。

```
postgres=# ALTER TABLE customer_t_copy RENAME TO customer_t;
```

----结束

9.6.3 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 TEMP 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原始表，以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 TEMP TABLE AS 语句创建表 customer_t 的临时表副本 customer_t_temp。

```
postgres=# CREATE TEMP 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。

```
postgres=# 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。

```
postgres=# INSERT INTO customer_t (SELECT * FROM customer_t_temp);
```

步骤 4 删除临时表副本 customer_t_temp。

```
postgres=# DROP TABLE customer_t_temp;
```

----结束

9.7 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《GBase 8c V5_3.0.1_SQL 手册》 ANALYZE | ANALYSE。

步骤 1 更新表统计信息。

以表 product_info 为例，ANALYZE 命令如下：

```
postgres=# ANALYZE product_info;
ANALYZE
```

----结束

表自动分析

GBase 8c 提供 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 50 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 50+10%*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。

9.8 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：

```
postgres=# VACUUM FULL product_info
VACUUM
```

----结束

9.9 管理并发写入操作

事务隔离说明

GBase 8c 数据库是基于 MVCC（多版本并发控制），并结合两阶段锁的方式进行事务管理的。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。

9.9.1 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。

9.9.2 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。

9.9.3 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));
```

9.9.3.1 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

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，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

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，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。

9.9.3.2 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

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

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。

9.9.3.3 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。

9.9.3.4 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

开启事务 T1，不提交的同时开启事务 T2，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

开启事务 T1，不提交的同时开启事务 T2，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。

10 导出数据

GBase 8c 提供 gs_dump、gs_dumpall 等工具用于数据导出，此外提供 wal2json 插件支持逻辑日志输出为 json 格式。

wal2json 插件支持在每个事务中生成一个 JSON 对象。JSON 对象中提供了所有新/旧元组，还包括事务时间戳、限定架构、数据类型、事务 ID 等属性。

10.1 使用 gs_dump 和 gs_dumpall 命令导出数据

10.1.1 概述

GBase 8c 数据库提供 gs_dump 和 gs_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。结合使用导入工具，将导出的数据信息再导入至新数据库，可以完成数据迁移。gs_dump 支持导出单个数据库或其内的对象，而 gs_dumpall 支持导出所有数据库或各库的公共全局对象。详细的使用场景见表 15-10-1。

表 10-1 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	**数据库级导出。** ● 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 9.3 使用 gsql 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，参见 9.4 使用 gs_restore 命令导入数据。
	模式级导出。		

	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。		
	表级导出。 ● 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	**数据库级导出。** ● 导出全量信息。使用导出的全量信息可以创建与当前主机相同的一个主机环境，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，可以快速创建与当前主机相同的一个主机环境，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 9.3 使用 gsql 元命令导入数据。
	各库公共全局对象导出。 ● 仅导出表空间信息。 ● 仅导出角色信息。 ● 导出角色与表空间。		

gs_dump 和 gs_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，gs_dump 和 gs_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 15-10-1，详细操作请参见 10.1.4 无权限角色导出数据。

gs_dump 和 gs_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。注意，使用 gs_dump 加密的纯文本格式文件，如果导出的数据库中包含存储过程，不支持使用 gsql 命令恢复文件，请使用另外三种模式导出数据库，并使用 gs_restore 恢复。

gs_dump 和 gs_dumpall 工具在进行数据导出时，其他用户可以访问数据库（读或写）。

gs_dump 和 gs_dumpall 工具支持导出完整一致的数据。例如，T1 时刻启动 gs_dump 导出 A 数据库，或者启动 gs_dumpall 导出 GBase 8c 数据库，那么导出数据结果将会是 T1 时刻 A 数据库或者该 GBase 8c 数据库的数据状态，T1 时刻之后对 A 数据库或 GBase 8c 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，建议通过 gs_guc 工具设置数据库节点的如下参数（如果参数值大于如下建议值，则无需设置）。例如：

```
gs_guc set -Z coordinator -N all -I all -c 'max_prepared_transactions = 1000'
gs_guc set -Z coordinator -N all -I all -c '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，gs_dump 和 gs_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 gs_dumpall 读取所有数据库中的表，因此必须以 GBase 8c 管理员身份进行连接，才能导出完整文件。在使用 gsql 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。

10.1.2 导出单个数据库

10.1.2.1 导出数据库

GBase 8c 支持使用 gs_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 gbase 登录数据库主节点。

步骤 2 使用 gs_dump 导出 userdatabase 数据库。

```
gs_dump -U jack -f /home/gbase/backup/userdatabase_backup.tar -p 5432 postgres -Ft
Password:
```

表 10-2 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 abcd@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/gbase/backup/postgres_backup.tar
-p	指定服务器所侦听的 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	postgres
-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	-Ft

	● t：tar 归档格式	

其他参数说明请参见《GBase 8c V5_3.0.x_工具参考》中“服务端工具 > gs_dump”章节。

----结束

示例

示例一：执行 gs_dump，导出 postgres 数据库全量信息，导出文件格式为 sql 文本格式。

```
gs_dump -f /home/gbase /backup/postgres_backup.sql -p 5432 postgres -F p
Password:
gs_dump[port='5432'][postgres][2022-06-07 13:55:04]: The total objects number is 431.
gs_dump[port='5432'][postgres][2022-06-07 13:55:04]: [100.00%] 431 objects have been
dumped.
gs_dump[port='5432'][postgres][2022-06-07 13:55:04]: dump database postgres successfully
gs_dump[port='5432'][postgres][2022-06-07 13:55:04]: total time: 1769  ms
```

示例二：执行 gs_dump，仅导出 postgres 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。

```
gs_dump -f /home/gbase/backup/postgres_data_backup.dmp -p 5432 postgres -a -F c
Password:
gs_dump[port='5432'][postgres][2022-06-07 13:57:01]: dump database postgres successfully
gs_dump[port='5432'][postgres][2022-06-07 13:57:01]: total time: 1758  ms
```

示例三：执行 gs_dump，仅导出 postgres 数据库所有对象的定义，导出文件格式为 sql 文本格式。

```
gs_dump -f /home/gbase/backup/postgres_def_backup.sql -p 5432 postgres -s -F p
Password:
gs_dump[port='5432'][postgres][2022-06-07 13:57:38]: The total objects number is 423.
gs_dump[port='5432'][postgres][2022-06-07 13:57:38]: [100.00%] 423 objects have been
dumped.
gs_dump[port='5432'][postgres][2022-06-07 13:57:38]: dump database postgres successfully
gs_dump[port='5432'][postgres][2022-06-07 13:57:38]: total time: 1674  ms
```

示例四：执行 gs_dump，仅导出 postgres 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。

```
gs_dump -U gbase  -f /home/gbase/backup/MPPDB_schema_backup -p 5432 postgres-n hr -n
public -Fd
Password:
```

10.1.2.2 导出模式

GBase 8c 目前支持使用 gs_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 gbase 登录数据库主节点。

步骤 2 使用 gs_dump 同时导出 hr 和 public 模式。

```
gs_dump -U gbase -f /home/gbase/backup/MPPDB_schema_backup -p 5432 postgres -n hr -n
public - F d
Password:
```

表 10-3 常用参数说明

参数	参数说明	举例
-U	连接数据库的用户名。 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 abcd@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/gbase/backup/ postgres_backup.tar
-p	指定服务器所侦听的 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	postgres
-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	-Ft

其他参数说明请参见《GBase 8c V5_3.0.x_工具参考》中“服务端工具 > gs_dump”章节。

----结束

示例

示例一：执行 gs_dump，导出 hr 模式全量信息，导出文件格式为文本格式。

```
gs_dump -f /home/gbase/backup/MPPDB_schema_backup.sql -p 5432 human_resource -n hr -F p
Password:
gs_dump[port='5432'][human_resource][2017-07-21 16:05:55]: dump database human_resource successfully
gs_dump[port='5432'][human_resource][2017-07-21 16:05:55]: total time: 2425 ms
```

示例二：执行 gs_dump，仅导出 hr 模式的数据，导出文件格式为 tar 归档格式。

```
gs_dump -f /home/gbase/backup/MPPDB_schema_data_backup.tar -p 5432 human_resource -n hr -a -F t
Password:
gs_dump[port='5432'][human_resource][2018-11-14 15:07:16]: dump database human_resource successfully
gs_dump[port='5432'][human_resource][2018-11-14 15:07:16]: total time: 1865 ms
```

示例三：执行 gs_dump，仅导出 hr 模式的定义，导出文件格式为目录归档格式。

```
gs_dump -f /home/gbase/backup/MPPDB_schema_def_backup -p 5432 human_resource -n hr -s -F d
Password:
```

```
gs_dump[port='5432'][human_resource][2018-11-14 15:11:34]: dump database
human_resource successfully
gs_dump[port='5432'][human_resource][2018-11-14 15:11:34]: total time: 1652 ms
```

示例四：执行 gs_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。

```
gs_dump -f /home/gbase/backup/MPPDB_schema_backup.dmp -p 5432 human_resource -N hr
-F c
Password:
gs_dump[port='5432'][human_resource][2017-07-21 16:06:31]: dump database
human_resource successfully
gs_dump[port='5432'][human_resource][2017-07-21 16:06:31]: total time: 2522 ms
```

示例五：执行 gs_dump，同时导出 hr 和 public 模式，且仅导出模式定义，导出文件格式为 tar 归档格式。

```
gs_dump -f /home/gbase/backup/MPPDB_schema_backup1.tar -p 5432 human_resource -n hr
-n public -s -F
t Password:
gs_dump[port='5432'][human_resource][2017-07-21 16:07:16]: dump database
human_resource successfully
gs_dump[port='5432'][human_resource][2017-07-21 16:07:16]: total time: 2132 ms
```

示例六：执行 gs_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。

```
gs_dump -f /home/gbase/backup/MPPDB_schema_backup2.dmp -p 5432 human_resource -N
hr -N public -Fc
Password:
gs_dump[port='5432'][human_resource][2017-07-21 16:07:55]: dump database
human_resource successfully
gs_dump[port='5432'][human_resource][2017-07-21 16:07:55]: total time: 2296 ms
```

示例七：执行 gs_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。

```
gs_dump -f /home/gbase/backup/MPPDB_backup3.dmp -p 5432 human_resource -t public.* -t
hr.staffs -F c
Password:
gs_dump[port='5432'][human_resource][2018-12-13 09:40:24]: dump database
human_resource successfully
gs_dump[port='5432'][human_resource][2018-12-13 09:40:24]: total time: 896 ms
```

10.1.2.3 导出表

GBase 8c 支持使用 gs_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 gbase 登录数据库主节点。

步骤 2 使用 gs_dump 同时导出指定表 hr.staffs 和 hr.employments。

```
gs_dump -U jack -f /home/gbase/backup/MPPDB_table_backup -p 5432 human_resource -t
hr.staffs -t hr.employments -F d
Password:
```

表 10-4 常用参数说明

参数	参数说明	举例
-U	连接数据库的用户名。 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 abcd@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/gbase/backup/ postgres_backup.tar
-p	指定服务器所侦听的 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	postgres
-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
-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 ● c：自定义归档 ● d：目录归档格式 ● t：tar 归档格式	-Ft
-T	不转储的表（或视图、或序列、或外表）对象列表，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。 当同时输入-t 和-T 时，会转储在-t 列表中，而不在-T 列表中的表对象。	-T table1

其他参数说明请参见《GBase 8c V5_3.0.x_工具参考》中“服务端工具 > gs_dump”章节。

----结束

示例

示例一：执行 gs_dump，导出表 hr.staffs 的定义和数据，导出文件格式为文本格式。

```
gs_dump -f /home/gbase/backup/MPPDB_table_backup.sql -p 5432 human_resource -t hr.staffs
-F p
Password:
gs_dump[port='5432'][human_resource][2017-07-21 17:05:10]: dump database
human_resource successfully
gs_dump[port='5432'][human_resource][2017-07-21 17:05:10]: total time: 3116 ms
```

示例二：执行 gs_dump，只导出表 hr.staffs 的数据，导出文件格式为 tar 归档格式。

```
gs_dump -f /home/gbase/backup/MPPDB_table_data_backup.tar -p 5432 human_resource -t
hr.staffs -a -F t
Password:
gs_dump[port='5432'][human_resource][2017-07-21 17:04:26]: dump database
human_resource successfully
gs_dump[port='5432'][human_resource][2017-07-21 17:04:26]: total time: 2570 ms
```

示例三：执行 gs_dump，导出表 hr.staffs 的定义，导出文件格式为目录归档格式。

```
gs_dump -f /home/gbase/backup/MPPDB_table_def_backup -p 5432 human_resource -t
hr.staffs -s -F d
Password:
gs_dump[port='5432'][human_resource][2017-07-21 17:03:09]: dump database
human_resource successfully
gs_dump[port='5432'][human_resource][2017-07-21 17:03:09]: total time: 2297 ms
```

示例四：执行 gs_dump，不导出表 hr.staffs，导出文件格式为自定义归档格式。

```
gs_dump -f /home/gbase/backup/MPPDB_table_backup4.dmp -p 5432 human_resource -T
hr.staffs -F c
Password:
gs_dump[port='5432'][human_resource][2017-07-21 17:14:11]: dump database
human_resource successfully
gs_dump[port='5432'][human_resource][2017-07-21 17:14:11]: total time: 2450 ms
```

示例五：执行 gs_dump，同时导出两个表 hr.staffs 和 hr.employments，导出文件格式为文本格式。

```
gs_dump -f /home/gbase/backup/MPPDB_table_backup1.sql -p 5432 human_resource -t
hr.staffs -t hr.employments -F p
Password:
gs_dump[port='5432'][human_resource][2017-07-21 17:19:42]: dump database
human_resource successfully
gs_dump[port='5432'][human_resource][2017-07-21 17:19:42]: total time: 2414 ms
```

示例六：执行 gs_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。

```
gs_dump -f /home/gbase/backup/MPPDB_table_backup2.sql -p 5432 human_resource -T
hr.staffs -T hr.employments -F p
Password:
gs_dump[port='5432'][human_resource][2017-07-21 17:21:02]: dump database
human_resource successfully
```

```
gs_dump[port='5432'][human_resource][2017-07-21 17:21:02]: total time: 3165 ms
```

示例七：执行 gs_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。

```
gs_dump -f /home/gbase/backup/MPPDB_table_backup3.tar -p 5432 human_resource -t
hr.staffs -t hr.employments --exclude-table-data hr.employments -F t
Password:
gs_dump[port='5432'][human_resource][2018-11-14 11:32:02]: dump database
human_resource successfully
gs_dump[port='5432'][human_resource][2018-11-14 11:32:02]: total time: 1645 ms
```

示例八：执行 gs_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。

```
gs_dump -f /home/gbase/backup/MPPDB_table_backup4.sql -p 5432 human_resource -t
hr.staffs --with- encryption AES128 --with-key abcdefg_?1234567 -F p
Password:
gs_dump[port='5432'][human_resource][2018-11-14 11:35:30]: dump database
human_resource successfully
gs_dump[port='5432'][human_resource][2018-11-14 11:35:30]: total time: 6708 ms
```

示例九：执行 gs_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。

```
gs_dump -f /home/gbase/backup/MPPDB_table_backup5.dmp -p 5432 human_resource -t
public.* -t hr.staffs -F c
Password:
gs_dump[port='5432'][human_resource][2018-12-13 09:40:24]: dump database
human_resource successfully
gs_dump[port='5432'][human_resource][2018-12-13 09:40:24]: total time: 896 ms
```

示例十： 执行 gs_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。

```
gs_dump -U jack -f /home/gbase/backup/MPPDB_view_backup6 -p 5432 human_resource -t
t1.test1 -- include-depend-objs --exclude-self -F d
Password:
gs_dump[port='5432'][jack][2018-11-14 17:21:18]: dump database human_resource
successfully
gs_dump[port='5432'][jack][2018-11-14 17:21:23]: total time: 4239 ms
```

10.1.3 导出所有数据库

10.1.3.1 导出所有数据库

GBase 8c 支持使用 gs_dumpall 工具导出所有数据库的全量信息，包含 GBase 8c 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 GBase 8c 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当前主机相同的一个主机环境，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，可以快速创建与当前主机相同的一个主机环境，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 gbase 登录数据库主节点。

步骤 2 使用 gs_dumpall 一次导出所有数据库信息。

```
gs_dumpall -U gbase -f /home/gbase/backup/MPPDB_backup.sql -p 5432
Password:
```

表 10-5 常用参数说明

参数	参数说明	举例
-U	连接数据库的用户名。 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 abcd@123
-f	将导出文件发送至指定目录文件夹。如果这里省略，则	-f

	使用标准输出。	home/gbase/backup/ postgres_backup.tar
-p	指定服务器所侦听的 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《GBase 8c V5_3.0.x_工具参考》中“服务端工具 > gs_dumpall”章节。

----结束

示例

示例一：执行 gs_dumpall，导出所有数据库全量信息（gbase 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。

```
gs_dumpall -U gbase -f /home/gbase/backup/MPPDB_backup.sql -p 5432
Password:
gs_dumpall[port='5432'][2017-07-21 15:57:31]: dumpall operation successful
gs_dumpall[port='5432'][2017-07-21 15:57:31]: total time: 9627 ms
```

示例二：执行 gs_dumpall，仅导出所有数据库定义（gbase 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。

```
gs_dumpall -U gbase -f /home/gbase/backup/MPPDB_backup.sql -p 5432 -s
Password:
gs_dumpall[port='5432'][2018-11-14 11:28:14]: dumpall operation successful
gs_dumpall[port='5432'][2018-11-14 11:28:14]: total time: 4147 ms
```

示例三：执行 gs_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。

```
gs_dumpall -f /home/gbase/backup/MPPDB_backup.sql -p 5432 -a --with-encryption AES128
--with-key abcdefg_?1234567
gs_dumpall[port='5432'][2018-11-14 11:32:26]: dumpall operation successful
gs_dumpall[port='5432'][2018-11-14 11:23:26]: total time: 4147 ms
```

10.1.3.2 导出全局对象

GBase 8c 支持使用 gs_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 gbase 登录数据库主节点。

步骤 2 使用 gs_dumpall 导出表空间对象信息。

```
gs_dumpall -U gbase -f /home/gbase/backup/MPPDB_tablespace.sql -p 5432 -t
Password:
```

表 10-6 常用参数说明

参数	参数说明	举例
-U	连接数据库的用户名。 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 abcd@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/gbase/backup/ postgres_backup.tar
-p	指定服务器所侦听的 TCP 端口或本地 Unix 域套接字后缀，以确保连接。	-p 5432
-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

其他参数说明请参见《GBase 8c V5_3.0.x_工具参考》中“服务端工具 > gs_dumpall”章节。

----结束

示例

示例一：执行 gs_dumpall，导出所有数据库的公共全局表空间信息和用户信息（gbase 用户为管理员用户），导出文件为文本格式。

```
gs_dumpall -U gbase -f /home/gbase/backup/MPPDB_globals.sql -p 5432 -g
Password:
gs_dumpall[port='5432'][2018-11-14 19:06:24]: dumpall operation successful
gs_dumpall[port='5432'][2018-11-14 19:06:24]: total time: 1150 ms
```

示例二： 执行 gs_dumpall，导出所有数据库的公共全局表空间信息（gbase 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。

```
gs_dumpall -U gbase -f /home/gbase/backup/MPPDB_tablespace.sql -p 5432 -t
--with-encryption AES128
--with-key abcdefg_?1234567
Password:
gs_dumpall[port='5432'][2018-11-14 19:00:58]: dumpall operation successful
gs_dumpall[port='5432'][2018-11-14 19:00:58]: total time: 186 ms
```

示例三：执行 gs_dumpall，导出所有数据库的公共全局用户信息（gbase 用户为管理员用户），导出文件为文本格式。

```
gs_dumpall -U gbase -f /home/gbase/backup/MPPDB_user.sql -p 5432 -r
Password:
gs_dumpall[port='5432'][2018-11-14 19:03:18]: dumpall operation successful
gs_dumpall[port='5432'][2018-11-14 19:03:18]: total time: 162 ms
```

10.1.4 无权限角色导出数据

gs_dump 和 gs_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，需要将有权限的用户赋权给无权限用户，然后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，gs_dump 和 gs_dumpall 会使用--role 参数指定的角色，完成导出动作。

操作步骤

步骤 1 以操作系统用户 gbase 登录数据库主节点。

步骤 2 使用 gs_dump 导出 human_resource 数据库数据。

用户 jack 不具备导出数据库 human_resource 的权限，而角色 role1 具备该权限，要实现导出数据库 human_resource，需要将 role1 赋权给 jack，然后可以在导出命令中设置-- role 角色为 role1，使用 role1 的权限，完成导出目的。导出文件格式为 tar 归档格式。

```
gs_dump -U jack -f /home/gbase/backup/MPPDB_backup.tar -p 5432 human_resource --role
role1 -- rolepassword abc@1234 -F t
Password:
```

表 10-7 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 abcd@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/gbase/backup/ postgres_backup.tar
-p	指定服务器所侦听的 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	postgres
-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
-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：	-Ft

	● 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	
-T	不转储的表（或视图、或序列、或外表）对象列表，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。 当同时输入-t 和-T 时，会转储在-t 列表中，而不在-T 列表中的表对象。	-T table1

其他参数说明请参见《GBase 8c V5_3.0.x_工具参考》中“服务端工具 > gs_dump”章节或“服务端工具> gs_dumpall”章节。

----结束

示例

示例一：执行 gs_dump 导出数据，用户 jack 不具备导出数据库 human_resource 的权限，而角色 role1 具备该权限，要实现导出数据库 human_resource，可以在导出命令中设置--role 角色为 role1，使用 role1 的权限，完成导出目的。导出文件格式为 tar 归档格式。

```
human_resource=# CREATE USER jack IDENTIFIED BY "1234@abc";
CREATE ROLE
human_resource=# GRANT role1 TO jack;
GRANT ROLE
gs_dump -U jack -f /home/gbase/backup/MPPDB_backup11.tar -p 5432 human_resource --role
role1 -- rolepassword abc@1234 -F t
Password:
gs_dump[port='5432'][human_resource][2017-07-21 16:21:10]: dump database
human_resource successfully
gs_dump[port='5432'][human_resource][2017-07-21 16:21:10]: total time: 4239 ms
```

示例二：执行 gs_dump 导出数据，用户 jack 不具备导出模式 public 的权限，而角色 role1 具备该权限，要实现导出模式 public，可以在导出命令中设置--role 角色为 role1，使用 role1 的权限，完成导出目的。导出文件格式为 tar 归档格式。

```
human_resource=# CREATE USER jack IDENTIFIED BY "1234@abc";
CREATE ROLE
human_resource=# GRANT role1 TO jack;
GRANT ROLE
```

```
gs_dump -U jack -f /home/gbase/backup/MPPDB_backup12.tar -p 5432 human_resource -n
public --role role1 --rolepassword abc@1234 -F t
Password:
gs_dump[port='5432'][human_resource][2017-07-21 16:21:10]: dump database
human_resource successfully
gs_dump[port='5432'][human_resource][2017-07-21 16:21:10]: total time: 3278 ms
```

示例三：执行 gs_dumpall 导出数据，用户 jack 不具备导出所有数据库的权限，而角色 role1（管理员）具备该权限，要实现导出所有数据库，可以在导出命令中设置--role 角色为 role1，使用 role1 的权限，完成导出目的。导出文件格式为文本归档格式。

```
human_resource=# CREATE USER jack IDENTIFIED BY "1234@abc"; CREATE ROLE
human_resource=# GRANT role1 TO jack;
GRANT ROLE
gs_dumpall -U jack -f /home/gbase/backup/MPPDB_backup.sql -p 5432 --role role1
--rolepassword abc@1234
Password:
gs_dumpall[port='5432'][human_resource][2018-11-14 17:26:18]: dumpall operation successful
gs_dumpall[port='5432'][human_resource][2018-11-14 17:26:18]: total time: 6437 ms
```

11 AI 特性

人工智能技术最早可以追溯到上世纪 50 年代，甚至比数据库系统的发展历史还要悠久。但是，由于各种各样客观因素的制约，在很长的一段时间内，人工智能技术并没有得到大规模的应用，甚至还经历了几次明显的低谷期。到了近些年，随着信息技术的进一步发展，从前限制人工智能发展的因素已经逐渐减弱，所谓的 ABC（AI、Big data、Cloud computing）技术也随之而诞生。

AI 特性子模块名为 DBMind，相对数据库其他功能更为独立，大致可分为 AI4DB、DB4AI 以及 AI in DB 三个部分。

- AI4DB 就是指用人工智能技术优化数据库的性能，从而获得更好地执行表现；也可以通过人工智能的手段实现自治、免运维等。主要包括自调优、自诊断、自安全、自运维、自愈等子领域；
- DB4AI 就是指打通数据库到人工智能应用的端到端流程，通过数据库来驱动 AI 任 务，统一人工智能技术栈，达到开箱即用、高性能、节约成本等目的。例如通过 SQL-like 语句实现推荐系统、图像检索、时序预测等功能，充分发挥数据库的高并行、列存储等优势，既可以避免数据和碎片化存储的代价，又可以避免因信息泄漏造成的安全风险；
- AI in DB 就是对数据库内核进行修改，实现原有数据库架构模式下无法实现的功能，

如利用 AI 算法改进数据库的优化器，实现更精确的代价估计等。

本章节所涉及的功能独立存在于数据库安装目录($GAUSSHOME)的 bin/dbmind 目录中，各个子功能存在于 dbmind 的子目录 components 中。提供 gs_dbmind 命令行供用户调用。与此同时，对于数据库内置 AI 的功能（如 DB4AI），以 SQL 语法和系统函数的形式呈现。

11.1 AI4DB：数据库自治运维

如上文所述，AI4DB 主要用于对数据库进行自治运维和管理，从而帮助数据库运维人员减少运维工作量。在实现上，DBMind 的 AI4DB 框架具有监控和服务化的性质，同时也提供即时 AI 工具包，提供开箱即用的 AI 运维功能（如索引推荐）。AI4DB 的监控平台以开源的 Prometheus 为主，DBMind 提供监控数据生产者 exporter，可与 Prometheus 平台完成对接。DBMind 的 AI4DB 服务架构如下图所示：

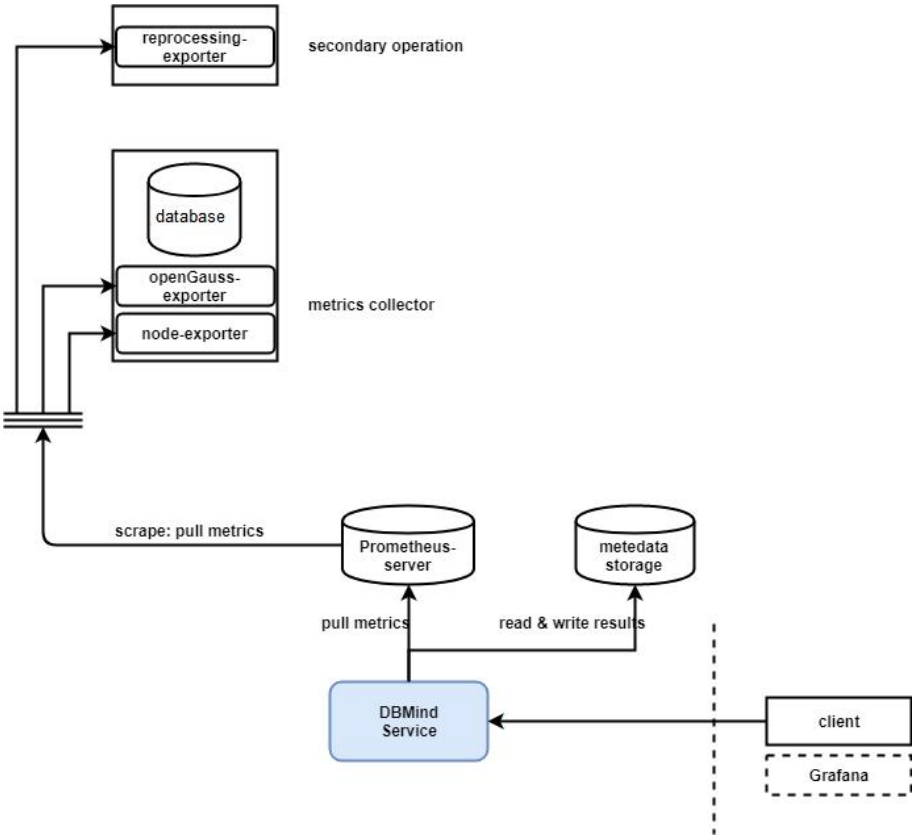


图 11-1 DBMind AI4DB 服务架构

图中各关键组件说明：

- DBMind Service: DBMind 后台服务，可用于定期离线计算，包括慢 SQL 根因分析、时序预测等；

- Prometheus-server: Prometheus 监控指标存储的服务器；
- metadatabase: DBMind 在离线计算结束后，将计算结果存储在此处，支持 openGauss、GBase 8c、SQLite 等数据库；
- client: 用户读取 DBMind 离线计算结果的客户端，目前仅实现命令行客户端；若采用 GBase 8c 等数据库存储计算 DBMind 计算结果，则用户可以自行配置 Grafana 等可视化工具对该结果进行可视化；
- openGauss-exporter: 用户从数据库节点上采集监控指标，供 DBMind 服务进行计算；
- node-exporter: Prometheus 官方提供的 exporter，可用于监控该节点的系统指标，如 CPU 和内存使用情况；
- reprocessing-exporter: 用于对 Prometheus 采集到的指标进行二次加工处理，例如计算 CPU 使用率等。

环境配置

DBMind 外置 AI 功能需要运行在 Python 3.6 版本及以上，需要的第三方依赖包记录在 AI 功能根目录（$GAUSSHOME/bin/dbmind）的 requirements.txt 文件中（包括 requirements-x86.txt 与 requirements-arrch64.txt，用户可根据自己平台类型选择） 中，可以通过 pip install 命令安装依赖，如：

```
pip install requirements-x86.txt
```

如果用户没有安装齐全所需的依赖，则当用户执行 gs_dbmind 命令时，会再次提醒用户安装第三方依赖。需要注意，该文件提供了 DBMind 所需的第三方依赖，若用户环境存在第三方包冲突等情况，可由用户根据实际情况进行处理。

11.1.1 DBMind 模式说明

用户可通过 gs_dbmind 命令调用 AI4DB 的全部功能，该命令可实现下列基本功能：

- 服务功能：service 子命令，包括创建并初始化配置目录、启动后台服务、关闭后台服务等；
- 调用组件：component 子命令，AI4DB 功能（如索引推荐、参数调优等）可通过该模式进行即时调用；
- 设置参数：set 子命令，通过该命令，可以一键修改配置目录中的配置文件值；当然，用户也可以通过文本编辑器进行手动修改；

用户可以通过 --help 选项获得上述模式的帮助信息，例如：

```
gs_dbmind --help
```

表 11-1 gs_dbmind 选项基本说明

参数	参数说明	取值范围
-h, --help	帮助命令	-
--version	版本号	-
service	服务功能相关的子命令	-
component	调用组件的子命令	-
set	修改配置文件的子命令	-

11.1.1.1 service 子命令

该子命令可用于对配置目录进行初始化，同时也可以实现启动和停止后台任务。

配置目录初始化

用户可通过 gs_dbmind service setup 子命令进行配置目录的初始化。该配置文件中可包括 DBMind 的配置文件、日志等内容。该目录中的部分文件说明：

- dbmind.conf：DBMind 的参数配置文件，用户可通过 gs_dbmind set 命令进行修改，也可通过文本编辑器进行手动修改。
- dynamic_config.db：DBMind 服务保存在本地节点的元信息，主要包括算法的超参数、监控阈值等；该文件为 DBMind 服务元信息，不可由用户直接配置。
- metric_map.conf：监控指标映射表，可用于适配到不同采集平台中。例如，在 DBMind 中，监控到的系统 cpu 使用率名为 os_cpu_usage，而用户自行实现的指标采集工具将 cpu 使用率命名为 my_cpu_usage_rate. 则在该种情况下，如果想要 DBMind 代表 cpu 使用率的指标名为 my_cpu_usage_rate，则需要修改该配置选项。即添加“os_cpu_usage = my_cpu_usage_rate”配置项进行映射。对于普通用户，建议直接使用 DBMind 配套的采集组件和方案，则不涉及修改该配置文件。
- logs：该目录中用于存储 DBMind 服务产生的日志。用户可通过两种方式进行配置目录的初始化，一种为交互式，另一种为非交互式。例如，待初始化的配置目录名为 confpath，则分别通过下述方法进行配置：

■ 交互式模式

```
gs_dbmind service setup -c confpath --interactive
```

执行完毕上述命令后，用户可通过命令行终端对配置项进行交互式配置。

■ 非交互式模式

非交互式模式总共分为三个步骤，即启动配置，修改配置项，初始化配置。其中第二个步骤需要用户通过文本编辑器手动编辑配置文件。具体步骤如下：

步骤 1 启动配置，执行下述命令：

```
gs_dbmind service setup -c confpath
```

步骤 2 执行完上述命令后，会在 confpath 目录下生成 dbmind.conf 配置文件，用户需要利用文本编辑器进行手动修改。需要修改的配置部分为“指标数据源数据库信息区【TSDB】”、“预测结果存储数据库信息区【METADATABASE】”和“自监控参数配置区【SELF-MONITORING】”相关参数的说明如下：

```
[TSDB] # 时序数据库相关信息，即指标数据源信息
name = prometheus # 时序数据库类型. 当前仅支持选择'prometheus'. host = # 时序数据库 IP 地址.
port = # 时序数据库端口号.
[METADATABASE] # 元数据库相关信息，及预测结果存储方式
dbtype = # 元数据库类型. 可选择: sqlite, gbase, postgresql。若该配置项名为 gbase，需关注 Python 驱动事宜，可参考下文中关于 Python 驱动的提示。
host = # 元数据库地址.
port = # 元数据库端口号.
username = # 元数据库用户信息.
password = # 元数据库密码，将会被加密存储
database = # 元数据库库名. 如果选择的数据库类型为 SQLite，则无需填写上述配置项，仅需填写此处。
DBMind 会根据 database 配置项的内容，在配置目录中生成对应文件名的 SQLite 数据库文件。
[SELF-MONITORING] # 自监控参数配置
detection_interval = 600 # 单位秒. 对 GBase 8c 进行健康检查的时间间隔（仅慢 SQL 根因诊断使用）.
last_detection_time = 600 # 单位秒. 用于健康检查的监控数据长度（仅慢 SQL 根因诊断使用）.
forecasting_future_time = 3600 # 单位秒，时序预测的长度（供时序预测特性使用），同时也是预测功能调度周期
# 待时序预测进行预测的关键指标项（仅供时序预测特性使用）
golden_kpi = os_cpu_usage, os_mem_usage, gaussdb_qps_by_instance
```

```
[LOG] # 日志相关信息
maxbytes = 10485760 # 默认值为 10Mb，单个日志文件的最大大小.如果 maxbytes 为零，则文件无限增长（建议该值不要设置太小，默认即可）.
backupcount = 1 # 日志文件最大数量.
level = INFO # 日志级别，也可配置为 DEBUG, INFO, WARNING, ERROR 模式。
```

步骤 3 待用户手动修改完上述参数后，需要执行下述命令进行配置项的初始化。在该阶段中，DBMind 会初步检查配置项的正确性、加密配置项中出现的明文密码、同时初始化用于存储结果数据的元数据库。

```
gs_dbmind service setup --initialize -c confpath
```

步骤 4 完成配置目录初始化过程，可基于该配置目录启动 DBMind 后台服务。

----结束

说明

(1) 配置文件注释信息用于在交互模式下对用户进行提示，有特殊含义不要手动修改或删除；

(2) 需要确保配置项的值与注释信息之间通过空格符分割，否则系统会将注释信息识别为配置项的值；

(3) 配置项中的特殊字符，如果需要转移，则通过转义符“百分号”（%）来转义，例如，用户配置的密码为 "password%"，则应通过“百分号”进行转义，即 "password%%"。

启动服务

当用户完成配置目录的初始化后，可基于此配置目录启动 DBMind 后台服务。例如配置目录为 confpath，则启动命令如下：

```
gs_dbmind service start -c confpath
```

当执行上述命令后，会提示服务已启动。在未指定任何附加参数时，该命令默认会启动所有的后台任务。如果用户只想启动某一个后台任务，需要添加参数 --only-run. 例如，用户只想启动慢 SQL 根因分析服务，则为：

```
gs_dbmind service start -c confpath --only-run slow_query_diagnosis
```

关闭服务

关闭服务与启动服务类似，其命令行结构更加简单，只需指定配置目录的地址即可。例如配置目录为 confpath，则为：

```
gs_dbmind service stop -c confpath
```

DBMind 服务会在后台执行完正在运行的任务后自行退出。

⚠ 注意

(1) [METADATABASE]中的元数据库用户需要具有在该数据库下的创表和数据插入更新权限，否则工具执行会出现异常。

(2) 当前不支持同一配置文件下分开启动多个服务。

(3) 工具提供了 requirement.txt 文件，用户可以通过该文件安装所需的第三方依赖。

命令参考

用户可以通过 --help 选项获得该模式的帮助信息，例如：

```
gs_dbmind service --help
usage: service [-h] -c DIRECTORY [--only-run {slow_query_diagnosis,forecast}] [--interactive
| --initialize]
{setup,start,stop}
positional arguments:
{setup,start,stop}
perform an action for service
optional arguments:
-h, --help       show this help message and exit
-c DIRECTORY, --conf DIRECTORY
set the directory of configuration files
--only-run {slow_query_diagnosis,forecast}
explicitly set a certain task running in the backend
--interactive   configure and initialize with interactive mode
--initialize     initialize and check configurations after configuring.
```

表 11-2 gs_dbmind service 子命令说明

参数	参数说明	取值范围
action	动作参数	setup：初始化配置目录。 start：服务启动。 stop：服务停止。
-c，--conf	配置文件目录地址	——
--initialize	配置参数初始化	——

--interactive	交互式输入配置参数	——
--only-run	选择只运行的模块	forecast：预测模块。 slow_query_diagnosis：慢 SQL 根因分析模块。
-h, --help	帮助命令	——

11.1.1.2 component 子命令

该子命令可以用于启动 DBMind 的组件，包括可用于监控指标的 exporter，以及 AI 功能等。该命令可以将用户通过命令行传入的命令转发给对应的子组件，故不同的子组件命令需参考其功能的对应说明，详见后文各个子组件对应章节，此处不再赘述。

命令参考

用户可以通过 --help 选项获得该模式的帮助信息，例如：

```
gs_dbmind component --help
usage: component [-h] COMPONENT_NAME ...
positional arguments:
COMPONENT_NAME choice a component to start. ['extract_log', 'forecast', 'index_advisor',
'opengauss_exporter', 'reprocessing_exporter', 'slow_query_diagnosis', 'sqldiag', 'xtuner']
ARGS    arguments for the component to start
optional arguments:
-h, --help      show this help message and exit
```

表 11-3 gs_dbmind component 子命令说明

参数	参数说明	取值范围
COMPONENT_NAME	子组件名	extract_log, forecast, index_advisor, opengauss_exporter, reprocessing_exporter, slow_query_diagnosis, sqldiag, xtuner
ARGS	子组件的参数	参考子组件的命令说明
-h, --help	帮助命令	-

11.1.1.3 set 子命令

该命令用于修改配置文件 dbmind.conf 中的参数值，与用户手动修改配置文件 dbmind.conf 一般无差异。例如修改配置目录 confpath 中的配置文件 dbmind.conf 中 TSDB 配置部分，host 参数的值，并将其设置为 127.0.0.1。则可通过下述命令实现：

```
gs_dbmind set TSDB host 127.0.0.1 -c confpath
```

在修改上述普通参数时，与手动修改配置文件 dbmind.conf 无差异。但是，当用户想要修改密码项时，则必须通过该命令才可以实现修改，否则通过用户手动修改的配置文件无法实现对明文密码的加密，例如：

```
gs_dbmind set METADATABASE password xxxxxx -c confpath
```

说明

该命令对于字符串是大小写敏感的，如果输错则可能出现执行过程错误。

命令参考

用户可以通过 --help 选项获得该模式的帮助信息，例如：

```
gs_dbmind set --help
usage: set [-h] -c DIRECTORY section option target
positional arguments:
section   which section (case sensitive) to set option   which option to set
target    the parameter target to set
optional arguments:
-h, --help      show this help message and exit
-c DIRECTORY, --conf DIRECTORY
set the directory of configuration files
```

表 11-4 模块命令行参数说明：python dbmind/ set xxx

参数	参数说明	取值范围
-h, –help	帮助命令	-
-c，–conf	配置文件目录 confpath	-
section	设置区	-
option	设置项	-
target	设置值	-

11.1.2 DBMind 的支持组件

支持组件是指 DBMind 提供的用于支撑整个服务、解决方案能够部署和实施的模块。它们本身不是 AI 功能，却是整个服务体系中非常重要的一环，用于支撑整个自治运维解决方案的快速实施，如用于采集数据库指标的 exporter 等。

11.1.2.1 Prometheus Exporter 组件

11.1.2.1.1 概述

Prometheus 是业内非常流行的开源监控系统，同时本身也是一款时序数据库。Prometheus 的采集端称之为 exporter，用来收集被监控模块的指标项。为了与 Prometheus 平台完成对接，AI 工具分别实现了两款 exporter，分别是用来采集数据库指标的 openGauss-exporter，以及对采集到的指标进行二次加工的 reprocessing- exporter。

须知

- Prometheus 和 exporter 是业内流行的监控和采集平台，部署在内网环境中，不对外部暴露接口，仅供内部监控平台使用。因此，为了增强该平台的安全性，一般需要用户或运维人员配置防火墙等，以便隔离外部访问，从而增强监控平台的安全性。
- Prometheus 平台在默认情况下，采用 Http 协议、并且没有任何安全访问限制。这是因为，该平台一般部署在内网环境中，攻击风险可控。如果用户希望提高安全性，可自行修改 Prometheus 的 TLS 配置选项，但仍不建议对外部直接暴露访问接口。

11.1.2.1.2 环境部署

步骤 1 通过命令行参数启动对应的 exporter 进程，同时在本地机器创建侦听端口号。

步骤 2 在 Promethues 的服务器端修改配置文件 prometheus.yml，将启动的 exporter 信息添加进去，例如：

```
scrape_configs:
...
- job_name: 'opengauss_exporter' static_configs:
- targets: ['127.0.0.1:9187']
...
```

----结束

其中，提供的 exporter 组件默认采用 Https 通信协议，因此需要用户默认提供 ssl 证书和

秘钥文件，并通过--ssl-keyfile 以及--ssl-certfile 提供。若用户不希望使用 Https 协议， 则可以通过--disable-https 选项禁用该模式。

说明

由于 GBase 8c 默认模式下的通信加密协议与 PostgreSQL 不兼容，导致通过 PyPI 源安装的 psycopg2-binary，由于是基于 PostgreSQL 编译的 Python 驱动，而默认无法连接至数据库。

11.1.2.1.3 使用指导

用户可通过 gs_dbmind 命令启动对应的 exporter。下面为用户演示一个完整的 Prometheus 监控平台的搭建过程。

步骤 1 部署 openGauss-exporter

启动 openGauss-exporter，采用默认侦听端口号 9187，侦听地址为 192.168.1.100，不采用 https 协议，则命令可以为：

```
gs_dbmind component opengauss_exporter --url postgresql://user:password@ip:port/dbname
--web.listen- address 192.168.1.100 --disable-https
```

步骤 2 部署 reprocessing-exporter

启动 reprocessing-exporter，采用默认侦听端口号 8181，侦听地址为 192.168.1.101，Prometheus-server IP 与端口号为 192.168.1.100:9090，采用 https 协议，则命令可以为：

```
gs_dbmind component reprocessing_exporter 192.168.1.100 9090 --web.listen-address
192.168.1.101 --ssl- keyfile server.key --ssl-certfile server.crt
```

步骤 3 部署 node-exporter

一般地，对于 Prometheus 监控平台都需要部署 node-exporter 用于监控 Linux 操作系统，后文提到的部分 AI 功能也需要依赖 node-exporter 采集 Linux 系统指标，故也需要用户来部署；只不过，该 node-exporter 为 Prometheus 自带组件，

需要用户自行下载安装，下载地址为 https://prometheus.io/download/ #node_exporter，

使用方法详见：https://prometheus.io/docs/guides/node-exporter/#installing-and-running-the-node-exporter。

用户可解压压缩包后直接运行该 node-exporter 进程，其默认端口号为 9100，启动命令行为：

```
./node_exporter
```

步骤 4 配置 Prometheus-server，修改配置文件 prometheus.yml，添加下述内容：

```
scrape_configs:
...
job_name: 'opengauss_exporter' static_configs:
- targets: ['192.168.1.100:9187']
job_name: 'reprocessing_exporter' scheme: https
tls_config: ca_file: xxx.crt
static_configs:
- targets: ['127.0.0.1:8181']
job_name: 'node_exporter' static_configs:
- targets: ['127.0.0.1:9100']
...
```

----结束

⚠ 注意

- openGauss-exporter 中连接数据库的用户需要 monitor admin 或以上权限，否则会出现部分指标无法采集的情况。
- openGauss-exporter 会从 dbe_perf.statement_history 中抽样慢 SQL 信息，dbe_perf.statement_history 视图慢 SQL 记录与 GUC 参数 log_min_duration_statement 和 track_stmnt_stat_level 相关，其中 log_min_duration_statement 是慢 SQL 阈值，单位毫秒，具体值由用户设置； track_stmnt_stat_level 是 SQL 记录级别，默认为'OFF,L0'，即只记录慢 SQL 信息，级别为 L0，用户在详细了解参数意义与作用情况下谨慎修改。
- openGauss-exporter 采集数据库相关信息，主要包括部分系统表和视图中的数据（具体参见代码中 opengauss_exporter 中的配置文件），node-exporter 采集系统 指标信息，主要与系统磁盘、CPU 等相关，reprocessing_exporter 基于 prometheus-server 中的某些指标（具体参见代码中 reprocessing_exporter 中的配置文件）进行二次加工，最终提供加工后的数据供用户使用。
- prometheus-server 在拉取 exporter 数据时有超时机制，超时时间由 scrape_timeout（默认 10s）控制，因此当 exporter 采集数据量较大时，用户可根据实际情况增大 scrape_timeout 以防止超时报错，另外需要注意的是 scrape_interval（采集间隔） 不能比 scrape_timeout 小，否则会出现异常。
- 如果数据库时区设置和系统不相同，可能会出现时间相关指标时间与系统时间不一致的情况，因此需要将用户时区与系统保持同步。

- 当使用 https 通信时，工具会检测证书与密钥文件权限以及证书有效期，如果文件权限大于 600 则会出现报警，证书有效期小于 90 天会出现报警。
- 当存在指标重复采集时，openGauss-exporter 会出现异常，异常信息会打印到日志中。
- openGauss-exporter 的 --config 、 --disable-settings-metrics 、 --disable-statement-history-metrics 三个参数需要注意，其存在以下几种情况：
    - 用户不指定其中任何参数，则工具会同时对 yamls 目录下的三个配置文件中的指标进行采集。
    - 用户显式指定--config，则工具不会采集 yamls 目录下 default.yml 中的指标，而会采集用户指定配置文件中的指标，同时 pg_settings.yml 和 statements.yml 正常 采集，此时用户需要注意指定的配置文件中的指标和 pg_settings.yml、statements.yml 中的指标不能存在重复采集的现象。
    - 用户显式指定--disable-settings-metrics，则工具不会采集 yamls 目录下 pg_settings.yml 中的指标，用户显式指定--disable-statement-history- metrics，则工具不会采集 yamls 目录下 statements.yml（慢 SQL 相关）中的指标。

11.1.2.1.4 获取帮助

用户可以通过--help 命令获取帮助信息，例如：

```
gs_dbmind component opengauss_exporter --help gs_dbmind component reprocessing_exporter --help
```

11.1.2.1.5 命令参考

reprocessing-exporter 的使用帮助详情：

```
gs_dbmind component reprocessing_exporter --help usage: [-h] [--disable-https] [--ssl-keyfile SSL_KEYFILE]
[--ssl-certfile SSL_CERTFILE]
[--web.listen-address WEB.LISTEN_ADDRESS] [--web.listen-port WEB.LISTEN_PORT]
[--collector.config COLLECTOR.CONFIG] [--log.filepath LOG.FILEPATH] [--log.level {debug,info,warn,error,fatal}] [--version]
prometheus_host prometheus_port
Reprocessing Exporter: A re-processing module for metrics stored in the Prometheus server.
positional arguments:
prometheus_host   from which host to pull data prometheus_port      the port to connect to the Prometheus host
optional arguments:
-h, --help        show this help message and exit
```

```
--disable-https      disable Https schema
--ssl-keyfile SSL_KEYFILE
set the path of ssl key file
--ssl-certfile SSL_CERTFILE
set the path of ssl certificate file
--web.listen-address WEB.LISTEN_ADDRESS
address on which to expose metrics and web interface
--web.listen-port WEB.LISTEN_PORT
listen port to expose metrics and web interface
--collector.config COLLECTOR.CONFIG
according to the content of the yaml file for metric collection
--log.filepath LOG.FILEPATH
the path to log
--log.level {debug,info,warn,error,fatal}
only log messages with the given severity or above. Valid levels: [debug, info, warn, error, fatal]
--version       show program's version number and exit
```

表 11-5 reprocessing-exporter 的命令行参数详情表

参数	参数说明	取值范围
prometheus_host	Prometheus-server 的 IP 地址	-
prometheus_port	Prometheus-server 的服务侦听端口号	1024-65535
-h, –help	帮助选项	-
–disable-https	禁用 Https 协议	-
–ssl-keyfile	Https 协议使用的证书私钥文件路径	-
–ssl-certfile	Https 协议使用的证书文件路径	-
–web.listen-address	该 exporter 服务的绑定 IP	-
–web.listen-port	该 exporter 服务的侦听端口	1024-65535
–collector.config	显性指定的待采集指标配置文件路径	-
–log.filepath	日志文件保存路径，默认保存在当前目录下	-
–log.level	日志文件的打印级别，默认为 INFO 级别	debug, info, warn, error, fatal

–version	显示版本信息	-

openGauss-exporter 的使用帮助详情：

```
gs_dbmind component opengauss_exporter --help
usage: [-h] --url URL [--config CONFIG] [--constant-labels CONSTANT_LABELS]
[--web.listen-address WEB.LISTEN_ADDRESS]
[--web.listen-port WEB.LISTEN_PORT]
[--web.telemetry-path WEB.TELEMETRY_PATH] [--disable-cache] [--disable-settings-metrics]
[--disable-statement-history-metrics] [--disable-https] [--ssl-keyfile SSL_KEYFILE]
[--ssl-certfile SSL_CERTFILE] [--parallel PARALLEL] [--log.filepath LOG.FILEPATH]
[--log.level {debug,info,warn,error,fatal}] [--version] openGauss Exporter (DBMind):
Monitoring for openGauss.
optional arguments:
-h, --help       show this help message and exit
--url URL        openGauss database target url.
--config CONFIG path to config dir or file.
--constant-labels CONSTANT_LABELS
a list of label=value separated by cgbasea(,).
--web.listen-address WEB.LISTEN_ADDRESS
address on which to expose metrics and web interface
--web.listen-port WEB.LISTEN_PORT
listen port to expose metrics and web interface
--web.telemetry-path WEB.TELEMETRY_PATH
path under which to expose metrics.
--disable-cache      force not using cache.
--disable-settings-metrics
not collect pg_settings.yml metrics.
--disable-statement-history-metrics
not collect statement-history metrics (including slow queries).
--disable-https      disable Https schema
--ssl-keyfile SSL_KEYFILE
set the path of ssl key file
--ssl-certfile SSL_CERTFILE
set the path of ssl certificate file
--parallel PARALLEL not collect pg_settings.yml metrics.
--log.filepath LOG.FILEPATH
the path to log
--log.level {debug,info,warn,error,fatal}
only log messages with the given severity or above. Valid levels: [debug, info, warn, error, fatal]
--version        show program's version number and exit
```

表 11-6 openGauss-exporter 的命令行参数详情表

参数	参数说明	取值范围
–url	openGauss server 的连接地址，例如 postgres://user:pwd@host:port/dbname	-
–constant-labels	常量标签，将采集到的指标项中强行添加该标签列表	1024-65535
-h, –help	帮助选项	-
–disable-https	禁用 Https 协议	-
–ssl-keyfile	Https 协议使用的证书私钥文件路径	-
–ssl-certfile	Https 协议使用的证书文件路径	-
–web.listen-address	该 exporter 服务的绑定 IP	-
–web.listen-port	该 exporter 服务的侦听端口	1024-65535
–web.telemetry-path	该 exporter 采集指标的 URI 地址，默认为 /metrics	-
–config	显性指定的待采集指标配置文件路径	-
–log.filepath	日志文件保存路径，默认保存在当前目录下	-
–log.level	日志文件的打印级别，默认为 INFO 级别	debug, info, warn, error, fatal
–version	显示版本信息	-
–disable-cache	禁止使用缓存	-
–disable-settings-metrics	禁止采集 pg_settings 表的值	-
–disable-statement-history-metrics	禁止收集 statement_history 表中的慢 SQL 信息	-
–parallel	连接到 GBase 8c 的数据库连接池的大小	正整数

11.1.2.1.6 常见问题处理

(1) 提示需要用户提供–ssl-keyfile 与–ssl-certfile 选项：

上述 exporter 默认采用 Https 模式通信，因此需要用户指定证书及其私钥文件的路径。相反，如果用户只想采用 Http 模式，则需要显性给定–disable-https 选项，从而禁用 Https 协议。

(2) 提示用户需要输入 PEM 密码（Enter PEM pass phrase）：

如果用户采用 Https 模式，并给定了证书及其秘钥文件的路径，且该秘钥文件是经过加密的，则需要用户输入该加密私钥证书文件的密码。

11.1.3 DBMind 的 AI 子功能

11.1.3.1 X-Tuner：参数调优与诊断

11.1.3.1.1 概述

X-Tuner 是一款数据库集成的参数调优工具，通过结合深度强化学习和全局搜索算法等 AI 技术，实现在无需人工干预的情况下，获取最佳数据库参数配置。本功能不强制与数据库环境部署到一起，支持独立部署，脱离数据库安装环境独立运行。

11.1.3.1.2 使用准备

前提条件与使用事项

- 数据库状态正常、客户端能够正常连接、且要求数据库内导入数据，以便调优程序可以执行 benchmark 测试调优效果。
- 使用本工具需要指定登录到数据库的用户身份，要求该登录到数据库上的用户具有足够的权限，以便可以获得充足的数据库状态信息。
- 使用登录到数据库宿主机上的 Linux 用户，需要将$GAUSSHOME/bin 添加到 PATH 环境变量中，即能够直接运行 gsql、gs_guc、gs_ctl 等数据库运维工具。
- 本工具支持以三种模式运行，其中 tune 和 train 模式要求用户配置好 benchmark 运行环境，并导入数据，本工具将会通过迭代运行 benchmark 来判断修改后的参数是否有性能提升。
- recommend 模式建议在数据库正在执行 workload 的过程中执行，以便获得更准确的实时 workload 信息。

- 本工具默认带有 TPC-C、TPC-H、TPC-DS 以及 sysbench 的 benchmark 运行脚本样 例，如果用户使用上述 benchmark 对数据库系统进行压力测试，则可以对上述配置文件进行适度修改或配置。如果需要适配用户自己的业务场景，需要您参照 benchmark 目录中的 template.py 文件编写驱动您自定义 benchmark 的脚本文件。

原理简介

调优程序是一个独立于数据库内核之外的工具，需要提供数据库及其所在实例的用户名和登录密码信息，以便控制数据库执行 benchmark 进行性能测试；在启动调优程序前，要求用户测试环境交互正常，能够正常跑通 benchmark 测试脚本、能够正常连接数据库。

说明

如果需要调优的参数中，包含重启数据库后才能使修改生效的参数，那么在调优过程中数据库将会重启多次。如果用户的数据库正在执行作业，请慎用 train 与 tune 模式。

调优程序 X-Tuner 包含三种运行模式，分别是:

- recommend：通过用户指定的用户名等信息登录到数据库环境中，获取当前正在运行的 workload 特征信息，根据上述特征信息生成参数推荐报告。报告当前数据库中不合理的参数配置和潜在风险等；输出根据当前正在运行的 workload 行为和特征；输出推荐的参数配置。该模式是秒级的，不涉及数据库的重启操作，其他模式可能需要反复重启数据库。
- train：通过用户提供的 benchmark 信息，不断地进行参数修改和 benchmark 的执行。通过反复的迭代过程，训练强化学习模型，以便用户在后面通过 tune 模式加载该模型进行调优。
- tune：使用优化算法进行数据库参数的调优，当前支持两大类算法，一种是深度强化学习，另一种是全局搜索算法（全局优化算法）。深度强化学习模式要求先运行 train 模式，生成训练后的调优模型，而使用全局搜索算法则不需要提前进行训练，可以直接进行搜索调优。

须知

如果在 tune 模式下，使用深度强化学习算法，要求必须有一个训练好的模型，且要求训练该模型时的参数与进行调优时的参数列表（包括 max 与 min）必须一致。

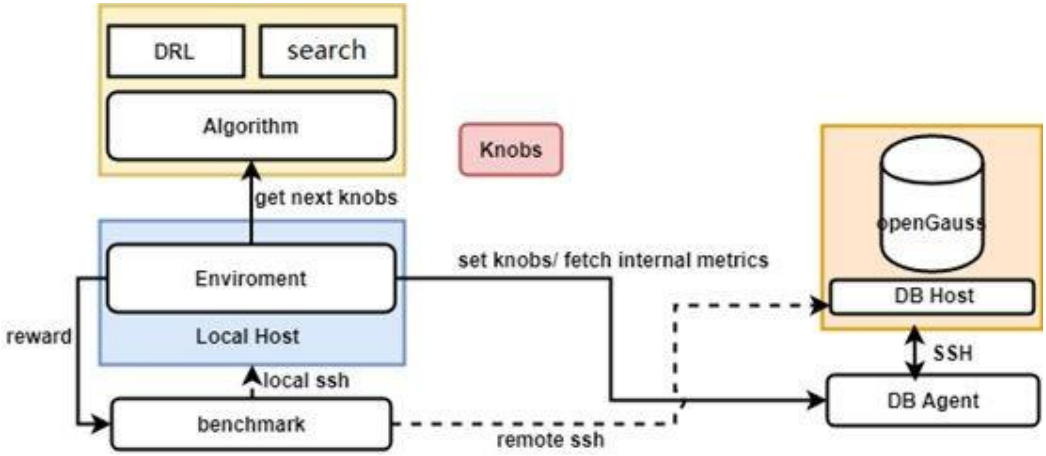


图 11-2 X-Tuner 结构图

- X-Tuner 的整体架构如图 11-2 X-Tuner 结构图所示，系统可以分为：
- DB 侧：通过 DB_Agent 模块对数据库实例进行抽象，通过该模块可以获取数据库内部的状态信息、当前数据库参数、以及设置数据库参数等。DB 侧包括登录数据库环境使用的 SSH 连接。
- 算法侧：用于调优的算法包，包括全局搜索算法（如贝叶斯优化、粒子群算法等）和深度强化学习（如 DDPG）。
- X-Tuner 主体逻辑模块：通过 Enviroment 模块进行封装，每一个 step 就是一次调优过程。整个调优过程通过多个 step 进行迭代。
- benchmark：由用户指定的 benchmark 性能测试脚本，用于运行 benchmark 作业，通过跑分结果反映数据库系统性能优劣。

  说明

  应确保 benchmark 脚本跑分结果越大表示性能越好。

  例如 TPCH 这种衡量 SQL 语句整体执行时长的 benchmark，可以通过取总体执行时间的相反数作为 benchmark 的输出分数。

X-Tuner 的运行和安装方法

执行下述命令即可获取 xtuner 功能帮助

```
gs_dbmind component xtuner --help
```

用户可据此给定不同的命令行执行相应的功能。

X-Tuner 的配置文件说明

X-Tuner 在运行前需要加载配置文件，用户可以通过 --help 命令查看默认加载的配置文件绝对路径：

```
...
-x TUNER_CONFIG_FILE, --tuner-config-file TUNER_CONFIG_FILE This is the path of the
core configuration file of the X-Tuner. You can specify the path of the new
configuration file. The default path is /path/to/xtuner/xtuner.conf.
You can modify the configuration file to control the tuning process.
...
```

修改配置文件的配置项可以指引 X-Tuner 执行不同的动作，用户可以根据自己的不同需求来修改配置文件的内容，配置文件的配置项说明详见表 15-11-8。如果需要修改配置文件的加载路径，则可以通过选项-x 命令行选项来指定。

Benchmark 的选择与配置

Benchmark 的驱动脚本存放路径为 X-Tuner 目录（即 $GAUSSHOME/bin/dbmind/components/xtuner，下同）的子目录 benchmark 中。X-Tuner 自带常用的 benchmark 驱动脚本，例如基于时间周期的探测脚本 (默认)、TPC-C、TPC-H 等。X-Tuner 通过调用 benchmark/init .py 文件中 get_benchmark_instance()命令来加载不同的 benchmark 驱动脚本，获取 benchmark 驱动实例。其中，benchmark 驱动脚本的格式说明如下：

- 驱动脚本文件名：表示 benchmark 的名字，该名字用于表示驱动脚本的唯一性，可通过在 X-Tuner 的配置文件中的配置项 benchmark_script 来指定选择加载哪个 benchmark 驱动脚本。
- 驱动脚本内容三要素：path 变量、cmd 变量以及 run 函数。下面分别介绍驱动脚本的内容三要素：
    - path 变量：表示 benchmark 脚本的存放地址，可以直接在驱动脚本中修改，也可以通过配置文件的 benchmark_path 配置项来指定。
    - cmd 变量：表示执行 benchmark 脚本需要运行的命令，可以直接在驱动脚本中修改，也可以通过配置文件的 benchmark_cmd 配置项来指定。cmd 中的文本允许使用占位符，用于获取某些运行 cmd 命令时的必要信息，使用示例参见 TPC-H 驱动脚本示例。这些占位符包括：
        - {host}：数据库宿主机的 IP 地址
        - {port}：数据库实例的侦听端口号
        - {user}：登录数据库系统上的用户名

- {password}：与登录数据库系统上的用户相匹配的密码
- {db}：正在进行调优的数据库名

- run 函数：该函数的函数签名为：

```
def run(remote_server, local_host) -> float:
```

其中，返回数据类型为 float，表示 benchmark 执行后的评估分数值，要求该值越大表示性能越好，例如使用 TPC-C 跑分结果 tpmC 即可作为返回值，TPC-H 的全部 SQL 语句执行总时间的相反数（取相反数后可保证返回值越大则性能越好）也可作为返回值。

remote_server 变量是 X-Tuner 程序传递给脚本使用的远端主机(数据库宿主机）的 shell 命令接口，local_host 变量是 X-Tuner 程序传递给脚本使用的本地主机（运行 X-Tuner 脚本的主机）的 shell 命令接口。上述 shell 命令接口提供的方法包括：

```
exec_command_sync(command, timeout) 功能：该方法用于在主机上执行 shell 命令。参数列表：
command 必选，数据类型可以是 str, 以及元素为 str 类型的 list 或 tuple; timeout 可选，表示命令执行的超时时长，单位是秒。
返回值：
返回二元组（stdout, stderr），stdout 表示标准输出流结果，stderr 表示标准错误流结果，数据类型均为 str. exit_status
功能：该属性表示最近一条 shell 命令执行后的退出状态码(exit status code)。
说明：一般情况，退出状态码为 0 表示执行正常，非 0 表示存在错误。
```

Benchmark 驱动脚本示例说明

(1) TPC-C 驱动脚本

```
from tuner.exceptions import ExecutionError
# WARN: You need to download the benchmark-sql test tool to the system, # replace the PostgreSQL JDBC driver with the openGauss driver,
# and configure the benchmark-sql configuration file.
# The program starts the test by running the following command:
path = '/path/to/benchmarksql/run' # TPC-C 测试脚本 benchmark-sql 的存放路径
cmd = "./runBenchmark.sh props.gs" # 自定义一个名为 props.gs 的 benchmark-sql 测试配置文件
def run(remote_server, local_host):
# 切换到 TPC-C 脚本目录下，清除历史错误日志，然后运行测试命令。
# 此处最好等待几秒钟，因为 benchmark-sql 测试脚本生成最终测试报告是通过一个 shell 脚本实现的， 整个过程会有延迟，
# 为了保证能够获取到最终的 tpmC 数值报告，我们这里选择等待 3 秒钟。
```

```
stdout, stderr = remote_server.exec_command_sync(['cd %s' % path, 'rm -rf benchmarksql-error.log', cmd, 'sleep 3'])
# 如果标准错误流中有数据，则报异常退出。
if len(stderr) > 0:
raise ExecutionError(stderr)
# 寻找最终 tpmC 结果
tpmC = None
split_string = stdout.split() # 对标准输出流结果进行分词。
for i, st in enumerate(split_string):
# 在 5.0 版本的 benchmark-sql 中，tpmC 最终测试结果数值在 ‘(NewOrders)’关键字的后两位，正常情况下，找到该字段后直接返回即可。
if "(NewOrders)" in  st: tpmC = split_string[i + 2] break
stdout, stderr = remote_server.exec_command_sync( "cat %s/benchmarksql-error.log" % path)
nb_err = stdout.count("ERROR:") # 判断整个 benchmark 运行过程中，是否有报错，记录报错的错误数
return float(tpmC) - 10 * nb_err # 这里将报错的错误数作为一个惩罚项，惩罚系数为 10，越高的惩罚系数表示越看中报错的数量.
```

(2) TPC-H 驱动脚本

```
import time
from tuner.exceptions import ExecutionError
# WARN: You need to import data into the database and SQL statements in the following path will be executed.
# The program automatically collects the total execution duration of these SQL statements. path = '/path/to/tpch/queries' # 存放 TPC-H 测试用的 SQL 脚本目录
cmd = "gsql -U {user} -W {password} -d {db} -p {port} -f {file}" # 需要运行 TPC-H 测试脚本的命令，一般使用'gsql -f 脚本文件' 来运行
def run(remote_server, local_host):
# 遍历当前目录下所有的测试用例文件名
find_file_cmd = "find . -type f -name '*.sql'"
stdout, stderr = remote_server.exec_command_sync(['cd %s' % path, find_file_cmd]) if len(stderr) > 0:
raise ExecutionError(stderr) files = stdout.strip().split('\n') time_start = time.time()
for file in files:
# 使用 file 变量替换 {file}，然后执行该命令行。perform_cmd = cmd.format(file=file)
stdout, stderr = remote_server.exec_command_sync(['cd %s' % path, perform_cmd]) if len(stderr) > 0:
print(stderr)
# 代价为全部测试用例的执行总时长
cost = time.time() - time_start
# 取相反数，适配 run 函数的定义：返回结果越大表示性能越好。
```

```
return - cost
```

11.1.3.1.3 使用示例

X-Tuner 支持三种模式，分别是获取参数诊断报告的 recommend 模式、训练强化学习模型的 train 模式、以及使用算法进行调优的 tune 模式。上述三种模式可以通过命令行参数来区别，通过配置文件来指定具体的细节。

配置数据库连接信息

三种模式连接数据库的配置项是相同的，有两种方式：一种是直接通过命令行输入详细的连接信息，另一种是通过 JSON 格式的配置文件输入，下面分别对两种指定数据库连接信息的方法进行说明。

(1) 通过命令行指定：

分别传递 --db-name --db-user --port --host --host-user 参数，可选 --host-ssh- port 参数，例如：

```
gs_dbmind component xtuner recommend --db-name postgres --db-user gbase --port 5678
--host 192.168.1.100 --host-user gbase
```

(2) 通过 JSON 格式的连接信息配置文件指定：

JSON 配置文件的示例如下，并假设文件名为 connection.json：

```
{
"db_name": "postgres", # 数据库名
"db_user": "dba",              # 登录到数据库上的用户名"host": "127.0.0.1", # 数据库宿主
机的 IP 地址"host_user": "dba",              # 登录到数据库宿主机的用户名"port": 5432, #
数据库的侦听端口号"ssh_port": 22          # 数据库宿主机的 SSH 侦听端口号
}
```

则可通过 -f connection.json 传递。

说明

为了防止密码泄露，配置文件和命令行参数中默认都不包含密码信息，用户在输入上述连接信息后，程序会采用交互式的方式要求用户输数据库密码以及操作系统登录用户的密码。

recommend 模式使用示例

对 recommend 模式生效的配置项为 scenario，若为 auto，则自动检测 workload 类型。执行下述命令，获取诊断结果：

```
gs_dbmind component xtuner recommend -f connection.json
```

则可以生成诊断报告如下：

```
Start to recommend knobs. Just a moment, please.
************************** Knob Recommendation Report **************************
INFO:
+--------------------------------------+----------------------+
|                Metric                |        Value         |
+--------------------------------------+----------------------+
|             workload_type            |          tp          |
|         average_connection_age       |          0           |
|         dirty_background_bytes       |          0           |
|             temp_file_size           |          0           |
|          current_connections         |         0.0          |
|          current_locks_count         |         0.0          |
|      current_prepared_xacts_count    |         0.0          |
|         rollback_commit_ratio        | 0.09168372786632421  |
|                uptime                |  0.122942879722222   |
| checkpoint_proactive_triggering_ratio |  0.488598416181662   |
|         fetched_returned_ratio       |  0.9915911452033203  |
|             cache_hit_rate           |  0.9979742937232552  |
|           read_write_ratio           |  123.86665549830312  |
|          all_database_size           |  134154882.48046875  |
|          search_modify_ratio         |  187.59523392981777  |
|                ap_index              |  2.3759498376861847  |
|           current_free_mem           |       31161892       |
|             os_mem_total             |       32779460       |
|        checkpoint_avg_sync_time      |   381.359603091308   |
|  checkpoint_dirty_writing_time_window |        450.0         |
|           max_processes              |          46          |
|         track_activity_size          |         46.0         |
|           write_tup_speed            |   6810.36309197048   |
|               used_mem               |     73988850.25      |
|            os_cpu_count              |          8           |
|             block_size               |         8.0          |
|           read_tup_speed             |   845237.440716804   |
|      shared_buffer_toast_hit_rate    |  98.16007359705611   |
|      shared_buffer_tidx_hit_rate     |  99.11667280088332   |
|       shared_buffer_idx_hit_rate     |  99.74473859023848   |
|      shared_buffer_heap_hit_rate     |  99.81099543813004   |
|           enable_autovacuum          |         True         |
|                is_64bit              |         True         |
|                 is_hdd               |         True         |
|            load_average              | [1.89, 3.175, 3.005] |
+--------------------------------------+----------------------+
p.s: The unit of storage is kB.
WARN:
[0]. The number of CPU cores is a little small. Please do not run too high concu
rrency. You are recommended to set max_connections based on the number of CPU co
res. If your job does not consume much CPU, you can also increase it.
[1]. The value of wal_buffers is a bit high. Generally, an excessively large val
ue does not bring better performance. You can also set this parameter to -1. The
 database automatically performs adaptation.
BAD:
[0]. The database runs for a short period of time, and the database description
may not be accumulated. The recommendation result may be inaccurate.
************************** Recommended Knob Settings ***************************
+---------------------------+-----------+---------+----------+---------+
|            name           | recommend |   min   |   max    | restart |
+---------------------------+-----------+---------+----------+---------+
|       shared_buffers      |  1638973  |  614614 | 1884818  |   True  |
|      max_connections      |     43    |    24   |   500    |   True  |
|    effective_cache_size   |  1638973  | 1638973 | 24584595 |  False  |
|        wal_buffers        |   51217   |   2048  |  51217   |   True  |
|      random_page_cost     |    3.0    |   2.0   |   3.0    |  False  |
| default_statistics_target |    100    |    10   |   150    |  False  |
+---------------------------+-----------+---------+----------+---------+
```

图 11-3 recommend 模式生成的报告示意图

在上述报告中，推荐了该环境上的数据库参数配置，并进行了风险提示。报告同时生成了当前 workload 的特征信息，其中有几个特征是比较有参考意义的：

- temp_file_size：产生的临时文件数量，如果该结果大于 0，则表明系统使用了临 时文件。使用过多的临时文件会导致性能不佳，如果可能的话，需要提高 work_mem 参数的配置。
- cache_hit_rate：shared_buffer 的缓存命中率，表明当前 workload 使用缓存的效率。
- read_write_ratio：数据库作业的读写比例。
- search_modify_ratio：数据库作业的查询与修改数据的比例。
- ap_index：表明当前 workload 的 AP 指数，取值范围是 0 到 10，该数值越大，表明越偏向于数据分析与检索。
- workload_type：根据数据库统计信息，推测当前负载类型，分为 AP、TP 以及 HTAP 三种类型。
- checkpoint_avg_sync_time：数据库在 checkpoint 时，平均每次同步刷新数据到 磁盘的时长，单位是毫秒。
- load_average：平均每个 CPU 核心在 1 分钟、5 分钟以及 15 分钟内的负载。一般 地，该数值在 1 左右表明当前硬件比较匹配 workload、在 3 左右表明运行当前作业压力比较大，大于 5 则表示当前硬件环境运行该 workload 压力过大（此时一般建议减少负载或升级硬件）。

  说明

  - recommend 模式会读取数据库中的 pg_stat_database 以及 pg_stat_bgwriter 等系统表中的信息，需要登录到数据库上的用户具有足够的权限（建议为管理员权限，可通过 alter user username sysadmin；授予 username 相应的权限）。
  - 由于某些系统表会一直记录统计信息，这可能会对负载特征识别造成干扰，因此建议最好先清空某些系统表的统计信息，运行一段时间的 workload 后再使用 recommend 模式进行诊断，以便获得更准确的结果。清除统计信息的方法为：

```
select pg_stat_reset_shared('bgwriter');
select pg_stat_reset();
```

train 模式使用示例

该模式是用来训练深度强化学习模型的，与该模式有关的配置项为：

- rl_algorithm：用于训练强化学习模型的算法，当前支持设置为 ddpg.
- rl_model_path：训练后生成的强化学习模型保存路径。
- rl_steps：训练过程的最大迭代步数。
- max_episode_steps：每个回合的最大步数。
- scenario：明确指定的 workload 类型，如果为 auto 则为自动判断。在不同模式下，推荐的调优参数列表也不一样。
- tuning_list：用户指定需要调哪些参数，如果不指定，则根据 workload 类型自动推荐应该调的参数列表。如需指定，则 tuning_list 表示调优列表文件的路径。一个调优列表配置文件的文件内容示例如下：

```
{
"work_mem": { "default": 65536,
"min": 65536,
"max": 655360,
"type": "int", "restart": false
},
"shared_buffers": { "default": 32000,
"min": 16000,
"max": 64000,
"type": "int", "restart": true
},
"random_page_cost": { "default": 4.0,
"min": 1.0,
"max": 4.0,
"type": "float", "restart": false
},
"enable_nestloop": { "default": true, "type": "bool", "restart": false
}
}
```

待上述配置项配置完成后，可以通过下述命令启动训练：

```
gs_dbmind component xtuner train -f connection.json
```

训练完成后，会在配置项 rl_model_path 指定的目录中生成模型文件。

tune 模式使用示例

tune 模式支持多种算法，包括基于强化学习（Reinforcement Learning, RL）的 DDPG 算法、基于全局搜索算法（Global OPtimization algorithm, GOP）算法的贝叶斯优化算法

（Bayesian Optimization）以及粒子群算法（Particle Swarm Optimization, PSO）。

与 tune 模式相关的配置项为：

- tune_strategy：指定选择哪种算法进行调优，支持 rl（使用强化学习模型进行调优）、gop（使用全局搜索算法）以及 auto（自动选择）。若该参数设置为 rl，则 rl 相关的配置项生效。除前文提到过的 train 模式下生效的配置项外，test_episode 配置项也生效，该配置项表明调优过程的最大回合数，该参数直接影响了调优过程的执行时间（一般地，数值越大越耗时）。
- gop_algorithm：选择何种全局搜索算法，支持 bayes 以及 pso。
- max_iterations：最大迭代轮次，数值越高搜索时间越长，效果往往越好。
- particle_nums：在 PSO 算法上生效，表示粒子数。
- scenario 与 tuning_list 见上文 train 模式中的描述。

待上述配置项配置完成后，可以通过下述命令启动调优：

```
gs_dbmind component xtuner tune -f connection.json
```

⚠ 注意

在使用 tune 和 train 模式前，用户需要先导入 benchmark 所需数据并检查 benchmark 能否正常跑通。调优过程结束后，调优程序会自动恢复调优前的数据库参数配置。

11.1.3.1.4 获取帮助

启动调优程序之前，可以通过如下命令获取帮助信息：

```
gs_dbmind component xtuner --help
```

输出帮助信息结果如下：

```
usage: [-h] [--db-name DB_NAME] [--db-user DB_USER] [--port PORT] [--host HOST]
[--host-user HOST_USER]
[--host-ssh-port HOST_SSH_PORT] [-f DB_CONFIG_FILE] [-x TUNER_CONFIG_FILE]
[-v]
{train,tune,recommend}
X-Tuner: a self-tuning tool integrated by openGauss. positional arguments:
{train,tune,recommend}
Train a reinforcement learning model or tune database
by model. And also can recommend best_knobs according to your workload.
optional arguments:
-h, --help      show this help message and exit
```

-f DB_CONFIG_FILE, --db-config-file DB_CONFIG_FILE
You can pass a path of configuration file otherwise you should enter database information by command arguments manually. Please see the template file share/server.json.template.
-x TUNER_CONFIG_FILE, --tuner-config-file TUNER_CONFIG_FILE This is the path of the core configuration file of the X-Tuner. You can specify the path of the new configuration file. The default path is xtuner.conf. You can modify the configuration file to control the tuning process.
-v, --version show program's version number and exit
Database Connection Information:
--db-name DB_NAME The name of database where your workload running on.
--db-user DB_USER Use this user to login your database. Note that the user must have sufficient permissions.
--port PORT Use this port to connect with the database.
--host HOST The IP address of your database installation host.
--host-user HOST_USER
The login user of your database installation host.
--host-ssh-port HOST_SSH_PORT
The SSH port of your database installation host.

11.1.3.1.5 命令参考

表 11-7 命令行参数

参数	参数说明	取值范围
mode	指定调优程序运行的模式	train，tune，recommend
--tuner- config-file, -x	X-Tuner 的核心参数配置文件路径，默认路径为安装目录下的 xtuner.conf	-
--db- config-file,-f	调优程序的用于登录到数据库宿主机上的连接信息配置文件路径，若通过该文件配置数据库连接信息，则下述数据库连接信息可省略	-
--db-name	指定需要调优的数据库名	-
--db-user	指定以何用户身份登陆到调优的数据库上	-
--host	数据库实例的宿主机 IP	-
--host-user	指定以何用户身份登陆到数据库实例的宿主机上，要求改用户名的环境变量中可以找到 gsql、gs_ctl 等数据库运维工具。	-

--host-ssh- port	数据库实例所在宿主机的 SSH 端口号，可选，默认为 22	-
--help, -h	返回帮助信息	-
--version, -v	返回当前工具版本号	-

表 11-8 配置文件中的参数详解

参数名	参数说明	取值范围
logfile	生成的日志存放路径	-
output_tuning_re sult	可选，调优结果的保存路径	-
verbose	是否打印详情	on, off
recorder_file	调优中间信息的记录日志存放路径	-
tune_strategy	调优模式下采取哪种策略	rl, gop, auto
drop_cache	是否在每一个迭代轮次中进行 drop cache，drop cache 可以使 benchmark 跑分结果更加稳定。若启动该参数，则需要将登录的系统用户加入到 /etc/ sudoers 列表中，同时为其增加 NOPASSWD 权限（由于该权限可能过高，建议临时启用该权限，调优结束后关闭）。	on, off
used_mem_pena lty_term	数据库使用总内存的惩罚系数，用于防止通过无限量占用内存而换取的性能表现。该数值越大，惩罚力度越大。	建议 0 ~ 1
rl_algorithm	选择何种 RL 算法	ddpg
rl_model_path	RL 模型保存或读取路径，包括保存目录名与文件名前缀。在 train 模式下该路径用于保存模型，在 tune 模式下则用于读取模型文件	-
rl_steps	深度强化学习算法迭代的步数	-
max_episode_ste ps	每个回合的最大迭代步数	-
test_episode	使用 RL 算法进行调优模式的回合数	-

gop_algorithm	采取何种全局搜索算法	bayes, pso
max_iterations	全局搜索算法的最大迭代轮次（并非确定数值，可能会根据实际情况多跑若干轮）	-
particle_nums	PSO 算法下的粒子数量	-
benchmark_scrip t	使用何种 benchmark 驱动脚本，该选项指定加载 benchmark 路径下同名文件，默认支持 TPC-C、TPC- H 等典型 benchmark	tpcc, tpch, tpcds, sysbenc h ...
benchmark_path	benchmark 脚本的存储路径，若没有配置该选项，则使用 benchmark 驱动脚本中的配置	-
benchmark_cmd	启动 benchmark 脚本的命令，若没有配置该选项，则使用 benchmark 驱动脚本中的配置	-
benchmark_perio d	仅对 period benchmark 有效，表明整个 benchmark 的测试周期是多少，单位是秒	-
scenario	用户指定的当前 workload 所属的类型	tp, ap, htap
tuning_list	准备调优的参数列表文件，可参考 share/ knobs.json.template 文件	-

11.1.3.1.6 常见问题处理

- 数据库实例连接失败：请检查数据库实例的情况，是否数据库实例出现了问题或安全权限配置（pg_hba.conf 文件中的配置项）不正确。
- 重启失败：请检查数据库实例健康情况，确保数据库实例工作正常。
- 跑 TPC-C 作业时发现性能越来越慢：TPC-C 等高并发场景下的压力测试，往往伴随着大量的数据修改。由于每一次测试并非是幂等的（TPC-C 数据库数据量的增加、没有进行 vacuum full 清理掉失效元组、数据库没有触发 checkpoint、没有进行 drop cache 等），因此一般建议 TPC-C 等伴随着较多数据写入的 benchmark 应该 每隔一段时间（视具体并发量、执行时长的不同而异）重新导入一次数据，比较简单的方法是备份$PGDATA 目录。
- TPC-C 跑作业时，TPC-C 驱动脚本报异常 “TypeError: float() argument must be a string or a number, not 'NoneType'” （不能将 None 转换为 float 类型）：这是因为没有获取到 TPC-C 的压测返回结果，造成该问题的原因比较多，请首先手动检测是否能够跑通

TPC-C 并能够获取返回结果。若无上述问题，则建议将 TPC-C 驱动脚本中的命令列表中的 "sleep" 命令延迟时间设得更大一些。

11.1.3.2 Index-advisor：索引推荐

本节介绍索引推荐的功能，共包含三个子功能：单 query 索引推荐、虚拟索引和 workload 级别索引推荐。

11.1.3.2.1 单 query 索引推荐

单 query 索引推荐功能支持用户在数据库中直接进行操作，本功能基于查询语句的语义信息和数据库的统计信息，对用户输入的单条查询语句生成推荐的索引。本功能涉及的函数接口如下。

表 11-9 单 query 索引推荐功能的接口

函数名	参数	功能
gs_index_advise	SQL 语句字符串	针对单条查询语句生成推荐索引

说明

- 本功能仅支持单条 SELECT 类型的语句，不支持其他类型的 SQL 语句。
- 本功能暂不支持列存表、段页式表、普通视图、物化视图、全局临时表以及密态数据库。

使用方法

使用上述函数，获取针对该 query 生成的推荐索引，推荐结果由索引的表名和列名组成。

例如：

```
gsql=> select "table", "column" from gs_index_advise('SELECT c_discount from
bmsql_customer where c_w_id = 10');
        table         |   column
----------------+----------
 bmsql_customer | c_w_id
(1 row)
```

上述结果表明：应当在 bmsql_customer 的 c_w_id 列上创建索引，例如可以通过下述 SQL 语句创建索引：

```
CREATE INDEX idx on bmsql_customer(c_w_id);
```

某些 SQL 语句，也可能被推荐创建联合索引，例如：

```
postgres=# select "table", "column" from gs_index_advise('select name, age, sex from t1 where age >= 18 and age < 35 and sex = "f";');
 table | column
-------+------------
 t1    | age, sex
(1 row)
```

则上述语句表明应该在表 t1 上创建一个联合索引 (age, sex)，则可以通过下述命令创建：

```
CREATE INDEX idx1 on t1(age, sex);
```

针对分区表可推荐具体索引类型，例如：

```
postgres=# select "table", "column", "indextype" from gs_index_advise('select name, age, sex from range_table where age = 20;');
 table | column | indextype
-------+--------+-----------
 t1    | age    | global
(1 row)
```

说明

系统函数 gs_index_advise()的参数是文本型，如果参数中存在如单引号 (') 等特殊字符，可以使用单引号 (') 进行转义，可参考上述示例。

11.1.3.2.2 虚拟索引

虚拟索引功能支持用户在数据库中直接进行操作，本功能将模拟真实索引的建立，避免真实索引创建所需的时间和空间开销，用户基于虚拟索引，可通过优化器评估该索引对指定查询语句的代价影响。

本功能涉及的系统函数接口如下表所示：

表 11-10 虚拟索引功能的接口

函数名	参数	功能
hypopg_create_index	创建索引语句的字符串	创建虚拟索引。
hypopg_display_index	无	显示所有创建的虚拟索引信息。
hypopg_drop_index	索引的 oid	删除指定的虚拟索引。

hypopg_reset_index	无	清除所有虚拟索引。
hypopg_estimate_size	索引的 oid	估计指定索引创建所需的空间大小。

本功能涉及的 GUC 参数如下：

表 11-11 虚拟索引功能的 GUC 参数

参数名	功能	默认值
enable_hypo_index	是否开启虚拟索引功能	off

使用步骤

(1) 使用函数 hypopg_create_index 创建虚拟索引。例如：

```
gsql=> select * from hypopg_create_index('create index on bmsql_customer(c_w_id)');
 indexrelid |                 indexname
------------+-------------------------------------
     329726 | <329726>btree_bmsql_customer_c_w_id
(1 row)
```

(2) 开启 GUC 参数 enable_hypo_index，该参数控制数据库的优化器进行 EXPLAIN 时是否考虑创建的虚拟索引。通过对特定的查询语句执行 explain，用户可根据优化器给出的执行计划评估该索引是否能够提升该查询语句的执行效率。例如：

```
gsql=> set enable_hypo_index = on;
SET
```

开启 GUC 参数前，执行 EXPLAIN + 查询语句：

```
gsql=> explain SELECT c_discount from bmsql_customer where c_w_id = 10;
                                 QUERY PLAN
-----------------------------------------------------------------------
 Seq Scan on bmsql_customer  (cost=0.00..52963.06 rows=31224 width=4)
   Filter: (c_w_id = 10)
(2 rows)
```

开启 GUC 参数后，执行 EXPLAIN + 查询语句：

```
gsql=> explain SELECT c_discount from bmsql_customer where c_w_id = 10;
                                                    QUERY PLAN
-------------------------------------------------------------------------------------------------------------------
 [Bypass]
```

```
 Index Scan using <329726>btree_bmsql_customer_c_w_id on bmsql_customer
(cost=0.00..39678.69 rows=31224 width=4)
    Index Cond: (c_w_id = 10)
(3 rows)
```

通过对比两个执行计划可以观察到，该索引预计会降低指定查询语句的执行代价，用户可考虑创建对应的真实索引。

(3) （可选）使用函数 hypopg_display_index 展示所有创建过的虚拟索引。例如：

```
gsql=> select * from hypopg_display_index();
                    indexname                    | indexrelid |       table        |
column
-----------------------------------------------+------------+----------------+------------------
 <329726>btree_bmsql_customer_c_w_id           |       329726 | bmsql_customer | (c_w_id)
 <329729>btree_bmsql_customer_c_d_id_c_w_id |       329729 | bmsql_customer | (c_d_id,
c_w_id)
(2 rows)
```

(4) （可选）使用函数 hypopg_estimate_size 估计虚拟索引创建所需的空间大小（单位：字节）。例如：

```
gsql=> select * from hypopg_estimate_size(329730);
 hypopg_estimate_size
----------------------
                15687680
(1 row)
```

(5) 删除虚拟索引。

使用函数 hypopg_drop_index 删除指定 oid 的虚拟索引。例如：

```
gsql=> select * from hypopg_drop_index(329726);
 hypopg_drop_index
-------------------
 t
(1 row)
```

使用函数 hypopg_reset_index 一次性清除所有创建的虚拟索引。例如：

```
gsql=> select * from hypopg_reset_index();
 hypopg_reset_index
--------------------

(1 row)
```

说明

- 执行 EXPLAIN ANALYZE 不会涉及虚拟索引功能。
- 创建的虚拟索引是数据库实例级别的，各个会话（session）之间可共享设置，关闭会话后虚拟索引仍可存在，但是重启数据库后将被清空。
- 本功能暂不支持视图、物化视图、列存表。

11.1.3.2.3 workload 级别索引推荐

对于 workload 级别的索引推荐，用户可通过运行数据库外的脚本使用此功能，本功能将包含有多条 DML 语句的 workload 作为输入，最终生成一批可对整体 workload 的执行表现进行优化的索引。同时，本功能提供从日志中抽取业务数据 SQL 流水的功能。

前提条件

- 数据库状态正常、客户端能够正常连接。
- 当前执行用户下安装有 gsql 工具，该工具路径已被加入到 PATH 环境变量中。
- 若使用本功能提供的业务数据抽取功能，需提前将要收集的节点的 GUC 参数按如下设置：

```
log_min_duration_statement = 0
log_statement= 'all'
```

说明

业务数据抽取完毕建议将上述 GUC 参数复原，否则容易导致日志文件膨胀。

业务数据抽取脚本使用步骤

(1) 按前提条件中要求设置相关 GUC 参数。

(2) 执行根据日志抽取 SQL 语句的功能，命令如下：

```
gs_dbmind component extract_log [l LOG_DIRECTORY] [f OUTPUT_FILE] [p
LOG_LINE_PREFIX] [-d DATABASE] [-U USERNAME][--start_time] [--sql_amount]
[--statement] [--json] [--max_reserved_period] [--max_template_num]
```

其中的输入参数依次为：

- LOG_DIRECTORY：pg_log 的存放目录。
- OUTPUT_PATH：输出 SQL 流水文件文件的保存路径，即抽取出的业务数据存放的文件路径。

- LOG_LINE_PREFIX：指定每条日志信息的前缀格式。
- DATABASE：（可选）数据库名称，不指定默认所有数据库。
- USERNAME：（可选）用户名称，不指定默认所有用户。
- start_time：（可选）日志收集的开始时间，不指定默认所有文件。
- sql_amount：（可选）收集 SQL 数量的最大值，不指定默认收集所有 SQL。
- statement：（可选）表示收集 pg_log 日志中 statement 标识开头的 SQL，不指定默认不收集。
- json：（可选）指定收集日志的文件存储格式为 SQL 归一化后的 json，不指定默认格式每条 SQL 占一行。
- max_reserved_period：（可选）指定 json 模式下，增量收集日志中保留的模板的最大的更新时长，不指定默认都保留，单位/天。
- max_template_num：（可选）指定 json 模式下保留的最大模板数量，不指定默认都保留。

使用示例：

```
gs_dbmind component extract_log $GAUSSLOG/pg_log/dn_6001 sql_log.txt
'%m %c %d %p %a %x %n %e' -d postgres -U gbase --start_time '2021-07-06 00:00:00'
--statement
```

说明

若指定-d/-U 参数，日志打印每条日志信息的前缀格式需包含%d、%u，若需要抽取事务，必须指定%p，详见 log_line_prefix 参数。max_template_num 参数设置建议不超 5000 条，避免 workload 索引推荐执行时间过长。

(3) 将 1 中设置的 GUC 参数还原为设置前的值。

索引推荐脚本使用步骤

(1) 准备好包含有多条 DML 语句的文件作为输入的 workload，文件中每条语句占据一行。用户可从数据库的离线日志中获得历史的业务语句。

(2) 运行本功能，命令如下：

```
gs_dbmind component index_advisor [p PORT] [d DATABASE] [f FILE] [--h HOST] [-U
USERNAME] [-W PASSWORD][--schema SCHEMA]
```

```
[--max_index_num MAX_INDEX_NUM][--max_index_storage MAX_INDEX_STORAGE]
[--multi_iter_mode] [--multi_node]  [--json] [--driver] [--show_detail]
```

其中的输入参数依次为：

- PORT：连接数据库的端口号。
- DATABASE：连接数据库的名字。
- FILE：包含 workload 语句的文件路径。
- HOST：（可选）连接数据库的主机号。
- USERNAME：（可选）连接数据库的用户名。
- PASSWORD：（可选）连接数据库用户的密码。
- SCHEMA：模式名称。
- MAX_INDEX_NUM：（可选）最大的索引推荐数目。
- MAX_INDEX_STORAGE：（可选）最大的索引集合空间大小。
- multi_node：（可选）指定当前是否为分布式数据库实例。
- multi_iter_mode：（可选）算法模式，可通过是否设置该参数来切换算法。
- json：（可选）指定 workload 语句的文件路径格式为 SQL 归一化后的 json，默认格式每条 SQL 占一行。
- driver：（可选）指定是否使用 python 驱动器连接数据库，默认 gsql 连接。
- show_detail：（可选）是否显示当前推荐索引集合的详细优化信息。

例如：

```
gs_dbmind component index_advisor 6001 postgres tpcc_log.txt --schema public
--max_index_num 10 --multi_iter_mode
```

推荐结果为一批索引，以多个创建索引语句的格式显示在屏幕上，结果示例。

```
create index ind0 on public.bmsql_stock(s_i_id,s_w_id);
create index ind1 on public.bmsql_customer(c_w_id,c_id,c_d_id);
create index ind2 on public.bmsql_order_line(ol_w_id,ol_o_id,ol_d_id);
create index ind3 on public.bmsql_item(i_id);
create index ind4 on public.bmsql_oorder(o_w_id,o_id,o_d_id);
create index ind5 on public.bmsql_new_order(no_w_id,no_d_id,no_o_id);
create index ind6 on public.bmsql_customer(c_w_id,c_d_id,c_last,c_first);
create index ind7 on public.bmsql_new_order(no_w_id);
```

```
create index ind8 on public.bmsql_oorder(o_w_id,o_c_id,o_d_id);
create index ind9 on public.bmsql_district(d_w_id);
```

说明

multi_node 参数需严格按照当前数据库架构进行指定，否则推荐结果不全，甚至导致无推荐结果。

11.1.3.3 AI4DB：慢 SQL 根因分析

11.1.3.3.1 概述

慢 SQL 一直是数据运维中的痛点问题，如何有效诊断慢 SQL 根因是当前一大难题，工具结合 GBase 8c 自身特点融合了现网 DBA 慢 SQL 诊断经验，该工具可以支持慢 SQL 根因 15+，能同时按照可能性大小输出多个根因并提供针对性的建议。

11.1.3.3.2 环境部署

- 数据库运行正常。
- 指标采集系统运行正常。

11.1.3.3.3 使用指导

假设用户已经初始化配置文件目录 confpath，则可以通过下述命令实现本特性的功能：

- 仅启动慢 SQL 诊断功能（输出 Top3 根因），启动命令如下（更多用法参考对 service 子命令的说明）：

```
gs_dbmind service start -c confpath --only-run slow_query_diagnosis
```

- 用户交互式慢 SQL 诊断，命令如下：

```
gs_dbmind component slow_query_diagnosis show -c confpath --query SQL --start-time
timestamps0
--end-time timestamps1
```

- 用户手动清理历史预测结果，命令如下：

```
gs_dbmind component slow_query_diagnosis clean -c confpath --retention-days DAYS
```

- 停止已启动的服务，命令如下：

```
gs_dbmind service stop -c confpath
```

11.1.3.3.4 获取帮助

模块命令行说明：

```
gs_dbmind component slow_query_diagnosis --help usage: [-h] -c DIRECTORY [--query
SLOW_QUERY]
[--start-time TIMESTAMP_IN_MICROSECONDS]
[--end-time TIMESTAMP_IN_MICROSECONDS] [--retention-days DAYS]
{show,clean}
Slow Query Diagnosis: Analyse the root cause of slow query positional arguments:
{show,clean} choose a functionality to perform
optional arguments:
-h, --help       show this help message and exit
-c DIRECTORY, --conf DIRECTORY
set the directory of configuration files
--query SLOW_QUERY set a slow query you want to retrieve
--start-time TIMESTAMP_IN_MICROSECONDS
set the start time of a slow SQL diagnosis result to be retrieved
--end-time TIMESTAMP_IN_MICROSECONDS
set the end time of a slow SQL diagnosis result to be retrieved
--retention-days DAYS
clear historical diagnosis results and set the maximum number of days to retain data
```

11.1.3.3.5 命令参考

表 11-12 gs_dbmind component slow_query_diagnosis 命令行说明

参数	参数说明	取值范围
-h, --help	帮助命令	-
action	动作参数	show：结果展示 clean：清理结果
-c，--conf	配置目录	-
--query	慢 SQL 文本	*
--start-time	显示开始时间的时间戳，单位毫秒	非负整数
--end-time	显示结束时间的时间戳，单位毫秒	非负整数
--retention-days	清理天数级结果	非负实数

11.1.3.3.6 常见问题处理

- 如果用户对没有执行过的慢 SQL 执行交互式诊断命令，则无法给出诊断结果。

- exporter 指标采集功能没有启动时运行慢 SQL 诊断功能，此时功能无法正常运行。
- 配置文件中的参数重新设置后，需要重新启动服务进程才能生效。

11.1.3.4 AI4DB：趋势预测

11.1.3.4.1 概述

趋势预测功能模块主要实现基于历史时序数据预测未来时序变化趋势。该模块框架解耦，可以实现不同预测算法的灵活替换，并且该模块功能可以实现不同特征时序的算法自动选择，支持线性特征时序预测 LR 回归算法和非线性特征预测 ARIMA 算法。目前该模块可以覆盖线性时序、非线性时序和周期时序的准确预测。

11.1.3.4.2 环境部署

指标采集系统运行正常。

11.1.3.4.3 使用指导

假设用户已经初始化配置文件目录 confpath，则可以通过下述命令实现本特性的功能：

- 仅启动趋势预测功能，启动命令如下（更多用法参考对 service 子命令的说明）：

```
gs_dbmind service start -c confpath --only-run forecast
```

- 用户交互式趋势预测，查看 timestamps0 到 timestamps1 时间段内的预测结果，命令如下：

```
gs_dbmind component forecast show -c confpath --start-time timestamps0 --end-time
timestamps1
```

- 用户手动清理手动清理历史预测结果，命令如下：

```
gs_dbmind component forecast clean -c confpath --retention-days DAYS
```

- 停止已启动的服务，命令如下：

```
gs_dbmind service stop -c confpath
```

11.1.3.4.4 获取帮助

模块命令行说明：

```
gs_dbmind component forecast --help
usage: [-h] -c DIRECTORY [--metric-name METRIC_NAME] [--host HOST] [--start-time
TIMESTAMP_IN_MICROSECONDS] [--end-time TIMESTAMP_IN_MICROSECONDS]
[--retention-days DAYS]
{show,clean}
```

```
Workload Forecasting: Forecast monitoring metrics positional arguments:
{show,clean} choose a functionality to perform
optional arguments:
-h, --help       show this help message and exit
-c DIRECTORY, --conf DIRECTORY
set the directory of configuration files
--metric-name METRIC_NAME
set a metric name you want to retrieve
--host HOST set a host you want to retrieve
--start-time TIMESTAMP_IN_MICROSECONDS
set a start time of for retrieving
--end-time TIMESTAMP_IN_MICROSECONDS
set a end time of for retrieving
--retention-days DAYS
clear historical diagnosis results and set the maximum number of days to retain data
```

11.1.3.4.5 命令参考

表 11-13 gs_dbmind component forecast 命令行说明

参数	参数说明	取值范围
-h, --help	帮助命令	-
action	动作参数	show：结果展示 clean：清理结果
-c，--conf	配置目录	-
--metric-name	指定显示指标名，用于过滤	-
--host	指定服务 IP 和端口号，用于过滤	-
--start-time	显示开始时间的时间戳，单位毫秒	非负实数
--end-time	显示结束时间的时间戳，单位毫秒	非负实数
--retention-days	保留结果天数	非负实数

11.1.3.4.6 常见问题处理

- 综合实际业务与模型预测效果考虑，趋势预测时长建议不要太短，建议大于 3600 秒（如果指标采集周期为 15 秒，则数据量为 240 个），否则预测效果会变差，并且数据量极小时，服务会异常，因此默认配置为 3600 秒

- 配置文件中的参数重新设置后，需要重新启动服务进程才能生效。

11.1.3.5 SQLdiag：慢 SQL 发现

SQLdiag 是 GBase 8c 中 SQL 语句执行时长预测工具。现有的预测技术主要基于执行计划的预测方法，但这些预测方案仅适用于 OLAP 场景且可以获取执行计划的任务，对于 OLTP 或者 HTAP 这样的快速、简单查询是没有太多使用价值的。与上述方案不同，SQLdiag 着眼于数据库的历史 SQL 语句，通过对历史 SQL 语句的执行表现进行总结归纳，将之再用于推断新的未知业务上。由于短时间内数据库 SQL 语句执行时长不会有太大的差距，SQLdiag 可以从历史数据中检测出与已执行 SQL 语句相似的语句结果集，并基于 SQL 向量化技术通过 SQL 模板化和深度学习这两种方法来预测 SQL 语句执行时长。本工具有如下优点：

(1) 不需要 SQL 语句的执行计划，对数据库性能不会有任何的影响。

(2) 使用场景广泛，目前业内的很多算法局限性比较高，比如只适用于 OLTP 或者 OLAP，而 SQLdiag 使用场景广泛。

(3) 该工具容易理解，只需要简单的操作，就可以训练出自己的预测模型。

本工具的典型应用场景是对一批即将上线的 SQL 语句进行透视，提前识别风险。

11.1.3.5.1 概述

SQLdiag 是一个 SQL 语句执行时间预测工具，通过模板化方法或者深度学习方法，实现在不获取 SQL 语句执行计划的前提下，依据语句逻辑相似度与历史执行记录，预测 SQL 语句的执行时间并以此发现异常 SQL。

11.1.3.5.2 使用指导

前提条件

- 需要保证用户提供训练数据。
- 如果用户通过提供的工具收集训练数据，则需要启用 WDR 功能，涉及到的参数为 track_stmt_stat_level 和 log_min_duration_statement，具体情况见下面小结。
- 为保证预测准确率，用户提供的历史语句日志应尽可能全面并具有代表性。

SQL 流水采集方法

本工具需要用户提前准备数据，训练数据格式如下，每个样本通过换行符分隔：

```
SQL,EXECUTION_TIME
```

预测数据格式如下：

```
SQL
```

其中 SQL 表示 SQL 语句的文本，EXECUTION_TIME 表示 SQL 语句的执行时间，样例数据见 sample_data 中的 train.csv 和 predict.csv。

用户可以按照要求格式自己收集训练数据，工具也提供了脚本自动采集（load_sql_from_rd），该脚本基于 WDR 报告获取 SQL 信息，涉及到的参数有 log_min_duration_statement 和 track_stmt_stat_level：

- 其中 log_min_duration_statement 表示慢 SQL 阈值，如果为 0 则全量收集，时间单位为毫秒；
- track_stmt_stat_level 表示信息捕获的级别，建议设置为 track_stmt_stat_level='L0,L0'

参数开启后，可能占用一定的系统资源，但一般不大。持续的高并发场景可能产生 5%以内的损耗，数据库并发较低的场景，性能损耗可忽略。下述脚本存在于 sqldiag 根目录（$GAUSSHOME/bin/components/sqldiag）中。

```
使用脚本获取训练集方式：
load_sql_from_wdr.py [-h] --port PORT --start_time START_TIME
                                    --finish_time FINISH_TIME [--save_path SAVE_PATH]
例如：
     python load_sql_from_wdr.py --start_time "2021-04-25 00:00:00" --finish_time
"2021-04-26 14:00:00" --port 5432    --save_path ./data.csv
```

操作步骤

(1) 提供历史日志以供模型训练

(2) 进行训练与预测操作。

```
基于模板法的训练与预测：
    gs_dbmind component sqldiag [train, predict] -f FILE --model template --model-path
template_model_path
基于 DNN 的训练与预测：
    gs_dbmind component sqldiag [train, predict] -f FILE --model dnn --model-path
dnn_model_path
```

使用方法示例

使用提供的测试数据进行模板化训练：

```
gs_dbmind component sqldiag train -f ./sample_data/train.csv --model template
--model-path ./template
```

使用提供的测试数据进行模板化预测：

```
gs_dbmind component sqldiag predict -f ./sample_data/predict.csv --model template
--model-path ./template --predicted-file ./result/t_result
```

使用提供的测试数据进行模板化模型更新：

```
gs_dbmind component sqldiag finetune -f ./sample_data/train.csv --model template
--model-path ./template
```

使用提供的测试数据进行 DNN 训练：

```
gs_dbmind component sqldiag train -f ./sample_data/train.csv --model dnn
--model-path ./dnn_model
```

使用提供的测试数据进行 DNN 预测：

```
gs_dbmind component sqldiag predict -f ./sample_data/predict.csv --model dnn
--model-path ./dnn_model --predicted-file
```

使用提供的测试数据进行 DNN 模型更新：

```
gs_dbmind component sqldiag finetune -f ./sample_data/train.csv --model dnn
--model-path ./dnn_model
```

11.1.3.5.3 获取帮助

使用 SQLdiag 工具前，您可以通过以下指令获取帮助。

```
gs_dbmind component sqldiag --help
```

显示如下帮助信息：

```
usage:      [-h] [-f CSV_FILE] [--predicted-file PREDICTED_FILE]
                    [--model {template,dnn}] --model-path MODEL_PATH
                    [--config-file CONFIG_FILE]
                    {train,predict,finetune}
SQLdiag integrated by openGauss.
positional arguments:
  {train,predict,finetune}
                                The training mode is to perform feature extraction and
                                model training based on historical SQL statements. The
                                prediction mode is to predict the execution time of a
                                new SQL statement through the trained model.
optional arguments:
  -h, --help                    show this help message and exit
  -f CSV_FILE, --csv-file CSV_FILE
                                The data set for training or prediction. The file
                                format is CSV. If it is two columns, the format is
                                (SQL statement, duration time). If it is three
```

```
                        columns, the format is (timestamp of SQL statement
                        execution time, SQL statement, duration time).
--predicted-file PREDICTED_FILE
                        The file path to save the predicted result.
--model {template,dnn}
                        Choose the model model to use.
--model-path MODEL_PATH
                        The storage path of the model file, used to read or
                        save the model file.
--config-file CONFIG_FILE
```

11.1.3.5.4 命令参考

表 11-14 命令行参数说明

参数	参数说明	取值范围
-f	训练或预测文件位置	-
–predicted-file	预测结果存储位置	-
–model	模型选择	template, dnn
–model-path	训练模型存储位置	-

11.1.3.5.5 常见问题处理

- 训练场景失败：请检查历史日志文件路径是否正确，且文件格式符合上文规定。
- 预测场景失败：请检查模型路径是否正确。确保待预测负载文件格式正确。

11.2 DB4AI：数据库驱动 AI

DB4AI 是指利用数据库的能力驱动 AI 任务，实现数据存储、技术栈的同构。通过在数据库内集成 AI 算法，令 GBase 8c 具备数据库原生 AI 计算引擎、模型管理、AI 算子、AI 原生执行计划的能力，为用户提供普惠 AI 技术。不同于传统的 AI 建模流程，DB4AI “一站式”建模可以解决数据在各平台的反复流转问题，同时简化开发流程，并可通过数据库规划出最优执行路径，让开发者更专注于具体业务和模型的调优上，具备同类产品不具备的易用性与性能优势。

11.2.1 原生 DB4AI 引擎

GBase 8c 支持了原生 DB4AI 能力，通过引入原生 AI 算子，简化操作流程，充分利用数据库优化器、执行器的优化与执行能力，获得高性能的数据库内模型训练能力。更简化的模型训练与预测流程、更高的性能表现，让开发者在更短时间内能更专注于模型的调优与数据分析上，而避免了碎片化的技术栈与冗余的代码实现。

关键字解析

表 11-15 DB4AI 语法及关键字

名称		描述
语法	CREATE MODEL	创建模型并进行训练，同时保存模型。
	PREDICT BY	利用已有模型进行推断。
	DROP MODEL	删除模型。
关键字	TARGET	训练/推断任务的目标列名。
	FEATURES	训练/推断任务的数据特征列名。
	MODEL	训练任务的模型名称。

使用指导

(1) 本版本支持的算法概述。

当前版本的 DB4AI 新增支持算法如下：

表 11-16 支持算法

优化算法	算法
GD	logistic_regression
	linear_regression
	svm_classification
	PCA
	multiclass

Kmeans	kmeans
xgboost	xgboost_regression_logistic
	xgboost_binary_logistic
	xgboost_regression_squarederror
	xgboost_regression_gamma

(2) 模型训练语法说明。

- CREATE MODEL

使用“CREATE MODEL”语句可以进行模型的创建和训练。模型训练 SQL 语句，选用公开数据集鸢尾花数据集 iris。

- 以 multiclass 为例，训练一个模型。从 tb_iris 训练集中指定 sepal_length, sepal_width,petal_length,petal_widt 为特征列，使用 multiclass 算法，创建并保存模型 iris_classification_model。

```
postgres=# CREATE MODEL iris_classification_model USING xgboost_regression_logistic
FEATURES sepal_length, sepal_width,petal_length,petal_width TARGET target_type < 2
FROM tb_iris_1 WITH nthread=4, max_depth=8;
MODEL CREATED. PROCESSED 1
```

上述命令中：

- CREATE MODEL 语句用于模型的训练和保存。
- USING 关键字指定算法名称。
- FEATURES 用于指定训练模模型的特征，需根据训练数据表的列名添加。
- TARGET 指定模型的训练目标，它可以是训练所需数据表的列名，也可以是一个表达式，例如: price > 10000。
- WITH 用于指定训练模型时的超参数。当超参未被用户进行设置的时候，框架会使用默认数值。

针对不同的算子，框架支持不同的超参组合：

表 11-17 算子支持的超参

算子	超参

GD (logistic_regression、linear_regression、svm_classification)	optimizer(char); verbose(bool); max_iterations(int); max_seconds(double); batch_size(int); learning_rate(double); decay(double); tolerance(double) 其中，SVM 限定超参 lambda(double)
Kmeans	max_iterations(int); num_centroids(int); tolerance(double); batch_size(int); num_features(int); distance_function(char); seeding_function(char); verbose(int);seed(int)
GD(pca)	batch_size(int);max_iterations(int);max_seconds(int);tolerance(float8);verbose(bool);number_components(int);seed(int)
GD(multiclass)	classifier(char) 注意：multiclass 的其他超参种类取决于选择的分类器中类
xgboost_regression_logistic、xgboost_binary_logistic、xgboost_regression_squarederror、xgboost_regression_gamma	batch_size(int);booster(char);tree_method(char);eval_metric(char*);seed(int);nthread(int);max_depth(int);gamma(float8);eta(float8);min_child_weight(int);verbosity(int)

当前各个超参数设置的默认值和取值范围如下：

表 11-18 超参的默认值以及取值范围

算子	超参(默认值)	取值范围	超参描述
GD: logistic_regression、linear_regression、svm_classification、pca	optimizer = gd（梯度下降法）	gd/ngd（自然梯度下降）	优化器
	verbose = false	T/F	日志显示
	max_iterations = 100	(0, 10000]	最大迭

			代次数
	max_seconds = 0 (不对运行时长设限制)	[0,INT_MAX_VALUE]	运行时长
	batch_size = 1000	(0, 1048575]	一次训练所选取的样本数
	learning_rate = 0.8	(0, DOUBLE_MAX_VALUE]	学习率
	decay = 0.95	(0, DOUBLE_MAX_VALUE]	权值衰减率
	tolerance = 0.0005	(0, DOUBLE_MAX_VALUE]	公差
	seed = 0 （对 seed 取随机值）	[0, INT_MAX_VALUE]	种子
	just for linear、SVM : kernel = “linear”	linear/gaussian/polynomial	核函数
	just for linear、SVM : components = MAX(2*features, 128)	[0, INT_MAX_VALUE]	高维空间维数
	just for linear、SVM : gamma = 0.5	(0, DOUBLE_MAX_VALUE]	gaussian 核函数参数
	just for linear、SVM : degree = 2	[2, 9]	polynomial 核函数参数
	just for linear、SVM : coef0 = 1.0	[0, DOUBLE_MAX_VALUE]	polynomial 核函数的参数
	just for SVM : lambda = 0.01	(0, DOUBLE_MAX_VALUE)	正则化参数

	just for pca : number_components	（0，INT_MAX_VALUE]	降维的目标维度
GD: multiclass	classifier=“svm_classification”	svm_classification\logistic_regression	多分类任务的分类器
Kmeans	max_iterations = 10	[1, 10000]	最大迭代次数
	num_centroids = 10	[1, 1000000]	簇的数目
	tolerance = 0.00001	(0,1]	中心点误差
	batch_size = 10	[1,1048575]	一次训练所选取的样本数
	num_features = 2	[1, INT_MAX_VALUE]	输入样本特征数
	distance_function = “L2_Squared”	L1\L2\L2_Squared\Linf	正则化方法
	seeding_function = “Random++”	“Random++”\“KMeans||”	初始化种子点方法
	verbose = 0U	{ 0, 1, 2 }	冗长模式
	seed = 0U	[0, INT_MAX_VALUE]	种子
xgboost: xgboost_regression_logistic、 xgboost_binary_logistic、	n_iter=10	(0, 10000]	迭代次数
	batch_size=10000	(0, 1048575]	一次训练所选取的样

xgboost_regression_gamma、xgboost_regression_squarederror			本数
	booster=“gbtree”	gbtree\gblinear\dart	booster 种类
	tree_method=“auto”	auto\exact\approx\hist\gpu_hist 注意：gpu_hist 参数需要相应的库 GPU 版本，否则 DB4AI 平台不支持该值。	树构建算法
	eval_metric=“rmse”	rmse\rmsle\map\mae\auc\aucpr	验证数据的评估指标
	seed=0	[0, 100]	种子
	nthread=1	(0, MAX_MEMORY_LIMIT]	并发量
	max_depth=5	(0, MAX_MEMORY_LIMIT]	树的最大深度，该超参仅对树型 booster 生效。
	gamma=0.0	[0, 1]	叶节点上进行进一步分区所需的最小损失减少
	eta=0.3	[0, 1]	更新中使用的步长收缩，以防止过拟合

	min_child_weight=1	[0, INT_MAX_VALUE]	孩子节点中所需的实例权重的最小总和
	verbosity=1	0 (silent)\1 (warning)\2 (info)\3 (debug)	打印信息的详细程度
MAX_MEMORY_LIMIT = 最大内存加载的元组数量			
GS_MAX_COLS = 数据库单表最大属性数量			

- 模型保存成功，则返回创建成功信息：

```
MODEL CREATED. PROCESSED x
```

(3) 查看模型信息。

当训练完成后模型会被存储到系统表 gs_model_warehouse 中。系统表 gs_model_warehouse 可以查看到关于模型本身和训练过程的相关信息。

关于模型的详细描述信息以二进制的形式存储在系统表中，用户可用过使用函数 gs_explain_model 完成对模型的查看，语句如下：

```
postgres=# select * from gs_explain_model("iris_classification_model");
 DB4AI MODEL
-------------------------------------------------------------
 Name: iris_classification_model
 Algorithm: xgboost_regression_logistic
 Query: CREATE MODEL iris_classification_model
 USING xgboost_regression_logistic
 FEATURES sepal_length, sepal_width,petal_length,petal_width
 TARGET target_type < 2
 FROM tb_iris_1
 WITH nthread=4, max_depth=8;
 Return type: Float64
 Pre-processing time: 0.000000
 Execution time: 0.001443
 Processed tuples: 78
 Discarded tuples: 0
 n_iter: 10
```

```
batch_size: 10000
max_depth: 8
min_child_weight: 1
gamma: 0.0000000000
eta: 0.3000000000
nthread: 4
verbosity: 1
seed: 0
booster: gbtree
tree_method: auto
eval_metric: rmse
rmse: 0.2648450136
model size: 4613
```

(4) 利用已存在的模型做推断任务。

使用“SELECT”和“PREDICT BY”关键字利用已有模型完成推断任务。

查询语法：SELECT…PREDICT BY…(FEATURES…)…FROM…;

```
postgres=# SELECT id, PREDICT BY iris_classification (FEATURES
sepal_length,sepal_width,petal_length,petal_width) as "PREDICT" FROM tb_iris limit 3;

id   | PREDICT
-----+---------
  84 |       2
  85 |       0
  86 |       0
(3 rows)
```

针对相同的推断任务，同一个模型的结果是大致稳定的。且基于相同的超参数和训练集训练的模型也具有稳定性，同时 AI 模型训练存在随机成分（每个 batch 的数据分布、随机梯度下降），所以不同的模型间的计算表现、结果允许存在小的差别。

(5) 查看执行计划。

使用 explain 语句可对“CREATE MODEL”和“PREDICT BY”的模型训练或预测过程中的执行计划进行分析。Explain 关键字后可直接拼接 CREATE MODEL/ PREDICT BY 语句（子句），也可接可选的参数，支持的参数如下：

表 11-19 EXPLAIN 支持的参数

参数名	描述

ANALYZE	布尔型变量，追加运行时间、循环次数等描述信息
VERBOSE	布尔型变量，控制训练的运行信息是否输出到客户端
COSTS	布尔型变量
CPU	布尔型变量
DETAIL	布尔型变量，不可用。
NODES	布尔型变量，不可用
NUM_NODES	布尔型变量，不可用
BUFFERS	布尔型变量
TIMING	布尔型变量
PLAN	布尔型变量
FORMAT	可选格式类型：TEXT / XML / JSON / YAML

示例：

```
postgres=# Explain CREATE MODEL patient_logisitic_regression USING logistic_regression
FEATURES second_attack, treatment TARGET trait_anxiety > 50 FROM patients WITH
batch_size=10, learning_rate = 0.05;
                                    QUERY PLAN
-------------------------------------------------------------------------
 Train Model - logistic_regression    (cost=0.00..0.00 rows=0 width=0)
   ->  Materialize    (cost=0.00..41.08 rows=1776 width=12)
         ->  Seq Scan on patients    (cost=0.00..32.20 rows=1776 width=12)
(3 rows)
```

(6) 异常场景。

- 训练阶段。
  - 场景一：当超参数的设置超出取值范围，模型训练失败，返回 ERROR，并提示错误，例如：

```
postgres=# CREATE MODEL patient_linear_regression USING linear_regression FEATURES
second_attack,treatment TARGET trait_anxiety    FROM patients WITH optimizer='aa';
ERROR:    Invalid hyperparameter value for optimizer. Valid values are: gd, ngd.
```

  - 场景二：当模型名称已存在，模型保存失败，返回 ERROR，并提示错误

原因，例如：

```
postgres=# CREATE MODEL patient_linear_regression USING linear_regression FEATURES
second_attack,treatment TARGET trait_anxiety    FROM patients;
ERROR:    The model name "patient_linear_regression" already exists in gs_model_warehouse.
```

- 场景三：FEATURE 或者 TARGETS 列是*，返回 ERROR，并提示错误原因，例如：

```
postgres=# CREATE MODEL patient_linear_regression USING linear_regression FEATURES
*    TARGET trait_anxiety    FROM patients;
ERROR:    FEATURES clause cannot be *
--------------------------------------------------------------------------------------------------------------
----
postgres=# CREATE MODEL patient_linear_regression USING linear_regression FEATURES
second_attack,treatment TARGET *    FROM patients;
ERROR:    TARGET clause cannot be *
```

- 场景四：对于无监督学习方法使用 TARGET 关键字，或者在监督学习方法中不适用 TARGET 关键字，均会返回 ERROR，并提示错误原因，例如：

```
postgres=# CREATE MODEL patient_linear_regression USING linear_regression FEATURES
second_attack,treatment FROM patients;
ERROR:    Supervised ML algorithms require TARGET clause
--------------------------------------------------------------------------------------------------------------
----------
CREATE MODEL patient_linear_regression USING linear_regression TARGET trait_anxiety
FROM patients;
ERROR:    Supervised ML algorithms require FEATURES clause
```

- 场景五：当进行分类任务时 TARGET 列的分类只有 1 种情况，会返回 ERROR，并提示错误原因，例如：

```
postgres=# CREATE MODEL ecoli_svmc USING multiclass FEATURES f1, f2, f3, f4, f5, f6,
f7 TARGET cat FROM (SELECT * FROM db4ai_ecoli WHERE cat='cp');
ERROR:    At least two categories are needed
```

- 场景六：DB4AI 在训练过程中会过滤掉含有空值的数据，当参与训练的模型数据为空的时候，会返回 ERROR，并提示错误原因，例如：

```
postgres=# create model iris_classification_model using xgboost_regression_logistic features
message_regular target error_level from error_code;
ERROR:    Training data is empty, please check the input data.
```

- 场景七：DB4AI 的算法对于支持的数据类型是有限制的。当数据类型不

在支持白名单中，会返回 ERROR，并提示非法的 oid，可通过 pg_type 查看 OID 确定非法的数据类型，例如：

```
postgres=# CREATE MODEL ecoli_svmc USING multiclass FEATURES f1, f2, f3, f4, f5, f6, f7, cat TARGET cat FROM db4ai_ecoli ;
ERROR:  Oid type 1043 not yet supported
```

- 场景八：当 GUC 参数 statement_timeout 设置了时长，训练超时执行的语句将被终止：执行 CREATE MODEL 语句。训练集的大小、训练轮数(iteration)、提前终止条件(tolerance、max_seconds)、并行线程数(nthread)等参数都会影响训练时长。当时长超过数据库限制，语句被终止模型训练失败。

- 模型解析。
  - 场景九：当模型名在系统表中查找不到，数据库会报 ERROR，例如：

```
postgres=# select gs_explain_model("ecoli_svmc");
ERROR:  column "ecoli_svmc" does not exist
```

- 推断阶段。
  - 场景十：当模型名在系统表中查找不到，数据库会报 ERROR，例如：

```
postgres=# select id, PREDICT BY patient_logistic_regression (FEATURES second_attack,treatment) FROM patients;
ERROR:  There is no model called "patient_logistic_regression".
```

  - 场景十一：当做推断任务 FEATURES 的数据维度和数据类型与训练集存在不一致，将报 ERROR，并提示错误原因，例如：

```
postgres=# select id, PREDICT BY patient_linear_regression (FEATURES second_attack) FROM patients;
ERROR:  Invalid number of features for prediction, provided 1, expected 2
CONTEXT:  referenced column: patient_linear_regression_pred
-----------------------------------------------------------------------------------------------------------------
postgres=# select id, PREDICT BY patient_linear_regression (FEATURES 1,second_attack,treatment) FROM patients;
ERROR:  Invalid number of features for prediction, provided 3, expected 2
CONTEXT:  referenced column: patient_linear_regression_pre
```

说明

DB4AI 特性需要读取数据参与计算，不适用于密态数据库等情况。

11.2.2 全流程 AI

传统的 AI 任务往往具有多个流程，如数据的收集过程包括数据的采集、数据清洗、数据存储等，在算法的训练过程中又包括数据的预处理、训练、模型的保存与管理等。其中，对于模型的训练过程，又包括超参数的调优过程。诸如此类机器学习模型生命周期的全过程，可大量集成于数据库内部。在距离数据存储侧最近处进行模型的训练、管理、优化等流程，在数据库端提供 SQL 语句式的开箱即用的 AI 全声明周期管理的功能，称之为全流程 AI.

GBase 8c 实现了部分全流程 AI 的功能，将在本章节中详细展开。

11.2.2.1 PLPython Fenced 模式

在 fenced 模式中添加 plpython 非安全语言。在数据库编译时需要将 python 集成进数据库中，在 configure 阶段加入--with-python 选项。同时也可指定安装 plpython 的 python 路径，添加选项--with-includes='/python-dir=path'。

在启动数据库之前配置 GUC 参数 unix_socket_directory ，指定 unix_socket 进程间通讯的文件地址。用户需要提前在 user-set-dir-path 下创建文件夹，并将文件夹权限修改为可读可写可执行状态。

```
unix_socket_directory = '/user-set-dir-path'
```

配置完成，启动数据库。

将 plpython 加入数据库编译，并设置好 GUC 参数 unix_socket_directory 后，在启动数据库的过程中，自动创建 fenced-Master 进程。在数据库不进行 python 编译的情况下， fenced 模式需要手动拉起 master 进程，在 GUC 参数设置完成后，输入创建 master 进程命令。

启动 fenced-Master 进程，命令为：

```
gaussdb --fenced -k /user-set-dir-path -D /user-set-dir-path &
```

完成 fence 模式配置，针对 plpython-fenced UDF 数据库将在 fenced-worker 进程中执行 UDF 计算。

使用指导

- 创建 extension
    - 当编译的 plpython 为 python2 时：

```
postgres=# create extension plpythonu;
CREATE EXTENSION
```

- 当编译的 plpython 为 python3 时：

```
postgres=# create extension plpython3u;
CREATE EXTENSION
```

下面示例是以 python2 为例。

- 创建 plpython-fenced UDF
- 查看 UDF 信息

```
postgres=# select * from pg_proc where proname='pymax';
postgres=# create or replace function pymax(a int, b int)
gsql-# returns INT
gsql-# language plpythonu fenced
gsql-# as $$
gsql$# import numpy
gsql$# if a > b:
gsql$#    return a;
gsql$# else:
gsql$#    return b;
gsql$# $$;
CREATE FUNCTION
```

- 运行 UDF
  - 创建一个数据表：

```
postgres=# create table temp (a int ,b int) ;
CREATE TABLE
postgres=# insert into temp values (1,2),(2,3),(3,4),(4,5),(5,6);
INSERT 0 5
```

  - 运行 UDF：

```
postgres=# select pymax(a,b) from temp;
pymax
-------
2
3
4
5
6
(5 rows)
```

11.2.2.2 DB4AI-Snapshots 数据版本管理

DB4AI-Snapshots 是 DB4AI 模块用于管理数据集版本的功能。通过 DB4ai-Snapshots 组件，开发者可以简单、快速地进行特征筛选、类型转换等数据预处理操作，同时还可以像 git 一样对训练数据集进行版本控制。数据表快照创建成功后可以像视图一样进行使用，但是一经发布后，数据表快照便固化为不可变的静态数据，如需修改该数据表快照的内容，需要创建一个版本号不同的新数据表快照。

DB4AI-Snapshots 的生命周期

DB4AI-Snapshots 的状态包括 published、archived 以及 purged。其中，published 可以用于标记该 DB4AI-Snapshots 已经发布，可以进行使用。archived 表示当前 DB4AI-Snapshots 处于“存档期”，一般不进行新模型的训练，而是利用旧数据对新的模型进行验证。purged 则是该 DB4AI-Snapshots 已经被删除的状态，在数据库系统中无法再检索到。

需要注意的是快照管理功能是为了给用户提供统一的训练数据，不同团队成员可以使用给定的训练数据来重新训练机器学习模型，方便用户间协同。为此私有用户和三权分立状态(enableSeparationOfDuty=ON)等涉及不支持用户数据转写等情况将不支持 Snapshot 特性。

用户可以通过“CREATE SNAPSHOT”语句创建数据表快照，创建好的快照默认即为 published 状态。可以采用两种模式创建数据表快照，即为 MSS 以及 CSS 模式，它们可以通过 GUC 参数 db4ai_snapshot_mode 进行配置。对于 MSS 模式，它是采用物化算法进行实现的，存储了原始数据集的数据实体；CSS 则是基于相对计算算法实现的，存储的是数据的增量信息。数据表快照的元信息存储在 DB4AI 的系统目录中。可以通过 db4ai.snapshot 系统表查看到。

可以通过“ARCHIVE SNAPSHOT”语句将某一个数据表快照标记为 archived 状态，可以通过“PUBLISH SNAPSHOT”语句将其再度标记为 published 状态。标记数据表快照的状态，是为了帮助数据科学家进行团队合作使用的。

当一个数据表快照已经丧失存在价值时，可以通过“PURGE SNAPSHOT”语句删除它，以便永久删除其数据并恢复存储空间。

DB4AI-Snapshots 使用指导

(1) 创建表以及插入表数据。

数据库内存在已有的数据表，可根据该已有的数据表创建对应的数据表快照。为了后续演示，在此处新建一个名为 t1 的数据表，并向其中插入测试数据。

```
create table t1 (id int, name varchar);
insert into t1 values (1, 'zhangsan');
insert into t1 values (2, 'lisi');
insert into t1 values (3, 'wangwu');
insert into t1 values (4, 'lisa');
insert into t1 values (5, 'jack');
```

通过 SQL 语句，查询搭配数据表内容。

```
SELECT * FROM t1;
id |     name
----+----------
  1 | zhangsan
  2 | lisi
  3 | wangwu
  4 | lisa
  5 | jack
(5 rows)
```

(2) 使用 DB4AI-Snapshots。

- 创建 DB4AI-Snapshots
  - 示例 1：CREATE SNAPSHOT…AS

示例如下，其中，默认版本分隔符为 “@”，默认子版本分割符为 “.”，该分割符可以分别通过 GUC 参数 db4ai_snapshot_version_delimiter 以及 db4ai_snapshot_version_separator 进行设置。

```
create snapshot s1@1.0 comment is 'first version' as select * from t1;
schema |    name
--------+--------
 public | s1@1.0
(1 row)
```

上述结果提示已经创建了数据表 s1 的快照，版本号为 1.0。创建好后的数据表快照可以像使用一般视图一样进行查询，但不支持通过 “INSERT INTO” 语句进行更新。例如下面几种语句都可以查询到数据表快照 s1 的对应版本 1.0 的内容：

```
SELECT * FROM s1@1.0;
SELECT * FROM public.s1@1.0;
SELECT * FROM public . s1 @ 1.0;
id |     name
----+----------
  1 | zhangsan
```

```
  2 | lisi
  3 | wangwu
  4 | lisa
  5 | jack
(5 rows)
```

可以通过下列 SQL 语句修改数据表 t1 的内容：

```
UPDATE t1 SET name = 'tom' where id = 4;
insert into t1 values (6, 'john');
insert into t1 values (7, 'tim');
```

再检索数据表 t1 的内容时，发现虽然数据表 t1 的内容已经发生变化，但是数据表快照 s1@1.0 版本的查询结果并未发生变化。由于数据表 t1 的数据已经发生了改变，如果将当前数据表的内容作为版本 2.0，则可创建快照 s1@2.0,创建的 SQL 语句如下：

```
create snapshot s1@2.0 as select * from t1;
```

通过上述例子，我们可以发现，数据表快照可以固化数据表的内容，避免中途对数据的改动造成机器学习模型训练时的不稳定，同时可以避免多用户同时访问、修改同一个表时造成的锁冲突。

- 示例 2：CREATE SNAPSHOT…FROM

SQL 语句可以对一个已经创建好的数据表快照进行继承，利用在此基础上进行的数据修改产生一个新的数据表快照。例如：

```
create snapshot s1@3.0 from @1.0 comment is 'inherits from @1.0' using (INSERT VALUES(6,
'john'), (7, 'tim'); DELETE WHERE id = 1);
schema |    name
--------+--------
 public | s1@3.0
(1 row)
```

其中，“@”为数据表快照的版本分隔符，from 子句后加上已存在的数据表快照，用法为“@”+版本号，USING 关键字后加入可选的几个操作关键字（INSERT …/UPDATE …/DELETE …/ALTER …），其中“INSERT INTO”以及“DELETE FROM”语句中的“INTO”、“FROM”等与数据表快照名字相关联的子句可以省略，具体可以参考 AI 特性函数。

示例中，基于前述 s1@1.0 快照，插入 2 条数据，删除 1 条新的数据，新生成的快照 s1@3.0，检索该 s1@3.0：

```
SELECT * FROM s1@3.0;
id |     name
```

```
----+----------
   2 | lisi
   3 | wangwu
   4 | lisa
   5 | jack
   6 | john
   7 | tim
(7 rows)
```

- 删除数据表快照 SNAPSHOT

```
purge snapshot s1@3.0;
schema |    name
--------+--------
 public | s1@3.0
(1 row)
```

此时，已经无法再从 s1@3.0 中检索到数据了，同时该数据表快照在 db4ai.snapshot 视图中的记录也会被清除。删除该版本的数据表快照不会影响其他版本的数据表快照。

- 从数据表快照中采样

示例：从 snapshot s1 中抽取数据，使用 0.5 抽样率。

```
sample snapshot s1@2.0 stratify by name as nick at ratio .5;
schema |       name
--------+------------
 public | s1nick@2.0
(1 row)
```

可以利用该功能创建训练集与测试集，例如：

```
SAMPLE SNAPSHOT s1@2.0    STRATIFY BY name AS _test AT RATIO .2, AS _train AT
RATIO .8 COMMENT IS 'training';
schema |          name
--------+----------------
 public | s1_test@2.0
 public | s1_train@2.0
(2 rows)
```

- 发布数据表快照

采用下述 SQL 语句将数据表快照 s1@2.0 标记为 published 状态：

```
publish snapshot s1@2.0;
schema |    name
```

```
--------+--------
 public | s1@2.0
(1 row)
```

- 存档数据表快照

  采用下述语句可以将数据表快照标记为 archived 状态：

```
archive snapshot s1@2.0;
schema |    name
--------+--------
 public | s1@2.0
(1 row)
```

  可以通过 db4ai-snapshots 提供的视图查看当前数据表快照的状态以及其他信息：

```
select * from db4ai.snapshot;
id | parent_id | matrix_id | root_id | schema |       name       | owner    |             commands
| comment | published | archived |                created | row_count
----+-----------+-----------+---------+--------+------------+--------+-----------------------------------
--+---------+-----------+----------+---------------------------+-----------
   1 |           |           |       1 | public | s1@2.0         | gbase | {"select *","from t1
where id > 3",NULL} |         | t         | f         | 2021-04-17 09:24:11.139868 |
2
   2 |         1 |           |       1 | public | s1nick@2.0 | gbase | {"SAMPLE nick .5
{name}"}                    |         | f         | f         | 2021-04-17 10:02:31.73923
|         0
```

(3) 异常场景

  数据表或 db4ai-snapshots 不存在时。

```
purge snapshot s1nick@2.0;
publish snapshot s1nick@2.0;
---------
ERROR:  snapshot public."s1nick@2.0" does not exist
CONTEXT:  PL/pgSQL function db4ai.publish_snapshot(name,name) line 11 at assignment

archive snapshot s1nick@2.0;
----------
ERROR:  snapshot public."s1nick@2.0" does not exist
CONTEXT:  PL/pgSQL function db4ai.archive_snapshot(name,name) line 11 at assignment
```

  删除 snapshot 时，有依赖该快照的其他 snapshot，需先确保删除对本快照所依赖的其他快照。

```
purge snapshot s1@1.0;
ERROR:    cannot purge root snapshot 'public."s1@1.0"' having dependent snapshots
HINT:    purge all dependent snapshots first
CONTEXT:    referenced column: purge_snapshot_internal
SQL statement "SELECT db4ai.purge_snapshot_internal(i_schema, i_name)"
PL/pgSQL function db4ai.purge_snapshot(name,name) line 71 at PERFORM
```

(4) 相关 GUC 参数

- db4ai_snapshot_mode：

Snapshot 有 2 种模式：MSS（物化模式，存储数据实体）和 CSS（计算模式，存储增量信息）。Snapshot 可在 MSS 和 CSS 之间切换快照模式，默认是 MSS 模式。

- db4ai_snapshot_version_delimiter：

该参数为数据表快照版本分隔符。“@”为数据表快照的默认版本分隔符。

- db4ai_snapshot_version_separator

该参数为数据表快照子版本分隔符。“.”为数据表快照的默认版本分隔符。

(5) DB4AI Schema 下的数据表快照详情 db4ai.snapshot。

```
postgres=# \d db4ai.snapshot
                                Table "db4ai.snapshot"
   Column     |                  Type                  |              Modifiers
-----------+-----------------------------+---------------------------
 id            | bigint                                  |
 parent_id | bigint                                  |
 matrix_id | bigint                                  |
 root_id     | bigint                                  |
 schema      | name                                    | not null
 name         | name                                    | not null
 owner        | name                                    | not null
 commands    | text[]                                  | not null
 comment     | text                                    |
 published | boolean                                 | not null default false
 archived    | boolean                                 | not null default false
 created     | timestamp without time zone | default pg_systimestamp()
 row_count | bigint                                  | not null
Indexes:
     "snapshot_pkey" PRIMARY KEY, btree (schema, name) TABLESPACE pg_default
     "snapshot_id_key" UNIQUE CONSTRAINT, btree (id) TABLESPACE pg_default
```

说明

命名空间 DB4AI 是本功能的私有域，不支持在 DB4AI 的命令空间下创建函数索引（functional index）。

11.3 AI in DB：数据库内 AI 功能

智能 Explain: SQL 语句查询时间预测

11.3.1 概述

本功能名为 Predictor，是基于机器学习且具有在线学习能力的查询时间预测工具。通过不断学习数据库内收集的历史执行信息，实现计划的执行时间预测功能。

本特性需要拉起 python 进程 AIEngine，用于模型的训练和推理。

该功能所在目录为 $GAUSSHOME/bin/dbmind/components/predictor. 由于该模块中某些功能涉及到相对复杂的搭建，因此，需要用户切换到该目录中寻找对应文件，并按照本章说明进行部署。

11.3.2 环境部署

前提条件

需要保证 GBase 8c 处于正常状态，用户通过身份验证成功登录数据库；用户执行的 SQL 语法正确无报错，且不会导致数据库异常等；历史性能数据窗口内 GBase 8c 并发量稳定，表结构、表数量不变，数据量无突变，涉及查询性能的 guc 参数不变；进行预测时，需要保证模型已训练并收敛；AiEngine 运行环境稳定。

请求样例

AiEngine 进程与内核进程使用 https 发送请求进行通信，请求样例如下：

```
curl -X POST -d '{"modelName":"modelname"}' -H 'Content-Type: application/json'
'https://IP-address:port/request-API'
```

表 11-20 AI-Engine 对外接口

Request-API	功能
/check	检查模型是否被正常拉起

/configure	设置模型参数
/train	模型训练
/track_process	查看模型训练日志
/setup	加载历史模型
/predict	模型预测

证书生成

使用此功能前需使用 openssl 工具生成通信双方认证所需的证书，保证通信安全。

(1) 搭建证书生成环境，证书文件保存路径为$GAUSSHOME/CA。

拷贝证书生成脚本及相关文件

```
cp path_to_predictor/install/ssl.sh $GAUSSHOME/
cp path_to_predictor/install/ca_ext.txt $GAUSSHOME/
```

copy 配置文件 openssl.cnf 到$GAUSSHOME 路径下

```
cp $GAUSSHOME/share/om/openssl.cnf    $GAUSSHOME/
```

修改 openssl.conf 配置参数

```
dir    = $GAUSSHOME/CA/demoCA
default_md          = sha256
```

至此证书生成环境准备完成

(2) 生成证书及密钥

```
cd $GAUSSHOME
sh ssl.sh
```

根据提示设置密码，假如为 Test@123：

密码要求至少 3 种不同类型字符，长度至少为 8 位

```
Please enter your password:
```

根据提示输入选项：

```
Certificate Details:
        Serial Number: 1 (0x1)
        Validity
            Not Before: May 15 08:32:44 2020 GMT
            Not After : May 15 08:32:44 2021 GMT
```

```
        Subject:
            countryName               = CN
            stateOrProvinceName       = SZ
            organizationName          = HW
            organizationalUnitName    = GS
            commonName                = CA
        X509v3 extensions:
            X509v3 Basic Constraints:
                CA:TRUE
Certificate is to be certified until May 15 08:32:44 2021 GMT (365 days)
Sign the certificate? [y/n]:y
1 out of 1 certificate requests certified, commit? [y/n]y
```

输入拉起 AIEngine 的 IP 地址，如 IP 为 127.0.0.1：

```
Please enter your aiEngine IP: 127.0.0.1
```

根据提示输入选项：

```
Certificate Details:
        Serial Number: 2 (0x2)
        Validity
            Not Before: May 15 08:38:07 2020 GMT
            Not After : May 13 08:38:07 2030 GMT
        Subject:
            countryName               = CN
            stateOrProvinceName       = SZ
            organizationName          = HW
            organizationalUnitName    = GS
            commonName                = 127.0.0.1
        X509v3 extensions:
            X509v3 Basic Constraints:
                CA:FALSE
Certificate is to be certified until May 13 08:38:07 2030 GMT (3650 days)
Sign the certificate? [y/n]:y
1 out of 1 certificate requests certified, commit? [y/n]y
```

输入启动 IP 地址，如 IP 为 127.0.0.1：

```
Please enter your gaussdb IP: 127.0.0.1
```

根据提示输入选项：

```
Certificate Details:
        Serial Number: 3 (0x3)
```

```
        Validity
            Not Before: May 15 08:41:46 2020 GMT
            Not After : May 13 08:41:46 2030 GMT
        Subject:
            countryName                 = CN
            stateOrProvinceName         = SZ
            organizationName            = HW
            organizationalUnitName      = GS
            commonName                  = 127.0.0.1
        X509v3 extensions:
            X509v3 Basic Constraints:
                CA:FALSE
Certificate is to be certified until May 13 08:41:46 2030 GMT (3650 days)
Sign the certificate? [y/n]:y
1 out of 1 certificate requests certified, commit? [y/n]y
```

至此，相关证书及密钥已生成，（$GAUSSHOME/CA）内容如下：

```
CA
|-- ca_ext.txt
|-- client.crt
|-- client.key
|-- client.key.cipher
|-- client.key.rand
|-- client.req
|-- demoCA
|   |-- cacert.pem
|   |-- careq.pem
|   |-- index.txt
|   |-- index.txt.attr
|   |-- index.txt.attr.old
|   |-- index.txt.old
|   |-- newcerts
|   |   |-- 01.pem
|   |   |-- 02.pem
|   |   `-- 03.pem
|   |-- private
|   |   `-- cakey.pem
|   |-- serial
|   `-- serial.old
|-- openssl.cnf
|-- server.crt
|-- server.key
|-- server.key.cipher
|-- server.key.rand
`-- server.req
```

环境准备

将工具代码文件夹拷贝至目标环境

假设安装路径为$INSTALL_FOLDER，目标环境路径为/home/ai_user：

```
scp -r $INSTALL_FOLDER/bin/dbmind/predictor  ai_user@127.0.0.1:path_to_Predictor
```

拷贝 CA 证书文件夹至 aiEngine 环境中某路径下：

```
cp -r $GAUSSHOME/CA  ai_user@127.0.0.1:path_to_CA
```

安装 predictor/install/requirements(-gpu).txt 工具（该功能比较特殊，与其他 AI 功能不同，需要独立安装依赖）：

```
有 GPU：pip install -r requirements-gpu.txt
无 GPU：pip install -r requirements.txt
```

拉起 AiEngine

切换至 aiEngine 环境（即拷贝 predictor 的目标环境 ai_user）：

设置 predictor/python/settings.py 中的相关参数：

```
DEFAULT_FLASK_SERVER_HOST = '127.0.0.1'  （aiEngine 运行 IP 地址）
DEFAULT_FLASK_SERVER_PORT = '5000'  （aiEngine 运行端口号）
PATH_SSL = "path_to_CA" (CA 文件夹路径)
```

运行 aiEngine 启动脚本：

```
python path_to_Predictor/python/run.py
```

此时，aiEngine 即在相应端口保持拉起状态，等待内核侧时间预测功能的请求指令。

至此，aiEngine 工具部署完成。从内核中发起执行时间预测功能指令步骤，请参考使用指导。

11.3.3 使用指导

数据收集

(1) 打开数据收集。

① 设置 ActiveSQL operator 信息相关参数:

```
enable_resource_track=on
resource_track_level=operator
enable_resource_record=on
resource_track_cost=10（默认值为 100000）
```

说明

resource_track_cost 需设置为小于需要收集的查询总代价，满足条件的信息才能被收集。

Cgroups 功能正常加载。

② 信息收集：

执行业务查询语句。

查看实时收集数据：

```
select * from gs_wlm_plan_operator_history;
```

预期：满足 resource_track_duration 和 resource_track_cost 的作业被全量收集。

(2) 关闭数据收集。

① 设置 ActiveSQL operator 信息相关参数：

```
enable_resource_track=off 或
resource_track_level=none 或
resource_track_level=query
```

② 执行业务查询语句。

等待三分钟之后查看当前节点上的数据：

```
select * from gs_wlm_plan_operator_info;
```

预期：所查表和视图无新增数据。

(3) 数据持久化保存。

① 设置 ActiveSQL operator 信息相关参数：

```
enable_resource_track=on
resource_track_level=operator
enable_resource_record=on
resource_track_duration=0（默认值为 60s）
resource_track_cost=10（默认值为 100000）
```

说明

resource_track_cost 需设置为小于需要收集的查询总代价，满足条件的信息才能被收集。

Cgroups 功能正常加载。

② 执行业务查询语句。

等待三分钟之后查看当前节点上的数据：

```
select * from gs_wlm_plan_operator_info;
```

预期：满足 resource_track_duration 和 resource_track_cost 的作业被全量收集。

模型管理（系统管理员用户）

说明

模型管理操作需要在数据库正常的状态下进行。

(1) 新增模型：

```
INSERT INTO gs_opt_model values('……');
```

示例：

```
INSERT INTO gs_opt_model values('rlstm', 'model_name', 'datname', '127.0.0.1', 5000, 2000, 1, -1, 64, 512, 0 , false, false, '{S, T}', '{0,0}', '{0,0}', 'Text');
```

说明

具体模型参数设置请参考 GS_OPT_MODEL。

目前 “template_name” 列只支持 “rlstm”。

“datname” 列请和用于模型使用和训练的数据库保持一致，否则无法使用。

“model_name” 一列需要满足 unique 约束。

其他参数设置见产品文档最佳实践部分。

(2) 修改模型参数：

```
UPDATE gs_opt_model SET <attribute> = <value> WHERE model_name = <target_model_name>;
```

(3) 删除模型：

```
DELETE FROM gs_opt_model WHERE model_name = <target_model_name>;
```

(4) 查询现有模型及其状态：

```
SELECT * FROM gs_opt_model;
```

模型训练（系统管理员用户）

(1) 配置/添加模型训练参数：参考模型管理（系统管理员用户）进行模型添加、模型参数修改，来指定训练参数。

模型添加：

```
INSERT INTO gs_opt_model values('rlstm', 'default', 'postgres', '127.0.0.1', 5000, 2000, 1, -1, 64, 512, 0 , false, false, '{S, T}', '{0,0}', '{0,0}', 'Text');
```

训练参数更新：

```
UPDATE gs_opt_model SET <attribute> = <value> WHERE model_name =
<target_model_name>;
```

(2) 前提条件为数据库状态正常且历史数据正常收集：

```
删除原有 encoding 数据：
DELETE FROM gs_wlm_plan_encoding_table;
```

进行数据编码，需要指定数据库名：

```
SELECT gather_encoding_info('postgres');
```

开始训练：

```
SELECT model_train_opt('rlstm', 'default');
```

(3) 获取 AI Engine 侧模型训练日志相对路径：

```
SELECT * FROM track_model_train_opt('rlstm', 'default');
```

模型预测

说明

模型预测功能需在数据库状态正常、指定模型已被训练且收敛的条件下进行。

目前，模型训练参数的标签设置中需要包含“S”标签，explain 中才可显示“p-time”预测值。

```
例：INSERT INTO gs_opt_model values('rlstm', 'default', 'postgres', '127.0.0.1', 5000, 1000, 1,
-1, 50, 500, 0 , false, false, '{S, T}', '{0,0}', '{0,0}', 'Text');
```

调用 explain 接口：

```
explain (analyze on, predictor <model_name>)
SELECT ...
```

预期结果：

```
例：Row Adapter  (cost=110481.35..110481.35 rows=100 p-time=99..182 width=100) (actual
time=375.158..375.160 rows=2 loops=1)
```

其中，“p-time”列为标签预测值。

其他功能

(1) 检查 AiEngine 是否可连接:

```
postgres=# select check_engine_status('aiEngine-ip-address',running-port);
```

(2) 查看模型对应日志在 AiEngine 侧的保存路径：

```
postgres=# select track_model_train_opt('template_name', 'model_name');
```

11.3.4 最佳实践

相关参数解释参考表 20.2.21 GS_OPT_MODEL。

表 11-21 模型参数及建议

模型参数	参数建议
template_name	‘rlstm’
model_name	自定义，如‘open_ai’，需满足 unique 约束。
datname	所服务 database 名称，如‘postgres’。
ip	aiEngine-ip 地址，如‘127.0.0.1’。
port	aiEngine 侦听端口，如‘5000’。
max_epoch	迭代次数，推荐较大数值，保证收敛效果，如‘2000’。
learning_rate	(0, 1]浮点数，推荐较大的学习率，助于加快收敛速度。
dim_red	特征值降维系数： ‘-1’：不采用 PCA 降维，全量特征； ‘（0，1]’区间浮点数：越小，训练维度越小，收敛速度越快，但影响训练准确率。
hidden_units	特征值维度较高时，建议适度增大此参数，提高模型复杂度，如‘64，128……’
batch_size	根据编码数据量，较大数据量推荐适度增大此参数，加快模型收敛，如‘256，512……’
其他参数	参考表 GS_OPT_MODEL

推荐参数配置：

```
INSERT INTO gs_opt_model values('rlstm', 'open_ai', 'postgres', '127.0.0.1', 5000, 2000， 1, -1, 64, 512, 0 , false, false, '{S, T}', '{0,0}', '{0,0}', 'Text');
```

11.3.5 常见问题处理

AI Engine 配置问题

- AiEngine 启动失败：请检查 ip 地址，端口是否可用；CA 证书路径是否存在。
- 发起请求 AiEngine 无响应：请检查通信双方 CA 证书是否一致。
- 训练，测试场景失败：请检查模型文件保存路径是否存在；训练预测文件是否在正确下载。
- 更换 AiEngine-IP 地址：按照证书生成步骤重新生成证书，在生成证书及密钥中替换成相应的 IP 地址即可。

数据库内部报错问题

问题：AiEngine 链接失败。

```
ERROR:  AI engine connection failed.
CONTEXT:  referenced column: model_train_opt
```

处理方法：检查 AIEngine 是否正常拉起或重启 AIEngine；检查通信双方 CA 证书是否一致；检查模型配置信息中的 ip 和端口是否匹配；

问题：模型不存在。

```
ERROR:  OPT_Model not found for model name XXX
CONTEXT:  referenced column: track_model_train_opt
```

处理方法：检查 GS_OPT_MODEL 表中是否存在执行语句中“model_name”对应的模型；使用预测功能报错时，检查模型是否已被训练；

12 内存表特性

12.1 MOT 介绍

本章介绍了 GBase 8c 内存优化表（Memory-Optimized Table，MOT）的特性及价值、关键技术、应用场景、性能基准和竞争优势。

12.1.1 MOT 简介

GBase 8c 提供的 MOT 存储引擎是一种事务性行存储，针对多核和大内存服务器进行了优化。MOT 为事务性工作负载提供更高的性能。MOT 完全支持 ACID 特性，并包括严格的

持久性和高可用性支持。企业可以在关键任务、性能敏感的在线事务处理（OLTP）中使用 MOT，以实现高性能、高吞吐、可预测低延迟以及多核服务器的高利用率。MOT 尤其适合在多路和多核处理器的现代服务器上运行。

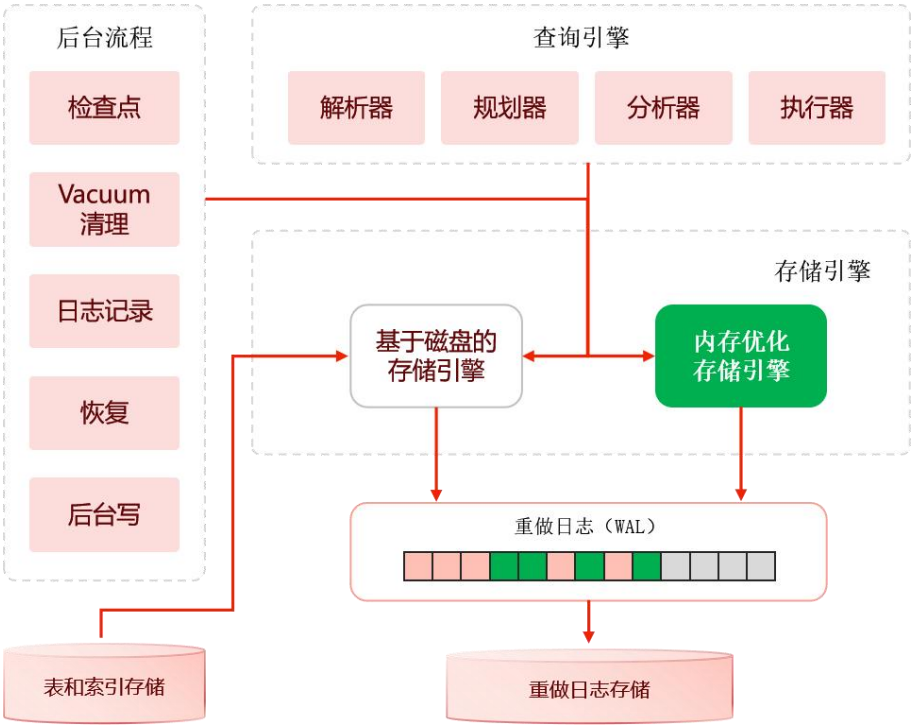


图 13-1 GBase 8c 内存优化存储引擎

如图 13-1 所示，GBase 8c 数据库内存优化存储引擎组件（绿色部分）负责管理 MOT 和事务。

MOT 与基于磁盘的普通表并排创建。MOT 的有效设计实现了几乎完全的 SQL 覆盖，并且支持完整的数据库功能集，如存储过程和自定义函数（限制参见 MOT SQL 覆盖和限制）。

通过完全存储在内存中的数据和索引、非统一内存访问感知（NUMA-aware）设计、消除锁和锁存争用的算法以及查询原生编译，MOT 可提供更快的数据访问和更高效的事务执行。

MOT 有效的几乎无锁的设计和高度调优的实现，使其在多核服务器上实现了卓越的近线性吞吐量扩展，这可能是业界最好的。

MOT 完全支持 ACID 特性：

- 原子性（Atomicity）：原子事务是一系列不可分割的数据库操作。在事务完成（分别提交或中止）之后，这些操作要么全部发生，要么全部不发生。
- 一致性（Consistency）：事务结束后，数据库处于一致状态，保留数据完整性。

- 隔离性（Isolation）：事务之间不能相互干扰。MOT 支持不同的重复读和读提交隔离级别。在下一个版本中，MOT 还将支持可序列化隔离。更多信息，请参见 MOT 隔离级别。
- 持久性（Durability）：即使发生崩溃和失败，成功完成（提交）的事务效果持久保存。MOT 完全集成了 GBase 8c 的基于 WAL 的日志记录。同时支持同步和异步日志记录选项。MOT 还支持同步+面向 NUMA 优化的组提交。详见 MOT 持久性概念。

12.1.2 MOT 特性及价值

MOT 在高性能（查询和事务延迟）、高可扩展性（吞吐量和并发量）以及高资源利用率（某些程度上节约成本）方面拥有显著优势。

- 低延迟（Low Latency）：提供快速的查询和事务响应时间。
- 高吞吐量（High Throughput）：支持峰值和持续高用户并发。
- 高资源利用率（High Resource Utilization）：充分利用硬件。

此外，高负载和高争用的场景是所有领先的行业数据库都会遇到的公认问题，而 MOT 能够在这种情况下极高地利用服务器资源。使用 MOT 后，4 路服务器的资源利用率达到 99%，远远领先其他行业数据库。

这种能力在现代的多核服务器上尤为明显和重要。

12.1.3 MOT 关键技术

MOT 的关键技术如下：

- 内存优化数据结构：以实现高并发吞吐量和可预测的低延迟为目标，所有数据和索引都在内存中，不使用中间页缓冲区，并使用持续时间最短的锁。数据结构和所有算法都是专门为内存设计而优化的。
- 免锁事务管理：MOT 在保证严格一致性和数据完整性的前提下，采用乐观的策略实现高并发和高吞吐。在事务过程中，MOT 不会对正在更新的数据行的任何版本加锁，从而大大降低了一些大内存系统中的争用。事务中的乐观并发控制（Optimistic Concurrency Control，OCC）语句是在没有锁的情况下实现的，所有的数据修改都是在内存中专门用于私有事务的部分（也称为私有事务内存）中进行的。这就意味着在事务过程中，相关数据在私有事务内存中更新，从而实现了无锁读写；而且只有在提交阶段才会短时间加锁。更多详细信息，请参见 MOT 并发控制机制。

- 免锁索引：由于内存表的数据和索引完全存储在内存中，因此拥有一个高效的索引数据结构和算法非常重要。MOT 索引机制基于最先进的 Masstree，这是一种用于多核系统的快速和可扩展的键值（Key Value，KV）存储索引，以 B+树的 Trie 实现。通过这种方式，高并发工作负载在多核服务器上可以获得卓越的性能。同时 MOT 应用了各种先进的技术以优化性能，如优化锁方法、高速缓存感知和内存预取。

- NUMA-aware 的内存管理：MOT 内存访问的设计支持非统一内存访问（NUMA）感知。NUMA-aware 算法增强了内存中数据布局的性能，使线程访问物理上连接到线程运行的核心的内存。这是由内存控制器处理的，不需要通过使用互连（如英特尔 QPI）进行额外的跳转。MOT 的智能内存控制模块，为各种内存对象预先分配了内存池，提高了性能，减少了锁，保证了稳定性。事务的内存对象的分配始终是 NUMA 本地的。本地处理的对象会返回到池中。同时在事务中尽量减少系统内存分配（OS malloc）的使用，避免不必要的锁。

- 高效持久性：日志和检查点是实现磁盘持久化的关键能力，也是 ACID 的关键要求之一（D 代表持久性）。目前所有的磁盘（包括 SSD 和 NVMe）都明显慢于内存，因此持久化是基于内存数据库引擎的瓶颈。作为一个基于内存的存储引擎，MOT 的持久化设计必须实现各种各样的算法优化，以确保持久化的同时还能达到设计时的速度和吞吐量目标。这些优化包括：

  - 并行日志，所有磁盘表都支持。
  - 每个事务的日志缓冲和无锁事务准备。
  - 增量更新记录，即只记录变化。
  - 除了同步和异步之外，创新的 NUMA 感知组提交日志记录。
  - 最先进的数据库检查点（CALC）使内存和计算开销降到最低。

- 高 SQL 覆盖率和功能集：MOT 通过扩展的 PostgreSQL 外部数据封装（FDW）以及索引，几乎支持完整的 SQL 范围，包括存储过程、用户定义函数和系统函数调用。有关不支持的功能的列表，请参阅 MOT SQL 覆盖和限制。

- 使用 PREPARE 语句的查询原生编译：通过使用 PREPARE 客户端命令，可以以交互方式执行查询和事务语句。这些命令已被预编译成原生执行格式，也称为 Code-Gen 或即时（Just-in-Time，JIT）编译。这样可以实现平均 30%的性能提升。在可能的情况下，应用编译和轻量级执行；否则，使用标准执行路径处理适用的查询。Cache Plan 模块已针对 OLTP 进行了优化，在整个会话中甚至使用不同的绑定设置以及在不同的会话中重

用编译结果。

- MOT 和数据库的无缝集成：MOT 是一个高性能的面向内存优化的存储引擎，已集成在 GBase 8c 数据库包中。MOT 的主内存引擎和基于磁盘的存储引擎并存，以支持多种应用场景，同时在内部重用数据库辅助服务，如 WAL 重做日志、复制、检查点和恢复高可用性等。用户可以从基于磁盘的表和 MOT 的统一部署、配置和访问中受益。根据特定需求，灵活且低成本地选择使用哪种存储引擎。例如，把会导致瓶颈的高度性能敏感数据放入内存中。

12.1.4 MOT 应用场景

MOT 可以根据负载的特点，显著加快应用程序的整体性能。MOT 通过提高数据访问和事务执行的效率，并通过消除并发执行事务之间的锁和锁存争用，最大程度地减少重定向，从而提高了事务处理的性能。

MOT 的极速不仅因为它在内存中，还因为它围绕并发内存使用管理进行了优化。数据存储、访问和处理算法从头开始设计，以利用内存和高并发计算的最新先进技术。

GBase 8c 允许应用程序随意组合 MOT 和基于标准磁盘的表。对于启用已证明是瓶颈的最活跃、高争用和对性能敏感的应用程序表，以及需要可预测的低延迟访问和高吞吐量的表来说，MOT 特别有用。

MOT 可用于各种应用，例如：

- 高吞吐事务处理：这是使用 MOT 的主要场景，因为它支持海量事务，同时要求单个事务的延迟较低。这类应用的例子有实时决策系统、支付系统、金融工具交易、体育博彩、移动游戏、广告投放等。
- 性能瓶颈加速：存在高争用现象的表可以通过使用 MOT 受益，即使该表是磁盘表。由于延迟更低、竞争和锁更少以及服务器吞吐量能力增加，此类表（除了相关表和在查询和事务中一起引用的表之外）的转换使得性能显著提升。
- 消除中间层缓存：云计算和移动应用往往会有周期性或峰值的高工作负载。此外，许多应用都有 80%以上负载是读负载，并伴有频繁的重复查询。为了满足峰值负载单独要求，以及降低响应延迟提供最佳的用户体验，应用程序通常会部署中间缓存层。这样的附加层增加了开发的复杂性和时间，也增加了运营成本。MOT 提供了一个很好的替代方案，通过一致的高性能数据存储来简化应用架构，缩短开发周期，降低 CAPEX 和 OPEX 成本。
- 大规模流数据提取：MOT 可以满足云端（针对移动、M2M 和物联网）、事务处理

（Transactional Processing，TP）、分析处理（Analytical Processing，AP）和机器学习（Machine Learning，ML）的大规模流数据的提取要求。MOT 尤其擅长持续快速地同时提取来自许多不同来源的大量数据。这些数据可以在以后进行处理、转换，并在速度较慢的基于磁盘的表中进行移动。另外，MOT 还可以查询到一致的、最新的数据，从而得出实时结果。在有许多实时数据流的物联网和云计算应用中，通常会有专门的数据摄取和处理。例如，一个 Apache Kafka 集群可以用来提取 10 万个事件/秒的数据，延迟为 10ms。一个周期性的批处理任务会将收集到的数据收集起来，并将转换格式，放入关系型数据库中进行进一步分析。MOT 可以通过将数据流直接存储在 MOT 关系表中，为分析和决策做好准备，从而支持这样的场景（同时消除单独的数据处理层）。这样可以更快地收集和处理数据，MOT 避免了代价高昂的分层和缓慢的批处理，提高了一致性，增加了分析数据的实时性，同时降低了总拥有成本（Total Cost of Ownership，TCO）。

- 降低 TCO：提高资源利用率和消除中间层可以节省 30%到 90%的 TCO。

12.2 使用 MOT

本章介绍如何部署、使用和管理 MOT。使用 MOT 的方法非常简单，相关命令语法与 GBase 8c 基于磁盘的表相同。只有 MOT 中的创建和删除表语句有所不同。本章主要介绍 MOT 入门、如何将基于磁盘的表转换为 MOT、如何使用 MOT 的查询原生编译功能以及 MOT 的局限性和覆盖面。还包括 MOT 管理选项，以及如何进行 TPC-C 基准测试。

12.2.1 MOT 使用概述

MOT 作为 GBase 8c 的一部分自动部署。有关如何计算和规划所需的内存和存储资源以维持工作负载的说明，请参阅 MOT 准备。参考 MOT 部署了解 MOT 中所有的配置，以及服务器优化的非必须选项。

使用 MOT 的方法非常简单。MOT 命令的语法与基于磁盘的表的语法相同，并支持大多数标准，如 PostgreSQL SQL、DDL 和 DML 命令和功能，如存储过程。只有 MOT 中的创建和删除表语句有所不同。您可以参考 MOT 使用了解这两个简单命令的说明，如何将基于磁盘的表转换为 MOT，如何使用查询原生编译和 PREPARE 语句获得更高的性能，以及了解外部工具支持和 MOT 引擎的限制。

MOT 管理介绍了如何进行数据库维护，以及监控和分析日志和错误报告。最后，MOT 样例 TPC-C 基准介绍了如何执行标准 TPC-C 基准测试。

阅读以下流程了解如何使用 MOT：

12.2.2 MOT 准备

下文介绍了使用 MOT 的前提条件以及内存和存储规划。

前提条件

以下是使用 GBase 8c MOT 的软硬件前提条件。

- 硬件支持

  MOT 支持最新硬件和现有硬件平台，支持 x86 架构和华为鲲鹏 Arm 架构。

  MOT 能够支持 GBase 8c 数据库支持的硬件。

- CPU

  MOT 在多核服务器（扩容）上提供卓越的性能。在这些环境中，MOT 的性能明显优于友商，并提供近线性扩展和极高的资源利用率。

  用户也可以开始在低端、中端和高端服务器上实现 MOT 的性能优势，无论 CPU 槽位是 1 或 2 个，还是 4 个，甚至是 8 个也没问题。在 16 路甚至 32 路的高端服务器上，性能和资源利用率也非常高。

- 内存

  MOT 支持标准 RAM/DRAM 用于其数据和事务管理。所有 MOT 数据和索引都驻留在内存中，因此内存容量必须能够支撑数据容量，并且还有进一步增长的空间。内存需求和规划，详见 MOT 内存和存储规划。

- 存储 IO

  MOT 是一个持久的数据库，使用永久性存储设备（磁盘/SSD/NVMe 驱动器）进行事务

日志操作和存储定期检查点。

推荐采用低延迟的存储设备，如配置 RAID-1 的 SSD、NVMe 或者任何企业级存储系统。当使用适当的硬件时，数据库事务处理和竞争将成为瓶颈，而非 IO。

详细的内存要求和规划，详见 MOT 内存和存储规划。

- 操作系统支持

  MOT 支持裸机和虚拟化环境

- 操作系统优化

  MOT 不需要任何特殊修改或安装新软件。但是，一些优化可以提高性能。有关实现最大性能的优化说明，请参阅 MOT 服务器优化。

MOT 内存和存储规划

本节描述了为满足特定应用程序需求，在评估、估计和规划内存和存储容量数量时，需要注意的事项和准则，以及影响所需内存数量的各种数据，例如计划表的数据和索引大小、维持事务管理的内存以及数据增长的速度。

- MOT 内存规划

  MOT 是一种内存数据库存储引擎（IMDB），其中所有表和索引完全驻留在内存中。

  说明

  内存存储是易失的，需要电力来维护所存储的信息。磁盘存储是持久的，写入磁盘是非易失性存储。MOT 使用两种存储，既把所有数据保存在内存中，也把事务性更改同步（通过 WAL 日志记录）到磁盘上以保持严格一致性（使用同步日志记录模式）。

服务器上必须有足够的物理内存以维持内存表的状态，并满足工作负载和数据的增长。所有这些都是在传统的基于磁盘的引擎、表和会话所需的内存之外的要求。因此，提前规划好足够的内存来容纳这些内容是非常有必要的。

开始可以使用任何数量的内存并执行基本任务和评估测试。但当准备好生产时，应解决以下问题：

- 内存配置

GBase 8c 数据库和标准 Postgres 类似，其内存上限是由 max_process_memory 设置的，该上限在 postgresql.conf 文件中定义。MOT 及其所有组件和线程，都驻留在进程中。因此，分配给 MOT 的内存也是在整个数据库进程的 max_process_memory 定义的上限内分配。

MOT 为自己保留的内存是 max_process_memory 的一部分。可以通过百分比或通过小于 max_process_memory 的绝对值定义。这个部分在 mot.conf 配置文件中由_mot__memory 配置项定义。

max_process_memory 中可以除了被 MOT 使用的部分之外，必须为 Postgres（GBase 8c）封装留下至少 2GB 的可用空间。为了确保这一点，MOT 在数据库启动过程中会进行如下校验：

```
(max_mot_global_memory + max_mot_local_memory) + 2GB < max_process_memory
```

如果违反此限制，则调整 MOT 内存内部限制，最大可能地满足上述限制范围。该调整在启动时进行，并据此计算 MOT 最大内存值。

说明

MOT 最大内存值是配置或调整值（max_mot_global_memory + max_mot_local_memory）的逻辑计算值。

此时，会向服务器日志发出警告，如下所示：

以下是报告问题的警告消息示例：

```
[WARNING] <Configuration> MOT engine maximum memory definitions (global: 9830 MB, local: 1843 MB, session large store: 0 MB, total: 11673 MB) breach GaussDB maximum process memory restriction (12288 MB) and/or total system memory (64243 MB). MOT values shall be adjusted accordingly to preserve required gap (2048 MB).
```

以下警告消息示例提示 MOT 正在自动调整内存限制：

```
[WARNING] <Configuration> Adjusting MOT memory limits: global = 8623 MB, local = 1617 MB, session large store = 0 MB, total = 10240 MB
```

新内存限制仅在此处显示。

此外，当总内存使用量接近所选内存限制时，MOT 不再允许插入额外数据。不再允许额外数据插入的阈值即是 MOT 最大内存百分比（如上所述，这是一个计算值）。MOT 最大内存百分比默认值为 90，即 90%。尝试添加超过此阈值的额外数据时，会向用户返回错误，并且也会注册到数据库日志文件中。

- 最小值和最大值

为了确保内存安全，MOT 根据最小的全局和本地设置预先分配内存。数据库管理员应指定 MOT 和会话维持工作负载所需的最小内存量。这样可以确保即使另一个消耗内存的应用程序与数据库在同一台服务器上运行，并且与数据库竞争内存资源，也能够将这个最小的

内存分配给 MOT。最大值用于限制内存增长。

- 全局和本地

MOT 使用的内存由两部分组成：

- 全局内存：全局内存是一个长期内存池，包含 MOT 的数据和索引。它平均分布在 NUMA 节点，由所有 CPU 核共享。
- 本地内存：本地内存是用于短期对象的内存池。它的主要使用者是处理事务的会话。这些会话将数据更改存储在专门用于相关特定事务的内存部分（称为事务专用内存）。在提交阶段，数据更改将被移动到全局内存中。内存对象分配以 NUMA-local 方式执行，以实现尽可能低的延迟。
- 被释放的对象被放回相关的内存池中。在事务期间尽量少使用操作系统内存分配（malloc）函数，避免不必要的锁和锁存。
- 这两个内存的分配由专用的 min/max_mot_global_memory 和 min/max_mot_local_memory 设置控制。如果 MOT 全局内存使用量太接近最大值，则 MOT 会保护自身，不接受新数据。超出此限制的内存分配尝试将被拒绝，并向用户报告错误。
- 最低内存要求

在开始执行对 MOT 性能的最小评估前，请确保：

除了磁盘表缓冲区和额外的内存，max_process_memory（在 postgresql.conf 中定义）还有足够的容量用于 MOT 和会话（由 mix/max_mot_global_memory 和 mix/max_mot_local_memory 配置）。对于简单的测试，可以使用 mot.conf 的默认设置。

- 生产过程中实际内存需求

在典型的 OLTP 工作负载中，平均读写比例为 80:20，每个表的 MOT 内存使用率比基于磁盘的表高 60%（包括数据和索引）。这是因为使用了更优化的数据结构和算法，使得访问速度更快，并具有 CPU 缓存感知和内存预取功能。

特定应用程序的实际内存需求取决于数据量、预期工作负载，特别是数据增长。

- 最大全局内存规划：数据和索引大小

要规划最大全局内存，需满足：

① 确定特定磁盘表(包括其数据和所有索引)的大小。如下统计查询可以确定 customer 表的数据大小和 customer_pkey 索引大小：

- 数据大小：选择 pg_relation_size（'customer'）；
- 索引：选择 pg_relation_size（'customer_pkey'）；

② 额外增加 60%的内存，相对于基于磁盘的数据和索引的当前大小，这是 MOT 中的常见要求。

③ 额外增加数据预期增长百分比。例如：

5%月增长率 = 80%年增长率（1.05^12）。因此，为了维持年增长，需分配比表当前使用的还多 80%的内存。

至此，max_mot_global_memory 值的估计和规划就完成了。实际设置可以用绝对值或 Postgres max_process_memory 的百分比来定义。具体的值通常在部署期间进行微调。

- 最大本地内存规划：并发会话支持

本地内存需求主要是并发会话数量的函数。平均会话的典型 OLTP 工作负载最大占用 8MB。此值乘以会话的数量，再加一点额外的值。

可以通过这种方式进行内存计算，然后进行微调：

```
SESSION_COUNT * SESSION_SIZE (8 MB) + SOME_EXTRA (100MB should be enough)
```

默认指定 Postgres 最大进程内存（默认为 12GB）的 15%。相当于 1.8GB 可满足 230 个会话，即 max_mot_local 内存需求。实际设置可以用绝对值或 Postgres max_process_memory 的百分比来定义。具体的值通常在部署期间进行微调。

- 异常大事务

某些事务非常大，因为它们将更改应用于大量行。这可能导致单个会话的本地内存增加到允许的最大限制，即 1GB。例如：

```
delete from SOME_VERY_LARGE_TABLE;
```

在配置 max_mot_local_memory 设置和应用程序开发时，请考虑此场景。

说明

有关配置的更多信息，请参阅内存（MOT）部分。

- 存储 IO

MOT 是一个内存优化的持久化数据库存储引擎。需要磁盘驱动器来存储 WAL 重做日志和定期检查点。

推荐采用低延迟的存储设备，如配置 RAID-1 的 SSD、NVMe 或者任何企业级存储系统。

当使用适当的硬件时，数据库事务处理和竞争将成为瓶颈，而非 IO。

由于持久性存储比 RAM 内存慢得多，因此 IO 操作（日志和检查点）可能成为内存中数据库和内存优化数据库的瓶颈。但是，MOT 具有针对现代硬件（如 SSD、NVMe）进行优化的高效持久性设计和实现。此外，MOT 最小化和优化了写入点（例如，使用并行日志记录、每个事务的单日志记录和 NUMA-aware 事务组写入），并且最小化了写入磁盘的数据（例如，只把更改记录的增量或更新列记录到日志，并且只记录提交阶段的事务）。

- 容量需求

  所需容量取决于检查点和记录的要求，如下所述：

  - 检查点

  检查点将所有数据的快照保存到磁盘。

  需要给检查点分配两倍数据大小的容量。不需要为检查点索引分配空间。

  检查点 = 2 x MOT 数据大小（仅表示行，索引非持久）。

  检查点之所以需要两倍大小，是因为快照会保存数据的全部大小到磁盘上，此外还应该为正在进行的检查点分配同样数量的空间。当检查点进程结束时，以前的检查点文件将被删除。

  说明

  MOT 增量检查点特性，这将大大降低存储容量需求。

  - 日志记录

  MOT 日志记录与基于磁盘的表的其它记录写入同一个数据库事务日志。

  日志的大小取决于事务吞吐量、数据更改的大小和检查点之间的时间（每次检查点，重做日志被截断并重新开始扩展）。

  与基于磁盘的表相比，MOT 使用较少的日志带宽和较低的 IO 争用。这由多种机制实现。

  例如，MOT 不会在事务完成之前记录每个操作。它只在提交阶段记录，并且只记录更新的增量记录（不像基于磁盘的表那样的完整记录）。

  为了确保日志 IO 设备不会成为瓶颈，日志文件必须放在具有低延迟的驱动器上。

  说明

  有关配置的更多信息，请参阅存储（MOT）。

12.2.3 MOT 部署

以下各小节介绍了各种必需和可选的设置，以达到最佳部署效果。

12.2.3.1 MOT 服务器优化

通常情况下，数据库由以下组件绑定：

- CPU：更快的 CPU 可以加速任何 CPU 绑定的数据库。
- 磁盘：高速 SSD/NVME 可加速任何 I/O 绑定数据库。
- 网络：更快的网络可以加速任何 SQL*Net 绑定数据库。

除以上内容外，以下通用服务器设置默认使用，可能会明显影响数据库的性能。

MOT 性能调优是确保快速的应用程序功能和数据检索的关键步骤。MOT 支持最新的硬件，因此调整每个系统以达到最大吞吐量是极为重要的。

BIOS

Hyper Threading 设置为 ON。

建议打开超线程（HT=ON）。建议在 MOT 上运行 OLTP 工作负载时打开超线程。当使用超线程时，某些 OLTP 工作负载显示高达 40%的性能增益。

操作系统环境设置

- NUMA

禁用 NUMA 平衡，如下所示。MOT 以极其高效的 NUMA-aware 方式进行内存管理，远远超过操作系统使用的默认方法。

```
echo 0 > /proc/sys/kernel/numa_balancing
```

- 服务

禁用如下服务：

```
service irqbalance stop                  # MANADATORY
service sysmonitor stop                   # OPTIONAL, performance
service rsyslog stop            # OPTIONAL, performance
```

- 调优服务

以下为必填项。

服务器必须运行 throughput-performance 配置文件。

```
[...]$ tuned-adm profile throughput-performance
```

throughput-performance 配置文件是广泛适用的调优，它为各种常见服务器工作负载提供卓越的性能。

其他不太适合 GBase 8c 和 MOT 服务器的配置可能会影响 MOT 的整体性能，包括：平衡配置、桌面配置、延迟性能配置、网络延迟配置、网络吞吐量配置和节能配置。

- 系统命令

  推荐使用下列操作系统设置以获得最佳性能。

  在/etc/sysctl.conf 文件中添加如下配置，然后执行 sysctl -p 命令：

```
net.ipv4.ip_local_port_range = 9000 65535
kernel.sysrq = 1
kernel.panic_on_oops = 1
kernel.panic = 5
kernel.hung_task_timeout_secs = 3600
kernel.hung_task_panic = 1
vm.oom_dump_tasks = 1
kernel.softlockup_panic = 1
fs.file-max = 640000
kernel.msgmnb = 7000000
kernel.sched_min_granularity_ns = 10000000
kernel.sched_wakeup_granularity_ns = 15000000
kernel.numa_balancing=0
vm.max_map_count = 1048576
net.ipv4.tcp_max_tw_buckets = 10000
net.ipv4.tcp_tw_reuse = 1
net.ipv4.tcp_tw_recycle = 1
net.ipv4.tcp_keepalive_time = 30
net.ipv4.tcp_keepalive_probes = 9
net.ipv4.tcp_keepalive_intvl = 30
net.ipv4.tcp_retries2 = 80
kernel.sem = 250 6400000 1000 25432
net.core.wmem_max = 21299200
net.core.rmem_max = 21299200
net.core.wmem_default = 21299200
net.core.rmem_default = 21299200
#net.sctp.sctp_mem = 94500000 915000000 927000000
#net.sctp.sctp_rmem = 8192 250000 16777216
#net.sctp.sctp_wmem = 8192 250000 16777216
```

```
net.ipv4.tcp_rmem = 8192 250000 16777216
net.ipv4.tcp_wmem = 8192 250000 16777216
net.core.somaxconn = 65535
vm.min_free_kbytes = 26351629
net.core.netdev_max_backlog = 65535
net.ipv4.tcp_max_syn_backlog = 65535
#net.sctp.addip_enable = 0
net.ipv4.tcp_syncookies = 1
vm.overcommit_memory = 0
net.ipv4.tcp_retries1 = 5
net.ipv4.tcp_syn_retries = 5
```

按如下方式修改/etc/security/limits.conf 对应部分：

```
<user> soft nofile 100000
<user> hard nofile 100000
```

软限制和硬限制设置可指定一个进程同时打开的文件数量。软限制可由各自运行这些限制的进程进行更改，直至达到硬限制值。

- 磁盘/SSD

下面以数据库同步提交模式为例，介绍如何保证磁盘读写性能适合数据库同步提交模式。

按如下方式运行磁盘/SSD 性能测试：

```
[...]$ sync; dd if=/dev/zero of=testfile bs=1M count=1024; sync
1024+0 records in
1024+0 records out
1073741824 bytes (1.1 GB) copied, 1.36034 s, 789 MB/s
```

当磁盘带宽明显低于 789MB/s 时，可能会造成 GBase 8c 性能瓶颈，尤其是造成 MOT 性能瓶颈。

网络

需要使用 10Gbps 以上网络。

运行 iperf 命令进行验证：

```
Server side: iperf -s
Client side: iperf -c <IP>
rc.local：网卡调优
```

以下可选设置对性能有显著影响：

① 将 https://gist.github.com/SaveTheRbtz/8875474 下的 set_irq_privacy.sh 文件拷贝到

/var/scripts/目录下。

② 进入/etc/rc.d/rc.local，执行 chmod 命令，确保在 boot 时执行以下脚本：

```
'chmod +x /etc/rc.d/rc.local'
var/scripts/set_irq_affinity.sh -x all <DEVNAME>
ethtool -K <DEVNAME> gro off
ethtool -C <DEVNAME> adaptive-rx on adaptive-tx on
Replace <DEVNAME> with the network card, i.e. ens5f1
```

12.2.3.2 MOT 配置

预置 MOT 用于创建工作 MOT。为了获得最佳效果，建议根据应用程序的特定要求和偏好自定义 MOT 配置（在 mot.conf 文件中定义）。

该文件在服务器启动时只读。如果在系统运行中编辑此文件，则必须重新加载服务器才能使修改内容生效。

mot.conf 文件与 postgresql.conf 配置文件在同一文件夹下。

在主备部署模式下，主备节点的 mot.conf 文件需要完全相同，否则，系统行为不明确。

阅读总体原则，根据需要查看和配置 mot.conf 文件。

说明

以上描述了 mot.conf 文件中的各个设置。除上述内容外，要了解特定 MOT 功能（如恢复），可参考本用户手册的相关章节。例如，MOT 恢复说明了 mot.conf 文件的恢复，包含影响 MOT 恢复的设置。此外，有关恢复的完整说明，请参阅“MOT 管理”章节的 MOT 恢复。下文各相关章节中还提供了参考链接。

以下介绍了 mot.conf 文件中的各个部分，其包含的设置以及默认值。

总体原则

以下是编辑 mot.conf 文件的总体原则。

- 每个设置项都带有默认值，如下所示：

```
# name = value
```

- 可以接受空格或留空。
- 在各行添加#号可进行注释。
- 每个设置项的默认值将作为注释显示在整个文件中。

- 如果参数没有注释并且置入了新值，则定义新设置。
- 对 mot.conf 文件的更改仅在数据库服务器启动或重装时生效。

内存单元的表示如下：

- KB：千字节
- MB：兆字节
- GB：吉字节
- TB：太字节

如果未指定内存单元，则假定为字节。

某些内存单位为 postgresql.conf 中的 max_process_memory 的百分比值。例如，20%。

时间单位表示如下：

- us：微秒
- ms：毫秒
- s：秒
- min：分钟
- h：小时
- d：天

如果未指定时间单位，则假定为微秒。

重做日志（MOT）

- enable_group_commit = false

是否使用组提交。

该选项仅在 GBase 8c 配置为使用同步提交时相关，即仅当 postgresql.conf 中的 synchronization_commit 设置为除 off 以外的任何值时相关。

有关 WAL 重做日志的详细信息，请参阅 MOT 日志记录：WAL 重做日志。

- group_commit_size = 16
- group_commit_timeout = 10 ms

只有当 MOT 引擎配置为同步组提交日志记录时，此选项才相关。即 postgresql.conf 中

的 synchronization_commit 配置为 true，mot.conf 配置文件中的 enable_group_commit 配置为 true。

当一组事务记录在 WAL 重做日志中时，需确定以下设置项取值：

group_commit_size：一组已提交的事务数。例如，16 表示当同一组中的 16 个事务已由它们的客户端应用程序提交时，则针对 16 个事务中的每个事务，在磁盘的 WAL 重做日志中写入一个条目。

group_commit_timeout：超时时间，单位为毫秒。例如，10 表示在 10 毫秒之后，为同一组由客户端应用程序在最近 10 毫秒内提交的每个事务，在磁盘的 WAL 重做日志中写入一个条目。

提交组在到达配置的事务数后或者在超时后关闭。组关闭后，组中的所有事务等待一个组落盘完成执行，然后通知客户端每个事务都已经结束。

有关同步组提交日志记录的详细信息，请参阅 MOT 日志类型。

检查点（MOT）

- checkpoint_dir =

  指定检查点数据存放目录。默认位置在每个数据节点的 data 文件夹中。

- checkpoint_segsize = 16 MB

  指定检查点时使用的段大小。分段执行检查点。当一个段已满时，它将被序列化到磁盘，并为后续的检查点数据打开一个新的段。

- checkpoint_workers = 3

  指定在检查点期间要使用的工作线程数。

检查点由多个 MOT 引擎工作线程并行执行。工作线程的数量可能会大大影响整个检查点操作的整体性能，以及其它正在运行的事务的操作。为了实现较短的检查点持续时间，应使用更多线程，直至达到最佳数量（根据硬件和工作负载的不同而不同）。但请注意，如果这个数目太大，可能会对其他正在运行的事务的执行时间产生负面影响。尽可能低这个数字，以最小化对其他运行事务的运行时的影响。当此数目过高时，检查点持续时间会较长。

说明

有关配置的更多信息，请参阅 MOT 持久性中的 MOT 检查点。

恢复（MOT）

- checkpoint_recovery_workers = 3

  指定在检查点数据恢复期间要使用的工作线程数。每个 MOT 引擎工作线程在自己的核上运行，通过将不同的表读入内存，可以并行处理不同的表。缺省值为 3，可将此参数设置为可处理的核数。恢复后，将停止并杀死这些线程。

  说明

  有关配置的详细信息，请参阅 MOT 恢复。

统计（MOT）

- enable_stats = false

  设置周期性统计打印信息。

- print_stats_period = 10 minute

  设置汇总统计报表打印的时间范围。

- print_full_stats_period = 1 hours

  设置全量统计报表打印的时间范围。

  以下设置为周期性统计报表中的各个部分。如果没有配置，则抑制统计报表。

- enable_log_recovery_stats = false

  日志恢复统计信息包含各种重做日志的恢复指标。

- enable_db_session_stats = false

  数据库会话统计信息包含事务事件，如提交、回滚等。

- enable_network_stats = false

  网络统计信息包括连接/断连事件。

- enable_log_stats = false

  日志统计信息包含重做日志详情。

- enable_memory_stats = false

  内存统计信息包含内存层详情。

- enable_process_stats = false

  进程统计信息包含当前进程的内存和 CPU 消耗总量。

- enable_system_stats = false

  系统统计信息包含整个系统的内存和 CPU 消耗总量。

- enable_jit_stats = false

  JIT 统计信息包含有关 JIT 查询编译和执行的信息。

错误日志（MOT）

- log_level = INFO

  设置 MOT 引擎下发的消息在数据库服务器的错误日志中记录的日志级别。有效值为 PANIC、ERROR、WARN、INFO、TRACE、DEBUG、DIAG1、DIAG2。

- Log/COMPONENT/LOGGER=LOG_LEVEL

  使用以下语法设置特定的日志记录器。

  例如，要为系统组件中的 ThreadIdPool 日志记录器配置 TRACE 日志级别，请使用以下语法：

- Log.System.ThreadIdPool.log_level=TRACE

  要为某个组件下的所有记录器配置日志级别，请使用以下语法：

- Log.COMPONENT.log_level=LOG_LEVEL

  例如：

```
Log.System.log_level=DEBUG
```

内存（MOT）

- enable_numa = true

  指定是否使用可识别 NUMA 的内存。禁用时，所有亲和性配置也将被禁用。MOT 引擎假定所有可用的 NUMA 节点都有内存。如果计算机具有某些特殊配置，其中某些 NUMA 节点没有内存，则 MOT 引擎初始化将因此失败，因此数据库服务器启动将失败。在此类计算机中，建议将此配置值设置为 false，以防止启动失败并让 MOT 引擎在不使用可识别 NUMA 的内存分配的情况下正常运行。

- affinity_mode = fill-physical-first

  设置用户会话和内部 MOT 任务的线程亲和模式。

  使用线程池时，用户会话将忽略此值，因为它们的亲和性由线程池控制。但内部 MOT

任务仍然使用。

有效值为 fill-socket-first、equal-per-socket、fill-physical-first、none。

- Fill-socket-first 将线程连接到同一个槽位的核上，直到槽位已满，然后移动到下一个槽位。
- Equal-per-socket 使线程均匀分布在所有槽位中。
- Fill-physical-first 将线程连接到同一个槽位中的物理核，直到用尽所有物理核，然后移动到下一个槽位。当所有物理核用尽时，该过程再次从超线程核开始。
- None 禁用任何亲和配置，并让系统调度程序确定每个线程调度在哪个核上运行。

- lazy_load_chunk_directory = true

设置块目录模式，用于内存块查找。

Lazy 模式将块目录设置为按需加载部分目录，从而减少初始内存占用（大约从 1GB 减少到 1MB）。然而，这可能会导致轻微的性能损失和极端情况下的内存损坏。相反，使用 non-lazy 块目录会额外分配 1GB 的初始内存，产生略高的性能，并确保在内存损坏期间避免块目录错误。

- reserve_memory_mode = virtual

设置内存预留模式（取值为 physical 或 virtual）。

每当从内核分配内存时，都会参考此配置值来确定所分配的内存是常驻（physical）还是非常驻（virtual）。这主要与预分配有关，但也可能影响运行时分配。对于 physical 保留模式，通过强制内存区域所跨越的所有页出现页错误，使整个分配的内存区域常驻。配置 virtual 内存预留可加速内存分配（特别是在预分配期间），但可能在初始访问期间出现页错误（因此导致轻微的性能影响），并在物理内存不可用时出现更多服务器错误。相反，物理内存分配速度较慢，但后续访问速度更快且有保障。

- store_memory_policy = compact

设置内存存储策略（取值为 compact 或 expanding）。

当定义了 compact 策略时，未使用的内存会释放回内核，直到达到内存下限（请参见下面的 min_mot_memory）。在 expanding 策略中，未使用的内存存储在 MOT 引擎中，以便后续再使用。compact 存储策略可以减少 MOT 引擎的内存占用，但偶尔会导致性能轻微下降。此外，在内存损坏时，它还可能导致内存不可用。相反，expanding 模式会占用更多的内存，但是会更快地分配内存，并且能够更好地保证在解分配后能够重新分配内存。

- chunk_alloc_policy = auto

  设置全局内存的块分配策略。

  MOT 内存以 2MB 的块为单位组织。源 NUMA 节点和每个块的内存布局会影响表数据在 NUMA 节点间的分布，因此对数据访问时间有很大影响。在特定 NUMA 节点上分配块时，会参考分配策略。

  可用值包括 auto、local、page-interleaved、chunk-interleaved、native。

  - Auto 策略根据当前硬件情况选择块分配策略。
  - Local 策略在各自的 NUMA 节点上分配每个数据块。
  - Page-interleaved 策略从所有 NUMA 节点分配由交插内存 4 千字节页组成的数据块。
  - Chunk-interleaved 策略以轮循调度方式从所有 NUMA 节点分配数据块。
  - Native 策略通过调用原生系统内存分配器来分配块。

- chunk_prealloc_worker_count = 8

  设置每个 NUMA 节点参与内存预分配的工作线程数。

- max_mot_global_memory = 80%

  设置 MOT 引擎全局内存的最大限制。

  指定百分比值与 postgresql.conf 中 max_process_memory 定义的总量有关。

  MOT 引擎内存分为全局（长期）内存，主要用于存储用户数据，以及本地（短期）内存，主要用于用户会话，以满足本地需求。

  任何试图分配超出此限制的内存的尝试将被拒绝，并向用户报告错误。请确保 max_mot_global_memory 与 max_mot_local_memory 之和不超过 postgresql.conf 中配置的 max_process_memory。

- min_mot_global_memory = 0 MB

  设置 MOT 引擎全局内存的最小限制。

  指定百分比值与 postgresql.conf 中 max_process_memory 定义的总量有关。

  此值用于启动期间的内存预分配，以及确保 MOT 引擎在正常运行期间有最小的内存可用量。当使用 compact 存储策略时（参阅上文 store_memory_policy），该值指定了下限，超过下限的内存不会释放回内核，而是保留在 MOT 引擎中以便后续重用。

- max_mot_local_memory = 15%

  设置 MOT 引擎本地内存的最大限制。

  指定百分比值与 postgresql.conf 中 max_process_memory 定义的总量有关。

  MOT 引擎内存分为全局（长期）内存，主要用于存储用户数据，以及本地（短期）内存，主要用于用户会话，以满足本地需求。

  任何试图分配超出此限制的内存的尝试将被拒绝，并向用户报告错误。请确保 max_mot_global_memory 与 max_mot_local_memory 之和不超过 postgresql.conf 中配置的 max_process_memory。

- min_mot_local_memory = 0 MB

  设置 MOT 引擎本地内存的最小限制。

  指定百分比值与 postgresql.conf 中 max_process_memory 定义的总量有关。

  此值用于在启动期间预分配内存，以及确保 MOT 引擎在正常运行期间有最小的可用内存。当使用 compact 存储策略时（参阅上文 store_memory_policy），该值指定了下限，超过下限的内存不会释放回内核，而是保留在 MOT 引擎中以便后续重用。

- max_mot_session_memory = 0 MB

  设置 MOT 引擎中单个会话的最大内存限制。

  指定百分比值与 postgresql.conf 中 max_process_memory 定义的总量有关。

  通常，MOT 引擎中的会话可以根据需要分配尽可能多的本地内存，只要没有超出本地内存限制即可。为了避免单个会话占用过多的内存，从而拒绝其他会话的内存，通过该配置项限制小会话的本地内存分配（最大 1022KB）。

  请确保该配置项不影响大会话的本地内存分配。

  0 表示不会限制每个小会话的本地分配，除非是由 max_mot_local_memory 配置的本地内存分配限制引起的。

- min_mot_session_memory = 0 MB

  设置 MOT 引擎中单个会话的最小内存预留。

  指定百分比值与 postgresql.conf 中 max_process_memory 定义的总量有关。

  此值用于在会话创建期间预分配内存，以及确保会话有最小的可用内存量来执行其正常操作。

- session_large_buffer_store_size = 0 MB

  设置会话的大缓冲区存储。

  当用户会话执行需要大量内存的查询时（例如，使用许多行），大缓冲区存储用于增加此类内存可用的确定级别，并更快地为这个内存请求提供服务。对于超过 1022KB 的会话，任何内存分配都是大内存分配。如果未使用或耗尽了大缓冲区存储，则这些分配将被视为直接从内核提供的巨大分配。

- session_large_buffer_store_max_object_size = 0 MB

  设置会话的大分配缓冲区存储中的最大对象大小。

  大缓冲区存储内部被划分为不同大小的对象。此值用于对源自大缓冲区存储的对象设置上限，以及确定缓冲区存储内部划分为不同大小的对象。

  此大小不能超过 session_large_buffer_store_size 的 1/8。如果超过，则将其调整到最大可能。

- session_max_huge_object_size = 1 GB

  设置会话单个大内存分配的最大尺寸。

  巨大分配直接从内核中提供，因此不能保证成功。

  此值也适用于全局（非会话相关）内存分配。

垃圾收集（MOT）

- enable_gc = true

  是否使用垃圾收集器（Garbage Collector，GC）。

- reclaim_threshold = 512 KB

  设置垃圾收集器的内存阈值。

  每个会话管理自己的待回收对象列表，并在事务提交时执行自己的垃圾回收。此值决定了等待回收的对象的总内存阈值，超过该阈值，会话将触发垃圾回收。

  一般来说，这里是在权衡未回收对象与垃圾收集频率。设置低值会使未回收的内存保持在较少的水平，但会导致频繁的垃圾回收，从而影响性能。设置高值可以减少触发垃圾回收的频率，但会导致未回收的内存过多。此设置取决于整体工作负载。

- reclaim_batch_size = 5432

  设置垃圾回收的批次大小。

垃圾收集器从对象中批量回收内存，以便限制在一次垃圾收集传递中回收的对象数量。此目的是最小化单个垃圾收集传递的操作时间。

- high_reclaim_threshold = 8 MB

  设置垃圾回收的高内存阈值。

  由于垃圾收集是批量工作的，因此会话可能有许多可以回收的对象，但这些对象不能回收。在这种情况下，为了防止垃圾收集列表变得过于膨胀，尽管已经达到批处理大小限制，此值继续单独回收对象，直到待回收对象小于该阈值，或者没有更多符合回收条件的对象。

JIT（MOT）

- enable_mot_codegen = true

  指定是否对计划查询使用 JIT 查询编译和执行。

  JIT 查询执行为在计划阶段准备好的查询准备了 JIT 编译的代码。每当调用准备好的查询时，都会执行生成的 JIT 编译函数。JIT 编译通常以 LLVM 的形式进行。在原生不支持 LLVM 的平台上，MOT 提供了基于软件的回退（Tiny Virtual Machine，TVM）。

- force_mot_pseudo_codegen = false

  当前平台支持 LLVM 时，是否使用 TVM（伪 LLVM）。

  在原生不支持 LLVM 的平台上，MOT 自动默认为 TVM。

  在原生支持 LLVM 的平台上，默认使用 LLVM。该配置项允许在支持 LLVM 的平台上使用 TVM 进行 JIT 编译和执行。

- enable_mot_codegen_print = false

  指定是否为 JIT 编译的查询打印发出的 LLVM/TVM IR 代码。

- mot_codegen_limit = 100

  限制每个用户会话允许的 JIT 查询数。

默认 MOT.conf 文件

最小设置和配置指定将 postgresql.conf 文件指向 MOT.conf 文件的位置：

```
postgresql.conf
mot_config_file = '/tmp/gauss/ MOT.conf'
```

确保 max_process_memory 设置的值足够包含 MOT 的全局（数据和索引）和本地（会话）内存。

MOT.conf 的默认内容满足开始使用的需求。设置内容后续可以优化。

12.2.4 MOT 使用

使用 MOT 非常简单，以下几个小节将会进行描述。

GBase 8c 允许应用程序使用 MOT 和基于标准磁盘的表。MOT 适用于最活跃、高竞争和对吞吐量敏感的应用程序表，也可用于所有应用程序的表。

以下命令介绍如何创建 MOT，以及如何将现有的基于磁盘的表转换为 MOT，以加速应用程序的数据库相关性能。MOT 尤其有利于已证明是瓶颈的表。

工作流程概述

以下是与使用 MOT 相关的任务的简单概述：

本小节还介绍了如何执行各种与 MOT 相关的附加任务，以及 MOT SQL 覆盖和限制。

12.2.4.1 授予用户权限

以授予数据库用户对 MOT 存储引擎的访问权限为例。每个数据库用户仅执行一次，通常在初始配置阶段完成。

说明

MOT 通过外部数据封装器 (Foreign Data Wrapper，FDW) 机制与 GBase 8c 数据库集成，所以需要授权用户权限。

要使特定用户能够创建和访问 MOT（DDL、DML、SELECT），以下语句只执行一次：

```
GRANT USAGE ON FOREIGN SERVER mot_server TO <user>;
```

所有关键字不区分大小写。

12.2.4.2 创建/删除 MOT

创建 MOT 非常简单。只有 MOT 中的创建和删除表语句与 GBase 8c 中基于磁盘的表的语句不同。SELECT、DML 和 DDL 的所有其他命令的语法对于 MOT 表和 GBase 8c 基于磁盘的表是一样的。

- 创建 MOT：

```
create FOREIGN table test(x int) [server mot_server];
```

- 以上语句中：
  - 始终使用 FOREIGN 关键字引用 MOT。
  - 在创建 MOT 表时，[server mot_server]部分是可选的，因为 MOT 是一个集成的引擎，而不是一个独立的服务器。
  - 上文以创建一个名为 test 的内存表（表中有一个名为 x 的整数列）为例。在下一节（创建索引）中将提供一个更现实的例子。
  - 如果 postgresql.conf 中开启了增量检查点，则无法创建 MOT。因此请在创建 MOT 前将 enable_incremental_checkpoint 设置为 off。
- 删除名为 test 的 MOT：

```
drop FOREIGN table test;
```

有关 MOT 的功能限制（如数据类型），请参见 MOT SQL 覆盖和限制。

12.2.4.3 为 MOT 创建索引

支持标准的 PostgreSQL 创建和删除索引语句。

例如：

```
create index    text_index1 on test(x) ;
```

创建一个用于 TPC-C 的 ORDER 表，并创建索引：

```
create FOREIGN table bmsql_oorder (
  o_w_id          integer          not null,
  o_d_id          integer          not null,
  o_id            integer          not null,
  o_c_id          integer not null,
  o_carrier_id integer,
  o_ol_cnt        integer,
```

```
    o_all_local    integer,
    o_entry_d      timestamp,
    primary key (o_w_id, o_d_id, o_id)
);
create index    bmsql_oorder_index1 on bmsql_oorder(o_w_id, o_d_id, o_c_id, o_id) ;
```

说明

在 MOT 名字之前不需要指定 FOREIGN 关键字，因为它仅用于创建和删除表的命令。

有关 MOT 索引限制，请参见 MOT SQL 覆盖和限制的索引部分内容。

12.2.4.4 将磁盘表转换为 MOT

磁盘表直接转换为 MOT 尚不能实现，这意味着尚不存在将基于磁盘的表转换为 MOT 的 ALTER TABLE 语句。

下面介绍如何手动将基于磁盘的表转换为 MOT，如何使用 gs_dump 工具导出数据，以及如何使用 gs_restore 工具导入数据。

前置条件检查

检查待转换为 MOT 的磁盘表的模式是否包含所有需要的列。

检查架构是否包含任何不支持的列数据类型，具体参见 MOT SQL 覆盖和限制中的不支持的数据类型。

如果不支持特定列，则建议首先创建一个更新了模式的备磁盘表。此模式与原始表相同，只是所有不支持的类型都已转换为支持的类型。

使用以下脚本导出该备磁盘表，然后导入到 MOT 中。

转换

要将基于磁盘的表转换为 MOT，请执行以下步骤：

(1) 暂停应用程序活动。

(2) 使用 gs_dump 工具将表数据转储到磁盘的物理文件中。请确保使用 data only。

(3) 重命名原始基于磁盘的表。

(4) 创建同名同模式的 MOT。请确保使用创建 FOREIGN 关键字指定该表为 MOT。

(5) 使用 gs_restore 将磁盘文件的数据加载/恢复到数据库表中。

(6) 浏览或手动验证所有原始数据是否正确导入到新的 MOT 中。下面将举例说明。

(7) 恢复应用程序活动。

须知

由于表名称保持不变，应用程序查询和相关数据库存储过程将能够无缝访问新的 MOT，而无需更改代码。请注意，MOT 目前不支持跨引擎多表查询（如使用 Join、Union 和子查询）和跨引擎多表事务。因此，如果在多表查询、存储过程或事务中访问原始表，则必须将所有相关的磁盘表转换为 MOT，或者更改应用程序或数据库中的相关代码。

转换示例

假设要将数据库 benchmarksql 中一个基于磁盘的表 customer 迁移到 MOT 中。将 customer 表迁移到 MOT，操作步骤如下：

(1) 检查源表列类型。验证 MOT 支持所有类型，详情请参阅 MOT SQL 覆盖和限制中的不支持的数据类型。

```
benchmarksql-# \d+ customer
                                Table "public.customer"
 Column |    Type      | Modifiers | Storage | Stats target | Description
--------+---------+-----------+---------+--------------+-------------
 x          | integer |                 | plain     |                    |
 y          | integer |                 | plain     |                    |
Has OIDs: no
Options: orientation=row, compression=no
```

(2) 请检查源表数据。

```
benchmarksql=# select * from customer;
 x | y
---+---
 1 | 2
 3 | 4
(2 rows)
```

(3) 只能使用 gs_dump 转储表数据。

```
$ gs_dump -Fc benchmarksql -a --table customer -f customer.dump -p 16000
gs_dump[port='15500'][benchmarksql][2020-06-04 16:45:38]: dump database benchmarksql successfully
gs_dump[port='15500'][benchmarksql][2020-06-04 16:45:38]: total time: 332    ms
```

(4) 重命名源表。

```
benchmarksql=# alter table customer rename to customer_bk;
```

```
ALTER TABLE
```

(5) 创建与源表完全相同的 MOT。

```
benchmarksql=# create foreign table customer (x int, y int);
CREATE FOREIGN TABLE
benchmarksql=# select * from customer;
 x | y
---+---
(0 rows)
```

(6) 将源转储数据导入到新 MOT 中。

```
$ gs_restore -C -d benchmarksql customer.dump -p 16000
restore operation successful
total time: 24    ms
Check that the data was imported successfully.
benchmarksql=# select * from customer;
 x | y
---+---
 1 | 2
 3 | 4
(2 rows)

benchmarksql=# \d
                                    List of relations
 Schema |       Name         |        Type          | Owner    |                    Storage
--------+-------------+---------------+--------+----------------------------------
 public | customer       | foreign table | aharon |
 public | customer_bk | table                | aharon | {orientation=row,compression=no}
(2 rows)
```

12.2.4.5 查询原生编译

MOT 的另一个特性是，在预编译的完整查询需要执行之前，能够以原生格式（使用 PREPARE 语句）准备并解析这些查询。

这种原生格式方便后续更有效地执行（使用 EXECUTE 命令）。这种执行类型速度要快得多，因为原生格式在执行期间绕过多个数据库处理层，从而获得更好的性能。

这种分工避免了重复的解析分析操作。查询和事务语句可以交互执行。此功能有时称为即时（Just-In-Time，JIT）查询编译。

查询编译：PREPARE 语句

若要使用 MOT 的原生查询编译，请在执行查询之前调用 PREPARE 客户端语句。MOT 将预编译查询和（或）从缓存预加载先前预编译的代码。

下面是 SQL 中 PREPARE 语法的示例：

```
PREPARE name [ ( data_type [, ...] ) ] AS statement
```

PREPARE 在数据库服务器中创建一个预处理语句，该语句是一个可用于优化性能的服务器端对象。

运行命令

发出 EXECUTE 命令时，将解析、分析、重写和执行预处理语句。这种分工避免了重复的解析分析操作，同时使执行计划依赖于特定的设置值。

下面是在 Java 应用程序中调用 PREPARE 和 EXECUTE 语句的示例。

```
conn = DriverManager.getConnection(connectionUrl, connectionUser, connectionPassword);
// Example 1: PREPARE without bind settings
String query = "SELECT * FROM getusers";
PreparedStatement prepStmt1 = conn.prepareStatement(query);
ResultSet rs1 = pstatement.executeQuery())
while (rs1.next()) {•••}
// Example 2: PREPARE with bind settings
String sqlStmt = "SELECT * FROM employees where first_name=? and last_name like ?";
PreparedStatement prepStmt2 = conn.prepareStatement(sqlStmt);
prepStmt2.setString(1, "Mark"); // first name “Mark”
prepStmt2.setString(2, "%n%"); // last name contains a letter “n”
ResultSet rs2 = prepStmt2.executeQuery())
while (rs2.next()) {•••}
```

MOT 编译支持的特性和不支持的特性见下文。

轻量执行支持的查询

以下查询类型适合轻量执行：

- 简单点查询
  - SELECT (including SELECT for UPDATE)
  - UPDATE
  - DELETE
- INSERT 查询

- 引用主键的完整前缀的范围 UPDATE 查询
- 引用主键的完整前缀的范围 SELECT 查询
- JOIN 查询，其中一部分或两部分重叠为点查询
- 引用每个连接表中主键的完整前缀的 JOIN 查询

轻量执行不支持的查询

任何特殊的查询属性都不适用于轻量执行。特别是如果以下条件中的任何一项适用，则该查询不适合轻量执行。有关更多信息，请参阅“原生编译和轻量执行不支持的查询”。

需要强调一点，如果查询语句不适用原生编译和轻量执行，不向客户端报告错误，查询仍以正常和规范的方式执行。

有关 MOT 原生编译功能的详细信息，请参阅 查询原生编译或查询原生编译（JIT）的有关内容。

12.2.4.6 重试中止事务

在乐观并发控制（OCC）中，在 COMMIT 阶段前的事务期间（使用任何隔离级别）不会对记录进行锁定。这是一个能显著提高性能的强大优势。它的缺点是，如果另一个会话尝试更新相同的记录，则更新可能会失败。所以必须中止整个事务。这些所谓的更新冲突是由 MOT 在提交时通过版本检查机制检测到的。

说明

使用悲观并发控制的引擎，如标准 Postgres 和 GBase 8c 基于磁盘的表，当使用 SERIALIZABLE 或 REPEATABLE-READ 隔离级别时，也会发生类似的异常中止。

这种更新冲突在常见的 OLTP 场景中非常少见，在使用 MOT 时尤其少见。但是，由于仍有可能发生这种情况，开发人员应该考虑使用事务重试代码来解决此问题。

下面以多个会话同时尝试更新同一个表为例，说明如何重试表命令。有关更多详细信息，请参阅“OCC 与 2PL 的区别举例”部分。下面以 TPC-C 支付事务为例。

```
int commitAborts = 0;
while (commitAborts < RETRY_LIMIT) {
    try {
        stmt =db.stmtPaymentUpdateDistrict;
        stmt.setDouble(1, 100);
        stmt.setInt(2, 1);
        stmt.setInt(3, 1);
```

```
            stmt.executeUpdate();
            db.commit();
            break;
        }
        catch (SQLException se) {
            if(se != null && se.getMessage().contains("could not serialize access due to
concurrent update")) {
                log.error("commmit abort = " + se.getMessage());
                commitAborts++;
                continue;
            }else {
                db.rollback();
            }
            break;
        }
}
```

12.2.4.7 MOT 外部支持工具

为了支持 MOT，修改了以下外部 GBase 8c 工具。请确保使用的工具是最新版本。下面将介绍与 MOT 相关的用法。

gs_ctl（全量和增量）

此工具用于从主服务器创建备服务器，以及当服务器的时间线偏离后，将服务器与其副本进行同步。

在操作结束时，工具将获取最新的 MOT 检查点，同时考虑 checkpoint_dir 配置值。

检查点从源服务器的 checkpoint_dir 读取到目标服务器的 checkpoint_dir。

目前 MOT 不支持增量检查点。因此，gs_ctl 增量构建对于 MOT 来说不是以增量方式工作，而是以全量方式工作。Postgres 磁盘表仍然可以增量构建。

gs_basebackup

gs_basebackup 用于准备运行中服务器的基础备份，不影响其他数据库客户端。

MOT 检查点也会在操作结束时获取。但是，检查点的位置是从源服务器中的 checkpoint_dir 获取的，并传输到源数据目录中，以便正确备份。

gs_dump

gs_dump 用于将数据库模式和数据导出到文件中。支持 MOT。

gs_restore

gs_restore 用于从文件中导入数据库模式和数据。支持 MOT。

12.2.4.8 MOT SQL 覆盖和限制

MOT 设计几乎能够覆盖 SQL 和未来特性集。例如，大多数支持标准的 Postgres SQL，也支持常见的数据库特性，如存储过程、自定义函数等。

下面介绍各种 SQL 覆盖和限制。

不支持的特性

MOT 不支持以下特性：

- 跨引擎操作：不支持跨引擎（磁盘+MOT）的查询、视图或事务。计划于 2021 年实现该特性。
- MVCC、隔离：不支持没有快照/可序列化隔离。计划于 2021 年实现该特性。
- 原生编译（JIT）：SQL 覆盖有限。此外，不支持存储过程的 JIT 编译。
- 本地内存限制为 1GB。一个事务只能更改小于 1GB 的数据。
- 容量（数据+索引）受限于可用内存。未来将提供 Anti-caching 和数据分层功能。
- 不支持全文检索索引。
- 不支持逻辑复制特性。

此外，下面详细列出了 MOT、MOT 索引、查询和 DML 语法的各种通用限制，以及查询原生编译的特点和限制。

MOT 限制

MOT 功能限制：

- 按范围分区
- AES 加密
- 流操作
- 自定义类型
- 子事务
- DML 触发器

- DDL 触发器
- “C”或“POSIX”以外的排序规则

不支持的 DDL 操作

- 修改表结构
- 创建 including 表
- 创建 as select 表
- 按范围分区
- 创建无日志记录子句（no-logging clause）的表
- 创建可延迟约束主键（DEFERRABLE）
- 重建索引
- 表空间
- 使用子命令创建架构

不支持的数据类型

- UUID
- User-Defined Type (UDF)
- Array data type
- NVARCHAR2(n)
- NVARCHAR(n)
- Clob
- Name
- Blob
- Raw
- Path
- Circle
- Reltime
- Bit varying(10)

- Tsvector
- Tsquery
- JSON
- HSTORE
- Box
- Text
- Line
- Point
- LSEG
- POLYGON
- INET
- CIDR
- MACADDR
- Smalldatetime
- BYTEA
- Bit
- Varbit
- OID
- Money
- 无限制的 varchar/character varying
- HSTORE
- XML

不支持的索引 DDL 和索引

- 在小数和数值类型上创建索引
- 在可空列上创建索引
- 单表创建索引总数>9

- 在键大小>256 的表上创建索引

键大小包括以字节为单位的列大小+列附加大小，这是维护索引所需的开销。下表列出了不同列类型的列附加大小。

此外，如果索引不是唯一的，额外需要 8 字节。

下面是伪代码计算键大小：

```
keySize =0;

for each (column in index){
        keySize += (columnSize + columnAddSize);
}
if (index is non_unique) {
        keySize += 8;
}
```

列类型	列大小	列附加大小
varchar	N	4
tinyint	1	1
smallint	2	1
int	4	1
longint	8	1
float	4	2
double	8	3

上表中未指定的类型，列附加大小为零（例如时间戳）。

不支持的 DML

- Merge into
- Select into
- Lock table
- Copy from table

原生编译和轻量执行不支持的查询

- 查询涉及两个以上的表
- 查询有以下任何一个情况：
  - 非原生类型的聚合
  - 窗口功能
  - 子查询子链接
  - Distinct-ON 修饰语（distinct 子句来自 DISTINCT ON）
  - 递归（已指定 WITH RECURSIVE）
  - 修改 CTE（WITH 中有 INSERT/UPDATE/DELETE）

  以下子句不支持轻量执行：
- Returning list
- Group By clause
- Grouping sets
- Having clause
- Windows clause
- Distinct clause
- Sort clause that does not conform to native index order
- Set operations
- Constraint dependencies

12.2.5 MOT 管理

12.2.5.1 MOT 持久性

持久性是指长期的数据保护（也称为磁盘持久化）。持久性意味着存储的数据不会遭受任何形式的退化或损坏，因此数据不会丢失或损坏。持久性可确保在有计划停机（例如维护）或计划外崩溃（例如电源故障）后数据和 MOT 引擎恢复到一致状态。

内存存储是易失的、需要电力来维护所存储的信息。另一方面，磁盘存储是非易失性的，这意味着它不需要电源来维护存储的信息，因此不用担心停电。MOT 同时使用这两种类型的存储。内存中存储了所有数据，同时将事务性更改持久化到磁盘，并保持频繁的定期 MOT

检查点，以确保在关机时恢复数据。

用户必须保证有足够的磁盘空间用于日志记录和检查点操作。检查点使用单独的驱动器，通过减少磁盘 I/O 负载来提高性能。

参考 MOT 关键技术了解如何在 MOT 引擎中实现持久化。

若要设置持久性：

为保证严格一致性，请在 postgresql.conf 配置文件中将参数 sync_commit 配置为 on。

MOT 的 WAL 重做日志和检查点开启持久性，下面将详细介绍。

MOT 日志记录：WAL 重做日志

为保证持久性，MOT 全面集成 GBase 8c 的 WAL 机制，通过 GBase 8c 的 XLOG 接口持久化 WAL 记录。这意味着，每次 MOT 记录的添加、更新和删除都记录在 WAL 中。确保了可以从这个非易失性日志重新生成和恢复最新的数据状态。例如，如果向表中添加了 3 行，删除了 2 行，更新了 1 行，那么日志中将记录 6 个条目。

MOT 日志记录和 GBase 8c 磁盘表的其他记录写入同一个 WAL 中。

MOT 只记录事务提交阶段的操作。

MOT 只记录更新的增量记录，以便最小化写入磁盘的数据量。

在恢复期间，从最后一个已知或特定检查点加载数据；然后使用 WAL 重做日志完成从该点开始发生的数据更改。

WAL 重做日志将保留所有表行修改，直到执行检查点（如上所述）。然后可以截断日志，以减少恢复时间和节省磁盘空间。

⚠ 注意

为确保日志 IO 设备不会成为瓶颈，日志文件必须放在具有低延迟的驱动器上。

MOT 日志类型

支持两个同步事务日志选项和一个异步事务日志选项。MOT 还支持同步的组提交日志记录与 NUMA-aware 优化，如下所述。

根据您的配置，实现以下类型的日志记录：

- 同步重做日志记录

同步重做日志记录选项是最简单、最严格的重做日志记录器。当客户端应用程序提交事

务时，事务重做条目记录在 WAL 重做日志中，如下所示：

(1) 当事务正在进行时，它存储在 MOT 内存中。

(2) 事务完成后，客户端应用程序发送 Commit 命令，该事务被锁定，然后写入磁盘上的 WAL 重做日志。当事务日志条目写入日志时，客户端应用程序仍在等待响应。

(3) 一旦事务的整个缓冲区被写入日志，就更改内存中的数据，然后提交事务。事务提交后，通知客户端应用程序事务完成。

总结

同步重做日志记录选项是最安全、最严格的，因为它确保了客户端应用程序和每个事务提交时的 WAL 重做日志条目的完全同步，从而确保了总的持久性和一致性，并且绝对不会丢失数据。此日志记录选项可防止客户端应用程序在事务尚未持久化到磁盘时将事务标记为成功的情况。

同步重做日志记录选项的缺点是，它是三个选项中最慢的日志机制。因为客户端应用程序必须等待所有数据都写入磁盘，并且磁盘写入过于频繁导致数据库变慢。

- 组同步重做日志记录

组同步重做日志记录选项与同步重做日志记录选项非常相似，它确保完全持久性，绝对不会丢失数据，并保证客户端应用程序和 WAL 重做日志条目的完全同步。不同的是，组同步重做日志记录选项将事务重做条目组同时写入磁盘上的 WAL 重做日志，而不是在提交时写入每个事务。使用组同步重做日志记录可以减少磁盘 I/O 数量，从而提高性能，特别是在运行繁重的工作负载时。

MOT 引擎通过根据运行事务的核的 NUMA 槽位自动对事务进行分组，使用 NUMA 感知优化来执行同步的组提交记录。

有关 NUMA-aware 内存访问的更多信息，请参阅 NUMA-aware 分配和亲和性。

当一个事务提交时，一组条目记录在 WAL 重做日志中，如下所示：

(1) 当事务正在进行时，它存储在内存中。MOT 引擎根据运行事务的核的 NUMA 槽位对桶中的事务进行分组。在同一槽位上运行的所有事务都被分组在一起，并且多个组将根据事务运行的核并行填充。

这样，将事务写入 WAL 更为有效，因为来自同一个槽位的所有缓冲区都一起写入磁盘。

注意：每个线程在属于单槽位的单核/CPU 上运行，每个线程只写在各自运行的核的槽位上。

(2) 在事务完成并且客户端应用程序发送 Commit 命令之后，事务重做日志条目将与同组的其他事务一起序列化。

(3) 当特定一组事务满足配置条件后，如重做日志（MOT）小节中描述的已提交的事务数或超时时间，该组中的事务将被写入磁盘的 WAL 中。当这些日志条目被写入日志时，发出提交请求的客户端应用程序正在等待响应。

(4) 一旦 NUMA-aware 组中的所有事务缓冲区都写入日志，该组中的所有事务都将对内存存储执行必要的更改，并且通知客户端这些事务已完成。

总结

组同步重做日志记录选项是一个极其安全和严格的日志记录选项，因为它确保了客户端应用程序和 WAL 重做日志条目的完全同步，从而确保了总的持久性和一致性，而且绝对不会丢失数据。此日志记录选项可防止客户端应用程序在事务尚未持久化到磁盘时将事务标记为成功的情况。

该选项的磁盘写入次数比同步重做日志记录选项少，这可能意味着它更快。缺点是事务被锁定的时间更长。在同一 NUMA 内存中的所有事务都写入磁盘的 WAL 重做日志之前，它们一直处于锁定状态。

是否使用此选项取决于事务工作负载的类型。例如，此选项有利于事务较多的系统。而对于事务少的系统而言，磁盘写入量也很少，因此不建议使用。

- 异步重做日志记录

异步重做日志记录选项是最快的日志记录方法，但是它不能确保数据不会丢失，某些仍位于缓冲区中且尚未写入磁盘的数据在电源故障或数据库崩溃时可能会丢失。当客户端应用程序提交事务时，事务重做条目将记录在内部缓冲区中，并按预先配置的时间间隔写入磁盘。客户端应用程序不会等待数据写入磁盘。它将继续进行下一个事务。这就是异步重做日志记录最快的原因。

当客户端应用程序提交事务时，事务重做条目记录在 WAL 重做日志中，如下所示：

(1) 当事务正在进行时，它存储在 MOT 内存中。

(2) 在事务完成并且客户端应用程序发送 Commit 命令后，事务重做条目将被写入内部缓冲区，但尚未写入磁盘。然后更改 MOT 数据内存，并通知客户端应用程序事务已提交。

(3) 后台运行的重做日志线程按预先配置的时间间隔收集所有缓存的重做日志条目，并将它们写入磁盘。

总结

异步重做日志记录选项是最快的日志记录选项，因为它不需要客户端应用程序等待数据写入磁盘。此外，它将许多事务重做条目分组并把它们写入一起，从而减少降低 MOT 引擎速度的磁盘 I/O 数量。

异步重做日志记录选项的缺点是它不能确保在崩溃或失败时数据不会丢失。已提交但尚未写入磁盘的数据在提交时是不持久的，因此在出现故障时无法恢复。异步重做日志记录选项对于愿意牺牲数据恢复（一致性）而不是性能的应用程序来说最为适用。

配置日志

标准 GBase 8c 磁盘引擎支持两个同步事务日志选项和一个异步事务日志选项。

配置日志记录

(1) 在 postgresql.conf 配置文件中的 sync_commit (On = Synchronous)参数中指定是否执行同步或异步事务日志记录。

(2) 在重做日志章节中的 mot.conf 配置文件里，将 enable_redo_log 参数设置为 True。

如果已选择事务日志记录的同步模式（如上文所述，synchronous_commit = on），则在 mot.conf 配置文件中的 enable_group_commit 参数中指定 Group Synchronous Redo Logging 选项或 Synchronous Redo Logging 选项。如果选择 Group Synchronous Redo Logging，必须在 mot.conf 文件中定义以下阈值，决定何时将一组事务记录在 WAL 中。

- group_commit_size：一组已提交的事务数。例如，16 表示当同一组中的 16 个事务已由它们的客户端应用程序提交时，则针对 16 个事务中的每个事务，在磁盘的 WAL 重做日志中写入一个条目。
- group_commit_timeout：超时时间，单位为毫秒。例如，10 表示在 10 毫秒之后，为同一组由客户端应用程序在最近 10 毫秒内提交的每个事务，在磁盘的 WAL 重做日志中写入一个条目。

说明

有关配置的详细信息，请参阅 MOT 持久性中的重做日志。

MOT 检查点

检查点是一个时间点。在这个时间点，表行的所有数据都保存在持久存储上的文件中，以便创建完整持久的数据库镜像。这是一个数据在某个时间点的快照。

检查点减少了为确保持久性而必须重放的 WAL 重做日志条目的数量，以此缩短数据库

的恢复时间。检查点还减少了保存所有日志条目所需的存储空间。

如果没有检查点，那么为了恢复数据库，所有 WAL 重做条目必须从开始时间进行重放，可能需要几天或几周的时间，这取决于数据库中的记录数量。检查点记录数据库的当前状态，并允许丢弃旧的重做条目。

检查点在恢复方案（特别是冷启动）中是必不可少的。首先，从最后一个已知或特定检查点加载数据；然后使用 WAL 完成此后发生的数据更改。

例如，如果同一表行被修改 100 次，则日志中将记录 100 个条目。当使用检查点后，即使表行被修改了 100 次，检查点也可以一次性记录。在记录检查点之后，可以基于该检查点执行恢复，并且只需要播放自该检查点之后发生的 WAL 重做日志条目。

12.2.5.2 MOT 恢复

MOT 恢复的主要目标是在有计划停机（例如维护）或计划外崩溃（例如电源故障后）后，将数据和 MOT 引擎恢复到一致状态。

MOT 恢复是随着 GBase 8c 数据库其余部分的恢复而自动执行的，并且完全集成到 GBase 8c 恢复过程（也称为冷启动）。

MOT 恢复包括两个阶段：

- 检查点恢复：必须通过将数据加载到内存行并创建索引，从磁盘上的最新检查点文件恢复数据。
- WAL 重做日志恢复：从检查点恢复中使用检查点后，必须通过重放之后添加到日志中的记录，从 WAL 重做日志中恢复最近的数据（在检查点中未捕获）。

  GBase 8c 管理和触发 WAL 重做日志恢复。
- 配置恢复。
- 虽然 WAL 恢复以串行方式执行，但可以将检查点恢复配置为以多线程方式运行（即由多个工作线程并行运行）。
- 在 mot.conf 文件中配置 checkpoint_recovery_workers 参数，见恢复（MOT）中的描述。

12.2.5.3 MOT 复制和高可用

由于 MOT 集成到 GBase 8c 中，并且使用或支持其复制和高可用，因此，MOT 原厂功能即支持同步复制和异步复制。

gs_ctl 工具用于可用性控制和数据库操作。这包括 gs_ctl 切换、gs_ctl 故障切换、gs_ctl 构建等。

有关配置复制和高可用性的更多信息，请参见《GBase 8c V5_3.0.x_工具参考》。

12.2.5.4 MOT 内存管理

规划和微调，请参见 MOT 内存和存储规划和 MOT 配置。

12.2.5.5 MOT VACUUM 清理

使用 VACUUM 进行垃圾收集，并有选择地分析数据库，如下所示。

- 【Postgres】

在 Postgres 中，VACUUM 用于回收死元组占用的存储空间。在正常的 Postgres 操作中，删除的元组或因更新而作废的元组不会从表中物理删除。只能由 VACUUM 清理。因此，需要定期执行 VACUUM，特别是在频繁更新的表上。

- 【MOT 扩展】

MOT 不需要周期性的 VACUUM 操作，因为新元组会重用失效元组和空元组。只有当 MOT 的大小急剧减少，并且不计划恢复到原来大小时，才需要 VACUUM 操作。

例如，应用程序定期（如每周一次）大量删除表数据的同时插入新数据，这需要几天时间，并且不一定是相同数量的行。在这种情况下，可以使用 VACUUM。

对 MOT 的 VACUUM 操作总是被转换为带有排他表锁的 VACUUM FULL。

- 支持的语法和限制

按规范激活 VACUUM 操作。

```
VACUUM [FULL | ANALYZE] [ table ];
```

只支持 FULL 和 ANALYZE VACUUM 两种类型。VACUUM 操作只能对整个 MOT 进行。

不支持以下 Postgres VACUUM 选项：

- FREEZE
- VERBOSE
- Column specification
- LAZY 模式（部分表扫描）

此外，不支持以下功能：

- AUTOVACUUM

12.2.5.6 MOT 统计

统计信息主要用于性能分析或调试。在生产环境中，通常不打开它们（默认是关闭的）。统计信息主要由数据库开发人员使用，数据库用户较少使用。

对性能有一定影响，特别是对服务器。对用户的影响可以忽略不计。

统计信息保存在数据库服务器日志中。该日志位于 data 文件夹中，命名为 postgresql-DATE-TIME.log。

有关详细的配置选项，请参阅 MOT 配置中的统计（MOT）。

12.2.5.7 MOT 监控

监控的所有语法支持基于 Postgres 的 FDW 表，包括下面的表或索引大小。此外，还存在用于监控 MOT 内存消耗的特殊函数，包括 MOT 全局内存、MOT 本地内存和单个客户端会话。

- 表和索引大小

  可以通过查询 pg_relation_size 来监控表和索引的大小。

  例如：

  数据大小

```
select pg_relation_size('customer');
```

  索引

```
select pg_relation_size('customer_pkey');
```

- MOT 全局内存详情

  检查 MOT 全局内存大小，主要是数据和索引。

```
select * from mot_global_memory_detail();
```

  结果如下。

```
numa_node    | reserved_size              | used_size
----------------+----------------+-------------
-1                  | 194716368896          | 25908215808
```

```
0                | 446693376          | 446693376
1                | 452984832          | 452984832
2                | 452984832          | 452984832
3                | 452984832          | 452984832
4                | 452984832          | 452984832
5                | 364904448          | 364904448
6                | 301989888          | 301989888
7                | 301989888          | 301989888
```

其中，-1 为总内存，0 -7 为 NUMA 内存节点。

- MOT 本地内存详情

  检查 MOT 本地内存大小，包括会话内存。

```
select * from mot_local_memory_detail();
```

结果如下。

```
numa_node    | reserved_size          | used_size
---------------+----------------+-------------
-1                | 144703488          | 144703488
0                 | 25165824           | 25165824
1                 | 25165824           | 25165824
2                 | 18874368           | 18874368
3                 | 18874368           | 18874368
4                 | 18874368           | 18874368
5                 | 12582912           | 12582912
6                 | 12582912           | 12582912
7                 | 12582912           | 12582912
```

其中，-1 为总内存，0 -7 为 NUMA 内存节点。

- 会话内存

  会话管理的内存从 MOT 本地内存中获取。

  所有活动会话（连接）的内存使用量可以通过以下查询。

```
select * from mot_session_memory_detail();
```

结果如下。

```
sessid                           | total_size | free_size | used_size
-------------------------------- - - - - - -+-----------+----------+----------
1591175063.139755603855104 | 6291456       | 1800704     | 4490752
```

其中，total_size：分配给会话的内存，free_size：未使用的内存，used_size：使用中的

内存。

DBA 可以通过以下查询确定当前会话使用的本地内存状态。

```
select * from mot_session_memory_detail()
 where sessid = pg_current_sessionid();
```

结果如下。

```
postgres=# select * from mot_session_memory_detail() where sessid = pg_current_sessionid();
          sessid          | total_size | free_size | used_size
--------------------------+------------+-----------+-----------
 1591955545.139837641324288 |    6291456 |   1861992 |   4429464
(1 row)
```

12.2.5.8 MOT 错误消息

错误可能由多种场景引起。所有错误都记录在数据库服务器日志文件中。此外，与用户相关的错误作为对查询、事务或存储过程执行或数据库管理操作的响应的一部分返回给用户。

- 服务器日志中报告的错误包括函数、实体、上下文、错误消息、错误描述和严重性。
- 向用户报告的错误被翻译成标准 PostgreSQL 错误码，可能由 MOT 特定的消息和描述组成。

错误提示、错误描述和错误码见下文。该错误码实际上是内部代码，不记录也不返回给用户。

写入日志文件的错误

所有错误都记录在数据库服务器日志文件中。以下列出了写入数据库服务器日志文件但未返回给用户的错误。该日志位于 data 文件夹中，命名为 postgresql-DATE-TIME.log。

表 13-1 只写入日志文件的错误

日志消息	内部错误代码
Error code denoting success	MOT_NO_ERROR 0
Out of memory	MOT_ERROR_OOM 1
Invalid configuration	MOT_ERROR_INVALID_CFG 2
Invalid argument passed to function	MOT_ERROR_INVALID_ARG 3
System call failed	MOT_ERROR_SYSTEM_FAILURE 4
Resource limit reached	MOT_ERROR_RESOURCE_LIMIT 5

Internal logic error	MOT_ERROR_INTERNAL 6
Resource unavailable	MOT_ERROR_RESOURCE_UNAVAILABLE 7
Unique violation	MOT_ERROR_UNIQUE_VIOLATION 8
Invalid memory allocation size	MOT_ERROR_INVALID_MEMORY_SIZE 9
Index out of range	MOT_ERROR_INDEX_OUT_OF_RANGE 10
Error code unknown	MOT_ERROR_INVALID_STATE 11

返回给用户的错误

下面列出了写入数据库服务器日志文件并返回给用户的错误。

MOT 使用返回码（Return Code，RC）返回 Postgres 标准错误代码至封装。某些 RC 会导致向正在与数据库交互的用户生成错误消息。

MOT 从内部返回 Postgres 代码（见下文）到数据库包，数据库封装根据标准的 Postgres 行为对其做出反应。

说明

提示信息中的%s、%u、%lu 指代相应的错误信息（如查询、表名或其他信息）。

- %s：字符串
- %u：数字
- %lu：数字

表 13-2 返回给用户并记录到日志文件的错误

返回给用户的短/长描述	Postgres 代码	内部错误码
Success. Denotes success	ERRCODESUCCESSFUL COMPLETION	RC_OK = 0
Failure Unknown error has occurred.	ERRCODE_FDW_ERROR	RC_ERROR = 1
Unknown error has occurred. Denotes aborted operation.	ERRCODE_FDW_ERROR	RC_ABORT

返回给用户的短/长描述	Postgres 代码	内部错误码
Column definition of %s is not supported. Column type %s is not supported yet.	ERRCODE_INVALID_COLUMN_DEFINITION	RC_UNSUPPORTED_COL_TYPE
Column definition of %s is not supported. Column type Array of %s is not supported yet.	ERRCODE_INVALID_COLUMN_DEFINITION	RC_UNSUPPORTED_COL_TYPE_ARR
Column size %d exceeds max tuple size %u. Column definition of %s is not supported.	ERRCODE_FEATURE_NOT_SUPPORTED	RC_EXCEEDS_MAX_ROW_SIZE
Column name %s exceeds max name size %u. Column definition of %s is not supported.	ERRCODE_INVALID_COLUMN_DEFINITION	RC_COL_NAME_EXCEEDS_MAX_SIZE
Column size %d exceeds max size %u. Column definition of %s is not supported.	ERRCODE_INVALID_COLUMN_DEFINITION	RC_COL_SIZE_INVLALID
Cannot create table. Cannot add column %s; as the number of declared columns exceeds the maximum declared columns.	ERRCODE_FEATURENOTSUPPORTED	RC_TABLE_EXCEEDSMAX DECLARED_COLS
Cannot create index. Total column size is greater than maximum index size %u.	ERRCODE_FDW_KEYSIZEEXCEEDS_MAX_ALLOWED	RC_INDEX_EXCEEDS_MAX_SIZE

返回给用户的短/长描述	Postgres 代码	内部错误码
Cannot create index. Total number of indexes for table %s is greater than the maximum number of indexes allowed %u.	ERRCODE_FDW_TOOMANY INDEXES	RC_TABLE_EXCEEDS_MAX_INDEXES
Cannot execute statement. Maximum number of DDLs per transaction reached the maximum %u.	ERRCODE_FDW_TOOMANY DDL_CHANGESIN TRANSACTIONNOT ALLOWED	RC_TXN_EXCEEDS_MAX_DDLS
Unique constraint violation Duplicate key value violates unique constraint \“%s\“”. Key %s already exists.	ERRCODEUNIQUE VIOLATION	RC_UNIQUE_VIOLATION
Table \“%s\” does not exist.	ERRCODE_UNDEFINED_TABLE	RC_TABLE_NOT_FOUND
Index \“%s\” does not exist.	ERRCODE_UNDEFINED_TABLE	RC_INDEX_NOT_FOUND
Unknown error has occurred.	ERRCODE_FDW_ERROR	RC_LOCAL_ROW_FOUND
Unknown error has occurred.	ERRCODE_FDW_ERROR	RC_LOCAL_ROW_NOT_FOUND
Unknown error has occurred.	ERRCODE_FDW_ERROR	RC_LOCAL_ROW_DELETED
Unknown error has occurred.	ERRCODE_FDW_ERROR	RC_INSERT_ON_EXIST
Unknown error has occurred.	ERRCODE_FDW_ERROR	RC_INDEX_RETRY_INSERT
Unknown error has occurred.	ERRCODE_FDW_ERROR	RC_INDEX_DELETE

返回给用户的短/长描述	Postgres 代码	内部错误码
Unknown error has occurred.	ERRCODE_FDW_ERROR	RC_LOCAL_ROW_NOT_VISIBLE
Memory is temporarily unavailable.	ERRCODE_OUT_OF_LOGICAL_MEMORY	RC_MEMORY_ALLOCATION_ERROR
Unknown error has occurred.	ERRCODE_FDW_ERROR	RC_ILLEGAL_ROW_STATE
Null constraint violated. NULL value cannot be inserted into non-null column %s at table %s.	ERRCODE_FDW_ERROR	RC_NULL_VIOLATION
Critical error. Critical error: %s.	ERRCODE_FDW_ERROR	RC_PANIC
A checkpoint is in progress – cannot truncate table.	ERRCODE_FDW_OPERATION_NOT_SUPPORTED	RC_NA
Unknown error has occurred.	ERRCODE_FDW_ERROR	RC_MAX_VALUE
<recovery message>	-	ERRCODE_CONFIG_FILE_ERROR
<recovery message>	-	ERRCODE_INVALIDTABLE DEFINITION
Memory engine – Failed to perform commit prepared.	-	ERRCODE_INVALIDTRANSACTION STATE
Invalid option <option name>	-	ERRCODE_FDW_INVALID OPTION NAME

返回给用户的短/长描述	Postgres 代码	内部错误码
Invalid memory allocation request size.	-	ERRCODE_INVALIDPARAMETER VALUE
Memory is temporarily unavailable.	-	ERRCODE_OUT_OFLOGICAL MEMORY
Could not serialize access due to concurrent update.	-	ERRCODE_T_RSERIALIZATION FAILURE
Alter table operation is not supported for memory table. Cannot create MOT tables while incremental checkpoint is enabled. Re-index is not supported for memory tables.	-	ERRCODE_FDW_OPERATIONNOT SUPPORTED
Allocation of table metadata failed.	-	ERRCODE_OUT_OF_MEMORY
Database with OID %u does not exist.	-	ERRCODE_UNDEFINED_DATABASE
Value exceeds maximum precision: %d.	-	ERRCODE_NUMERIC_VALUEOUT OF_RANGE
You have reached a maximum logical capacity %lu of allowed %lu.	-	ERRCODE_OUT_OFLOGICAL MEMORY

12.3 MOT 的概念

本章介绍 GBase 8c MOT 的设计和工作原理，阐明其高级特性、功能及使用方法，旨在

让读者了解 MOT 操作上的技术细节、重要特性细节和创新点。本章内容有助于决策 MOT 是否适合于特定的应用需求，以及进行最有效的使用和管理。

12.3.1 MOT 纵向扩容架构

纵向扩容即为同一台机器添加额外的核以增加算力。纵向扩容是传统上为单对控制器和多核的机器增加算力的常见形式。纵向扩容架构受限于控制器的可扩展性。

技术要求

MOT 旨在实现以下目标：

- 线性扩容：MOT 提供事务性存储引擎，利用单个 NUMA 架构服务器的所有核，以提供近线性的扩容性能。这意味着 MOT 的目标是在机器的核数和性能提升倍数之间实现直接的、近线性的关系。

  说明

  MOT 的近线性扩容效果明显优于所有现有方案，并且尽可能接近于获得最佳效果，因现有方案皆受限于硬件（如电线）的物理限制和局限性。

- 无最大核数限制：MOT 对最大核数不做任何限制。这意味着 MOT 可从单核扩展到高达 1000 秒的多核，并且新增的核退化速度最小，即便是在跨 NUMA 槽位边界的情况下。
- 极高的事务性吞吐量：MOT 提供了一个事务性存储引擎，与市场上任何其他 OLTP 供应商相比，它能够实现极高的事务性吞吐量。
- 极低的事务性时延：与市场上任何其他 OLTP 供应商相比，MOT 提供事务性存储引擎，可以达到极低的事务时延。
- 无缝集成和利用产品：MOT 事务引擎与 GBase 8c 产品标准无缝集成。通过这种方式，MOT 最大限度地重用了位于其事务性存储引擎顶部的 GBase 8c 层功能。

设计原则

为了实现上述要求（特别是在多核的环境中），我们存储引擎的体系结构实施了以下技术和策略：

- 数据和索引只存在于内存中。
- 数据和索引不用物理分区来布局（因为对于某些类型的应用程序，这些分区的性能可能会降低）。
- 事务并发控制基于乐观并发控制（OCC），没有集中的争用点。有关 OCC 的详细信息，

请参见 MOT 并发控制机制。

- 使用平行重做日志（最后单位为核）来有效避免中央锁定点。
- 使用免锁索引。有关免锁索引的详细信息，请参见 MOT 索引。
- 使用 NUMA 感知内存分配，避免跨槽位访问，特别是会话生命周期对象。有关 NUMA 感知的更多信息，请参见 NUMA-aware 分配和亲和性。
- 使用带有预缓存对象池的自定义 MOT 内存管理分配器，避免昂贵的运行时间分配和额外的争用点。这种专用的 MOT 内存分配器按需预先访问操作系统中较大的内存块，然后按需将内存分配给 MOT，使内存分配更加高效。

使用外部数据封装（FDW）进行集成

MOT 遵循并利用了 GBase 8c 的标准扩展机制——外部数据封装（FDW），如下图所示。

在 PostgreSQL 外部数据封装特性的支持下，作为其他数据源的代理的 MOT 数据库可以创建外表，如 MySQL、Oracle、PostgreSQL 等。当对外表执行查询时，FDW 将查询外部数据源并返回结果，就像查询内表一样。

GBase 8c 依赖 PostgreSQL 外部数据封装和索引支持，因此 SQL 完全覆盖，包括存储过程、用户定义函数、系统函数调用。

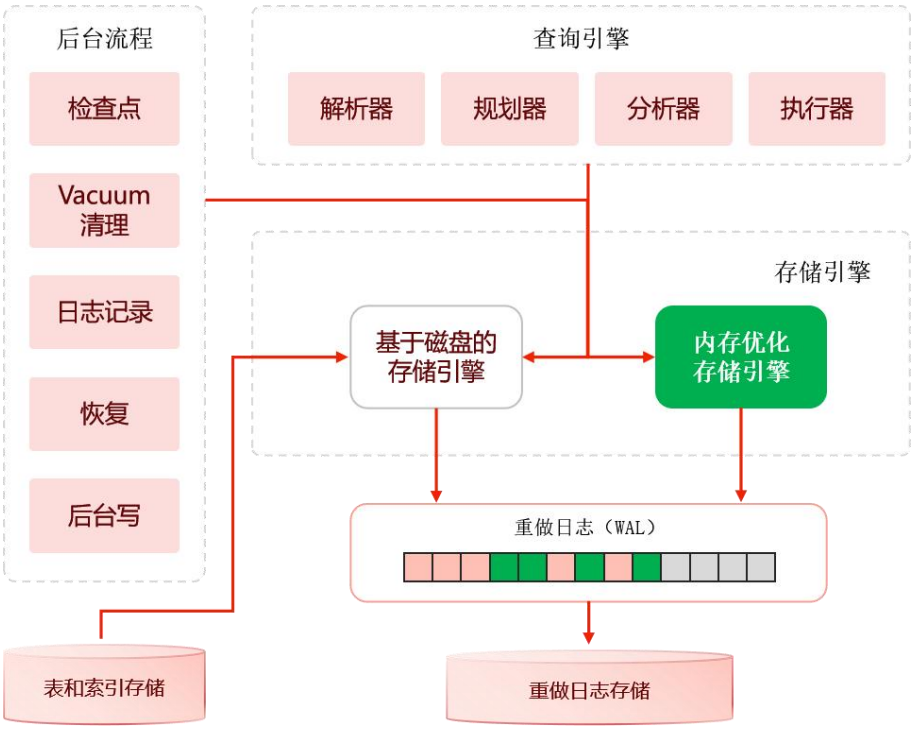


图 13-2 MOT 架构

上图中绿色表示 MOT 引擎，蓝色表示现有的 GBase 8c 组件。由此可见，FDW 在 MOT 引擎和 GBase 8c 组件之间进行中介。

与 MOT 相关的 FDW 定制

通过 FDW 集成 MOT 可以重用最上层的 GBase 8c 功能，从而显著缩短 MOT 的上市时间，同时不影响 SQL 的覆盖范围。

但是，GBase 8c 中原有的 FDW 机制并不是为存储引擎扩展而设计的，因此缺少以下基本功能：

- 查询规划阶段待计算的外表的索引感知
- 完整的 DDL 接口
- 完整的事务生命周期接口
- 检查点接口
- 重做日志接口
- 恢复接口
- 真空接口

为了支持所有缺失的功能，SQL 层和 FDW 接口层已扩展，从而为插入 MOT 事务存储引擎提供必要的基础设施。

结果：线性扩容

以下是上述 MOT 设计原则和实现的结果：

MOT 在符合 ACID 工作负载的事务吞吐量方面优于所有现有的工业级 OLTP 数据库。

12.3.2 MOT 并发控制机制

通过大量研究，我们找到了最佳的并发控制机制，结论为：基于 SILO 的 OCC 算法是 MOT 中最符合 ACID 特性的 OCC 算法。SILO 为满足 MOT 的挑战性需求提供了最好的基础。

说明

MOT 完全符合原子性、一致性、隔离性、持久性（ACID）特性，如 MOT 简介所述。

下面介绍 MOT 的并发控制机制。

12.3.3 MOT 本地内存和全局内存

SILO 管理本地内存和全局内存，如图 13-3 所示。

全局内存是所有核共享的长期内存，主要用于存储所有的表数据和索引。

本地内存是短期内存，主要由会话使用，用于处理事务及将数据更改存储到事务内存中，直到提交阶段。

当事务需要更改时，SILO 将该事务的所有数据从全局内存复制到本地内存。使用 OCC 方法，全局内存中放置的是最小的锁，因此争用时间极短。事务更改完成后，该数据从本地内存回推到全局内存中。

本地内存与 SILO 增强并发控制的基本交互式事务流如下所示：

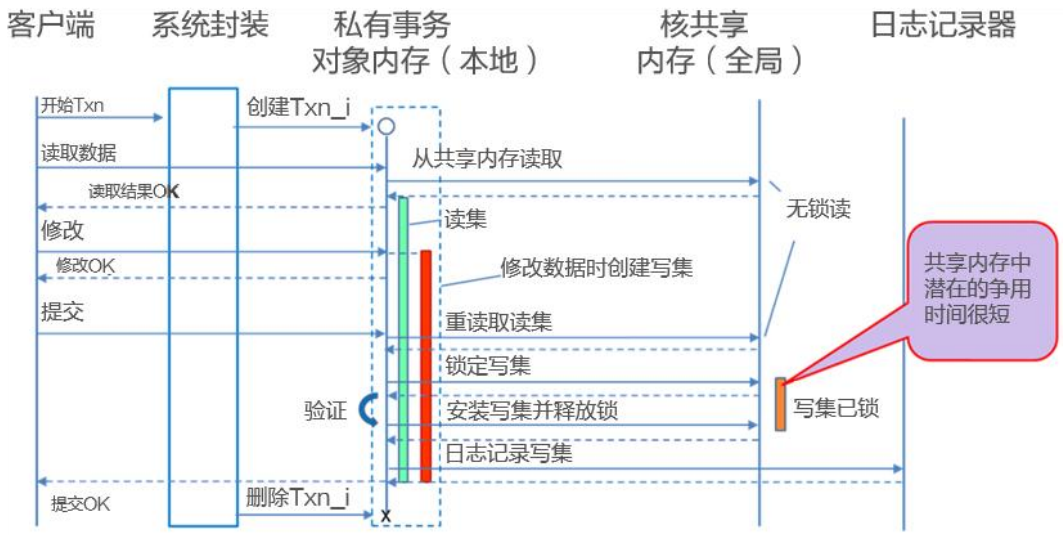


图 13-3 私有（本地）内存（每个事务）和全局内存（所有核的所有事务）

12.3.4 MOT SILO 增强特性

SILO 凭借其基本算法流程，优于我们在研究实验中测试的许多其他符合 ACID 的 OCC 算法。然而，为了使 SILO 成为产品级机制，我们必须用许多在最初设计中缺失的基本功能来增强它，例如：

- 新增对交互式事务的支持，其中事务的 SQL 运行在客户端实现，而不是作为服务器端的单个步骤运行。
- 新增乐观插入。
- 新增对非唯一索引的支持。

- 新增对事务中写后读校验（RAW）的支持，使用户能够在提交之前查看更改。
- 新增对无锁协同垃圾回收的支持。
- 新增对无锁检查点的支持。
- 新增对快速恢复的支持。
- 新增对分布式部署两阶段提交的支持。

在不破坏原始 SILO 的可扩展特性的前提下添加这些增强是非常具有挑战性的。

12.3.5 MOT 隔离级别

即使 MOT 完全兼容 ACID，GBase 8c 并非支持所有的隔离级别。下表介绍了各隔离级别，以及 MOT 支持和不支持的内容。

表 13-3 隔离级别

隔离级别	说明
READ UNCOMMITTED	MOT 不支持
READ COMMITTED	MOT 支持 READ COMMITTED（读已提交）隔离级别保证任何正在读取的数据在上一次读取时都已提交。它只是限制读者看到任何中间数据、未提交数据或脏读。数据被读取后可以自由更改，因此，读已提交隔离级别并不保证事务再次读取时能找到相同的数据。
SNAPSHOT	MOT 不支持 SNAPSHOT（快照）隔离级别提供与 SERIALIZABLE（可序列化）相同的保证，同时支持并发事务修改数据。相反，它迫使每个读者看到自己的世界版本（自己的快照）。不阻止并发更新使得编程非常容易，且可扩展性很强。然而，在许多实现中，这种隔离级别需要更高的服务器资源。
REPEATABLE READ	MOT 支持 REPEATABLE READ（可重复读）是一个更高的隔离级别，除了 READ COMMITTED 隔离级别的保证之外，它还保证任何读取的数据都不能更改。如果一个事务再次读取相同的数据，它将找出该数据，不做更改，并且保证它可读取。 乐观模型使得并发事务能更新该事务读取的行。在提交时，该

	事务将验证 REPEATABLE READ 隔离级别是否被违反。若违反，则回滚该事务，必须重试。
SERIALIZABLE	MOT 不支持 SERIALIZABLE（可序列化）隔离提供了更强的保证。除了 REPEATABLE READ 隔离级别保证的所有内容外，它还保证后续读取不会看到新数据。 它之所以被命名为 SERIALIZABLE，是因为隔离非常严格，几乎有点像事务串行运行，而不是并行运行。

下表显示了不同隔离级别启用的并发副作用。

表 13-4 隔离级别启用的并发副作用

隔离级别	说明	不可重复读	幻读
READ UNCOMMITTED	是	是	是
READ COMMITTED	否	是	是
REPEATABLE READ	否	否	是
SNAPSHOT	否	否	否
SERIALIZABLE	否	否	否

12.3.6 MOT 乐观并发控制

并发控制模块（简称 CC 模块）提供了主内存引擎的所有事务性需求。CC 模块的主要目标是为主内存引擎提供各种隔离级别的支持。

12.3.6.1 乐观 OCC 与悲观 2PL

悲观 2PL（2 阶段锁定）和乐观并发控制（OCC）的功能差异在于对事务完整性分别采用悲观和乐观方法。

基于磁盘的表使用悲观方法，这是最常用的数据库方法。MOT 引擎使用的是乐观方法。

悲观方法和乐观方法的主要功能区别在于，如果冲突发生，

- 悲观的方法会导致客户端等待；

- 而乐观方法会导致其中一个事务失败，使得客户端必须重试失败的事务。

乐观并发控制方法（MOT 使用）

乐观并发控制（OCC）方法在冲突发生时检测冲突，并在提交时执行验证检查。

乐观方法开销较小，而且通常效率更高，原因之一是事务冲突在大多数应用程序中并不常见。

当强制执行 REPEATABLE READ 隔离级别时，乐观方法与悲观方法之间的函数差异更大，而当强制执行 SERIALIZABLE 隔离级别时，函数差异最大。

悲观方法（MOT 未使用）

悲观并发控制（2PL，或称 2 阶段锁定）方法使用锁阻止在潜在冲突的发生。执行语句时应用锁，提交事务时释放锁。基于磁盘的行存储使用这种方法，并且添加了多版本并发控制（Multi-version Concurrency Control，MVCC）。

在 2PL 算法中，当一个事务正在写入行时，其他事务不能访问该行；当一个行正在读取时，其他事务不能覆盖该行。在访问时锁定每个行，以进行读写；在提交时释放锁。这些算法需要一个处理和避免死锁的方案。死锁可以通过计算等待图中的周期来检测。死锁可以通过使用 TSO 保持时序或使用某种回退方案来避免。

遇时锁定（ETL）

另一种方法是遇时锁定（ETL），它以乐观的方式处理读取，但写入操作锁定它们访问的数据。因此，来自不同 ETL 事务的写入操作相互感知，并可以决定中止。实验证明，ETL 通过两种方式提高 OCC 的性能：

- 首先，ETL 会在早期检测冲突，并通常能增加事务吞吐量。这是因为事务不会执行无用的操作。（通常）在提交时发现的冲突无法在不中止至少一个事务的情况下解决。
- 其次，ETL 写后读校验（RAW）运行高效，无需昂贵或复杂的机制。

结论：

OCC 是大多数工作负载最快的选项。这一点我们在初步研究阶段已经发现。

其中一个原因是，当每个核执行多个线程时，锁很可能被交换线程持有，特别是在交互模式下。另一个原因是悲观算法涉及死锁检测（产生开销），并通常使用读写锁（比标准自旋锁效率低）。

我们选择 Silo 是因为它比其他现有选项（如 TicToc）简单，同时对大多数工作负载保持相同的性能。ETL 有时比 OCC 更快，但它引入了假中止，可能会使用户混淆，而 OCC

则只在提交时中止。

12.3.6.2 OCC 与 2PL 的区别举例

下面是会话同时更新同一个表时，两种用户体验的区别：悲观（针对基于磁盘的表）和乐观（针对 MOT 表）。

本例中，使用如下表测试命令：

```
table “TEST” – create table test (x int, y int, z int, primary key(x));
```

本示例描述同一测试的两个方面：用户体验（本示例中的操作）和重试要求。

悲观方法示例——用于基于磁盘的表

下面是一个悲观方法例子（非 MOT）。任何隔离级别都可能适用。

以下两个会话执行尝试更新单个表的事务。

WAIT LOCK 操作发生，客户端体验是：会话 2 卡住，直到会话 1 完成 COMMIT，会话 2 才能进行。

但是，使用这种方法时，两个会话都成功，并且不会发生异常中止（除非应用了 SERIALIZABLE 或 REPEATABLE-READ 隔离级别），这会导致整个事务需要重试。

表 13-5 悲观方法代码示例

时间	会话 1	会话 2
t0	Begin	Begin
t1	update test set y=200 where x=1;	
t2	y=200	Update test set y=300 where x=1; – Wait on lock
t4	Commit	
		Unlock
		Commit (in READ-COMMITTED this will succeed, in SERIALIZABLE it will fail)
		y = 300

乐观方法示例——用于 MOT

下面是一个乐观方法的例子。

它描述了创建一个 MOT 表，然后有两个并发会话同时更新同一个 MOT 表的情况。

```
create foreign table test (x int, y int, z int, primary key(x));
```

- OCC 的优点是，在 COMMIT 之前没有锁。
- OCC 的缺点是，如果另一个会话更新了相同的记录，则更新可能会失败。如果更新失败（在所有支持的隔离级别中），则必须重试整个会话#2 事务。
- 更新冲突由内核在提交时通过版本检查机制检测。
- 会话 2 将不会等待其更新操作，并且由于在提交阶段检测到冲突而中止。

表 13-6 乐观方法代码示例——用于 MOT

时间	会话 1	会话 2
t0	Begin	Begin
t1	update test set y=200 where x=1;	
t2	y=200	Update test set y=300 where x=1;
t4	Commit	y = 300
		Commit
		ABORT
		y = 200

12.3.7 扩展 FDW 与其他特性

GBase 8c 是基于 PostgreSQL 的。但 PostgreSQL 没有内置存储引擎适配器，如 MySQL 的 handlerton。为了在 GBase 8c 数据库中集成 MOT 存储引擎，利用并扩展了现有的 FDW 机制，可以将这些外表和数据源呈现为统一、本地可访问的关系来访问外部管理的数据库。

和 PostgreSQL 不同的是，MOT 存储引擎是嵌入在 GBase 8c 数据库内部的，表由 GBase 8c 管理。GBase 8c 规划器和执行器控制表的访问。MOT 从 GBase 8c 获取日志和检查点服

务，并参与恢复过程和其他过程。

我们把正在使用或正在访问 MOT 存储引擎的所有组件称为封装。

下图显示了 MOT 存储引擎如何嵌入到数据库中，及其对数据库功能的双向访问。

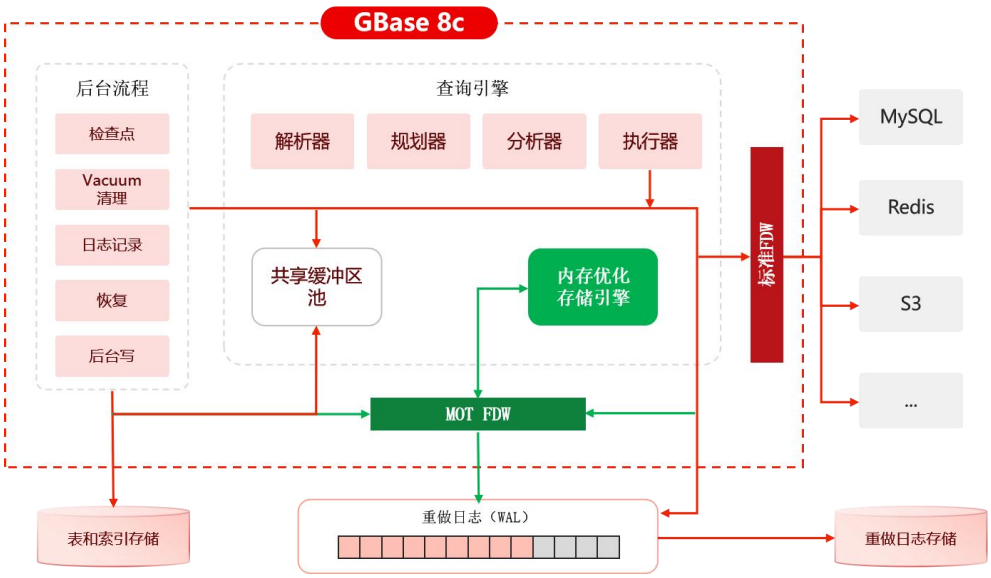


图 13-4 GBase 8c 内置 MOT 存储引擎——外部数据库的 FDW 访问

我们通过扩展和修改 FdwRoutine 结构来扩展 FDW 的能力，以便引入在 MOT 引入之前不需要的特性和调用。例如，新增了对以下功能的支持：添加索引、删除索引/表、截断、真空和表/索引内存统计。重点放在了 FdwRoutine 结构与日志、复制和检查点机制的集成，以便通过故障为跨表事务提供一致性。在这种情况下，MOT 本身有时会通过 FDW 层发起对 GBase 8c 功能的调用。

创建表和索引

为了支持 MOT 表的创建，重用了标准的 FDW 语法。

例如，创建 FOREIGN 表。

MOT DW 机制将指令传递给 MOT 存储引擎，用于实际建表。同样，我们支持创建索引（create index …）。此功能以前在 FDW 中不可用，因为表由外部管理，不需要此功能。

为了在 MOT FDW 中支持两者，ValidateTableDef 函数实际上创建了指定的表。它还处理该关系的索引创建、DROP TABLE 和 DROP INDEX 以及先前在 FDW 中不支持的 VACUUM 和 ALTER TABLE。

索引规划与执行的使用方法

查询分为两个阶段：规划和执行。在规划阶段（可能在多次执行中才出现一次），选择扫描的最佳索引。该选择基于匹配查询的 WHERE 子句、JOIN 子句和 ORDER BY 条件。在执行期间，查询迭代相关的表行，并执行各种任务，如每次迭代的更新或删除。插入是一种特殊情况——表将行添加到所有索引中，且不需要扫描。

- 规划器：在标准 FDW 中，将查询传递给外部数据源执行。这意味着索引过滤和实际规划（例如索引的选择）不在数据库中本地执行，而是在外部数据源中执行。在内部，FDW 向数据库规划器返回总体计划。MOT 表的处理方式与磁盘表类似。这意味着相关的 MOT 索引得到过滤和匹配，最小化遍历行集的索引被选择并添加到计划中。
- 执行器：查询执行器使用所选的 MOT 索引来迭代表的相关行。每个行都由 GBase 8c 封装检查，根据查询条件调用 update 或 delete 处理相应的行。

持久性、复制性和高可用性

存储引擎负责存储、读取、更新和删除底层内存和存储系统中的数据。存储引擎不处理日志、检查点和恢复，特别是因为某些事务包含多个不同存储引擎的表。因此，为了数据持久化和复制，GBase 8c 封装使用如下高可用性设施：

- 持久性：MOT 引擎通过 WAL 记录使数据持久化，WAL 记录使用 GBase 8c 的 XLOG 接口。这为 GBase 8c 提供了使用相同 API 进行复制的好处。具体请参见 MOT 持久性概念。
- 检查点设定：通过向 GBase 8c Checkpointer 注册回调来启用 MOT 检查点每当执行通用数据库检查点时，MOT 检查点也被调用。MOT 保留了检查点的日志序列号（LSN），以便与 GBase 8c 恢复对齐。MOT Checkpointing 算法是高度优化的异步算法，不会停止并发事务。具体请参见 MOT 检查点概念。
- 恢复：启动时，GBase 8c 首先调用 MOT 回调，通过加载到内存行并创建索引来恢复 MOT 检查点，然后根据检查点的 LSN 重放记录来执行 WAL 恢复。MOT 检查点使用多线程并行恢复，每个线程读取不同的数据段。这使 MOT 检查点在多核硬件上的恢复速度相当快，尽管可能比仅重放 WAL 记录的基于磁盘的表慢一些。具体请参见 MOT 恢复概念。

VACUUM 和 DROP

为了最大化 MOT 功能，我们增加了对 VACUUM、DROP TABLE 和 DROP INDEX 的支持。这三个操作都使用排他表锁执行，这意味着不允许在表上并发事务。系统 VACUUM 调用一个新的 FDW 函数执行 MOT 真空，而 ValidateTableDef()函数中增加了 DROP。

删除内存池

每个索引和表都跟踪它使用的所有内存池。DROP INDEX 命令用于删除元数据。内存池作为单个连续块被删除。MOT VACUUM 只对已使用的内存进行压缩，因为内存回收由基于 epoch 的垃圾收集器（GC）在后台持续进行。为了执行压缩，我们将索引或表切换到新的内存池，遍历所有实时数据，删除每行并使用新池插入数据，最后删除池，就像执行 DROP 那样。

查询本机编译（JIT）

MOT 引擎的 FDW 适配器还包含一个轻量级执行路径，该路径使用 LLVM 编译器执行 JIT 编译查询。有关 MOT 查询本机编译的更多信息可以在 MOT 查询原生编译（JIT）一节中找到。

12.3.8 NUMA-aware 分配和亲和性

非统一内存访问（NUMA）是一种计算机内存设计，用于多重处理，其中内存访问时间取决于内存相对于处理器的位置。处理器可以利用 NUMA 的优势，优先访问本地内存（速度更快），而不是访问非本地内存（这意味着它不会访问另一个处理器的本地内存或处理器之间共享的内存）。

MOT 内存访问设计时采用了 NUMA 感知。即 MOT 意识到内存不是统一的，而是通过访问最快和最本地的内存来获得最佳性能。

NUMA 的优点仅限于某些类型的工作负载，特别是数据通常与某些任务或用户强相关的服务器上的工作负载。

在 NUMA 平台上运行的内储存数据库系统面临一些问题，例如访问远程主内存时，时延增加和带宽降低。为了应对这些 NUMA 相关问题，NUMA 感知必须被看作是数据库系统基本架构的主要设计原则。

为了便于快速操作和高效利用 NUMA 节点，MOT 为每个表的行分配一个指定的内存池，同时为索引的节点分配一个指定的内存池。每个内存池由多个 2MB 的块组成。指定 API 从本地 NUMA 节点、来自所有节点的页面或通过轮询分配这些块，每个块在下一个节点上分配。默认情况下，共享数据池以轮询方式分配，以保持访问均衡，同时避免在不同 NUMA 节点之间拆分行。但是，线程专用内存是从一个本地节点分配的，必须验证线程始终运行在同一个 NUMA 节点中。

总结

MOT 有一个智能内存控制模块，它预先为各种类型的内存对象分配了内存池。这种智能内存控制可以提高性能，减少锁并保证稳定性。事务的内存对象分配始终是 NUMA-local，从而保证了 CPU 内存访问的最佳性能，降低时延和争用。被释放的对象返回到内存池中。在事务期间最小化使用操作系统的 malloc 函数可以避免不必要的锁。

12.3.9 MOT 索引

MOT 索引基于最先进的 Masstree 的免锁索引，用于多核系统的快速和可扩展的键值（KV）存储，通过 B+树的 Trie 实现。在多核服务器和高并发工作负载上，性能优异。它使用各种先进的技术，如乐观锁方法、缓存感知和内存预取。

在比较了各种最先进的解决方案之后，我们选择了 Masstree 作为索引，因为它显示了点查询、迭代和修改的最佳总体性能。Masstree 是 Trie 和 B+树的组合，用以谨慎利用缓存、预取、乐观导航和细粒度锁定。它针对高争用进行了优化，并对其前代产品增加了许多优化，如 OLFIT。然而，Masstree 索引的缺点是它的内存消耗更高。虽然行数据占用相同的内存大小，但每个索引（主索引或辅助索引）的每行内存平均高了 16 字节——基于磁盘的表使用基于锁的 B 树，大小为 29 字节，而 MOT 的 Masstree 大小为 45 字节。

我们的实证研究表明，成熟的免锁 Masstree 实现与我们对 Silo 的强大改进相结合，恰能为我们解决这一方面的问题。

另一个挑战是对具有多个索引的表使用乐观插入。

Masstree 索引是用于数据和索引管理的 MOT 内存布局的核心。我们的团队增强并显著改进了 Masstree，同时提交了一些关键贡献给 Masstree 开源。这些改进包括：

- 每个索引都有专用内存池：高效分配和快速索引下移
- Masstree 全球 GC：快速按需内存回收
- 具有插入键访问的大众树迭代器实现
- ARM 架构支持

MOT 的主要创新是增强了原有的 Masstree 数据结构和算法，它不支持非唯一索引（作为二级索引）。设计细节请参见非唯一索引。

MOT 支持主索引、辅助索引和无键索引 (受不支持的索引 DDL 和索引中提到的限制)。

非唯一索引

一个非唯一索引可以包含多个具有相同键的行。非唯一索引仅用于通过维护频繁使用的

数据值的排序来提高查询性能。例如，数据库可能使用非唯一索引对来自同一家庭的所有人员进行分组。但是，Masstree 数据结构实现不允许将多个对象映射到同一个键。我们用于创建非唯一索引的解决方案（如下图所示）是为映射行的键添加一个打破对称的后缀。这个添加的后缀是指向行本身的指针，该行具有 8 个字节的常量大小，并且值对该行是唯一的。当插入到非唯一索引时，哨兵的插入总是成功的，这使执行事务分配的行能够被使用。这种方法还使 MOT 能够为非唯一索引提供一个快速、可靠、基于顺序的迭代器。

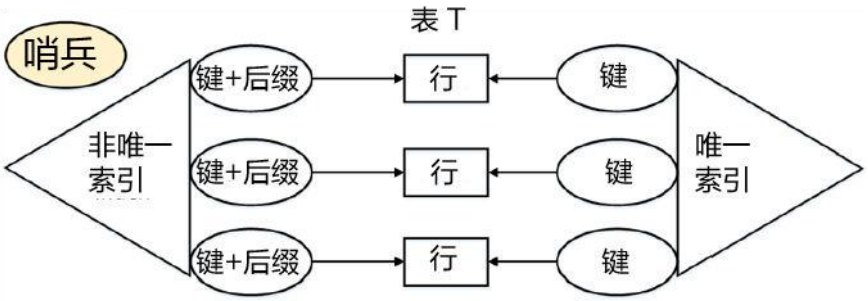


图 13-5 非唯一索引

上图描述了一个 MOT 的 T 表的结构，它有三个行和两个索引。矩形表示数据行，索引指向指向行的哨兵（椭圆形）。哨兵用键插入唯一索引，用键+后缀插入非唯一索引。哨兵可以方便维护操作，无需接触索引数据结构就可替换行。此外，在哨兵中嵌入了各种标志和参考计数，以便于乐观插入。

查找非唯一辅助索引时，会使用所需的键（如姓氏）。全串联键只用于插入和删除操作。插入和删除操作总是将行作为参数获取，从而可以创建整个键，并在执行删除或插入索引的特定行时使用它。

12.3.10MOT 持久性概念

持久性是指长期的数据保护（也称为磁盘持久性）。持久性意味着存储的数据不会遭受任何形式的退化或破坏，因此数据不会丢失或损坏。持久性可确保在有计划停机（例如维护）或计划外崩溃（例如电源故障）后数据和 MOT 引擎恢复到一致状态。

内存存储是易失的，需要电源来维护所存储的信息。另一方面，磁盘存储是非易失性的，这意味着它不需要电源来维护存储的信息，因此它不用担心停电。MOT 使用这两种类型的存储，它拥有内存中的所有数据，同时将事务性更改持久化到磁盘，并保持频繁的定期 MOT 检查点，以确保在关机时恢复数据。

用户必须保证有足够的磁盘空间用于日志记录和检查点操作。检查点使用单独的驱动器，通过减少磁盘 I/O 负载来提高性能。

有关如何在 MOT 引擎中实现持久性的概述，请参见 MOT 关键技术。

MOT 的 WAL 重做日志和检查点启用了持久性，如下所述。

12.3.10.1 MOT 日志记录：WAL 重做日志概念

概述

预写日志记录（WAL）是确保数据持久性的标准方法。WAL 的主要概念是，数据文件（表和索引所在的位置）的更改只有在记录这些更改之后才会写入，即只有在描述这些更改的日志记录被刷新到永久存储之后才会写入。

MOT 全面集成 GBase 8c 的封装日志记录设施。除持久性外，这种方法的另一个好处是能够将 WAL 用于复制目的。

支持三种日志记录方式：两种标准同步和一种异步方式。标准 GBase 8c 磁盘引擎也支持这三种日志记录方式。此外，在 MOT 中，组提交（Group-Commit）选项还提供了特殊的 NUMA 感知优化。Group-Commit 在维护 ACID 属性的同时提供最高性能。

为保证持久性，MOT 全面集成 GBase 8c 的 WAL 机制，通过 GBase 8c 的 XLOG 接口持久化 WAL 记录。这意味着，每次 MOT 记录的添加、更新和删除都记录在 WAL 中。这确保了可以从这个非易失性日志中重新生成和恢复最新的数据状态。例如，如果向表中添加了 3 个新行，删除了 2 个，更新了 1 个，那么日志中将记录 6 个条目。

- MOT 日志记录和 GBase 8c 磁盘表的其他记录写入同一个 WAL 中。
- MOT 只记录事务提交阶段的操作。
- MOT 只记录更新的增量记录，以便最小化写入磁盘的数据量。
- 在恢复期间，从最后一个已知或特定检查点加载数据；然后使用 WAL 重做日志完成从该点开始发生的数据更改。
- WAL 重做日志将保留所有表行修改，直到执行一个检查点为止（如上所述）。然后可以截断日志，以减少恢复时间和节省磁盘空间。

说明

为了确保日志 IO 设备不会成为瓶颈，日志文件必须放在具有低时延的驱动器上。

日志类型

支持两个同步事务日志选项和一个异步事务日志选项（标准 GBase 8c 磁盘引擎也支持这些选项）。MOT 还支持同步的组提交日志记录与 NUMA 感知优化，如下所述。

根据您的配置，实现以下类型的日志记录：

- 同步重做日志记录

同步重做日志记录选项是最简单、最严格的重做日志记录器。当客户端应用程序提交事务时，事务重做条目记录在 WAL 重做日志中，如下所示：

(1) 当事务正在进行时，它存储在 MOT 内存中。

(2) 事务完成后，客户端应用程序发送提交命令，该事务被锁定，然后写入磁盘上的 WAL 重做日志。这意味着，当事务日志条目写入日志时，客户端应用程序仍在等待响应。

(3) 一旦事务的整个缓冲区被写入日志，就更改内存中的数据，然后提交事务。事务提交后，客户端应用程序收到事务完成通知。

技术说明

当事务结束时，同步重做日志处理程序（SynchronousRedoLogHandler）序列化事务缓冲区，并写入 XLOG iLogger 实现。

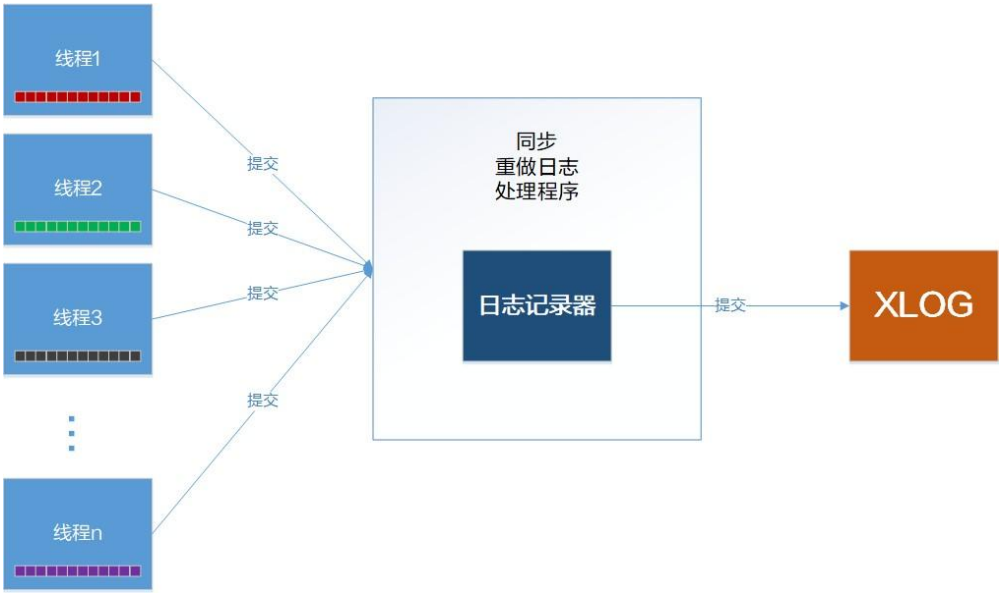


图 13-6 同步日志记录

总结

同步重做日志记录选项是最安全、最严格的，因为它确保了客户端应用程序和每个事务提交时的 WAL 重做日志条目的完全同步，从而确保了总的持久性和一致性，并且绝对不会丢失数据。此日志记录选项可防止客户端应用程序在事务尚未持久化到磁盘时将事务标记为成功的情况。

同步重做日志记录选项的缺点是，它是三个选项中最慢的日志机制。这是因为客户端应用程序必须等到所有数据都写入磁盘，并且磁盘频繁写入（这通常使数据库变慢）。

- 组同步重做日志记录

组同步重做日志记录选项与同步重做日志记录选项非常相似，因为它还确保完全持久性，绝对不会丢失数据，并保证客户端应用程序和 WAL 重做日志条目的完全同步。不同的是，组同步重做日志记录选项将事务重做条目组同时写入磁盘上的 WAL 重做日志，而不是在提交时写入每个事务。使用组同步重做日志记录可以减少磁盘 I/O 数量，从而提高性能，特别是在运行繁重的工作负载时。

MOT 引擎通过根据运行事务的核的 NUMA 槽位自动对事务进行分组，使用非统一内存访问（NUMA）感知优化来执行同步的组提交记录。

有关 NUMA 感知内存访问的更多信息，请参见 NUMA-aware 分配和亲和性。

当一个事务提交时，一组条目记录在 WAL 重做日志中，如下所示：

(1) 当事务正在进行时，它存储在内存中。MOT 引擎根据运行事务的核的 NUMA 槽位对桶中的事务进行分组。这意味着在同一槽位上运行的所有事务都被分在一组，并且多个组将根据事务运行的核心并行填充。

这样，将事务写入 WAL 更为有效，因为来自同一个槽位的所有缓冲区都一起写入磁盘。

说明

每个线程在属于单个槽位的单核/CPU 上运行，每个线程只写运行于其上的核的槽位。

(2) 在事务完成并且客户端应用程序发送 Commit 命令之后，事务重做日志条目将与属于同一组的其他事务一起序列化。

(3) 当特定一组事务满足配置条件后，如重做日志（MOT）小节中描述的已提交的事务数或超时时间，该组中的事务将被写入磁盘的 WAL 中。这意味着，当这些日志条目被写入日志时，发出提交请求的客户端应用程序正在等待响应。

(4) 一旦 NUMA-aware 组中的所有事务缓冲区都写入日志，该组中的所有事务都将对内存存储执行必要的更改，并且通知客户端这些事务已完成。

技术说明

4 种颜色分别代表 4 个 NUMA 节点。因此，每个 NUMA 节点都有自己的内存日志，允许多个连接的组提交。

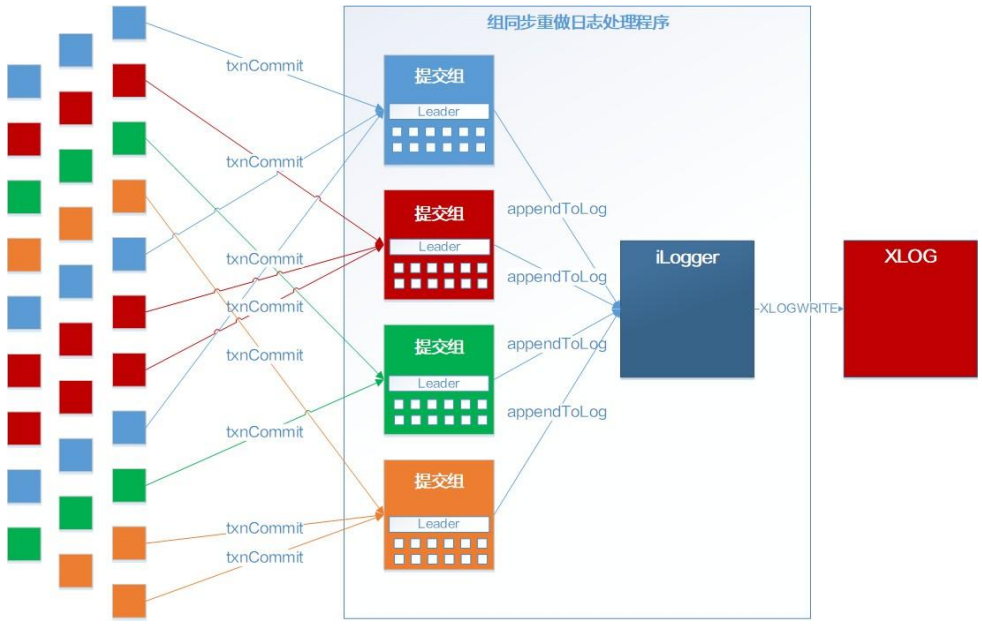


图 13-7 组提交——具有 NUMA 感知

总结

组同步重做日志记录选项是一个极其安全和严格的日志记录选项，因为它保证了客户端应用程序和 WAL 重做日志条目的完全同步，从而确保总的持久性和一致性，并且绝不会丢失数据。此日志记录选项可防止客户端应用程序在事务尚未持久化到磁盘时将事务标记为成功的情况。

一方面，该选项的磁盘写入次数比同步重做日志记录选项少，这可能意味着它更快。缺点是事务被锁定的时间更长，这意味着它们被锁定，直到同一 NUMA 内存中的所有事务都写入磁盘上的 WAL 重做日志为止。

使用此选项的好处取决于事务工作负载的类型。例如，此选项有利于事务多的系统（而对于事务少的系统而言，则较少使用，因为磁盘写入量也很少）。

- 异步重做日志记录

异步重做日志记录选项是最快的日志记录方法，但是，它不能确保数据不会丢失。也就是说，某些仍位于缓冲区且尚未写入磁盘的数据在电源故障或数据库崩溃时可能会丢失。当客户端应用程序提交事务时，事务重做条目将记录在内部缓冲区中，并按预先配置的时间间隔写入磁盘。客户端应用程序不会等待数据写入磁盘，而是继续到下一个事务。因此异步重做日志记录的速度最快。

当客户端应用程序提交事务时，事务重做条目记录在 WAL 重做日志中，如下所示：

(1) 当事务正在进行时，它存储在 MOT 内存中。

(2) 在事务完成并且客户端应用程序发送 Commit 命令后，事务重做条目将被写入内部缓冲区，但尚未写入磁盘。然后更改 MOT 数据内存，并通知客户端应用程序事务已提交。

(3) 后台运行的重做日志线程按预先配置的时间间隔收集所有缓存的重做日志条目，并将它们写入磁盘。

技术说明

在事务提交时，事务缓冲区被移到集中缓冲区（指针分配，而不是数据副本），并为事务分配一个新的事务缓冲区。一旦事务缓冲区移动到集中缓冲区，且事务线程不被阻塞，事务就会被释放。实际写入日志使用 Postgres WALWRITER 线程。当 WALWRITER 计时器到期时，它首先调用异步重做日志处理程序（通过注册的回调）来写缓冲区，然后继续其逻辑，并将数据刷新到 XLOG 中。

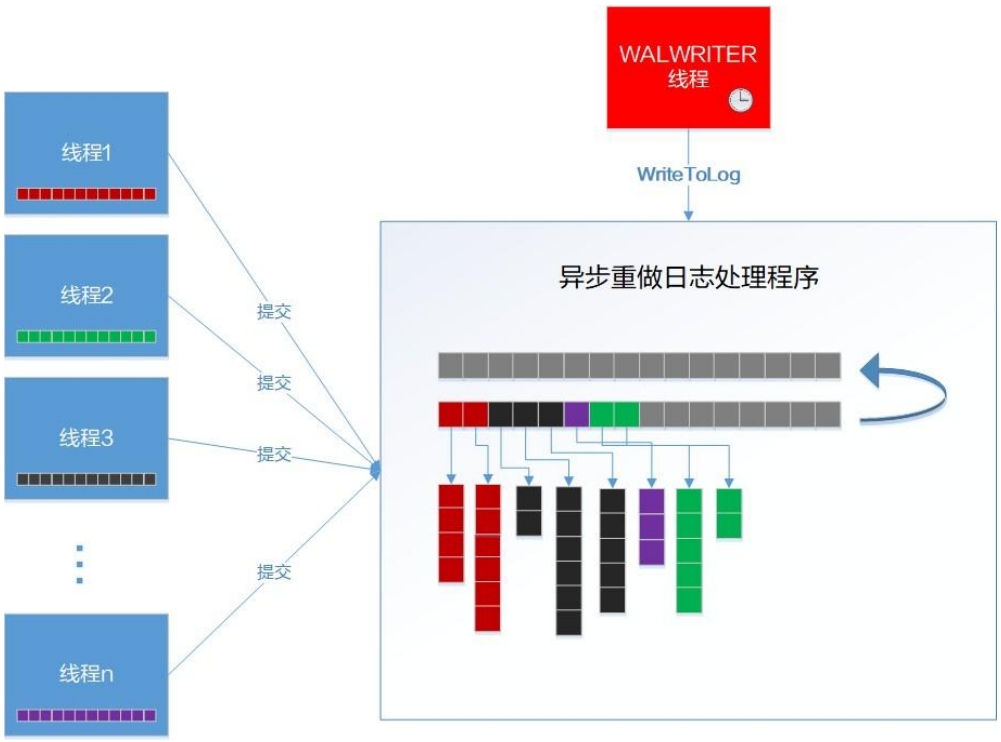


图 13-8 异步日志记录

总结

异步重做日志记录选项是最快的日志记录选项，因为它不需要客户端应用程序等待数据写入磁盘。此外，它将许多事务重做条目分组并把它们写入一起，从而减少降低 MOT 引擎速度的磁盘 I/O 数量。

异步重做日志记录选项的缺点是它不能确保在崩溃或失败时数据不会丢失。已提交但尚

未写入磁盘的数据在提交时是不持久的，因此在出现故障时无法恢复。异步重做日志记录选项对于愿意牺牲数据恢复（一致性）而不是性能的应用程序来说最为相关。

- 日志记录设计细节

下面将详细介绍内储存引擎模块中与持久化相关的各个组件的设计细节。

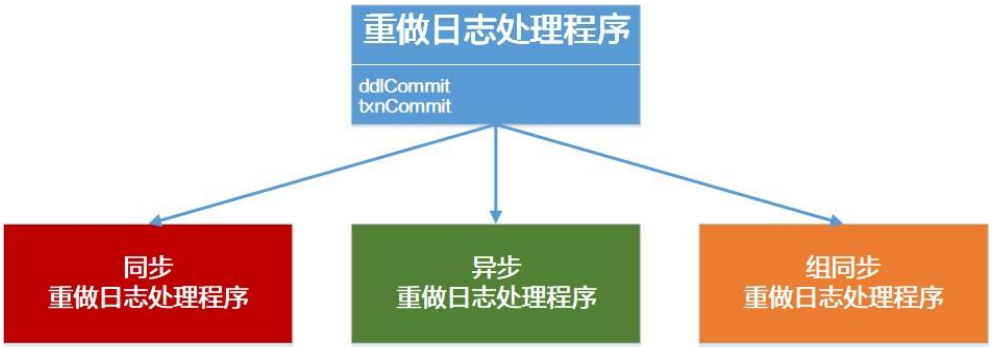


图 13-9 三种日志记录选项

重做日志组件由使用内储存引擎的后端线程和 WAL 编写器使用，以便持久化其数据。检查点通过 Checkpoint 管理器执行，由 Postgres 的 Checkpointer 触发。

- 日志记录设计概述

预写日志记录（WAL）是确保数据持久性的标准方法。WAL 的核心概念是，数据文件（表和索引所在的位置）的更改只有在记录了这些更改之后才会写入，这意味着在描述这些更改的日志记录被刷新到永久存储之后。

在内储存引擎中，我们使用现有的 GBase 8c 日志设施，并没有从头开始开发低级别的日志 API，以减少开发时间并使其可用于复制目的。

- 单事务日志记录

在内储存引擎中，事务日志记录存储在事务缓冲区中，事务缓冲区是事务对象（TXN）的一部分。在调用 addToLog()时记录事务缓冲区 -如果缓冲区超过阈值，则将其刷新并重新使用。当事务提交并通过验证阶段（OCC SILO[对比：磁盘与 MOT]验证）或由于某种原因中止时，相应的消息也会保存在日志中，以便能够在恢复期间确定事务的状态。

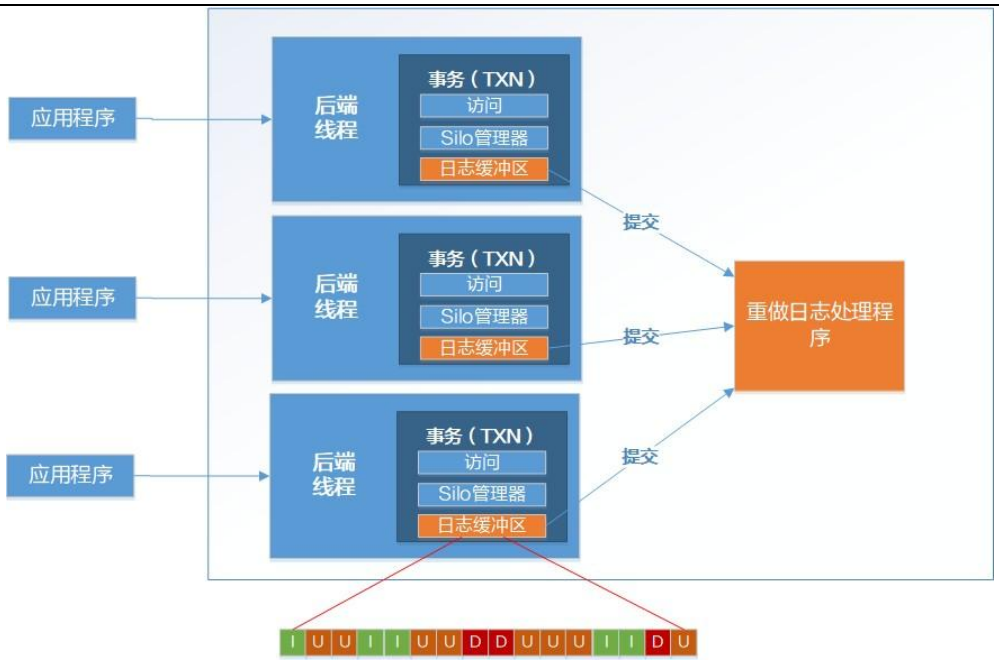


图 13-10 单事务日志记录

并行日志记录由 MOT 和磁盘引擎执行。但是，MOT 引擎通过每个事务的日志缓冲区、无锁准备和单个日志记录增强了这种设计。

- 异常处理

持久化模块通过 Postgres 错误报告基础设施（ereport）处理异常。系统日志中会记录每个错误情况的错误信息。此外，使用 Postgres 内置的错误报告基础设施将错误报告到封装。

该模块上报有如下异常：

表 13-7 异常处理

异常条件	异常码	场景描述	最终结果
WAL 写入失败	ERRCODE_FDW_ERROR	在任何情况下，WAL 写入失败	事务终止
文件 IO 错误：写入、打开等	ERRCODE_IO_ERROR	检查点：在任何文件访问错误时调用	严重：进程存在
内存不足	ERRCODE_INSUFFICIENT_RESOURCES	检查点：本地内存分配失败	严重：进程存在

逻辑、DB 错误	ERRCODEINTERNAL 错误	检查点：算法失败或无法检索表数据或索引	严重：进程存在

12.3.10.2 MOT 检查点概念

在 GBase 8c 数据库中，检查点是事务序列中一个点的快照，在该点上，可以保证堆和索引数据文件已经同步了检查点之前写入的所有信息。

在执行检查点时，所有脏数据页都会刷新到磁盘，并将一个特殊的检查点记录写入日志文件。

数据直接存储在内存中。MOT 没有像 GBase 8c 那样存储数据，因此不存在脏页的概念。为此，使用 CALC 算法，用于主内存数据库系统中的低开销异步检查点。

CALC 检查点算法：内存和计算开销低

检查点算法具有以下优点：

降低内存使用量-每条记录在任何时候最多存储两个副本。在记录处于活动且稳定版本相同或没有记录任何检查点时，仅存储记录的一个物理副本，可以最大限度地减少内存使用。

- 低开销：CALC 的开销比其他异步检查点算法小。
- 使用虚拟一致性点：CALC 不需要静默数据库以实现物理一致性点。

检查点激活

MOT 检查点被集成到 GBase 8c 的封装的检查点机制中。检查点流程可以通过执行 CHECKPOINT；命令手动触发，也可以根据封装的检查点触发设置（时间/大小）自动触发。

检查点配置在 mot.conf 文件中执行，请参见检查点（MOT）部分。

12.3.11 MOT 恢复概念

MOT 恢复模块提供了恢复 MOT 表数据所需的所有功能。恢复模块的主要目标是在计划（例如维护）关闭或计划外（例如电源故障）崩溃后，将数据和 MOT 引擎恢复到一致的状态。

GBase 8c 数据库恢复（有时也称为冷启动）包括 MOT 表，并且随着数据库其余部分的恢复而自动执行。MOT 恢复模块无缝、全面地集成到 GBase 8c 恢复过程中。

MOT 恢复有两个主要阶段：检查点恢复和 WAL 恢复（重做日志）。

MOT 检查点恢复在封装的恢复发生之前执行。仅在冷启动事件（PG 进程的启动）中执行此操作。它首先恢复元数据，然后插入当前有效检查点的所有行，这由 checkpoint_recovery_workers 并行完成，每个行都在不同的表中工作。索引在插入过程中创建。

在检查点时，表被分成多个 16MB 的块，以便多个恢复工作进程可以并行地恢复表。这样做是为了加快检查点恢复速度，它被实现为一个多线程过程，其中每个线程负责恢复不同的段。不同段之间没有依赖关系，因此线程之间没有争用，在更新表或插入新行时也不需要使用锁。

WAL 记录作为封装的 WAL 恢复的一部分进行恢复。GBase 8c 封装会迭代 XLOG，根据 xlog 记录类型执行必要的操作。如果是记录类型为 MOT 的条目，封装将它转发给 MOT 恢复管理器进行处理。如果 XLOG 条目太旧（即 XLOG 条目的 LSN 比检查点的 LSN 旧），MOT 恢复将忽略该条目。

在主备部署中，备用服务器始终处于 Recovery 状态，以便自动 WAL 恢复过程。

MOT 恢复参数在 mot.conf 文件中配置，参见 MOT 恢复。

12.3.12 MOT 查询原生编译（JIT）

MOT 使您可以在执行之前以原生格式（使用 PREPARE 语句）准备并分析预编译的完整查询。

这种本机格式以后可以更有效地执行（使用 EXECUTE 命令）。这种类型的执行效率要高得多，因为在执行期间，本机格式绕过了多个数据库处理层。这种分工避免了重复的解析分析操作。Lite Executor 模块负责执行预准备查询，其执行路径比封装执行的常规通用计划要快得多。这是通过 LLVM 使用实时（JIT）编译来实现的。此外，以伪 LLVM 的形式提供具有潜在相似性能的类似解决方案。

下面是 SQL 中的 PREPARE 语法示例：

```
PREPARE name [ ( data_type [, ...] ) ] AS statement
```

下面是一个如何在 Java 应用程序中调用 PREPARE 和 EXECUTE 语句的示例：

```
conn = DriverManager.getConnection(connectionUrl, connectionUser, connectionPassword);
// Example 1: PREPARE without bind settings
String query = "SELECT * FROM getusers";
PreparedStatement prepStmt1 = conn.prepareStatement(query);
```

```
ResultSet rs1 = pstatement.executeQuery())
while (rs1.next()) {•••}
// Example 2: PREPARE with bind settings
String sqlStmt = "SELECT * FROM employees where first_name=? and last_name like ?";
PreparedStatement prepStmt2 = conn.prepareStatement(sqlStmt);
prepStmt2.setString(1, "Mark"); // first name  “Mark”
prepStmt2.setString(2, "%n%"); // last name contains a letter  “n”
ResultSet rs2 = prepStmt2.executeQuery())
while (rs2.next()) {•••}
```

Prepare

Prepare 创建一个预处理语句。预处理语句是服务器端对象，可用于优化性能。执行 PREPARE 语句时，将解析、分析和重写指定的语句。

如果查询语句中提到的表是 MOT 表，则 MOT 编译负责对象准备，并基于 LLVM 将查询编译成 IR 字节码进行特殊优化。

每当需要新的查询编译时，都会分析查询，并使用实用程序 GsCodeGen 对象和标准 LLVM JIT API（IRBuilder）为查询生成合适的 IR 字节代码。完成字节代码生成后，代码将被 JIT 编译到单独的 LLVM 模块中。编译的代码生成一个 C 函数指针，以后可以调用该指针直接执行。请注意，这个 C 函数可以被许多线程并发调用，只要每个线程提供不同的执行上下文（详细信息如下）。每个这样的执行上下文称为“JIT 上下文”。

为了进一步提高性能，MOT JIT 对其 LLVM 代码结果应用缓存策略，使它们能够被在不同会话中的相同查询重用。

执行

当发出 EXECUTE 命令时，会计划并执行预准备语句（上文所述）。这种分工避免了重复的解析分析工作，同时使执行计划依赖于提供的特定设置值。

当生成的执行查询命令到达数据库时，它使用相应的 IR 字节代码，在 MOT 引擎中直接执行该代码，并且执行效率更高。这称为“轻量级执行”。

此外，为了可用性，Lite Executor 维护了一个预先分配的 JIT 源池。每个会话预分配自己的会话本地 JIT 上下文对象池（用于重复执行预编译查询）。

JIT 编译对比-GBase 8c 盘表与 MOT 表

目前，GBase 8c 包含针对其基于磁盘的表的 JIT/CodeGen 查询优化的两种主要形式：

- 加速表达式计算，例如在 WHERE 子句、目标列表、聚合和投影中。

- 内联小函数调用。

这些优化是局部的（从某种意义上说，它们不优化整个解释的运算符树或完全替换它），并且主要针对 CPU 绑定的复杂查询，通常在 OLAP 用例中可见。查询的执行是在使用解释运算符树的拉模型（Volcano 样式处理）中执行的。激活后，每次执行查询时都会执行编译。目前，尚未提供生成的 LLVM 代码的缓存及其跨会话和查询的重用。

相反，MOT JIT 优化为符合 MOT JIT 优化条件的整个查询提供了 LLVM 代码。结果代码用于直接执行 MOT 表，而解释的运算符模型则完全放弃。结果实际上是为整个特定查询执行生成的手写 LLVM 代码。

另一个显着的概念差异是 MOT LLVM 代码只在查询的 PREPARE 阶段为准备查询生成，而不是在查询执行时生成。由于 OLTP 查询的运行时间相当短，因此这对于 OLTP 场景尤其重要，这无法在每次查询执行期间生成代码和执行较长的查询编译时间。

最后，在 GBase 8c 中，激活 PREPARE 意味着于同一个会话中具有不同参数的执行之间重用生成的计划。同样，MOT JIT 对其 LLVM 代码结果应用了缓存策略，并扩展了缓存策略，以便在不同会话之间重用。因此，单个查询可以只编译一次，其 LLVM 代码可以在多个会话中重用，这同样有利于 OLTP 场景。

12.3.13对比：磁盘与 MOT

下表简要对比了存储引擎和 MOT 存储引擎的各种特性。

表 13-8 对比：基于磁盘与 MOT

特性	GBase 8c 磁盘存储	GBase 8c MOT 引擎
英特尔 x86+鲲鹏 ARM	是	是
SQL 和功能集覆盖率	100%	98%
纵向扩容（多核，NUMA）	低效	高效
吞吐量	高	极高
时延	低	极低
分布式	是	是
隔离级别	RC+SI	RC

	RR 序列化	RR RC+SI（V2 版本）
并发控制策略	悲观	乐观
数据容量（数据+索引）	不受限制	受限于 DRAM
本地编译	否	是
复制、恢复	是	是
复制选项	2（同步，异步）	3（同步、异步、组提交）

其中，

- RR=可重复读取
- RC=读已提交
- SI=快照隔离

12.4 附录

表 13-9 术语表

缩略语	定义描述
2PL	2 阶段锁（2-Phase Locking）
ACID	原子性（Atomicity）、一致性（Consistency）、隔离性（Isolation）、持久性（Durability）
AP	分析处理（Analytical Processing）
Arm	高级 RISC 机器（Advanced RISC Machine）、x86 的替代硬件架构
CC	并发控制（Concurrency Control）
CPU	中央处理器（Central Processing Unit）
DB	数据库（Database）
DBA	数据库管理员（Database Administrator）

DBMS	数据库管理系统（DataBase Management System）
DDL	数据定义语言（Data Definition Language）数据库模式管理语言
DML	数据修改语言（Data Modification Language）
ETL	提取、转换、加载或遇时锁定（Extract、Transform、 Load or Encounter Time Locking）
FDW	外部数据封装（Foreign Data Wrapper）
GC	垃圾收集器（Garbage Collector）
HA	高可用性（High Availability）
HTAP	事务分析混合处理（Hybrid Transactional-Analytical Processing）
IoT	物联网（Internet of Things）
IM	内储存（In-Memory）
IMDB	内储存数据库（In-Memory Database）
IR	源代码的中间表示（Intermediate Representation），用于编译和优化
JIT	准时（Just In Time）
JSON	JavaScript 对象表示法（JavaScript Object Notation）
KV	键值（Key Value）
LLVM	低级虚拟机（Low-Level Virtual Machine），指编译代码或 IR 查询
M2M	机对机（Machine-to-Machine）
ML	机器学习（Machine Learning）
MM	主内存（Main Memory）
MO	内存优化（Memory Optimized）

MOT	内存优化表存储引擎（SE），读作/em/ /oh/ /tee/
MVCC	多版本并发控制（Multi-Version Concurrency Control）
NUMA	非一致性内存访问（Non-Uniform Memory Access）
OCC	乐观并发控制（Optimistic Concurrency Control）
OLTP	在线事务处理（On-Line Transaction Processing），多用户在线交易类业务
PG	PostgreSQL
RAW	写后读校验（Reads-After-Writes）
RC	返回码（Return Code）
RTO	目标恢复时间（Recovery Time Objective）
SE	存储引擎（Storage Engine）
SQL	结构化查询语言（Structured Query Language）
TCO	总体拥有成本（Total Cost of Ownership）
TP	事务处理（Transactional Processing）
TPC-C	一种联机事务处理基准
Tpm-C	每分钟事务数-C. TPC-C 基准的性能指标，用于统计新订单事务
TVM	微小虚拟机（Tiny Virtual Machine）
TSO	分时选项（Time Sharing Option）
UDT	自定义类型
WAL	预写日志（Write Ahead Log）
XLOG	事务日志的 PostgreSQL 实现（WAL，如上文所述）

13 性能调优

13.1 总体调优思路

GBase 8c 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。

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

性能调优有时候需要重启数据库，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数据库时，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 14-1 所示。

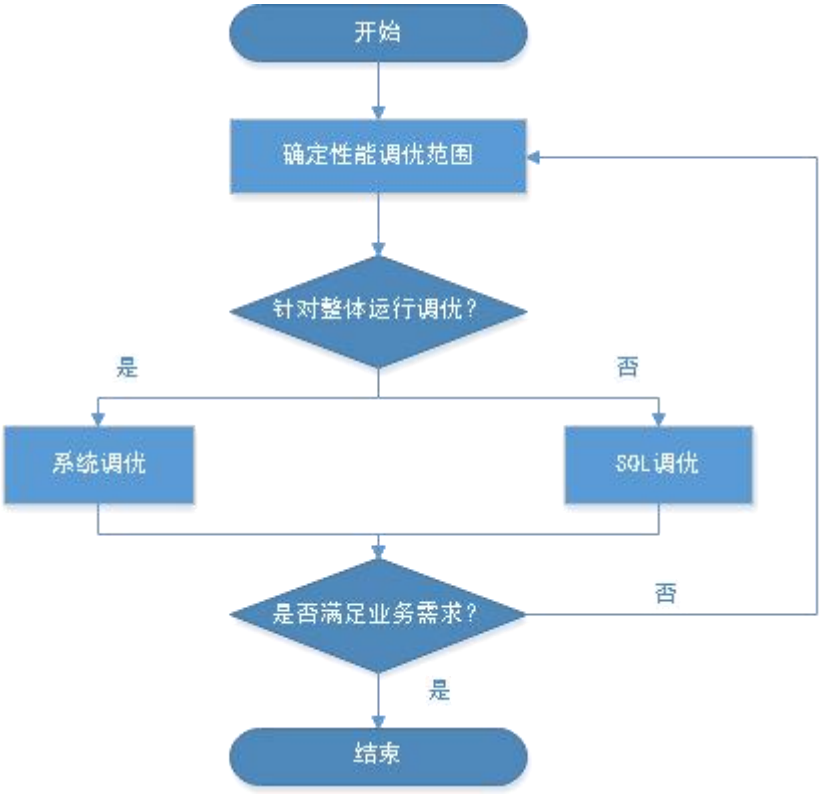


图 14-1 性能调优流程

调优各阶段说明，如表 15-14-1 所示。

表 14-1 性能调优流程说明

阶段	描述
确定性能调优范围	获取 GBase 8c 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。
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。
SQL 调优指南	审视业务所用 SQL 语句是否存在可优化空间，包括： 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。

13.2 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 GBase 8c 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 GBase 8c 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 GBase 8c 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：

  ① 检查关键的操作系统参数和数据库参数是否合理设置，进行系统调优指南。

  ② 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 SQL 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 SQL 调优指南。

- 硬件瓶颈点分析

获取 GBase 8c 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。

- 查询最耗性能的 SQL
- 分析作业是否被阻塞

13.2.1 硬件瓶颈点分析

获取 GBase 8c 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。

- CPU：通过 top 命令查看 GBase 8c 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。
- 内存：通过 top 命令查看 GBase 8c 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。
- I/O：通过 iostat、pidstat 命令或 GBase 8c 健康检查工具查看 GBase 8c 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。
- 网络：通过 sar 或 ifconfig 命令查看 GBase 8c 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。

13.2.1.1 CPU

通过 top 命令查看数据库节点 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 gbase    20   0  822m 421m 128m S     0  5.4   5:28.15 gaussdb
    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

使用“top -H”命令查看 CPU，显示内容如下所示。

```
top - 15:41:24 up 35 days,  2:53,  1 user,  load average: 0.12, 0.17, 0.20
Threads: 499 total,   1 running, 498 sleeping,   0 stopped,   0 zombie
%Cpu(s):  5.3 us,  3.0 sy,  0.0 ni, 91.7 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
KiB Mem :  8009296 total,   212276 free,  1081672 used,  6715348 buff/cache
KiB Swap:  8257532 total,  8236796 free,    20736 used.  5453088 avail Mem

  PID USER      PR  NI    VIRT    RES    SHR S %CPU %MEM     TIME+
COMMAND
    1 root      20   0  125868   3924   2348 S  2.0  0.0 601:21.01 systemd
  495 root      20   0   40236   3104   2880 S  1.7  0.0 411:22.96 systemd-jou+
 5146 gbase     20   0 7600388   1.3g 841064 S  1.7 16.5  27:43.21 ashworker
 8856 gbase     20   0  162528   2696   1600 R  1.0  0.0   0:00.06 top
 2319 gbase     20   0 2978508  42920   9288 S  0.7  0.5  10:04.47 python3
 5147 gbase     20   0 7600388   1.3g 841064 S  0.7 16.5  10:25.31 TrackStmtWo+
29995 etcd      20   0   10.5g  38984  11192 S  0.7  0.5   1:22.91 etcd
  836 dbus      20   0   60484   2440   1704 S  0.3  0.0  84:25.88 dbus-daemon
  856 root      20   0  305280   4548   3464 S  0.3  0.1  42:48.65 vmtoolsd
 1323 root      20   0  756392  32392  11120 S  0.3  0.4  10:06.16 dockerd-cur+
 1300 root      20   0  478680   4368   2744 S  0.3  0.1  85:20.07 in:imjournal
 2308 gbase     20   0 1515752  41104   9404 S  0.3  0.5   0:32.70 python3
 6068 gbase     20   0 1515752  41104   9404 S  0.3  0.5   0:00.80 python3
 5132 gbase     20   0 7600388   1.3g 841064 S  0.3 16.5   3:13.09 pagewriter
 5145 gbase     20   0 7600388   1.3g 841064 S  0.3 16.5   2:34.44 percentwork+
 5148 gbase     20   0 7600388   1.3g 841064 S  0.3 16.5   0:07.54 auditor
 5305 gbase     20   0  717620  18148   6488 S  0.3  0.2   1:44.40 node_export+
 5307 gbase     20   0  717620  18148   6488 S  0.3  0.2   1:01.56 node_export+
 5309 gbase     20   0  717620  18148   6488 S  0.3  0.2   1:55.39 node_export+
```

```
4960 gbase      20   0  717620   18148    6488 S   0.3   0.2    2:03.07 node_export+
8799 etcd       20   0   10.5g   38984   11192 S   0.3   0.5    1:27.71 etcd
```

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 gbase 的 GBase 8c 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 5085 为主，分析其为何占用 CPU 过高。

```
top - 15:43:40 up 35 days,  2:55,  1 user,  load average: 0.10, 0.14, 0.18
Threads:  33 total,   0 running,  33 sleeping,   0 stopped,   0 zombie
%Cpu(s):  4.0 us,  3.0 sy,  0.0 ni, 92.8 id,  0.2 wa,  0.0 hi,  0.0 si,  0.0 st
KiB Mem :  8009296 total,   212848 free,  1080076 used,  6716372 buff/cache
KiB Swap:  8257532 total,  8236796 free,    20736 used.  5454684 avail Mem

  PID USER      PR  NI    VIRT    RES    SHR S %CPU %MEM     TIME+ COMMAND
 5146 gbase     20   0 7600388   1.3g 841064 S  1.7 16.5  27:45.73 ashworker
 5132 gbase     20   0 7600388   1.3g 841064 S  0.7 16.5   3:13.39 pagewriter
 5147 gbase     20   0 7600388   1.3g 841064 S  0.7 16.5  10:26.25 TrackStmtWo+
 5145 gbase     20   0 7600388   1.3g 841064 S  0.3 16.5   2:34.67 percentwork+
 5085 gbase     20   0 7600388   1.3g 841064 S  0.0 16.5   0:19.22 gaussdb
 5086 gbase     20   0 7600388   1.3g 841064 S  0.0 16.5   0:00.00 jemalloc_bg+
 5097 gbase     20   0 7600388   1.3g 841064 S  0.0 16.5   0:00.00 gaussdb
 5098 gbase     20   0 7600388   1.3g 841064 S  0.0 16.5   0:02.53 syslogger
 5099 gbase     20   0 7600388   1.3g 841064 S  0.0 16.5   0:22.45 alarm
 5100 gbase     20   0 7600388   1.3g 841064 S  0.0 16.5   0:00.00 jemalloc_bg+
 5101 gbase     20   0 7600388   1.3g 841064 S  0.0 16.5   0:00.34 reaper
 5102 gbase     20   0 7600388   1.3g 841064 S  0.0 16.5   0:00.00 jemalloc_bg+
 5104 gbase     20   0 7600388   1.3g 841064 S  0.0 16.5   0:00.00 jemalloc_bg+
 5130 gbase     20   0 7600388   1.3g 841064 S  0.0 16.5   0:17.61 checkpointer
 5131 gbase     20   0 7600388   1.3g 841064 S  0.0 16.5   0:00.05 Spbgwriter
 5133 gbase     20   0 7600388   1.3g 841064 S  0.0 16.5   0:01.46 pagewriter
 5134 gbase     20   0 7600388   1.3g 841064 S  0.0 16.5   0:01.48 pagewriter
 5135 gbase     20   0 7600388   1.3g 841064 S  0.0 16.5   0:01.48 pagewriter
```

```
5136 gbase    20   0 7600388   1.3g 841064 S   0.0 16.5   0:01.47 pagewriter
5137 gbase    20   0 7600388   1.3g 841064 S   0.0 16.5   0:12.32 WALwriter
5138 gbase    20   0 7600388   1.3g 841064 S   0.0 16.5   0:00.00 WALwriteraux
5139 gbase    20   0 7600388   1.3g 841064 S   0.0 16.5   0:00.11 AVClauncher
5140 gbase    20   0 7600388   1.3g 841064 S   0.0 16.5   0:11.49 Jobscheduler
5141 gbase    20   0 7600388   1.3g 841064 S   0.0 16.5   0:00.04 asyncundola+
5142 gbase    20   0 7600388   1.3g 841064 S   0.0 16.5   0:01.80 globalstats
5143 gbase    20   0 7600388   1.3g 841064 S   0.0 16.5   0:00.09 applylaunch+
5144 gbase    20   0 7600388   1.3g 841064 S   0.0 16.5   0:12.45 statscollec+
5148 gbase    20   0 7600388   1.3g 841064 S   0.0 16.5   0:07.54 auditor
5149 gbase    20   0 7600388   1.3g 841064 S   0.0 16.5   0:08.22 2pccleaner
5150 gbase    20   0 7600388   1.3g 841064 S   0.0 16.5   0:00.13 faultmonitor
5152 gbase    20   0 7600388   1.3g 841064 S   0.0 16.5   0:01.07 undorecycler
10398 gbase   20   0 7600388   1.3g 841064 S   0.0 16.5   0:00.01 worker
10399 gbase   20   0 7600388   1.3g 841064 S   0.0 16.5   0:00.00 worker
```

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  5085
```

查询结果如下所示。

```
 Thread 33 (Thread 0x7f7311bff700 (LWP 5086)):
#0  0x00007f7313182a35 in pthread_cond_wait@@GLIBC_2.3.2 () from
/lib64/libpthread.so.0
#1  0x00005615d19bfad9 in ?? ()
#2  0x00007f731317eea5 in start_thread () from /lib64/libpthread.so.0
#3  0x00007f7312ea78dd in clone () from /lib64/libc.so.6
Thread 32 (Thread 0x7f71d9d0f700 (LWP 5097)):
#0  0x00007f73131863c1 in sigwait () from /lib64/libpthread.so.0
#1  0x00005615d0c67e23 in ?? ()
#2  0x00007f731317eea5 in start_thread () from /lib64/libpthread.so.0
#3  0x00007f7312ea78dd in clone () from /lib64/libc.so.6
Thread 31 (Thread 0x7f71cf84f700 (LWP 5098)):
#0  0x00007f7312e9cc3d in poll () from /lib64/libc.so.6
#1  0x00005615d080f4ec in WaitLatchOrSocket(Latch volatile*, int, int, long) ()
#2  0x00005615d0f6786b in SysLoggerMain(int) ()
#3  0x00005615d0f52df3 in int GaussDbThreadMain<(knl_thread_role)17>(knl_thread_arg*)
()
#4  0x00005615d0f37985 in ?? ()
#5  0x00007f731317eea5 in start_thread () from /lib64/libpthread.so.0
#6  0x00007f7312ea78dd in clone () from /lib64/libc.so.6
```

```
……
```

13.2.1.2 内存

通过 top 命令查看数据库节点内存使用情况，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 - 17:52:01 up 34 days,  5:03,  1 user,  load average: 0.09, 0.10, 0.17
Tasks: 153 total,   2 running, 151 sleeping,   0 stopped,   0 zombie
%Cpu(s):  6.9 us,  3.7 sy,  0.0 ni, 89.3 id,  0.2 wa,  0.0 hi,  0.0 si,  0.0 st
KiB Mem :  8009296 total,   181180 free,  1057004 used,  6771112 buff/cache
KiB Swap:  8257532 total,  8240124 free,    17408 used.  5478076 avail Mem

  PID USER      PR  NI    VIRT    RES    SHR S  %CPU %MEM     TIME+ COMMAND
 5085 gbase     20   0 7586052   1.2g 831648 S   6.6 16.2   9:33.11 gaussdb
    1 root      20   0  125868   3924   2348 S   2.3  0.0 574:26.02 systemd
  495 root      20   0   56620  12628  12404 S   1.7  0.2 392:14.28 systemd-jo+
 5464 gbase     20   0  845048  15168   5764 S   1.3  0.2   1:04.54 opengauss_+
20868 root      20   0  156300   6476   2368 R   1.3  0.1   0:00.04 rhsmcertd-+
29985 etcd      20   0   10.5g  34920  11156 S   1.0  0.4   1:33.19 etcd
  856 root      20   0  305280   4548   3464 S   0.7  0.1  41:40.21 vmtoolsd
 1285 root      20   0  495764  10856   9244 S   0.7  0.1 105:26.03 rsyslogd
 2319 gbase     20   0 2978508  42920   9288 S   0.7  0.5   1:21.68 python3
    9 root      20   0       0      0      0 S   0.3  0.0  13:08.82 rcu_sched
  836 dbus      20   0   60328   2356   1704 S   0.3  0.0  80:37.66 dbus-daemon
 1282 root      20   0  756392  32360  11120 S   0.3  0.4  72:05.25 dockerd-cu+
 1351 root      20   0  320276  10636   2364 S   0.3  0.1  48:02.61 docker-con+
 2180 gbase     20   0 1515752  40984   9400 S   0.3  0.5   0:29.67 python3
    2 root      20   0       0      0      0 S   0.0  0.0   0:00.61 kthreadd
```

分析时，主要关注 gaussdb 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

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  buff/cache   available
Mem:        8009296     1086448      193632     1168640     6729216
5448296
Swap:       8257532       20736     8236796
```

若用户内存占用过高，需查看执行计划，重点分析：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。

13.2.1.3 I/O

通过 iostat、pidstat 命令或健康检查工具，查看数据库节点 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 的方法主要有以下三种方式：

- 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 为采样间隔时间，10 为采样次数
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   gaussdb
```

“kB_rd/s”为每秒读取的 kB 数，“kB_wr/s”为每秒写入的 kB 数。

- 使用 gs_checkperf 工具对 GBase 8c 进行性能检查，需要以 gbase 用户登录。

```
gs_checkperf
Cluster statistics information:
     Host CPU busy time ratio                     :     2.17          %
     MPPDB CPU time % in busy time                :     3.77          %
     Shared Buffer Hit ratio                      :   100.00        %
     In-memory sort ratio                         :   100.00        %
     Physical Reads                               :      976
     Physical Writes                              :     1204
     DB size                                      :       48             MB
     Total Physical writes                        :     1204
     Active SQL count                             :      4
     Session count                                :      8
```

显示结果包括每个节点的 I/O 使用情况，物理读写次数。

也可以使用 gs_checkperf –detail 命令查询每个节点的详细性能信息。

性能参数分析

(1) 检查磁盘空间使用率，建议不要超过 60%。

```
df -T
```

(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。
- 对查询相关表做 VACUUM FULL。

```
vacuum full 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。

13.2.1.4 网络

通过 sar 或 ifconfig 命令查看数据库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:XXXXX  Bcast:XXXXX  Mask:XXXXX
              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。

13.2.2 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

(1) 以操作系统用户 gbase 登录数据库节点。

(2) 使用如下命令连接数据库。

```
gsql -d postgres -p 5432
```

postgres 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：

```
gsql ((GBase 8c V5 build a4c7bcc5) compiled at 2024-01-16 14:49:56 commit 0 last mr 1632 )
Non-SSL connection (SSL connection is recommended when requiring high-security)
Last login: 2024-01-19 16:22:27+08 from [local] (user gbase)
Type "help" for help.

postgres=#
```

(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';
```

(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。

(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  |
----------+---------+--------+
 postgres |  gbase  | idle   |
 postgres |  gbase  | 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';
```

(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。

13.2.3 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

(1) 以操作系统用户 gbase 登录数据库节点。

(2) 使用如下命令连接数据库。

```
gsql -d postgres -p 5432
```

postgres 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：

```
gsql ((GBase 8c V5 build a4c7bcc5) compiled at 2024-01-16 14:49:56 commit 0 last mr 1632 )
Non-SSL connection (SSL connection is recommended when requiring high-security)
Last login: 2024-01-19 16:22:27+08 from [local] (user gbase)
Type "help" for help.

postgres=#
```

(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。

(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

gsql 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。

13.3 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。

13.3.1 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 GBase 8c 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 15-1。

```
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 14-2 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	表示系統默认的 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。

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。通过如下命令，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，使参数生效。

```
reboot
```

13.3.2 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”。

13.3.2.1 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盘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。

13.3.2.2 数据库并发队列参数调优

数据库提供两种手段进行并发队列的控制，全局并发队列和局部并发队列。

全局并发队列

全局并发队列采用 GUC 参数 max_active_statements 控制数据库主节点上运行并发执行的作业数量。采用全局并发队列机制将控制所有普通用户的执行作业，不区分复杂度，即执行语句都将作为一个执行单元，当并发执行的作业数量达到此参数阈值时，将进入队列等待。对于管理员执行的作业，不走全局并发控制逻辑。

设置该 GUC 参数数值时，需要考虑系统的承受能力，主要关注内存的使用情况及 IO 的使用情况，综合判断。若普通用户关联资源池，并且资源池中的优先级比例不同，全局并发队列在数据库内部将使用两维队列，即优先级高低排队和同一优先级进行排队。在唤醒时，将优先唤醒高优先级队列的作业。

说明

- 在事务类大并发业务场景下，参数 max_active_statements 建议设置为-1，即不限制全局并发数。
- 在分析类查询的场景下，参数 max_active_statements 的值设置为 CPU 的核数除以数据库节点个数，一般可以设置 5~8 个。

局部并发队列

采用资源池局部并发控制机制的目的是控制在数据库主节点上同一资源池内的并发作业数量。局部并发控制机制根据执行作业的 cost，控制复杂查询的并发作业数量。

参数 parctl_min_cost 数值用于判断执行作业是否是复杂作业。

13.3.3 配置 SMP

介绍 SMP 模块的使用限制与适用场景，并给出 SMP 配置指南。

13.3.3.1 SMP 适用场景与限制

背景信息

SMP 特性通过算子并行来提升性能，同时会占用更多的系统资源，包括 CPU、内存、I/O 等等。本质上 SMP 是一种以资源换取时间的方式，在合适的场景以及资源充足的情况下，能够起到较好的性能提升效果；但是如果在不合适的场景下，或者资源不足的情况下，反而可能引起性能的劣化。SMP 特性适用于分析类查询场景，这类场景的特点是单个查询时间较长、业务并发度低。通过 SMP 并行技术能够降低查询时延，提高系统吞吐性能。然而在事务类大并发业务场景下，由于单个查询本身的时延很短，使用多线程并行技术反而会增加查询时延，降低系统吞吐性能。

适用场景

- 支持并行的算子：计划中存在以下算子支持并行。
  - Scan：支持行存普通表和行存分区表顺序扫描 、列存普通表和列存分区表顺序扫描。

- Join：HashJoin、NestLoop。
- Agg：HashAgg、SortAgg、PlainAgg、WindowAgg（只支持 partition by，不支持 order by）。
- Stream：Local Redistribute、Local Broadcast。
- 其他：Result、Subqueryscan、Unique、Material、Setop、Append、VectoRow。

- SMP 特有算子：为了实现并行，新增了并行线程间的数据交换 Stream 算子供 SMP 特性使用。这些新增的算子可以看做 Stream 算子的子类。
  - Local Gather：实现实例内部并行线程的数据汇总。
  - Local Redistribute：在实例内部各线程之间，按照分布键进行数据重分布。
  - Local Broadcast：将数据广播到实例内部的每个线程。
  - Local RoundRobin：在实例内部各线程之间实现数据轮询分发。
- 示例说明，以 TPCH Q1 的并行计划为例。

在这个计划中，实现了 Scan 以及 HashAgg 算子的并行，并新增了 Local Gather 数据交换算子。其中 3 号算子为 Local Gather 算子，上面标有的“dop: 1/4”表明该算子的发送端线程的并行度为 4，而接受端线程的并行度为 1，即下层的 4 号 HashAggregate 算子按照 4 并行度执行，而上层的 1~2 号算子按照串行执行，3 号算子实现了实例内并行线程的数据汇总。

通过计划 Stream 算子上表明的 dop 信息即可看出各个算子的并行情况。

- 非适用场景
  - 索引扫描不支持并行执行。
  - MergeJoin 不支持并行执行。
  - WindowAgg order by 不支持并行执行。
  - cursor 不支持并行执行。
  - 存储过程和函数内的查询不支持并行执行。
  - 不支持子查询 subplan 和 initplan 的并行，以及包含子查询的算子的并行。
  - 查询语句中带有 median 操作的查询不支持并行执行。
  - 带全局临时表的查询不支持并行执行。
  - 物化视图的更新不支持并行执行。

13.3.3.2 资源对 SMP 性能的影响

SMP 架构是一种利用富余资源来换取时间的方案，计划并行之后必定会引起资源消耗的增加，包括 CPU、内存、I/O 等资源的消耗都会出现明显的增长，而且随着并行度的增大，资源消耗也随之增大。当上述资源成为瓶颈的情况下，SMP 无法提升性能，反而可能导致集群整体性能的劣化。下面对各种资源对 SMP 性能的影响情况分别进行说明。

- CPU 资源

在一般客户场景中，系统 CPU 利用率不高的情况下，利用 SMP 并行架构能够更充分地利用系统 CPU 资源，提升系统性能。但当数据库服务器的 CPU 核数较少，CPU 利用率已经比较高的情况下，如果打开 SMP 并行，不仅性能提升不明显，反而可能因为多线程间的资源竞争而导致性能劣化。

- 内存资源

查询并行后会导致内存使用量的增长，但每个算子使用内存上限仍受到 work_mem 等参数的限制。假设 work_mem 为 4GB，并行度为 2，那么每个并行线程所分到的内存上限为 2GB。在 work_mem 较小或者系统内存不充裕的情况下，使用 SMP 并行后，可能出现数据下盘，导致查询性能劣化的问题。

- I/O 资源

要实现并行扫描必定会增加 I/O 的资源消耗，因此只有在 I/O 资源充足的情况下，并行扫描才能够提高扫描性能。

13.3.3.3 其他因素对 SMP 性能的影响

除了资源因素外，还有一些因素也会对 SMP 并行性能造成影响。例如分区表中分区数据不均，以及系统并发度等因素。

- 数据倾斜对 SMP 性能的影响

当数据中存在严重数据倾斜时，并行效果较差。例如某表 join 列上某个值的数据量远大于其他值，开启并行后，根据 join 列的值对该表数据做 hash 重分布，使得某个并行线程的数据量远多于其他线程，造成长尾问题，导致并行后效果差。

- 系统并发度对 SMP 性能的影响

SMP 特性会增加资源的使用，而在高并发场景下资源剩余较少。所以，如果在高并发场景下，开启 SMP 并行，会导致各查询之间严重的资源竞争问题。一旦出现了资源竞争的

现象，CPU、I/O、内存都会导致整体性能的下降。因此在高并发场景下，开启 SMP 往往不能达到性能提升的效果，甚至可能引起性能劣化。

13.3.3.4 SMP 使用建议

使用限制

想要利用 SMP 提升查询性能需要满足以下条件：

系统的 CPU、内存、I/O 和网络带宽等资源充足。SMP 架构是一种利用富余资源来换取时间的方案，计划并行之后必定会引起资源消耗的增加，当上述资源成为瓶颈的情况下，SMP 无法提升性能，反而可能导致性能的劣化。在出现资源瓶颈的情况下，建议关闭 SMP。

配置步骤

(1) 观察当前系统负载情况，如果系统资源充足（资源利用率小于 50%），执行 2；否则退出。

(2) 设置 query_dop=1（默认值），利用 explain 打出执行计划，观察计划是否符合 SMP 适用场景与限制小节中的适用场景。如果符合，进入 3。

(3) 设置 query_dop=value，不考虑资源情况和计划特征，强制选取 dop 为 1 或 value。

(4) 在符合条件的查询语句执行前设置合适的 query_dop 值，在语句执行结束后关闭 query_dop。举例如下。

```
postgres=# SET query_dop = 4;
postgres=# SELECT COUNT(*) FROM t1 GROUP BY a;
......
postgres=# SET query_dop = 1;
```

说明

- 资源许可的情况下，并行度越高，性能提升效果越好。
- SMP 并行度支持会话级设置，推荐客户在执行符合要求的查询前，打开 smp，执行结束后，关闭 smp。以免在业务峰值时，对业务造成冲击。

13.3.4 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。

13.3.4.1 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：

  - Case…when… 表达式
  - In 表达式
  - Bool 表达式（And/Or/Not）
  - BooleanTest 表达式（IS_NOT_KNOWN/IS_UNKNOWN/IS_TRUE/IS_NOT_TRUE/IS_FALSE/IS_NOT_FALSE）
  - NullTest 表达式（IS_NOT_NULL/IS_NULL）
  - Operator 表达式
  - Function 表达式（lpad、substring、btrim、rtrim、length）
  - 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：
  - Join：HashJoin
  - Agg：HashAgg
  - 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。

13.3.4.2 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。

13.3.4.3 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：

(1) 设置合理的 work_mem，在允许的条件下尽可能设置较大的 work_mem，如果出现大量数据落盘，则建议关闭 LLVM 动态编译优化（通过设置 enable_codegen=off 实现）。

(2) 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。

(3) 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。

13.4 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，确定合理的部署方案和表定义。这是数据库在多数情况下满足性能要求的基础。在合理的数据库设计和安装基础上，进行下述调优操作。

13.4.1 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 14-2 和表 15-1 所示。其中，红色字体部分为 DBA 可以介入实施调优的环节。

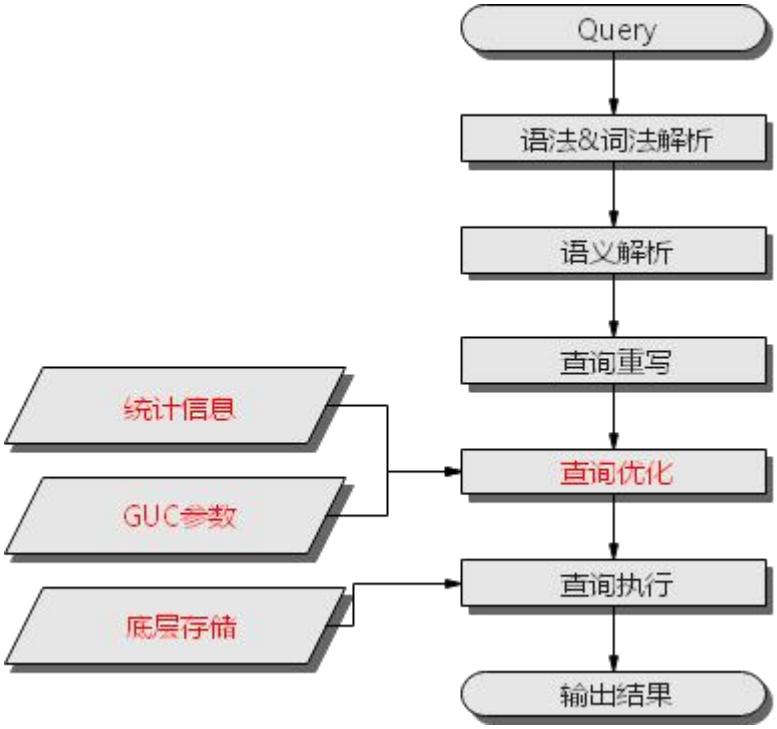


图 14-2 SQL 引擎执行查询类 SQL 语句的流程

表 14-3 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

GBase 8c 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。从上面的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。一般来说我们会通过 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 ANALYZE。

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，GBase 8c 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

GBase 8c 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。

13.4.2 SQL 执行计划介绍

13.4.2.1 SQL 执行计划概述

SQL 执行计划是一个节点树，显示 GBase 8c 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 14-3 所示。

```
openGauss=# explain select *from t1, t2 where t1.c1=t2.c2;
                              QUERY PLAN
-----------------------------------------------------------------------
 Hash Join (cost=58.35..355.67 rows=23091 width=16)
 -> 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)
```

图 14-3 SQL 执行计划示例

- 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 EXPLAIN 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。

EXPLAIN PERFORMANCE 轻量化执行方式与 EXPLAIN PERFORMANCE 保持一致，在原来的基础上减少了性能分析的时间，执行时间与 SQL 执行时间的差异显著减少。

13.4.2.2 详解

如 SQL 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。

13.4.2.2.1 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：

```
openGauss=# explain select *from t1, t2 where t1.c1=t2.c2;
                              QUERY PLAN
-----------------------------------------------------------------------
 Hash Join (cost=58.35..355.67 rows=23091 width=16)
 -> 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（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：

(1) 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，然后上层规划节点真实地从表中抓取出那些行。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

使用简单索引搜索，该方式表的数据行是以索引顺序抓取的，这样就令读取它们的开销更大，但是这里的行少得可怜，因此对行位置的额外排序并不值得。最常见的就是看到这种规划类型只抓取一行，以及那些要求 ORDER BY 条件匹配索引顺序的查询。因为那时候没有多余的排序步骤是必要的以满足 ORDER BY。

(2) 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。

(3) 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。

13.4.2.2.2 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：

```
openGauss=# 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
        (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.c2, t2.c2
            (CPU: ex c/r=0, ex row=0, ex cyc=0, inc cyc=0)
Tatol runtime: 1.087 ms
(19 rows)
```

13.4.3 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

(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。具体请参见更新统计信息。

(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 SQL 执行计划介绍。

(3) 审视和修改表定义。

(4) 针对 EXPLAIN 或 EXPLAIN PERFORMANCE 信息，定位 SQL 慢的具体原因以及改进措施，具体参见典型 SQL 调优点。

(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。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。

13.4.4 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。

```
--更新单个表的统计信息。
ANALYZE tablename;
--更新全库的统计信息。
ANALYZE;
```

使用以下命令进行多列统计信息相关操作。

```
   --收集 tablename 表的 column_1、column_2 列的多列统计信息。
ANALYZE tablename ((column_1, column_2));
--添加 tablename 表的 column_1、column_2 列的多列统计信息声明。
ALTER TABLE tablename ADD STATISTICS ((column_1, column_2));
--收集单列统计信息，并收集已声明的多列统计信息。
ANALYZE tablename;
  --删除 tablename 表的 column_1、column_2 列的多列统计信息或其声明。
ALTER TABLE tablename DELETE STATISTICS ((column_1, column_2));
```

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。

13.4.5 审视和修改表定义

13.4.5.1 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。

13.4.5.2 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）。

13.4.5.3 使用局部聚簇

局部聚簇（Partial Cluster Key）是列存下的一种技术。这种技术可以通过 min/max 稀疏

索引较快的实现基表扫描的 filter 过滤。Partial Cluster Key 可以指定多列，但是一般不建议超过 2 列。Partial Cluster Key 的选取原则：

(1) 受基表中的简单表达式约束。这种约束一般形如 col op const，其中 col 为列名，op 为操作符 =、>、>=、<=、<，const 为常量值。

(2) 尽量采用选择度比较高（过滤掉更多数据）的简单表达式中的列。

(3) 尽量把选择度比较低的约束 col 放在 Partial Cluster Key 中的前面。

(4) 尽量把枚举类型的列放在 Partial Cluster Key 中的前面。

13.4.5.4 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。

GBase 8c 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。

13.4.5.5 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。

13.4.6 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，则通过查看执行计划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。

13.4.6.1 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 PG_CONTROL_GROUP_CONFIG、GS_SESSION_MEMORY_DETAIL 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。调优方法可以参考更新统计信息和统计信息调优。

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

(1) 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"
```

(2) 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。

13.4.6.2 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。

GBase 8c 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 GBase 8c 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 GBase 8c 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)
```

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 GBase 8c 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：

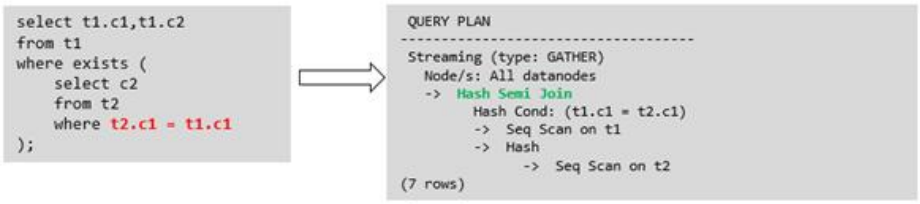
```
QUERY PLAN
-----------------------------------
Hash Join
Hash Cond: (t1.c1 = t2.c2)
->  Seq Scan on t1
->  Hash
->  HashAggregate
Group By Key: t2.c2
->  Seq Scan on t2
Filter: (c1 = 1)
(8 rows)
```

  - 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：

(1) 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)

(2) 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);
```

(3) 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 GBase 8c 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。

13.4.6.3 统计信息调优

统计信息调优介绍

GBase 8c 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。

13.4.6.4 算子级调优

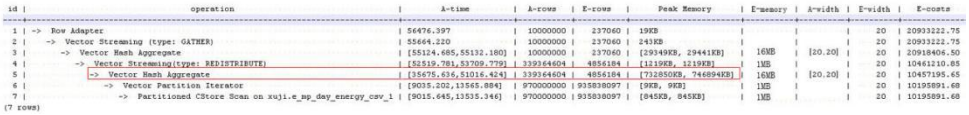
算子级调优介绍

一个查询语句要经过多个算子步骤才会输出最终的结果。由于个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。

```
postgres=#    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
postgres=#    create index idx on store_sales_row(ss_sold_date_sk);
CREATE INDEX
postgres=#    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。

```
openGauss=# 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] | 12968211302 |   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)
```

```
openGauss=#  set enable_nestloop=off;
SET
openGauss=#  set enable_mergejoin=off;
SET
openGauss=#  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。

```
openGauss=# 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)
```

```
openGauss=#  set enable_sort=off;
SET
openGauss=#  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)
```

13.4.7 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。

GBase 8c 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。

13.4.8 SQL 调优关键参数调整

本节将介绍影响 SQL 调优性能的关键数据库主节点配置参数，配置方法参见配置运行参数。

表 14-4 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。

13.4.9 使用 Plan Hint 进行调优

13.4.9.1 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 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，为外层查询的 hint，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)              ,
```

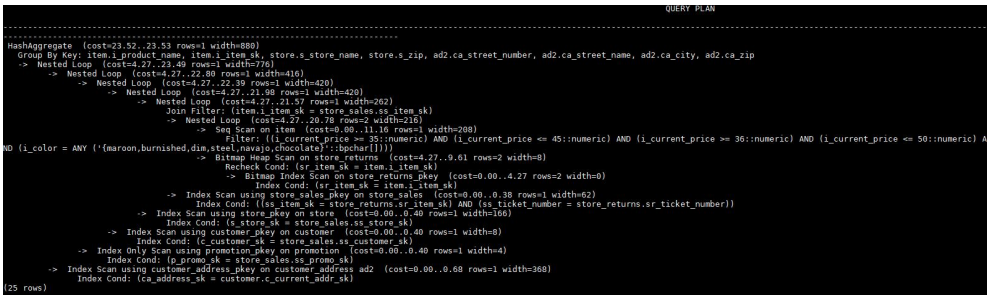
```
    s_company_id              integer               ,
    s_company_name             varchar(50)          ,
    s_street_number           varchar(10)           ,
    s_street_name             varchar(60)           ,
    s_street_type             char(15)              ,
    s_suite_number            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
```



13.4.9.2 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：t2、t3 先 join，t2 做内表；然后再和 t1 join，

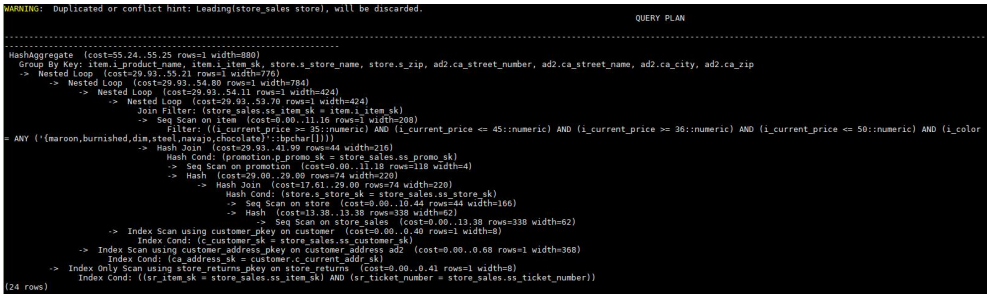
t2、t3 的 join 表做内表；然后再依次跟 t4、t5 做 join，t4、t5 做内表。

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：



图中计划顶端 warning 的提示详见 Hint 的错误、冲突及告警的说明。

13.4.9.3 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：

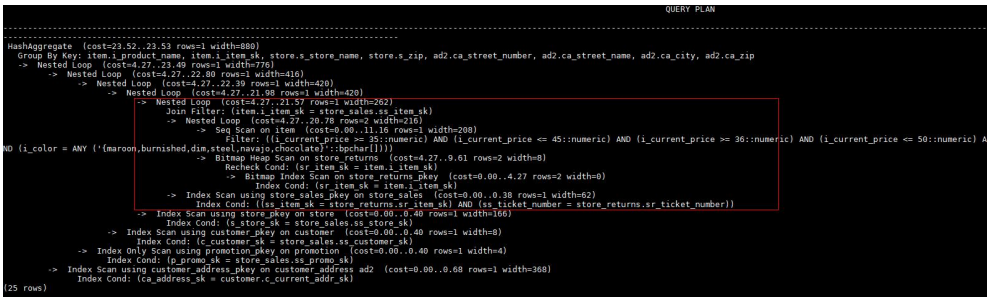
no nestloop(t1 t2 t3)表示：生成 t1、t2、t3 三表连接计划时，不使用 nestloop。三表连接计划可能是 t2 t3 先 join，再跟 t1 join，或 t1 t2 先 join，再跟 t3 join。此 hint 只 hint 最后一次 join 的 join 方式，对于两表连接的方法不 hint。如果需要，可以单独指定，例如：任意表均不允许 nestloop 连接，且希望 t2 t3 先 join，则增加 hint：no nestloop(t2 t3)。

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：



13.4.9.4 Join 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 Join 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。

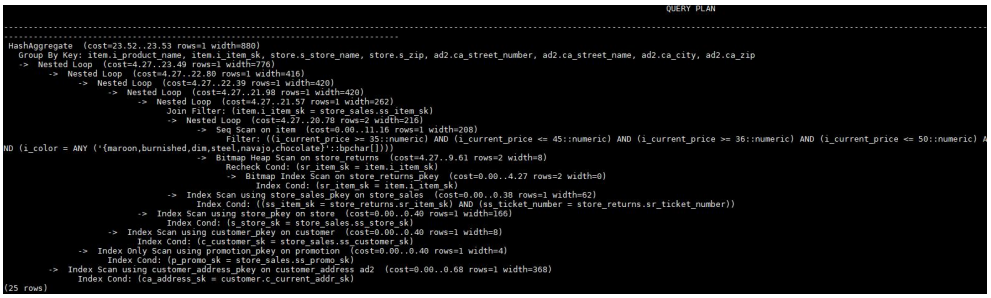
rows hint 支持在单表、多表、function table 及 subquery scan table 的结果集上指定 hint。

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：



13.4.9.5 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

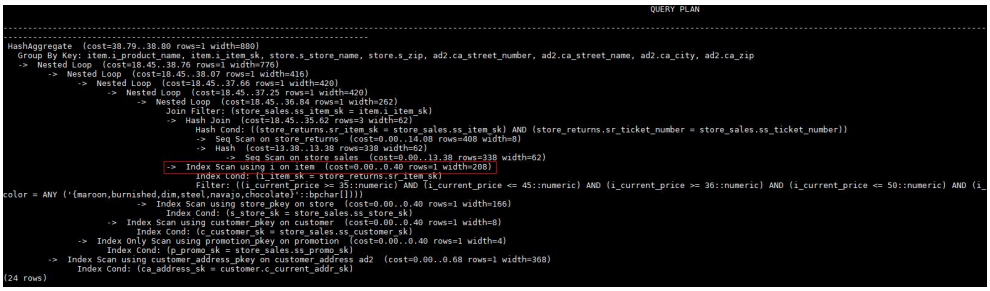
为了 hint 使用索引扫描，需要首先在表 item 的 i_item_sk 列上创建索引，名称为 i。

```
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：



13.4.9.6 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没有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)
```

13.4.9.7 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。

13.4.9.8 优化器 GUC 参数的 Hint

功能描述

设置本次查询执行内生效的查询优化相关 GUC 参数。hint 的推荐使用场景可以参考各 guc 参数的说明，此处不作赘述。

语法格式

```
set(param value)
```

参数说明

- param 表示参数名。
- value 表示参数的取值。
- 目前支持使用 Hint 设置生效的参数有
  - 布尔类：

enable_bitmapscan 、 enable_hashagg 、 enable_hashjoin 、 enable_indexscan 、 enable_indexonlyscan 、 enable_material 、 enable_mergejoin 、 enable_nestloop 、 enable_index_nestloop、enable_seqscan、enable_sort、enable_tidscan

  - 整形类：

query_dop

  - 浮点类：

cost_weight_index、default_limit_rows、seq_page_cost、random_page_cost、cpu_tuple_cost、cpu_index_tuple_cost、cpu_operator_cost、effective_cache_size

- 枚举类型：

try_vector_engine_strategy

说明

- 设置不在白名单中的参数，参数取值不合法，或 hint 语法错误时，不会影响查询执行的正确性。使用 explain(verbose on)执行可以看到 hint 解析错误的报错提示。
- GUC 参数的 hint 只在最外层查询生效——子查询内的 GUC 参数 hint 不生效。
- 视图定义内的 GUC 参数 hint 不生效。
- CREATE TABLE … AS … 查询最外层的 GUC 参数 hint 可以生效。

13.4.9.9 Custom Plan 和 Generic Plan 选择的 Hint

功能描述

对于以 PBE 方式执行的查询语句和 DML 语句，优化器会基于规则、代价、参数等因素选择生成 Custom Plan 或 Generic Plan 执行。用户可以通过 use_cplan/use_gplan 的 hint 指定使用哪种计划执行方式。

语法格式

指定使用 Custom Plan：

```
use_cplan
```

指定使用 Generic Plan：

```
use_gplan
```

说明

对于非 PBE 方式执行的 SQL 语句，设置本 hint 不会影响执行方式。 - 本 Hint 的优先级仅高于基于代价的选择和 plan_cache_mode 参数，即 plan_cache_mode 无法强制选择执行方式的语句本 hint 也无法生效。

示例

强制使用 Custom Plan

```
create table t (a int, b int, c int);
prepare p as select /*+ use_cplan */ * from t where a = $1;
explain execute p(1);
```

计划如下。可以看到过滤条件为入参的实际值，即此计划为 Custom Plan。

```
                          QUERY PLAN
-----------------------------------------------------------
 Seq Scan on t  (cost=0.00..34.31 rows=10 width=12)
   Filter: (a = 1)
(2 rows)
```

强制使用 Generic Plan

```
deallocate p;
prepare p as select /*+ use_gplan */ * from t where a = $1;
explain execute p(1);
```

计划如下。可以看到过滤条件为待填充的入参，即此计划为 Generic Plan。

```
                          QUERY PLAN
-----------------------------------------------------------
 Seq Scan on t  (cost=0.00..34.31 rows=10 width=12)
   Filter: (a = $1)
(2 rows)
```

13.4.9.10 指定子查询不展开的 Hint

功能描述

数据库在对查询进行逻辑优化时通常会将可以提升的子查询提升到上层来避免嵌套执行，但对于某些本身选择率较低且可以使用索引过滤访问页面的子查询，嵌套执行不会导致性能下降过多，而提升之后扩大了查询路径的搜索范围，可能导致性能变差。对于此类情况，可以使用 no_expand Hint 进行调试。大多数情况下不建议使用此 hint。

语法格式

```
no_expand
```

示例

正常的查询执行

```
explain select * from t1 where t1.a in (select t2.a from t2);
```

计划

```
                                QUERY PLAN
-------------------------------------------------------------------------
 Hash Join  (cost=38.81..92.58 rows=972 width=12)
   Hash Cond: (t1.a = t2.a)
   ->  Seq Scan on t1  (cost=0.00..29.45 rows=1945 width=12)
   ->  Hash  (cost=36.31..36.31 rows=200 width=4)
         ->  HashAggregate  (cost=34.31..36.31 rows=200 width=4)
               Group By Key: t2.a
               ->  Seq Scan on t2  (cost=0.00..29.45 rows=1945 width=4)
(7 rows)
```

加入 no_expand

```
explain select * from t1 where t1.a in (select /*+ no_expand*/ t2.a from t2);
```

计划

```
                              QUERY PLAN
-----------------------------------------------------------------------
 Seq Scan on t1  (cost=34.31..68.62 rows=972 width=12)
   Filter: (hashed SubPlan 1)
   SubPlan 1
     ->  Seq Scan on t2  (cost=0.00..29.45 rows=1945 width=4)
(4 rows)
```

13.4.9.11 指定不使用全局计划缓存的 Hint

功能描述

全局计划缓存打开时，可以通过 no_gpc Hint 来强制单个查询语句不在全局共享计划缓存，只保留会话生命周期的计划缓存。

语法格式

```
no_gpc
```

说明

本参数仅在 enable_global_plancache=on 时对 PBE 执行的语句生效。

示例

dbe_perf.global_plancache_status 视图中无结果即没有计划被全局缓存。

13.4.9.12 同层参数化路径的 Hint

功能描述

通过 predpush_same_level Hint 来指定同层表或物化视图之间参数化路径生成。

语法格式

```
predpush_same_level(src, dest)
predpush_same_level(src1 src2 ..., dest)
```

说明

本参数仅在 rewrite_rule 中的 predpushforce 选项打开时生效。

示例

准备参数和表及索引：

```
postgres=# set rewrite_rule = 'predpushforce';
SET
postgres=# create table t1(a int, b int);
CREATE TABLE
postgres=# create table t2(a int, b int);
CREATE TABLE
postgres=# create index idx1 on t1(a);
CREATE INDEX
postgres=# create index idx2 on t2(a);
CREATE INDEX
```

执行语句查看计划：

```
postgres=# explain select * from t1, t2 where t1.a = t2.a;
                                QUERY PLAN
-------------------------------------------------------------------
 Hash Join  (cost=27.50..56.25 rows=1000 width=16)
   Hash Cond: (t1.a = t2.a)
   ->  Seq Scan on t1  (cost=0.00..15.00 rows=1000 width=8)
   ->  Hash  (cost=15.00..15.00 rows=1000 width=8)
         ->  Seq Scan on t2  (cost=0.00..15.00 rows=1000 width=8)
(5 rows)
```

可以看到 t1.a = t2.a 条件过滤在 Join 上面，此时可以通过 predpush_same_level(t1, t2)将条件下推至 t2 的扫描算子上：

```
postgres=# explain select /*+predpush_same_level(t1, t2)*/ * from t1, t2 where t1.a = t2.a;
                                QUERY PLAN
-------------------------------------------------------------------
 Nested Loop  (cost=0.00..335.00 rows=1000 width=16)
   ->  Seq Scan on t1  (cost=0.00..15.00 rows=1000 width=8)
   ->  Index Scan using idx2 on t2  (cost=0.00..0.31 rows=1 width=8)
         Index Cond: (a = t1.a)
(4 rows)
```

须知

predpush_same_level 可以指定多个 src，但是所有的 src 必须在同一个条件中。

如果指定的 src 和 dest 条件不存在，或该条件不符合参数化路径要求，则本 hint 不生效。

13.5 使用向量化执行引擎进行调优

GBase 8c 数据库支持行执行引擎和向量化执行引擎，分别对应行存表和列存表。

- 一次一个 batch，读取更多数据，节省 IO。
- batch 中记录较多，CPU cache 命中率提升。
- Pipeline 模式执行，函数调用次数少。
- 一次处理一批数据，效率高。

GBase 8c 数据库所以对于分析类的复杂查询能够获得更好的查询性能。但列存表在数据插入和数据更新上表现不佳，对于存在数据频繁插入和更新的业务无法使用列存表。

为了提升行存表在分析类的复杂查询上的查询性能，GBase 8c 数据库提供行存表使用向量化执行引擎的能力。通过设置 GUC 参数 try_vector_engine_strategy，可以将包含行存表的查询语句转换为向量化执行计划执行。

行存表转换为向量化执行引擎执行不是对所有的查询场景都适用。参考向量化引擎的优势，如果查询语句中包含表达式计算、多表 join、聚集等操作时，通过转换为向量化执行能够获得性能提升。从原理上分析，行存表转换为向量化执行，会产生转换的开销，导致性能下降。而上述操作的表达式计算、join 操作、聚集操作转换为向量化执行之后，能够获得获得性能提升。所以查询转换为向量化执行后，性能是否提升，取决于查询转换为向量化之后获得的性能提升能否高于转换产生的性能开销。

以 TPCH Q1 为例，使用行执行引擎时，扫描算子的执行时间为 405210ms，聚集操作的执行时间为 2618964ms；而转换为向量化执行引擎后，扫描算子（SeqScan + VectorAdapter）的执行时间为 470840ms，聚集操作的执行时间为 212384ms，所以查询能够获得性能提升。

TPCH Q1 行执行引擎执行计划：

```
                                                                                                QUERY
PLAN
------------------------------------------------------------------------------------------------------------
------------------------
  Sort   (cost=43539570.49..43539570.50 rows=6 width=260) (actual
time=3024174.439..3024174.439 rows=4 loops=1)
     Sort Key: l_returnflag, l_linestatus
     Sort Method: quicksort   Memory: 25kB
     ->  HashAggregate   (cost=43539570.30..43539570.41 rows=6 width=260) (actual
time=3024174.396..3024174.403 rows=4 loops=1)
```

```
            Group By Key: l_returnflag, l_linestatus
            ->  Seq Scan on lineitem  (cost=0.00..19904554.46 rows=590875396 width=28)
(actual time=0.016..405210.038 rows=596140342 loops=1)
                  Filter: (l_shipdate <= '1998-10-01 00:00:00'::timestamp without time zone)
                  Rows Removed by Filter: 3897560
 Total runtime: 3024174.578 ms
(9 rows)
TPCH Q1 向量化执行引擎执行计划:

 QUERY PLAN
------------------------------------------------------------------------------------------------------------------------
-----------------------------------------------
 Row Adapter  (cost=43825808.18..43825808.18 rows=6 width=298) (actual
time=683224.925..683224.927 rows=4 loops=1)
   ->  Vector Sort  (cost=43825808.16..43825808.18 rows=6 width=298) (actual
time=683224.919..683224.919 rows=4 loops=1)
         Sort Key: l_returnflag, l_linestatus
         Sort Method: quicksort  Memory: 3kB
         ->  Vector Sonic Hash Aggregate  (cost=43825807.98..43825808.08 rows=6
width=298) (actual time=683224.837..683224.837 rows=4 loops=1)
               Group By Key: l_returnflag, l_linestatus
               ->  Vector Adapter(type: BATCH MODE)
(cost=19966853.54..19966853.54 rows=596473861 width=66) (actual time=0.982..470840.274
rows=596140342 loops=1)
                     Filter: (l_shipdate <= '1998-10-01 00:00:00'::timestamp without
time zone)
                     Rows Removed by Filter: 3897560
                     ->  Seq Scan on lineitem  (cost=0.00..19966853.54
rows=596473861 width=66) (actual time=0.364..199301.737 rows=600037902 loops=1)
 Total runtime: 683225.564 ms
(11 rows)
```

13.6 TPCC 性能调优测试指导

13.6.1 概述

本章节主要介绍 GBase 8c 数据库为了达到最佳 TPMC 性能，所依赖的关键系统级调优。

13.6.2 搭建性能测试环境

13.6.2.1 硬件要求

服务器数量：2 台服务器（1 台用于数据库服务器，1 台用于 TPCC 客户端）。

服务器硬件规格：

- Memory：大于 512GB。
- Free Disk：4 块 NVME 硬盘，每块容量大于 1TB。
- 网卡：Hi1822 千兆网卡，光纤互连。

13.6.2.2 软件要求

操作系统

操作系统要求：aarch64 架构

软件依赖

表 14-5 软件依赖要求

所需软件	建议版本
numactl	–
jdk	1.8.0-242
ant	1.10.5
htop	–

13.6.2.3 前置软件安装

- 安装 jdk。

  a.下载并安装 jdk。

  b. 配置环境变量，具体如下：

```
export JAVA_HOME=your_path/jdk1.8.0_232
export JRE_HOME=${JAVA_HOME}/jre
export CLASSPATH=.:${JAVA_HOME}/lib:${JRE_HOME}/jre
```

```
export PATH=${PATH}:${JAVA_HOME}/bin
export JAVA_TOOL_OPTIONS=-Dfile.encoding=UTF8
```

- 安装 numactl。

  执行如下命令安装 numactl。

```
yum install –y numactl
```

- 安装 ant。

  a. 执行如下命令安装 ant。

```
yum install –y ant
```

  b. 设置环境变量，修改文件/etc/profile，添加下面两行。

```
export ANT_HOME=/usr/share/ant/
export PATH=${PATH}:${ANT_HOME}/bin
```

  说明

  可能对于不同的操作系统 ANT_HOME 的安装路径不一样，可以通过 find 命令找打其对应的安装路径。

- 安装 htop 工具，具体请参见 https://hisham.hm/htop/。

13.6.2.4 BIOS 配置

- 恢复 BIOS 出厂设置。

  修改相关 BIOS 设置，如下所示：

  a. BIOS>Advanced>MISC Config，配置 Support Smmu 为 Disabled，如图 1 所示。

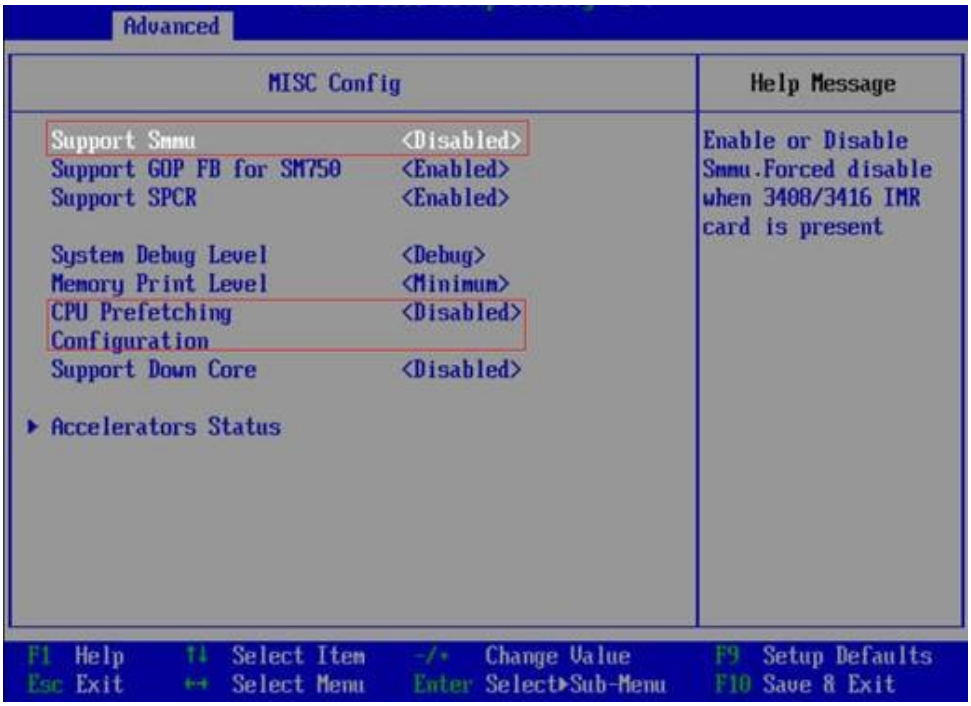


图 14-4 修改 BIOS 设置（1）

b.BIOS>Advanced>MISC Config，配置 CPU Prefetching Configuration 为 Disabled，如图 14-4 所示。

c.BIOS>Advanced>Memory Config，配置 Die Interleaving 为 Disable，如图 14-5 所示。

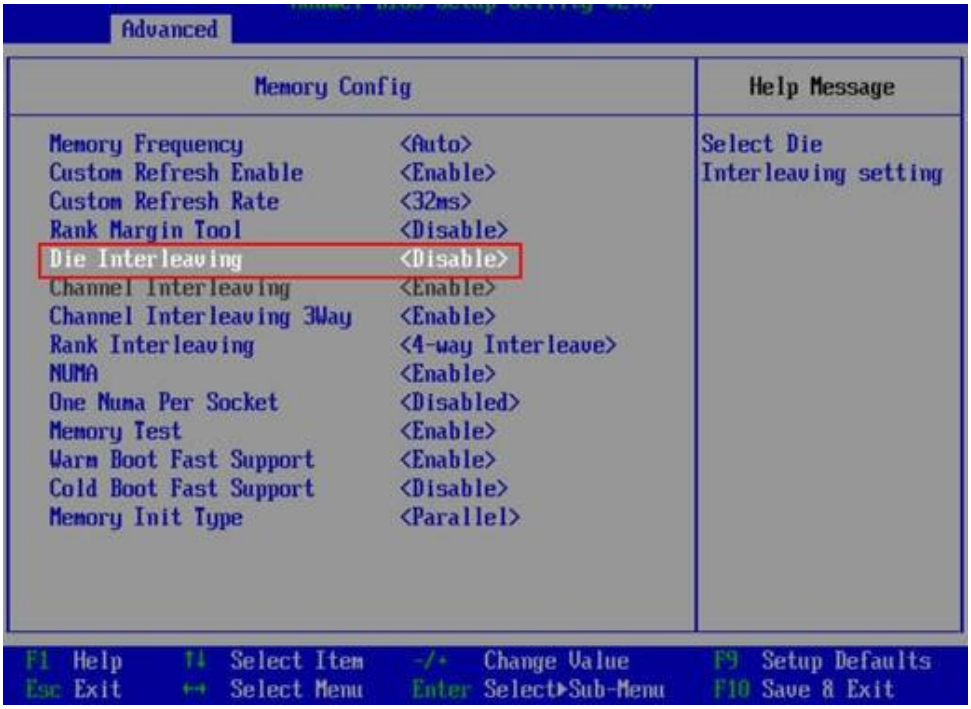


图 14-5 修改 BIOS 设置（2）

重启 BIOS 系统。

13.6.2.5 操作系统配置

1.安装操作系统。

2.修改操作系统内核 PAGESIZE 为 64KB。

3.关闭 CPU 中断的服务 irqbalance。irqbalance 负责均衡 CPU 中断，避免单 CPU 处理中断负载过重。

以 root 用户登录操作系统，执行如下命令。

```
service irqbalance stop     # 关闭 Irq balance。
echo 0 > /proc/sys/kernel/numa_balancing
echo 'never' > /sys/kernel/mm/transparent_hugepage/enabled
echo 'never' > /sys/kernel/mm/transparent_hugepage/defrag
echo none > /sys/block/nvme*n*/queue/scheduler    # 针对 nvme 磁盘 io 队列调度机制设置。
```

13.6.2.6 文件系统配置

为了提升数据库的 IO 效率，我们使用 NVME 盘作为数据盘，需要采用的文件系统类型为 xfs，数据块大小为 8KB。具体操作如下：

- 查看当前数据盘的文件系统类型。

a. 执行如下命令查看当前挂载的 nvme 盘。

```
df -h | grep nvme
```

结果如下。

```
/dev/nvme0n1                3.7T   2.6T   1.2T   69% /data1
/dev/nvme1n1                3.7T   1.9T   1.8T   51% /data2
/dev/nvme2n1                3.7T   2.2T   1.6T   59% /data3
/dev/nvme3n1                3.7T   1.4T   2.3T   39% /data4
```

b.执行如下命令查看 nvme 盘的信息。

```
xfs_info
```

例如执行命令 xfs_info /data1，如图 14-6 所示，block 的大小正好为 8KB，不需要修改；若 block 不满足 8kB 大小的要求，需要重新格式化，格式化前请将数据进行备份。

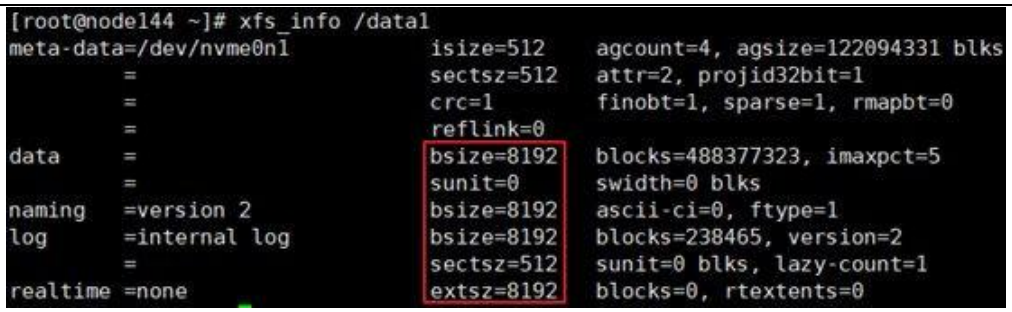


图 14-6 查看 nvme 盘的信息

对于需要修改的磁盘，备份所需的数据至其他磁盘或其他服务器。

格式化磁盘为 xfs 文件系统。以/dev/nvme0n1 盘，加载路径为/data1 为例，参考命令如下。此步骤需根据实际环境情况进行操作。

```
umount /data1
mkfs.xfs -b size=8192 /dev/nvme0n1 -f
mount /dev/nvme0n1 /data1
```

并确认 blocksize 是否已修改为 8KB。

13.6.2.7 网络配置

13.6.2.7.1 网卡多中断队列设置

产品需要在服务器端，客户端均使用网卡多中断队列（默认 16 队列）的规格，网卡至少为千兆网卡，客户端与服务端光纤互连。

当前推荐的配置为：

- 服务器端网卡配置 16 中断队列。
- 客户端网卡配置 48 中断队列。

操作步骤

(1) 下载 IN500_solution_5.1.0.SPC401.zip。

(2) 解压 IN500_solution_5.1.0.SPC401.zip，进入 tools\linux_arm 目录。

(3) 解压 nic - ZIP ，在 root 用户下安装 hinicadm。

(4) 确定当前连接的物理端口对应哪个网卡，不同硬件平台的网口和网卡名有差别。以当前举例的服务器为例，当前使用 enp3s0 的小网网口，属于 hinic0 网卡。

(5) 进入 config 目录， 利用配置工具 hinicconfig 配置中断队列 FW 配置文件。根据实际需

要进行修改。

- 64 队列配置文件：std_sh_4x25ge_dpdk_cfg_template0.ini；
- 16 队列配置文件：std_sh_4x25ge_nic_cfg_template0.ini；

a. 修改系统支持的最大中断队列数。

对 hinic0 卡配置为不同队列数（默认 16 队列，可以按需要调整）

```
./hinicconfig hinic0 -f std_sh_4x25ge_dpdk_cfg_template0.ini
```

执行命令 reboot 重启操作系统使生效。

执行命令 ethtool -l enp3s0 查看是否修改成功，比如下图表示修改为 64。

b. 修改当前使用的队列数。

执行如下命令，将网卡的中断队列调整为 48 个。

```
ethtool -L enp3s0 combined 48
```

说明

不同平台，不同应用的优化值可能不同，当前 128 核的平台，服务器端调优值为 16，客户端调优值为 48。

13.6.2.7.2 中断调优

在 GBase 8c 数据库 CPU 占比 90%以上的情况下，CPU 成为瓶颈，需要开启 offloading，将网络分片 offloading 到网卡上。

执行如下命令，开启 tso、lro、gro、gso 特性。

```
ethtool –K enp3s0 tso on
ethtool –K enp3s0 lro on
ethtool –K enp3s0 gro on
ethtool –K enp3s0 gso on
```

执行如下命令，将网卡中断队列与 CPU 核进行绑定。

```
sh bind_net_irq.sh    16
```

13.6.2.7.3 网卡固件确认与更新

执行命令 ethtool -i enp3s0 确认当前环境的网卡固件版本是否为 2.4.1.0，如果不是 2.4.1.0，建议更换为 2.4.1.0，以获得更佳性能。

```
# ethtool -i enp3s0
driver: hinic
```

```
version: 2.3.2.11
firmware-version: 2.4.1.0
expansion-rom-version:
bus-info: 0000:03:00.0
```

- 更新网卡固件。

a.在..\firmware\update_bin 路径下，获取 cfg_data_nic_prd_1h_4x25G.bin 文件。

b.使用 root 用户执行如下命令更新网卡固件。

```
hinicadm updatefw -i <物理网卡设备名> -f <固件文件路径>
```

涉及的参数说明如下：

- "物理网卡设备名"为网卡在系统中的名称，例如"hinic0"表示第一张网卡，"hinic1"表示第二张网卡，查找方法参见前文网卡多中断队列设置。
- "固件文件路径"为 cfg_data_nic_prd_1h_4x25G.bin 文件的路径。

例如：

```
#
Please do not remove driver or network device
Loading...
[>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>]    [100%] [\]
Loading firmware image succeed.
Please reboot OS to take firmware effect.
```

c.重启服务器，再确认网卡固件版本成功更新为 2.4.1.0。

```
# ethtool -i enp3s0
driver: hinic
version: 2.3.2.11
firmware-version: 2.4.1.0
expansion-rom-version:
bus-info: 0000:03:00.0
```

13.6.2.8 数据库服务端及客户端绑核

安装 GBase 8c 数据库，具体操作请参考《GBase 8c V5_3.0.x_安装指南》。

停止数据库，具体操作请参考《GBase 8c V5_3.0.x_GBase 8c 管理员指南》中的"启停 GBase 8c"章节。

使用 gs_guc 工具修改数据库端口、IP 等，gs_guc 的使用请参考《GBase 8c V5_3.0.x_GBase

8c 工具参考》中的“服务端工具>gs_guc”章节。

使用 gs_guc 工具设置如下参数。

```
advance_xlog_file_num = 100
numa_distribute_mode = 'all'
thread_pool_attr = '464,4,(cpubind:1-27,32-59,64-91,96-123)'
xloginsert_locks = 16
wal_writer_cpu=0
wal_file_init_num = 20
xlog_idle_flushes_before_sleep = 500000000
pagewriter_sleep = 10ms
```

执行如下命令以绑核方式启动服务端数据库。

```
numactl -C 1-27,32-59,64-91,96-123 gaussdb --single_node -D /data1/gaussdata    -p 3625 &
```

其中 0 核用于 wal_writer、1-27、32-59、64-91、96-123 表示使用 111 个核运行 TPCC 程序，其余的 16 个核用来处理服务端的网络中断。

执行如下命令，将客户端 CPU 的 48 个核与网卡中断队列进行绑定：

```
``` sh bind_net_irq.sh 48
```

13.6.3 测试 TPCC 性能

下载 TPCC 标准测试工具 BenchmarkSQL5.0。

将目录 lib/postgresql 下面的*.jar 替换为 GBase 8c 适配的 jar 包。

```
$ pwd
/your path/benchmarksql-5.0/lib/postgres
$ ls
postgresql.jar  # GBase 8c jdbc 驱动。
postgresql-9.3-1102.jdbc41.jar.bak       #  自带 jar 备份。
```

进入 benchmarksql-5.0 根目录，输入 ant 命令进行编译。

```
$ cd /your path/benchmarksql-5.0/
$ ant
```

编译成功后会生成 build 和 dist 两个目录。

创建 benchmarkSQL 配置文件，使用 benchmarkSQL 前需要配置数据库相关的信息，包括数据库账号、密码、端口、数据库名称。

```
$ cd /your path/benchmarksql-5.0/run
$ cp props.pg props.opengauss.1000w
```
```

```
$ vim props.opengauss.1000w
```

从 props.pg 拷贝一份配置文件并按如下修改，斜体处请根据真实情况进行修改。

```
db=postgres
driver=org.postgresql.Driver
// 修改连接字符串，包含 IP、端口号、数据库，其中 8.92.4.238 为数据库服务端的千兆网卡 IP。
conn=jdbc:postgresql://8.92.4.238:21579/tpcc1000?prepareThreshold=1&batchMode=on&fetchsize=10
// 设置数据库登录用户和密码。
user=bot
password=gbase;123

warehouses=1000
loadWorkers=200

// 设置最大并发数量，跟服务端最大 work 数对应。
terminals=812
//要为每个终端运行指定事务--runMins 必须等于零
runTxnsPerTerminal=0
//要运行指定的分钟 - runTxnsPerTerminal 必须等于零
runMins=5
//每分钟总事务数
limitTxnsPerMin=0

//在 4.x 兼容模式下运行时，设置为 True。
//设置为 false 以均匀使用整个配置的数据库。
terminalWarehouseFixed=false

//以下五个值相加之和为 100。
//45、43、4、4 和 4 的默认百分比与 TPC-C 规范匹配。
newOrderWeight=45
paymentWeight=43
orderStatusWeight=4
deliveryWeight=4
stockLevelWeight=4

//创建文件夹以收集详细结果数据。
//通过注释取消此内容。
resultDirectory=my_result_%tY-%tm-%td_%tH%tM%tS
osCollectorScript=./misc/os_collector_linux.py
```

```
osCollectorInterval=1
//收集 OS 负载信息。
osCollectorSSHAddr=osuer@10.44.133.78
osCollectorDevices=net_enp3s0 blk_nvme0n1 blk_nvme1n1 blk_nvme2n1 blk_nvme3n1
```

TPCC 导入数据前准备。

使用如下文件替换 benchmarkSQL 中的文件，路径为 benchmarksql-5.0/run/sql.common/。该文件主要增加了两个表空间和一些附加数据属性。

```
CREATE TABLESPACE example2 relative location 'tablespace2';
CREATE TABLESPACE example3 relative location 'tablespace3';

create table bmsql_config (
    cfg_name        varchar(30),
    cfg_value       varchar(50)
);

create table bmsql_warehouse (
    w_id            integer      not null,
    w_ytd           decimal(12,2),
    w_tax           decimal(4,4),
    w_name          varchar(10),
    w_street_1      varchar(20),
    w_street_2      varchar(20),
    w_city          varchar(20),
    w_state         char(2),
    w_zip           char(9)
) WITH (FILLFACTOR=80);

create table bmsql_district (
    d_w_id          integer       not null,
    d_id            integer       not null,
    d_ytd           decimal(12,2),
    d_tax           decimal(4,4),
    d_next_o_id     integer,
    d_name          varchar(10),
    d_street_1      varchar(20),
    d_street_2      varchar(20),
    d_city          varchar(20),
    d_state         char(2),
    d_zip           char(9)
 ) WITH (FILLFACTOR=80);
```

```
create table bmsql_customer (
    c_w_id              integer             not null,
    c_d_id              integer             not null,
    c_id                integer             not null,
    c_discount          decimal(4,4),
    c_credit            char(2),
    c_last              varchar(16),
    c_first             varchar(16),
    c_credit_lim        decimal(12,2),
    c_balance           decimal(12,2),
    c_ytd_payment       decimal(12,2),
    c_payment_cnt       integer,
    c_delivery_cnt integer,
    c_street_1          varchar(20),
    c_street_2          varchar(20),
    c_city              varchar(20),
    c_state             char(2),
    c_zip               char(9),
    c_phone             char(16),
    c_since             timestamp,
    c_middle            char(2),
    c_data              varchar(500)
) WITH (FILLFACTOR=80) tablespace example2;

create sequence bmsql_hist_id_seq;

create table bmsql_history (
    hist_id    integer,
    h_c_id     integer,
    h_c_d_id integer,
    h_c_w_id integer,
    h_d_id     integer,
    h_w_id     integer,
    h_date     timestamp,
    h_amount decimal(6,2),
    h_data     varchar(24)
) WITH (FILLFACTOR=80);

create table bmsql_new_order (
    no_w_id    integer     not null,
```

```
    no_d_id    integer      not null,
    no_o_id    integer      not null
) WITH (FILLFACTOR=80);

create table bmsql_oorder (
    o_w_id          integer          not null,
    o_d_id          integer          not null,
    o_id            integer          not null,
    o_c_id          integer,
    o_carrier_id integer,
    o_ol_cnt        integer,
    o_all_local     integer,
    o_entry_d       timestamp
) WITH (FILLFACTOR=80);

create table bmsql_order_line (
    ol_w_id             integer      not null,
    ol_d_id             integer      not null,
    ol_o_id             integer      not null,
    ol_number           integer      not null,
    ol_i_id             integer      not null,
    ol_delivery_d       timestamp,
    ol_amount           decimal(6,2),
    ol_supply_w_id      integer,
    ol_quantity         integer,
    ol_dist_info        char(24)
) WITH (FILLFACTOR=80);

create table bmsql_item (
    i_id         integer          not null,
    i_name       varchar(24),
    i_price      decimal(5,2),
    i_data       varchar(50),
    i_im_id      integer
);

create table bmsql_stock (
    s_w_id           integer          not null,
    s_i_id           integer          not null,
    s_quantity       integer,
    s_ytd            integer,
```

```
  s_order_cnt    integer,
  s_remote_cnt integer,
  s_data           varchar(50),
  s_dist_01       char(24),
  s_dist_02       char(24),
  s_dist_03       char(24),
  s_dist_04       char(24),
  s_dist_05       char(24),
  s_dist_06       char(24),
  s_dist_07       char(24),
  s_dist_08       char(24),
  s_dist_09       char(24),
  s_dist_10       char(24)
) WITH (FILLFACTOR=80) tablespace example3;
```

- 导入数据。

  创建数据库用户。

```
create user bot identified by 'Gaussdba@Mpp' profile default;
alter user bot sysadmin;
create database tpcc1000 encoding 'UTF8' template=template0 owner tpcc5q;
```

  执行如下命令导入数据。

```
./runDatabaseBuild.sh props.opengauss.1000w
```

- 备份数据。

  为了方便多次测试，减少导入数据的时间，可以通过停止数据库，将整个数据目录执行一次拷贝对数据库进行备份。

- 对数据进行分盘。

  在性能测试过程中，为了增加 IO 的吞吐量，需要将数据分散到不同的存储介质上。由于机器上有 4 块 NVME 盘，可以将数据分散到不同的盘上。将 pg_xlog、tablespace2、tablespace3 这三个目录放置在其他 3 个 NVME 盘上，并在原有的位置给出指向真实位置的软连接。pg_xlog 位于数据库目录下，tablespace2、tablespace3 分别位于数据库目录 pg_location 下。对 tablespace2 分盘的命令如下:

```
mv $DATA_DIR/pg_location/tablespace2 $TABSPACE2_DIR/tablespace2
cd $DATA_DIR/pg_location/
ln -svf $TABSPACE2_DIR/tablespace2 ./
```

- 运行 TPCC 程序。

```
numactl  -C 0-19,32-51,64-83,96-115 ./runBenchmark.sh props.opengauss.1000w
```

- 验证数据测试过程正确性。

使用 htop 监控数据库服务端和 tpcc 客户端 CPU 利用情况，最佳性能测试情况下，各个业务 CPU 的占用率都非常高（> 90%）。如果有 CPU 占用率没有达标，可能是绑核方式不对或其他问题，需要定位找到根因进行调整。

如果为了避免数据的干扰，需要进行重新测试，可以通过备份的数据通过拷贝的方式恢复数据。重复对数据进行分盘、运行 TPCC 程序、验证数据测试过程正确性这三个步骤，可以重新进行测试。

13.6.4 TPCC 测试用例

13.6.4.1 环境配置

13.6.4.1.1 硬件配置

服务器	CPU	内存	硬盘	网络信息
100.0.0.50	Intel(R) Xeon(R) Silver 4216 CPU @ 2.10GHz，64U	256G;2400 MHZ	SSD 6.1T	万兆网
100.0.0.55	Intel(R) Xeon(R) Silver 4216 CPU @ 2.10GHz，40U	128G;2400 MHZ	SSD 6.1T	万兆网

通过环境的查看，可以将数据库部署在读写速率最好的磁盘上。

```
[root@gbasehost_0_50 run]# hdparm -Tt /dev/sdd


/dev/sdd:
 Timing cached reads:   16742 MB in  1.99 seconds = 8400.78 MB/sec
 Timing buffered disk reads: 1656 MB in  3.00 seconds = 551.72 MB/sec
```

13.6.4.1.2 软件配置

操作系统：CentOS Linux release 7.8.2003

数据库版本：GBase8c 3.0.1B07 build 73c8d716

Benchmarksql 版本：benchmarksql-5.0

13.6.4.2 性能调优

13.6.4.2.1 系统参数调优

```
1)文件句柄：ulimit -n 1000000
2)sysctl.conf 系统参数配置
vm.max_map_count = 999999
#kernel.sem = 4096 32000 1024 200
kernel.shmall = 65894144
kernel.shmmax = 161941448294
kernel.shmmni = 4096
kernel.sem = 501000 641280000 501000 12800
net.ipv4.ip_local_port_range =  9000 65500
net.core.rmem_default = 4194304
net.core.rmem_max = 4194304
net.core.wmem_default = 262144
net.core.wmem_max = 1048586
fs.file-max = 6815744
fs.aio-max-nr = 1048576
```

13.6.4.2.2 数据库参数调优

```
fsync='on';
full_page_writes='off';
shared_buffers='38GB';
max_connections =4096;
max_prepared_transactions =2048;
wal_buffers='128MB';
checkpoint_completion_target = 0.9;
audit_enabled=off;
cstore_buffers='16MB';
enable_alarm=off;
max_files_per_process=100000;
use_workload_manager=off;
wal_buffers='1GB';
autovacuum_mode=vacuum;
autovacuum_max_workers=20;
autovacuum_naptime='5s';
wal_log_hints=off;
autovacuum_analyze_scale_factor = 0.02;
autovacuum_vacuum_scale_factor =0.1;
```

```
advance_xlog_file_num =100;
enable_instr_cpu_timer =off;
enable_instance_metric_persistent =off;
enable_logical_io_statistics =off;
enable_page_lsn_check =off;
enable_user_metric_persistent =off;
enable_xlog_prune =off;
enable_resource_track=off;
instr_unique_sql_count =100;
remote_read_mode = non_authentication;
wal_level = archive;
hot_standby =off;
enable_asp =off;
enable_bbox_dump =off;
bgwriter_flush_after =32;
checkpoint_segments =8000;
enable_stmt_track =false;
bgwriter_delay ='5s';
xloginsert_locks =16;
wal_file_init_num=20;
pagewriter_sleep ='10ms';

synchronous_commit='on';                conf
maintenance_work_mem='10240MB';    conf
commit_delay=200;                       conf
temp_buffers='128MB';                   conf
random_page_cost=1.1;          conf
commit_siblings = 10;          conf
effective_cache_size = '64GB';      conf
log_min_duration_statement = 1000;      conf
work_mem='1MB';                 conf
vacuum_cost_limit=10000;        ---conf
update_lockwait_timeout='20min';    ---conf
enable_mergejoin=off;    conf
enable_nestloop=off;        conf
enable_hashjoin=off;        conf
enable_material=off;      conf
log_duration=off;        conf
query_dop =8;          conf
enable_save_datachanged_timestamp =false;      --conf
track_sql_count =off;          --conf
```

```
session_timeout =0;         conf
client_min_messages = ERROR;      conf
log_min_messages = FATAL;        conf
enable_seqscan =off;      conf
enable_beta_opfusion =on      conf
```

13.6.4.2.3 TPC-C 参数配置

```
db=postgres
driver=org.postgresql.Driver
conn=jdbc:postgresql://100.0.0.50:5432/tpcc?loggerLevel=off
user=tpcc_user
Password=******
warehouses=300
terminals=100
runMins=5
limitTxnsPerMin=1000000
terminalWarehouseFixed=true
newOrderWeight=45
paymentWeight=43
orderStatusWeight=4
deliveryWeight=4
stockLevelWeight=4
```

13.6.4.2.4 建表语句

```
create tablespace example2 relative location 'tablespace2';
create tablespace example3 relative location 'tablespace3';
create table bmsql_config (
    cfg_name        varchar(30) primary key,
    cfg_value       varchar(50)
);

create table bmsql_warehouse (
    w_id             integer      not null,
    w_ytd            decimal(12,2),
    w_tax            decimal(4,4),
    w_name             varchar(10),
    w_street_1     varchar(20),
    w_street_2     varchar(20),
    w_city           varchar(20),
    w_state          char(2),
    w_zip            char(9)
```

```
)with (fillfactor=80)
;

create table bmsql_district (
    d_w_id          integer          not null,
    d_id            integer          not null,
    d_ytd           decimal(12,2),
    d_tax           decimal(4,4),
    d_next_o_id     integer,
    d_name          varchar(10),
    d_street_1      varchar(20),
    d_street_2      varchar(20),
    d_city          varchar(20),
    d_state         char(2),
    d_zip           char(9))with (fillfactor=80);

create table bmsql_customer (
    c_w_id          integer          not null,
    c_d_id          integer          not null,
    c_id            integer          not null,
    c_discount      decimal(4,4),
    c_credit        char(2),
    c_last          varchar(16),
    c_first         varchar(16),
    c_credit_lim    decimal(12,2),
    c_balance       decimal(12,2),
    c_ytd_payment   decimal(12,2),
    c_payment_cnt   integer,
    c_delivery_cnt integer,
    c_street_1      varchar(20),
    c_street_2      varchar(20),
    c_city          varchar(20),
    c_state         char(2),
    c_zip           char(9),
    c_phone         char(16),
    c_since         timestamp,
    c_middle        char(2),
    c_data          varchar(500)
)with (fillfactor=80)
tablespace example2;
```

```
--create sequence bmsql_hist_id_seq cache 100;

create table bmsql_history (
  hist_id  integer,
  h_c_id   integer,
  h_c_d_id integer,
  h_c_w_id integer,
  h_d_id   integer,
  h_w_id   integer,
  h_date   timestamp,
  h_amount decimal(6,2),
  h_data   varchar(24)
)with (fillfactor=80);
--DISTRIBUTE BY HASH(h_w_id);

create table bmsql_new_order (
  no_w_id  integer   not null,
  no_d_id  integer   not null,
  no_o_id  integer   not null,
  primary key (no_w_id, no_d_id, no_o_id)
)with (fillfactor=80);
--DISTRIBUTE BY HASH(no_w_id);

create table bmsql_oorder (
  o_w_id       integer      not null,
  o_d_id       integer      not null,
  o_id         integer      not null,
  o_c_id       integer,
  o_carrier_id integer,
  o_ol_cnt     integer,
  o_all_local  integer,
  o_entry_d    timestamp
)with (fillfactor=80) ;
--DISTRIBUTE BY HASH(o_w_id);

create table bmsql_order_line (
  ol_w_id      integer   not null,
  ol_d_id      integer   not null,
  ol_o_id      integer   not null,
  ol_number    integer   not null,
  ol_i_id      integer   not null,
```

```
    ol_delivery_d     timestamp,
    ol_amount           decimal(6,2),
    ol_supply_w_id    integer,
    ol_quantity         integer,
    ol_dist_info       char(24)
)with (fillfactor=80);
--DISTRIBUTE BY HASH(ol_w_id);

create table bmsql_item (
    i_id          integer          not null,
    i_name      varchar(24),
    i_price     decimal(5,2),
    i_data      varchar(50),
    i_im_id     integer
)with (fillfactor=80);

create table bmsql_stock (
    s_w_id            integer            not null,
    s_i_id            integer            not null,
    s_quantity      integer,
    s_ytd             integer,
    s_order_cnt     integer,
    s_remote_cnt integer,
    s_data            varchar(50),
    s_dist_01       char(24),
    s_dist_02       char(24),
    s_dist_03       char(24),
    s_dist_04       char(24),
    s_dist_05       char(24),
    s_dist_06       char(24),
    s_dist_07       char(24),
    s_dist_08       char(24),
    s_dist_09       char(24),
    s_dist_10       char(24)
)with (fillfactor=80)
tablespace example3;
--DISTRIBUTE BY HASH(s_w_id);
```

13.6.4.2.5 建索引语句

```
alter table bmsql_warehouse add constraint bmsql_warehouse_pkey
    primary key (w_id);
```

```
alter table bmsql_district add constraint bmsql_district_pkey
   primary key (d_w_id, d_id);
alter table bmsql_customer add constraint bmsql_customer_pkey
   primary key (c_w_id, c_d_id, c_id);
create index bmsql_customer_idx1
   on    bmsql_customer (c_w_id, c_d_id, c_last, c_first);
alter table bmsql_oorder add constraint bmsql_oorder_pkey
   primary key (o_w_id, o_d_id, o_id);
create unique index bmsql_oorder_idx1
   on    bmsql_oorder (o_w_id, o_d_id, o_carrier_id, o_id);
alter table bmsql_new_order add constraint bmsql_new_order_pkey
   primary key (no_w_id, no_d_id, no_o_id) using index tablespace example2;
alter table bmsql_order_line add constraint bmsql_order_line_pkey
   primary key (ol_w_id, ol_d_id, ol_o_id, ol_number);
alter table bmsql_stock add constraint bmsql_stock_pkey
   primary key (s_w_id, s_i_id);
alter table bmsql_item add constraint bmsql_item_pkey
   primary key (i_id);
```

13.6.4.2.6 调优效果展示

```
11:52:07,043 [main] INFO   jTPCC : Term-00,
11:52:07,043 [main] INFO   jTPCC : Term-00, db=postgres
11:52:07,044 [main] INFO   jTPCC : Term-00, driver=org.postgresql.Driver
11:52:07,044 [main] INFO   jTPCC : Term-00, conn=jdbc:postgresql://100.0.0.50:5432/tpcc?prepareThreshold=5&batchMode=on&fetchsize=10&loggerLevel=off
11:52:07,044 [main] INFO   jTPCC : Term-00, user=tpcc_user
11:52:07,044 [main] INFO   jTPCC : Term-00,
11:52:07,044 [main] INFO   jTPCC : Term-00, warehouses=300
11:52:07,044 [main] INFO   jTPCC : Term-00, terminals=100
11:52:07,045 [main] INFO   jTPCC : Term-00, runMins=5
11:52:07,046 [main] INFO   jTPCC : Term-00, limitTxnsPerMin=0
11:52:07,046 [main] INFO   jTPCC : Term-00, terminalWarehouseFixed=true
11:52:07,046 [main] INFO   jTPCC : Term-00,
11:52:07,046 [main] INFO   jTPCC : Term-00, newOrderWeight=45
11:52:07,046 [main] INFO   jTPCC : Term-00, paymentWeight=43
11:52:07,046 [main] INFO   jTPCC : Term-00, orderStatusWeight=4
11:52:07,046 [main] INFO   jTPCC : Term-00, deliveryWeight=4
11:52:07,046 [main] INFO   jTPCC : Term-00, stockLevelWeight=4
11:52:07,046 [main] INFO   jTPCC : Term-00,
11:52:07,046 [main] INFO   jTPCC : Term-00, resultDirectory=null
11:52:07,046 [main] INFO   jTPCC : Term-00, osCollectorScript=null
11:52:07,047 [main] INFO   jTPCC : Term-00,
11:52:07,365 [main] INFO   jTPCC : Term-00, C value for C_LAST during load: 15
11:52:07,365 [mTerm-00, Running Average tpmTOTAL: 1293228.68    Current tpmTOTAL: 42805116    Memory Usage: 437MB / 1832MB
11:57:16,160 [Thread-48] INFO   jTPCC : Term-00,
11:57:16,161 [Thread-48] INFO   jTPCC : Term-00,
11:57:16,161 [Thread-48] INFO   jTPCC : Term-00, Measured tpmC (NewOrders) = 581680.5
11:57:16,161 [Thread-48] INFO   jTPCC : Term-00, Measured tpmTOTAL = 1293129.9
11:57:16,161 [Thread-48] INFO   jTPCC : Term-00, Session Start     = 2022-05-18 11:52:16
11:57:16,161 [Thread-48] INFO   jTPCC : Term-00, Session End       = 2022-05-18 11:57:16
11:57:16,161 [Thread-48] INFO   jTPCC : Term-00, Transaction Count = 6466252
[gbase@gbase8c_0_50_run]$
```

13.7 实际调优案例

rewrite_rule 包含了多个查询重写规则：magicset、partialpush、uniquecheck、disablerep、intargetlist、predpush。下面简要说明一下其中重要的几个规则的使用场景：

目标列子查询提升参数 intargetlist

通过将目标列中子查询提升，转为 JOIN，往往可以极大提升查询性能。举例如下查询：

```
postgres=# set rewrite_rule='none';
```

```
SET
postgres=# create table t1(c1 int,c2 int);
CREATE TABLE
postgres=# create table t2(c1 int,c2 int);
CREATE TABLE
postgres=#    explain (verbose on, costs off) select c1,(select avg(c2) from t2 where t2.c2=t1.c2)
from t1 where t1.c1<100 order by t1.c2;
                            QUERY PLAN
-----------------------------------------------
 Sort
    Output: t1.c1, ((SubPlan 1)), t1.c2
    Sort Key: t1.c2
    ->   Seq Scan on public.t1
            Output: t1.c1, (SubPlan 1), t1.c2
            Filter: (t1.c1 < 100)
            SubPlan 1
              ->   Aggregate
                      Output: avg(t2.c2)
                      ->   Seq Scan on public.t2
                              Output: t2.c1, t2.c2
                              Filter: (t2.c2 = t1.c2)
(12 rows)
```

由于目标列中的相关子查询(select avg(c2) from t2 where t2.c2=t1.c2)无法提升的缘故，导致每扫描 t1 的一行数据，就会触发子查询的一次执行，效率低下。如果打开 intargetlist 参数会把子查询提升转为 JOIN，来提升查询的性能：

```
postgres=# set rewrite_rule='intargetlist';
SET
postgres=#    explain (verbose on, costs off) select c1,(select avg(c2) from t2 where t2.c2=t1.c2)
from t1 where t1.c1<100 order by t1.c2;
                            QUERY PLAN
-----------------------------------------------
 Sort
    Output: t1.c1, (avg(t2.c2)), t1.c2
    Sort Key: t1.c2
    ->   Hash Left Join
            Output: t1.c1, (avg(t2.c2)), t1.c2
            Hash Cond: (t1.c2 = t2.c2)
            ->   Seq Scan on public.t1
                    Output: t1.c1, t1.c2
                    Filter: (t1.c1 < 100)
```

```
            ->   Hash
                    Output: (avg(t2.c2)), t2.c2
                    ->   HashAggregate
                            Output: avg(t2.c2), t2.c2
                            Group By Key: t2.c2
                            ->   Seq Scan on public.t2
                                    Output: t2.c2
(16 rows)
```

提升无 agg 的子查询 uniquecheck

子链接提升需要保证对于每个条件只有一行输出，对于有 agg 的子查询可以自动提升，对于无 agg 的子查询如：

```
select t1.c1 from t1 where t1.c1 = (select t2.c1 from t2 where t1.c1=t2.c2);
```

重写为：

```
select t1.c1 from t1 join (select t2.c1 from t2 where t2.c1 is not null group by t2.c1(unique
check)) tt(c1) on tt.c1=t1.c1;
```

为了保证语义等价，子查询 tt 必须保证对于每个 group by t2.c1 只能有一行输出。打开 uniquecheck 查询重写参数保证可以提升并且等价，如果在运行时输出了多于一行的数据，就会报错。

```
postgres=# set rewrite_rule='uniquecheck';
SET
postgres=# explain verbose select t1.c1 from t1 where t1.c1 = (select t2.c1 from t2 where
t1.c1=t2.c1);
                                                QUERY PLAN
-----------------------------------------------------------------------------------------
 Hash Join    (cost=43.36..104.40 rows=2149 distinct=[200, 200] width=4)
    Output: t1.c1
    Hash Cond: (t1.c1 = subquery."?column?")
    ->   Seq Scan on public.t1    (cost=0.00..31.49 rows=2149 width=4)
            Output: t1.c1, t1.c2
    ->   Hash    (cost=40.86..40.86 rows=200 width=8)
            Output: subquery."?column?", subquery.c1
            ->   Subquery Scan on subquery    (cost=36.86..40.86 rows=200 width=8)
                    Output: subquery."?column?", subquery.c1
                    ->   HashAggregate    (cost=36.86..38.86 rows=200 width=4)
                            Output: t2.c1, t2.c1
                            Group By Key: t2.c1
                            Filter: (t2.c1 IS NOT NULL)
```

```
                        Unique Check Required
                        ->  Seq Scan on public.t2  (cost=0.00..31.49 rows=2149 width=4)
                              Output: t2.c1
(16 rows)
```

注意：因为分组 group by t2.c1 unique check 发生在过滤条件 tt.c1=t1.c1 之前，可能导致原来不报错的查询重写之后报错。举例：

有 t1、t2 表，其中的数据为：

```
postgres=# select * from t1 order by c2;
 c1 | c2
----+----
  1 |  1
  2 |  2
  3 |  3
(3 rows)
postgres=# select * from t2 order by c2;
 c1 | c2
----+----
  1 |  1
  2 |  2
  3 |  3
  4 |  4
  4 |  4
  5 |  5
(6 rows)
```

分别关闭和打开 uniquecheck 参数对比，打开之后报错。

```
postgres=# select t1.c1 from t1 where t1.c1 = (select t2.c1 from t2 where t1.c1=t2.c2) ;
 c1
----
  1
  2
  3
(3 rows)
postgres=# set rewrite_rule='uniquecheck';
SET
postgres=# select t1.c1 from t1 where t1.c1 = (select t2.c1 from t2 where t1.c1=t2.c2) ;
ERROR:  more than one row returned by a subquery used as an expression
```

13.8 性能日志

性能日志主要关注外部资源的访问性能问题。

性能日志指的是数据库系统在运行时检测物理资源的运行状态的日志，在对外部资源进行访问时的性能检测，包括磁盘、OBS、Hadoop 集群等外部资源的访问检测信息。GBase 8c 不支持 OBS、Hadoop，所以只有磁盘访问的监控信息。

磁盘监控的访问信息主要在磁盘文件 IO 读写的时候进行统计。例如，拷贝文件时的读文件 IO，正常 SQL 执行时访问 OS 表文件的 pread 系统调用。

在出现性能问题时，可以借助性能日志及时定位问题发生的原因，能极大地提升问题解决效率。

- 日志文件存储路径

  性能日志目录在“$GAUSSLOG/gs_profile”中各自对应的目录下。

- 日志文件命名格式

  性能日志的命名规则：postgresql-创建时间.prf

  默认情况下，每日 0 点或者日志文件大于 20MB 或者数据库实例（CN、DN）重新启动后，会生成新的日志文件。

- 日志内容说明

  每一行日志内容的默认格式：主机名称+日期+时间+实例名称+线程号+日志内容。

性能日志收集的配置参数

- logging_collector：该参数控制是否开启日志收集功能。默认值为 on，表示开启日志收集功能。
- plog_merge_age：控制性能日志数据输出的周期，即多久进行一次性能日志汇聚，单位为毫秒。默认值为 3s。

当 logging_collector 参数为 on，plog_merge_age 大于 0，且主机正常运行时，恢复过程不进行性能收集。

14 配置运行参数

14.1 查看参数当前取值

GBase 8c 安装后使用默认的运行参数。为了使数据库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 GUC 参数调整。

操作步骤

(1) 以操作系统用户 gbase 登录数据库主节点。

(2) 使用如下命令连接数据库。

```
gsql -d postgres -p 5432
```

postgres 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：

```
gsql ((GBase 8c V5 build a4c7bcc5) compiled at 2024-01-16 14:49:56 commit 0 last mr 1632 )
Non-SSL connection (SSL connection is recommended when requiring high-security)
Last login: 2024-01-19 16:22:27+08 from [local] (user gbase)
Type "help" for help.

postgres=#
```

(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：

```
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：

```
SHOW ALL;
```

方法二：使用 pg_settings 视图。

使用如下命令查看单个参数：

```
SELECT * FROM pg_settings WHERE NAME='server_version';
```

使用如下命令查看所有参数：

```
SELECT * FROM pg_settings;
```

示例

查看服务器的版本号。

```
postgres=# SHOW server_version;
 server_version
----------------
 9.2.4
(1 row)
```

14.2 重设参数

14.2.1 背景信息

GBase 8c 提供了多种修改 GUC 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 GUC 参数说明。

14.2.2 GUC 参数设置

GBase 8c 提供了六类 GUC 参数，具体分类和设置方式参考表 15-1。

表 15-1 GUC 参数说明

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	无

	修改，只能通过 show 语法或者 pg_settings 视图进行查看。	
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 2 中的方式一、方式四。
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 2 中的方式一、方式二、方式四。
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 2 中的方式一、方式二、方式四。 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、数据库启动后或者数据库管理员通过 SQL 进行设置。	支持表 2 中的方式一、方式二或由数据库管理员通过方式三设置。
USERSET	普通用户参数。可被任何用户在任何时刻设置。	支持表 2 中的方式一、方式二或方式三设置。

GBase 8c 提供了四种方式来修改 GUC 参数，具体操作参考表 15-2。

表 15-2 GUC 参数设置方式

序号	设置方法

方式一	(1) 使用如下命令修改参数。
	`gs_guc set [-Z NODETYPE] -D datadir -c “paraname=value“`
	说明：
	● 如果参数是一个字符串变量，则使用-c parameter=“'value'“或者使用-c “parameter = 'value'“。
	● -Z 参数仅在分布式情况下需要指定。
	使用以下命令在数据库节点上同时设置某个参数。
	`gs_guc set [-Z NODETYPE] -N all -I all -c “paraname=value“`

	(2) 重启数据库使参数生效。 说明： 重启 GBase 8c 数据库的操作，会导致用户执行操作中断，请在操作之前规划好合适的执行窗口。 gha_ctl stop all -l http://<dcs_ip>:2379 gha_ctl start all -l http://<dcs_ip>:2379
方式二	gs_guc reload -D datadir -c “paraname=value“ 说明： 使用以下命令在数据库节点上同时设置某个参数。 gs_guc reload [-Z NODETYPE] -N all -I all -c “paraname=value“

方式三	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
	postgres=# ALTER DATABASE dbname SET paraname TO value;
	在下次会话中生效。
	● 设置用户级别的参数
	postgres=# ALTER USER username SET paraname TO value;
	在下次会话中生效。
	● 设置会话级别的参数
	postgres=# SET paraname TO value;
	修改本次会话中的取值。退出会话后，设置将失效。
	说明：
	SET 设置的会话级参数优先级最高，其次是 ALTER 设置的，其中 ALTER DATABASE 设置的参数值优先级高于 ALTER USER 设置，这三种设置方式设置的优先级都高于 gs_guc 设置方式。

方式四	使用 ALTER SYSTEM SET 修改数据库参数。 ● 设置 POSTMASERT 级别的参数 postgres=# ALTER SYSTEM SET paraname TO value; 重启后生效。 ● 设置 SIGHUP 级别的参数 postgres=# ALTER SYSTEM SET paraname TO value;

	立刻生效(实际等待线程重新加载参数略有延迟)。 ● 设置 BACKEND 级别的参数 postgres=# ALTER SYSTEM SET paraname TO value; 在下次会话中生效。

⚠ 注意

使用方式一和方式二设置参数时，若所设参数不属于当前环境，数据库会提示参数不在支持范围内的相关信息。

14.2.3 操作步骤

以主备式集群为例。

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。

(1) 以操作系统用户 gbase 登录数据库主节点。

(2) 查看 archive_mode 参数。

```
cat /home/gbase/data/dn1/dn1_1/postgresql.conf | grep archive_mode
archive_mode = on
```

on 表示日志要进行归档操作。

(3) 设置 archive_mode 参数为 off，关闭日志的归档操作。例如：

```
gs_guc set -D /home/gbase/data/dn1/dn1_1 -c "archive_mode=off"
```

说明： 可以使用以下命令在数据库节点上设置 archive_mode 参数为 off。例如：

```
gs_guc set -N all -I all -c "archive_mode=off"
```

(4) 重启数据库使参数生效。

```
gha_ctl stop all -l http://<dcs_ip>:2379
gha_ctl start all -l http://<dcs_ip>:2379
```

(5) 使用如下命令连接数据库。

```
gsql -d postgres -p 5432
```

postgres 为需要连接的数据库名称，5432 为数据库主节点的端口号（根据实际情况指定端口号。以下端口说明与此相同）。

连接成功后，系统显示类似如下信息：

```
gsql ((GBase 8c V5 build a4c7bcc5) compiled at 2024-01-16 14:49:56 commit 0 last mr 1632 )
Non-SSL connection (SSL connection is recommended when requiring high-security)
Last login: 2024-01-19 16:22:27+08 from [local] (user gbase)
Type "help" for help.

postgres=#
```

(6) 检查参数设置的正确性。

```
postgres=# SHOW archive_mode;
 archive_mode
--------------
 off
(1 row)
```

使用方式二设置参数，以在数据库主节点设置 authentication_timeout 参数为例。

(1) 以操作系统用户 gbase 登录数据库主节点。

(2) 查看 authentication_timeout 参数。

```
cat /home/gbase/data/dn1/dn1_1/postgresql.conf | grep authentication_timeout
authentication_timeout = 1min
```

(3) 设置 authentication_timeout 参数为 59s。

```
gs_guc reload -N all -I all -c "authentication_timeout = 59s"

Total instances: 2. Failed instances: 0.
Success to perform gs_guc!
```

说明： 可以使用以下命令在数据库节点上设置 authentication_timeout 参数为 59s。

```
gs_guc reload -N all -I all -c "authentication_timeout = 59s"
```

(4) 使用如下命令连接数据库。

```
gsql -d postgres -p 5432
```

postgres 为需要连接的数据库名称，5432 为数据库主节点的端口号（根据实际情况，指定相应端口号）。

连接成功后，系统显示类似如下信息：

```
gsql ((GBase 8c V5 build a4c7bcc5) compiled at 2024-01-16 14:49:56 commit 0 last mr 1632 )
Non-SSL connection (SSL connection is recommended when requiring high-security)
Last login: 2024-01-19 16:22:27+08 from [local] (user gbase)
Type "help" for help.
```

```
postgres=#
```

(5) 检查参数设置的正确性。

```
postgres=# SHOW authentication_timeout;
 authentication_timeout
------------------------
 59s
(1 row)
```

使用方式三设置参数，以设置 explain_perf_mode 参数为例。

(1) 以操作系统用户 gbase 登录数据库主节点。

(2) 使用如下命令连接数据库。

```
gsql -d postgres -p 5432
```

postgres 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：

```
gsql ((GBase 8c V5 build a4c7bcc5) compiled at 2024-01-16 14:49:56 commit 0 last mr 1632 )
Non-SSL connection (SSL connection is recommended when requiring high-security)
Last login: 2024-01-19 16:22:27+08 from [local] (user gbase)
Type "help" for help.

postgres=#
```

(3) 查看 explain_perf_mode 参数。

```
postgres=# SHOW explain_perf_mode;
 explain_perf_mode
-------------------
 normal
(1 row)
```

(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

```
postgres=# ALTER DATABASE postgres 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

```
postgres=# ALTER USER gbase 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

```
postgres=# SET explain_perf_mode TO pretty;
```

当结果显示为如下信息，则表示设置成功。

```
SET
```

(5) 检查参数设置的正确性。

```
postgres=# SHOW explain_perf_mode;
 explain_perf_mode
--------------
 pretty
(1 row)
```

14.2.4 示例

以主备式集群为例。

示例 1：使用方式一修改 GBase 8c 数据库主节点的最大连接数。

(1) 以操作系统用户 gbase 登录数据库主节点。

(2) 使用如下命令连接数据库。

```
gsql -d postgres -p 5432
```

postgres 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：

```
gsql ((GBase 8c V5 build a4c7bcc5) compiled at 2024-01-16 14:49:56 commit 0 last mr 1632 )
Non-SSL connection (SSL connection is recommended when requiring high-security)
Last login: 2024-01-19 16:22:27+08 from [local] (user gbase)
Type "help" for help.

postgres=#
```

(3) 查看最大连接数。

```
postgres=# SHOW max_connections;
 max_connections
-----------------
 200
(1 row)
```

(4) 使用如下命令退出数据库。

```
postgres=# \q
```

(5) 修改 GBase 8c 数据库主节点的最大连接数。

```
gs_guc set -N all -I all -c "max_connections = 800"
```

(6) 重启 GBase 8c。

```
gs_om -t stop && gs_om -t start
```

(7) 使用如下命令连接数据库。

```
gsql -d postgres -p 5432
```

postgres 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：

```
gsql ((GBase 8c V5 build a4c7bcc5) compiled at 2024-01-16 14:49:56 commit 0 last mr 1632 )
Non-SSL connection (SSL connection is recommended when requiring high-security)
Last login: 2024-01-19 16:22:27+08 from [local] (user gbase)
Type "help" for help.

postgres=#
```

(8) 查看最大连接数。

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

示例 2：使用方式二设置数据库主节点的客户端认证最长时间参数“authentication_timeout”

(1) 以操作系统用户 gbase 登录数据库主节点。

(2) 使用如下命令连接数据库。

```
gsql -d postgres -p 5432
```

postgres 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：

```
gsql ((GBase 8c V5 build a4c7bcc5) compiled at 2024-01-16 14:49:56 commit 0 last mr 1632 )
Non-SSL connection (SSL connection is recommended when requiring high-security)
Last login: 2024-01-19 16:22:27+08 from [local] (user gbase)
Type "help" for help.

postgres=#
```

(3) 查看客户端认证的最长时间。

```
postgres=# SHOW authentication_timeout;
 authentication_timeout
------------------------
 1min
(1 row)
```

(4) 使用如下命令退出数据库。

```
postgres=# \q
```

(5) 修改数据库主节点的客户端认证最长时间。

```
gs_guc reload -N all -I all -c "authentication_timeout = 59s"
```

(6) 使用如下命令连接数据库。

```
gsql -d postgres -p 5432
```

postgres 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：

```
gsql ((GBase 8c V5 build a4c7bcc5) compiled at 2024-01-16 14:49:56 commit 0 last mr 1632 )
Non-SSL connection (SSL connection is recommended when requiring high-security)
Last login: 2024-01-19 16:22:27+08 from [local] (user gbase)
Type "help" for help.

postgres=#
```

(7) 查看客户端认证的最长时间。

```
postgres=# SHOW authentication_timeout;
 authentication_timeout
------------------------
```

```
 59s
(1 row)
```

示例 3：修改 GBase 8c 数据库节点的最大连接数。

(1) 以操作系统用户 gbase 登录数据库主节点。

(2) 使用如下命令连接数据库。

```
gsql -d postgres -p 5432
```

postgres 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：

```
gsql ((GBase 8c V5 build a4c7bcc5) compiled at 2024-01-16 14:49:56 commit 0 last mr 1632 )
Non-SSL connection (SSL connection is recommended when requiring high-security)
Last login: 2024-01-19 16:22:27+08 from [local] (user gbase)
Type "help" for help.

postgres=#
```

(3) 查看最大连接数。

```
postgres=# SHOW max_connections;
 max_connections
-----------------
 200
(1 row)
```

(4) 使用如下命令退出数据库。

```
postgres=# \q
```

(5) 修改数据库节点的最大连接数。

```
gs_guc set -N all -I all -c "max_connections = 500"
```

(6) 重启 GBase 8c。

```
gs_om -t stop
gs_om -t start
```

(7) 使用如下命令连接数据库。

```
gsql -d postgres -p 5432
```

postgres 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：

```
gsql ((GBase 8c V5 build a4c7bcc5) compiled at 2024-01-16 14:49:56 commit 0 last mr 1632 )
Non-SSL connection (SSL connection is recommended when requiring high-security)
Last login: 2024-01-19 16:22:27+08 from [local] (user gbase)
Type "help" for help.

postgres=#
```

(8) 查看最大连接数。

```
postgres=# SHOW max_connections;
 max_connections
-----------------
 500
(1 row)
```

示例 4：设置数据库节点的客户端认证最长时间参数“authentication_timeout”

(1) 以操作系统用户 gbase 登录数据库主节点。

(2) 使用如下命令连接数据库。

```
gsql -d postgres -p 5432
```

postgres 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：

```
gsql ((GBase 8c V5 build a4c7bcc5) compiled at 2024-01-16 14:49:56 commit 0 last mr 1632 )
Non-SSL connection (SSL connection is recommended when requiring high-security)
Last login: 2024-01-19 16:22:27+08 from [local] (user gbase)
Type "help" for help.

postgres=#
```

(3) 查看客户端认证的最长时间。

```
postgres=# SHOW authentication_timeout;
 authentication_timeout
------------------------
 1min
(1 row)
```

(4) 使用如下命令退出数据库。

```
postgres=# \q
```

(5) 修改数据库节点的客户端认证最长时间。

```
gs_guc reload -N all -I all -c "authentication_timeout = 30s"
```

(6) 使用如下命令连接数据库。

```
gsql -d postgres -p 5432
```

postgres 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：

```
gsql ((GBase 8c V5 build a4c7bcc5) compiled at 2024-01-16 14:49:56 commit 0 last mr 1632 )
Non-SSL connection (SSL connection is recommended when requiring high-security)
Last login: 2024-01-19 16:22:27+08 from [local] (user gbase)
Type "help" for help.

postgres=#
```

(7) 查看客户端认证的最长时间。

```
postgres=# SHOW authentication_timeout;
 authentication_timeout
------------------------
 30s
(1 row)
```

15 用户自定义函数

15.1 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。一些常见函数，例如 gs_extend_library。

应用 PL/pgSQL 创建函数的语法为 CREATE FUNCTION。PL/pgSQL 是一种可载入的过程语言。其应用方法与 18 存储过程相似，只是存储过程无返回值，函数有返回值。

16 存储过程

16.1 存储过程

商业规则和业务逻辑可以通过程序存储在 GBase 8c 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 CREATE PROCEDURE。

PL/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。

16.2 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。GBase 8c 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， GBase 8c 根据数据类型有相应函数对其内容进行操作，例如 GBase 8c 可对数值型 数据进行加、减、乘、除操作。

16.3 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 GBase 8c 提供的类型转换函 数，例如 CAST 进行数据类型强转。

GBase 8c 数据库常见的隐式类型转换，请参见表 15-18-1。

须知：

GBase 8c 支持的 DATE 的效限范围是：公元前 4713 年到公元 294276 年。

表 18-1 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	-
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	-
INT4	BOOLEAN	-
BOOLEAN	INT4	-

16.4 数组和 record

16.4.1 数组

16.4.1.1 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。定义方法如下。

```
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 GBase 8c 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，GBase 8c 会优先选择存储过程中定义的数组类型来声明数组变量。
- data_type 也可以为存储过程中定义的 record 类型（匿名块不支持），但不可以为存储过程中　定义的数组或集合类型。

GBase 8c 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、prior、exists、 trim、next、delete 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 语句推荐使用　中括号来访问数组元素，使用小括号默认识别为数组访问，若数组不存在，则识别为函数表达式。
- 如果 clob 类型大于 1GB，则存储过程中的 table of 类型、record 类型、clob 作为出入参、游标、raise info 等功能不支持。

16.4.2 集合

16.4.2.1　集合类型的使用

在使用集合之前，需要自定义一个集合类型。

在存储过程中紧跟 AS 关键字后面定义集合类型。定义方法如下。

其中：

- table_type：要定义的集合类型名。
- TABLE：表示要定义集合类型。
- data_type：要创建的集合中成员的类型。
- indexby_type：　创建集合索引的类型。

📖 说明

- 在 GBase 8c 中，集合会自动增长，访问越界会返回一个 NULL，不会报错。

- 在存储过程中定义的集合类型，其作用域仅在该存储过程中。
- 索引的类型仅支持 integer 和 varchar 类型，其中 varchar 的长度暂不约束。
- NOT NULL 只支持语法不支持功能。
- data_type 可以为存储过程内定义的 record 类型，集合类型（匿名块不支持），不可以为数组　类型。
- 不支持跨 package 的嵌套集合类型变量使用。
- 不支持 record 嵌套 table of index by 类型的变量作为存储过程的出入参。
- 不支持 table of index by 类型的变量作为函数的出入参。
- 不支持通过 raise info 打印整个嵌套 table of 变量。
- 不支持跨自治事务传递 table of 变量。
- 不支持存储过程的出入参定义为嵌套 table of 类型。

GBase 8c 支持使用圆括号来访问集合元素，且还支持一些特有的函数，如 extend，count，first，last, prior, next, delete 来访问集合的内容。

集合函数支持 multiset union/intersect/except all/distinct 函数。

说明

同一个表达式里不支持两个以上 table of index by 类型变量的函数调用。

16.4.3 record

16.4.3.1 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。

16.4.3.2 语法

record 类型的语法参见图 17-1。

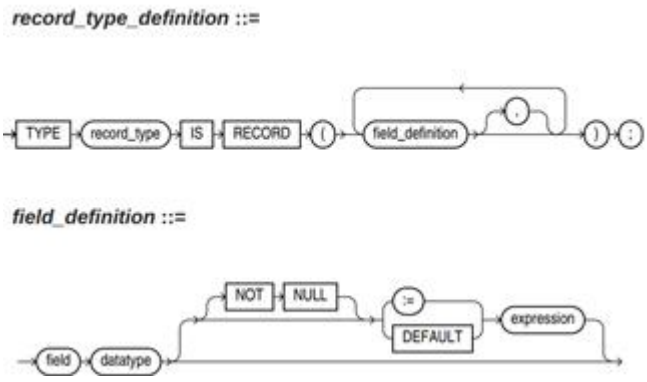


图 17-1 record 类型的语法

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 GBase 8c 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。
- datatype 可以为存储过程中定义 record 类型、数组类型和集合类型(匿名块不支持)。

16.4.3.3 示例

下面示例中用到的表定义如下：

```
postgres=# \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)|
--演示在函数中对数组进行操作。
postgres=# 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;
--调用该函数。
postgres=# CALL regress_record('abc');
--删除函数。
postgres=# DROP FUNCTION regress_record;
```

16.5 声明语法

16.5.1 基本结构

16.5.1.1 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 gsql 工具带“-r”参数执行 PL/SQL 块时出现异常。

16.5.1.2 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 17-2。
- 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。

16.5.2 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。

16.5.2.1 语法

匿名块的语法参见图 17-2。

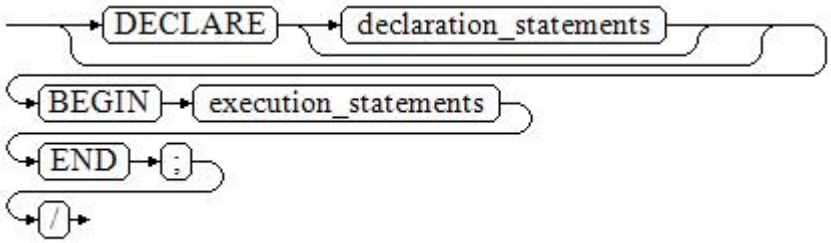


图 17-2 anonymous_block::=语法

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。

16.5.3 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。

16.6 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。

16.6.1 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。

16.6.1.1 变量声明

变量声明语法请参见图 17-1。

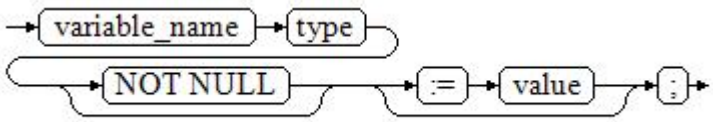


图 17-1 declare_variable::=

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

```
postgres=# 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。

```
TYPE employee_record is record (id INTEGER, firstname VARCHAR2(20));
my_employee employee_record;
my_id my_employee.id%TYPE;
my_id_copy my_id%TYPE;
```

%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
```

同样可以使用在 cursor 上面，该组数据的字段名称与字段类型都与 employee 表相同（对于 PACKAGE 中的 cursor，可以省略%ROWTYPE）。%TYPE 也可以引用 cursor 中某一列的类型，我们可以通过如下定义：

```
cursor cur is select * from employee;
my_employee cur%ROWTYPE
my_name cur.firstname%TYPE
```

```
my_employee2 cur -- 对于 PACKAGE 中定义的 cursor，可以省略%ROWTYPE 字段
```

须知： - %TYPE 不支持引用复合类型或 RECORD 类型变量的类型、RECORD 类型的某列类型、跨 PACKAGE 复合类型变量的某列类型、跨 PACKAGE cursor 变量的某列类型等。 - %ROWTYPE 不支持引用复合类型或 RECORD 类型变量的类型、跨 PACKAGE cursor 的类型。

16.6.1.2 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。

16.6.2 赋值语句

16.6.2.1 语法

给变量赋值的语法请参见图 17-2。

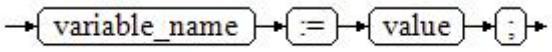


图 17-2 assignment_value::=

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：

```
postgres=# DECLARE
    emp_id    INTEGER := 7788;--赋值
BEGIN
    emp_id := 5;--赋值
    emp_id := 5*7784;
END;
/
```

16.6.2.2 嵌套赋值

给变量嵌套赋值的语法请参见图 17-3。

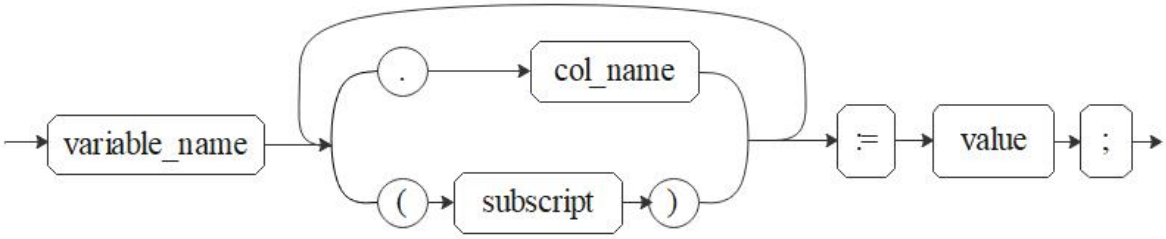


图 17-3 nested_assignment_value::=

对以上语法格式的解释如下：

- variable_name：变量名。
- col_name：列名。
- subscript：下标，针对数组变量使用，可以是值或表达式，类型必须为 int。
- 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：

```
postgres=#CREATE TYPE o1 as (a int, b int);
postgres=# DECLARE
```

```
    TYPE r1 is VARRAY(10) of o1;
    emp_id   r1;
BEGIN
    emp_id(1).a := 5;--赋值
    emp_id(1).b := 5*7784;
END;
/
```

须知

- INTO 方式赋值仅支持对第一层列赋值，且不支持二维及以上数组；
- 引用嵌套的列值时，若存在数组下标，目前仅支持在前三层列中只存在一个小括号情况，建议使用方括号[ ]引用下标；

16.6.2.3 INTO/BULK COLLECT INTO

将存储过程内语句返回的值存储到变量内，BULK COLLECT INTO 允许将部分或全部返回值暂存到数组内部。

示例：

```
postgres=# DECLARE
    my_id integer;
BEGIN
    select id into my_id from customers limit 1; --  赋值
END;
/
postgres=# DECLARE
    type id_list is varray(6) of customers.id%type;
    id_arr id_list;
```

```
BEGIN
    select id bulk collect into id_arr from customers order by id DESC limit 20; -- 批量赋值
END;
/
```

须知： BULK COLLECT INTO 只支持批量赋值给数组。合理使用 LIMIT 字段避免操作过量数据导致性能下降。

16.6.3 调用语句

16.6.3.1 语法

调用一个语句的语法请参见图 1。

图 1 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。

16.6.3.2 示例

```
--创建存储过程 proc_staffs
postgres=# 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 hr.staffs where section_id = section;

END;

/

--调用存储过程 proc_return.

postgres=# CALL proc_staffs(2,8,6);

--清除存储过程

postgres=# DROP PROCEDURE proc_staffs;
```

16.7 动态语句

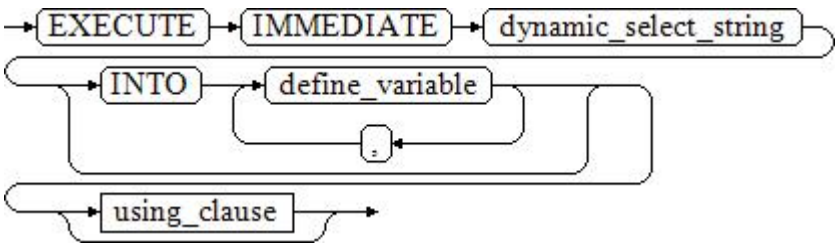
16.7.1 执行动态查询语句

介绍执行动态查询语句。GBase 8c 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。

16.7.1.1 EXECUTE IMMEDIATE

语法图请参见图 18-7。

图 18-7 EXECUTE IMMEDIATE dynamic_select_clause::=



using_clause 子句的语法图参见图 18-8。

图 18-8 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。

16.7.1.2 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 18-9。

图 18-9 open_for::=

参数说明：

- cursor_name：要打开的游标名。
- dynamic_string：动态查询语句。

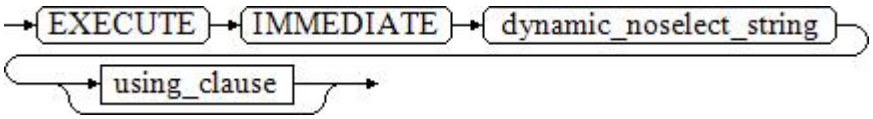
- USING value：在 dynamic_string 中存在占位符时使用。游标的使用请参考游标。

16.7.2 执行动态非查询语句

16.7.2.1 语法

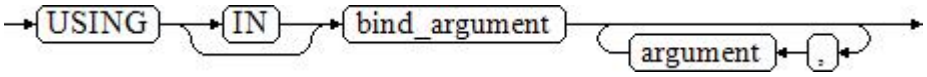
语法请参见图 18-10。

图 18-10 noselect::=



using_clause 子句的语法参见图 18-11。

图 18-11 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。

16.7.2.2 示例

```
--创建表

postgres=# CREATE TABLE sections_t1 (

section    NUMBER(4) , section_name VARCHAR2(30), manager_id NUMBER(6),
place_id  NUMBER(4)

);
```

```
--声明变量 postgres=# 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

postgres=# SELECT * FROM sections_t1;

--删除表

postgres=# DROP TABLE sections_t1;
```

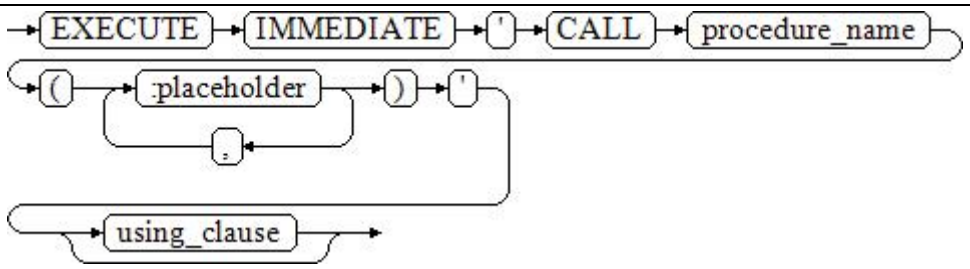
16.7.3 动态调用存储过程

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。

16.7.3.1 语法

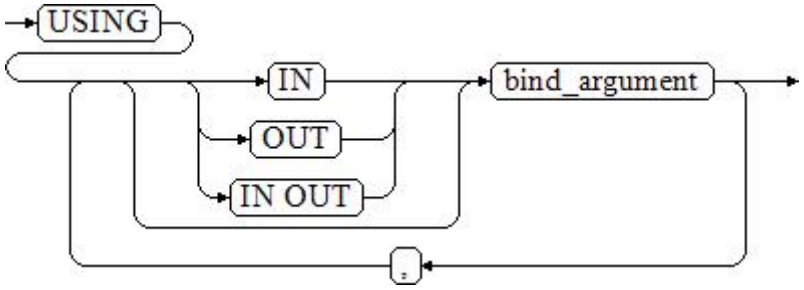
语法请参见图 18-12。

图 18-12 call_procedure::=



using_clause 子句的语法参见图 18-13。

图 18-13 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。

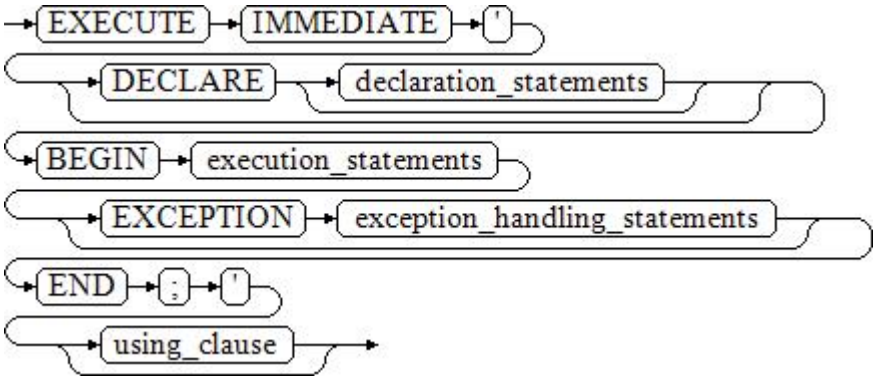
16.7.4 动态调用匿名块

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。

16.7.4.1 语法

语法请参见图 18-14。

图 18-14 call_anonymous_block::=



using_clause 子句的语法参见图 18-15。

图 18-15 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 GBase 8c 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。

16.8 控制语句

16.8.1 返回语句

GBase 8c 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。

16.8.1.1 RETURN

语法

返回语句的语法请参见图 18-16。

图 18-16 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。

16.8.1.2 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

```
postgres=# CREATE TABLE t1(a int); postgres=# INSERT INTO t1 VALUES(1),(10);

--RETURN NEXT

postgres=# 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;

postgres=# call fun_for_return_next(); a

--- 1

10

(2 rows)

-- RETURN QUERY

postgres=# 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;

postgres=# call fun_for_return_query(); a

---

1

10

(2 rows)
```

16.8.2 条件语句

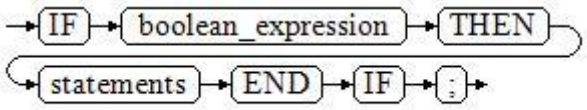
条件语句的主要作用是判断参数或者语句是否满足已给定的条件，根据判定结果执行相

应的操作。

GBase 8c 有五种形式的 IF：

- **IF_THEN**

图 1 IF_THEN::=



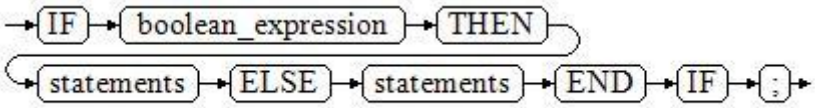
IF_THEN 语句是 IF 的最简单形式。如果条件为真，statements 将被执行。否则，将忽略它们的结果使该 IF_THEN 语句执行结束。

示例

```
postgres=# IF v_user_id <> 0 THEN
    UPDATE users SET email = v_email WHERE user_id = v_user_id;
END IF;
```

- **IF_THEN_ELSE**

图 2 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

```
postgres=# 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**

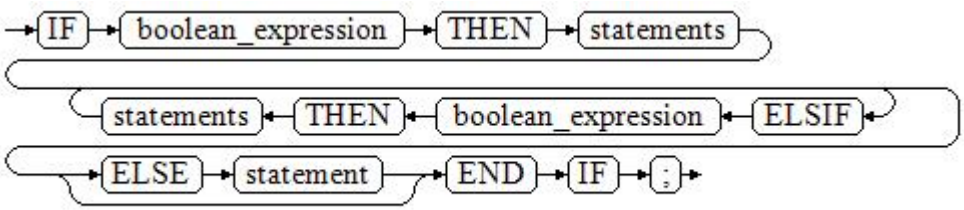
IF 语句可以嵌套，嵌套方式如下：

```
postgres=# IF sex = 'm' THEN
    pretty_sex := 'man';
ELSE
    IF sex = 'f' THEN
        pretty_sex := 'woman';
    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 3 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。

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);
```

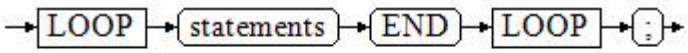
```
--删除存储过程

DROP PROCEDURE proc_control_structure;
```

16.8.3 循环语句

16.8.3.1 简单 LOOP 语句

语法图



示例

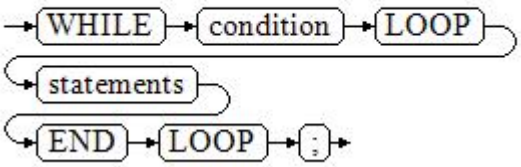
```
CREATE OR REPLACE PROCEDURE proc_loop(i in integer, count out integer)
AS
    BEGIN
        count:=0;
        LOOP
        IF count > i THEN
            raise info 'count is %. ', count;
            EXIT;
        ELSE
            count:=count+1;
        END IF;
        END LOOP;
    END;
/
```

```
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。

16.8.3.2 WHILE_LOOP 语句

语法图



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;
```

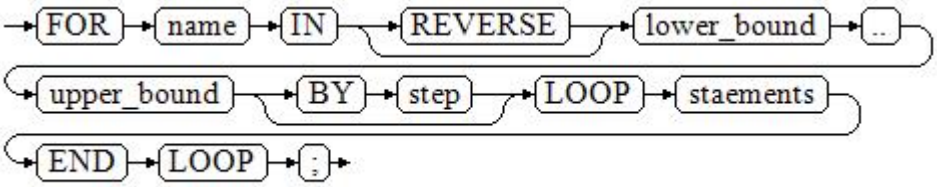
```
/


--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;
```

16.8.3.3 FOR_LOOP（integer 变量）语句

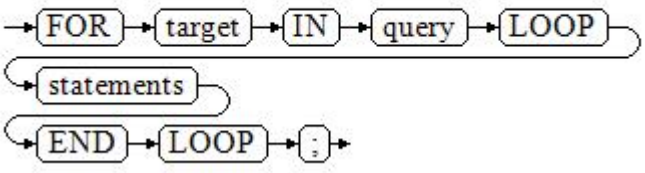
语法图



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。

16.8.3.4 FOR_LOOP 查询语句

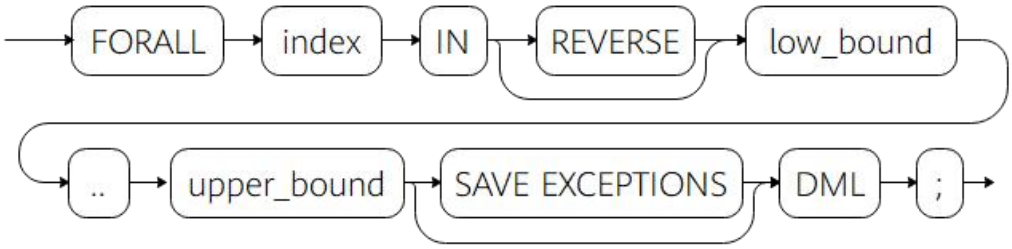
语法图



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。

16.8.3.5 FORALL 批量查询语句

语法图



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。 - 如果声明了 SAVE EXCEPTIONS，则会将循环体 DML 执行过程中每次遇到的异常保存在 SQL&BULK_EXCEPTIONS 中，并在执行结束后统一抛出一个异常，循环过程中没有异常的执行的结果在当前子事务内不会回滚。

示例

```
CREATE TABLE hdfs_t1 (
    title NUMBER(6),
    did VARCHAR2(20),
    data_period 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();

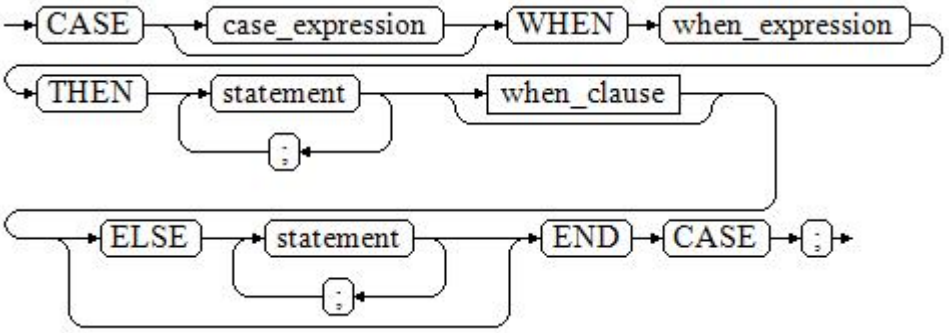

--查询存储过程调用结果
SELECT * FROM hdfs_t1 WHERE title BETWEEN 100 AND 120;


--删除存储过程和表
DROP PROCEDURE proc_forall;
DROP TABLE hdfs_t1;
```

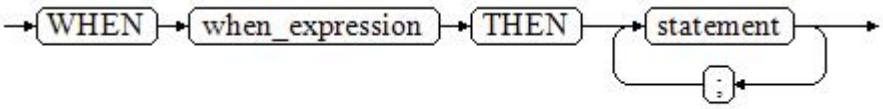
16.8.4 分支语句

语法

分支语句的语法请参见下图。



when_clause 子句的语法图参见下图。



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;
```

16.8.5 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;
/
```

16.8.6 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。错误码可以捕获同一类的其他错误码。

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');

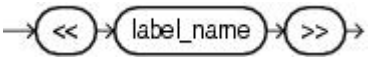
--删除 FUNCTION 和 TABLE
DROP FUNCTION merge_db;
DROP TABLE db;
```

16.8.7 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

```
postgres=# 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;
/
```

16.9 事务管理

存储过程本身就处于一个事务中，开始调用最外围存储过程时会自动开启一个事务，在调用结束时自动提交或者发生异常时回滚。除了系统自动的事务控制外，也可以使用 COMMIT/ROLLBACK 来控制存储过程中的事务。在存储过程中调用 COMMIT/ROLLBACK 命令，将提交/回滚当前事务并自动开启一个新的事务，后续的所有操作都会在此新事务中运行。

保存点 SAVEPOINT 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。存储过程中允许使用保存点来进行事务管

理，当前支持保存点的创建、回滚和释放操作。存储过程中使用回滚保存点只是回退当前事务的修改，而不会改变存储过程的执行流程，也不会回退存储过程中的局部变量值等。

16.9.1 语法格式

定义保存点

```
SAVEPOINT savepoint_name;
```

回滚保存点

```
ROLLBACK TO [SAVEPOINT] savepoint_name;
```

释放保存点

```
RELEASE [SAVEPOINT] savepoint_name;
```

16.9.2 使用场景

支持调用的上下文环境：

- 支持在 PLSQL 的存储过程内使用 COMMIT/ROLLBACK/SAVEPOINT。
- 支持含有 EXCEPTION 的存储过程使用 COMMIT/ROLLBACK/SAVEPOINT。
- 支持在存储过程的 EXCEPTION 语句内使用 COMMIT/ROLLBACK/SAVEPOINT。
- 支持在事务块里调用含有 COMMIT/ROLLBACK/SAVEPOINT 的存储过程，即通过/BEGIN/START/END 等开启控制的外部事务。
- 支持在子事务中调用含有 SAVEPOINT 的存储过程，即存储过程中使用外部定义的 SAVEPOINT，回退事务状态到存储过程外定义的 SAVEPOINT 位置。
- 支持存储过程外部对存储过程内定义的 SAVEPOINT 可见，即存储过程外可以将事务修改回滚到存储过程中定义 SAVEPOINT 的位置。
- 支持多数 PLSQL 的上下文和语句内调用 COMMIT/ROLLBACK/SAVEPOINT，包括常用的 IF/FOR/CURSOR LOOP/WHILE。
- 支持存储过程返回值与简单表达式计算中调用含有 COMMIT/ROLLBACK/SAVEPOINT 的存储过程或者函数。

支持提交/回滚的内容：

- 支持 DDL 在 COMMIT/ROLLBACK 后的提交/回滚。
- 支持 DML 的 COMMIT/ROLLBACK 后的提交。
- 支持存储过程内 GUC 参数的回滚提交。

16.9.3 使用限制

不支持调用的上下文环境：

- 不支持除 PLSQL 的其他存储过程中调用 COMMIT/ROLLBACK/SAVEPOINT，例如 PLJAVA、PLPYTHON 等。
- 不支持函数中调用 COMMIT/ROLLBACK/SAVEPOINT，包括函数调用含有 COMMIT/ROLLBACK/SAVEPOINT 的存储过程。
- 不支持事务块中调用了 SAVEPOINT 后，调用含有 COMMIT/ROLLBACK 的存储过程。
- 不支持 TRIGGER 中调用含有 COMMIT/ROLLBACK/SAVEPOINT 语句的存储过程。
- 不支持 EXECUTE 语句中调用 COMMIT/ROLLBACK/SAVEPOINT 语句。
- 不支持在 CURSOR 语句中打开一个含有 COMMIT/ROLLBACK/SAVEPOINT 的存储过程。
- 不支持带有 IMMUTABLE 以及 SHIPPABLE 的存储过程调用 COMMIT/ROLLBACK/SAVEPOINT，或调用带有 COMMIT/ROLLBACK/SAVEPOINT 语句的存储过程。
- 不支持 SQL 中调用含有 COMMIT/ROLLBACK/SAVEPOINT 语句的存储过程，除了 SELECT PROC 以及 CALL PROC。
- 存储过程头带有 GUC 参数设置的不允许调用 COMMIT/ROLLBACK/SAVEPOINT 语句。
- 不支持 CURSOR/EXECUTE 语句，以及各类表达式内调用 COMMIT/ROLLBACK/SAVEPOINT。

- 自治事务和存储过程事务是两个独立的事务，不能互相使用对方事务中定义的保存点。

不支持提交回滚的内容：

- 不支持存储过程内声明变量以及传入变量的提交/回滚。
- 不支持存储过程内必须重启生效的 GUC 参数的提交/回滚。

16.9.4 示例

示例 1：支持在 PLSQL 的存储过程内使用 COMMIT/ROLLBACK。

```
CREATE TABLE EXAMPLE1(COL1 INT);

CREATE OR REPLACE PROCEDURE TRANSACTION_EXAMPLE()
AS
BEGIN
    FOR i IN 0..20 LOOP
        INSERT INTO EXAMPLE1(COL1) VALUES (i);
        IF i % 2 = 0 THEN
            COMMIT;
        ELSE
            ROLLBACK;
        END IF;
    END LOOP;
END;
/
```

示例 2：

支持含有 EXCEPTION 的存储过程使用 COMMIT/ROLLBACK。

支持在存储过程的 EXCEPTION 语句内使用 COMMIT/ROLLBACK。

支持 DDL 在 COMMIT/ROLLBACK 后的提交/回滚。

```
CREATE OR REPLACE PROCEDURE TEST_COMMIT_INSERT_EXCEPTION_ROLLBACK()
AS
BEGIN
    DROP TABLE IF EXISTS TEST_COMMIT;
    CREATE TABLE TEST_COMMIT(A INT, B INT);
    INSERT INTO TEST_COMMIT SELECT 1, 1;
    COMMIT;
```

```
        CREATE TABLE TEST_ROLLBACK(A INT, B INT);
    RAISE EXCEPTION 'RAISE EXCEPTION AFTER COMMIT';
EXCEPTION
    WHEN OTHERS THEN
    INSERT INTO TEST_COMMIT SELECT 2, 2;
    ROLLBACK;
END;
/
```

示例 3：支持在事务块里调用含有 COMMIT/ROLLBACK 的存储过程，即通过/BEGIN/START/END 等开启控制的外部事务。

```
    BEGIN;
        CALL TEST_COMMIT_INSERT_EXCEPTION_ROLLBACK();
    END;
```

示例 4：支持多数 PLSQL 的上下文和语句内调用 COMMIT/ROLLBACK，包括常用的 IF/FOR/CURSOR LOOP/WHILE。

```
CREATE OR REPLACE PROCEDURE TEST_COMMIT2()
IS
BEGIN
    DROP TABLE IF EXISTS TEST_COMMIT;
    CREATE TABLE TEST_COMMIT(A INT);
    FOR I IN REVERSE 3..0 LOOP
    INSERT INTO TEST_COMMIT SELECT I;
    COMMIT;
    END LOOP;
    FOR I IN REVERSE 2..4 LOOP
    UPDATE TEST_COMMIT SET A=I;
    COMMIT;
    END LOOP;
EXCEPTION
WHEN OTHERS THEN
    INSERT INTO TEST_COMMIT SELECT 4;
    COMMIT;
END;
/
```

示例 5：支持存储过程返回值与简单表达式计算。

```
CREATE OR REPLACE PROCEDURE exec_func3(RET_NUM OUT INT)
AS
BEGIN
  RET_NUM := 1+1;
COMMIT;
END;
/
CREATE OR REPLACE PROCEDURE exec_func4(ADD_NUM IN INT)
AS
SUM_NUM INT;
BEGIN
SUM_NUM := ADD_NUM + exec_func3();
COMMIT;
END;
/
```

示例 6：支持存储过程内 GUC 参数的回滚提交。

```
SHOW explain_perf_mode;
SHOW enable_force_vector_engine;

CREATE OR REPLACE PROCEDURE GUC_ROLLBACK()
AS
BEGIN
    SET enable_force_vector_engine = on;
    COMMIT;
    SET explain_perf_mode TO pretty;
    ROLLBACK;
END;
/

call GUC_ROLLBACK();
SHOW explain_perf_mode;
SHOW enable_force_vector_engine;
SET enable_force_vector_engine = off;
```

示例 7：函数（Function）中不允许调用 commit/rollback 语句，同时不允许函数调用含有 commit/rollback 的存储过程。

```
CREATE OR REPLACE FUNCTION FUNCTION_EXAMPLE1() RETURN INT
AS
EXP INT;
```

```
BEGIN
    FOR i IN 0..20 LOOP
        INSERT INTO EXAMPLE1(col1) VALUES (i);
        IF i % 2 = 0 THEN
            COMMIT;
        ELSE
            ROLLBACK;
        END IF;
    END LOOP;
    SELECT COUNT(*) FROM EXAMPLE1 INTO EXP;
    RETURN EXP;
END;
/
```

示例 8：函数（Fucntion）中不允许调用带有 commit/rollback 语句的存储过程。

```
CREATE OR REPLACE FUNCTION FUNCTION_EXAMPLE2() RETURN INT
AS
EXP INT;
BEGIN
    --transaction_example 为存储过程，带有 commit/rollback 语句
    CALL transaction_example();
    SELECT COUNT(*) FROM EXAMPLE1 INTO EXP;
    RETURN EXP;
END;
/
```

示例 9：不允许 Trigger 的存储过程包含 commit/rollback 语句，或调用带有 commit/rollback 语句的存储过程。

```
CREATE OR REPLACE FUNCTION FUNCTION_TRI_EXAMPLE2() RETURN TRIGGER
AS
EXP INT;
BEGIN
    FOR i IN 0..20 LOOP
        INSERT INTO EXAMPLE1(col1) VALUES (i);
        IF i % 2 = 0 THEN
            COMMIT;
        ELSE
            ROLLBACK;
        END IF;
    END LOOP;
```

```
    SELECT COUNT(*) FROM EXAMPLE1 INTO EXP;
END;
/

CREATE TRIGGER TRIGGER_EXAMPLE AFTER DELETE ON EXAMPLE1
FOR EACH ROW EXECUTE PROCEDURE FUNCTION_TRI_EXAMPLE2();

DELETE FROM EXAMPLE1;
```

示例 10：不支持带有 IMMUABLE 以及 SHIPPABLE 的存储过程调用 commit/rollback，或调用带有 commit/rollback 语句的存储过程。

```
CREATE OR REPLACE PROCEDURE TRANSACTION_EXAMPLE1()
IMMUTABLE
AS
BEGIN
    FOR i IN 0..20 LOOP
        INSERT INTO EXAMPLE1 (col1) VALUES (i);
        IF i % 2 = 0 THEN
            COMMIT;
        ELSE
            ROLLBACK;
        END IF;
    END LOOP;
END;
/
```

示例 11：不支持出现在 SQL 中的调用（除了 Select Procedure）。

```
CREATE OR REPLACE PROCEDURE TRANSACTION_EXAMPLE3()
AS
BEGIN
    FOR i IN 0..20 LOOP
        INSERT INTO EXAMPLE1 (col1) VALUES (i);
        IF i % 2 = 0 THEN
            EXECUTE IMMEDIATE 'COMMIT';
        ELSE
            EXECUTE IMMEDIATE 'ROLLBACK';
        END IF;
    END LOOP;
END;
/
```

示例 12：存储过程头带有 GUC 参数设置的不允许调用 commit/rollback 语句。

```
CREATE OR REPLACE PROCEDURE TRANSACTION_EXAMPLE4()
SET ARRAY_NULLS TO "ON"
AS
BEGIN
    FOR i IN 0..20 LOOP
        INSERT INTO EXAMPLE1 (col1) VALUES (i);
        IF i % 2 = 0 THEN
            COMMIT;
        ELSE
            ROLLBACK;
        END IF;
    END LOOP;
END;
/
```

示例 13：游标 open 的对象不允许为带有 commit/rollback 语句的存储过程。

```
CREATE OR REPLACE PROCEDURE TRANSACTION_EXAMPLE5(INTIN IN INT, INTOUT OUT INT)
AS
BEGIN
INTOUT := INTIN + 1;
COMMIT;
END;
/

CREATE OR REPLACE PROCEDURE TRANSACTION_EXAMPLE6()
AS
CURSOR CURSOR1(EXPIN INT)
IS SELECT TRANSACTION_EXAMPLE5(EXPIN);
INTEXP INT;
BEGIN
    FOR i IN 0..20 LOOP
        OPEN CURSOR1(i);
        FETCH CURSOR1 INTO INTEXP;
        INSERT INTO EXAMPLE1(COL1) VALUES (INTEXP);
        IF i % 2 = 0 THEN
            COMMIT;
        ELSE
            ROLLBACK;
        END IF;
```

```
        CLOSE CURSOR1;
    END LOOP;
END;
/
```

示例 14：不支持 CURSOR/EXECUTE 语句，以及各类表达式内调用 COMMIT/ROLLBACK。

```
CREATE OR REPLACE PROCEDURE exec_func1()
AS
BEGIN
    CREATE TABLE TEST_exec(A INT);
COMMIT;
END;
/
CREATE OR REPLACE PROCEDURE exec_func2()
AS
BEGIN
EXECUTE exec_func1();
COMMIT;
END;
/
```

示例 15：存储过程使用保存点回退事务部分修改。

```
CREATE OR REPLACE PROCEDURE STP_SAVEPOINT_EXAMPLE1()
AS
BEGIN
    INSERT INTO EXAMPLE1 VALUES(1);
    SAVEPOINT s1;
    INSERT INTO EXAMPLE1 VALUES(2);
    ROLLBACK TO s1;  -- 回退插入记录 2
    INSERT INTO EXAMPLE1 VALUES(3);
END;
/
```

示例 16：存储过程中使用保存点回退到存储过程外部定义的保存点。

```
CREATE OR REPLACE PROCEDURE STP_SAVEPOINT_EXAMPLE2()
AS
BEGIN
    INSERT INTO EXAMPLE1 VALUES(2);
    ROLLBACK TO s1;  -- 回退插入记录 2
```

```
    INSERT INTO EXAMPLE1 VALUES(3);
END;
/

BEGIN;
INSERT INTO EXAMPLE1 VALUES(1);
SAVEPOINT s1;
CALL STP_SAVEPOINT_EXAMPLE2();
SELECT * FROM EXAMPLE1;
COMMIT;
```

示例 17：存储过程外部回退到存储过程中定义的保存点。

```
CREATE OR REPLACE PROCEDURE STP_SAVEPOINT_EXAMPLE3()
AS
BEGIN
    INSERT INTO EXAMPLE1 VALUES(1);
    SAVEPOINT s1;
    INSERT INTO EXAMPLE1 VALUES(2);
END;
/

BEGIN;
INSERT INTO EXAMPLE1 VALUES(3);
CALL STP_SAVEPOINT_EXAMPLE3();
ROLLBACK TO SAVEPOINT s1; --回退存储过程中插入记录 2
SELECT * FROM EXAMPLE1;
COMMIT;
```

16.10 其他语句

16.10.1锁操作

GBase 8c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 GBase 8c 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。

16.10.2 游标操作

GBase 8c 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显式游标。

16.11 游标

16.11.1 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 17-1。

表 17-1 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

16.11.2 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。

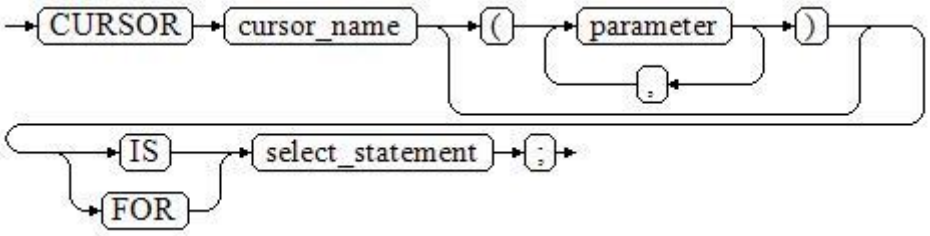
16.11.2.1 处理步骤

显式游标处理需六个 PL/SQL 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 1。

图 1 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 17-4 和图 17-5。

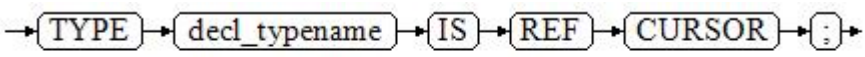


图 17-4 cursor_typename::=

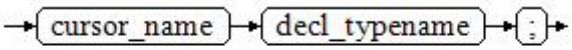


图 17-5 dynamic_cursor_define::=

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 17-6。

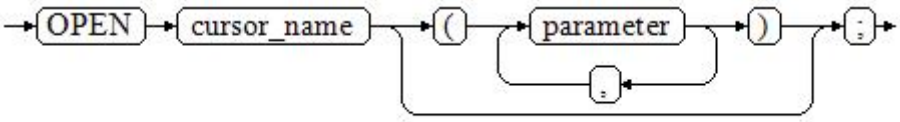


图 17-6 open_static_cursor::=

   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 17-7。

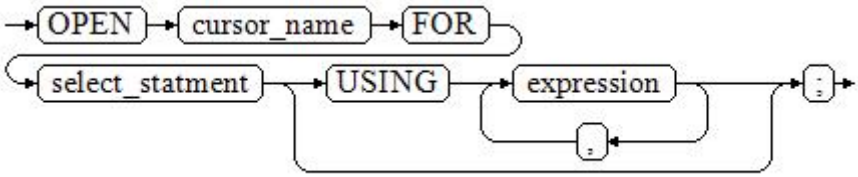


图 17-7 open_dynamic_cursor::=

   PL/SQL 程序不能用 OPEN 语句重复打开一个游标。

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 17-8。

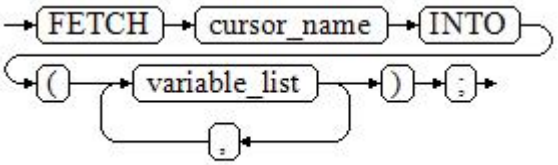


图 17-8 fetch_cursor::=

4. 对该记录进行处理。

5. 继续处理，直到活动集合中没有记录。

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 17-9。

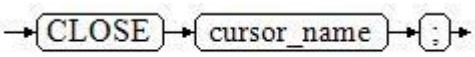


图 17-9 close_cursor::=

16.11.2.2 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。

16.11.3 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。

16.11.3.1 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。
- 兼容 O 模式下，GUC 参数 behavior_compat_options 为 compat_cursor 时，隐式游标跨存储过程有效。

16.11.3.2 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。

16.11.3.3 示例

```
--删除员工表 hr.staffs 表中某部门的所有员工，如果该部门中已没有员工，则在部门表 hr.sections 中删除该部门。

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 WHERE section_ID = V_DEPTNO;
        END IF;
    END;
/

CALL proc_cursor3();

--删除存储过程和临时表
DROP PROCEDURE proc_cursor3;
```

16.11.4 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。

16.11.4.1 语法

FOR AS 循环的语法请参见图 17-10。

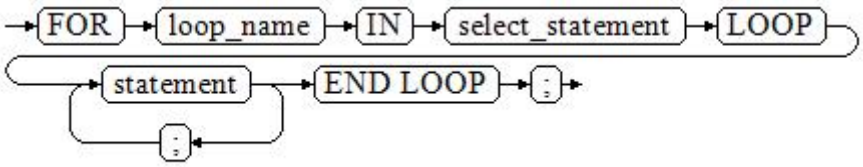


图 17-10 FOR_AS_loop::=

16.11.4.2 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 GBase 8c 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。

16.12 高级包

本章介绍高级包的基础接口。

16.12.1基础接口

16.12.1.1 PKG_SERVICE

PKG_SERVICE 支持的所有接口请参见表 17-1。

表 17-1 PKG_SERVICE

接口名称	描述
PKG_SERVICE.SQL_IS_CONTEXT_ACTIVE	确认该 CONTEXT 是否已注册。
PKG_SERVICE.SQL_CLEAN_ALL_CONTEXTS	取消所有注册的 CONTEXT。
PKG_SERVICE.SQL_REGISTER_CONTEXT	注册一个 CONTEXT。
PKG_SERVICE.SQL_UNREGISTER_CONTEXT	取消注册该 CONTEXT。
PKG_SERVICE.SQL_SET_SQL	向 CONTEXT 设置一条 SQL 语句，目前只支持 SELECT。
PKG_SERVICE.SQL_RUN	在一个 CONTEXT 上执行设置的 SQL 语句。
PKG_SERVICE.SQL_NEXT_ROW	读取该 CONTEXT 中的下一行数据。
PKG_SERVICE.SQL_GET_VALUE	读取该 CONTEXT 中动态定义的列值
PKG_SERVICE.SQL_SET_RESULT_TYPE	根据类型 OID 动态定义该 CONTEXT 的一个列。
PKG_SERVICE.JOB_CANCEL	通过任务 ID 来删除定时任务。
PKG_SERVICE.JOB_FINISH	禁用或者启用定时任务。
PKG_SERVICE.JOB_SUBMIT	提交一个定时任务。作业号由系统自动生成

接口名称	描述
	或由用户指定。
PKG_SERVICE.JOB_UPDATE	修改定时任务的属性，包括任务内容、下次执行时间、执行间隔。
PKG_SERVICE.SUBMIT_ON_NODES	提交一个任务到所有节点，作业号由系统自动生成。
PKG_SERVICE.ISUBMIT_ON_NODES	提交一个任务到所有节点，作业号由用户指定。
PKG_SERVICE.SQL_GET_ARRAY_RESULT	获取该 CONTEXT 中返回的数组值。
PKG_SERVICE.SQL_GET_VARIABLE_RESULT	获取该 CONTEXT 中返回的列值。

- **PKG_SERVICE.SQL_IS_CONTEXT_ACTIVE**

该函数用来确认一个 CONTEXT 是否已注册。该函数传入想查找的 CONTEXT ID，如果该 CONTEXT 存在返回 TRUE，反之返回 FALSE。

PKG_SERVICE.SQL_IS_CONTEXT_ACTIVE 函数原型为：

```
PKG_SERVICE.SQL_IS_CONTEXT_ACTIVE(
 context_id     IN INTEGER
)
RETURN BOOLEAN;
```

表 17-2 PKG_SERVICE.SQL_IS_CONTEXT_ACTIVE 接口说明

参数名称	描述
context_id	想查找的 CONTEXT ID 号

- **PKG_SERVICE.SQL_CLEAN_ALL_CONTEXTS**

该函数用来取消所有 CONTEXT

PKG_SERVICE.SQL_CLEAN_ALL_CONTEXTS 函数原型为：

```
PKG_SERVICE.SQL_CLEAN_ALL_CONTEXTS(
```

```
)
RETURN VOID;
```

- **PKG_SERVICE.SQL_REGISTER_CONTEXT**

该函数用来打开一个 CONTEXT，是后续对该 CONTEXT 进行各项操作的前提。该函数不传入任何参数，内部自动递增生成 CONTEXT ID，并作为返回值返回给 integer 定义的变量。

PKG_SERVICE.SQL_REGISTER_CONTEXT 函数原型为：

```
DBE_SQL.REGISTER_CONTEXT(
)
RETURN INTEGER;
```

- **PKG_SERVICE.SQL_UNREGISTER_CONTEXT**

该函数用来关闭一个 CONTEXT，是该 CONTEXT 中各项操作的结束。如果在存储过程结束时没有调用该函数，则该 CONTEXT 占用的内存仍然会保存，因此关闭 CONTEXT 非常重要。由于异常情况的发生会中途退出存储过程，导致 CONTEXT 未能关闭，因此建议存储过程中有异常处理，将该接口包含在内。

PKG_SERVICE.SQL_UNREGISTER_CONTEXT 函数原型为：

```
PKG_SERVICE.SQL_UNREGISTER_CONTEXT(
 context_id     IN INTEGER
)
RETURN INTEGER;
```

表 17-3 PKG_SERVICE.SQL_UNREGISTER_CONTEXT 接口说明

参数名称	描述
context_id	打算关闭的 CONTEXT ID 号

- **PKG_SERVICE.SQL_SET_SQL**

该函数用来解析给定游标的查询语句，被传入的查询语句会立即执行。目前仅支持 SELECT 查询语句的解析，且语句参数仅可通过 text 类型传递，长度不大于 1G。

PKG_SERVICE.SQL_SET_SQL 函数的原型为：

```
PKG_SERVICE.SQL_SET_SQL(
context_id      IN INTEGER,
query_string    IN TEXT,
language_flag   IN INTEGER
)
RETURN BOOLEAN;
```

表 17-4 PKG_SERVICE.SQL_SET_SQL 接口说明

参数名称	描述
context_id	执行查询语句解析的 CONTEXT ID
query_string	执行的查询语句
language_flag	版本语言号，目前只支持 1

- **PKG_SERVICE.SQL_RUN**

该函数用来执行一个给定的 CONTEXT。该函数接收一个 CONTEXT ID，运行后获得的数据用于后续操作。目前仅支持 SELECT 查询语句的执行。

PKG_SERVICE.SQL_RUN 函数的原型为：

```
PKG_SERVICE.SQL_RUN(
context_id      IN INTEGER,
)
RETURN INTEGER;
```

表 17-5 PKG_SERVICE.SQL_RUN 接口说明

参数名称	描述
context_id	执行查询语句解析的 CONTEXT ID

- **PKG_SERVICE.SQL_NEXT_ROW**

该函数返回符合查询条件的数据行数，每一次运行该接口都会获取到新的行数的集合，直到数据读取完毕获取不到新行为止。

PKG_SERVICE.SQL_NEXT_ROW 函数的原型为：

```
PKG_SERVICE.SQL_NEXT_ROW(
```

```
context_id      IN INTEGER,
)
RETURN INTEGER;
```

表 17-6 PKG_SERVICE.SQL_NEXT_ROW 接口说明

参数名称	描述
context_id	执行的 CONTEXT ID

- **PKG_SERVICE.SQL_GET_VALUE**

该函数用来返回给定 CONTEXT 中给定位置的 CONTEXT 元素值，该接口访问的是 PKG_SERVICE.SQL_NEXT_ROW 获取的数据。

PKG_SERVICE.SQL_GET_VALUE 函数的原型为：

```
PKG_SERVICE.SQL_GET_VALUE(
context_id          IN     INTEGER,
pos                 IN     INTEGER,
col_type            IN     ANYELEMENT
)
RETURN ANYELEMENT;
```

表 17-7 PKG_SERVICE.SQL_GET_VALUE 接口说明

参数名称	描述
context_id	执行的 CONTEXT ID
pos	动态定义列在查询中的位置
col_type	任意类型变量，定义列的返回值类型

- **PKG_SERVICE.SQL_SET_RESULT_TYPE**

该函数用来定义从给定 CONTEXT 返回的列，该接口只能应用于 SELECT 定义的 CONTEXT 中。定义的列通过查询列表的相对位置来标识，PKG_SERVICE.SQL_SET_RESULT_TYPE 函数的原型为：

```
PKG_SERVICE.SQL_SET_RESULT_TYPE(
context_id      IN INTEGER,
pos             IN INTEGER,
```

```
coltype_oid     IN ANYELEMENT,
maxsize         IN INTEGER
)
RETURN INTEGER;
```

表 17-8 PKG_SERVICE.SQL_SET_RESULT_TYPE 接口说明

参数名称	描述
context_id	执行的 CONTEXT ID。
pos	动态定义列在查询中的位置。
coltype_oid	任意类型的变量，可根据变量类型得到对应类型 OID。
maxsize	定义的列的长度。

- **PKG_SERVICE.JOB_CANCEL**

  存储过程 CANCEL 删除指定的定时任务。

  PKG_SERVICE.JOB_CANCEL 函数原型为：

```
PKG_SERVICE.JOB_CANCEL(
job   IN   INTEGER);
```

表 17-9 PKG_SERVICE.JOB_CANCEL 接口参数说明

参数	类型	入参/出参	是否可以为空	描述
id	integer	IN	否	指定的作业号。

  示例：

```
CALL PKG_SERVICE.JOB_CANCEL(101);
```

- **PKG_SERVICE.JOB_FINISH**

  存储过程 FINISH 禁用或者启用定时任务。

  PKG_SERVICE.JOB_FINISH 函数原型为：

```
PKG_SERVICE.JOB_FINISH(
```

```
id          IN   INTEGER,
broken       IN   BOOLEAN,
next_time    IN   TIMESTAMP  DEFAULT  sysdate);
```

表 17-10 PKG_SERVICE.JOB_FINISH 接口参数说明

参数	类型	入参/出参	是否可以为空	描述
id	integer	IN	否	指定的作业号。
broken	Boolean	IN	否	状态标志位，true 代表禁用，false 代表启用。根据 true 或 false 值更新当前 job；如果为空值，则不改变原有 job 的状态。
next_time	timestamp	IN	是	下次运行时间，默认为当前系统时间。如果参数 broken 状态为 true，则更新该参数为'4000-1-1'；如果参数 broken 状态为 false，且如果参数 next_time 不为空值，则更新指定 job 的 next_time 值，如果 next_time 为空值，则不更新 next_time 值。该参数可以省略，为默认值。

- **PKG_SERVICE.JOB_SUBMIT**

  存储过程 JOB_SUBMIT 提交一个系统提供的定时任务。

  PKG_SERVICE.JOB_SUBMIT 函数原型为：

```
PKG_SERVICE.JOB_SUBMIT(
id             IN   BIGINT DEFAULT,
content        IN   TEXT,
next_date      IN   TIMESTAMP DEFAULT sysdate,
interval_time IN   TEXT  DEFAULT 'null',
job            OUT  INTEGER);
```

说明： 当创建一个定时任务（JOB）时，系统默认将当前数据库和用户名与当前创建的定时任务绑定起来。该接口函数可以通过 call 或 select 调用，如果通过 select 调用，可以不填写出参。如果在存储过程中，则需要通过 perform 调用该接口函数。如果提交的 sql 语句任务使用到非 public 的 schema，应该指定表或者函数的 schema，或者在 sql 语句前添加 set current_schema = xxx;语句。

表 17-11 PKG_SERVICE.JOB_SUBMIT 接口参数说明

参数	类型	入参/出参	是否可以为空	描述
id	bigint	IN	否	作业号。如果传入 id 为 NULL，则内部会生成作业 ID。
context	text	IN	否	要执行的 SQL 语句。支持一个或多个‘DML’，‘匿名块’，‘调用存储过程的语句’或 3 种混合的场景。
next_time	timestamp	IN	否	下次作业运行时间。默认值为当前系统时间（sysdate）。如果是过去时间，在提交作业时表示立即执行。
interval_time	text	IN	是	用来计算下次作业运行时间的时间表达式，可以是 interval 表达式，也可以是 sysdate 加上一个 numeric 值（例如：sysdate+1.0/24）。如果为空值或字符串”null”表示只执行一次，执行后 JOB 状态 STATUS 变成'd' 不再执行。
job	integer	OUT	否	作业号。范围为 1￣32767。当使用 select 调用 pkg_service.job_submit 时，该参数可以省略。

示例：

```
SELECT PKG_SERVICE.JOB_SUBMIT(NULL, 'call pro_xxx();',
to_date('20180101','yyyymmdd'),'sysdate+1');

SELECT PKG_SERVICE.JOB_SUBMIT(NULL, 'call pro_xxx();',
to_date('20180101','yyyymmdd'),'sysdate+1.0/24');

CALL PKG_SERVICE.JOB_SUBMIT(NULL,'INSERT INTO T_JOB  VALUES(1);  call pro_1();
call pro_2();', add_months(to_date('201701','yyyymm'),1),
'date_trunc(''day'',SYSDATE) + 1 +(8*60+30.0)/(24*60)' ,:jobid);

SELECT PKG_SERVICE.JOB_SUBMIT (101, 'insert_msg_statistic1;', sysdate,
'sysdate+3.0/24');
```

- **PKG_SERVICE.JOB_UPDATE**

存储过程 UPDATE 修改定时任务的属性，包括任务内容、下次执行时间、执行间隔。

PKG_SERVICE.JOB_UPDATE 函数原型为：

```
PKG_SERVICE.JOB_UPDATE(
id             IN   BIGINT,
next_time      IN   TIMESTAMP,
interval_time  IN   TEXT,
content        IN   TEXT);
```

表 17-12 PKG_SERVICE.JOB_UPDATE 接口参数说明

参数	类型	入参/出参	是否可以为空	描述
id	integer	IN	否	指定的作业号。
next_time	timestamp	IN	是	下次运行时间。如果该参数为空值，则不更新指定 job 的 next_time 值，否则更新指定 job 的 next_time 值。
interval_time	text	IN	是	用来计算下次作业运行时间的时间表达式。如果该参数为空值，则不更新指定 job 的 interval_time 值；如果该参数不为空值，会校验 interval_time 是否为有效的时间类型或 interval 类型，则更新指定 job 的 interval_time 值。如果为字符串”null”表示只执行一次，执行后 JOB 状态 STATUS 变成'd'不再执行。
content	text	IN	是	执行的存储过程名或者 sql 语句块。如果该参数为空值，则不更新指定 job 的 content 值，否则更新指定 job 的 content 值。

示例：

```
CALL PKG_SERVICE.JOB_UPDATE(101, 'call userproc();', sysdate, 'sysdate +
1.0/1440');
CALL PKG_SERVICE.JOB_UPDATE(101, 'insert into tbl_a values(sysdate);', sysdate,
'sysdate + 1.0/1440');
```

- **PKG_SERVICE.SUBMIT_ON_NODES**

存储过程 SUBMIT_ON_NODES 创建一个结点上的定时任务，仅 sysadmin/monitor admin

有此权限。

PKG_SERVICE.SUBMIT_ON_NODES 函数原型为：

```
PKG_SERVICE.SUBMIT_ON_NODES(
node_name     IN   TEXT,
database      IN   TEXT
what          IN   TEXT,
next_date     IN   TIMESTAMP DEFAULT sysdate,
job_interval IN   TEXT  DEFAULT 'null',
job           OUT  INTEGER);
```

表 17-13 PKG_SERVICE.SUBMIT_ON_NODES 接口参数说明

参数	类型	入参/出参	是否可以为空	描述
node_name	text	IN	否	指定作业的执行节点，当前仅支持值为'ALL_NODE'（在所有节点执行）与'CCN'（注：CCN在集中式/小型化环境下无意义）。
database	text	IN	否	数据库实例作业所使用的 database，节点类型为'ALL_NODE'时仅支持值为'postgres'。
what	text	IN	否	要执行的 SQL 语句。支持一个或多个‘DML’，‘匿名块’，‘调用存储过程的语句’或 3 种混合的场景。
nextdate	timestamp	IN	否	下次作业运行时间。默认值为当前系统时间（sysdate）。如果是过去时间，在提交作业时表示立即执行。
job_interval	text	IN	否	用来计算下次作业运行时间的时间表达式，可以是 interval 表达式，也可以是 sysdate 加上一个 numeric 值（例如：sysdate+1.0/24）。如果为空值或字符串”null”表示只执行一次，执行后 JOB 状态 STATUS 变成'd'不再执行。
job	integer	OUT	否	作业号。范围为 1￣32767。当使用 select 调用 dbms.submit_on_nodes 时，该参数可以省略。

示例：

```
select pkg_service.submit_on_nodes('ALL_NODE', 'postgres', 'select
capture_view_to_json(''dbe_perf.statement'', 0);', sysdate, 'interval ''60
second''');
select pkg_service.submit_on_nodes('CCN', 'postgres', 'select
capture_view_to_json(''dbe_perf.statement'', 0);', sysdate, 'interval ''60
second''');
```

- **PKG_SERVICE.ISUBMIT_ON_NODES**

ISUBMIT_ON_NODES 与 SUBMIT_ON_NODES 语法功能相同，但其第一个参数是入参，即指定的作业号，SUBMIT 最后一个参数是出参，表示系统自动生成的作业号。仅 sysadmin/monitor admin 有此权限。

- **PKG_SERVICE.SQL_GET_ARRAY_RESULT**

该函数用来返回绑定的数组类型的 OUT 参数的值，可以用来获取存储过程中的 OUT 参数。

PKG_SERVICE.SQL_GET_ARRAY_RESULT 函数原型为：

```
PKG_SERVICE.SQL_GET_ARRAY_RESULT(
    context_id in int,
    pos in VARCHAR2,
    column_value inout anyarray,
    result_type in anyelement
);
```

表 17-14 PKG_SERVICE.SQL_GET_ARRAY_RESULT 接口说明

参数名称	描述
context_id	想查找的 CONTEXT ID 号。
pos	绑定的参数名。
column_value	返回值。
result_type	返回值类型。

- **PKG_SERVICE.SQL_GET_VARIABLE_RESULT**

该函数用来返回绑定的非数组类型的 OUT 参数的值，可以用来获取存储过程中的 OUT

参数。

PKG_SERVICE.SQL_GET_VARIABLE_RESULT 函数原型为：

```
PKG_SERVICE.SQL_GET_VARIABLE_RESULT(
    context_id in int,
    pos in VARCHAR2,
    result_type in anyelement
)
RETURNS anyelement;
```

表 17-15 PKG_SERVICE.SQL_GET_VARIABLE_RESULT 接口说明

参数名称	描述
context_id	想查找的 CONTEXT ID 号。
pos	绑定的参数名。
result_type	返回值类型。

16.12.1.2 PKG_UTIL

PKG_UTIL 支持的所有接口请参见表 17-16：

表 17-16 PKG_UTIL

接口名称	描述
PKG_UTIL.LOB_GET_LENGTH	获取 lob 的长度。
PKG_UTIL.LOB_READ	读取 lob 对象的一部分。
PKG_UTIL.LOB_WRITE	将源对象按照指定格式写入到目标对象。
PKG_UTIL.LOB_APPEND	将 lob 源对象指定个数的字符追加到目标 lob 对象。
PKG_UTIL.LOB_COMPARE	根据指定长度比较两个 lob 对象。
PKG_UTIL.LOB_MATCH	返回一个字符串在 LOB 中第 N 次出现的位置。

接口名称	描述
PKG_UTIL.LOB_RESET	将 lob 的指定位置重置为指定字符。
PKG_UTIL.IO_PRINT	将字符串打印输出。
PKG_UTIL.RAW_GET_LENGTH	获取 raw 的长度。
PKG_UTIL.RAW_CAST_FROM_VARCHAR2	将 varchar2 转化为 raw。
PKG_UTIL.RAW_CAST_FROM_BINARY_INTEGER	将 binary integer 转化为 raw。
PKG_UTIL.RAW_CAST_TO_BINARY_INTEGER	将 raw 转化为 binary integer。
PKG_UTIL.SET_RANDOM_SEED	设置随机种子。
PKG_UTIL.RANDOM_GET_VALUE	返回随机值。
PKG_UTIL.FILE_SET_DIRNAME	设置当前操作的目录。
PKG_UTIL.FILE_OPEN	根据指定文件名和设置的目录打开一个文件。
PKG_UTIL.FILE_SET_MAX_LINE_SIZE	设置写入文件一行的最大长度。
PKG_UTIL.FILE_IS_CLOSE	检测一个文件句柄是否关闭。
PKG_UTIL.FILE_READ	从一个打开的文件句柄中读取指定长度的数据。
PKG_UTIL.FILE_READLINE	从一个打开的文件句柄中读取一行数据。
PKG_UTIL.FILE_WRITE	将 BUFFER 中的数据写入到文件中。
PKG_UTIL.FILE_WRITELINE	将 buffer 写入文件，并追加换行符。
PKG_UTIL.FILE_NEWLINE	新起一行。
PKG_UTIL.FILE_READ_RAW	从一个打开的文件句柄中读取指定长度的二进制数据。
PKG_UTIL.FILE_WRITE_RAW	将二进制数据写入到文件中。

接口名称	描述
PKG_UTIL.FILE_FLUSH	将一个文件句柄中的数据写入到物理文件中。
PKG_UTIL.FILE_CLOSE	关闭一个打开的文件句柄。
PKG_UTIL.FILE_REMOVE	删除一个物理文件，操作需要有对应权限。
PKG_UTIL.FILE_RENAME	对于磁盘上的文件进行重命名，类似 Unix 的 mv。
PKG_UTIL.FILE_SIZE	返回文件大小。
PKG_UTIL.FILE_BLOCK_SIZE	返回文件含有的块数量。
PKG_UTIL.FILE_EXISTS	判断文件是否存在。
PKG_UTIL.FILE_GETPOS	返回文件的偏移量，单位字节。
PKG_UTIL.FILE_SEEK	设置文件位置为指定偏移。
PKG_UTIL.FILE_CLOSE_ALL	关闭一个会话中打开的所有文件句柄。
PKG_UTIL.EXCEPTION_REPORT_ERROR	抛出一个异常。
PKG_UTIL.RANDOM_SET_SEED	设置一个随机数种子。
pkg_util.app_read_client_info	读取 client_info 信息。
pkg_util.app_set_client_info	设置 client_info 信息。
pkg_util.lob_converttoblob	clob 类型转换成 blob 类型。
pkg_util.lob_converttoclob	blob 类型转换成 clob 类型。
pkg_util.lob_rawtotext	raw 类型转成 text 类型。
pkg_util.lob_reset	清空一个 lob 类型的数据。
pkg_util.lob_texttoraw	text 类型转成 raw 类型。
pkg_util.lob_write	将数据写入 lob 类型。
pkg_util.match_edit_distance_similarity	计算两个字符串的差距。

接口名称	描述
pkg_util.raw_cast_to_varchar2	raw 类型转成 varchar2 类型。
pkg_util.session_clear_context	清空 session_context 中的属性值。
pkg_util.session_search_context	查找一个属性值。
pkg_util.session_set_context	设置一个属性值。
pkg_util.utility_format_call_stack	查看存储过程的调用堆栈。
pkg_util.utility_format_error_backtrace	查看存储过程的错误堆栈。
pkg_util.utility_format_error_stack	查看存储过程的报错信息。
pkg_util.utility_get_time	查看系统 unix 时间戳。

- **PKG_UTIL.LOB_GET_LENGTH**

  该函数 LOB_GET_LENGTH 获取输入数据的长度。

  PKG_UTIL.LOB_GET_LENGTH 函数原型为：

```
PKG_UTIL.LOB_GET_LENGTH(
lob       IN   anyelement
)
RETURN INTEGER;
```

表 17-17 PKG_UTIL.LOB_GET_LENGTH 接口参数说明

参数	类型	入参/出参	是否可以为空	描述
lob	clob/blob	IN	否	待获取长度的对象。

- **PKG_UTIL.LOB_READ**

  该函数 LOB_READ 读取一个对象，并返回指定部分。

  PKG_UTIL.LOB_READ 函数原型为：

```
PKG_UTIL.LOB_READ(
lob       IN   anyelement,
len       IN   int,
```

```
start      IN    int,
mode       IN    int
)
RETURN ANYELEMENT
```

表 17-18 PKG_UTIL.LOB_READ 接口参数说明

参数	类型	入参/出参	是否可以为空	描述
lob	clob/blob	IN	否	clob 或者 blob 类型数据。
len	int	IN	否	返回结果长度。
start	int	IN	否	相较于第一个字符的偏移量。
mode	int	IN	否	判断读取操作的类型， 0 ： read; 1 ： trim; 2 ： substr。

- **PKG_UTIL.LOB_WRITE**

  该函数 LOB_WRITE 将源对象按照指定的参数写入目标对象，并返回目标对象。

  PKG_UTIL.LOB_WRITE 函数原型为：

```
PKG_UTIL.LOB_WRITE(
dest_lob     INOUT    anyelement,
src_lob      IN       varchar2
len          IN       int,
start        IN       int
)
RETURN ANYELEMENT;
```

表 17-19 PKG_UTIL.LOB_WRITE 接口参数说明

参数	类型	入参/出参	是否可以为空	描述
dest_lob	clob/blob	INOUT	否	写入的目标对象。
src_lob	clob/blob	IN	否	被写入的源对象。

参数	类型	入参/出参	是否可以为空	描述
len	int	IN	否	源对象的写入长度。
start	int	IN	否	目标对象的写入起始位置。

- **PKG_UTIL.LOB_APPEND**

  该函数 LOB_APPEND 将源 blob/clob 对象追加到目标 blob/clob 对象，并返回目标对象。

  PKG_UTIL.LOB_APPEND 函数原型为：

```
PKG_UTIL.LOB_APPEND(
dest_lob     INOUT    blob,
src_lob      IN       blob,
len          IN       int default NULL
)
RETURN BLOB;

PKG_UTIL.LOB_APPEND(
dest_lob     INOUT    clob,
src_lob      IN       clob,
len          IN       int default NULL
)
RETURN CLOB;
```

表 17-20 PKG_UTIL.LOB_APPEND 接口参数说明

参数	类型	入参/出参	是否可以为空	描述
dest_lob	blob/clob	INOUT	否	写入的目标 blob/clob 对象。
src_lob	blob/clob	IN	否	被写入的源 blob/clob 对象。
len	int	IN	是	写入源对象的长度，为 NULL 则默认写入源对象全部。

- **PKG_UTIL.LOB_COMPARE**

  该函数 LOB_COMPARE 按照指定的起始位置、个数比较对象是否相同，lob1 大则返回

1，lob2 大返回-1，相等返回 0。

PKG_UTIL.LOB_COMPARE 函数原型为：

```
PKG_UTIL.LOB_COMPARE(
lob1         IN   anyelement,
lob2         IN   anyelement,
len          IN   int,
start1       IN   int,
start2       IN   int
)
RETURN INTEGER;
```

表 17-21 PKG_UTIL.LOB_COMPARE 接口参数说明

参数	类型	入参/出参	是否可以为空	描述
lob1	clob/blob	IN	否	待比较的字符串。
lob2	clob/blob	IN	否	待比较的字符串。
len	int	IN	否	比较的长度。
start1	int	IN	否	lob1 起始偏移量。
start2	int	IN	否	lob2 起始偏移量。

- **PKG_UTIL.LOB_MATCH**

  该函数 LOB_MATCH 返回 pattern 出现在 lob 对象中第 match_nth 次的位置。

  PKG_UTIL.LOB_MATCH 函数原型为：

```
PKG_UTIL.LOB_MATCH(
lob          IN   anyelement,
pattern      IN   anyelement,
start        IN   int,
match_nth    IN   int
)
RETURN INTEGER;
```

表 17-22 PKG_UTIL.LOB_MATCH 接口参数说明

参数	类型	入参/出参	是否可以为空	描述
lob	clob/blob	IN	否	待比较的字符串。
pattern	clob/blob	IN	否	待匹配的 pattern。
start	int	IN	否	lob 的起始比较位置。
match_nth	int	IN	否	第几次匹配到。

- **PKG_UTIL.LOB_RESET**

  该函数 LOB_RESET 清除一段数据为字符 value。

  PKG_UTIL.LOB_RESET 函数原型为：

```
PKG_UTIL.LOB_RESET(
lob           INOUT   bytea,
len           INOUT   int,
start         IN   int DEFAUTL 1,
value         IN   char default 0
)
RETURN record;
```

表 17-23 PKG_UTIL.LOB_RESET 接口参数说明

参数	类型	入参/出参	是否可以为空	描述
lob	bytea	IN	否	待重置的字符串。
len	int	IN	否	重置的长度。
start	int	IN	否	重置的起始位置。
value	int	IN	是	设置的字符。默认值‘0’。

- **PKG_UTIL.IO_PRINT**

  该函数 IO_PRINT 将一段字符串打印输出。

  PKG_UTIL.IO_PRINT 函数原型为：

```
PKG_UTIL.IO_PRINT(
format       IN   text,
is_one_line  IN   boolean
)
RETURN void;
```

表 17-24 PKG_UTIL.IO_PRINT 接口参数说明

参数	类型	入参/出参	是否可以为空	描述
format	text	IN	否	待打印输出的字符串。
is_one_line	boolean	IN	否	是否输出为一行。

- **PKG_UTIL.RAW_GET_LENGTH**

  该函数 RAW_GET_LENGTH 获取 raw 的长度。

  PKG_UTIL.RAW_GET_LENGTH 函数原型为：

```
PKG_UTIL.RAW_GET_LENGTH(
value       IN   raw
)
RETURN integer;
```

表 17-25 PKG_UTIL.RAW_GET_LENGTH 接口参数说明

参数	类型	入参/出参	是否可以为空	描述
raw	raw	IN	否	待获取长度的对象。

- **PKG_UTIL.RAW_CAST_FROM_VARCHAR2**

该函数 RAW_CAST_FROM_VARCHAR2 将 varchar2 转化为 raw。

PKG_UTIL.RAW_CAST_FROM_VARCHAR2 函数原型为：

```
PKG_UTIL.RAW_CAST_FROM_VARCHAR2(
str        IN    varchar2
)
RETURN raw;
```

表 17-26 PKG_UTIL.RAW_CAST_FROM_VARCHAR2 接口参数说明

参数	类型	入参/出参	是否可以为空	描述
str	varchar2	IN	否	需要转化的源数据。

- **PKG_UTIL.CAST_FROM_BINARY_INTEGER**

该函数 CAST_FROM_BINARY_INTEGER 将 binary integer 数据转化为 raw 。

PKG_UTIL.CAST_FROM_BINARY_INTEGER 函数原型为：

```
PKG_UTIL.CAST_FROM_BINARY_INTEGER(
value        IN    integer,
endianess    IN    integer
)
RETURN raw;
```

表 17-27 PKG_UTIL.CAST_FROM_BINARY_INTEGER 接口参数说明

参数	类型	入参/出参	是否可以为空	描述
value	integer	IN	否	需要转化为 raw 的源数据。
endianess	integer	IN	否	表示字节序的整型值 1 或 2（1 代表 BIG_ENDIAN，2 代表 LITTLE-ENDIAN）。

- **PKG_UTIL.CAST_TO_BINARY_INTEGER**

该函数 CAST_TO_BINARY_INTEGER 将 raw 数据转化为 binary integer 。

PKG_UTIL.CAST_TO_BINARY_INTEGER 函数原型为：

```
PKG_UTIL.CAST_TO_BINARY_INTEGER(
value       IN   raw,
endianess   IN   integer
)
RETURN integer;
```

表 17-28 PKG_UTIL.CAST_TO_BINARY_INTEGER 接口参数说明

参数	类型	入参/出参	是否可以为空	描述
value	raw	IN	否	需要转化为 binary integer 的 raw 类型源数据。
endianess	integer	IN	否	表示字节序的整型值 1 或 2（1 代表 BIG_ENDIAN，2 代表 LITTLE-ENDIAN）。

PKG_UTIL.SET_RANDOM_SEED

该函数 SET_RANDOM_SEED 设置随机数种子。

PKG_UTIL.SET_RANDOM_SEED 函数原型为：

```
PKG_UTIL.RANDOM_SET_SEED(
seed         IN   int
)
RETURN integer;
```

表 17-29 PKG_UTIL.SET_RANDOM_SEED 接口参数说明

参数	类型	入参/出参	是否可以为空	描述
seed	int	IN	否	随机数种子。

- **PKG_UTIL.RANDOM_GET_VALUE**

该函数 RANDOM_GET_VALUE 返回 0~1 区间的一个随机数。

PKG_UTIL.RANDOM_GET_VALUE 函数原型为：

```
PKG_UTIL.RANDOM_GET_VALUE(
)
RETURN numeric;
```

- **PKG_UTIL.FILE_SET_DIRNAME**

设置当前操作的目录，基本上所有涉及单个目录的操作，都需要调用此方法先设置操作的目录。

PKG_UTIL.FILE_SET_DIRNAME 函数原型为：

```
PKG_UTIL.FILE_SET_DIRNAME(
dir  IN  text
)
RETURN bool
```

表 17-30 PKG_UTIL.FILE_SET_DIRNAME 接口参数说明

参数	描述
dirname	文件的目录位置，这个字符串是一个目录对象名。 说明： 文件目录的位置，需要添加到系统表 PG_DIRECTORY 中，如果传入的路径和 PG_DIRECTORY 中的路径不匹配，会报路径不存在的错误，下面的涉及 location 作为参数的函数也是同样的情况。

- **PKG_UTIL.FILE_OPEN**

该函数用来打开一个文件，最多可以同时打开 50 个文件。并且该函数返回 INTEGER 类型的一个句柄。

PKG_UTIL.FILE_OPEN 函数原型为：

```
PKG_UTIL.FILE_OPEN(
file_name     IN  text,
open_mode     IN  integer)
```

表 17-31 PKG_UTIL.FILE_OPEN 接口参数说明

参数	描述

参数	描述
file_name	文件名，包含扩展（文件类型），不包括路径名。如果文件名中包含路径，在 OPEN 中会被忽略，在 Unix 系统中，文件名不能以/.结尾。
open_mode	指定文件的打开模式，包含 r：read text，w：write text 和 a：append text。 说明： 对于写操作，会检测文件类型，如果写入 elf 文件，将会报错并退出。

- **PKG_UTIL.FILE_SET_MAX_LINE_SIZE**

  设置写入文件一行的最大长度。

  PKG_UTIL.FILE_SET_MAX_LINE_SIZE 函数原型为：

```
PKG_UTIL.FILE_SET_MAX_LINE_SIZE(
max_line_size in integer)
RETURN BOOL
```

表 17-32 PKG_UTIL.FILE_SET_MAX_LINE_SIZE 接口参数说明

参数	描述
max_line_size	每行最大字符数，包含换行符（最小值是 1，最大值是 32767）。如果没有指定，会指定一个默认值 1024。

- **PKG_UTIL.FILE_IS_CLOSE**

  检测一个文件句柄是否关闭。

  PKG_UTIL.FILE_IS_CLOSE 函数原型为：

```
PKG_UTIL.FILE_IS_CLOSE(
file in integer
)
RETURN BOOL
```

表 17-33 PKG_UTIL.FILE_IS_CLOSE 接口参数说明

参数	描述
file	一个打开的文件句柄。

- **PKG_UTIL.FILE_READ**

根据指定的长度从一个打开的文件句柄中读取出数据。

PKG_UTIL.FILE_READ 函数原型为：

```
PKG_UTIL.FILE_READ(
file      IN   integer,
buffer    OUT  text,
len       IN   integer)
```

表 17-34 PKG_UTIL.FILE_READ 接口参数说明

参数	描述
file	通过调用 OPEN 打开的文件句柄，文件必须以读的模式打开，否则会抛出 INVALID_OPERATION 的异常。
buffer	用于接收数据的 BUFFER。
len	从文件中读取的字节数。

- **PKG_UTIL.FILE_READLINE**

根据指定的长度从一个打开的文件句柄中读取出一行数据。

PKG_UTIL.FILE_READLINE 函数原型为：

```
PKG_UTIL.FILE_READLINE(
file     IN  integer,
buffer   OUT text,
len      IN  integer default 1024)
```

表 17-35 PKG_UTIL.FILE_READLINE 接口参数说明

参数	描述
file	通过调用 OPEN 打开的文件句柄，文件必须以读的模式打开，否则会抛出 INVALID_OPERATION 的异常。
buffer	用于接收数据的 BUFFER。
len	从文件中读取的字节数，默认是 NULL。如果是默认 NULL，会使用 max_line_size 来指定大小。

- **PKG_UTIL.FILE_WRITE**

  将 BUFFER 中指定的数据写入到文件中。

  PKG_UTIL.FILE_WRITE 函数原型为：

```
PKG_UTIL.FILE_WRITE(
file in integer,
buffer in text
)
RETURN BOOL
```

表 17-36 PKG_UTIL.FILE_WRITE 接口参数说明

参数	描述
file	一个打开的文件句柄。
buffer	要写入文件的文本数据，BUFFER 的最大值是 32767 个字节。如果没有指定值，默认是 1024 个字节，没有刷新到文件之前，一系列的 PUT 操作的 BUFFER 总和不能超过 32767 个字节。 说明： 对于写操作，会检测文件类型，如果写入 elf 文件，将会报错并退出。

- **PKG_UTIL.FILE_NEWLINE**

  向一个打开的文件中写入一个行终结符。行终结符和平台相关。

  PKG_UTIL.FILE_NEWLINE 函数原型为：

```
PKG_UTIL.FILE_NEWLINE(
file in integer
)
RETURN BOOL
```

表 17-37 PKG_UTIL.FILE_NEWLINE 接口参数说明

参数	描述
file	一个打开的文件句柄。

- **PKG_UTIL.FILE_WRITELINE**

  向一个打开的文件中写入一行。

PKG_UTIL.FILE_WRITELINE 函数原型为：

```
PKG_UTIL.FILE_WRITELINE(
file in integer,
buffer in text,
flush in bool default false
)
RETURN BOOL
```

表 17-38 PKG_UTIL.FILE_WRITELINE 接口参数说明

参数	描述
file	一个打开的文件句柄。
buffer	要写入的内容。
flush	是否落盘。

- **PKG_UTIL.FILE_READ_RAW**

从一个打开的文件句柄中读取指定长度的二进制数据，返回读取的二进制数据，返回类型为 raw。

PKG_UTIL.FILE_READ_RAW 函数原型为：

```
PKG_UTIL.FILE_READ_RAW(
file      in integer,
length    in integer default NULL
)
RETURN raw
```

表 17-39 PKG_UTIL.FILE_READ_RAW 接口参数说明

参数	描述
file	一个打开的文件句柄。
length	要读取的长度，默认为 NULL。默认情况下读取文件中所有数据，最大为 1 G。

- **PKG_UTIL.FILE_WRITE_RAW**

向一个打开的文件中写入传入二进制对象 RAW。插入成功返回 true。

PKG_UTIL.FILE_WRITE_RAW 函数原型为：

```
PKG_UTIL.FILE_WRITE_RAW(
file in integer,
r    in raw
)
RETURN BOOL
```

表 17-40 PKG_UTIL.FILE_NEWLINE 接口参数说明

参数	描述
file	一个打开的文件句柄。
r	准备传入文件的数据 说明：对于写操作，会检测文件类型，如果写入 elf 文件，将会报错并退出。

- **PKG_UTIL.FILE_FLUSH**

一个文件句柄中的数据要写入到物理文件中，缓冲区中的数据必须要有一个行终结符。当文件必须在打开时读取，刷新非常有用。例如，调试信息可以刷新到文件中，以便立即读取。

PKG_UTIL.FILE_FLUSH 函数原型为：

```
PKG_UTIL.FILE_FLUSH (
file in integer
)
RETURN VOID
```

表 17-41 PKG_UTIL.FILE_FLUSH 接口参数说明

参数	描述
file	一个打开的文件句柄。

- **PKG_UTIL.FILE_CLOSE**

关闭一个打开的文件句柄。

PKG_UTIL.FILE_CLOSE 函数原型为：

```
PKG_UTIL.FILE_CLOSE (
```

```
file in integer
)
RETURN BOOL
```

表 17-42 PKG_UTIL.FILE_CLOSE 接口参数说明

参数	描述
file	一个打开的文件句柄。

- **PKG_UTIL.FILE_REMOVE**

  删除一个磁盘文件，操作的时候需要有充分的权限。

  PKG_UTIL.FILE_REMOVE 函数原型为：

```
PKG_UTIL.FILE_REMOVE(
file_name in text
)
RETURN VOID
```

表 17-43 PKG_UTIL.FILE_REMOVE 接口参数说明

参数	描述
filen_ame	要删除的文件名

- **PKG_UTIL.FILE_RENAME**

  对于磁盘上的文件进行重命名，类似 Unix 的 mv。

  PKG_UTIL.FILE_RENAME 函数原型为：

```
PKG_UTIL.FILE_RENAME(
text src_dir in text,
text src_file_name in text,
text dest_dir in text,
text dest_file_name in text,
 overwrite boolean default false)
```

表 17-44 PKG_UTIL.FILE_RENAME 接口参数说明

参数	描述

参数	描述
src_dir	源文件目录（大小写敏感）。
src_file_name	源文件名。
dest_dir	目标文件目录（大小写敏感）。
dest_file_name	目标文件名。
overwrite	默认是 false，如果目的目录下存在一个同名的文件，不会进行重写。

- **PKG_UTIL.FILE_SIZE**

  返回指定的文件大小。

  PKG_UTIL.FILE_SIZE 函数原型为：

```
bigint PKG_UTIL.FILE_SIZE(
file_name in text
)
```

表 17-45 PKG_UTIL.FILE_SIZE 接口参数说明

参数	描述
file_name	文件名

- **PKG_UTIL.FILE_BLOCK_SIZE**

  返回指定的文件含有的块数量。

  PKG_UTIL.FILE_BLOCK_SIZE 函数原型为：

```
bigint PKG_UTIL.FILE_BLOCK_SIZE(
file_name in text
)
```

表 17-46 PKG_UTIL.FILE_BLOCK_SIZE 接口参数说明

参数	描述
file_name	文件名

- **PKG_UTIL.FILE_EXISTS**

判断指定的文件是否存在。

PKG_UTIL.FILE_EXISTS 函数原型为：

```
PKG_UTIL.FILE_EXISTS(
file_name in text
)
RETURN BOOL
```

表 17-47 PKG_UTIL.FILE_EXISTS 接口参数说明

参数	描述
file_name	文件名

- **PKG_UTIL.FILE_GETPOS**

  返回文件的偏移量，单位字节。

  PKG_UTIL.FILE_GETPOS 函数原型为：

```
PKG_UTIL.FILE_GETPOS(
file in integer
)
RETURN BIGINT
```

表 17-48 PKG_UTIL.FILE_GETPOS 接口参数说明

参数	描述
file	一个打开的文件句柄。

- **PKG_UTIL.FILE_SEEK**

  根据用户指定的字节数向前或者向后调整文件指针的位置。

  PKG_UTIL.FILE_SEEK 函数原型为：

```
void PKG_UTIL.FILE_SEEK(
file in integer,
start in bigint default null
)
RETURN VOID
```

表 17-49 PKG_UTIL.FILE_SEEK 接口参数说明

参数	描述
file	一个打开的文件句柄。
start	文件偏移，字节。

- **PKG_UTIL.FILE_CLOSE_ALL**

  关闭一个会话中打开的所有的文件句柄。

  PKG_UTIL.FILE_CLOSE_ALL 函数原型为：

```
PKG_UTIL.FILE_CLOSE_ALL(
)
RETURN VOID↵
```

- **PKG_UTIL.EXCEPTION_REPORT_ERROR**

  抛出一个异常。

  PKG_UTIL.EXCEPTION_REPORT_ERROR 函数原型为：

```
PKG_UTIL.EXCEPTION_REPORT_ERROR(
code integer,
log text,
flag boolean DEFAULT false
)
RETURN INTEGER
```

表 17-50 PKG_UTIL.EXCEPTION_REPORT_ERROR 接口参数说明

参数	描述
code	抛异常所打印的错误码。
log	抛异常所打印的日志提示信息。
flag	保留字段，默认为 false。

- **PKG_UTIL.app_read_client_info**

  读取 client_info 信息。

PKG_UTIL.app_read_client_info 函数原型为：

```
PKG_UTIL.app_read_client_info(
OUT buffer text
)
```

表 17-51 PKG_UTIL.app_read_client_info 接口参数说明

参数	描述
buffer	返回的 client_info 信息。

- **PKG_UTIL.app_set_client_info**

  设置 client_info 信息。

  PKG_UTIL.app_set_client_info 函数原型为：

```
PKG_UTIL.app_set_client_info(
str text
)
RETURN INTEGER
```

表 17-52 PKG_UTIL.app_set_client_info 接口参数说明

参数	描述
str	要设置的 client_info 信息。

- **PKG_UTIL.lob_converttoblob**

  将 clob 转成 blob，amout 为要转换的长度。

  PKG_UTIL.lob_converttoblob 函数原型为：

```
PKG_UTIL.lob_converttoblob(
dest_lob blob,
src_clob clob,
amount integer,
dest_offset integer,
src_offset integer
)
```

表 17-53 PKG_UTIL.lob_converttoblob 接口参数说明

参数	描述
dest_lob	目标 lob。
src_clob	要转换的 clob。
amount	转换的长度。
dest_offset	目标 lob 的起始位置。
src_offset	源 clob 的起始位置。

- **PKG_UTIL.lob_converttoclob**

  将 blob 转成 clob，amout 为要转换的长度。

  PKG_UTIL.lob_converttoclob 函数原型为：

```
PKG_UTIL.lob_converttoclob(
dest_lob clob,
src_blob blob,
amount integer,
dest_offset integer,
src_offset integer
)
```

表 17-54 PKG_UTIL.lob_converttoclob 接口参数说明

参数	描述
dest_lob	目标 lob。
src_blob	要转换的 blob。
amount	转换的长度。
dest_offset	目标 lob 的起始位置。
src_offset	源 clob 的起始位置。

- **PKG_UTIL.lob_texttoraw**

  将 text 转成 raw。

  PKG_UTIL.lob_texttoraw 函数原型为：

```
PKG_UTIL.lob_texttoraw(
src_lob clob
)
RETURN raw
```

表 17-55 PKG_UTIL.lob_texttoraw 接口参数说明

参数	描述
src_lob	要转换的 lob 数据。

- **PKG_UTIL.match_edit_distance_similarity**

  计算两个字符串的差别。

  PKG_UTIL.match_edit_distance_similarity 函数原型为：

```
PKG_UTIL.match_edit_distance_similarity(
str1 text,
str2 text
)
RETURN INTEGER
```

表 17-56 PKG_UTIL.match_edit_distance_similarity 接口参数说明

参数	描述
str1	第一个字符串。
str2	第二个字符串。

- **PKG_UTIL.raw_cast_to_varchar2**

  raw 类型转成 varchar2。

  PKG_UTIL.raw_cast_to_varchar2 函数原型为：

```
PKG_UTIL.raw_cast_to_varchar2(
str1 text,
str2 text
)
RETURN INTEGER
```

表 17-57 PKG_UTIL.raw_cast_to_varchar2 接口参数说明

参数	描述
str1	第一个字符串。
str2	第二个字符串。

- **PKG_UTIL.session_clear_context**

  清除 session_context 信息。

  PKG_UTIL.session_clear_context 函数原型为：

```
PKG_UTIL.session_clear_context(
namespace text,
client_identifier text,
attribute text
)
RETURN INTEGER
```

表 17-58 PKG_UTIL.session_clear_context 接口参数说明

参数	描述
namespace	属性的命名空间。
client_identifier	client_identifier，一般与 namespace 即可，当为 null 时，默认修改所有 namesapce。
attribute	要清除的属性值。

- **PKG_UTIL.session_search_context**

  查找属性值。

  PKG_UTIL.session_clear_context 函数原型为：

```
PKG_UTIL.session_clear_context(
namespace text,
attribute text
)
```

```
RETURN INTEGER
```

表 17-59 PKG_UTIL.session_clear_context 接口参数说明

参数	描述
namespace	属性的命名空间。
attribute	要清除的属性值。

- **PKG_UTIL.session_set_context**

  设置属性值。

  PKG_UTIL.session_set_context 函数原型为：

```
PKG_UTIL.session_set_context(
namespace text,
attribute text,
value text
)
RETURN INTEGER
```

表 17-60 PKG_UTIL.session_set_context 接口参数说明

参数	描述
namespace	属性的命名空间
attribute	要设置的属性
value	属性对应的值

- **PKG_UTIL.utility_get_time**

  打印 unix 时间戳。

  PKG_UTIL.utility_get_time 函数原型为：

```
PKG_UTIL.utility_get_time()
RETURN text
```

- **PKG_UTIL.utility_format_error_backtrace**

  查看存储过程调用堆栈。

PKG_UTIL.utility_format_error_backtrace 函数原型为：

```
PKG_UTIL.utility_format_error_backtrace()
RETURN text
```

- **PKG_UTIL.utility_format_error_stack**

  查看存储过程错误信息。

  PKG_UTIL.utility_format_error_stack 函数原型为：

```
PKG_UTIL.utility_format_error_stack()
RETURN text
```

- **PKG_UTIL.utility_format_call_stack**

  查看存储过程调用堆栈。

  PKG_UTIL.utility_format_call_stack 函数原型为：

```
PKG_UTIL.utility_format_call_stack()
RETURN text
```

16.13 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：
postgres=# 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;
/

postgres=# CALL retry_basic(1);
```

16.14 调试

16.14.1语法

16.14.1.1 RAISE 语法

有以下五种语法格式：

图 1 raise_format::=

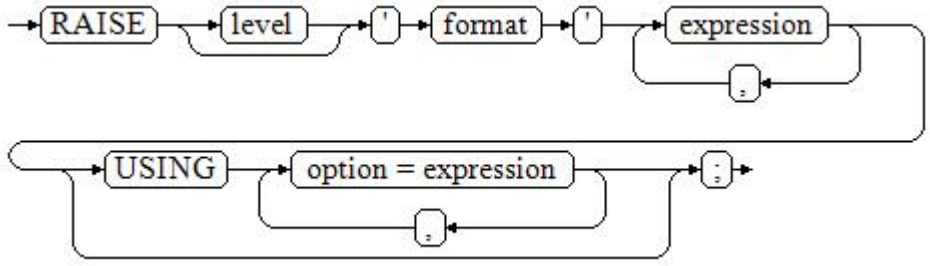


图 17-11 raise_format::=

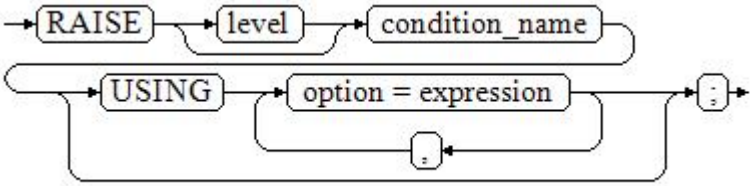


图 17-12 raise_condition::=

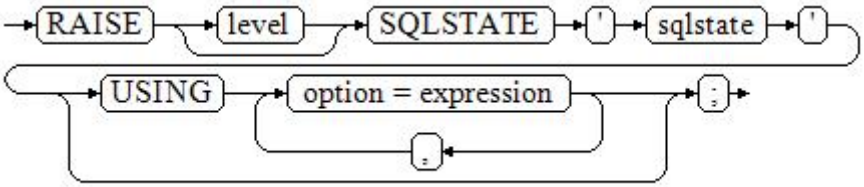


图 17-13 raise_sqlstate::=

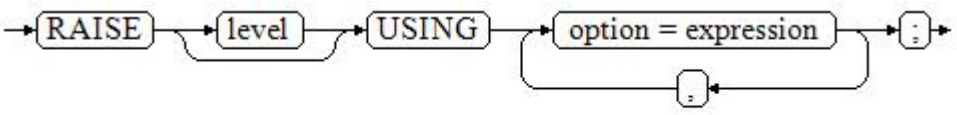


图 17-14 raise_option::=

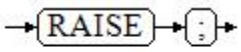


图 17-15 raise::=

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。- 兼容 O 模式下，SQLCODE 等于 SQLSTATE。

说明： 图 17-15 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。

16.14.1.2 EXCEPTION_INIT 语法

兼容 O 模式下，支持使用 EXCEPTION_INIT 语法自定义错误码 SQLCODE。语法格式如下：

图 17-16 exception_init::=

参数说明：

exception_name 为用户申明的异常名，EXCEPTION_INIT 语法必须出现在与申明异常相同部分，位于申明异常之后。

sqlcode 为自定义的 SQLCODE，必须为负整数，取值范围-2147483647~-1。

须知： 使用 EXCEPTION_INIT 语法自定义错误码 SQLCODE 时，SQLSTATE 与 SQLCODE 相同，SQLERRM 格式为” xxx: non-GaussDB Exception”。比如自定义 SQLCODE=-1，则 SQLSTATE=“-1”，SQLERRM=” 1: non-GaussDB Exception”。

16.14.2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
```

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;
```

兼容 O 模式下，支持使用语法 EXCEPTION_INIT 自定义错误码 SQLCODE:

```
declare
    deadlock_detected exception;
    pragma exception_init(deadlock_detected, -1);
begin
    if 1 > 0 then
        raise deadlock_detected;
    end if;
exception
    when deadlock_detected then
        raise notice
'sqlcode:%,sqlstate:%,sqlerrm:%',sqlcode,sqlstate,sqlerrm;
end;
/
--执行结果
NOTICE:  sqlcode:-1,sqlstate:-1,sqlerrm: 1: non-GaussDB Exception
```

16.15 package

package 是一组相关存储过程、函数、变量、常量、游标等 PL/SQL 程序的组合，具有面向对象的特点，可以对 PL/SQL 程序设计元素进行封装。package 中的函数具有统一性，创建、删除、修改都统一进行。

package 包含包头（Package Specification）和 Package Body 两个部分，其中包头所包含

的声明可以被外部函数、匿名块等访问，而在包体中包含的声明不能被外部函数、匿名块等访问，只能被包体内函数和存储过程等访问。

PACKAGE 的创建请参见 CREATE PACKAGE。

须知：- 跨 PACKAGE 变量不支持作为 FOR 循环中控制变量使用。- PACKAGE 中定义类型不支持删除、修改等操作，也不支持定义表。 - 不支持以 SCHEMA.PACKAGE.CUROSR 的形式引用 cursor 变量。 - 带参数的 CURSOR 仅支持在当前 PACKAGE 内打开。

17 自治事务

自治事务（Autonomous Transaction），在主事务执行过程中新启的独立的事务。自治事务的提交和回滚不会影响主事务已提交的数据，同时自治事务也不受主事务影响。

自治事务在存储过程、函数和匿名块中定义，用 PRAGMA AUTONOMOUS_TRANSACTION 关键字来声明。

17.1 存储过程支持自治事务

自治事务可以在存储过程中定义，标识符为 PRAGMA AUTONOMOUS_TRANSACTION，其余语法与创建存储过程语法相同，示例如下。

```
--建表
create table t2(a int, b int);
insert into t2 values(1,2);
select * from t2;

--创建包含自治事务的存储过程
CREATE OR REPLACE PROCEDURE autonomous_4(a int, b int)  AS
DECLARE
   num3 int := a;
   num4 int := b;
   PRAGMA AUTONOMOUS_TRANSACTION;
BEGIN
   insert into t2 values(num3, num4);
```

```
END;
/
--创建调用自治事务存储过程的普通存储过程
CREATE OR REPLACE PROCEDURE autonomous_5(a int, b int)  AS
DECLARE
BEGIN

   insert into t2 values(666, 666);
   autonomous_4(a,b);
   rollback;
END;
/
--调用普通存储过程
select autonomous_5(11,22);
--查看表结果
select * from t2 order by a;
```

上述例子，最后在回滚的事务块中执行包含自治事务的存储过程，直接说明了自治事务的特性，即主事务的回滚，不会影响自治事务已经提交的内容。

17.2 匿名块支持自治事务

自治事务可以在匿名块中定义，标识符为 PRAGMA AUTONOMOUS_TRANSACTION，其余语法与创建匿名块语法相同，示例如下。

```
create table t1(a int ,b text);

START TRANSACTION;
DECLARE
   PRAGMA AUTONOMOUS_TRANSACTION;
BEGIN

   insert into t1 values(1,'you are so cute,will commit!');
END;
/
insert into t1 values(1,'you will rollback!');
rollback;

select * from t1;
```

上述例子，最后在回滚的事务块前执行包含自治事务的匿名块，也能直接说明了自治事

务的特性，即主事务的回滚，不会影响自治事务已经提交的内容。

17.3 用户自定义函数支持自治事务

自治事务可以在函数中定义，标识符为 PRAGMA AUTONOMOUS_TRANSACTION，执行的函数块中使用包含 start transaction 和 commit/rollback 的 sql，其余语法与 CREATE FUNCTION 创建函数语法类似，一个简单的用例如下：

```
--创建表。
CREATE TABLE test1 (a int, b text);
--创建包含自治事务的函数。
CREATE OR REPLACE FUNCTION autonomous_easy_2(i int) RETURNS integer
LANGUAGE plpgsql
AS $$
DECLARE
    PRAGMA AUTONOMOUS_TRANSACTION;
BEGIN
    START TRANSACTION;
    INSERT INTO test1 VALUES (2, 'a');
    IF i % 2 = 0 THEN
        COMMIT;
    ELSE
        ROLLBACK;
    END IF;
  RETURN i % 2 = 0;
END;
$$;
--执行命令。
select autonomous_easy_2(1);
--执行结果。
 autonomous_easy_2
-------------------
                  0
(1 row)
--执行命令，查询表数据。
select * from test1;
--执行结果。
 a | b
```

```
---+---
(0 rows)
--执行命令。
select autonomous_easy_2(2);
--执行结果。
 autonomous_easy_2
-------------------
                 1
(1 row)
--执行命令，查询表数据。
select * from test1;
--执行结果
 a | b
---+---
 2 | a
(1 row)
--清空表数据。
truncate table test1;
--在回滚的事务块中执行包含自治事务的函数。
begin;
insert into test1 values(1,'b');
select autonomous_easy_2(2);
rollback;
--检查表数据。
select * from test1;
--执行结果如下。
 a | b
---+---
 2 | a
(1 row)
```

上述例子，最后在回滚的事务块中执行包含自治事务的函数，直接说明了自治事务的特性，即主事务的回滚，不会影响自治事务已经提交的内容。

17.4 规格约束

注意：

- 自治事务执行时，将会在后台启动自治事务 session，我们可以通过 max_concurrent_autonomous_transactions 设置自治事务执行的最大并行数量，该参数取值范

围为 0~1024，默认值为 10。

- 当 max_concurrent_autonomous_transactions 参数设置为 0 时，自治事务将无法执行。

- 自治事务新启 session 后，将使用默认 session 参数，不共享主 session 下对象（包括 session 级别变量，本地临时变量，全局临时表的数据等）。

- 触发器函数不支持自治事务。

```
CREATE TABLE test_trigger_des_tbl(id1 INT, id2 INT, id3 INT);

CREATE OR REPLACE FUNCTION tri_insert_func() RETURNS TRIGGER AS
$$
DECLARE
 PRAGMA AUTONOMOUS_TRANSACTION;
BEGIN
INSERT INTO test_trigger_des_tbl VALUES(NEW.id1, NEW.id2, NEW.id3);
RETURN NEW;
END
$$ LANGUAGE PLPGSQL;
```

- 自治事务不支持非顶层匿名块调用（仅支持顶层自治事务,包括存储过程、函数、匿名块）。

- 自治事务不支持 ref_cursor 参数传递。

```
create table sections(section_ID int);
insert into sections values(1);
insert into sections values(1);
insert into sections values(1);
insert into sections values(1);

CREATE OR REPLACE function proc_sys_ref()
return SYS_REFCURSOR
IS
declare
  PRAGMA AUTONOMOUS_TRANSACTION;
        C1 SYS_REFCURSOR;
```

```
BEGIN
 OPEN C1 FOR SELECT section_ID FROM sections ORDER BY section_ID;
 return C1;
END;
/

CREATE OR REPLACE PROCEDURE proc_sys_call() AS
DECLARE
  C1 SYS_REFCURSOR;
  TEMP NUMBER(4);
BEGIN
  c1 = proc_sys_ref();
  if c1%isopen then
    raise notice '%','ok';
 end if;

  LOOP
    FETCH C1 INTO TEMP;
    raise notice '%',C1%ROWCOUNT;
    EXIT WHEN C1%NOTFOUND;
  END LOOP;
END;
/

select proc_sys_call();

CREATE OR REPLACE function proc_sys_ref(OUT C2 SYS_REFCURSOR, OUT a int)
return SYS_REFCURSOR
IS
declare
  PRAGMA AUTONOMOUS_TRANSACTION;
        C1 SYS_REFCURSOR;
BEGIN
 OPEN C1 FOR SELECT section_ID FROM sections ORDER BY section_ID;
 return C1;
END;
/

CREATE OR REPLACE PROCEDURE proc_sys_call() AS
DECLARE
  C1 SYS_REFCURSOR;
```

```
  TEMP NUMBER(4);
  a int;
BEGIN
  OPEN C1 FOR SELECT section_ID FROM sections ORDER BY section_ID;
  c1 = proc_sys_ref(C1,a);
  if c1%isopen then
    raise notice '%','ok';
 end if;

  LOOP
    FETCH C1 INTO TEMP;
    raise notice '%',C1%ROWCOUNT;
    EXIT WHEN C1%NOTFOUND;
  END LOOP;
END;
/

select proc_sys_call();
```

- 自治事务函数不支持返回非 out 形式的 record 类型。
- 不支持修改自治事务的隔离级别。
- 不支持自治事务返回集合类型（setof）。

```
create table test_in (id int,a date);
create table test_main (id int,a date);
insert into test_main values (1111,'2021-01-01'),(2222,'2021-02-02');
truncate test_in,test_main;
CREATE OR REPLACE FUNCTION autonomous_f_022(num1  int) RETURNS SETOF test_in
LANGUAGE plpgsql AS $$
DECLARE
count int :=3;
test_row test_in%ROWTYPE;
PRAGMA AUTONOMOUS_TRANSACTION;
BEGIN
    while true
    loop
    if count=3 then
    null;
    else
    if count=2 then
```

```
        insert into test_main values (count,'2021-03-03');
        goto pos1;
    end if;
    end if;
    count=count-1;
    end loop;
    insert into test_main values (1000,'2021-04-04');
     <<pos1>>
     for test_row in select * from test_main
     loop
        return next test_row;
       end loop;
      return;
END;
$$
;
```

18 系统表和系统视图

18.1 系统表和系统视图概述

系统表是 GBase 8c 存放结构元数据的地方，它是 GBase 8c 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。后续的系统表和系统视图章节有些标识了需要管理员权限，这些系统表和系统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 GBase 8c 不可用。

18.2 系统表

18.2.1 GS_ASP

GS_ASP 显示被持久化的 ACTIVE SESSION PROFILE 样本，该表只在系统库下查询，在用户库下查询无数据。

名称	类型	描述
sampleid	bigint	采样 ID。
sample_time	timestamp with time zone	采样的时间。
need_flush_sample	boolean	该样本是否需要刷新到磁盘。 t（true）：表示需要。 f（false）：表示不需要。
databaseid	oid	数据库 ID。
thread_id	bigint	线程的 ID。
sessionid	bigint	会话的 ID。
start_time	timestamp with time zone	会话的启动时间。
event	text	具体的事件名称。
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。与执行计划的层级（id）相对应。
smpid	integer	smp 执行模式下并行线程的并行编号。
userid	oid	session 用户的 id。
application_name	text	应用的名字。
client_addr	inet	client 端的地址。

名称	类型	描述
client_hostname	text	client 端的名字。
client_port	integer	客户端用于与后端通讯的 TCP 端口号。
query_id	bigint	debug query id。
unique_query_id	bigint	unique query id。
user_id	oid	unique query 的 key 中的 user_id。
cn_id	integer	表示下发该 unique sql 的节点 id。unique query 的 key 中的 cn_id。
unique_query	text	-规范化后的 Unique SQL 文本串。
locktag	text	会话等待锁信息，可通过 locktag_decode 解析。
lockmode	text	会话等待锁模式： LW_EXCLUSIVE：排他锁 LW_SHARED：共享锁 LW_WAIT_UNTIL_FREE：等待 LW_EXCLUSIVE 可用
block_sessionid	bigint	如果会话正在等待锁，阻塞该会话获取锁的会话标识。
wait_status	text	描述 event 列的更多详细信息。
global_sessionid	text	全局会话 ID。

18.2.2 GS_AUDITING_POLICY

GS_AUDITING_POLICY 系统表记录统一审计的主体信息，每条记录对应一个设计策略。需要有系统管理员或安全策略管理员权限才可以访问此系统表。

名称	类型	描述
oid	oid	行标识符（隐含属性，必须明确选择）。
polname	name	策略名称，需要唯一，不可重复。

名称	类型	描述
polcomments	name	策略描述字段，记录策略相关的描述信息，通过 COMMENTS 关键字体现。
modifydate	timestamp without time zone	策略创建或修改的最新时间戳。
polenabled	boolean	用来表示策略启动开关。

18.2.3 GS_AUDITING_POLICY_ACCESS

GS_AUDITING_POLICY_ACCESS 系统表记录与 DML 数据库相关操作的统一审计信息。需要有系统管理员或安全策略管理员权限才可以访问此系统表。

名称	类型	描述
oid	oid	行标识符（隐含属性，必须明确选择）。
accesstype	name	DML 数据库操作相关类型。例如 SELECT、INSERT、DELETE 等。
labelname	name	资源标签名称。对应系统表 gs_auditing_policy 中的 polname 字段。
policyoid	oid	所属审计策略的 oid，对应系统表 GS_AUDITING_POLICY 中的 oid。
modifydate	timestamp without time zone	创建或修改的最新时间戳。

18.2.4 GS_AUDITING_POLICY_FILTERS

GS_AUDITING_POLICY_FILTERS 系统表记录统一审计相关的过滤策略相关信息，每条记录对应一个设计策略。需要有系统管理员或安全策略管理员权限才可以访问此系统表。

名称	类型	描述
oid	oid	行标识符（隐含属性，必须明确选择）。
filtertype	name	过滤类型。目前值仅为 logical_expr。
labelname	name	名称。目前值仅为 logical_expr。

名称	类型	描述
policyoid	oid	所属审计策略的 oid，对应系统表 GS_AUDITING_POLICY 中的 oid。
modifydate	timestamp	创建或修改的最新时间戳。
logicaloperator	text	过滤条件的逻辑字符串。

18.2.5 GS_AUDITING_POLICY_PRIVILEGES

GS_AUDITING_POLICY_PRIVILEGES 系统表记录统一审计 DDL 数据库相关操作信息，每条记录对应一个设计策略。需要有系统管理员或安全策略管理员权限才可以访问此系统表。

名称	类型	描述
oid	oid	行标识符（隐含属性，必须明确选择）。
privilegetype	name	DDL 数据库操作相关类型。例如 CREATE、ALTER、DROP 等。
labelname	name	资源标签名称。对应系统表 gs_auditing_policy 中的 polname 字段。
policyoid	oid	对应审计策略系统表 GS_AUDITING_POLICY 中的 oid。
modifydate	timestamp	创建或修改的最新时间戳。

18.2.6 GS_CLIENT_GLOBAL_KEYS

GS_CLIENT_GLOBAL_KEYS 系统表记录密态等值特性中客户端加密主密钥相关信息，每条记录对应一个客户端加密主密钥。

名称	类型	描述
oid	oid	行标识符（隐含字段）。
global_key_name	name	客户端加密主密钥（cmk）名称。
key_namespace	oid	包含此客户端加密主密钥（cmk）的命名空间 oid。

名称	类型	描述
key_owner	oid	客户端加密主密钥（cmk）的所有者。
key_acl	aclitem[]	创建该密钥时所拥有的访问权限。
create_date	timestamp	创建密钥的时间。

18.2.7 GS_CLIENT_GLOBAL_KEYS_ARGS

GS_CLIENT_GLOBAL_KEYS_ARGS 系统表记录密态等值特性中客户端加密主密钥相关元数据信息，每条记录对应客户端加密主密钥的一个键值对信息。

名称	类型	描述
oid	oid	行标识符（隐含字段）。
global_key_id	oid	客户端加密主密钥（cmk）oid。
function_name	name	值为 encryption。
key	name	客户端加密主密钥（cmk）的元数据信息对应的名称。
value	bytea	客户端加密主密钥（cmk）的元数据信息名称的值。

18.2.8 GS_COLUMN_KEYS

GS_COLUMN_KEYS 系统表记录密态等值特性中列加密密钥相关信息，每条记录对应一个列加密密钥。

名称	类型	描述
oid	oid	行标识符（隐含字段）。
column_key_name	name	列加密密钥（cek）名称。
column_key_distributed_id	oid	根据加密密钥（cek）全称域名 hash 值得到的 id。
global_key_id	oid	外键。客户端加密主密钥（cmk）的 oid。

名称	类型	描述
key_namespace	oid	包含此列加密密钥（cek）的命名空间 oid。
key_owner	oid	列加密密钥（cek）的所有者。
create_date	timestamp	创建列加密密钥的时间。
key_acl	aclitem[]	创建该列加密密钥时所拥有的访问权限。

18.2.9 GS_COLUMN_KEYS_ARGS

GS_COLUMN_KEYS_ARGS 系统表记录密态等值特性中客户端加密主密钥相关元数据信息，每条记录对应客户端加密主密钥的一个键值对信息。

名称	类型	描述
oid	oid	行标识符（隐含字段）。
column_key_id	oid	列加密密钥（cek）oid。
function_name	name	值为 encryption。
key	name	列加密密钥（cek）的元数据信息对应的名称。
value	bytea	列加密密钥（cek）的元数据信息名称的值。

18.2.10GS_DB_PRIVILEGE

GS_DB_PRIVILEGE 系统表记录 ANY 权限的授予情况，每条记录对应一条授权信息。

名称	类型	描述
oid	oid	行标识符（隐含字段，必须明确选择）。
roleid	oid	用户标识。
privilege_type	text	用户拥有的 ANY 权限，取值参考《GBase 8c V5_3.0.1_SQL 参考手册》中 GRANT 章节 ANY 权限列表。

名称	类型	描述
admin_option	boolean	是否具有 privilege_type 列记录的 ANY 权限的再授权权限。 t：表示具有。 f：表示不具有。

18.2.11 GS_ENCRYPTED_COLUMNS

GS_ENCRYPTED_COLUMNS 系统表记录密态等值特性中表的加密列相关信息，每条记录对应一条加密列信息。

名称	类型	描述
oid	oid	行标识符（隐含字段）。
rel_id	oid	表的 OID。
column_name	name	加密列的名称。
column_key_id	oid	外键，列加密密钥的 OID。
encryption_type	int1	加密类型，取值为 2（DETERMINISTIC）或者 1（RANDOMIZED）。
data_type_original_oid	oid	加密列的原始数据类型 id，参考系统表 PG_TYPE 中的 oid。
data_type_original_mod	int4	加密列的原始数据类型修饰符，参考系统表 PG_ATTRIBUTE 中的 atttypmod。其值对那些不需要的类型 data_type_original_mod 通常为-1。
create_date	timestamp	创建加密列的时间。

18.2.12 GS_ENCRYPTED_PROC

GS_ENCRYPTED_PROC 系统表提供了密态函数/存储过程函数参数、返回值的原始数据类型，加密列等信息。

名称	类型	描述
oid	oid	行标识符（隐含字段）。

名称	类型	描述
func_id	oid	function 的 oid，对应 pg_proc 系统表中的 oid 行标识符。
prorettype_orig	int4	返回值的原始数据类型。
proargcachedcol	oidvector	函数 INPUT 参数对应的加密列的 oid，对应 gs_encrypted_columns 系统表中的 oid 行标识符。
proallargtypes_orig	oid[]	所有函数参数的原始数据类型。

18.2.13GS_GLOBAL_CHAIN

GS_GLOBAL_CHAIN 系统表记录用户对防篡改用户表的修改操作信息，每条记录对应一次表级修改操作。具有审计管理员权限的用户可以查询此系统表，所有用户均不允许修改此系统表。

名称	类型	描述
blocknum	bigint	区块号，当前用户操作在账本中记录的序号。
dbname	name	数据库名称。被修改的防篡改用户表所属的database。
username	name	用户名，执行用户表修改操作的用户名。
starttime	timestamp with time zone	用户操作执行的最新时间戳。
relid	oid	用户表 Oid，被修改的防篡改用户表 Oid。
relnsp	name	模式 Oid，被修改的防篡改用户表所属的 namesapce oid。
relname	name	用户表名，被修改的防篡改用户表名。
relhash	hash16	当前操作产生的表级别 hash 变化量。
globalhash	hash32	全局摘要，由当前行信息与前一行 globalhash 计算而来，将整个表串联起来，用于验证 GS_GLOBAL_CHAIN 数据完整性。
txcommand	text	被记录操作的 SQL 语句。

18.2.14GS_GLOBAL_CONFIG

GS_GLOBAL_CONFIG 记录了数据库实例初始化时，用户指定的参数值。除此之外，还存放了用户设置的弱口令，支持 ALTER 和 DROP 语法对系统表中的参数进行写入、修改和删除。

名称	类型	描述
name	name	数据库实例初始化时系统内置的指定参数名称。当前版本第一行默认为 buckets_len，第二行起存放弱口令名称。
value	text	数据库实例初始化时系统内置的指定参数值。当前版本第一行默认为 bucketmap 长度；第二行起存放弱口令。

18.2.15GS_MASKING_POLICY

GS_MASKING_POLICY 系统表记录动态数据脱敏策略的主体信息，每条记录对应一个脱敏策略。需要有系统管理员或安全策略管理员权限才可以访问此系统表。

名称	类型	描述
oid	oid	行标识符（隐含属性，必须明确选择）。
polname	name	策略名称，唯一不可重复。
polcomments	name	策略描述字段，记录策略相关的描述信息，通过 COMMENTS 关键字体现。
modifydate	timestamp	策略创建或修改的最新时间戳。
polenabled	Boolean	策略启动开关。

18.2.16GS_MASKING_POLICY_ACTIONS

GS_MASKING_POLICY_ACTIONS 系统表记录动态数据脱敏策略中相应的脱敏策略，包含的脱敏行为，一个脱敏策略对应着该表的一行或多行记录。需要有系统管理员或安全策略管理员权限才可以访问此系统表。

名称	类型	描述
oid	oid	行标识符（隐含属性，必须明确选择）。

名称	类型	描述
actiontype	name	脱敏函数，标识脱敏策略使用的脱敏函数。
actparams	name	向脱敏函数中传递的参数信息。
actlabelname	name	被脱敏的 label 名称。
policyoid	oid	该条记录所属的脱敏策略 oid，对应 GS_MASKING_POLICY 中的 oid。
modifydate	timestamp	该条记录创建或修改的最新时间戳。

18.2.17GS_MASKING_POLICY_FILTERS

GS_MASKING_POLICY_FILTERS 系统表记录动态数据脱敏策略对应的用户过滤条件，当用户条件满足 FILTER 条件时，对应的脱敏策略才会生效。需要有系统管理员或安全策略管理员权限才可以访问此系统表。

名称	类型	描述
oid	oid	行标识符（隐含属性，必须明确选择）。
filtertype	name	过滤类型。目前值仅为 logical_expr。
filterlabelname	name	过滤范围。目前值仅为 logical_expr。
policyoid	oid	该条用户过滤条件所属的脱敏策略 oid，对应 GS_MASKING_POLICY 中的 oid。
modifydate	timestamp	该条用户过滤条件创建或修改的最新时间戳。
logicaloperator	text	过滤条件的波兰表达式。

18.2.18GS_MATVIEW

GS_MATVIEW 系统表提供了关于数据库中每一个物化视图的信息。

名称	类型	描述
oid	oid	行标识符（隐含属性，必须明确选择）。

名称	类型	描述
matviewid	oid	物化视图的 oid。
mapid	oid	物化视图 map 表的 oid，map 表为物化视图关联表，与物化视图一一对应。全量物化视图不存在对应的 map 表，该字段为 0。
ivm	boolean	物化视图的类型，t 为增量物化视图，f 为全量物化视图。
needrefresh	boolean	保留字段。
refreshtime	timestamp	物化视图上一次刷新时间，若未刷新则为 null。仅对增量物化视图维护该字段，全量物化视图为 null。

18.2.19GS_MATVIEW_DEPENDENCY

GS_MATVIEW_DEPENDENCY 系统表提供了关于数据库中每一个增量物化视图、基表和 mlog 表的关联信息。全量物化视图不存在与基表对应的 mlog 表，不会写入记录。

名称	类型	描述
oid	oid	行标识符（隐含属性，必须明确选择）。
matviewid	oid	物化视图的 oid。
relid	oid	物化视图基表的 oid。
mlogid	oid	物化视图 mlog 表的 oid，mlog 表为物化视图日志表，与基表一一对应。
mxmin	int4	保留字段。

18.2.20GS_MODEL_WAREHOUSE

GS_MODEL_WAREHOUSE 系统表用于存储 AI 引擎训练模型，其中包含模型，训练过程的详细描述。

名称	数据类型	描述
oid	oid	隐含列。

名称	数据类型	描述
modelname	name	唯一约束。
modelowner	oid	模型拥有者的 OID。
createtime	timestamp	模型创建的时间。
processedtuples	int	训练涉及的元组数。
discardedtuples	int	未参加训练的不合格元组数。
preprocesstime	real	数据预处理时长。
exectime	real	训练时长。
iterations	int	迭代轮次。
outputtype	oid	模型输出的数据类型 OID。
modeltype	text	AI 算子的类型名称。
query	text	创建模型所执行的 query 语句。
modeldata	bytea	保存的二进制模型信息。
weight	real[]	目前只适用于 GD 算子模型。
hyperparametersnames	text[]	涉及的超参名称。
hyperparametersvalues	text[]	超参所对应的取值。
hyperparametersoids	oid[]	超参对应的数据类型 OID。
coefnames	text[]	模型参数名称。
coefvalues	text[]	模型参数对应的取值。
coefoids	oid[]	模型参数对应的数据类型 OID。
trainingscoresname	text[]	度量模型性能方法的名称。

名称	数据类型	描述
trainingscoresvalue	real[]	度量模型性能方法的数值。
modeldescribe	text[]	模型的描述信息。

18.2.21GS_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。

名称	类型	描述
oid	oid	数据库对象 id。
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。
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 端所侦听的端口号。
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。

名称	类型	描述
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}。
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]标识模型各任务的准确率。
description	text	模型注释。

18.2.22GS_PACKAGE

GS_PACKAGE 系统表记录 PACKAGE 内的信息。

名称	类型	描述

名称	类型	描述
oid	oid	行标识符（隐含属性，必须明确选择）。
pkgnamespace	oid	package 所属 schema。
pkgowner	oid	package 的所属者。
pkgname	name	package 的名字。
pkgspecsrc	text	package specification 的内容。
pkgbodydeclsrc	text	package body 的内容。
pkgbodyinitsrc	text	package init 的内容。
pkgacl	aclitem	访问权限。
pkgsecdef	boolean	package 是否是定义者权限。

18.2.23GS_POLICY_LABEL

GS_POLICY_LABEL 系统表记录资源标签配置信息，一个资源标签对应着一条或多条记录，每条记录标记了数据库资源所属的资源标签。需要有系统管理员或安全策略管理员权限才可以访问此系统表。

FQDN（Fully Qualified Domain Name）标识了数据库资源所属的绝对路径。

名称	类型	描述
oid	oid	行标识符（隐含属性，必须明确选择）。
labelname	name	资源标签名称。
labeltype	name	资源标签类型，目前仅为 RESOURCE。
fqdnnamespace	oid	被标识的数据库资源所属的 namespace oid。
fqdnid	oid	被标识的数据库资源的 oid，若数据库资源为列，则该列为所属表的 oid。

名称	类型	描述
relcolumn	name	列名，若被标识的数据库资源为列，该列指出列名，否则该列为空。
fqdntype	name	被标识的数据库资源的类型名称，例如：schema、table、column、view 等。

18.2.24GS_RECYCLEBIN

gs_recyclebin 描述了回收站对象的详细信息。

名称	类型	描述
oid	oid	系统列。
rcybaseid	oid	基表对象 id，引用 gs_recyclebin.oid。
rcydbid	oid	当前对象所属数据库 oid。
rcyrelid	oid	当前对象 oid。
rcyname	name	回收站对象名称，格式“BIN$unique_id$oid$0”，其中 unique_id 为最多 16 字符唯一标识，oid 为对象标识符。
rcyoriginname	name	原始对象名称。
rcyoperation	“char”	操作类型。 d 表示 drop。 t 表示 truncate。
rcytype	int	对象类型。 0 表示 table。 1 表示 index。 2 表示 toast table。 3 表示 toast index。 4 表示 sequence，指 serial、bigserial、smallserial、largeserial 类型自动关联的序列对象。 5 表示 partition。 6 表示 global index。
rcyrecyclecsn	bigint	对象 drop、truncate 时 csn。
rcyrecycletime	timestamptz	对象 drop、truncate 时间。

名称	类型	描述
rcycreatecsn	bigint	对象创建时 csn。
rcychangecsn	bigint	对象定义改变的 csn。
rcynamespace	oid	包含这个关系的名字空间的 OID。
rcyowner	oid	关系所有者。
rcytablespace	oid	这个关系存储所在的表空间。如果为 0，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。
rcyrelfilenode	oid	回收站对象在磁盘上的文件的名称，如果没有则为 0，用于 TRUNCATE 对象恢复时纹理文件还原。
rcycanrestore	bool	是否可以被单独闪回。
rcycanpurge	bool	是否可以被单独 purge。
rcyfrozenxid	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。
rcyfrozenxid64	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。

18.2.25GS_TXN_SNAPSHOT

GS_TXN_SNAPSHOT 是“时间戳-CSN”映射表，周期性采样，并维护适当的时间范围，用于估算范围内的时间戳对应的 CSN 值。

名称	类型	描述
snptime	timestamptz	快照捕获时间。
snpxmin	bigint	快照 xmin。
snpcsn	bigint	快照 csn。
snpsnapshot	TEXT	快照序列化文本。

18.2.26GS_UID

GS_UID 系统表存储了数据库中使用 hasuids 属性表的唯一标识元信息。

名称	类型	描述
relid	oid	表的 oid 信息。
uid_backup	bigint	当前可以为表分配唯一标识的最大值。

18.2.27GS_WLM_EC_OPERATOR_INFO

GS_WLM_EC_OPERATOR_INFO 系统表存储执行 EC（Extension Connector）作业结束后的算子相关的记录。当设置 GUC 参数 enable_resource_record 为 on 时，系统会每 3 分钟将 GS_WLM_EC_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。查询该系统表需要 sysadmin 权限。

名称	类型	描述
queryid	bigint	EC 语句执行使用的内部 query_id。
plan_node_id	integer	EC 算子对应的执行计划的 plan node id。
start_time	timestamp with time zone	EC 算子处理第一条数据的开始时间。
duration	bigint	EC 算子到结束时候总的执行时间（ms）。
tuple_processed	bigint	EC 算子返回的元素个数。
min_peak_memory	integer	EC 算子在所有 DN 上的最小内存峰值（MB）。
max_peak_memory	integer	EC 算子在所有 DN 上的最大内存峰值（MB）。
average_peak_memory	integer	EC 算子在所有 DN 上的平均内存峰值（MB）。
ec_status	text	EC 作业的执行状态。
ec_execute_datanode	text	执行 EC 作业的 DN 名称。

名称	类型	描述
ec_dsn	text	EC 作业所使用的 DSN。
ec_username	text	EC 作业访问远端数据库实例的 USERNAME（远端数据库实例为 SPARK 类型时该值为空）。
ec_query	text	EC 作业发送给远端数据库实例执行的语句。
ec_libodbc_type	text	EC 作业使用的 unixODBC 驱动类型。

18.2.28GS_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。查询该系统表需要 sysadmin 权限，且仅在数据库 postgres 下面查询时有数据。

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_writ	bigint	实例对应进程向磁盘写数据的写速率（不包括向磁盘

名称	类型	描述
e		pagecache 中写入的字节数，10 秒均值），单位 KB/s。
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。

18.2.29GS_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。查询该系统表需要 sysadmin 权限，且仅在数据库 postgres 下面查询时有数据。

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。

名称	类型	描述
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。

名称	类型	描述
		Spill times is greater than 3。 Spill on memory adaptive。 Hash table conflict。

18.2.30GS_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。

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。
parent_node_id	integer	当前算子的父节点 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。

18.2.31GS_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。

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。

名称	类型	描述
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。
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。

18.2.32GS_WLM_SESSION_QUERY_INFO_ALL

GS_WLM_SESSION_QUERY_INFO_ALL 系统表显示当前数据库实例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。当设置 GUC 参数 enable_resource_record 为 on 时，系统会定时（周期为 3 分钟）将内核中 query 信息导入 GS_WLM_SESSION_QUERY_INFO_ALL 系统表。查询该系统表需要 sysadmin 权限，且仅在数据库 postgres 下面查询时有数据。

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。

名称	类型	描述
schemaname	text	模式的名称。
nodename	text	语句执行的数据库实例名称。
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。
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。

名称	类型	描述
resource_pool	text	用户使用的资源池。
control_group	text	语句所使用的 Cgroup。
estimate_memory	integer	语句估算内存大小。
min_peak_memory	integer	语句在数据库实例上的最小内存峰值，单位 MB。
max_peak_memory	integer	语句在数据库实例上的最大内存峰值，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	语句数据库实例间的内存使用倾斜率。
spill_info	text	语句在数据库实例上的下盘信息： None：数据库实例均未下盘。 All：数据库实例均下盘。 [a:b]：数量为 b 个数据库实例中有 a 个数据库实例下盘。
min_spill_size	integer	若发生下盘，数据库实例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，数据库实例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，数据库实例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，数据库实例间下盘倾斜率。
min_dn_time	bigint	语句在数据库实例上的最小执行时间，单位 ms。
max_dn_time	bigint	语句在数据库实例上的最大执行时间，单位 ms。
average_dn_time	bigint	语句在数据库实例上的平均执行时间，单位 ms。

名称	类型	描述
dntime_skew_percent	integer	语句在数据库实例间的执行时间倾斜率。
min_cpu_time	bigint	语句在数据库实例上的最小 CPU 时间，单位 ms。
max_cpu_time	bigint	语句在数据库实例上的最大 CPU 时间，单位 ms。
total_cpu_time	bigint	语句在数据库实例上的 CPU 总时间，单位 ms。
cpu_skew_percent	integer	语句在数据库实例间的 CPU 时间倾斜率。
min_peak_iops	integer	语句在数据库实例上的每秒最小 IO 峰值（列存单位是次/s，行存单位是万次/s）。
max_peak_iops	integer	语句在数据库实例上的每秒最大 IO 峰值（列存单位是次/s，行存单位是万次/s）。
average_peak_iops	integer	语句在数据库实例上的每秒平均 IO 峰值（列存单位是次/s，行存单位是万次/s）。
iops_skew_percent	integer	语句在数据库实例间的 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数据库实例。
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 5 的节点名称。
mem_top3_node_name	text	内存使用量第 5 的节点名称。
mem_top4_node_name	text	内存使用量第 5 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。

名称	类型	描述
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。
n_returned_rows	bigint	Select 返回的结果集行数。
n_tuples_fetched	bigint	随机扫描行数。
n_tuples_returned	bigint	顺序扫描行数。
n_tuples_inserted	bigint	插入行数。
n_tuples_updated	bigint	更新行数。
n_tuples_deleted	bigint	删除行数。
n_blocks_fetched	bigint	Cache 加载次数。
n_blocks_hit	bigint	Cache 命中数。
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）。

名称	类型	描述
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）。
is_slow_query	bigint	标记是否为慢查询。 取值为 1 时表示其为慢查询。

18.2.33GS_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。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。查询该系统表需要 sysadmin 权限，且仅在数据库 postgres 下面查询时有数据。

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。

名称	类型	描述
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。
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。

18.2.34PG_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。

名称	类型	引用	描述
aggfnoid	regproc	PG_PROC.proname	此聚集函数的 PG_PROC pronam e。
aggtransfn	regproc	PG_PROC.proname	转换函数。
aggcollectfn	regproc	PG_PROC.proname	收集函数。
aggfinalfn	regproc	PG_PROC.proname	最终处理函数（如果没有则为零）。
aggsortop	oid	PG_OPERATOR.oid	关联排序操作符（如果没有则为零）。
aggtranstype	oid	PG_TYPE.oid	此聚集函数的内部转换（状态）数据的数据类型。 可能取值及其含义见于 pg_type.h 中诸 type 定义，主要分为多态（isPolymorphicType）和非多态两类。
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。

18.2.35PG_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 1 PG_AM 字段

名称	类型	引用	描述
oid	oid	-	行标识符（隐含属性，必须明确选择）。
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。
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。

名称	类型	引用	描述
amclusterable	boolean	-	是否允许在一个这种类型的索引上聚簇。
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	PG_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	PG_PROC.proname	“插入这个行”函数。
ambeginscan	regproc	PG_PROC.proname	“准备索引扫描”函数。
amgettuple	regproc	PG_PROC.proname	“下一个有效行”函数，如果没有则为 0。
amgetbitmap	regproc	PG_PROC.proname	“抓取所有的有效行”函数，如果没有则为 0。
amrescan	regproc	PG_PROC.proname	“（重新）开始索引扫描”函数。
amendscan	regproc	PG_PROC.proname	“索引扫描后清理”函数。
ammarkpos	regproc	PG_PROC.proname	“标记当前扫描位置”函数。
amrestrpos	regproc	PG_PROC.proname	“恢复已标记的扫描位置”函数。
ammerge	regproc	PG_PROC.proname	“归并多个索引对象”函数。
ambuild	regproc	PG_PROC.proname	“建立新索引”函数。
ambuildempty	regproc	PG_PROC.proname	“建立空索引”函数。

名称	类型	引用	描述
ambulkdelete	regproc	PG_PROC.proname	批量删除函数。
amvacuumcleanup	regproc	PG_PROC.proname	VACUUM 后的清理函数。
amcanreturn	regproc	PG_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	PG_PROC.proname	估计一个索引扫描开销的函数。
amoptions	regproc	PG_PROC.proname	为一个索引分析和确认 reloptions 的函数。

18.2.36PG_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。

名称	类型	引用	描述
oid	oid	-	行标识符（隐含属性，必须明确选择）。
amopfamily	oid	PG_OPFAMILY.oid	这个项的操作符族。
amoplefttype	oid	PG_TYPE.oid	操作符的左输入类型。可能取值及其描述见于 pg_type.h。
amoprighttype	oid	PG_TYPE.oid	操作符的右输入类型。可能取值及其描述见于 pg_type.h。
amopstrategy	smallint	-	操作符策略数。
amoppurpose	“char”	-	操作符目的，s 为搜索或 o

名称	类型	引用	描述
			为排序。
amopopr	oid	PG_OPERATOR.oid	该操作符的 OID。
amopmethod	oid	PG_AM.oid	索引访问方式操作符族。
amopsortfamily	oid	PG_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。

18.2.37PG_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。

名称	类型	引用	描述
oid	oid	-	行标识符（隐含属性，必须明确选择）。
amprocfamily	oid	PG_OPFAMILY.oid	该项的操作符族。
amproclefttype	oid	PG_TYPE.oid	相关操作符的左输入数据类型。可能取值及其描述见于 pg_type.h。
amprocrighttype	oid	PG_TYPE.oid	相关操作符的右输入数据类型。可能取值及其描述见于 pg_type.h。

名称	类型	引用	描述
amprocnum	smallint	-	支持过程编号。
amproc	regproc	PG_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。

18.2.38PG_APP_WORKLOADGROUP_MAPPING

PG_APP_WORKLOADGROUP_MAPPING 系统表提供了数据库负载映射组的信息。

名称	类型	描述
oid	oid	行标识符（隐含属性，必须明确选择）。
appname	name	应用名称。
workload_gpname	name	映射到的负载组名称。

18.2.39PG_ATTRDEF

PG_ATTRDEF 系统表存储列的默认值。

名称	类型	描述
oid	oid	行标识符（隐含属性，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	字段缺省值或生成表达式的内部表现形式。
adsrc	text	可读缺省值或生成表达式的内部表现形式。
adgenc	“char”	标识该列是否为生成列。取值为's'表示该列为生成列，取值为'\0'

名称	类型	描述
ol		表示该列为普通列，默认值为'\0'。

18.2.40PG_ATTRIBUTE

PG_ATTRIBUTE 系统表存储关于表字段的信息。

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。
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

名称	类型	描述
		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。
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[]	字段属性。目前支持以下两种属性： n_distinct，表示该字段的 distinct 值数量（不包含字表） n_distinct_inherited，表示该字段的 distinct 值数量（包含字表）
attfdwoptions	text[]	外表字段属性。当前支持的 dist_fdw、file_fdw、log_fdw 未使用外表字段属性。
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。

名称	类型	描述
		3. ATT_KV_TIMETAG：时间列。

18.2.41PG_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。

GBase 8c 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。

名称	类型	描述
oid	oid	行标识符（隐含属性，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。 t（true）：表示是。 f（false）：表示不是。
rolinherit	boolean	角色是否自动继承其所属角色的权限。 t（true）：表示自动继承。 f（false）：表示不自动继承。
rolcreaterole	boolean	角色是否可以创建更多角色。 t（true）：表示可以。 f（false）：表示不可以。
rolcreatedb	boolean	角色是否可以创建数据库。 t（true）：表示可以。 f（false）：表示不可以。
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。 t（true）：表示可以。 f（false）：表示不可以。
rolcanlogin	boolean	角色是否可以登录，也就是说，这个角色可以给予

名称	类型	描述
		会话认证标识符。 t（true）：表示可以。 f（false）：表示不可以。
rolreplication	boolean	角色是否具有复制权限： t（true）：表示有。 f（false）：表示没有。
rolauditadmin	boolean	角色是否具有审计管理员权限： t（true）：表示有。 f（false）：表示没有。
rolsystemadmin	boolean	角色是否具有系统管理员权限： t（true）：表示有。 f（false）：表示没有。
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 zone	帐户的有效开始时间，如果没有开始时间，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。 t（true）：表示可以。 f（false）：表示不可以。
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。

名称	类型	描述
roltempspace	text	用户临时表的最大空间限额，单位为 KB。
rolspillspace	text	用户执行作业时下盘数据的最大空间限额，单位为 KB。
rolexcpdata	text	用户可以设置的查询规则（当前未使用）。
rolmonitoradmin	boolean	角色是否具有监控管理员权限： t（true）：表示有。 f（false）：表示没有。
roloperatoradmin	boolean	角色是否具有运维管理员权限： t（true）：表示有。 f（false）：表示没有。
rolpolicyadmin	boolean	角色是否具有安全策略管理员权限： t（true）：表示有。 f（false）：表示没有。

18.2.42 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。

名称	类型	描述
oid	oid	行标识符（隐含属性，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	角色密码密文，加密方式由 GUC 参数 password_encryption_type 确定。

18.2.43 PG_AUTH_MEMBERS

PG_AUTH_MEMBERS 系统表存储显示角色之间的成员关系。

名称	类型	描述

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。

18.2.44PG_CAST

PG_CAST 系统表存储数据类型之间的转化关系。

名称	类型	描述
oid	oid	行标识符（隐含属性，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。

18.2.45PG_CLASS

PG_CLASS 系统表存储数据库对象信息及其之间的关系。

名称	类型	描述
oid	oid	行标识符（隐含属性，必须明确选择）。
relname	name	表、索引、视图等对象的名称。

名称	类型	描述
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。

名称	类型	描述
		CU 描述表（Desc 表）附属于列存表。用于控制表目录中存储数据的可见性。
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 GBase 8c 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。 g：表示临时表。
relkind	“char”	r：表示普通表。 i：表示索引。 I：表示分区表 GLOBAL 索引。 S：表示序列。 L：表示 Large 序列。 v：表示视图。 c：表示复合类型。 t：表示 TOAST 表。 f：表示外表。 m：表示物化视图。
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。
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。

名称	类型	描述
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。
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。 v：表示该表为 HDFS 的 Value 分区表。 s：表示该表为二级分区表。
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 15-15-2。
relreplident	“char”	逻辑解码中解码列的标识：

名称	类型	描述
		d = 默认 （主键，如果存在）。 n = 无。 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	pg_hashbucket 中的桶信息。
relbucketkey	int2vector	哈希分区列号。
relminmxid	xid	该表中所有在这个之前的多事务 ID 已经被一个事务 ID 替换。这用于跟踪该表是否需要为了防止多事务 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)。

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

参数	参数说明
c	CONNECT
T	TEMPORARY
A	ALTER
P	DROP
m	COMMENT
i	INDEX
v	VACUUM
*	给前面权限的授权选项

18.2.46PG_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。

名称	类型	引用	描述
oid	oid	-	行标识符（隐含属性，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	PG_NAMESPACE.oid	包含这个排序规则的名称空间的 OID。
collowner	oid	PG_AUTHID.oid	排序规则的所有者。
collencoding	integer	-	排序规则可用的编码，兼容 PostgreSQL 所有的字符编码类型，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_CTYP

名称	类型	引用	描述
			E。

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。

18.2.47PG_CONSTRAINT

PG_CONSTRAINT 系统表存储表上的检查约束、主键和唯一约束。

名称	类型	描述
oid	oid	行标识符（隐含属性，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。

名称	类型	描述
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。
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。
confmatchtype	“char”	外键匹配类型。 f = 全部。 p = 部分。 u = 未指定（在 f 的基础上允许匹配 NULL 值）。
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。
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。
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。
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。
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。
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。

名称	类型	描述
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。
conincluding	smallint[]	不用做约束，但是会包含在 INDEX 中的属性列。

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。

18.2.48PG_CONVERSION

PG_CONVERSION 系统表描述编码转换信息。

名称	类型	引用	描述
oid	oid	-	行标识符（隐藏属性，必须明确选择）。
conname	name	-	转换名称（在一个名称空间里是唯一的）。
connamespace	oid	PG_NAMESPACE.oid	包含这个转换的名称空间的 OID。
conowner	oid	PG_AUTHID.oid	编码转换的属主。
conforencoding	integer	-	源编码 ID。
contoencoding	integer	-	目的编码 ID。
conproc	regproc	PG_PROC.proname	转换过程。
condefault	boolean	-	如果这是缺省转换则为真。

18.2.49PG_DATABASE

PG_DATABASE 系统表存储关于可用数据库的信息。

名称	类型	描述
oid	oid	行标识符（隐含属性，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。
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。当前版本该字段已经废弃使用，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，当前支持四种兼容模式：A、B、C、PG，分别表示兼容 O、MY、TD 和 POSTGRES。
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。
datminmxid	xid	该数据库中中所有在这个之前的多事务 ID 已经被一个事务 ID 替换。这用于跟踪该数据库是否需要

名称	类型	描述
		为了防止事务 ID 重叠或者允许收缩 pg_clog 而进行清理。它是此数据库中所有表的 pg_class.relminmxid 中的最小值。

18.2.50PG_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。

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，配置方法参考表 15-15-2。

18.2.51PG_DEFAULT_ACL

PG_DEFAULT_ACL 系统表存储为新建对象设置的初始权限。

名称	类型	描述
oid	oid	行标识符（隐含属性，必须明确选择）。
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。 r 表示表或视图。 S 表示序列。 f 表示函数。 T 表示类型。 K 表示客户端主密钥。 k 表示列加密密钥。
defaclacl	aclitem[]	创建该类型时所拥有的访问权限。

18.2.52 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 PG_SHDEPEND，用于记录那些在 GBase 8c 之间共享的对象之间的依赖性关系。

名称	类型	引用	描述
classid	oid	PG_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 0。
refclassid	oid	PG_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 0。
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 PG_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 gs_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。

18.2.53PG_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 PG_SHDESCRIPTION，用于记录 GBase 8c 范围内共享对象的注释。

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	PG_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。

18.2.54PG_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。

名称	类型	描述

名称	类型	描述
oid	oid	行标识符（隐含属性，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。

18.2.55PG_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。

名称	类型	引用	描述
oid	oid	-	行标识符（隐含属性，必须明确选择）。
enumtypid	oid	PG_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。

18.2.56PG_EXTENSION

PG_EXTENSION 系统表存储关于所安装扩展的信息。GBase 8c 默认扩展是 PLPGSQL

和 MOT_FDW。

名称	类型	描述
oid	oid	数据库对象 id。
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。

18.2.57PG_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。

名称	类型	引用	描述
oid	oid	-	行标识符（隐含属性，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。

名称	类型	引用	描述
srcoptions	text[]	-	外部数据源对象的指定选项，使用“keyword=value”格式的字符串。

18.2.58PG_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。

名称	类型	引用	描述
oid	oid	-	行标识符（隐含属性，必须明确选择）。
fdwname	name	-	外部数据封装器名。
fdwowner	oid	PG_AUTHID.oid	外部数据封装器的所有者。
fdwhandler	oid	PG_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	PG_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。

18.2.59PG_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。

名称	类型	引用	描述
oid	oid	-	行标识符（隐含属性，必须明确选择）。

名称	类型	引用	描述
srvname	name	-	外部服务器名。
srvowner	oid	PG_AUTHID.oid	外部服务器的所有者。
srvfdw	oid	PG_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。

18.2.60PG_FOREIGN_TABLE

PG_FOREIGN_TABLE 系统表存储外部表的辅助信息。

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。取值如下： t（true）：表示可写。 f（false）：表示不可写。
ftoptions	text[]	外部表的可选项，具体参考 CREATE FOREIGN TABLE 语法说明。

18.2.61PG_HASHBUCKET

PG_HASHBUCKET 系统表存储 hash bucket 信息。

名称	类型	描述
oid	oid	行标识符（隐含字段，必须明确选择）。
bucketid	oid	对 bucketvector 计算的 hash 值，通过 hash 值可以加速对 bu

名称	类型	描述
		cketvector 的查找。
bucketcnt	integer	包含分片的个数。
bucketmapsize	integer	所有 DN 上包含的分片总数。
bucketref	integer	预留字段，默认值为 1。
bucketvector	oidvector_extend	记录此行 bucket 信息包含的所有 bucket 的 id，在此列上建立唯一索引，具有相同 bucketid 信息的表共享同一行 pg_hashbucket 数据。

18.2.62PG_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。

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。
indisunique	Boolean	如果为真，这是个唯一索引。 如果为假，这不是唯一索引。
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。 如果为假，该索引不是该表的主键。
indisexclusion	Boolean	如果为真，该索引支持排他约束。 如果为假，该索引不支持排他约束。
indimmediate	Boolean	如果为真，在插入数据时会立即进行唯一性检查。 如果为假，在插入数据时不会进行唯一性检查。
indisclustered	Boolean	如果为真，则该表最后在这个索引上建了簇。 如果为假，则该表没有再这个索引上建簇。
indisusable	Boolean	如果为真，该索引对 insert/select 可用。

名称	类型	描述
		如果为假，该索引对 insert/select 不可用。
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。 如果为 false，查询可以用于索引。
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。
indisreplident	Boolean	如果为真，则此索引的列成为逻辑解码的解码列。 如果为假，则此索引的列不是逻辑解码的解码列。
indnkeyatts	smallint	索引中的总字段数，超出 indnatts 的部分不参与索引查询。

18.2.63PG_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。

名称	类型	引用	描述
inhrelid	oid	PG_CLASS.oid	子表的 OID。
inhparent	oid	PG_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。

18.2.64PG_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。

名称	类型	描述
oid	oid	行标识符（隐含属性，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 gaussdb 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
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：

名称	类型	描述
		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。
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。
nspname	name	标识作业执行时的 schema 的名称。
job_name	text	DBE_SCHEDULER 定时任务专用，定时任务名称。
end_date	timestamp without time zone	DBE_SCHEDULER 定时任务专用，定时任务失效时间，时间精确到毫秒。
enable	boolean	DBE_SCHEDULER 定时任务专用，定时任务启用状态：

名称	类型	描述
		true：启用 false：未启用
failure_msg	text	最新一次执行任务报错信息。

18.2.65PG_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。

名称	类型	描述
oid	oid	行标识符（隐含属性，必须明确选择）。
job_id	integer	外键，关联 pg_job 表中的 job_id。
what	text	作业内容，DBE_SCHEDULER 定时任务中的程序内容。
job_name	text	DBE_SCHEDULER 定时任务专用，定时任务或程序名称。

18.2.66PG_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。

名称	类型	引用	描述
oid	oid	-	行标识符（隐含属性，必须明确选择）。
lanname	name	-	语言的名称。
lanowner	oid	PG_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，gs_dump 仍然使用这个东西判断哪种语言需要转

名称	类型	引用	描述
			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。
lanplcallfoid	oid	PG_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	PG_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	PG_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。

18.2.67PG_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。

名称	类型	引用	描述
loid	oid	PG_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。

18.2.68PG_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。

名称	类型	引用	描述
oid	oid	-	行标识符（隐含属性，必须明确选择）。
lomowner	oid	PG_AUTHID.oid	大对象的所有者。
lomacl	aclitem[]	-	访问权限。

18.2.69PG_NAMESPACE

PG_NAMESPACE 系统表存储名称空间，即存储 schema 相关的信息。

名称	类型	描述
oid	oid	行标识符（隐含属性，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。
in_redistribution	“char”	是否处于重发布状态。
nspblockchain	Boolean	如果为真，则该模式为防篡改模式。 如果为假，则此模式为非防篡改模式。

18.2.70PG_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。

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。 l 表示 Large 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。
createcsn	int8	对象创建时的 CSN。
changecsn	int8	对表或索引执行 DDL 操作时的 CSN。

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。

对于升级前创建的对象，再次修改时会记录其修改时间（mtime），对表或索引执行 DDL 操作时会记录其所属事务的事务提交序列号（changecsn）。由于无法得知该对象创建时间，因此 ctime 和 createcsn 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。

createcsn 和 changecsn 记录的是用户当次操作所属事务的事务提交序列号。

18.2.71 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。

名称	类型	引用	描述
oid	oid	-	行标识符（隐含属性，必须明确选择）。
opcmethod	oid	PG_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。
opcnamespace	oid	PG_NAMESPACE.oid	这个操作符类的名称空间。
opcowner	oid	PG_AUTHID.oid	操作符类属主。
opcfamily	oid	PG_OPFAMILY.oid	包含该操作符类的操作符族。
opcintype	oid	PG_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	PG_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。

18.2.72 PG_OPERATOR

PG_OPERATOR 系统表存储有关操作符的信息。

名称	类型	引用	描述
oid	oid	-	行标识符（隐含属性，必须明确选择）。

名称	类型	引用	描述
oprname	name	-	操作符的名称。
oprnamespace	oid	PG_NAMESPACE.oid	包含此操作符的名称空间的 OID。
oprowner	oid	PG_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	-	这个操作符是否支持合并连接。 t（true）：表示支持合并连接。 f（false）：表示不支持合并连接。
oprcanhash	Boolean	-	这个操作符是否支持 Hash 连接。 t（true）：表示支持 Hash 连接。 f（false）：表示不支持 Hash 连接。
oprleft	oid	PG_TYPE.oid	左操作数的类型。
oprright	oid	PG_TYPE.oid	右操作数的类型。
oprresult	oid	PG_TYPE.oid	结果类型。
oprcom	oid	PG_OPERATOR.oid	此操作符的交换符，如果存在的话。
oprnegate	oid	PG_OPERATOR.oid	此操作符的反转器，如果存在的话。
oprcode	regproc	PG_PROC.proname	实现这个操作符的函数。
oprrest	regproc	PG_PROC.proname	此操作符的约束选择性计算函数。
oprjoin	regproc	PG_PROC.proname	此操作符的连接选择性计算函数。

18.2.73PG_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。

名称	类型	引用	描述
oid	oid	-	行标识符（隐含属性，必须明确选择）。
opfmethod	oid	PG_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	PG_NAMESPACE.oid	这个操作符的名称空间。
opfowner	oid	PG_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 PG_AMOP，PG_AMPROC 和 PG_OPCLASS 里。

18.2.74PG_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。

名称	类型	描述
oid	oid	行标识符（隐含属性，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。 's'：table subpartition。

名称	类型	描述
		'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。 'i'：间隔分区。 'l'：list 分区。 'h'：hash 分区。 'n'：无效分区。
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。
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。

名称	类型	描述
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。
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。
relfrozenxid64	xid	冻结事务 ID 号。
relminmxid	xid	冻结多事务 ID 号。

18.2.75PG_PLTEMPLATE

PG_PLTEMPLATE 系统表存储过程语言的“模板”信息。

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。

名称	类型	描述
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）。

18.2.76PG_PROC

PG_PROC 系统表存储函数或过程的信息。

名称	类型	描述
oid	oid	行标识符（隐含属性，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。
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。
proisagg	boolean	函数是聚集函数。 t（true）：表示是。 f（false）：表示不是。
proiswindow	boolean	函数是窗口函数。 t（true）：表示是。 f（false）：表示不是。

名称	类型	描述
prosecdef	boolean	函数是一个安全定义器（也就是一个“setuid”函数）。 t（true）：表示是。 f（false）：表示不是。
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。 t（true）：表示没副作用。 f（false）：表示有副作用。
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依赖于它的输入参数，或者还会被外界因素影响。 i：“不可变的”（immutable）函数，这样的函数对于相同的输入总是产生相同的结果。 s：“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。 v：“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，而因为历史原因，proargtypes 的下标起点为 0。

名称	类型	描述
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。 v 表示 VARIADIC 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。具体请参见《GBase 8c V5_3.0.1_SQL 参考手册》GRANT 和 REVOKE。
prodefaultargpos	int2vector	函数具有默认值的入参的位置。
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。 用户创建的 C 函数，fencedmode 字段默认值均为 true，即 fence 模式；系统内建函数，fencedmode 字段均为 false，即 not fence 模式。
proshippable	boolean	表示该函数是否可以下推到数据库节点上执行，默认

名称	类型	描述
		值是 false。 对于 IMMUTABLE 类型的函数，函数始终可以下推到数据库节点上执行。 对于 STABLE/VOLATILE 类型的函数，仅当函数的属性是 SHIPPABLE 的时候，函数可以下推到数据库节点执行。
propackage	boolean	表示该函数是否支持重载，默认值是 false。 t（true）：表示支持。 f（false）：表示不支持。
prokind	“char”	表示该对象为函数还是存储过程： 值为'f'表示该对象为函数。 值为'p'表示该对象为存储过程。
proargsrc	text	描述兼容 oracle 语法定义的函数或存储过程的参数输入字符串，包括参数注释。默认值为 NULL。
proisprivate	boolean	描述函数是否是 PACKAGE 内的私有函数，默认为 false。
propackageid	oid	函数所属的 package oid，如果不在 package 内，则为 0。
proargtypesext	oidvector_extend	当函数参数较多时，用来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）。
prodefaultargposext	int2vector_extend	当函数参数较多时，函数具有默认值的入参的位置。
allargtypes	oidvector	不区分参数类型，包含存储过程所有参数（包含入参、出参、INOUT 参数）。
allargtypesext	oidvector_extend	当函数参数较多时，用来存放函数参数的数据类型的数组。数组里包含所有参数（包含入参、出参、INOUT 参数）。

18.2.77PG_PUBLICATION

系统表 pg_publication 包含当前数据库中创建的所有 publication。

名称	类型	描述
pubname	text	publication 的名称。
pubowner	oid	publication 的拥有者。
puballtables	bool	如果为真，这个 publication 自动包括数据库中的所有表，包括未来将会创建的任何表。
pubinsert	bool	如果为真，为 publication 中的表复制 INSERT 操作。
pubupdate	bool	如果为真，为 publication 中的表复制 UPDATE 操作。
pubdelete	bool	如果为真，为 publication 中的表复制 DELETE 操作。

18.2.78PG_PUBLICATION_REL

系统表 PG_PUBLICATION_REL 包含当前数据库中的表和 publication 之间的映射，这是一种多对多映射。

名称	类型	引用	描述
prpubid	oid	-	对 publication 的引用。
prrelid	oid	-	对表的引用。

18.2.79PG_RANGE

PG_RANGE 系统表存储关于范围类型的信息。除了 PG_TYPE 里类型的记录。

名称	类型	引用	描述
rngtypid	oid	PG_TYPE.oid	范围类型的 OID。
rngsubtype	oid	PG_TYPE.oid	这个范围类型的元素类型（子类型）的 OID。
rngcollation	oid	PG_COLLATION.oid	用于范围比较的排序规则的 OID，如果没有则为零。
rngsubopc	oid	PG_OPCLASS.oid	用于范围比较的子类型的操作符类的 OID。
rngcanonical	regproc	PG_PROC.proname	转换范围类型为规范格式的函数

名称	类型	引用	描述
			名，如果没有则为 0。
rngsubdiff	regproc	PG_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。

18.2.80PG_REPLICATION_ORIGIN

PG_REPLICATION_ORIGIN 系统表包含所有已创建的复制源，该表为全局共享表，即在每个节点上只有一份 pg_replication_origin，而不是每个数据库一份。

名称	类型	描述
roident	oid	一个集群范围内唯一的复制源标识符。
roname	text	外部的由用户定义的复制源名称。

18.2.81PG_RESOURCE_POOL

PG_RESOURCE_POOL 系统表提供了数据库资源池的信息。

名称	类型	描述
oid	oid	行标识符（隐含属性，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。用作扩容的接口，表示数据重分布时，扫描

名称	类型	描述
		并发度。
memory_limit	name	资源池最大的内存。
parentid	oid	父资源池 OID。
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 GBase 8c 的名称。
is_foreign	boolean	表示资源池是否用于逻辑 GBase 8c 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。
max_worker	integer	只用于扩容的接口，表示扩容数据重分布时，表内并发度。

注：max_dop 和 max_worker 用户扩容，不适用于 GBase 8c。

18.2.82 PG_REWRITE

PG_REWRITE 系统表存储为表和视图定义的重写规则。

名称	类型	描述
oid	oid	行标识符（隐含属性，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。

名称	类型	描述
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。
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）。

18.2.83PG_RLSPOLICY

PG_RLSPOLICY 系统表存储行级访问控制策略。

名称	类型	描述
oid	oid	行标识符（隐含属性，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。

18.2.84PG_SECLABEL

PG_SECLABEL 系统表存储数据对象上的安全标签。

PG_SHSECLABEL 的作用类似，只是它是用于在一个 GBase 8c 内共享的数据库对象的安全标签上的。

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	PG_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。

18.2.85PG_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 GBase 8c 保证在企图删除这些对象之前，这些对象是没有被引用的。

PG_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 GBase 8c 里面所有的数据库之间共享的：每个 GBase 8c 只有一个 PG_SHDEPEND，而不是每个数据库一个。

名称	类型	引用	描述
dbid	oid	PG_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	PG_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	PG_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。

名称	类型	引用	描述
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。

- SHARED_DEPENDENCY_ DBPRIV(d)

  被引用的对象（必须是一个角色）具有依赖对象所对应的 ANY 权限（指定的依赖对象的 OID 对应的是系统表 gs_db_privilege 中一行）。

18.2.86PG_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 psql 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 GBase 8c 里面所有的数据库之间共享的：每个 GBase 8c 只有一个 PG_SHDESCRIPTION，而不是每个数据库一个。

名称	类型	引用	描述

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	PG_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。

18.2.87PG_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 PG_SECLABELS。

PG_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 GBase 8c 中的所有数据库中共享：每个 GBase 8c 只有一个 PG_SHSECLABEL，而不是每个数据库一个。

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	PG_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。

18.2.88PG_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。

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。

名称	类型	描述
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，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，没有更好地办法。 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。

须知： PG_STATISTIC 系统表存储了统计对象的一些敏感信息，如高频值 MCV。系统管理员和授权后的其他用户可以通过访问 PG_STATISTIC 系统表查询到统计对象的这些敏感信息。

18.2.89PG_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。

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，'c'表示普通表，'p'表示分区表。
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。
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

名称	类型	描述
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，没有更好地办法。 n 的取值范围：1¯5
staexprs	pg_node_tree	扩展统计信息对应的表达式。

须知： PG_STATISTIC_EXT 系统表存储了统计对象的一些敏感信息，如高频值 MCV。系统管理员和授权后的其他用户可以通过访问 PG_STATISTIC_EXT 系统表查询到统计对象的这些敏感信息。

18.2.90PG_SUBSCRIPTION

系统表 PG_SUBSCRIPTION 包含所有现有的逻辑复制订阅。需要有系统管理员权限才可以访问此系统表。

和大部分系统表不同，pg_subscription 在数据库实例的所有数据库之间共享，即在每个节点上有只有一份 pg_replication_origin，而不是每个数据库一份。

名称	类型	描述	其它
subdbid	oid	订阅所在的数据库的 OID。	订阅所在的数据库的 OID。
subname	text	订阅的名称。	

名称	类型	描述	其它
subowner	oid	订阅的拥有者。	
subenabled	bool	如果为真，订阅被启用并且应该被复制。	
subconninfo	text	到发布端数据库的连接信息。	
subslotname	text	发布端数据库中复制槽的名称。空表示为 NONE。	
subpublications	text[]	被订阅的 publication 名称的数组。这些引用的是发布者服务器上的 publication。	

18.2.91PG_SYNONYM

PG_SYNONYM 系统表存储同义词对象名与其他数据库对象名间的映射信息。

名称	类型	描述
oid	oid	数据库对象 id。
synname	name	同义词名称。
synnamespace	oid	包含该同义词的名字空间的 OID。
synowner	oid	同义词的所有者，通常是创建它的用户 OID。
synobjschema	name	关联对象指定的模式名。
synobjname	name	关联对象名。

18.2.92PG_TABLESPACE

PG_TABLESPACE 系统表存储表空间信息。

名称	类型	描述
oid	oid	行标识符（隐含属性，必须明确选择）。
spcname	name	表空间名称。

名称	类型	描述
spcowner	oid	表空间的所有者，通常是创建它的人。
spcacl	aclitem[]	详见《GBase 8c V5_3.0.1_SQL 参考手册》中 GRANT 和 REVOKE 章节。
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。

18.2.93PG_TRIGGER

PG_TRIGGER 系统表存储触发器信息。

名称	类型	描述
oid	oid	行标识符（隐含属性，必须明确选择）。
tgrelid	oid	触发器所在表的 OID。
tgname	name	触发器名。
tgfoid	oid	要被触发器调用的函数。
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。
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。

名称	类型	描述
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	触发器的所有者。

18.2.94PG_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 PG_TS_CONFIG_MAP 里面的辅助记录定义的。

名称	类型	引用	描述
oid	oid	-	行标识符（隐含属性，必须明确选择）。
cfgname	name	-	文本搜索配置名。
cfgnamespace	oid	PG_NAMESPACE.oid	包含这个配置的名称空间的 OID。
cfgowner	oid	PG_AUTHID.oid	配置的所有者。
cfgparser	oid	PG_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。

18.2.95PG_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。

名称	类型	引用	描述
mapcfg	oid	PG_TS_CONFIG.oid	拥有这个映射记录的 PG_TS_CONFIG 记录的 OID。

名称	类型	引用	描述
maptokentype	integer	-	由配置的解析器产生的一个符号类型值。
mapseqno	integer	-	在相同 mapcfg 或 maptokentype 值的情况下，该符号类型的顺序号。
mapdict	oid	PG_TS_DICT.oid	要咨询的文本搜索字典的 OID。

18.2.96PG_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。

名称	类型	引用	描述
oid	oid	-	行标识符（隐含属性，必须明确选择）。
dictname	name	-	文本搜索字典名。
dictnamespace	oid	PG_NAMESPACE.oid	包含这个字典的名称空间的 OID。
dictowner	oid	PG_AUTHID.oid	字典的所有者。
dicttemplate	oid	PG_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。

18.2.97PG_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。

名称	类型	引用	描述

名称	类型	引用	描述
oid	oid	-	行标识符（隐含属性，必须明确选择）。
prsname	name	-	文本搜索解析器名。
prsnamespace	oid	PG_NAMESPACE.oid	包含这个解析器的名称空间的 OID。
prsstart	regproc	PG_PROC.proname	解析器的启动函数名。
prstoken	regproc	PG_PROC.proname	解析器的下一个标记函数名。
prsend	regproc	PG_PROC.proname	解析器的关闭函数名。
prsheadline	regproc	PG_PROC.proname	解析器的标题函数名。
prslextype	regproc	PG_PROC.proname	解析器的 lextype 函数名。

18.2.98PG_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。

名称	类型	引用	描述
oid	oid	-	行标识符（隐含属性，必须明确选择）。
tmplname	name	-	文本搜索模板名。
tmplnamespace	oid	PG_NAMESPACE.oid	包含这个模板的名称空间的 OID。
tmplinit	regproc	PG_PROC.proname	模板的初始化函数名。
tmpllexize	regproc	PG_PROC.proname	模板的 lexize 函数名。

18.2.99PG_TYPE

PG_TYPE 系统表存储数据类型的相关信息。

名称	类型	描述
oid	oid	行标识符（隐含属性，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（该值为 true）还是传引用（该值为 false）。如果该类型的 TYPLEN 不是 1、2、4、8， TYPBYVAL 最好为 false。变长类型通常是传引用。即使 TYPLEN 允许传值，TYPBYVAL 也可以为 false。
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。
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

名称	类型	描述
		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 GBase 8c 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。 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。

名称	类型	描述
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	该类型是否存在 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。取值参考 PG_COLLATION 系统表。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 为非 NULL，则 typdefault 必须包含一个 typdefaultbin 代表的缺省表达式。 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，可以把它作为该类型的输入，转换器生成一个常量。
typacl	aclitem[]	访问权限。

18.2.100 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 PG_USER_MAPPINGS 进行查询。

名称	类型	引用	描述
oid	oid	-	行标识符（隐含属性，必须明确选择）。
umuser	oid	PG_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	PG_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。

18.2.101 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

名称	类型	描述
oid	oid	行标识符（隐含字段，必须明确选择）。
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。
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。

名称	类型	描述
tempspace	bigint	角色已经使用的临时表存储空间大小。
passwordexpired	smallint	密码是否失效。 0：密码有效。 1：密码失效。

18.2.102 PG_WORKLOAD_GROUP

PG_WORKLOAD_GROUP 系统表提供了数据库负载组的信息。

名称	类型	描述
oid	oid	行标识符（隐含属性，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。

18.2.103 PGXC_CLASS

PGXC_CLASS 系统表存储每张表的复制或分布信息。PGXC_CLASS 系统表仅在分布式场景下有具体含义，GBase 8c 只能查询表定义。

名称	类型	描述
pcrelid	oid	表的 OID。
pclocatortype	“char”	定位器类型。 H：hash G：Range L：List M：Modulo N：Round Robin R：Replication
pchashalgorithm	smallint	使用哈希算法分布元组。
pchashbuckets	smallint	哈希容器的值。

名称	类型	描述
pgroup	name	节点群的名称。
redistributed	“char”	表已经完成重分布。
redis_order	integer	重分布的顺序。该值等于 0 的表在本轮重分布过程中不进行重分布。
pcattnum	int2vector	用作分布键的列标号。
nodeoids	oidvector_extend	表分布的节点 OID 列表。
options	text	系统内部保留字段，存储扩展状态信息。

18.2.104 PGXC_GROUP

PGXC_GROUP 系统表存储节点组信息。PGXC_GROUP 系统表仅在分布式场景下有具体含义，GBase 8c 只能查询表定义。

名称	类型	描述
oid	oid	行标识符（隐含字段，必须明确选择）。
group_name	name	节点组名称。
in_redistribution	“char”	是否需要重分布。取值包括 n，y，t。 n：表示 NodeGroup 没有再进行重分布。 y：表示 NodeGroup 是重分布过程中的源节点组。 t：表示 NodeGroup 是重分布过程中的目的节点组。
group_members	oidvector_extend	节点组的节点 OID 列表。
group_buckets	text	分布数据桶的集合。
is_installation	Boolean	是否安装子集群。 t（true）：表示安装。 f（false）：表示不安装。
group_acl	aclitem[]	访问权限。
group_kind	“char”	node group 类型，取值包括 i，n，v，e。

名称	类型	描述
		i：表示 installation node group。 n：表示普通非逻辑集群 node group。 v：表示逻辑集群 node group。 e：表示弹性集群。
group_parent	oid	如果是子 node group，该字段表示父 node group 的 OID，如果是父 node group，该字段值为空。

18.2.105 PGXC_NODE

PGXC_NODE 系统表存储集群节点信息。PGXC_NODE 系统表仅在分布式场景下有具体含义，GBase 8c 只能查询表定义。

名称	类型	描述
oid	oid	行标识符（隐含字段，必须明确选择）。
node_name	name	节点名称。
node_type	“char”	节点类型。 C：协调节点。 D：数据节点。 S：数据节点的备节点。
node_port	integer	节点的端口号。
node_host	name	节点的主机名称或者 IP（如配置为虚拟 IP，则为虚拟 IP）。
node_port1	integer	复制节点的端口号。
node_host1	name	复制节点的主机名称或者 IP（如配置为虚拟 IP，则为虚拟 IP）。
hostis_primary	Boolean	表明当前节点是否发生主备切换。 t（true）：表示发生。 f（false）：表示不发生。
nodeis_primary	Boolean	在 replication 表下，是否优选当前节点作为优先执行的节点进行非查询操作。 t（true）：表示优选。

名称	类型	描述
		f（false）：表示不优选。
nodeis_preferred	Boolean	在 replication 表下，是否优选当前节点作为首选的节点进行查询。 t（true）：表示优选。 f（false）：表示不优选。
node_id	integer	节点标志符。由 node_name 经过 hash 函数计算后得到。
sctp_port	integer	主节点使用 TCP 代理通信库或 SCTP 通信库的数据通道侦听端口。
control_port	integer	主节点使用 TCP 代理通信库或 SCTP 通信库的控制通道侦听端口。
sctp_port1	integer	备节点使用 TCP 代理通信库或 SCTP 通信库的数据通道侦听端口。
control_port1	integer	备节点使用 TCP 代理通信库或 SCTP 通信库的控制通道侦听端口。
nodeis_central	Boolean	表明当前节点是否为中心控制节点，只用于 CN，对 DN 无效。 t（true）：表示是。 f（false）：表示不是。
nodeis_active	Boolean	表明当前节点是否是正常状态，用于标记 CN 是否被剔除，对 DN 无效。 t（true）：表示是。 f（false）：表示不是。

18.2.106 PGXC_SLICE

PGXC_SLICE 表是针对 range 范围分布和 list 分布创建的系统表，用来记录分布具体信息，当前不支持 range interval 自动扩展分片，不过在系统表中预留。

PGXC_SLICE 系统表仅在分布式场景下有具体含义，GBase 8c 只能查询表定义。

名称	类型	描述
relname	name	表名或者分片名，通过 type 区分

名称	类型	描述
type	char	当为’t’时 relname 是表名，当为’s’时 relname 是分片的名字
strategy	char	‘r’：为 range 分布表 ‘l’：为 list 分布表 后续 interval 分片会扩展该值
relid	oid	该 tuple 隶属的分布表 oid
referenceoid	oid	所参考分布表的 oid,用于 slice reference 建表语法
sindex	int	当为 list 分布表时，表示当前 boundary 在某个分片内的位置
interval	text[]	预留字段
transitboundary	text[]	预留字段
transitno	int	预留字段
nodeoid	oid	当 relname 为分片名时，表示该分片的数据存放在哪个 DN 上，nodeoid 表示这个 DN 的 oid
boundaries	text[]	当 relname 为分片名时，对应该分片的边界值
specified	boolean	当前分片对应的 DN 是否是用户在 DDL 中显示指定的
sliceorder	int	用户定义分片的顺序

18.2.107 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。

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

名称	类型	描述
		限制。
statement_id	varchar2(30)	用户输入的查询标签。
plan_id	bigint	查询标识。该标识在计划生成阶段自动产生，供内核工程师调试使用。
id	int	计划中的节点编号。
operation	varchar2(30)	操作描述。
options	varchar2(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	varchar2(30)	对象类型。
object_owner	name	对象所属 schema，来自于用户定义。
projection	varchar2(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 PLAN_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 PLAN_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。

18.2.108 STREAMING_CONT_QUERY

STREAMING_CONT_QUERY 系统表存储所有 CONTVIEW 对象的元数据信息。

名称	类型	描述
id	int4	CONTVIEW 对象唯一的标识符，不可重复。
type	char	标识 CONTVIEW 的类型。 'c'表示该 CONTVIEW 是基于列存存储模型。 'r'表示该 CONTVIEW 是基于行存存储模型。 'p'表示该 CONTVIEW 是基于分区列存存储模型。
relid	oid	CONTVIEW 对象的 OID。
defrelid	oid	CONTVIEW 对应的持续计算规则 VIEW 的 OID。
active	bool	标识 CONTVIEW 是否处于持续计算状态。 t（true）：表示是。 f（false）：表示不是。
streamrelid	oid	CONTVIEW 对应的 STREAM 的 OID。
matrelid	oid	CONTVIEW 对应物化表的 OID。
lookupidxid	oid	CONTVIEW 对应 GROUP LOOK UP INDEX 的 OID，此字段内部使用，仅行存具有。
step_factor	int2	标识 CONTVIEW 的步进模式。主要取值为 0（无重叠窗口）和 1（滑动窗口，步长为 1）。
ttl	int4	CONTVIEW 设置的 ttl_interval 参数值。
ttl_attno	int2	CONTVIEW 设置的 TTL 功能对应时间列的字段编号。
dictrelid	oid	CONTVIEW 对应字典表的 OID。
grpnum	int2	CONTVIEW 持续计算规则中维度列的个数，此字段内部使用。
grpidx	int2vector	CONTVIEW 持续计算规则中维度列在 TARGET LIST 的索引，此字段内部使用。

18.2.109 STATEMENT_HISTORY

获得当前节点的执行语句的信息。查询系统表必须具有 sysadmin 权限。只可在系统库中查询到结果，用户库中无法查询。

对于此系统表查询有如下约束：

必须在 postgres 库内查询，其它库中不存数据。

此系统表受 track_stmt_stat_level 控制，默认为“OFF,L0”，第一部分控制 Full SQL，第二部分控制 Slow SQL，具体字段记录级别见下表。

对于 Slow SQL，当 track_stmt_stat_level 的值为非 OFF 时，且 SQL 执行时间超过 log_min_duration_statement，会记录为慢 SQL。

名称	类型	描述	记录级别
db_name	name	数据库名称。	L0
schema_name	name	schema 名称。	L0
origin_node	integer	节点名称。	L0
user_name	name	用户名。	L0
application_name	text	用户发起的请求的应用程序名称。	L0
client_addr	text	用户发起的请求的客户端地址。	L0
client_port	integer	用户发起的请求的客户端端口。	L0
unique_query_id	bigint	归一化 SQL ID。	L0
debug_query_id	bigint	唯一 SQL ID。	L0
query	text	归一化 SQL。	L0
start_time	timestamp with time zone	语句启动的时间。	L0
finish_time	timestamp with	语句结束的时间。	L0

名称	类型	描述	记录级别
	time zone		
slow_sql_threshold	bigint	语句执行时慢 SQL 的标准。	L0
transaction_id	bigint	事务 ID。	L0
thread_id	bigint	执行线程 ID。	L0
session_id	bigint	用户 session id。	L0
n_soft_parse	bigint	软解析次数，n_soft_parse + n_hard_parse 可能大于 n_calls，因为子查询未计入 n_calls。	L0
n_hard_parse	bigint	硬解析次数，n_soft_parse + n_hard_parse 可能大于 n_calls，因为子查询未计入 n_calls。	L0
query_plan	text	语句执行计划。	L1
n_returned_rows	bigint	SELECT 返回的结果集行数。	L0
n_tuples_fetched	bigint	随机扫描行。	L0
n_tuples_returned	bigint	顺序扫描行。	L0
n_tuples_inserted	bigint	插入行。	L0
n_tuples_updated	bigint	更新行。	L0
n_tuples_deleted	bigint	删除行。	L0
n_blocks_fetched	bigint	buffer 的块访问次数。	L0

名称	类型	描述	记录级别
n_blocks_hit	bigint	buffer 的块命中次数。	L0
db_time	bigint	有效的 DB 时间花费，多线程将累加（单位：微秒）。	L0
cpu_time	bigint	CPU 时间（单位：微秒）。	L0
execution_time	bigint	执行器内执行时间（单位：微秒）。	L0
parse_time	bigint	SQL 解析时间（单位：微秒）。	L0
plan_time	bigint	SQL 生成计划时间（单位：微秒）。	L0
rewrite_time	bigint	SQL 重写时间（单位：微秒）。	L0
pl_execution_time	bigint	plpgsql 上的执行时间（单位：微秒）。	L0
pl_compilation_time	bigint	plpgsql 上的编译时间（单位：微秒）。	L0
data_io_time	bigint	IO 上的时间花费（单位：微秒）。	L0
net_send_info	text	通过物理连接发送消息的网络状态，包含时间（微秒）、调用次数、吞吐量（字节）。通过该字段可以分析 SQL 在分布式系统下的网络开销，单机模式下不支持该字段。例如：{“time”:xxx，“n_calls”:xxx，“size”:xxx}。	L0
net_recv_info	text	通过物理连接接收消息的网络状态，包含时间（微秒）、调用次数、吞吐量（字节）。通过该字段可以分析 SQL 在分布式系统下的网络开销，单机模式下不支持该字段。例如：{“time”:xxx，“n_calls”:xxx，“size”:xxx}。	L0
net_stream_send_info	text	通过逻辑连接发送消息的网络状态，包含时间（微秒）、调用次数、吞吐量（字节）。通过该字段可以分析 SQL 在分布式系统下的网络开销，单机模式下不支持该字段。例如：{“time”:	L0

名称	类型	描述	记录级别
		xxx，“n_calls”:xxx，“size”:xxx}。	
net_stream_recv_info	text	通过逻辑连接接收消息的网络状态，包含时间（微秒）、调用次数、吞吐量（字节）。通过该字段可以分析 SQL 在分布式系统下的网络开销，单机模式下不支持该字段。例如：{“time”:xxx，“n_calls”:xxx，“size”:xxx}。	L0
lock_count	bigint	加锁次数。	L0
lock_time	bigint	加锁耗时。	L1
lock_wait_count	bigint	加锁等待次数。	L0
lock_wait_time	bigint	加锁等待耗时。	L1
lock_max_count	bigint	最大持锁数量。	L0
lwlock_count	bigint	轻量级加锁次数（预留）。	L0
lwlock_wait_count	bigint	轻量级等锁次数。	L0
lwlock_time	bigint	轻量级加锁时间（预留）。	L1
lwlock_wait_time	bigint	轻量级等锁时间。	L1
details	bytea	语句锁事件的列表，该列表按时间书序记录事件，记录的数量受参数 track_stmt_details_size 的影响。该字段为二进制，需要借助解析函数 pg_catalog.statement_detail_decode 读取，见《GBase 8c V5_3.0.1_SQL 参考手册》中其他函数章节 statement_detail_decode 参数说明。 事件包括： 加锁开始 加锁结束 等锁开始	L2

名称	类型	描述	记录级别
		等锁结束 放锁开始 放锁结束 轻量级等锁开始 轻量级等锁结束	
is_slow_sql	boolean	该 SQL 是否为 slow SQL。 t（true）：表示是。 f（false）：表示不是。	L0
trace_id	text	驱动传入的 trace id，与应用的一次请求相关联。	L0

18.2.110 STREAMING_REAPER_STATUS

STREAMING_REAPER_STATUS 系统表存储流引擎 reaper 线程的状态信息。

名称	类型	描述
id	int4	CONTVIEW 对象唯一的标识符，不可重复。
contquery_name	name	CONTVIEW 对象的名称。
gather_interval	text	CONTVIEW 对象设置的 gather_interval 参数值（自动聚合特定时间前历史数据的时间参数）。
gather_completion_time	text	CONTVIEW 对象最近一次的 GATHER（历史数据聚合）的完成时间。

18.2.111 STREAMING_STREAM

STREAMING_STREAM 系统表存储所有 STREAM 对象的元数据信息。

名称	类型	描述
relid	oid	STREAM 对象的 OID。

名称	类型	描述
queries	bytea	该 STREAM 对应 CONTVIEW 的位图映射。

18.2.112 SYS_DUMMY

SYS_DUMMY 系统表用于计算表达式常量的值。

名称	类型	描述
DUMMY	VARCHAR(1)	表达式常量的值。

18.3 系统视图

18.3.1 DV_SESSIONS

DV_SESSIONS 视图存储当前会话的所有会话信息。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。

名称	类型	描述
sid	bigint	当前活动的后台线程的 OID。
serial#	integer	当前活动的后台线程的序号，在 GBase 8c 中为 0。
user#	oid	登录此后台线程的用户的 OID。oid 为 0 表示此后台线程为全局辅助线程(auxiliary)。
username	name	登录此后台线程的用户名。username 为空表示此后台线程为全局辅助线程(auxiliary)。 可以通过和 pg_stat_get_activity() 关联查询，识别出 application_name。 例如： select s.*,a.application_name from DV_SESSIONS as s left join pg_stat_get_activity(NULL) as a on s.sid=a.sessionid;

18.3.2 DV_SESSION_LONGOPS

DV_SESSION_LONGOPS 视图存储当前正在执行的操作的进度。该视图需要授权访问。

名称	类型	描述
sid	bigint	当前正在执行的后台进程的 OID。
serial#	integer	当前正在执行的后台进程的序号，在 GBase 8c 中为 0。
sofar	integer	目前完成的工作量，在 GBase 8c 中为空。
totalwork	integer	工作总量，在 GBase 8c 中为空。

18.3.3 GS_AUDITING

GS_AUDITING 视图显示对数据库相关操作的所有审计信息。需要有系统管理员或安全策略管理员权限才可以访问此视图。

名称	类型	描述
polname	name	策略名称，需要唯一，不可重复。
pol_type	text	审计策略类型，值为‘access’或者‘privilege’。 access：表示审计 DML 操作。 privilege：表示审计 DDL 操作。
polenabled	boolean	用来表示策略启动开关。 t（true）：表示启动。 f（false）：表示不启动。
access_type	bigint	DML 数据库操作相关类型。例如 SELECT、INSERT、DELETE 等。
label_name	name	资源标签名称。对应系统表 gs_auditing_policy 中的 polname 字段。
priv_object	name	用来描述数据库资产的路径。
filter_name	text	过滤条件的逻辑字符串。

18.3.4 GS_AUDITING_ACCESS

GS_AUDITING_ACCESS 视图显示对数据库 DML 相关操作的所有审计信息。需要有系统管理员或安全策略管理员权限才可以访问此视图。

名称	类型	描述
polname	name	策略名称，需要唯一，不可重复。
pol_type	text	审计策略类型，值为‘access’，表示审计 DML 操作。
polenabled	boolean	用来表示策略启动开关。
access_type	name	DML 数据库操作相关类型。例如 SELECT、INSERT、DELETE 等。
label_name	name	资源标签名称。对应系统表 gs_auditing_policy 中的 polname 字段。
access_object	text	用来描述数据库资产的路径。
filter_name	text	过滤条件的逻辑字符串。

18.3.5 GS_AUDITING_PRIVILEGE

GS_AUDITING_PRIVILEGE 视图显示对数据库 DDL 相关操作的所有审计信息。需要有系统管理员或安全策略管理员权限才可以访问此视图。

名称	类型	描述
polname	name	策略名称，需要唯一，不可重复。
pol_type	text	审计策略类型，值为‘privilege’，表示审计 DDL 操作。
polenabled	boolean	用来表示策略启动开关。
access_type	name	DDL 数据库操作相关类型。例如 CREATE、ALTER、DROP 等。
label_name	name	资源标签名称。对应系统表 gs_auditing_policy 中的 polname 字段。
priv_object	text	带有数据库对象的全称域名。

名称	类型	描述
filter_name	text	过滤条件的逻辑字符串。

18.3.6GS_DB_PRIVILEGES

GS_DB_PRIVILEGES 系统视图记录 ANY 权限的授予情况，每条记录对应一条授权信息。

名称	类型	描述
rolename	name	用户名。
privilege_type	text	用户拥有的 ANY 权限，取值参考《GBasee 8c V5_3.0.1_SQL 参考手册》中“GRANT”章节 ANY 权限列表。
admin_option	boolean	是否具有 privilege_type 列记录的 ANY 权限的再授权权限。 yes：表示具有。 no：表示不具有。

18.3.7 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。

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。

名称	类型	描述
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。

18.3.8 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销。
- CPU_TIME：CPU 的时间花销。
- EXECUTION_TIME：执行器内的时间花销。
- PARSE_TIME：SQL 解析的时间花销。
- PLAN_TIME：生成 Plan 的时间花销。
- REWRITE_TIME：SQL 重写的时间花销。
- PL_EXECUTION_TIME：plpgsql（存储过程）执行的时间花销。
- PL_COMPILATION_TIME：plpgsql（存储过程）编译的时间花销。
- NET_SEND_TIME：网络上的时间花销。
- DATA_IO_TIME：IO 的时间花销。

名称	类型	描述
stat_id	integer	统计编号。
stat_name	text	类型名称。

名称	类型	描述
value	bigint	时间值（单位：微秒）。

18.3.9 GS_LABELS

GS_LABELS 视图显示所有已配置的资源标签信息。需要有系统管理员或安全策略管理员权限才可以访问此视图。

名称	类型	描述
labelname	name	资源标签的名称。
labeltype	name	资源标签的类型。对应系统表 GS_POLICY_LABEL 中的 labeltype 字段。
fqdntype	name	数据库资源的类型。如 table、schema、index 等。
schemaname	name	数据库资源所属的 schema 名称。
fqdnname	name	数据库资源名称。
columnname	name	数据库资源列名称，若标记的数据库资源不为表的列则该项为空。

18.3.10GS_MASKING

GS_MASKING 视图显示所有已配置的动态脱敏策略信息。需要有系统管理员或安全策略管理员权限才可以访问此视图。

名称	类型	描述
polname	name	脱敏策略名称。
polenabled	boolean	脱敏策略开关。
maskaction	name	脱敏函数。
labelname	name	脱敏函数作用的标签名称。
masking_object	text	脱敏数据库资源对象。
filter_name	text	过滤条件的逻辑表达式。

18.3.11 GS_MATVIEWS

GS_MATVIEWS 视图提供了关于数据库中每一个物化视图的信息。

名称	类型	引用	描述
schemaname	name	PG_NAMESPACE.nspname	物化视图的模式名。
matviewname	name	PG_CLASS.relname	物化视图名。
matviewowner	name	PG_AUTHID.Erolname	物化视图的所有者。
tablespace	name	PG_TABLESPACE.spcname	物化视图的表空间名（如果使用数据库默认表空间则为空）。
hasindexes	boolean	-	如果物化视图有（或者最近有过）任何索引，则此列为真。
definition	text	-	物化视图的定义（一个重构的 SELECT 查询）。

18.3.12 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。

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。

18.3.13 GS_REDO_STAT

GS_REDO_STAT 视图用于统计回话线程日志回放情况。

名称	类型	描述
phywrts	bigint	日志回放过程中写数据的次数。

名称	类型	描述
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。

18.3.14GS_SESSION_CPU_STATISTICS

GS_SESSION_CPU_STATISTICS 视图显示当前用户执行的正在运行的复杂作业的 CPU 使用的负载管理信息。

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。
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。

18.3.15GS_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 gaussdb 线程和 Stream 线程分配的所有内存。当 GUC 参数 enable_memory_limit 的值为 off 时，本视图不可用。

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。

18.3.16GS_SESSION_MEMORY_CONTEXT

GS_SESSION_MEMORY_CONTEXT 视图统计所有的会话的内存使用情况，以 MemoryContext 节点来统计。该视图仅在开启线程池（enable_thread_pool = on）时生效。当 GUC 参数 enable_memory_limit 的值为 off 时，本视图不可用。

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。

名称	类型	描述
sessid	text	会话启动时间+会话标识（字符串信息为 timestamp.sessionid）。
threadid	bigint	会话绑定的线程标识，如果未绑定线程，该值为-1。
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。
parent	text	父内存上下文名称。

名称	类型	描述
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。

18.3.17GS_SESSION_MEMORY_DETAIL

GS_SESSION_MEMORY_DETAIL 统计会话的内存使用情况，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。当 GUC 参数 enable_memory_limit 的值为 off 时，本视图不可用。

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“$GAUSSLOG/pg_log/${node_name}/dumpmem”目录下的“threadid_timestamp.log”文件中。其中 threadid 可通过下表 sessid 中获得。

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。
parent	text	父内存上下文名称。

名称	类型	描述
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。

18.3.18GS_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。

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。 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。

18.3.19GS_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。

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。

18.3.20GS_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。

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。
value	bigint	会话值。

18.3.21GS_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 GBase 8c 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 GBase 8c 内部进行的查询。

名称	类型	描述

名称	类型	描述
node_name	text	节点名称。
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	DML 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）。

名称	类型	描述
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）。

18.3.22GS_THREAD_MEMORY_CONTEXT

GS_THREAD_MEMORY_CONTEXT 视图统计所有的线程的内存使用情况，以 MemoryContext 节点来统计。该视图在关闭线程池（enable_thread_pool = off）时等价于 GS_SESSION_MEMORY_DETAIL 视图。当 GUC 参数 enable_memory_limit 的值为 off 时，本视图不可用。

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。

名称	类型	描述
threadid	text	线程启动时间+线程标识（字符串信息为 timestamp.sessionid）。
tid	bigint	线程标识。
thrdtype	text	线程类型。
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。
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。

名称	类型	描述
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。

18.3.23GS_TOTAL_MEMORY_DETAIL

GS_GSC_MEMORY_DETAIL 视图描述当前节点当前进程的全局 SysCache 的内存占用情况，仅在开启 GSC 的模式下有数据。需要注意的是，这个查询由于是以数据库内存上下文分隔的，因此会缺少一部分内存的统计，缺失的内存统计对应的内存上下文名称为 GlobalSysDBCache。

名称	类型	描述
db_id	integer	数据库 id。
totalsize	bigint	共享内存总大小，单位 Byte。
freesize	bigint	共享内存剩余大小，单位 Byte。
usedsize	bigint	共享内存使用大小，单位 Byte。

18.3.24GS_WLM_CGROUP_INFO

GS_WLM_CGROUP_INFO 视图显示当前执行作业的控制组的信息。

名称	类型	描述
cgoup_name	text	控制组的名称。
priority	interger	作业的优先级。
usage_pecent	interger	控制组占用的百分比。
shares	bigint	控制组分配的 CPU 资源配额。
cpuacct	bigint	CPU 配额分配。
cpuset	text	CPU 限额分配。

名称	类型	描述
relpath	text	控制组的相对路径。
valid	text	该控制组是否有效。
node_group	text	逻辑数据库实例名称。

18.3.25GS_WLM_EC_OPERATOR_STATISTICS

GS_WLM_EC_OPERATOR_STATISTICS 视图显示当前用户正在执行的 EC（Extension Connector）作业的算子相关信息。查询该视图需要 sysadmin 权限。

名称	类型	描述
queryid	bigint	EC 语句执行使用的内部 query_id。
plan_node_id	integer	EC 算子对应的执行计划的 plan node id。
start_time	timestamp with time zone	EC 算子处理第一条数据的开始时间。
ec_status	text	EC 作业的执行状态。 EC_STATUS_INIT：初始化。 EC_STATUS_CONNECTED：已连接。 EC_STATUS_EXECUTED：已执行。 EC_STATUS_FETCHING：获取中。 EC_STATUS_END：已结束。
ec_execute_datanode	text	执行 EC 作业的 DN 名称。
ec_dsn	text	EC 作业所使用的 DSN。
ec_username	text	EC 作业访问远端数据库实例的 USERNAME（远端集群为 SPARK 类型时该值为空）。
ec_query	text	EC 作业发送给远端数据库实例执行的语句。
ec_libodbc_type	text	EC 作业使用的 unixODBC 驱动类型。 类型 1：对应 libodbc.so.1。 类型 2：对应 libodbc.so.2。

名称	类型	描述
ec_fetch_count	bigint	EC 作业当前处理的数据条数。

18.3.26GS_WLM_EC_OPERATOR_HISTORY

GS_WLM_EC_OPERATOR_HISTORY 视图显示的是当前用户在当前 CN 上执行 EC（Extension Connector）作业结束后的算子相关记录。此视图查询数据的记录会被定时清理，清理周期为 3 分钟。

- 当 GUC 参数 enable_resource_record 为 on 时，视图中的记录每隔 3 分钟被转储到系统表 GS_WLM_EC_OPERATOR_INFO 中一次，同时视图中的记录被删除；
- 当 GUC 参数 enable_resource_record 为 off 时，记录在视图中的存留时间达到超期时间（超期时间为 3 分钟）后会被删除。记录的数据同 GS_WLM_EC_OPERATOR_INFO 的字段。

18.3.27GS_WLM_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。

记录的数据同 GS_WLM_PLAN_OPERATOR_INFO。

18.3.28GS_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。查询该视图需要 sysadmin 权限。

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 wi	该算子处理第一条数据的开始时间。

名称	类型	描述
	th time zone	
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	当前算子在数据库实例上的最小内存峰值（MB）。
max_peak_memory	integer	当前算子在数据库实例上的最大内存峰值（MB）。
average_peak_memory	integer	当前算子在数据库实例上的平均内存峰值（MB）。
memory_skew_percent	integer	当前算子在数据库实例间的内存使用倾斜率。
min_spill_size	integer	若发生下盘，数据库实例上下盘的最小数据量（MB），默认为 0。
max_spill_size	integer	若发生下盘，数据库实例上下盘的最大数据量（MB），默认为 0。
average_spill_size	integer	若发生下盘，数据库实例上下盘的平均数据量（MB），默认为 0。
spill_skew_percent	integer	若发生下盘，数据库实例间下盘倾斜率。
min_cpu_time	bigint	该算子在数据库实例上的最小执行时间（ms)。
max_cpu_time	bigint	该算子在数据库实例上的最大执行时间（ms)。
total_cpu_time	bigint	该算子在数据库实例上的总执行时间（ms）。
cpu_skew_percent	integer	数据库实例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

名称	类型	描述
		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

18.3.29GS_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。

记录的数据同表 GS_WLM_PLAN_OPERATOR_INFO。

18.3.30GS_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。查询该视图需要 sysadmin 权限。

名称	类型	描述
gs_wlm_rebuild_user_resource_pool	boolean	重建内存中用户资源池信息结果。t 为成功，f 为失败。

18.3.31GS_WLM_RESOURCE_POOL

这是资源池上的一些统计信息。

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。
parentid	oid	父资源池的 OID。
ref_count	integer	关联到该资源池上的作业数量。

名称	类型	描述
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。

18.3.32GS_WLM_SESSION_HISTORY

GS_WLM_SESSION_HISTORY 视图显示当前用户在数据库实例上执行作业结束后的负载管理记录。查询该视图需要 sysadmin 或者 monitor admin 权限。

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	语句执行的数据库实例名称。
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。

名称	类型	描述
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	语句所使用的 Cgroup。
estimate_memory	integer	语句估算内存大小。
min_peak_memory	integer	语句在数据库实例上的最小内存峰值，单位 MB。
max_peak_memory	integer	语句在数据库实例上的最大内存峰值，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	语句数据库实例间的内存使用倾斜率。
spill_info	text	语句在数据库实例上的下盘信息： None：数据库实例均未下盘。 All：数据库实例均下盘。 [a:b]：数量为 b 个数据库实例中有 a 个数据库实例下盘。

名称	类型	描述
min_spill_size	integer	若发生下盘，数据库实例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，数据库实例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，数据库实例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，数据库实例间下盘倾斜率。
min_dn_time	bigint	语句在数据库实例上的最小执行时间，单位 ms。
max_dn_time	bigint	语句在数据库实例上的最大执行时间，单位 ms。
average_dn_time	bigint	语句在数据库实例上的平均执行时间，单位 ms。
dntime_skew_percent	integer	语句在数据库实例间的执行时间倾斜率。
min_cpu_time	bigint	语句在数据库实例上的最小 CPU 时间，单位 ms。
max_cpu_time	bigint	语句在数据库实例上的最大 CPU 时间，单位 ms。
total_cpu_time	bigint	语句在数据库实例上的 CPU 总时间，单位 ms。
cpu_skew_percent	integer	语句在数据库实例间的 CPU 时间倾斜率。
min_peak_iops	integer	语句在数据库实例上的每秒最小 IO 峰值（列存单位是次/s，行存单位是万次/s）。
max_peak_iops	integer	语句在数据库实例上的每秒最大 IO 峰值（列存单位是次/s，行存单位是万次/s）。
average_peak_iops	integer	语句在数据库实例上的每秒平均 IO 峰值（列存单位是次/s，行存单位是万次/s）。
iops_skew_percent	integer	语句在数据库实例间的 IO 倾斜率。

名称	类型	描述
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数据库实例。
cpu_top1_node_name	text	cpu 使用率第 1 的节点名称。
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。

名称	类型	描述
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。

18.3.33GS_WLM_SESSION_INFO

GS_WLM_SESSION_INFO 视图显示数据库实例执行作业结束后的负载管理记录。查询该视图需要 sysadmin 权限。

具体的字段请参考表 GS_WLM_SESSION_HISTORY 字段中的信息。

18.3.34GS_WLM_SESSION_INFO_ALL

GS_WLM_SESSION_INFO_ALL 视图显示在数据库实例上执行作业结束后的负载管理记录。查询该视图需要 sysadmin 或者 monitor admin 权限。

名称	类型	描述
userid	oid	用户 OID。

名称	类型	描述
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。
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。
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）。

名称	类型	描述
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）。
is_slow_query	bigint	是否是慢 SQL 记录。

18.3.35GS_WLM_SESSION_STATISTICS

GS_WLM_SESSION_STATISTICS 视图显示当前用户在数据库实例上正在执行的作业的负载管理记录。查询该视图需要 sysadmin 权限。

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	语句执行的数据库实例名称。
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 Uni

名称	类型	描述
		x 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	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	语句在数据库实例上的最小内存峰值，单位 MB。
max_peak_memory	integer	语句在数据库实例上的最大内存峰值，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	语句在数据库实例间的内存使用倾斜率。
spill_info	text	语句在数据库实例上的下盘信息： None：数据库实例均未下盘。

名称	类型	描述
		All：数据库实例均下盘。 [a:b]：数量为 b 个数据库实例中有 a 个数据库实例下盘。
min_spill_size	integer	若发生下盘，数据库实例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，数据库实例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，数据库实例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，数据库实例间下盘倾斜率。
min_dn_time	bigint	语句在数据库实例上的最小执行时间，单位 ms。
max_dn_time	bigint	语句在数据库实例上的最大执行时间，单位 ms。
average_dn_time	bigint	语句在数据库实例上的平均执行时间，单位 ms。
dntime_skew_percent	integer	语句在数据库实例间的执行时间倾斜率。
min_cpu_time	bigint	语句在数据库实例上的最小 CPU 时间，单位 ms。
max_cpu_time	bigint	语句在数据库实例上的最大 CPU 时间，单位 ms。
total_cpu_time	bigint	语句在数据库实例上的 CPU 总时间，单位 ms。
cpu_skew_percent	integer	语句在数据库实例间的 CPU 时间倾斜率。
min_peak_iops	integer	语句在数据库实例上的每秒最小 IO 峰值（列存单位是次/s，行存单位是万次/s）。
max_peak_iops	integer	语句在数据库实例上的每秒最大 IO 峰值（列存单位是次/s，行存单位是万次/s）。
average_peak_iops	integer	语句在数据库实例上的每秒平均 IO 峰值（列存单位是次/s，行存单位是万次/s）。
iops_skew_percent	integer	语句在数据库实例间的 IO 倾斜率。

名称	类型	描述
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数据库实例。
top_cpu_dn	text	cpu 使用量 topN 信息。
top_mem_dn	text	内存使用量 topN 信息。

18.3.36GS_WLM_USER_INFO

用户统计信息视图。

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。

名称	类型	描述
childlist	text	子用户列表。

18.3.37GS_STAT_SESSION_CU

GS_STAT_SESSION_CU 视图查询 GBase 8c 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。GBase 8c 重启后，统计数据也会清零。

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。

18.3.38GS_WRITE_TERM_LOG

写入一条日志记录 DN 当前的 term 值。备 DN 返回 false，主 DN 写入成功后返回 true。

18.3.39MPP_TABLES

MPP_TABLES 视图显示信息如下。

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。

18.3.40PG_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。

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。

18.3.41PG_AVAILABLE_EXTENSIONS

PG_AVAILABLE_EXTENSIONS 视图显示数据库中某些特性的扩展信息。

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。

18.3.42PG_CURSORS

PG_CURSORS 视图列出了当前可用的游标。

名称	类型	描述
name	text	游标名。

名称	类型	描述
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。

18.3.43PG_COMM_DELAY

PG_COMM_DELAY 视图展示单个节点的通信库时延状态。

名称	类型	描述
node_name	text	节点名称。
remote_name	text	连接对端节点名称。
remote_host	text	连接对端 IP 地址。
stream_num	integer	当前物理连接使用的 stream 逻辑连接数量。
min_delay	integer	当前物理连接一分钟内探测到的最小时延，单位微秒。 说明： 负数结果无效，请重新等待时延状态更新后再执行。
average	integer	当前物理连接一分钟内探测时延的平均值，单位微秒。
max_delay	integer	当前物理连接一分钟内探测到的最大时延，单位微秒。

18.3.44PG_COMM_RECV_STREAM

PG_COMM_RECV_STREAM 视图展示节点上所有的通信库接收流状态。

名称	类型	描述
node_name	text	节点名称。
local_tid	bigint	使用此通信流的线程 ID。
remote_name	text	连接对端节点名称。
remote_tid	bigint	连接对端线程 ID。
idx	integer	通信对端 DN 在本 DN 内的标识编号。
sid	integer	通信流在物理连接中的标识编号。
tcp_sock	integer	通信流所使用的 tcp 通信 socket。
state	text	通信流当前的状态。 UNKNOWN：表示当前逻辑连接状态未知。 READY：表示逻辑连接已就绪。 RUN：表示逻辑连接发送报文正常。 HOLD：表示逻辑连接发送报文等待中。 CLOSED：表示关闭逻辑连接。 TO_CLOSED：表示将会关闭逻辑连接。
query_id	bigint	通信流对应的 debug_query_id 编号。
pn_id	integer	通信流所执行查询的 plan_node_id 编号。
send_smp	integer	通信流所执行查询 send 端的 smpid 编号。
recv_smp	integer	通信流所执行查询 recv 端的 smpid 编号。
recv_bytes	bigint	通信流接收的数据总量，单位 Byte。
time	bigint	通信流当前生命周期使用时长，单位 ms。
speed	bigint	通信流的平均接收速率，单位 Byte/s。
quota	bigint	通信流当前的通信配额值，单位 Byte。
buff_usize	bigint	通信流当前缓存的数据大小，单位 Byte。

18.3.45PG_COMM_SEND_STREAM

PG_COMM_SEND_STREAM 视图展示节点上所有的通信库发送流状态。

名称	类型	描述
node_name	text	节点名称。
local_tid	bigint	使用此通信流的线程 ID。
remote_name	text	连接对端节点名称。
remote_tid	bigint	连接对端线程 ID。
idx	integer	通信对端 DN 在本 DN 内的标识编号。
sid	integer	通信流在物理连接中的标识编号。
tcp_sock	integer	通信流所使用的 tcp 通信 socket。
state	text	通信流当前的状态。 UNKNOWN：表示当前逻辑连接状态未知。 READY：表示逻辑连接已就绪。 RUN：表示逻辑连接发送报文正常。 HOLD：表示逻辑连接发送报文等待中。 CLOSED：表示关闭逻辑连接。 TO_CLOSED：表示将会关闭逻辑连接。
query_id	bigint	通信流对应的 debug_query_id 编号。
pn_id	integer	通信流所执行查询的 plan_node_id 编号。
send_smp	integer	通信流所执行查询 send 端的 smpid 编号。
recv_smp	integer	通信流所执行查询 recv 端的 smpid 编号。
send_bytes	bigint	通信流发送的数据总量，单位 Byte。
time	bigint	通信流当前生命周期使用时长，单位 ms。
speed	bigint	通信流的平均发送速率，单位 Byte/s。
quota	bigint	通信流当前的通信配额值，单位 Byte。
wait_quota	bigint	通信流等待 quota 值产生的额外时间开销，单位 ms。

18.3.46PG_COMM_STATUS

PG_COMM_STATUS 视图展示节点的通信库状态。

名称	类型	描述
node_name	text	节点名称。
rxpck_rate	integer	节点通信库接收速率，单位 Byte/s。
txpck_rate	integer	节点通信库发送速率，单位 Byte/s。
rxkbyte_rate	bigint	bigint 节点通信库接收速率，单位 KByte/s。
txkbyte_rate	bigint	bigint 节点通信库发送速率，单位 KByte/s。
buffer	bigint	cmailbox 的 buffer 大小。
memkbyte_libcomm	bigint	libcomm 进程通信内存大小，单位 Byte。
memkbyte_libpq	bigint	libpq 进程通信内存大小，单位 Byte。
used_pm	integer	postmaster 线程实时使用率。
used_sflow	integer	gs_sender_flow_controller 线程实时使用率。
used_rflow	integer	gs_receiver_flow_controller 线程实时使用率。
used_rloop	integer	多个 gs_receivers_loop 线程中高的实时使用率。
stream	integer	当前使用的逻辑连接总数。

18.3.47PG_CONTROL_GROUP_CONFIG

PG_CONTROL_GROUP_CONFIG 视图存储系统的控制组配置信息。查询该视图需要 sysadmin 权限。

名称	类型	描述
pg_control_group_config	text	控制组的配置信息。

18.3.48PG_EXT_STATS

PG_EXT_STATS 视图提供对存储在 PG_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。

名称	类型	引用	描述

名称	类型	引用	描述
schemaname	name	PG_NAMESPACE.nspname	包含表的模式名。
tablename	name	PG_CLASS.relname	表名。
attname	int2vector	PG_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。
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。 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； 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。

名称	类型	引用	描述
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。
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。

18.3.49PG_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。

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。

18.3.50PG_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。

名称	类型	描述
pid	bigint	当前 sender 的线程 ID。
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	texts	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。

18.3.51PG_GROUP

PG_GROUP 视图查看数据库认证角色及角色之间的成员关系。

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。

18.3.52PG_GTT_RELSTATS

PG_GTT_RELSTATS 视图查看当前会话所有全局临时表基本信息，调用 pg_get_gtt_relstats 函数。

名称	类型	描述
schemaname	name	schema 名称。
tablename	name	全局临时表名称。
relfilenode	oid	文件对象的 ID。

名称	类型	描述
relpages	integer	全局临时表的磁盘页面数。
reltuples	real	全局临时表的记录数。
relallvisible	integer	被标识为全可见的页面数。
relfrozenxid	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。
relminmxid	xid	预留接口，暂未启用。

18.3.53PG_GTT_STATS

PG_GTT_STATS 视图查看当前会话所有全局临时表单列统计信息，调用 pg_get_gtt_statistics 函数。

名称	类型	描述
schemaname	name	schema 名称。
tablename	name	全局临时表名称。
attname	name	属性名称。
inherited	boolean	是否统计有继承关系的对象。
null_frac	real	该字段中为 NULL 的记录的比率。
avg_width	integer	非 NULL 记录的平均存储宽度，以字节计算。
n_distinct	real	标识全局统计信息中字段里唯一的非 NULL 数据值的数目。
most_common_vals	text[]	高频值列表，按照出现的频率排序。
most_common_freqs	real[]	高频值的频率。
histogram_bounds	text[]	等频直方图描述列中的数据分布（不包含高频值）。
correlation	real	相关系数。
most_common_elems	text[]	类型高频值列表，用于数组类型或一些其他类型。

名称	类型	描述
most_common_elem_freqs	real[]	类型高频值的频率。
elem_count_histogram	real[]	数组类型直方图。

18.3.54PG_GTT_ATTACHED_PIDS

PG_GTT_ATTACHED_PIDS 视图查看哪些会话正在使用全局临时表，调用 pg_get_attached_pid 函数。

名称	类型	描述
schemaname	name	schema 名称。
tablename	name	全局临时表名称。
relid	oid	全局临时表的 oid。
pids	bigint[]	线程 pid 列表。

18.3.55PG_INDEXES

PG_INDEXES 视图提供对数据库中每个索引的有用信息的访问。

名称	类型	引用	描述
schemaname	name	PG_NAMESPACE.nspname	包含表和索引的模式名称。
tablename	name	PG_CLASS.relname	此索引所服务的表的名称。
indexname	name	PG_CLASS.relname	索引的名称。
tablespace	name	PG_TABLESPACE.nspname	包含索引的表空间名称。
indexdef	text	-	索引定义（一个重建的 C

名称	类型	引用	描述
			REATE INDEX 命令）。

18.3.56PG_LOCKS

PG_LOCKS 视图存储各打开事务所持有的锁信息。

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	PG_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	PG_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	-	子表对应的 bucket number。如果目标不是表的话，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	PG_CLASS.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

名称	类型	引用	描述
			其他对象类型，这个字段是 0；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。
locktag	text	-	会话等待锁信息，可通过 locktag_decode()函数解析。
global_sessionid	text		全局会话 ID。

18.3.57PG_NODE_ENV

PG_NODE_ENV 视图提供获取当前节点的环境变量信息。

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。

名称	类型	描述
datapath	text	当前节点的数据目录。
log_directory	text	当前节点的日志目录。

18.3.58PG_OS_THREADS

PG_OS_THREADS 视图提供当前节点下所有线程的状态信息。

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。

18.3.59PG_PREPARED_STATEMENTS

PG_PREPARED_STATEMENTS 视图显示当前会话所有可用的预备语句。

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。
from_sql	Boolean	如果该预备语句是通过 PREPARE 语句创建的则为

名称	类型	描述
		true。 如果是通过前/后端协议创建的则为 false。

18.3.60PG_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。

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	PG_AUTHID.rolname	执行该事务的用户的名称。
database	name	PG_DATABASE.datname	执行该事务所在的数据库名。

18.3.61PG_PUBLICATION_TABLES

视图 PG_PUBLICATION_TABLES 提供 publication 与其所发布的表之间的映射信息。和底层的系统表 pg_publication_rel 不同，这个视图展开了定义为 FOR ALL TABLES 的 publication，这样对这类 publication 来说，每一个合格的表都有一行。

名称	类型	描述
pubname	name	发布的名称。
schemaname	name	包含表的模式名称。
tablename	name	表的名称。

18.3.62PG_REPLICATION_ORIGIN_STATUS

获取复制源的复制状态。

名称	类型	描述

名称	类型	描述
local_id	oid	复制源 ID。
external_id	text	复制源名称。
remote_lsn	LSN	复制源的 lsn 位置。
local_lsn	LSN	本地的 lsn 位置。

18.3.63PG_REPLICATION_SLOTS

PG_REPLICATION_SLOTS 视图查看复制节点的信息。

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。
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。

18.3.64PG_RLSPOLICIES

PG_RLSPOLICIES 视图提供查询行级访问控制策略。

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。

名称	类型	描述
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。

18.3.65PG_ROLES

PG_ROLES 视图提供访问数据库角色的相关信息，初始化用户和具有 sysadmin 属性或 createrole 属性的用户可以查看全部角色的信息，其他用户只能查看自己的信息。

名称	类型	引用	描述
rolname	name	-	角色名称。
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。
rolreplication	Boolean	-	该角色是否可以复制。
rolauditadmin	Boolean	-	该角色是否为审计管理员。

名称	类型	引用	描述
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	PG_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，单位为 KB。
rolspillspace	text	-	用户算子落盘空间限额，单位为 KB。
rolconfig	text[]	-	运行时配置变量的会话缺省。
oid	oid	PG_AUTHID.oid	角色的 ID。
roluseft	Boolean	PG_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。
rolmonitoradmin	Boolean	-	该角色是否为监控管理员。

名称	类型	引用	描述
roloperatoradmin	Boolean	-	该角色是否为运维管理员。
rolpolicyadmin	Boolean	-	该角色是否为安全策略管理员。

18.3.66PG_RULES

PG_RULES 视图提供对查询重写规则的有用信息访问的接口。

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）。

18.3.67PG_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。

名称	类型	描述
handle	integer	事务对应的事务管理器中的槽位句柄，该值恒为-1。
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。 t（true）：表示是。 f（false）：表示否。
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

名称	类型	描述
		态，值为 0。
pid	bigint	事务对应的线程 id。
next_xid	xid	其余节点发送给当前节点的事务 id，该值恒为 0。

18.3.68PG_SECLABELS

PG_SECLABELS 视图提供关于安全标签的信息。

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	PG_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 0。
objtype	text	-	这个标签出现的对象的类型，文本格式。
objnamespace	oid	PG_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	PG_SECLABEL.provider	与这个标签相关的标签提供者。
label	text	PG_SECLABEL.label	适用于这个对象的安全标签。

18.3.69PG_TOTAL_MEMORY_DETAIL

PG_TOTAL_MEMORY_DETAIL 视图显示某个数据库节点内存使用情况。

名称	类型	描述
nodename	text	节点名称。

名称	类型	描述
memorytype	text	内存的名称。
memorymbytes	integer	内存使用的大小，单位为 MB。

18.3.70PG_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。

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。
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。
read_kbytes	bigint	读磁盘数据量，单位 KB。
write_kbytes	bigint	写磁盘数据量，单位 KB。

名称	类型	描述
read_counts	bigint	读磁盘次数。
write_counts	bigint	写磁盘次数。
read_speed	double precision	读磁盘速率，单位 B/ms。
write_speed	double precision	写磁盘速率，单位 B/ms。

18.3.71PG_SETTINGS

PG_SETTINGS 视图显示数据库运行时参数的相关信息。

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 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。

名称	类型	描述
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。

18.3.72PG_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。PG_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。

名称	类型	引用	描述
usename	name	PG_AUTHID.rolname	用户名。
usesysid	oid	PG_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 PG_AUTHID 获取加密的口令是如何存储的信息。
valbegin	timestamp wit	-	帐户的有效开始时间；如果没有设置

名称	类型	引用	描述
	h time zone		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。

18.3.73PG_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。该视图记录的统计信息更新时间间隔由参数 autovacuum_naptime 设置。

名称	类型	引用	描述
schemaname	name	PG_NAMESPACE.nspname	包含表的模式名。
tablename	name	PG_CLASS.relname	表名。
attname	name	PG_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。
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。

名称	类型	引用	描述
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。 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； 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 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。

名称	类型	引用	描述
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。

18.3.74PG_STAT_ACTIVITY

PG_STAT_ACTIVITY 视图显示和当前用户查询相关的信息，字段保存的是上一次执行的信息。

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

名称	类型	描述
		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	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。

名称	类型	描述
		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 omm 的 state 信息为空： SELECT datname, usename, usesysid, state,pid FROM pg_stat_activity; datname | usename | usesysid | state | pid ———-+———+———-+———+——— postgres | gbase | 10 | | 139968752121616 postgres | gbase | 10 | | 139968903116560 db_tpcc | judy | 16398 | active | 139968391403280 postgres | gbase | 10 | | 139968643069712 postgres | gbase | 10 | | 139968680818448 postgres | 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）。
unique_sql_id	bigint	语句的 unique sql id。

名称	类型	描述
trace_id	text	驱动传入的 trace id，与应用的一次请求相关联。

18.3.75PG_STAT_ACTIVITY_NG

PG_STAT_ACTIVITY_NG 视图显示在当前用户所属的逻辑数据库实例下，所有查询的相关信息。

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 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 wi	启动当前事务的时间，如果没有事务是活

名称	类型	描述
	th time zone	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。否则为 false。
enqueue	text	语句当前排队状态。可能值是： waiting in queue：表示语句在排队中。 空：表示语句正在运行。
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 omm 的 state 信息为空： SELECT datname, usename, usesysid, state,pid FROM pg_stat_activity_ng; datname | usename | usesysid | state | pid ————-+————+————-+———+—————

名称	类型	描述
		postgres | gbase | 10 | | 139968752121616 postgres | gbase | 10 | | 139968903116560 db_tpcds | judy | 16398 | active | 139968391403280 postgres | gbase | 10 | | 139968643069712 postgres | gbase | 10 | | 139968680818448 postgres | 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。
node_group	text	语句所属用户对应的逻辑数据库实例。

18.3.76PG_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。

名称	类型	描述
relid	oid	这个索引的表的 OID。

名称	类型	描述
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。

18.3.77PG_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。

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

名称	类型	描述
		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	最后一次清理该表的时间。
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	这个表被清理的次数。
autovacuum_count	bigint	这个表被 autovacuum 清理的次数。
analyze_count	bigint	这个表被分析的次数。
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。

18.3.78PG_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。

名称	类型	描述
nodename	text	节点名。
databaseid	integer	数据库 OID。

名称	类型	描述
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。取值如下： 0，数据主文件。 1，FSM 文件。 2，VM 文件。 3，BCM 文件。 大于 4，列存表每个字段的数据文件。
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。

18.3.79PG_STAT_BGWRITER

PG_STAT_BGWRITER 视图显示关于后端写进程活动的统计信息。

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。

名称	类型	描述
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。

18.3.80PG_STAT_DATABASE

PG_STAT_DATABASE 视图将包含 GBase 8c 中每个数据库的数据库统计信息。

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 GBase 8c 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 PG_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。

18.3.81PG_STAT_DATABASE_CONFLICTS

PG_STAT_DATABASE_CONFLICTS 视图显示数据库冲突状态的统计信息。

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。

名称	类型	描述
confl_bufferpin	bigint	冲突的缓冲区数目。
confl_deadlock	bigint	冲突的死锁数目。

18.3.82 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。

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。

18.3.83 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。

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。

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。

18.3.84PG_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。

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）。
last_autovacuum	timestamp with time zone	上次被 autovacuum 守护进程清理的

名称	类型	描述
		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 守护进程分析的次数。
last_data_changed	timestamp with time zone	这个表数据最近修改时间。

18.3.85PG_STAT_REPLICATION

PG_STAT_REPLICATION 视图用于描述日志同步状态信息，例如发起端发送日志位置，接收端接收日志位置等。

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。

名称	类型	描述
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。 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。

18.3.86PG_STAT_SUBSCRIPTION

获取订阅的详细同步信息。

名称	类型	描述
subid	oid	订阅的 OID。
subname	name	订阅的名称。
pid	thread_id	后台 apply 线程的 thread id。
received_lsn	LSN	从发布端接收到的最近的 lsn。

名称	类型	描述
last_msg_send_time	timestamp	最近发布端发送消息的时间。
last_msg_receipt_time	timestamp	最新订阅端收到消息的时间。
latest_end_lsn	LSN	最近一次收到保活消息时发布端的 lsn。
latest_end_time	timestamp	最近一次收到保活消息的时间。

18.3.87PG_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。

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。

18.3.88PG_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。

名称	类型	描述

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时间（不计算 VACUUM FULL）。
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。

名称	类型	描述
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。

18.3.89PG_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。

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）。

18.3.90PG_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。

名称	类型	描述

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）。

18.3.91PG_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。

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。

18.3.92PG_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。

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）。

18.3.93PG_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。

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。

名称	类型	描述
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。

18.3.94PG_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。

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。

18.3.95PG_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。

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）。

名称	类型	描述
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）。

18.3.96PG_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。

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。

18.3.97PG_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。

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。

18.3.98PG_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。

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。

18.3.99PG_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。

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。

18.3.100 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。

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）。

18.3.101 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。

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。

名称	类型	描述
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）。

18.3.102 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。

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。
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 15-15-2。
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。
locktag	text	当前线程正在等待锁的信息。
lockmode	text	当前线程正等待获取的锁模式。包含表级锁、行级锁、页级

名称	类型	描述
		锁下的各模式。
block_sessionid	bigint	阻塞当前线程获取锁的会话标识。
global_sessionid	text	全局会话 ID。

wait_status 列的等待状态有以下状态。

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	等待接收与某节点的连接上的数据，当前正在等待

wait_status 值	含义
N, [phase]	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。
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。

wait_status 值	含义
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。
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 - [fetch tuple | write file]	当前是 Sort 算子做排序，fetch tuple 表示 Sort 算子正在获取 tuple，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。）

轻量级锁等待事件列表

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。

wait_event 类型	类型描述
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。
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。

wait_event 类型	类型描述
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。
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。
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。

wait_event 类型	类型描述
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。
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 GBase 8c 连接信息的全局哈希表。
RPNumberLock	用于加速 GBase 8c 的数据库节点对正在执行计划的任务线程的计数。
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。

wait_event 类型	类型描述
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。
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。

wait_event 类型	类型描述
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。
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。

wait_event 类型	类型描述
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。

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。
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。

wait_event 类型	类型描述
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。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。

wait_event 类型	类型描述
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。
WALWrite	写入 WAL 文件。
WALBufferAccess	WAL Buffer 访问（出于性能考虑，内核代码里只统计访问次数，未统计其访问耗时）。
WALBufferFull	WAL Buffer 满时，写 wal 文件相关的处理。

wait_event 类型	类型描述
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。

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。

wait_event 类型	类型描述
cstore_freespace	对列存空闲空间加锁。
userlock	加用户锁。
advisory	加 advisory 锁。

18.3.103 PG_TABLES

PG_TABLES 视图提供了对数据库中每个表访问的有用信息。

名称	类型	引用	描述
schemaname	name	PG_NAMESPACE.nspname	包含表的模式名。
tablename	name	PG_CLASS.relname	表名。
tableowner	name	pg_get_userbyid(PG_CLASS.relowner)	表的所有者。
tablespace	name	PG_TABLESPACE.spcname	包含表的表空间，默认为 NULL。
hasindexes	Boolean	PG_CLASS.relhasindex	如果表上有索引（或者最近拥有）则为 TRUE，否则为 FALSE。
hasrules	Boolean	PG_CLASS.relhasruls	如果表上有规则，则为 TRUE，否则为 FALSE。
hastriggers	Boolean	PG_CLASS.RELHASTRIGGERS	如果表上有触发器，则为 TRUE，否则为 FALSE。
tablecreator	name	pg_get_userbyid(po.creator)	创建表的名称。
created	timestamp with time zone	pg_object.ctime	对象的创建时间。
last_ddl_time	timestamp wit	pg_object.mtime	对象的最后修改时间。

名称	类型	引用	描述
	h time zone		

18.3.104 PG_TDE_INFO

PG_TDE_INFO 视图提供了 GBase 8c 加密信息。

名称	类型	描述
is_encrypt	Boolean	是否加密 GBase 8c。 f：非加密 GBase 8c。 t：加密 GBase 8c。
g_tde_algo	text	加密算法。 SM4-CTR-128。 AES-CTR-128。
remain	text	保留字段。

18.3.105 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。

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。

18.3.106 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。

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）。

名称	类型	描述
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）。 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）。

18.3.107 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。

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 z	帐户的有效开始时间；如果没有设置有效

名称	类型	描述
	one	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 GBase 8c 名称，如果该用户没有管理逻辑 GBase 8c，则该字段为空。

18.3.108 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 PG_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。

名称	类型	引用	描述
umid	oid	PG_USER_MAPPING.oid	用户映射的 OID。
srvid	oid	PG_FOREIGN_SERVER.oid	包含这个映射的外部服务器的 OID。
srvname	name	PG_FOREIGN_SERVER.srvname	外部服务器的名称。

名称	类型	引用	描述
umuser	oid	PG_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，则为用户映射指定选项，使用“keyword=value”字符串，否则为 null。

18.3.109 PG_VIEWS

PG_VIEWS 视图提供访问数据库中每个视图的有用信息。

名称	类型	引用	描述
schemaname	name	PG_NAMESPACE.nspname	包含视图的模式名。
viewname	name	PG_CLASS.relname	视图名。
viewowner	name	PG_AUTHID.Erolname	视图的所有者。
definition	text	-	视图的定义。

18.3.110 PG_TIMEZONE_ABBREVS

PG_TIMEZONE_ABBREVS 视图提供了显示了所有可用的时区信息。

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。

18.3.111 PG_VARIABLE_INFO

PG_VARIABLE_INFO 视图用于查询 GBase 8c 中当前节点的 xid、oid 的状态。

名称	类型	描述

名称	类型	描述
node_name	text	节点名称。
next_oid	oid	该节点下一次生成的 oid。
next_xid	xid	该节点下一次生成的事务号。
oldest_xid	xid	该节点最老的事务号。
xid_vac_limit	xid	强制 autovacuum 的临界点。
oldest_xid_db	oid	该节点 datafrozenxid 最小的数据库 oid。
last_extend_csn_logpage	xid	最后一次扩展 csnlog 的页面号。
start_extend_csn_logpage	xid	csnlog 扩展的起始页面号。
next_commit_seqno	xid	该节点下次生成的 csn 号。
latest_completed_xid	xid	该节点提交或者回滚后节点上的最新事务号。
startup_max_xid	xid	该节点关机前的最后一个事务号。

18.3.112 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。查询该视图需要 sysadmin 权限或者 monitor admin 权限。

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。

名称	类型	描述
query_id	bigint	作业 id。
mincurriops	integer	该作业当前 io 在数据库实例中的最小值。
maxcurriops	integer	该作业当前 io 在数据库实例中的最大值。
minpeakiops	integer	在作业运行时，作业 io 峰值中，数据库实例的最小值。

名称	类型	描述
maxpeakiops	integer	在作业运行时，作业 io 峰值中，数据库实例的最大值。
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。
curr_io_limits	integer	使用 io_priority 管控 io 时的实时 io_limits 值。

18.3.113 PG_SESSION_WLMSTAT

PG_SESSION_WLMSTAT 视图显示当前用户执行作业正在运行时的负载管理相关信息。查询该视图需要 sysadmin 权限。

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。

名称	类型	描述
		Unknown：未知。
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实例上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实例上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。（当 enqueue 字段为 StoredProc 或 Transaction 时，仅代表语句中的部分作业已经执行完毕，该状态会持续到该语句完全执行完毕。） 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。

18.3.114 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。查询该视图需要 sysadmin 权限。

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实例上 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实例上 CPU 使用的倾斜率。
control_group	text	语句执行异常处理时所使用的 Cgroups。
status	text	语句执行异常处理后的状态，包括： 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。

18.3.115 PGXC_PREPARED_XACTS

PGXC_PREPARED_XACTS 视图显示当前处于 prepared 阶段的两阶段事务。只有 system admin 和 monitor admin 用户有权限查看。

名称	类型	描述
pgxc_prepared_xact	text	查看当前处于 prepared 阶段的两阶段事务。

18.3.116 PGXC_THREAD_WAIT_STATUS

通过 CN 节点查看 PGXC_THREAD_WAIT_STATUS 视图，可以查看集群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，从而更容易定位进程停止响应问题以及类似现象的原因。

PGXC_THREAD_WAIT_STATUS 视图和 PG_THREAD_WAIT_STATUS 视图列定义完全相同，这是由于 PGXC_THREAD_WAIT_STATUS 视图本质是到集群中各个节点上查询 PG_THREAD_WAIT_STATUS 视图汇总的结果。

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。
tid	bigint	当前线程的线程号。
lwtid	integer	当前线程的轻量级线程号。
ptid	integer	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 PG_THREAD_WAIT_STATUS 中等待状态列表。

名称	类型	描述
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。

例如：

在 coordinator1 执行一条语句之后长时间没有响应。可以创建另外一个连接到 coordinator1 上，查询 coordinator1 上的线程状态。

```
select * from pg_thread_wait_status where query_id > 0;
 node_name    | db_name   | thread_name   | query_id |          tid          | lwtid | ptid   |
tlevel | smpid |       wait_status     |     wait_event
--------------+----------+--------------+----------+-----------------+-------+-------+--------+-------+---------
-------------
coordinator1 | postgres | gsql              | 20971544 | 140274089064208 | 22579 |          |
0 |        0 | wait node: datanode4 |(1 rows)
```

此外，可以查看该语句在全局范围内各个节点上的工作情况。如下所示，每个 DN 上都没有在等待的阻塞资源，因为读取的数据太多而执行较慢。

```
select * from pgxc_thread_wait_status where query_id=20971544;
 node_name    | db_name   | thread_name   | query_id |          tid          | lwtid | ptid   |
tlevel | smpid |       wait_status     |   wait_event
--------------+----------+--------------+----------+-----------------+-------+-------+--------+-------+---------
-------------
datanode1      | postgres | coordinator1 | 20971544 | 139902867994384 | 22735 |          |
0 |        0 | wait node: datanode3 |
datanode1      | postgres | coordinator1 | 20971544 | 139902838634256 | 22970 | 22735 |
5 |        0 | synchronize quit        |
datanode1      | postgres | coordinator1 | 20971544 | 139902607947536 | 22972 | 22735 |
5 |        1 | synchronize quit        |
datanode2      | postgres | coordinator1 | 20971544 | 140632156796688 | 22736 |          |
0 |        0 | wait node: datanode3 |
datanode2      | postgres | coordinator1 | 20971544 | 140632030967568 | 22974 | 22736 |
5 |        0 | synchronize quit        |
```

```
        datanode2       | postgres | coordinator1 | 20971544 | 140632081299216 | 22975 | 22736 |
5 |       1 | synchronize quit        |
        datanode3       | postgres | coordinator1 | 20971544 | 140323627988752 | 22737 |           |
0 |       0 | wait node: datanode3 |
        datanode3       | postgres | coordinator1 | 20971544 | 140323523131152 | 22976 | 22737 |
5 |       0 | net flush data          |
        datanode3       | postgres | coordinator1 | 20971544 | 140323548296976 | 22978 | 22737 |
5 |       1 | net flush data
        datanode4       | postgres | coordinator1 | 20971544 | 140103024375568 | 22738 |           |
0 |       0 | wait node: datanode3
        datanode4       | postgres | coordinator1 | 20971544 | 140102919517968 | 22979 | 22738 |
5 |       0 | synchronize quit        |
        datanode4       | postgres | coordinator1 | 20971544 | 140102969849616 | 22980 | 22738 |
5 |       1 | synchronize quit        |
        coordinator1 | postgres | gsql             | 20971544 | 140274089064208 | 22579 |           |
0 |       0 | wait node: datanode4    |(13 rows)
```

18.3.117 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。

名称	类型	描述
statement_id	varchar2(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	varchar2(30)	计划中算子的操作描述。
options	varchar2(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	varchar2(30)	对象类型。

名称	类型	描述
object_owner	name	对象所属 schema，来自于用户定义。
projection	varchar2(4000)	操作输出的列信息。

说明：

object_type 取值范围为 PG_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。

18.3.118 GS_DB_PRIVILEGES

GS_DB_PRIVILEGES 系统视图记录 ANY 权限的授予情况，每条记录对应一条授权信息。

名称	类型	描述
rolename	name	用户名。
privilege_type	text	用户拥有的 ANY 权限，取值参考《GBase 8c V5_3.0.1_SQL 参考手册》中 GRANT 章节 ANY 权限列表。
admin_option	boolean	是否具有 privilege_type 列记录的 ANY 权限的再授权权限。 yes：表示具有。 no：表示不具有。

18.3.119 GS_GSC_MEMORY_DETAIL

GS_GSC_MEMORY_DETAIL 视图描述当前节点当前进程的全局 SysCache 的内存占用情况，仅在开启 GSC 的模式下有数据。需要注意的是，这个查询由于是以数据库内存上下文分隔的，因此会缺少一部分内存的统计，缺失的内存统计对应的内存上下文名称为 GlobalSysDBCache。

名称	类型	描述
db_id	integer	数据库 id。
totalsize	bigint	共享内存总大小，单位 Byte。
freesize	bigint	共享内存剩余大小，单位 Byte。
usedsize	bigint	共享内存使用大小，单位 Byte。

18.3.120 GS_LSC_MEMORY_DETAIL

GS_LSC_MEMORY_DETAIL 视图统计所有的线程的本地 SysCache 内存使用情况，以 MemoryContext 节点来统计，仅在开启 GSC 的模式下有数据。

名称	类型	描述
threadid	text	线程启动时间+线程标识（字符串信息为 timestamp.sessionid）。
tid	bigint	线程标识。
thrdtype	text	线程类型。可以是系统内存在的任何线程类型，如 postgresql、wlmmonitor 等。
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。
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。

名称	类型	描述
usedsize	bigint	当前内存上下文中已使用的内存总数，单位 Byte。

19 Schema

GBase 8c 的 Schema 如下表所示。

Schema 名称	描述
blockchain	用于存储账本数据库特性中创建防篡改表时自动创建的用户历史表。
cstore	该模式用于储存列存表相关的辅助表如 cudesc 或者 delta 表。
db4ai	用于管理 AI 训练中不同版本的数据信息。
dbe_perf	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，默认监控管理员具有模式 dbe_perf 的权限，有权查看该模式下的视图和函数。
dbe_pldebugger	用于调试 plpgsql 函数及存储过程。
snapshot	用于管理 WDR snapshot 的相关的数据信息，默认初始化用户或监控管理员用户可以访问。
sqladvisor	用于分布列推荐，GBase 8c 不可用。
sys	用于提供系统信息视图接口。
pg_catalog	用于维护系统的 catalog 信息，包含系统表和所有内置数据类型、函数、操作符。
pg_toast	用于存储大对象（系统内部使用）。
public	公共模式，缺省时，创建的表（以及其它对象）自动放入该模式。
pkg_service	用于管理 package 服务相关信息。
pkg_util	用于管理 package 工具相关信息。

19.1 Information Schema

信息模式本身是一个名为 information_schema 的模式。这个模式自动存在于所有数据库中。信息模式由一组视图构成，它们包含定义在当前数据库中对象的信息。这个模式的拥有者是初始数据库用户，并且该用户自然地拥有这个模式上的所有特权，包括删除它的能力。

信息模式继承自开源 PGXC/PG，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/information-schema.html

https://www.postgresql.org/docs/9.2/information-schema.html

下面章节只显示未在上述链接内的视图信息。

19.1.1 _PG_FOREIGN_DATA_WRAPPERS

显示外部数据封装器的信息。该视图只有 sysadmin 权限可以查看。

名称	类型	描述
oid	oid	外部数据封装器的 oid。
fdwowner	oid	外部数据封装器的所有者的 oid。
fdwoptions	text[]	外部数据封装器指定选项，使用“keyword=value”格式的字符串。
foreign_data_wrapper_catalog	information_schema.sql_identifier	外部封装器所在的数据库名称（永远为当前数据库）。
foreign_data_wrapper_name	information_schema.sql_identifier	外部数据封装器名称。
authorization_identifier	information_schema.sql_identifier	外部数据封装器所有者的角色名称。
foreign_data_wrapper_language	information_schema.character_data	外部数据封装器的实现语言。

19.1.2 _PG_FOREIGN_SERVERS

显示外部服务器的信息。该视图只有 sysadmin 权限可以查看。

名称	类型	描述
oid	oid	外部服务器的 oid。
srvoptions	text[]	外部服务器指定选项，使用“keyword=value”格式的字符串。
foreign_server_catalog	information_schema.sql_identifier	外部服务器所在 database 名称（永远为当前数据库）。
foreign_server_name	information_schema.sql_identifier	外部服务器名称。
foreign_data_wrapper_catalog	information_schema.sql_identifier	外部数据封装器所在 database 名称（永远为当前数据库）。
foreign_data_wrapper_name	information_schema.sql_identifier	外部数据封装器名称。
foreign_server_type	information_schema.character_data	外部服务器的类型。
foreign_server_version	information_schema.character_data	外部服务器的版本。
authorization_identifier	information_schema.sql_identifier	外部服务器的所有者的角色名称。

19.1.3 _PG_FOREIGN_TABLE_COLUMNS

显示外部表的列信息。该视图只有 sysadmin 权限可以查看。

名称	类型	描述
nspname	name	schema 名称。
relname	name	表名称。
attname	name	列名称。
attfdwoptions	text[]	外部数据封装器的属性选项，使用“keyword=value”格式的字符串。

19.1.4 _PG_FOREIGN_TABLES

存储所有的定义在本数据库的外部表信息。只显示当前用户有权访问的外部表信息。该视图只有 sysadmin 权限可以查看。

名称	类型	描述
foreign_table_catalog	information_schema.sql_identifier	外部表所在的数据库名称(永远是当前数据库）。
foreign_table_schema	name	外部表的 schema 名称。
foreign_table_name	name	外部表的名称。
ftoptions	text[]	外部表的可选项。
foreign_server_catalog	information_schema.sql_identifier	外部服务器所在的数据库名称(永远是当前数据库）。
foreign_server_name	information_schema.sql_identifier	外部服务器的名称。
authorization_identifier	information_schema.sql_identifier	所有者的角色名称。

19.1.5 _PG_USER_MAPPINGS

存储从本地用户到远程的映射。该视图只有 sysadmin 权限可以查看。

名称	类型	描述
oid	oid	从本地用户到远程的映射的 oid。
umoptions	text[]	用户映射指定选项，使用“keyword=value”格式的字符串。
umuser	oid	被映射的本地用户的 OID，如果用户映射是公共的则为 0。
authorization_identifier	information_schema.sql_identifier	本地用户角色名称。
foreign_server_catalog	information_schema.sql_identifier	外部服务器定义所在的 database 名称。

foreign_server_name	information_schema.sql_identifier	外部服务器名称。
srvowner	information_schema.sql_identifier	外部服务器所有者。

19.1.6 INFORMATION_SCHEMA_CATALOG_NAME

用来显示当前所在的 database 的名称。

名称	类型	描述
catalog_name	information_schema.sql_identifier	当前 database 的名称。

19.2 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。若是由旧版本升级而来，为保持权限的前向兼容，模式 dbe_perf 的权限与旧版本保持一致。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 GBase 8c 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。

19.2.1 OS

19.2.1.1 OS_RUNTIME

显示当前操作系统运行的状态信息。

名称	类型	描述
id	integer	编号。

名称	类型	描述
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。

19.2.1.2 GLOBAL_OS_RUNTIME

提供 GBase 8c 中所有正常节点下的操作系统运行状态信息。

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。

19.2.1.3 OS_THREADS

提供当前节点下所有线程的状态信息。

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。

19.2.1.4 GLOBAL_OS_THREADS

提供 GBase 8c 中所有正常节点下的线程状态信息。

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。

19.2.2 Instance

19.2.2.1 INSTANCE_TIME

提供当前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销。
- CPU_TIME：CPU 的时间花销。
- EXECUTION_TIME：执行器内的时间花销。
- PARSE_TIME：SQL 解析的时间花销。
- PLAN_TIME：生成 Plan 的时间花销。
- REWRITE_TIME：SQL 重写的时间花销。
- PL_EXECUTION_TIME：plpgsql（存储过程）执行的时间花销。
- PL_COMPILATION_TIME：plpgsql（存储过程）编译的时间花销。
- NET_SEND_TIME：网络上的时间花销。
- DATA_IO_TIME：IO 上的时间花销。

名称	类型	描述

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）。

19.2.2.2 GLOBAL_INSTANCE_TIME

提供 GBase 8c 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)。

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）。

19.2.3 Memory

19.2.3.1 MEMORY_NODE_DETAIL

显示某个数据库节点内存使用情况。

名称	类型	描述
nodename	text	节点名称。
memorytype	text	内存的名称。 ● max_process_memory：GBase 8c 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。

名称	类型	描述
		● 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。

19.2.3.2 GLOBAL_MEMORY_NODE_DETAIL

显示当前 GBase 8c 中所有正常节点下的内存使用情况。

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：GBase 8c 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。

名称	类型	描述
		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。

19.2.3.3 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。

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）。
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）。

19.2.3.4 GLOBAL_SHARED_MEMORY_DETAIL

查询 GBase 8c 中所有正常节点下的共享内存上下文的使用信息。

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）。

19.2.4 File

19.2.4.1 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。

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）。

名称	类型	描述
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）。

19.2.4.2 SUMMARY_FILE_IOSTAT

通过 GBase 8c 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。

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。
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）。

19.2.4.3 GLOBAL_FILE_IOSTAT

得到所有节点上的数据文件 IO 统计信息。

名称	类型	描述

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）。
maxiowtm	bigint	读写文件的最大时长（单位：微秒）。

19.2.4.4 FILE_REDO_IOSTAT

本节点 Redo（WAL）相关的统计信息。

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）。

名称	类型	描述
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）。

19.2.4.5 SUMMARY_FILE_REDO_IOSTAT

GBase 8c 内汇聚所有的 Redo（WAL）相关的统计信息。

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）。
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）。

19.2.4.6 GLOBAL_FILE_REDO_IOSTAT

得到 GBase 8c 内各节点的 Redo（WAL）相关统计信息。

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。

名称	类型	描述
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）。

19.2.4.7 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。

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。
phyblkwrt	bigint	写物理文件的块的数目。

19.2.4.8 GLOBAL_REL_IOSTAT

获取所有节点上的数据文件 IO 统计信息。

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。

19.2.4.9 SUMMARY_REL_IOSTAT

获取所有节点上的数据文件 IO 统计信息。

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。

19.2.5 Object

19.2.5.1 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。

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）。

名称	类型	描述
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）。
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。

19.2.5.2 SUMMARY_STAT_USER_TABLES

GBase 8c 内汇聚所有命名空间中用户自定义普通表的状态信息。

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。

名称	类型	描述
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。
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。

19.2.5.3 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。

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。
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）。

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。

19.2.5.4 STAT_USER_INDEXES

显示数据库中用户自定义普通表的索引状态信息。

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。

19.2.5.5 SUMMARY_STAT_USER_INDEXES

GBase 8c 内汇聚所有数据库中用户自定义普通表的索引状态信息。

名称	类型	描述
schemaname	name	索引中模式名。
relname	name	索引的表名。

名称	类型	描述
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。

19.2.5.6 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。

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。

19.2.5.7 STAT_SYS_TABLES

显示单节点内 pg_catalog、information_schema 以及 pg_toast 模式下的所有系统表的统计信息。

名称	类型	描述
relid	oid	表的 OID。
schemaname	name	该表的模式名。

名称	类型	描述
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）。
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_co	bigint	该表被 autovacuum 守护进程分析的次数。

名称	类型	描述
unt		

19.2.5.8 SUMMARY_STAT_SYS_TABLES

GBase 8c 内汇聚 pg_catalog、information_schema 以及 pg_toast 模式下的所有系统表的统计信息。

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。
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。

19.2.5.9 GLOBAL_STAT_SYS_TABLES

得到各节点 pg_catalog、information_schema 以及 pg_toast 模式下的所有系统表的统计信息。

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。

19.2.5.10 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。

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。

名称	类型	描述
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。

19.2.5.11 SUMMARY_STAT_SYS_INDEXES

GBase 8c 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。

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。

19.2.5.12 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。

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。

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	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。

19.2.5.13 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。

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）。

名称	类型	描述
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）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。

19.2.5.14 SUMMARY_STAT_ALL_TABLES

GBase 8c 内汇聚数据库中每个表的一行（包括 TOAST 表）的统计信息。

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)。
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。

19.2.5.15 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。

名称	类型	描述

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）。
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）。

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。

19.2.5.16 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。

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。

19.2.5.17 SUMMARY_STAT_ALL_INDEXES

将包含 GBase 8c 内汇聚数据库中的每个索引行，显示访问特定索引的统计。

名称	类型	描述
schemaname	name	索引中模式名。
relname	name	索引的表名。

名称	类型	描述
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。

19.2.5.18 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。

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。

19.2.5.19 STAT_DATABASE

视图将包含本节点中每个数据库的统计信息。

名称	类型	描述
datid	oid	数据库的 OID。
datname	name	此数据库的名称。

名称	类型	描述
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。
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 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。

名称	类型	描述
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。

19.2.5.20 SUMMARY_STAT_DATABASE

视图将包含数据库内汇聚的每个数据库的每一行，显示数据库统计。

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 GBase 8c 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 STAT_DATABASE_CONFLICTS 获取更多信息。

名称	类型	描述
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。

19.2.5.21 GLOBAL_STAT_DATABASE

视图将包含 GBase 8c 中各节点的每个数据库的每一行，显示数据库统计。

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。

名称	类型	描述
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 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。

19.2.5.22 STAT_DATABASE_CONFLICTS

显示当前节点数据库冲突状态的统计信息。

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。

19.2.5.23 SUMMARY_STAT_DATABASE_CONFLICTS

显示 GBase 8c 内汇聚的数据库冲突状态的统计信息。

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。

19.2.5.24 GLOBAL_STAT_DATABASE_CONFLICTS

显示每个节点的数据库冲突状态的统计信息。

名称	类型	描述
node_name	name	数据库进程名称。
datid	oid	数据库标识。

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。

19.2.5.25 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。

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_u	bigint	HOT 更新行数（比如没有更新所需的单独索引）。

名称	类型	描述
pd		

19.2.5.26 SUMMARY_STAT_XACT_ALL_TABLES

显示 GBase 8c 内汇聚的命名空间中所有普通表和 toast 表的事务状态信息。

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）。

19.2.5.27 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。

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。

名称	类型	描述
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）。

19.2.5.28 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。

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）。

19.2.5.29 SUMMARY_STAT_XACT_SYS_TABLES

显示 GBase 8c 内汇聚的命名空间中系统表的事务状态信息。

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）。

19.2.5.30 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。

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。

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）。

19.2.5.31 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。

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）。

19.2.5.32 SUMMARY_STAT_XACT_USER_TABLES

显示数据库内汇聚的命名空间中用户表的事务状态信息。

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）。

19.2.5.33 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。

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。

名称	类型	描述
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）。

19.2.5.34 STAT_XACT_USER_FUNCTIONS

视图包含当前节点本事务内函数执行的统计信息。

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。

19.2.5.35 SUMMARY_STAT_XACT_USER_FUNCTIONS

视图包含 GBase 8c 内汇聚的本事务内函数执行的统计信息。

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。

19.2.5.36 GLOBAL_STAT_XACT_USER_FUNCTIONS

视图包含各节点本事务内函数执行的统计信息。

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）。

19.2.5.37 STAT_BAD_BLOCK

获得当前节点表、索引等文件的读取失败信息。

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 z	坏块最后出现的时间。

名称	类型	描述
	one	

19.2.5.38 SUMMARY_STAT_BAD_BLOCK

获得 GBase 8c 内汇聚的表、索引等文件的读取失败信息。

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。

19.2.5.39 GLOBAL_STAT_BAD_BLOCK

获得各节点的表、索引等文件的读取失败信息。

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 zo	坏块第一次出现的时间。

名称	类型	描述
	ne	
last_time	timestamp with time zone	坏块最后出现的时间。

19.2.5.40 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。

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）。

19.2.5.41 SUMMARY_STAT_USER_FUNCTIONS

SUMMARY_STAT_USER_FUNCTIONS 用来统计所数据库节点用户自定义函数的相关统计信息。

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。

名称	类型	描述
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）。

19.2.5.42 GLOBAL_STAT_USER_FUNCTIONS

提供 GBase 8c 中各个节点的用户所创建的函数的状态的统计信息。

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。

19.2.6 Workload

19.2.6.1 WORKLOAD_SQL_COUNT

显示当前节点 workload 上的 SQL 数量分布。

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。

19.2.6.2 SUMMARY_WORKLOAD_SQL_COUNT

显示 GBase 8c 内各数据库主节点的 workload 上的 SQL 数量分布。

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。

19.2.6.3 WORKLOAD_TRANSACTION

当前节点上负载的事务信息。

名称	类型	描述
workload	name	负载的名称。

名称	类型	描述
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）。

19.2.6.4 SUMMARY_WORKLOAD_TRANSACTION

显示 GBase 8c 内汇聚的负载事务信息。

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）。

19.2.6.5 GLOBAL_WORKLOAD_TRANSACTION

显示各节点上的 workload 的负载信息。

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）。

名称	类型	描述
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）。

19.2.6.6 WORKLOAD_SQL_ELAPSE_TIME

WORKLOAD_SQL_ELAPSE_TIME 用来统计 workload（业务负载）上的 SUID 信息。

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）。

名称	类型	描述
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）。

19.2.6.7 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。

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）。

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）。
avg_delete_elapse	bigint	平均 delete 的时间花费（单位：微秒）。

19.2.6.8 USER_TRANSACTION

USER_TRANSACTION 用来统计用户执行的事务信息。monadmin 用户能看到所有用户执行事务的信息，普通用户只能查询到自己执行的事务信息。

名称	类型	描述
username	name	用户的名称。
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）。

19.2.6.9 GLOBAL_USER_TRANSACTION

GLOBAL_USER_TRANSACTION 用来统计全局用户执行的事务信息。

名称	类型	描述
node_name	name	节点名称。
username	name	用户的名称。
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）。

19.2.7 Session/Thread

19.2.7.1 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。

名称	类型	描述
sessid	text	线程启动时间+线程标识。
statid	integer	统计编号。

名称	类型	描述
statname	text	统计会话名称。
statunit	text	统计会话单位。
value	bigint	统计会话值。

19.2.7.2 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。

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。

19.2.7.3 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。

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。

19.2.7.4 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。

名称	类型	描述

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。

19.2.7.5 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 GBase 8c 线程和 Stream 线程分配的所有内存，单位为 MB。

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。

19.2.7.6 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 GBase 8c 线程和 Stream 线程分配的所有内存，单位为 MB。

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。

19.2.7.7 SESSION_MEMORY_DETAIL

统计线程的内存使用情况，以 MemoryContext 节点来统计。

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。
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）。

19.2.7.8 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。

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）。

19.2.7.9 SESSION_STAT_ACTIVITY

显示当前节点上正在运行的线程相关的信息。

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。

名称	类型	描述
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。 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 omm 的 stat 信息为空。 postgres=# SELECT datname, usename, usesysid,state,pid FROM pg_stat_activity; datname | usename | usesysid | state | pid ———-+———+———-+—— -+———— -postgres | gbase | 10 | |139968752121616 postgres | gbase | 10 | |139968903116560 db_tpcds | judy | 16398 | active |139968391403280 postgres | gbase | 10 | |139968643069712 postgres | gbase | 10 | |139968680818448 postgres | joe | 16390 | |139968563377936 (6 rows)

名称	类型	描述
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。
unique_sql_id	bigint	语句的 unique sql id。
trace_id	text	驱动传入的 trace id，与应用的一次请求相关联。

19.2.7.10 GLOBAL_SESSION_STAT_ACTIVITY

显示 GBase 8c 内各节点上正在运行的线程相关的信息。

名称	类型	描述
coorname	text	数据库进程名称。
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。

名称	类型	描述
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。 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 及初始用户的 state 信息为空。 postgres=# SELECT datname, usename,

名称	类型	描述
		usesysid,state,pid FROM pg_stat_activity; datname | usename | usesysid | state | pid ————-+————+————-+————+————————postgres | gbase | 10 | |139968752121616 postgres | gbase | 10 | |139968903116560 db_tpcds | judy | 16398 | active |139968391403280 postgres | gbase | 10 | |139968643069712 postgres | gbase | 10 | |139968680818448 postgres | 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。
unique_sql_id	bigint	语句的 unique sql id。
trace_id	text	驱动传入的 trace id，与应用的一次请求相关联。

19.2.7.11 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 15-15-2、PG_THREAD_WAIT_STATUS 中的等待状态列表、轻量级锁等待事件列表、IO 等待事件列表和事务锁等待事件列表。

名称	类型	描述
node_name	text	数据库进程名称。
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。
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 PG_THREAD_WAIT_STATUS 中等待状态列表。
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。

19.2.7.12 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。具体事件信息请参见 PG_THREAD_WAIT_STATUS 中的等待状态列表、轻量级锁等待事件列表、IO 等待事件列表和事务锁等待事件列表

通过 GLOBAL_THREAD_WAIT_STATUS 视图，可以查看 GBase 8c 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 GBase 8c 中各个节点上

查询 THREAD_WAIT_STATUS 视图汇总的结果。

名称	类型	描述
node_name	text	数据库进程名称。
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。
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 15.3.67-表 GUC 参数设置方式 等待状态列表。
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。

19.2.7.13 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。

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。

名称	类型	描述
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。 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。

19.2.7.14 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 LOCAL_THREADPOOL_STATUS。

19.2.7.15 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。

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。

名称	类型	描述
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	语句所属用户对应的逻辑 GBase 8c。
top_cpu_dn	text	cpu 使用量 topN 信息。

19.2.7.16 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。

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

名称	类型	描述
		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 GBase 8c。
top_mem_dn	text	mem 使用量 topN 信息。

19.2.7.17 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。

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 GBase 8c。

19.2.7.18 LOCAL_ACTIVE_SESSION

LOCAL_ACTIVE_SESSION 视图显示本节点上的 ACTIVE SESSION PROFILE 内存中的样本。

名称	类型	描述

名称	类型	描述
sampleid	bigint	采样 ID。
sample_time	timestamp with time zone	采样的时间。
need_flush_sample	boolean	该样本是否需要刷新的磁盘。
databaseid	oid	数据库 ID。
thread_id	bigint	线程的 ID。
sessionid	bigint	会话的 ID。
start_time	timestamp with time zone	会话的启动时间。
event	text	具体的事件名称。
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。与执行计划的层级（ID）相对应。
smpid	integer	smp 执行模式下并行线程的并行编号。
userid	oid	session 用户的 ID。
application_name	text	应用的名称。
client_addr	inet	client 端的地址。
client_hostname	text	client 端的名称。
client_port	integer	客户端用于与后端通讯的 TCP 端口号。
query_id	bigint	debug query id。
unique_query_id	bigint	unique query id。
user_id	oid	unique query 的 key 中的 user_id。

名称	类型	描述
cn_id	integer	cn id，在 DN 上表示下发该 unique sql 的节点 id，unique query 的 key 中的 cn_id。
unique_query	text	规范化后的 UniqueSQL 文本串。
locktag	text	会话等待锁信息，可通过 locktag_decode 解析。
lockmode	text	会话等待锁模式。
block_sessionid	bigint	如果会话正在等待锁，阻塞该会话获取锁的会话标识。
final_block_sessionid	bigint	表示源头阻塞会话 ID。
wait_status	text	描述 event 列的更多详细信息。
global_sessionid	text	全局会话 ID。

19.2.8 Transaction

19.2.8.1 TRANSACTIONS_PREPARED_XACTS

显示当前准备好进行两阶段提交的事务的信息。

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。

19.2.8.2 SUMMARY_TRANSACTIONS_PREPARED_XACTS

显示 GBase 8c 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。

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。

19.2.8.3 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。

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。

19.2.8.4 TRANSACTIONS_RUNNING_XACTS

显示当前节点运行事务的信息。

名称	类型	描述
handle	integer	事务对应的事务管理器中的槽位句柄，该值恒为-1。

名称	类型	描述
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	其余节点发送给当前节点的事务 id，该值恒为 0。

19.2.8.5 SUMMARY_TRANSACTIONS_RUNNING_XACTS

显示集群中各个节点运行事务的信息，字段内容和 transactions_running_xacts 一致。

名称	类型	描述
handle	integer	事务对应的事务管理器中的槽位句柄，该值恒为-1。
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。

名称	类型	描述
pid	bigint	事务对应的线程 id。
next_xid	xid	其余节点发送给当前节点的事务 id，该值恒为 0。

19.2.8.6 GLOBAL_TRANSACTIONS_RUNNING_XACTS

显示集群中各个节点运行事务的信息。

名称	类型	描述
handle	integer	事务对应的事务管理器中的槽位句柄，该值恒为-1
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	其余节点发送给当前节点的事务 id，该值恒为 0。

19.2.9 Query

19.2.9.1 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 权限或者 monitor admin 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。

名称	类型	描述
node_name	name	数据库进程名称。
node_id	integer	节点的 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。
n_soft_parse	bigint	软解析次数，n_soft_parse + n_hard_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_hard_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）。

名称	类型	描述
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）。
sort_count	bigint	排序执行的次数。
sort_time	bigint	排序执行的时间（单位：微秒）。
sort_mem_used	bigint	排序过程中使用的 work memory 大小(单位:KB)。
sort_spill_count	bigint	排序过程中，若发生落盘，写文件的次数。
sort_spill_size	bigint	排序过程中，若发生落盘，使用的文件大小(单位:KB）。
hash_count	bigint	hash 执行的次数。
hash_time	bigint	hash 执行的时间（单位：微秒）。
hash_mem_used	bigint	hash 过程中使用的 work memory 大小(单位:KB)。
hash_spill_count	bigint	hash 过程中，若发生落盘，写文件的次数。
hash_spill_size	bigint	hash 过程中，若发生落盘，使用的文件大小(单位:KB）。
last_updated	timestamp with time zone	最后一次更新该语句的时间。

19.2.9.2 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）。

名称	类型	描述
node_name	name	数据库进程名称。
node_id	integer	节点的 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。
n_soft_parse	bigint	软解析次数。
n_hard_parse	bigint	硬解析次数。

名称	类型	描述
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）。
last_updated	timestamp with time zone	最后一次更新该语句的时间。
sort_count	bigint	排序执行的次数。
sort_time	bigint	排序执行的时间（单位：微秒）。
sort_mem_used	bigint	排序过程中使用的 work memory 大小（单位：KB）。
sort_spill_count	bigint	排序过程中，若发生落盘，写文件的次数。
sort_spill_size	bigint	排序过程中，若发生落盘，使用的文件大小（单位：KB）。
hash_count	bigint	hash 执行的次数。
hash_time	bigint	hash 执行的时间（单位：微秒）。
hash_mem_used	bigint	hash 过程中使用的 work memory 大小（单位：KB）。

名称	类型	描述
hash_spill_count	bigint	hash 过程中，若发生落盘，写文件的次数。
hash_spill_size	bigint	hash 过程中，若发生落盘，使用的文件大小（单位：KB）。

19.2.9.3 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 GBase 8c 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，GBase 8c 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。

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。

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）。

19.2.9.4 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。

名称	类型	描述
node_name	text	数据库进程名称。
user_name	text	用户名。

名称	类型	描述
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）。
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）。

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）。

19.2.9.5 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。

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）。

名称	类型	描述
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）。
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）。

19.2.9.6 GLOBAL_STATEMENT_COMPLEX_HISTORY

显示各个节点执行作业结束后的负载管理记录。

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

名称	类型	描述
		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。
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。

名称	类型	描述
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。
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）。

名称	类型	描述
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 GBase 8c。
cpu_top1_node_name	text	cpu 使用率第 1 的节点名称。
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。

名称	类型	描述
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。

19.2.9.7 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 GLOBAL_STATEMENT_COMPLEX_HISTORY 中的字段。

19.2.9.8 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。

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。

名称	类型	描述
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。
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 M

名称	类型	描述
		B。
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

名称	类型	描述
		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 GBase 8c。
top_cpu_dn	text	cpu 使用量 topN 信息。
top_mem_dn	text	内存使用量 topN 信息。

19.2.9.9 STATEMENT_RESPONSETIME_PERCENTILE

获取 GBase 8cSQL 响应时间 P80，P95 分布信息。

名称	类型	描述

名称	类型	描述
p80	bigint	GBase 8c80%的 SQL 的响应时间（单位：微秒）。
p95	bigint	GBase 8c95%的 SQL 的响应时间（单位：微秒）。

19.2.9.10 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 GS_SESSION_MEMORY_DETAIL。

19.2.9.11 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 GS_SESSION_MEMORY_DETAIL。

19.2.9.12 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。具体的字段请参考表 GS_SESSION_MEMORY_DETAIL。

19.2.9.13 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。

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。

名称	类型	描述
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。
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

名称	类型	描述
		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 m

名称	类型	描述
		s。
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 GBase 8c。
top_cpu_dn	text	cpu 使用量 topN 信息。
top_mem_dn	text	内存使用量 topN 信息。

19.2.9.14 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。

名称	类型	描述

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。
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。

名称	类型	描述
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。 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 GBase 8c 模式下，语句当前是否占用其他逻辑 GBase 8c 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 GBase 8c。

19.2.9.15 STATEMENT_HISTORY

获得当前节点的执行语句的信息。查询系统表必须具有 sysadmin 权限。只可在系统库中查询到结果，用户库中无法查询。

对于此系统表查询有如下约束：

必须在 postgres 库内查询，其它库中不存数据。

此系统表受 track_stmt_stat_level 控制，默认为“OFF,L0”，第一部分控制 Full SQL，第二部分控制 Slow SQL，具体字段记录级别见下表。

对于 Slow SQL，当 track_stmt_stat_level 的值为非 OFF 时，且 SQL 执行时间超过

log_min_duration_statement，会记录为慢 SQL。

名称	类型	描述	记录级别
db_name	name	数据库名称。	L0
schema_name	name	schema 名称。	L0
origin_node	integer	节点名称。	L0
user_name	name	用户名。	L0
application_name	text	用户发起的请求的应用程序名称。	L0
client_addr	text	用户发起的请求的客户端地址。	L0
client_port	integer	用户发起的请求的客户端端口。	L0
unique_query_id	bigint	归一化 SQL ID。	L0
debug_query_id	bigint	唯一 SQL ID。	L0
query	text	归一化 SQL。	L0
start_time	timestamp with time zone	语句启动的时间。	L0
finish_time	timestamp with time zone	语句结束的时间。	L0
slow_sql_threshold	bigint	语句执行时慢 SQL 的标准。	L0
transaction_id	bigint	事务 ID。	L0
thread_id	bigint	执行线程 ID。	L0
session_id	bigint	用户 session id。	L0
n_soft_parse	bigint	软解析次数，n_soft_parse + n_hard_parse 可能大于 n_calls，因为子查询未计入 n_calls。	L0

名称	类型	描述	记录级别
n_hard_parse	bigint	硬解析次数，n_soft_parse + n_hard_parse 可能大于 n_calls，因为子查询未计入 n_calls。	L0
query_plan	text	语句执行计划。	L1
n_returned_rows	bigint	SELECT 返回的结果集行数。	L0
n_tuples_fetched	bigint	随机扫描行。	L0
n_tuples_returned	bigint	顺序扫描行。	L0
n_tuples_inserted	bigint	插入行。	L0
n_tuples_updated	bigint	更新行。	L0
n_tuples_deleted	bigint	删除行。	L0
n_blocks_fetched	bigint	buffer 的块访问次数。	L0
n_blocks_hit	bigint	buffer 的块命中次数。	L0
db_time	bigint	有效的 DB 时间花费，多线程将累加（单位：微秒）。	L0
cpu_time	bigint	CPU 时间（单位：微秒）。	L0
execution_time	bigint	执行器内执行时间（单位：微秒）。	L0
parse_time	bigint	SQL 解析时间（单位：微秒）。	L0
plan_time	bigint	SQL 生成计划时间（单位：微秒）。	L0
rewrite_time	bigint	SQL 重写时间（单位：微秒）。	L0

名称	类型	描述	记录级别
pl_execution_time	bigint	plpgsql 上的执行时间（单位：微秒）。	L0
pl_compilation_time	bigint	plpgsql 上的编译时间（单位：微秒）。	L0
data_io_time	bigint	IO 上的时间花费（单位：微秒）。	L0
net_send_info	text	通过物理连接发送消息的网络状态，包含时间（微秒）、调用次数、吞吐量（字节）。通过该字段可以分析 SQL 在分布式系统下的网络开销，单机模式下不支持该字段。例如：{“time”:xxx, “n_calls”:xxx, “size”:xxx}。	L0
net_recv_info	text	通过物理连接接收消息的网络状态，包含时间（微秒）、调用次数、吞吐量（字节）。通过该字段可以分析 SQL 在分布式系统下的网络开销，单机模式下不支持该字段。例如：{“time”:xxx, “n_calls”:xxx, “size”:xxx}。	L0
net_stream_send_info	text	通过逻辑连接发送消息的网络状态，包含时间（微秒）、调用次数、吞吐量（字节）。通过该字段可以分析 SQL 在分布式系统下的网络开销，单机模式下不支持该字段。例如：{“time”:xxx, “n_calls”:xxx, “size”:xxx}。	L0
net_stream_recv_info	text	通过逻辑连接接收消息的网络状态，包含时间（微秒）、调用次数、吞吐量（字节）。通过该字段可以分析 SQL 在分布式系统下的网络开销，单机模式下不支持该字段。例如：{“time”:xxx, “n_calls”:xxx, “size”:xxx}。	L0
lock_count	bigint	加锁次数。	L0
lock_time	bigint	加锁耗时。	L1

名称	类型	描述	记录级别
lock_wait_count	bigint	加锁等待次数。	L0
lock_wait_time	bigint	加锁等待耗时。	L1
lock_max_count	bigint	最大持锁数量。	L0
lwlock_count	bigint	轻量级加锁次数（预留）。	L0
lwlock_wait_count	bigint	轻量级等锁次数。	L0
lwlock_time	bigint	轻量级加锁时间（预留）。	L1
lwlock_wait_time	bigint	轻量级等锁时间。	L1
details	bytea	语句锁事件的列表，该列表按时间书序记录事件，记录的数量受参数 track_stmt_details_size 的影响。该字段为二进制，需要借助解析函数 pg_catalog.statement_detail_decode 读取。 事件包括： 加锁开始 加锁结束 等锁开始 等锁结束 放锁开始 放锁结束 轻量级等锁开始 轻量级等锁结束	L2
is_slow_sql	boolean	该 SQL 是否为 slow SQL。 t（true）：表示是。 f（false）：表示不是。	L0
trace_id	text	驱动传入的 trace id，与应用的一次请求相关联。	L0

19.2.10Cache/IO

19.2.10.1 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。

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）。

19.2.10.2 SUMMARY_STATIO_USER_TABLES

SUMMARY_STATIO_USER_TABLES 视图显示 GBase 8c 内汇聚的命名空间中所有用户关系表的 IO 状态信息。

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。

名称	类型	描述
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）。

19.2.10.3 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。

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）。

19.2.10.4 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。

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。

19.2.10.5 SUMMARY_STATIO_USER_INDEXES

SUMMARY_STATIO_USER_INDEXES 视图显示 GBase 8c 内汇聚的命名空间中所有用户关系表索引的 IO 状态信息。

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。

19.2.10.6 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。

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。

19.2.10.7 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。

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。

19.2.10.8 SUMMARY_STATIO_USER_SEQUENCES

SUMMARY_STATIO_USER_SEQUENCES 视图显示 GBase 8c 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。

名称	类型	描述
schemaname	name	序列中模式名。

名称	类型	描述
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。

19.2.10.9 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。

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。

19.2.10.10 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。

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。

名称	类型	描述
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）。

19.2.10.11 SUMMARY_STATIO_SYS_TABLES

SUMMARY_STATIO_SYS_TABLES 视图显示 GBase 8c 内汇聚的命名空间中所有系统表的 IO 状态信息。

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）。

19.2.10.12 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。

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。
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）。

19.2.10.13 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。

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。

19.2.10.14 SUMMARY_STATIO_SYS_INDEXES

SUMMARY_STATIO_SYS_INDEXES 视图显示 GBase 8c 内汇聚的命名空间中所有系统表索引的 IO 状态信息。

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。

19.2.10.15 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。

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。

19.2.10.16 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。

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。

19.2.10.17 SUMMARY_STATIO_SYS_SEQUENCES

SUMMARY_STATIO_SYS_SEQUENCES 视图显示 GBase 8c 内汇聚的命名空间中所有系统表为序列的 IO 状态信息。

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。

19.2.10.18 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。

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。

名称	类型	描述
blks_hit	bigint	序列中缓存命中数。

19.2.10.19 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。

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）。

19.2.10.20 SUMMARY_STATIO_ALL_TABLES

SUMMARY_STATIO_ALL_TABLES 视图将包含 GBase 8c 内汇聚的数据库中每个表(包括 TOAST 表）的 I/O 的统计。

名称	类型	描述
schemaname	name	该表模式名。
relname	name	表名。

名称	类型	描述
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）。

19.2.10.21 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。

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。
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）。

名称	类型	描述
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）。

19.2.10.22 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。

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。

19.2.10.23 SUMMARY_STATIO_ALL_INDEXES

SUMMARY_STATIO_ALL_INDEXES 视图包含含 GBase 8c 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。

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。

19.2.10.24 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。

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。

19.2.10.25 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。

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。

19.2.10.26 SUMMARY_STATIO_ALL_SEQUENCES

SUMMARY_STATIO_ALL_SEQUENCES 视图包含 GBase 8c 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。

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。

19.2.10.27 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。

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。

19.2.10.28 GLOBAL_STAT_SESSION_CU

GLOBAL_STAT_DB_CU 视图用于查询 GBase 8c，每个数据库的 CU 命中情况。可以通过 pg_stat_reset()进行清零。

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_re	bigint	硬盘同步读次数。

名称	类型	描述
ad		
hdd_asyn_read	bigint	硬盘异步读次数。

19.2.10.29 GLOBAL_STAT_DB_CU

GLOBAL_STAT_SESSION_CU 用于查询 GBase 8c 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。GBase 8c 重启后，统计数据也会清零。

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。

19.2.11 Utility

19.2.11.1 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。

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 wi	程序启动时间。

名称	类型	描述
	th time zone	
state	text	日志复制的状态： 追赶状态。 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。

19.2.11.2 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。

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。

名称	类型	描述
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。 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。

19.2.11.3 REPLICATION_SLOTS

REPLICATION_SLOTS 视图用于查看复制节点的信息。

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。

名称	类型	描述
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。

19.2.11.4 GLOBAL_REPLICATION_SLOTS

GLOBAL_REPLICATION_SLOTS 视图用于查看 GBase 8c 各节点的复制节点的信息。

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。
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。

名称	类型	描述
dummy_standby	boolean	复制节点是否为假备。

19.2.11.5 BGWRITER_STAT

BGWRITER_STAT 视图显示关于后端写进程活动的统计信息。

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。
stats_reset	timestamp with time zone	这些统计被重置的时间。

19.2.11.6 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。

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。

19.2.11.7 GLOBAL_CKPT_STATUS

GLOBAL_CKPT_STATUS 视图用于显示 GBase 8c 所有实例的检查点信息和各类日志刷页情况。

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。

19.2.11.8 GLOBAL_DOUBLE_WRITE_STATUS

GLOBAL_DOUBLE_WRITE_STATUS 视图显示 GBase 8c 所有实例的双写文件的情况。它是由每个节点的 local_double_write_stat 视图组成，属性完全一致。

名称	类型	描述
node_name	text	节点名称。
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 页面）。
high_threshold_wr	bigint	高效率写双写文件的 I/O 次数（一次 I/O 刷页数量

名称	类型	描述
ites		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。
file_id	bigint	当前双写文件的 id 号。

19.2.11.9 GLOBAL_PAGEWRITER_STATUS

GLOBAL_PAGEWRITER_STATUS 视图显示 GBase 8c 实例的刷页信息和检查点信息。

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。

19.2.11.10 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 GBase 8c 统计信息时间。

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。

19.2.11.11 GLOBAL_REDO_STATUS

GLOBAL_REDO_STATUS 视图显示 GBase 8c 实例的日志回放情况。

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。

名称	类型	描述
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。
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。

19.2.11.12 GLOBAL_RECOVERY_STATUS

GLOBAL_RECOVERY_STATUS 视图显示关于主机和备机的日志流控信息。

名称	类型	描述
node_name	text	主机进程名称，包含主机和备机。
standby_node	text	备机进程名称。

名称	类型	描述
_name		
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。

19.2.11.13 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。

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。

19.2.11.14 USER_LOGIN

USER_LOGIN 用来记录用户登录和退出次数的相关信息。

名称	类型	描述

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。

19.2.11.15 SUMMARY_USER_LOGIN

USER_LOGIN 用来记录用户登录和退出次数的相关信息。

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。

19.2.11.16 GLOBAL_GET_BGWRITER_STATUS

GLOBAL_GET_BGWRITER_STATUS 视图显示所有实例 bgwriter 线程刷页信息，候选 buffer 链中页面个数，buffer 淘汰信息。

名称	类型	描述
node_name	text	实例名称。
bgwr_actual_flush_total_num	bigint	从启动到当前时间 bgwriter 线程总计刷脏页数量。
bgwr_last_flush_n	integer	bgwriter 线程上一批刷脏页数量。

名称	类型	描述
um		
candidate_slots	integer	当前候选 buffer 链中页面个数。
get_buffer_from_list	bigint	buffer 淘汰从候选 buffer 链中获取页面的次数。
get_buffer_clock_sweep	bigint	buffer 淘汰从原淘汰方案中获取页面的次数。

19.2.11.17 GLOBAL_SINGLE_FLUSH_DW_STATUS

GLOBAL_SINGLE_FLUSH_DW_STATUS 视图显示数据库所有实例单页面淘汰双写文件信息。展示内容中，/前是第一个版本双写文件刷页情况，/后是第二个版本双写文件刷页情况。

名称	类型	描述
node_name	text	实例名称。
curr_dwn	text	当前双写文件的序列号。
curr_start_page	text	当前双写文件 start 位置。
total_writes	text	当前双写文件总计写数据页面个数。
file_trunc_num	text	当前双写文件复用的次数。
file_reset_num	text	当前双写文件写满后发生重置的次数。

19.2.11.18 GLOBAL_CANDIDATE_STATUS

GLOBAL_CANDIDATE_STATUS 视图显示整个数据库所有实例候选 buffer 个数，buffer 淘汰信息。

名称	类型	描述
node_name	text	节点名称。

名称	类型	描述
candidate_slots	integer	当前 Normal Buffer Pool 候选 buffer 链中页面个数。
get_buf_from_list	bigint	Normal Buffer Pool，buffer 淘汰从候选 buffer 链中获取页面的次数。
get_buf_clock_sweep	bigint	Normal Buffer Pool，buffer 淘汰从原淘汰方案中获取页面的次数。
seg_candidate_slots	integer	当前 Segment Buffer Pool 候选 buffer 链中页面个数。
seg_get_buf_from_list	bigint	Segment Buffer Pool，buffer 淘汰从候选 buffer 链中获取页面的次数。
seg_get_buf_clock_sweep	bigint	Segment Buffer Pool，buffer 淘汰从原淘汰方案中获取页面的次数。

19.2.12Lock

19.2.12.1 LOCKS

LOCKS 视图用于查看各打开事务所持有的锁信息。

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。 如果是一个事务 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。

名称	类型	描述
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 0；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。
locktag	text	会话等待锁信息，可通过 locktag_decode()函数解析。
global_sessionid	text	全局会话 ID。

19.2.12.2 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。

名称	类型	描述
node_name	name	数据库进程名称。

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。 如果是一个事务 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。

19.2.13Wait Events

19.2.13.1 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 PG_THREAD_WAIT_STATUS 中等待状态列表、轻量级锁等待事件列表、IO 等待事件列表和事务锁等待事件列表。关于每种事务锁对业务的影响程度，请参考 LOCK 语法小节的详细描述。

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）。
last_updated	timestamp with time zone	最后一次更新该事件的时间。

19.2.13.2 GLOBAL_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 PG_THREAD_WAIT_STATUS 中等待状态列表、轻量级锁等待事件列表、IO 等待事件列表和事务锁等待事件列表。关于每种事务锁对业务的影响程度，请参考 LOCK 语法小节的详细描述。

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）。
last_updated	timestamp with time zone	最后一次更新该事件的时间。

19.2.14Configuration

19.2.14.1 CONFIG_SETTINGS

CONFIG_SETTINGS 视图显示数据库运行时参数的相关信息。

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、

名称	类型	描述
		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。
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。

19.2.14.2 GLOBAL_CONFIG_SETTINGS

GLOBAL_CONFIG_SETTINGS 显示各节点数据库运行时参数的相关信息。

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。

名称	类型	描述
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 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。

19.2.15Operator

19.2.15.1 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 1 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。

名称	类型	描述
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。
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）。

名称	类型	描述
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。

19.2.15.2 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 GS_WLM_OPERATOR_INFO。

19.2.15.3 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。

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
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。
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：

名称	类型	描述
		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。

19.2.15.4 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。

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。
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。

19.2.15.5 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 GLOBAL_OPERATOR_HISTORY。

19.2.15.6 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。

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)，

名称	类型	描述
		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。

19.2.16Workload Manager

19.2.16.1 WLM_USER_RESOURCE_CONFIG

WLM_USER_RESOURCE_CONFIG 视图显示用户的资源配置信息。

名称	类型	描述
userid	oid	用户 oid。
username	name	用户名称。
sysadmin	boolean	是否是 sysadmin。

名称	类型	描述
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。

19.2.16.2 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。

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。

名称	类型	描述
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。

19.2.17Global Plancache

GPC 相关视图在 enable_global_plancache 打开且线程池打开的状态下才有效。

19.2.17.1 GLOBAL_PLANCACHE_STATUS

GLOBAL_PLANCACHE_STATUS 视图显示 GPC 全局计划缓存状态信息。

名称	类型	描述
nodename	text	所属节点名称。
query	text	查询语句 text。
refcount	integer	被引用次数。
valid	bool	是否合法。
databaseid	oid	所属数据库 id。
schema_name	text	所属 schema。
params_num	integer	参数数量。
func_id	oid	该 plancache 所在存储过程 oid，如果不属于存储过程则为 0。

19.2.17.2 GLOBAL_PLANCACHE_CLEAN

GLOBAL_PLANCACHE_CLEAN 视图用于清理所有节点上无人使用的全局计划缓存。返回值为 Boolean 类型。

19.2.18RTO & RPO

19.2.18.1 global_rto_status

global_rto_status 视图显示关于主机和备机的日志流控信息（本节点除外、备 DN 上不可使用）。

参数	类型	描述
node_name	text	节点的名称，包含主机和备机。
rto_info	text	流控的信息，包含了备机当前的日志流控时间（单位：秒），备机通过 GUC 参数设置的预期流控时间（单位：秒），为了达到这个预期主机所需要的睡眠时间（单位：微秒）。

19.3 WDR Snapshot Schema

WDR Snasphot 在启动后(打开参数 enable_wdr_snapshot)，会在用户表空间”pg_default”，数据库”postgres”下新建 schema “snapshot”，用于持久化 WDR 快照数据。默认初始化用户或 monadmin 用户可以访问 Snapshot Schema。

根据参数 wdr_snapshot_retention_days 来自动管理快照的生命周期。

19.3.1 WDR Snapshot 原信息表

19.3.1.1.1 SNAPSHOT.SNAPSHOT

SNAPSHOT 表记录当前系统中存储的 WDR 快照数据的索引信息、开始时间和结束时间。只能在系统库中查询到结果，在用户库中无法查询。

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

19.3.1.1.2 SNAPSHOT.TABLES_SNAP_TIMESTAMP

TABLES_SNAP_TIMESTAMP 表记录所有存储的 WDR snapshot 中数据库、表对象、以及数据采集的开始和结束时间。

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

须知：

用户应该禁止对 Snapshot schema 下的表进行增删改等操作，人为对这些表的修改或破坏可能会导致 WDR 各种异常情况甚至 WDR 不可用。

19.3.1.1.3 SNAP_SEQ

snap_seq 是一个递增的 sequence，其为 WDR snapshot 提供快照的 ID。

19.3.2 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 下的视图。

19.3.3 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，默认初始化用户或监控管理员

用户可以生成报告，在 V500R001C20SPC003 之前的版本初始化用户或者 sysadmin 用户可以生成报告。

前提条件

WDR Snasphot 启动（即参数 enable_wdr_snapshot 为 on 时），且快照数量大于等于 2。

操作步骤

1、执行如下命令新建报告文件。

```
touch    /home/om/wdrTestNode.html
```

2、执行以下命令连接 postgres 数据库。

```
gsql -d postgres -p  端口号  -r
```

3、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;
```

4、（可选）在 CCN 上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看在当前时间段数据库的监控数据，可以选择手动执行快照操作，该命令需要用户具有 sysadmin 权限。

```
select create_wdr_snapshot();
```

说明： 执行“cm_ctl query -Cdvi”，回显中“Central Coordinator State”下显示的信息即为 CCN 信息。

5、执行如下命令，在本地生成 HTML 格式的 WDR 报告。

- 执行如下命令，设置报告格式。\a: 不显示表行列符号， \t: 不显示列名 ，\o: 指定输出文件。

```
gsql> \a
gsql> \t
gsql> \o /home/om/wdrTestNode.html
```

- 执行如下命令，生成 HTML 格式的 WDR 报告。

```
gsql> select generate_wdr_report(begin_snap_id Oid, end_snap_id Oid, int report_type, int report_scope, int node_name );
```

示例一，生成集群级别的报告：

```
select generate_wdr_report(1, 2, 'all', 'cluster',null);
```

示例二，生成某个节点的报告：

```
select generate_wdr_report(1, 2, 'all', 'node', pgxc_node_str()::cstring);
```

说明： 当前 GBase 8c 的节点名固定是“dn_6001_6002_6003”，也可直接代入。

表 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）。	-
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，该值可以省略或者指定为空或 NULL。	node：GBase 8c 中的节点名称。 cluster：省略/空/NULL。

- 执行如下命令关闭输出选项及格式化输出命令。

```
\o \a \t
```

6、在/home/om/下根据需要查看 WDR 报告。

示例

```
--创建报告文件
touch    /home/om/wdrTestNode.html


--连接数据库
gsql -d postgres -p  端口号  -r


--查询已经生成的快照。
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
(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。
postgres=# select generate_wdr_report(1, 2, 'all', 'node', 'dn_6001_6002_6003');


--关闭性能报告 wdrTestNode.html。
postgres=# \o


--生成格式化性能报告 wdrTestCluster.html。
postgres=# \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。
postgres=# \o \a \t
Output format is aligned.
Tuples only is off.
```

19.3.4 查看 WDR 报告

19.3.4.1 Database Stat

Database Stat 列名称及描述如下表所示。

列名称	描述
DB Name	数据库名称。
Backends	连接到该数据库的后端数。
Xact Commit	此数据库中已经提交的事务数。
Xact Rollback	此数据库中已经回滚的事务数。
Blks Read	在这个数据库中读取的磁盘块的数量。
Blks Hit	高速缓存中已经发现的磁盘块的次数。
Tuple Returned	顺序扫描的行数。
Tuple Fetched	随机扫描的行数。
Tuple Inserted	通过数据库查询插入的行数。
Tuple Updated	通过数据库查询更新的行数。
Tup Deleted	通过数据库查询删除的行数。
Conflicts	由于数据库恢复冲突取消的查询数量。
Temp Files	通过数据库查询创建的临时文件数量。

列名称	描述
Temp Bytes	通过数据库查询写入临时文件的数据总量。
Deadlocks	在该数据库中检索的死锁数。
Blk Read Time	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	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	重置当前状态统计的时间。

19.3.4.2 Load Profile

Load Profile 指标名称及描述如下表所示。

指标名称	描述
DB Time(us)	作业运行的 elapse time 总和。
CPU Time(us)	作业运行的 CPU 时间总和。
Redo size(blocks)	产生的 WAL 的大小（块数）。
Logical read (blocks)	表或者索引文件的逻辑读（块数）。
Physical read (blocks)	表或者索引的物理读（块数）。
Physical write (blocks)	表或者索引的物理写（块数）。
Read IO requests	表或者索引的读次数。
Write IO requests	表或者索引的写次数。
Read IO (MB)	表或者索引的读大小（MB）。
Write IO (MB)	表或者索引的写大小（MB）。
Logons	登录次数。
Executes (SQL)	SQL 执行次数。
Rollbacks	回滚事务数。

指标名称	描述
Transactions	事务数。
SQL response time P95(us)	95%的 SQL 的响应时间。
SQL response time P80(us)	80%的 SQL 的响应时间。

19.3.4.3 Instance Efficiency Percentages

Instance Efficiency Percentages 指标名称及描述如下表所示。

指标名称	描述
Buffer Hit %	Buffer Pool 命中率。
Effective CPU %	CPU time 占 DB time 的比例。
WalWrite NoWait %	访问 WAL Buffer 的 event 次数占总 wait event 的比例。
Soft Parse %	软解析的次数占总的解析次数的比例。
Non-Parse CPU %	非 parse 的时间占执行总时间的比例。

19.3.4.4 Top 10 Events by Total Wait Time

Top 10 Events by Total Wait Time 列名称及描述如下表所示。

列名称	描述
Event	Wait Event 名称。
Waits	wait 次数。
Total Wait Time(us)	总 wait 时间（微秒）。
Avg Wait Time(us)	平均 wait 时间（微秒）。
Type	Wait Event 类别。

19.3.4.5 Wait Classes by Total Wait Time

Wait Classes by Total Wait Time 列名称及描述如下表所示。

列名称	描述
Type	Wait Event 类别名称： STATUS。 LWLOCK_EVENT。 LOCK_EVENT。 IO_EVENT。
Waits	Wait 次数。
Total Wait Time(us)	总 Wait 时间（微秒）。
Avg Wait Time(us)	平均 Wait 时间（微秒）。

19.3.4.6 Host CPU

Host CPU 列名称及描述如下表所示。

列名称	描述
Cpus	CPU 数量。
Cores	CPU 核数。
Sockets	CPU Sockets 数量。
Load Average Begin	开始 snapshot 的 Load Average 值。
Load Average End	结束 snapshot 的 Load Average 值。
%User	用户态在 CPU 时间上的占比。

列名称	描述
%System	内核态在 CPU 时间上的占比。
%WIO	Wait IO 在 CPU 时间上的占比。
%Idle	空闲时间在 CPU 时间上的占比。

19.3.4.7 IO Profile

IO Profile 指标名称及描述如下表所示。

指标名称	描述
Database requests	Database IO 次数。
Database (MB)	Database IO 数据量。
Database (blocks)	Database IO 数据块。
Redo requests	Redo IO 次数。
Redo (MB)	Redo IO 量。

19.3.4.8 Memory Statistics

Memory Statistics 指标名称及描述如下表所示。

指标名称	描述
shared_used_memory	已经使用共享内存大小（MB）。
max_shared_memory	最大共享内存（MB）。
process_used_memory	进程已经使用内存（MB）。
max_process_memory	最大进程内存（MB）。

19.3.4.9 Time Model

Time Model 名称及描述如下表所示。

名称	描述
DB_TIME	所有线程端到端的墙上时间（WALL TIME）消耗总和（单位：微秒）。
EXECUTION_TIME	消耗在执行器上的时间总和（单位：微秒)。
PL_EXECUTION_TIME	消耗在 PL/SQL 执行上的时间总和（单位：微秒）。
CPU_TIME	所有线程 CPU 时间消耗总和（单位：微秒）。
PLAN_TIME	消耗在执行计划生成上的时间总和（单位：微秒）。
REWRITE_TIME	消耗在查询重写上的时间总和（单位：微秒）。
PL_COMPILATION_TIME	消耗在 SQL 编译上的时间总和（单位：微秒）。
PARSE_TIME	消耗在 SQL 解析上的时间总和（单位：微秒）。
NET_SEND_TIME	消耗在网络发送上的时间总和（单位：微秒）。
DATA_IO_TIME	消耗在数据读写上的时间总和（单位：微秒）。

19.3.4.10 SQL Statistics

SQL Statistics 列名称及描述如下表所示。

列名称	描述
Unique SQL Id	归一化的 SQL ID。
Node Name	节点名称。

列名称	描述
User Name	用户名称。
Tuples Read	访问的元组数量。
Calls	调用次数。
Min Elapse Time(us)	最小执行时间（us）。
Max Elapse Time(us)	最大执行时间（us）。
Total Elapse Time(us)	总执行时间（us）。
Avg Elapse Time(us)	平均执行时间（us）。
Returned Rows	SELECT 返回行数。
Tuples Affected	Insert/Update/Delete 行数。
Logical Read	Buffer 逻辑读次数。
Physical Read	Buffer 物理读次数。
CPU Time (us)	CPU 时间（us）。
Data IO Time(us)	IO 上的时间花费（us）。
Sort Count	排序执行的次数。
Sort Time(us)	排序执行的时间（us）。
Sort Mem	排序过程中使用的 work memory 大小（KB）。

列名称	描述
Used(KB)	
Sort Spill Count	排序过程中，若发生落盘，写文件的次数。
Sort Spill Size(KB)	排序过程中，若发生落盘，使用的文件大小（KB）。
Hash Count	hash 执行的次数。
Hash Time (us)	hash 执行的时间（us）。
Hash Mem Used(KB)	hash 过程中使用的 work memory 大小（KB）。
Hash Spill Count	hash 过程中，若发生落盘，写文件的次数。
Hash Spill Size(KB)	hash 过程中，若发生落盘，使用的文件大小（KB）。
SQL Text	归一化 SQL 字符串。

19.3.4.11 Wait Events

Wait Events 列名称及描述如下表所示。

列名称	描述
Type	Wait Event 类别名称： STATUS。 LWLOCK_EVENT。 LOCK_EVENT。 IO_EVENT。
Event	Wait Event 名称。
Total Wait Time (us)	总 Wait 时间（us）。

列名称	描述
Waits	总 Wait 次数。
Failed Waits	Wait 失败次数。
Avg Wait Time (us)	平均 Wait 时间（us）。
Max Wait Time (us)	最大 Wait 时间（us）。

19.3.4.12 Cache IO Stats

Cache IO Stats 包含 User table 和 User index 两张表，列名称及描述如下所示。

User table IO activity ordered by heap blks hit ratio

列名称	描述
DB Name	Database 名称。
Schema Name	Schema 名称。
Table Name	Table 名称。
%Heap Blks Hit Ratio	此表的 Buffer Pool 命中率。
Heap Blks Read	该表中读取的磁盘块数。
Heap Blks Hit	此表缓存命中数。
Idx Blks Read	表中所有索引读取的磁盘块数。
Idx Blks Hit	表中所有索引命中缓存数。
Toast Blks Read	此表的 TOAST 表读取的磁盘块数（如果存在）。
Toast Blks Hit	此表的 TOAST 表命中缓冲区数（如果存在）。
Tidx Blks Read	此表的 TOAST 表索引读取的磁盘块数（如果存在）。

列名称	描述
Tidx Blks Hit	此表的 TOAST 表索引命中缓冲区数（如果存在）。

User index IO activity ordered by idx blks hit ratio

列名称	描述
DB Name	Database 名称。
Schema Name	Schema 名称。
Table Name	Table 名称。
Index Name	Index 名称。
%Idx Blks Hit Ratio	Index 的命中率。
Idx Blks Read	所有索引读取的磁盘块数。
Idx Blks Hit	所有索引命中缓存数。

19.3.4.13 Utility status

Utility status 包含 Replication slot 和 Replication stat 两张表，列名称及描述如下所示。

Replication slot

列名称	描述
Slot Name	复制节点名。
Slot Type	复制节点类型。
DB Name	复制节点数据库名称。
Active	复制节点状态。
Xmin	复制节点事务标识。
Restart Lsn	复制节点的 Xlog 文件信息。
Dummy Standby	复制节点假备。

Replication stat

列名称	描述
Thread Id	线程的 PID。
Usesys Id	用户系统 ID。
Usename	用户名称。
Application Name	应用程序。
Client Addr	客户端地址。
Client Hostname	客户端主机名。
Client Port	客户端端口。
Backend Start	程序起始时间。
State	日志复制状态。
Sender Sent Location	发送端发送日志位置。
Receiver Write Location	接收端 write 日志位置。
Receiver Flush Location	接收端 flush 日志位置。
Receiver Replay Location	接收端 replay 日志位置。
Sync Priority	同步优先级。
Sync State	同步状态。

19.3.4.14 Object stats

Object stats 包含 User Tables stats、User index stats 和 Bad lock stats 三张表，列名称及描述如下所示。

User Tables stats

列名称	描述
DB Name	Database 名称。

列名称	描述
Schema	Schema 名称。
Relname	Relation 名称。
Seq Scan	此表发起的顺序扫描数。
Seq Tup Read	顺序扫描抓取的活跃行数。
Index Scan	此表发起的索引扫描数。
Index Tup Fetch	索引扫描抓取的活跃行数。
Tuple Insert	插入行数。
Tuple Update	更新行数。
Tuple Delete	删除行数。
Tuple Hot Update	HOT 更新行数（即没有更新所需的单独索引）。
Live Tuple	估计活跃行数。
Dead Tuple	估计死行数。
Last Vacuum	最后一次此表是手动清理的（不计算 VACUUM FULL）时间。
Last Autovacuum	上次被 autovacuum 守护进程清理的时间。
Last Analyze	上次手动分析这个表的时间。
Last Autoanalyze	上次被 autovacuum 守护进程分析的时间。
Vacuum Count	这个表被手动清理的次数（不计算 VACUUM FULL）。
Autovacuum Count	这个表被 autovacuum 清理的次数。
Analyze Count	这个表被手动分析的次数。

列名称	描述
Autoanalyze Count	这个表被 autovacuum 守护进程分析的次数。

User index stats

列名称	描述
DB Name	Database 名称。
Schema	Schema 名称。
Relname	Relation 名称。
Index Relname	Index 名称。
Index Scan	索引上开始的索引扫描数。
Index Tuple Read	通过索引上扫描返回的索引项数。
Index Tuple Fetch	通过使用索引的简单索引扫描抓取的表行数。

Bad lock stats

列名称	描述
DB Id	数据库的 OID。
Tablespace Id	表空间的 OID。
Relfilenode	文件对象 ID。
Fork Number	文件类型。
Error Count	失败计数。
First Time	第一次发生时间。
Last Time	最近一次发生时间。

19.3.4.15 Configuration settings

Configuration settings 列名称及描述如下表所示。

列名称	描述
Name	GUC 名称。
Abstract	GUC 描述。
Type	数据类型。
Curent Value	当前值。
Min Value	合法最小值。
Max Value	合法最大值。
Category	GUC 类别。
Enum Values	如果是枚举值，列举所有枚举值。
Default Value	数据库启动时参数默认值。
Reset Value	数据库重置时参数默认值。

19.3.4.16 SQL Detail

SQL Detail 列名称及描述如下表所示。

列名称	描述
Unique SQL Id	归一化 SQL ID。
User Name	用户名称。
Node Name	节点名称。Node 模式下不显示该字段。
SQL Text	归一化 SQL 文本。

19.4 DBE_PLDEBUGGER Schema

DBE_PLDEBUGGER 下系统函数用于单机下调试存储过程，目前支持的接口及其描述如下所示。仅管理员有权限执行这些调试接口。

⚠ 须知： 当在函数体中创建用户时，调用 attach、next、continue、 info_code、step、info_breakpoint、backtrace、 finish 中会返回密码的明文。因此不建议用户在函数体中创建用户。

对应权限角色为 gs_role_pldebugger，可以由管理员用户通过如下命令将 debugger 权限赋权给该用户。

```
GRANT gs_role_pldebugger to user;
```

需要有两个客户端连接数据库，一个客户端负责执行调试接口作为 debug 端，另一个客户端执行调试函数，控制 server 端存储过程执行。示例如下。

- **准备调试**

通过 PG_PROC，查找到待调试存储过程的 oid，并执行 DBE_PLDEBUGGER.turn_on(oid)。本客户端就会作为 server 端使用。

```
postgres=# CREATE OR REPLACE PROCEDURE test_debug ( IN    x INT)
AS
BEGIN
      INSERT INTO t1 (a) VALUES (x);
      DELETE FROM t1 WHERE a = x;
END;
/
CREATE PROCEDURE
postgres=# SELECT OID FROM PG_PROC WHERE PRONAME='test_debug';
   oid
-------
  16389
(1 row)
```

```
postgres=# SELECT * FROM DBE_PLDEBUGGER.turn_on(16389);
 nodename | port
----------+------
 datanode |    0
(1 row)
```

- 开始调试

server 端执行存储过程，会在存储过程内第一条 SQL 语句前 hang 住，等待 debug 端发送的调试消息。仅支持直接执行存储过程的调试，不支持通过 trigger 调用执行的存储过程调试。

```
postgres=# call test_debug(1);
```

再起一个客户端，作为 debug 端，通过 turn_on 返回的数据，调用 DBE_PLDEBUGGER.attach 关联到该存储过程上进行调试。

```
postgres=# SELECT * FROM DBE_PLDEBUGGER.attach('datanode',0);
 funcoid |   funcname   | lineno |                    query
---------+------------+--------+----------------------------------
    16389 | test_debug |      3 |    INSERT INTO t1 (a) VALUES (x);
(1 row)
```

在执行 attach 的客户端调试，执行下一条 statement。

```
postgres=# SELECT * FROM DBE_PLDEBUGGER.next();
 funcoid |   funcname   | lineno |          query
---------+------------+--------+----------------------
    16389 | test_debug |      0 | [EXECUTION FINISHED]
(1 row)
```

在执行 attach 的客户端调试，可以执行以下变量操作

```
postgres=# SELECT * FROM DBE_PLDEBUGGER.info_locals(); --打印全部变量
 varname | vartype | value | package_name | isconst
---------+---------+-------+--------------+---------
```

```
 x          | int4      | 1       |                     | f
(1 row)
postgres=# SELECT * FROM DBE_PLDEBUGGER.set_var('x', 2); --变量赋值
 set_var
---------
 t
(1 row)
postgres=# SELECT * FROM DBE_PLDEBUGGER.print_var('x'); --打印单个变量
 varname | vartype | value | package_name | isconst
---------+---------+-------+--------------+---------
 x          | int4      | 2       |                     | f
(1 row)
```

直接执行完成当前正在调试的存储过程。

```
postgres=# SELECT * FROM DBE_PLDEBUGGER.continue();
 funcoid |   funcname   | lineno |             query
---------+------------+--------+---------------------
    16389 | test_debug |          0 | [EXECUTION FINISHED]
(1 row)
```

直接退出当前正在调试的存储过程，不执行尚未执行的语句。

```
postgres=# SELECT * FROM DBE_PLDEBUGGER.abort();
 abort
-------
 t
(1 row)
```

client 端查看代码信息并识别可以设置断点行号。

```
postgres=# SELECT * FROM DBE_PLDEBUGGER.info_code(16389);
 lineno |                                query                                | canbreak
```

```
--------+----------------------------------------------------------+----------
          | CREATE OR REPLACE PROCEDURE public.test_debug( IN    x INT) | f
        1 | AS    DECLARE                                                    | f
        2 | BEGIN                                                            | f
        3 |         INSERT INTO t1 (a) VALUES (x);                           | t
        4 |         DELETE FROM t1 WHERE a = x;                              | t
        5 | END;                                                             | f
        6 | /                                                                | f
(7 rows)
```

设置断点。

```
postgres=# SELECT * FROM DBE_PLDEBUGGER.add_breakpoint(16389,4);
 lineno |                                query                               | canbreak
--------+----------------------------------------------------------+----------
          | CREATE OR REPLACE PROCEDURE public.test_debug( IN    x INT) | f
        1 | AS    DECLARE                                                    | f
        2 | BEGIN                                                            | f
        3 |         INSERT INTO t1 (a) VALUES (x);                           | t
        4 |         DELETE FROM t1 WHERE a = x;                              | t
        5 | END;                                                             | f
        6 | /                                                                | f
(7 rows)
```

查看断点信息。

```
postgres=# SELECT * FROM DBE_PLDEBUGGER.info_breakpoints();
 breakpointno | funcoid | lineno |                query                | enable
--------------+---------+--------+-------------------------------------+--------
            0 |   16389 |      4 |       DELETE FROM t1 WHERE a = x; | t
(1 row)
```

执行至断点。

```
postgres=# SELECT * FROM DBE_PLDEBUGGER.continue();
 funcoid |   funcname   | lineno |                query
---------+--------------+--------+--------------------------------
   16389 | test_debug |      4 |       DELETE FROM t1 WHERE a = x;
(1 row)
```

存储过程执行结束后，调试会自动退出，再进行调试需要重新 attach 关联。如果 server 端不需要继续调试，可执行 turn_off 关闭，或退出 session。具体调试接口请见下面列表。

表 DBE_PLDEBUGGER

接口名称	描述
DBE_PLDEBUGGER.turn_on	server 端调用，标记存储过程可以调试，调用后执行该存储过程时会 hang 住等待调试信息。
DBE_PLDEBUGGER.turn_off	server 端调用，标记存储过程关闭调试。
DBE_PLDEBUGGER.local_debug_server_info	server 端调用，打印本 session 内所有已 turn_on 的存储过程。
DBE_PLDEBUGGER.attach	debug 端调用，关联到正在调试存储过程。
DBE_PLDEBUGGER.info_locals	debug 端调用，打印正在调试的存储过程中的变量当前值。
DBE_PLDEBUGGER.next	debug 端调用，单步执行。
DBE_PLDEBUGGER.continue	debug 端调用，继续执行，直到断点或存储过程结束。
DBE_PLDEBUGGER.abort	debug 端调用，停止调试，server 端报错长跳转。
DBE_PLDEBUGGER.print_var	debug 端调用，打印正在调试的存储过程中指定的变量当前值。
DBE_PLDEBUGGER.info_code	debug 和 server 端都可以调用，打印指定存储过程的源语句和各行对应的行号。
DBE_PLDEBUGGER.step	debug 端调用，单步进入执行。

接口名称	描述
DBE_PLDEBUGGER.add_breakpoint	debug 端调用，新增断点。
DBE_PLDEBUGGER.delete_breakpoint	debug 端调用，删除断点。
DBE_PLDEBUGGER.info_breakpoints	debug 端调用，查看当前的所有断点。
DBE_PLDEBUGGER.backtrace	debug 端调用，查看当前的调用栈。
DBE_PLDEBUGGER.enable_breakpoint	debug 端调用，激活被禁用的断点。
DBE_PLDEBUGGER.disable_breakpoint	debug 端调用，禁用已激活的断点。
DBE_PLDEBUGGER.finish	debug 端调用，继续调试，直到断点或返回上一层调用栈。
DBE_PLDEBUGGER.set_var	debug 端调用，为变量进行赋值操作。

19.4.1 DBE_PLDEBUGGER.turn_on

该函数用于标记某一存储过程为可调试，执行 turn_on 后 server 端可以执行该存储过程来进行调试。需要用户根据系统表 PG_PROC 手动获取存储过程 oid，传入函数中。turn_on 后本 session 内执行该存储过程会停在第一条 sql 前等待 debug 端的调试操作。该设置会在 session 断连后默认被清理掉。目前不支持对启用自治事务的存储过程/函数进行调试。

函数原型为：

```
DBE_PLDEBUGGER.turn_on(Oid)
RETURN Record;
```

表　turn_on 入参和返回值列表

名称	类型	描述

名称	类型	描述
func_oid	IN oid	函数 oid。
nodename	OUT text	节点名称。
port	OUT integer	连接端口号。

19.4.2 DBE_PLDEBUGGER.turn_off

用于去掉 turn_on 添加的调试标记，返回值表示成功或失败。可通过 DBE_PLDEBUGGER.local_debug_server_info 查找已经 turn_on 的存储过程 oid。

函数原型为：

```
DBE_PLDEBUGGER.turn_off(Oid)
RETURN boolean;
```

名称	类型	描述
func_oid	IN oid	函数 oid。
turn_off	OUT boolean	turn off 是否成功。

19.4.3 DBE_PLDEBUGGER.local_debug_server_info

用于查找当前连接中已经 turn_on 的存储过程 oid。便于用户确认在调试哪些存储过程，需要通过 funcoid 和 pg_proc 配合使用。

名称	类型	描述
nodename	OUT text	节点名称。
port	OUT bigint	端口号。
funcoid	OUT oid	存储过程 oid。

19.4.4 DBE_PLDEBUGGER.attach

server 端执行存储过程，停在第一条语句前，等待 debug 端关联。debug 端调用 attach，传入 nodename 和 port，关联到该存储过程上。

如果调试过程中报错，attach 会自动失效；如果调试过程中 attach 到其他存储过程上，当前 attach 的调试也会失效。

表 attach 入参和返回值列表

名称	类型	描述
nodename	IN text	节点名称。
port	IN integer	连接端口号。
funcoid	OUT oid	函数 id。
funcname	OUT text	函数名。
lineno	OUT integer	当前调试运行的下一行行号。
query	OUT text	当前调试的下一行函数源码。

19.4.5 DBE_PLDEBUGGER.next

执行存储过程中当前的 sql，返回执行的下一条的行数和对应 query。

表 next 返回值列表

名称	类型	描述
funcoid	OUT oid	函数 id。
funcname	OUT text	函数名。
lineno	OUT integer	当前调试运行的下一行行号。
query	OUT text	当前调试的下一行函数源码。

19.4.6 DBE_PLDEBUGGER.continue

执行当前存储过程，直到下一个断点或结束。返回值表示执行的下一条的行数和对应 query。

函数原型为：

```
DBE_PLDEBUGGER.continue()
RETURN Record;
```

表 continue 返回值列表

名称	类型	描述
funcoid	OUT oid	函数 id。
funcname	OUT text	函数名。
lineno	OUT integer	当前调试运行的下一行行号。
query	OUT text	当前调试的下一行函数源码。

19.4.7 DBE_PLDEBUGGER.abort

令 server 端执行的存储过程报错跳出。返回值表示是否成功发送 abort。

函数原型为：

```
DBE_PLDEBUGGER.abort()
RETURN boolean;
```

表 abort 返回值列表

名称	类型	描述
abort	OUT boolean	表示成功或失败。

19.4.8 DBE_PLDEBUGGER.print_var

debug 端调试过程中，调用 print_var，打印当前存储过程内变量中指定的变量名及其取值。该函数入参 frameno 表示查询遍历的栈层数，支持不加入该参数调用，缺省为查看最上层栈变量。

表 print_var 入参和返回值列表

名称	类型	描述

名称	类型	描述
var_name	IN text	变量。
frameno	IN integer（可选）	指定的栈层数，缺省为最顶层。
varname	OUT text	变量名。
vartype	OUT text	变量类型。
value	OUT text	变量值。
package_name	OUT text	变量对应的 package 名，预留使用，当前均为空。
isconst	OUT boolean	是否为常量。

19.4.9 DBE_PLDEBUGGER.info_code

debug 端调试过程中，调用 info_code，查看指定存储过程的源语句和各行对应的行号，行号从函数体开始，函数头部分行号为空。

表 info_code 入参和返回值列表

名称	类型	描述
funcoid	IN oid	函数 ID。
lineno	OUT integer	行号。
query	OUT text	源语句。
canbreak	OUT bool	当前行是否支持断点。

19.4.10DBE_PLDEBUGGER.step

debug 端调试过程中，如果当前执行的是一个存储过程，则进入该存储过程继续调试，返回该存储过程第一行的行号等信息，如果当前执行的不是存储过程，则和 next 行为一致，执行该 sql 后返回下一行的行号等信息。

表 step 入参和返回值列表

名称	类型	描述
funcoid	OUT oid	函数 id。
funcname	OUT text	函数名。
lineno	OUT integer	当前调试运行的下一行行号。
query	OUT text	当前调试的下一行函数源码。

19.4.11 DBE_PLDEBUGGER.delete_breakpoint

debug 端调试过程中，调用 delete_breakpoint 删除已有的断点。

表 delete_breakpoint 入参和返回值列表

名称	类型	描述
breakpointno	IN integer	断点编号。
result	OUT bool	是否成功。

19.4.12 DBE_PLDEBUGGER.info_breakpoints

debug 端调试过程中，调用 info_breakpoints，查看当前的函数断点。

表 info_breakpoints 返回值列表

名称	类型	描述
breakpointno	OUT integer	断点编号。
funcoid	OUT oid	函数 ID。
lineno	OUT integer	行号。
query	OUT text	断点内容。
enable	OUT boolean	是否有效

19.4.13 DBE_PLDEBUGGER.backtrace

debug 端调试过程中，调用 backtrace，查看当前的调用堆栈。

表 backtrace 返回值列表

名称	类型	描述
frameno	OUT integer	调用栈编号。
funcname	OUT oid	函数名。
lineno	OUT integer	行号。
query	OUT text	断点内容。
funcoid	OUT oid	函数 oid。

19.4.14DBE_PLDEBUGGER.enable_breakpoint

debug 端调试过程中，调用 enable_breakpoint 激活已被禁用的断点。

表 enable_breakpoint 入参和返回值列表

名称	类型	描述
breakpointno	IN integer	断点编号
result	OUT bool	是否成功

19.4.15DBE_PLDEBUGGER.add_breakpoint

debug 端调试过程中，调用 add_breakpoint 增加新的断点。如果返回-1 则说明指定的断点不合法，请参考 DBE_PLDEBUGGER.info_code 的 canbreak 字段确定断点合适的位置。

表 add_breakpoint 入参和返回值列表

名称	类型	描述
funcoid	IN text	函数 ID。
lineno	IN integer	行号。
breakpointno	OUT integer	断点编号。

19.4.16DBE_PLDEBUGGER.disable_breakpoint

debug 端调试过程中，调用 disable_breakpoint 禁用已被激活的断点。

表 disable_breakpoint 入参和返回值列表

名称	类型	描述
breakpointno	IN integer	断点编号
result	OUT bool	是否成功

19.4.17DBE_PLDEBUGGER.finish

执行存储过程中当前的 SQL 直到下一个断点触发或执行到上层栈的下一行。

表 finish 入参和返回值列表

名称	类型	描述
funcoid	OUT oid	函数 id。
funcname	OUT text	函数名。
lineno	OUT integer	当前调试运行的下一行行号。
query	OUT text	当前调试的下一行函数源码。

19.4.18DBE_PLDEBUGGER.set_var

将指定的调试的存储过程中最上层栈上的变量修改为入参的取值。如果存储过程中包含同名的变量，set_var 只支持第一个变量值的设置。

表 set_var 入参和返回值列表

名称	类型	描述
var_name	IN text	变量名。
value	IN text	修改值。
result	OUT boolean	结果，是否成功。

19.5 DB4AI Schema

DB4AI 模式在 AI 特性中主要是用来存储和管理数据集版本。模式中保存数据表的原始视图快照，每一个数据版本的更改记录以及版本快照的管理信息。模式面向普通用户，用户可在该模式下查找特性 DB4AI.SNAPSHOT 创建的快照版本信息。

19.5.1 DB4AI.SNAPSHOT

SNAPSHOT 表记录当前用户通过特性 DB4AI.SNAPSHOT 存储的快照。

名称	类型	描述	实例
id	bigint	当前快照的 ID。	1
parent_id	bigint	父快照的 ID。	0
matrix_id	bigint	CSS 模式下快照的矩阵 ID，否则为 NULL。	0
root_id	bigint	初始快照的 ID，通过 db4ai.create_snapshot()从操作数据构建。	0
schema	name	导出快照视图的模式。	public
name	name	快照的名称，包括版本后缀。	example0@1.1.0
owner	name	创建此快照的用户的名称。	nw
commands	text[]	记录如何从其根快照生成到此快照的 SQL 语句的完整列表。	{DELETE,“WHERE id > 7”}
comment	text	快照说明。	inherits from @1.0.0
published	boolean	TRUE，当且仅当快照当前已发布。	f
archived	boolean	TRUE，当且仅当快照当前已存档。	f
created	timestamp without time zone	快照创建日期的时间戳。	2021-08-25 10:59:52.955604

名称	类型	描述	实例
row_count	bigint	此快照中的数据行数。	8

19.5.2 DB4AI.CREATE_SNAPSHOT

CREATE_SNAPSHOT 是 DB4AI 特性用于创建快照的接口函数。通过语法 CREATE SNAPSHOT 调用。

参数	类型	描述
i_schema	IN NAME	快照存储的模式名字，默认值是当前用户或者 PUBLIC。
i_name	IN NAME	快照名称。
i_commands	IN TEXT[]	定义数据获取的 SQL 命令。
i_vers	IN NAME	版本后缀。
i_comment	IN TEXT	快照描述。
res	OUT db4ai.snapshot_name	结果。

19.5.3 DB4AI.CREATE_SNAPSHOT_INTERNAL

CREATE_SNAPSHOT_INTERNAL 是 db4ai.create_snapshot 函数的内置执行函数。函数存在信息校验，无法直接调用。

参数	类型	描述
s_id	IN BIGINT	快照 ID。
i_schema	IN NAME	快照存储的名字空间。
i_name	IN NAME	快照名称。

参数	类型	描述
i_commands	IN TEXT[]	定义数据获取的 SQL 命令。
i_comment	IN TEXT	快照描述。
i_owner	IN NAME	快照拥有者。

19.5.4 DB4AI.PREPARE_SNAPSHOT

PREPARE_SNAPSHOT 是 DB4AI 特性中数据准备模型训练和解释快照进行协作。快照为所有应用更改的数据和文档提供了完整的序列。通过语法 PREPARE SNAPSHOT 调用。

参数	类型	描述
i_schema	IN NAME	快照存储的模式名字，默认值是当前用户或者 PUBLIC。
i_parent	IN NAME	父快照名称。
i_commands	IN TEXT[]	定义快照修改的 DDL 和 DML 命令。
i_vers	IN NAME	版本后缀。
i_comment	IN TEXT	此数据策展单元的说明。
res	OUT db4ai.snapshot_name	结果。

19.5.5 DB4AI.PREPARE_SNAPSHOT_INTERNAL

PREPARE_SNAPSHOT_INTERNAL 是 db4ai.prepare_snapshot 函数的内置执行函数。函数存在信息校验，无法直接调用。

参数	类型	描述
s_id	IN BIGINT	快照 ID。

参数	类型	描述
p_id	IN BIGINT	父快照 ID。
m_id	IN BIGINT	矩阵 id。
r_id	IN BIGINT	根快照 ID。
i_schema	IN NAME	快照模式。
i_name	IN NAME	快照名称。
i_commands	IN TEXT[]	定义快照修改的 DDL 和 DML 命令。
i_comment	IN TEXT	快照描述。
i_owner	IN NAME	快照所有者。
i_idx	INOUT INT	exec_cmds 的索引。
i_exec_cmds	INOUT TEXT[]	用于执行的 DDL 和 DML。
i_mapping	IN NAME[]	将用户列映射到备份列；如果不为 NULL，则生成规则。

19.5.6 DB4AI.ARCHIVE_SNAPSHOT

ARCHIVE_SNAPSHOT 是 DB4AI 特性用于存档快照的接口函数。通过语法 ARCHIVE SNAPSHOT 调用。生效后的快照无法参数训练等任务。

参数	类型	描述
i_schema	IN NAME	快照存储的模式名字，默认值是当前用户
i_name	IN NAME	快照名称
res	OUT db4ai.snapshot_name	结果

19.5.7 DB4AI.PUBLISH_SNAPSHOT

PUBLISH_SNAPSHOT 是 DB4AI 特性用于发布快照的接口函数。通过语法 PUBLISH SNAPSHOT 调用。

参数	类型	描述
i_schema	IN NAME	快照存储的模式名字，默认值是当前用户或者 PUBLIC
i_name	IN NAME	快照名称
res	OUT db4ai.snapshot_name	结果

19.5.8 DB4AI.MANAGE_SNAPSHOT_INTERNAL

MANAGE_SNAPSHOT_INTERNAL 是 DB4AI.PUBLISH_SNAPSHOT 和 DB4AI.ARCHIVE_SNAPSHOT 函数的内置执行函数。函数存在信息校验，无法直接调用。

表 1 DB4AI.MANAGE_SNAPSHOT_INTERNAL 入参和返回值列表

参数	类型	描述
i_schema	IN NAME	快照存储的模式名字
i_name	IN NAME	快照名称
publish	IN BOOLEN	是否是发布状态
res	OUT db4ai.snapshot_name	结果

19.5.9 DB4AI.SAMPLE_SNAPSHOT

SAMPLE_SNAPSHOT 是 DB4AI 特性用于对基数据进行采样生成快照的接口函数。通过语法 SAMPLE SNAPSHOT 调用。

参数	类型	描述
i_schema	IN NAME	快照存储的模式名字

参数	类型	描述
i_parent	IN NAME	父快照名称
i_sample_infixes	IN NAME[]	示例快照名称中缀
i_sample_ratios	IN NUMBER[]	每个样本的大小，作为父集的比率
i_stratify	IN NAME[]	分层策略
i_sample_comments	IN TEXT[]	示例快照描述
res	OUT db4ai.snapshot_name	结果

19.5.10 DB4AI.PURGE_SNAPSHOT

PURGE_SNAPSHOT 是 DB4AI 特性用于删除快照的接口函数。通过语法 PURGE SNAPSHOT 调用。

参数	类型	描述
i_schema	IN NAME	快照存储的模式名字
i_name	IN NAME	快照名称
res	OUT db4ai.snapshot_name	结果

19.5.11 DB4AI.PURGE_SNAPSHOT_INTERNAL

PURGE_SNAPSHOT_INTERNAL 是 DB4AI.PURGE_SNAPSHOT 函数的内置执行函数。函数存在信息校验，无法直接调用

参数	类型	描述
i_schema	IN NAME	快照存储的模式名字
i_name	IN NAME	快照名称

19.6 DBE_PLDEVELOPER

DBE_PLDEVELOPER 下系统表用于记录 PLPGSQL 包、函数及存储过程编译过程中需要记录的信息。

19.6.1 DBE_PLDEVELOPER.gs_source

用于记录 PLPGSQL 对象（存储过程、函数、包、包体）编译相关信息，具体内容见下列字段描述。

打开 plsql_show_all_error 参数后，会把成功或失败的 PLPGSQL 对象编译信息记录在此表中，如果关闭 plsql_show_all_error 参数则只会将正确的编译相关信息插入此表中。

表 1 DBE_PLDEVELOPER.gs_source 字段

名称	类型	描述
id	oid	对象的 ID。
owner	bigint	对象创建用户 ID。
nspid	oid	对象的模式 ID。
name	name	对象名。
type	text	对象类型（procedure/function/package/package body）。
status	boolean	是否创建成功。
src	text	对象创建的原始语句。

19.6.2 DBE_PLDEVELOPER.gs_errors

用于记录 PLPGSQL 对象（存储过程、函数、包、包体）编译过程中遇到的报错信息，具体内容见下列字段描述。

打开 plsql_show_all_error 参数后，如果编译过程中存在报错，则会跳过报错继续编译并把报错信息记录在 gs_errors 中，如果关闭 plsql_show_all_error 参数则不会将相关信息插入此表中。

名称	类型	描述
id	oid	对象的 ID。
owner	bigint	对象创建用户 ID。
nspid	oid	对象的模式 ID。
name	name	对象名。
type	text	对象类型（procedure/function/package/package body）。
line	integer	行号。
src	text	对象创建的原始语句。

20 逻辑复制

20.1 逻辑解码

20.1.1 逻辑解码概述

功能描述

GBase 8c 对数据复制能力的支持情况为：

支持通过数据迁移工具定期向异构数据库（如 Oracle 等）进行数据同步，不具备实时数据复制能力。不足以支撑与异构数据库间并网运行实时数据同步的诉求。

GBase 8c 提供了逻辑解码功能，通过反解 xlog 的方式生成逻辑日志。目标数据库解析逻辑日志以实时进行数据复制。具体如图 21-1 所示。逻辑复制降低了对目标数据库的形态限制，支持异构数据库、同构异形数据库对数据的同步，支持目标库进行数据同步期间的数据可读写，数据同步时延低。

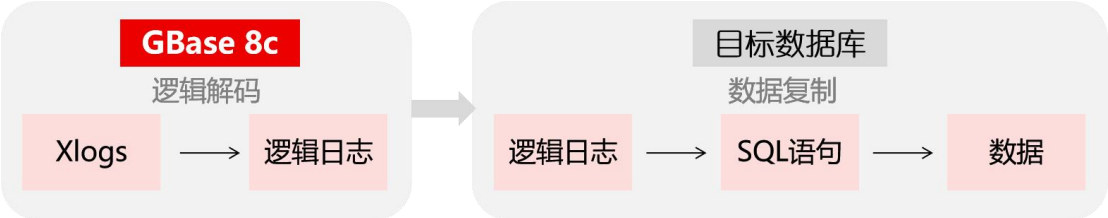


图 21-1 逻辑复制

逻辑复制由两部分组成：逻辑解码和数据复制。逻辑解码会输出以事务为单位组织的逻辑日志。业务或数据库中间件将会对逻辑日志进行解析并最终实现数据复制。GBase 8c 当前只提供逻辑解码功能，因此本章节只涉及逻辑解码的说明。

逻辑解码为逻辑复制提供事务解码的基础能力，GBase 8c 使用 SQL 函数接口进行逻辑解码。此方法调用方便，不需使用工具，对接外部工具接口也比较清晰，不需要额外适配。

由于逻辑日志是以事务为单位的，在事务提交后才能输出，且逻辑解码是由用户驱动的；因此为了防止事务开始时的 xlog 被系统回收，或所需的事务信息被 VACUUM 回收，GBase 8c 新增了逻辑复制槽，用于阻塞 xlog 的回收。

一个逻辑复制槽表示一个更改流，这些更改可以在其它数据库中以它们在原数据库上产生的顺序被重播。逻辑复制槽，由每个逻辑日志的获取者维护一个。

注意事项

- 不支持 DDL 语句解码，在执行特定的 DDL 语句（例如普通表 truncate 或分区表 exchange）时，可能造成解码数据丢失。
- 不支持列存、数据页复制的解码。
- 不支持级联备机进行逻辑解码。
- 当执行 DDL 语句（如 alter table）后，该 DDL 语句前尚未解码的物理日志可能会丢失。
- 单条元组大小不超过 1GB，考虑解码结果可能大于插入数据，因此建议单条元组大小不超过 500MB。
- GBase 8c 支持解码的数据类型为：INTEGER、BIGINT、SMALLINT、TINYINT、SERIAL、SMALLSERIAL、BIGSERIAL、FLOAT、DOUBLE PRECISION、DATE、TIME[WITHOUT

TIME ZONE]、TIMESTAMP[WITHOUT TIME ZONE]、CHAR(n)、VARCHAR(n)、TEXT。

- 如果需要 ssl 连接需要保证前置设置 GUC 参数 ssl=on。
- 逻辑复制槽名称必须小于 64 个字符，且只包含小写字母、数字或者下划线中的一种或几种。
- 当前逻辑复制不支持 MOT 特性。
- 当逻辑复制槽所在数据库被删除后，这些复制槽变为不可用状态，需要用户手动删除。
- 仅支持 utf-8 字符集。
- 对多库的解码需要分别在库内创建流复制槽并开始解码，每个库的解码都需要单独扫一遍日志。
- 不支持强起，强起后需要重新全量导出数据。
- 备机解码时，switchover 和 failover 时可能出现解码数据变多，需用户手动过滤。Qurem 协议下，switchover 和 failover 选择升主的备机，需要与当前主机日志同步。
- 不允许主备，多个备机同时使用同一个复制槽解码，否则会产生数据不一致。
- 只支持主机创建删除复制槽。
- 数据库故障重启或逻辑复制进程重启后，解码数据存在重复，用户需自己过滤。
- 计算机内核故障后，解码存在乱码，需手动或自动过滤。
- 当前备机逻辑解码，不支持开启极致 RTO。
- 请确保在创建逻辑复制槽过程中长事务未启动，启动长事务会阻塞逻辑复制槽的创建。
- 不支持 interval partition 表复制。
- 不支持全局临时表。
- 在事务中执行 DDL 语句后，该 DDL 语句与之后的语句不会被解码。
- 如需进行备机解码，需在对应主机上设置 guc 参数 enable_slot_log = on。
- 禁止在使用逻辑复制槽时在其他节点对该复制槽进行操作，删除复制槽进行操作的操作

需在该复制槽停止解码后执行。

20.1.2 使用 SQL 函数接口进行逻辑解码

GBase 8c 可以通过调用 SQL 函数，进行创建、删除、推进逻辑复制槽，获取解码后的事务日志。

前提条件

- 逻辑日志目前从主机节点中抽取，默认关闭 SSL 连接，如果进行逻辑复制，需要先上配置 GUC 参数 ssl=on。

  说明： 为避免安全风险，请保证启用 SSL 连接。

- 设置 GUC 参数 wal_level=logical。
- 设置 GUC 参数 max_replication_slots>每个节点所需的(物理流复制槽数+逻辑复制槽数)。

物理流复制槽提供了一种自动化的方法来确保主节点在所有备节点或从备节点收到 xlog 之前，xlog 不会被移除。也就是说物理流复制槽用于支撑主备节点 HA。数据库所需要的物理流复制槽数为：备节点加从备的和与主节点之间的比例。例如，假设数据库的高可用方案为 1 主、1 备、1 从备，则所需物理流复制槽数为 2。

关于逻辑复制槽数，请按如下规则考虑。

  - 一个逻辑复制槽只能解码一个 Database 的修改，如果需要解码多个 Database，则需要创建多个逻辑复制槽。
  - 如果需要多路逻辑复制同步给多个目标数据库，在源端数据库需要创建多个逻辑复制槽，每个逻辑复制槽对应一条逻辑复制链路。

- 仅限初始用户和拥有 REPLICATION 权限的用户进行操作。三权分立关闭时数据库管理员可进行逻辑复制操作，三权分立开启时不允许数据库管理员进行逻辑复制操作。
- 目前默认不支持主备从部署模式。

操作步骤

1. 以数据库安装用户登录 GBase 8c 数据库主节点。

2. 使用如下命令通过连接默认数据库 postgres。

```
gsql -d postgres -p 16000 -r
```

其中，16000 为数据库端口号，用户可根据实际情况替换。

3. 创建名称为 slot1 的逻辑复制槽。

```
postgres=# SELECT * FROM pg_create_logical_replication_slot('slot1', 'mppdb_decoding');
slotname | xlog_position
----------+---------------
slot1     | 0/601C150
(1 row)
```

4. 在数据库中创建表 t，并向表 t 中插入数据。

```
postgres=# CREATE TABLE t(a int PRIMARY KEY, b int);
postgres=# INSERT INTO t VALUES(3,3);
```

5. 读取复制槽 slot1 解码结果，解码条数为 4096。

```
postgres=# SELECT * FROM pg_logical_slot_peek_changes('slot1', NULL, 4096);
location   |   xid   | data
-----------+-------+------------------------------------------------------------------------------------------------------------------------------------------
-----------------------------------------
 0/601C188 | 1010023 | BEGIN 1010023
 0/601ED60 | 1010023 | COMMIT 1010023 CSN 1010022
 0/601ED60 | 1010024 | BEGIN 1010024
 0/601ED60 | 1010024 |
{"table_name":"public.t","op_type":"INSERT","columns_name":["a","b"],"columns_type":["integ
er","integer"],"columns_val":["3","3"],"old_keys_name":[],"old_keys_type":[],"old_keys_val":[]}
 0/601EED8 | 1010024 | COMMIT 1010024 CSN 1010023
(5 rows)
```

6. 删除逻辑复制槽 slot1。

```
postgres=#    SELECT * FROM pg_drop_replication_slot('slot1');
 pg_drop_replication_slot
--------------------------


(1 row)
```

20.2 发布订阅

发布和订阅基于逻辑复制实现，其中有一个或者更多订阅者订阅一个发布者节点上的一个或者更多发布。订阅者从它们所订阅的发布拉取数据。

发布者上的更改会被实时发送给订阅者。订阅者以与发布者相同的顺序应用那些数据，这样在一个订阅中能够保证发布的事务一致性。这种数据复制的方法有时候也被称为事务性复制。

发布订阅的典型用法是：

在一个数据库或者一个数据库的子集中发生更改时，把增量的改变发送给订阅者。

在更改到达订阅者时引发触发器。

把多个数据库联合到单一数据库中（例如用于分析目的）。

订阅者数据库的行为与任何其他 GBase 8c 实例相同，并且可以被用作其他数据库的发布者，只需要定义它自己的发布。当订阅者被应用当作只读时，单一的订阅中不会有冲突。在另一方面，如果应用或者对相同表集合的订阅者执行了其他的写动作，冲突可能会发生。

20.2.1 发布

发布可以被定义在任何物理复制的主服务器上。定义有发布的节点被称为发布者。发布是从一个表或者一组表生成的改变的集合，也可以被描述为更改集合或者复制集合。每个发布都只存在于一个数据库中。

发布与模式不同，不会影响表的访问方式。如果需要，每个表都可以被加入到多个发布。当前，发布只能包含表。对象必须被明确地加入到发布，除非发布是用 ALL TABLES 创建的。

发布可以选择把它们产生的更改限制为 INSERT、UPDATE、DELETE 的任意组合，类似于触发器如何被特定事件类型触发的方式。默认情况下，所有操作类型都会被复制。

为了能够复制 UPDATE 和 DELETE 操作，被发布的表必须配置有一个“复制标识”，这样在订阅者那一端才能标识对于更新或删除合适的行。默认情况下，复制标识就是主键（如果有主键）。也可以在复制标识上设置另一个唯一索引（有特定的额外要求）。如果表没有合适的键，那么可以设置成复制标识“full”，它表示整个行都成为那个键。不过，这样做效率很低，只有在没有其他方案的情况下才应该使用。如果在发布者端设置了“full”之外的复制标识，在订阅者端也必须设置一个复制标识，它应该由相同的或者少一些的列组成。如何设置复制标识的细节请参考 REPLICA IDENTITY。如果在复制 UPDATE 或 DELETE 操作的发布中加入了没有复制标识的表，那么订阅者上后续的 UPDATE 或 DELETE 操作将导致错误。不管有没有复制标识，INSERT 操作都能继续下去。

每一个发布都可以有多个订阅者。

Publication 通过使用 CREATE PUBLICATION 命令创建并且可以在之后使用相应的命令进行修改或者删除。

表可以使用 ALTER PUBLICATION 动态地增加或者移除。ADD TABLE 以及 DROP TABLE 操作都是事务性的，因此一旦该事务提交，该表将以正确的快照开始或者停止复制。

20.2.2 订阅

订阅是逻辑复制的下游端。订阅被定义在其中的节点被称为订阅者。一个订阅会定义到另一个数据库的连接以及它想要订阅的发布集合（一个或者多个）。

订阅者数据库的行为与任何其他 GBase 8c 实例相同，并且可以被用作其他数据库的发布者，只需要定义它自己的发布。

如果需要，一个订阅者节点可以有多个订阅。可以在一对发布者-订阅者之间定义多个订阅，在这种情况下要确保被订阅的发布对象不会重叠。

每一个订阅都将通过一个复制槽接收更改。预先存在的表的初始数据暂时不支持同步。

如果当前用户是一个具有 SYSADMIN 权限用户，则订阅会被 gs_dump 转储。否则订阅会被跳过并且写出一个警告，因为不具有 SYSADMIN 权限用户不能从 pg_subscription 目录中读取所有的订阅信息。

可以使用 CREATE SUBSCRIPTION 增加订阅，并且使用 ALTER SUBSCRIPTION 在任何时刻修改订阅，还可以使用 DROP SUBSCRIPTION 删除订阅。

在一个订阅被删除并且重建时，同步信息会丢失。这意味着数据必须被重新同步。

模式定义不会被复制，并且被发布的表必须在订阅者上存在。只有常规表可以成为复制的目标。例如，不能复制视图。

表在发布者和订阅者之间使用完全限定的表名进行匹配。不支持复制到订阅者上命名不同的表。

表的列也通过名称匹配。订阅表中的列顺序不需要与发布表中的顺序一样。 列的数据类型也不需要一样，只要可以将数据的文本表示形式转换为目标类型即可。 例如，您可以从 integer 类型的列复制到 bigint 类型的列。 目标表还可以具有发布表中不存在的额外列。额外列都将使用目标表的定义中指定的默认值填充。

20.2.3 冲突处理

逻辑复制的行为类似于正常的 DML 操作，即便数据在订阅者节点本地被修改，逻辑复制也会根据收到的更改来更新数据。如果流入的数据违背了任何约束，复制将停止。这种情况被称为一个冲突。在复制 UPDATE 或 DELETE 操作时，缺失的数据将不会产生冲突并且这类操作将被简单地跳过。

冲突将会产生错误并且停止复制，它必须由用户手工解决。在订阅者的服务器日志中可以找到有关冲突的详细情况。

通过更改订阅者上的数据（这样它就不会与到来的数据发生冲突）或者跳过与已有数据冲突的事务可以解决这种冲突。通过调用 pg_replication_origin_advance()函数可以跳过该事务，函数的参数是对应于该订阅名称的 node_name 以及一个位置。复制源头的当前位置可以在 pg_replication_origin_status 系统视图中看到。

20.2.4 限制

发布订阅基于逻辑复制实现，继承所有逻辑复制的限制，同时发布订阅还有下列额外的限制或者缺失的功能。

- 数据库模式和 DDL 命令不会被复制。初始模式可以手工使用 gs_dump –schema-only 进行拷贝。后续的模式改变需要手工保持同步。
- 序列数据不被复制。后台由序列支撑的 serial 或者标识列中的数据当然将被作为表的一部分复制，但是序列本身在订阅者上仍将显示开始值。如果订阅者被用作一个只读数据库，那么这通常不会是什么问题。不过，如果订阅者数据库预期有某种转换或者容错，那么序列需要被更新到最后的值，要么通过从发布者拷贝当前数据的防范（也许使用 gs_dump），要么从表本身决定一个足够高的值。
- 只有表支持复制，包括分区表。试图复制其他类型的关系，例如视图、物化视图或外部表，将会导致错误。
- 同一数据库内的多个订阅不应当订阅内容重复的发布（指发布相同的表），否则会产生数据重复或者主键冲突。
- 如果被发布的表中包含不支持 btree/hash 索引的数据类型（如地理类型等），那么该表需要有主键，才能成功的复制 UPDATE/DELETE 操作到订阅端。否则复制会失败，同时订阅端会出现“FATAL: could not identify an equality operator for type xx”的日志。
- 使用 gs_probackup 备份发布端时，由于 gs_probackup 工具本身不支持备份逻辑复制槽，所以备份恢复后，会由于逻辑复制槽不存在，而使得原有的发布订阅关系无法正常建立。建议使用 gs_basebackup 工具备份发布端。

20.2.5 架构

发布者上的更改会在它们发生时实时传送给订阅者。订阅者按照数据在发布者上被提交的顺序应用数据，这样任意单一订阅中的发布的事务一致性才能得到保证。

逻辑复制被构建在一种类似于物理流复制的架构上。它由“walsender”和“apply”进程实现。

walsender 进程开始对 WAL 的逻辑解码并且载入标准逻辑解码插件（pgoutput）。该插件把从 WAL 中读取的更改转换成逻辑复制协议并且根据发布说明过滤数据。然后数据会被连续地使用流复制协议传输到应用工作者，应用工作者会把数据映射到本地表并且以正确的事务顺序应用它们接收到的更改

订阅者数据库上的应用进程总是将 session_replication_role 设置为 replica 运行，这会产生触发器和约束上通常的效果。

逻辑复制应用进程当前仅会引发行触发器，而不会引发语句触发器。不过，初始的表同步是以类似一个 COPY 命令的方式实现的，因此会引发 INSERT 的行触发器和语句触发器。

20.2.6 监控

因为逻辑复制是基于与物理流复制相似的架构的，一个发布节点上的监控也类似于对物理复制主节点的监控。

有关订阅的监控信息在 pg_stat_subscription 中可以看到。 每一个订阅工作者在这个视图都有一行。一个订阅能有零个或者多个活跃订阅工作者取决于它的状态。

通常，对于一个已启用的订阅会有单一的应用进程运行。一个被禁用的订阅或者崩溃的订阅在这个视图中不会有行存在。如果有任何表的数据同步正在进行，对正在被同步的表会有额外的工作者。

20.2.7 安全性

用于复制连接的角色必须具有 REPLICATION 属性(或者是具有 SYSADMIN 权限用户)。如果角色缺少 SUPERUSER 和 BYPASSRLS，发布者的行安全策略可以执行。 角色的访问权限必须在 pg_hba.conf 中配置，并且必须具有 LOGIN 属性。

要创建发布，用户必须在数据库中有 CREATE 特权。

要把表加入到一个发布，用户必须在该表上有拥有权。要创建一个自动发布所有表的发布，用户必须是一个具有 SYSADMIN 权限用户。

要创建订阅，用户必须是一个具有 SYSADMIN 权限用户。

订阅的应用过程将在本地数据库上以具有 SYSADMIN 权限用户的特权运行。

特权检查仅在复制连接开始时被执行一次。在从发布者读到每一个更改记录时不会重新检查特权，在每一个更改被应用时也不会重新检查特权。

20.2.8 配置设置

发布订阅要求设置一些配置选项。

在发布者端，wal_level 必须被设置为 logical，而 max_replication_slots 中设置的值必须至少是预期要连接的订阅数。max_wal_senders 应该至少被设置为 max_replication_slots 加上同时连接的物理复制体的数量。

订阅者还要求 max_replication_slots 被设置。在这种情况下，它必须至少被设置为将被加入到该订阅者的订阅数。max_logical_replication_workers 必须至少被设置为订阅数，

20.2.9 快速设置

首先在 postgresql.conf 中设置配置选项：

```
wal_level = logical
```

对于一个基础设置来说，其他所需的设置使用默认值就足够了。

需要调整 pg_hba.conf 以允许复制（这里的值取决于实际的网络配置以及用于连接的用户）：

```
host        all        repuser        0.0.0.0/0        sha256
```

然后在发布者数据库上：

```
CREATE PUBLICATION mypub FOR TABLE users, departments;
```

并且在订阅者数据库上：

```
CREATE SUBSCRIPTION mysub CONNECTION 'dbname=foo host=bar user=repuser' PUBLICATION mypub;
```

上面的语句将开始复制过程，复制对那些表的增量更改。

21 Foreign Data Wrapper

GBase 8c 的 FDW（Foreign Data Wrapper）可以实现各个 GBase 8c 数据库及远程服务器（包括数据库、文件系统）之间的跨库操作。目前支持的远程数据库类型包括 Oracle、MySQL（MariaDB）、GBase 8c（postgres_fdw）、openGauss（postgres_fdw）、file_fdw 、dblink。

21.1 oracle_fdw

oracle_fdw(foreign data wrapper for oracle)用于 Oracle 的外部数据包装器，可以使 GBase 8c 轻松跨库操作 Oracle 中的对象。

21.1.1 客户端配置

(1) 选择合适的运行环境和版本，下载 Basic Package、SDK package、SQLPlus Package 对应版本的安装包，三者版本号必须一致。

(2) 传至数据库节点（分布式场景下读写 CN 节点，主备式场景下主节点），并以 gbase 用户身份解压文件。例如，在/home/gbase.tools/oracle_fdw 路径下解压：

```
[root@gbasehost oracle_fdw]# su gbase
[gbase@gbasehost oracle_fdw]$ unzip
instanclient-basic-linux.x86-21.5.0.0.0dbru.zip
[gbase@gbasehost oracle_fdw]$ unzip
instanclient-sdk-linux.x86-21.5.0.0.0dbru.zip
[gbase@gbasehost oracle_fdw]$ unzip
instanclient-sqlplus-linux.x86-21.5.0.0.0dbru.zip
```

(3) 安装后配置环境变量，即可使用 oracle_fdw。根据实际情况修改 ORACLE_HOME 具体目录。例如：

```
[gbase@gbasehost oracle_fdw]$ export
ORACLE_HOME=/home/gbase/tools/oracle_fdw/instantclient_21_5
[gbase@gbasehost oracle_fdw]$ export
LD_LIBRARY_PATH=$ORACLE_HOME:$LD_LIBRARY_PATH
[gbase@gbasehost oracle_fdw]$ export TNS_ADMIN=$ORACLE_HOME/network/admin
```

(4) 执行以下命令，使环境变量生效。

```
[gbase@gbasehost oracle_fdw]$ source ~/.bashrc
```

(5) 通过客户端连接 Oracle Server，进行配置。

```
[gbase@gbasehost oracle_fdw]$ sqlplus
```

进入 Oracle 客户端目录的 network/admin 目录，添加 tnsnames.ora 文件。

```
[gbase@gbasehost oracle_fdw]$ touch tnsnames.ora
[gbase@gbasehost oracle_fdw]$ vim tnsnames.ora
```

新增文件具体内容如下（host、port、sid、instance_name 请以实际情况为准）：

```
orcl =
   (DESCRIPTION =
      (ADDRESS_LIST =
         (ADDRESS = (PROTOCOL = TCP) (HOST = ip) (PORT = port))
      )
      (CONNECT_DATA =
         (SID = sid)
         (SERVER = DEDICATED)
         (INSTANCE_NAME = instance_name)
      )
   )
```

在 Oracle Server 端添加当前机器的访问权限，进入 product/network/admin 目录，修改 tnsname.ora 配置文件，添加内容如下：

```
    (ADDRESS = (PROTOCOL = TCP) (HOST = ip) (PORT = port))
    (CONNECT_DATA = (SERVER = DEDICATED) (SERVER_NAME = sid)
```

(6) 切换 root 用户，进行如下操作（分布式、主备式每个节点机器都需配置）：

```
[gbase@gbasehost oracle_fdw]$ su root
[root@gbasehost oracle_fdw]# yum install -y numactl
[root@gbasehost oracle_fdw]# yum install -y ndctl
```

其中 Oracle 客户端基础包内含 libclntsh.so.21.1、libnnz21.so、libclntshcore.so.21.1、libociei.so 四个依赖包，需要将其复制到/lib64 目录下。并重启数据库生效。

```
[root@gbasehost oracle_fdw]# cp libclntsh.so.21.1 libnnz21.so
libclntshcore.so.21.1 libociei.so /lib64
[root@gbasehost oracle_fdw]# su gbase
[gbase@gbasehost oracle_fdw]$ gha_ctl stop all -l dcslist
[gbase@gbasehost oracle_fdw]$ gha_ctl start all -l dcslist
```

21.1.2 使用 oracle_fdw

使用 oracle_fdw 前需要完成以上客户端配置。

- 创建 oracle_fdw 插件

```
CREATE EXTENSION oracle_fdw;
```

- 创建使用用户并授权

```
CREATE USER user_name IDENTIFIED BY 'password';
GRANT USAGE ON FOREIGN DATA WRAPPER oracle_fdw TO user_name;
```

- 创建服务器对象

```
CREATE SERVER server_name foreign data wrapper oracle_fdw options(dbserver
'host_ip:port/servername');
```

- 创建用户映射

```
CREATE USER MAPPING for user_name server server_name options(user
'user',password 'password');
```

- 创建外部表

```
CREATE FOREIGN TABLE table_name(col_name col_type) server server_name
options(schema 'schema_name',table 'table', prefetch 'value');
```

其中，外表的表结构需要与 Oracle 数据库中的表结构保持一致。注意 Oracle server 侧的表的第一个字段必须具有唯一性约束（如 PRIMARY KEY、UNIQUE 等）。

- 对外表做正常的操作，如 INSERT、UPDATE、DELETE、SELECT、EXPLAIN、ANALYZE、COPY 等。
- 删除外表

```
DROP FOREIGN TABLE table_name;
```

- 删除用户映射

```
DROP USER MAPPING;
```

- 删除服务器对象

```
DROP SERVER server_name CASCADE;
```

- 删除扩展

```
DROP EXTENSION oracle_fdw;
```

21.1.3 常见问题

- 在 GBase 8c 上建立外表时，不会在 Oracle 数据库中同步建表，需要自行在 Oracle 数据库中建表。
- 执行 CREATE USER MAPPING 时使用的 Oracle 用户需要有远程连接 Oracle 数据库及对表相关操作的权限。使用外表前，可以在 GBase 8c server 所在的机器上，使用 Oracle 的客户端，使用对应的用户名密码确认能否成功连接 Oracle 并进行操作。
- 执行 CREATE EXTENSION oracle_fdw;时，报错显示类似“ERROR: could not load library “/home/gbase/GBase8cV5_S3.0.1B12/lib/……oracle_fdw.so”: libclntsh.so.21.1: cannot open shared objeck file: No such file or directory”。原因是 Oracle 的开发库 libclntsh.so 不在系统的相关路径中，可以进入客户端目录找到 libclntsh.so 的具体路径，然后将其中包含所需要的依赖包放入每个节点的/lib64 目录下。
- 查询外部表时报错显示类似“ERROR : error connecting ro Oracle: OCIEnvCreate failed to create enviroment handle”信息。原因可能是缺少 libociei.so、libnuma.so、libndctl.so。解决方法是将客户端下 libociei.so 依赖包复制到/lib64 路径下；libnuma.so 则需要通过下载 numactl，yum install -y numactl；libndctl.so 需要通过下载 ndctl，yum install -y ndctl，并重启数据库生效。

21.1.4 注意事项

- 两个 Oracle 外表间的 SELECT JOIN 不支持下推到 Oracle server 执行，会被分成两条 SQL 语句传递到 Oracle 执行，然后在 GBase 8c 处汇总处理结果。
- 不支持 IMPORT FOREIGN SCHEMA 语法。
- 不支持对外表进行 CREATE TRIGGER 操作。

21.2 mysql_fdw

mysql_fdw（foreign data wrapper for mysql）GBase 8c 用于 Oracle 的外部数据包装器。

21.2.1 安装 mysql_fdw

mysql_fdw 同样已集成在 GBase 8c 的安装包中。支持 MariaDB 环境，需要安装客户端，并配置环境变量即可。

(1) 选择合适的运行环境和版本，下载对应安装包，并通过执行 rpm -ivh 命令安装对应文件。MariaDB-client 是 MySQL/MariaDB 的客户端工具，也可以根据需要安装，用于连接 MySQL/MariaDB 进行测试。

(2) 安装后配置环境变量，即可使用 mysql_fdw。例如：

```
export LD_LIBRARY_PATH=/usr/lib/mysql/lib:LD_LIBRARY_PATH
```

(3) 执行 source ~/.bashrc 使环境变量生效。

(4) 创建用于访问 mysql_fdw 的用户

```
gs_guc generate -U user_name -S password -D dir -o usermapping
```

21.2.2 使用 mysql_fdw

使用 mysql_fdw 需要连接 MySQL/MariaDB。MySQL 数据库和实例请自行安装准备。

- 加载 mysql_fdw 扩展

```
CREATE EXTENSION mysql_fdw;
```

- 授予用户所有权限

```
GRANT ALL PRIVILEGES TO user_name;
```

- 授予用户访问 mysql_fdw 权限

```
GRANT USAGE ON FOREIGN DATA WRAPPER MYSQL_FDW TO user_name;
```

- 创建服务器对象

```
CREATE SERVER server_name FOREIGN DATA WRAPPER MYSQL_FDW OPTIONS (host 'host_ip', port 'port');
```

- 创建用户映射

```
CREATE USER MAPPING FOR user_name server server_name options (username 'user', password 'password');
```

- 创建外表

```
CREATE FOREIGN TABLE table_name(col_name col_type) server server_name options(dbname 'dbname', table_name 'table_name');
```

其中，外表的表结构需要与 MySQL/MariaDB 侧的表结构保持一致。注意 MySQL/MariaDB 侧的表的第一个字段必须具有唯一性约束（如 PRIMARY KEY、UNIQUE 等）。

- 对外表做正常的操作，如 INSERT、UPDATE、DELETE、SELECT、EXPLAIN、ANALYZE、

COPY 等。

- 删除外表

```
DROP FOREIGN TABLE table_name;
```

- 删除用户映射

```
DROP USER MAPPING;
```

- 删除服务器对象

```
DROP SERVER server_name;
```

- 删除扩展

```
DROP EXTENSION mysql_fdw;
```

21.2.3 常见问题

- 在 GBase 8c 上建立外表时，不会同步在 MariaDB/MySQL Server 上建表，需要自己利用 MariaDB/MySQL Server 的客户端连接 MariaDB/MySQL Server 建表。
- 创建 USER MAPPING 时使用的 MariaDB/MySQL Server 用户需要有远程连接 MariaDB/MySQL Server 及对表相关操作的权限。使用外表前，可以在 GBase 8c server 所在的机器上，使用 MariaDB/MySQL Server 的客户端，使用对应的用户名密码确认能否成功连接 MariaDB/MySQL Server 并进行操作。
- 对外表执行 DML 操作时，出现 Can't initialize character set SQL_ASCII (path: compiled_in) 错误。由于 MariaDB 不支持 SQL_ASCII 编码格式，目前只能通过修改 GBase 8c database 的编码格式解决该问题。修改 database 编码格式的方式为 update pg_database set encoding = pg_char_to_encoding('UTF-8') where datname = 'postgres'; datname 根据实际情况填写。注意修改完编码格式后，需要重新开启一个 gsql 会话，才能使 mysql_fdw 使用更新后的参数。也可以通过在执行 gs_initdb 时，使用–locale=LOCALE，指定默认的编码格式为非 SQL_ASCII 编码。

21.2.4 注意事项

- 两个 mysql 外表间的 SELECT JOIN 不支持下推到 MariaDB/MySQL Server 执行，会被分成两条 SQL 语句传递到 MariaDB/MySQL Server 执行，然后在 GBase 8c 处汇总处理结果。
- 不支持 IMPORT FOREIGN SCHEMA 语法。

- 不支持对外表进行 CREATE TRIGGER 操作。

21.3 postgres_fdw

postgres_fdw 是一款开源插件，其代码随 PostgreSQL 源码一同发布。GBase 8c 基于 postgres_fdw 源码进行开发适配。

postgres_fdw 插件默认参与编译，使用安装包安装好 GBase 8c 后，可直接使用无须其他操作。

21.3.1 使用 postgres_fdw

- 加载 postgres_fdw 扩展

```
CREATE EXTENSION postgres_fdw;
```

- 授予用户 postgres_fdw 的使用权限

```
GRANT USAGE ON FOREIGN DATA WRAPPER postgres_fdw TO user_name;
```

- 创建服务器对象

```
CREATE SERVER server_name FOREIGN DATA WRAPPER postgres_fdw OPTIONS (host 'host_ip', port 'port', dbname 'dbname');
```

- 授予用户外部服务器的使用权限

```
GRANT USAGE ON FOREIGN SERVER server_name TO user_name;
```

- 创建用户映射

```
CREATE USER MAPPING FOR user_name SERVER server_name OPTIONS (user 'user', password 'password');
```

- 创建外表

```
CREATE FOREIGN TABLE table_name (col_name col_type) SERVER server_name OPTIONS (schema_name 'schema', table_name 'table');
```

其中，外表的表结构需要与远端 GBase 8c 侧的表结构保持一致。

- 对外表做正常的操作，如 INSERT、UPDATE、DELETE、SELECT、EXPLAIN、ANALYZE、COPY 等。

- 删除外表

```
DROP FOREIGN TABLE table_name;
```

- 删除用户映射

```
DROP USER MAPPING;
```

- 删除服务器对象

```
DROP SERVER server_name;
```

- 删除扩展

```
DROP EXTENSION postgres_fdw;
```

21.3.2 常见问题

- 在 GBase 8c 上建立外表时，不会同步在远端的 GBase 8c 上建表，需要自己利用客户端连接远端 GBase 8c 建表。
- 执行 CREATE USER MAPPING 时使用的 GBase 8c 用户需要有远程连接 GBase 8c 及对表相关操作的权限。使用外表前，可以在本地机器上，使用 gsql 的客户端，使用对应的用户名密码确认能否成功连接远端 GBase 8c 并进行操作。

21.3.3 注意事项

- 两个 postgres_fdw 外表间的 SELECT JOIN 不支持下推到远端 GBase 8c 执行，会被分成两条 SQL 语句传递到远端 GBase 8c 执行，然后在本地汇总处理结果。
- 不支持 IMPORT FOREIGN SCHEMA 语法。
- 不支持对外表进行 CREATE TRIGGER 操作。

21.4 file_fdw

file_fdw 模块提供了外部数据封装器 file_fdw，可以用来在服务器的文件系统中访问数据文件。数据文件必须是 COPY FROM 可读的格式；具体可参照 COPY 语句的介绍。访问这样的数据文件当前只是可读的。当前不支持对该数据文件的写入操作。

当前 GBase 8c 会默认编译 file_fdw，在 initdb 的时候会在 pg_catalog schema 中创建该插

件。

使用 file_fdw 创建的外部表可以有下列选项：

- **filename**
  指定要读取的文件，必需的参数，且必须是一个绝对路径名。
- **format**
  远端 server 的文件格式，支持 text/csv/binary 三种格式，和 COPY 语句的 FORMAT 选项相同。
- **header**
  指定的文件是否有标题行，与 COPY 语句的 HEADER 选项相同。
- **delimiter**
  指定文件的分隔符，与 COPY 的 DELIMITER 选项相同。
- **quote**
  指定文件的引用字符，与 COPY 的 QUOTE 选项相同。
- **escape**
  指定文件的转义字符，与 COPY 的 ESCAPE 选项相同。
- **null**
  指定文件的 null 字符串，与 COPY 的 NULL 选项相同。
- **encoding**
  指定文件的编码，与 COPY 的 ENCODING 选项相同。
- **force_not_null**
  这是一个布尔选项。如果为真，则声明字段的值不应该匹配空字符串（也就是， 文件级别 null 选项）。与 COPY 的 FORCE_NOT_NULL 选项里的字段相同。

file_fdw 不支持 COPY 的 OIDS 和 FORCE_QUOTE 选项。

注意这些选项只能为外部表或它的字段声明，不是在 file_fdw 外部数据封装器的选项里，也不是在使用该封装器的服务器或用户映射的选项里。

修改表级别的选项需要系统管理员权限，因为安全原因：只有系统管理员用户能够决定读哪个文件。

对于一个使用 file_fdw 的外部表，EXPLAIN 显示要读取的文件名。除非指定了 COSTS OFF，否则也显示文件大小（字节计）。

21.4.1 使用 file_fdw

- 使用 file_fdw 需要指定要读取的文件，请先准备好该文件，并让数据库有读取权限。
- 创建服务器对象：CREATE SERVER

- 创建用户映射：CREATE USER MAPPING
- 创建外表：CREATE FOREIGN TABLE 外表的表结构需要与指定的文件的数据保持一致。
- 对外表做查询操作，写操作不被允许。
- 删除外表：DROP FOREIGN TABLE
- 删除用户映射：DROP USER MAPPING
- 删除服务器对象：DROP SERVER

21.4.2 注意事项

- 不支持 DROP EXTENSION file_fdw 操作。

21.5 dblink

dblink 是一个可以在一个 GBase 8c 数据库会话中连接到其它 GBase 8c 数据库的工具，同 libpq 支持的连接参数一致，可参考链接字符。

21.5.1 常用的 dblink 函数

- 加载 dblink 扩展

```
CREATE EXTENSION dblink;
```

- 打开一个到远程数据库的持久连接

```
SELECT dblink_connect(text connstr);
```

- 关闭一个到远程数据库的持久连接

```
SELECT dblink_disconnect();
```

- 在远程数据库执行查询

```
SELECT * FROM dblink(text connstr, text sql);
```

- 在远程数据库执行命令

```
SELECT dblink_exec(text connstr, text sql);
```

- 返回所有打开的命名 dblink 连接的名称

```
SELECT dblink_get_connections();
```

- 发送一个异步查询到远程数据库

```
SELECT dblink_send_query(text connname, text sql);
```

- 检查连接是否正在忙于一个异步查询

```
SELECT dblink_is_busy(text connname);
```

- 删除扩展

```
DROP EXTENSION dblink;
```

21.5.2 注意事项

目前 dblink 仅支持 GBase 8c 数据库访问另一个 GBase 8c 数据库，不支持 GBase 8c 数据库访问 PostgreSQL 数据库。

22 物化视图

物化视图是一种特殊的物理表，物化视图是相对普通视图而言的。普通视图是虚拟表，应用的局限性较大，任何对视图的查询实际上都是转换为对 SQL 语句的查询，性能并没有实际上提高。物化视图实际上就是存储 SQL 执行语句的结果，起到缓存的效果。

目前 Ustore 引擎不支持创建、使用物化视图。

22.1 全量物化视图

22.1.1 概述

全量物化视图仅支持对已创建的物化视图进行全量更新，而不支持进行增量更新。创建全量物化视图语法和 CREATE TABLE AS 语法类似。

22.1.2 使用

语法格式

- 创建全量物化视图

```
CREATE MATERIALIZED VIEW [ view_name ] AS { query_block };
```

- 全量刷新物化视图

```
REFRESH MATERIALIZED VIEW [ view_name ];
```

- 删除物化视图

```
DROP MATERIALIZED VIEW [ view_name ];
```

- 查询物化视图

```
SELECT * FROM [ view_name ];
```

示例

```
--准备数据。
postgres=# CREATE TABLE t1(c1 int, c2 int);
postgres=# INSERT INTO t1 VALUES(1, 1);
postgres=# INSERT INTO t1 VALUES(2, 2);

--创建全量物化视图。
postgres=# CREATE MATERIALIZED VIEW mv AS select count(*) from t1;
CREATE MATERIALIZED VIEW

--查询物化视图结果。
postgres=# SELECT * FROM mv;
 count
-------
     2
(1 row)

--向物化视图中基表插入数据。
postgres=# INSERT INTO t1 VALUES(3, 3);

--对全量物化视图做全量刷新。
postgres=# REFRESH MATERIALIZED VIEW mv;
```

```
REFRESH MATERIALIZED VIEW

--查询物化视图结果。
postgres=# SELECT * FROM mv;
 count
-------
     3
(1 row)

--删除物化视图。
postgres=# DROP MATERIALIZED VIEW mv;
DROP MATERIALIZED VIEW
```

22.1.3 支持和约束

支持场景

- 通常全量物化视图所支持的查询范围与 CREATE TABLE AS 语句一致。
- 全量物化视图上支持创建索引。
- 支持 analyze、explain。

不支持场景

物化视图不支持增删改操作，只支持查询语句。

约束

全量物化视图的刷新、删除过程中会给基表加高级别锁，若物化视图的定义涉及多张表，需要注意业务逻辑，避免死锁产生。

22.2 增量物化视图

22.2.1 概述

增量物化视图可以对物化视图增量刷新，需要用户手动执行语句完成对物化视图在一段时间内的增量数据刷新。与全量创建物化视图的不同在于目前增量物化视图所支持场景较小。目前物化视图创建语句仅支持基表扫描语句或者 UNION ALL 语句。

22.2.2 使用

语法格式

- 创建增量物化视图

```
CREATE INCREMENTAL MATERIALIZED VIEW [ view_name ] AS { query_block };
```

- 全量刷新物化视图

```
REFRESH MATERIALIZED VIEW [ view_name ];
```

- 增量刷新物化视图

```
REFRESH INCREMENTAL MATERIALIZED VIEW [ view_name ];
```

- 删除物化视图

```
DROP MATERIALIZED VIEW [ view_name ];
```

- 查询物化视图

```
SELECT * FROM [ view_name ];
```

示例

```
--准备数据。
postgres=# CREATE TABLE t1(c1 int, c2 int);
postgres=# INSERT INTO t1 VALUES(1, 1);
postgres=# INSERT INTO t1 VALUES(2, 2);


--创建增量物化视图。
postgres=# CREATE INCREMENTAL MATERIALIZED VIEW mv AS SELECT * FROM
t1;
```

```
CREATE MATERIALIZED VIEW

--插入数据。
postgres=# INSERT INTO t1 VALUES(3, 3);
INSERT 0 1

--增量刷新物化视图。
postgres=# REFRESH INCREMENTAL MATERIALIZED VIEW mv;
REFRESH MATERIALIZED VIEW

--查询物化视图结果。
postgres=# SELECT * FROM mv;
 c1 | c2
----+----
  1 |  1
  2 |  2
  3 |  3
(3 rows)

--插入数据。
postgres=# INSERT INTO t1 VALUES(4, 4);
INSERT 0 1

--全量刷新物化视图。
postgres=# REFRESH MATERIALIZED VIEW mv;
REFRESH MATERIALIZED VIEW

--查询物化视图结果。
postgres=# select * from mv;
 c1 | c2
```

```
----+----
   1 |    1
   2 |    2
   3 |    3
   4 |    4
(4 rows)


--删除物化视图。
postgres=# DROP MATERIALIZED VIEW mv;
DROP MATERIALIZED VIEW
```

22.2.3 支持和约束

支持场景

- 单表查询语句。
- 多个单表查询的 UNION ALL。
- 物化视图上支持创建索引。
- 物化视图支持 Analyze 操作。

不支持场景

- 物化视图中不支持多表 Join 连接计划以及 subquery 计划。
- 除少部分 ALTER 操作外，不支持对物化视图中基表执行绝大多数 DDL 操作。
- 物化视图不支持增删改操作，只支持查询语句。
- 不支持用临时表/hashbucket/unlog/分区表创建物化视图。
- 不支持物化视图嵌套创建（即物化视图上创建物化视图）。
- 仅支持行存表，不支持列存表。
- 不支持 UNLOGGED 类型的物化视图，不支持 WITH 语法。

约束

- 物化视图定义如果为 UNION ALL，则其中每个子查询需使用不同的基表。
- 增量物化视图的创建、全量刷新、删除过程中会给基表加高级别锁，若物化视图的定义为 UNION ALL，需要注意业务逻辑，避免死锁产生。

23 GUC 参数说明

23.1 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。

23.2 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。

23.2.1 data_directory

参数说明：设置 GBase 8c 的数据目录（data 目录）。此参数可以通过如下方式指定。

- 在安装 GBase 8c 时指定。
- 该参数属于 POSTMASTER 类型参数，请参考表 15-15-1 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。

23.2.2 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 15-15-1 中对应设置方法进行设置，不支持使用表 GUC 参数设置方式中的方式四进行修改。

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）

23.2.3 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 15-15-1 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）

23.2.4 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 15-15-1 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）

23.2.5 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 15-15-1 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

23.3 连接和认证

23.3.1 连接设置

介绍设置客户端和服务器连接方式相关的参数。

23.3.1.1 listen_addresses

参数说明：声明服务器侦听客户端的 TCP/IP 地址。

该参数指定 GBase 8c 服务器使用哪些 IP 地址进行侦听，如 IPV4 或 IPV6（若支持）。服务器主机上可能存在多个网卡，每个网卡可以绑定多个 IP 地址，该参数就是控制 GBase 8c 到底绑定在哪个或者哪几个 IP 地址上。而客户端则可以通过该参数中指定的 IP 地址来连接 GBase 8c 或者给 GBase 8c 发送请求。

该参数属于 POSTMASTER 类型参数，请参考表 15-15-1 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”表示侦听所有 IP 地址。配置侦听所有 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。同时，主备环境下配置为“*”或“0.0.0.0”时，主节点数据库路径下 postgresql.conf 文件中的 localport 端口号不能为数据库 dataPortBase+1，否则会导致数据库无法启动。
- 置空则服务器不会侦听任何 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：数据库实例安装好后，根据 XML 配置文件中不同实例的 IP 地址配置不同默认值。DN 的默认参数值为：listen_addresses = 'x.x.x.x'。

23.3.1.2 local_bind_address

参数说明：声明当前节点连接 GBase 8c 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 15-15-1 中对应设置方法进行设置。

默认值：数据库实例安装好后，根据 XML 配置文件中不同实例的 IP 地址配置不同默认值。DN 的默认参数值为：local_bind_address = 'x.x.x.x'。

23.3.1.3 port

参数说明：GBase 8c 服务侦听的 TCP 端口号。

该参数属于 POSTMASTER 类型参数，请参考表 15-15-1 中对应设置方法进行设置。

说明： 该参数由安装时的配置文件指定，请勿轻易修改，否则修改后会影响数据库正常通信。

取值范围：整型，1～65535

说明：

设置端口号时，请设置一个未被占用的端口号。设置多个实例的端口号，不可冲突。

1~1023 为操作系统保留端口号，请不要使用。

通过配置文件安装数据库实例时，配置文件中的端口号需要注意通信矩阵预留端口。如：DN 还需保留 dataPortBase+1 作为内部工具使用端口，保留 dataPortBase+6 作为流引擎消息队列通信端口等。故数据库实例安装阶段，port 最大值为：DN 可设置 65529，同时需要保证端口号不冲突。

默认值：5432（实际值由安装时的配置文件指定）

23.3.1.4 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 GBase 8c 的并发能力。

该参数属于 POSTMASTER 类型参数，请参考表 15-15-1 中对应设置方法进行设置。

取值范围：整型。最小值为 10（要大于 max_wal_senders），理论最大值为 262143，实际最大值为动态值，计算公式为“262143 - job_queue_processes - autovacuum_max_workers - AUXILIARY_BACKENDS - AV_LAUNCHER_PROCS - max_inner_tool_connections”。job_queue_processes、autovacuum_max_workers 和 max_inner_tool_connections 的值取决于对应 GUC 参数的设置。AUXILIARY_BACKENDS 为预留辅助线程数，固定为 20。AV_LAUNCHER_PROCS 为预留 autovacuum 的 lancher 线程数，固定为 2。

默认值：

- 200：编译安装数据库或极简安装数据库的情况下。
- 5000：使用 om 安装数据库的情况下。

设置建议：

数据库主节点中此参数建议保持默认值。

配置不当时影响：

- 若配置 max_connections 过大，超过计算公式所描述的最大动态值，会出现节点拉起失败问题，报错提示“invalid value for parameter “max_connections””。
- 若未按照对外出口规格配置仅调大 max_connections 参数值，未同比例调整内存参数。业务压力大时，容易出现内存不足，报错提示“memory is temporarily unavailable”。

说明：

- 对于管理员用户的连接数限制会略超过 max_connections 设置，目的是为了让管理员在链接被普通用户占满后仍可以连接上数据库，再超过一定范围（sysadmin_reserved_connections 参数）后才会报错。即管理员用户的最大连接数等于 max_connections + sysadmin_reserved_connections。
- 对于普通用户来说，由于内部作业也会使用一些链接，因此会略小于 max_connections，具体值取决于内部链接个数。

23.3.1.5 max_inner_tool_connections

参数说明：允许和数据库连接的工具的最大并发连接数。此参数会影响 GBase 8c 的工具连接并发能力。

该参数属于 POSTMASTER 类型参数，请参考表 15-15-1 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 MIN(262143, max_connections)，max_connections 的计算方法见上文。

默认值：数据库节点为 50。如果该默认值超过内核支持的最大值（在执行 gs_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。

增大此参数可能导致 GBase 8c 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。

23.3.1.6 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。该参数和 max_connections 参数配合使用，管理员用户的最大连接数等于 max_connections + sysadmin_reserved_connections。

该参数属于 POSTMASTER 类型参数，请参考表 15-15-1 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)，max_connections 的计算方法见上文。

默认值：3

23.3.1.7 unix_socket_directory

参数说明：设置 GBase 8c 服务器侦听客户端连接的 Unix 域套接字目录。

该参数属于 POSTMASTER 类型参数，请参考表 15-1 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）

23.3.1.8 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 15-1 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

23.3.1.9 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 15-1 中对应设置方法进行设置。

取值范围：0000-0777

默认值：0777

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。

23.3.1.10 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 15-1 中对应设置方法进行设置。

在备机请求主机进行日志复制时，如果该参数非空串，那么会被用来作为备机在主机上的流复制槽名字。此时，如果该参数长度超过 61 个字节，那么流复制槽名字只会截取使用前 61 个字节的字符。

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）

23.3.1.11 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。

驱动连接数据库的时候自行拼接的 connection_info 参数格式如下：

```
{"driver_name":"ODBC","driver_version": "(GBase 8c X.X.X build 13b34b53) compiled at 2020-05-08 02:59:43 commit 2143 last mr 131 debug","driver_path":"/usr/local/lib/psqlodbcw.so","os_user":"gbase"}
```

默认显示 driver_name 和 driver_version，driver_path 和 os_user 的显示由用户控制（参见连接数据库和 Linux 下配置数据源）。

23.3.2 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。

23.3.2.1 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 15-1 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

23.3.2.2 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 15-1 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。

23.3.2.3 session_authorization

参数说明：当前会话的用户标识。

该参数属于 USERSET 类型参数，此参数为内部参数，只能通过 SET SESSION AUTHORIZATION 语法设置，不支持直接设置。

取值范围：字符串。

默认值：NULL

23.3.2.4 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 15-1 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：

GBase 8c gsql 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 gsql 表现的现象为断开后重连。

23.3.2.5 ssl

参数说明：启用 SSL 连接。请在使用这个选项之前阅读使用 gsql 连接。

该参数属于 POSTMASTER 类型参数，请参考表 15-1 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：

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 GBase 8c 无法正常启动。

默认值：on

23.3.2.6 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使用 gsql 连接。

该参数属于 SIGHUP 类型参数，请参考表 15-1 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：

GBase 8c 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

23.3.2.7 ssl_ciphers

参数说明：指定 SSL 支持的加密算法列表。ssl_ciphers 设置错误会导致数据库不能正常启动。

该参数属于 POSTMASTER 类型参数，请参考表 15-1 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。详细请参见用 SSL 进行安全的 TCP/IP 连接获取支持的加密算法。

默认值：ALL

23.3.2.8 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 15-1 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

23.3.2.9 ssl_cert_file

参数说明：指定包含 SSL 服务器证书的文件的名称，其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 15-1 中对应设置方法进行设置。

取值范围：字符串

默认值：server.crt

23.3.2.10 ssl_key_file

参数说明：指定包含 SSL 私钥的文件名称，其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 15-1 中对应设置方法进行设置。

取值范围：字符串

默认值：server.key

23.3.2.11 ssl_ca_file

参数说明：指定包含 CA 信息的文件的名称，其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 15-1 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 CA 文件被加载，不进行客户端证书验证。

默认值：cacert.pem

23.3.2.12 ssl_crl_file

参数说明：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。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 15-1 中对应设置方法进行设置。

取值范围：字符串，空字符串表示没有吊销列表。

默认值：空

23.3.2.13 g_instance.attr.attr_security.encryption_card_type

参数说明：指定加密卡类型。

取值范围：0、1。0 表示软加密，1 表示使用渔翁加密卡。

默认值：0

23.3.2.14 g_instance.attr.attr_security.encryption_card_version

参数说明：加密卡版本，用于指定加密卡的版本号。

当该参数与 g_instance.attr.attr_security.encryption_card_type 同时有效时，会在进程初始化时尝试打开加密卡。

23.3.2.15 krb_server_keyfile

参数说明：指定 Kerberos 服务主配置文件的位置，详细请参见配置客户端接入认证。

该参数属于 SIGHUP 类型参数，请参考表 15-1 中对应设置方法进行设置。

取值范围：字符串

默认值：空

23.3.2.16 krb_srvname

参数说明：设置 Kerberos 服务名，详细请参见配置客户端接入认证。

该参数属于 SIGHUP 类型参数，请参考表 15-1 中对应设置方法进行设置。

取值范围：字符串

默认值：postgres

23.3.2.17 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 15-1 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

23.3.2.18 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 GBase 8c 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设置密码安全策略。

该参数属于 SIGHUP 类型参数，请参考表 15-1 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 表示不采用密码复杂度校验策略。
- 1 表示采用默认密码复杂度校验策略。

默认值：1

23.3.2.19 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设置密码安全策略。

该参数属于 SIGHUP 类型参数，请参考表 15-1 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，最小值为 0，最大值为 3650，单位为天。

- 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

23.3.2.20 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设置密码安全策略。

该参数属于 SIGHUP 类型参数，请参考表 15-1 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

23.3.2.21 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设置密码安全策略。

该参数属于 SIGHUP 类型参数，请参考表 15-1 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：1d

23.3.2.22 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 参数设定的值，则当前帐户会被锁定。password_lock_time 参数设定的天数过后，帐户自动解锁。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设置密码安全策略。

该参数属于 SIGHUP 类型参数，请参考表 15-1 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

23.3.2.23 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 15-1 中对应设置方法进行设置。

取值范围：0、1、2、3

- 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。
- 3 表示采用 sm3 方式对密码加密。

须知：

md5 加密算法安全性低，存在安全风险，不建议单独使用。

默认值：主备式场景下默认值为 2；分布式场景下默认值为 1

23.3.2.24 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 15-1 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

23.3.2.25 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 15-1 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

23.3.2.26 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 15-1 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

23.3.2.27 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 15-1 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

23.3.2.28 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 15-1 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

23.3.2.29 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 15-1 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

23.3.2.30 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 15-1 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

23.3.2.31 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 15-1 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

23.3.3 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。

23.3.3.1 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 15-15-1 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。
- 将该值设置为 0 时，将使用系统的值。
- 该参数在不同的会话之间不共享，也就是说不同的会话连接可能有不同的值。
- 查看该参数时查出来的是当前会话连接内的参数值，而不是 guc 副本的值。

取值范围：0-3600，单位为 s。

默认值：0

23.3.3.2 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 15-15-1 中对应设置方法进行设置。

取值范围：0-180，单位为 s。

默认值：0

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。
- 将该值设置为 0 时，将使用系统的值。
- 该参数在不同的会话之间不共享，也就是说不同的会话连接可能有不同的值。
- 查看该参数时查出来的是当前会话连接内的参数值，而不是 guc 副本的值。

23.3.3.3 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 GBase 8c 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 15-15-1 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。
- 将该值设置为 0 时，将使用系统的值。
- 该参数在不同的会话之间不共享，也就是说不同的会话连接可能有不同的值。
- 查看该参数时查出来的是当前会话连接内的参数值，而不是 guc 副本的值。

取值范围：0-100，其中 0 表示 GBase 8c 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：0

23.3.3.4 comm_proxy_attr

参数说明：通信代理库相关参数配置。

说明： - 该参数仅支持欧拉 2.9 系统下的集中式 ARM 单机。 - 本功能在线程池开启状态下生效，即 enable_thread_pool 为 on。 - 配置该参数时需同步配置 GUC 参数 local_bind_address 为 libos_kni 的网卡 IP。- 参数模板 :comm_proxy_attr = '{enable_libnet:true, enable_dfx:false, numa_num:4, numa_bind:[[30,31],[62,63],[94,95],[126,127]]}' - 可配置参数说明。 - enable_libnet：是否开启用户态协议，取值范围：true、false。 - enable_dfx：是否开

启通信代理库视图，取值范围：true、false。 - numa_num：机器环境中 numa 的数量，支持 2P、4P 服务器，取值范围：4、8。 - numa_bind：代理线程绑核参数，每个 numa 两个 CPU 绑核，共 numa_num 组，取值范围：[0，cpu 数-1]。

该参数属于 POSTMASTER 类型参数，请参考表 15-15-1 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：'none'

23.4 资源消耗

23.4.1 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。

23.4.1.1 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 15-1 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

23.4.1.2 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 15-1 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

23.4.1.3 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 15-1 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

注意：

- 若 max_process_memory-shared_buffers-cstore_buffers-元数据少于 2G，GBase 8c 强制把 enable_memory_limit 设置为 off。其中元数据是 GBase 8c 内部使用的内存和部分并发参数，如 max_connections、thread_pool_attr、max_prepared_transactions 等参数相关。
- 当该值为 off 时，不对数据库使用的内存做限制，在大并发或者复杂查询时，使用内存过多，可能导致操作系统 OOM 问题。

23.4.1.4 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 15-1 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：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\ - vm.min_free_kbytes) * 0.7 / (1 + 主节点个数)。该系数的目的是尽可能保证系统的可靠性，不会因数据库内存膨胀导致节点 OOM。这个公式中提到 vm.min_free_kbytes，其含义是预留操作系统内存供内核使用，通常用作操作系统内核中通信收发内存分配，至少为 5%内存。即，max_process_memory = 物理内存 * 0.665 / (1 + 主

节点个数)。

注意： 当该值设置不合理，即大于服务器物理内存，可能导致操作系统 OOM 问题。

23.4.1.5 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 15-1 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

23.4.1.6 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 15-1 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

23.4.1.7 shared_buffers

参数说明：设置 GBase 8c 使用的共享内存大小。增加此参数的值会使 GBase 8c 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 15-1 中对应设置方法进行设置。

取值范围：整型，16 ~ 1073741823，单位为 8KB。

shared_buffers 需要设置为 BLCKSZ 的整数倍，BLCKSZ 目前设置为 8KB，即

shared_buffers 需要设置为 8KB 整数倍。改变 BLCKSZ 的值会改变最小值。

默认值：8MB

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。

23.4.1.8 segment_buffers

参数说明：设置 GBase 8c 段页式元数据页的内存大小。

该参数属于 POSTMASTER 类型参数，请参考表 15-1 中对应设置方法进行设置。

取值说明：整型，16 ~ 1073741823，单位为 8KB。

segment_buffers 需要设置为 BLCKSZ 的整数倍，BLCKSZ 目前设置为 8KB，即 segment_buffers 需要设置为 8KB 整数倍。改变 BLCKSZ 的值会改变最小值。

默认值：8MB

设置建议：

segment_buffers 用来缓存段页式段头的内容，属于关键元数据信息，为了提高性能建议常用的表的段头都能缓存在 buffer 中，不被置换出去。建议按照表的个数(包括索引和 toast 表）* 分区数 * 3 + 128 来设置。乘以 3 是因为每个表（分区）会有一些额外的元数据段，一般一个表有 3 个段。最后+128 因为段页式表空间管理需要一定数量的 buffer。

23.4.1.9 bulk_write_ring_size

参数说明：大批量数据写入触发时（例如 copy 动作），该操作使用的环形缓冲区大小。

该参数属于 USERSET 类型参数，请参考表 15-1 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。

23.4.1.10 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 15-1 中对应设置方法进行设置。

取值范围：双精度浮点型，0.1~1.0

默认值：0.3

23.4.1.11 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 15-1 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：1MB

23.4.1.12 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 GBase 8c 比系统默认设置需要更多的 System V 共享内存。

当 GBase 8c 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 15-1 中对应设置方法进行设置。

取值范围：整型，0~262143。

默认值：10

说明： 一般不需要对事务显式进行 PREPARE 操作，如果业务对事务进行显示 PREPARE 操作，为避免在准备步骤失败，需要调大该值，大于需要进行 PREPARE 业务的并发数。

23.4.1.13 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 15-1 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。 >- 对于 BitmapScan 的哈希表也会受到 work_mem 的限制，但不会被严格管控下盘。完全 Lossify 的情况下，哈希表每占用 1MB 的内存，对应一次 BitmapHeapScan 的 16GB 的页面（Ustore 为 32GB），达到 work_mem 上限后，会按此比例随数据访问量线性增长。

23.4.1.14 query_mem

参数说明：设置执行作业所使用的内存。

该参数属于 USERSET 类型参数，请参考表 15-1 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。

默认值：0

须知：

- 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。
- 如果设置值为负数或小于 32MB，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。

23.4.1.15 query_max_mem

参数说明：设置执行作业所能够使用的最大内存。

该参数属于 USERSET 类型参数，请参考表 15-1 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。

默认值：0

须知：

- 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。
- 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。

23.4.1.16 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 15-1 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：16MB

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。
- 当自动清理线程运行时，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。

23.4.1.17 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 15-1 中对应设置方法进行设置。

须知： 同样，多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 的几倍。

取值范围：整型 64~2147483647，单位为 KB。

默认值：512MB

23.4.1.18 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 15-1 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

23.4.1.19 max_stack_depth

参数说明：设置 GBase 8c 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 15-1 中对应设置方法进行设置。

取值范围：整型，100~INT_MAX，单位为 KB。

默认值：2MB

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=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线程崩溃。在 GBase 8c 能够检测内核限制的操作系统上，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。 >- 默认值 2MB，这个值相对比较小，不容易导致系统崩溃。

23.4.1.20 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 15-1 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：1GB

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。

23.4.1.21 bulk_read_ring_size

参数说明：大批量数据查询时（例如大表扫描），该操作使用的环形缓冲区大小。

该参数属于 USERSET 类型参数，请参考表 15-1 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

23.4.1.22 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 15-1 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

23.4.1.23 local_syscache_threshold

参数说明：系统表 cache 在单个 session 缓存的大小。

该参数属于 PG_SIGHUP 类型参数，请参考表 15-1 中对应设置方法进行设置。

如果 enable_global_plancache 已打开，为保证 GPC 生效，local_syscache_threshold 设置值小于 16MB 时不会生效，最小为 16MB。

如果 enable_global_plancache 和 enable_thread_pool 打开，该参数描述的是当前线程和绑定到当前线程上的 session 缓存的总大小。

取值范围：整型，1*1024 ~ 512*1024，单位 KB。

默认值：on

23.4.1.24 resilience_memory_reject_percent

参数说明：用于控制内存过载逃生的动态内存占用百分比。该参数仅在 GUC 参数 use_workload_manager 和 enable_memory_limit 打开时生效。该参数属于 SIGHUP 类型参数，请参考表 15-1 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 recover_memory_percent,、overload_memory_percent 2 部分，这 2 个部分的

具体含义如下：

recover_memory_percent：内存从过载状态恢复正常状态的动态内存使用占最大动态内存的百分比，当动态内存使用小于最大动态内存乘以该值对应的百分比后，停止过载逃生并放开新连接接入，取值为 0~100，设置为多少表示百分之多少。

overload_memory_percent：内存过载时动态内存使用占最大动态内存的百分比，当动态内存使用大于最大动态内存乘以该值对应的百分比后，表示当前内存已经过载，触发过载逃生 kill 会话并禁止新连接接入，取值为 0~100，设置为多少表示百分之多少。

默认值：'0,0'，表示关闭内存过载逃生功能。

示例：

```
resilience_memory_reject_percent = ’70,90’
```

表示内存使用超过最大内存上限的 90%后禁止新连接接入并 kill 堆积的会话，kill 会话过程中内存恢复到最大内存的 70%以下时停止 kill 会话并允许新连接接入。

须知：

- 最大动态内存和已使用的动态内存可以通过 gs_total_memory_detail 视图查询获得，最大动态内存：max_dynamic_memory，已使用的动态内存：dynamic_used_memory。
- 该参数如果设置的百分比过小，则会频繁触发内存过载逃生流程，会使正在执行的会话被强制退出，新连接短时间接入失败，需要根据实际内存使用情况慎重设置。

23.4.2 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。

23.4.2.1 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、管控的空间包括普通表、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 15-1 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

23.4.2.2 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 15-1 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

23.4.3 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。

23.4.3.1 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的文件数。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 15-1 中对应设置方法进行设置。

取值范围：整型，25~2147483647。

默认值：1000

23.4.3.2 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 GBase 8c 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 15-1 中对应设置方法进行设置。

⚠ 须知：

- 如果被声明的库不存在，GBase 8c 服务将会启动失败。

- 每一个支持 GBase 8c 的库都有一个特殊的标记用于保证兼容性。因此，不支持 GBase 8c 的库不能用这种方法加载。

取值范围：字符串

默认值：空

23.4.4 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 GBase 8c 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

23.4.4.1 背景信息

在 ANALYZE | ANALYSE 和 VACUUM 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。如需开启，需要把 vacuum_cost_delay 变量设置为一个非零值。

23.4.4.2 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 15-1 中对应设置方法进行设置。

取值范围：整型，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

23.4.4.3 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。它代表锁住缓冲池、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 15-1 中对应设置方法进行设置。

取值范围：整型，0~10000。

默认值：1

23.4.4.4 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。它代表锁住缓冲池、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 15-1 中对应设置方法进行设置。

取值范围：整型，0~10000。

默认值：10

23.4.4.5 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。它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 15-1 中对应设置方法进行设置。

取值范围：整型，0~10000

默认值：20

23.4.4.6 vacuum_cost_limit

参数说明：设置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 15-1 中对应设置方法进行设置。

取值范围：整型，1~10000。

默认值：200

23.4.5 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。

23.4.5.1 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 15-1 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：2s

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。

23.4.5.2 candidate_buf_percent_target

参数说明：设置用于增量检查点打开时，候选 buffer 链中可用 buffer 数目占据 shared_buffer 内存缓冲区百分比的期望值，当前候选链中的数目少于目标值时，bgwriter 线程会启动将满足条件的脏页刷盘。

该参数属于 SIGHUP 类型参数，请参考表 15-1 中对应设置方法进行设置。

取值范围：双精度浮点类型，0.1—0.85

默认值：0.3

23.4.5.3 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 15-1 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

23.4.5.4 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 15-1 中对应设置方法进行设置。

取值范围：浮点型，0~10。

默认值：2

23.4.5.5 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，主要是按照脏页置脏的顺序刷盘，用于推进 recovery 点。

该参数属于 POSTMASTER 类型参数，请参考表 15-1 中对应设置方法进行设置。

取值范围：整型，1~16

默认值：4

23.4.5.6 dirty_page_percent_max

参数说明：设置用于增量检查点打开后脏页数量占 shared_buffers 的百分比。达到这个设定值时，后台刷页线程将以设置的 max_io_capacity 计算出的最大值刷脏页。

该参数属于 SIGHUP 类型参数，请参考表 15-1 中对应设置方法进行设置。

取值范围：浮点型，0.1~1

默认值：0.9

23.4.5.7 pagewriter_sleep

参数说明：设置用于增量检查点打开后，pagewrite 线程每隔 pagewriter_sleep 的时间刷一批脏页下盘。当脏页占据 shared_buffers 的比例达到 dirty_page_percent_max 时，每批页面数量以设定的 max_io_capacity 计算出的值刷页，其余情况每批页面数量按比例相对减少。

该参数属于 SIGHUP 类型参数，请参考表 15-1 中对应设置方法进行设置。

取值范围：整型，0~3600000（毫秒）

默认值：2000ms

23.4.5.8 max_io_capacity

参数说明：设置后端写进程批量刷页每秒的 IO 上限，需要根据具体业务场景和机器磁盘 IO 能力进行设置。要求 RTO 很短时间或者数据量比共享内存大多倍的情况，业务访问数据量又是随机访问时，该值不宜过小。该参数设置较小会减小后端写进程刷页个数，如果业务触发页面淘汰多时，该值设置小会影响业务。

该参数属于 SIGHUP 类型参数，请参考表 15-1 中对应设置方法进行设置。

取值范围：整型，30720~10485760。单位是 KB。

默认值：512000KB（500MB）

23.4.5.9 enable_consider_usecount

参数说明：设置 backend 线程在页面置换时是否考虑页面热度，建议大容量场景下开启此参数。

该参数属于 SIGHUP 类型参数，请参考表 15-1 中对应设置方法进行设置。

取值范围：布尔型

- on/true 表示考虑页面热度。
- off/false 表示不考虑页面热度。

默认值：off

23.4.5.10 dw_file_num

参数说明：设置批量双写文件的数量，该值与 pagewriter_thread_num 有关，不会大于 pagwriter_thread_num，如果设置过大，内部会纠正为 pagewriter_thread_num。

该参数属于 POSTMASTER 类型参数，请参考表 15-1 中对应设置方法进行设置。

取值范围：整型，1~16

默认值：1

23.4.5.11 dw_file_size

参数说明：设置每个批量双写文件的大小。

该参数属于 POSTMASTER 类型参数，请参考表 15-1 中对应设置方法进行设置。

取值范围：整型，32~256

默认值：256

23.4.6 异步 IO

23.4.6.1 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 15-1 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

说明： 当前版本暂不支持打开该开关，即使用户手动设置为打开，系统内部也会自动设置为关闭状态。

23.4.6.2 enable_adio_function

参数说明：是否开起 ADIO 功能。

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。

说明： 当前版本暂不支持开启异步 IO 功能，默认该功能关闭。

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

23.4.6.3 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 15-1 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

23.4.6.4 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 15-1 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)

23.4.6.5 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 15-1 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)

23.4.6.6 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 15-1 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

23.4.6.7 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 15-1 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

23.4.6.8 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 15-1 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

23.4.6.9 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 15-1 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

23.4.6.10 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 15-1 中对应设置方法进行设置。

取值范围：整型，0~1000

默认值：1

23.4.6.11 checkpoint_flush_after

参数说明：设置 checkpointer 线程刷页个数超过设定的阈值时，告知操作系统开始将操作系统缓存中的页面异步刷盘。GBase 8c 中，磁盘页大小为 8KB。

该参数属于 SIGHUP 类型参数，请参考表 15-1 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

23.4.6.12 bgwriter_flush_after

参数说明：设置 background writer 线程刷页个数超过设定的阈值时，告知操作系统开始将操作系统缓存中的页面异步刷盘。GBase 8c 中，磁盘页大小为 8KB。

该参数属于 SIGHUP 类型参数，请参考表 15-1 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（8KB）。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：512KB（即 64 个页面）

23.4.6.13 backend_flush_after

参数说明：设置 backend 线程刷页个数超过设定的阈值时，告知操作系统开始将操作系统缓存中的页面异步刷盘。GBase 8c 中，磁盘页大小为 8KB。

该参数属于 USERSET 类型参数，请参考表 15-1 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（8KB）。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

23.5 预写式日志

23.5.1 设置

23.5.1.1 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 15-1 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。

- **logical**

这个参数表示 WAL 日志支持逻辑复制。

默认值：logical

23.5.1.2 fsync

参数说明：设置 GBase 8c 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 15-1 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

23.5.1.3 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 15-1 中对应设置方法进行设置。

通常情况下，一个事务产生的日志的同步顺序如下：

1. 主机将日志内容写入本地内存。
2. 主机将本地内存中的日志写入本地文件系统。
3. 主机将本地文件系统中的日志内容刷盘。
4. 主机将日志内容发送给备机。
5. 备机接受到日志内容，存入备机内存。
6. 备机将备机内存中的日志写入备机文件系统。
7. 备机将备机文件系统中的日志内容刷盘。
8. 备机回放日志，完成对数据文件的增量更新。

取值范围：枚举类型

- on：表示主机事务提交需要等待备机将对应日志刷新到磁盘。
- off：表示主机事务提交无需等待主机自身将对应日志刷新到磁盘，通常也称为异步提交。
- local：表示主机事务提交需要等待主机自身将对应日志刷新到磁盘，通常也称为本地提交。
- remote_write：表示主机事务提交需要等待备机将对应日志写到文件系统（无需刷新到磁盘）。
- remote_receive：表示主机事务提交需要等待备机接收到对应日志数据（无需写入文件系统）。
- remote_apply：表示主机事务提交需要等待备机完成对应日志的回放操作。
- true：同 on。
- false：同 off。
- yes：同 on。
- no：同 off。
- 1：同 on。
- 0：同 off。
- 2：同 remote_apply。

默认值：on

23.5.1.4 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 15-1 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，Windows 平台支持将 wal_sync_method 设置为 fsync_writethrough。在 Windows 平台上 fsync_writethrough 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）

说明： 不是所有的平台都支持以上参数。

默认值：fdatasync

23.5.1.5 full_page_writes

参数说明：设置 GBase 8c 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。当增量检查点开关和 enable_double_write 同时打开时，则不使用 full_page_writes。

该参数属于 SIGHUP 类型参数，请参考表 15-1 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

23.5.1.6 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 15-1 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

23.5.1.7 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 15-1 中对应设置方法进行设置。

取值范围：-1~218

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，这种情况下最小值为 8 个 XLOG_BLCKSZ，最大值为 2048 个 XLOG_BLCKSZ，自动调整后的值小于最小值时会调整为最小值，大于最大值时会调整为最大值。

如果设置为其他值，当小于 4 时，会被默认设置为 4。

默认值：2048，即 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。

23.5.1.8 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 15-1 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，单位为毫秒。

默认值：200ms

23.5.1.9 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 15-1 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，单位为微秒，其中 0 表示无延迟。

默认值：0

23.5.1.10 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 15-1 中对应设置方法进行设置。

取值范围：整型， 0—1000

默认值：5

23.5.1.11 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

23.5.1.12 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)

23.5.1.13 walwriter_cpu_bind

参数说明：绑定到 WAL 写入线程的 CPU 核，与 thread pool 配合使用。

该参数属于 POSTMASTER 类型参数，请参考表 15-1 中对应设置方法进行设置。

取值范围：整型，-1~核数减 1。

默认值：-1

23.5.1.14 walwriter_sleep_threshold

参数说明：xlogflusher 进入 sleep 之前空闲 xlog 刷新的次数，达到阈值会休眠。

该参数属于 SIGHUP 类型参数，请参考表 15-1 中对应设置方法进行设置。

取值范围：整型，1~50000。

默认值：500

23.5.1.15 wal_file_init_num

参数说明：WAL 编写器将创建的 xlog 段文件的数量。

该参数属于 POSTMASTER 类型参数，请参考表 15-1 中对应设置方法进行设置。

取值范围：整型，0~1000000。

默认值：10

23.5.1.16 force_promote

参考说明：备机强切功能开关。

备机强切在集群故障状态下，以丢失部分数据为代价换取集群尽可能快的恢复服务；是集群状态为不可用时的一个逃生方法，不建议频繁触发。如果操作者不清楚备机强切后丢失数据对业务的影响，请勿使用本功能。

取值范围：整型，0 或 1

0 表示关闭，1 表示开启

默认值：0

23.5.1.17 wal_flush_timeout

参数说明：遍历 WalInsertStatusEntryTbl 的超时时间。Xlog 刷盘自适应控制的刷盘 IO 遍历 WalInsertStatusEntryTbl 等待的最大时间。

该参数属于 SIGHUP 类型参数，请参考表 15-1 中对应设置方法进行设置。

须知： 如果时间过长可能造成 Xlog 刷盘频率降低，降低 Xlog 处理性能。

取值范围：整型， 0～90000000（微秒）

默认值：2us

23.5.1.18 wal_flush_delay

参数说明：遍历 WalInsertStatusEntryTbl 时，遇到 WAL_NOT_COPIED 状态 entry 时等待的时间间隔。

该参数属于 SIGHUP 类型参数，请参考表 15-1 中对应设置方法进行设置。

取值范围：整型， 0～90000000（微秒）

默认值：1us

23.5.2 检查点

23.5.2.1 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 15-1 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

23.5.2.2 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 15-1 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

23.5.2.3 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 15-1 中对应设置方法进行设置。

取值范围：双精度浮点类型，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。

23.5.2.4 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 15-1 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

23.5.2.5 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 15-1 中对应设置方法进行设置。

取值范围：整型，2~3600（秒）

默认值：1min

23.5.2.6 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 15-1 中对应设置方法进行设置。

取值范围：布尔型

默认值：on

23.5.2.7 enable_double_write

参数说明：双写开关。当增量检查点开关打开时，同时 enable_double_write 打开，则使用 enable_double_write 双写特性保护，不再使用 full_page_writes 防止半页写问题。

该参数属于 POSTMASTER 类型参数，请参考表 15-1 中对应设置方法进行设置。

取值范围：布尔型

默认值：on

23.5.2.8 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 15-1 中对应设置方法进行设置。

取值范围：整型，1~3600（秒）

默认值：1min

23.5.2.9 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 15-1 中对应设置方法进行设置。

取值范围：布尔型

- 设置为 on 时，如果任一备机断联时，主机回收日志。
- 设置为 off 时，如果任一备机断联时，主机不回收日志。

默认值：on

23.5.2.10 max_redo_log_size

参数说明：备 DN 表示当前回放的最新检查点位置和当前日志回放位置之间日志量的期

望值，主 DN 表示恢复点到当前最新日志之间日志量的期望值，关注 RTO 的情况下，这个值建议不宜过大。

该参数属于 SIGHUP 类型参数，请参考表 15-1 中对应设置方法进行设置。

取值范围：整型，163840—2147483647，单位为 KB。

默认值：1GB

23.5.2.11 max_size_for_xlog_prune

参数说明：在 enable_xlog_prune、synchronous_commit 都打开时生效，如果有备机断连且 xlog 日志大小大于此阈值，则回收日志。所有备机断联且无逻辑复制槽时，不回收日志。

该参数属于 SIGHUP 类型参数，请参考表 15-1 中对应设置方法进行设置。

取值范围：整型，0—2147483647，单位为 KB

默认值：2147483647，单位 KB

23.5.3 日志回放

23.5.3.1 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 15-1 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。另外，由于极致 RTO 自带流控，所以同时开启极致 RTO 与流控时会以极致 RTO 优先，在运行期间使流控不生效。

默认值：0

23.5.3.2 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 15-1 中对应设置方法进行设置。

取值范围：整型，0~20

默认值：1（安装工具默认设置为 4，以获得更好的性能）

23.5.3.3 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 15-1 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 模式和主备从模式，仅在参数 hot_standby 设置成 off，replication_type 设置成 1 时可以设置 recovery_parse_workers 为>1。另外，极致 RTO 也不支持列存，在已经使用列存表或者即将使用列存表的系统中，请关闭极致 RTO。由于极致 RTO 自带流控，所以同时开启极致 RTO 与流控时会以极致 RTO 优先，在运行期间使流控不生效。

默认值：1

23.5.3.4 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 15-1 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。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

23.5.3.5 recovery_parallelism

参数说明：查询实际回放线程个数，该参数为只读参数，无法修改。

该参数属于 POSTMASTER 类型参数，受 recovery_max_workers 以及 recovery_parse_workers 参数影响，任意一值大于 0 时，recover_parallelism 将被重新计算。

取值范围：整型，1~2147483647

默认值：1

23.5.3.6 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 15-1 中对应设置方法进行设置。

取值范围：布尔型

默认值：on

recovery_min_apply_delay

参数说明：设置备节点回放的延迟时间。

该参数属于 SIGHUP 类型参数，请参考表 15-1 中对应设置方法进行设置。

须知：

- 此参数主节点设置无效，必须设置在需要延迟的备节点上，推荐设置在异步备上，设置了延时的异步备如果升主 RTO 时间会比较长。
- 延迟时间是根据主服务器上事务提交的时间戳与备机上的当前时间来计算，因此需要保证主备系统时钟一致。
- 延迟时间设置过长时，可能会导致该备机 XLOG 文件所在的磁盘满，需要平衡考虑磁盘大小来设置延迟时间。
- 没有事务的操作不会被延迟。
- 主备切换之后，原主机若需延迟，需要再手动配置此参数。

- 当 synchronous_commit 被设置为 remote_apply 时，同步复制会受到这个延时的影响，每一个 COMMIT 都需要等待备机回放结束后才会返回。
- 使用这个特性也会让 hot_standby_feedback 被延迟，这可能导致主服务器的膨胀，两者一起使用时要小心。
- 主机执行了持有 AccessExclusive 锁的 DDL 操作，比如 DROP 和 TRUNCATE 操作，在备机延迟回放该条记录期间，在备机上对该操作对象执行查询操作会等待锁释放之后才会返回。
- 不支持 MOT 表。

取值范围：整型，0~INT_MAX，单位为毫秒。

默认值：0（不增加延迟）

23.5.3.7 redo_bind_cpu_attr

参数说明：用于控制回放线程的绑核操作，仅 sysadmin 用户可以访问。该参数属于 POSTMASTER 类型参数，请参考表 15-1 中对应设置方法进行设置。

取值范围：字符串，长度大于 0，该参数不区分大小写。

- 'nobind'：线程不做绑核。
- 'nodebind: 1, 2'：利用 NUMA 组 1,2 中的 CPU core 进行绑核。
- 'cpubind: 0-30'：利用 0-30 号 CPU core 进行绑核。

默认值：'nobind'

23.5.4 归档

23.5.4.1 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 15-1 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。

无论是同步备机还是异步备机都能够开启归档，归档开启的方式与单机开启归档一致，将 archive_mode 置为 on，并设置正确的 archive_dest 或者 archive_command 即可。

注：若未开启最大可用模式以及有同步备机与主机断开连接时，主机会因为业务阻塞的原因无法给备机发送归档的位置，从而导致归档失败。

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

23.5.4.2 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 15-1 中对应设置方法进行设置。

须知：

- 当 archive_dest 和 archive_command 同时配置时，WAL 日志优先保存到 archive_dest 所设置的目录中，archive_command 配置的命令不生效。
- 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。
- 如果归档命令有多条，则需将其写入 SHELL 脚本文件中，然后将 archive_command

配置为执行该脚本的命令。示例如下：

```
--假设多条命令如下。
test ! -f dir/%f && cp %p dir/%f
--则 test.sh 脚本内容如下。
test ! -f dir/$2 && cp $1 dir/$2
--归档命令如下。
archive_command='sh dir/test.sh %p %f'
```

取值范围：字符串

默认值：(disabled)

23.5.4.3 archive_dest

参数说明：由管理员设置的用于归档 WAL 日志的目录，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 15-1 中对应设置方法进行设置。

须知：

- 当 archive_dest 和 archive_command 同时配置时，WAL 日志优先保存到 archive_dest 所设置的目录中，archive_command 配置的命令不生效。
- 字符串中如果是相对路径为相对于数据目录的。示例如下。

```
archive_dest = '/mnt/server/archivedir/'
```

取值范围：字符串

默认值：空字符串

23.5.4.4 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 15-1 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

23.6 双机复制

23.6.1 发送端服务器

23.6.1.1 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 15-1 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—1024

- 建议取值范围为 8～100。
- 只有当使用单 DN 实例无主备场景下才可以设置为 0。

默认值：16

23.6.1.2 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 15-1 中对应设置方法进行设置。

取值范围：整型，2～INT_MAX

默认值：16

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。

若 synchronous_commit 级别小于 LOCAL_FLUSH，重建备机时，建议调大改参数为 1000，

避免重建过程中，主机日志回收导致重建失败。

23.6.1.3 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 15-1 中对应设置方法进行设置。

须知：

如果主机数据较大，重建备机数据库时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：60s

23.6.1.4 max_replication_slots

参数说明：设置主机端的日志复制 slot 个数。

该参数属于 POSTMASTER 类型参数，请参考表 15-1 中对应设置方法进行设置。

取值范围：整型，0~1024（建议取值范围：8~100）

默认值：8

设置建议：

当使用双机复制、逻辑解码时，该参数值建议设为：当前物理流复制槽数+所需的逻辑复制槽数。如果实际设置值比上述建议值要小，那么可能造成这些功能不可用或异常。

物理流复制槽提供了一种自动化的方法来确保主节点在所有备节点或从备节点收到 xlog 之前，xlog 不会被移除。也就是说物理流复制槽用于支撑主备 HA。数据库所需要的物理流复制槽数为备节点加从备的和与主节点之间的比例。例如，假设数据库高可用方案为 1 主、1 备、1 从备，则所需物理流复制槽数为 2。假设数据库的高可用方案为 1 主 3 备，则所需物理流复制槽数为 3。

目前默认不支持主备从部署方式。

关于逻辑复制槽数，请按如下规则考虑：

- 一个逻辑复制槽只能解码一个数据库的修改，如果需要解码多个数据库，则需要创建多个逻辑复制槽。
- 如果需要多路逻辑复制同步给多个目标数据库，在源端数据库需要创建多个逻辑复制槽，每个逻辑复制槽对应一条逻辑复制链路。

23.6.1.5 enable_slot_log

参数说明：是否开启逻辑复制槽主备同步特性。

该参数属于 USERSET 类型参数，请参考表 15-1 中对应设置方法进行设置。

取值范围：布尔型

- on 表示开启逻辑复制槽主备同步特性。
- off 表示不开启逻辑复制槽主备同步特性。

默认值：off

23.6.1.6 max_changes_in_memory

参数说明：逻辑解码时单条事务在内存中缓存的大小上限，单位字节。

该参数属于 POSTMASTER 类型参数，请参考表 15-1 中对应设置方法进行设置。

取值范围：整型，1~2147483647

默认值：4096

23.6.1.7 max_cached_tuplebufs

参数说明：逻辑解码时总元组信息在内存中缓存的大小上限，单位字节。建议设置为 max_changes_in_memory 的两倍以上。

该参数属于 POSTMASTER 类型参数，请参考表 15-1 中对应设置方法进行设置。

取值范围：整型，1~2147483647

默认值：8192

23.6.1.8 enable_wal_shipping_compression

参数说明：在流式容灾模式下设置启动跨数据库实例日志压缩功能。

该参数属于 SIGHUP 类型参数，请参考表 15-1 中对应设置方法进行设置。

须知： 该参数仅作用于流式容灾中跨数据库实例传输的一对 walsender 与 walreceiver 中，在主数据库实例上配置。

取值范围：布尔型

true 表示打开流式容灾跨数据库实例日志压缩

false 表示关闭流式容灾跨数据库实例日志压缩

默认值：false

23.6.1.9 repl_auth_mode

参数说明：设置主备复制和备机重建的验证模式。

该参数属于 SIGHUP 类型参数，请参考表 15-1 中对应设置方法进行设置。

须知：

- 如果主机上开启了 UUID 验证功能、且配置了非空字符串的 repl_uuid 验证码，那么备机也需要开启 UUID 验证功能、且配置相同的 repl_uuid 验证码，否则主备日志复制和备机重建请求将被主机拒绝。
- 该参数支持 SIGHUP 动态加载新值。修改之后，不影响已建连的主备连接，对后续主备复制请求和主备重建请求生效。
- 支持 Quorum、DCF 协议下的备机重建验证；支持 Quorum 协议下的主备复制验证；不支持 DCF 协议下的主备复制验证。
- UUID 验证功能主要为了防止主、备误连导致的数据串扰和污染，不是用于安全目的。
- 该参数不支持主、备间自动同步。

取值范围：枚举类型

off 表示关闭 UUID 验证功能。

default 表示关闭 UUID 验证功能。

uuid 表示开启 UUID 验证功能。

默认值：default

23.6.1.10 repl_uuid

参数说明：设置用于主备 UUID 验证的 UUID 码。

该参数属于 SIGHUP 类型参数，请参考表 15-1 中对应设置方法进行设置。

须知：

- 如果主机上开启了 UUID 验证功能、且配置了非空字符串的 repl_uuid 验证码，那么备机也需要开启 UUID 验证功能、且配置相同的 repl_uuid 验证码，否则主备日志复制和备机重建请求将被主机拒绝。
- 该参数支持 SIGHUP 动态加载新值。修改之后，不影响已建连的主备连接，对后续主备复制请求和主备重建请求生效。
- 支持 Quorum、DCF 协议下的备机重建验证；支持 Quorum 协议下的主备复制验证；不支持 DCF 协议下的主备复制验证。
- UUID 验证功能主要为了防止主、备误连导致的数据串扰和污染，不是用于安全目

的。

- 该参数不支持主、备间自动同步。

取值范围：字符串类型。长度 0 – 63 个字符，字母和数字的组合，大小写不敏感，内部统一转换为小写存储。空字符串表示不启用 UUID 验证功能。

默认值：空字符串

23.6.1.11 replconninfo1

参数说明：设置本端侦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 15-1 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

23.6.1.12 replconninfo2

参数说明：设置本端侦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 15-1 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

23.6.1.13 replconninfo3

参数说明：设置本端侦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 15-1 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

23.6.1.14 replconninfo4

参数说明：设置本端侦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 15-1 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

23.6.1.15 replconninfo5

参数说明：设置本端侦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 15-1 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

23.6.1.16 replconninfo6

参数说明：设置本端侦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 15-1 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

23.6.1.17 replconninfo7

参数说明：设置本端侦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 15-1 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

23.6.1.18 replconninfo8

参数说明：设置本端侦听和鉴权的第八个节点信息。

该参数属于 SIGHUP 类型参数，请参考表 15-1 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八个节点信息。

默认值：空字符串

23.6.1.19 cross_cluster_replconninfo1

参数说明：设置跨集群的本端侦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 15-1 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

23.6.1.20 cross_cluster_replconninfo2

参数说明：设置跨集群的本端侦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 15-1 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

23.6.1.21 cross_cluster_replconninfo3

参数说明：设置跨集群的本端侦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 15-1 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

23.6.1.22 cross_cluster_replconninfo4

参数说明：设置跨集群的本端侦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 15-1 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

23.6.1.23 cross_cluster_replconninfo5

参数说明：设置跨集群的本端侦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 15-1 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

23.6.1.24 cross_cluster_replconninfo6

参数说明：设置跨集群的本端侦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 15-1 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

23.6.1.25 cross_cluster_replconninfo7

参数说明：设置跨集群的本端侦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 15-1 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

23.6.1.26 cross_cluster_replconninfo8

参数说明：设置跨集群的本端侦听和鉴权的第八个节点信息。

该参数属于 SIGHUP 类型参数，请参考表 15-1 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八个节点信息。

默认值：空字符串

23.6.1.27 available_zone

参数说明：设置本端节点所在区域信息。

该参数属于 POSTMASTER 类型参数，请参考表 15-1 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节点信息。

默认值：空字符串

23.6.2 主服务器

23.6.2.1 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 15-1 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*，表示匹配任意提供同步复制的备机名称。支持按如下格式配置：

- ANY num_sync(standby_name [, …]) [, ANY num_sync (standby_name [, …])]
- [FIRST] num_sync (standby_name [, …])
- standby_name [, …]

说明：- 其中 num_sync 是事务需要等待其回复的同步复制的备机的数量，standby_name 是备机的名称，FIRST 以及 ANY 指定从所列服务器中选取同步复制的备机的策略。

- ANY N (node1,node2,…) 表示在括号内任选 N 个主机名称作为同步复制的备机名称列表。例如，ANY 1 (node1,node2) 表示在 node1 和 node2 中任选一个作为同步复制的备机名称。
- ANY N1 (node1,node2,…), ANY N2 (node3,node4,…) 表示分组潜在同步复制的备机名称列表，在第一组括号内任选 N1 个主机名称作为第一组同步复制的备机名称列表，在第二组括号内任选 N2 个主机名称作为第二组同步复制的备机名称列表。此时两个分组之间为且关系，必须两个分组均达到各自需求的同步备机数，本地事务才可以被提交。

- FIRST N (node1,node2,…)表示在括号内按出现顺序的先后作为优先级选择前 N 个主机名称作为同步复制的备机名称列表。例如，FIRST 1 (node1,node2)表示选择 node1 作为同步复制的备机名称。
- node1,node2,…和 FIRST 1 (node1,node2,…) 具有的含义相同。
- 若使用 gs_guc 工具设置该参数，需要如下设置：

```
gs_guc reload -Z datanode -N @NODE_NAME@ -D @DN_PATH@ -c "synchronous_standby_names='ANY NODE 1(dn_instanceId1, dn_instanceId2)'";
```

或者：

```
gs_guc reload -Z datanode -N @NODE_NAME@ -D @DN_PATH@ -c "synchronous_standby_names='ANY 1(AZ1, AZ2)'";
```

默认值：*

说明：

- 备机名称列表中不可出现重复的名称，配置中 num_sync 不可大于备机列表数量。
- 多分组同步备机配置 如 ANY N1 (node1,node2,…), ANY N2 (node3,node4,…) 的时候，多个分组之间为且关系，当前仅支持多 ANY 分组。不允许使用 * 来作为模糊匹配，不允许出现配置重复的备机。

23.6.2.2 most_available_sync

参数说明：在有同步备机故障时，主机事务不因同步备机故障而被阻塞。比如有两个同步备机，一个故障，另一个正常，这个时候主机事务只会等好的这个同步备，而不被故障的同步备所阻塞； 再比如走 quorum 协议时，一主三同步备，配置 ANY 2(node1,node2,node3)，当 node1、node3 故障，node2 正常时，主机业务同样不被阻塞。

该参数属于 SIGHUP 类型参数，请参考表 15-1 中对应设置方法进行设置。

取值范围：布尔型

- on 表示在有同步备机故障时，不阻塞主机。

- off 表示在有同步备机故障时，阻塞主机。

  默认值：off

23.6.2.3 keep_sync_window

参数说明：延迟进入最大可用模式的时间。

- 当最大可用模式 most_available_sync 配置为 on，在主备场景下，当存在同步备发生故障，导致不满足当前所配置的同步备数量(详细可参考 synchonous_standby_name 的含义)时，如果配置了 keep_sync_window 参数，则在 keep_sync_window 设置的时间窗口内，继续保持最大保护模式，即阻塞主机的事务提交，延缓进入最大可用模式的时间。
- 若在 keep_sync_window 超时窗口内，同步备机故障恢复，且满足当前所配置的同步备数量，则不阻塞事务，恢复到正常状态.
- 如果设置 keep_sync_window，推荐最小配置为 5s，以避免监控系统监控到网络不稳定的误报。

  该参数属于 SIGHUP 类型参数，请参考表 15-1 中对应设置方法进行设置。

  取值范围：整形，范围 0~INT_MAX，单位为秒。

  0 表示不设置 keep_sync_window 超时时间窗口，即直接进入最大可用模式。

  其余表示 keep_sync_window 超时时间窗口的大小。

  默认值：0

须知：

配置该参数可能会对 RPO 造成影响，若主机在所配置的超时时间窗口内发生故障，则从开始阻塞到主机故障这段时间窗口内的数据可能丢失。

23.6.2.4 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 15-1 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。
- 当前版本默认不支持主备从部署模式。

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

23.6.2.5 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。
- 当前版本默认不支持主备从部署模式。

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

23.6.2.6 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 15-1 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

23.6.2.7 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 15-1 中对应设置方法进行设置。

取值范围：整型，4096~1072693248，单位为 KB。

默认值：16MB（即 16384KB）

23.6.2.8 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 15-1 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）

23.6.2.9 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 15-1 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

23.6.2.10 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 15-1 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

23.6.2.11 enable_incremental_catchup

参数说明：控制主备之间数据追赶（catchup）的方式，目前默认不支持主备从部署模式。

该参数属于 SIGHUP 类型参数，请参考表 15-1 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

23.6.2.12 wait_dummy_time

参数说明：同时控制增量数据追赶（catchup）时，GBase 8c 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 15-1 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。
- 当前版本默认不支持主备从部署模式。

23.6.2.13 catchup2normal_wait_time

参数说明：打开最大可用模式 most_available_sync，主备场景下，控制备机数据追赶（catchup）阻塞主机的最长时间。该时间为估算值，实际结果可能与参数值有偏差。

该参数属于 SIGHUP 类型参数，请参考表 15-1 中对应设置方法进行设置。

取值范围：整型，范围-1~10000，单位为毫秒。

- -1 表示主机阻塞直到备机数据追赶完成。
- 0 表示备机数据追赶时始终不阻塞主机。
- 其余值表示备机数据追赶时阻塞主机的最长时间。例如，取值 5000，表示当备机数据追赶完成时间还剩 5s 时，阻塞主机等待其完成。

默认值：-1

23.6.2.14 sync_config_strategy

参数说明：主机和备机、备机和级联备之间配置文件的同步策略。

该参数属于 POSTMASTER 类型参数，请参考表 15-1 中对应设置方法进行设置。

取值范围：枚举类型

- all_node: 主机配置为 all_node 时，表示允许主机向所有备机主动同步配置文件；备机配置为 all_node 时，表示允许当前备机向其主机发送同步请求，允许当前备机向其所有级联备主动同步配置文件；级联备配置为 all_node 时，表示允许当前级联备向上一级备机发送同步请求。
- only_sync_node: 主机配置为 only_sync_node 时，表示仅允许主机向所有同步备机主动同步配置文件；备机配置为 only_sync_node 时，表示允许当前备机向其主机发送同步请求，不允许当前备机向其所有级联备机主动同步配置文件；级联备配置为

only_sync_node 时，表示允许当前级联备向其备机发送同步请求。

- none_node: 主机配置为 none_node 时，表示不允许主机向任何备机主动同步配置文件；备机配置为 none_node 时，表示不允许当前备机向其主机发送同步请求，不允许当前备机向其所有级联备主动同步配置文件；级联备配置为 none_node 时，表示不允许当前级联备向其备机发送同步请求。

默认值：all_node

须知：

- 在一个包含了主机、备机和级联备的 GBase 8c 集群中，主机相对于备机是发送端，备机相对于主机是接收端，备机相对于级联备是发送端，级联备相对于备机是接收端。
- 发送端主动向接收端同步配置文件、接收端请求发送端同步配置文件是两个独立的事件，均会使得配置文件同步。若不希望配置文件同步，则需要将集群中所有节点的 sync_config_strategy 参数配置为 none_node；若仅希望主机与同步备机同步配置文件，则需要将主机的 sync_config_strategy 参数配置为 only_sync_node，其余节点配置为 none_node；若希望所有节点同步配置文件，则需要将所有节点的 sync_config_strategy 参数配置为 all_node。目前暂不支持自定义指定任意节点间的同步策略。
- 配置参数同步的具体表现为，发送端发送配置文件，对接收端配置文件中的对应参数直接覆盖。若设置了配置文件需要同步的策略，则修改接收端配置参数后，发送端会立刻覆盖接收端的配置参数，使得接收端修改不生效。
- 即使设置了配置文件需要同步的策略，仍有部分配置参数不会被同步。它们是：

“application_name”、 “archive_command”、 “audit_directory”、 “available_zone”、“comm_control_port”、“comm_sctp_port”、“listen_addresses”、“log_directory”、“port”、“replconninfo1”、“replconninfo2”、“replconninfo3”、“replconninfo4”、“replconninfo5”、“replconninfo6”、“replconninfo7”、“replconninfo8”、“ssl”、“ssl_ca_file”、“ssl_cert_file”、

“ssl_ciphers”、“ssl_crl_file”、“ssl_key_file”、“ssl_renegotiation_limit”、“ssl_cert_notify_time”、“synchronous_standby_names”、“local_bind_address”、“perf_directory”、“query_log_directory”、“asp_log_directory”、“streaming_router_port”、“enable_upsert_to_merge”、“archive_dest”、“recovery_min_apply_delay”、“sync_config_strategy”。

23.6.2.15 enable_save_confirmed_lsn

参数说明： 启用该参数后，主机会将每次事务操作时与当前同步备达成多数派一致性的位置持久化到磁盘上。当主机发生故障后，原主作为备机发起 build 时，检测源端（新主）是否存在相同的 confirmed LSN。如果不存在，build 失败，避免原主的数据被 build 覆盖。

该参数属于 POSTMASTER 类型参数，请参考表 15-1 中对应设置方法进行设置。

取值范围： 布尔型

on: 表示启用该功能。在 1 主多备且配置有同步备机的场景下，在主机每次执行数据变化的事务操作时（DML/DDL），且主机等待与同步备机达成多数派一致性位置时，将与当前同步备达成多数派一致性的位置持久化到磁盘上。持久化的文件对应同步备的复制槽的状态文件。该参数打开后同时影响不指定模式的自动 build 和增量 build，当主机发生故障后，原主作为备机发起 build 时，检测源端（新主）是否存在相同的 confirmed LSN。如果不存在，build 失败，避免原主的数据被 build 覆盖。

off: 表示不启用该功能，主机事务提交时的行为与原来保持一致。自动 build 和增量 build 的行为与原来保持一致。此时在 1 主多备且配置有异步备机的场景下，如果主机突然发生故障宕机，而主机此刻达成的多数派一致性位置（比如 LSN100）又没有同步到异步备机时，如果强行将异步备机作为新主机启动，且在新主上执行一些事务操作，那么新主上的数据会覆盖 LSN100，此时再将原主作为备机发起 build，主机上会丢失自己最近一次达成多数派一致性位置 LSN100 的业务数据。

默认值： off

须知：

如果最大可用模式 most_available_sync 配置为 on，且所有同步备机都故障时，该功能不生效。因为没有同步备可以触发该 LSN 的持久化。

该功能只会影响增量 build 或不指定 build 模式的自动 build，如果用户强制指定全量 build 模式，该功能不生效。

如果在执行 build 前，主机的 pg_replslot 下的文件被人为删除或破坏，本功能不生效。

该功能开启后，如果主机在等待同步备机达成多数派一致性的过程中被主动停止，不会提示“该事务已在本地提交，可能未同步到远端”，避免上层业务以为数据已经达成一致。

该功能开启后，因为等待同步的时间会由于持久化数据而变长，带有同步备的主备集群的性能会受到影响。测试数据显示，与不开启该功能相比，性能约下降 20%。

23.6.3 备服务器

23.6.3.1 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 15-1 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。 - 如果 hot_standby 参数曾经被关闭，且 wal_level 参数曾被设置低于 hot_standby 等级，那么，再次打开 hot_standby 参数之前，为了确保主备环境下备机上待回放的日志都可以支持备机查询功能，需要进行如下操作： 1. 将主、备的 wal_level 参数调整到 hot_standby 等级或以上，并重启实例生效。 2. 在主机上执行 checkpoint 操作，并通过查询 pg_stat_get_wal_senders()系统函数，确认各个备机的 receiver_replay_location 追上主机当前的 sender_flush_location，保证 wal_level 的调整同步到备机并生效，且备机不需要再回放之前低等级的日志。 3. 将主、备的 hot_standby 参数打开（设为 on），并重启实例生效。

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

23.6.3.2 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 15-1 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）

23.6.3.3 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 15-1 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）

23.6.3.4 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 15-1 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）

须知：

当该参数设置为 0 时，表示关闭备机向主机反馈日志接收位置等信息，可能会导致主机事务提交阻塞、switchover 操作失败等异常现象。正常业务场景，不建议将该参数设置为 0。

23.6.3.5 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 15-1 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。

23.6.3.6 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 15-1 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）

23.6.3.7 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 15-1 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

23.6.3.8 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 15-1 中对应设置方法进行设置。

取值范围：整型，1~ INT_MAX。

默认值：1

23.6.3.9 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，目前默认不支持主备从部署模式。

该参数属于 POSTMASTER 类型参数，请参考表 15-1 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）

23.6.3.10 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 15-1 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

23.6.3.11 max_logical_replication_workers

参数说明：订阅端 apply worker 线程的最大数量。

该参数属于 POSTMASTER 类型参数，请参考表 15-1 中对应设置方法进行设置。

取值范围：整型，0~262143

默认值：4

23.7 内存表

介绍内存表的配置参数。

23.7.1.1 enable_codegen_mot

参数说明：设置是否启用原生 LLVM Lite 执行简单查询。如果当前平台上不支持原生 LLVM，那么将使用伪 LLVM。

该参数属于 POSTMASTER 类型参数，请参考表 15-1 中对应设置方法进行设置。

取值范围：布尔值

默认值：true

23.7.1.2 force_pseudo_codegen_mot

参数说明:设置是否强制伪 LLVM Lite 执行简单查询,即使当前平台上支持原生 LLVM。

该参数属于 POSTMASTER 类型参数，请参考表 15-1 中对应设置方法进行设置。

取值范围：布尔值

默认值：true

说明:

即使将 force_pseudo_codegen_mot 设置为 true，而当前平台不支持原生 LLVM，那么仍然会使用伪 LLVM。

23.7.1.3 enable_codegen_mot_print

参数说明：设置是否打印生成函数的 IR 字节码（如果使用伪 LLVM，则打印伪 IR 字节码）。

该参数属于 POSTMASTER 类型参数，请参考表 15-1 中对应设置方法进行设置。

取值范围：布尔值

默认值：true

23.7.1.4 codegen_mot_limit

参数说明：设置全局缓存计划源的数量限制以及每个会话的克隆计划。

该参数属于 POSTMASTER 类型参数，请参考表 15-1 中对应设置方法进行设置。

取值范围：uint32

默认值：100

23.7.1.5 mot_allow_index_on_nullable_column

参数说明：设置是否允许在内存表 nullable 列上创建索引。

该参数属于 POSTMASTER 类型参数，请参考表 15-1 中对应设置方法进行设置。

取值范围：布尔值

默认值：true

23.7.1.6 mot_config_file

参数说明：指定 MOT 的主配置文件。

该参数属于 POSTMASTER 类型参数，请参考表 15-1 中对应设置方法进行设置。

取值范围：字符串

默认值：NULL

23.8 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。

23.8.1 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。

23.8.1.1 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 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

23.8.1.2 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 15-1 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

23.8.1.3 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 15-1 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

23.8.1.4 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 15-1 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

23.8.1.5 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 15-1 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

23.8.1.6 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 15-1 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

23.8.1.7 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 15-1 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

23.8.1.8 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 15-1 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

23.8.1.9 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 15-1 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

23.8.1.10 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 15-1 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

23.8.1.11 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 15-1 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

23.8.1.12 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 15-1 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

23.8.1.13 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 15-1 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

23.8.1.14 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 15-1 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

23.8.1.15 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 15-1 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

23.8.1.16 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 15-1 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

23.8.1.17 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 15-1 中对应设置方法进行设置。

取值范围：整型，0~INT_MAX。

默认值：200

23.8.1.18 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 15-1 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

23.8.1.19 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 15-1 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

23.8.1.20 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 15-1 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

23.8.1.21 enable_valuepartition_pruning

参数说明：控制是否对 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 15-1 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

23.8.1.22 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 15-1 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

23.8.1.23 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 15-1 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

23.8.1.24 default_limit_rows

参数说明：设置生成 genericplan 的缺省 limit 估算行数。此参数设置为正数时表示直接将设置的值作为估算 limit 的行数，设置为负数时表示使用百分比的形式设置默认的估算值，负数转换为默认百分比，即-5 代表 5%。

该参数属于 USERSET 类型参数，请参考表 15-1 中对应设置方法进行设置。

取值范围：浮点型，-100~DBL_MAX。

默认值：-10

23.8.1.25 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 15-1 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

23.8.1.26 cost_weight_index

参数说明：设置 index_scan 的代价权重。

该参数属于 USERSET 类型参数，请参考表 15-1 中对应设置方法进行设置。

取值范围：浮点型，1e-10~1e+10。

默认值：1

23.8.1.27 try_vector_engine_strategy

参数说明：设置行存表走向量化执行引擎的策略。通过设置该参数，可以使包含行存表的查询可以转换为向量化的执行计划执行计算，从而提升类 AP 场景的复杂查询的执行性能。

该参数属于 USERSET 类型参数，请参考表 15-1 中对应设置方法进行设置。

取值范围：枚举型

- off，为默认取值，表示关闭本功能，即行存表不会转换为向量的执行计划执行。
- force，表示只要查询中不包含向量化引擎不支持的类型或者表达式，则不论查询的基表为行存表、列存表，还是行列混合存储的，强制将查询转换为向量化的执行计划执行计算。在这种情况下，针对不同的查询场景可能出现性能下降。
- optimal，表示在 force 的基础上，由优化器根据查询的复杂度进行选择是否将查询语句转换为向量化的执行计划，尽可能避免转换为向量化的执行计划后出现性能下

降。

默认值：off

23.8.2 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。

23.8.2.1 seq_page_cost

参数说明：设置优化器计算一次顺序磁盘抓取页面的开销。

该参数属于 USERSET 类型参数，请参考表 15-1 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

23.8.2.2 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 15-1 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。

23.8.2.3 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 15-1 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

23.8.2.4 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 15-1 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

23.8.2.5 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 15-1 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

23.8.2.6 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 GBase 8c 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 GBase 8c 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 15-1 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

23.8.2.7 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 15-1 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

23.8.3 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。

23.8.3.1 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 15-1 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

23.8.3.2 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 15-1 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

23.8.3.3 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 15-1 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

23.8.3.4 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 15-1 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 00，表示使用系统自适应方式，GBase 8c 会基于 geqo_effort 和表的个数选取合适的值。

默认值：0

23.8.3.5 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 15-1 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

23.8.3.6 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 15-1 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

23.8.3.7 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 15-1 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

23.8.4 其他优化器选项

23.8.4.1 enable_stream_recursive

参数说明：控制 with recursive 语句是否允许使用 stream 分布式执行计划。

取值范围：on、off

默认值：off。表示默认不使用 stream 分布式执行计划。

23.8.4.2 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 15-1 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

23.8.4.3 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 15-1 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 GBase 8c 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：代表是在 pretty 的基础上增加了对打印信息的分析。
- run：代表在 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（当前版本参数取值仅 normal 生效，若设置为非 normal，显示格式依然为 normal）

23.8.4.4 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 15-1 中对应设置方法进行设置。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, …)'。

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。

说明： pretty 模式当前版本不支持。

默认值：ALL,on(),off(LLVM_COMPILE、HASH_CONFLICT、STREAM_DATA_CHECK)，不开启任何定位功能。

23.8.4.5 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。目前，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 15-1 中对应设置方法进行设置。

取值范围：整型，0~INT_MAX

默认值：0

23.8.4.6 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 15-1 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

23.8.4.7 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 15-1 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）。
- magicset：使用 Magic Set 查询重写规则（从主查询中下推条件到子查询）。
- partialpush：使用 Partial Push 查询重写规则。
- uniquecheck：使用 Unique Check 查询重写规则（提升目标列中无 agg 的子查询语句，在执行时检查返回行数是否为 1 行）。
- disablerep：使用 Disable Replicate 查询重写规则。
- intargetlist：使用 In Target List 查询重写规则（提升目标列中的子查询）。
- predpushnormal：使用 Predicate Push 查询重写规则（下推谓词条件到子查询中）。
- predpushforce：使用 Predicate Push 查询重写规则（下推谓词条件到子查询中，尽可能的利用索引加速）。
- predpush：在 predpushnormal 和 predpushforce 中根据代价选择最优计划。

默认值：magicset

23.8.4.8 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 15-1 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

23.8.4.9 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。

在打开 enable_global_plancache 的情况下，为保证 GPC 生效，默认 local_syscache_threshold 不小于 16MB。即如当前 local_syscache_threshold 小于 16MB，则设置为 16MB，如大于 16MB，则不改变。

该参数属于 POSTMASTER 类型参数，请参考表 15-1 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

23.8.4.10 gpc_clean_timeout

参数说明：开启 enable_global_plancache 的情况下，如果共享计划列表里的计划超过 gpc_clean_timeout 的时间没有被使用，则会被清理掉。本参数用于控制没有使用的共享计划的保留时间。

该参数属于 SIGHUP 类型参数，请参考表 15-1 中对应设置方法进行设置。

取值范围：整型，300\~86400，单位为秒

默认值：1800，即 30min

23.8.4.11 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 GBase 8c

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 15-1 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

23.8.4.12 enable_opfusion

参数说明：控制是否对简单增删改查进行优化。

该参数属于 USERSET 类型参数，请参考表 15-1 中对应设置方法进行设置。

简单查询限制如下：

- 只支持 indexscan 和 indexonlyscan，且全部 WHERE 语句的过滤条件都在索引上。
- 只支持单表增删改查，不支持 join、using。
- 只支持行存表，不支持分区表，表不支持有触发器。
- 不支持 active sql、QPS 等信息统计特性。
- 不支持正在扩容和缩容的表。
- 不支持查询或者修改系统列。
- 只支持简单 SELECT 语句，例如：

```
SELECT c3 FROM t1 WHERE c1 = ? and c2 =10;
```

仅可以查询目标表的列，c1 和 c2 列为索引列，后边可以是常量或者参数，可以使用 for update。

- 只支持简单 INSERT 语句，例如：

```
INSERT INTO t1 VALUES (?,10,?);
```

仅支持一个 VALUES，VALUES 里面的类型可以是常量和参数，不支持 returning。

- 只支持简单 DELETE 语句，例如：

```
DELETE FROM t1 WHERE c1 = ? and c2 = 10;
```

c1 和 c2 列为索引列，后边可以是常量或者参数。

- 只支持简单 UPDATE 语句，例如：

```
UPDATE t1 SET c3 = c3+? WHERE c1 = ? and c2 = 10;
```

c3 列修改的值可以是常量和参数，也可以是一个简单的表达式，c1 和 c2 列为索引列，后边可以是常量或者参数。

取值范围：布尔型

- on 表示使用。
- off 表示不使用。

默认值：on

23.8.4.13 enable_partition_opfusion

参数说明：在 enable_opfusion 参数打开的状态下，如果开启该参数，可以对分区表的简单查询进行查询优化，提升 SQL 执行性能。

该参数属于 USERSET 类型参数，请参考表 15-1 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

23.8.4.14 sql_beta_feature

参数说明：标识开启的可选 SQL 引擎 Beta 特性，其中包括对行数估算、查询等价估算等优化。

开启它们可以对特定的场景进行优化，但也可能会导致部分没有被测试覆盖的场景发生

性能劣化。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 SQL 引擎 Beta 特性的组合，比如有多个 Beta 特性：feature1、feature2、feature3、feature4。可以设置：

```
--启用 SQL 引擎 Beta 特性 feature1。
set  sql_beta_feature=feature1;
--启用 SQL 引擎 Beta 特性 feature2 和 feature3。
set  sql_beta_feature=feature2,feature3;
--关闭所有可选 SQL 引擎 Beta 特性。
set  sql_beta_feature=none;
```

该参数属于 USERSET 类型参数，请参考表 15-1 中对应设置方法进行设置。

取值范围：字符串

- none：不使用任何 Beta 优化器特性。
- sel_semi_poisson：使用泊松分布对等值的半连接和反连接选择率进行校准。
- sel_expr_instr：使用字符串匹配的行数估算方法对 instr(col, 'const') > 0、= 0、 = 1 进行更准确的估算。
- param_path_gen：生成更多可能的参数化路径。
- rand_cost_opt：对小数据量表的随机读取代价进行优化。
- param_path_opt：利用表的膨胀系数优化索引 analyze 信息。
- page_est_opt：优化对非列存表索引 analyze 信息的 relpages 估算。
- no_unique_index_first：关闭主键索引扫描路径优先的优化。
- join_sel_with_cast_func：估算 join 行数的时候支持类型转换函数。
- canonical_pathkey：正则化 pathkey 生成置后（pathkey 是指标记数据有序性键值的集合）。

- index_cost_with_leaf_pages_only：估算索引代价时考虑索引叶子结点。
- partition_opfusion：开启分区表优化。
- a_style_coerce：开启 Decode 类型转换规则兼容 O，详见对于 case，在 ORA 兼容模式下的处理。
- partition_fdw_on：支持基于分区表创建 postgres foreign table 下的相关 SQL。
- predpush_same_level：开启 predpush hint 控制同层参数化路径的功能。

默认值：none

23.8.4.15 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 15-1 中对应设置方法进行设置。

取值范围：整型，1~4

默认值：2

23.8.4.16 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 15-1 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

23.8.4.17 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 15-1 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

23.8.4.18 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。此参数设置为正数时，代表统计信息的样本数量；设置为负数时，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 15-1 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：
  1. 恢复默认的统计信息。
  2. 使用 plan hint 来调整到之前的查询计划。（详细参见使用 Plan Hint 进行调优）
- 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

23.8.4.19 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 15-1 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。如果所有的表都打开它，对于简单的查询强加了额外的规划，并且对简单查询没有益处。如果不用分区表，可以关掉它。

23.8.4.20 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 15-1 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

23.8.4.21 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 15-1 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

23.8.4.22 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 15-1 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

23.8.4.23 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 guc 参数值为-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。 如果设置 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 15-1 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。

23.8.4.24 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 15-1 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

23.8.4.25 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 15-1 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

23.8.4.26 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 15-1 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

23.8.4.27 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 15-1 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

23.8.4.28 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 15-1 中对应设置方法进行设置。

取值范围：整型，0～2147483647。

默认值：10000

23.8.4.29 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 15-1 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

23.8.4.30 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 15-1 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

23.8.4.31 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 15-1 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

23.8.4.32 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 15-1 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：off

23.8.4.33 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 15-1 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”，Explain 详解请参见详解。

默认值：on

23.8.4.34 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 15-1 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”，Explain 详解请参见详解。

默认值：on

23.8.4.35 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 15-1 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

23.8.4.36 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 15-1 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

23.8.4.37 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 15-1 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

23.8.4.38 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 15-1 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

23.8.4.39 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 15-1 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

23.8.4.40 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 15-1 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

23.8.4.41 enable_hypo_index

参数说明：控制优化器执行 EXPLAIN 命令时是否考虑虚拟索引。

该参数属于 USERSET 类型参数，请参考表 15-1 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

23.8.4.42 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 15-1 中对应设置方法进行设置。

取值范围：布尔型

- on 表示可以向量化的算子强制生成向量化。
- off 表示由向量化算子优化器决定是否向量化。

默认值：off

23.8.4.43 enable_auto_explain

参数说明：控制是否开启自动打印执行计划。该参数是用来定位慢存储过程或慢查询，只对当前连接的数据库主节点有效。

该参数属于 USERSET 类型参数，请参考表 15-1 中对应设置方法进行设置。

取值范围：布尔型

- true 表示开启。
- false 表示关闭。

默认值：false

23.8.4.44 auto_explain_level

参数说明：控制自动打印执行计划的日志等级。

该参数属于 USERSET 类型参数，请参考表 15-1 中对应设置方法进行设置。

取值范围：枚举型，LOG 或 NOTICE。NOTICE 表示以提示知的形式打印出计划。

- LOG 表示在日志中打印执行计划。
- NOTICE 表示以提示知的形式打印出计划。

默认值：LOG

23.8.4.45 query_dop

参数说明：用户自定义的查询并行度。

该参数属于 USERSET 类型参数，请参考表 15-1 中对应设置方法进行设置。

取值范围：整型，1~64。打开固定 SMP 功能，系统会使用固定并行度。

说明： 在开启并行查询后，请保证系统 CPU、内存、网络等资源充足，以达到最佳效果。

默认值：2

23.8.4.46 enable_startwith_debug

参数说明:该参数为 start with/connect by 用于 debug 的参数,打开参数可以显示 start with 特性所有涉及的尾列相关信息。

该参数属于 USERSET 类型参数，请参考表 15-1 中对应设置方法进行设置。

取值范围：布尔型

- true 表示开启。
- false 表示关闭。

默认值：false

23.9 错误报告和日志

23.9.1 记录日志的位置

23.9.1.1 log_destination

参数说明：GBase 8c 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 15-1 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、eventlog。

- 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。GBase 8c 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/gbase
```

默认值：stderr

23.9.1.2 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 15-1 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

23.9.1.3 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）。log_directory 支持动态修改，可以通过 gs_guc reload 实现。

该参数属于 SIGHUP 类型参数，请参考表 15-1 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 GBase 8c 无法重新启动。
- 通过 gs_guc reload 动态修改 log_directory 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。
- 在沙箱环境，路径中不可以包含/var/chroot，例如 log 的绝对路径是

/var/chroot/var/lib/log/Ruby/pg_log/cn_log ， 则 只 需 要 设 置 为 /var/lib/log/Ruby/pg_log/cn_log。

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。

23.9.1.4 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 15-1 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

23.9.1.5 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 15-1 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

23.9.1.6 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 15-1 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 GBase 8c 以覆盖写入的方式写服务器日志消息。
- off 表示 GBase 8c 将日志消息附加到同名的现有日志文件上。

默认值：off

23.9.1.7 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 15-1 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)

23.9.1.8 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 15-1 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

23.9.1.9 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 15-1 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

23.9.1.10 syslog_ident

参数说明：log_destination 设置为 syslog 时，syslog_ident 设置在 syslog 日志中 GBase 8c 日志消息的标识。

该参数属于 SIGHUP 类型参数，请参考表 15-1 中对应设置方法进行设置。

取值范围：字符串

默认值：postgres

23.9.1.11 event_source

参数说明：该参数仅在 windows 环境下生效，GBase 8c 暂不支持。log_destination 设置为 eventlog 时，event_source 设置在日志中 GBase 8c 日志消息的标识。

该参数属于 POSTMASTER 类型参数，请参考表 15-1 中对应设置方法进行设置。

取值范围：字符串

默认值：PostgreSQL

23.9.2 记录日志的时间

23.9.2.1 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 15-1 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 15-1。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

23.9.2.2 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 15-1 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。

部分日志信息的打印需要同时配置该参数与 logging_modules，即设置该参数打开后可能还需要设置 logging_modules 打开对应模块的日志打印开关。

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 15-1。

默认值：warning

23.9.2.3 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 15-1 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 15-1。

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

23.9.2.4 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 15-1 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

23.9.2.5 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 15-1 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 15-1。

默认值：panic

下表解释 GBase 8c 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，GBase 8c 版本不涉及该参数）时，GBase 8c 进行如表中的转换。

表 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
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

23.9.2.6 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 15-1 中对应设置方法进行设置。

须知：

该参数以毫秒为单位，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/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：0

23.9.3 记录日志的内容

23.9.3.1 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 15-1 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

23.9.3.2 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 15-1 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

23.9.3.3 debug_print_plan

参数说明：用于设置是否将查询的执行计划打印到日志中。

该参数属于 SIGHUP 类型参数，请参考表 15-1 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。

23.9.3.4 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 15-1 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

23.9.3.5 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 15-1 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

23.9.3.6 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 15-1 中对应设置方法进行设置。

须知：

有些客户端程序（例如 gsql），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

23.9.3.7 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 15-1 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

23.9.3.8 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 15-1 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

23.9.3.9 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 15-1 中对应设置方法进行设置。

取值范围：枚举类型

- terse 代表输出不包括 DETAIL、HINT、QUERY 及 CONTEXT 错误信息的记录。
- verbose 代表输出包括 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 代表输出包括 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

23.9.3.10 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 15-1 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

23.9.3.11 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 15-1 中的状态信息。

该参数属于 SIGHUP 类型参数，请参考表 15-1 中对应设置方法进行设置。

表 1 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
%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 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。
%S	会话 ID。
%%	字符%。

说明：

转义字符%c 打印一个会话 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 设置为非空值时，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。

23.9.3.12 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 15-1 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

23.9.3.13 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 15-1 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

23.9.3.14 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 15-1 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 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

23.9.3.15 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 15-1 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：PRC

说明：

gs_initdb 进行相应系统环境设置时会对默认值进行修改。

23.9.3.16 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，不建议通过 gs_guc 工具来设置。

该参数属于 USERSET 类型参数，设置请参考表 15-1 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：

```
postgres=# show logging_module;
logging_module
----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------
ALL,on(),off(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,T,PL
```

```
ANHINT,SNAPSHOT,XACT,HANDLE,CLOG,EC,REMOTE,CN_RETRY,PLSQL,TEXTSEAR
CH,SEQ,REDO,FUNCTION,PARSER,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：

```
    postgres=# set logging_module='on(SSL)';
    SET
    postgres=# show logging_module;
logging_module
    ----------------------------------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------------------------------
----------------------------------------------------------------
ALL,on(SSL),off(DFS,GUC,ORC,SLRU,MEM_CTL,AUTOVAC,CACHE,ADIO,TBLSPC,WL
M,EXECUTOR,VEC_EXECUTOR,LLVM,OPT,OPT_REWRITE,OPT_JOIN,OPT_AGG,OPT_S
UBPLAN,OPT_SETOP,OPT_SKEW,UDF,COOP_ANALYZE,WLMCP,ACCELERATE,,PLANH
INT,SNAPSHOT,XACT,HANDLE,CLOG,EC,REMOTE,CN_RETRY,PLSQL,TEXTSEARCH,S
EQ,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。

```
    postgres=# set logging_module='off(ALL)';
    SET
    postgres=# show logging_module;
logging_module
    ----------------------------------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------------------------------
----------------------------------------------------------------
ALL,on(),off(DFS,GUC,ORC,SLRU,MEM_CTL,AUTOVAC,CACHE,ADIO,SSL,TBLSPC,WL
M,EXECUTOR,VEC_EXECUTOR,LLVM,OPT,OPT_REWRITE,OPT_JOIN,OPT_AGG,OPT_S
UBPLAN,OPT_SETOP,OPT_SKEW,UDF,COOP_ANALYZE,WLMCP,ACCELERATE,PLANHI
NT,SNAPSHOT,XACT,HANDLE,CLOG,EC,REMOTE,CN_RETRY,PLSQL,TEXTSEARCH,SE
Q,REDO,FUNCTION,PARSER,INSTR,INCRE_CKPT,DBL_WRT,RTO,HEARTBEAT)
    (1 row)
```

```
postgres=# set logging_module='on(ALL)';
SET
postgres=# 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,PLANHINT,SNAPSHOT,XACT,HANDLE,CLOG,EC,REMOTE,CN_RETRY,PLSQL,TEXTSEARCH,SEQ,REDO,FUNCTION,PARSER,INSTR,INCRE_CKPT,DBL_WRT,RTO,HEARTBEAT),off()
(1 row)
```

依赖关系：该参数依赖于 log_min_messages 参数的设置。

23.9.3.17 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 15-1 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

23.9.3.18 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 15-1 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

23.9.4 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 gbase_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
);
```

详细说明请参见下表。

表 25-1csvlog 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
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）

字段名	字段含义	字段名	字段含义
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 GBase 8c 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 gbase_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。

23.10 告警检测

在 GBase 8c 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 GBase 8c 的错误。

23.10.1enable_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 15-1 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

23.10.2connection_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 15-1 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

23.10.3alarm_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 15-1 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

23.10.4alarm_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 15-1 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

23.10.5table_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 15-1 中对应设置方法进行设置。

取值范围：浮点型，0~1

默认值：1

23.10.6table_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 15-1 中对应设置方法进行设置。

取值范围：整型，0~INT_MAX

默认值：100000

23.11 运行时统计

23.11.1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。

23.11.1.1 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 15-1 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

23.11.1.2 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 15-1 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

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。

23.11.1.3 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 15-1 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

23.11.1.4 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 15-1 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

23.11.1.5 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 15-1 中对应设置方法进行设置。

取值范围：整型，100~102400

默认值：1024

23.11.1.6 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 15-1 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

23.11.1.7 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 15-1 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

23.11.1.8 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 15-1 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

23.11.1.9 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。

在 x86 平台集中式部署下，硬件配置规格为 32 核 CPU/256GB 内存，使用 Benchmark SQL 5.0 工具测试性能，开关此参数性能影响约 0.8%。

该参数属于 SUSET 类型参数，请参考表 15-1 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 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。

23.11.2 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 GBase 8c 提供的性能统计方法，可以方便定位性能问题。

23.11.2.1 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 15-1 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

23.12 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。GBase 8c 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。

23.12.1use_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 GUC 参数设置方式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 GUC 参数设置方式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：off

23.12.2cgroup_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 GUC 参数设置方式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

23.12.3cpu_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 GUC 参数设置方式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

23.12.4memory_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 GUC 参数设置方式中对应设置方法进行设置。

取值范围：

- none：表示不启动内存统计功能。
- peak：表示统计 query 级内存 peak 值，此数值计入数据库日志，也可以由 explain analyze 输出。
- normal：表示仅做内存实时统计，不生成文件。
- executor：表示生成统计文件，包含执行层使用过的所有已分配内存的上下文信息。
- fullexec：表示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

23.12.5memory_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 GUC 参数设置方式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。

23.12.6enable_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 GUC 参数设置方式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

23.12.7enable_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 GUC 参数设置方式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

23.12.8enable_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 GUC 参数设置方式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

23.12.9enable_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 GS_WLM_EC_OPERATOR_INFO 系统表中。

该参数属于 SIGHUP 类型参数，请参考表 GUC 参数设置方式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

23.12.10 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 GUC 参数设置方式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

23.12.11 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 GS_WLM_INSTANCE_HISTORY 系统表中。

该参数属于 SIGHUP 类型参数，请参考表 GUC 参数设置方式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

23.12.12 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 GUC 参数设置方式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

23.12.13 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 GUC 参数设置方式中对应设置方法进行设置。

取值范围：枚举型

- none：表示不开启资源监控功能。
- query：表示开启 query 级别资源监控功能。
- operator：表示开启 query 级别和算子级别资源监控功能。

默认值：query

23.12.14 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 GUC 参数设置方式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 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

23.12.15 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 GUC 参数设置方式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

23.12.16 disable_memory_protect

参数说明：设置是否禁止内存保护功能。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 GUC 参数设置方式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

23.12.17 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 GUC 参数设置方式中对应设置方法进行设置。

取值范围：字符型

默认值：空

23.12.18 memory_fault_percent

参数说明：内存故障测试时内存申请失败的比例，仅用在 DEBUG 版本。

该参数属于 SIGHUP 类型参数，请参考表 GUC 参数设置方式中对应设置方法进行设置。

取值范围：整型，0~2147483647

默认值：0

23.12.19 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 GUC 参数设置方式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：on

须知： 黑匣子功能生成 core 文件依赖操作系统开放 ptrace 接口。若发生权限不足(errno = 1)，请确保/proc/sys/kernel/yama/ptrace_scope 配置合理。

23.12.20 bbox_dump_count

参数说明：在 bbox_dump_path 定义的路径下，允许存储的 GBase 8c 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 GUC 参数设置方式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。

23.12.21 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 GUC 参数设置方式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/kernel/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。

23.12.22 bbox_blanklist_items

参数说明：黑匣子 core 文件的脱敏数据选项。此参数只有当 enable_bbox_dump 为 on 时才生效。

该参数属于 POSTMASTER 类型参数，请参考表 2 中对应设置方法进行设置。

取值范围：字符型，以逗号分隔的敏感数据选项的字符串。

默认值：空。表示 bbox 生成的 core 文件脱敏所有支持的敏感数据项。

目前支持脱敏的数据项：

- SHARED_BUFFER：buffer 数据缓冲区
- XLOG_BUFFER：redo 日志缓冲区
- DW_BUFFER：双写数据缓冲区
- XLOG_MESSAGE_SEND：主备日复制日志发送缓冲区
- WALRECIVER_CTL_BLOCK：主备复制日志接收缓冲区
- DATA_MESSAGE_SEND：主备复制数据发送缓冲区
- DATA_WRITER_QUEUE：主备复制数据接收缓冲区

23.12.23 enable_ffic_log

参数说明：是否开启 FFIC（First Failure Info Capture）功能。

该参数属于 POSTMASTER 类型参数，请参考表 GUC 参数设置方式中对应设置方法进行设置。

取值范围：布尔型

- on 表示打开 FFIC 功能。
- off 表示关闭 FFIC 功能。

默认值：on

23.12.24 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 GUC 参数设置方式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

23.12.25 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 GUC 参数设置方式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

23.12.26 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 GUC 参数设置方式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

23.12.27 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 GUC 参数设置方式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

23.12.28 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 GUC 参数设置方式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

23.12.29 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 GUC 参数设置方式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

23.12.30 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 GUC 参数设置方式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

23.12.31 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 15-1 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 时，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 时，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

23.13 自动清理

系统自动清理线程（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。

23.13.1 autovacuum

参数说明：控制数据库自动清理线程（autovacuum）的启动。自动清理线程运行的前提是将 track_counts 设置为 on。

该参数属于 SIGHUP 类型参数，请参考表 GUC 参数设置方式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线程。

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线程。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。

取值范围：布尔型

- on 表示开启数据库自动清理线程。
- off 表示关闭数据库自动清理线程。

默认值：on

23.13.2autovacuum_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 GUC 参数设置方式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

23.13.3autoanalyze_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 GUC 参数设置方式中对应设置方法进行设置。

取值范围：整型，0~2147483，单位是秒。

默认值：5min（即 300s）

23.13.4autovacuum_io_limits

参数说明：控制 autovacuum 线程每秒触发 IO 的上限。

该参数属于 SIGHUP 类型参数，请参考表 GUC 参数设置方式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

23.13.5log_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 GUC 参数设置方式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、非 0 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

23.13.6autovacuum_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 GUC 参数设置方式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 的值 - job_queue_processes 的值 - 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

23.13.7autovacuum_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 GUC 参数设置方式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）

23.13.8autovacuum_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 GUC 参数设置方式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

23.13.9autovacuum_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 GUC 参数设置方式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

23.13.10 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 GUC 参数设置方式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

23.13.11 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 GUC 参数设置方式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

23.13.12 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。
- 即使自动清理线程被禁止，系统也会调用自动清理线程来防止循环重复。

该参数属于 POSTMASTER 类型参数，请参考表 GUC 参数设置方式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：4000000000

23.13.13 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 GUC 参数设置方式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

23.13.14 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 GUC 参数设置方式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

23.13.15 defer_csn_cleanup_time

参数说明：用来指定本地回收时间间隔。

该参数属于 SIGHUP 类型参数，请参考表 GUC 参数设置方式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：5s（即 5000ms）

23.14 客户端连接缺省设置

23.14.1语句行为

介绍 SQL 语句执行过程的相关默认参数。

23.14.1.1 exclude_reserved_words

参数说明：该参数值表示需要进行转义的关键字，以实现关键词屏蔽功能。在用户使用数据库进行创建表时，会遇到创建的字段名与数据库中的关键词相冲突，默认创建时会报错。因此增加一个屏蔽数据库关键词的参数，避免与用户创建的表名、列名冲突。需要屏蔽的关键词例如：maxvalue、body、verify 等。

需要注意的是，如果屏蔽了数据库关键字，那该关键字相关的数据库对应功能应当无法使用。

该参数属于 SIGHUP 类型参数。

参数值类型：字符串。

默认值：空字符，表示不对任何关键字进行屏蔽。

使用说明：

（1）可设置多个关键字，使用英文逗号分隔。

（2）如果屏蔽了数据库关键字，那该关键字相关的数据库对应功能将无法使用。

23.14.1.2 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 GUC 参数设置方式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。

23.14.1.3 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 GUC 参数设置方式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。

23.14.1.4 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 GUC 参数设置方式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

23.14.1.5 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，GBase 8c 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，GBase 8c 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 GUC 参数设置方式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

23.14.1.6 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 GUC 参数设置方式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

23.14.1.7 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 GUC 参数设置方式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，GBase 8c 目前功能上不支持此隔离级别，等价于 repeatable read。

默认值：read committed

23.14.1.8 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 USERSET 类型参数，请参考表 GUC 参数设置方式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

23.14.1.9 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。

GBase 8c 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 GUC 参数设置方式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

23.14.1.10 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 GUC 参数设置方式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

23.14.1.11 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 GUC 参数设置方式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

23.14.1.12 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 GUC 参数设置方式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：2000000000

23.14.1.13 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 GUC 参数设置方式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

23.14.1.14 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 GUC 参数设置方式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

23.14.1.15 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 GUC 参数设置方式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

23.14.1.16 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 GUC 参数设置方式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

23.14.1.17 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 GUC 参数设置方式中对应设置方法进行设置。

取值范围：整型，1~2147483647。

默认值：1000

23.14.1.18 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 GUC 参数设置方式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

23.14.2 区域和格式化

介绍时间格式设置的相关参数。

23.14.2.1 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 GUC 参数设置方式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：

gs_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。

23.14.2.2 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 GUC 参数设置方式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

23.14.2.3 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 GUC 参数设置方式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：PRC

说明：

gs_initdb 将设置一个与其系统环境一致的时区值。

23.14.2.4 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 GUC 参数设置方式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。

23.14.2.5 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 GUC 参数设置方式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

23.14.2.6 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 GUC 参数设置方式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，gs_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，不建议通过 gs_guc 工具或其他方式直接在 postgresql.conf 文件中设置 client_encoding 参数，即使设置也不会生效，以保证 GBase 8c 内部通信编码格式一致。

默认值：UTF8

推荐值：SQL_ASCII/UTF8

23.14.2.7 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 GUC 参数设置方式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，gs_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

23.14.2.8 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 GUC 参数设置方式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，gs_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

23.14.2.9 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 GUC 参数设置方式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，gs_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

23.14.2.10 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 GUC 参数设置方式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，gs_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

23.14.2.11 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，gs_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 GUC 参数设置方式中对应设置方法进行设置。

取值范围：字符串

说明：

GBase 8c 支持 pg_catalog.english、pg_catalog.simple 两种配置。

默认值：pg_catalog.english

23.14.3其他缺省

主要介绍数据库系统默认的库加载参数。

23.14.3.1 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 GBase 8c 发布提供的模块安装路径。例如：

```
dynamic_library_path = '/usr/local/lib/gaussdb:/opt/testgs/lib:$libdir'
```

该参数属于 SUSET 类型参数，请参考表 GUC 参数设置方式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

23.14.3.2 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 GUC 参数设置方式中对应设置方法进行设置。

取值范围：整型，0~2147483647

默认值：0

23.14.3.3 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 GBase 8c 的库都有一个“magic block”用于确保兼容性，因此不支持 GBase 8c 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 GUC 参数设置方式中对应设置方法进行设置。

取值范围：字符串

默认值：空

23.15 锁管理

在 GBase 8c 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。

23.15.1 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。
- 当设置 log_lock_waits 为 on 时，deadlock_timeout 决定一个等待时间来将查询执行过程中的锁等待耗时信息写入日志。如果要研究锁延时情况，可以设置 deadlock_timeout 的值比正常情况小。

该参数属于 SUSET 类型参数，请参考表 GUC 参数设置方式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

23.15.2lockwait_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 GUC 参数设置方式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

23.15.3update_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 GUC 参数设置方式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

23.15.4max_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 GBase 8c 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 GUC 参数设置方式中对应设置方法进行设置。

取值范围：整型，10 ~ INT_MAX

默认值：256

23.15.5max_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 GBase 8c 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 GUC 参数设置方式中对应设置方法进行设置。

取值范围：整型，10 ~ INT_MAX

默认值：64

23.15.6gs_clean_timeout

参数说明：控制主节点周期性清理临时表的时间，是一个平均值。

- 数据库连接异常终止时，通常会有临时表残留，此时需要对数据库中的临时表进行清理。

- 增大这个参数可能导致 GBase 8c 临时表清理时间延长。

该参数属于 SIGHUP 类型参数，请参考表 GUC 参数设置方式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：1min

23.15.7partition_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 GUC 参数设置方式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

23.15.8fault_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 GUC 参数设置方式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

23.15.9enable_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 GBase 8c 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 GUC 参数设置方式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

23.15.10 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 GUC 参数设置方式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

23.15.11 num_internal_lock_partitions

参数说明：控制内部轻量级锁分区的个数。主要用于各类场景的性能调优。内容以关键字和数字的 KV 方式组织，各个不同类型锁之间以逗号隔开。先后顺序对设置结果不影响，例如“CLOG_PART=256,CSNLOG_PART=512”等同于“CSNLOG_PART=512,CLOG_PART=256”。重复设置同一关键字时，以最后一次设置为准，

例如“CLOG_PART=256,CLOG_PART=2”，设置的结果为 CLOG_PART=2。当没有设置关键字时，则为默认值，各类锁的使用描述和最大、最小、默认值如下。

- CLOG_PART：CLOG 文件控制器的个数。增大该值可以提高 CLOG 日志写入效率，提升事务提交性能，但是会增大内存使用；减小该值会减少相应内存使用，但可能使得 CLOG 日志写入冲突变大，影响性能。最小值为 1，最大值为 256。
- CSNLOG_PART：CSNLOG 文件控制器的个数。增大该值可以提高 CSNLOG 日志写入效率，提升事务提交性能，但是会增大内存使用；减小该值会减少相应内存使用，但可能使得 CSNLOG 日志写入冲突变大，影响性能。最小值为 1，最大值为 512。
- LOG2_LOCKTABLE_PART：常规锁表锁分区个数的 2 对数。增大该值可以提升正常流程常规锁获取锁的并行度，但是可能增加锁转移和锁消除时的耗时，对于等待事件在 LockMgrLock 时，可以调大该锁增加性能。最小值为 4，即锁分区数为 16；最大值为 16，即锁分区数为 65536。
- TWOPHASE_PART：两阶段事务锁的分区数。调大该值可以提高两阶段事务提交的并发数。最小值为 1，最大值为 64。

该参数属于 POSTMASTER 类型参数，参考表 GUC 参数设置方式中对应设置方法进行设置。

取值范围：字符串

默认值：

- CLOG_PART：256
- CSNLOG_PART：512
- LOG2_LOCKTABLE_PART：4
- TWOPHASE_PART：1

23.16 版本和平台兼容性

23.16.1历史版本兼容性

GBase 8c 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。

23.16.1.1 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 GUC 参数设置方式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

23.16.1.2 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 GUC 参数设置方式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

23.16.1.3 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 GUC 参数设置方式中对应设置方法进行设置。

取值范围：布尔型

默认值：on

23.16.1.4 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 GUC 参数设置方式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

23.16.1.5 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 gs_dump 的–quote-all-identifiers 选项。

该参数属于 USERSET 类型参数，请参考表 GUC 参数设置方式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

23.16.1.6 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。这是为了兼容旧版本而设置的。

该参数属于 USERSET 类型参数，请参考表 GUC 参数设置方式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

23.16.1.7 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 GUC 参数设置方式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

23.16.1.8 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 GUC 参数设置方式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

23.16.1.9 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 GUC 参数设置方式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

23.16.2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。

23.16.2.1 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 15-1 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： 该参数调整会修改内部数据类型转换规则，导致不可预期的行为，请谨慎操作。

23.16.2.2 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 15-1 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

23.16.2.3 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：8192

23.16.2.4 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 15-1 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。

23.16.2.5 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 15-1 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

23.16.2.6 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，用户无法修改，只能查看。

取值范围：枚举型

- A 表示同 O 数据库兼容。
- B 表示同 MY 数据库兼容。
- C 表示同 TD 数据库兼容。

- PG 表示同 POSTGRES 数据库兼容。

默认值：A

须知：

- 该参数只能在执行 CREATE DATABASE 命令创建数据库的时候设置。
- 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，请勿任意改动，否则会导致数据库行为不一致。

23.16.2.7 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 15-1 中对应设置方法进行设置。

取值范围：字符串

默认值：空

说明：

- 当前只支持下表。
- 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 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。

兼容性配置项	兼容性行为控制
	• 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， postgres=# select add_months('2018-02-28',3) from sys_dummy; 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， postgres=# select add_months('2018-02-28',3) from sys_dummy; 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。 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：

兼容性配置项	兼容性行为控制
	• 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 O 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。 postgres=# select length(lpad('123',0,'')) from sys_dummy; length ——- (1 row) • 设置此配置项时，空字符串显示为”。 postgres=# select length(lpad('123',0,'')) from sys_dummy; 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

兼容性配置项	兼容性行为控制
	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 MY 无 variadic 类型，所以该选项对 MY 无影响。
merge_update_multi	控制在使用 MERGE INTO … WHEN MATCHED THEN UPDATE（参考 MERGE INTO）和 INSERT … ON DUPLICATE KEY UPDATE（参考 INSERT）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。
hide_tailing_zero	numeric 显示配置项。不设置此项时，numeric 按照指定精度显示。设置此项时，隐藏小数点后的末尾 0。 set behavior compat options='hide tailing zero'; select cast(123.123 as numeric(15,10)); numeric ------- 123.123 (1 row)
rownum_type_compat	控制 ROWNUM 的类型，ROWNUM 默认类型为 INT8，设置此参数后，ROWNUM 类型变更为 NUMERIC 类型。
aformat_null_test	控制 rowtype 类型判空逻辑,设置此项时，对于 rowtype is not null 判断，当一行数据有一列不为空的时候返回 ture。 否则，对于 rowtype is not null 判断，当一行数据所有列不为空的时候返回 ture.
aformat_regexp_match	控制正则表达式函数的匹配行为。 设置此项，且 sql_compatibility 参数的值为 A 或 B 时，正则表达式的 flags 参数支持的选项含义有变更： . 默认不能匹配 '\n' 字符。 flags 中包含 n 选项时， . 能够匹配 '\n' 字符。 regexp_replace(source, pattern replacement) 函数替换所有匹配的子串。 regexp_replace(source, pattern, replacement, flags) 在 flags

兼容性配置项	兼容性行为控制
	值为”或者 null 时，返回值为 null。 否则，正则表达式的 flags 参数支持的选项含义： . 默认能匹配 '\n' 字符。 flags 中的 n 选项表示按照多行模式匹配。 regexp_replace(source, pattern replacement) 函数仅替换第一个匹配到的子串。 regexp_replace(source, pattern, replacement, flags) 在 flags 值为”或者 null 时，返回值为替换后的字符串。
compat_cursor	控制隐式游标状态兼容行为。设置此项，且兼容 O，隐式游标状态（SQL%FOUND、SQL%NOTFOUND、SQL%ISOPNE、SQL%ROWCOUNT）由原先的仅在当前执行的函数有效，拓展到包括本函数调用的子函数有效。
proc_outparam_override	控制存储过程出参的重载行为，打开该参数后，对于存储过程只有 out 出参部分不同的情况下，也可以正常调用。
proc_implicit_for_loop_variable	控制存储过程中 FOR_LOOP 查询语句行为设置此项时，在 FOR rec IN query LOOP 语句中，若 rec 已经定义，不会复用已经定义的 rec 变量，而且重新建立一个新的变量。否则，会复用已经定义的 rec 变量，不会建立新的变量。
allow_procedure_compile_check	控制存储过程中 select 语句和 open cursor 语句的编译检查设置此项时，在存储过程中执行 select 语句、open cursor for 语句、cursor%rowtype 语句、for rec in 语句时，若查询的表不存在，则无法创建创建存储过程，不支持 trigger 函数的编译检查，若查询的表存在，则成功创建存储过程。
char_coerce_compat	控制 char(n)类型向其它变长字符串类型转换时的行为。默认情况下 char(n)类型转换其它变长字符串类型时会省略尾部的空格，开启该参数后，转换时不再省略尾部的空格，并且在转换时如果 char(n)类型的长度超过其它变长字符串类型时将会报错。该参数仅在 sql_compatibility 参数的值为 A 时生效。

23.16.2.8 plpgsql.variable_conflict

参数说明：设置同名的存储过程变量和表的列的使用优先级。

该参数属于 USERSET 类型参数，仅支持表 GUC 参数设置方式中对应设置方法 3 进行设置。

取值范围：字符串

- error 表示遇到存储过程变量和表的列名同名则编译报错。
- use_variable 表示存储过程变量和表的列名同名则优先使用变量。
- use_column 表示存储过程变量和表的列名同名则优先使用列名。

默认值：error

23.16.2.9 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 15-1 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

23.16.2.10 uppercase_attribute_name

参数说明：设置列名以大写形式返回给客户端。该参数仅限于 ORA 兼容模式和集中式环境下使用。

该参数属于 USERSET 类型参数，请参考表 1 中对应设置方法进行设置。

取值范围：布尔型

- on 表示开启列名以大写形式返回给客户端。
- off 表示关闭列名以大写形式返回给客户端。

默认值：off

23.16.2.11 lastval_supported

参数说明：控制是否可以使用 lastval 函数。

该参数属于 POSTMASTER 类型参数，请参考表 15-1 中对应设置方法进行设置。

取值范围：布尔型

- on 表示支持 lastval 函数，同时 nextval 函数不支持下推。
- off 表示不支持 lastval 函数，同时 nextval 函数可以下推。

默认值：off

23.17 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。

23.17.1exit_on_error

参数说明：打开该开关，ERROR 级别报错会升级为 PANIC 报错，从而可以产生 core 堆栈。主要用于问题定位和业务测试。

该参数属于 USERSET 类型参数，请参考表 GUC 参数设置方式中对应设置方法进行设置。

取值范围：布尔型

- on 表示 ERROR 级别报错会升级为 PANIC 报错。
- off 表示不会对 ERROR 级别报错进行升级。

默认值：off

23.17.2 restart_after_crash

参数说明：设置为 on，后端进程崩溃时，GBase 8c 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 GUC 参数设置方式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 GBase 8c 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

23.17.3 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 GUC 参数设置方式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

23.17.4 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 GUC 参数设置方式中对应设置方法进行设

置。

取值范围：整型，0~20。

默认值：0

23.17.5cn_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 GUC 参数设置方式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

23.17.6max_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 GUC 参数设置方式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

23.17.7retry_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 GUC 参数设置方式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

23.17.8data_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 GUC 参数设置方式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

23.17.9 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。

该参数属于 POSTMASTER 类型参数，请参考表 15-1 中对应设置方法进行设置。

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

23.18 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。

23.18.1 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。

说明： 此参数在该版本不生效。

该参数属于 USERSET 类型参数，请参考表 GUC 参数设置方式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）

23.18.2 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。

说明： 此参数在该版本不生效。

该参数属于 USERSET 类型参数，请参考表 GUC 参数设置方式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

23.18.3 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 GUC 参数设置方式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

23.19 GBase 8c 事务

介绍 GBase 8c 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。

23.19.1 autocommit

参数说明：控制是否开启自动提交，为 USERSET 参数类型，表示可以被任何用户在任何时刻设置。可选值范围为（on，off）、（true,false）或者（1,0）。

取值范围：

on：表示开启自动提交，为默认值。

off：表示关闭自动提交。

默认值：on

23.19.2 enable_pgstat

参数说明：控制全局性能统计功能的开启和关闭。需要重启生效。

取值范围：on 表示开启性能统计功能，off 表示关闭性能统计功能。

默认值：off

23.19.3 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 15-1 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：GBase 8c 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 default_transaction_isolation 所设隔离级别。

默认值：read committed

23.19.4 transaction_read_only

参数说明：设置当前事务是只读事务。

该参数在数据库恢复过程中或者在备机里，固定为 on；否则，固定为 default_transaction_read_only 的值。

该参数属于 USERSET 类型参数，请参考表 15-1 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

23.19.5xc_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 15-1 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 GBase 8c 数据不一致。

默认值：off

23.19.6allow_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 15-1 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

23.19.7transaction_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。该参数为预留参数，该版本不生效。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 15-1 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

23.19.8enable_show_any_tuples

参数说明：该参数只有在只读事务中可用，用于分析。

该参数属于 USERSET 类型参数，请参考表 15-1 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

23.19.9replication_type

参数说明：标记当前 HA 模式是单主机模式、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 15-1 中对应设置方法进行设置。

该参数用户不能自己去设置参数值。

取值范围：0~2

- 2 表示单主机模式，此模式无法扩展备机。

- 1 表示使用一主多备模式，全场景覆盖，推荐使用。
- 0 表示主备从模式，目前此模式暂不支持。

默认值：1

23.19.10 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 15-1 中对应设置方法进行设置。

在备机请求主机进行日志复制时，如果 application_name 参数没有设置，那么该参数会被用来作为备机在主机上的流复制槽名字。该流复制槽的命名方式为“该参数值_备机 ip_备机 port”。其中，备机 ip 和备机 port 取自 replconninfo 参数中指定的备机 ip 和端口号。该流复制槽最大长度为 61 个字符，如果拼接后的字符串超过该长度，则会使用截断后的 pgxc_node_name 进行拼接，以保证流复制槽名字长度小于等于 61 个字符。

注意： 此参数修改后会导致连接数据库实例失败，不建议进行修改。

取值范围：字符串

默认值：当前节点名称

23.19.11 enable_defer_calculate_snapshot

参数说明：延迟计算快照的 xmin 和 oldestxmin，执行 1000 个事务或者间隔 1s 才触发计算，设置为 on 时可以在高负载场景下减少计算快照的开销，但是会导致 oldestxmin 推进较慢，影响垃圾元组回收，设置为 off 时 xmin 和 oldestxmin 可以实时推进，但是会增加计算快照时的开销。

该参数属于 SIGHUP 类型参数，改请参考表 GUC 参数设置方式进行设置

取值范围：布尔型。

- on 表示延迟计算快照 xmin 和 oldestxmin。
- off 表示实时计算快照 xmin 和 oldestxmin。

默认值：on

23.20 双数据库实例复制参数

23.20.1 RepOriginId

参数说明：该参数是一个会话级别的 GUC 参数，在双向逻辑复制的场景下，为避免数据循环复制，需要设置为一个非 0 的值。

该参数属于 USERSET 类型参数，请参考表 15-15-1 中方式三对应设置方法进行设置。

取值范围：整型，0~2147483647

默认值：0

23.21 开发人员选项

23.21.1 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 15-1 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

注意： 不建议修改该参数默认值，若设置为 on，可能导致系统表损坏，甚至数据库无法启动。

23.21.2 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 GBase 8c 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 15-1 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 GBase 8c 时，debug_assertions 缺省值为 on 。

默认值：off

23.21.3ignore_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 15-1 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

23.21.4ignore_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 15-1 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

23.21.5post_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 15-1 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

说明： 此参数只用于调试和问题定位，为避免影响正常业务运行，生产环境下请确保参数值为默认值 0。参数设置为非 0 时可能会因认证延迟时间过长导致数据库实例状态异常。

23.21.6pre_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 15-1 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

说明： 此参数只用于调试和问题定位，为避免影响正常业务运行，生产环境下请确保参数值为默认值 0。参数设置为非 0 时可能会因认证延迟时间过长导致数据库实例状态异常。

23.21.7trace_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 15-1 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

23.21.8trace_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 15-1 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。

23.21.9trace_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 GBase 8c 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 15-1 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

23.21.10 zero_damaged_pages

参数说明：控制检测导致 GBase 8c 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 15-1 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

23.21.11 remotetype

参数说明：设置远程连接类型。

该参数不支持修改。

取值范围：枚举类型，有效值有 application、datanode、internaltool。

默认值：application

23.21.12 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 15-1 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

23.21.13 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 15-1 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

23.21.14 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 15-1 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

23.21.15 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 15-1 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

23.21.16 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 15-1 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

说明： 当前版本不支持加速数据库实例，因此该参数设置后不生效。

23.21.17 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 15-1 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

23.21.18 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 15-1 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。

说明： 当前版本 x86 平台下不支持 numa_distribute_mode 设置为 all。

默认值：'none'

23.21.19 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 true。

该参数属于 SIGHUP 类型参数，请参考表 15-1 中对应设置方法进行设置。

取值范围：布尔型

默认值：on

23.21.20 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 15-1 中对应设置方法进行设置。

取值范围：整型，0~1000000（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

23.21.21 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，排序两类 SQL 语句的加速执行，提升 SQL 执行性能。

该参数属于 USERSET 类型参数，请参考表 15-1 中对应设置方法进行设置。

取值范围：布尔型

- on 表示开启。
- off 表示不开启。

默认值：off

23.21.22 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 15-1 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

23.21.23 pldebugger_timeout

参数说明：该参数用来控制 pldebugger server 端等待 debug 端响应的超时时间。

该参数属于 USERSET 类型参数，请参考表 15-1 中对应设置方法进行设置。

取值范围：整型，1 ~ 86400，单位为秒。

默认值：15min

23.21.24 plsql_show_all_error

参数说明：该参数用来控制编译 PLPGSQL 对象时是否支持跳过报错继续编译，具体影响请参考 DBE_PLDEVELOPER 内的说明。

该参数属于 USERSET 类型参数，请参考表 15-1 中对应设置方法进行设置。

取值范围：布尔型

默认值：off

23.21.25 ustore_attr

参数说明：该参数主要用来控制 USTORE 存储引擎表的信息统计，回滚类型，重点模块(包括数据、索引、回滚段、回放等)运行时数据的校验，主要用于协助研发问题定位。

该参数属于 USERSET 类型参数，请参考表 15-1 中对应设置方法进行设置。

取值范围：字符串，该参数值的设置方式采用 key-value 模式，key 和 value 取值对应关系和说明如下。如果是多个 key-value 组合设置，中间使用";“作为分隔符，例如：

```
ustore_attr=’ustore_verify_level=NORMAL;ustore_verify_module=UPAGE:UBTREE:UNDO:REDO’。
```

说明： ustore_attr 设置参数值时，key 和 value 之间的”=“前后不要有空格或者其他字符，例如 ustore_attr='ustore_verify_level = NORMAL;，内核代码校验会发现参数不合法，导致参数设置失败。

ustore_verify_level：控制校验的级别。

取值范围：字符串，见下述表格详细描述。

表 1 ustore_verify_level 取值含义说明

参数取值	含义
FAST	FAST 表示快速校验，校验内容少，性能影响最小。
NORMAL	NORMAL 表示正常校验，校验内容相比快速校验增多，性能影响中等。
SLOW	SLOW 表示慢速校验，校验内容最多，性能影响比较大。

默认值：空字符串

ustore_verify_module：控制校验的模块。

取值范围：字符串，设置值 UPAGE，UBTREE，UNDO, REDO 中的一个或者多个，或

者单独设置 ALL 或者 NULL(不区分大小写)。当设置 UPAGE，UBTREE，UNDO，REDO 中的多个值时，使用":" 作为连接符。例如 ustore_verify_module=UPAGE:UBTREE:UNDO:REDO。

表 2 ustore_verify_module 取值含义说明

参数取值	含义
UPAGE	表示开启数据页面校验
UBTREE	表示开启 UBTREE 索引校验
UNDO	表示开启回滚段数据校验
REDO	表示开启 REDO 流程的数据页面校验
ALL	表示同时开启 UPAGE，UBTREE，UNDO，REDO 模块数据的校验
NULL	表示同时关闭 UPAGE，UBTREE，UNDO，REDO 模块数据的校验

默认值：空字符串

index_trace_level：控制开启索引追踪并控制打印级别，开启后在索引扫描的过程中，会根据不同的打印级别对符合条件的索引元组的信息进行打印。

取值范围：字符串，取值下表格描述。

默认值：NO

表 3 index_trace_level 取值含义说明

参数取值	含义

NO	不打印任何额外信息。
NORMAL	打印可见索引元组相关信息，包括： 当前索引元组所在索引页面号以及偏移。 当前元组状态。 当前元组对应的 TID 以及 partOid。 当前元组对应的 xmin 和 xmax 信息。 当前元组内容（如果开启 enable_log_tuple）。
VISIBILITY	在 NORMAL 的基础上，额外打印没有通过可见性检查的索引元组的信息，并标明是否可见。
SHOWHIKEY	在 VISIBILITY 的基础上，尝试打印页面上 HIKEY 元组的信息。
ALL	打印扫描的索引页面上所有元组的相关信息。

enable_log_tuple：打印日志级提示信息时，是否允许同时将相关元组的内容打印出来，以便进行问题排查和定位。

取值范围：on 或者 off（不区分大小写）

默认值：off

备注：该参数已弃用

enable_ustore_sync_rollback：控制 USTORE 表是否开启同步回滚。

取值范围：布尔值

默认值：true

enable_ustore_async_rollback：控制 USTORE 表是否开启异步回滚。

取值范围：布尔值

默认值：true

enable_ustore_page_rollback：控制 USTORE 表是否开启页面回滚。

取值范围：布尔值

默认值：true

enable_ustore_partial_seqscan：是否允许 USTORE 表开启部分扫描。

取值范围：布尔值

默认值：false

enable_candidate_buf_usage_count：是否开启缓存区使用计数统计。

取值范围：布尔值

默认值：false

ustats_tracker_naptime：控制 USTORE 表统计信息周期。

取值范围：1~INT_MAX/1000

默认值：20，单位(秒)

umax_search_length_for_prune：控制 USTORE 表 prune 操作搜索的最大深度。

取值范围：1~INT_MAX/1000

默认值：10，单次(次)

默认值：空字符串

说明：该参数只适用于集中式或者单机上的 USTORE 存储引擎表。

23.22 审计

23.22.1审计开关

23.22.1.1 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 15-1 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

23.22.1.2 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 15-1 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 GBase 8c，则审计日志路径为“$GAUSSLOG/pg_audit/实例名称”。

23.22.1.3 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 15-1 中对应设置方法进行设置。

取值范围：字符串

默认值：binary

23.22.1.4 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 15-1 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。

23.22.1.5 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 15-1 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。

23.22.1.6 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 15-1 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

23.22.1.7 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 15-1 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

23.22.1.8 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 15-1 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

23.22.1.9 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 15-1 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。

23.22.1.10 audit_thread_num

参数说明：审计线程的个数。

该参数属于 POSTMASTER 类型参数，请参考表 15-1 中对应设置方法进行设置。

取值范围：整型，1~48

默认值：1

须知： 当 audit_dml_state 开关打开且在高性能场景下，建议增大此参数保证审计消息可以被及时处理和记录。

23.22.2用户和权限审计

23.22.2.1 audit_login_logout

参数说明：这个参数决定是否审计 GBase 8c 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 15-1 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

23.22.2.2 audit_database_process

参数说明：该参数决定是否对 GBase 8c 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 15-1 中对应设置方法进行设置。

取值范围：整型，0、1。

- 0 表示关闭 GBase 8c 启动、停止、恢复和切换审计功能。
- 1 表示开启 GBase 8c 启动、停止、恢复和切换审计功能。

默认值：1

说明：

GBase 8c 启动时 DN 执行备升主流程，因此 DN 启动时审计日志中类型为 system_switch。

23.22.2.3 audit_user_locked

参数说明：该参数决定是否审计 GBase 8c 用户的锁定和解锁。

该参数属于 SIGHUP 类型参数，请参考表 15-1 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

23.22.2.4 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 15-1 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

23.22.2.5 audit_grant_revoke

参数说明：该参数决定是否审计 GBase 8c 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 15-1 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

23.22.3操作审计

23.22.3.1 audit_system_object

参数说明：该参数决定是否对 GBase 8c 数据库对象的 CREATE、DROP、ALTER 操作进行审计。GBase 8c 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 15-1 中对应设置方法进行设置。

取值范围：整型，0~67108863

- 0 代表关闭 GBase 8c 数据库对象的 CREATE、DROP、ALTER 操作审计功能。
- 非 0 代表只审计 GBase 8c 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 26 个二进制位的组合求出，这 26 个二进制位分别代表 GBase 8c 的 26 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 26 个二进制位代表的具体审计内容请参见下表。

默认值：12295

表 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 C	• 0 表示不审计该对象的 CREATE、

二进制位	含义	取值说明
	REATE、DROP、ALTER 操作。	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/MATVIEW 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； • 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	• 0 表示不审计该对象的 CREATE、

二进制位	含义	取值说明
	对象的 CREATE、DROP、ALTER 操作。	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、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 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。

二进制位	含义	取值说明
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。
第 19 位	是否审计 SEQUENCE 对象的 CREATE、DROP、ALTER 操作。	• 0 表示不审计 SEQUENCE 对象的 CREATE、DROP、ALTER 操作； • 1 表示审计 SEQUENCE 对象的 CREATE、DROP、ALTER 操作。
第 20 位	是否审计 CMK、CEK 对象的 CREATE、DROP 操作。	• 0 表示不审计 CMK、CEK 对象的 CREATE、DROP 操作； • 1 表示审计 CMK、CEK 对象的 CREATE、DROP 操作。
第 21 位	是否审计 PACKAGE 对象的 CREATE、DROP、ALTER 操作。	• 0 表示不审计 PACKAGE 对象的 CREATE、DROP、ALTER 操作； • 1 表示审计 PACKAGE 对象的 CREATE、DROP、ALTER 操作。
第 22 位	是否审计 MODEL 对象的 CREATE、DROP 操作。	• 0 表示不审计 MODEL 对象的 CREATE、ALTER 操作； • 1 表示审计 MODEL 对象的 CREATE、DROP 操作。
第 23 位	是否审计 PUBLICATION 和 SUBSCRIPTION 对象的 CREATE、DROP、ALTER 操作。	• 0 表示不审计 PUBLICATION 和 SUBSCRIPTION 对象的 CREATE、DROP、ALTER 操作； • • 1 表示审计 PUBLICATION 和 SUBSCRIPTION 对象的 CREATE、DROP、ALTER 操作。

二进制位	含义	取值说明
		•
第 24 位	是否审计对 gs_global_config 全局对象的 ALTER、DROP 操作。	• 0 表示不审计对系统表 gs_global_config 全局对象的 ALTER、DROP 操作； • 1 表示审计对系统表 gs_global_config 全局对象的 ALTER、DROP 操作。
第 25 位	是否审计 FOREIGN DATA WRAPPER 对象的 CREATE、DROP、ALTER 操作。	• 0 表示不审计 FOREIGN DATA WRAPPER 对象的 CREATE、DROP、ALTER 操作； • 1 表示审计 FOREIGN DATA WRAPPER 对象的 CREATE、DROP、ALTER 操作。

23.22.3.2 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 15-1 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

23.22.3.3 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 15-1 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

23.22.3.4 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 15-1 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

23.22.3.5 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 15-1 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：1

23.22.3.6 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 15-1 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

23.22.3.7 audit_xid_info

参数说明：这个参数决定是否在审计日志字段 detail_info 中记录 SQL 语句的事务 ID。

该参数属于 SIGHUP 类型参数，请参考表 15-1 中对应设置方法进行设置。

取值范围：整型，0、1。

- 0 表示关闭审计日志记录事务 ID 功能。
- 1 表示开启审计日志记录事务 ID 功能。

默认值：0

须知： 如果开启此开关，审计日志中 detail_info 信息则以 xid 开始，例如：

```
detail_info: xid=14619 , create table t1(id int);
```

对于不存在事务 ID 的审计行为，记录 xid=NA。

23.22.3.8 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 15-1 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

23.22.3.9 enable_nonsysadmin_execute_direct

参数说明：是否允许非系统管理员和非监控管理员执行 EXECUTE DIRECT ON 语句。

该参数属于 POSTMASTER 类型参数，请参考表 15-1 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和监控管理员执行 EXECUTE DIRECT ON 语句。

默认值：off

23.22.3.10 enable_access_server_directory

参数说明：是否开启非初始用户创建、修改和删除 DIRECTORY 的权限。

该参数属于 SIGHUP 类型参数，请参考表 15-1 中对应设置方法进行设置。

取值范围：布尔型

- on 表示开启非初始用户创建、修改和删除 DIRECTORY 的权限。
- off 表示不开启非初始用户创建、修改和删除 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、修改和删除 DIRECTORY 对象。 - 如果开启了 enable_access_server_directory，具有 SYSADMIN 权限的用户和继承了内置角色 gs_role_directory_create 权限的用户可以创建 directory 对象；具有 SYSADMIN 权限的用户、directory 对象的属主、被授予了该 directory 的 DROP 权限的用户或者继承了内置角色 gs_role_directory_drop 权限的用户可以删除 directory 对象；具有 SYSADMIN 权限的用户和 directory 对象的属主可以修改 directory 对象的所有者，且要求该用户是新属主的成员。

23.23 CM 相关参数

CM 相关参数的修改对 Kernel 的运行机制有影响，谨慎修改。修改 CM 相关参数的方法，请参考表 GUC 参数设置方式中方式一进行设置。

23.23.1cm_agent 参数

23.23.1.1 log_dir

参数说明：log_dir 决定存放 cm_agent 日志文件的目录。可以是绝对路径，或者是相对路径（相对于$GAUSSLOG 的路径）。

取值范围：字符串。修改后需要重启 cm_agent 才能生效。参数修改请参考表 GUC 参数设置方式进行设置。

默认值：“log”，表示在$GAUSSLOG 下对应的 cm 目录下生成 cm_agent 日志。

23.23.1.2 log_file_size

参数说明：控制日志文件的大小。当日志文件达到指定大小时，则重新创建一个日志文件记录日志信息。

取值范围：整型，取值范围 0~2047，单位为 MB。修改后需要重启 cm_agent 才能生效。参数修改请参考表 GUC 参数设置方式进行设置。

默认值：16MB

23.23.1.3 log_min_messages

参数说明：控制写到 cm_agent 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。

取值范围：枚举类型，有效值有 debug5、debug1、warning、error、log、fatal。修改后需要重启 cm_agent 才能生效。参数修改请参考表 GUC 参数设置方式进行设置。

默认值：warning

23.23.1.4 incremental_build

参数说明：控制重建备 DN 模式是否为增量。打开这个开关，则增量重建备 DN；否则，全量重建备 DN。

取值范围：布尔型，有效值有 on、off。修改后需要重启 cm_agent 才能生效。参数修改请参考表 GUC 参数设置方式进行设置。

默认值：on

23.23.1.5 alarm_component

参数说明：设置用于处理告警内容的告警组件的位置。

取值范围：字符串。修改后需要重启 cm_agent 才能生效。参数修改请参考表 GUC 参数设置方式进行设置。

- 若前置脚本 gs_preinstall 中的–alarm-type 参数设置为 5 时，表示未对接第三方组件，

告警写入 system_alarm 日志，此时 GUC 参数 alarm_component 的取值为：/opt/huawei/snas/bin/snas_cm_cmd。

- 若前置脚本 gs_preinstall 中的–alarm-type 参数设置为 1 时，表示对接第三方组件，此时 GUC 参数 alarm_component 的值为第三方组件的可执行程序的绝对路径。

默认值：/opt/huawei/snas/bin/snas_cm_cmd

23.23.1.6 alarm_report_interval

参数说明：指定告警上报的时间间隔。参数修改请参考表 GUC 参数设置方式进行设置。

取值范围：非负整型，单位为秒。

默认值：1

23.23.1.7 alarm_report_max_count

参数说明：指定告警上报的最大次数。参数修改请参考表 GUC 参数设置方式进行设置。

取值范围：非负整型。

默认值：1

23.23.1.8 agent_report_interval

参数说明：cm_agent 上报实例状态的时间间隔。

取值范围：整型，单位为秒。修改后需要重启 cm_agent 才能生效。参数修改请参考表 GUC 参数设置方式进行设置。

默认值：1

23.23.1.9 agent_phony_dead_check_interval

参数说明：cm_agent 检测 DN 进程是否僵死的时间间隔。

取值范围：整型，单位为秒。修改后需要重启 cm_agent 才能生效。参数修改请参考表 GUC 参数设置方式进行设置。

默认值：10

23.23.1.10 agent_check_interval

参数说明：cm_agent 查询 DN 等实例状态的时间间隔。

取值范围：整型，单位为秒。修改后需要重启 cm_agent 才能生效。参数修改请参考表 GUC 参数设置方式进行设置。

默认值：2

23.23.1.11 agent_heartbeat_timeout

参数说明：cm_server 心跳超时时间。

取值范围：整型，2~231 - 1，单位为秒。修改后需要重启 cm_agent 才能生效。参数修改请参考表 GUC 参数设置方式进行设置。

默认值：8

23.23.1.12 agent_connect_timeout

参数说明：cm_agent 连接 cm_server 超时时间。

取值范围：整型，单位为秒。修改后需要重启 cm_agent 才能生效。参数修改请参考表 GUC 参数设置方式进行设置。

默认值：1

23.23.1.13 agent_connect_retries

参数说明：cm_agent 连接 cm_server 尝试次数。

取值范围：整型。修改后需要重启 cm_agent 才能生效。参数修改请参考表 GUC 参数设置方式进行设置。

默认值：15

23.23.1.14 agent_kill_instance_timeout

参数说明：当 cm_agent 在无法连接 cm_server 主节点后，发起一次杀死本节点上所有实例的操作之前，所需等待的时间间隔。

取值范围：整型。修改后需要重启 cm_agent 才能生效。参数修改请参考表 GUC 参数

设置方式进行设置。

默认值：0，不发起杀死本节点上所有实例的操作。

23.23.1.15 security_mode

参数说明：控制是否以安全模式启动 DN。打开这个开关，则以安全模式启动 DN；否则，以非安全模式启动 DN。

取值范围：布尔型，有效值有 on、off。修改后需要重启 cm_agent 才能生效。参数修改请参考表 GUC 参数设置方式进行设置。

默认值：off

23.23.1.16 upgrade_from

参数说明：就地升级过程中使用，用于标示升级前数据库的内部版本号，此参数禁止手动修改。

取值范围：非负整型。修改后需要重启 cm_agent 才能生效。参数修改请参考表 GUC 参数设置方式进行设置。

默认值：0

23.23.1.17 process_cpu_affinity

参数说明：控制是否以绑核优化模式启动主 DN 进程。配置该参数为 0，则不进行绑核优化；否则，进行绑核优化，且物理 CPU 片数为 2n 个。数据库、cm_agent 重启生效。仅支持 ARM。参数修改请参考表 GUC 参数设置方式进行设置。

取值范围：整型，0~2。

默认值：0

23.23.1.18 log_threshold_check_interval

参数说明：日志压缩和清除的时间间隔，每 1800 秒压缩和清理一次。

取值范围：整型，0~2147483647，单位为秒。修改后需要重启 cm_agent 才能生效。参数修改请参考表 GUC 参数设置方式进行设置。

默认值：1800

23.23.1.19 dilatation_shard_count_for_disk_capacity_alarm

参数说明：扩容场景下，设置新增的扩容分片数，用于上报磁盘容量告警时的阈值计算。

说明： 该分片数请与实际扩容分片数设置为一致。

取值范围：整型，0~232 - 1，单位为个。该参数设置为 0，表示关闭磁盘扩容告警上报；该参数设置为大于 0，表示开启磁盘扩容告警上报，且告警上报的阈值根据此参数设置的分片数量进行计算。修改后需要重启 cm_agent 才能生效。参数修改请参考表 GUC 参数设置方式进行设置。

默认值：1

23.23.1.20 log_max_size

参数说明：控制日志最大存储值。

取值范围：整型，0~2147483647，单位为 MB。修改后需要重启 cm_agent 才能生效。参数修改请参考表 GUC 参数设置方式进行设置。

默认值：10240

23.23.1.21 log_max_count

参数说明：硬盘上可存储的最多日志数量。

取值范围：整型，0~10000，单位为个。修改后需要重启 cm_agent 才能生效。参数修改请参考表 GUC 参数设置方式进行设置。

默认值：10000

23.23.1.22 log_saved_days

参数说明：日志保存的天数。

取值范围：整型，0~1000，单位为天。修改后需要重启 cm_agent 才能生效。参数修改请参考表 GUC 参数设置方式进行设置。

默认值：90

23.23.1.23 enable_log_compress

参数说明：控制压缩日志功能。

取值范围：布尔型。修改后需要重启 cm_agent 才能生效。参数修改请参考表 GUC 参数设置方式进行设置。

- on 表示允许压缩日志。
- off 表示不允许压缩日志。

默认值：on

23.23.1.24 log_pattern_%s

参数说明：enable_log_compress=on 时使用，定义压缩日志的格式，压缩含有关键字%s 的所有日志，%s 为服务端工具名称。

取值范围：表 log_pattern_%s 中所有参数对应的值。参数修改请参考表 GUC 参数设置方式进行设置。

默认值：表 log_pattern_%s 中各参数对应的默认值。不支持用户修改默认值。

表 log_pattern_%s

参数（logpattern%s）	默认值
log_pattern_cm_ctl	cm_ctl-
log_pattern_gs_clean	gs_clean-
log_pattern_gs_ctl	gs_ctl-
log_pattern_gs_guc	gs_guc-
log_pattern_gs_dump	gs_dump-
log_pattern_gs_dumpall	gs_dumpall-
log_pattern_gs_restore	gs_restore-
log_pattern_gs_initcm	gs_initcm-

参数（logpattern%s）	默认值
log_pattern_gs_initdb	gs_initdb-
log_pattern_gs_initgtm	gs_initgtm-
log_pattern_gtm_ctl	gtm_ctl-
log_pattern_cm_agent	cm_agent-
log_pattern_system_call	system_call-
log_pattern_cm_server	cm_server-
log_pattern_om_monitor	om_monitor-
log_pattern_gs_local	gs_local-
log_pattern_gs_preinstall	gs_preinstall-
log_pattern_gs_install	gs_install-
log_pattern_gs_replace	gs_replace-
log_pattern_gs_uninstall	gs_uninstall-
log_pattern_gs_om	gs_om-
log_pattern_gs_upgradectl	gs_upgradectl-
log_pattern_gs_expand	gs_expand-
log_pattern_gs_shrink	gs_shrink-
log_pattern_gs_postuninstall	gs_postuninstall-
log_pattern_gs_backup	gs_backup-
log_pattern_gs_checkos	gs_checkos-
log_pattern_gs_collector	gs_collector-
log_pattern_GaussReplace	GaussReplace-
log_pattern_GaussOM	GaussOM-

参数（logpattern%s）	默认值
log_pattern_gs_checkperf	gs_checkperf-
log_pattern_gs_check	gs_check-
log_pattern_roach-agent	roach-agent-
log_pattern_roach-controller	roach-controller-
log_pattern_postgresql	postgresql-
log_pattern_gtm	gtm-
log_pattern_sessionstat	sessionstat-
log_pattern_sync	sync-
log_pattern_system_alarm	system_alarm-
log_pattern_pg_perf	pg_perf-
log_pattern_slow_query_log	slow_query_log-
log_pattern_asp	asp-
log_pattern_etcd	etcd-
log_pattern_gs_cgroup	gs_cgroup-
log_pattern_pscp	pscp-
log_pattern_postgresql-query-log	postgresql-query-log-
log_pattern_gs_hotpatch	gs_hotpatch-
log_pattern_pssh	pssh-

23.23.1.25 agent_backup_open

参数说明：灾备数据库实例设置，开启后 CM 按照灾备数据库实例模式运行。

取值范围：整型，0~1。修改后需要重启 cm_agent 才能生效。参数修改请参考表 log_pattern_%s 进行设置。

- 0 表示关闭。
- 1 表示开启。

默认值：0

23.23.1.26 enable_xc_maintenance_mode

参数说明：在数据库实例为只读模式下，控制是否可以修改 pgxc_node 系统表。

取值范围：布尔型。参数修改请参考表 15-1 进行设置。

- on 表示开启可以修改 pgxc_node 系统表功能。
- off 表示关闭可以修改 pgxc_node 系统表功能。

默认值：on

23.23.1.27 unix_socket_directory

参数说明：unix 套接字的目录位置。

取值范围：字符串。修改后需要重启 cm_agent 才能生效。参数修改请参考表 log_pattern_%s 进行设置。

默认值：”

23.23.1.28 enable_dcf

参数说明：DCF 模式开关。

取值范围：布尔型。修改后需要重启 cm_agent 才能生效。参数修改请参考表 log_pattern_%s 进行设置。

- 0 表示关闭。
- 1 表示开启。

默认值：off

23.23.1.29 disaster_recovery_type

参数说明：主备数据库实例灾备关系的类型。

取值范围：整型，0~2。修改后需要重启 cm_agent 才能生效。参数修改请参考表 log_pattern_%s 进行设置。

- 0 表示未搭建灾备关系。
- 1 表示搭建了 obs 灾备关系。
- 2 表示搭建了流式灾备关系

默认值：0

23.23.2cm_server 参数

23.23.2.1 log_dir

参数说明：log_dir 决定存放 cm_server 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$GAUSSLOG 的路径）。

取值范围：字符串。修改后需要重启 cm_server 才能生效。参数修改请参考表 GUC 参数设置方式进行设置。

默认值：“log”，表示在$GAUSSLOG 下对应的 cm 目录下生成 cm_server 日志。

23.23.2.2 log_file_size

参数说明：控制日志文件的大小。当日志文件达到指定大小时，则重新创建一个日志文件记录日志信息。

取值范围：整型，取值范围 0~2047，单位为 MB。修改后需要重启 cm_server 才能生效。参数修改请参考表 GUC 参数设置方式进行设置。

默认值：16MB

23.23.2.3 log_min_messages

参数说明：控制写到 cm_server 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。

取值范围：枚举类型，有效值有 debug5、debug1、log、warning、error、fatal。修改后需要重启 cm_server 才能生效。参数修改请参考表 2 进行设置。

默认值：warning

23.23.2.4 thread_count

参数说明：cm_server 线程池的线程数。

取值范围：整型，2~1000。修改后需要重启 cm_server 才能生效。参数修改请参考表 GUC 参数设置方式进行设置。

默认值：1000

23.23.2.5 alarm_component

参数说明：设置用于处理告警内容的告警组件的位置。

取值范围：字符串。修改后需要重启 cm_server 才能生效。参数修改请参考表 2 进行设置。

- 若前置脚本 gs_preinstall 中的–alarm-type 参数设置为 5 时，表示未对接第三方组件，告警写入 system_alarm 日志，此时 GUC 参数 alarm_component 的取值为：/opt/huawei/snas/bin/snas_cm_cmd。
- 若前置脚本 gs_preinstall 中的–alarm-type 参数设置为 1 时，表示对接第三方组件，此时 GUC 参数 alarm_component 的值为第三方组件的可执行程序的绝对路径。

默认值：/opt/huawei/snas/bin/snas_cm_cmd

23.23.2.6 instance_failover_delay_timeout

参数说明：cm_server 检测到主机宕机，failover 备机的延迟时间。

取值范围：整型，单位为秒。修改后需要重启 cm_server 才能生效。参数修改请参考表 GUC 参数设置方式进行设置。

默认值：0

23.23.2.7 instance_heartbeat_timeout

参数说明：实例心跳超时时间。

取值范围：整型，单位为秒。修改后需要重启 cm_server 才能生效。参数修改请参考表

GUC 参数设置方式进行设置。

默认值：6

23.23.2.8 cmserver_ha_connect_timeout

参数说明：cm_server 主备连接超时时间。

取值范围：整型，单位为秒。修改后需要重启 cm_server 才能生效。参数修改请参考表 GUC 参数设置方式进行设置。

默认值：2

23.23.2.9 cmserver_ha_heartbeat_timeout

参数说明：cm_server 主备心跳超时时间。

取值范围：整型，单位为秒。修改后需要重启 cm_server 才能生效。参数修改请参考表 GUC 参数设置方式进行设置。

默认值：6

23.23.2.10 phony_dead_effective_time

参数说明：用于 DN 进程的僵死检测，当检测到的僵死次数大于该参数值，认为进程僵死，将进程重启。

取值范围：整型，单位为次数。修改后需要重启 cm_server 才能生效。参数修改请参考表 GUC 参数设置方式进行设置。

默认值：5

23.23.2.11 enable_transaction_read_only

参数说明：控制数据库是否为只读模式开关。

取值范围：布尔型，有效值有 on，off，true，false，yes，no，1，0。参数修改请参考表 GUC 参数设置方式进行设置。

默认值：on

23.23.2.12 datastorage_threshold_check_interval

参数说明：检测磁盘占用的时间间隔。间隔用户指定时间，检测一次磁盘占用。

取值范围：整型，单位为秒。参数修改请参考表 2 进行设置。

默认值：10

23.23.2.13 datastorage_threshold_value_check

参数说明：设置数据库只读模式的磁盘占用阈值，当数据目录所在磁盘占用超过这个阈值，自动将数据库设置为只读模式。

取值范围：整型，1～99，表示百分比。参数修改请参考表 GUC 参数设置方式进行设置。

默认值：85

23.23.2.14 max_datastorage_threshold_check

参数说明：设置磁盘使用率的最大检测间隔时间。当用户手动修改只读模式参数后，会自动在指定间隔时间后开启磁盘满检测操作。

取值范围：整型，单位为秒。参数修改请参考表 GUC 参数设置方式进行设置。

默认值：43200

23.23.2.15 cmserver_ha_status_interval

参数说明：cm_server 主备同步状态信息间隔时间。

取值范围：整型，单位为秒。修改后需要重启 cm_server 才能生效。参数修改请参考表 GUC 参数设置方式进行设置。

默认值：1

23.23.2.16 cmserver_self_vote_timeout

参数说明：cm_server 自仲裁超时时间。

取值范围：整型，单位为秒。修改后需要重启 cm_server 才能生效。参数修改请参考表 GUC 参数设置方式行设置。

默认值：6

23.23.2.17 alarm_report_interval

参数说明：指定告警上报的时间间隔。

取值范围：非负整型，单位为秒。参数修改请参考表 GUC 参数设置方式进行设置。

默认值：3

23.23.2.18 alarm_report_max_count

参数说明：指定告警上报的最大次数。

取值范围：非负整型。参数修改请参考表 2 进行设置。

默认值：1

23.23.2.19 enable_az_auto_switchover

参数说明：AZ 自动切换开关，若打开，则表示允许 cm_server 自动切换 AZ。否则当发生 dn 故障等情况时，即使当前 AZ 已经不再可用，也不会自动切换到其它 AZ 上，除非手动执行切换命令。

取值范围：非负整型，0 或 1，0 表示开关关闭，1 表示开关打开。参数修改请参考表 GUC 参数设置方式进行设置。

默认值：1

23.23.2.20 instance_keep_heartbeat_timeout

参数说明：cm_agent 会定期检测实例状态并上报给 cm_server，若实例状态长时间无法成功检测，累积次数超出该数值，则 cm_server 将下发命令给 agent 重启该实例。

取值范围：整型，单位为秒。参数修改请参考表 GUC 参数设置方式进行设置。

默认值：40

23.23.2.21 az_switchover_threshold

参数说明：若一个 AZ 内 DN 分片的故障率（故障的 dn 分片数 / 总 dn 分片数 * 100%）超过该数值，则会触发 AZ 自动切换。

取值范围：整型，0~100。参数修改请参考表 GUC 参数设置方式进行设置。

默认值：100

23.23.2.22 az_check_and_arbitrate_interval

参数说明：当某个 AZ 状态不正常时，会触发 AZ 自动切换，该参数是检测 AZ 状态的时间间隔。

取值范围：整型，单位为秒。参数修改请参考表 GUC 参数设置方式进行设置。

默认值：2

23.23.2.23 az_connect_check_interval

参数说明：定时检测 AZ 间的网络连接，该参数表示连续两次检测之间的间隔时间。

取值范围：整型，单位为秒。参数修改请参考表 GUC 参数设置方式进行设置。

默认值：60

23.23.2.24 az_connect_check_delay_time

参数说明：每次检测 AZ 间的网络连接时有多次重试，该参数表示两次重试之间的延迟时间。

取值范围：整型，单位为秒。参数修改请参考表 2 进行设置。

默认值：150

23.23.2.25 cmserver_demote_delay_on_etcd_fault

参数说明：因为 etcd 不健康而导致 cm_server 从主降为备的时间间隔。

取值范围：整型，单位为秒。参数修改请参考表 GUC 参数设置方式进行设置。

默认值：8

23.23.2.26 instance_phony_dead_restart_interval

参数说明：当 dn 实例僵死时，会被 cm_agent 重启，相同的实例连续因僵死被杀时，其间隔时间不能小于该参数数值，否则 cm_agent 不会下发命令。

取值范围：整型，单位为秒。最小生效值为 1800，如果设置小于此值实际生效值为 1800。

参数修改请参考表 GUC 参数设置方式进行设置。

默认值：21600

23.23.2.27 cm_auth_method

参数说明：CM 模块端口认证方式，trust 表示未配置端口认证，gss 表示采用 kerberos 端口认证。必须注意的是：只有当 kerberos 服务端和客户端成功安装后才能修改为 gss，否则 CM 模块无法正常通信，将影响数据库状态。

取值范围：枚举类型，有效值有 trust, gss。修改后需要重启 cm_server 才能生效。参数修改请参考表 GUC 参数设置方式进行设置。

默认值：trust

23.23.2.28 cm_krb_server_keyfile

参数说明：kerberos 服务端 key 文件所在位置，需要配置为绝对路径。该文件通常为 ${GAUSSHOME}/kerberos 路径下，以 keytab 格式结尾，文件名与数据库运行所在用户名相同。与上述 cm_auth_method 参数是配对的，当 cm_auth_method 参数修改为 gss 时，该参数也必须配置为正确路径，否则将影响数据库状态

取值范围：字符串类型，修改后需要重启 cm_server 才能生效。参数修改请参考表 GUC 参数设置方式进行设置。

默认值：${GAUSSHOME}/kerberos/{UserName}.keytab，默认值无法生效，仅作为提示

23.23.2.29 cm_server_arbitrate_delay_base_time_out

参数说明：cm_server 仲裁延迟基础时长。cm_server 主断连后，仲裁启动计时开始，经过仲裁延迟时长后，将选出新的 cm_server 主。其中仲裁延迟时长由仲裁延迟基础时长、节点 index（server ID 序号）和增量时长共同决定。公式为：仲裁延迟时长=仲裁延迟基础时长+节点 index*仲裁延迟增量时长参数

取值范围：整型，index>0，单位为秒。修改后需要重启 cm_server 才能生效。参数修改请参考表 2 进行设置。

默认值：10

23.23.2.30 cm_server_arbitrate_delay_incrememtal_time_out

参数说明：cm_server 仲裁延迟增量时长。cm_server 主断连后，仲裁启动计时开始，经过仲裁延迟时长后，将选出新的 cm_server 主。其中仲裁延迟时长由仲裁延迟基础时长、节点 index（server ID 序号）和增量时长共同决定。公式为：仲裁延迟时长=仲裁延迟基础时长+节点 index*仲裁延迟增量时长参数

取值范围：整型，index>0，单位为秒。修改后需要重启 cm_server 才能生效。参数修改请参考表 GUC 参数设置方式进行设置。

默认值：3

23.23.2.31 force_promote

参数说明：cm_server 是否打开强起逻辑（指数据库状态为 Unknown 的时候以丢失部分数据为代价保证数据库基本功能可用）的开关。0 代表功能关闭，1 代表功能开启。该参数同时适用于 dn。

取值范围：整型，0~1。在 cm_server 上修改后需要重启 cm_server 才能生效。

默认值：0

23.23.2.32 switch_rto

参数说明：cm_server 强起逻辑等待时延。在 force_promote 被置为 1 时，当数据库的某一分片处于无主状态开始计时，等待该延迟时间后开始执行强起逻辑。

取值范围：整型，60~2147483647，单位为秒。修改后需要重启 cm_server 才能生效。

默认值：0

23.23.2.33 backup_open

参数说明：灾备数据库实例设置，开启后 CM 按照灾备数据库实例模式运行

取值范围：整型，0~1。修改后需要重启 cm_server 才能生效。非灾备数据库实例不能开启该参数。参数修改请参考表 GUC 参数设置方式进行设置。

- 0 表示关闭。

- 1 表示开启。

默认值：0

23.23.2.34 enable_dcf

参数说明：DCF 模式开关。

取值范围：布尔型。修改后需要重启 cm_server 才能生效。参数修改请参考表 GUC 参数设置方式进行设置。

- 0 表示关闭。
- 1 表示开启。

默认值：off

23.23.2.35 install_type

参数说明：容灾数据库实例相关的设置，用来区别是否是基于 dorado 的数据库实例。

取值范围：整型，0~2。修改后需要重启 cm_server 才能生效。非灾备数据库实例不能开启该参数。参数修改请参考表 GUC 参数设置方式进行设置。

- 0 表示未搭建容灾关系的数据库实例。
- 1 表示基于 dorado 的数据库实例。
- 2 表示基于流式的数据库实例。

默认值：0

23.23.2.36 enable_ssl

参数说明：ssl 证书开关。

取值范围：布尔型。打开后使用 ssl 证书加密通信。修改后需要重启才能生效。参数修改请参考表 GUC 参数设置方式进行设置。

- on 表示启用 ssl。
- off 表示不启用 ssl。

默认值：off

须知： 出于安全性考虑，建议不要关闭该配置。关闭后 cm 将不使用加密通信，所有信息明文传播，可能带来窃听、篡改、冒充等安全风险。

23.23.2.37 ssl_cert_expire_alert_threshold

参数说明：ssl 证书过期告警时间。

取值范围：整型，单位为天。证书过期时间少于该时间时，上报证书即将过期告警。参数修改请参考表 GUC 参数设置方式进行设置。

默认值：90

23.23.2.38 ssl_cert_expire_check_interval

参数说明：ssl 证书过期检测周期。

取值范围：整型，单位为秒。参数修改请参考表 GUC 参数设置方式进行设置。

默认值：86400

23.23.2.39 delay_arbitrate_timeout

参数说明：设置等待跟主 DN 同 AZ 节点 redo 回放，优先选择同 AZ 升主的时间。

取值范围：整型，[0, 21474836]，单位：秒。参数修改请参考表 GUC 参数设置方式进行设置。

默认值：0

23.23.2.40 ddb_type

参数说明：etcd，dcc 模式切换开关。

取值范围：整型。0：etcd；1：dcc。修改后需要重启 cm_server 才能生效。参数修改请参考表 2 进行设置。

默认值：0

23.23.2.41 ddb_log_level

参数说明：设置 ddb 日志级别。

关闭日志：“NONE”，NONE 表示关闭日志打印，不能与以下日志级别混合使用。

开启日志：“RUN_ERR|RUN_WAR|RUN_INF|DEBUG_ERR|DEBUG_WAR|DEBUG_INF|TRACE|PROFILE|OPER”日志级别可以从上述字符串中选取字符串并使用竖线组合使用，不能配置空串。

取值范围：字符串，RUN_ERR|RUN_WAR|RUN_INF|DEBUG_ERR|DEBUG_WAR|DEBUG_INF|TRACE|PROFILE|OPER。参数修改请参考表 2 进行设置。

默认值：RUN_ERR|RUN_WAR|DEBUG_ERR|OPER|RUN_INF|PROFILE

23.23.2.42 ddb_log_backup_file_count

参数说明：最大保存日志文件个数。

取值范围：整型，[1, 100]。参数修改请参考表 GUC 参数设置方式进行设置。

默认值：10

23.23.2.43 ddb_max_log_file_size

参数说明：单条日志最大字节数。

取值范围：字符串，[1M, 1000M]。参数修改请参考表 GUC 参数设置方式进行设置。

默认值：10M

23.23.2.44 ddb_log_suppress_enable

参数说明：是否开启日志抑制功能。

取值范围：整型，0：关闭；1：开启。参数修改请参考表 GUC 参数设置方式进行设置。

默认值：1

23.23.2.45 ddb_election_timeout

参数说明：dcc 选举超时时间。

取值范围：整型，[1, 600],单位：秒。参数修改请参考表 GUC 参数设置方式进行设置。

默认值：3

23.24 升级参数

23.24.1IsInplaceUpgrade

参数说明：标示是否在升级的过程中。该参数用户无法修改。

该参数属于 SUSET 类型参数，请参考表 15-1 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

23.24.2inplace_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。该参数用户无法修改。

该参数属于 SUSET 类型参数，请参考表 15-1 中对应设置方法进行设置。

取值范围：字符串

默认值：空

23.24.3upgrade_mode

参数说明：标示升级模式。

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，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

说明： 特殊情况：在使用灰度升级的情况下，若选择策略为大版本升级，即需要执行升级脚本和替换二进制包，会将 upgrade_mode 设置为 2，选择策略为小版本升级，只替换二进制包，则不会设置 upgrade_mode 设置为 2。

23.25 其它选项

23.25.1enable_l2k

参数说明：用于打开 l2k 功能开关。

参数类型：布尔。

23.25.2enable_slot_log

参数说明：打开复制槽主备同步功能。

参数类型：布尔。

23.25.3l2k_conf

参数说明：用于配置 l2k 工具信息

参数类型：字符串（json 串）

参数	描述	格式举例
dbname	数据库名	postgres
kafka_broker	kafka 的 broker 配置，用","分隔	100.0.1.54:9092,100.0.1.54:9192
kafka_topic	kafka 的 topic 配置，与 kafka_topic_table 互斥	test1
kafka_topic_tab	记录表名和 kafkatopic 映射关系的数据库表名，与 kafka_topic 互斥。	testft1

le		
kafka_param	rdkafka 连接需要额外配置的参数，json 格式	{"security.protocol":"sasl_ssl","sasl.mechanisms":"PLAIN","sasl.username":"admin","sasl.password":"admin","ssl.ca.location":"/home/gbase/ssl/ca-cert","ssl.key.password":"abcdefgh"}
schema2db	type 为 8A 类型使用。为 schema 与 db 的映射关系。用","分隔，schema 与 db 间用":"分隔	test1:public,test2:public2
type	转换类型（8A 为转换为 8A 格式，不填写为默认的 wal2json）	8A
verbose	详细日志，默认关闭	on

23.25.4enable_stream_smp

参数说明：开启 smp 必须配置参数，控制是否使能 smp。

取值范围：on 表示开启，off 表示关闭。

默认值：off

23.25.5enable_force_smp

参数说明：该参数为可选参数。如果配置在 enable_stream_smp 使能下，所有 select 查询强制走 smp。

取值范围：on 表示开启，off 表示关闭。

默认值：off。

23.25.6enable_ustore

参数说明：**指定是否开启 Ustore 存储引擎，该参数为 on 时，支持创建 Ustore 表。特别需要注意，使用 Ustore 表，必须要开启 track_counts 和 track_activities 参数，否则会引起空间膨胀。

该参数属于 POSTMASTER 类型，请参考表 GUC 参数分类对应设置方法进行设置。

取值范围：[off,on]

默认值：on

23.25.7enable_default_ustore_table

参数说明：指定是否开启默认支持 Ustore 存储引擎。该参数为 on 时，创建的表类型都为 Ustore 表。

该参数属于 USERSET 类型，请参考表 GUC 参数分类对应设置方法进行设置。

取值范围：[off,on]

默认值：off

23.25.8reserve_space_for_nullable_atts

参数说明：指定是否为 Ustore 表的可空属性预留空间。该参数为 on 时默认为 Ustore 表的可空属性预留空间。

该参数属于 USERSET 类型，请参考表 GUC 参数分类对应设置方法进行设置。

取值范围：[off,on]

默认值：off

23.25.9ustore_attr

参数说明：Ustore 测试参数。

该参数属于 USERSET 类型，可以设置包括 enable_ustore_partial_seqscan（仅在 ustore 表中顺序扫描时复制选择性列）、enable_candidate_buf_usage_count（是否脏页淘汰加入使用次数权重）、ustats_tracker_naptime（重新加载统计文件所用的时间）、umax_search_length_for_prune（扩展表前要修剪的块数）、ustore_unit_test（开启 Ustore 白盒测试）。设置方法为 ustore_attr='需要设置的参数'，例如需要设置 enable_ustore_partial_seqscan 时，ustore_attr='enable_ustore_partial_seqscan=on'。

取值范围：字符串

23.25.10 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。该

参数继承自 PostgreSQL 内核，表示当前数据库内核兼容 PostgreSQL 对应的 server_version 版本，无实际含义，为保持北向对外工具接口的生态兼容性（工具连接时查询），保留该参数。该参数不推荐使用，如想查询服务器版本号，可通过函数 opengauss_version()获取。

取值范围：字符串

默认值：9.2.4

23.25.11 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。该参数继承自 PostgreSQL 内核，表示当前数据库内核兼容 PostgreSQL 对应的 server_version_num 版本，无实际含义，为保持北向对外工具接口的生态兼容性（工具连接时查询），保留该参数。

取值范围：整数

默认值：90204

23.25.12 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

23.25.13 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

23.25.14 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

23.25.15 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

23.25.16 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 GBase 8c 安装部署时的配置。

23.25.17 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 GBase 8c 安装部署时的配置。

23.25.18 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

23.25.19 server_encoding

参数说明：报告当前数据库的服务端编码字符集。

默认情况下，gs_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。

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由当前系统环境决定的。

23.25.20 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 GUC 参数分类中对应设置方法进行设置。

取值范围： 布尔型

- on：表示提升 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：表示在对表的 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

23.25.21 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 GUC 参数分类中对应设置方法进行设

置。该参数当前版本只适用于 DWS 场景。

取值范围：字符串，设置为空表示 GBase 8c 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 GBase 8c 不可用。

23.25.22 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/。该参数当前版本只适用于 DWS 场景。

该参数属于 POSTMASTER 类型参数，请参考表 GUC 参数分类中对应设置方法进行设置。

取值范围：字符串

默认值：空

23.25.23 transparent_encrypt_kms_region

参数说明：它存储的是 GBase 8c 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。该参数当前版本只适用于 DWS 场景。

该参数属于 POSTMASTER 类型参数，请参考表 GUC 参数分类中对应设置方法进行设置。

取值范围：字符串

默认值：空

23.25.24 basebackup_timeout

参数说明：备份传输完成后连接无读写的超时时间。

通过 gs_basebackup 工具作传输时，如果指定较高压缩率时，可能在传输表空间完成后超时（客户端需要压缩传输数据）。

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：600s

23.25.25 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 GUC 参数分类中对应设置方法进行设置。

取值范围：整型，1000—60000，单位为毫秒。

默认值：1s

23.25.26 max_concurrent_autonomous_transactions

参数说明：自治事务最大链接数，同一时间自治事务执行的最大并发数。当设置为 0 时，将无法执行自治事务。

该参数属于 INTERNAL 类型参数，请参考表 GUC 参数分类中对应设置方法进行设置。

取值范围：0-1024

默认值：10

23.26 等待事件

23.26.1 enable_instr_track_wait

参数说明：是否开启等待事件信息实时收集功能。

在 x86 平台集中式部署下，硬件配置规格为 32 核 CPU/256GB 内存，使用 Benchmark SQL 5.0 工具测试性能，开关此参数性能影响约 1.4%。

该参数属于 SIGHUP 类型参数，请参考表 15-1 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

23.27 Query

23.27.1instr_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（备机不支持此能力）；从小变大不受影响。

当系统中产生的 unique sql 条目数量大于 instr_unique_sql_count 时，若开启了 unique sql 自动淘汰，则系统会按 unique sql 的更新时间由远到近自动淘汰一定比例的条目，使得新产生的 unique sql 信息可以继续被统计。若没有开启自动淘汰，则系统产生的新的 unique sql 信息将不再被统计。

在 x86 平台集中式部署下，硬件配置规格为 32 核 CPU/256GB 内存，使用 Benchmark SQL 5.0 工具测试性能，开关此参数性能影响约 3%。

该参数属于 SIGHUP 类型参数，请参考表 15-1 中对应设置方法进行设置。

取值范围：整型，0~2147483647

默认值：100

注意： - 在开启自动淘汰的情况下，如果该值设置的较小，可能会导致系统频繁的进行自动淘汰，有可能会影响数据库系统性能，所以实际场景中建议不要将该值设置的过小，建议值为 200000。

在开启自动淘汰的情况下，如果该值设置的较大（例如 38347922），清理过程中可能会引发大内存问题而无法清理。

23.27.2instr_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 15-1 中对应设置方法进行设置。

取值范围：枚举类型

top：代表只记录顶层 SQL。

默认值：top

23.27.3enable_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 15-1 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

23.27.4percentile

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 15-1 中对应设置方法进行设置。

取值范围：字符串。

默认值：80,95

23.27.5instr_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 15-1 中对应设置方法进行设置。

取值范围：整型，0~3600，单位为秒。

默认值：10s

23.27.6enable_instr_cpu_timer

参数说明：是否捕获 SQL 执行的 cpu 时间消耗 。

在 x86 平台集中式部署下，硬件配置规格为 32 核 CPU/256GB 内存，使用 Benchmark SQL 5.0 工具测试性能，开关此参数性能影响约 3.5%。

该参数属于 SIGHUP 类型参数，请参考表 15-1 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

23.27.7enable_stmt_track

参数说明：控制是否启用 Full /Slow SQL 特性。

在 x86 平台集中式部署下，硬件配置规格为 32 核 CPU/256GB 内存，使用 Benchmark SQL 5.0 工具测试性能，开关此参数性能影响约 1.2%。

该参数属于 SIGHUP 类型参数，请参考表 15-1 中对应设置方法进行设置。

取值范围：布尔型

- on：表示开启 Full /Slow SQL 捕获。
- off：表示关闭 Full /Slow SQL 捕获。

默认值：on

23.27.8track_stmt_session_slot

参数说明：设置一个 session 缓存的最大的全量/慢 SQL 的数量，超过这个数量，新的语句执行将不会被跟踪，直到落盘线程将缓存语句落盘，留出空闲的空间。

该参数属于 SIGHUP 类型参数，请参考表 15-1 中对应设置方法进行设置。

取值范围：整型，0～2147483647

默认值：1000

23.27.9track_stmt_details_size

参数说明：设置单语句可以收集的最大的执行事件的大小（byte）。

该参数属于 USERSET 类型参数，请参考表 15-1 中对应设置方法进行设置。

取值范围：整型，0 ~ 100000000

默认值：4096

23.27.10 track_stmt_retention_time

参数说明：组合参数，控制全量/慢 SQL 记录的保留时间。以 60 秒为周期读取该参数，并执行清理超过保留时间的记录，仅 sysadmin 用户可以访问。

该参数属于 SIGHUP 类型参数，请参考表 15-1 中对应设置方法进行设置。

取值范围：字符型

该参数分为两部分，形式为'full sql retention time, slow sql retention time'

full sql retention time 为全量 SQL 的保留时间，取值范围为 0 ~ 86400

slow sql retention time 为慢 SQL 的保留时间，取值范围为 0 ~ 604800

默认值：3600,604800

23.27.11 track_stmt_stat_level

参数说明：控制语句执行跟踪的级别。

该参数属于 USERSET 类型参数，请参考表 15-1 中对应设置方法进行设置，不区分英文字母大小写。

取值范围：字符型

该参数分为两部分，形式为'full sql stat level, slow sql stat level'

full sql stat level 为全量 SQL 跟踪级别，取值范围为 OFF、L0、L1、L2

slow sql stat level 为慢 SQL 的跟踪级别，取值范围为 OFF、L0、L1、L2

说明： 若全量 SQL 跟踪级别值为非 OFF 时，当前 SQL 跟踪级别值为全量 SQL 和慢 SQL 的较高级别（L2 > L1 > L0），级别说明请参见表 15-1。

默认值：OFF,L0

23.27.12 enable_auto_clean_unique_sql

参数说明：当系统中产生的 unique sql 条目数量大于等于 instr_unique_sql_count 时，是否启用 unique sql 自动淘汰功能。

该参数属于 POSTMASTER 类型参数，请参考表 15-1 中对应设置方法进行设置。

取值范围：布尔型

默认值：off

注意： 由于快照有部分信息是来源于 unique sql，所以开启自动淘汰的情况下，在生成 wdr 报告时，如果选择的起始快照和终止快照跨过了淘汰发生的时间，会导致无法生成 wdr 报告。

23.28 系统性能快照

23.28.1enable_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 15-1 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

23.28.2wdr_snapshot_retention_days

参数说明：系统中数据库监控快照数据的保留天数。当数据库运行过程期间所生成的快照量数超过保留天数内允许生成的快照数量的最大值时，系统将每隔 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 15-1 中对应设置方法进行设置。

取值范围：整型，1~8。

默认值：8

23.28.3wdr_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 15-1 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

23.28.4wdr_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 15-1 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

23.28.5asp_flush_mode

参数说明：ASP 刷新到磁盘上的方式分为写文件和写系统表，当为‘file’时，默认写文件，为‘table’时写系统表，为‘all’时，即写文件也写系统表，仅 sysadmin 用户可以访问。

该参数属于 SIGHUP 类型参数，请参考表 15-1 中对应设置方法进行设置。

取值范围：字符串，‘table’、‘file’、‘all’。

默认值：‘table’

23.28.6asp_flush_rate

参数说明：当内存中样本个数达到 asp_sample_num 时，会按一定比例把内存中样本刷

新到磁盘上，asp_flush_rate 为刷新比例。该参数为 10 时表示按 10：1 进行刷新。

该参数属于 SIGHUP 类型参数，请参考表 15-1 中对应设置方法进行设置。

取值范围：整型，1~10。

默认值：10

23.28.7asp_log_filename

参数说明：当 ASP 写文件时，该参数设置文件名的格式，仅 sysadmin 用户可以访问。

该参数属于 SIGHUP 类型参数，请参考表 15-1 中对应设置方法进行设置。

取值范围：字符串。

默认值：“asp-%Y-%m-%d_%H%M%S.log”

23.28.8asp_retention_days

参数说明：当 ASP 样本写到系统表时，该参数表示保留的最大天数。

该参数属于 SIGHUP 类型参数，请参考表 15-1 中对应设置方法进行设置。

取值范围：整型，1~7。

默认值：2

23.28.9asp_sample_interval

参数说明：每次采样的间隔。

该参数属于 SIGHUP 类型参数，请参考表 15-1 中对应设置方法进行设置。

取值范围：整型，1~10，单位为秒。

默认值：1s

23.28.10 asp_sample_num

参数说明：LOCAL_ACTIVE_SESSION 视图最大的样本个数，仅 sysadmin 用户可以访问。

该参数属于 POSTMASTER 类型参数，请参考表 15-1 中对应设置方法进行设置。

取值范围：整型，10￣100000。

默认值：100000

23.28.11 enable_asp

参数说明：是否开启活跃会话信息 active session profile。

该参数属于 SIGHUP 类型参数，请参考表 15-1 中对应设置方法进行设置。

取值范围：布尔型

- on：打开 active session profile 功能。
- off：关闭 active session profile 功能。

默认值：on

23.29 安全配置

23.29.1 elastic_search_ip_addr

参数说明：Elastic Search 系统 IP 地址。

该参数属于 POSTMASTER 类型参数，请参考表 15-1 中对应设置方法进行设置。

取值范围：字符串。

默认值：'https:127.0.0.1'

23.29.2 enable_security_policy

参数说明：安全策略开关，控制统一审计和数据动态脱敏策略是否生效。

该参数属于 SIGHUP 类型参数，请参考表 15-1 中对应设置方法进行设置。

取值范围：布尔型。

- on：安全策略开关打开。
- off：安全策略开关关闭。

默认值：off

23.29.3use_elastic_search

参数说明：使能统一审计发送日志至 Elastic Search 系统，enable_security_policy 打开且本参数打开后，统一审计日志会通过 http（https）传递至 Elastic Search 系统（默认使用 https 安全协议）。此参数打开后需要保证 elastic_search_ip_addr 对应的 es 服务可正常连通，否则进程启动失败。

该参数属于 POSTMASTER 类型参数，请参考表 15-1 中对应设置方法进行设置。

取值范围：布尔型。

- on：使能统一审计日志发送至 Elastic Search。
- off：关闭统一审计日志发送至 Elastic Search。

默认值：off

23.29.4is_sysadmin

参数说明：表示当前用户是否是初始用户。

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：是初始用户。
- off：不是初始用户。

默认值：off

23.29.5enable_tde

参数说明：透明数据加密功能开关。创建加密表前需要将此参数置为 on。当前参数值为 off 时，禁止创建新的加密表，对于已经创建的加密表只在读取数据时解密，写入数据时不再加密。

该参数属于 POSTMASTER 类型参数，请参考表 15-1 中对应设置方法进行设置。

取值范围：布尔型。

- on：开启透明数据加密功能。
- off：关闭透明数据加密功能。

默认值：off

23.29.6tde_cmk_id

参数说明：透明数据加密功能使用的数据库实例主密钥 CMK 的 ID 编号，由使用的密钥管理服务 KMS 生成。数据库实例主密钥 CMK 用于对数据加密密钥 DEK 进行加密保护，当前需要对 DEK 进行解密时，需要给 KMS 发起请求报文，将 DEK 密文和对应 CMK 的 ID 编号一起发送给 KMS。

该参数属于 SIGHUP 类型参数，请参考表 15-1 中对应设置方法进行设置。

取值范围：字符串。

默认值：””

23.30 全局临时表

23.30.1max_active_global_temporary_table

参数说明：全局临时表功能开关，控制是否可以创建全局临时表。

该参数属于 USERSET 类型参数，请参考表 15-1 中对应设置方法进行设置。

取值范围：整型，0 ~ 1000000

- 0：全局临时表功能关闭。
- > 0：全局临时表功能打开。

默认值：1000

23.30.2vacuum_gtt_defer_check_age

参数说明：vacuum 执行后检查全局临时表 relfrozenxid 与普通表的差异。如果全局临时表 relfrozenxid 落后超过指定参数值，就产生 WARNING。一般不用修改。

该参数 USERSET 类型参数，请参考表 15-1 中对应设置方法进行设置。

取值范围：整型，0～1000000

默认值：10000

23.30.3enable_gtt_concurrent_truncate

参数说明：是否支持全局临时表 truncate table 和 DML 的并发执行，以及全局临时表 truncate table 和 truncate table 的并发执行。

该参数 SIGHUP 类型参数，请参考表 15-1 中对应设置方法进行设置。

取值范围：**布尔型

on/true 表示支持上述操作并发。

off/false 表示不支持上述操作并发。

默认值：on

23.31 HyperLogLog

23.31.1hll_default_log2m

参数说明：该参数可以指定 hll 数据结构桶的个数。桶的个数会影响 hll 计算 distinct 值的精度，桶的个数越多，误差越小。误差范围为：[-1.04/2log2m*1⁄2,+1.04/2log2m*1⁄2]。

该参数属于 USERSET 类型参数，请参考表 15-1 中对应设置方法进行设置。

取值范围：整型，10~16。

默认值：14

23.31.2hll_default_log2explicit

参数说明：该参数可以用来设置从 Explicit 模式到 Sparse 模式的默认阈值大小。

该参数属于 USERSET 类型参数，请参考表 15-1 中对应设置方法进行设置。

取值范围：整型，0~12。0 表示跳过 Explicit 模式，取 1-12 表示在基数到达 2hll_default_log2explicit 时切换模式。

默认值：10

23.31.3hll_default_log2sparse

参数说明：该参数可以用来设置从 Sparse 模式到 Full 模式的默认阈值大小。

该参数属于 USERSET 类型参数，请参考表 15-1 中对应设置方法进行设置。

取值范围：整型，0~14。0 表示跳过 Explicit 模式，取 1-14 表示在基数到达 2hll_default_log2sparse 时切换模式。

默认值：12

23.31.4hll_duplicate_check

参数说明：该参数可以用来指定是否默认开启 duplicatecheck。

该参数属于 USERSET 类型参数，请参考表 15-1 中对应设置方法进行设置。

取值范围：0，1。 - 0 表示默认关闭。 - 1 表示默认开启。

默认值：0

23.32 用户自定义函数

23.32.1udf_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。本参数当前版本不生效，请使用 FencedUDFMemoryLimit 和 UDFWorkerMemHardLimit 参数控制 fenced udf worker 虚存。

该参数属于 POSTMASTER 类型参数，请参考表 15-1 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

23.32.2FencedUDFMemoryLimit

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 15-1 中对应设置方法进行设置。

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

23.32.3UDFWorkerMemHardLimit

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 15-1 中对应设置方法进行设置。

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

23.32.4pljava_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，仅 sysadmin 用户可以访问。

该参数属于 SUSET 类型参数，请参考表 15-1 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 PL/pgSQL 语言函数。

默认值：空

23.33 定时任务

23.33.1job_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，通过 gs_guc 设置，需要重启 GBase 8c 数据库才能生效。

该参数属于 POSTMASTER 类型参数，请参考表 15-1 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 时，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 设置为大于 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

23.33.2 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 15-1 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

23.34 线程池

23.34.1 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 15-1 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

23.34.2 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 15-1 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。线程池大小推荐根据硬件配置设置，计算公式为 thread_num=CPU 核数*3~5，thread_num 最大值 4096。
- 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：
  - '(nobind)' ，线程不做绑核；
  - '(allbind)'，利用当前系统所有能查询到的 CPU core 做线程绑核；

- '(nodebind: 1, 2)'，利用 NUMA 组 1，2 中的 CPU core 进行绑核；
- '(cpubind: 0-30)'，利用 0-30 号 CPU core 进行绑核。
- '(numabind: 0-30)'，在 NUMA 组内利用 0-30 号 CPU core 进行绑核。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)'

23.34.3 resilience_threadpool_reject_cond

参数说明：用于控制线程池过载逃生的堆积会话数占比。该参数仅在 GUC 参数 use_workload_manager 和 enable_thread_pool 打开时生效。该参数属于 SIGHUP 类型参数，请参考表 15-1 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 recover_threadpool_percent、overload_threadpool_percent 2 部分，这 2 个部分的具体含义如下：

recover_threadpool_percent：线程池恢复正常状态的接入会话占线程池初始设置线程数的百分比，当已经接入的会话数小于线程池初始设置数乘以该值对应的百分比后，停止过载逃生并放开新连接接入，取值为 0~INT_MAX，设置为多少表示百分之多少。

overload_threadpool_percent：线程池过载时的接入会话占线程池初始设置线程数的百分比，当已经接入的会话数大于线程池初始设置数乘以该值对应的百分比后，表示当前线程池已经过载，触发过载逃生 kill 会话并禁止新连接接入，取值为 0~INT_MAX，设置为多少表示百分之多少。

默认值：'0,0'，表示关闭线程池逃生功能。

示例：

```
resilience_threadpool_reject_cond = '100, 200'
```

表示已经堆积的会话数超过线程池初始设置的线程数的 200%后禁止新连接接入并 kill 堆积的会话，kill 会话过程中会话数恢复到线程池初始设置的线程数的 100%以下时停止 kill 会话并允许新连接接入。

须知：

- 已经堆积的会话数可以通过查询 pg_stat_activity 视图有多少条数据获得，需要过滤少量后台线程；线程池设置的初试线程池线程数目可以通过查询 thread_pool_attr

参数获得。

- 该参数如果设置的百分比过小，则会频繁触发线程池过载逃生流程，会使正在执行的会话被强制退出，新连接短时间接入失败，需要根据实际线程池使用情况慎重设置。

23.35 备份恢复

23.35.1 operation_mode

参数说明：标示系统进入备份恢复模式。

该参数属于 SIGHUP 类型参数，请参考表 15-1 中对应设置方法进行设置。

取值范围：布尔型

- on 表示在备份恢复过程中。
- off 表示不在备份恢复过程中。

默认值：off

23.35.2 enable_cbm_tracking

参数说明：当使用 roach 执行数据库实例的全量和增量备份时需要开启此参数，如果关闭会导致备份失败。

该参数属于 SIGHUP 类型参数，请参考表 15-1 中对应设置方法进行设置。

取值范围：布尔型

- on 表示追踪功能开启。
- off 表示追踪功能关闭。

默认值：off

23.35.3 hadr_max_size_for_xlog_receiver

参数说明：该参数为异地容灾参数，表示灾备数据库实例中实例获取 obs 端日志和本地回放日志的最大允许差距，若差距大于此值时停止获取 obs 端日志。

该参数属于 SIGHUP 类型参数，请参考表 15-1 中方式对应设置方法进行设置。

修改建议：该参数的取值应和本地磁盘大小相关，建议设置为磁盘大小的 50%。

取值范围：整型，0~2147483647

默认值：256GB

23.36 Undo

23.36.1 undo_space_limit_size

参数说明：用于控制 undo 强制回收阈值，达到阈值的 80%启动强制回收，用户需要根据自己的业务情况，设置该值，可以通过先设置一个较大值，然后观察实际业务运行占用 undo 空间，再将该值调整为合理值。

该参数属于 SIGHUP 类型参数，请参考表 15-1 中对应设置方法进行设置。

取值范围：整型，800MB~16TB

默认值：256GB

23.36.2 undo_limit_size_per_transaction

参数说明：用于控制单事务 undo 分配空间阈值，达到阈值时事务报错回滚。

该参数属于 SIGHUP 类型参数，请参考表 15-1 中对应设置方法进行设置。

取值范围：整型，2MB~16TB

默认值：32GB

23.37 DCF 参数设置

23.37.1 enable_dcf

参数说明：是否开启 DCF 模式，该参数不允许修改。

该参数属于 POSTMASTER 类型参数，请参考表 15-1 中对应设置方法进行设置。

取值范围：布尔型，on/off。on 表示当前安装部署方式为 DCF 模式，off 表示当前安装部署方式为非 DCF 模式。

默认值：off

23.37.2dcf_ssl

参数说明：是否开启 SSL，重启生效。

该参数属于 POSTMASTER 类型参数，请参考表 15-1 中对应设置方法进行设置。

取值范围：布尔型，on/off。on 表示使用 SSL，off 表示不使用 SSL。

默认值：on

23.37.3dcf_config

参数说明：用户安装时自定义配置信息，该参数不允许修改。

该参数属于 POSTMASTER 类型参数，请参考表 15-1 中对应设置方法进行设置。

默认值：字符串，安装时用户自定义配置

23.37.4dcf_data_path

参数说明：DCF 数据路径，该参数不允许修改。

该参数属于 POSTMASTER 类型参数，请参考表 15-1 中对应设置方法进行设置。

默认值：字符串，DN 数据目录下的 dcf_data 目录

23.37.5dcf_log_path

参数说明：DCF 日志路径，该参数不允许修改。

该参数属于 POSTMASTER 类型参数，请参考表 15-1 中对应设置方法进行设置。

默认值：字符串，DN 数据目录下的 dcf_log 目录

23.37.6dcf_node_id

参数说明：DCF 所在 DN 节点 ID，用户安装时自定义，该参数不允许修改。

该参数属于 POSTMASTER 类型参数，请参考表 15-1 中对应设置方法进行设置。

默认值：整型，安装时用户自定义配置

23.37.7dcf_max_workers

参数说明：DCF 回调函数线程个数。如果节点数量超过 7 个，需要增加这个参数的数值（比如增加到 40），否则可能会出现主节点一直处于 promoting 状态，主备节点日志不推进的状态。

该参数属于 POSTMASTER 类型参数，请参考表 15-1 中对应设置方法进行设置。

取值范围：整型，10~262143

默认值：10

23.37.8dcf_truncate_threshold

参数说明：DN 对 DCF 日志进行 truncate 的门限阈值。

该参数属于 SIGHUP 类型参数，请参考表 15-1 中对应设置方法进行设置。

取值范围：整型，1~2147483647

默认值：100000

23.37.9dcf_election_timeout

参数说明：DCF leader 和 follower 选举超时时间。选举超时时间数值依赖于当前 DN 之间的网络状况，在超时时间较小且网络极差的情形下，会有超时选举发生，待网络恢复选举恢复正常。建议根据当前网络状态合理设置超时时间。对 DCF 节点时钟的约束：DCF 节点间最大时钟差异小于选举超时时间的一半。

该参数属于 SIGHUP 类型参数，请参考表 15-1 中对应设置方法进行设置。

取值范围：整型，单位 s，1~600

默认值：3

23.37.10 dcf_enable_auto_election_priority

参数说明：DCF 优先级选主是否允许内部自动调整优先级值。0 表示不允许，1 表示允许内部自动调整。

该参数属于 SIGHUP 类型参数，请参考表 15-1 中对应设置方法进行设置。

取值范围：整型，0~1

默认值：1

23.37.11 dcf_election_switch_threshold

参数说明：DCF 防频繁切主门限。推荐根据用户业务可接受的最大故障时间配置。

该参数属于 SIGHUP 类型参数，请参考表 15-1 中对应设置方法进行设置。

取值范围：整型，单位 s，0~2147483647

默认值：0

23.37.12 dcf_run_mode

参数说明：DCF 选举模式，0 表示自动选举模式，2 表示去使能选举模式。目前去使能选举模式只限定少数派恢复场景使用，修改会导致数据库实例不可用。

该参数属于 SIGHUP 类型参数，请参考表 15-1 中对应设置方法进行设置。

取值范围：枚举类型，0、2

默认值：0

23.37.13 dcf_log_level

参数说明：DCF 日志级别。

该参数属于 SIGHUP 类型参数，请参考表 15-1 中对应设置方法进行设置。

取值范围：字符串

- 关闭日志：“NONE”，NONE 表示关闭日志打印，不能与以下日志级别混合使用。
- 开启日志：

  “RUN_ERR|RUN_WAR|RUN_INF|DEBUG_ERR|DEBUG_WAR|DEBUG_INF|TRACE|PROFILE|OPER”
- 日志级别可以从上述字符串中选取字符串并使用竖线组合使用，不能配置空串。

默认值：“RUN_ERR|RUN_WAR|DEBUG_ERR|OPER|RUN_INF|PROFILE”

23.37.14 dcf_log_backup_file_count

参数说明：DCF 运行日志备份保留个数。

该参数属于 SIGHUP 类型参数，请参考表 15-1 中对应设置方法进行设置。

取值范围：整型，1~100

默认值：10

23.37.15 dcf_max_log_file_size

参数说明：DCF 运行日志单个文件最大大小。

该参数属于 SIGHUP 类型参数，请参考表 15-1 中对应设置方法进行设置。

取值范围：整型，单位 MB，1~1000

默认值：10

23.37.16 dcf_socket_timeout

参数说明：DCF 通信模块连接 socket 超时时间，参数重启生效。对于网络环境比较差的环境，若配置很小的超时时间，可能会导致建链不成功，此时需要适当增大此值。

该参数属于 POSTMASTER 类型参数，请参考表 15-1 中对应设置方法进行设置。

取值范围：整型，单位 ms，10~600000

默认值：5000

23.37.17 dcf_connect_timeout

参数说明：DCF 通信模块建立连接超时时间，参数重启生效。对于网络环境比较差的环境，若配置很小的超时时间，可能会导致建链不成功，此时需要适当增大此值。

该参数属于 POSTMASTER 类型参数，请参考表 15-1 中对应设置方法进行设置。

取值范围：整型，单位 ms，10~600000

默认值：60000

23.37.18 dcf_mec_fragment_size

参数说明：DCF 通信模块 fragment 大小，参数重启生效。

该参数属于 POSTMASTER 类型参数，请参考表 15-1 中对应设置方法进行设置。

取值范围：整型，单位 KB，32~10240

默认值：64

23.37.19 dcf_stg_pool_max_size

参数说明：DCF 存储模内存池最大值，参数重启生效。

该参数属于 POSTMASTER 类型参数，请参考表 15-1 中对应设置方法进行设置。

取值范围：整型，单位 MB，32~2147483647

默认值：2048

23.37.20 dcf_stg_pool_init_size

参数说明：DCF 存储模块内存池最小值，参数重启生效。

该参数属于 POSTMASTER 类型参数，请参考表 15-1 中对应设置方法进行设置。

取值范围：整型，单位 MB，32~2147483647

默认值：32

23.37.21 dcf_mec_pool_max_size

参数说明：DCF 通信模块内存池最大值，参数重启生效。

该参数属于 POSTMASTER 类型参数，请参考表 15-1 中对应设置方法进行设置。

取值范围：整型，单位 MB，32~2147483647

默认值：200

23.37.22 dcf_flow_control_disk_rawait_threshold

参数说明：DCF 流控功能的磁盘等待阈值。

该参数属于 SIGHUP 类型参数，请参考表 15-1 中对应设置方法进行设置。

取值范围：整型，单位 us，0~2147483647

默认值：100000

23.37.23 dcf_flow_control_net_queue_message_num_threshold

参数说明：DCF 流控功能的网络队列消息数阈值。

该参数属于 SIGHUP 类型参数，请参考表 15-1 中对应设置方法进行设置。

取值范围：整型，0~2147483647

默认值：1024

23.37.24 dcf_flow_control_cpu_threshold

参数说明：DCF CPU 流控阈值。

该参数属于 SIGHUP 类型参数，请参考表 15-1 中对应设置方法进行设置。

取值范围：整型，单位百分比，0~2147483647

默认值：100

23.37.25 dcf_mec_batch_size

参数说明：DCF 通信批量消息数，数值为 0 时，DCF 会根据网络以及写入数据量自适应调整，参数重启生效。

该参数属于 POSTMASTER 类型参数，请参考表 15-1 中对应设置方法进行设置。

取值范围：整型，0~1024

默认值：0

23.37.26 dcf_mem_pool_max_size

参数说明：DCF 内存最大值，参数重启生效。

该参数属于 POSTMASTER 类型参数，请参考表 15-1 中对应设置方法进行设置。

取值范围：整型，单位 MB，32~2147483647

默认值：2048

23.37.27 dcf_mem_pool_init_size

参数说明：DCF 内存初始化大小，参数重启生效。

该参数属于 POSTMASTER 类型参数，请参考表 15-1 中对应设置方法进行设置。

取值范围：整型，单位 MB，32~2147483647

默认值：32

23.37.28 dcf_compress_algorithm

参数说明：DCF 运行日志传输压缩算法，参数重启生效。

该参数属于 POSTMASTER 类型参数，请参考表 15-1 中对应设置方法进行设置。

取值范围：整型

- 0 表示不压缩
- 1 表示 ZSTD 压缩算法
- 2 表示 LZ4 压缩算法

默认值：0

23.37.29 dcf_compress_level

参数说明：DCF 日志传输压缩级别，参数重启生效，此参数生效前提必须配置有效的压缩算法，即设置合法的 dcf_compress_algorithm 参数。

该参数属于 POSTMASTER 类型参数，请参考表 15-1 中对应设置方法进行设置。

取值范围：整型，1~22

若不开启压缩，配置的压缩级别将不生效。

默认值：1

23.37.30 dcf_mec_channel_num

参数说明：DCF 通信通道数量，参数重启生效。

该参数属于 POSTMASTER 类型参数，请参考表 15-1 中对应设置方法进行设置。

取值范围：整型，1~64

默认值：1

23.37.31 dcf_rep_append_thread_num

参数说明：DCF 日志复制线程数量，参数重启生效。

该参数属于 POSTMASTER 类型参数，请参考表 15-1 中对应设置方法进行设置。

取值范围：整型，1~1000

默认值：2

23.37.32 dcf_mec_agent_thread_num

参数说明：DCF 通信工作线程数量，参数重启生效。dcf_mec_agent_thread_num 值建议不少于 2*节点数*dcf_mec_channel_num。

该参数属于 POSTMASTER 类型参数，请参考表 15-1 中对应设置方法进行设置。

取值范围：整型，1~1000

默认值：10

23.37.33 dcf_mec_reactor_thread_num

参数说明：DCF 使用 reactor 线程数量，参数重启生效。dcf_mec_reactor_thread_num 与 dcf_mec_agent_thread_num 比例建议 1：40。

该参数属于 POSTMASTER 类型参数，请参考表 15-1 中对应设置方法进行设置。

取值范围：整型，1~100

默认值：1

23.37.34 dcf_log_file_permission

参数说明：DCF 运行日志文件属性，参数重启生效，参数安装阶段配置，后续不支持修改。若用户需要支持同组的其他用户访问日志，首先需要所有的父目录都支持同组的其他用户也能访问。即若参数 dcf_log_path_permission 配置为 750，dcf_log_file_permission 只能为 600 或者 640。若参数 dcf_log_path_permission 配置为 700，dcf_log_file_permission 只能

为 600。

该参数属于 POSTMASTER 类型参数，请参考表 15-1 中对应设置方法进行设置。

取值范围：枚举型，600、640

默认值：600

23.37.35 dcf_log_path_permission

参数说明：DCF 运行日志目录属性，参数重启生效，参数安装阶段配置，后续不支持修改。若用户需要支持同组的其他用户访问日志路径，需选择参数 750，否则选择 700。

该参数属于 POSTMASTER 类型参数，请参考表 15-1 中对应设置方法进行设置。

取值范围：枚举型，700、750

默认值：700

23.38 闪回相关参数

本章节介绍闪回功能相关参数。闪回功能详见《GBase 8c V5_3.0.1_管理员手册》中“5.4 闪回恢复”章节。

23.38.1 enable_recyclebin

参数说明：用来控制回收站的实时打开和关闭。

该参数属于 SIGHUP 类型参数，请参考表 15-1 中对应设置方法进行设置。

取值范围：布尔型

默认值：off

23.38.2 recyclebin_retention_time

参数说明：设置回收站对象保留时间，超过该时间的回收站对象将被自动清理。

该参数属于 SIGHUP 类型参数，请参考表 15-1 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 2147483647。

默认值：15min（即 900s）

23.38.3vacuum_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。也可以通过设置该参数，配置闪回功能旧版本保留期限。

该参数属于 SIGHUP 类型参数，请参考表 15-1 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。取值范围需要扩展到 1 亿。

默认值：0

注意： 在进行 Ustore 闪回时，无需关注该参数。其服务于之前版本的 astore 闪回功能，同时具有其他用途。本版本闪回功能已不使用。

23.38.4undo_retention_time

参数说明：设置 undo 旧版本保留时间。

该参数属于 SIGHUP 类型参数，请参考表 15-1 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 0，最大值为 2147483647。

默认值：0

23.39 回滚相关参数

23.39.1max_undo_workers

参数说明：异步回滚调用的 undoworker 线程数量，参数重启生效。

该参数属于 SIGHUP 类型参数，请参考表 15-1 中对应设置方法进行设置。

取值范围：整型，1~100

默认值：5

23.40 预留参数

说明： 下列参数为预留参数，该版本不生效。

- acce_min_datasize_per_thread
- cstore_insert_mode
- enable_constraint_optimization
- enable_hadoop_env
- enable_hdfs_predicate_pushdown
- enable_orc_cache
- schedule_splits_threshold
- backend_version
- undo_zone_count
- version_retention_age

23.41 AI 特性

23.41.1 enable_hypo_index

参数说明：该参数控制数据库的优化器进行 EXPLAIN 时是否考虑创建虚拟索引。通过对特定的查询语句执行 explain，用户可根据优化器给出的执行计划评估该索引是否能够提升该查询语句的执行效率。

该参数属于 USERSET 类型参数，请参考表 15-1 中对应设置方法进行设置。

取值范围：布尔型

- on 表示在进行 EXPLAIN 时创建虚拟索引。
- off 表示在进行 EXPLAIN 时不创建虚拟索引。

默认值：off

23.41.2db4ai_snapshot_mode

参数说明：snapshot 有 2 种模式：MSS（物化模式，存储数据实体）和 CSS（计算模式，存储增量信息）。

该参数属于 USERSET 类型参数，请参考表 15-1 中对应设置方法进行设置。

取值范围：字符串，MSS/CSS

- MSS 表示物化模式，db4ai 在创建快照的时候存储数据实体。
- CSS 表示计算模式，db4ai 在创建快照的时候存储增量信息。

默认值：MSS

23.41.3db4ai_snapshot_version_delimiter

参数说明：该参数为数据表快照版本分隔符。

该参数属于 USERSET 类型参数，请参考表 15-1 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：@

23.41.4db4ai_snapshot_version_separator

参数说明：该参数用于指定数据表快照子版本分隔符。

该参数属于 USERSET 类型参数，请参考表 15-1 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：.

23.41.5unix_socket_directory

参数说明：用于指定 unix_socket 通信方式中，文件存放的路径。此参数只能在配置文件 postgresql.conf 中指定。再启动 fenced 模式前需要设定该 GUC 参数。

该参数属于 POSTMASTER 类型参数，请参考表 15-1 中对应设置方法进行设置。

取值范围：字符串，长度大于等于 0

默认值：“

23.42 Global SysCache 参数

23.42.1enable_global_syscache

参数说明：控制是否使用全局系统缓存功能。该参数属于 POSTMASTER 类型参数，请参考表 15-1 中对应设置方法进行设置。

取值范围：布尔型

- on 表示开启全局系统缓存功能。
- off 表示不开启全局系统缓存功能。

默认值：on

推荐结合线程池参数使用。打开该参数后，如果需要访问备机，建议设置备机 wal_level 级别为 hot_standby 以上。

23.42.2global_syscache_threshold

参数说明：全局系统缓存内存最大占用大小。

该参数属于 PGC_SIGHUP 类型参数，请参考表 15-1 中对应设置方法进行设置。

需要打开 enable_global_syscache 参数。

取值范围：整型，16384~1073741824，单位为 kB。

默认值：163840

推荐计算公式：热点 DB 个数和线程个数的最小值乘以每个 DB 分配的内存大小

即 global_syscache_threshold = min(count(hot dbs)，count(threads)) * memofdb

热点 DB 数即访问较为频繁的数据库，线程数在线程池模式下取线程池线程个数和后台线程个数之和，非线程池模式不需要计算这个值，直接使用热点 DB 数。

memofdb 即平均每个 db 应该分配的内存，每个 DB 的底噪内存是 2M，平均每增加一个表或者索引，增加 11k 内存。

如果设置的值过小，会导致内存频繁淘汰，内存存在大量碎片无法回收，导致内存控制失效。

GBASE®

南大通用数据技术股份有限公司
General Data Technology Co., Ltd.

微信二维码

技术支持热线：400-013-9696