分布式服务开发指南

目录

[1、前言 8](#_Toc3796157)

[1.1、范围 8](#_Toc3796158)

[1.2、术语 8](#_Toc3796159)

[1.2.1、腾讯微服务平台 8](#_Toc3796160)

[1.2.2、TBase 9](#_Toc3796161)

[2、TSF概述 9](#_Toc3796162)

[2.1、微服务应用基本框架 9](#_Toc3796163)

[2.2、服务依赖 10](#_Toc3796164)

[2.3、工具准备 11](#_Toc3796165)

[2.4、标准开发环境配置 11](#_Toc3796166)

[2.5、代码版本管理 14](#_Toc3796167)

[3、TSF开发规范 15](#_Toc3796168)

[3.1、Yaml格式 15](#_Toc3796169)

[3.2、常量 16](#_Toc3796170)

[3.3、控制器 19](#_Toc3796171)

[3.4、服务接口 20](#_Toc3796172)

[3.5、服务实现 21](#_Toc3796173)

[3.6、Dao数据交互层 23](#_Toc3796174)

[3.7、异常处理 23](#_Toc3796175)

[3.8、工具 24](#_Toc3796176)

[3.9、代理与远程接口 25](#_Toc3796177)

[3.10、业务日志 27](#_Toc3796178)

[3.11、本地调试 28](#_Toc3796179)

[4、 Spring Cloud应用接入 29](#_Toc3796180)

[4.1、服务注册与发现 29](#_Toc3796181)

[4.1.1、创建服务提供者 29](#_Toc3796182)

[4.1.2、创建服务消费者 31](#_Toc3796183)

[4.1.3、本地开发调试 34](#_Toc3796184)

[4.1.4、TSF 中部署应用 36](#_Toc3796185)

[4.1.5、从 Eureka 迁移 36](#_Toc3796186)

[4.2、轻量级服务注册中心 37](#_Toc3796187)

[4.2.1、下载轻量级服务注册中心 37](#_Toc3796188)

[4.2.2、本地使用轻量服务注册中心 37](#_Toc3796189)

[4.2.3、多微服务联调环境的轻量服务注册中心 38](#_Toc3796190)

[4.3、分布式配置 39](#_Toc3796191)

[4.3.1、依赖项 39](#_Toc3796192)

[4.3.2、代码中引用配置 39](#_Toc3796193)

[4.3.3、通过 TSF 平台下发动态配置 41](#_Toc3796194)

[4.4、服务鉴权 42](#_Toc3796195)

[4.5、服务限流 44](#_Toc3796196)

[4.6、参数传递 44](#_Toc3796197)

[4.6.1、元数据类别 44](#_Toc3796198)

[4.6.2、元数据的传递性 45](#_Toc3796199)

[4.6.3、使用元数据 46](#_Toc3796200)

[4.7、调用链 47](#_Toc3796201)

[4.8、API注册 48](#_Toc3796202)

[5、TBase数据库概述 49](#_Toc3796203)

[5.1、TBase设计目标 49](#_Toc3796204)

[5.2、TBase集群构架 50](#_Toc3796205)

[5.3、TBase安装 51](#_Toc3796206)

[5.3.1、运行环境硬件要求 51](#_Toc3796207)

[5.3.2、操作系统要求 51](#_Toc3796208)

[5.3.3、如何获取TBase安装包 52](#_Toc3796209)

[5.3.4、安装过程 52](#_Toc3796210)

[5.4 、配置tbase的客户端使用环境 52](#_Toc3796211)

[5.5、使用TBase对开发者的基本要求 53](#_Toc3796212)

[5.6 、TBase每个cn/dn节点的系统限制 53](#_Toc3796213)

[6、创建TBase默认的分布式使用环境 53](#_Toc3796214)

[6.1、创建数据表默认的default group 53](#_Toc3796215)

[6.2、为default group创建shardmap 54](#_Toc3796216)

[6.3、更多group的使用方法 55](#_Toc3796217)

[6.3.1、创建扩展group 55](#_Toc3796218)

[6.3.2、删除group 55](#_Toc3796219)

[6.3.3、查询集群中group的数量 55](#_Toc3796220)

[7、开发者工具介绍和使用 56](#_Toc3796221)

[7.1、可视化开发工具TStudio 56](#_Toc3796222)

[7.1.1、如何打开TStudio 56](#_Toc3796223)

[7.1.2、TStudio主界面说明 57](#_Toc3796224)

[7.1.3、如何添加要管理的节点 57](#_Toc3796225)

[7.1.4、如何创建数据表、索引 58](#_Toc3796226)

[7.1.5、运行手工编写脚本 63](#_Toc3796227)

[7.1.6、表数据导入导出数据 64](#_Toc3796228)

[7.2、shell交互客户端psql 66](#_Toc3796229)

[7.2.1、连接到一个数据库 66](#_Toc3796230)

[7.2.2、建立一个新连接 68](#_Toc3796231)

[7.2.3、显示和设置该连接当前运行参数 68](#_Toc3796232)

[7.2.4、退出连接 74](#_Toc3796233)

[7.2.5、psql执行一个sql命令 74](#_Toc3796234)

[7.2.6、psql执行一个sql文件中所有命令 74](#_Toc3796235)

[7.2.7、调用编辑器编写sql脚本 75](#_Toc3796236)

[7.2.8、调用外部命令 76](#_Toc3796237)

[7.2.9、将执行的结果保存到文件 76](#_Toc3796238)

[7.2.10、改变当前的工作目录 77](#_Toc3796239)

[7.2.11、插件管理 77](#_Toc3796240)

[7.2.12、数据库相关操作 78](#_Toc3796241)

[7.2.13、模式相关操作 79](#_Toc3796242)

[7.2.14、用户相关操作 80](#_Toc3796243)

[7.2.15、表相关操作 82](#_Toc3796244)

[7.2.16、视图相关操作 84](#_Toc3796245)

[7.2.17、物化视图相关操作 87](#_Toc3796246)

[7.2.18、序列相关操作 90](#_Toc3796247)

[7.2.19、索引相关操作 93](#_Toc3796248)

[7.2.20、函数相关操作 95](#_Toc3796249)

[7.2.21、自定义数据类型相关操作 101](#_Toc3796250)

[7.2.22、存储过程语句相关操作 103](#_Toc3796251)

[7.2.23、列出表、视图和序列和它们相关的访问权限 103](#_Toc3796252)

[7.2.24、列出库或用户定义的配置 105](#_Toc3796253)

[7.2.25、copy命令的使用 105](#_Toc3796254)

[7.2.26、copy FROM stdin使用方法 106](#_Toc3796255)

[7.2.27、打印当前查询缓冲区到标准输出 106](#_Toc3796256)

[7.2.28、自定义显示格式 106](#_Toc3796257)

[7.2.29、显示psql内部操作 108](#_Toc3796258)

[7.2.30、重复执行上一条语句 109](#_Toc3796259)

[7.2.31、sql命令帮助查看 110](#_Toc3796260)

[8、TBase数据库开发基础 111](#_Toc3796261)

[8.1、TBase的数据类型 111](#_Toc3796262)

[8.1.1、数字类型 111](#_Toc3796263)

[8.1.2、字符类型 112](#_Toc3796264)

[8.1.3、二进制数据类型 112](#_Toc3796265)

[8.1.4、日期类型 112](#_Toc3796266)

[8.1.5、布尔类型 113](#_Toc3796267)

[8.1.6、更多的数据类型介绍 113](#_Toc3796268)

[8.1.7、异构数据库类型对照表 113](#_Toc3796269)

[8.2、自增列于序列的用法 116](#_Toc3796270)

[8.2.1、序列的创建和访问 116](#_Toc3796271)

[8.2.2、序列在DML中使用 118](#_Toc3796272)

[8.2.3、序列做为字段的默认值使用 118](#_Toc3796273)

[8.2.4、序列做为字段类型使用 119](#_Toc3796274)

[8.2.5、删除序列 119](#_Toc3796275)

[8.3、库/模式/表/索引/视图/物化视图等DDL操作 120](#_Toc3796276)

[8.3.1、数据库管理 120](#_Toc3796277)

[8.3.2、模式管理 125](#_Toc3796278)

[8.3.3、创建和删除数据表 131](#_Toc3796279)

[8.3.4、创建和删除索引 147](#_Toc3796280)

[8.3.5、修改表结构 163](#_Toc3796281)

[8.3.6、视图创建和删除 169](#_Toc3796282)

[8.3.7、物化视图使用 172](#_Toc3796283)

[8.3.8、truncate操作 174](#_Toc3796284)

[8.4、select语句 177](#_Toc3796285)

[8.4.1、访问函数 177](#_Toc3796286)

[8.4.2、数据排序 177](#_Toc3796287)

[8.4.3、where条件使用 182](#_Toc3796288)

[8.4.4、分页查询 186](#_Toc3796289)

[8.4.5、合并多个查询结果 187](#_Toc3796290)

[8.4.6、返回两个结果的交集 188](#_Toc3796291)

[8.4.7、返回两个结果的差集 189](#_Toc3796292)

[8.4.8、any用法 190](#_Toc3796293)

[8.4.9、all用法 190](#_Toc3796294)

[8.4.10、聚集查询 191](#_Toc3796295)

[8.4.11、多表关联 193](#_Toc3796296)

[8.4.12、聚合函数并发计算 194](#_Toc3796297)

[8.4.13、not in中包含了null，结果全为真 195](#_Toc3796298)

[8.4.14、只查某个dn的数据 196](#_Toc3796299)

[8.4.15、特殊应用 197](#_Toc3796300)

[8.5、insert语句 198](#_Toc3796301)

[8.5.1、插入单条记录 198](#_Toc3796302)

[8.5.2、插入多数记录 200](#_Toc3796303)

[8.5.3、使用子查询插入数据 200](#_Toc3796304)

[8.5.4、从另外一个表取数据进行批量插入 200](#_Toc3796305)

[8.5.5、大批量的生成数据 201](#_Toc3796306)

[8.5.6、返回插入数据，轻松获取插入记录的serial值 201](#_Toc3796307)

[8.5.7、insert..update更新 202](#_Toc3796308)

[8.6、update语句 203](#_Toc3796309)

[8.6.1、单表更新 203](#_Toc3796310)

[8.6.2、多表关联更新 203](#_Toc3796311)

[8.6.3、返回更新的数据 204](#_Toc3796312)

[8.6.4、多列匹配更新 204](#_Toc3796313)

[8.6.5、shard key不准许更新 204](#_Toc3796314)

[8.7、delete语句 205](#_Toc3796315)

[8.7.1、带条件删除 205](#_Toc3796316)

[8.7.2、多表关联删除数据 205](#_Toc3796317)

[8.7.3、返回删除数据 206](#_Toc3796318)

[8.7.4、删除所有数据 206](#_Toc3796319)

[8.8、游标的使用 207](#_Toc3796320)

[8.8.1、定义一个游标 207](#_Toc3796321)

[8.8.2、提取下一行数据 207](#_Toc3796322)

[8.8.3、提取前一行数据 208](#_Toc3796323)

[8.8.4、提取最后一行 208](#_Toc3796324)

[8.8.5、提取第一行 209](#_Toc3796325)

[8.8.6、提取该查询的第x行 209](#_Toc3796326)

[8.8.7、提取当前位置后的第x行 210](#_Toc3796327)

[8.8.8、向前提取x行数据 210](#_Toc3796328)

[8.8.9、向前提取剩下的所有数据 211](#_Toc3796329)

[8.8.10、向后提取x行数据 211](#_Toc3796330)

[8.8.11、向后提取剩下的所有数据 212](#_Toc3796331)

[8.9、copy的使用 212](#_Toc3796332)

[8.9.1、实验表结构及数据 212](#_Toc3796333)

[8.9.2、copy to用法详解--复制数据到文件中 213](#_Toc3796334)

[8.9.3、copy from用法详解--复制文件内容到数据表中 222](#_Toc3796335)

[8.10、json/jsonb的使用 233](#_Toc3796336)

[8.10.1、json应用 233](#_Toc3796337)

[8.10.2、jsonb应用 235](#_Toc3796338)

[8.10.3、jsonb函数应用 238](#_Toc3796339)

[8.10.4、jsonb索引使用 240](#_Toc3796340)

[9、TBase的进阶开发 242](#_Toc3796341)

[9.1、TBase事务控制 242](#_Toc3796342)

[9.1.1、开始一个事务 242](#_Toc3796343)

[9.1.2、提交事务 242](#_Toc3796344)

[9.1.3、回滚事务 244](#_Toc3796345)

[9.1.4、事务读一致性REPEATABLE READ 245](#_Toc3796346)

[9.1.5、行锁在事务中的运用 246](#_Toc3796347)

[9.2、TBase相关开发规范 251](#_Toc3796348)

[9.2.1、命名规范 251](#_Toc3796349)

[9.2.2、COLUMN设计 251](#_Toc3796350)

[9.2.3、Constraints 设计 251](#_Toc3796351)

[9.2.4、Index 设计 252](#_Toc3796352)

[9.2.5、关于NULL 252](#_Toc3796353)

[9.2.6、开发相关规范 256](#_Toc3796354)

[9.3、高级sql语句编写 264](#_Toc3796355)

[9.3.1、窗口函数的使用 264](#_Toc3796356)

[10、TBase-PL/pgsql开发 287](#_Toc3796357)

[10.1、应用程序语法介绍 287](#_Toc3796358)

[10.1.1、建立函数语法 287](#_Toc3796359)

[10.1.2、[OR REPLACE] 更新函数介绍 288](#_Toc3796360)

[10.1.3、[模式名.]函数名介绍 289](#_Toc3796361)

[10.2、参数详细介绍 291](#_Toc3796362)

[10.2.1、参数模式 291](#_Toc3796363)

[10.2.2、参数引用 296](#_Toc3796364)

[10.2.3、参数数据类型 298](#_Toc3796365)

[10.3、返回值详细介绍 309](#_Toc3796366)

[10.3.1、返回值介绍 309](#_Toc3796367)

[10.3.2、返回值类型介绍 309](#_Toc3796368)

[10.4、变量使用 328](#_Toc3796369)

[10.4.1、变量使用介绍 328](#_Toc3796370)

[10.4.2、变量使用实例 328](#_Toc3796371)

[10.5、控制结构 334](#_Toc3796372)

[10.5.1、判断语句 334](#_Toc3796373)

[10.5.2、循环语句 338](#_Toc3796374)

[10.5.3、其它控制语句 346](#_Toc3796375)

[10.5.4、俘获错误 349](#_Toc3796376)

[10.6、触发器函数 352](#_Toc3796377)

[10.6.1、INSERT事件触发器函数 352](#_Toc3796378)

[10.6.2、UPDATE事件触发器函数 353](#_Toc3796379)

[10.6.3、DELETE事件触发器函数 354](#_Toc3796380)

[10.6.4、删除触发器 355](#_Toc3796381)

[10.6.5、更多的触发器函数使用 355](#_Toc3796382)

[10.7、消息及异常输出 356](#_Toc3796383)

[10.7.1、RAISE NOTICE 356](#_Toc3796384)

[10.7.2、RAISE EXCEPTION 356](#_Toc3796385)

[10.7.3、RAISE EXCEPTION 自定义ERRCODE 357](#_Toc3796386)

[10.8、pg/pgsql函数实战 358](#_Toc3796387)

[10.8.1、批量设置表owner的函数 358](#_Toc3796388)

[10.8.2、批量设置表的加密规则函数 360](#_Toc3796389)

[10.8.3、oracle to\_date函数的实现 361](#_Toc3796390)

[11、Tbase问题定位及性能优化 362](#_Toc3796391)

[11.1、访问日志管理 362](#_Toc3796392)

[11.1.1、配置TBase日志 362](#_Toc3796393)

[11.1.2、TBase日志格式说明 365](#_Toc3796394)

[11.1.3、对日志进行分析 367](#_Toc3796395)

[11.1.4、配置只收集慢的sql语句 370](#_Toc3796396)

[11.2、如何查询数据是否倾斜 370](#_Toc3796397)

[11.3、如何优化有问题的Sql语句 372](#_Toc3796398)

[11.3.1、查看是否为分布键查询 372](#_Toc3796399)

[11.3.2、查看是否使用上索引 373](#_Toc3796400)

[11.3.3、查看是否为分布key join 374](#_Toc3796401)

[11.3.4、查看join发生的节点 375](#_Toc3796402)

[11.3.5、查看并行的worker数 376](#_Toc3796403)

[11.3.6、检查各个节点的执行计划是否一致 377](#_Toc3796404)

[11.4、优化实例 378](#_Toc3796405)

[11.4.1、count(distinct xx)优化 378](#_Toc3796406)

[11.4.2、增大work\_mem减少io访问 381](#_Toc3796407)

[11.4.3、not in改写为anti join 384](#_Toc3796408)

[11.4.4、分布key jon+limit优化 386](#_Toc3796409)

[11.4.5、非分布key join使用hash join性能一般最好 390](#_Toc3796410)

[11.4.6、exists的优化 395](#_Toc3796411)

[12、TBase-插件使用 397](#_Toc3796412)

[12.1、插件查看，添加和删除 397](#_Toc3796413)

[12.1.1、查看数据库加载了那些插件 397](#_Toc3796414)

[12.1.2、添加插件 398](#_Toc3796415)

[12.1.3、删除插件 398](#_Toc3796416)

[12.2、插件uuid-ossp使用 398](#_Toc3796417)

[12.2.1、uuid-ossp功能介绍及添加方法 398](#_Toc3796418)

[12.2.2、uuid常量函数 399](#_Toc3796419)

[12.2.3、uuid生成函数 401](#_Toc3796420)

[12.2.4、在数据表中使用uuid默认值 403](#_Toc3796421)

[12.2.5、uuid各函数性能对比 404](#_Toc3796422)

[12.2.6、uuid与serial 在TBase中性能对比 406](#_Toc3796423)

[12.2.7、占用空间对比 409](#_Toc3796424)

[12.2.8、使用uuid做为分布列的方案 410](#_Toc3796425)

[12.2.9、使用建议 411](#_Toc3796426)

[12.3、插件pg\_stat\_statements使用 411](#_Toc3796427)

[12.3.1、pg\_stat\_statements功能介绍及添加方法 411](#_Toc3796428)

[12.3.2、获取执行次数最多的语句 412](#_Toc3796429)

[12.3.3、获取执行总时间最长的语句 412](#_Toc3796430)

[12.3.4、获取每句平均执行时间最长的语句 413](#_Toc3796431)

[12.3.5、获取buffer读最多的语句 413](#_Toc3796432)

[12.4、插件pg\_trgm使用 414](#_Toc3796433)

[12.4.1、pg\_trgm功能介绍及添加方法 414](#_Toc3796434)

[12.4.2、添加pg\_trgm 插件 414](#_Toc3796435)

[12.4.3、测试环境准备 415](#_Toc3796436)

[12.4.4、gist索引测试 415](#_Toc3796437)

[12.4.5、gin索引测试 421](#_Toc3796438)

[12.4.6、无索引字段测试 427](#_Toc3796439)

[12.4.7、数据对比总结及例外 430](#_Toc3796440)

[13、Tbase应用程序样例 431](#_Toc3796441)

[13.1、java 431](#_Toc3796442)

[13.1.1、创建数据表 431](#_Toc3796443)

[13.1.2、插入数据 433](#_Toc3796444)

[13.1.4、扩展协议插入数据 434](#_Toc3796445)

[13.1.3、copy from 入库 435](#_Toc3796446)

[13.1.4、copy to出库 437](#_Toc3796447)

[13.1.5、jdbc下载 438](#_Toc3796448)

[13.2、C程序 438](#_Toc3796449)

[13.2.1、连接服务 438](#_Toc3796450)

[13.2.2、建立数据表 439](#_Toc3796451)

[13.2.3、插入数据 441](#_Toc3796452)

[13.2.4、查询数据 442](#_Toc3796453)

[13.2.5、copy入库 444](#_Toc3796454)

[13.3、shell程序 446](#_Toc3796455)

[13.4、python程序 448](#_Toc3796456)

[13.4.1、安装psycopg2模块 448](#_Toc3796457)

[13.4.2、连接服务 448](#_Toc3796458)

[13.4.3、创建数据表 448](#_Toc3796459)

[13.4.4、新增数据 449](#_Toc3796460)

[13.4.5、查询数据 450](#_Toc3796461)

[13.4.6、copy from方法 451](#_Toc3796462)

[13.5、php程序 452](#_Toc3796463)

[13.5.1、连接服务 452](#_Toc3796464)

[13.5.2、创建数据表 453](#_Toc3796465)

[13.5.3、插入数据 455](#_Toc3796466)

[13.5.4、查询记录 456](#_Toc3796467)

[13.5.5、copy from 方法 458](#_Toc3796468)

[13.5.6、copy to 方法 460](#_Toc3796469)

[13.5.7、入库去重方法 461](#_Toc3796470)

[13.6、golang程序 467](#_Toc3796471)

[13.6.1、连接服务 467](#_Toc3796472)

[13.6.2、建立数据表 470](#_Toc3796473)

[13.6.3、插入数据 472](#_Toc3796474)

[13.6.4、查询数据 477](#_Toc3796475)

[13.6.5、copy from 方法 482](#_Toc3796476)

[13.6.6、go相关资源包 485](#_Toc3796477)

# 1、前言

## 1.1、范围

本文从分布式服务开发的两大核心部分——应用程序和数据库，分别阐述分布式服务开的规范和指南。

应用于所有研发。

## 1.2、术语

### 1.2.1、腾讯微服务平台

腾讯微服务平台 （Tencent Service Framework） 是一个围绕着应用和微服务的 PaaS 平台，提供应用全生命周期管理、数据化运营、立体化监控和服务治理等功能。TSF 拥抱 Spring Cloud 、Service Mesh 微服务框架，帮助企业客户解决传统集中式架构转型的困难，打造大规模高可用的分布式系统架构，实现业务、产品的快速落地。

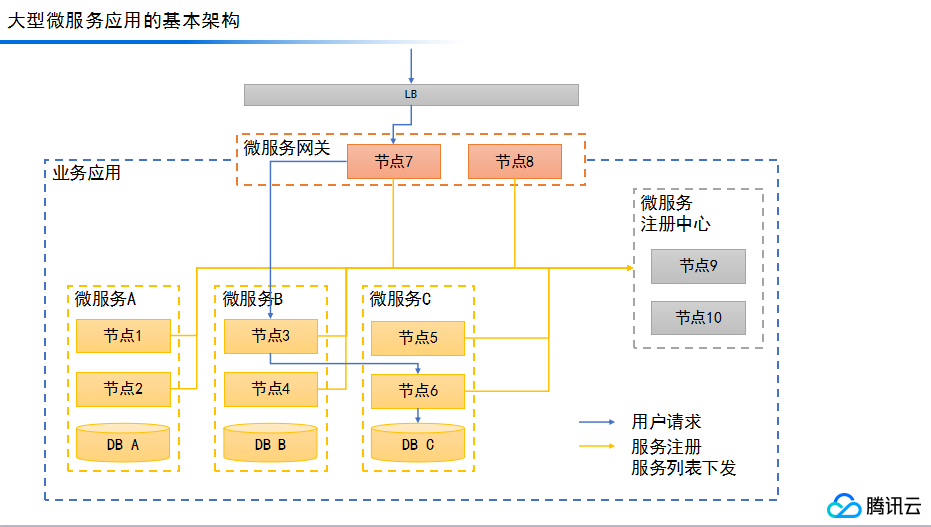
### 1.2.2、TBase

腾讯TBase分布式关系型数据库是一款面向海量在线实时分布式事务交易和MPP实时数据分析通用型高性能数据库系统。面对应用业务产生的不定性数据爆炸需求，不管是高并发的交易还是海量的实时数据分析，TBase都有足够能力处理。TBase产品原型起源于技术成熟，功能强大的PostgreSQL，PostgreSQL经历了30多年的发展，是目前世界上最好的企业级开源关系型数据库。在此基础上我们构造和发行了具有功能更丰富、稳定性更好、兼容性更广、安全性更高、性能更强、扩展性极好的分布式数据库TBase产品。腾讯公司对TBase完全自主可控,具有完全自主知识产权. 具备在中高端市场可以规模替代国外数据库的能力，在数据库基础软件层面支撑国家自主可控战略发展。当前TBase已经在腾讯微信支付、金融、保险、电力、公安、消防、政务等行业的核心交易系统上线运行。

# 2、TSF概述

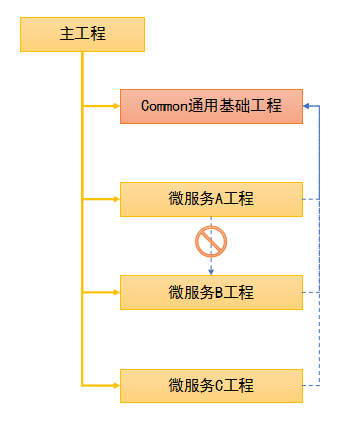
## 2.1、微服务应用基本框架

* 包括微服务网关在内的全部服务都是微服务。
* 所有服务节点都向服务注册中心注册自身，并接受服务列表下发。



## 2.2、服务依赖

* 业务的微服务间不产生依赖
* 所有业务微服务依赖同一个基础工程
* 微服务间不相互依赖



## 2.3、工具准备

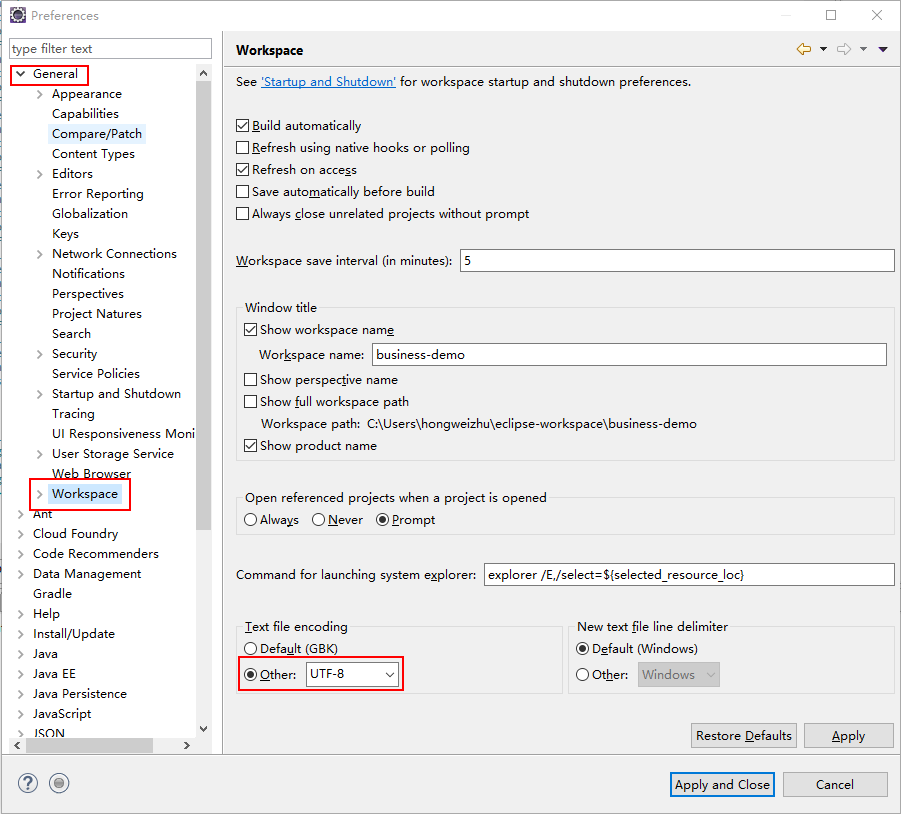
* 包仓库管理器：Maven
* IDE：Eclipse
* 版本控制工具：GIT
* 本地数据库/开发环境数据库：MySQL，Oracle
* 一个单机版的Consul
* Lombok的插件jar包

## 2.4、标准开发环境配置

* 统一字符集：UTF-8
* 以Eclipse为例：

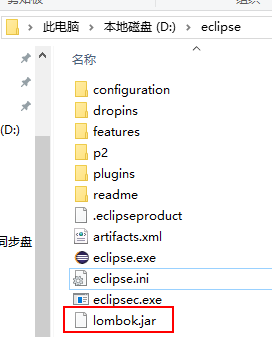
Window > Preferences > General > Workspace

右侧窗口中间下部红框位置选择UTF-8



* 安装Lombok插件

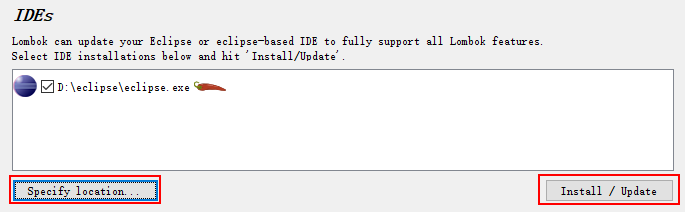
将下载的lombok.jar放到eclipse安装目录下



在命令行中进入eclipse安装目录运行该jar包

* java -jar lombok.jar

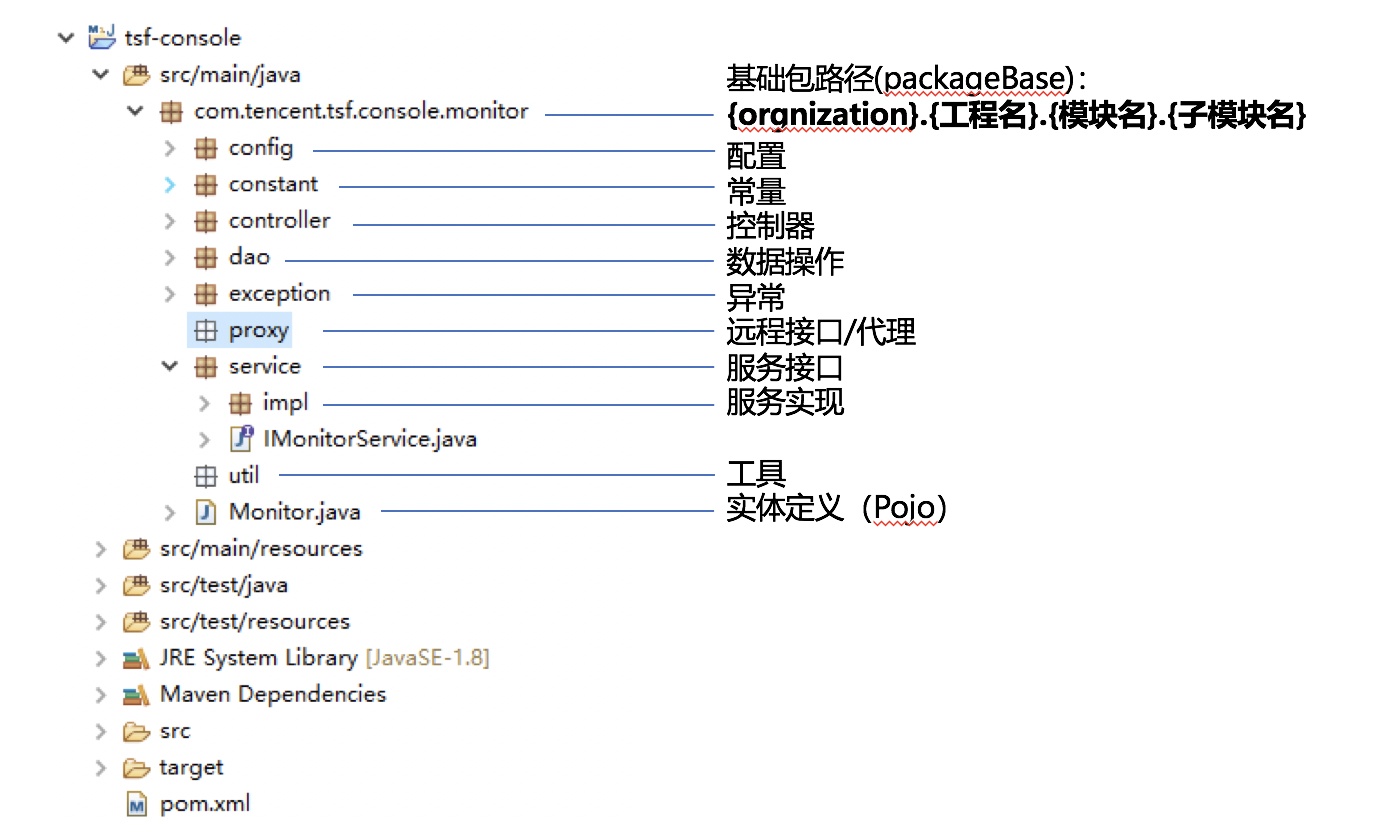
忽略提示的错误，选择eclipse安装目录，并点击Install / Update按钮



安装完毕后，确认eclipse.ini中新增了如下配置



* 单微服务工程的包结构



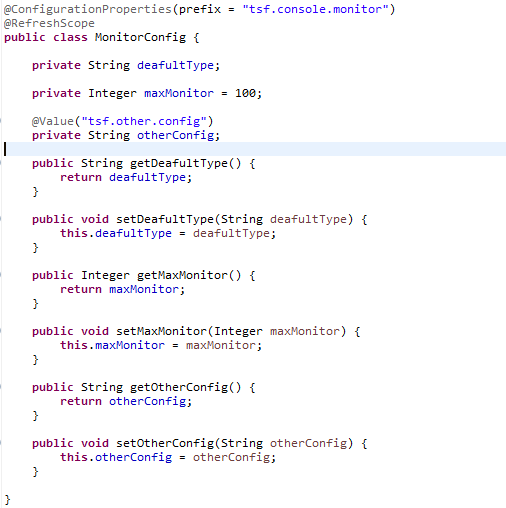
* Config类用于加载应用需要的配置或者指定注入某些定制Bean

通常会使用@ConfigurationProperties注解加载某一前缀下下的所有配置到结构化对象中。

使用@RefreshScope标记该配置能够被动态刷新。

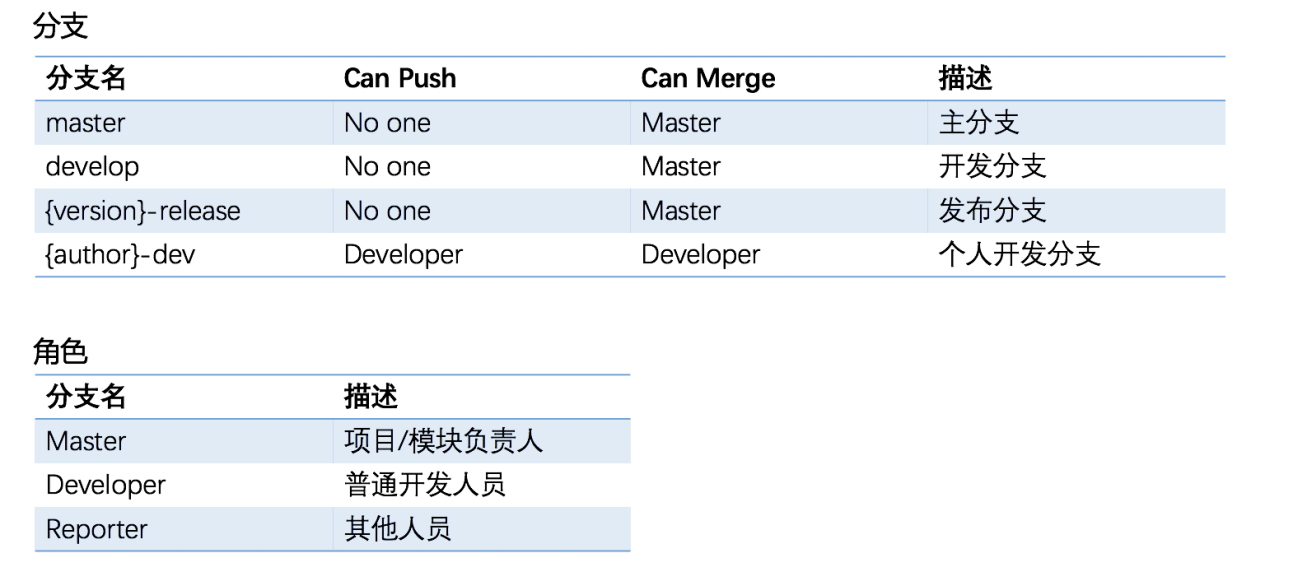
注：@Value方式的配置不会被动态刷新。

有些时候，我们也可以从spring的上下文中直接获取配置，此时配置总是动态刷新的。



## 2.5、代码版本管理

* 代码版本管理建议按照如下方式：



* master分支：

master分支保持任何时候可发布状态。

* develop分支：

develop分支是变动最频繁的分支，可向开发/联调环境发布代码。

* release分支：

当预发布准备时，从develop分支拉取当前版本的release分支。

当确认版本结束时，将release分支合并到master分支，并打上版本的Tag。

* 个人开发分支：

用户开发时，从develop分支拉取到自己的开发分支，开发完成后再通过提交MR的方式合入develop分支。

当存在release分支时，除提交MR到develop分支之外还需要额外提交Cherry Pick到release分支。

# 3、TSF开发规范

## 3.1、Yaml格式

* 语法规则

YAML 的基本语法规则如下：

- 大小写敏感。

- 使用缩进表示层级关系。

- 缩进时 不允许 使用Tab键，只允许使用空格。

- 缩进的空格数目不重要，只要相同层级的元素左侧对齐即可。

* 数据结构

YAML 支持三种数据结构：对象、数组和纯量。

- 对象：键值对的集合，又称为映射（mapping）/ 哈希（hashes） / 字典（dictionary）

- 数组：一组按次序排列的值，又称为序列（sequence） / 列表（list）

- 纯量（scalars）：单个的、不可再分的值

## 3.2、常量

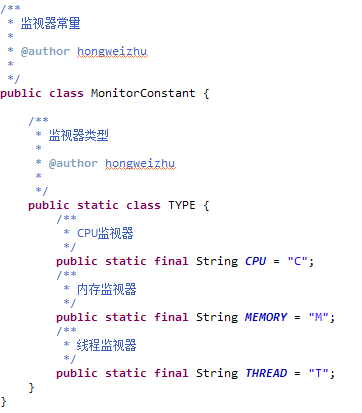
* 通常我们会定义一个模块级的常量便于统一配置

常量中仅有静态变量和静态类。

静态Map的初始化可以使用static语句块。

我们偶尔会对某些枚举值进行合法性校验，可以使用常量定义和一个标准工具进行





* 目前建议将异常信息同意定义在一个枚举中

统一定义的异常信息结合我们的标准异常能够在业务代码中标准化的快速跑出业务异常信息。

集中定义可以极大程度的降低维护/统一化异常的工作量。

统一的异常结构，利于进行统一化处理。

标准的java doc注释能够提升我们的代码可读性。

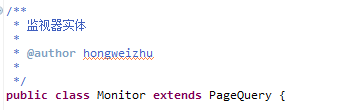
* 根据业务实体创建模型

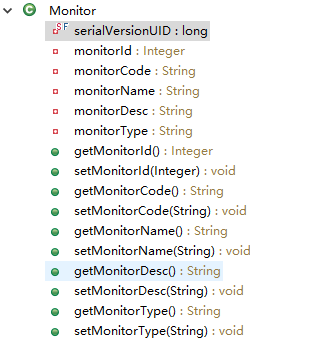
每个实体只创建实体相关的必要字段。

每个实体都要继承基础实体。

字段请使用私有封装对象，并声明setter/getter方法。

标准字段可以统一声明在基础实体中。





* 使用Lombok插件

Lombok插件是一个截止到编译时的插件。

Lombok通过类注解启用。

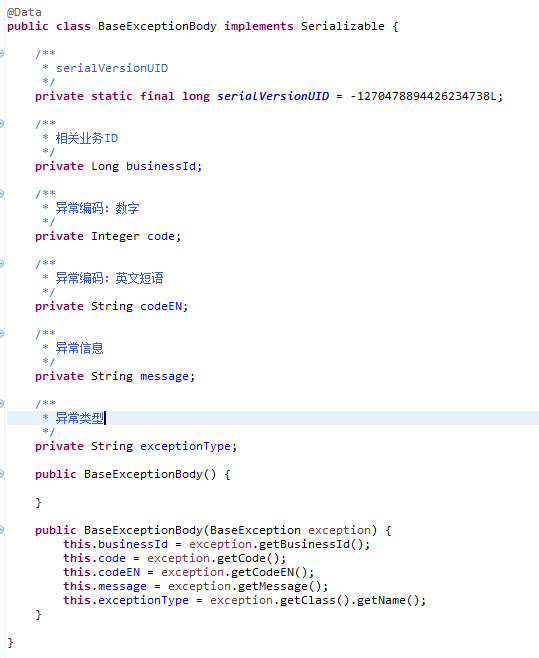
@Data：生成getter/setter，equalHash，toString

@Getter：仅生成getter

@Setter：仅生成setter

Lombok可以简化我们编写的源代码，其运行机制会在编译时为我们生成相关getter/setter方法。

在使用Lombok时，需要注意父类可能需要额外



## 3.3、控制器

* 控制器是微服务暴露REST接口的主要途径

Controller中的接口是面向功能级的。

请注解@RestController。

在类上使用@RequestMapping注解统一路径前缀。

在方法上使用@RequestMapping注解接口路径。

接口的完整URI = 类上的path + 方法上的path。

参数有多种接收方式：

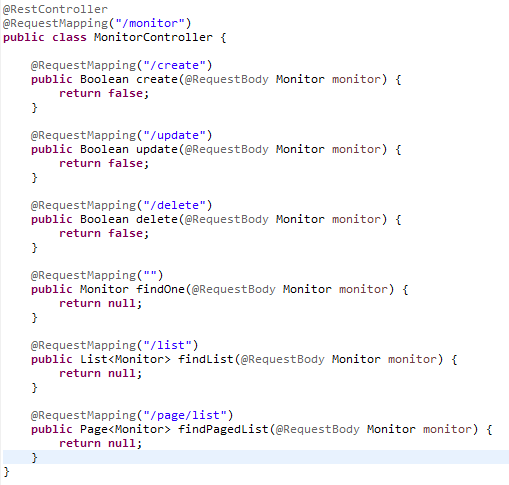
@RequestBody可以从请求体中接收。

@PathVariable可以从路径中接收。

@RequestParam可以从请求参数中接收。

返回值：直接返回可序列化对象。

异常：直接抛出BaseException或其子类。



## 3.4、服务接口

* Service接口定义了需要暴露的服务接口

Service中的接口是服务级的。

仅定义需要暴露的接口，服务内部方法可以不暴露。

请在接口中写清楚注释。



## 3.5、服务实现

* Service实现实现了之前的服务接口定义的服务

实现接口的Override方法不需要写注释。

不需要暴露的方法可以尽量设置为私有。

需要使用@Service注解声明为Bean。

一个接口通常需要包含完整的业务逻辑：

-参数校验

-处理数据

-数据存取

-返回数据

返回值：直接返回可序列化对象。

异常：直接抛出BaseException或其子类。



* Service有时候会需要在不同场景下有多重实现

一个比较常规的做法就是使用注解@ConditionalOnXXXXX系列注解

例如：@ConditionalOnProperty

name为读取的配置，havingValue为配置的值，matchIfMissing为没有改配置时是否认为匹配。

在tsf.monitor.service值为MonitorService或未配置时注入



* 在tsf.monitor.service值为MonitorExtService时注入



## 3.6、Dao数据交互层

* Dao层主要进行数据交互

Dao层需要注入为Bean。

Dao层面向实体，专注于操作实体本身的数据。

不要出现跨微服务的关联查询。

Dao层实现应该根据各业务模块的需求自行选择持久化技术。

JdbcTemplate属于一种相对较为灵活和接近源生的实现，适用范围非常广泛。

所有的参数绑定都应该是变量绑定方式，不应该拼接SQL。

通常数据库层面的异常都应该在Dao层处理完毕后抛出标准业务异常。



## 3.7、异常处理

* 异常类继承标准异常

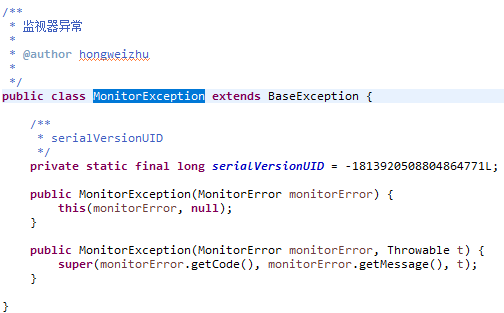
继承标准异常后，我们能够对应用抛出的异常进行统一处理。

统一异常的处理包含：

-后台本地异常处理。

-远程调用异常处理。

-前端异常处理。



## 3.8、工具

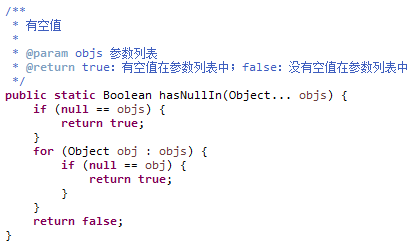
* 工具类处理多次复用的标准计算逻辑

工具类只提供静态方法。

如何界定一个标准计算逻辑是私有方法还是工具类：

-私有方法：仅在小范围内使用，少量甚至没有复用的逻辑。

-工具类：具有通用性，能够在模块/系统中多次复用的逻辑。



## 3.9、代理与远程接口

* 远程接口和代理方法单独放到proxy中

单独管理的proxy层，可以很好地分离远程接口和本地逻辑。

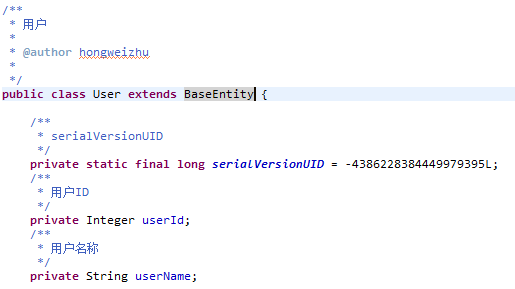
远程接口使用的实体也可以放在proxy中管理。

远程接口使用的实体，重点关注需要的字段而不是接口提供的字段。

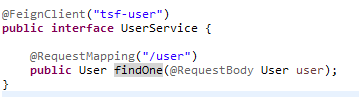
proxy层可以单独编写单元测试进行独立测试。

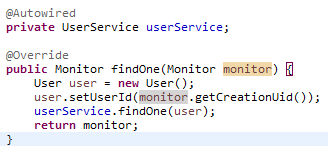
远程访问可以使用Feign Client等实现。

开发时，建议先使用单元测试独立联调远程接口的连通性，这一部分的联调时长是最难以控制的。



* Feign Client是一种远程调用的客户端插件
* Feign Client很好的集成了服务注册发现与SpringMVC的能力，能够通过注解很便捷的实现远程调用。
* 具体远程调用底层还是http，Feign Client通过注解和interface的定义信息收集到必要数据通过动态代理生成默认的实现。本地应用只需要像调用本地服务方法一样注入并使用即可。



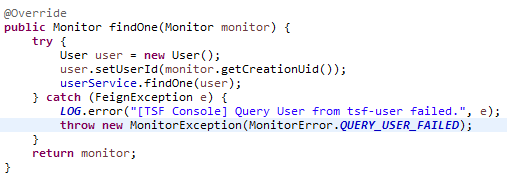


* 使用Feign Client调用远程接口出现异常时

使用catch(FeignException e)来捕获异常。

被捕获的异常会带有异常根因。

如果远程异常抛出的是一个标准的异常，可以使用标准异常转换工具来解析异常。



## 3.10、业务日志

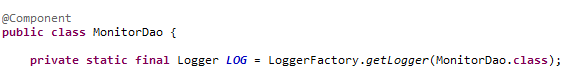
* 业务日志使用常规日志输出

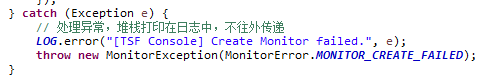
各业务代码在需要输出日志的类中定义Logger。这个定义在后续配置日志输出文件时可以用于拆分不日志到指定文件。

通常在异常处理时如果不将堆栈抛出，会打印到日志中以便定位问题，除非是明确可以忽略的异常。

打印堆栈异常时通常使用LOG.error(String msg, Throwable t)接口打印，此时能够正确打印堆栈信息。

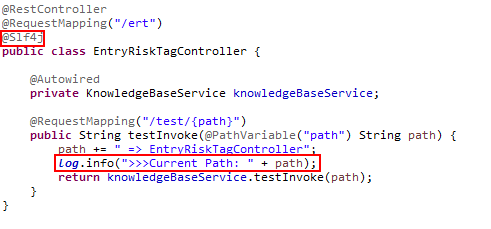
日志提供了格式化接口，使用得当的话可以提升代码可读性。





* 使用Lombok框架时请在业务模块上注解@Slf4j或者@XSlf4j即可。

在业务代码中可以直接使用log.debug/info/error/fault来打印日志。



## 3.11、本地调试

* 本地调试基于本地的标准consul即可

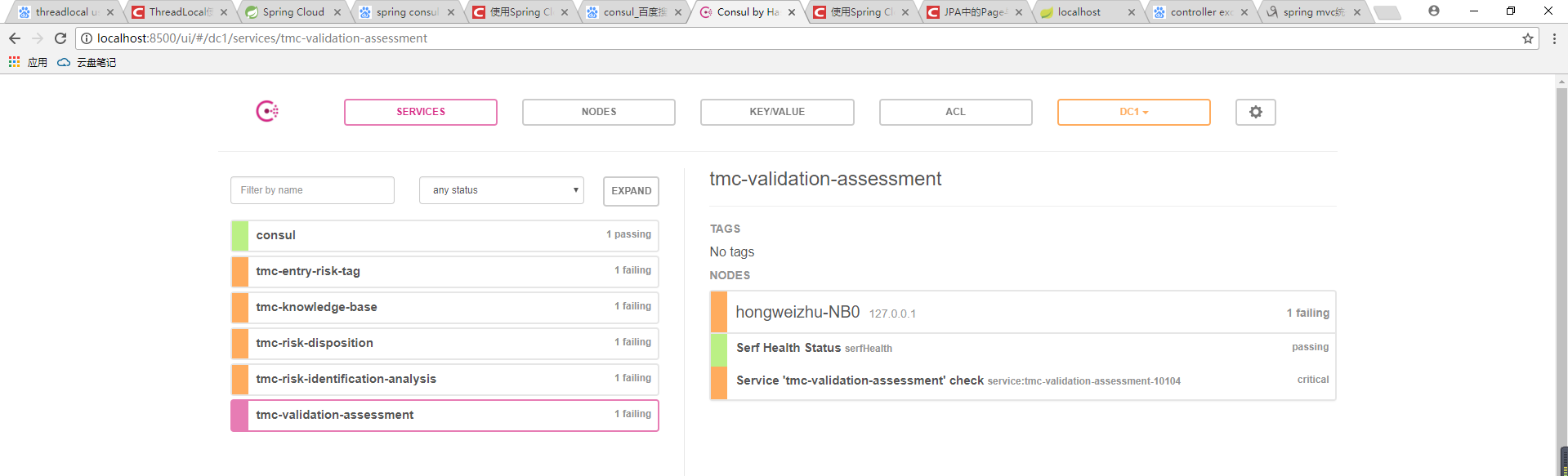
使用标准的consul服务端包即可完成本地调试。

使用consul agent -dev启动本地后，需要调试的微服务将consul的host和port调整为localhost:8500即可。

多服务注册在本地的consul能够正常使用FeignClient进行调用。

本地的consul控制台地址为localhost:8500。

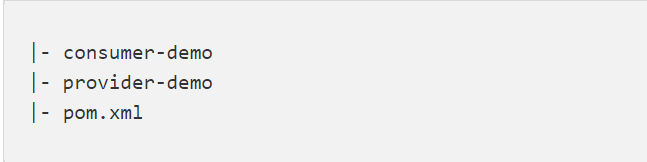
配置内容可以通过本地配置文件application.yml等进行配置。



# Spring Cloud应用接入

## 4.1、服务注册与发现

通过一个简单的示例说明如何实践服务的注册和发现。创建 tsf-demo 工程，文件结构如下：



其中 pom.xml 文件结构如下：



### 4.1.1、创建服务提供者

此服务提供者提供一个简单的 echo 服务，并将自身注册到服务注册中心。

* 创建 provider 工程

创建一个 Spring Cloud 工程，命名为provider-demo。

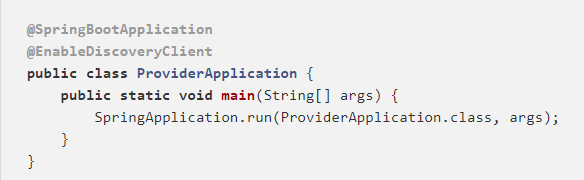
* 修改 pom 依赖

在 pom.xml 中引入需要的依赖内容：



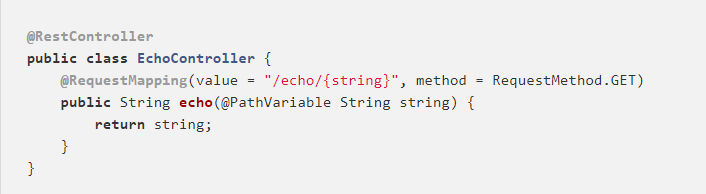
* 开启服务注册发现

添加服务提供端的代码，其中 @EnableDiscoveryClient 注解表明此应用需开启服务注册与发现功能。



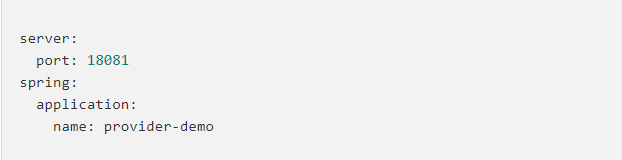
* 提供 echo 服务

创建一个 EchoController，提供简单的echo服务。



* 修改配置

在 resource 目录下的 application.yml 文件中配置应用名与监听端口号。



### 4.1.2、创建服务消费者

本示例中，我们将创建一个服务消费者，消费者通过 RestTemplate、AsyncRestTemplate、FeignClient 这三个客户端去调用服务提供者。

* 创建 consumer 工程

创建一个 Spring Cloud 工程，命名为consumer-demo

* 修改 pom 依赖

在 pom.xml 中引入需要的依赖内容：



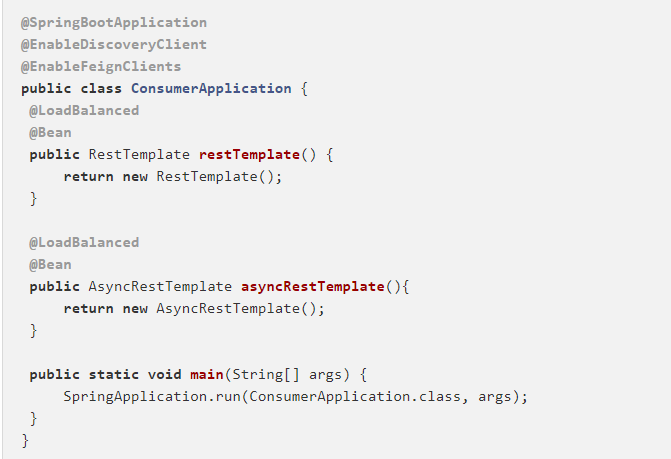
因为此处需要演示 FeignClient 的使用，所以与 provider-demo 相比，pom.xml 文件中的依赖增加了一个 spring-cloud-starter-feign。

* 开启服务注册发现

与服务提供者 provider-demo 相比，除了开启服务与注册外，还需要添加两项配置才能使用 RestTemplate、AsyncRestTemplate、FeignClient 这三个客户端：

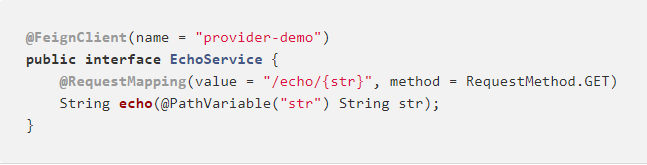
-添加 @LoadBalanced 注解将 RestTemplate 与 AsyncRestTemplate 与服务发现结合。

-使用 @EnableFeignClients 注解激活 FeignClients。



* 设置调用信息

在使用 EchoService 的 FeignClient 之前，还需要完善它的配置。配置服务名以及方法对应的 HTTP 请求，服务名 为 provider-demo工程中配置的服务名 provider-demo，代码如下：



* 创建 Controller

创建一个 Controller 供调用测试。

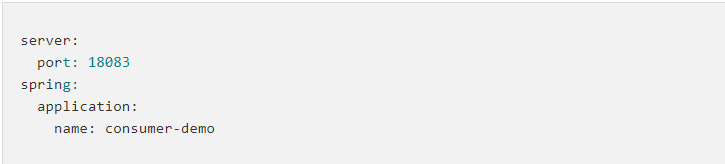
/echo-rest/\* 验证通过 RestTemplate 去调用服务提供者。

/echo-async-rest/\* 验证通过 AsyncRestTemplate 去调用服务提供者。

/echo-feign/\* 验证通过 FeignClient 去调用服务提供者。



* 修改配置



**注意：**

运行在 TSF 平台上的应用无须配置服务注册中心地址，SDK 会通过环境变量自动获取注册中心地址。

### 4.1.3、本地开发调试

* 启动轻量级注册中心

本地开发调试时，需要使用轻量级注册中心，轻量级注册中心包含了 TSF 服务注册发现服务端的轻量版，详请参见下一章。

* 启动应用

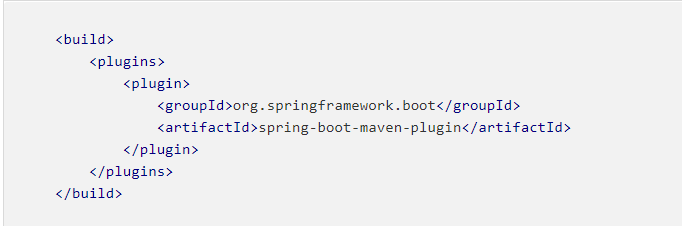
本地启动应用可以通过 IDE 和 FatJar 两种方式。

* IDE 中启动

在IDE中启动，通过 VM options 配置启动参数 -Dtsf\_consul\_ip=127.0.0.1 -Dtsf\_consul\_port=8500 -Dtsf\_application\_id=a -Dtsf\_group\_id=b，通过 main 方法直接启动。其中 IP 和 port 取值为轻量级服务注册中心的地址，使用了分布式配置功能的模块，需要设置-Dtsf\_application\_id=a -Dtsf\_group\_id=b， 取值可为任意值。

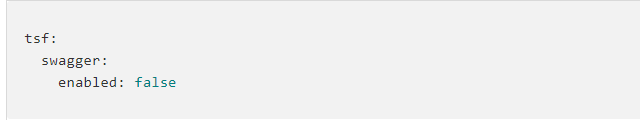
* FatJar 启动

**添加 FatJar 打包方式**



**修改配置文件：**

由于轻量级服务注册中心（原生的 consul）的 metadata 只能支持512个字节，因此您需要关闭 TSF 的 API 上报能力。  
 在配置文件 application.yml 中关闭 swagger：



如果省略了该步骤，在本地运行 Demo 时将会报错，错误信息中包含 Value is too long (limit 512 characters)。

**打包 FatJar 文件：**

添加完插件后，在工程的主目录下，使用 Maven 命令 mvn clean package 进行打包，及可在 target 目录下找到打包好的 FatJar 文件。

**通过 Java 命令启动：**



其中127.0.0.1和8500为轻量级服务注册中心地址，在本地调试时 tsf\_application\_id 和 tsf\_group\_id 可以填任意值。

### 4.1.4、TSF 中部署应用

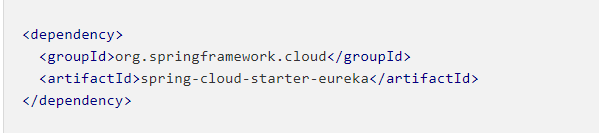
将打包好的 FatJar 程序包上传到 TSF 控制台，进行部署操作，无需关心额外配置。。

### 4.1.5、从 Eureka 迁移

已经接入 Eureka 服务注册与发现的应用，只需要修改 pom.xml 依赖，就可以将服务接入 TSF 服务注册发现中心。

在工程根目录的 pom.xml 中增加 spring-cloud-tsf-dependencies 的 parent。参考上文中的 Demo 工程。

在单个 Spring Cloud 应用的 pom.xml 中，将 spring-cloud-starter-eureka 替换成 spring-cloud-starter-consul-discovery。  
替换前：



替换后：



修改代码中的 Eureka 的相关注解。



**异常检查：**

通过 TSF 平台部署改造后的应用，发现应用无法启动，可以在工程中检查 spring-cloud-starter-consul-discovery 的版本，如果发现版本号中没有 TSF-RELEASE ，说明没有正确依赖 TSF 相关依赖。

## 4.2、轻量级服务注册中心

轻量级服务注册中心给开发者提供在开发、调试、测试的过程中的服务发现、注册和查询功能。

在一个公司内部，通常只需要在一台机器上安装轻量级服务注册中心。具体安装和使用的步骤请参见下文。

### 4.2.1、下载轻量级服务注册中心

推荐您找一台专门的机器启动轻量服务注册中心，比如某台测试机器。根据是否涉及到多个微服务联调测试，分为单机调试和多微服务联调两种场景进行说明。

### 4.2.2、本地使用轻量服务注册中心

如果 不涉及 到多个微服务联调场景，可以通过本地机器启动一个 Consul 作为轻量服务注册中心。单机调试支持 Windows 和 Linux / macOS 操作系统。

确保机器以下的端口是空闲的：8300, 8301, 8302, 8500, 8600。

* 启动轻量级服务注册中心

本地使用轻量服务注册中心场景下支持 Windows 和 Linux / macOS 操作系统。  
进入解压目录，启动服务注册中心。

Windows 操作系统：



Linux / macOS 操作系统：



* 验证服务注册中心启动

查看 8301 和 8500 的端口是否被监听；

通过浏览器查看服务注册中心页面（http://127.0.0.1:8500）。

### 4.2.3、多微服务联调环境的轻量服务注册中心

在多个微服务联调场景下，可以找一台可以被微服务访问的机器来部署轻量服务注册中心。目前本场景下仅支持 Linux 系统的 Consul，[Linux 系统 Consul 下载地址](https://releases.hashicorp.com/consul/1.0.0/)。

* 启动轻量级服务注册中心

将 consul 二进制文件放到任意一个目录，比如 /data/。

将 start.sh 也放到同一个目录。下载脚本 [start.sh](https://main.qcloudimg.com/raw/6d24d47c93b173f962bd874ea598c68b.zip)。

执行如下命令：



其中 local\_ip 填写本机 IP。比如，linux 机器上的 IP 为 192.168.1.10，那么执行的命令是：./start.sh 192.168.1.10。

* 验证服务注册中心启动



如果有正常输出，代表 Consul 已经正常启动。

## 4.3、分布式配置

### 4.3.1、依赖项

添加 pom.xml 依赖：



### 4.3.2、代码中引用配置

用户可通过两种方式更新代码中的配置信息：使用配置类@ConfigurationProperties和@Value 注解。@Value比较适用于配置比较少的场景，而 @ConfigurationProperties 则更适用于有较多配置的情况。

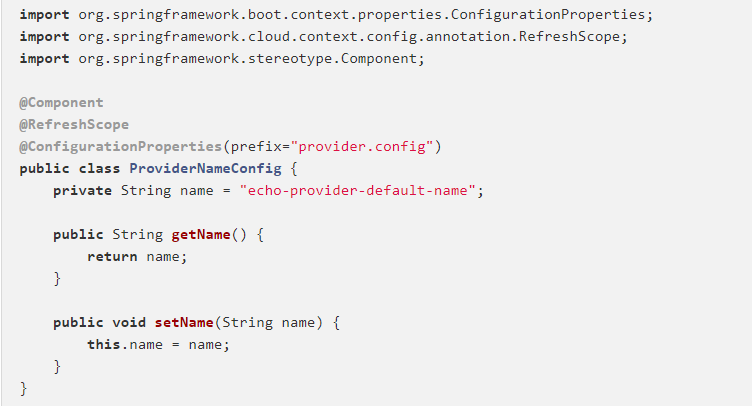
用户也可以动态更新应用配置文件（如 application.yml）中的配置，如动态更改 redis 的地址或者鉴权功能开关等。

* 使用配置类 @ConfigurationProperties

在provider-demo的 ProviderNameConfig 类中，有一个字符串类型的变量 name。其中：

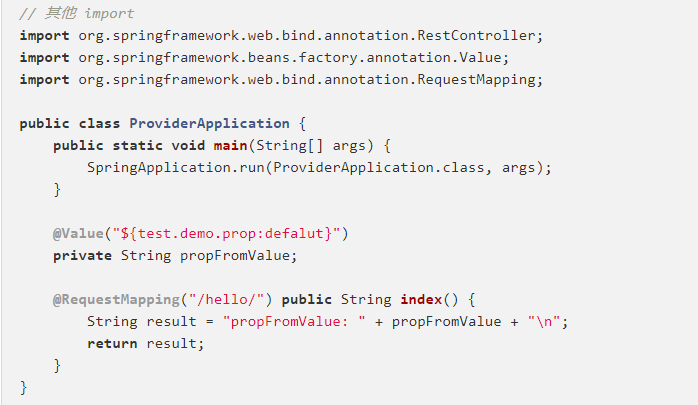
-使用@ConfigurationProperties注解来标明这个类是一个配置类。

-使用@RefreshScope注解开启 refresh 机制。



* 使用 @Value 注解

在启动类 ProviderApplication 中，使用 @Value 注解来标识一个配置变量。下面的示例中 ${test.demo.prop:default} 中 test.demo.prop 是在动态配置下发中使用的 key，value 默认是 default 。



* 配置更新触发回调

配置更新触发回调功能允许程序在不重启的情况下动态修改业务逻辑。当配置更新时，触发配置回调方法的调用。配置更新触发回调功能的使用场景包括：

-程序使用一个防刷开关配置，当开关为启用状态时，启动防刷逻辑，当开关为关闭状态时，停用防刷逻辑。

-程序使用一个 ReidsConfig 的动态配置类，包含 redis 的 host 和 port，当更新这个配置时，更新 Redis 实例。

-配置更新后发出通知消息，通知本地或者远程的其他模块执行变更逻辑

TSF 分布式配置支持使用 @ConfigChangeListener 注解且实现 ConfigChangeCallback 接口。在 SimpleConfigurationListener类中，设置 callback回调方法。支持场景包括：

-支持按照配置项前缀模糊匹配方式。

-支持配置项精确匹配方式。

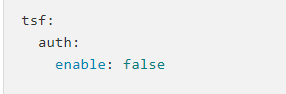
-支持回调方法同步 sync 或者异步 async 方式执行。

* 更新配置文件中的配置

在 provider-demo 中， bootstrap.yml 文件有一行鉴权开关的配置项：



用户可以通过控制台创建配置，将鉴权开关关闭：



### 4.3.3、通过 TSF 平台下发动态配置

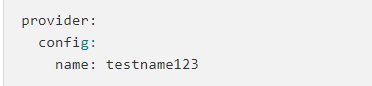
用户可以通过 TSF 平台来下发动态配置，前提条件：

- 已经在 TSF 平台上部署了 provider-demo 和 consumer-demo 应用。

-部署 provider-demo 的部署组的日志配置项的日志路径中包含了 /tsf-demo-logs/provider-demo/root.log，以确保打印的日志被采集后，可以通过控制台查看应用的日志。

关于如何通过控制台创建及下发更新的配置，请参考 [应用配置](https://cloud.tencent.com/document/product/649/15539)。

如果希望修改 ProviderNameConfig 类 中的 providerName 的值，创建配置时，配置内容填写：



将配置发布到已部署 provider-demo 的部署组上，检查打印的日志中是否 name 的值已更新。如果已更新，说明更新的配置生效。



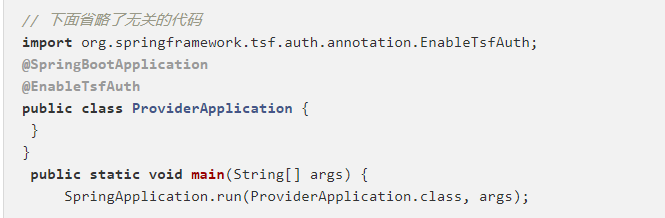
## 4.4、服务鉴权

这里演示如何快速实践服务鉴权功能。假如现在有两个微服务 provider 和 consumer，想实现 consumer 调用 provider 时，provider 对请求做鉴权。

向 provider 和 consumer 工程都添加依赖。在 pom.xml 中添加以下代码：



向 Application 类中添加注解 @EnableTsfAuth：



此时您已经对 provider 微服务开启了鉴权功能，任何到达 provider 的请求都会被鉴权，鉴权不通过时会返回 HTTP 403 Forbidden。

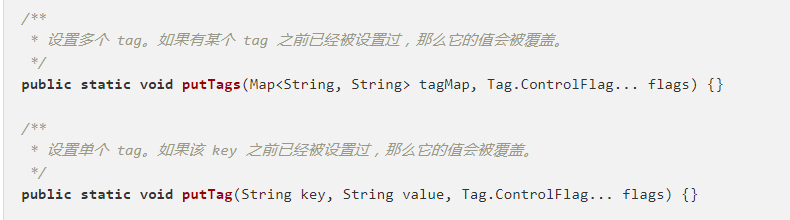
TSF 提供了两种类型的鉴权能力，一种根据调用方服务名鉴权，一种根据调用方设置的 tag 鉴权。在控制台上可以配置相应的规则。如果在控制台上对 provider 启用了鉴权功能，并且配置了至少一条规则，那么调用 provider 的微服务（如 comsumer）也需要引入 spring-cloud-tsf-auth 的依赖并且加上 @EnableTsfAuth 注解，否则到 provider 的请求会被返回 HTTP 403 Forbidden。

如果请求双方想使用基本 tag 的鉴权规则，那么：

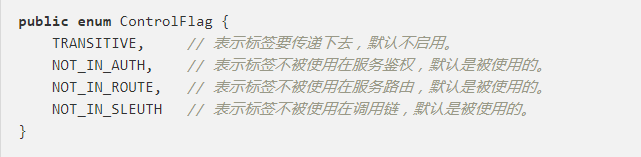
- 对于 provider 而言，需要在控制台上设置 tag 鉴权规则。

- 对于 consumer 而言，需要在业务代码中设置 tag 的内容。

在 consumer 中设置 tag ，使用 org.springframework.tsf.core 包中的 TsfContext 类。设置 Tag 的方法签名如下：



其中 flags 决定 tag 的使用场景，如果您没有特殊需要，不传即可：



TSF 提供的 Demo

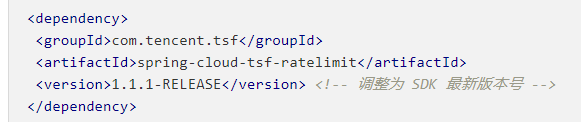
consumer-demo/src/main/java/com/tsf/demo/consumer/Controller.java 中提供了一个设置 tag 的例子：



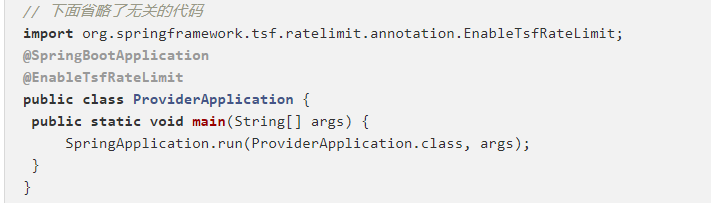
## 4.5、服务限流

如果用户要对某个服务开启限流的能力，即对调用它的请求做限流，可以按下面的步骤打开限流开关。

向工程中添加依赖。在 pom.xml 中添加以下代码：



向 Application 类中添加注解 @EnableTsfRateLimit：



此时您已经对服务开启了限流功能，任何到达的请求都会被限流模块处理。如果该服务上的配额已经消耗完，会对请求返回 HTTP 429 Too Many Requests；否则会正常放行。

## 4.6、参数传递

### 4.6.1、元数据类别

TSF 提供两种类型的元数据供开发者在代码中进行设置：

|  |  |
| --- | --- |
| 类型 | 特点 |
| 标签信息（Tags) | 可设置传递性，仅支持 key-value 数据结构，key 和 value 均为字符串类型。 |
| 辅助信息  （CustomMetadata) | 仅供展示，不支持筛选，不具备传递性。 |

场景说明：

标签信息 ：用于信息分类，使用场景包括：

- 服务鉴权：被调方通过标签来决定是否提供服务。

- 服务路由：通过标签来判断应该访问什么服务，可用于实现金丝雀发布等。

- 调用链：可用于调用链的筛选和附带业务信息。

辅助信息 （CustomMetadata）：仅供展示，不支持筛选，不具备传递性。

### 4.6.2、元数据的传递性

以调用关系 A => B => C，说明传递性的概念：

- 可传递（Transitive）：需要传递的标签，在整条链路都传递，即用户在 A 设置的标签，会传递到 B 再传递到 C。

- 不可传递：不需要传递的标签，即用户在 A 设置的标签，会传递到 B，但是不会传递到 C。

标签信息 支持允许用户设置是否支持传递，辅助信息 不支持传递。

不同的标签可以设置不同传递性，比如一些业务场景：

- userid 标签是需要传递的：

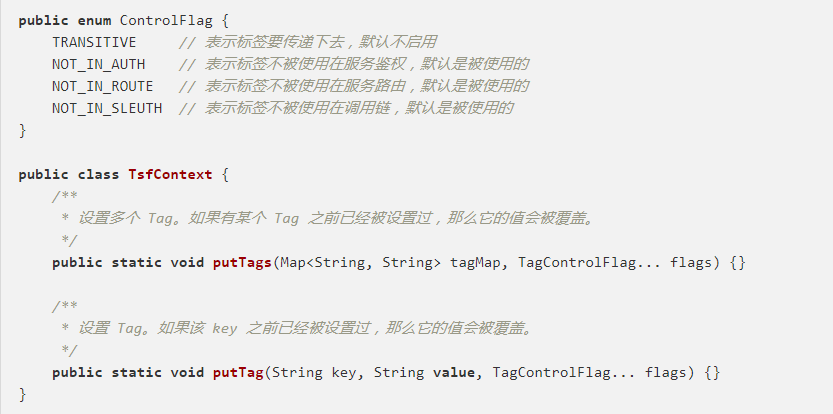
可以作为整条调用链上的服务的 上下文信息。

可以实现按 uin 区分的服务路由，比如 A、B、C 三个服务同时做滚动发布，那么可以让一批 uin 都走新版本的 A、B、C 服务，其他用户走老版本。

- level=高级会员 这种标签，很可能就不需要在调用间传递下去。

### 4.6.3、使用元数据

**接口**



**场景 1：设置鉴权 Tag**

TSF 提供的 Demo

consumer-demo/src/main/java/com/tsf/demo/consumer/Controller.java 中设置了键为 user，请求参数作为值的 Tag。



**场景 2：设置调用链 Tag**

设置 user=12345678 的标签，用户可以在控制台 调用链查询界面 通过标签 user=12345678 来检索调用链。



**Tag 的长度限制**

用户传递到下流的 Tag （包含从上流带过来的有传递性的 Tag），数量上限为 16 个。Key 的长度上限为 UTF-8 编码后 32 字节，value 的长度上限为 UTF-8 编码后 128 字节。

## 4.7、调用链

为了节约用户的开发成本和提升使用效率，TSF 提供了Spring Cloud 全链路跟踪的组件。用户只需要在代码中配置组件依赖，即可直接使用 TSF 的全链路跟踪功能，无需关心日志采集、分析、存储等过程。 用户仅需要安装了对应的依赖包及添加依赖项即可，无须其他配置。

* 添加依赖

添加 pom.xml 依赖：



* 标签与自定义元数据

调用链支持用户在代码中设置标签（Tag） 和自定义元数据（CustomMetada），分别用于调用链的筛选和附带业务信息。

接口定义：

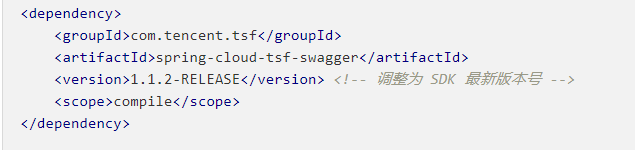


## 4.8、API注册

TSF 框架在微服务注册时，会自动收集并注册微服务提供的 API 接口，用户可通过 TSF 控制台实时掌握当前微服务提供的 API 情况。

* 添加依赖

在 pom.xml 中添加以下代码：



添加依赖包后，TSF API 注册功能即生效。

* 配置选项

API 注册功能，基于 Swagger 原生规范来实现。提供多个配置来适配 Swagger 不同配置应用场景，可用配置如下表所示：

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 配置项 | 类型 | 必填 | 默认值 | 说明 |
| tsf.swagger.enabled | Boolean | 否 | true | 是否开启 TSF API 注册功能 |
| tsf.swagger.basePackage | String | 否 | ApplicationMainClass 所在包路径 | 注册 API 的扫描包路径。  推荐将 ApplicationMainClass 写在外层 Package |
| tsf.swagger.excludePath | String | 否 | （空） | 排除扫描的包路径 |
| tsf.swagger.group | String | 否 | default | swagger docket 分组 |

# 5、TBase数据库概述

## 5.1、TBase设计目标

1、提供全局实时分布式事务能力

1. 弹性扩缩容和负载再平衡能力
2. 冷热数据分离，数据倾斜治理能力
3. 三权分立管理能力

5、支持两地三中心部署，支持数据同步，半同步，异步复制能力

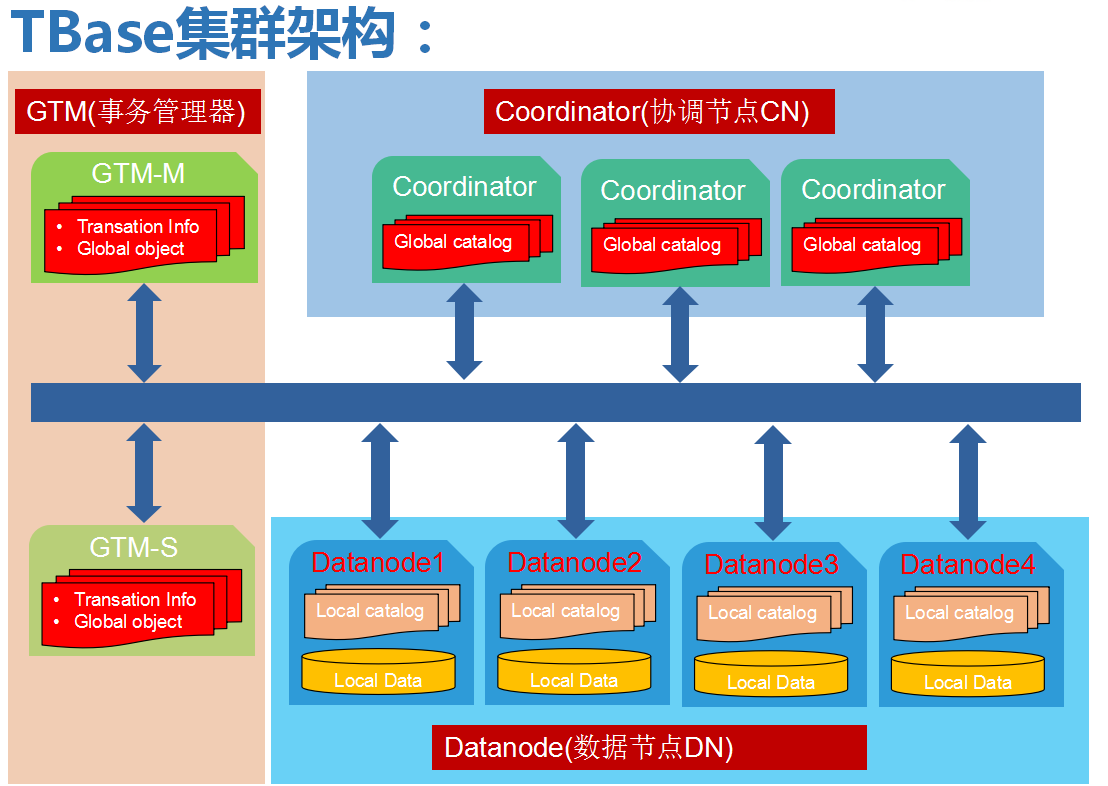
6、可以提供多个主节点同时处理来自应用端发出的 SQL 语句，这些节点称为coordinator（协调节点）。

7、任何一个协调节点都有全局数据库视图，也就是说当任一台协调节点接收更新数据并提交后，在另外的主节点可以迅速地看到变化的数据视图。

8、表以distributed by shard 或者 replicated 方式分布式存储，并且这对应用来说是透明的。

9、不需要架构师在应用层再构建sharding应用组件，大大的降低应用程序的复杂度，提高开发的效率。

## 5.2、TBase集群构架



* Coordinator

Coordinator（简称 CN） 是协调节点， 是数据库服务的对外入口，负责数据的分发和查询规划，多个节点位置对等。业务请求发送给 CN 后，无需关心数据计算和存储的细节， 由 CN 统一返回执行结果。 CN 上只存储系统的元数据，并不存储实际的业务数据， 可以配合支持业务接入增长动态增加。

* Datanode

Datanode（简称 DN） 是数据节点，执行协调节点分发的执行请求，实际存储业务数据。 各个 DN 可以部署在不同的物理机上， 也支持同物理机部署多个 DN 节点， 注意互为主备 DN 不建议部署在同物理主机上。 DN 节点存储空间彼此之间独立、 隔离，是标准的 share nothing 存储拓扑结构。另外TBase-V2与V1最大的不同地方是DN与DN之间可以通信，互相交换数据。

* GlobalTransactionManager

GlobalTransactionManager（简称 GTM）， 是全局事务管理器，负责全局事务管理。GTM 上不存储业务数据。

## 5.3、TBase安装

### 5.3.1、运行环境硬件要求

|  |  |  |  |
| --- | --- | --- | --- |
| 用途（体验环境） | 配置 | 数量（台） | 部署组件 |
| 管理节点、协调节点、数据节点、集群管理平台、可视化开发工具 | 8核8G 200G | 1 | OssCenterMaster|ConfdbMaster|TStudio |
| 管理高可用节点、协调节点、数据节点 | 8核8G 200G | 1 | OssCenterSlave|ConfdbSlave|Alarm |

### 5.3.2、操作系统要求

|  |  |  |
| --- | --- | --- |
| 支持系统版本 | 内核 | glibc版本 |
| CentOS 7.3 | Linux version 3.10.0 | glibc-2.17 |

部署请使用最小化安装方式

Centos7.3minimal发行版本，下载地址

<http://mirror.neu.edu.cn/centos/7/isos/x86_64/CentOS-7-x86_64-Minimal-1611.iso>

### 5.3.3、如何获取TBase安装包

请联系腾讯商务代表，或在TBase官网<http://tbase.qq.com/download/>获取TBase安装包和许可证文件。

### 5.3.4、安装过程

参考TBase部署文档

## 5.4 、配置tbase的客户端使用环境

Tbase-v2是基于多租户设计，安装后所有机器都不会配置tbase的客户端默认使用环境，所以使用psql

需要配置其环境变量，如下

export PGXZ\_HOME=/usr/local/install/tbase\_pgxz

export PATH=$PGXZ\_HOME/bin:$PATH

export LD\_LIBRARY\_PATH=$PGXZ\_HOME/lib:${LD\_LIBRARY\_PATH}

PGUSER=tbase

PGHOST=127.0.0.1

PGDATABASE=postgres

PGPORT=11000

export PGHOST PGUSER PGDATABASE PGPORT

这样就能使用psql客户端了，如果机器上面不准备安装任何节点，则你可以直接将上面的环境变量配置到

~/.bashrc 中

## 5.5、使用TBase对开发者的基本要求

通常情况下，只要开发者有过Postgresql，mysql，oracle，sql server，db2，sybase...等关系型数据库使用经验，使用起TBase来就没有什么困难，最佳的体验者为Postgresql应用开发人员。如果你没有这方面的经常，建议你先购买一本sql语言的书籍先进行阅读。

## 5.6 、TBase每个cn/dn节点的系统限制

|  |  |
| --- | --- |
| 项目 | 值 |
| 单个数据库最大容量 | 无限制 |
| 单个表的最大容量 | 32TB |
| 一行记录最大容量 | 1.6TB |
| 一个字段的最大容量 | 1GB |
| 一个表里最大记录行数 | 无限制 |
| 一个表里最大列数 | 250-1600，与列类型有关 |
| 一个表里最大索引个数 | 无限制 |
| 字段名长度 | 63 |

# 6、创建TBase默认的分布式使用环境

## 6.1、创建数据表默认的default group

TBase做为分布式数据库系统，使用前我们必需配置数据表默认分布的数据节点（DN），下面演示如何创建一个default group

--切换为tbase用户

su tbase

--连接数据库--注意是连接到cn节点(后面没特别说明，所有数据库操作都是连接到cn节点)

psql -h 172.16.0.29 -p 15432 -U tbase -d postgres

psql (PostgreSQL 10 (TBase 2.01))

Type "help" for help.

--查询当前什么数据节点（DN）,这些DN节点就是上面初始化集群时建立的

postgres=# select \* from pgxc\_node where node\_type='D';

node\_name | node\_type | node\_port | node\_host | nodeis\_primary | nodeis\_preferred | node\_id | node\_cluster\_name

-----------+-----------+-----------+-------------+----------------+------------------+-------------+-------------------

dn\_0 | D | 23001 | 172.16.0.29 | f | f | 1485981022 | tbase\_cluster

dn\_1 | D | 23002 | 172.16.0.47| f | f | -1300059100 | tbase\_cluster

(2 rows)

--建立数据表默认使用的group

postgres=# create default node group default\_group with(dn\_0, dn\_2);

## 6.2、为default group创建shardmap

配置完成数据表默认使用的DN节点后，我们接下来需要配置记录的分区方案，shardmap就是TBase各个哈希值与DN的对照表，下面演示如何创建一个shardmap给default\_group 使用

postgres=# create sharding group to group default\_group;

postgres=# clean sharding;

至此我们就可以像单机一样使用TBase集群了。

## 6.3、更多group的使用方法

### 6.3.1、创建扩展group

* 建立group

postgres=# create node group ext\_group with(dn03,dn04);

CREATE NODE GROUP

* 为group创建shard

postgres=# create extension sharding group to group ext\_group;

CREATE SHARDING GROUP

postgres=# clean sharding;

CLEAN SHARDING

postgres=#

### 6.3.2、删除group

postgres=# drop sharding in group ext\_group;

DROP SHARDING GROUP

postgres=# drop node group ext\_group ;

DROP NODE GROUP

### 6.3.3、查询集群中group的数量

postgres=# select \* from pgxc\_group;

group\_name | default\_group | group\_members

---------------+---------------+---------------

default\_group | 1 | 16386 16387

ext\_group | 0 | 16388 16389

(2 rows)

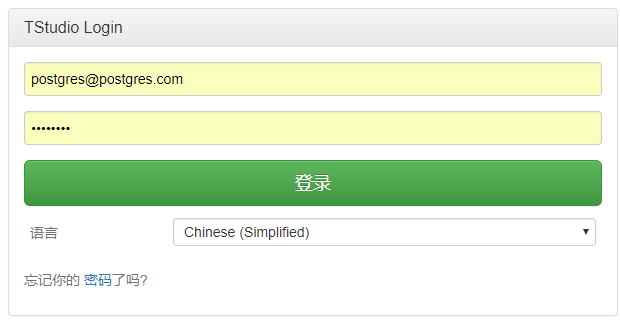
# 7、开发者工具介绍和使用

## 7.1、可视化开发工具TStudio

### 7.1.1、如何打开TStudio

打开浏览器（目前只支持chrome和firefox两种浏览器），输入下面运行Tbase机器的网址进入TBase TStudio数据对象可视化管理平台，如

[http://172.16.0.29:5050/](http://192.168.32.131:5050/)



登录用户名：postgres@postgres.com 登录密码：postgres

### 7.1.2、TStudio主界面说明



### 7.1.3、如何添加要管理的节点

点击上图的“添加新的服务器”在弹出的操作窗口录入相应的信息后按“保存”即可添加一个管理节点，如下图所示。



### 7.1.4、如何创建数据表、索引

#### 7.1.4.1、创建数据表

1. 、点击菜单项目 Servers－〉TBase-sales-10－〉postgres
2. 、在“表”菜单项上右击鼠标，在弹出菜单选择 创建－〉表



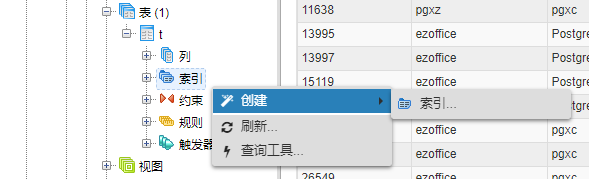
3、）系统弹出创建表的对话窗口，如下所示



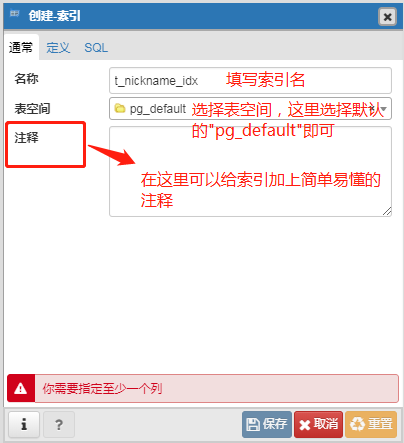
1. ）编辑完第一页和第二页框信息后按“保存”按钮即可建立表名为“t”的数据表

#### 7.1.4.2、给表建立索引

1. 、点击刚才建立的数据表t
2. 、在“索引”菜单项上右击鼠标，在弹出菜单选择 创建－〉索引



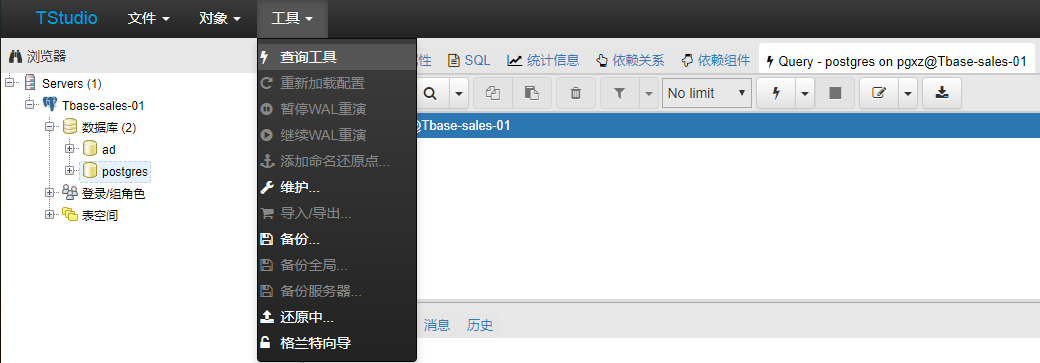
3、）系统弹出创建索引的对话窗口，如下所示



1. 、编辑完第一页和第二页框信息后按“保存”按钮即可为表“t”建立索引“t\_nickname\_idx”

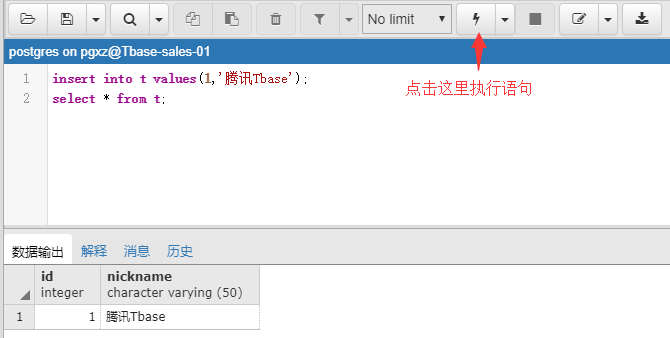
### 7.1.5、运行手工编写脚本

1）、选择左边菜单项目“postgres”库



2）、点击顶部菜单 工具－〉查询工具 即可在右边工作区域弹出查询窗口

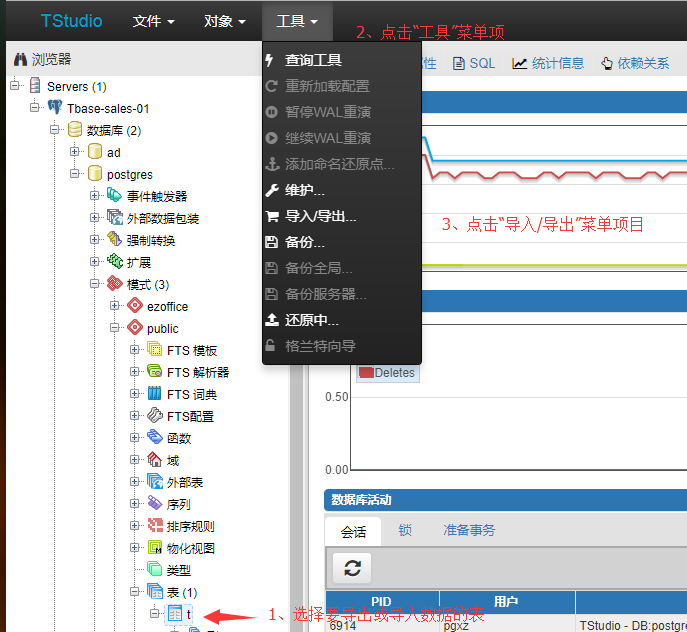
3）、录入要执行的sql语句后按窗口上面的“执行”图标执行sql脚本，如下图所示



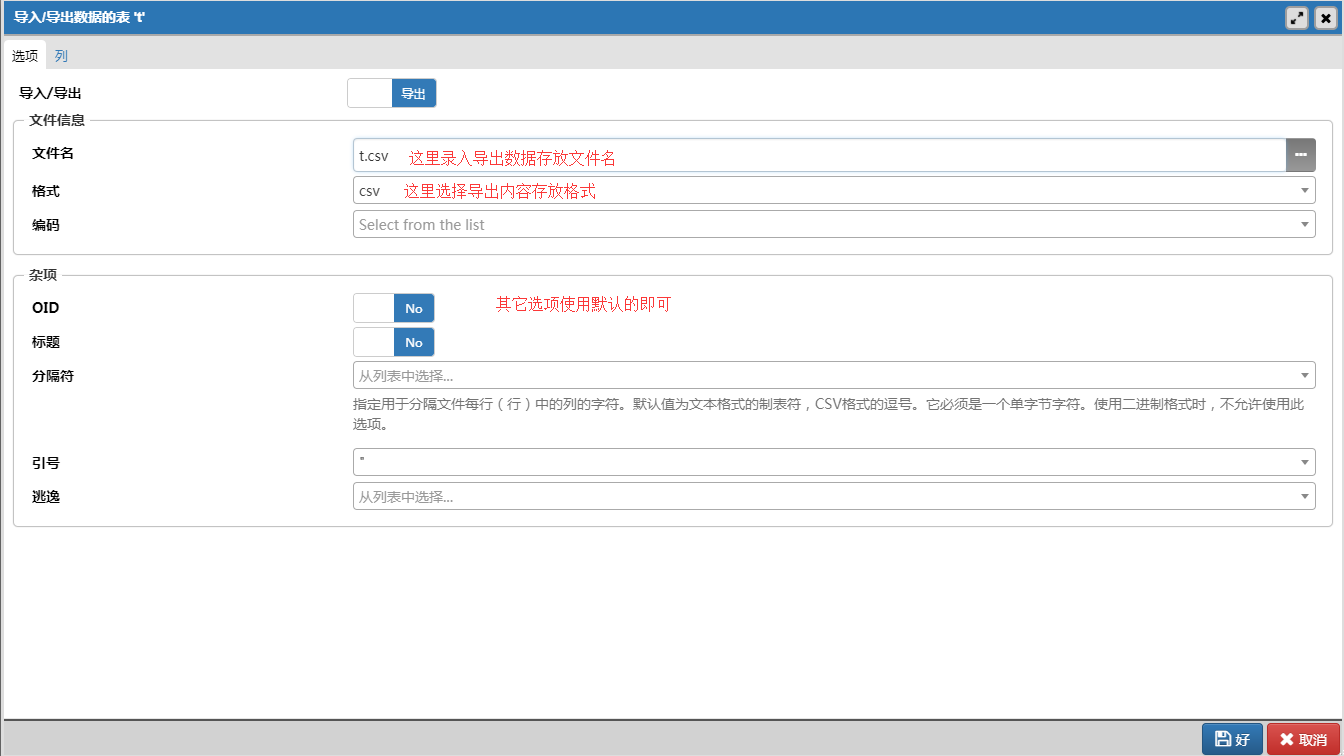
上面语句先插入一条记录，再将插入的记录找出来

### 7.1.6、表数据导入导出数据

1）、选择左边菜单项目要导出或导入数据的表



2）、点击顶部菜单 工具－〉导入/导出 即可弹出操作对话窗口，如下图所示



数据存放于目录/usr/local/install/tstudio/storage/postgres/目录下面

[root@VM\_0\_29\_centos tbase\_mgr]# cd /usr/local/install/tstudio/storage/postgres

[root@VM\_0\_29\_centos tbase\_mgr]# cat t.csv

1,腾讯TBase

[root@VM\_0\_29\_centos tbase\_mgr]#

## 7.2、shell交互客户端psql

### 7.2.1、连接到一个数据库

* 使用参数连接

[tbase@VM\_0\_29\_centos root]$ psql -h 172.16.0.29 -p 15432 -U tbase -d postgres

psql (PostgreSQL 10 (TBase 2.01))

Type "help" for help.

postgres=#

* 使用conninfo字符串或者一个URI

[tbase@VM\_0\_29\_centos root]$ psql postgresql://tbase@172.16.0.29:15432/postgres

psql (PostgreSQL 10 (TBase 2.01))

Type "help" for help.

postgres=#

* 配置证书连接

[tbase@VM\_0\_29\_centos ~]$ psql 'host=172.16.0.29 port=15432 dbname=postgres user=tbase sslmode=verify-ca sslcert=postgresql.crt sslkey=postgresql.key sslrootcert=root.crt'

Password:

psql (PostgreSQL 10 (TBase 2.01))

SSL connection (cipher: DHE-RSA-AES256-SHA, bits: 256)

Type "help" for help.

postgres=> \q

* 配置环境变量后快捷连接

[tbase@VM\_0\_29\_centos root]$ PGUSER=tbase

[tbase@VM\_0\_29\_centos root]$ PGHOST=127.0.0.1

[tbase@VM\_0\_29\_centos root]$ PGDATABASE=postgres

[tbase@VM\_0\_29\_centos root]$ PGPORT=15432

[tbase@VM\_0\_29\_centos root]$ export PGHOST PGUSER PGDATABASE PGPORT

[tbase@VM\_0\_29\_centos root]$ psql

psql (PostgreSQL 10 (TBase 2.01))

Type "help" for help.

postgres=#

也可以配置在用户环境变量中

[tbase@VM\_0\_29\_centos root]$ cat ~/.bashrc

# .bashrc

PGUSER=tbase

PGHOST=127.0.0.1

PGDATABASE=postgres

PGPORT=15432

export PGHOST PGUSER PGDATABASE PGPORT

### 7.2.2、建立一个新连接

* 连接到另外一个库（也可以是当前库）

postgres=# select pg\_backend\_pid();

pg\_backend\_pid

----------------

2408

(1 row)

postgres=# \c

You are now connected to database "postgres" as user "tbase".

postgres=# select pg\_backend\_pid();

pg\_backend\_pid

----------------

2426

(1 row)

postgres=# \c template1

You are now connected to database "template1" as user "tbase".

template1=#

* 连接到外一台服务

postgres=# \c postgres tbase 172.16.0.47 15432

You are now connected to database "postgres" as user "tbase" on host "172.16.0.47" at port "15432".

### 7.2.3、显示和设置该连接当前运行参数

* 显示当前连接的运行参数

postgres=# SELECT CURRENT\_USER;

current\_user

--------------

tbase

(1 row)

postgres=# show search\_path ;

search\_path

----------------

"$user",public

(1 row)

postgres=# show work\_mem ;

work\_mem

----------

4MB

(1 row)

* 设置当前连接的运行参数

postgres=# set search\_path = "$user",public,pg\_catalog;

SET

postgres=# set work\_mem = '8MB';

SET

* 打开和关闭显示每个sql语句执行的时间

postgres=# \timing on

Timing is on.

postgres=# select count(1) from tbase;

count

-------

10000

(1 row)

Time: 5.139 ms

postgres=# \timing off

Timing is off.

postgres=# select count(1) from tbase;

count

-------

10000

(1 row)

* 打开和关闭显示每个快捷操作符实际运行的sql语句

postgres=# \set ECHO\_HIDDEN on

postgres=# \dt

\*\*\*\*\*\*\*\*\* QUERY \*\*\*\*\*\*\*\*\*\*

SELECT n.nspname as "Schema",

c.relname as "Name",

CASE c.relkind WHEN 'r' THEN 'table' WHEN 'v' THEN 'view' WHEN 'm' THEN 'materialized view' WHEN 'i' THEN 'index' WHEN 'S' THEN 'sequence' WHEN 's' THEN 'special' WHEN 'f' THEN 'foreign table' END as "Type",

pg\_catalog.pg\_get\_userbyid(c.relowner) as "Owner"

FROM pg\_catalog.pg\_class c

LEFT JOIN pg\_catalog.pg\_namespace n ON n.oid = c.relnamespace

WHERE c.relkind IN ('r','')

AND n.nspname <> 'pg\_catalog'

AND n.nspname <> 'information\_schema'

AND n.nspname !~ '^pg\_toast'

AND pg\_catalog.pg\_table\_is\_visible(c.oid)

ORDER BY 1,2;

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

List of relations

Schema | Name | Type | Owner

--------+--------------+-------+-------

public | t\_time\_range | table | tbase

public | tbase | table | tbase

(2 rows)

postgres=# \set ECHO\_HIDDEN off

postgres=# \dt

List of relations

Schema | Name | Type | Owner

--------+--------------+-------+-------

public | t\_time\_range | table | tbase

public | tbase | table | tbase

(2 rows)

* 配置输出结果为HTML格式

postgres=# \pset format html

Output format is html.

postgres=# \d tbase

<table border="1">

<caption>Table &quot;public.tbase&quot;</caption>

<tr>

<th align="center">Column</th>

<th align="center">Type</th>

<th align="center">Modifiers</th>

</tr>

<tr valign="top">

<td align="left">id</td>

<td align="left">integer</td>

<td align="left">&nbsp; </td>

</tr>

<tr valign="top">

<td align="left">mc</td>

<td align="left">text</td>

<td align="left">&nbsp; </td>

</tr>

</table>

恢复为对齐模式

postgres=# \pset format aligned

Output format is aligned.

postgres=# \d tbase

Table "public.tbase"

Column | Type | Modifiers

--------+---------+-----------

id | integer |

mc | text |

* 配置行列显示格式

postgres=# \x on

Expanded display is on.

postgres=# select \* from tbase where id=1;

-[ RECORD 1 ]

id | 1

mc | 1

-[ RECORD 2 ]

id | 1

mc | 2

-[ RECORD 3 ]

id | 1

mc | 2

postgres=# \x off

Expanded display is off.

postgres=# select \* from tbase where id=1;

id | mc

----+----

1 | 1

1 | 2

1 | 2

(3 rows)

* 显示和配置客户端编码

postgres=# \encoding

UTF8

配置客户端编码为SQL\_ASCII

postgres=# \encoding sql\_ascii

postgres=# \encoding

SQL\_ASCII

### 7.2.4、退出连接

postgres=# \q

### 7.2.5、psql执行一个sql命令

* 显示标题

[tbase@VM\_0\_29\_centos root]$ psql -h 172.16.0.29 -p 15432 -U tbase -d postgres -c "select count(1) from pg\_class"

count

-------

317

(1 row)

* 不显示标题

[tbase@VM\_0\_29\_centos root]$ psql -h 172.16.0.29 -p 15432 -U tbase -d postgres -t -c "select count(1) from pg\_class"

317

### 7.2.6、psql执行一个sql文件中所有命令

* 在外部执行

[tbase@VM\_0\_29\_centos ~]$ cat /data/tbase/tbase.sql

set search\_path = public;

insert into tbase values(1,2);

select count(1) from tbase;

[tbase@VM\_0\_29\_centos ~]$ psql -h 172.16.0.29 -p 15432 -U tbase -d postgres -f /data/tbase/tbase.sql

SET

INSERT 0 1

count

-------

10001

(1 row)

* 在内部执行

[tbase@VM\_0\_29\_centos ~]$ psql -h 172.16.0.29 -p 15432 -U tbase -d postgres

psql (PostgreSQL 10 (TBase 2.01))

Type "help" for help.

postgres=# \i /data/tbase/tbase.sql

SET

INSERT 0 1

count

-------

10002

(1 row)

### 7.2.7、调用编辑器编写sql脚本

postgres=# \e

"/tmp/psql.edit.5532.sql" 2L, 35C written

编辑完成后保证退出执行

### 7.2.8、调用外部命令

postgres=# \! cat /data/tbase/tbase.sql

set search\_path = public;

insert into tbase values(1,2);

select count(1) from tbase;

postgres=# \! ls

data1 data2 data3 install

postgres=# \! ls -l

total 0

drwxrwxr-x 3 tbase tbase 28 Sep 18 11:05 data1

drwxrwxr-x 3 tbase tbase 28 Sep 18 11:05 data2

drwxrwxr-x 3 tbase tbase 28 Sep 18 11:17 data3

drwxr-xr-x 3 tbase tbase 23 Sep 18 10:55 install

postgres=#

### 7.2.9、将执行的结果保存到文件

postgres=# \o /data/tbase/log.txt

postgres=# select \* from tbase where id=1;

postgres=# \! cat /data/tbase/log.txt

id | mc

----+----

1 | 1

1 | 2

1 | 2

(3 rows)

postgres=# \o

postgres=# select \* from tbase where id=1;

id | mc

----+----

1 | 1

1 | 2

1 | 2

(3 rows)

### 7.2.10、改变当前的工作目录

postgres=# \! ls

backup data install java log.txt pgbench tbase tbaseconf shell tbase.sql

postgres=# \cd /data

postgres=# \! ls

package pgsql tbase tbaseConfdb

postgres=#

### 7.2.11、插件管理

* 查看当前库加载了那些插件

postgres=# \dx

List of installed extensions

Name | Version | Schema | Description

--------------------+---------+------------+-----------------------------------------------------------

pg\_stat\_statements | 1.1 | public | track execution statistics of all SQL statements executed

plpgsql | 1.0 | pg\_catalog | PL/pgSQL procedural language

(2 rows)

* 给当前库加载插件

postgres=# create extension pg\_stat\_statements;

CREATE EXTENSION

* 删除当前库某个插件

postgres=# drop extension pg\_stat\_statements;

DROP EXTENSION

### 7.2.12、数据库相关操作

* \l显示当前集群中所有数据库

postgres=# \l

List of databases

Name | Owner | Encoding | Collate | Ctype | Access privileges

-----------+-------+----------+------------+------------+-------------------

postgres | tbase | UTF8 | en\_US.utf8 | en\_US.utf8 |

template0 | tbase | UTF8 | en\_US.utf8 | en\_US.utf8 | =c/tbase +

| | | | | tbase=CTc/tbase

template1 | tbase | UTF8 | en\_US.utf8 | en\_US.utf8 | =c/tbase +

| | | | | tbase=CTc/tbase

(3 rows)

* \l+显示当前当前集群中所有数据库（包含库大小及注释）

注意，如果节点特别多，数据表特别多，使用\l+时统计比较耗时

postgres=# \l+

List of databases

Name | Owner | Encoding | Collate | Ctype | Access privileges | Size | Tablespace | Description

-----------+-------+----------+------------+------------+-------------------+-------+------------+------------------------------------

postgres | tbase | UTF8 | en\_US.utf8 | en\_US.utf8 | | 17 MB | pg\_default |

template0 | tbase | UTF8 | en\_US.utf8 | en\_US.utf8 | =c/tbase +| 14 MB | pg\_default | unmodifiable empty database

| | | | | tbase=CTc/tbase | | |

template1 | tbase | UTF8 | en\_US.utf8 | en\_US.utf8 | =c/tbase +| 14 MB | pg\_default | default template for new databases

| | | | | tbase=CTc/tbase | | |

(3 rows)

* 创建一个新库

postgres=# create database mydb;

CREATE DATABASE

postgres=#

### 7.2.13、模式相关操作

* \dn显示当前库所有模式

postgres=# \dn

List of schemas

Name | Owner

--------+-------

pgxc | tbase

public | tbase

(2 rows)

* \dn+显示当前库所有模式（包含注释）

postgres=# \dn+

List of schemas

Name | Owner | Access privileges | Description

--------+-------+-------------------+------------------------

pgxc | tbase | |

public | tbase | tbase=UC/tbase +| standard public schema

| | =UC/tbase |

(2 rows)

* 创建一个新模式

postgres=# create schema mysche;

CREATE SCHEMA

### 7.2.14、用户相关操作

* \du显示当前集群中所有数据库用户

postgres=# \du

List of roles

Role name | Attributes | Member of

---------------+------------------------------------------------------------+-----------

audit\_admin | No inheritance | {}

mls\_admin | No inheritance | {}

tbase | Superuser, Create role, Create DB, Replication, Bypass RLS | {}

tbase01\_admin | Superuser, Create role, Create DB | {}

* \du+显示当前集群中所有数据库用户（包含注释）

postgres=# \du+

List of roles

Role name | Attributes | Member of | Description

---------------+------------------------------------------------------------+-----------+-------------

audit\_admin | No inheritance | {} |

mls\_admin | No inheritance | {} |

tbase | Superuser, Create role, Create DB, Replication, Bypass RLS | {} |

tbase01\_admin | Superuser, Create role, Create DB | {} |

* 创建一个新的用户

postgres=# create role pgxc with login ;

CREATE ROLE

* 配置用户密码

postgres=# \password pgxc

Enter new password:

Enter it again:

### 7.2.15、表相关操作

* 建立数据表

postgres=# create table tbase(id int,mc text) distribute by shard(id);

CREATE TABLE

* \d查看表结构，包括使用的触发器

postgres=# \d tbase

Table "public.tbase"

Column | Type | Modifiers

--------+---------+-----------

id | integer |

mc | text |

* \d+查看表结构（包含注释），表类型，分布节点

postgres=# \d+ tbase

Table "public.tbase"

Column | Type | Modifiers | Storage | Stats target | Description

--------+---------+-----------+----------+--------------+-------------

id | integer | | plain | |

mc | text | | extended | |

Has OIDs: no

Distribute By SHARD(id)

Location Nodes: ALL DATANODES

* \dt查看表列表

postgres=# \dt

List of relations

Schema | Name | Type | Owner

--------+--------------+-------+-------

public | t\_time\_range | table | tbase

public | tbase | table | tbase

(2 rows)

* \dt+查看表列表详细信息，包含表大小和注释

这里连接的节点如果是cn的话，表大小为所有dn节点大小之和，否则为只是该节点的表大小

postgres=# \dt+

List of relations

Schema | Name | Type | Owner | Size | Description

--------+--------------+-------+-------+---------+--------------------

public | t\_time\_range | table | tbase | 0 bytes | 这是一个日期分区表

public | tbase | table | tbase | 576 kB |

(2 rows)

* \dt+显示某个模式下的所有表

postgres=# \dt+ pgxc.\*

List of relations

Schema | Name | Type | Owner | Size | Description

--------+------------+-------+-------+---------+-------------

pgxc | order\_main | table | tbase | 0 bytes |

(1 row)

* \dt+表名 显示某个表的详细信息

postgres=# \dt+ tbase

List of relations

Schema | Name | Type | Owner | Size | Description

--------+-------+-------+-------+--------+-------------

public | tbase | table | tbase | 576 kB |

(1 row)

* \dt+通配符列出适配的表

postgres=# \dt+ t\*

List of relations

Schema | Name | Type | Owner | Size | Description

--------+--------------+-------+-------+---------+--------------------

public | t\_time\_range | table | tbase | 0 bytes | 这是一个日期分区表

public | tbase | table | tbase | 576 kB |

(2 rows)

postgres=# \dt+ t\_\*

List of relations

Schema | Name | Type | Owner | Size | Description

--------+--------------+-------+-------+---------+--------------------

public | t\_time\_range | table | tbase | 0 bytes | 这是一个日期分区表

(1 row)

### 7.2.16、视图相关操作

* 建立视图

postgres=# create or replace view tbase\_view as select \* from tbase;

CREATE VIEW

* \d查视图结构

postgres=# \d tbase

Table "public.tbase"

Column | Type | Modifiers

--------+---------+-----------

id | integer |

mc | text |

* \d+查看视图结构（包含注释），包含创建视图的sql语句

postgres=# \d+ tbase\_view

View "public.tbase\_view"

Column | Type | Modifiers | Storage | Description

--------+---------+-----------+----------+-------------

id | integer | | plain |

mc | text | | extended |

View definition:

SELECT tbase.id,

tbase.mc

FROM tbase;

* \dv查看视图列表

postgres=# \dv

List of relations

Schema | Name | Type | Owner

--------+-------------------+------+-------

pgxc | t\_time\_range\_view | view | tbase

public | tbase\_view | view | tbase

(2 rows)

* \dv+查看视图列表详细信息（包含注释）

postgres=# \dv+

List of relations

Schema | Name | Type | Owner | Size | Description

--------+-------------------+------+-------+---------+-------------

pgxc | t\_time\_range\_view | view | tbase | 0 bytes |

public | tbase\_view | view | tbase | 0 bytes | 我的视图

(2 rows)

* \dv+显示某个模式下的所有视图

postgres=# \dv+ pgxc.\*

List of relations

Schema | Name | Type | Owner | Size | Description

--------+-------------------+------+-------+---------+-------------

pgxc | t\_time\_range\_view | view | tbase | 0 bytes |

(1 row)

* \dv+视图名 显示某个视图的详细信息

postgres=# \dv+ tbase\_view

List of relations

Schema | Name | Type | Owner | Size | Description

--------+------------+------+-------+---------+-------------

public | tbase\_view | view | tbase | 0 bytes | 我的视图

(1 row)

* \dv+通配符列出适配的视图

postgres=# \dv+ t\*

List of relations

Schema | Name | Type | Owner | Size | Description

--------+-------------------+------+-------+---------+-------------

pgxc | t\_time\_range\_view | view | tbase | 0 bytes |

public | tbase\_view | view | tbase | 0 bytes | 我的视图

(2 rows)

postgres=# \dv+ tb\*

List of relations

Schema | Name | Type | Owner | Size | Description

--------+------------+------+-------+---------+-------------

public | tbase\_view | view | tbase | 0 bytes | 我的视图

(1 row)

### 7.2.17、物化视图相关操作

* 建立物化视图

postgres=# create MATERIALIZED VIEW tbase\_count as select count(1) as num from tbase;

SELECT 1

postgres=# select \* from tbase\_count;

num

-------

10000

(1 row)

* \d查物化视图结构

postgres=# \d tbase\_count

Materialized view "public.tbase\_count"

Column | Type | Modifiers

--------+--------+-----------

num | bigint |

* \d+查看物化视图结构（包含注释），包含创建物化视图的sql语句

postgres=# \d+ tbase\_count

Materialized view "public.tbase\_count"

Column | Type | Modifiers | Storage | Stats target | Description

--------+--------+-----------+---------+--------------+-------------

num | bigint | | plain | |

View definition:

SELECT count(1) AS num

FROM tbase;

* \dm查看视图列表

postgres=# \dm

List of relations

Schema | Name | Type | Owner

--------+-------------+-------------------+-------

pgxc | tbase\_sum | materialized view | tbase

public | tbase\_count | materialized view | tbase

(2 rows)

* \dm+查看物化视图列表详细信息（包含注释），占用空间大小

postgres=# \dm+

List of relations

Schema | Name | Type | Owner | Size | Description

--------+-------------+-------------------+-------+------------+---------------

pgxc | tbase\_sum | materialized view | tbase | 8192 bytes |

public | tbase\_count | materialized view | tbase | 8192 bytes | tbase总记录数

(2 rows)

* \dm+显示某个模式下的所有物化视图

postgres=# \dm+ pgxc.\*

List of relations

Schema | Name | Type | Owner | Size | Description

--------+-----------+-------------------+-------+------------+-------------

pgxc | tbase\_sum | materialized view | tbase | 8192 bytes |

(1 row)

* \dm+视图名 显示某个物化视图的详细信息

postgres=# \dm+ tbase\_count

List of relations

Schema | Name | Type | Owner | Size | Description

--------+-------------+-------------------+-------+------------+---------------

public | tbase\_count | materialized view | tbase | 8192 bytes | tbase总记录数

(1 row)

* \dm+通配符列出适配的物化视图

postgres=# \dm t\*

List of relations

Schema | Name | Type | Owner

--------+-------------+-------------------+-------

pgxc | tbase\_sum | materialized view | tbase

public | tbase\_count | materialized view | tbase

(2 rows)

postgres=# \dm tbase\_c\*

List of relations

Schema | Name | Type | Owner

--------+-------------+-------------------+-------

public | tbase\_count | materialized view | tbase

(1 row)

### 7.2.18、序列相关操作

* 建立序列

postgres=# create sequence tbase\_seq;

CREATE SEQUENCE

postgres=# create sequence pgxc.tbase\_seq;

CREATE SEQUENCE

* \d查看序列定义和使用情况

postgres=# \d tbase\_seq

Sequence "public.tbase\_seq"

Column | Type | Value

------------+---------+-------

last\_value | bigint | 1

log\_cnt | bigint | 0

is\_called | boolean | f

* \ds查看序列列表

postgres=# \ds

List of relations

Schema | Name | Type | Owner

--------+-----------+----------+-------

pgxc | tbase\_seq | sequence | tbase

public | tbase\_seq | sequence | tbase

(2 rows)

* \ds+查看序列列表详细信息（包含注释），占用空间大小

postgres=# \ds+

List of relations

Schema | Name | Type | Owner | Size | Description

--------+-----------+----------+-------+------------+-------------

pgxc | tbase\_seq | sequence | tbase | 8192 bytes |

public | tbase\_seq | sequence | tbase | 8192 bytes | tbase序列

(2 rows)

* \ds+显示某个模式下的所有序列

postgres=# \ds+ pgxc.\*

List of relations

Schema | Name | Type | Owner | Size | Description

--------+----------+----------+-------+------------+-------------

pgxc | tbase\_seq | sequence | tbase | 8192 bytes |

(1 row)

* \ds+序列名 显示某个序列的详细信息

postgres=# \ds+ tbase\_seq

List of relations

Schema | Name | Type | Owner | Size | Description

--------+-----------+----------+-------+------------+-------------

public | tbase\_seq | sequence | tbase | 8192 bytes | tbase序列

(1 row)

* \ds+通配符列出适配的序列

postgres=# \ds \*\_seq

List of relations

Schema | Name | Type | Owner

--------+-----------+----------+-------

pgxc | tbase\_seq | sequence | tbase

public | tbase\_seq | sequence | tbase

(2 rows)

postgres=# \ds t\*\_seq

List of relations

Schema | Name | Type | Owner

--------+-----------+----------+-------

public | tbase\_seq | sequence | tbase

(1 row)

### 7.2.19、索引相关操作

* 建立索引

postgres=# create unique index tbase\_id\_uidx on tbase(id);

CREATE INDEX

postgres=# create index tbase\_mc\_idx on tbase(mc);

CREATE INDEX

postgres=#

* \di查看索引列表

postgres=# \di

List of relations

Schema | Name | Type | Owner | Table

--------+---------------+-------+-------+-------

public | tbase\_id\_uidx | index | tbase | tbase

public | tbase\_mc\_idx | index | tbase | tbase

(2 rows)

* \di+查看索引列表详细信息（包含注释），占用空间大小--只能在dn上面查看索引大小

postgres=# \di+

List of relations

Schema | Name | Type | Owner | Table | Size | Description

--------+---------------+-------+-------+-------+------------+---------------

public | tbase\_id\_uidx | index | tbase | tbase | 8192 bytes | tbase唯一索引

public | tbase\_mc\_idx | index | tbase | tbase | 8192 bytes |

(2 rows)

* \di+显示某个模式下的所有索引--只能在dn上面查看索引大小

postgres=# \di+ public.\*

List of relations

Schema | Name | Type | Owner | Table | Size | Description

--------+---------------+-------+-------+-------+------------+---------------

public | tbase\_id\_uidx | index | tbase | tbase | 8192 bytes | tbase唯一索引

public | tbase\_mc\_idx | index | tbase | tbase | 8192 bytes |

(2 rows)

* \di+索引名 显示某个索引的详细信息--只能在dn上面查看索引大小

postgres=# \di+ public.tbase\_id\_uidx

List of relations

Schema | Name | Type | Owner | Table | Size | Description

--------+---------------+-------+-------+-------+------------+---------------

public | tbase\_id\_uidx | index | tbase | tbase | 8192 bytes | tbase唯一索引

(1 row)

* \di+通配符列出适配的索引--只能在dn上面查看索引大小

postgres=# \di+ \*idx

List of relations

Schema | Name | Type | Owner | Table | Size | Description

--------+-------------------+-------+-------+------------+------------+---------------

pgxc | order\_main\_id\_idx | index | tbase | order\_main | 8192 bytes |

public | tbase\_id\_uidx | index | tbase | tbase | 8192 bytes | tbase唯一索引

public | tbase\_mc\_idx | index | tbase | tbase | 8192 bytes |

(3 rows)

postgres=# \di+ \*uidx--只能在dn上面查看索引大小

List of relations

Schema | Name | Type | Owner | Table | Size | Description

--------+---------------+-------+-------+-------+------------+---------------

public | tbase\_id\_uidx | index | tbase | tbase | 8192 bytes | tbase唯一索引

(1 row)

### 7.2.20、函数相关操作

* 建立函数

postgres=# CREATE OR REPLACE FUNCTION tbase\_f1(a\_1 text) returns text as

postgres-# $$

postgres$# begin

postgres$# return a\_1;

postgres$# end;

postgres$# $$

postgres-# language plpgsql;

CREATE FUNCTION

postgres=# CREATE OR REPLACE FUNCTION pgxc.tbase\_f2(a\_2 text) returns text as

postgres-# $$

postgres$# begin

postgres$# return a\_2;

postgres$# end;

postgres$# $$

postgres-# language plpgsql;

CREATE FUNCTION

postgres=#

* \df查看函数列表

postgres=# \df

List of functions

Schema | Name | Result data type | Argument data types | Type

--------+----------+------------------+---------------------+--------

pgxc | tbase\_f2 | text | a\_2 text | normal

public | tbase\_f1 | text | a\_1 text | normal

(2 rows)

* \df+查看函数列表详细信息（包含注释），定义

postgres=# \x

Expanded display is on.

postgres=# \df+ tbase\*

List of functions

-[ RECORD 1 ]-------+--------------------

Schema | pgxc

Name | tbase\_f2

Result data type | text

Argument data types | a\_2 text

Type | normal

Volatility | volatile

Parallel | unsafe

Owner | tbase

Security | invoker

Access privileges |

Language | plpgsql

Source code | +

| begin +

| return a\_2; +

| end; +

|

Description |

-[ RECORD 2 ]-------+--------------------

Schema | public

Name | tbase\_f1

Result data type | text

Argument data types | a\_1 text

Type | normal

Volatility | volatile

Parallel | unsafe

Owner | tbase

Security | invoker

Access privileges |

Language | plpgsql

Source code | +

| begin +

| return a\_1;+

| end; +

|

Description |

* \df+显示某个模式下的所有索引

-[ RECORD 1 ]-------+--------------------

Schema | public

Name | tbase\_f1

Result data type | text

Argument data types | a\_1 text

Type | normal

Volatility | volatile

Parallel | unsafe

Owner | tbase

Security | invoker

Access privileges |

Language | plpgsql

Source code | +

| begin +

| return a\_1;+

| end; +

|

Description |

* \df+函数名 显示某个函数的详细信息

postgres=# \df+ tbase\_f1

List of functions

-[ RECORD 1 ]-------+--------------------

Schema | public

Name | tbase\_f1

Result data type | text

Argument data types | a\_1 text

Type | normal

Volatility | volatile

Parallel | unsafe

Owner | tbase

Security | invoker

Access privileges |

Language | plpgsql

Source code | +

| begin +

| return a\_1;+

| end; +

|

Description |

* \df+通配符列出适配的函数

postgres=# \df+ tbase\*

List of functions

-[ RECORD 1 ]-------+--------------------

Schema | pgxc

Name | tbase\_f2

Result data type | text

Argument data types | a\_2 text

Type | normal

Volatility | volatile

Parallel | unsafe

Owner | tbase

Security | invoker

Access privileges |

Language | plpgsql

Source code | +

| begin +

| return a\_2; +

| end; +

|

Description |

-[ RECORD 2 ]-------+--------------------

Schema | public

Name | tbase\_f1

Result data type | text

Argument data types | a\_1 text

Type | normal

Volatility | volatile

Parallel | unsafe

Owner | tbase

Security | invoker

Access privileges |

Language | plpgsql

Source code | +

| begin +

| return a\_1;+

| end; +

|

Description |

postgres=# \df+ \*f1

List of functions

-[ RECORD 1 ]-------+--------------------

Schema | public

Name | tbase\_f1

Result data type | text

Argument data types | a\_1 text

Type | normal

Volatility | volatile

Parallel | unsafe

Owner | tbase

Security | invoker

Access privileges |

Language | plpgsql

Source code | +

| begin +

| return a\_1;+

| end; +

|

Description |

### 7.2.21、自定义数据类型相关操作

* 建立数据类型

postgres=# CREATE TYPE bug\_status AS ENUM ('new', 'open', 'closed');

CREATE TYPE

* \dT查看自定义数据类型列表

postgres=# \dT

List of data types

Schema | Name | Description

-----------+------------+--------------------------

pg\_oracle | nvarchar2 | oracle nvarchar2(length)

pg\_oracle | varchar2 | oracle varchar2(length)

public | bug\_status |

* \dT+查看自定义数据类型列表详细信息（包含enum类型的值）

postgres=# \dT+

List of data types

Schema | Name | Internal name | Size | Elements | Access privileges | Description

-----------+------------+---------------+-------+----------+-------------------+--------------------------

pg\_oracle | nvarchar2 | nvarchar2 | var | | | oracle nvarchar2(length)

pg\_oracle | varchar2 | varchar2 | var | | | oracle varchar2(length)

public | bug\_status | bug\_status | 4 | new +| |

| | | | open +| |

| | | | closed | |

* \dT+显示某个模式下的所有自定义类型

postgres=# \dT+ public.\*

List of data types

Schema | Name | Internal name | Size | Elements | Access privileges | Description

--------+------------+---------------+-------+----------+-------------------+-------------

public | bug\_status | bug\_status | 4 | new +| |

| | | | open +| |

| | | | closed | |

* \dT+自定义数据类型 显示某个数据类型的详细信息

postgres=# \dT+ bug\_status

List of data types

Schema | Name | Internal name | Size | Elements | Access privileges | Description

--------+------------+---------------+------+----------+-------------------+-------------

public | bug\_status | bug\_status | 4 | new +| |

| | | | open +| |

| | | | closed | |

(1 row)

* \dT+通配符列出适配的数据类型

postgres=# \dT+ bug\_\*

List of data types

Schema | Name | Internal name | Size | Elements | Owner | Access privileges | Description

--------+------------+---------------+------+----------+-------+-------------------+-------------

public | bug\_status | bug\_status | 4 | new +| tbase | |

| | | | open +| | |

| | | | closed | | |

### 7.2.22、存储过程语句相关操作

* 加载某个存储过程语言

postgres=# create language plpgsql;

CREATE LANGUAGE

* \dL查看存储过程语言列表

postgres=# \dL

List of languages

Name | Owner | Trusted | Description

---------+-------+---------+------------------------------

plpgsql | tbase | t | PL/pgSQL procedural language

(1 row)

### 7.2.23、列出表、视图和序列和它们相关的访问权限

* 显示所有对象的访问权限

postgres=# \dp

Access privileges

Schema | Name | Type | Access privileges | Column privileges | Policies

--------+--------------------+----------+-------------------+-------------------+----------

public | pg\_stat\_statements | view | tbase=arwdDxt/tbase+| |

| | | =r/tbase | |

public | tbase | table | tbase=arwdDxt/tbase+| |

| | | pgxc=r/tbase | |

public | tbase\_seq | sequence | tbase=rwU/tbase +| |

| | | pgxc=rwU/tbase | |

(3 rows)

* \dp+某个对象名，只显示匹配的对象

postgres=# \dp tbase

Access privileges

Schema | Name | Type | Access privileges | Column privileges | Policies

--------+-------+-------+-------------------+-------------------+----------

public | tbase | table | tbase=arwdDxt/tbase+| |

| | | pgxc=r/tbase | |

(1 row)

postgres=# \dp public.\*

Access privileges

Schema | Name | Type | Access privileges | Column privileges | Policies

--------+--------------------+----------+-------------------+-------------------+----------

public | pg\_stat\_statements | view | tbase=arwdDxt/tbase+| |

| | | =r/tbase | |

public | tbase | table | tbase=arwdDxt/tbase+| |

| | | pgxc=r/tbase | |

public | tbase\_seq | sequence | tbase=rwU/tbase +| |

| | | pgxc=rwU/tbase | |

(3 rows)

### 7.2.24、列出库或用户定义的配置

postgres=# \drds

List of settings

Role | Database | Settings

------+----------+-----------------------------------

pgxc | | log\_statement=none

| postgres | search\_path="$user", public, pgxc

(2 rows)

### 7.2.25、copy命令的使用

这是一个针对客户端文件操作的copy应用，服务器端copy使用见开发工程师文档4.9

* \copy to 将数据复制到本地文件中

postgres=# \copy tbase to '/data/tbase/tbase.txt';

postgres=# \! cat /data/tbase/tbase.txt

1 tbase

* \copy from 将本地文件复制到数据表中

postgres=# \copy tbase from '/data/tbase/tbase.txt';

### 7.2.26、copy FROM stdin使用方法

postgres=# COPY tbase (id, mc) FROM stdin;

Enter data to be copied followed by a newline.

End with a backslash and a period on a line by itself.

>> 1 tbase

>> 2 \N

>> 3 pgxc

>> \.

postgres=# select \* from tbase;

id | mc

----+-------

1 | tbase

2 |

3 | pgxc

(3 rows)

### 7.2.27、打印当前查询缓冲区到标准输出

postgres=# select \* from tbase;

id | mc

----+-------

1 | tbase

(1 row)

postgres=# \p

select \* from tbase;

### 7.2.28、自定义显示格式

* 配置null的显示代替字符串

postgres=# \pset null '(null)'

Null display is "(null)".

postgres=# insert into tbase values(2,null);

INSERT 0 1

postgres=# select \* from tbase;

id | mc

----+--------

1 | tbase

2 | (null)

(2 rows)

* 不显示边框

postgres=# \pset border 0

Border style is 0.

postgres=# select \* from tbase;

id mc

-- ------

1 tbase

2 (null)

(2 rows)

* 只显示记录

postgres=# \pset tuples\_only

Showing only tuples.

postgres=# select \* from tbase;

1 tbase

2 (null)

* 列之间使用逗号分隔

postgres=# \pset format unaligned

Output format is unaligned.

postgres=# \pset fieldsep ,

Field separator is ",".

postgres=# select \* from tbase;

1,tbase

2,(null)

### 7.2.29、显示psql内部操作

postgres=# \set ECHO\_HIDDEN on

postgres=# \dt+ t

\*\*\*\*\*\*\*\*\* QUERY \*\*\*\*\*\*\*\*\*\*

SELECT n.nspname as "Schema",

c.relname as "Name",

CASE c.relkind WHEN 'r' THEN 'table' WHEN 'v' THEN 'view' WHEN 'm' THEN 'materialized view' WHEN 'i' THEN 'index' WHEN 'S' THEN 'sequence' WHEN 's' THEN 'special' WHEN 'f' THEN 'foreign table' WHEN 'p' THEN 'table' END as "Type",

pg\_catalog.pg\_get\_userbyid(c.relowner) as "Owner",

pg\_catalog.pg\_size\_pretty(pg\_catalog.pg\_table\_size(c.oid)) as "Size",

pg\_catalog.pg\_size\_pretty(pg\_catalog.pg\_allocated\_table\_size(c.oid)) as "Allocated Size",

pg\_catalog.obj\_description(c.oid, 'pg\_class') as "Description"

FROM pg\_catalog.pg\_class c

LEFT JOIN pg\_catalog.pg\_namespace n ON n.oid = c.relnamespace

WHERE c.relkind IN ('r','p','s','')

AND n.nspname !~ '^pg\_toast'

AND c.relname ~ '^(t)$'

AND pg\_catalog.pg\_table\_is\_visible(c.oid)

ORDER BY 1,2;

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

List of relations

Schema | Name | Type | Owner | Size | Allocated Size | Description

--------+------+-------+-------+-------+----------------+-------------

public | t | table | tbase | 16 kB | 0 bytes |

(1 row)

#禁用显示psql内部操作

postgres=# \set ECHO\_HIDDEN off

### 7.2.30、重复执行上一条语句

postgres=# select count(1) from pg\_stat\_activity where state!='idle';

count

-------

1

(1 row)

postgres=# \watch 1

Fri 28 Sep 2018 04:58:51 PM CST (every 1s)

count

-------

1

(1 row)

Fri 28 Sep 2018 04:58:52 PM CST (every 1s)

count

-------

1

(1 row)

上面的语句为每次自动查询活跃的进程数，相当于临时监控作用

### 7.2.31、sql命令帮助查看

postgres=# \h

Available help:

ABORT ALTER USER CREATE ROLE DROP INDEX LOCK

ALTER AGGREGATE ALTER USER MAPPING CREATE RULE DROP LANGUAGE MOVE

ALTER COLLATION ALTER VIEW CREATE SCHEMA DROP MATERIALIZED VIEW NOTIFY

。。。

postgres=# \h begin;

Command: BEGIN

Description: start a transaction block

Syntax:

BEGIN [ WORK | TRANSACTION ] [ transaction\_mode [, ...] ]

where transaction\_mode is one of:

ISOLATION LEVEL { SERIALIZABLE | REPEATABLE READ | READ COMMITTED | READ UNCOMMITTED }

READ WRITE | READ ONLY

[ NOT ] DEFERRABLE

postgres=#

# 8、TBase数据库开发基础

## 8.1、TBase的数据类型

### 8.1.1、数字类型

|  |  |  |  |
| --- | --- | --- | --- |
| **名字** | **存储尺寸** | **描述** | **范围** |
| smallint | 2字节 | 小范围整数 | -32768 to +32767 |
| integer | 4字节 | 整数的典型选择 | -2147483648 to +2147483647 |
| bigint | 8字节 | 大范围整数 | -9223372036854775808 to +9223372036854775807 |
| decimal | 可变 | 用户指定精度，精确 | 最高小数点前131072位，以及小数点后16383位 |
| numeric | 可变 | 用户指定精度，精确 | 最高小数点前131072位，以及小数点后16383位 |
| real | 4字节 | 可变精度，不精确 | 6位十进制精度 |
| double precision | 8字节 | 可变精度，不精确 | 15位十进制精度 |
| smallserial | 2字节 | 自动增加的小整数 | 1到32767 |
| serial | 4字节 | 自动增加的整数 | 1到2147483647 |
| bigserial | 8字节 | 自动增长的大整数 | 1到9223372036854775807 |

### 8.1.2、字符类型

|  |  |
| --- | --- |
| **名字** | **描述** |
| character varying(n), varchar(n) | 有限制的变长 |
| character(n), char(n) | 定长，空格填充 |
| text | 1G |

### 8.1.3、二进制数据类型

|  |  |  |
| --- | --- | --- |
| **名字** | **存储尺寸** | **描述** |
| bytea | 1或4字节外加真正的二进制串 | 变长二进制串 |

### 8.1.4、日期类型

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **名字** | **存储尺寸** | **描述** | **最小值** | **最大值** | **解析度** |
| timestamp [ (*p*) ] [ without time zone ] | 8字节 | 包括日期和时间（无时区） | 4713 BC | 294276 AD | 1微秒 / 14位 |
| timestamp [ (*p*) ] with time zone | 8字节 | 包括日期和时间，有时区 | 4713 BC | 294276 AD | 1微秒 / 14位 |
| date | 4字节 | 日期（没有一天中的时间） | 4713 BC | 5874897 AD | 1日 |
| time [ (*p*) ] [ without time zone ] | 8字节 | 一天中的时间（无日期） | 0:00:00 | 24:00:00 | 1微秒 / 14位 |
| time [ (*p*) ] with time zone | 12字节 | 仅仅是一天中的时间，带有时区 | 00:00:00+1459 | 24:00:00-1459 | 1微秒 / 14位 |
| interval [ *fields* ] [ (*p*) ] | 16字节 | 时间间隔 | -178000000年 | 178000000年 | 1微秒 / 14位 |

### 8.1.5、布尔类型

|  |  |  |
| --- | --- | --- |
| **名字** | **存储字节** | **描述** |
| boolean | 1字节 | 状态为真或假 |

### 8.1.6、更多的数据类型介绍

请点击连接<https://www.postgresql.org/docs/10/static/datatype.html>查看更多的数据类型说明

### 8.1.7、异构数据库类型对照表

#### 8.1.7.1、与oracle对照表

|  |  |
| --- | --- |
| Oracle | TBase |
| numeric | 可以对应TBase的smallint, integer, bigint, numeric(p,s)等多种数据类型。由于smallint，Integer，bigint的算术运算效率比numberic高的多,所以要视业务需要转换成对应的smallint, integer, bigint，无法转换时才转换成numeric(p,s) |
| float | double precision |
| binary\_float | real |
| binary\_double | double precision |
| char | char |
| nchar | char |
| varchar2 | varchar |
| nvarchar2 | varchar |
| rowid | char(18) |
| urowid | varchar |
| long | text |
| clob | text |
| nclob | text |
| blob | bytea |
| bfile | bytea |
| Long raw | bytea |
| raw | bytea |
| date | Timestamp(0) |
| timestamp | Timestamp |
| Interval | interval |

#### 8.1.7.2、与mysql对照表

|  |  |
| --- | --- |
| Mysql | TBase |
| int | int |
| smallint | smallint |
| bigint | bigint |
| int AUTO\_INCREMENT | serial |
| smallint AUTO\_INCREMENT | smallserial |
| bigint AUTO\_INCREMENT | bigserial |
| bit | bit |
| tinyint | boolean |
| float | real |
| double | double precision |
| decimal | numeric |
| char | char |
| varchar | varchar |
| text | text |
| date | date |
| time | time |
| datetime | timestamp |
| longblob | bytea |
| Longtext | text |
| ENUM  CREATE TABLE TYPE022(COL1 ENUM('S','M','L','XL','XXL','XXXL') ,COL2 INT PRIMARY KEY); | 自定义类型  CREATE TYPE mood AS ENUM ('S','M','L','XL','XXL','XXXL');  CREATE TABLE TYPE022(COL1 mood ,COL2 INT PRIMARY KEY) |
| SET类型  CREATE TABLE TYPE023(COL1 SET('A','B', 'C','D') ,COL2 INT PRIMARY KEY) | CREATE TABLE TYPE023(COL1 VARCHAR check(regexp\_split\_to\_array(col1,',') <@ array['A','B','C','D']) ,COL2 INT PRIMARY KEY); |

#### 8.1.7.3、与sqlserver对照表

|  |  |
| --- | --- |
| SQLSERVER | TBase |
| smallint | smallint |
| int | int |
| bigint | bigint |
| tinyint | smallint |
| real | real |
| float | double precision |
| numeric | numeric |
| bit | bit |
| char | char |
| nchar | char |
| varchar | varchar |
| nvarchar | varchar |
| text | text |
| ntext | text |
| date | date |
| time | time |
| datetime | timestamp |
| datetime2 | timestamp |
| smalldatetime | timestamp |
| datetimeoffset | Timestamp |
| timestamp | money |
| uniqueidentifier | uuid |
| image | bytea |
| binary | bytea |
| varbinary | bytea |

## 8.2、自增列于序列的用法

### 8.2.1、序列的创建和访问

[tbase@VM\_0\_29\_centos tbase\_mgr]$ psql -p 15432 -U tbase -d postgres

psql (PostgreSQL 10 (TBase 2.01))

Type "help" for help.

**--建立序列**

postgres=# create sequence tbase\_seq;

CREATE SEQUENCE

**--建立序列，不存在时才创建**

postgres=# create sequence IF NOT EXISTS tbase\_seq;

NOTICE: relation "tbase\_seq" already exists, skipping

CREATE SEQUENCE

**--查看序列当前的使用状况**

postgres=# \x

Expanded display is on.

postgres=# select \* from tbase\_seq ;

-[ RECORD 1 ]-

last\_value | 1

log\_cnt | 0

is\_called | f

**--获取序列的下一个值**

postgres=# select nextval('tbase\_seq');

nextval

---------

1

**--获取序列的当前值，这个需要在访问nextval()后才能使用**

postgres=# select currval('tbase\_seq');

currval

---------

1

你可以后下面的方式来获取序列当前使用到那一个值

postgres=# select last\_value from tbase\_seq ;

last\_value

------------

3

**--设置序列当前值**

postgres=# select setval('tbase\_seq',1);

setval

--------

1

### 8.2.2、序列在DML中使用

postgres=# INSERT INTO t (id,nickname) VALUES(nextval('tbase\_seq'),'TBase好牛');

INSERT 0 1

postgres=# select \* from t;

id | nickname

----+-----------

1 | 腾讯TBase

2 | TBase好牛

(2 rows)

### 8.2.3、序列做为字段的默认值使用

postgres=# alter table t alter column id set default nextval('tbase\_seq');

postgres=# INSERT INTO t (nickname) VALUES('hello TBase');

INSERT 0 1

postgres=# select \* from t;

id | nickname

----+-------------

3 | hello TBase

1 | 腾讯TBase

2 | TBase好牛

(3 rows)

### 8.2.4、序列做为字段类型使用

postgres=# drop table t;

DROP TABLE

postgres=# create table t (id serial not null,nickname text);

CREATE TABLE

postgres=# INSERT INTO t (nickname) VALUES('hello TBase');

INSERT 0 1

postgres=# select \* from t;

id | nickname

----+-------------

1 | hello TBase

(1 row)

postgres=#

### 8.2.5、删除序列

postgres=# drop sequence tbase\_seq;

DROP SEQUENCE

**--删除序列，不存在时跳过**

postgres=# drop sequence IF EXISTS tbase\_seq;

NOTICE: sequence "tbase\_seq" does not exist, skipping

DROP SEQUENCE

postgres=#

## 8.3、库/模式/表/索引/视图/物化视图等DDL操作

### 8.3.1、数据库管理

#### 8.3.1.1、创建数据库

要创建一个数据库，你必须是一个超级用户或者具有特殊的CREATEDB特权，默认情况下，新数据库将通过克隆标准系统数据库template1被创建。可以通过写TEMPLATE name指定一个不同的模板。特别地，通过写TEMPLATE template0你可以创建一个干净的数据库，它将只包含你的TBase版本所预定义的标准对象。

* 默认参数创建数据库

postgres=# create database tbase\_db;

CREATE DATABASE

* 指定克隆库

postgres=# create database tbase\_db\_template TEMPLATE template0;

CREATE DATABASE

* 指定所有者

postgres=# create role pgxz with login;

CREATE ROLE

postgres=# create database tbase\_db\_owner owner pgxz;

CREATE DATABASE

postgres=# \l+ tbase\_db\_owner

List of databases

Name | Owner | Encoding | Collate | Ctype | Access privileges | Size | Tablespace | Description

----------------+-------+----------+------------+------------+-------------------+-------+------------+-------------

tbase\_db\_owner | pgxz | UTF8 | en\_US.utf8 | en\_US.utf8 | | 18 MB | pg\_default |

(1 row)

* 指定编码

postgres=# create database tbase\_db\_encoding ENCODING UTF8;

CREATE DATABASE

postgres=# \l+ tbase\_db\_encoding

List of databases

Name | Owner | Encoding | Collate | Ctype | Access privileges | Size | Tablespace | Description

-------------------+-------+----------+------------+------------+-------------------+-------+------------+-------------

tbase\_db\_encoding | tbase | UTF8 | en\_US.utf8 | en\_US.utf8 | | 18 MB | pg\_default |

(1 row)

* 指定排序规则

postgres=# create database tbase\_db\_lc\_collate lc\_collate 'C';

CREATE DATABASE

postgres=# \l+ tbase\_db\_lc\_collate

List of databases

Name | Owner | Encoding | Collate | Ctype | Access privileges | Size | Tablespace | Description

---------------------+-------+----------+---------+------------+-------------------+-------+------------+-------------

tbase\_db\_lc\_collate | tbase | UTF8 | C | en\_US.utf8 | | 18 MB | pg\_default |

(1 row)

* 指定分组规则

postgres=# create database tbase\_db\_lc\_ctype LC\_CTYPE 'C' ;

CREATE DATABASE

postgres=# \l+ tbase\_db\_lc\_ctype

List of databases

Name | Owner | Encoding | Collate | Ctype | Access privileges | Size | Tablespace | Description

-------------------+-------+----------+------------+-------+-------------------+-------+------------+-------------

tbase\_db\_lc\_ctype | tbase | UTF8 | en\_US.utf8 | C | | 18 MB | pg\_default |

(1 row)

* 配置数据可连接

postgres=# create database tbase\_db\_allow\_connections ALLOW\_CONNECTIONS true;

CREATE DATABASE

postgres=# select datallowconn from pg\_database where datname='tbase\_db\_allow\_connections';

datallowconn

--------------

t

(1 row)

* 配置连接数

postgres=# create database tbase\_db\_connlimit CONNECTION LIMIT 100;

CREATE DATABASE

postgres=# select datconnlimit from pg\_database where datname='tbase\_db\_connlimit';

datconnlimit

--------------

100

(1 row)

* 配置数据库可以被复制

postgres=# create database tbase\_db\_istemplate is\_template true;

CREATE DATABASE

postgres=# select datistemplate from pg\_database where datname='tbase\_db\_istemplate';

datistemplate

---------------

t

(1 row)

* 多个参数一起配置

postgres=# create database tbase\_db\_mul owner pgxz CONNECTION LIMIT 50 template template0 encoding 'utf8' lc\_collate 'C';

CREATE DATABASE

#### 8.3.1.2、修改数据库配置

* 修改数据库名称

postgres=# alter database tbase\_db rename to tbase\_db\_new;

ALTER DATABASE

* 修改连接数

postgres=# alter database tbase\_db\_new connection limit 50;

ALTER DATABASE

* 修改数据库所有者

postgres=# alter database tbase\_db\_new owner to tbase;

ALTER DATABASE

* 配置数据默认运行参行

postgres=# alter database tbase\_db\_new set search\_path to public,pg\_catalog,pg\_oracle;

ALTER DATABASE

更多set的用法参考运维文档

* Alter database不支持的项目

|  |  |
| --- | --- |
| 项目 | 备注 |
| encoding | 编码 |
| lc\_collate | 排序规则 |
| lc\_ctype | 分组规则 |

#### 8.3.1.3、删除数据库

postgres=# drop database tbase\_db\_new;

DROP DATABASE

删除数据库时，如果该数据库已经有session连接上来，则会提示如下错误

postgres=# drop database tbase\_db\_template;

ERROR: database "tbase\_db\_template" is being accessed by other users

DETAIL: There is 1 other session using the database.

使用下面方法可以把session断开，然后再删除

postgres=# select pg\_terminate\_backend(pid) from pg\_stat\_activity where datname='tbase\_db\_template';

pg\_terminate\_backend

----------------------

t

(1 row)

postgres=# drop database tbase\_db\_template;

DROP DATABASE

### 8.3.2、模式管理

模式本质上是一个名字空间，oracle里一般叫用户，SQLSERVER中叫框架，MYSQL中叫数据库，模式里面包含表、数据类型、函数以及操作符，对象名称可以与在其他模式中存在的对象重名，访问某个模式中的对象时可以使用"模式名.对象名"。

#### 8.3.2.1、创建模式

* 标准语句

postgres=# create schema tbase;

CREATE SCHEMA

* 扩展语法，不存在时才创建

postgres=# create schema if not exists tbase ;

NOTICE: schema "tbase" already exists, skipping

CREATE SCHEMA

postgres=#

* 指定所属用户

postgres=# create schema tbase\_pgxz AUTHORIZATION pgxz;

CREATE SCHEMA

postgres=# \dn tbase\_pgxz

List of schemas

Name | Owner

------------+-------

tbase\_pgxz | pgxz

(1 row)

#### 8.3.2.2、修改模式属性

* 修改模式名

postgres=# alter schema tbase rename to tbase\_new;

ALTER SCHEMA

* 修改所有者

postgres=# alter schema tbase\_pgxz owner to tbase;

ALTER SCHEMA

#### 8.3.2.3、删除模式

postgres=# drop schema tbase\_new;

DROP SCHEMA

当模式中存在对象时，则会删除失败，提示如下

postgres=# create table tbase\_pgxz.t(id int);

NOTICE: Replica identity is needed for shard table, please add to this table through "alter table" command.

CREATE TABLE

postgres=# drop schema tbase\_pgxz;

ERROR: cannot drop schema tbase\_pgxz because other objects depend on it

DETAIL: table tbase\_pgxz.t depends on schema tbase\_pgxz

HINT: Use DROP ... CASCADE to drop the dependent objects too.

可以这样强制删除

postgres=# drop schema tbase\_pgxz CASCADE;

NOTICE: drop cascades to table tbase\_pgxz.t

DROP SCHEMA

#### 8.3.2.4、配置用户访问模式权限

普通用户对于某个模式下的对象访问除了访问对象要授权外，模式也需要授权

[tbase@VM\_0\_37\_centos root]$ psql -U tbase

psql (PostgreSQL 10.0 TBase V2)

Type "help" for help.

postgres=# create table tbase.t(id int);

NOTICE: Replica identity is needed for shard table, please add to this table through "alter table" command.

CREATE TABLE

--这里授权用户可以访问tbase.t表

postgres=# grant select on tbase.t to pgxz;

GRANT

postgres=# \q

[tbase@VM\_0\_37\_centos root]$ psql -U pgxz

psql (PostgreSQL 10.0 TBase V2)

Type "help" for help.

--在没授权用户可以使用tbase模式前，还是访问不了

postgres=> select \* from tbase.t;

ERROR: permission denied for schema tbase

LINE 1: select \* from tbase.t;

^

postgres=> \q

[tbase@VM\_0\_37\_centos root]$ psql -U tbase

psql (PostgreSQL 10.0 TBase V2)

Type "help" for help.

postgres=# grant USAGE on SCHEMA tbase to pgxz;

GRANT

postgres=# \q

[tbase@VM\_0\_37\_centos root]$ psql -U pgxz

psql (PostgreSQL 10.0 TBase V2)

Type "help" for help.

postgres=> select \* from tbase.t;

id

----

(0 rows)

postgres=>

#### 8.3.2.5、配置访问模式的顺序

TBase数据库有一个运行变量叫search\_path，其值为模式名列表，用于配置访问数据对象的顺序，如下所示

--当前连接用户

postgres=# select current\_user;

current\_user

--------------

tbase

(1 row)

--总共三个模式

postgres=# \dn

List of schemas

Name | Owner

--------------+-------

public | tbase

tbase | tbase

tbase\_schema | tbase

(3 rows)

--搜索路径只配置为"$user", public，其中"$user"为当前用户名，即上面的current\_user值“tbase”

postgres=# show search\_path ;

search\_path

-----------------

"$user", public

(1 row)

--不指定模式创建数据表，则该表存放于第一个搜索模式下面

postgres=# create table t(id int,mc text);

NOTICE: Replica identity is needed for shard table, please add to this table through "alter table" command.

CREATE TABLE

postgres=# \dt

List of relations

Schema | Name | Type | Owner

--------+------+-------+-------

tbase | t | table | tbase

(1 row)

--指定表位于某个模式下，不同模式下表名可以相同哦

postgres=# create table public.t(id int,mc text);

NOTICE: Replica identity is needed for shard table, please add to this table through "alter table" command.

CREATE TABLE

postgres=# \dt public.t

List of relations

Schema | Name | Type | Owner

--------+------+-------+-------

public | t | table | tbase

(1 row)

postgres=# create table tbase\_schema.t1(id int,mc text);

NOTICE: Replica identity is needed for shard table, please add to this table through "alter table" command.

CREATE TABLE

postgres=#

--访问不在搜索路径对象时，需要写全路径

postgres=# select \* from t1;

ERROR: relation "t1" does not exist

LINE 1: select \* from t1;

^

postgres=# select \* from tbase\_schema.t1;

id | mc

----+----

(0 rows)

上面出错就是因为模式tbase\_schema没有配置在search\_path搜索路径中

### 8.3.3、创建和删除数据表

#### 8.3.3.1、不用指定shard key 建表方式

不指定shard key建表方法，系统默认使用第一个字段做为表的shard key

postgres=# create table t\_first\_col\_share(id serial not null,nickname text);

CREATE TABLE

postgres=# \d+ t\_first\_col\_share

Table "public.t\_first\_col\_share"

Column | Type | Modifiers | Storage | Stats target | Description

----------+---------+----------------------------------------------------------------+----------+--------------+-------------

id | integer | not null default nextval('t\_first\_col\_share\_id\_seq'::regclass) | plain | |

nickname | text | | extended | |

Has OIDs: no

Distribute By SHARD(id)

Location Nodes: ALL DATANODES

#### 8.3.3.2、指定shard key 建表方式

postgres=# create table t\_appoint\_col(id serial not null,nickname text) distribute by shard(nickname);

CREATE TABLE

postgres=# \d+ t\_appoint\_col

Table "public.t\_appoint\_col"

Column | Type | Modifiers | Storage | Stats target | Description

----------+---------+------------------------------------------------------------+----------+--------------+-------------

id | integer | not null default nextval('t\_appoint\_col\_id\_seq'::regclass) | plain | |

nickname | text | | extended | |

Has OIDs: no

Distribute By SHARD(nickname)

Location Nodes: ALL DATANODES

#### 8.3.3.3、指定group 建表方式

postgres=# create table t (id integer,nc text) distribute by shard (id) to group default\_group;

CREATE TABLE

postgres=# \d+ t

Table "public.t"

Column | Type | Modifiers | Storage | Stats target | Description

--------+---------+-----------+----------+--------------+-------------

id | integer | | plain | |

nc | text | | extended | |

Has OIDs: no

Distribute By SHARD(id)

Location Nodes: ALL DATANODES

#### 8.3.3.4、创建范围分区表

postgres=#create table t\_range (f1 bigint,f2 timestamp default now(), f3 integer) partition by range (f3) begin (1) step (50) partitions (3) distribute by shard(f1) to group default\_group;

CREATE TABLE

postgres=# insert into t\_range(f1,f3) values(1,1),(2,50),(3,100),(2,110);

INSERT 0 4

1.36版本前数字分区只能支持integer类型，请不要使用bigint和smalint，否则后面无法进行trucate分区表操作。大家猜一下这些数据是怎样分布的

postgres=# insert into t\_range(f1,f3) values(1,151);

这样的语句执行会出错,提示超出范围

postgres=# insert into t\_range(f1,f3) values(1,151);

ERROR: node:16385, error inserted value is not in range of partitioned table, please check the value of paritition key

解决办法再扩展一些分区后就可以使用了

postgres=# ALTER TABLE t\_range ADD PARTITIONS 2;

ALTER TABLE

postgres=# insert into t\_range(f1,f3) values(1,151);

INSERT 0 1

postgres=#

#### 8.3.3.5、创建时间范围分区表

postgres=# create table t\_time\_range(f1 bigint, f2 timestamp ,f3 bigint) partition by range (f2) begin (timestamp without time zone '2017-09-01 0:0:0') step (interval '1 month') partitions (12) distribute by shard(f1) to group default\_group;

CREATE TABLE

postgres=# \d+ t\_time\_range

Table "public.t\_time\_range"

Column | Type | Modifiers | Storage | Stats target | Description

--------+-----------------------------+-----------+---------+--------------+-------------

f1 | bigint | | plain | |

f2 | timestamp without time zone | | plain | |

f3 | bigint | | plain | |

Has OIDs: no

Distribute By SHARD(f1)

Location Nodes: ALL DATANODES

Partition By: RANGE(f2)

# Of Partitions: 12

Start With: 2017-09-01

Interval Of Partition: 1 MONTH

#### 8.3.3.6、逻辑分区表

##### 8.3.3.6.1、range分区表

* 创建主分区

postgres=# create table t\_native\_range (f1 bigint,f2 timestamp default now(), f3 integer) partition by range ( f2 ) distribute by shard(f1) to group default\_group;

NOTICE: Replica identity is needed for shard table, please add to this table through "alter table" command.

CREATE TABLE

* 建立两个子表

postgres=# create table t\_native\_range\_201709 partition of t\_native\_range (f1 ,f2 , f3 ) for values from ('2017-09-01') to ('2017-10-01');

NOTICE: Replica identity is needed for shard table, please add to this table through "alter table" command.

CREATE TABLE

postgres=# create table t\_native\_range\_201710 partition of t\_native\_range (f1 ,f2 , f3 ) for values from ('2017-10-01') to ('2017-11-01');

NOTICE: Replica identity is needed for shard table, please add to this table through "alter table" command.

CREATE TABLE

postgres=#

* 查看表结构

postgres=# \d+ t\_native\_range

Table "tbase.t\_native\_range"

Column | Type | Collation | Nullable | Default | Storage | Stats target | Description

--------+-----------------------------+-----------+----------+---------+---------+--------------+-------------

f1 | bigint | | | | plain | |

f2 | timestamp without time zone | | | now() | plain | |

f3 | integer | | | | plain | |

Partition key: RANGE (f2)

Partitions: t\_native\_range\_201709 FOR VALUES FROM ('2017-09-01 00:00:00') TO ('2017-10-01 00:00:00'),

t\_native\_range\_201710 FOR VALUES FROM ('2017-10-01 00:00:00') TO ('2017-11-01 00:00:00')

Distribute By: SHARD(f1)

Location Nodes: ALL DATANODES

##### 8.3.3.6.2、list分区表

* 创建主分区

postgres=# create table t\_native\_list(f1 bigserial not null,f2 text, f3 integer,f4 date) partition by list( f2 ) distribute by shard(f1) to group default\_group;

NOTICE: Replica identity is needed for shard table, please add to this table through "alter table" command.

CREATE TABLE

* 建立两个子表,分别存入“广东”和“北京”

postgres=# create table t\_list\_gd partition of t\_native\_list(f1 ,f2 , f3,f4) for values in ('广东');

NOTICE: Replica identity is needed for shard table, please add to this table through "alter table" command.

CREATE TABLE

postgres=# create table t\_list\_bj partition of t\_native\_list(f1 ,f2 , f3,f4) for values in ('北京');

NOTICE: Replica identity is needed for shard table, please add to this table through "alter table" command.

CREATE TABLE

* 查看表结构

postgres=# \d+ t\_native\_list

Table "tbase.t\_native\_list"

Column | Type | Collation | Nullable | Default | Storage | Stats target | Description

--------+---------+-----------+----------+-------------------------------------------+----------+--------------+-------------

f1 | bigint | | not null | nextval('t\_native\_list\_f1\_seq'::regclass) | plain | |

f2 | text | | | | extended | |

f3 | integer | | | | plain | |

f4 | date | | | | plain | |

Partition key: LIST (f2)

Partitions: t\_list\_bj FOR VALUES IN ('北京'),

t\_list\_gd FOR VALUES IN ('广东')

Distribute By: SHARD(f1)

Location Nodes: ALL DATANODES

postgres=#

##### 8.3.3.6.2、多级分区表

* 创建主表

postgres=# create table t\_native\_mul\_list(f1 bigserial not null,f2 integer,f3 text,f4 text, f5 date) partition by list ( f3 ) distribute by shard(f1) to group default\_group;

NOTICE: Replica identity is needed for shard table, please add to this table through "alter table" command.

CREATE TABLE

* 创建二级表

postgres=# create table t\_native\_mul\_list\_gd partition of t\_native\_mul\_list for values in ('广东') partition by range(f5);

NOTICE: Replica identity is needed for shard table, please add to this table through "alter table" command.

CREATE TABLE

postgres=# create table t\_native\_mul\_list\_bj partition of t\_native\_mul\_list for values in ('北京') partition by range(f5);

NOTICE: Replica identity is needed for shard table, please add to this table through "alter table" command.

CREATE TABLE

postgres=# create table t\_native\_mul\_list\_sh partition of t\_native\_mul\_list for values in ('上海');

NOTICE: Replica identity is needed for shard table, please add to this table through "alter table" command.

CREATE TABLE

* 创建三级表

postgres=# create table t\_native\_mul\_list\_gd\_201701 partition of t\_native\_mul\_list\_gd(f1,f2,f3,f4,f5) for values from ('2017-01-01') to ('2017-02-01');

NOTICE: Replica identity is needed for shard table, please add to this table through "alter table" command.

CREATE TABLE

postgres=# create table t\_native\_mul\_list\_gd\_201702 partition of t\_native\_mul\_list\_gd(f1,f2,f3,f4,f5) for values from ('2017-02-01') to ('2017-03-01');

NOTICE: Replica identity is needed for shard table, please add to this table through "alter table" command.

CREATE TABLE

postgres=# create table t\_native\_mul\_list\_bj\_201701 partition of t\_native\_mul\_list\_bj(f1,f2,f3,f4,f5) for values from ('2017-01-01') to ('2017-02-01');

NOTICE: Replica identity is needed for shard table, please add to this table through "alter table" command.

CREATE TABLE

postgres=# create table t\_native\_mul\_list\_bj\_201702 partition of t\_native\_mul\_list\_bj(f1,f2,f3,f4,f5) for values from ('2017-02-01') to ('2017-03-01');

NOTICE: Replica identity is needed for shard table, please add to this table through "alter table" command.

CREATE TABLE

TBase支持存在1级和2级分区混用，大家不需要都平级。

#### 8.3.3.7、复制表

复制表是所有dn节点都存储一份相同的数据

postgres=# create table t\_rep (id int,mc text) distribute by replication to group default\_group;

CREATE TABLE

postgres=# insert into t\_rep values(1,'TBase'),(2,'pgxz');

INSERT 0 2

postgres=# EXECUTE DIRECT ON (dn001) 'select \* from t\_rep';

id | mc

----+-------

1 | TBase

2 | pgxz

(2 rows)

postgres=# EXECUTE DIRECT ON (dn002) 'select \* from t\_rep';

id | mc

----+-------

1 | TBase

2 | pgxz

(2 rows)

我们可以看到所有节点都保存了一份相同的数据。

#### 8.3.3.8、分区表性能测试

##### 8.3.3.8.1、逻辑分区表range分区表

postgres=# \d+ t\_native\_range

Table "tbase.t\_native\_range"

Column | Type | Collation | Nullable | Default | Storage | Stats target | Description

--------+-----------------------------+-----------+----------+---------+---------+--------------+-------------

f1 | bigint | | | | plain | |

f2 | timestamp without time zone | | | now() | plain | |

f3 | integer | | | | plain | |

Partition key: RANGE (f2)

Partitions: t\_native\_range\_201709 FOR VALUES FROM ('2017-09-01 00:00:00') TO ('2017-10-01 00:00:00'),

t\_native\_range\_201710 FOR VALUES FROM ('2017-10-01 00:00:00') TO ('2017-11-01 00:00:00'),

t\_native\_range\_201711 FOR VALUES FROM ('2017-11-01 00:00:00') TO ('2017-12-01 00:00:00'),

t\_native\_range\_201712 FOR VALUES FROM ('2017-12-01 00:00:00') TO ('2018-01-01 00:00:00'),

t\_native\_range\_201801 FOR VALUES FROM ('2018-01-01 00:00:00') TO ('2018-02-01 00:00:00'),

t\_native\_range\_201802 FOR VALUES FROM ('2018-02-01 00:00:00') TO ('2018-03-01 00:00:00'),

t\_native\_range\_201803 FOR VALUES FROM ('2018-03-01 00:00:00') TO ('2018-04-01 00:00:00'),

t\_native\_range\_201804 FOR VALUES FROM ('2018-04-01 00:00:00') TO ('2018-05-01 00:00:00'),

t\_native\_range\_201805 FOR VALUES FROM ('2018-05-01 00:00:00') TO ('2018-06-01 00:00:00'),

t\_native\_range\_201806 FOR VALUES FROM ('2018-06-01 00:00:00') TO ('2018-07-01 00:00:00'),

t\_native\_range\_201807 FOR VALUES FROM ('2018-07-01 00:00:00') TO ('2018-08-01 00:00:00'),

t\_native\_range\_201808 FOR VALUES FROM ('2018-08-01 00:00:00') TO ('2018-09-01 00:00:00')

Distribute By: SHARD(f1)

Location Nodes: ALL DATANODES

[tbase@VM\_0\_37\_centos pgbench]$ pgbench -h 127.0.0.1 -p 11008 -d postgres -U tbase -c 4 -j 1 -n -M prepared -T 60 -r -f insert\_t\_native\_range.sql > insert\_t\_native\_range.log 2>&1

[tbase@VM\_0\_37\_centos pgbench]$ tail insert\_t\_native\_range.log

duration: 60 s

number of transactions actually processed: 13220

latency average = 18.162 ms

tps = 220.241817 (including connections establishing)

tps = 220.260935 (excluding connections establishing)

script statistics:

- statement latencies in milliseconds:

0.013 \set f1 random(1, 10000000)

0.006 \set f2 random(1, 364)

18.130 insert into t\_native\_range values(:f1,('2017-09-01'::date+:f2::integer)::timestamp,:f1);

##### 8.3.3.8.2、时间范围分区表

postgres=# \d+ t\_time\_range

Table "tbase.t\_time\_range"

Column | Type | Collation | Nullable | Default | Storage | Stats target | Description

--------+-----------------------------+-----------+----------+---------+---------+--------------+-------------

f1 | bigint | | | | plain | |

f2 | timestamp without time zone | | | | plain | |

f3 | bigint | | | | plain | |

Distribute By: SHARD(f1)

Location Nodes: ALL DATANODES

Partition By: RANGE(f2)

# Of Partitions: 12

Start With: 2017-09-01

Interval Of Partition: 1 MONTH

postgres=#

[tbase@VM\_0\_37\_centos pgbench]$ pgbench -h 127.0.0.1 -p 11008 -d postgres -U tbase -c 4 -j 1 -n -M prepared -T 60 -r -f insert\_t\_time\_range.sql > insert\_t\_time\_range.log 2>&1

[tbase@VM\_0\_37\_centos pgbench]$ tail -20 insert\_t\_time\_range.log

client 3 receiving

client 0 receiving

client 1 receiving

client 2 receiving

pghost: 127.0.0.1 pgport: 11008 nclients: 4 duration: 60 dbName: postgres

transaction type: insert\_t\_time\_range.sql

scaling factor: 1

query mode: prepared

number of clients: 4

number of threads: 1

duration: 60 s

number of transactions actually processed: 55464

latency average = 4.328 ms

tps = 924.229841 (including connections establishing)

tps = 924.343708 (excluding connections establishing)

script statistics:

- statement latencies in milliseconds:

0.011 \set f1 random(1, 10000000)

0.006 \set f2 random(1, 364)

4.307 insert into t\_time\_range values(:f1,('2017-09-01'::date+:f2::integer)::timestamp,:f1);

#### 8.3.3.9、使用IF NOT EXISTS

带IF NOT EXISTS关键字作用表示表不存在时才创建

postgres=# create table t(id int,mc text);

CREATE TABLE

postgres=# create table t(id int,mc text);

ERROR: relation "t" already exists

postgres=# create table IF NOT EXISTS t(id int,mc text);

NOTICE: relation "t" already exists, skipping

CREATE TABLE

postgres=#

#### 8.3.3.10、指定模式创建表

postgres=# create table public.t(id int,mc text);

NOTICE: Replica identity is needed for shard table, please add to this table through "alter table" command.

CREATE TABLE

#### 8.3.3.11、使用将查询结果创建数据表

postgres=# create table t(id int,mc text) distribute by shard(mc);

NOTICE: Replica identity is needed for shard table, please add to this table through "alter table" command.

CREATE TABLE

postgres=# insert into t values(1,'tbase');

INSERT 0 1

postgres=# create table t\_as as select \* from t;

NOTICE: Replica identity is needed for shard table, please add to this table through "alter table" command.

INSERT 0 1

postgres=# select \* from t\_as;

id | mc

----+-------

1 | tbase

(1 row)

postgres=# \d+ t

Table "tbase.t"

Column | Type | Collation | Nullable | Default | Storage | Stats target | Description

--------+---------+-----------+----------+---------+----------+--------------+-------------

id | integer | | | | plain | |

mc | text | | | | extended | |

Distribute By: SHARD(mc)

Location Nodes: ALL DATANODES

postgres=# \d+ t\_as

Table "tbase.t\_as"

Column | Type | Collation | Nullable | Default | Storage | Stats target | Description

--------+---------+-----------+----------+---------+----------+--------------+-------------

id | integer | | | | plain | |

mc | text | | | | extended | |

Distribute By: SHARD(id)

Location Nodes: ALL DATANODES

postgres=#

#### 8.3.3.12、删除数据表

--删除当前模式下的数据表

postgres=# drop table t;

DROP TABLE

--删除某个模式下数据表

postgres=# drop table public.t;

DROP TABLE

--删除数据表，不存在时不执行，不报错

postgres=# drop table IF EXISTS t;

NOTICE: table "t" does not exist, skipping

DROP TABLE

--使用CASCADE无条件删除数据表

postgres=# create view tbase\_schema.t1\_view as select \* from tbase\_schema.t1 ;

CREATE VIEW

postgres=# drop table tbase\_schema.t1 ;

ERROR: cannot drop table tbase\_schema.t1 because other objects depend on it

DETAIL: view tbase\_schema.t1\_view depends on table tbase\_schema.t1

HINT: Use DROP ... CASCADE to drop the dependent objects too.

postgres=# drop table tbase\_schema.t1 CASCADE;

NOTICE: drop cascades to view tbase\_schema.t1\_view

DROP TABLE

postgres=#

### 8.3.4、创建和删除索引

#### 8.3.4.1、普通索引

postgres=# create index t\_appoint\_id\_idx on t\_appoint\_col using btree(id);

CREATE INDEX

#### 8.3.4.2、唯一索引

创建唯一索引

postgres=# create unique index t\_first\_col\_share\_id\_uidx on t\_first\_col\_share using btree(id);

CREATE INDEX

非shard key字段不能建立唯一索引

postgres=# create unique index t\_first\_col\_share\_nickname\_uidx on t\_first\_col\_share using btree(nickname);

ERROR: Unique index of partitioned table must contain the hash/modulo distribution column.

#### 8.3.4.3、表达式索引

postgres=# create table t\_upper(id int,mc text);

NOTICE: Replica identity is needed for shard table, please add to this table through "alter table" command.

CREATE TABLE

postgres=# create index t\_upper\_mc on t\_upper(mc);

CREATE INDEX

postgres=# insert into t\_upper select t,md5(t::text) from generate\_series(1,10000) as t;

INSERT 0 10000

postgres=# analyze t\_upper;

ANALYZE

postgres=# explain select \* from t\_upper where upper(mc)=md5('1');

QUERY PLAN

------------------------------------------------------------------------

Remote Fast Query Execution (cost=0.00..0.00 rows=0 width=0)

Node/s: dn001, dn002

-> Seq Scan on t\_upper (cost=0.00..135.58 rows=25 width=37)

Filter: (upper(mc) = 'c4ca4238a0b923820dcc509a6f75849b'::text)

(4 rows)

postgres=# create index t\_upper\_mc on t\_upper(upper(mc));

CREATE INDEX

postgres=# explain select \* from t\_upper where upper(mc)=md5('1');

QUERY PLAN

----------------------------------------------------------------------------------

Remote Fast Query Execution (cost=0.00..0.00 rows=0 width=0)

Node/s: dn001, dn002

-> Bitmap Heap Scan on t\_upper (cost=2.48..32.43 rows=25 width=37)

Recheck Cond: (upper(mc) = 'c4ca4238a0b923820dcc509a6f75849b'::text)

-> Bitmap Index Scan on t\_upper\_mc (cost=0.00..2.47 rows=25 width=0)

Index Cond: (upper(mc) = 'c4ca4238a0b923820dcc509a6f75849b'::text)

(6 rows)

#### 8.3.4.4、条件索引

postgres=# create table t\_sex(id int,sex text) ;

NOTICE: Replica identity is needed for shard table, please add to this table through "alter table" command.

CREATE TABLE

postgres=# create index t\_sex\_sex\_idx on t\_sex (sex);

CREATE INDEX

postgres=# insert into t\_sex select t,'男' from generate\_series(1,1000000) as t;

INSERT 0 1000000

postgres=# insert into t\_sex select t,'女' from generate\_series(1,100) as t;

INSERT 0 100

postgres=# analyze t\_sex ;

ANALYZE

postgres=#

postgres=# explain select \* from t\_sex where sex ='女';

QUERY PLAN

----------------------------------------------------------------------------------

Remote Fast Query Execution (cost=0.00..0.00 rows=0 width=0)

Node/s: dn001, dn002

-> Index Scan using t\_sex\_sex\_idx on t\_sex (cost=0.42..5.81 rows=67 width=8)

Index Cond: (sex = '女'::text)

(4 rows)

#索引对于条件为男的情况下无效

postgres=# explain select \* from t\_sex where sex ='男';

QUERY PLAN

-------------------------------------------------------------------

Remote Fast Query Execution (cost=0.00..0.00 rows=0 width=0)

Node/s: dn001, dn002

-> Seq Scan on t\_sex (cost=0.00..9977.58 rows=500539 width=8)

Filter: (sex = '男'::text)

(4 rows)

#连接dn节点查看索引点用空间大，而且度数也高

[tbase@VM\_0\_37\_centos shell]$ psql -p 11010

psql (PostgreSQL 10.0 TBase V2)

Type "help" for help.

postgres=# \di+

List of relations

Schema | Name | Type | Owner | Table | Size | Allocated Size | Description

--------+---------------+-------+-------+---------+-------+----------------+-------------

tbase | t\_sex\_sex\_idx | index | tbase | t\_sex | 14 MB | 14 MB |

tbase | t\_upper\_mc | index | tbase | t\_upper | 14 MB | 14 MB |

(2 rows)

postgres=# \q

[tbase@VM\_0\_37\_centos shell]$ psql

psql (PostgreSQL 10.0 TBase V2)

Type "help" for help.

postgres=# drop index t\_sex\_sex\_idx;

DROP INDEX

postgres=# create index t\_sex\_sex\_idx on t\_sex (sex) where sex='女';

CREATE INDEX

postgres=# analyze t\_sex;

ANALYZE

postgres=# explain select \* from t\_sex where sex ='女';

QUERY PLAN

----------------------------------------------------------------------------------

Remote Fast Query Execution (cost=0.00..0.00 rows=0 width=0)

Node/s: dn001, dn002

-> Index Scan using t\_sex\_sex\_idx on t\_sex (cost=0.14..6.69 rows=33 width=8)

(3 rows)

postgres=# explain select \* from t\_sex where sex ='男';

QUERY PLAN

-------------------------------------------------------------------

Remote Fast Query Execution (cost=0.00..0.00 rows=0 width=0)

Node/s: dn001, dn002

-> Seq Scan on t\_sex (cost=0.00..9977.58 rows=500573 width=8)

Filter: (sex = '男'::text)

(4 rows)

postgres=# \q

[tbase@VM\_0\_37\_centos shell]$ psql -p 11010

psql (PostgreSQL 10.0 TBase V2)

Type "help" for help.

postgres=# \di+

List of relations

Schema | Name | Type | Owner | Table | Size | Allocated Size | Description

--------+---------------+-------+-------+---------+-------+----------------+-------------

tbase | t\_sex\_sex\_idx | index | tbase | t\_sex | 16 kB | 16 kB |

tbase | t\_upper\_mc | index | tbase | t\_upper | 14 MB | 14 MB |

(2 rows)

postgres=#

#### 8.3.4.5、gist索引

postgres=# create table t\_trgm (id int,trgm text,no\_trgm text) ;

NOTICE: Replica identity is needed for shard table, please add to this table through "alter table" command.

CREATE TABLE

postgres=# create index t\_trgm\_trgm\_idx on t\_trgm using gist(trgm gist\_trgm\_ops);

CREATE INDEX

#### 8.3.4.6、gin索引

* pg\_trgm索引

postgres=# drop index t\_trgm\_trgm\_idx;

DROP INDEX

Time: 55.954 ms

postgres=# create index t\_trgm\_trgm\_idx on t\_trgm using gin(trgm gin\_trgm\_ops);

CREATE INDEX

* jsonb索引

postgres=# create table t\_jsonb(id int,f\_jsonb json);

NOTICE: Replica identity is needed for shard table, please add to this table through "alter table" command.

CREATE TABLE

postgres=# create index t\_jsonb\_f\_jsonb\_idx on t\_jsonb using gin(f\_jsonb);

CREATE INDEX

* 数组索引

postgres=# create table t\_array(id int, mc text[]);

NOTICE: Replica identity is needed for shard table, please add to this table through "alter table" command.

CREATE TABLE

postgres=# insert into t\_array select t,('{'||md5(t::text)||'}')::text[] from generate\_series(1,1000000) as t;

INSERT 0 1000000

postgres=# analyze;

ANALYZE

postgres=# \timing

Timing is on.

postgres=# explain select \* from t\_array where mc @> ('{'||md5('1')||'}')::text[];

QUERY PLAN

---------------------------------------------------------------------------------------

Remote Fast Query Execution (cost=0.00..0.00 rows=0 width=0)

Node/s: dn001, dn002

-> Gather (cost=1000.00..12060.25 rows=2503 width=61)

Workers Planned: 2

-> Parallel Seq Scan on t\_array (cost=0.00..10809.95 rows=1043 width=61)

Filter: (mc @> ('{c4ca4238a0b923820dcc509a6f75849b}'::cstring)::text[])

(6 rows)

Time: 4.105 ms

postgres=# select \* from t\_array where mc @> ('{'||md5('1')||'}')::text[];

id | mc

----+------------------------------------

1 | {c4ca4238a0b923820dcc509a6f75849b}

(1 row)

Time: 494.371 ms

postgres=# create index t\_array\_mc\_idx on t\_array using gin(mc);

CREATE INDEX

Time: 8195.387 ms (00:08.195)

postgres=# explain select \* from t\_array where mc @> ('{'||md5('1')||'}')::text[];

QUERY PLAN

-------------------------------------------------------------------------------------------

Remote Fast Query Execution (cost=0.00..0.00 rows=0 width=0)

Node/s: dn001, dn002

-> Bitmap Heap Scan on t\_array (cost=29.40..3172.64 rows=2503 width=61)

Recheck Cond: (mc @> ('{c4ca4238a0b923820dcc509a6f75849b}'::cstring)::text[])

-> Bitmap Index Scan on t\_array\_mc\_idx (cost=0.00..28.78 rows=2503 width=0)

Index Cond: (mc @> ('{c4ca4238a0b923820dcc509a6f75849b}'::cstring)::text[])

(6 rows)

Time: 1.716 ms

postgres=# select \* from t\_array where mc @> ('{'||md5('1')||'}')::text[];

id | mc

----+------------------------------------

1 | {c4ca4238a0b923820dcc509a6f75849b}

(1 row)

Time: 2.980 ms

* Btree\_gin任意字段索引

postgres=# create table gin\_mul(f1 int, f2 int, f3 timestamp, f4 text, f5 numeric, f6 text);

NOTICE: Replica identity is needed for shard table, please add to this table through "alter table" command.

CREATE TABLE

postgres=#

postgres=# insert into gin\_mul select random()\*5000, random()\*6000, now()+((30000-60000\*random())||' sec')::interval , md5(random()::text), round((random()\*100000)::numeric,2), md5(random()::text) from generate\_series(1,1000000);

INSERT 0 1000000

postgres=# create extension btree\_gin;

CREATE EXTENSION

postgres=# create index gin\_mul\_gin\_idx on gin\_mul using gin(f1,f2,f3,f4,f5,f6);

CREATE INDEX

#单字段查询

postgres=# explain select \* from gin\_mul where f1=10;

QUERY PLAN

---------------------------------------------------------------------------------------

Remote Fast Query Execution (cost=0.00..0.00 rows=0 width=0)

Node/s: dn002

-> Bitmap Heap Scan on gin\_mul (cost=11.51..369.70 rows=194 width=90)

Recheck Cond: (f1 = 10)

-> Bitmap Index Scan on gin\_mul\_gin\_idx (cost=0.00..11.46 rows=194 width=0)

Index Cond: (f1 = 10)

(6 rows)

postgres=#

postgres=# explain select \* from gin\_mul where f3='2019-02-18 23:01:01';

QUERY PLAN

-------------------------------------------------------------------------------------

Remote Fast Query Execution (cost=0.00..0.00 rows=0 width=0)

Node/s: dn001, dn002

-> Bitmap Heap Scan on gin\_mul (cost=10.01..12.02 rows=1 width=90)

Recheck Cond: (f3 = '2019-02-18 23:01:01'::timestamp without time zone)

-> Bitmap Index Scan on gin\_mul\_gin\_idx (cost=0.00..10.01 rows=1 width=0)

Index Cond: (f3 = '2019-02-18 23:01:01'::timestamp without time zone)

(6 rows)

postgres=# explain select \* from gin\_mul where f4='2364d9969c8b66402c9b7d17a6d5b7d3';

QUERY PLAN

-------------------------------------------------------------------------------------

Remote Fast Query Execution (cost=0.00..0.00 rows=0 width=0)

Node/s: dn001, dn002

-> Bitmap Heap Scan on gin\_mul (cost=10.01..12.02 rows=1 width=90)

Recheck Cond: (f4 = '2364d9969c8b66402c9b7d17a6d5b7d3'::text)

-> Bitmap Index Scan on gin\_mul\_gin\_idx (cost=0.00..10.01 rows=1 width=0)

Index Cond: (f4 = '2364d9969c8b66402c9b7d17a6d5b7d3'::text)

(6 rows)

postgres=# explain select \* from gin\_mul where f5=85375.30;

QUERY PLAN

-------------------------------------------------------------------------------------

Remote Fast Query Execution (cost=0.00..0.00 rows=0 width=0)

Node/s: dn001, dn002

-> Bitmap Heap Scan on gin\_mul (cost=10.01..12.02 rows=1 width=90)

Recheck Cond: (f5 = 85375.30)

-> Bitmap Index Scan on gin\_mul\_gin\_idx (cost=0.00..10.01 rows=1 width=0)

Index Cond: (f5 = 85375.30)

(6 rows)

#二个字段组合

postgres=# explain select \* from gin\_mul where f1=2 and f3='2019-02-18 16:59:52.872523';

QUERY PLAN

-----------------------------------------------------------------------------------------------------------

Remote Fast Query Execution (cost=0.00..0.00 rows=0 width=0)

Node/s: dn001

-> Bitmap Heap Scan on gin\_mul (cost=18.00..20.02 rows=1 width=90)

Recheck Cond: ((f1 = 2) AND (f3 = '2019-02-18 16:59:52.872523'::timestamp without time zone))

-> Bitmap Index Scan on gin\_mul\_gin\_idx (cost=0.00..18.00 rows=1 width=0)

Index Cond: ((f1 = 2) AND (f3 = '2019-02-18 16:59:52.872523'::timestamp without time zone))

(6 rows)

#三字段组合查询

postgres=# explain select \* from gin\_mul where f1=2 and f3='2019-02-18 16:59:52.872523' and f6='fa627dc16c2bd026150afa0453a0991d';

QUERY PLAN

---------------------------------------------------------------------------------------------------------------------------------------------------------------

Remote Fast Query Execution (cost=0.00..0.00 rows=0 width=0)

Node/s: dn001

-> Bitmap Heap Scan on gin\_mul (cost=26.00..28.02 rows=1 width=90)

Recheck Cond: ((f1 = 2) AND (f3 = '2019-02-18 16:59:52.872523'::timestamp without time zone) AND (f6 = 'fa627dc16c2bd026150afa0453a0991d'::text))

-> Bitmap Index Scan on gin\_mul\_gin\_idx (cost=0.00..26.00 rows=1 width=0)

Index Cond: ((f1 = 2) AND (f3 = '2019-02-18 16:59:52.872523'::timestamp without time zone) AND (f6 = 'fa627dc16c2bd026150afa0453a0991d'::text))

(6 rows)

postgres=#

#### 8.3.4.7、多字段索引

postgres=# create table t\_mul\_idx (f1 int,f2 int,f3 int,f4 int);

NOTICE: Replica identity is needed for shard table, please add to this table through "alter table" command.

CREATE TABLE

Time: 308.109 ms

postgres=# create index t\_mul\_idx\_idx on t\_mul\_idx(f1,f2,f3);

CREATE INDEX

Time: 108.734 ms

多字段使用注意事项

* or查询条件bitmap scan最多支持两个不同字段条件

postgres=# insert into t\_mul\_idx select t,t,t,t from generate\_series(1,1000000) as t;

INSERT 0 1000000

postgres=# analyze ;

ANALYZE

postgres=# explain select \* from t\_mul\_idx where f1=1 or f2=2 ;

QUERY PLAN

-------------------------------------------------------------------------------------------

Remote Fast Query Execution (cost=0.00..0.00 rows=0 width=0)

Node/s: dn001, dn002

-> Bitmap Heap Scan on t\_mul\_idx (cost=7617.08..7621.07 rows=2 width=16)

Recheck Cond: ((f1 = 1) OR (f2 = 2))

-> BitmapOr (cost=7617.08..7617.08 rows=2 width=0)

-> Bitmap Index Scan on t\_mul\_idx\_idx (cost=0.00..2.43 rows=1 width=0)

Index Cond: (f1 = 1)

-> Bitmap Index Scan on t\_mul\_idx\_idx (cost=0.00..7614.65 rows=1 width=0)

Index Cond: (f2 = 2)

(9 rows)

Time: 3.655 ms

postgres=# explain select \* from t\_mul\_idx where f1=1 or f2=2 or f1=3 ;

QUERY PLAN

-------------------------------------------------------------------------------------------

Remote Fast Query Execution (cost=0.00..0.00 rows=0 width=0)

Node/s: dn001, dn002

-> Bitmap Heap Scan on t\_mul\_idx (cost=7619.51..7625.49 rows=3 width=16)

Recheck Cond: ((f1 = 1) OR (f2 = 2) OR (f1 = 3))

-> BitmapOr (cost=7619.51..7619.51 rows=3 width=0)

-> Bitmap Index Scan on t\_mul\_idx\_idx (cost=0.00..2.43 rows=1 width=0)

Index Cond: (f1 = 1)

-> Bitmap Index Scan on t\_mul\_idx\_idx (cost=0.00..7614.65 rows=1 width=0)

Index Cond: (f2 = 2)

-> Bitmap Index Scan on t\_mul\_idx\_idx (cost=0.00..2.43 rows=1 width=0)

Index Cond: (f1 = 3)

(11 rows)

Time: 3.429 ms

postgres=# explain select \* from t\_mul\_idx where f1=1 or f2=2 or f3=3 ;

QUERY PLAN

--------------------------------------------------------------------

Remote Fast Query Execution (cost=0.00..0.00 rows=0 width=0)

Node/s: dn001, dn002

-> Seq Scan on t\_mul\_idx (cost=0.00..12979.87 rows=3 width=16)

Filter: ((f1 = 1) OR (f2 = 2) OR (f3 = 3))

(4 rows)

Time: 1.679 ms

* 如果返回字段全部在索引文件中，则只需要扫描索引，io开销会更少

postgres=# explain select f1,f2,f3 from t\_mul\_idx where f1=1 ;

QUERY PLAN

-------------------------------------------------------------------------------------------

Remote Fast Query Execution (cost=0.00..0.00 rows=0 width=0)

Node/s: dn001

-> Index Only Scan using t\_mul\_idx\_idx on t\_mul\_idx (cost=0.42..4.44 rows=1 width=12)

Index Cond: (f1 = 1)

(4 rows)

Time: 1.564 ms

* 更新性能比单字段多索引文件要好

--多字段

postgres=# insert into t\_simple\_idx select t,t,t,t from generate\_series(1,1000000) as t;

INSERT 0 1000000

Time: 7143.754 ms (00:07.144)

--单字段

postgres=# insert into t\_mul\_idx select t,t,t,t from generate\_series(1,1000000) as t;

INSERT 0 1000000

Time: 4034.208 ms (00:04.034)

* 多字段索引走非第一字段查询时性能比独立的单字段差

--多字段

postgres=# select \* from t\_mul\_idx where f1=1;

f1 | f2 | f3 | f4

----+----+----+----

1 | 1 | 1 | 1

(1 row)

Time: 1.769 ms

postgres=# select \* from t\_mul\_idx where f2=1;

f1 | f2 | f3 | f4

----+----+----+----

1 | 1 | 1 | 1

(1 row)

Time: 25.423 ms

postgres=# select \* from t\_mul\_idx where f3=1;

f1 | f2 | f3 | f4

----+----+----+----

1 | 1 | 1 | 1

(1 row)

Time: 27.791 ms

--独立字段

postgres=# select \* from t\_simple\_idx where f1=1;

f1 | f2 | f3 | f4

----+----+----+----

1 | 1 | 1 | 1

(1 row)

Time: 1.530 ms

postgres=# select \* from t\_simple\_idx where f2=1;

f1 | f2 | f3 | f4

----+----+----+----

1 | 1 | 1 | 1

(1 row)

Time: 2.315 ms

postgres=# select \* from t\_simple\_idx where f3=1;

f1 | f2 | f3 | f4

----+----+----+----

1 | 1 | 1 | 1

(1 row)

Time: 2.390 ms

#### 8.3.4.8、删除索引

postgres=# drop index t\_appoint\_id\_idx;

DROP INDEX

### 8.3.5、修改表结构

#### 8.3.5.1、修改表名

postgres=# alter table t rename to tbase;

ALTER TABLE

#### 8.3.5.2、给表或字段添加注释

postgres=# comment on table tbase is 'TBase分布式关系型数据库系统';

COMMENT

postgres=# \dt+

List of relations

Schema | Name | Type | Owner | Size | Description

--------+-------------------+-------+-------+-------+-----------------------------

public | t\_appoint\_col | table | pgxz | 16 kB |

public | t\_first\_col\_share | table | pgxz | 16 kB |

public | TBase | table | pgxz | 24 kB | TBase分布式关系型数据库系统

(3 rows)

postgres=# comment on column tbase.nickname is 'TBase呢称是大象';

COMMENT

postgres=# \d+ tbase

Table "public.tbase"

Column | Type | Modifiers | Storage | Stats target | Description

----------+---------+------------------------------------------------+----------+--------------+-----------------

id | integer | not null default nextval('t\_id\_seq'::regclass) | plain | |

nickname | text | | extended | | TBase呢称是大象

Has OIDs: no

Distribute By SHARD(id)

Location Nodes: ALL DATANODES

postgres=#

#### 8.3.5.3、给表增加字段

postgres=# alter table tbase add column age integer;

ALTER TABLE

postgres=# \d+ tbase

Table "public.tbase"

Column | Type | Modifiers | Storage | Stats target | Description

----------+---------+------------------------------------------------+----------+--------------+-----------------

id | integer | not null default nextval('t\_id\_seq'::regclass) | plain | |

nickname | text | | extended | | TBase呢称是大象

age | integer | | plain | |

Has OIDs: no

Distribute By SHARD(id)

Location Nodes: ALL DATANODES

#### 8.3.5.4、修改字段类型

postgres=# alter table tbase alter column age type float8;

ALTER TABLE

postgres=# \d+ tbase

Table "public.tbase"

Column | Type | Modifiers | Storage | Stats target | Description

----------+------------------+------------------------------------------------+----------+--------------+-----------------

id | integer | not null default nextval('t\_id\_seq'::regclass) | plain | |

nickname | text | | extended | | TBase呢称是大象

age | double precision | | plain | |

Has OIDs: no

Distribute By SHARD(id)

Location Nodes: ALL DATANODES

postgres=#

#### 8.3.5.5、修改字段默认值

postgres=# alter table tbase alter column age set default 0.0;

ALTER TABLE

postgres=# \d+ tbase

Table "public.tbase"

Column | Type | Modifiers | Storage | Stats target | Description

----------+------------------+------------------------------------------------+----------+--------------+-----------------

id | integer | not null default nextval('t\_id\_seq'::regclass) | plain | |

nickname | text | | extended | | TBase呢称是大象

age | double precision | default 0.0 | plain | |

Has OIDs: no

Distribute By SHARD(id)

Location Nodes: ALL DATANODES

#### 8.3.5.6、删除字段

postgres=# alter table tbase drop column age;

ALTER TABLE

postgres=# \d+ tbase

Table "public.tbase"

Column | Type | Modifiers | Storage | Stats target | Description

----------+---------+------------------------------------------------+----------+--------------+-----------------

id | integer | not null default nextval('t\_id\_seq'::regclass) | plain | |

nickname | text | | extended | | TBase呢称是大象

Has OIDs: no

Distribute By SHARD(id)

Location Nodes: ALL DATANODES

#### 8.3.5.7、添加主键

postgres=# ALTER TABLE t ADD CONSTRAINT t\_id\_pkey PRIMARY KEY (id) ;

ALTER TABLE

postgres=# \d+ t

Table "tbase.t"

Column | Type | Collation | Nullable | Default | Storage | Stats target | Description

--------+---------+-----------+----------+---------+----------+--------------+-------------

id | integer | | not null | | plain | |

mc | text | | | | extended | |

Indexes:

"t\_id\_pkey" PRIMARY KEY, btree (id)

Distribute By: SHARD(id)

Location Nodes: ALL DATANODES

#### 8.3.5.8、删除主键

postgres=# ALTER TABLE t DROP CONSTRAINT t\_id\_pkey ;

ALTER TABLE

postgres=# \d+ t

Table "tbase.t"

Column | Type | Collation | Nullable | Default | Storage | Stats target | Description

--------+---------+-----------+----------+---------+----------+--------------+-------------

id | integer | | not null | | plain | |

mc | text | | | | extended | |

Distribute By: SHARD(id)

Location Nodes: ALL DATANODES

#### 8.3.5.9、修改表所属模式

postgres=# \dt t

List of relations

Schema | Name | Type | Owner

--------+------+-------+-------

tbase | t | table | tbase

(1 row)

postgres=# alter table t set schema public;

ALTER TABLE

postgres=# \dt t

List of relations

Schema | Name | Type | Owner

--------+------+-------+-------

public | t | table | tbase

(1 row)

#### 8.3.5.10、修改表所属用户

postgres=# \dt tbase

List of relations

Schema | Name | Type | Owner

--------+-------+-------+-------

public | tbase | table | tbase

(1 row)

postgres=# alter table tbase owner to pgxz;

ALTER TABLE

postgres=# \dt tbase

List of relations

Schema | Name | Type | Owner

--------+-------+-------+-------

public | tbase | table | pgxz

(1 row)

### 8.3.6、视图创建和删除

#### 8.3.6.1、创建视图

postgres=# create view t\_range\_view as select \* from t\_range;

CREATE VIEW

postgres=# select \* from t\_range\_view;

f1 | f2 | f3 | f4

----+----------------------------+-----+----

1 | 2017-09-27 23:17:39.674318 | 1 |

2 | 2017-09-27 23:17:39.674318 | 50 |

2 | 2017-09-27 23:17:39.674318 | 110 |

1 | 2017-09-27 23:39:45.841093 | 151 |

3 | 2017-09-27 23:17:39.674318 | 100 |

(5 rows)

数据类型重定义

postgres=# create view t\_range\_view as select f1,f2::date from t\_range;

CREATE VIEW

postgres=# select \* from t\_range\_view;

f1 | f2

----+------------

1 | 2017-09-27

2 | 2017-09-27

2 | 2017-09-27

1 | 2017-09-27

3 | 2017-09-27

(5 rows)

数据类型重定义,以及取别名

postgres=# create view t\_range\_view as select f1,f2::date as mydate from t\_range;

CREATE VIEW

postgres=# select \* from t\_range\_view;

f1 | mydate

----+------------

1 | 2017-09-27

2 | 2017-09-27

2 | 2017-09-27

1 | 2017-09-27

3 | 2017-09-27

(5 rows)

tbase支持视图引用表或字段改名联动，不受影响

postgres=# \d+ t\_view

View "tbase.t\_view"

Column | Type | Collation | Nullable | Default | Storage | Description

--------+---------+-----------+----------+---------+----------+-------------

id | integer | | | | plain |

mc | text | | | | extended |

View definition:

SELECT t.id,

t.mc

FROM t;

postgres=# alter table t rename to t\_new;

ALTER TABLE

Time: 62.875 ms

postgres=# alter table t\_new rename mc to mc\_new;

ALTER TABLE

Time: 22.081 ms

postgres=# \d+ t\_view

View "tbase.t\_view"

Column | Type | Collation | Nullable | Default | Storage | Description

--------+---------+-----------+----------+---------+----------+-------------

id | integer | | | | plain |

mc | text | | | | extended |

View definition:

SELECT t\_new.id,

t\_new.mc\_new AS mc

FROM t\_new;

#### 8.3.6.2、删除视图

postgres=# drop table t;

DROP TABLE

postgres=# create table t (id int,mc text);

CREATE TABLE

postgres=# create view t\_view as select \* from t;

CREATE VIEW

postgres=# create view t\_view\_1 as select \* from t\_view;

CREATE VIEW

postgres=# create view t\_view\_2 as select \* from t\_view;

CREATE VIEW

postgres=# drop view t\_view\_2;

DROP VIEW

#使用cascade强制删除依赖对象

postgres=# drop view t\_view;

ERROR: cannot drop view t\_view because other objects depend on it

DETAIL: view t\_view\_1 depends on view t\_view

HINT: Use DROP ... CASCADE to drop the dependent objects too.

postgres=# drop view t\_view cascade;

NOTICE: drop cascades to view t\_view\_1

DROP VIEW

### 8.3.7、物化视图使用

#### 8.3.7.1、创建物化视图

postgres=# CREATE MATERIALIZED VIEW t\_range\_mv AS select f1,f2::date from t\_range;

SELECT 5

postgres=# select \* from t\_range\_mv;

f1 | f2

----+------------

1 | 2017-09-27

2 | 2017-09-27

2 | 2017-09-27

1 | 2017-09-27

3 | 2017-09-27

(5 rows)

#### 8.3.7.2、访问物化视图

postgres=# select \* from t\_range\_mv;

f1 | f2

----+------------

1 | 2017-09-27

2 | 2017-09-27

2 | 2017-09-27

1 | 2017-09-27

3 | 2017-09-27

(5 rows)

postgres=# insert into t\_range(f1,f3) values(5,10);

INSERT 0 1

postgres=# select \* from t\_range;

f1 | f2 | f3 | f4

----+----------------------------+-----+----

1 | 2017-09-27 23:17:39.674318 | 1 |

2 | 2017-09-27 23:17:39.674318 | 50 |

5 | 2017-09-27 23:50:51.576173 | 10 |

2 | 2017-09-27 23:17:39.674318 | 110 |

1 | 2017-09-27 23:39:45.841093 | 151 |

3 | 2017-09-27 23:17:39.674318 | 100 |

(6 rows)

#### 8.3.7.3、增量数据刷新

postgres=# select \* from t\_range\_mv ;

f1 | f2

----+------------

1 | 2017-09-27

2 | 2017-09-27

2 | 2017-09-27

1 | 2017-09-27

3 | 2017-09-27

(5 rows)

postgres=# REFRESH MATERIALIZED VIEW t\_range\_mv;

REFRESH MATERIALIZED VIEW

postgres=# select \* from t\_range\_mv ;

f1 | f2

----+------------

1 | 2017-09-27

2 | 2017-09-27

5 | 2017-09-27

2 | 2017-09-27

1 | 2017-09-27

3 | 2017-09-27

(6 rows)

注意：物化视图数据存储在cn节点上面，每个cn节点各有一份相同的数据

### 8.3.8、truncate操作

truncate功能用于对表数据进行快速清除，truncate属于ddl级别，会给truncate表加上ACCESS EXCLUSIVE最高级别的锁

#### 8.3.8.1、truncate普通表

postgres=# truncate table t1;

TRUNCATE TABLE

#也可以一次truncate多个数据表

postgres=# truncate table t1,t2;

TRUNCATE TABLE

postgres=#

#### 8.3.8.2、truncate分区表

* truncate一个时间分区表

postgres=# \d+ t\_time\_range

Table "pgxz.t\_time\_range"

Column | Type | Modifiers | Storage | Stats target | Description

--------+-----------------------------+-----------+----------+--------------+-------------

f1 | bigint | | plain | |

f2 | timestamp without time zone | | plain | |

f3 | character varying(20) | | extended | |

Has OIDs: no

Distribute By SHARD(f1)

Location Nodes: dn001, dn002

Partition By: RANGE(f2)

# Of Partitions: 12

Start With: 2017-09-01

Interval Of Partition: 1 MONTH

postgres=# select \* from t\_time\_range;

f1 | f2 | f3

----+---------------------+-------

1 | 2017-09-01 00:00:00 | tbase

2 | 2017-10-01 00:00:00 | pgxz

(2 rows)

postgres=# truncate t\_time\_range partition for ('2017-09-01' ::timestamp without time zone);

TRUNCATE TABLE

postgres=# select \* from t\_time\_range;

f1 | f2 | f3

----+---------------------+------

2 | 2017-10-01 00:00:00 | pgxz

(1 row)

postgres=#

* truncate一个数字分区表

postgres=# \d+ t\_range

Table "pgxz.t\_range"

Column | Type | Modifiers | Storage | Stats target | Description

--------+-----------------------------+---------------+---------+--------------+-------------

f1 | integer | | plain | |

f2 | timestamp without time zone | default now() | plain | |

f3 | integer | | plain | |

Has OIDs: no

Distribute By SHARD(f1)

Location Nodes: dn01, dn02

Partition By: RANGE(f3)

# Of Partitions: 3

Start With: 1

Interval Of Partition: 50

postgres=# select \* from t\_range ;

f1 | f2 | f3

----+----------------------------+-----

1 | 2017-12-22 11:47:39.153234 | 1

2 | 2017-12-22 11:47:39.153234 | 50

2 | 2017-12-22 11:47:39.153234 | 110

3 | 2017-12-22 11:47:39.153234 | 100

(4 rows)

postgres=# truncate t\_range partition for (1);

TRUNCATE TABLE

postgres=# select \* from t\_range ;

f1 | f2 | f3

----+----------------------------+-----

2 | 2017-12-22 11:47:39.153234 | 110

3 | 2017-12-22 11:47:39.153234 | 100

(2 rows)

postgres=#

## 8.4、select语句

### 8.4.1、访问函数

postgres=# select md5(random()::text);

md5

----------------------------------

3eb6c0c8f8355f0b0f0cad7a8f0f7491

### 8.4.2、数据排序

**--按某一列排序**

postgres=# INSERT into tbase (nickname) VALUES('TBase好牛');

INSERT 0 1

postgres=# INSERT into tbase (id,nickname) VALUES(1,'TBase分布式数据库的时代来了');

INSERT 0 1

postgres=# select \* from tbase order by id;

id | nickname

----+-----------------------------

1 | hello TBase

1 | TBase分布式数据库的时代来了

2 | TBase好牛

(3 rows)

**--按第一列排序**

postgres=# select \* from tbase order by 1;

id | nickname

----+-----------------------------

1 | hello TBase

1 | TBase分布式数据库的时代来了

2 | TBase好牛

(3 rows)

**--按id升级排序，再按nickname降序排序**

postgres=# select \* from tbase order by id,nickname desc;

id | nickname

----+-----------------------------

1 | TBase分布式数据库的时代来了

1 | hello TBase

2 | TBase好牛

(3 rows)

**--效果与上面的语句一样**

postgres=# select \* from tbase order by 1,2 desc;

id | nickname

----+-----------------------------

1 | TBase分布式数据库的时代来了

1 | hello TBase

2 | TBase好牛

(3 rows)

**--随机排序**

postgres=# select \* from tbase order by random();

id | nickname

----+-----------------------------

1 | TBase分布式数据库的时代来了

2 | TBase好牛

1 | hello TBase

(3 rows)

**--计算排序**

postgres=# select \* from tbase order by md5(nickname);

id | nickname

----+-----------------------------

2 | TBase好牛

1 | TBase分布式数据库的时代来了

1 | hello TBase

(3 rows)

**--排序都能用子查询，牛**

postgres=# select \* from tbase order by (select id from tbase order by random() limit 1);

id | nickname

----+-----------------------------

1 | hello TBase

2 | TBase好牛

1 | TBase分布式数据库的时代来了

(3 rows)

**--null值排序结果处理**

postgres=# insert into tbase values(4,null);

INSERT 0 1

null值记录排在最前面

postgres=# select \* from tbase order by nickname nulls first;

id | nickname

----+-----------------------------

4 |

1 | hello TBase

1 | TBase分布式数据库的时代来了

2 | TBase好牛

(4 rows)

null值记录排在最后

postgres=# select \* from tbase order by nickname nulls last;

id | nickname

----+-----------------------------

1 | hello TBase

1 | TBase分布式数据库的时代来了

2 | TBase好牛

4 |

(4 rows)

**--按拼音排序**

postgres=# select \* from (values ('张三'), ('李四'),('陈五')) t(myname) order by myname;

myname

--------

张三

李四

陈五

(3 rows)

如果不加处理，则按汉字的utf8编码进行排序，不符合中国人使用习惯

postgres=# select \* from (values ('张三'), ('李四'),('陈五')) t(myname) order by convert(myname::bytea,'UTF-8','GBK');

myname

--------

陈五

李四

张三

(3 rows)

使用convert函数实现汉字按拼音进行排序

postgres=# select \* from (values ('张三'), ('李四'),('陈五')) t(myname) order by convert\_to(myname,'GBK');

myname

--------

陈五

李四

张三

(3 rows)

使用convert\_to函数实现汉字按拼音进行排序

postgres=# select \* from (values ('张三'), ('李四'),('陈五')) t(myname) order by myname collate "zh\_CN.utf8";

myname

--------

陈五

李四

张三

(3 rows)

通过指定排序规则collact来实现汉字按拼音进行排序

### 8.4.3、where条件使用

**--单条件查询**

postgres=# select \* from tbase where id=1;

id | nickname

----+-----------------------------

1 | hello TBase

1 | TBase分布式数据库的时代来了

**--多条件and**

postgres=# select \* from tbase where id=1 and nickname like '%h%';

id | nickname

----+-------------

1 | hello TBase

(1 row)

**--多条件or**

postgres=# select \* from tbase where id=2 or nickname like '%h%';

id | nickname

----+-------------

1 | hello TBase

2 | TBase好牛

(2 rows)

**--ilike不区分大小写匹配**

postgres=# create table t\_ilike(id int,mc text);

NOTICE: Replica identity is needed for shard table, please add to this table through "alter table" command.

CREATE TABLE

postgres=# insert into t\_ilike values(1,'tbase'),(2,'TBase');

INSERT 0 2

postgres=# select \* from t\_ilike where mc ilike '%tb%';

id | mc

----+-------

1 | tbase

2 | TBase

(2 rows)

**--where条件也能支持子查询**

postgres=# select \* from tbase where id=(select (random()\*2)::integer from tbase order by random() limit 1);

id | nickname

----+----------

(0 rows)

postgres=# select \* from tbase where id=(select (random()\*2)::integer from tbase order by random() limit 1);

id | nickname

----+-----------------------------

1 | hello TBase

1 | TBase分布式数据库的时代来了

(2 rows)

**--null值查询方法**

postgres=# select \* from tbase where nickname is null;

id | nickname

----+----------

4 |

(1 row)

postgres=# select \* from tbase where nickname is not null;

id | nickname

----+-----------------------------

1 | hello TBase

2 | TBase好牛

1 | TBase分布式数据库的时代来了

(3 rows)

**--exists，只要有记录返回就为真**

postgres=# create table t\_exists1(id int,mc text);

NOTICE: Replica identity is needed for shard table, please add to this table through "alter table" command.

CREATE TABLE

postgres=# insert into t\_exists1 values(1,'tbase'),(2,'TBase');

INSERT 0 2

postgres=# create table t\_exists2(id int,mc text);

NOTICE: Replica identity is needed for shard table, please add to this table through "alter table" command.

CREATE TABLE

postgres=# insert into t\_exists2 values(1,'tbase'),(1,'TBase');

INSERT 0 2

postgres=# select \* from t\_exists1 where exists(select 1 from t\_exists2 where t\_exists1.id=t\_exists2.id) ;

id | mc

----+-------

1 | tbase

(1 row)

**--exists等价写法**

postgres=# select t\_exists1.\* from t\_exists1,(select distinct id from t\_exists2) as t where t\_exists1.id=t.id;;

id | mc

----+-------

1 | tbase

(1 row)

### 8.4.4、分页查询

**--默认从第一条开始，返回一条记录**

postgres=# select \* from tbase limit 1;

id | nickname

----+-------------

1 | hello TBase

(1 row)

**--使用offset指定从第几条开始，0表示第一条开如，返回1条记录**

postgres=# select \* from tbase limit 1 offset 0;

id | nickname

----+-------------

1 | hello TBase

(1 row)

**--从第3条开始，返回二条记录**

postgres=# select \* from tbase limit 1 offset 2;

id | nickname

----+-----------------------------

1 | TBase分布式数据库的时代来了

(1 row)

**--上面的语句没有使用排序，返回结果不可预知，使用order by可以获得一个有序的结果**

postgres=# select \* from tbase order by id limit 1 offset 2;

id | nickname

----+-----------

2 | TBase好牛

(1 row)

### 8.4.5、合并多个查询结果

**--不过虑重复的记录**

postgres=# select \* from tbase union all select \* from t\_appoint\_col;

id | nickname

----+-----------------------------

1 | hello TBase

2 | TBase好牛

1 | TBase分布式数据库的时代来了

1 | hello TBase

(4 rows)

**--过虑重复的记录**

postgres=# select \* from tbase union select \* from t\_appoint\_col;

id | nickname

----+-----------------------------

1 | TBase分布式数据库的时代来了

1 | hello TBase

2 | TBase好牛

(3 rows)

**--每个子查询分布在合并结果中的使用**

postgres=# select \* from ( select \* from tbase limit 1) as t union all select \* from (select \* from t\_appoint\_col limit 1) as t ;

id | nickname

----+-------------

1 | hello TBase

1 | hello TBase

(2 rows)

### 8.4.6、返回两个结果的交集

postgres=# create table t\_intersect1(id int,mc text);

NOTICE: Replica identity is needed for shard table, please add to this table through "alter table" command.

CREATE TABLE

postgres=# insert into t\_intersect1 values(1,'tbase'),(2,'tbase');

INSERT 0 2

postgres=# create table t\_intersect2(id int,mc text);

NOTICE: Replica identity is needed for shard table, please add to this table through "alter table" command.

CREATE TABLE

postgres=# insert into t\_intersect2 values(1,'tbase'),(3,'tbase');

INSERT 0 2

postgres=# select \* from t\_intersect1 INTERSECT select \* from t\_intersect2;

id | mc

----+-------

1 | tbase

(1 row)

### 8.4.7、返回两个结果的差集

postgres=# create table t\_except1(id int,mc text);

NOTICE: Replica identity is needed for shard table, please add to this table through "alter table" command.

CREATE TABLE

postgres=# insert into t\_except1 values(1,'tbase'),(2,'tbase');

INSERT 0 2

postgres=# create table t\_except2(id int,mc text);

NOTICE: Replica identity is needed for shard table, please add to this table through "alter table" command.

CREATE TABLE

postgres=# insert into t\_except2 values(1,'tbase'),(3,'tbase');

INSERT 0 2

postgres=# select \* from t\_except1 except select \* from t\_except2;

id | mc

----+-------

2 | tbase

(1 row)

### 8.4.8、any用法

postgres=# create table t\_any(id int,mc text);

NOTICE: Replica identity is needed for shard table, please add to this table through "alter table" command.

CREATE TABLE

postgres=# insert into t\_any values(1,'tbase'),(2,'TBase');

INSERT 0 2

postgres=# select \* from t\_any where id>any (select 1 union select 3);

id | mc

----+-------

2 | TBase

(1 row)

只需要大于其中一个值即为真

### 8.4.9、all用法

postgres=# create table t\_all(id int,mc text);

NOTICE: Replica identity is needed for shard table, please add to this table through "alter table" command.

CREATE TABLE

postgres=# insert into t\_all values(2,'tbase'),(3,'TBase');

INSERT 0 2

postgres=# select \* from t\_all where id>all (select 1 union select 2);

id | mc

----+-------

3 | TBase

(1 row)

需要大于所有值才为真

### 8.4.10、聚集查询

**--统计记录数**

postgres=# select count(1) from tbase;

count

-------

3

(1 row)

**--统计不重复值的记录表**

postgres=# select count(distinct id) from tbase;

count

-------

2

(1 row)

**--求和**

postgres=# select sum(id) from tbase;

sum

-----

4

(1 row)

**--求最大值**

postgres=# select max(id) from tbase;

max

-----

2

(1 row)

**--求最小值**

postgres=# select min(id) from tbase;

min

-----

1

(1 row)

**--求平均值**

postgres=# select avg(id) from tbase;

avg

--------------------

1.3333333333333333

(1 row)

### 8.4.11、多表关联

**--内连接**

postgres=# select \* from tbase inner join t\_appoint\_col on tbase.id=t\_appoint\_col.id;

id | nickname | id | nickname

----+-----------------------------+----+-------------

1 | hello TBase | 1 | hello TBase

1 | TBase分布式数据库的时代来了 | 1 | hello TBase

(2 rows)

**--左外连接**

postgres=# select \* from tbase left join t\_appoint\_col on tbase.id=t\_appoint\_col.id;

id | nickname | id | nickname

----+-----------------------------+----+-------------

1 | hello TBase | 1 | hello TBase

2 | TBase好牛 | |

1 | TBase分布式数据库的时代来了 | 1 | hello TBase

(3 rows)

**--右外连接**

postgres=# select \* from tbase right join t\_appoint\_col on tbase.id=t\_appoint\_col.id;

id | nickname | id | nickname

----+-----------------------------+----+-------------

1 | TBase分布式数据库的时代来了 | 1 | hello TBase

1 | hello TBase | 1 | hello TBase

| | 5 | Power TBase

(3 rows)

**--全连接**

postgres=# select \* from tbase full join t\_appoint\_col on tbase.id=t\_appoint\_col.id;

id | nickname | id | nickname

----+-----------------------------+----+-------------

1 | hello TBase | 1 | hello TBase

2 | TBase好牛 | |

1 | TBase分布式数据库的时代来了 | 1 | hello TBase

| | 5 | Power TBase

(4 rows)

### 8.4.12、聚合函数并发计算

* 单核计算

postgres=# \timing

Timing is on.

postgres=# set max\_parallel\_workers\_per\_gather to 0;

SET

Time: 0.633 ms

postgres=# select count(1) from t\_count;

count

----------

20000000

(1 row)

Time: 3777.518 ms (00:03.778)

* 二核并行

postgres=# set max\_parallel\_workers\_per\_gather to 2;

SET

Time: 0.478 ms

postgres=# select count(1) from t\_count;

count

----------

20000000

(1 row)

Time: 2166.481 ms (00:02.166)

* 四核并行

postgres=# set max\_parallel\_workers\_per\_gather to 4;

SET

Time: 0.315 ms

postgres=# select count(1) from t\_count;

count

----------

20000000

(1 row)

Time: 1162.433 ms (00:01.162)

postgres=#

### 8.4.13、not in中包含了null，结果全为真

postgres=# create table t\_not\_in(id int,mc text);

NOTICE: Replica identity is needed for shard table, please add to this table through "alter table" command.

CREATE TABLE

postgres=# insert into t\_not\_in values(1,'tbase'),(2,'pgxz');

INSERT 0 2

postgres=# select \* from t\_not\_in where id not in (3,5);

id | mc

----+-------

1 | tbase

2 | pgxz

(2 rows)

postgres=# select \* from t\_not\_in where id not in (3,5,null);

id | mc

----+----

(0 rows)

### 8.4.14、只查某个dn的数据

postgres=# create table t\_direct(id int,mc text);

NOTICE: Replica identity is needed for shard table, please add to this table through "alter table" command.

CREATE TABLE

postgres=# insert into t\_direct values(1,'tbase'),(3,'pgxz');

INSERT 0 2

postgres=# EXECUTE DIRECT ON (dn001) 'select \* from t\_direct';

id | mc

----+-------

1 | tbase

(1 row)

postgres=# EXECUTE DIRECT ON (dn002) 'select \* from t\_direct';

id | mc

----+------

3 | pgxz

(1 row)

postgres=# select \* from t\_direct ;

id | mc

----+-------

1 | tbase

3 | pgxz

(2 rows)

postgres=#

### 8.4.15、特殊应用

**--多行变成单行**

postgres=# create table t\_mulcol\_tosimplecol(id int,mc text);

NOTICE: Replica identity is needed for shard table, please add to this table through "alter table" command.

CREATE TABLE

postgres=# insert into t\_mulcol\_tosimplecol values(1,'tbase'),(2,'TBase');

INSERT 0 2

postgres=# select array\_to\_string(array(select mc from t\_mulcol\_tosimplecol),',');

array\_to\_string

-----------------

tbase,TBase

(1 row)

**--一列变成多行**

postgres=# create table t\_col\_to\_mulrow(id int,mc text);

NOTICE: Replica identity is needed for shard table, please add to this table through "alter table" command.

CREATE TABLE

postgres=# insert into t\_col\_to\_mulrow values(1,'tbase,TBase');

INSERT 0 1

postgres=# select regexp\_split\_to\_table((select mc from t\_col\_to\_mulrow where id=1 limit 1), ',');

regexp\_split\_to\_table

-----------------------

tbase

TBase

(2 rows)

## 8.5、insert语句

### 8.5.1、插入单条记录

指定所有字段

postgres=# insert into tbase(id,nickname) values(1,'hello TBase');

INSERT 0 1

指定某些字段，不指的有默认值的插入时带上默认值

postgres=# insert into tbase(nickname) values('TBase好牛');

INSERT 0 1

不指定字段则默认为所有字段与建表时一致

postgres=# insert into tbase values(nextval('t\_id\_seq'::regclass),'TBase好牛');

INSERT 0 1

字段顺序可以任意排列

postgres=# insert into tbase(nickname,id) values('TBase swap',5);

INSERT 0 1

使用default关键字，即值为建表时指定的默认值方式

postgres=# insert into tbase(id,nickname) values(default,'TBase default');

INSERT 0 1

上面五次插入记录后产生的数据

postgres=# select \* from tbase;

id | nickname

----+---------------

1 | hello TBase

2 | TBase好牛

5 | TBase swap

3 | TBase好牛

4 | TBase default

(5 rows)

### 8.5.2、插入多数记录

postgres=# insert into tbase(id,nickname) values(1,'hello TBase'),(2,'TBase好牛');

INSERT 0 2

postgres=# select \* from tbase;

id | nickname

----+-------------

1 | hello TBase

2 | TBase好牛

(2 rows)

### 8.5.3、使用子查询插入数据

postgres=# insert into tbase(id,nickname) values(1,(select relname from pg\_class limit 1));

INSERT 0 1

postgres=# select \* from tbase;

id | nickname

----+--------------

1 | pg\_statistic

(1 row)

### 8.5.4、从另外一个表取数据进行批量插入

postgres=# insert into tbase(nickname) select relname from pg\_class limit 3;

INSERT 0 3

postgres=# select \* from tbase;

id | nickname

----+---------------

5 | pg\_type

6 | pg\_toast\_2619

4 | pg\_statistic

(3 rows)

### 8.5.5、大批量的生成数据

postgres=# insert into tbase select t,md5(random()::text) from generate\_series(1,10000) as t;

INSERT 0 10000

postgres=# select count(1) from tbase;

count

-------

10000

(1 row)

### 8.5.6、返回插入数据，轻松获取插入记录的serial值

postgres=# insert into tbase(nickname) values('TBase好牛') returning \*;

id | nickname

----+-----------

7 | TBase好牛

(1 row)

指定返回的字段

INSERT 0 1

postgres=# insert into tbase(nickname) values('hello TBase') returning id;

id

----

8

1. row)

### 8.5.7、insert..update更新

* 使用ON CONFLICT

postgres=# \d+ t

Table "public.t"

Column | Type | Modifiers | Storage | Stats target | Description

--------+---------+-----------+----------+--------------+-------------

id | integer | | plain | |

nc | text | | extended | |

Indexes:

"t\_id\_udx" UNIQUE, btree (id)

Has OIDs: no

Distribute By SHARD(id)

Location Nodes: ALL DATANODES

postgres=# select \* from t;

id | nc

----+------

1 | pgxz

(1 row)

postgres=# insert into t values(1,'pgxz') ON CONFLICT (id) DO UPDATE SET nc = 'tbase';

INSERT 0 1

postgres=# select \* from t;

id | nc

----+-------

1 | tbase

1. row)

## 8.6、update语句

### 8.6.1、单表更新

postgres=# update tbase set nickname ='Hello TBase' where id=1;

UPDATE 1

--null条件的表达方法

postgres=# update tbase set nickname = 'Good TBase' where nickname is null;

UPDATE 1

postgres=# select \* from tbase;

id | nickname

----+-------------

2 | TBase好牛

1 | Hello TBase

3 | Good TBase

(3 rows)

### 8.6.2、多表关联更新

postgres=# update tbase set nickname ='Good TBase' from t\_appoint\_col where t\_appoint\_col.id=tbase.id;

UPDATE 1

postgres=# select \* from tbase;

id | nickname

----+------------

2 | TBase好牛

1 | Good TBase

(2 rows)

### 8.6.3、返回更新的数据

postgres=# update tbase set nickname = nickname where id = (random()\*2)::integer returning \*;

id | nickname

----+-----------

2 | TBase好牛

(1 row)

上面的语句我们随机更新了一些数据，然后返回更新过的记录，returning机制大在的降低的应用的复杂度

### 8.6.4、多列匹配更新

postgres=# update tbase set ( nickname , age ) = ( 'TBase好牛' , (random()\*2)::integer );

UPDATE 2

postgres=# select \* from tbase;

id | nickname | age

----+-----------+-----

1 | TBase好牛 | 2

2 | TBase好牛 | 0

(2 rows)

### 8.6.5、shard key不准许更新

postgres=# update tbase set id=8 where id=1;

ERROR: Distribute column or partition column can't be updated in current version

## 8.7、delete语句

### 8.7.1、带条件删除

postgres=# select \* from tbase;

id | nickname

----+-------------

2 | TBase好牛

1 | Hello TBase

3 |

4 | TBase good

(4 rows)

postgres=# delete from tbase where id=4;

DELETE 1

--null条件的表达方式

postgres=# delete from tbase where nickname is null;

DELETE 1

postgres=# select \* from tbase;

id | nickname

----+-------------

2 | TBase好牛

1 | Hello TBase

(2 rows)

### 8.7.2、多表关联删除数据

postgres=# select \* from tbase;

id | nickname

----+-------------

2 | TBase好牛

1 | Hello TBase

(2 rows)

postgres=# set prefer\_olap to on;

SET

postgres=# delete from tbase using t\_appoint\_col where tbase.id=t\_appoint\_col.id;

DELETE 1

postgres=# select \* from tbase;

id | nickname

----+-----------

2 | TBase好牛

(1 row)

### 8.7.3、返回删除数据

postgres=# delete from tbase returning \*;

id | nickname

----+-----------

2 | TBase好牛

(1 row)

returning机制大在的降低的应用的复杂度

### 8.7.4、删除所有数据

postgres=# insert into tbase select t,random()::text from generate\_series(1,100000) as t;

postgres=# \timing

Timing is on.

postgres=# delete from tbase ;

DELETE 100000

Time: 100.808 ms

使用truncate方法是全表删除更高效的方法

postgres=# insert into tbase select t,random()::text from generate\_series(1,100000) as t;

INSERT 0 100000

Time: 13178.429 ms

postgres=# truncate table tbase;

TRUNCATE TABLE

Time: 24.242 ms

## 8.8、游标的使用

### 8.8.1、定义一个游标

postgres=# begin;

BEGIN

postgres=# DECLARE tbase\_cur SCROLL CURSOR FOR SELECT \* from tbase ORDER BY id;

DECLARE CURSOR

注意：游标需要放在一个事务中使用

### 8.8.2、提取下一行数据

postgres=# DECLARE tbase\_cur SCROLL CURSOR FOR SELECT \* from tbase ORDER BY id;

DECLARE CURSOR

postgres=# FETCH NEXT from tbase\_cur ;

id | nickname

----+-------------

1 | hello TBase

(1 row)

postgres=# FETCH NEXT from tbase\_cur ;

id | nickname

----+-----------

2 | TBase好牛

(1 row)

### 8.8.3、提取前一行数据

postgres=# FETCH PRIOR from tbase\_cur ;

id | nickname

----+-------------

1 | hello TBase

(1 row)

postgres=# FETCH PRIOR from tbase\_cur ;

id | nickname

----+----------

(0 rows)

### 8.8.4、提取最后一行

postgres=# FETCH LAST from tbase\_cur ;

id | nickname

----+------------

5 | TBase swap

(1 row)

### 8.8.5、提取第一行

postgres=# FETCH FIRST from tbase\_cur ;

id | nickname

----+-------------

1 | hello TBase

(1 row)

### 8.8.6、提取该查询的第x行

postgres=# FETCH ABSOLUTE 2 from tbase\_cur ;

id | nickname

----+-----------

2 | TBase好牛

(1 row)

postgres=# FETCH ABSOLUTE -1 from tbase\_cur ;

id | nickname

----+------------

5 | TBase swap

(1 row)

postgres=# FETCH ABSOLUTE -2 from tbase\_cur ;

id | nickname

----+---------------

4 | TBase default

(1 row)

X为负数时则从尾部向上提

### 8.8.7、提取当前位置后的第x行

postgres=# FETCH ABSOLUTE 1 from tbase\_cur ;

id | nickname

----+-------------

1 | hello TBase

(1 row)

postgres=# FETCH RELATIVE 2 from tbase\_cur ;

id | nickname

----+-----------

3 | TBase好牛

(1 row)

postgres=# FETCH RELATIVE 2 from tbase\_cur ;

id | nickname

----+------------

5 | TBase swap

(1 row)

每提取一次数据，游标的位置都是会前行

### 8.8.8、向前提取x行数据

postgres=# FETCH FORWARD 2 from tbase\_cur ;

id | nickname

----+-------------

1 | hello TBase

2 | TBase好牛

(2 rows)

postgres=# FETCH FORWARD 2 from tbase\_cur ;

id | nickname

----+---------------

3 | TBase好牛

4 | TBase default

(2 rows)

### 8.8.9、向前提取剩下的所有数据

postgres=# FETCH FORWARD 2 from tbase\_cur ;

id | nickname

----+-------------

1 | hello TBase

2 | TBase好牛

(2 rows)

postgres=# FETCH FORWARD all from tbase\_cur ;

id | nickname

----+---------------

3 | TBase好牛

4 | TBase default

5 | TBase swap

(3 rows)

### 8.8.10、向后提取x行数据

postgres=# FETCH BACKWARD 2 from tbase\_cur ;

id | nickname

----+---------------

5 | TBase swap

4 | TBase default

(2 rows)

### 8.8.11、向后提取剩下的所有数据

postgres=# FETCH BACKWARD all from tbase\_cur ;

id | nickname

----+-------------

3 | TBase好牛

2 | TBase好牛

1 | hello TBase

1. rows)

## 8.9、copy的使用

COPY用于TBase表和标准文件系统文件之间数据互相复制。COPY TO可以把一个表的内容复制到一个文件，COPY FROM可以从一个文件复制数据到一个表（数据以追加形式入库），COPY TO也能复制一个SELECT查询的结果到一个文件。如果指定了一个列列表，COPY将只把指定列的数据复制到文件或者从文件复制数据到指定列。如果表中有列不在列列表中，COPY FROM将会为那些列插入默认值。 使用COPY时TBase服务器直接从“本地”一个文件读取或者写入到一个文件。该文件必须是TBase用户（运行服务器的用户 ID）可访问的并且应该以服务器的视角来指定其名称。

### 8.9.1、实验表结构及数据

postgres=# \d+ t

Table "public.t"

Column | Type | Modifiers | Storage | Stats target | Description

--------+-----------------------------+---------------+----------+--------------+-------------

f1 | integer | not null | plain | |

f2 | character varying(32) | not null | extended | |

f3 | timestamp without time zone | default now() | plain | |

f4 | integer | | plain | |

Has OIDs: yes

Distribute By SHARD(f1)

Location Nodes: ALL DATANODES

数据测试过程可以行再录入修改

postgres=# select \* from t;

f1 | f2 | f3 | f4

----+-------+----------------------------+----

3 | pgxz | 2017-10-28 18:24:05.645691 |

1 | Tbase | | 7

2 | | 2017-10-28 18:24:05.643102 | 3

(3 rows)

### 8.9.2、copy to用法详解--复制数据到文件中

#### 8.9.2.1、导出所有列

postgres=# copy public.t to '/data/pgxz/t.txt';

COPY 3

postgres=# \! cat /data/pgxz/t.txt

1 Tbase \N 7

2 2017-10-28 18:24:05.643102 3

3 pgxz 2017-10-28 18:24:05.645691 \N

默认生成的文件内容为表的所有列，列与列之间使用tab分隔开来。NULL值生成的值为\N

#### 8.9.2.2、导出部分列

postgres=# copy public.t(f1,f2) to '/data/pgxz/t.txt';

COPY 3

postgres=# \! cat /data/pgxz/t.txt

1 Tbase

2

3 pgxz

postgres=#

只导出f1和f2列

#### 8.9.2.3、导出查询结果

postgres=# copy (select f2,f3 from public.t order by f3) to '/data/pgxz/t.txt';

COPY 3

postgres=# \! cat /data/pgxz/t.txt

2017-10-28 18:24:05.643102

pgxz 2017-10-28 18:24:05.645691

Tbase \N

postgres=#

查询可以是任何复杂查询

#### 8.9.2.4、指定生成文件格式

* 生成csv格式

postgres=# copy public.t to '/data/pgxz/t.txt' with csv;

COPY 3

postgres=# \! cat /data/pgxz/t.txt

1,Tbase,,7

2,pgxc,2017-10-28 18:24:05.643102,3

3,pgxz,2017-10-28 18:24:05.645691,

* 生成二进制格式

postgres=# copy public.t to '/data/pgxz/t.txt' with binary;

COPY 3

postgres=# \1

postgres=# \! cat /data/pgxz/t.txt

PGCOPY

Tbase

默认为TEXT格式

#### 8.9.2.5、使用delimiter指定列与列之间的分隔符

postgres=# copy public.t to '/data/pgxz/t.txt' with delimiter '@';

COPY 3

postgres=# \! cat /data/pgxz/t.txt

1@Tbase@\N@7

2@pgxc@2017-10-28 18:24:05.643102@3

3@pgxz@2017-10-28 18:24:05.645691@\N

postgres=# copy public.t to '/data/pgxz/t.txt' with csv delimiter '@';

COPY 3

postgres=# \! cat /data/pgxz/t.txt

1@Tbase@@7

2@pgxc@2017-10-28 18:24:05.643102@3

3@pgxz@2017-10-28 18:24:05.645691@

postgres=# copy public.t to '/data/pgxz/t.txt' with csv delimiter '@@';

ERROR: COPY delimiter must be a single one-byte character

postgres=# copy public.t to '/data/pgxz/t.txt' with binary delimiter '@';

ERROR: cannot specify DELIMITER in BINARY mode

指定分隔文件各列的字符。文本格式中默认是一个制表符， 而CSV格式中默认是一个逗号。分隔符必须是一个单一的单字节字符，即汉字是不支持的。使用binary格式时不允许这个选项。

#### 8.9.2.6、NULL值的处理

postgres=# copy public.t to '/data/pgxz/t.txt' with NULL 'NULL';

COPY 3

postgres=# \! cat /data/pgxz/t.txt

1 Tbase NULL 7

2 pgxc 2017-10-28 18:24:05.643102 3

3 pgxz 2017-10-28 18:24:05.645691 NULL

postgres=# copy public.t to '/data/pgxz/t.txt' with CSV NULL 'NULL';

COPY 3

postgres=# \! cat /data/pgxz/t.txt

1,Tbase,NULL,7

2,pgxc,2017-10-28 18:24:05.643102,3

3,pgxz,2017-10-28 18:24:05.645691,NULL

postgres=# copy public.t to '/data/pgxz/t.txt' with binary NULL 'NULL';

ERROR: cannot specify NULL in BINARY mode

postgres=#

记录值为NULL时导出为NULL字符。使用binary格式时不允许这个选项。

#### 8.9.2.7、生成列标题名

postgres=# copy public.t to '/data/pgxz/t.txt' with csv HEADER;

COPY 3

postgres=# \! cat /data/pgxz/t.txt

f1,f2,f3,f4

1,Tbase,,7

2,pgxc,2017-10-28 18:24:05.643102,3

3,pgxz,2017-10-28 18:24:05.645691,

postgres=# copy public.t to '/data/pgxz/t.txt' with HEADER;

ERROR: COPY HEADER available only in CSV mode

只有使用 CSV格式时才允许这个选项。

#### 8.9.2.8、导出oids系统列

postgres=# drop table t;

DROP TABLE

postgres=# CREATE TABLE t (

postgres(# f1 integer NOT NULL,

postgres(# f2 text NOT NULL,

postgres(# f3 timestamp without time zone,

postgres(# f4 integer

postgres(# )

postgres-# with oids DISTRIBUTE BY SHARD (f1);

CREATE TABLE

postgres=# copy t from '/data/pgxz/t.txt' with csv ;

COPY 3

postgres=# select \* from t;

f1 | f2 | f3 | f4

----+---------------+----------------------------+----

1 | Tbase | | 7

2 | pg'", xc | 2017-10-28 18:24:05.643102 | 3

3 | pgxz | 2017-10-28 18:24:05.645691 |

(3 rows)

postgres=# copy t to '/data/pgxz/t.txt' with oids ;

COPY 3

postgres=# \! cat /data/pgxz/t.txt

35055 1 Tbase \N 7

35056 2 pg'", xc 2017-10-28 18:24:05.643102 3

35177 3 pgxz 2017-10-28 18:24:05.645691 \N

创建表时使用了with oids才能使用oids 选项

#### 8.9.2.9、使用quote自定义引用字符

postgres=# copy t to '/data/pgxz/t.txt' with csv;

COPY 3

postgres=# \! cat /data/pgxz/t.txt

1,Tbase,,7

2,"pg'"", xc%",2017-10-28 18:24:05.643102,3

3,pgxz,2017-10-28 18:24:05.645691,

默认引用字符为“双引号”

postgres=# copy t to '/data/pgxz/t.txt' with quote '%' csv;

COPY 3

postgres=# \! cat /data/pgxz/t.txt

1,Tbase,,7

2,%pg'", xc%%%,2017-10-28 18:24:05.643102,3

3,pgxz,2017-10-28 18:24:05.645691,

上面指定了引用字符为百分号，系统自动把字段值为%的字符替换为双个%

postgres=# copy t to '/data/pgxz/t.txt' with quote '%';

ERROR: COPY quote available only in CSV mode

只有使用 CSV格式时才允许这个选项。

postgres=# copy t to '/data/pgxz/t.txt' with quote '%%' csv;

ERROR: COPY quote must be a single one-byte character

postgres=#

引用字符必须是一个单一的单字节字符，即汉字是不支持的。

#### 8.9.2.10、使用escape自定义逃逸符

postgres=# copy t to '/data/pgxz/t.txt' with quote '%' csv;

COPY 3

postgres=# \! cat /data/pgxz/t.txt

1,Tbase,,7

2,%pg'", xc%%%,2017-10-28 18:24:05.643102,3

3,pgxz,2017-10-28 18:24:05.645691,

不指定escape逃逸符，默认和QUOTE值一样

postgres=# copy t to '/data/pgxz/t.txt' with quote '%' escape '@' csv;

COPY 3

postgres=# \! cat /data/pgxz/t.txt

1,Tbase,,7

2,%pg'", xc@%%,2017-10-28 18:24:05.643102,3

3,pgxz,2017-10-28 18:24:05.645691,

指定逃逸符为’@’

postgres=# copy t to '/data/pgxz/t.txt' with quote '%' escape '@@' csv;

ERROR: COPY escape must be a single one-byte character

这必须是一个单一的单字节字符。

postgres=# copy t to '/data/pgxz/t.txt' with quote '%' escape '@';

ERROR: COPY quote available only in CSV mode

只有使用CSV格式时才允许这个选项。

#### 8.9.2.11、强制给某个列添加引用字符

postgres=# copy t to '/data/pgxz/t.txt' (format 'csv',force\_quote (f1,f2));

COPY 3

postgres=# \! cat /data/pgxz/t.txt

"1","Tbase",,7

"2","pg'"", xc%",2017-10-28 18:24:05.643102,3

"3","pgxz",2017-10-28 18:24:05.645691,

指定2列强制添加引用字符

postgres=# copy t to '/data/pgxz/t.txt' (format 'csv',force\_quote (f1,f4,f3,f2));

COPY 3

postgres=# \! cat /data/pgxz/t.txt

"1","Tbase",,"7"

"2","pg'"", xc%","2017-10-28 18:24:05.643102","3"

"3","pgxz","2017-10-28 18:24:05.645691",

指定4列强制添加引用字符，字段的顺序可以任意排列

postgres=# copy t to '/data/pgxz/t.txt' (format 'text',force\_quote (f1,f2,f3,f4));

ERROR: COPY force quote available only in CSV mode

postgres=#

只有使用CSV格式时才允许这个选项。

#### 8.9.2.12、使用encoding指定导出文件内容编码

postgres=# copy t to '/data/pgxz/t.csv' (encoding utf8);

COPY 3

导出文件编码为UTF8

postgres=# copy t to '/data/pgxz/t.csv' (encoding gbk);

COPY 3

postgres=#

导出文件编码为gbk

使用set\_client\_encdoing to gbk;也可以将文件的内容设置为需要的编码，如下所示

postgres=# set client\_encoding to gbk;

SET

postgres=# copy t to '/data/pgxz/t.csv' with csv ;

COPY 4

postgres=#

### 8.9.3、copy from用法详解--复制文件内容到数据表中

#### 8.9.3.1、导入所有列

postgres=# \! cat /data/pgxz/t.txt

1 Tbase \N 7

2 pg'", xc% 2017-10-28 18:24:05.643102 3

3 pgxz 2017-10-28 18:24:05.645691 \N

postgres=# truncate table t;

TRUNCATE TABLE

postgres=# copy t from '/data/pgxz/t.txt';

COPY 3

postgres=# select \* from t;

f1 | f2 | f3 | f4

----+----------------+----------------------------+----

1 | Tbase | | 7

2 | pg'", xc% | 2017-10-28 18:24:05.643102 | 3

3 | pgxz | 2017-10-28 18:24:05.645691 |

1. rows)

#### 8.9.3.2、导入部分指定列

postgres=# copy t(f1,f2) to '/data/pgxz/t.txt';

postgres=# \! cat /data/pgxz/t.txt

1 Tbase

2 pg'", xc%

3 pgxz

postgres=# truncate table t;

TRUNCATE TABLE

postgres=# copy t(f1,f2) from '/data/pgxz/t.txt';

COPY 3

postgres=# select \* from t;

f1 | f2 | f3 | f4

----+----------------+----------------------------+----

1 | Tbase | 2017-10-30 11:54:16.559579 |

2 | pg'", xc% | 2017-10-30 11:54:16.559579 |

3 | pgxz | 2017-10-30 11:54:16.560283 |

1. rows)

有默认值的字段在没有导入时，会自动的将默认值付上

postgres=# \! cat /data/pgxz/t.txt

1 \N Tbase

2 2017-10-28 18:24:05.643102 pg'", xc%

3 2017-10-28 18:24:05.645691 pgxz

postgres=# truncate table t;

TRUNCATE TABLE

postgres=# copy t(f1,f3,f2) from '/data/pgxz/t.txt';

COPY 3

postgres=# select \* from t;

f1 | f2 | f3 | f4

----+----------------+----------------------------+----

1 | Tbase | |

2 | pg'", xc% | 2017-10-28 18:24:05.643102 |

3 | pgxz | 2017-10-28 18:24:05.645691 |

1. rows)

字段的顺序可以任意调整，但需要与导放文件的存放顺序一致

postgres=# \! cat /data/pgxz/t.txt;

1 Tbase \N 7

2 pg'", xc% 2017-10-28 18:24:05.643102 3

3 pgxz 2017-10-28 18:24:05.645691 \N

postgres=# copy t (f1,f2) from '/data/pgxz/t.txt';

ERROR: extra data after last expected column

CONTEXT: COPY t, line 1: "1 Tbase \N 7"

数据文件的列表不能多于要导入的列数，否则会出错。

#### 8.9.3.3、指定导入文件格式

postgres=# \! cat /data/pgxz/t.txt

1 Tbase \N 7

2 pg'", xc% 2017-10-28 18:24:05.643102 3

3 pgxz 2017-10-28 18:24:05.645691 \N

postgres=# copy t from '/data/pgxz/t.txt' (format 'text');

COPY 3

TRUNCATE TABLE

postgres=# \! cat /data/pgxz/t.csv

1,Tbase,,7

2,"pg'"", xc%",2017-10-28 18:24:05.643102,3

3,pgxz,2017-10-28 18:24:05.645691,

postgres=# copy t from '/data/pgxz/t.csv' (format 'csv');

COPY 3

postgres=# truncate table t;

TRUNCATE TABLE

postgres=# \! od -c /data/pgxz/t.bin

0000000 P G C O P Y \n 377 \r \n \0 \0 \0 \0 \0 \0

0000020 \0 \0 \0 \0 004 \0 \0 \0 004 \0 \0 \0 001 \0 \0 \0

0000040 005 T b a s e 377 377 377 377 \0 \0 \0 004 \0 \0

0000060 \0 \a \0 004 \0 \0 \0 004 \0 \0 \0 002 \0 \0 \0 016

0000100 p g ' " , x c % \0 \0

0000120 \0 \b \0 001 377 236 G w 213 ^ \0 \0 \0 004 \0 \0

0000140 \0 003 \0 004 \0 \0 \0 004 \0 \0 \0 003 \0 \0 \0 004

0000160 p g x z \0 \0 \0 \b \0 001 377 236 G w 225 {

0000200 377 377 377 377 377 377

0000206

postgres=# copy t from '/data/pgxz/t.bin' (format 'binary');

COPY 3

postgres=# select \* from t;

f1 | f2 | f3 | f4

----+----------------+----------------------------+----

1 | Tbase | | 7

2 | pg'", xc% | 2017-10-28 18:24:05.643102 | 3

3 | pgxz | 2017-10-28 18:24:05.645691 |

(3 rows)

#### 8.9.3.4、使用delimiter指定列与列之间的分隔符

postgres=# \! cat /data/pgxz/t.txt

1%Tbase%\N%7

2%pg'", xc\%%2017-10-28 18:24:05.643102%3

3%pgxz%2017-10-28 18:24:05.645691%\N

postgres=# copy t from '/data/pgxz/t.txt' (format 'text',delimiter '%');

COPY 3

postgres=# select \* from t;

f1 | f2 | f3 | f4

----+----------------+----------------------------+----

1 | Tbase | | 7

2 | pg'", xc% | 2017-10-28 18:24:05.643102 | 3

3 | pgxz | 2017-10-28 18:24:05.645691 |

(3 rows)

postgres=# \! cat /data/pgxz/t.csv

1%Tbase%%7

2%"pg'"", xc%"%2017-10-28 18:24:05.643102%3

3%pgxz%2017-10-28 18:24:05.645691%

postgres=# truncate table t;

TRUNCATE TABLE

postgres=# copy t from '/data/pgxz/t.csv' (format 'csv',delimiter '%');

COPY 3

postgres=# select \* from t;

f1 | f2 | f3 | f4

----+----------------+----------------------------+----

1 | Tbase | | 7

2 | pg'", xc% | 2017-10-28 18:24:05.643102 | 3

3 | pgxz | 2017-10-28 18:24:05.645691 |

1. rows)

#### 8.9.3.5、NULL值处理

postgres=# \! cat /data/pgxz/t.txt

1 Tbase NULL 7

2 pg'", xc% 2017-10-28 18:24:05.643102 3

3 pgxz 2017-10-28 18:24:05.645691 NULL

postgres=# copy t from '/data/pgxz/t.txt' (null 'NULL');

COPY 3

postgres=# select \* from t;

f1 | f2 | f3 | f4

----+----------------+----------------------------+----

1 | Tbase | | 7

2 | pg'", xc% | 2017-10-28 18:24:05.643102 | 3

3 | pgxz | 2017-10-28 18:24:05.645691 |

(3 rows)

将文件中的NULL字符串当成NULL值处理,SQL SERVER导出来的文件中把NULL值替换成字符串NULL,所以入库时可以这样处理一下，注意字符串是区分大小写，如下面语句导入数据就会出错

postgres=# copy t from '/data/pgxz/t.txt' (null 'null');

ERROR: invalid input syntax for type timestamp: "NULL"

CONTEXT: COPY t, line 1, column f3: "NULL"

#### 8.9.3.6、自定义quote字符

postgres=# \! cat /data/pgxz/t.csv

1,Tbase,,7

2,%pg'", xc%%%,2017-10-28 18:24:05.643102,3

3,pgxz,2017-10-28 18:24:05.645691,

如果不配置quote字符则无法导入文件

postgres=# copy t from '/data/pgxz/t.csv' (format 'csv');

ERROR: unterminated CSV quoted field

CONTEXT: COPY t, line 4: "2,%pg'", xc%%%,2017-10-28 18:24:05.643102,3

3,pgxz,2017-10-28 18:24:05.645691,

"

postgres=# copy t from '/data/pgxz/t.csv' (format 'csv',quote '%');

COPY 3

postgres=#

postgres=# copy t from '/data/pgxz/t.csv' (format 'text',quote '%');

ERROR: COPY quote available only in CSV mode

只有csv格式导入时才能配置quote字符

#### 8.9.3.7、自定义escape字符

postgres=# \! cat /data/pgxz/t.csv

1,Tbase,,7

2,"pg'@", xc%",2017-10-28 18:24:05.643102,3

3,pgxz,2017-10-28 18:24:05.645691,

postgres=# copy t from '/data/pgxz/t.csv' (format 'csv');

ERROR: unterminated CSV quoted field

CONTEXT: COPY t, line 4: "2,"pg'@", xc%",2017-10-28 18:24:05.643102,3

3,pgxz,2017-10-28 18:24:05.645691,

"

不指定escape字符时，系统默认就是双写的quote字符，如双倍的“双引号”

postgres=# copy t from '/data/pgxz/t.csv' (format 'csv',escape '@');

COPY 3

postgres=# select \* from t;

f1 | f2 | f3 | f4

----+----------------+----------------------------+----

1 | Tbase | | 7

2 | pg'", xc% | 2017-10-28 18:24:05.643102 | 3

3 | pgxz | 2017-10-28 18:24:05.645691 |

(3 rows)

postgres=#

#### 8.9.3.8、csv header忽略首行

postgres=# \! cat /data/pgxz/t.csv;

f1,f2,f3,f4

1,Tbase,,7

2,"pg'"", xc%",2017-10-28 18:24:05.643102,3

3,pgxz,2017-10-28 18:24:05.645691,

postgres=# copy t from '/data/pgxz/t.csv' (format 'csv');

ERROR: invalid input syntax for integer: "f1"

CONTEXT: COPY t, line 1, column f1: "f1"

postgres=# copy t from '/data/pgxz/t.csv' (format 'csv',header true);

COPY 3

postgres=# select \* from t;

f1 | f2 | f3 | f4

----+----------------+----------------------------+----

1 | Tbase | | 7

2 | pg'", xc% | 2017-10-28 18:24:05.643102 | 3

3 | pgxz | 2017-10-28 18:24:05.645691 |

(3 rows)

如果不忽略首行，则系统会把首行当成数据，造成导入失败

#### 8.9.3.9、导入oid列值

postgres=# \! cat /data/pgxz/t.txt

35242 1 Tbase \N 7

35243 2 pg'", xc% 2017-10-28 18:24:05.643102 3

35340 3 pgxz 2017-10-28 18:24:05.645691 \N

postgres=# truncate table t;

TRUNCATE TABLE

postgres=# copy t from '/data/pgxz/t.txt' (oids true);

COPY 3

postgres=# select oid,\* from t;

oid | f1 | f2 | f3 | f4

-------+----+----------------+----------------------------+----

35242 | 1 | Tbase | | 7

35243 | 2 | pg'", xc% | 2017-10-28 18:24:05.643102 | 3

35340 | 3 | pgxz | 2017-10-28 18:24:05.645691 |

(3 rows)

#### 8.9.3.10、使用FORCE\_NOT\_NULL把某列中空值变成长度为0的字符串，而不是NULL值

postgres=# truncate table t;

TRUNCATE TABLE

postgres=# \! cat '/data/pgxz/t.csv' ;

1,Tbase,,7

2,"pg'"", xc%",2017-10-28 18:24:05.643102,3

3,pgxz,2017-10-28 18:24:05.645691,

4,,2017-10-30 16:14:14.954213,4

postgres=# copy t from '/data/pgxz/t.csv' (format 'csv');

ERROR: node:16386, error null value in column "f2" violates not-null constraint

DETAIL: Failing row contains (4, null, 2017-10-30 16:14:14.954213, 4).

postgres=# select \* from t where f2='';

f1 | f2 | f3 | f4

----+----+----+----

(0 rows)

不使用FORCE\_NOT\_NULL处理的话就变成NULL值

postgres=# truncate table t;

TRUNCATE TABLE

postgres=# copy t from '/data/pgxz/t.csv' (format 'csv' ,FORCE\_NOT\_NULL (f2));

COPY 4

postgres=# select \* from t where f2='';

f1 | f2 | f3 | f4

----+----+----------------------------+----

4 | | 2017-10-30 16:14:14.954213 | 4

1. row)

使用FORCE\_NOT\_NULL处理就变成长度为0的字符串

#### 8.9.3.11、encoding指定导入文件的编码

postgres=# \! enca -L zh\_CN /data/pgxz/t.txt

Simplified Chinese National Standard; GB2312

postgres=# copy t from '/data/pgxz/t.txt' ;

COPY 4

postgres=# select \* from t;

f1 | f2 | f3 | f4

----+----------------+----------------------------+----

1 | Tbase | | 7

2 | pg'", xc% | 2017-10-28 18:24:05.643102 | 3

3 | pgxz | 2017-10-28 18:24:05.645691 |

4 | | 2017-10-30 16:41:09.157612 | 4

(4 rows)

不指定导入文件的编码格式，则无法正确导入中文字符

postgres=# truncate table t;

TRUNCATE TABLE

postgres=# copy t from '/data/pgxz/t.txt' (encoding gbk) ;

COPY 4

postgres=# select \* from t;

f1 | f2 | f3 | f4

----+----------------+----------------------------+----

1 | Tbase | | 7

2 | pg'", xc% | 2017-10-28 18:24:05.643102 | 3

3 | pgxz | 2017-10-28 18:24:05.645691 |

4 | 腾讯 | 2017-10-30 16:41:09.157612 | 4

1. rows)

使用encoding gbk后便可以正确导入文件的内容，你也可以使用下面的方式转换导入文件的编码后再导入数据

postgres=# truncate table t;

TRUNCATE TABLE

postgres=# \! enconv -L zh\_CN -x UTF-8 /data/pgxz/t.txt

postgres=# copy t from '/data/pgxz/t.txt';

COPY 4

postgres=# select \* from t;

f1 | f2 | f3 | f4

----+----------------+----------------------------+----

1 | Tbase | | 7

2 | pg'", xc% | 2017-10-28 18:24:05.643102 | 3

3 | pgxz | 2017-10-28 18:24:05.645691 |

4 | 腾讯 | 2017-10-30 16:41:09.157612 | 4

(4 rows)

postgres=#

## 8.10、json/jsonb的使用

TBase不只是一个分布式关系型数据库系统，同时它还支持非关系数据类型json，JSON 数据类型是用来存储 JSON（JavaScript Object Notation） 数据的。这种数据也可以被存储为text，但是 JSON 数据类型的 优势在于能强制要求每个被存储的值符合 JSON 规则。 也有很多 JSON 相关的函数和操作符可以用于存储在这些数据类型中的数据。JSON 数据类型有json 和 jsonb。它们接受完全相同的值集合作为输入。主要的实际区别是效率。json数据类型存储输入文本的精准拷贝，处理函数必须在每次执行时必须重新解析该数据。而jsonb数据被存储在一种分解好的二进制格式中，它在输入时要稍慢一些，因为需要做附加的转换。但是 jsonb在处理时要快很多，因为不需要解析。jsonb也支持索引，这也是一个令人瞩目的优势。

### 8.10.1、json应用

#### 8.10.1.1、创建json类型字段表

postgres=# create table t\_json(id int,f\_json json);

NOTICE: Replica identity is needed for shard table, please add to this table through "alter table" command.

CREATE TABLE

#### 8.10.1.2、插入数据

postgres=# insert into t\_json values(1,'{"col1":1,"col2":"tbase"}');

INSERT 0 1

postgres=# insert into t\_json values(2,'{"col1":1,"col2":"tbase","col3":"pgxz"}');

INSERT 0 1

postgres=# select \* from t\_json;

id | f\_json

----+-----------------------------------------

1 | {"col1":1,"col2":"tbase"}

2 | {"col1":1,"col2":"tbase","col3":"pgxz"}

(2 rows)

#### 8.10.1.3、通过键获得 JSON 对象域

postgres=# select f\_json ->'col2' as col2 ,f\_json -> 'col3' as col3 from t\_json;

col2 | col3

---------+--------

"tbase" |

"tbase" | "pgxz"

(2 rows)

#### 8.10.1.4、以文本形式获取对象值

postgres=# select f\_json ->>'col2' as col2 ,f\_json ->> 'col3' as col3 from t\_json;

col2 | col3

-------+------

tbase |

tbase | pgxz

(2 rows)

postgres=# select f\_json ->>'col2' as col2 ,f\_json ->> 'col3' as col3 from t\_json where f\_json ->> 'col3' is not null;

col2 | col3

-------+------

tbase | pgxz

(1 row)

### 8.10.2、jsonb应用

#### 8.10.2.1、创建jsonb类型字段表

postgres=# create table t\_jsonb(id int,f\_jsonb jsonb);

NOTICE: Replica identity is needed for shard table, please add to this table through "alter table" command.

CREATE TABLE

postgres=#

#### 8.10.2.2、插入数据

postgres=# insert into t\_jsonb values(1,'{"col1":1,"col2":"tbase"}');

INSERT 0 1

postgres=# insert into t\_jsonb values(2,'{"col1":1,"col2":"tbase","col3":"pgxz"}');

INSERT 0 1

postgres=# select \* from t\_jsonb;

id | f\_jsonb

----+----------------------------------------------

1 | {"col1": 1, "col2": "tbase"}

2 | {"col1": 1, "col2": "tbase", "col3": "pgxz"}

(2 rows)

--jsonb插入时会移除重复的键，如下所示

postgres=# insert into t\_jsonb values(3,'{"col1":1,"col2":"tbase","col2":"pgxz"}');

INSERT 0 1

postgres=# select \* from t\_jsonb;

id | f\_jsonb

----+----------------------------------------------

1 | {"col1": 1, "col2": "tbase"}

3 | {"col1": 1, "col2": "pgxz"}

2 | {"col1": 1, "col2": "tbase", "col3": "pgxz"}

(3 rows)

#### 8.10.2.3、更新数据

--增加元素

postgres=# update t\_jsonb set f\_jsonb = f\_jsonb || '{"col3":"pgxz"}'::jsonb where id=1;

UPDATE 1

--更新原来的元素

postgres=# update t\_jsonb set f\_jsonb = f\_jsonb || '{"col2":"tbase"}'::jsonb where id=3;

UPDATE 1

postgres=# select \* from t\_jsonb;

id | f\_jsonb

----+----------------------------------------------

2 | {"col1": 1, "col2": "tbase", "col3": "pgxz"}

1 | {"col1": 1, "col2": "tbase", "col3": "pgxz"}

3 | {"col1": 1, "col2": "tbase"}

(3 rows)

--删除某个键

postgres=# update t\_jsonb set f\_jsonb = f\_jsonb - 'col3';

UPDATE 3

postgres=# select \* from t\_jsonb;

id | f\_jsonb

----+------------------------------

2 | {"col1": 1, "col2": "tbase"}

1 | {"col1": 1, "col2": "tbase"}

3 | {"col1": 1, "col2": "tbase"}

(3 rows)

#### 8.10.2.4、jsonb\_set()函数更新数据

jsonb\_set(target jsonb, path text[], new\_value jsonb, [create\_missing boolean]) 说明：target指要更新的数据源，path指路径，new\_value指更新后的键值，create\_missing值为true表示如果键不存在则添加，create\_missing值为false表示如果键不存在则不添加。

postgres=# update t\_jsonb set f\_jsonb = jsonb\_set( f\_jsonb , '{col}' , '"pgxz"' , true ) where id=1;

UPDATE 1

postgres=# update t\_jsonb set f\_jsonb = jsonb\_set( f\_jsonb , '{col}' , '"pgxz"' , false ) where id=2;

UPDATE 1

postgres=# update t\_jsonb set f\_jsonb = jsonb\_set( f\_jsonb , '{col2}' , '"pgxz"' , false ) where id=3;

UPDATE 1

postgres=# select \* from t\_jsonb;

id | f\_jsonb

----+---------------------------------------------

1 | {"col": "pgxz", "col1": 1, "col2": "tbase"}

2 | {"col1": 1, "col2": "tbase"}

3 | {"col1": 1, "col2": "pgxz"}

(3 rows)

### 8.10.3、jsonb函数应用

#### 8.10.3.1、jsonb\_each()将json对象转变键和值

postgres=# select f\_jsonb from t\_jsonb where id=1;

f\_jsonb

---------------------------------------------

{"col": "pgxz", "col1": 1, "col2": "tbase"}

(1 row)

postgres=# select \* from jsonb\_each((select f\_jsonb from t\_jsonb where id=1));

key | value

------+---------

col | "pgxz"

col1 | 1

col2 | "tbase"

(3 rows)

#### 8.10.3.2、jsonb\_each\_text()将json对象转变文本类型的键和值

postgres=# select \* from jsonb\_each\_text((select f\_jsonb from t\_jsonb where id=1));

key | value

------+-------

col | pgxz

col1 | 1

col2 | tbase

(3 rows)

#### 8.10.3.3、row\_to\_json()将一行记录变成一个json对象

postgres=# \d+ tbase

Table "public.tbase"

Column | Type | Collation | Nullable | Default | Storage | Stats target | Description

----------+---------+-----------+----------+---------+----------+--------------+-------------

id | integer | | not null | | plain | |

nickname | text | | | | extended | |

Indexes:

"tbase\_pkey" PRIMARY KEY, btree (id)

Distribute By: SHARD(id)

Location Nodes: ALL DATANODES

postgres=# select \* from tbase;

id | nickname

----+----------

1 | tbase

2 | pgxz

(2 rows)

postgres=# select row\_to\_json(tbase) from tbase;

row\_to\_json

-----------------------------

{"id":1,"nickname":"tbase"}

{"id":2,"nickname":"pgxz"}

(2 rows)

#### 8.10.3.4、json\_object\_keys()返回一个对象中所有的键

postgres=# select \* from json\_object\_keys((select f\_jsonb from t\_jsonb where id=1)::json);

json\_object\_keys

------------------

col

col1

col2

(3 rows)

### 8.10.4、jsonb索引使用

TBase为文档jsonb提供了GIN索引，GIN 索引可以被用来有效地搜索在大量jsonb文档（数据）中出现 的键或者键值对。

#### 8.10.4.1、创建立jsonb索引

postgres=# create index t\_jsonb\_f\_jsonb\_idx on t\_jsonb using gin(f\_jsonb);

CREATE INDEX

postgres=# \d+ t\_jsonb

Table "public.t\_jsonb"

Column | Type | Collation | Nullable | Default | Storage | Stats target | Description

---------+---------+-----------+----------+---------+----------+--------------+-------------

id | integer | | | | plain | |

f\_jsonb | jsonb | | | | extended | |

Indexes:

"t\_jsonb\_f\_jsonb\_idx" gin (f\_jsonb)

Distribute By: SHARD(id)

Location Nodes: ALL DATANODES

#### 8.10.4.2、测试查询的性能

postgres=# select count(1) from t\_jsonb;

count

----------

10000000

(1 row)

postgres=# analyze t\_jsonb;

ANALYZE

--没有索引开销

postgres=# select \* from t\_jsonb where f\_jsonb @> '{"col1":9999}';

id | f\_jsonb

------+--------------------------------

9999 | {"col1": 9999, "col2": "9999"}

9999 | {"col1": 9999, "col2": "9999"}

9999 | {"col1": 9999, "col2": "9999"}

9999 | {"col1": 9999, "col2": "9999"}

9999 | {"col1": 9999, "col2": "9999"}

(5 rows)

Time: 2473.488 ms (00:02.473)

--有索引开销

postgres=# select \* from t\_jsonb where f\_jsonb @> '{"col1":9999}';

id | f\_jsonb

------+--------------------------------

9999 | {"col1": 9999, "col2": "9999"}

9999 | {"col1": 9999, "col2": "9999"}

9999 | {"col1": 9999, "col2": "9999"}

9999 | {"col1": 9999, "col2": "9999"}

9999 | {"col1": 9999, "col2": "9999"}

(5 rows)

Time: 217.968 ms

# 9、TBase的进阶开发

## 9.1、TBase事务控制

### 9.1.1、开始一个事务

postgres=# begin;

BEGIN

或者

postgres=# begin TRANSACTION ;

BEGIN

也可以定义事务的级别

postgres=# begin transaction isolation level read committed ;

BEGIN

### 9.1.2、提交事务

进程#1访问

postgres=# begin;

BEGIN

postgres=# delete from tbase where id=5;

DELETE 1

postgres=#

postgres=# select \* from tbase order by id;

id | nickname

----+---------------

1 | hello TBase

2 | TBase好牛

3 | TBase好牛

4 | TBase default

TBase也是完全支持ACID特性，没提交前开启另一个连接查询，你会看到是5条记录，这是TBase隔离性和多版本视图的实现，如下所示

进程#2访问

postgres=# select \* from tbase order by id;

id | nickname

----+---------------

1 | hello TBase

2 | TBase好牛

3 | TBase好牛

4 | TBase default

5 | TBase swap

(5 rows)

进程#1提交数据

postgres=# commit;

COMMIT

postgres=#

进程#2再查询数据，这时能看到已经提交的数据了，这个级别叫“读已提交”

postgres=# select \* from tbase order by id;

id | nickname

----+---------------

1 | hello TBase

2 | TBase好牛

3 | TBase好牛

4 | TBase default

(4 rows)

### 9.1.3、回滚事务

postgres=# begin;

BEGIN

postgres=# delete from tbase where id in (3,4);

DELETE 2

postgres=# select \* from tbase;

id | nickname

----+-------------

1 | hello TBase

2 | TBase好牛

(2 rows)

postgres=# rollback;

ROLLBACK

Rollback后数据又回来了

postgres=# select \* from tbase;

id | nickname

----+---------------

1 | hello TBase

2 | TBase好牛

3 | TBase好牛

4 | TBase default

(4 rows)

### 9.1.4、事务读一致性REPEATABLE READ

这种事务级别表示事务自始至终读取的数据都是一致的，如下所示

#session1

postgres=# create table t\_repeatable\_read (id int,mc text);

NOTICE: Replica identity is needed for shard table, please add to this table through "alter table" command.

CREATE TABLE

postgres=# insert into t\_repeatable\_read values(1,'tbase');

INSERT 0 1

postgres=# begin isolation level repeatable read ;

BEGIN

postgres=# select \* from t\_repeatable\_read ;

id | mc

----+-------

1 | tbase

(1 row)

#session2

postgres=# insert into t\_repeatable\_read values(1,'pgxz');

INSERT 0 1

postgres=# select \* from t\_repeatable\_read;

id | mc

----+-------

1 | tbase

1 | pgxz

(2 rows)

#session1

postgres=# select \* from t\_repeatable\_read ;

id | mc

----+-------

1 | tbase

(1 row)

postgres=#

### 9.1.5、行锁在事务中的运用

#### 9.1.5.1、环境准备

postgres=# create table t\_row\_lock(id int,mc text,primary key (id));

NOTICE: Replica identity is needed for shard table, please add to this table through "alter table" command.

CREATE TABLE

postgres=#

postgres=# insert into t\_row\_lock values(1,'tbase'),(2,'pgxz');

INSERT 0 2

postgres=# select \* from t\_row\_lock;

id | mc

----+-------

1 | tbase

2 | pgxz

(2 rows)

#### 9.1.5.2、直接update获取

#session1

postgres=# begin;

BEGIN

postgres=# set lock\_timeout to 1;

SET

postgres=# update t\_row\_lock set mc='postgres' where mc='pgxz';

UPDATE 1

postgres=#

#session2

postgres=# begin;

BEGIN

postgres=# set lock\_timeout to 1;

SET

postgres=# update t\_row\_lock set mc='postgresql' where mc='tbase';

UPDATE 1

postgres=#

上面session1与session2分别持有mc=pgxz行和mc=tbase的行锁

#### 9.1.5.3、select...for update获取

#session1

postgres=#

BEGIN

postgres=# set lock\_timeout to 1;

SET

postgres=# select \* from t\_row\_lock where mc='pgxz' for update;

id | mc

----+------

2 | pgxz

(1 row)

#session2

postgres=# begin;

BEGIN

postgres=# set lock\_timeout to 1;

SET

postgres=# select \* from t\_row\_lock where mc='tbase' for update;

id | mc

----+------

2 | pgxz

(1 row)

上面session1与session2分别持有mc=pgxz行和mc=tbase的行锁

#### 9.1.5.4、与mysql获取行级锁的区别

mysql> select version();

+-----------+

| version() |

+-----------+

| 5.6.36 |

+-----------+

1 row in set (0.00 sec)

#sessin1

mysql> begin;

Query OK, 0 rows affected (0.00 sec)

mysql> select \* from t\_row\_lock where mc='pgxz' for update;

+----+------+

| id | mc |

+----+------+

| 2 | pgxz |

+----+------+

1 row in set (0.00 sec)

#session2

mysql> select \* from t\_row\_lock where mc='tbase' for update;

ERROR 1205 (HY000): Lock wait timeout exceeded; try restarting transaction

mysql>

这是因为mysql要使用行级锁需要有索引来配合使用，如下所示,使用id主键来获取行锁

#session1

mysql> begin;

Query OK, 0 rows affected (0.00 sec)

mysql> select \* from t\_row\_lock where id=1 for update;

+----+-------+

| id | mc |

+----+-------+

| 1 | tbase |

+----+-------+

1 row in set (0.00 sec)

#session2

mysql> begin;

Query OK, 0 rows affected (0.00 sec)

mysql> select \* from t\_row\_lock where id=2 for update;

+----+------+

| id | mc |

+----+------+

| 2 | pgxz |

+----+------+

1 row in set (0.00 sec)

## 9.2、TBase相关开发规范

### 9.2.1、命名规范

1）、DB object: database, schema, table, column, view, index, sequence, function, trigger等名称：

(1) 建议使用小写字母、数字、下划线的组合。

(2) 建议不使用双引号即"包围，除非必须包含大写字母或空格等特殊字符。

(3) 长度不能超过63个字符。

(4) 不建议以pg\_开头或者pgxc\_（避免与系统DB object混淆），不建议以数字开头。

(5) 禁止使用 SQL 关键字例如 type, order 等。

2）、 table能包含的column数目,根据字段类型的不同，数目在 250 到 1600 之间；

3）、临时或备份的DB object:table,view 等,建议加上日期, 如dba\_ops.b2c\_product\_summay\_2014\_07\_12 (其中dba\_ops 为DBA专用schema)；

4）、index命名规则为: 普通索引为 表名\_列名\_idx，唯一索引表名\_列名\_uidx,如student\_name\_idx,student\_id\_uidx。

### 9.2.2、COLUMN设计

1）、建议能用数值类型的，就不用字符类型；

2）、建议能用varchar(N) 就不用char(N),以利于节省存储空间；

3）、建议能用varchar(N) 就不用text,varchar；

4）、建议使用default NULL,而不用default '',以节省存储空间；

5）、建议如有国际货业务的话，使用timestamp with time zone(timestamptz),而不用timestamp without time zone,避免时间函数在对于不同时区的时间点返回值不同,也为业务国际化扫清障碍；

6）、建议使用NUMERIC(precision, scale)来存储货币金额和其它要求精确计算的数值, 而不建议使用real, double precision；

### 9.2.3、Constraints 设计

1）、建议每个table都使用shard key做为主键或者唯一索引。

2）、建议建表时一步到位把主键或者唯一过引也一起建立。

3）、注意，非shard key不可以建立primary key或者 unique index。

### 9.2.4、Index 设计

1）、TBase提供的index类型: B-tree, Hash, GiST (Generalized Search Tree), SP-GiST (space-partitioned GiST), GIN (Generalized Inverted Index), BRIN (Block Range Index),目前不建议使用Hash，通常情况下使用B-tree；

2）、建议create 或 drop index 时,加 CONCURRENTLY参数,这是个好习惯，达到与写入数据并发的效果；

3）、建议对于频繁update, delete的包含于index 定义中的column的table, 用create index CONCURRENTLY , drop index CONCURRENTLY 的方式进行维护其对应index；

4）、建议用unique index 代替unique constraints,便于后续维护；

5）、建议对where 中带多个字段and条件的高频 query，参考数据分布情况，建多个字段的联合index；

6）、建议对固定条件的（一般有特定业务含义）且选择比好（数据占比低）的query，建带where的Partial Indexes；

select \* from test where status=1 and col=?; -- 其中status=1为固定的条件

create index on test (col) where status=1;

7）、建议对经常使用表达式作为查询条件的query，可以使用表达式或函数索引加速query；

select \* from test where exp(xxx);

create index on test ( exp(xxx) );

8）、建议不要建过多index，一般不要超过6个，核心table（产品，订单）可适当增加index个数。

### 9.2.5、关于NULL

1）、 NULL 的判断：IS NULL ，IS NOT NULL；

2）、 注意boolean 类型取值 true，false， NULL；

3）、小心NOT IN 集合中带有NULL元素；

postgres=# select \* from tbase;

id | nickname

----+---------------

1 | hello TBase

2 | TBase好牛

3 | TBase好牛

4 | TBase default

(4 rows)

postgres=# select \* from tbase where id not in (null);

id | nickname

----+----------

(0 rows)

4）、建议对字符串型NULL值处理后，进行 || 操作；

postgres=# select id,nickname from tbase limit 1;

id | nickname

----+-------------

1 | hello TBase

(1 row)

postgres=# select id,nickname||null from tbase limit 1;

id | ?column?

----+----------

1 |

(1 row)

postgres=# select id,nickname||coalesce(null,'') from tbase limit 1;

id | ?column?

----+-------------

1 | hello TBase

(1 row)

5）、建议使用count(1) 或count(\*) 来统计行数，而不建议使用count(col) 来统计行数，因为NULL值不会计入；

注意: count(多列列名)时，多列列名必须使用括号，例如count( (col1,col2,col3) ); 注意多列的count，即使所有列都为NULL，该行也被计数，所以效果与count(\*) 一致；

postgres=# select \* from tbase ;

id | nickname

----+---------------

1 | hello TBase

2 | TBase好牛

5 |

3 | TBase好牛

4 | TBase default

(5 rows)

postgres=# select count(1) from tbase;

count

-------

5

(1 row)

postgres=# select count(\*) from tbase;

count

-------

5

(1 row)

postgres=# select count(nickname) from tbase;

count

-------

4

(1 row)

postgres=# select count((id,nickname)) from tbase;

count

-------

5

(1 row)

6）、count(distinct col) 计算某列的非NULL不重复数量，NULL不被计数；

注意: count(distinct (col1,col2,...) ) 计算多列的唯一值时，NULL会被计数，同时NULL与NULL会被认为是相同的；

postgres=# select count(distinct nickname) from tbase;

count

-------

3

(1 row)

postgres=# select count(distinct (id,nickname)) from tbase;

count

-------

5

(1 row)

7）、两个null的对比方法

postgres=# select null is not distinct from null as TBasenull;

TBasenull

-----------

t

(1 row)

### 9.2.6、开发相关规范

1. 、 建议对DB object 尤其是COLUMN 加COMMENT，便于后续新人了解业务及维护；

注释前后的数据表可读性对比，有注释的一看就明白

postgres=# \d+ TBase\_main

Table "public.tbase\_main"

Column | Type | Modifiers | Storage | Stats target | Description

--------+---------+-----------+----------+--------------+-------------

id | integer | | plain | |

mc | text | | extended | |

Indexes:

"TBase\_main\_id\_uidx" UNIQUE, btree (id)

Has OIDs: no

Distribute By SHARD(id)

Location Nodes: ALL DATANODES

postgres=# comment on column TBase\_main.id is 'id号';

COMMENT

postgres=# comment on column TBase\_main.mc is '产品名称';

COMMENT

postgres=# \d+ TBase\_main

Table "public.tbase\_main"

Column | Type | Modifiers | Storage | Stats target | Description

--------+---------+-----------+----------+--------------+-------------

id | integer | | plain | | id号

mc | text | | extended | | 产品名称

Indexes:

"TBase\_main\_id\_uidx" UNIQUE, btree (id)

Has OIDs: no

Distribute By SHARD(id)

Location Nodes: ALL DATANODES

2）、 建议非必须时避免select \*，只取所需字段，以减少包括不限于网络带宽消耗

postgres=# explain (verbose) select \* from tbase\_main where id=1;

QUERY PLAN

---------------------------------------------------------------------------------------------

Index Scan using TBase\_main\_id\_uidx on public.tbase\_main (cost=0.15..8.17 rows=1 width=36)

Output: id, mc

Index Cond: (TBase\_main.id = 1)

(3 rows)

postgres=# explain (verbose) select tableoid from tbase\_main where id=1;

QUERY PLAN

--------------------------------------------------------------------------------------------

Index Scan using TBase\_main\_id\_uidx on public.tbase\_main (cost=0.15..8.17 rows=1 width=4)

Output: tableoid

Index Cond: (TBase\_main.id = 1)

(3 rows)

\*是返回36个字符，而另一个一条记录只能4个字段的长度

1. 、 建议update 时尽量做 <> 判断,比如update table\_a set column\_b = c where column\_b <> c；

postgres=# update tbase\_main set mc='TBase' ;

UPDATE 1

postgres=# select xmin,\* from tbase\_main;

xmin | id | mc

------+----+-------

2562 | 1 | TBase

(1 row)

postgres=# update tbase\_main set mc='TBase' ;

UPDATE 1

postgres=# select xmin,\* from tbase\_main;

xmin | id | mc

------+----+-------

2564 | 1 | TBase

(1 row)

postgres=# update tbase\_main set mc='TBase' where mc!='TBase';

UPDATE 0

postgres=# select xmin,\* from tbase\_main;

xmin | id | mc

------+----+-------

2564 | 1 | TBase

(1 row)

上面的效果是一样的，但带条件的更新不会产生一个新的版本记录，不需要系统执行vacuum回收垃圾数据。

1. 、 建议将单个事务的多条SQL操作,分解、拆分，或者不放在一个事务里，让每个事务的粒度尽可能小，尽量lock少的资源，避免lock 、dead lock的产生；

#sesseion1把所有数据都更新而不提交，一下子锁了2000千万条记录

postgres=# begin;

BEGIN

postgres=# update tbase\_main set mc='TBase\_1.3';

UPDATE 200000000

#sesseion2 等待

postgres=# update tbase\_main set mc='TBase\_1.4' where id=1;

#sesseion3 等待

postgres=# update tbase\_main set mc='TBase\_1.5' where id=2;

如果#sesseion1分布批更新的话，如下所示

postgres=# begin;

BEGIN

postgres=# update tbase\_main set mc='TBase\_1.3' where id>0 and id <=100000;

UPDATE 100000

postgres=#COMMIT;

postgres=# begin;

BEGIN

postgres=# update tbase\_main set mc='TBase\_1.3' where id>100000 and id <=200000;

UPDATE 100000

postgres=#COMMIT;

则session2和session3中就能部分提前完成，这样可以避免大量的锁等待和出现大量的session占用系统资源，在做全表更新时请使用这种方法来执行

1. 、 建议 大批量的数据入库时， 使用copy ，不建议使用insert，以提高写入速度；

postgres=# insert into tbase\_main select t,'xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx' from generate\_series(1,100000) as t;

INSERT 0 100000

Time: 9511.755 ms

postgres=# copy TBase\_main to '/data/pgxz/TBase\_main.txt';

COPY 100002

Time: 179.428 ms

postgres=# copy TBase\_main from '/data/pgxz/TBase\_main.txt';

COPY 100002

Time: 1625.803 ms

postgres=#

性能相差5倍

1. 、 建议对报表类的或生成基础数据的查询，使用物化视图(MATERIALIZED VIEW)定期固化数据快照， 避免对多表（尤其多写频繁的表）重复跑相同的查询，且物化视图支持 REFRESH MATERIALIZED VIEW CONCURRENTLY, 支持并发更新；

如有一个程序需要不断查询TBase\_main的总记录数，那么我们这样做

postgres=# select count(1) from tbase\_main;

count

--------

200004

(1 row)

Time: 27.948 ms

postgres=# create MATERIALIZED VIEW TBase\_main\_count as select count(1) as num from tbase\_main;

SELECT 1

Time: 322.372 ms

postgres=# select num from TBase\_main\_count ;

num

--------

200004

(1 row)

Time: 0.421 ms

性能提高上百倍

有数据变化时刷新方法

postgres=# copy TBase\_main from '/data/pgxz/TBase\_main.txt';

COPY 100002

Time: 1201.774 ms

postgres=# select count(1) from tbase\_main;

count

--------

300006

(1 row)

Time: 23.164 ms

postgres=# REFRESH MATERIALIZED VIEW TBase\_main\_count;

REFRESH MATERIALIZED VIEW

Time: 49.486 ms

postgres=# select num from tbase\_main\_count ;

num

--------

300006

(1 row)

Time: 0.301 ms

1. 、 建议复杂的统计查询可以尝试窗口函数

请参考5.3.2SQL-CET用法

1. 、两表join时尽量的使用分布key进行join

所似在建立业务的主表，明细表时，就需要使用他们的关联键来做分布键，如下所示

[pgxz@VM\_0\_29\_centos pgxz]$ psql -p 15001

psql (PostgreSQL 10 (TBase 2.01))

Type "help" for help.

postgres=# create table tbase\_main(id integer,mc text) distribute by shard(id);

CREATE TABLE

postgres=# create table tbase\_detail(id integer,TBase\_main\_id integer,mc text) distribute by shard(TBase\_main\_id);

CREATE TABLE

postgres=# explain select TBase\_detail.\* from tbase\_main,TBase\_detail where TBase\_main.id=TBase\_detail.TBase\_main\_id;

QUERY PLAN

----------------------------------------------------------------------------

Data Node Scan on "\_\_REMOTE\_FQS\_QUERY\_\_" (cost=0.00..0.00 rows=0 width=0)

Node/s: dn001, dn002

(2 rows)

postgres=# explain (verbose) select TBase\_detail.\* from tbase\_main,TBase\_detail where TBase\_main.id=TBase\_detail.TBase\_main\_id;

QUERY PLAN

------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

Data Node Scan on "\_\_REMOTE\_FQS\_QUERY\_\_" (cost=0.00..0.00 rows=0 width=0)

Output: TBase\_detail.id, TBase\_detail.TBase\_main\_id, TBase\_detail.mc

Node/s: dn001, dn002

Remote query: SELECT TBase\_detail.id, TBase\_detail.TBase\_main\_id, TBase\_detail.mc FROM public.tbase\_main, public.tbase\_detail WHERE (TBase\_main.id = TBase\_detail.TBase\_main\_id)

(4 rows)

postgres=#

1. 、分布键用唯一索引代替主键

postgres=# create unique index TBase\_main\_id\_uidx on TBase\_main using btree(id);

CREATE INDEX

因为唯一索引后期的维护成本比主键要低很多

1. 、分布键无法建立唯一索引则要建立普通索引，提高查询的效率

postgres=# create index TBase\_detail\_TBase\_main\_id\_idx on TBase\_detail using btree(TBase\_main\_id);

CREATE INDEX

这样两表在join查询时返回少量数据时的效率才会高。

1. 、不要对字段建立外键

目前TBase还不支持多dn外键约束，除非你能确定数据关联键的数据全部落在同一个dn上面

## 9.3、高级sql语句编写

### 9.3.1、窗口函数的使用

#### 9.3.1.1、环境准备

drop table if exists bills ;

create table bills

(

id serial not null,

goodsdesc text not null,

beginunit text not null,

begincity text not null,

pubtime timestamp not null,

amount float8 not null default 0,

primary key (id)

);

COMMENT ON TABLE bills is '运单记录';

COMMENT ON COLUMN bills.id IS 'id号';

COMMENT ON COLUMN bills.goodsdesc IS '货物名称';

COMMENT ON COLUMN bills.beginunit IS '启运省份';

COMMENT ON COLUMN bills.begincity IS '启运城市';

COMMENT ON COLUMN bills.pubtime IS '发布时间';

COMMENT ON COLUMN bills.amount IS '运费';

INSERT INTO bills(id,goodsdesc,beginunit,begincity,pubtime,amount)

VALUES(default,'衣服','海南省','三亚市','2015-10-05 09:32:01',ROUND((random()\*10000)::NUMERIC,2));

INSERT INTO bills(id,goodsdesc,beginunit,begincity,pubtime,amount)

VALUES(default,'建筑设备','福建省','三明市','2015-10-05 07:21:22',ROUND((random()\*10000)::NUMERIC,2));

INSERT INTO bills(id,goodsdesc,beginunit,begincity,pubtime,amount)

VALUES(default,'设备','福建省','三明市','2015-10-05 11:21:54',ROUND((random()\*10000)::NUMERIC,2));

INSERT INTO bills(id,goodsdesc,beginunit,begincity,pubtime,amount)

VALUES(default,'普货','福建省','三明市','2015-10-05 15:19:17',ROUND((random()\*10000)::NUMERIC,2));

INSERT INTO bills(id,goodsdesc,beginunit,begincity,pubtime,amount)

VALUES(default,'5 0铲车，后八轮翻斗车','河南省','三门峡市','2015-10-05 07:53:13',ROUND((random()\*10000)::NUMERIC,2));

INSERT INTO bills(id,goodsdesc,beginunit,begincity,pubtime,amount)

VALUES(default,'鲜香菇2000斤','河南省','三门峡市','2015-10-05 10:38:29',ROUND((random()\*10000)::NUMERIC,2));

INSERT INTO bills(id,goodsdesc,beginunit,begincity,pubtime,amount)

VALUES(default,'旋挖附件38吨','河南省','三门峡市','2015-10-05 10:48:38',ROUND((random()\*10000)::NUMERIC,2));

INSERT INTO bills(id,goodsdesc,beginunit,begincity,pubtime,amount)

VALUES(default,'旋挖附件35吨','河南省','三门峡市','2015-10-05 10:48:38',ROUND((random()\*10000)::NUMERIC,2));

INSERT INTO bills(id,goodsdesc,beginunit,begincity,pubtime,amount)

VALUES(default,'旋挖附件39吨','河南省','三门峡市','2015-10-05 11:38:38',ROUND((random()\*10000)::NUMERIC,2));

INSERT INTO bills(id,goodsdesc,beginunit,begincity,pubtime,amount)

VALUES(default,'设备','上海市','上海市','2015-10-05 07:59:35',ROUND((random()\*10000)::NUMERIC,2));

INSERT INTO bills(id,goodsdesc,beginunit,begincity,pubtime,amount)

VALUES(default,'普货40吨需13米半挂一辆','上海市','上海市','2015-10-05 08:13:59',ROUND((random()\*10000)::NUMERIC,2));

#### 9.3.1.2、row\_number() --返回行号，不分组

postgres=# select row\_number() over(),\* from bills limit 2;

row\_number | id | goodsdesc | beginunit | begincity | pubtime | amount

------------+----+-----------+-----------+-----------+---------------------+---------

1 | 1 | 衣服 | 海南省 | 三亚市 | 2015-10-05 09:32:01 | 1915.86

2 | 2 | 建筑设备 | 福建省 | 三明市 | 2015-10-05 07:21:22 | 2022.31

(2 rows)

postgres=# select row\_number() over(),\* from bills limit 2 offset 2;

row\_number | id | goodsdesc | beginunit | begincity | pubtime | amount

------------+----+-----------------------+-----------+-----------+---------------------+---------

3 | 6 | 5 0铲车，后八轮翻斗车 | 河南省 | 三门峡市 | 2015-10-05 07:53:13 | 1030.9

4 | 8 | 旋挖附件38吨 | 河南省 | 三门峡市 | 2015-10-05 10:48:38 | 5365.04

(2 rows)

#### 9.3.1.3、row\_number() --返回行号，按amount排序

postgres=# select row\_number() over(order by amount),\* from bills;

row\_number | id | goodsdesc | beginunit | begincity | pubtime | amount

------------+----+------------------------+-----------+-----------+---------------------+---------

1 | 11 | 设备 | 上海市 | 上海市 | 2015-10-05 07:59:35 | 971.54

2 | 6 | 5 0铲车，后八轮翻斗车 | 河南省 | 三门峡市 | 2015-10-05 07:53:13 | 1030.9

3 | 4 | 普货 | 福建省 | 三明市 | 2015-10-05 15:19:17 | 1316.27

4 | 1 | 衣服 | 海南省 | 三亚市 | 2015-10-05 09:32:01 | 1915.86

5 | 2 | 建筑设备 | 福建省 | 三明市 | 2015-10-05 07:21:22 | 2022.31

6 | 7 | 鲜香菇2000斤 | 河南省 | 三门峡市 | 2015-10-05 10:38:29 | 4182.68

7 | 8 | 旋挖附件38吨 | 河南省 | 三门峡市 | 2015-10-05 10:48:38 | 5365.04

8 | 10 | 旋挖附件39吨 | 河南省 | 三门峡市 | 2015-10-05 11:38:38 | 8290.5

9 | 3 | 设备 | 福建省 | 三明市 | 2015-10-05 11:21:54 | 8771.11

10 | 9 | 旋挖附件35吨 | 河南省 | 三门峡市 | 2015-10-05 10:48:38 | 9621.37

11 | 5 | 普货40吨需13米半挂一辆 | 上海市 | 上海市 | 2015-10-05 08:13:59 | 9886.15

(11 rows)

#### 9.3.1.4、row\_number() --返回行号，按begincity分组，pubtime排序，注意绿色记录行号不间断

postgres=# select row\_number() over(partition by begincity order by pubtime),\* from bills;

row\_number | id | goodsdesc | beginunit | begincity | pubtime | amount

------------+----+------------------------+-----------+-----------+---------------------+---------

1 | 1 | 衣服 | 海南省 | 三亚市 | 2015-10-05 09:32:01 | 1915.86

1 | 2 | 建筑设备 | 福建省 | 三明市 | 2015-10-05 07:21:22 | 2022.31

2 | 3 | 设备 | 福建省 | 三明市 | 2015-10-05 11:21:54 | 8771.11

3 | 4 | 普货 | 福建省 | 三明市 | 2015-10-05 15:19:17 | 1316.27

1 | 6 | 5 0铲车，后八轮翻斗车 | 河南省 | 三门峡市 | 2015-10-05 07:53:13 | 1030.9

2 | 7 | 鲜香菇2000斤 | 河南省 | 三门峡市 | 2015-10-05 10:38:29 | 4182.68

3 | 8 | 旋挖附件38吨 | 河南省 | 三门峡市 | 2015-10-05 10:48:38 | 5365.04

4 | 9 | 旋挖附件35吨 | 河南省 | 三门峡市 | 2015-10-05 10:48:38 | 9621.37

5 | 10 | 旋挖附件39吨 | 河南省 | 三门峡市 | 2015-10-05 11:38:38 | 8290.5

1 | 11 | 设备 | 上海市 | 上海市 | 2015-10-05 07:59:35 | 971.54

2 | 5 | 普货40吨需13米半挂一辆 | 上海市 | 上海市 | 2015-10-05 08:13:59 | 9886.15

(11 rows)

#### 9.3.1.5、rank()--返回行号,对比值重复时行号重复并间断，即返回1,2,2,4...

postgres=# select rank() over(partition by begincity order by pubtime),\* from bills;

rank | id | goodsdesc | beginunit | begincity | pubtime | amount

------+----+------------------------+-----------+-----------+---------------------+---------

1 | 1 | 衣服 | 海南省 | 三亚市 | 2015-10-05 09:32:01 | 1915.86

1 | 2 | 建筑设备 | 福建省 | 三明市 | 2015-10-05 07:21:22 | 2022.31

2 | 3 | 设备 | 福建省 | 三明市 | 2015-10-05 11:21:54 | 8771.11

3 | 4 | 普货 | 福建省 | 三明市 | 2015-10-05 15:19:17 | 1316.27

1 | 6 | 5 0铲车，后八轮翻斗车 | 河南省 | 三门峡市 | 2015-10-05 07:53:13 | 1030.9

2 | 7 | 鲜香菇2000斤 | 河南省 | 三门峡市 | 2015-10-05 10:38:29 | 4182.68

3 | 8 | 旋挖附件38吨 | 河南省 | 三门峡市 | 2015-10-05 10:48:38 | 5365.04

3 | 9 | 旋挖附件35吨 | 河南省 | 三门峡市 | 2015-10-05 10:48:38 | 9621.37

5 | 10 | 旋挖附件39吨 | 河南省 | 三门峡市 | 2015-10-05 11:38:38 | 8290.5

1 | 11 | 设备 | 上海市 | 上海市 | 2015-10-05 07:59:35 | 971.54

2 | 5 | 普货40吨需13米半挂一辆 | 上海市 | 上海市 | 2015-10-05 08:13:59 | 9886.15

(11 rows)

#### 9.3.1.6、dance\_rank()--返回行号,对比值重复时行号重复但不间断，即返回1,2,2,3...

postgres=# select dense\_rank() over(partition by begincity order by pubtime),\* from bills;

dense\_rank | id | goodsdesc | beginunit | begincity | pubtime | amount

------------+----+------------------------+-----------+-----------+---------------------+---------

1 | 1 | 衣服 | 海南省 | 三亚市 | 2015-10-05 09:32:01 | 1915.86

1 | 2 | 建筑设备 | 福建省 | 三明市 | 2015-10-05 07:21:22 | 2022.31

2 | 3 | 设备 | 福建省 | 三明市 | 2015-10-05 11:21:54 | 8771.11

3 | 4 | 普货 | 福建省 | 三明市 | 2015-10-05 15:19:17 | 1316.27

1 | 6 | 5 0铲车，后八轮翻斗车 | 河南省 | 三门峡市 | 2015-10-05 07:53:13 | 1030.9

2 | 7 | 鲜香菇2000斤 | 河南省 | 三门峡市 | 2015-10-05 10:38:29 | 4182.68

3 | 8 | 旋挖附件38吨 | 河南省 | 三门峡市 | 2015-10-05 10:48:38 | 5365.04

3 | 9 | 旋挖附件35吨 | 河南省 | 三门峡市 | 2015-10-05 10:48:38 | 9621.37

4 | 10 | 旋挖附件39吨 | 河南省 | 三门峡市 | 2015-10-05 11:38:38 | 8290.5

1 | 11 | 设备 | 上海市 | 上海市 | 2015-10-05 07:59:35 | 971.54

2 | 5 | 普货40吨需13米半挂一辆 | 上海市 | 上海市 | 2015-10-05 08:13:59 | 9886.15

(11 rows)

#### 9.3.1.7、percent\_rank()从当前开始，计算在分组中的比例 (行号-1)\*(1/(总记录数-1))

postgres=# select percent\_rank() over(partition by begincity order by id),\* from bills;

percent\_rank | id | goodsdesc | beginunit | begincity | pubtime | amount

--------------+----+------------------------+-----------+-----------+---------------------+---------

0 | 1 | 衣服 | 海南省 | 三亚市 | 2015-10-05 09:32:01 | 1915.86

0 | 2 | 建筑设备 | 福建省 | 三明市 | 2015-10-05 07:21:22 | 2022.31

0.5 | 3 | 设备 | 福建省 | 三明市 | 2015-10-05 11:21:54 | 8771.11

1 | 4 | 普货 | 福建省 | 三明市 | 2015-10-05 15:19:17 | 1316.27

0 | 6 | 5 0铲车，后八轮翻斗车 | 河南省 | 三门峡市 | 2015-10-05 07:53:13 | 1030.9

0.25 | 7 | 鲜香菇2000斤 | 河南省 | 三门峡市 | 2015-10-05 10:38:29 | 4182.68

0.5 | 8 | 旋挖附件38吨 | 河南省 | 三门峡市 | 2015-10-05 10:48:38 | 5365.04

0.75 | 9 | 旋挖附件35吨 | 河南省 | 三门峡市 | 2015-10-05 10:48:38 | 9621.37

1 | 10 | 旋挖附件39吨 | 河南省 | 三门峡市 | 2015-10-05 11:38:38 | 8290.5

0 | 5 | 普货40吨需13米半挂一辆 | 上海市 | 上海市 | 2015-10-05 08:13:59 | 9886.15

1 | 11 | 设备 | 上海市 | 上海市 | 2015-10-05 07:59:35 | 971.54

(11 rows)

#### 9.3.1.8、cume\_dist() --返回行数除以记录数值

postgres=# select ROUND((cume\_dist() over(partition by begincity order by id))::NUMERIC,2) AS cume\_dist,\* from bills;

cume\_dist | id | goodsdesc | beginunit | begincity | pubtime | amount

-----------+----+------------------------+-----------+-----------+---------------------+---------

1.00 | 1 | 衣服 | 海南省 | 三亚市 | 2015-10-05 09:32:01 | 1915.86

0.33 | 2 | 建筑设备 | 福建省 | 三明市 | 2015-10-05 07:21:22 | 2022.31

0.67 | 3 | 设备 | 福建省 | 三明市 | 2015-10-05 11:21:54 | 8771.11

1.00 | 4 | 普货 | 福建省 | 三明市 | 2015-10-05 15:19:17 | 1316.27

0.20 | 6 | 5 0铲车，后八轮翻斗车 | 河南省 | 三门峡市 | 2015-10-05 07:53:13 | 1030.9

0.40 | 7 | 鲜香菇2000斤 | 河南省 | 三门峡市 | 2015-10-05 10:38:29 | 4182.68

0.60 | 8 | 旋挖附件38吨 | 河南省 | 三门峡市 | 2015-10-05 10:48:38 | 5365.04

0.80 | 9 | 旋挖附件35吨 | 河南省 | 三门峡市 | 2015-10-05 10:48:38 | 9621.37

1.00 | 10 | 旋挖附件39吨 | 河南省 | 三门峡市 | 2015-10-05 11:38:38 | 8290.5

0.50 | 5 | 普货40吨需13米半挂一辆 | 上海市 | 上海市 | 2015-10-05 08:13:59 | 9886.15

1.00 | 11 | 设备 | 上海市 | 上海市 | 2015-10-05 07:59:35 | 971.54

(11 rows)

#### 9.3.1.9、ntile(分组数量)--让所有记录尽可以的均匀分布

postgres=# select ntile(2) over(partition by begincity order by id),\* from bills;

ntile | id | goodsdesc | beginunit | begincity | pubtime | amount

-------+----+------------------------+-----------+-----------+---------------------+---------

1 | 1 | 衣服 | 海南省 | 三亚市 | 2015-10-05 09:32:01 | 1915.86

1 | 2 | 建筑设备 | 福建省 | 三明市 | 2015-10-05 07:21:22 | 2022.31

1 | 3 | 设备 | 福建省 | 三明市 | 2015-10-05 11:21:54 | 8771.11

2 | 4 | 普货 | 福建省 | 三明市 | 2015-10-05 15:19:17 | 1316.27

1 | 6 | 5 0铲车，后八轮翻斗车 | 河南省 | 三门峡市 | 2015-10-05 07:53:13 | 1030.9

1 | 7 | 鲜香菇2000斤 | 河南省 | 三门峡市 | 2015-10-05 10:38:29 | 4182.68

1 | 8 | 旋挖附件38吨 | 河南省 | 三门峡市 | 2015-10-05 10:48:38 | 5365.04

2 | 9 | 旋挖附件35吨 | 河南省 | 三门峡市 | 2015-10-05 10:48:38 | 9621.37

2 | 10 | 旋挖附件39吨 | 河南省 | 三门峡市 | 2015-10-05 11:38:38 | 8290.5

1 | 5 | 普货40吨需13米半挂一辆 | 上海市 | 上海市 | 2015-10-05 08:13:59 | 9886.15

2 | 11 | 设备 | 上海市 | 上海市 | 2015-10-05 07:59:35 | 971.54

(11 rows)

postgres=# select ntile(3) over(partition by begincity order by id),\* from bills;

ntile | id | goodsdesc | beginunit | begincity | pubtime | amount

-------+----+------------------------+-----------+-----------+---------------------+---------

1 | 1 | 衣服 | 海南省 | 三亚市 | 2015-10-05 09:32:01 | 1915.86

1 | 2 | 建筑设备 | 福建省 | 三明市 | 2015-10-05 07:21:22 | 2022.31

2 | 3 | 设备 | 福建省 | 三明市 | 2015-10-05 11:21:54 | 8771.11

3 | 4 | 普货 | 福建省 | 三明市 | 2015-10-05 15:19:17 | 1316.27

1 | 6 | 5 0铲车，后八轮翻斗车 | 河南省 | 三门峡市 | 2015-10-05 07:53:13 | 1030.9

1 | 7 | 鲜香菇2000斤 | 河南省 | 三门峡市 | 2015-10-05 10:38:29 | 4182.68

2 | 8 | 旋挖附件38吨 | 河南省 | 三门峡市 | 2015-10-05 10:48:38 | 5365.04

2 | 9 | 旋挖附件35吨 | 河南省 | 三门峡市 | 2015-10-05 10:48:38 | 9621.37

3 | 10 | 旋挖附件39吨 | 河南省 | 三门峡市 | 2015-10-05 11:38:38 | 8290.5

1 | 5 | 普货40吨需13米半挂一辆 | 上海市 | 上海市 | 2015-10-05 08:13:59 | 9886.15

2 | 11 | 设备 | 上海市 | 上海市 | 2015-10-05 07:59:35 | 971.54

(11 rows)

#### 9.3.1.10、lag(value any [, offset integer [, default any]] )--返回偏移量值

offset integer是偏移值，正数时取前值，负数时取后值，没有取到值时用default代替

postgres=# select lag(amount,1,null) over(partition by begincity order by id),\* from bills;

lag | id | goodsdesc | beginunit | begincity | pubtime | amount

---------+----+------------------------+-----------+-----------+---------------------+---------

| 1 | 衣服 | 海南省 | 三亚市 | 2015-10-05 09:32:01 | 1915.86

| 2 | 建筑设备 | 福建省 | 三明市 | 2015-10-05 07:21:22 | 2022.31

2022.31 | 3 | 设备 | 福建省 | 三明市 | 2015-10-05 11:21:54 | 8771.11

8771.11 | 4 | 普货 | 福建省 | 三明市 | 2015-10-05 15:19:17 | 1316.27

| 6 | 5 0铲车，后八轮翻斗车 | 河南省 | 三门峡市 | 2015-10-05 07:53:13 | 1030.9

1030.9 | 7 | 鲜香菇2000斤 | 河南省 | 三门峡市 | 2015-10-05 10:38:29 | 4182.68

4182.68 | 8 | 旋挖附件38吨 | 河南省 | 三门峡市 | 2015-10-05 10:48:38 | 5365.04

5365.04 | 9 | 旋挖附件35吨 | 河南省 | 三门峡市 | 2015-10-05 10:48:38 | 9621.37

9621.37 | 10 | 旋挖附件39吨 | 河南省 | 三门峡市 | 2015-10-05 11:38:38 | 8290.5

| 5 | 普货40吨需13米半挂一辆 | 上海市 | 上海市 | 2015-10-05 08:13:59 | 9886.15

9886.15 | 11 | 设备 | 上海市 | 上海市 | 2015-10-05 07:59:35 | 971.54

(11 rows)

postgres=# select lag(amount,2,0::float8) over(partition by begincity order by id),\* from bills;

lag | id | goodsdesc | beginunit | begincity | pubtime | amount

---------+----+------------------------+-----------+-----------+---------------------+---------

0 | 1 | 衣服 | 海南省 | 三亚市 | 2015-10-05 09:32:01 | 1915.86

0 | 2 | 建筑设备 | 福建省 | 三明市 | 2015-10-05 07:21:22 | 2022.31

0 | 3 | 设备 | 福建省 | 三明市 | 2015-10-05 11:21:54 | 8771.11

2022.31 | 4 | 普货 | 福建省 | 三明市 | 2015-10-05 15:19:17 | 1316.27

0 | 6 | 5 0铲车，后八轮翻斗车 | 河南省 | 三门峡市 | 2015-10-05 07:53:13 | 1030.9

0 | 7 | 鲜香菇2000斤 | 河南省 | 三门峡市 | 2015-10-05 10:38:29 | 4182.68

1030.9 | 8 | 旋挖附件38吨 | 河南省 | 三门峡市 | 2015-10-05 10:48:38 | 5365.04

4182.68 | 9 | 旋挖附件35吨 | 河南省 | 三门峡市 | 2015-10-05 10:48:38 | 9621.37

5365.04 | 10 | 旋挖附件39吨 | 河南省 | 三门峡市 | 2015-10-05 11:38:38 | 8290.5

0 | 5 | 普货40吨需13米半挂一辆 | 上海市 | 上海市 | 2015-10-05 08:13:59 | 9886.15

0 | 11 | 设备 | 上海市 | 上海市 | 2015-10-05 07:59:35 | 971.54

(11 rows)

postgres=# select lag(amount,-2,0::float8) over(partition by begincity order by id),\* from bills;

lag | id | goodsdesc | beginunit | begincity | pubtime | amount

---------+----+------------------------+-----------+-----------+---------------------+---------

0 | 1 | 衣服 | 海南省 | 三亚市 | 2015-10-05 09:32:01 | 1915.86

1316.27 | 2 | 建筑设备 | 福建省 | 三明市 | 2015-10-05 07:21:22 | 2022.31

0 | 3 | 设备 | 福建省 | 三明市 | 2015-10-05 11:21:54 | 8771.11

0 | 4 | 普货 | 福建省 | 三明市 | 2015-10-05 15:19:17 | 1316.27

5365.04 | 6 | 5 0铲车，后八轮翻斗车 | 河南省 | 三门峡市 | 2015-10-05 07:53:13 | 1030.9

9621.37 | 7 | 鲜香菇2000斤 | 河南省 | 三门峡市 | 2015-10-05 10:38:29 | 4182.68

8290.5 | 8 | 旋挖附件38吨 | 河南省 | 三门峡市 | 2015-10-05 10:48:38 | 5365.04

0 | 9 | 旋挖附件35吨 | 河南省 | 三门峡市 | 2015-10-05 10:48:38 | 9621.37

0 | 10 | 旋挖附件39吨 | 河南省 | 三门峡市 | 2015-10-05 11:38:38 | 8290.5

0 | 5 | 普货40吨需13米半挂一辆 | 上海市 | 上海市 | 2015-10-05 08:13:59 | 9886.15

0 | 11 | 设备 | 上海市 | 上海市 | 2015-10-05 07:59:35 | 971.54

(11 rows)

#### 9.3.1.11、lead(value any [,offset integer [, default any]] )--返回偏移量值

offset integer是偏移值，正数时取后值，负数时取前值，没有取到值时用default代替

postgres=# select lead(amount,2,null) over(partition by begincity order by id),\* from bills;

lead | id | goodsdesc | beginunit | begincity | pubtime | amount

---------+----+------------------------+-----------+-----------+---------------------+---------

| 1 | 衣服 | 海南省 | 三亚市 | 2015-10-05 09:32:01 | 1915.86

1316.27 | 2 | 建筑设备 | 福建省 | 三明市 | 2015-10-05 07:21:22 | 2022.31

| 3 | 设备 | 福建省 | 三明市 | 2015-10-05 11:21:54 | 8771.11

| 4 | 普货 | 福建省 | 三明市 | 2015-10-05 15:19:17 | 1316.27

5365.04 | 6 | 5 0铲车，后八轮翻斗车 | 河南省 | 三门峡市 | 2015-10-05 07:53:13 | 1030.9

9621.37 | 7 | 鲜香菇2000斤 | 河南省 | 三门峡市 | 2015-10-05 10:38:29 | 4182.68

8290.5 | 8 | 旋挖附件38吨 | 河南省 | 三门峡市 | 2015-10-05 10:48:38 | 5365.04

| 9 | 旋挖附件35吨 | 河南省 | 三门峡市 | 2015-10-05 10:48:38 | 9621.37

| 10 | 旋挖附件39吨 | 河南省 | 三门峡市 | 2015-10-05 11:38:38 | 8290.5

| 5 | 普货40吨需13米半挂一辆 | 上海市 | 上海市 | 2015-10-05 08:13:59 | 9886.15

| 11 | 设备 | 上海市 | 上海市 | 2015-10-05 07:59:35 | 971.54

(11 rows)

postgres=# select lead(amount,-2,null) over(partition by begincity order by id),\* from bills;

lead | id | goodsdesc | beginunit | begincity | pubtime | amount

---------+----+------------------------+-----------+-----------+---------------------+---------

| 1 | 衣服 | 海南省 | 三亚市 | 2015-10-05 09:32:01 | 1915.86

| 2 | 建筑设备 | 福建省 | 三明市 | 2015-10-05 07:21:22 | 2022.31

| 3 | 设备 | 福建省 | 三明市 | 2015-10-05 11:21:54 | 8771.11

2022.31 | 4 | 普货 | 福建省 | 三明市 | 2015-10-05 15:19:17 | 1316.27

| 6 | 5 0铲车，后八轮翻斗车 | 河南省 | 三门峡市 | 2015-10-05 07:53:13 | 1030.9

| 7 | 鲜香菇2000斤 | 河南省 | 三门峡市 | 2015-10-05 10:38:29 | 4182.68

1030.9 | 8 | 旋挖附件38吨 | 河南省 | 三门峡市 | 2015-10-05 10:48:38 | 5365.04

4182.68 | 9 | 旋挖附件35吨 | 河南省 | 三门峡市 | 2015-10-05 10:48:38 | 9621.37

5365.04 | 10 | 旋挖附件39吨 | 河南省 | 三门峡市 | 2015-10-05 11:38:38 | 8290.5

| 5 | 普货40吨需13米半挂一辆 | 上海市 | 上海市 | 2015-10-05 08:13:59 | 9886.15

| 11 | 设备 | 上海市 | 上海市 | 2015-10-05 07:59:35 | 971.54

(11 rows)

#### 9.3.1.12、first\_value(value any)返回第一值

postgres=# select first\_value(amount) over(partition by begincity order by id),\* from bills;

first\_value | id | goodsdesc | beginunit | begincity | pubtime | amount

-------------+----+------------------------+-----------+-----------+---------------------+---------

1915.86 | 1 | 衣服 | 海南省 | 三亚市 | 2015-10-05 09:32:01 | 1915.86

2022.31 | 2 | 建筑设备 | 福建省 | 三明市 | 2015-10-05 07:21:22 | 2022.31

2022.31 | 3 | 设备 | 福建省 | 三明市 | 2015-10-05 11:21:54 | 8771.11

2022.31 | 4 | 普货 | 福建省 | 三明市 | 2015-10-05 15:19:17 | 1316.27

1030.9 | 6 | 5 0铲车，后八轮翻斗车 | 河南省 | 三门峡市 | 2015-10-05 07:53:13 | 1030.9

1030.9 | 7 | 鲜香菇2000斤 | 河南省 | 三门峡市 | 2015-10-05 10:38:29 | 4182.68

1030.9 | 8 | 旋挖附件38吨 | 河南省 | 三门峡市 | 2015-10-05 10:48:38 | 5365.04

1030.9 | 9 | 旋挖附件35吨 | 河南省 | 三门峡市 | 2015-10-05 10:48:38 | 9621.37

1030.9 | 10 | 旋挖附件39吨 | 河南省 | 三门峡市 | 2015-10-05 11:38:38 | 8290.5

9886.15 | 5 | 普货40吨需13米半挂一辆 | 上海市 | 上海市 | 2015-10-05 08:13:59 | 9886.15

9886.15 | 11 | 设备 | 上海市 | 上海市 | 2015-10-05 07:59:35 | 971.54

(11 rows)

#### 9.3.1.13、last\_value(value any)返回最后值

postgres=# select last\_value(amount) over(partition by begincity order by pubtime),\* FROM bills;

last\_value | id | goodsdesc | beginunit | begincity | pubtime | amount

------------+----+------------------------+-----------+-----------+---------------------+---------

1915.86 | 1 | 衣服 | 海南省 | 三亚市 | 2015-10-05 09:32:01 | 1915.86

2022.31 | 2 | 建筑设备 | 福建省 | 三明市 | 2015-10-05 07:21:22 | 2022.31

8771.11 | 3 | 设备 | 福建省 | 三明市 | 2015-10-05 11:21:54 | 8771.11

1316.27 | 4 | 普货 | 福建省 | 三明市 | 2015-10-05 15:19:17 | 1316.27

1030.9 | 6 | 5 0铲车，后八轮翻斗车 | 河南省 | 三门峡市 | 2015-10-05 07:53:13 | 1030.9

4182.68 | 7 | 鲜香菇2000斤 | 河南省 | 三门峡市 | 2015-10-05 10:38:29 | 4182.68

9621.37 | 8 | 旋挖附件38吨 | 河南省 | 三门峡市 | 2015-10-05 10:48:38 | 5365.04

9621.37 | 9 | 旋挖附件35吨 | 河南省 | 三门峡市 | 2015-10-05 10:48:38 | 9621.37

8290.5 | 10 | 旋挖附件39吨 | 河南省 | 三门峡市 | 2015-10-05 11:38:38 | 8290.5

971.54 | 11 | 设备 | 上海市 | 上海市 | 2015-10-05 07:59:35 | 971.54

9886.15 | 5 | 普货40吨需13米半挂一辆 | 上海市 | 上海市 | 2015-10-05 08:13:59 | 9886.15

(11 rows)

postgres=# select last\_value(amount) over(partition by begincity),\* FROM bills;

last\_value | id | goodsdesc | beginunit | begincity | pubtime | amount

------------+----+------------------------+-----------+-----------+---------------------+---------

1915.86 | 1 | 衣服 | 海南省 | 三亚市 | 2015-10-05 09:32:01 | 1915.86

1316.27 | 2 | 建筑设备 | 福建省 | 三明市 | 2015-10-05 07:21:22 | 2022.31

1316.27 | 3 | 设备 | 福建省 | 三明市 | 2015-10-05 11:21:54 | 8771.11

1316.27 | 4 | 普货 | 福建省 | 三明市 | 2015-10-05 15:19:17 | 1316.27

9621.37 | 7 | 鲜香菇2000斤 | 河南省 | 三门峡市 | 2015-10-05 10:38:29 | 4182.68

9621.37 | 10 | 旋挖附件39吨 | 河南省 | 三门峡市 | 2015-10-05 11:38:38 | 8290.5

9621.37 | 6 | 5 0铲车，后八轮翻斗车 | 河南省 | 三门峡市 | 2015-10-05 07:53:13 | 1030.9

9621.37 | 8 | 旋挖附件38吨 | 河南省 | 三门峡市 | 2015-10-05 10:48:38 | 5365.04

9621.37 | 9 | 旋挖附件35吨 | 河南省 | 三门峡市 | 2015-10-05 10:48:38 | 9621.37

971.54 | 5 | 普货40吨需13米半挂一辆 | 上海市 | 上海市 | 2015-10-05 08:13:59 | 9886.15

971.54 | 11 | 设备 | 上海市 | 上海市 | 2015-10-05 07:59:35 | 971.54

(11 rows)

注意不要加上order by id，默认情况下，带了order by 参数会从分组的起始值开始一直叠加，

直到当前值（不是当前记录）不同为止，当忽略order by 参数则是整个分组。下面通过修改分组的统计范围就可以实现order by参数取最后值

postgres=# select last\_value(amount) over(partition by begincity order by id range between unbounded preceding and unbounded following),\* FROM bills;

last\_value | id | goodsdesc | beginunit | begincity | pubtime | amount

------------+----+------------------------+-----------+-----------+---------------------+---------

1915.86 | 1 | 衣服 | 海南省 | 三亚市 | 2015-10-05 09:32:01 | 1915.86

1316.27 | 2 | 建筑设备 | 福建省 | 三明市 | 2015-10-05 07:21:22 | 2022.31

1316.27 | 3 | 设备 | 福建省 | 三明市 | 2015-10-05 11:21:54 | 8771.11

1316.27 | 4 | 普货 | 福建省 | 三明市 | 2015-10-05 15:19:17 | 1316.27

8290.5 | 6 | 5 0铲车，后八轮翻斗车 | 河南省 | 三门峡市 | 2015-10-05 07:53:13 | 1030.9

8290.5 | 7 | 鲜香菇2000斤 | 河南省 | 三门峡市 | 2015-10-05 10:38:29 | 4182.68

8290.5 | 8 | 旋挖附件38吨 | 河南省 | 三门峡市 | 2015-10-05 10:48:38 | 5365.04

8290.5 | 9 | 旋挖附件35吨 | 河南省 | 三门峡市 | 2015-10-05 10:48:38 | 9621.37

8290.5 | 10 | 旋挖附件39吨 | 河南省 | 三门峡市 | 2015-10-05 11:38:38 | 8290.5

971.54 | 5 | 普货40吨需13米半挂一辆 | 上海市 | 上海市 | 2015-10-05 08:13:59 | 9886.15

971.54 | 11 | 设备 | 上海市 | 上海市 | 2015-10-05 07:59:35 | 971.54

(11 rows)

#### 9.3.1.14、nth\_value(value any, nth integer)：返回窗口框架中的指定值

postgres=# select nth\_value(amount,2) over(partition by begincity order by id),\* from bills;

nth\_value | id | goodsdesc | beginunit | begincity | pubtime | amount

-----------+----+------------------------+-----------+-----------+---------------------+---------

| 1 | 衣服 | 海南省 | 三亚市 | 2015-10-05 09:32:01 | 1915.86

| 2 | 建筑设备 | 福建省 | 三明市 | 2015-10-05 07:21:22 | 2022.31

8771.11 | 3 | 设备 | 福建省 | 三明市 | 2015-10-05 11:21:54 | 8771.11

8771.11 | 4 | 普货 | 福建省 | 三明市 | 2015-10-05 15:19:17 | 1316.27

| 6 | 5 0铲车，后八轮翻斗车 | 河南省 | 三门峡市 | 2015-10-05 07:53:13 | 1030.9

4182.68 | 7 | 鲜香菇2000斤 | 河南省 | 三门峡市 | 2015-10-05 10:38:29 | 4182.68

4182.68 | 8 | 旋挖附件38吨 | 河南省 | 三门峡市 | 2015-10-05 10:48:38 | 5365.04

4182.68 | 9 | 旋挖附件35吨 | 河南省 | 三门峡市 | 2015-10-05 10:48:38 | 9621.37

4182.68 | 10 | 旋挖附件39吨 | 河南省 | 三门峡市 | 2015-10-05 11:38:38 | 8290.5

| 5 | 普货40吨需13米半挂一辆 | 上海市 | 上海市 | 2015-10-05 08:13:59 | 9886.15

971.54 | 11 | 设备 | 上海市 | 上海市 | 2015-10-05 07:59:35 | 971.54

(11 rows)

#### 9.3.1.15、统计各个城市的总运费及平均每单的运费

postgres=# select sum(amount) over(partition by begincity),avg(amount) over(partition by begincity),begincity,amount from bills;

sum | avg | begincity | amount

----------+------------------+-----------+---------

1915.86 | 1915.86 | 三亚市 | 1915.86

12109.69 | 4036.56333333333 | 三明市 | 2022.31

12109.69 | 4036.56333333333 | 三明市 | 8771.11

12109.69 | 4036.56333333333 | 三明市 | 1316.27

28490.49 | 5698.098 | 三门峡市 | 4182.68

28490.49 | 5698.098 | 三门峡市 | 8290.5

28490.49 | 5698.098 | 三门峡市 | 1030.9

28490.49 | 5698.098 | 三门峡市 | 5365.04

28490.49 | 5698.098 | 三门峡市 | 9621.37

10857.69 | 5428.845 | 上海市 | 9886.15

10857.69 | 5428.845 | 上海市 | 971.54

(11 rows)

#### 9.3.1.16、窗口函数别名使用

postgres=# select sum(amount) over w,avg(amount) over w,begincity,amount from bills window w as (partition by begincity);

sum | avg | begincity | amount

----------+------------------+-----------+---------

1915.86 | 1915.86 | 三亚市 | 1915.86

12109.69 | 4036.56333333333 | 三明市 | 2022.31

12109.69 | 4036.56333333333 | 三明市 | 8771.11

12109.69 | 4036.56333333333 | 三明市 | 1316.27

28490.49 | 5698.098 | 三门峡市 | 4182.68

28490.49 | 5698.098 | 三门峡市 | 8290.5

28490.49 | 5698.098 | 三门峡市 | 1030.9

28490.49 | 5698.098 | 三门峡市 | 5365.04

28490.49 | 5698.098 | 三门峡市 | 9621.37

10857.69 | 5428.845 | 上海市 | 9886.15

10857.69 | 5428.845 | 上海市 | 971.54

(11 rows)

#### 9.3.1.17、获取每个城市运费前两名订单

postgres=# select \* from (select row\_number() over(partition by begincity order by amount desc),\* from bills) where row\_number<3;

row\_number | id | goodsdesc | beginunit | begincity | pubtime | amount

------------+----+------------------------+-----------+-----------+---------------------+---------

1 | 1 | 衣服 | 海南省 | 三亚市 | 2015-10-05 09:32:01 | 1915.86

1 | 3 | 设备 | 福建省 | 三明市 | 2015-10-05 11:21:54 | 8771.11

2 | 2 | 建筑设备 | 福建省 | 三明市 | 2015-10-05 07:21:22 | 2022.31

1 | 9 | 旋挖附件35吨 | 河南省 | 三门峡市 | 2015-10-05 10:48:38 | 9621.37

2 | 10 | 旋挖附件39吨 | 河南省 | 三门峡市 | 2015-10-05 11:38:38 | 8290.5

1 | 5 | 普货40吨需13米半挂一辆 | 上海市 | 上海市 | 2015-10-05 08:13:59 | 9886.15

2 | 11 | 设备 | 上海市 | 上海市 | 2015-10-05 07:59:35 | 971.54

(7 rows)

# 10、TBase-PL/pgsql开发

## 10.1、应用程序语法介绍

### 10.1.1、建立函数语法

CREATE [OR REPLACE] FUNCTION [模式名.]函数名 ([参数模式 [参数名] 数据类型 [default 默认值] [,…]]) RETRUNS [SETOF] 数据类型 AS

[标签]

[DECLARE

--变量定义]

BEGIN

--注释

/\*注释\*/

--语句执行

END;

[标签]

LANGUAGE PLPGSQL;

### 10.1.2、[OR REPLACE] 更新函数介绍

带OR REPLACE的作用就函数存在时则替换的功能，建立PL/pgsql函数时不带OR REPLACE关键字， 则遇到函数已经存系统则会报错，如下所示

postgres=# select prosrc from pg\_proc where proname='f';

prosrc

--------------------------

+

BEGIN +

RAISE NOTICE 'TBase';+

END; +

(1 行记录)

postgres=# CREATE FUNCTION f() RETURNS VOID AS

postgres-# $$

postgres$# BEGIN

postgres$# RAISE NOTICE 'Hello ,TBase';

postgres$# END;

postgres$# $$

postgres-# LANGUAGE PLPGSQL;

ERROR: function "f" already exists with same argument types

postgres=# CREATE OR REPLACE FUNCTION f() RETURNS VOID AS

postgres-# $$

postgres$# BEGIN

postgres$# RAISE NOTICE 'Hello ,TBase';

postgres$# END;

postgres$# $$

postgres-# LANGUAGE PLPGSQL;

CREATE FUNCTION

postgres=# select prosrc from pg\_proc where proname='f';

prosrc

---------------------------------

+

BEGIN +

RAISE NOTICE 'Hello ,TBase';+

END; +

(1 行记录)

postgres=# SELECT f();

NOTICE: Hello ,TBase

f

---

(1 行记录)

### 10.1.3、[模式名.]函数名介绍

建立函数名称，模式名可以指定，也可以不指定，不指存则存放在当前模式下，如上面例子就没有指定模式名，则就存放在当前模式下，如下所示

postgres=# select \* from pg\_namespace;

nspname | nspowner | nspacl

--------------------+----------+-------------------------------------

pg\_toast | 10 |

pg\_temp\_1 | 10 |

pg\_toast\_temp\_1 | 10 |

pg\_catalog | 10 | {postgres=UC/postgres,=U/postgres}

public | 10 | {postgres=UC/postgres,=UC/postgres}

information\_schema | 10 | {postgres=UC/postgres,=U/postgres}

(6 行记录)

postgres=# show search\_path;

search\_path

----------------

"$user",public

(1 行记录)

postgres=# select pg\_namespace.nspname,pg\_proc.prosrc from pg\_proc,pg\_namespace where

pg\_proc.pronamespace=pg\_namespace.oid and pg\_proc.proname='f';

nspname | prosrc

---------+---------------------------------

public | +

| BEGIN +

| RAISE NOTICE 'Hello ,TBase';+

| END; +

|

(1 行记录)

因为$user模式不存在，所以存在public模式下

## 10.2、参数详细介绍

### 10.2.1、参数模式

参数模式总共有三种，分别是IN (传入参数，这个是默认方式)、OUT(返回值参数)、INOUT(传入返回值参数)，下面说说这几个参数模式的用法法

#### 10.2.1.1、IN模式

IN模式指的是执行函数时需要输入参数值，如下所示

postgres=# CREATE OR REPLACE FUNCTION f1(IN a\_xm text) RETURNS TEXT AS

postgres-# $$

postgres$# BEGIN

postgres$# RETURN a\_xm;

postgres$# END;

postgres$# $$

postgres-# LANGUAGE PLPGSQL;

CREATE FUNCTION

postgres=#

postgres=# SELECT f1('TBase');

f1

------

TBase

(1 行记录)

postgres=# CREATE OR REPLACE FUNCTION f1(a\_xm text) RETURNS TEXT AS

postgres-# $$

postgres$# BEGIN

postgres$# RETURN a\_xm;

postgres$# END;

postgres$# $$

postgres-# LANGUAGE PLPGSQL;

CREATE FUNCTION

postgres=#

postgres=# SELECT \* FROM f1('TBase');

f1

------

TBase

(1 行记录)

上面两种方式定义的参数效果是一样的

#### 10.2.1.2、OUT模式

OUT模式参数是指定了函数执行时返回的字段名及类型

postgres=# CREATE OR REPLACE FUNCTION f1(OUT a\_xm TEXT) RETURNS TEXT AS

postgres-# $$

postgres$# BEGIN

postgres$# a\_xm:='TBase';

postgres$# END;

postgres$# $$

postgres-# LANGUAGE PLPGSQL;

CREATE FUNCTION

postgres=# SELECT \* FROM f1();

a\_xm

------

TBase

(1 行记录)

采用OUT模式参数不能用RETURN返回，而是要对返回的OUT参数直接付值。返回值类型与参数的数据类型必需一致。参数名就是返回的字段名

#### 10.2.1.3、INOUT模式

INOUT模式是指参数即传入，同时又指定了返回值的字段名和类型

postgres=# CREATE OR REPLACE FUNCTION f1(INOUT a\_xm text) RETURNS TEXT AS

postgres-# $$

postgres$# BEGIN

postgres$# END;

postgres$# $$

postgres-# LANGUAGE PLPGSQL;

CREATE FUNCTION

postgres=#

postgres=# SELECT \* FROM f1('TBase');

a\_xm

------

TBase

(1 行记录)

值得注意的是，上面的函数跟下面的函数是相同的，即重新定义会覆盖掉

postgres=# CREATE OR REPLACE FUNCTION f1(IN a\_xm text) RETURNS TEXT AS

postgres-# $$

postgres$# BEGIN

postgres$# RETURN 'TBase';

postgres$# END;

postgres$# $$

postgres-# LANGUAGE PLPGSQL;

CREATE FUNCTION

postgres=# SELECT \* FROM f1('TBase');

f1

------

TBase

(1 行记录)

#### 10.2.1.4、VARIADIC模式

VARIADICS模式是参数个数可变模式，系统用一个数组对传入的参数进行处理，VARIADIC参数必需是所有最后一个声明，如下所示

postgres=# CREATE OR REPLACE FUNCTION f1(VARIADIC a\_int integer[]) RETURNS void AS

postgres-# $$

postgres$# BEGIN

postgres$# RAISE NOTICE 'a\_int = %',a\_int;

postgres$# RAISE NOTICE 'a\_int[1] = %',a\_int[1];

postgres$# END;

postgres$# $$

postgres-# LANGUAGE PLPGSQL;

CREATE FUNCTION

postgres=#

postgres=# SELECT f1(1);

NOTICE: a\_int = {1}

NOTICE: a\_int[1] = 1

f1

----

(1 行记录)

postgres=# SELECT f1(1,2);

NOTICE: a\_int = {1,2}

NOTICE: a\_int[1] = 1

f1

----

(1 行记录)

postgres=# CREATE OR REPLACE FUNCTION f1(a\_xm TEXT,VARIADIC a\_int integer[]) RETURNS void AS

postgres-# $$

postgres$# BEGIN

postgres$# RAISE NOTICE 'a\_int = %',a\_int;

postgres$# RAISE NOTICE 'a\_int[1] = %',a\_int[1];

postgres$# RAISE NOTICE 'a\_xm = %',a\_xm;

postgres$# END;

postgres$# $$

postgres-# LANGUAGE PLPGSQL;

CREATE FUNCTION

postgres=#

postgres=# SELECT f1('TBase',1,2);

NOTICE: a\_int = {1,2}

NOTICE: a\_int[1] = 1

NOTICE: a\_xm = TBasea

f1

----

(1 行记录)

### 10.2.2、参数引用

PL/pgsql函数的参数是以$1,$2这样标识符来进行传递，也支持命名参数，所以参数的定义可以用下面的方式

#### 10.2.2.1、无命名参数

postgres=# CREATE OR REPLACE FUNCTION f2(text) RETURNS TEXT AS

postgres-# $$

postgres$# BEGIN

postgres$# RETURN $1;

postgres$# END;

postgres$# $$

postgres-# LANGUAGE PLPGSQL;

CREATE FUNCTION

postgres=#

postgres=# SELECT \* FROM f2('TBase');

f2

------

TBase

(1 行记录)

#### 10.2.2.2、给标识符指定别名

postgres=# CREATE OR REPLACE FUNCTION f2(text) RETURNS TEXT AS

postgres-# $$

postgres$# DECLARE

postgres$# a\_xm ALIAS FOR $1; --a\_xm是$1的别名

postgres$# BEGIN

postgres$# RETURN a\_xm;

postgres$# END;

postgres$# $$

postgres-# LANGUAGE PLPGSQL;

CREATE FUNCTION

postgres=#

postgres=# SELECT \* FROM f2('TBase');

f2

------

TBase

(1 行记录)

#### 10.2.2.3、命名参数

postgres=# CREATE OR REPLACE FUNCTION f2(a\_xm text) RETURNS TEXT AS

postgres-# $$

postgres$# DECLARE

postgres$# v\_xm ALIAS FOR $1;

postgres$# BEGIN

postgres$# RAISE NOTICE 'a\_xm = % ; v\_xm = % ; $1 = %',a\_xm,v\_xm,$1;

postgres$# RETURN $1;

postgres$# END;

postgres$# $$

postgres-# LANGUAGE PLPGSQL;

CREATE FUNCTION

postgres=#

postgres=# SELECT \* FROM f2('TBase');

NOTICE: a\_xm = TBase ; v\_xm = TBase ; $1 = TBase

f2

------

TBase

(1 行记录)

### 10.2.3、参数数据类型

数据类型(可以有模式修饰)，可以是基本类型，复合类型、域类型、游标、或者可以引用一个现有表类型、字段类型(建立时转换为对应的类型)、还可以是多态类型anyelement、anyarray，也可以是各种数据类型的数组形式

#### 10.2.3.1、基本类型

postgres=# CREATE OR REPLACE FUNCTION f3 (a\_int integer,a\_str text) RETURNS VOID AS

postgres-# $$

postgres$# BEGIN

postgres$# RAISE NOTICE 'a\_int = % ; a\_str = %',a\_int,a\_str;

postgres$# END;

postgres$# $$

postgres-# LANGUAGE PLPGSQL;

CREATE FUNCTION

postgres=# SELECT \* FROM f3(1,'TBase');

NOTICE: a\_int = 1 ; a\_str = TBase

f3

----

(1 行记录)

postgres=# CREATE OR REPLACE FUNCTION f3 (a\_int integer[],a\_str text[]) RETURNS VOID AS

postgres-# $$

postgres$# BEGIN

postgres$# RAISE NOTICE 'a\_int = % ; a\_str = %',a\_int,a\_str;

postgres$# END;

postgres$# $$

postgres-# LANGUAGE PLPGSQL;

CREATE FUNCTION

postgres=# SELECT f3(ARRAY[1,2,3],ARRAY['TBase','pgxz']);

NOTICE: a\_int = {1,2,3} ; a\_str = {TBase,pgxz}

f3

----

(1 行记录)

#### 10.2.3.2、复合类型

postgres=# CREATE TYPE t\_per AS

postgres-# (

postgres(# id integer,

postgres(# mc text

postgres(# );

ERROR: type "t\_per" already exists

postgres=# CREATE OR REPLACE FUNCTION f3 (a\_row public.t\_per) RETURNS VOID AS

postgres-# $$

postgres$# BEGIN

postgres$# RAISE NOTICE 'id = % ; mc = %',a\_row.id,a\_row.mc;

postgres$# END;

postgres$# $$

postgres-# LANGUAGE PLPGSQL;

CREATE FUNCTION

postgres=#

postgres=# SELECT f3(ROW(1,'TBase')::public.t\_per);

NOTICE: id = 1 ; mc = TBase

f3

----

(1 行记录)

postgres=# CREATE OR REPLACE FUNCTION f3 (a\_rec public.t\_per[]) RETURNS VOID AS

postgres-# $$

postgres$# BEGIN

postgres$# RAISE NOTICE 'a\_rec = %',a\_rec;

postgres$# RAISE NOTICE 'a\_rec[1].id = %',a\_rec[1].id;

postgres$# END;

postgres$# $$

postgres-# LANGUAGE PLPGSQL;

CREATE FUNCTION

postgres=#

postgres=# SELECT f3(ARRAY[ROW(1,'TBase'),ROW(1,'pgxz')]::public.t\_per[]);

NOTICE: a\_rec = {"(1,TBase)","(1,pgxz)"}

NOTICE: a\_rec[1].id = 1

f3

----

(1 行记录)

#### 10.2.3.3、行类型

postgres=# \d t

资料表 "public.t"

栏位 | 型别 | 修饰词

------+---------+--------

id | integer |

mc | text |

postgres=# CREATE OR REPLACE FUNCTION f3 (a\_row public.t) RETURNS VOID AS

postgres-# $$

postgres$# BEGIN

postgres$# RAISE NOTICE 'id = % ; mc = %',a\_row.id,a\_row.mc;

postgres$# END;

postgres$# $$

postgres-# LANGUAGE PLPGSQL;

CREATE FUNCTION

postgres=#

postgres=# SELECT f3(ROW(1,'TBase'));

NOTICE: id = 1 ; mc = TBase

f3

----

(1 行记录)

postgres=# SELECT f3(t.\*) FROM t LIMIT 1;

NOTICE: id = 1 ; mc = TBase

f3

----

(1 行记录)

postgres=# CREATE OR REPLACE FUNCTION f3 (a\_rec public.t[]) RETURNS VOID AS

postgres-# $$

postgres$# BEGIN

postgres$# RAISE NOTICE 'a\_rec = %',a\_rec;

postgres$# RAISE NOTICE 'a\_rec[1].id = %',a\_rec[1].id;

postgres$# END;

postgres$# $$

postgres-# LANGUAGE PLPGSQL;

CREATE FUNCTION

postgres=#

postgres=# SELECT f3(array[row(1,'TBase'),row(1,'pgxz')]::public.t[]);

NOTICE: a\_rec = {"(1,TBase)","(1,pgxz)"}

NOTICE: a\_rec[1].id = 1

f3

----

(1 行记录)

postgres=# SELECT f3(array[t.\*,t.\*]::public.t[]) FROM t LIMIT 2;

NOTICE: a\_rec = {"(1,TBase)","(1,TBase)"}

NOTICE: a\_rec[1].id = 1

NOTICE: a\_rec = {"(2,pgxz)","(2,pgxz)"}

NOTICE: a\_rec[1].id = 2

f3

----

(2 行记录)

#### 10.2.3.4、域类型

postgres=# CREATE DOMAIN xb AS TEXT CHECK

postgres-# (

postgres(# VALUE = '男'

postgres(# OR VALUE ='女'

postgres(# OR VALUE = ''

postgres(# );

CREATE DOMAIN

postgres=#

postgres=# CREATE OR REPLACE FUNCTION f4 (a\_xb public.xb) RETURNS VOID AS

postgres-# $$

postgres$# BEGIN

postgres$# RAISE NOTICE 'a\_xb = %',a\_xb;

postgres$# END;

postgres$# $$

postgres-# LANGUAGE PLPGSQL;

CREATE FUNCTION

postgres=#

postgres=# SELECT \* FROM f4('男');

NOTICE: a\_xb = 男

f4

----

(1 行记录)

postgres=# SELECT \* FROM f4('她');

ERROR: value for domain xb violates check constraint "xb\_check"

postgres=#

域类型输入参数值时会检查是否违反规则

#### 10.2.3.5、游标类型

postgres=# CREATE OR REPLACE FUNCTION f5 (a\_ref refcursor) RETURNS void AS

postgres-# $$

postgres$# DECLARE

postgres$# v\_rec record;

postgres$# BEGIN

postgres$# OPEN a\_ref FOR SELECT \* FROM t LIMIT 1;

postgres$# FETCH a\_ref INTO v\_rec;

postgres$# RAISE NOTICE 'v\_rec = % ',v\_rec;

postgres$# END;

postgres$# $$

postgres-# LANGUAGE PLPGSQL;

CREATE FUNCTION

postgres=#

postgres=# SELECT \* FROM f5('a');

NOTICE: v\_rec = (1,TBase)

f5

----

(1 行记录)

postgres=# CREATE OR REPLACE FUNCTION f6 (a\_ref refcursor) RETURNS refcursor AS

postgres-# $$

postgres$# BEGIN

postgres$# OPEN a\_ref FOR SELECT \* FROM t LIMIT 1;

postgres$# RETURN a\_ref;

postgres$# END;

postgres$# $$

postgres-# LANGUAGE PLPGSQL;

CREATE FUNCTION

postgres=#

#注意这里需要开启一个事务

postgres=# BEGIN;

BEGIN

postgres=# SELECT \* FROM f6('a');

f6

----

a

(1 行记录)

postgres=# FETCH ALL FROM a;

id | mc

----+------

1 | TBase

(1 行记录)

#### 10.2.3.6、多态类型

postgres=# CREATE OR REPLACE FUNCTION f\_any(a\_arg anyelement) RETURNS VOID AS

postgres-# $$

postgres$# BEGIN

postgres$# RAISE NOTICE '%',a\_arg;

postgres$# END;

postgres$# $$

postgres-# LANGUAGE PLPGSQL;

CREATE FUNCTION

postgres=#

postgres=# SELECT f\_any(1::integer);

NOTICE: 1

f\_any

-------

(1 行记录)

postgres=# SELECT f\_any('TBase'::TEXT);

NOTICE: TBase

f\_any

-------

(1 行记录)

postgres=# SELECT f\_any(ROW(1,'TBase')::public.t\_rec);

NOTICE: (1,TBase)

f\_any

-------

(1 行记录)

postgres=# SELECT f\_any(ARRAY[1,2]::INTEGER[]);

NOTICE: {1,2}

f\_any

-------

(1 行记录)

postgres=# SELECT f\_any(ARRAY[[1,2],[3,4],[5,6]]::INTEGER[][][]);

NOTICE: {{1,2},{3,4},{5,6}}

f\_any

-------

(1 行记录)

注意多态类型参数函数调用时最好直接声明参数类型，否则有可能出错

postgres=# CREATE OR REPLACE FUNCTION f\_any\_array(a\_arg anyarray) RETURNS VOID AS

postgres-# $$

postgres$# BEGIN

postgres$# RAISE NOTICE '%',a\_arg;

postgres$# END;

postgres$# $$

postgres-# LANGUAGE PLPGSQL;

CREATE FUNCTION

postgres=#

postgres=# SELECT f\_any\_array(ARRAY['TBase','pgxz']::TEXT[]);

NOTICE: {TBase,pgxz}

f\_any\_array

-------------

(1 行记录)

postgres=# SELECT f\_any\_array(ARRAY[ARRAY['TBase','pgxz'],ARRAY['TBase','Tencent']]::TEXT[][]);

NOTICE: {{TBase,pgxz},{TBase,Tencent}}

f\_any\_array

-------------

(1 行记录)

注意：Anyelement参数如果写成数组，其意义就跟anyarray参数一致，所以 f\_any(a\_arg anyelement)与f\_any(a\_arg anyarray)在调用f\_any(ARRAY[1,2])时就会出现函数不是唯一化的错误(ERROR: function f\_any(…) is not unique)提示

#### 10.2.3.7、参数默认值

PL/pgsql扩展语言函数支持给参数设置默认值

postgres=# CREATE OR REPLACE FUNCTION f7 (a\_int INTEGER DEFAULT 1) RETURNS VOID AS

postgres-# $$

postgres$# BEGIN

postgres$# RAISE NOTICE 'a\_int = %',a\_int;

postgres$# END;

postgres$# $$

postgres-# LANGUAGE PLPGSQL;

CREATE FUNCTION

postgres=#

postgres=# SELECT \* FROM f7();

NOTICE: a\_int = 1

f7

----

(1 行记录)

备注：如果原来存在一个f7()这样的函数，则上面的执行就会出错，因为系统无法清楚到你到底要执行那个函数，如下所示

postgres=# CREATE OR REPLACE FUNCTION f7() RETURNS void AS

postgres-# $$

postgres$# BEGIN

postgres$# RAISE NOTICE '无参数';

postgres$# END;

postgres$# $$

postgres-# LANGUAGE plpgsql ;

CREATE FUNCTION

postgres=#

postgres=# SELECT \* FROM f7();

ERROR: function f7() is not unique

第1行SELECT \* FROM f7();

^

提示: Could not choose a best candidate function. You might need to add explicit type casts.

postgres=#

出错提示，f7()函数不是唯一的，这是使用上一个需要特别注意的地方

## 10.3、返回值详细介绍

### 10.3.1、返回值介绍

返回值可以是一个简单数据类型、复合类型、RECORD、已经存在的表行类型、表字段类型、游标、另外还可以返回一个记录集、如果不需要返回值，则可以用RETURN void。返回值的字段名及类型可以在参数在用OUT,INOUT模式中声明

### 10.3.2、返回值类型介绍

#### 10.3.2.1、没有返回值

postgres=# CREATE OR REPLACE FUNCTION f8() RETURNS void AS

postgres-# $$

postgres$# BEGIN

postgres$# RAISE NOTICE '不用返回值，函数体可以有或没有return语句';

postgres$# RETURN ;--这一句可以有，也可以没有

postgres$# END;

postgres$# $$

postgres-# LANGUAGE plpgsql;

CREATE FUNCTION

postgres=#

postgres=# SELECT \* FROM f8();

NOTICE: 不用返回值，函数体可以有或没有return语句

f8

----

(1 行记录)

#### 10.3.2.2、返回简单类型

postgres=# CREATE OR REPLACE FUNCTION f9() RETURNS TEXT AS

postgres-# $$

postgres$# BEGIN

postgres$# RETURN 'TBase';

postgres$# END;

postgres$# $$

postgres-# LANGUAGE plpgsql;

CREATE FUNCTION

postgres=#

postgres=# SELECT \* FROM f9() t(a\_xm);

a\_xm

------

TBase

(1 行记录)

postgres=#

postgres=# CREATE OR REPLACE FUNCTION f9(OUT a\_xm TEXT) RETURNS TEXT AS

postgres-# $$

postgres$# BEGIN

postgres$# a\_xm:='TBase';

postgres$# END;

postgres$# $$

postgres-# LANGUAGE plpgsql;

CREATE FUNCTION

postgres=#

postgres=# SELECT \* FROM f9();

a\_xm

------

TBase

(1 行记录)

上面两个函数其实就是同一个函数，建立时如果不加OR REPLACE则会提示已经存在

postgres=# CREATE OR REPLACE FUNCTION f10() RETURNS TEXT[] AS

postgres-# $$

postgres$# BEGIN

postgres$# RETURN ARRAY['TBase','pgxz'];

postgres$# END;

postgres$# $$

postgres-# LANGUAGE plpgsql;

CREATE FUNCTION

postgres=#

postgres=# SELECT \* FROM f10();

f10

---------------

{TBase,pgxz}

(1 行记录)

#### 10.3.2.3、返回一个复合类型

postgres=# CREATE TYPE t\_rec AS

postgres-# (

postgres(# id integer,

postgres(# mc text

postgres(# );

CREATE TYPE

postgres=#

postgres=# CREATE OR REPLACE FUNCTION f11() RETURNS t\_rec AS

postgres-# $$

postgres$# DECLARE

postgres$# v\_rec public.t\_rec;

postgres$# BEGIN

postgres$# v\_rec.id:=1;

postgres$# v\_rec.mc='TBase';

postgres$# RETURN v\_rec;

postgres$# END;

postgres$# $$

postgres-# LANGUAGE plpgsql;

CREATE FUNCTION

postgres=#

postgres=# SELECT \* FROM f11();

id | mc

----+------

1 | TBase

(1 行记录)

postgres=# CREATE OR REPLACE FUNCTION f12() RETURNS t\_rec[] AS

postgres-# $$

postgres$# BEGIN

postgres$# RETURN ARRAY[ROW(1,'TBase'),ROW(1,'pgxz')]::t\_rec[];

postgres$# END;

postgres$# $$

postgres-# LANGUAGE plpgsql;

CREATE FUNCTION

postgres=#

postgres=# SELECT \* FROM f12();

f12

---------------------------

{"(1,TBase)","(1,pgxz)"}

(1 行记录)

#### 10.3.2.4、返回行类型

postgres=# \d t

资料表 "public.t"

栏位 | 型别 | 修饰词

------+---------+--------

id | integer |

mc | text |

postgres=# CREATE OR REPLACE FUNCTION f13() RETURNS public.t AS

postgres-# $$

postgres$# DECLARE

postgres$# v\_rec public.t%ROWTYPE;

postgres$# BEGIN

postgres$# SELECT \* INTO v\_rec FROM public.t LIMIT 1;

postgres$# RETURN v\_rec;

postgres$# END;

postgres$# $$

postgres-# LANGUAGE plpgsql;

CREATE FUNCTION

postgres=#

postgres=# SELECT \* FROM f13();

id | mc

----+------

1 | TBase

(1 行记录)

postgres=# CREATE OR REPLACE FUNCTION f14() RETURNS public.t[] AS

postgres-# $$

postgres$# DECLARE

postgres$# v\_rec public.t[];

postgres$# BEGIN

postgres$# SELECT ARRAY[ROW(t.\*),ROW(t.\*)]::public.t[] INTO v\_rec FROM public.t LIMIT 1;

postgres$# RETURN v\_rec;

postgres$# END;

postgres$# $$

postgres-# LANGUAGE plpgsql;

CREATE FUNCTION

postgres=#

postgres=# SELECT \* FROM f14();

f14

-------------------------

{"(1,TBase)","(1,TBase)"}

1. 行记录)

#### 10.3.2.5、返回TABLE类型

postgres=# DROP FUNCTION f14();

DROP FUNCTION

postgres=# CREATE FUNCTION f14() RETURNS TABLE(a\_id integer, a\_nc text) AS

postgres-# $$

postgres$# BEGIN

postgres$# RETURN QUERY SELECT 1::integer,'TBase'::Text;

postgres$# END;

postgres$# $$LANGUAGE plpgsql;

CREATE FUNCTION

postgres=# SELECT \* FROM f14();

a\_id | a\_nc

------+-------

1 | TBase

(1 row)

#### 10.3.2.6、返回RECORD类型

postgres=# CREATE OR REPLACE FUNCTION f15() RETURNS RECORD AS

postgres-# $$

postgres$# DECLARE

postgres$# v\_rec RECORD;

postgres$# BEGIN

postgres$# v\_rec:=ROW(1::integer,'TBase'::text,'pgxz'::text);

postgres$# RETURN v\_rec;

postgres$# END;

postgres$# $$

postgres-# LANGUAGE plpgsql;

CREATE FUNCTION

postgres=# SELECT f15();

f15

-----------------

(1,TBase,pgxz)

(1 行记录)

postgres=# SELECT \* FROM f15() t(id integer,xm text,xl text);

id | xm | xl

----+------+--------

1 | TBase | pgxz

(1 行记录)

#### 10.3.2.7、返回一个游标

postgres=# CREATE OR REPLACE FUNCTION f16() RETURNS refcursor AS

postgres-# $$

postgres$# DECLARE

postgres$# v\_ref refcursor;

postgres$# BEGIN

postgres$# OPEN v\_ref FOR SELECT \* FROM public.t;

postgres$# RETURN v\_ref;

postgres$# END;

postgres$# $$

postgres-# LANGUAGE plpgsql;

CREATE FUNCTION

postgres=#

postgres=# BEGIN;

BEGIN

postgres=#

postgres=# SELECT \* FROM f16();

f15

--------------------

<unnamed portal 1>

(1 行记录)

postgres=# FETCH ALL FROM "<unnamed portal 1>";

id | mc

----+--------

1 | TBase

2 | pgxz

(2 行记录)

postgres=# END;

postgres=# CREATE OR REPLACE FUNCTION f16(a\_ref refcursor) RETURNS refcursor AS

postgres-# $$

postgres$# BEGIN

postgres$# OPEN a\_ref FOR SELECT \* FROM public.t;

postgres$# RETURN a\_ref;

postgres$# END;

postgres$# $$

postgres-# LANGUAGE plpgsql;

CREATE FUNCTION

postgres=#

postgres=# BEGIN;

BEGIN

postgres=# SELECT \* FROM f16('a');

f15

-----

a

(1 行记录)

postgres=# FETCH ALL FROM a;

id | mc

----+--------

1 | TBase

2 | pgxz

(2 行记录)

postgres=# END;

COMMIT

#### 10.3.2.8、返回记录集

postgres=# CREATE OR REPLACE FUNCTION f17() RETURNS SETOF TEXT AS

postgres-# $$

postgres$# BEGIN

postgres$# RETURN NEXT 'TBase'::text;

postgres$# RETURN NEXT 'pgxz'::text;

postgres$# RETURN ;--最后的RETURN可以加，也可以不加上去

postgres$# END;

postgres$# $$

postgres-# LANGUAGE PLPGSQL;

CREATE FUNCTION

postgres=#

postgres=# SELECT \* FROM f17();

f17

--------

TBase

pgxz

(2 行记录)

postgres=#

postgres=#

postgres=# CREATE OR REPLACE FUNCTION f18() RETURNS SETOF public.t AS

postgres-# $$

postgres$# DECLARE

postgres$# --使用行类型返回

postgres$# v\_rec public.t%ROWTYPE;

postgres$# BEGIN

postgres$# FOR v\_rec IN SELECT \* FROM t ORDER BY id LOOP

postgres$# RETURN NEXT v\_rec;

postgres$# END LOOP;

postgres$# RETURN ;--最后的RETURN可以加，也可以不加上去

postgres$# END;

postgres$# $$

postgres-# LANGUAGE PLPGSQL;

CREATE FUNCTION

postgres=#

postgres=# SELECT \* FROM f18();

id | mc

----+--------

1 | TBase

2 | pgxz

(2 行记录)

postgres=# \d t1

资料表 "public.t1"

栏位 | 型别 | 修饰词

------+---------------+--------

id | integer | 非空

yhm | text | 非空

nc | text | 非空

mm | character(32) | 非空

索引：

"t1\_pkey" PRIMARY KEY, btree (id)

"t1\_yhm\_key" UNIQUE CONSTRAINT, btree (yhm)

postgres=# CREATE OR REPLACE FUNCTION f19() RETURNS SETOF public.t\_rec AS

postgres-# $$

postgres$# DECLARE

postgres$# --使用已经定义的结构类型返回

postgres$# v\_rec public.t\_rec;

postgres$# BEGIN

postgres$# FOR v\_rec IN SELECT id,yhm FROM t1 ORDER BY id LOOP

postgres$# RETURN NEXT v\_rec;

postgres$# END LOOP;

postgres$# RETURN ;--最后的RETURN可以加，也可以不加上去

postgres$# END;

postgres$# $$

postgres-# LANGUAGE PLPGSQL;

CREATE FUNCTION

postgres=#

postgres=# SELECT \* FROM f19();

id | mc

----+--------

1 | TBase

2 | pgxc

3 | pgxz

(3 行记录)

postgres=# CREATE OR REPLACE FUNCTION f20(a\_int integer) RETURNS SETOF record AS

postgres-# $$

postgres$# DECLARE

postgres$# --a\_int定义返回的字段数，实现动态列返回

postgres$# v\_rec record;

postgres$# v\_sql text;

postgres$# BEGIN

postgres$# IF a\_int = 2 THEN

postgres$# v\_sql:='SELECT id,yhm FROM t1 ORDER BY id ';

postgres$# ELSE

postgres$# v\_sql:='SELECT id,yhm,nc FROM t1 ORDER BY id';

postgres$# END IF;

postgres$# FOR v\_rec IN EXECUTE v\_sql LOOP

postgres$# RETURN NEXT v\_rec;

postgres$# END LOOP;

postgres$# RETURN ;--最后的RETURN可以加，也可以不加上去

postgres$# END;

postgres$# $$

postgres-# LANGUAGE PLPGSQL;

CREATE FUNCTION

postgres=# SELECT \* FROM f20(2) t(id integer,yhm text);

id | yhm

----+--------

1 | TBase

2 | pgxc

3 | pgxz

(3 行记录)

postgres=# SELECT \* FROM f20(3) t(id integer,yhm text,nc text);

id | yhm | nc

----+--------+--------

1 | TBase | TBase

2 | pgxc | pgxc

3 | pgxz | pgxz

(3 行记录)

postgres=# CREATE OR REPLACE FUNCTION f21(OUT a\_id integer,OUT a\_yhm TEXT) RETURNS SETOF record AS

postgres-# $$

postgres$# DECLARE

postgres$# --使用out返回

postgres$# v\_rec record;

postgres$# BEGIN

postgres$# FOR v\_rec IN SELECT id,yhm FROM t1 LOOP

postgres$# a\_id:=v\_rec.id;

postgres$# a\_yhm:=v\_rec.yhm;

postgres$# RETURN NEXT;

postgres$# END LOOP;

postgres$# RETURN ;--最后的RETURN可以加，也可以不加上去

postgres$# END;

postgres$# $$

postgres-# LANGUAGE PLPGSQL;

CREATE FUNCTION

postgres=#

postgres=# SELECT \* FROM f21();

a\_id | a\_yhm

------+--------

1 | TBase

2 | pgxc

3 | pgxz

(3 行记录)

postgres=# CREATE OR REPLACE FUNCTION f22() RETURNS SETOF refcursor AS

postgres-# $$

postgres$# DECLARE

postgres$# --返回游标集

postgres$# v\_ref1 REFCURSOR;

postgres$# v\_ref2 REFCURSOR;

postgres$# BEGIN

postgres$# OPEN v\_ref1 FOR SELECT \* FROM t;

postgres$# OPEN v\_ref2 FOR SELECT \* FROM t1;

postgres$# RETURN NEXT v\_ref1;

postgres$# RETURN NEXT v\_ref2;

postgres$# RETURN ;--最后的RETURN可以加，也可以不加上去

postgres$# END;

postgres$# $$

postgres-# LANGUAGE PLPGSQL;

CREATE FUNCTION

postgres=#

postgres=# BEGIN;

BEGIN

postgres=# SELECT \* FROM f22();

f22

---------------------

<unnamed portal 13>

<unnamed portal 14>

(2 行记录)

postgres=# FETCH ALL FROM "<unnamed portal 13>";

id | mc

----+--------

1 | TBase

2 | pgxz

(2 行记录)

postgres=# FETCH ALL FROM "<unnamed portal 14>";

id | yhm | nc | mm

----+--------+--------+----------------------------------

1 | TBase | TBase | 202cb962ac59075b964b07152d234b70

2 | pgxc | pgxc | 202cb962ac59075b964b07152d234b70

3 | pgxz | pgxz | 202cb962ac59075b964b07152d234b70

(3 行记录)

postgres=# COMMIT;

COMMIT

postgres=# CREATE OR REPLACE FUNCTION f22(a\_ref1 refcursor,a\_ref2 refcursor) RETURNS SETOF refcursor AS

postgres-# $$

postgres$# BEGIN

postgres$# --指定游标名称

postgres$# OPEN a\_ref1 FOR SELECT \* FROM t;

postgres$# OPEN a\_ref2 FOR SELECT \* FROM t1;

postgres$# RETURN NEXT a\_ref1;

postgres$# RETURN NEXT a\_ref2;

postgres$# RETURN ;--最后的RETURN可以加，也可以不加上去

postgres$# END;

postgres$# $$

postgres-# LANGUAGE PLPGSQL;

CREATE FUNCTION

postgres=#

postgres=# BEGIN;

BEGIN

postgres=#

postgres=# SELECT \* FROM f22('a','b');

f22

-----

a

b

(2 行记录)

postgres=# FETCH ALL FROM "a";

id | mc

----+--------

1 | TBase

2 | pgxz

(2 行记录)

postgres=# FETCH ALL FROM "b";

id | yhm | nc | mm

----+--------+--------+----------------------------------

1 | TBase | TBase | 202cb962ac59075b964b07152d234b70

2 | pgxc | pgxc | 202cb962ac59075b964b07152d234b70

3 | pgxz | pgxz | 202cb962ac59075b964b07152d234b70

(3 行记录)

postgres=# COMMIT;

COMMIT

#### 10.3.2.9、返回多态类型

postgres=# CREATE OR REPLACE FUNCTION f23(a\_arg anyelement) RETURNS anyelement AS

postgres-# $$

postgres$# BEGIN

postgres$# RETURN a\_arg;

postgres$# END;

postgres$# $$

postgres-# LANGUAGE PLPGSQL;

CREATE FUNCTION

postgres=#

postgres=# SELECT \* FROM f23('TBase'::text);

f23

------

TBase

(1 行记录)

postgres=# SELECT \* FROM f23(1::integer);

f23

-----

1

(1 行记录)

postgres=# SELECT \* FROM f23(ARRAY['TBase','pgxz']);

f23

---------------

{TBase,pgxz}

(1 行记录)

postgres=# SELECT \* FROM f23(ROW(1,'TBase')::public.t\_rec);

id | mc

----+------

1 | TBase

(1 行记录)

postgres=# CREATE OR REPLACE FUNCTION f24(a\_arg ANYARRAY) RETURNS anyarray AS

postgres-# $$

postgres$# BEGIN

postgres$# RETURN a\_arg;

postgres$# END;

postgres$# $$

postgres-# LANGUAGE PLPGSQL;

CREATE FUNCTION

postgres=#

postgres=# SELECT \* FROM f24(ARRAY[1,2]::INTEGER[]);

f24

-------

{1,2}

(1 行记录)

postgres=# SELECT f24(ARRAY[t1.\*]) FROM t1;

f24

--------------------------------------------------------

{"(1,TBase,TBase,202cb962ac59075b964b07152d234b70)"}

{"(2,pgxc,pgxc,202cb962ac59075b964b07152d234b70)"}

{"(3,pgxz,pgxz,202cb962ac59075b964b07152d234b70)"}

(3 行记录)

返回数据类型如果是多态，则函数最少需要定义一个多态参数

## 10.4、变量使用

### 10.4.1、变量使用介绍

在一个块中使用的所有变量必须在该块的声明小节中事先进行声明,PL/pgSQL变量可以是任意 SQL 数据类型，可以是一个简单数据类型、复合类型、RECORD、已经存在的表行类型、表字段类型、游标.

### 10.4.2、变量使用实例

#### 10.4.2.1、变量声明语法

name [ CONSTANT ] type [ COLLATE collation\_name ] [ NOT NULL ] [ { DEFAULT | := | = } expression ];

如果给定DEFAULT子句，它会指定进入该块时分 配给该变量的初始值。如果没有给出DEFAULT子句， 则该变量被初始化为SQL空值。 CONSTANT选项阻止该变量在初始化之后被赋值， 这样它的值在块的持续期内保持不变。COLLATE 选项指定用于该变量的一个排序规则（见 第 41.3.6 节）。如果指 定了NOT NULL，对该变量赋值为空值会导致一个 运行时错误。所有被声明为NOT NULL的变量必须 被指定一个非空默认值。 等号（=）可以被用来代替 PL/SQL-兼容的 :=。

#### 10.4.2.2、定义一个普通变量

postgres=# CREATE OR REPLACE FUNCTION f25() RETURNS VOID AS

postgres-# $$

postgres$# DECLARE

postgres$# --所有变量的声明都要放在这里,建议变量以v\_开头,参数以a\_开头

postgres$# v\_int integer := 1;

postgres$# v\_text text;

postgres$# BEGIN

postgres$# v\_text = 'TBase';

postgres$# RAISE NOTICE 'v\_int = %',v\_int;

postgres$# RAISE NOTICE 'v\_text = %',v\_text;

postgres$# END;

postgres$# $$

postgres-# LANGUAGE plpgsql;

CREATE FUNCTION

postgres=# SELECT f25();

NOTICE: v\_int = 1

NOTICE: v\_text = TBase

f25

-----

(1 row)

postgres=#

#### 10.4.2.3、定义CONSTANT变量

postgres=# CREATE OR REPLACE FUNCTION f25() RETURNS VOID AS

postgres-# $$

postgres$# DECLARE

postgres$# v\_int CONSTANT integer := 1;

postgres$# BEGIN

postgres$# RAISE NOTICE 'v\_int = %',v\_int;

postgres$# END;

postgres$# $$

postgres-# LANGUAGE plpgsql;

CREATE FUNCTION

postgres=# select f25();

NOTICE: v\_int = 1

f25

-----

(1 row)

CONSTANT 不能再次赋值

postgres=# CREATE OR REPLACE FUNCTION f25() RETURNS VOID AS

postgres-# $$

postgres$# DECLARE

postgres$# v\_int CONSTANT integer := 1;

postgres$# BEGIN

postgres$# RAISE NOTICE 'v\_int = %',v\_int;

postgres$# v\_int = 10;

postgres$# RAISE NOTICE 'v\_int = %',v\_int;

postgres$# END;

postgres$# $$

postgres-# LANGUAGE plpgsql;

ERROR: "v\_int" is declared CONSTANT

#### 10.4.2.4、定义NOT NULL变量

postgres=# CREATE OR REPLACE FUNCTION f25() RETURNS VOID AS

postgres-# $$

postgres$# DECLARE

postgres$# v\_int integer NOT NULL := 1;

postgres$# BEGIN

postgres$# RAISE NOTICE 'v\_int = %',v\_int;

postgres$# SELECT NULL INTO v\_int;

postgres$# RAISE NOTICE 'v\_int = %',v\_int;

postgres$# END;

postgres$# $$

postgres-# LANGUAGE plpgsql;

CREATE FUNCTION

postgres=# SELECT f25();

NOTICE: v\_int = 1

ERROR: null value cannot be assigned to variable "v\_int" declared NOT NULL

CONTEXT: PL/pgSQL function f25() line 6 at SQL statement

postgres=#

定义为NOT NULL变量，则该变量受NOT NULL约束

#### 10.4.2.5、定义COLLATE变量

按unicode值对比大小

postgres=# CREATE OR REPLACE FUNCTION f25() RETURNS VOID AS

postgres-# $$

postgres$# DECLARE

postgres$# v\_txt1 TEXT COLLATE "C" := '严';

postgres$# v\_txt2 TEXT COLLATE "C" := '丰';

postgres$# BEGIN

postgres$# IF v\_txt1 > v\_txt2 THEN

postgres$# RAISE NOTICE ' % -> % ',v\_txt1,v\_txt2;

postgres$# ELSE

postgres$# RAISE NOTICE ' % -> % ',v\_txt2,v\_txt1;

postgres$# END IF;

postgres$# END;

postgres$# $$

postgres-# LANGUAGE plpgsql;

CREATE FUNCTION

postgres=# SELECT f25();

NOTICE: 丰 -> 严

f25

-----

(1 row)

postgres=# select '严'::bytea;

bytea

----------

\xe4b8a5

(1 row)

postgres=# select '丰'::bytea;

bytea

----------

\xe4b8b0

(1 row)

按汉字的拼音对比大小

postgres=# CREATE OR REPLACE FUNCTION f25() RETURNS VOID AS

postgres-# $$

postgres$# DECLARE

postgres$# v\_txt1 TEXT COLLATE "zh\_CN.utf8" := '严';

postgres$# v\_txt2 TEXT COLLATE "zh\_CN.utf8" := '丰';

postgres$# BEGIN

postgres$# IF v\_txt1 > v\_txt2 THEN

postgres$# RAISE NOTICE ' % -> % ',v\_txt1,v\_txt2;

postgres$# ELSE

postgres$# RAISE NOTICE ' % -> % ',v\_txt2,v\_txt1;

postgres$# END IF;

postgres$# END;

postgres$# $$

postgres-# LANGUAGE plpgsql;

CREATE FUNCTION

postgres=# SELECT f25();

NOTICE: 严 -> 丰

f25

-----

(1 row)

#### 10.4.2.6、变量赋值

postgres=# CREATE OR REPLACE FUNCTION f25() RETURNS VOID AS

postgres-# $$

postgres$# DECLARE

postgres$# --定义时赋值

postgres$# v\_int1 integer = 1;

postgres$# --使用 :=兼容于plsql

postgres$# v\_int2 integer := 1;

postgres$# v\_txt1 text;

postgres$# v\_float float8;

postgres$# --使用查询赋值

postgres$# v\_relname text = (select relname FROM pg\_class LIMIT 1);

postgres$# v\_relpages integer;

postgres$# v\_rec RECORD;

postgres$# BEGIN

postgres$# --在函数体中赋值

postgres$# v\_txt1 = 'TBase';

postgres$# v\_float = random();

postgres$# --使用查询赋值的另一种方式

postgres$# SELECT relname,relpages INTO v\_relname,v\_relpages FROM pg\_class ORDER BY random() LIMIT 1;

postgres$# RAISE NOTICE 'v\_relname = % , relpages = %',v\_relname,v\_relpages;

postgres$# END;

postgres$# $$

postgres-# LANGUAGE plpgsql;

CREATE FUNCTION

postgres=# SELECT \* FROM f25();

NOTICE: v\_relname = pg\_ts\_parser , relpages = 1

f25

-----

(1 row)

## 10.5、控制结构

### 10.5.1、判断语句

#### 10.5.1.1、IF...THEN...END IF

postgres=# CREATE OR REPLACE FUNCTION f26() RETURNS VOID AS

postgres-# $$

postgres$# BEGIN

postgres$# IF random()>0.5 THEN

postgres$# RAISE NOTICE '随机数大于0.5';

postgres$# END IF;

postgres$# END;

postgres$# $$

postgres-# LANGUAGE plpgsql;

CREATE FUNCTION

postgres=# select f26();

NOTICE: 随机数大于0.5

f26

-----

(1 row)

postgres=#

#### 10.5.1.2、IF...THEN...ELSE...END IF

postgres=# CREATE OR REPLACE FUNCTION f26() RETURNS VOID AS

postgres-# $$

postgres$# BEGIN

postgres$# IF random()>0.99 THEN

postgres$# RAISE NOTICE '随机数大于0.99';

postgres$# ELSE

postgres$# RAISE NOTICE '随机数小于或等于0.99';

postgres$# END IF;

postgres$# END;

postgres$# $$

postgres-# LANGUAGE plpgsql;

CREATE FUNCTION

postgres=# select f26();

NOTICE: 随机数小于或等于0.99

f26

-----

(1 row)

postgres=#

#### 10.5.1.3、IF...THEN...ELSIF...THEN...ELSE...END IF

postgres=# CREATE OR REPLACE FUNCTION f26() RETURNS VOID AS

postgres-# $$

postgres$# DECLARE

postgres$# v\_float8 float8 := random();

postgres$# BEGIN

postgres$# IF v\_float8>0.99 THEN

postgres$# RAISE NOTICE '随机数大于0.99';

postgres$# ELSIF v\_float8>0.5 THEN

postgres$# RAISE NOTICE '随机数大于0.50';

postgres$# ELSIF v\_float8>0.25 THEN

postgres$# RAISE NOTICE '随机数大于0.25';

postgres$# ELSE

postgres$# RAISE NOTICE '随机数小于或等于0.25';

postgres$# END IF;

postgres$# END;

postgres$# $$

postgres-# LANGUAGE plpgsql;

CREATE FUNCTION

postgres=# SELECT f26();

NOTICE: 随机数大于0.50

f26

-----

1. row)

#### 10.5.1.4、CASE语句

postgres=# CREATE OR REPLACE FUNCTION f26() RETURNS VOID AS

postgres-# $$

postgres$# DECLARE

postgres$# v\_float8 float8 := random();

postgres$# BEGIN

postgres$# CASE

postgres$# WHEN v\_float8>0.99 THEN

postgres$# RAISE NOTICE '随机数大于0.99';

postgres$# WHEN v\_float8>0.5 THEN

postgres$# RAISE NOTICE '随机数大于0.50';

postgres$# WHEN v\_float8>0.25 THEN

postgres$# RAISE NOTICE '随机数大于0.25';

postgres$# ELSE

postgres$# RAISE NOTICE '随机数小于或等于0.25';

postgres$# END CASE;

postgres$# END;

postgres$# $$

postgres-# LANGUAGE plpgsql;

CREATE FUNCTION

postgres=# SELECT f26();

NOTICE: 随机数大于0.50

f26

-----

(1 row)

### 10.5.2、循环语句

#### 10.5.2.1、LOOP循环

postgres=# CREATE OR REPLACE FUNCTION f27() RETURNS VOID AS

postgres-# $$

postgres$# DECLARE

postgres$# v\_id INTEGER := 1;

postgres$# BEGIN

postgres$# LOOP

postgres$# RAISE NOTICE '%',v\_id;

postgres$# EXIT WHEN random()>0.8;

postgres$# v\_id := v\_id + 1;

postgres$# END LOOP ;

postgres$# END;

postgres$# $$

postgres-# LANGUAGE plpgsql;

CREATE FUNCTION

postgres=# SELECT f27();

NOTICE: 1

NOTICE: 2

f27

-----

(1 row)

使用EXIT退出循环

postgres=# CREATE OR REPLACE FUNCTION f27() RETURNS VOID AS

postgres-# $$

postgres$# DECLARE

postgres$# v\_id INTEGER := 1;

postgres$# v\_random float8 ;

postgres$# BEGIN

postgres$# LOOP

postgres$# RAISE NOTICE '%',v\_id;

postgres$# v\_id := v\_id + 1;

postgres$# v\_random := random();

postgres$# IF v\_random > 0.8 THEN

postgres$# RETURN;

postgres$# END IF;

postgres$# END LOOP ;

postgres$# END;

postgres$# $$

postgres-# LANGUAGE plpgsql;

CREATE FUNCTION

postgres=# SELECT f27();

NOTICE: 1

NOTICE: 2

NOTICE: 3

NOTICE: 4

NOTICE: 5

f27

-----

(1 row)

postgres=#

使用RETURN退出循环返回

#### 10.5.2.2、WHILE循环

postgres=# CREATE OR REPLACE FUNCTION f27() RETURNS VOID AS

postgres-# $$

postgres$# DECLARE

postgres$# v\_id INTEGER := 1;

postgres$# v\_random float8 := random() ;

postgres$# BEGIN

postgres$# WHILE v\_random > 0.8 LOOP

postgres$# RAISE NOTICE '%',v\_id;

postgres$# v\_id := v\_id + 1;

postgres$# v\_random = random();

postgres$# END LOOP;

postgres$# END;

postgres$# $$

postgres-# LANGUAGE plpgsql;

CREATE FUNCTION

postgres=# SELECT f27();

NOTICE: 1

f27

-----

(1 row)

#### 10.5.2.3、FOR循环

postgres=# CREATE OR REPLACE FUNCTION f27() RETURNS VOID AS

postgres-# $$

postgres$# BEGIN

postgres$# FOR i IN 1..3 LOOP

postgres$# RAISE NOTICE 'i = %',i;

postgres$# END LOOP;

postgres$# END;

postgres$# $$

postgres-# LANGUAGE plpgsql;

CREATE FUNCTION

postgres=# SELECT f27();

NOTICE: i = 1

NOTICE: i = 2

NOTICE: i = 3

f27

-----

(1 row)

postgres=# CREATE OR REPLACE FUNCTION f27() RETURNS VOID AS

postgres-# $$

postgres$# BEGIN

postgres$# FOR i IN REVERSE 3..1 LOOP

postgres$# RAISE NOTICE 'i = %',i;

postgres$# END LOOP;

postgres$# END;

postgres$# $$

postgres-# LANGUAGE plpgsql;

CREATE FUNCTION

postgres=# SELECT f27();

NOTICE: i = 3

NOTICE: i = 2

NOTICE: i = 1

f27

-----

(1 row)

使用REVERSE递减

postgres=# CREATE OR REPLACE FUNCTION f27() RETURNS VOID AS

postgres-# $$

postgres$# BEGIN

postgres$# FOR i IN 1..8 BY 2 LOOP

postgres$# RAISE NOTICE 'i = %',i;

postgres$# END LOOP;

postgres$# END;

postgres$# $$

postgres-# LANGUAGE plpgsql;

CREATE FUNCTION

postgres=# SELECT f27();

NOTICE: i = 1

NOTICE: i = 3

NOTICE: i = 5

NOTICE: i = 7

f27

-----

(1 row)

使用BY设置步长

#### 10.5.2.4、FOR循环查询结果

postgres=# CREATE OR REPLACE FUNCTION f27() RETURNS VOID AS

postgres-# $$

postgres$# DECLARE

postgres$# v\_rec RECORD;

postgres$# BEGIN

postgres$# FOR v\_rec IN SELECT \* FROM public.t LOOP

postgres$# RAISE NOTICE '%',v\_rec;

postgres$# END LOOP;

postgres$# END;

postgres$# $$

postgres-# LANGUAGE plpgsql;

CREATE FUNCTION

postgres=# SELECT f27();

NOTICE: (1,TBase)

NOTICE: (2,pgxz)

f27

-----

(1 row)

#### 10.5.2.5、FOREACH循环一个数组

postgres=# CREATE OR REPLACE FUNCTION f27() RETURNS VOID AS

postgres-# $$

postgres$# DECLARE

postgres$# v\_random\_arr float8[]:=ARRAY[random(),random()];

postgres$# v\_random float8;

postgres$# BEGIN

postgres$# FOREACH v\_random IN ARRAY v\_random\_arr LOOP

postgres$# RAISE NOTICE '%',v\_random ;

postgres$# END LOOP;

postgres$# END;

postgres$# $$

postgres-# LANGUAGE plpgsql;

CREATE FUNCTION

postgres=# SELECT f27();

NOTICE: 0.452758576720953

NOTICE: 0.975814974401146

f27

-----

1. row)

postgres=# CREATE OR REPLACE FUNCTION f27() RETURNS VOID AS

postgres-# $$

postgres$# DECLARE

postgres$# v\_random\_arr float8[][]:=ARRAY[ARRAY[random(),random()],ARRAY[random(),random()]];

postgres$# v\_random float8;

postgres$# BEGIN

postgres$# FOREACH v\_random SLICE 0 IN ARRAY v\_random\_arr LOOP

postgres$# RAISE NOTICE '%',v\_random ;

postgres$# END LOOP;

postgres$# END;

postgres$# $$

postgres-# LANGUAGE plpgsql;

CREATE FUNCTION

postgres=# SELECT f27();

NOTICE: 0.0588191924616694

NOTICE: 0.368828620761633

NOTICE: 0.813376842066646

NOTICE: 0.415377039927989

f27

-----

1. row)

循环会通过计算expression得到的数组的个体元素进行迭代

postgres=# CREATE OR REPLACE FUNCTION f27() RETURNS VOID AS

postgres-# $$

postgres$# DECLARE

postgres$# v\_random\_arr float8[][]:=ARRAY[ARRAY[random(),random()],ARRAY[random(),random()]];

postgres$# v\_random float8[];

postgres$# BEGIN

postgres$# FOREACH v\_random SLICE 1 IN ARRAY v\_random\_arr LOOP

postgres$# RAISE NOTICE '%',v\_random ;

postgres$# END LOOP;

postgres$# END;

postgres$# $$

postgres-# LANGUAGE plpgsql;

CREATE FUNCTION

postgres=# SELECT f27();

NOTICE: {0.578366641886532,0.78098024148494}

NOTICE: {0.783956411294639,0.450278480071574}

f27

-----

(1 row)

通过一个正SLICE值，FOREACH通过数组的切片而不是单一元素迭代

### 10.5.3、其它控制语句

#### 10.5.3.1、动态执行

postgres=# CREATE OR REPLACE FUNCTION f27(a\_id INTEGER) RETURNS text AS

postgres-# $$

postgres$# DECLARE

postgres$# v\_sql TEXT;

postgres$# v\_mc TEXT;

postgres$# BEGIN

postgres$# v\_sql := 'SELECT mc FROM t WHERE id='||a\_id::TEXT;

postgres$# EXECUTE v\_sql INTO v\_mc;

postgres$# RETURN v\_mc;

postgres$# END;

postgres$# $$

postgres-# LANGUAGE plpgsql;

CREATE FUNCTION

postgres=# SELECT f27(1);

f27

-------

TBase

(1 row)

动态执行就是拼sql语句,然后使用EXECUTE命令执行

#### 10.5.3.2、执行一个没有结果的命令

postgres=# CREATE OR REPLACE FUNCTION f27() RETURNS void AS

postgres-# $$

postgres$# BEGIN

postgres$# perform f27(1);

postgres$# END;

postgres$# $$

postgres-# LANGUAGE plpgsql;

CREATE FUNCTION

postgres=# SELECT f27();

f27

-----

(1 row)

postgres=#

#### 10.5.3.3、获取执行结果

postgres=# DROP FUNCTION f27(INTEGER);

DROP FUNCTION

postgres=# CREATE OR REPLACE FUNCTION f27(a\_id INTEGER) RETURNS VOID AS

postgres-# $$

postgres$# DECLARE

postgres$# v\_mc TEXT;

postgres$# BEGIN

postgres$# SELECT mc INTO v\_mc FROM t WHERE id=a\_id;

postgres$# IF FOUND THEN

postgres$# RAISE NOTICE '查询到记录，值为%',v\_mc;

postgres$# ELSE

postgres$# RAISE NOTICE '查不到记录' ;

postgres$# END IF;

postgres$# END;

postgres$# $$

postgres-# LANGUAGE plpgsql;

CREATE FUNCTION

postgres=# SELECT f27(1);

NOTICE: 查询到记录，值为TBase

f27

-----

(1 row)

postgres=# SELECT f27(3);

NOTICE: 查不到记录

f27

-----

(1 row)

#### 10.5.3.4、获取影响行数

postgres=# CREATE OR REPLACE FUNCTION f27(a\_id INTEGER) RETURNS VOID AS

postgres-# $$

postgres$# DECLARE

postgres$# v\_mc TEXT;

postgres$# v\_row\_count BIGINT;

postgres$# BEGIN

postgres$# SELECT mc INTO v\_mc FROM t WHERE id=a\_id;

postgres$# GET DIAGNOSTICS v\_row\_count = ROW\_COUNT;

postgres$# RAISE NOTICE '查询到的记录数为 % ',v\_row\_count;

postgres$# END;

postgres$# $$

postgres-# LANGUAGE plpgsql;

CREATE FUNCTION

postgres=# SELECT f27(1);

NOTICE: 查询到的记录数为 1

f27

-----

(1 row)

postgres=# SELECT f27(3);

NOTICE: 查询到的记录数为 0

f27

-----

(1 row)

### 10.5.4、俘获错误

#### 10.5.4.1、错误俘获处理

postgres=# CREATE TABLE t\_exception (id integer not null,nc text);

CREATE TABLE

postgres=# create unique index t\_exception\_id\_uidx on t\_exception using btree(id);

CREATE INDEX

postgres=# CREATE OR REPLACE FUNCTION f27(a\_id integer,a\_nc text) RETURNS TEXT AS

postgres-# $$

postgres$# BEGIN

postgres$# INSERT INTO t\_exception VALUES(a\_id,a\_nc);

postgres$# RETURN '';

postgres$# EXCEPTION WHEN OTHERS THEN

postgres$# RETURN '执行出错';

postgres$# END;

postgres$# $$

postgres-# LANGUAGE plpgsql;

CREATE FUNCTION

postgres=# SELECT f27(1,'TBase');

f27

-----

(1 row)

postgres=# SELECT f27(1,'TBase');

f27

----------

执行出错

(1 row)

#### 10.5.4.2、获取错误相关信息

postgres=# CREATE OR REPLACE FUNCTION f27(a\_id integer,a\_nc text) RETURNS TEXT AS

postgres-# $$

postgres$# DECLARE

postgres$# v\_sqlstate text;

postgres$# v\_context text;

postgres$# v\_message\_text text;

postgres$# BEGIN

postgres$# INSERT INTO t\_exception VALUES(a\_id,a\_nc);

postgres$# RETURN '';

postgres$# EXCEPTION WHEN OTHERS THEN

postgres$# GET STACKED DIAGNOSTICS v\_sqlstate = RETURNED\_SQLSTATE,

postgres$# v\_message\_text = MESSAGE\_TEXT,

postgres$# v\_context = PG\_EXCEPTION\_CONTEXT;

postgres$# RAISE NOTICE '错误代码 : %',v\_sqlstate;

postgres$# RAISE NOTICE '出错信息 : %',v\_message\_text;

postgres$# RAISE NOTICE '发生异常语句 : %',v\_context;

postgres$# RETURN '错误代码 : '||v\_sqlstate || '\n出错信息 : '||v\_message\_text|| '发生异常语句 : '||v\_context;

postgres$# END;

postgres$# $$

postgres-# LANGUAGE plpgsql;

CREATE FUNCTION

postgres=# SELECT f27(1,'TBase');

NOTICE: 错误代码 : 23505

NOTICE: 出错信息 : node:16385, error duplicate key value violates unique constraint "t\_exception\_id\_uidx"

NOTICE: 发生异常语句 : SQL statement "INSERT INTO t\_exception VALUES(a\_id,a\_nc)"

PL/pgSQL function f27(integer,text) line 7 at SQL statement

f27

---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

错误代码 : 23505\n出错信息 : node:16385, error duplicate key value violates unique constraint "t\_exception\_id\_uidx"发生异常语句 : SQL statement "INSERT INTO t\_exception VALUES(a\_id,a\_nc)"+

PL/pgSQL function f27(integer,text) line 7 at SQL statement

(1 row)

## 10.6、触发器函数

### 10.6.1、INSERT事件触发器函数

函数功能实现字段值t\_trigger.nc值重写

postgres=# CREATE TABLE t\_trigger

postgres-# (

postgres(# id integer NOT NULL,

postgres(# nc text NOT NULL

postgres(# );

CREATE TABLE

postgres=# CREATE OR REPLACE FUNCTION t\_trigger\_insert\_trigger\_func() RETURNS trigger AS

postgres-# $$

postgres$# BEGIN

postgres$# IF NEW.nc = '' THEN

postgres$# NEW.nc = 'TBase\_' || random()::text;

postgres$# END IF;

postgres$# RETURN NEW;

postgres$# END;

postgres$# $$

postgres-# LANGUAGE plpgsql;

CREATE FUNCTION

postgres=# CREATE TRIGGER t\_trigger\_insert\_trigger BEFORE INSERT ON t\_trigger FOR EACH ROW EXECUTE PROCEDURE t\_trigger\_insert\_trigger\_func();

CREATE TRIGGER

postgres=# INSERT INTO t\_trigger values(1,'');

INSERT 0 1

postgres=# SELECT \* FROM t\_trigger ;

id | nc

----+-------------------------

1 | TBase\_0.426093454472721

(1 row)

注意使用BEFORE,不能使用AFTER,否则重写失效

### 10.6.2、UPDATE事件触发器函数

不准许更新t\_trigger.nc字段值为 TBase

postgres=# CREATE OR REPLACE FUNCTION t\_trigger\_update\_trigger\_func() RETURNS trigger AS

postgres-# $$

postgres$# BEGIN

postgres$# --不准许t\_trigger.nc值为 TBase

postgres$# IF NEW.nc = 'TBase' THEN

postgres$# NEW.nc = OLD.nc ;

postgres$# END IF;

postgres$# RETURN NEW;

postgres$# END;

postgres$# $$

postgres-# LANGUAGE plpgsql;

CREATE FUNCTION

postgres=# CREATE TRIGGER t\_trigger\_update\_trigger BEFORE UPDATE ON t\_trigger FOR EACH ROW EXECUTE PROCEDURE t\_trigger\_update\_trigger\_func();

CREATE TRIGGER

postgres=# UPDATE t\_trigger SET nc='TBase' WHERE id=1;

UPDATE 1

postgres=# SELECT \* FROM t\_trigger ;

id | nc

----+-------------------------

1 | TBase\_0.426093454472721

(1 row)

postgres=#

### 10.6.3、DELETE事件触发器函数

限制TBase记录不能被删除

postgres=# CREATE OR REPLACE FUNCTION t\_trigger\_delete\_trigger\_func() RETURNS trigger AS

postgres-# $$

postgres$# BEGIN

postgres$# --不准许t\_trigger.nc值为 TBase

postgres$# IF OLD.nc = 'TBase' THEN

postgres$# RETURN NULL;

postgres$# --RAISE EXCEPTION 'TBase不能被删除';

postgres$# END IF;

postgres$# RETURN OLD;

postgres$# END;

postgres$# $$

postgres-# LANGUAGE plpgsql;

CREATE FUNCTION

postgres=# CREATE TRIGGER t\_trigger\_delete\_trigger BEFORE DELETE ON t\_trigger FOR EACH ROW EXECUTE PROCEDURE t\_trigger\_delete\_trigger\_func();

CREATE TRIGGER

postgres=# INSERT INTO t\_trigger VALUES(2,'TBase');

INSERT 0 1

postgres=# SELECT \* t\_trigg

postgres=# SELECT \* FROM t\_trigger ;

id | nc

----+-------------------------

1 | TBase\_0.426093454472721

2 | TBase

(2 rows)

postgres=# DELETE FROM t\_trigger WHERE id=2;

DELETE 0

postgres=# SELECT \* FROM t\_trigger ;

id | nc

----+-------------------------

1 | TBase\_0.426093454472721

2 | TBase

(2 rows)

### 10.6.4、删除触发器

postgres=# drop TRIGGER t\_trigger\_insert\_trigger on t\_trigger;

DROP TRIGGER

### 10.6.5、更多的触发器函数使用

更多的触发器函数使用见文档

[http://www.postgres.cn/docs/10/plpgsql-trigger.html](http://www.postgres.cn/docs/9.4/plpgsql-trigger.html)

## 10.7、消息及异常输出

### 10.7.1、RAISE NOTICE

postgres=# CREATE OR REPLACE FUNCTION f28() RETURNS VOID AS

postgres-# $$

postgres$# DECLARE

postgres$# v\_int INTEGER := 1;

postgres$# BEGIN

postgres$# RAISE NOTICE 'v\_int = %, 随机数 = %',v\_int,random();

postgres$# END;

postgres$# $$

postgres-# LANGUAGE plpgsql;

CREATE FUNCTION

postgres=# SELECT f28();

NOTICE: v\_int = 1, 随机数 = 0.236714988015592

f28

-----

(1 row)

使用raise notice 向终端输出一个消息,也有可能写到日志中(需要调整日志的保存级别)

### 10.7.2、RAISE EXCEPTION

postgres=# CREATE OR REPLACE FUNCTION f28() RETURNS VOID AS

postgres-# $$

postgres$# DECLARE

postgres$# v\_int INTEGER := 1;

postgres$# BEGIN

postgres$# RAISE EXCEPTION '程序EXCEPTION ';

postgres$# --下面的语句不会再执行

postgres$# RAISE NOTICE 'v\_int = %, 随机数 = %',v\_int,random();

postgres$# END;

postgres$# $$

postgres-# LANGUAGE plpgsql;

CREATE FUNCTION

postgres=# SELECT f28();

ERROR: 程序EXCEPTION

如果在事务中执行这个函数,则事务会中止(abort)

### 10.7.3、RAISE EXCEPTION 自定义ERRCODE

postgres=# CREATE OR REPLACE FUNCTION f28() RETURNS VOID AS

postgres-# $$

postgres$# DECLARE

postgres$# v\_int INTEGER := 1;

postgres$# BEGIN

postgres$# RAISE EXCEPTION ' 程序EXCEPTION ' USING ERRCODE = '23505';

postgres$# END;

postgres$# $$

postgres-# LANGUAGE plpgsql;

CREATE FUNCTION

postgres=# SELECT f28();

ERROR: 程序EXCEPTION

日志中会记录这个ERRCODE

2017-10-03 18:40:16.710 CST,"pgxz","postgres",15072,"[local]",59d33b65.3ae0,225,"idle",2017-10-03 15:25:25 CST,4/367159,0,LOG,00000,"statement: SELECT f28();",,,,,,,,,"psql"

2017-10-03 18:40:16.710 CST,"pgxz","postgres",15072,"[local]",59d33b65.3ae0,226,"SELECT",2017-10-03 15:25:25 CST,4/367159,0,ERROR,23505," 程序EXCEPTION ",,,,,,,,,"psql"

## 10.8、pg/pgsql函数实战

### 10.8.1、批量设置表owner的函数

CREATE OR REPLACE FUNCTION public.alter\_owner(a\_schema\_name varchar,a\_role\_name varchar) RETURNS TEXT AS

$$

DECLARE

-- a\_schema\_name:指定某个模式下，不对定是对数据库的所有表

-- a\_role\_name：表所有者

v\_rec RECORD;

v\_sql TEXT;

BEGIN

IF a\_schema\_name != '' THEN--如果指模式，检查模式是否存

PERFORM 1 FROM pg\_namespace WHERE nspname = a\_schema\_name;

IF NOT FOUND THEN

RETURN '指定的模式 ' || a\_schema\_name || ' 不存在!';

END IF;

END IF;

PERFORM 1 FROM pg\_roles WHERE rolname = a\_role\_name ;

IF NOT FOUND THEN--检查用户是否存在

RETURN '指定的用户 ' || a\_role\_name || ' 不存在!';

END IF;

IF a\_schema\_name != '' THEN --指定了模式

v\_sql:='SELECT schemaname,tablename FROM pg\_tables WHERE schemaname=''' || a\_sche || '''';

ELSE

v\_sql:='SELECT schemaname,tablename FROM pg\_tables WHERE schemaname!=''pg\_catalog'' AND schemaname!=''information\_schema'' ';

END IF;

FOR v\_rec IN EXECUTE v\_sql LOOP

EXECUTE 'ALTER TABLE "'||v\_rec.schemaname||'"."'||v\_rec.tablename||'" OWNER TO '||a\_role\_name;

END LOOP;

RETURN 'ok';

END;

$$

LANGUAGE PLPGSQL;

COMMENT ON FUNCTION public.alter\_owner(a\_schema\_name varchar,a\_role\_name varchar) IS '批量设置表所有者';

CREATE OR REPLACE FUNCTION public.alter\_owner(a\_role\_name varchar) RETURNS TEXT AS

$$

BEGIN

RETURN public.alter\_owner('',a\_role\_name);

END;

$$

LANGUAGE PLPGSQL;

COMMENT ON FUNCTION public.alter\_owner(a\_role\_name varchar) IS '批量设置表所有者重载';

### 10.8.2、批量设置表的加密规则函数

create or replace function MLS\_TRANSPARENT\_CRYPT\_ALGORITHM\_BIND\_ALL\_TABLE(a\_schema text,a\_algo\_id int) returns text as

$$

declare

v\_rec record;

v\_algorithm\_name text;

v\_raise\_notice text;

v\_sqlstate text;

v\_context text;

v\_message\_text text;

begin

perform 1 from pg\_namespace where nspname=a\_schema ;

if not found then

return '模式['||a\_schema||']不存在';

end if;

--显示使用的加密算法

select algorithm\_name INTO v\_algorithm\_name from pg\_transparent\_crypt\_policy\_algorithm where algorithm\_id=a\_algo\_id;

if not found then

return '加密算法id['||a\_algo\_id::text||']不存在';

else

raise notice '你使用的密码算法为--%',v\_algorithm\_name;

end if;

for v\_rec in select pg\_tables.schemaname,pg\_tables.tablename,pg\_transparent\_crypt\_policy\_map.tblname from pg\_tables left outer join pg\_transparent\_crypt\_policy\_map on pg\_tables.schemaname=pg\_transparent\_crypt\_policy\_map.nspname and pg\_tables.tablename=pg\_transparent\_crypt\_policy\_map.tblname where pg\_tables.schemaname=a\_schema and pg\_transparent\_crypt\_policy\_map.tblname is null loop

begin

PERFORM MLS\_TRANSPARENT\_CRYPT\_ALGORITHM\_BIND\_TABLE(v\_rec.schemaname,v\_rec.tablename, a\_algo\_id);

EXCEPTION WHEN OTHERS THEN

GET STACKED DIAGNOSTICS v\_sqlstate = RETURNED\_SQLSTATE,

v\_message\_text = MESSAGE\_TEXT,

v\_context = PG\_EXCEPTION\_CONTEXT;

RAISE NOTICE '出错信息 : %',v\_message\_text;

end;

end loop;

return '配置表加密完成';

end;

$$

language plpgsql;

### 10.8.3、oracle to\_date函数的实现

Oracle的to\_date函数可以精确处理到年-月-日 时:分:秒，TBase只能处理 年-月-日

TBase可以使用to\_timestamp函数代替，如果应用程序中已经大量的使用的oracle to\_date函数并且是精细到时:分:秒，那我们可以自定义一个to\_date函数存放到一个优先访问的schema中，函数的内容为to\_timestamp，这样就可以兼容原来的oracle应用了，如下所示

postgres=# create schema oracle ;

CREATE SCHEMA

postgres=# CREATE OR REPLACE FUNCTION oracle.to\_date(a\_date text,a\_style text ) RETURNS timestamp with time zone AS

postgres-# $$

postgres$# BEGIN

postgres$# RETURN to\_timestamp(a\_date,a\_style);

postgres$# END;

postgres$# $$

postgres-# LANGUAGE PLPGSQL;

CREATE FUNCTION

postgres=# set search\_path = oracle ,"$user", public,pg\_catalog;;

SET

postgres=# select to\_date('2028-01-01 13:14:20','yyyy-MM-dd HH24:MI');

to\_date

------------------------

2028-01-01 13:14:00+08

1. row)

# 11、Tbase问题定位及性能优化

## 11.1、访问日志管理

### 11.1.1、配置TBase日志

TBase运行参数配置文件Postgresql.conf中涉及日志配置参数如下所示

#日志相关配置

#用户访问日志格式支持多种方法来记录服务器消息，包括stderr、csvlog

#如果csvlog被包括在log\_destination中，日志项会以"逗号分隔值" （CSV）格式被输出，这样可以很方便地把日志载入到程序中

log\_destination = 'csvlog'

#启用用户访问日志收集器

logging\_collector = on

#日志存放目录，可以是相对路径（相对了data目录）或绝对路径

log\_directory = 'pg\_log'

#设置日志文件的权限

log\_file\_mode = 0600

#按我先前经常是只有log\_rotation\_age这个值有效果

log\_truncate\_on\_rotation = 'on'

#120分钟切换一次

log\_rotation\_age = 120

#64MB切换一次

log\_rotation\_size = 64MB

#会话的当前执行命令保留长度，用于在字段pg\_stat\_activity.query中显示

track\_activity\_query\_size = 4096

#配置sql语句执行超过多少毫秒数时，语句将被记录,值为-1时禁用，0时记录所有语句

log\_min\_duration\_statement = 1000

#是否记录checkpoint

log\_checkpoints = on

#是否记录连接

log\_connections = on

#是否记录断开连接

log\_disconnections = on

#是否记录autovacuum执行日志

log\_autovacuum\_min\_duration = 0

#是否记录所有语句的执行时间，值为on时将单独记录所有语句的执行时间，这里不记录语句

log\_duration = off

#stderr配置日志记录的内容

log\_line\_prefix = '%h'

#控制那些类型的语句被记录，有效值是 none (off)、ddl、mod和 all（所有语句）。

#如果log\_min\_duration\_statement 值为0时，则log\_statement什么值的效果都一样

#如果log\_min\_duration\_statement 值大于0，并且log\_statement为ddl则ddl语句全部表被记录，为两条log

#dml超时才被记录，为一条记录

log\_statement = 'none'

#设置在服务器日志中写入的时间戳的时区

log\_timezone = 'PRC'

log\_filename = 'postgresql-%A-%H.log'

#是否记录autovacuum执行日志

log\_autovacuum\_min\_duration = 0

#控制被发送给客户端的消息级别

#其值有debug5,debug4,debug3,debug2,debug1,log,notice,warning,error

client\_min\_messages = notice

#控制哪些消息级别被写入到服务器日

#其值有debug5,debug4,debug3,debug2,debug1,log,notice,warning,error

log\_min\_messages = warning

#控制哪些导致一个错误情况的 SQL 语句被记录在服务器日志中

#默认值是ERROR,它表示导致错误、日志消息、致命错误或恐慌错误的语句将被记录在日志中

log\_min\_error\_statement = error

### 11.1.2、TBase日志格式说明

执行正确的语句产生的日志

2017-10-11 16:23:55.178 CST,"pgxz","postgres",11499,"1211.0.0.1:2329",59ddd50c.2ceb,1,"idle",2017-10-11 16:23:40 CST,3/26053,0,LOG,00000,"statement: select \* from pg\_class limit 1;",,,,,,,,,"psql"

执行时间 | 2017-10-11 16:23:55.178

用户名 | pgxz

数据库 | postgres

进程id | 11499

客户端id | 127.0.0.1:2329

会话 ID | 59ddd50c.2ceb

每个会话的行号 | 1

命令标签 | idle

登录时间 | 2017-10-11 16:23:40

虚拟事务 ID| 3/26053

普通事务 ID、 | 0

级别 | LOG

SQLSTATE 代码 | 00000

执行信息 | statement: select \* from pg\_class limit 1;

详情 |

提示 |

导致错误的内部查询 |

错误位置所在的字符计数 |

错误上下文 |

导致错误的用户查询（如果有且被log\_min\_error\_statement启用） |

错误位置所在的字符计数 |

在 PostgreSQL 源代码中错误的位置（如果log\_error\_verbosity被设置为verbose） |

应用名 | psql

错误日志解释

2017-10-11 16:24:10.233 CST,"pgxz","postgres",11499,"127.0.0.1:2329",59ddd50c.2ceb,2,"idle",2017-10-11 16:23:40 CST,3/26054,0,LOG,00000,"statement: select \* from pgxc\_nodes limit 1;",,,,,,,,,"psql"

2017-10-11 16:24:10.233 CST,"pgxz","postgres",11499,"127.0.0.1:2329",59ddd50c.2ceb,3,"SELECT",2017-10-11 16:23:40 CST,3/26054,0,ERROR,42P01,"relation ""pgxc\_nodes"" does not exist",,,,,,,,,"psql"

向TBase执行错误的语句会产生两条日志，一条是执行语句，一条提示出错的原因

### 11.1.3、对日志进行分析

#### 11.1.3.1、创建日志表

CREATE table tbase\_log

(

log\_time timestamp without time zone,

user\_name text,

database\_name text,

process\_id integer,

connection\_from text,

session\_id text,

session\_line\_num bigint,

command\_tag text,

session\_start\_time timestamp without time zone,

virtual\_transaction\_id text,

transaction\_id bigint,

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

);

#### 11.1.3.2、导入日志数据

TBase日志文件默认存储在“数据目录/pg\_log”目录下面

postgres=# COPY tbase\_log FROM '/data/pgxz/data/pgxz/dn001/pg\_log/postgresql-Tuesday-16.csv' WITH csv;

COPY 10790

#### 11.1.3.3、统计日志数据

--按照session连接及操作时间排序

postgres=# select \* from tbase\_log order by process\_id,log\_time;

--查询错误日志

SELECT \* FROM TBase\_log WHERE error\_severity='ERROR' limit 1;

--统计session操作数统计

postgres=# select count(1),process\_id,user\_name,database\_name from tbase\_log group by process\_id,user\_name,database\_name order by count(1) desc limit 10;

count | process\_id | user\_name | database\_name

-------+------------+-----------+---------------

2770 | 48067 | pgxz | postgres

10 | 22143 | pgxz | postgres

10 | 28778 | pgxz | postgres

9 | 28367 | pgxz | postgres

9 | 44280 | pgxz | postgres

8 | 32442 | pgxz | postgres

7 | 17911 | pgxz | postgres

7 | 21865 | pgxz | postgres

7 | 26159 | pgxz | postgres

7 | 45471 | pgxz | postgres

1. rows)

--用户操作统计

postgres=# select count(1),user\_name from tbase\_log group by user\_name order by count(1) desc limit 10;

count | user\_name

-------+-----------

10790 | pgxz

--数据库访问次数统计

postgres=# select count(1),database\_name from tbase\_log group by database\_name order by count(1) desc limit 10;

count | database\_name

-------+---------------

10790 | postgres

1. row)

--错误信息统计

postgres=# select count(1),user\_name,database\_name from tbase\_log where error\_severity='ERROR' group by user\_name,database\_name order by count(1) desc limit 10;

count | user\_name | database\_name

-------+-----------+---------------

1390 | pgxz | postgres

(1 row)

### 11.1.4、配置只收集慢的sql语句

#用户访问日志格式

log\_destination = 'csvlog'

#启用用户访问日志收集器

logging\_collector = on

#配置sql语句执行超过多少毫秒数时，语句将被记录,值为-1时禁用，0时记录所有语句

#下面配置只收集运行超过1秒的语句

log\_min\_duration\_statement = 1000

#默认不记录任何日志

log\_statement = 'none'

收集到的日志文件内容如下所示

2017-10-15 10:25:54.106 CST,"postgres","postgres",43799,"127.0.0.1:17899",59e2c65b.ab17,4,"SELECT",2017-10-15 10:22:19 CST,2/0,0,LOG,00000,"duration: 1338.366 ms statement: select \* from t where id=20000 or id=2000000;",,,,,,,,,"psql"

系统记录运行的语句及运行时间

## 11.2、如何查询数据是否倾斜

连接上不同的dn节点，查询表的容量大小，如果大小偏差较大就可以判断存在数据倾斜

[pgxz@VM\_0\_29\_centos pgxz]$ psql -p 15432 -h 172.16.0.47

psql (PostgreSQL 10 (TBase 2.01))

Type "help" for help.

postgres=# select pg\_size\_pretty(pg\_table\_size('TBase\_1'));

pg\_size\_pretty

----------------

2408 kB

(1 row)

postgres=# select pg\_size\_pretty(pg\_table\_size('TBase\_2'));

pg\_size\_pretty

----------------

896 kB

(1 row)

postgres=# \q

[pgxz@VM\_0\_29\_centos pgxz]$ psql -p 5431 -h 172.16.0.47

psql (PostgreSQL 10 (TBase 2.01))

Type "help" for help.

postgres=# select pg\_size\_pretty(pg\_table\_size('TBase\_1'));

pg\_size\_pretty

----------------

2408 kB

(1 row)

postgres=# select pg\_size\_pretty(pg\_table\_size('TBase\_2'));

pg\_size\_pretty

----------------

464 kB

(1 row)

上面数据表“TBase\_2”容量相差为一半，基本可以判定存在数据倾斜。更方便的办法请参考运维文档7.5.3。

## 11.3、如何优化有问题的Sql语句

### 11.3.1、查看是否为分布键查询

postgres=# explain select \* from tbase\_1 where f1=1;

QUERY PLAN

--------------------------------------------------------------------------------

Remote Fast Query Execution (cost=0.00..0.00 rows=0 width=0)

Node/s: dn001, dn002

-> Gather (cost=1000.00..7827.20 rows=1 width=14)

Workers Planned: 2

-> Parallel Seq Scan on tbase\_1 (cost=0.00..6827.10 rows=1 width=14)

Filter: (f1 = 1)

(6 rows)

postgres=# explain select \* from tbase\_1 where f2=1;

QUERY PLAN

--------------------------------------------------------------------------------

Remote Fast Query Execution (cost=0.00..0.00 rows=0 width=0)

Node/s: dn001

-> Gather (cost=1000.00..7827.20 rows=1 width=14)

Workers Planned: 2

-> Parallel Seq Scan on tbase\_1 (cost=0.00..6827.10 rows=1 width=14)

Filter: (f2 = 1)

(6 rows)

上面第一个查询为非分布键查询，需要发往所有节点，这样最慢的节点决定了整个业务的速度，需要保持所有节点的响应性能一致，业务设计查询时尽可能带上分布键

### 11.3.2、查看是否使用上索引

postgres=# create index tbase\_2\_f2\_idx on tbase\_2(f2);

CREATE INDEX

postgres=# explain select \* from tbase\_2 where f2=1;

QUERY PLAN

-------------------------------------------------------------------------------------

Remote Fast Query Execution (cost=0.00..0.00 rows=0 width=0)

Node/s: dn001, dn002

-> Index Scan using tbase\_2\_f2\_idx on tbase\_2 (cost=0.42..4.44 rows=1 width=14)

Index Cond: (f2 = 1)

(4 rows)

postgres=# explain select \* from tbase\_2 where f3='1';

QUERY PLAN

--------------------------------------------------------------------------------

Remote Fast Query Execution (cost=0.00..0.00 rows=0 width=0)

Node/s: dn001, dn002

-> Gather (cost=1000.00..7827.20 rows=1 width=14)

Workers Planned: 2

-> Parallel Seq Scan on tbase\_2 (cost=0.00..6827.10 rows=1 width=14)

Filter: (f3 = '1'::text)

(6 rows)

postgres=#

第一个查询使用了索引，第二个没有使用索引，通常情况下，使用索引可以加速查询速度，但要记住索引也会增加更新的开销

### 11.3.3、查看是否为分布key join

postgres=# explain select tbase\_1.\* from tbase\_1,tbase\_2 where tbase\_1.f1=tbase\_2.f1 ;

QUERY PLAN

------------------------------------------------------------------------------------------------

Remote Subquery Scan on all (dn001,dn002) (cost=29.80..186.32 rows=3872 width=40)

-> Hash Join (cost=29.80..186.32 rows=3872 width=40)

Hash Cond: (tbase\_1.f1 = tbase\_2.f1)

-> Remote Subquery Scan on all (dn001,dn002) (cost=100.00..158.40 rows=880 width=40)

Distribute results by S: f1

-> Seq Scan on tbase\_1 (cost=0.00..18.80 rows=880 width=40)

-> Hash (cost=18.80..18.80 rows=880 width=4)

-> Seq Scan on tbase\_2 (cost=0.00..18.80 rows=880 width=4)

(8 rows)

postgres=# explain select tbase\_1.\* from tbase\_1,tbase\_2 where tbase\_1.f2=tbase\_2.f1 ;

QUERY PLAN

---------------------------------------------------------------------------------

Remote Fast Query Execution (cost=0.00..0.00 rows=0 width=0)

Node/s: dn001, dn002

-> Hash Join (cost=18904.69..46257.08 rows=500564 width=14)

Hash Cond: (tbase\_1.f2 = tbase\_2.f1)

-> Seq Scan on tbase\_1 (cost=0.00..9225.64 rows=500564 width=14)

-> Hash (cost=9225.64..9225.64 rows=500564 width=4)

-> Seq Scan on tbase\_2 (cost=0.00..9225.64 rows=500564 width=4)

(7 rows)

第一查询需要数据重分布，而第二个是不需要，分布键join查询性能会更高。

### 11.3.4、查看join发生的节点

postgres=# explain select tbase\_1.\* from tbase\_1,tbase\_2 where tbase\_1.f1=tbase\_2.f1 ;

QUERY PLAN

-----------------------------------------------------------------------------------------------

Hash Join (cost=29.80..186.32 rows=3872 width=40)

Hash Cond: (tbase\_1.f1 = tbase\_2.f1)

-> Remote Subquery Scan on all (dn001,dn002) (cost=100.00..158.40 rows=880 width=40)

-> Seq Scan on tbase\_1 (cost=0.00..18.80 rows=880 width=40)

-> Hash (cost=126.72..126.72 rows=880 width=4)

-> Remote Subquery Scan on all (dn001,dn002) (cost=100.00..126.72 rows=880 width=4)

-> Seq Scan on tbase\_2 (cost=0.00..18.80 rows=880 width=4)

(7 rows)

postgres=# set prefer\_olap to on;

SET

postgres=# explain select tbase\_1.\* from tbase\_1,tbase\_2 where tbase\_1.f1=tbase\_2.f1 ;

QUERY PLAN

------------------------------------------------------------------------------------------------

Remote Subquery Scan on all (dn001,dn002) (cost=29.80..186.32 rows=3872 width=40)

-> Hash Join (cost=29.80..186.32 rows=3872 width=40)

Hash Cond: (tbase\_1.f1 = tbase\_2.f1)

-> Remote Subquery Scan on all (dn001,dn002) (cost=100.00..158.40 rows=880 width=40)

Distribute results by S: f1

-> Seq Scan on tbase\_1 (cost=0.00..18.80 rows=880 width=40)

-> Hash (cost=18.80..18.80 rows=880 width=4)

-> Seq Scan on tbase\_2 (cost=0.00..18.80 rows=880 width=4)

(8 rows)

上面join在cn节点执行，下面的在dn上重分布后再join，业务上设计一般oltp类业务在cn上进行少数据量join性能会更好

### 11.3.5、查看并行的worker数

postgres=# explain select count(1) from tbase\_1;

QUERY PLAN

---------------------------------------------------------------------------------------

Finalize Aggregate (cost=118.81..118.83 rows=1 width=8)

-> Remote Subquery Scan on all (dn001,dn002) (cost=118.80..118.81 rows=1 width=0)

-> Partial Aggregate (cost=18.80..18.81 rows=1 width=8)

-> Seq Scan on tbase\_1 (cost=0.00..18.80 rows=880 width=0)

(4 rows)

postgres=# analyze tbase\_1;

ANALYZE

postgres=# explain select count(1) from tbase\_1;

QUERY PLAN

----------------------------------------------------------------------------------------------------

Parallel Finalize Aggregate (cost=14728.45..14728.46 rows=1 width=8)

-> Parallel Remote Subquery Scan on all (dn001,dn002) (cost=14728.33..14728.45 rows=1 width=0)

-> Gather (cost=14628.33..14628.44 rows=1 width=8)

Workers Planned: 2

-> Partial Aggregate (cost=13628.33..13628.34 rows=1 width=8)

-> Parallel Seq Scan on tbase\_1 (cost=0.00..12586.67 rows=416667 width=0)

(6 rows)

上面第一个查询没走并行，analyze后走并行才是正确的，建议大数据量更新再执行analyze。

### 11.3.6、检查各个节点的执行计划是否一致

./tbase\_run\_sql\_dn\_master.sh "explain select \* from tbase\_2 where f2=1"

dn006 --- psql -h 172.16.0.13 -p 11227 -d postgres -U tbase -c "explain select \* from tbase\_2 where f2=1"

QUERY PLAN

-----------------------------------------------------------------------------

Bitmap Heap Scan on tbase\_2 (cost=2.18..7.70 rows=4 width=40)

Recheck Cond: (f2 = 1)

-> Bitmap Index Scan on tbase\_2\_f2\_idx (cost=0.00..2.18 rows=4 width=0)

Index Cond: (f2 = 1)

(4 rows)

dn002 --- psql -h 172.16.0.42 -p 11012 -d postgres -U tbase -c "explain select \* from tbase\_2 where f2=1"

QUERY PLAN

-------------------------------------------------------------------------------

Index Scan using tbase\_2\_f2\_idx on tbase\_2 (cost=0.42..4.44 rows=1 width=14)

Index Cond: (f2 = 1)

(2 rows)

这两个dn的执行计划不一致，最大可能可以是数据倾斜或者是执行计划给禁用。

如果有可能的话，DBA可以配置在系统空闲时执行全库analyze和vacuum。

## 11.4、优化实例

### 11.4.1、count(distinct xx)优化

postgres=# CREATE TABLE t1(f1 serial not null unique,f2 text,f3 text,f4 text,f5 text,f6 text,f7 text,f8 text,f9 text,f10 text,f11 text,f12 text) distribute by shard(f1);

NOTICE: Replica identity is needed for shard table, please add to this table through "alter table" command.

CREATE TABLE

Time: 89.938 ms

postgres=# insert into t1 select t,md5(t::text),md5(t::text),md5(t::text),md5(t::text),md5(t::text),md5(t::text),md5(t::text),md5(t::text),md5(t::text),md5(t::text),md5(t::text) from generate\_series(1,1000000) as t;

INSERT 0 1000000

Time: 14849.045 ms (00:14.849)

postgres=# analyze t1;

ANALYZE

Time: 1340.387 ms (00:01.340)

postgres=# explain (verbose) select count(distinct f2) from t1;

QUERY PLAN

---------------------------------------------------------------------------------------------------------------------------------------

Aggregate (cost=103320.00..103320.01 rows=1 width=8)

Output: count(DISTINCT f2)

-> Remote Subquery Scan on all (dn01,dn02,dn03,dn04,dn05,dn06,dn07,dn08,dn09,dn10) (cost=100.00..100820.00 rows=1000000 width=33)

Output: f2

-> Seq Scan on public.t1 (cost=0.00..62720.00 rows=1000000 width=33)

Output: f2

(6 rows)

Time: 0.748 ms

postgres=# select count(distinct f2) from t1;

count

---------

1000000

(1 row)

Time: 6274.684 ms (00:06.275)

postgres=# select count(distinct f2) from t1 where f1 <10;

count

-------

9

(1 row)

Time: 19.261 ms

上面发现count(distinct f2)是发生在cn节点，对于TP类业务，需要操作的数据量少的情况下，性能开销是没有问题的，而且往往比下推执行的性能开销还要小。但如果一次要操作的数据量比较大的ap类业务，则网络传输就会成功瓶颈，下面看看改写后的执行计划

postgres=# explain (verbose) select count(1) from (select f2 from t1 group by f2) as t ;

QUERY PLAN

-----------------------------------------------------------------------------------------------------------------------------------------------------------------

Finalize Aggregate (cost=355600.70..355600.71 rows=1 width=8)

Output: count(1)

-> Remote Subquery Scan on all (dn01,dn02,dn03,dn04,dn05,dn06,dn07,dn08,dn09,dn10) (cost=355600.69..355600.70 rows=1 width=0)

Output: PARTIAL count(1)

-> Partial Aggregate (cost=355500.69..355500.70 rows=1 width=8)

Output: PARTIAL count(1)

-> Group (cost=340500.69..345500.69 rows=1000000 width=33)

Output: t1.f2

Group Key: t1.f2

-> Sort (cost=340500.69..343000.69 rows=1000000 width=0)

Output: t1.f2

Sort Key: t1.f2

-> Remote Subquery Scan on all (dn01,dn02,dn03,dn04,dn05,dn06,dn07,dn08,dn09,dn10) (cost=216192.84..226192.84 rows=1000000 width=0)

Output: t1.f2

Distribute results by S: f2

-> Group (cost=216092.84..221092.84 rows=1000000 width=33)

Output: t1.f2

Group Key: t1.f2

-> Sort (cost=216092.84..218592.84 rows=1000000 width=33)

Output: t1.f2

Sort Key: t1.f2

-> Seq Scan on public.t1 (cost=0.00..62720.00 rows=1000000 width=33)

Output: t1.f2

(23 rows)

改写后，并行推到dn去执行，现在看看执行的效果

postgres=# select count(1) from (select f2 from t1 group by f2) as t ;

count

---------

1000000

(1 row)

Time: 1328.431 ms (00:01.328)

postgres=# select count(1) from (select f2 from t1 where f1<10 group by f2) as t ;

count

-------

9

(1 row)

Time: 24.991 ms

postgres=#

我们可以看到对于大量数据计算的AP类业务，性能提高了5倍

### 11.4.2、增大work\_mem减少io访问

postgres=# CREATE TABLE t1(f1 serial not null unique,f2 text,f3 text,f4 text,f5 text,f6 text,f7 text,f8 text,f9 text,f10 text,f11 text,f12 text) distribute by shard(f1);

NOTICE: Replica identity is needed for shard table, please add to this table through "alter table" command.

CREATE TABLE

Time: 70.545 ms

postgres=# CREATE TABLE t2(f1 serial not null unique,f2 text,f3 text,f4 text,f5 text,f6 text,f7 text,f8 text,f9 text,f10 text,f11 text,f12 text) distribute by shard(f1);

NOTICE: Replica identity is needed for shard table, please add to this table through "alter table" command.

CREATE TABLE

Time: 61.913 ms

postgres=# insert into t1 select t,md5(t::text),md5(t::text),md5(t::text),md5(t::text),md5(t::text),md5(t::text),md5(t::text),md5(t::text),md5(t::text),md5(t::text),md5(t::text) from generate\_series(1,1000) as t;

INSERT 0 1000

Time: 48.866 ms

postgres=# insert into t2 select t,md5(t::text),md5(t::text),md5(t::text),md5(t::text),md5(t::text),md5(t::text),md5(t::text),md5(t::text),md5(t::text),md5(t::text),md5(t::text) from generate\_series(1,50000) as t;

INSERT 0 50000

Time: 792.858 ms

postgres=# analyze t1;

ANALYZE

Time: 175.946 ms

postgres=# analyze t2;

ANALYZE

Time: 318.802 ms

postgres=#

postgres=# explain select \* from t1 where f2 not in (select f2 from t2);

QUERY PLAN

-----------------------------------------------------------------------------------------------------------------------------------------------

Remote Subquery Scan on all (dn01,dn02,dn03,dn04,dn05,dn06,dn07,dn08,dn09,dn10) (cost=0.00..2076712.50 rows=500 width=367)

-> Seq Scan on t1 (cost=0.00..2076712.50 rows=500 width=367)

Filter: (NOT (SubPlan 1))

SubPlan 1

-> Materialize (cost=0.00..4028.00 rows=50000 width=33)

-> Remote Subquery Scan on all (dn01,dn02,dn03,dn04,dn05,dn06,dn07,dn08,dn09,dn10) (cost=0.00..3240.00 rows=50000 width=33)

-> Seq Scan on t2 (cost=0.00..3240.00 rows=50000 width=33)

(7 rows)

Time: 0.916 ms

postgres=# select \* from t1 where f2 not in (select f2 from t2);

f1 | f2 | f3 | f4 | f5 | f6 | f7 | f8 | f9 | f10 | f11 | f12

----+----+----+----+----+----+----+----+----+-----+-----+-----

(0 rows)

Time: 4226.825 ms (00:04.227)

postgres=# set work\_mem to '8MB';

SET

Time: 0.289 ms

postgres=# explain select \* from t1 where f2 not in (select f2 from t2);

QUERY PLAN

-----------------------------------------------------------------------------------------------------------------------------------------

Remote Subquery Scan on all (dn01,dn02,dn03,dn04,dn05,dn06,dn07,dn08,dn09,dn10) (cost=3365.00..3577.50 rows=500 width=367)

-> Seq Scan on t1 (cost=3365.00..3577.50 rows=500 width=367)

Filter: (NOT (hashed SubPlan 1))

SubPlan 1

-> Remote Subquery Scan on all (dn01,dn02,dn03,dn04,dn05,dn06,dn07,dn08,dn09,dn10) (cost=0.00..3240.00 rows=50000 width=33)

-> Seq Scan on t2 (cost=0.00..3240.00 rows=50000 width=33)

(6 rows)

Time: 0.890 ms

postgres=# select \* from t1 where f2 not in (select f2 from t2);

f1 | f2 | f3 | f4 | f5 | f6 | f7 | f8 | f9 | f10 | f11 | f12

----+----+----+----+----+----+----+----+----+-----+-----+-----

(0 rows)

Time: 105.249 ms

postgres=#

增大work\_mem后，性能提高了40倍，因为work\_mem足够放下filter的数据，不需要再做 Materialize物化，filter由原来的subplan变成了hash subplan，直接在内存hash表中filter，性能就上去了

注意，work\_mem默认不宜过大，建议在某个具体的查询语句中再根据需要进行调整即可。

### 11.4.3、not in改写为anti join

上面11.4.2通过增大计算内存达到提高性能，但内存不可能无限扩大，下面通过改写语句也可以达到提高查询的性能。

postgres=# explain select \* from t1 left outer join t2 on t1.f2 = t2.f2 where t2.f2 is null;

QUERY PLAN

-------------------------------------------------------------------------------------------------------------------------------------------------

Remote Subquery Scan on all (dn01,dn02,dn03,dn04,dn05,dn06,dn07,dn08,dn09,dn10) (cost=6405.00..9260.75 rows=1 width=734)

-> Hash Anti Join (cost=6405.00..9260.75 rows=1 width=734)

Hash Cond: (t1.f2 = t2.f2)

-> Remote Subquery Scan on all (dn01,dn02,dn03,dn04,dn05,dn06,dn07,dn08,dn09,dn10) (cost=100.00..682.00 rows=1000 width=367)

Distribute results by S: f2

-> Seq Scan on t1 (cost=0.00..210.00 rows=1000 width=367)

-> Hash (cost=21940.00..21940.00 rows=50000 width=367)

-> Remote Subquery Scan on all (dn01,dn02,dn03,dn04,dn05,dn06,dn07,dn08,dn09,dn10) (cost=100.00..21940.00 rows=50000 width=367)

Distribute results by S: f2

-> Seq Scan on t2 (cost=0.00..3240.00 rows=50000 width=367)

(10 rows)

Time: 1.047 ms

postgres=# select \* from t1 left outer join t2 on t1.f2 = t2.f2 where t2.f2 is null;

f1 | f2 | f3 | f4 | f5 | f6 | f7 | f8 | f9 | f10 | f11 | f12 | f1 | f2 | f3 | f4 | f5 | f6 | f7 | f8 | f9 | f10 | f11 | f12

----+----+----+----+----+----+----+----+----+-----+-----+-----+----+----+----+----+----+----+----+----+----+-----+-----+-----

(0 rows)

Time: 107.233 ms

postgres=#

也可以修改not exists

postgres=# explain select \* from t1 where not exists( select 1 from t2 where t1.f2=t2.f2);

QUERY PLAN

-----------------------------------------------------------------------------------------------------------------------------------------------

Remote Subquery Scan on all (dn01,dn02,dn03,dn04,dn05,dn06,dn07,dn08,dn09,dn10) (cost=3865.00..4078.75 rows=1 width=367)

-> Hash Anti Join (cost=3865.00..4078.75 rows=1 width=367)

Hash Cond: (t1.f2 = t2.f2)

-> Remote Subquery Scan on all (dn01,dn02,dn03,dn04,dn05,dn06,dn07,dn08,dn09,dn10) (cost=100.00..682.00 rows=1000 width=367)

Distribute results by S: f2

-> Seq Scan on t1 (cost=0.00..210.00 rows=1000 width=367)

-> Hash (cost=5240.00..5240.00 rows=50000 width=33)

-> Remote Subquery Scan on all (dn01,dn02,dn03,dn04,dn05,dn06,dn07,dn08,dn09,dn10) (cost=100.00..5240.00 rows=50000 width=33)

Distribute results by S: f2

-> Seq Scan on t2 (cost=0.00..3240.00 rows=50000 width=33)

(10 rows)

Time: 0.974 ms

postgres=# select \* from t1 where not exists( select 1 from t2 where t1.f2=t2.f2);

f1 | f2 | f3 | f4 | f5 | f6 | f7 | f8 | f9 | f10 | f11 | f12

----+----+----+----+----+----+----+----+----+-----+-----+-----

(0 rows)

Time: 42.944 ms

postgres=#

### 11.4.4、分布key jon+limit优化

--数据准备

postgres=# CREATE TABLE t1(f1 serial not null unique,f2 text,f3 text,f4 text,f5 text,f6 text,f7 text,f8 text,f9 text,f10 text,f11 text,f12 text) distribute by shard(f1);

NOTICE: Replica identity is needed for shard table, please add to this table through "alter table" command.

CREATE TABLE

postgres=# CREATE TABLE t2(f1 serial not null unique,f2 text,f3 text,f4 text,f5 text,f6 text,f7 text,f8 text,f9 text,f10 text,f11 text,f12 text) distribute by shard(f1);

NOTICE: Replica identity is needed for shard table, please add to this table through "alter table" command.

CREATE TABLE

postgres=# insert into t1 select t,md5(t::text),md5(t::text),md5(t::text),md5(t::text),md5(t::text),md5(t::text),md5(t::text),md5(t::text),md5(t::text),md5(t::text),md5(t::text) from generate\_series(1,1000000) as t;

INSERT 0 1000000

postgres=# insert into t2 select t,md5(t::text),md5(t::text),md5(t::text),md5(t::text),md5(t::text),md5(t::text),md5(t::text),md5(t::text),md5(t::text),md5(t::text),md5(t::text) from generate\_series(1,1000000) as t;

INSERT 0 1000000

postgres=# analyze t1;

ANALYZE

postgres=# analyze t2;

ANALYZE

postgres=#

postgres=# \timing

Timing is on.

postgres=# explain select t1.\* from t1,t2 where t1.f1=t2.f1 limit 10;

QUERY PLAN

----------------------------------------------------------------------------------------------------------------------------------------------

Limit (cost=0.25..1.65 rows=10 width=367)

-> Merge Join (cost=0.25..140446.26 rows=1000000 width=367)

Merge Cond: (t1.f1 = t2.f1)

-> Remote Subquery Scan on all (dn01,dn02,dn03,dn04,dn05,dn06,dn07,dn08,dn09,dn10) (cost=100.12..434823.13 rows=1000000 width=367)

-> Index Scan using t1\_f1\_key on t1 (cost=0.12..62723.13 rows=1000000 width=367)

-> Remote Subquery Scan on all (dn01,dn02,dn03,dn04,dn05,dn06,dn07,dn08,dn09,dn10) (cost=100.12..71823.13 rows=1000000 width=4)

-> Index Only Scan using t2\_f1\_key on t2 (cost=0.12..62723.13 rows=1000000 width=4)

(7 rows)

Time: 1.372 ms

postgres=# explain analyze select t1.\* from t1,t2 where t1.f1=t2.f1 limit 10;

QUERY PLAN

--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

Limit (cost=0.25..1.65 rows=10 width=367) (actual time=2675.437..2948.199 rows=10 loops=1)

-> Merge Join (cost=0.25..140446.26 rows=1000000 width=367) (actual time=2675.431..2675.508 rows=10 loops=1)

Merge Cond: (t1.f1 = t2.f1)

-> Remote Subquery Scan on all (dn01,dn02,dn03,dn04,dn05,dn06,dn07,dn08,dn09,dn10) (cost=100.12..434823.13 rows=1000000 width=367) (actual time=1.661..1.704 rows=10 loops=1)

-> Remote Subquery Scan on all (dn01,dn02,dn03,dn04,dn05,dn06,dn07,dn08,dn09,dn10) (cost=100.12..71823.13 rows=1000000 width=4) (actual time=2673.761..2673.783 rows=10 loops=1)

Planning time: 0.358 ms

Execution time: 2973.948 ms

(7 rows)

Time: 2976.008 ms (00:02.976)

postgres=#

看执行计划是在cn上面执行，merge join需要把要join的数据拉回cn再排序，然后再join，这里主切的开销在于网络，优化的话方法就是让语句其推下去计算

postgres=# set prefer\_olap to on;

SET

Time: 0.291 ms

postgres=# explain select t1.\* from t1,t2 where t1.f1=t2.f1 limit 10;

QUERY PLAN

--------------------------------------------------------------------------------------------------------------------------------

Limit (cost=100.25..101.70 rows=10 width=367)

-> Remote Subquery Scan on all (dn01,dn02,dn03,dn04,dn05,dn06,dn07,dn08,dn09,dn10) (cost=100.25..101.70 rows=10 width=367)

-> Limit (cost=0.25..1.65 rows=10 width=367)

-> Merge Join (cost=0.25..140446.26 rows=1000000 width=367)

Merge Cond: (t1.f1 = t2.f1)

-> Index Scan using t1\_f1\_key on t1 (cost=0.12..62723.13 rows=1000000 width=367)

-> Index Only Scan using t2\_f1\_key on t2 (cost=0.12..62723.13 rows=1000000 width=4)

(7 rows)

Time: 1.061 ms

postgres=# explain analyze select t1.\* from t1,t2 where t1.f1=t2.f1 limit 10;

QUERY PLAN

---------------------------------------------------------------------------------------------------------------------------------------------------------------------------

Limit (cost=100.25..101.70 rows=10 width=367) (actual time=1.527..3.899 rows=10 loops=1)

-> Remote Subquery Scan on all (dn01,dn02,dn03,dn04,dn05,dn06,dn07,dn08,dn09,dn10) (cost=100.25..101.70 rows=10 width=367) (actual time=1.525..1.529 rows=10 loops=1)

Planning time: 0.360 ms

Execution time: 18.193 ms

(4 rows)

Time: 19.921 ms

相差150倍的性能，一般情况下，如果需要拉大量的数据回cn计算，则下推执行的效率会更好

### 11.4.5、非分布key join使用hash join性能一般最好

为了提高tp类业务查询的性能，我们经常需要对一些字段建立索引，使用有索引字段join时系统往往也会使用Merge Cond和nestloop

mydb=# CREATE TABLE t1(f1 serial not null,f2 text,f3 text,f4 text,f5 text,f6 text,f7 text,f8 text,f9 text,f10 text,f11 text,f12 text) distribute by shard(f1);

NOTICE: Replica identity is needed for shard table, please add to this table through "alter table" command.

CREATE TABLE

Time: 481.042 ms

mydb=# create index t1\_f1\_idx on t1(f2);

CREATE INDEX

Time: 85.521 ms

mydb=# CREATE TABLE t2(f1 serial not null,f2 text,f3 text,f4 text,f5 text,f6 text,f7 text,f8 text,f9 text,f10 text,f11 text,f12 text) distribute by shard(f1);

NOTICE: Replica identity is needed for shard table, please add to this table through "alter table" command.

CREATE TABLE

Time: 75.973 ms

mydb=# create index t2\_f1\_idx on t2(f2);

CREATE INDEX

Time: 29.890 ms

mydb=# insert into t1 select t,md5(t::text),md5(t::text),md5(t::text),md5(t::text),md5(t::text),md5(t::text),md5(t::text),md5(t::text),md5(t::text),md5(t::text),md5(t::text) from generate\_series(1,1000000) as t;

INSERT 0 1000000

Time: 16450.623 ms (00:16.451)

mydb=# insert into t2 select t,md5(t::text),md5(t::text),md5(t::text),md5(t::text),md5(t::text),md5(t::text),md5(t::text),md5(t::text),md5(t::text),md5(t::text),md5(t::text) from generate\_series(1,1000000) as t;

INSERT 0 1000000

Time: 17218.738 ms (00:17.219)

mydb=# analyze t1;

ANALYZE

Time: 2219.341 ms (00:02.219)

mydb=# analyze t2;

ANALYZE

Time: 1649.506 ms (00:01.650)

mydb=#

--merge join

mydb=# explain select t1.\* from t1,t2 where t1.f2=t2.f2 limit 10;

QUERY PLAN

-------------------------------------------------------------------------------------------------------------------------

Limit (cost=100.25..102.78 rows=10 width=367)

-> Remote Subquery Scan on all (dn001,dn002) (cost=100.25..102.78 rows=10 width=367)

-> Limit (cost=0.25..2.73 rows=10 width=367)

-> Merge Join (cost=0.25..248056.80 rows=1000000 width=367)

Merge Cond: (t1.f2 = t2.f2)

-> Remote Subquery Scan on all (dn001,dn002) (cost=100.12..487380.85 rows=1000000 width=367)

Distribute results by S: f2

-> Index Scan using t1\_f1\_idx on t1 (cost=0.12..115280.85 rows=1000000 width=367)

-> Materialize (cost=100.12..155875.95 rows=1000000 width=33)

-> Remote Subquery Scan on all (dn001,dn002) (cost=100.12..153375.95 rows=1000000 width=33)

Distribute results by S: f2

-> Index Only Scan using t2\_f1\_idx on t2 (cost=0.12..115275.95 rows=1000000 width=33)

(12 rows)

Time: 4.183 ms

mydb=# explain analyze select t1.\* from t1,t2 where t1.f2=t2.f2 limit 10;

QUERY PLAN

-------------------------------------------------------------------------------------------------------------------------------------------

Limit (cost=100.25..102.78 rows=10 width=367) (actual time=6555.346..6556.296 rows=10 loops=1)

-> Remote Subquery Scan on all (dn001,dn002) (cost=100.25..102.78 rows=10 width=367) (actual time=6555.343..6555.349 rows=10 loops=1)

Planning time: 0.473 ms

Execution time: 6569.828 ms

(4 rows)

Time: 6614.439 ms (00:06.614)

--nested loop

mydb=# set enable\_mergejoin to off;

SET

Time: 0.422 ms

mydb=# explain select t1.\* from t1,t2 where t1.f2=t2.f2 limit 10;

QUERY PLAN

--------------------------------------------------------------------------------------------------------------------

Limit (cost=100.12..103.57 rows=10 width=367)

-> Remote Subquery Scan on all (dn001,dn002) (cost=100.12..103.57 rows=10 width=367)

-> Limit (cost=0.12..3.52 rows=10 width=367)

-> Nested Loop (cost=0.12..339232.00 rows=1000000 width=367)

-> Remote Subquery Scan on all (dn001,dn002) (cost=100.00..434740.00 rows=1000000 width=367)

Distribute results by S: f2

-> Seq Scan on t1 (cost=0.00..62640.00 rows=1000000 width=367)

-> Materialize (cost=100.12..100.31 rows=1 width=33)

-> Remote Subquery Scan on all (dn001,dn002) (cost=100.12..100.30 rows=1 width=33)

Distribute results by S: f2

-> Index Only Scan using t2\_f1\_idx on t2 (cost=0.12..0.27 rows=1 width=33)

Index Cond: (f2 = t1.f2)

(12 rows)

Time: 1.033 ms

mydb=# explain analyze select t1.\* from t1,t2 where t1.f2=t2.f2 limit 10;

QUERY PLAN

-------------------------------------------------------------------------------------------------------------------------------------------

Limit (cost=100.12..103.57 rows=10 width=367) (actual time=5608.326..5609.571 rows=10 loops=1)

-> Remote Subquery Scan on all (dn001,dn002) (cost=100.12..103.57 rows=10 width=367) (actual time=5608.323..5608.349 rows=10 loops=1)

Planning time: 0.347 ms

Execution time: 5669.901 ms

(4 rows)

Time: 5672.584 ms (00:05.673)

mydb=# set enable\_nestloop to off;

SET

Time: 0.436 ms

mydb=# explain select t1.\* from t1,t2 where t1.f2=t2.f2 limit 10;

QUERY PLAN

-------------------------------------------------------------------------------------------------------------------------

Limit (cost=85983.00..85984.94 rows=10 width=367)

-> Remote Subquery Scan on all (dn001,dn002) (cost=85983.00..85984.94 rows=10 width=367)

-> Limit (cost=85883.00..85884.89 rows=10 width=367)

-> Hash Join (cost=85883.00..274580.00 rows=1000000 width=367)

Hash Cond: (t1.f2 = t2.f2)

-> Remote Subquery Scan on all (dn001,dn002) (cost=100.00..434740.00 rows=1000000 width=367)

Distribute results by S: f2

-> Seq Scan on t1 (cost=0.00..62640.00 rows=1000000 width=367)

-> Hash (cost=100740.00..100740.00 rows=1000000 width=33)

-> Remote Subquery Scan on all (dn001,dn002) (cost=100.00..100740.00 rows=1000000 width=33)

Distribute results by S: f2

-> Seq Scan on t2 (cost=0.00..62640.00 rows=1000000 width=33)

(12 rows)

Time: 1.141 ms

mydb=# explain analyze select t1.\* from t1,t2 where t1.f2=t2.f2 limit 10;

QUERY PLAN

-----------------------------------------------------------------------------------------------------------------------------------------------

Limit (cost=85983.00..85984.94 rows=10 width=367) (actual time=1083.691..1085.962 rows=10 loops=1)

-> Remote Subquery Scan on all (dn001,dn002) (cost=85983.00..85984.94 rows=10 width=367) (actual time=1083.688..1083.699 rows=10 loops=1)

Planning time: 0.530 ms

Execution time: 1108.830 ms

(4 rows)

Time: 1117.713 ms (00:01.118)

mydb=#

### 11.4.6、exists的优化

exists在数据量比较大情况下，一般使用的是Semi Join ，在work\_mem足够大的情况下走的是hash join，性能会更好

postgres=# show work\_mem;

work\_mem

----------

4MB

(1 row)

Time: 0.298 ms

postgres=# explain select count(1) from t1 where exists(select 1 from t2 where t2.t1\_f1=t1.f1);

QUERY PLAN

-----------------------------------------------------------------------------------------------------------------------

Finalize Aggregate (cost=242218.32..242218.33 rows=1 width=8)

-> Remote Subquery Scan on all (dn001,dn002) (cost=242218.30..242218.32 rows=1 width=0)

-> Partial Aggregate (cost=242118.30..242118.31 rows=1 width=8)

-> Hash Semi Join (cost=110248.00..242118.30 rows=505421 width=0)

Hash Cond: (t1.f1 = t2.t1\_f1)

-> Seq Scan on t1 (cost=0.00..17420.00 rows=1000000 width=4)

-> Hash (cost=79340.00..79340.00 rows=3000000 width=4)

-> Remote Subquery Scan on all (dn001,dn002) (cost=100.00..79340.00 rows=3000000 width=4)

Distribute results by S: t1\_f1

-> Seq Scan on t2 (cost=0.00..52240.00 rows=3000000 width=4)

(10 rows)

Time: 1.091 ms

postgres=# select count(1) from t1 where exists(select 1 from t2 where t2.t1\_f1=t1.f1);

count

--------

500000

(1 row)

Time: 3779.401 ms (00:03.779)

postgres=# set work\_mem to '128MB';

SET

Time: 0.368 ms

postgres=# explain select count(1) from t1 where exists(select 1 from t2 where t2.t1\_f1=t1.f1);

QUERY PLAN

------------------------------------------------------------------------------------------------------------------

Finalize Aggregate (cost=101763.76..101763.77 rows=1 width=8)

-> Remote Subquery Scan on all (dn001,dn002) (cost=101763.75..101763.76 rows=1 width=0)

-> Partial Aggregate (cost=101663.75..101663.76 rows=1 width=8)

-> Hash Join (cost=89660.00..101663.75 rows=505421 width=0)

Hash Cond: (t2.t1\_f1 = t1.f1)

-> Remote Subquery Scan on all (dn001,dn002) (cost=59840.00..69443.00 rows=505421 width=4)

Distribute results by S: t1\_f1

-> HashAggregate (cost=59740.00..64794.21 rows=505421 width=4)

Group Key: t2.t1\_f1

-> Seq Scan on t2 (cost=0.00..52240.00 rows=3000000 width=4)

-> Hash (cost=17420.00..17420.00 rows=1000000 width=4)

-> Seq Scan on t1 (cost=0.00..17420.00 rows=1000000 width=4)

(12 rows)

Time: 4.739 ms

postgres=# select count(1) from t1 where exists(select 1 from t2 where t2.t1\_f1=t1.f1);

count

--------

500000

(1 row)

Time: 1942.037 ms (00:01.942)

postgres=#

大约有一倍性能的提升

# 12、TBase-插件使用

## 12.1、插件查看，添加和删除

12.1.1、查看数据库加载了那些插件

postgres=# SELECT e.extname AS "Name", e.extversion AS "Version", n.nspname AS "Schema", c.description AS "Description" FROM pg\_catalog.pg\_extension e LEFT JOIN pg\_catalog.pg\_namespace n ON n.oid = e.extnamespace LEFT JOIN pg\_catalog.pg\_description c ON c.objoid = e.oid AND c.classoid = 'pg\_catalog.pg\_extension'::pg\_catalog.regclass ORDER BY 1;

Name | Version | Schema | Description

--------------------+---------+------------+-----------------------------------------------------------

pageinspect | 1.1 | public | inspect the contents of database pages at a low level

pg\_errcode\_stat | 1.1 | public | track error code of all processes

pg\_stat\_statements | 1.1 | public | track execution statistics of all SQL statements executed

plpgsql | 1.0 | pg\_catalog | PL/pgSQL procedural language

shard\_statistic | 1.0 | public | tools for get shard statistic

(5 rows)

12.1.2、添加插件

postgres=# create extension "uuid-ossp" with schema tbase;

CREATE EXTENSION

上面的语句把"uuid-ossp"创建到模式tbase下

12.1.3、删除插件

postgres=# drop extension "uuid-ossp" ;

DROP EXTENSION

## 12.2、插件uuid-ossp使用

12.2.1、uuid-ossp功能介绍及添加方法

uuid-ossp模块提供函数使用几种标准算法之一产生通用唯一标识符（UUID）。还提供产生某些特殊 UUID 常量的函数。该模块提供的功能函数可用于替代序列的，而且性能比序列更好。

给数据库添加uuid-ossp插件

postgres=# create extension "uuid-ossp" with schema tbase;

CREATE EXTENSION

12.2.2、uuid常量函数

#### 12.2.2.1、uuid\_nil()

这是一个"nil" UUID 常量

postgres=# select uuid\_nil();

uuid\_nil

--------------------------------------

00000000-0000-0000-0000-000000000000

1. row)

#### 12.2.2.2、uuid\_ns\_dns()

为 UUID 指定 DNS 名字空间的常量

postgres=# select uuid\_ns\_dns();

uuid\_ns\_dns

--------------------------------------

6ba7b810-9dad-11d1-80b4-00c04fd430c8

(1 row)

#### 12.2.2.3、uuid\_ns\_url()

为 UUID 指定 URL 名字空间的常量。

postgres=# select uuid\_ns\_url();

uuid\_ns\_url

--------------------------------------

6ba7b811-9dad-11d1-80b4-00c04fd430c8

1. row)

#### 12.2.2.4、uuid\_ns\_oid()

为 UUID 指定 ISO 对象标识符（OID） 名字空间的常量

postgres=# select uuid\_ns\_oid();

uuid\_ns\_oid

--------------------------------------

6ba7b812-9dad-11d1-80b4-00c04fd430c8

(1 row)

#### 12.2.2.5、uuid\_ns\_x500()

为 UUID 指定 X.500 可识别名（DN）名字空间的常量

postgres=# select uuid\_ns\_x500();

uuid\_ns\_x500

--------------------------------------

6ba7b814-9dad-11d1-80b4-00c04fd430c8

(1 row)

12.2.3、uuid生成函数

#### 12.2.3.1、uuid\_generate\_v1()

这个函数产生一个版本 1 的 UUID。这涉及到计算机的 MAC 地址和一个时间戳。

postgres=# select uuid\_generate\_v1(),clock\_timestamp();

uuid\_generate\_v1 | clock\_timestamp

--------------------------------------+------------------------------

e909c7f0-c36b-11e7-984d-525400efe0ca | 2017-11-07 11:29:49.67391+08

(1 row)

postgres=# select uuid\_generate\_v1(),clock\_timestamp();

uuid\_generate\_v1 | clock\_timestamp

--------------------------------------+-------------------------------

e9f24458-c36b-11e7-9f81-525400efe0ca | 2017-11-07 11:29:51.197516+08

1. row)

#### 12.2.3.2、uuid\_generate\_v1mc()

这个函数产生一个版本 1 的 UUID，但是使用一个随机广播 MAC 地址而不是该计算机真实的 MAC 地址。

postgres=# select uuid\_generate\_v1mc();

uuid\_generate\_v1mc

--------------------------------------

1538d848-c36c-11e7-9043-5749d89bc820

1. row)

#### 12.2.3.3、uuid\_generate\_v3(namespace uuid, name text)

这个函数使用指定的输入名称在给定的名字空间中产生一个版本 3 的 UUID,名称参数将使用 MD5 进行哈希，因此从产生的 UUID 中得不到明文

postgres=# SELECT uuid\_generate\_v3(uuid\_ns\_url(), 'http://tbase.qq.com');

uuid\_generate\_v3

--------------------------------------

af6371f5-bcd5-300b-8a35-7186d54009d5

(1 row)

postgres=# SELECT uuid\_generate\_v3(uuid\_ns\_dns(), 'http://tbase.qq.com');

uuid\_generate\_v3

--------------------------------------

246a9a9c-cff2-3b34-9ddd-80cc6e30a222

(1 row)

#### 12.2.3.4、uuid\_generate\_v4()

这个函数产生一个版本 4 的 UUID，它完全从随机数产生。

postgres=# select uuid\_generate\_v4();

uuid\_generate\_v4

--------------------------------------

dfc68e17-6b97-496e-a992-531aca74ef18

(1 row)

postgres=# select uuid\_generate\_v4();

uuid\_generate\_v4

--------------------------------------

e96b64c2-cf1e-423b-845c-541e9b3901a3

(1 row)

#### 12.2.3.5、uuid\_generate\_v5(namespace uuid, name text)

这个函数产生一个版本 5 的 UUID，它和版本 3 的 UUID 相似，但是采用的是 SHA-1 作为哈希方法。版本 5 比版本 3 更好，因为 SHA-1 被认为比 MD5 更安全。

postgres=# SELECT uuid\_generate\_v5(uuid\_ns\_url(), 'http://tbase.qq.com');

uuid\_generate\_v5

--------------------------------------

d8381c8e-6899-5e43-ab26-18d3660d7db1

(1 row)

postgres=# SELECT uuid\_generate\_v5(uuid\_ns\_dns(), 'http://tbase.qq.com');

uuid\_generate\_v5

--------------------------------------

d5051eb0-2051-53be-8d36-967851c3946d

1. row)

12.2.4、在数据表中使用uuid默认值

postgres=# create table t\_uuid(f1 uuid not null default uuid\_generate\_v1(),f2 varchar(256) );

CREATE TABLE

postgres=# \d+ t\_uuid

Table "public.t\_uuid"

Column | Type | Modifiers | Storage | Stats target | Description

--------+------------------------+-------------------------------------+----------+--------------+-------------

f1 | uuid | not null default uuid\_generate\_v1() | plain | |

f2 | character varying(256) | | extended | |

Has OIDs: no

Distribute By SHARD(f2)

Location Nodes: ALL DATANODES

postgres=# insert into t\_uuid (f2) values(uuid\_generate\_v4());

INSERT 0 1

postgres=# select \* from t\_uuid;

f1 | f2

--------------------------------------+--------------------------------------

75dfb6b0-c382-11e7-9f82-525400efe0ca | 2b99799c-d77e-4023-81db-3e5c6ad901b0

1. row)

12.2.5、uuid各函数性能对比

* 数据表结构

postgres=# \d+ t\_uuid

Table "public.t\_uuid"

Column | Type | Modifiers | Storage | Stats target | Description

--------+------------------------+-------------------------------------+----------+--------------+-------------

f1 | uuid | not null default uuid\_generate\_v1() | plain | |

f2 | character varying(256) | | extended | |

Has OIDs: no

Distribute By SHARD(f2)

Location Nodes: ALL DATANODES

* uuid\_generate\_v3函数

postgres=# insert into t\_uuid(f1,f2) select uuid\_generate\_v3(uuid\_ns\_dns(), 'http://tbase.qq.com'),uuid\_generate\_v3(uuid\_ns\_dns(), 'http://tbase.qq.com') from generate\_series(1,10000) as t;

INSERT 0 10000

Time: 58.407 ms

* uuid\_generate\_v5函数

postgres=# insert into t\_uuid(f1,f2) select uuid\_generate\_v5(uuid\_ns\_dns(), 'http://tbase.qq.com'),uuid\_generate\_v5(uuid\_ns\_dns(), 'http://tbase.qq.com') from generate\_series(1,10000) as t;

INSERT 0 10000

Time: 47.049 ms

* uuid\_generate\_v1函数

postgres=# insert into t\_uuid(f1,f2) select uuid\_generate\_v1(),uuid\_generate\_v1() from generate\_series(1,10000) as t;

INSERT 0 10000

Time: 111.204 ms

* uuid\_generate\_v4函数

postgres=# insert into t\_uuid(f1,f2) select uuid\_generate\_v4(),uuid\_generate\_v4() from generate\_series(1,10000) as t;

INSERT 0 10000

Time: 137.062 ms

* uuid\_generate\_v1mc函数

postgres=# insert into t\_uuid(f1,f2) select uuid\_generate\_v1mc(),uuid\_generate\_v1mc() from generate\_series(1,10000) as t;

INSERT 0 10000

Time: 125.663 ms

* 测试数据对比

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| V1 | v1mc | V4 | V3 | V5 |
| 111.204 ms | 125.663 ms | 137.062 ms | 58.407 ms | 47.049 ms |

可以看出V5的性能是最好的

12.2.6、uuid与serial 在TBase中性能对比

* uuid插入1万条数据

postgres=# \d+ t\_uuid

Table "public.t\_uuid"

Column | Type | Modifiers | Storage | Stats target | Description

--------+------------------------+-------------------------------------+----------+--------------+-------------

f1 | uuid | not null default uuid\_generate\_v1() | plain | |

f2 | character varying(256) | | extended | |

Has OIDs: no

Distribute By SHARD(f2)

Location Nodes: ALL DATANODES

postgres=# insert into t\_uuid(f2) select t from generate\_series(1,1000000) as t;

INSERT 0 1000000

Time: 4869.140 ms (00:04.869)

* serial插入1万条数据

postgres=# \d+ t\_serial

Table "public.t\_serial"

Column | Type | Modifiers | Storage | Stats target | Description

--------+------------------------+-------------------------------------------------------+----------+--------------+-------------

f1 | integer | not null default nextval('t\_serial\_f1\_seq'::regclass) | plain | |

f2 | character varying(256) | | extended | |

Has OIDs: no

Distribute By SHARD(f1)

Location Nodes: ALL DATANODES

postgres=# insert into t\_serial(f2) select t from generate\_series(1,1000000) as t;

INSERT 0 1000000

Time: 3683.533 ms (00:03.684)

* uuid pgbench结果

pghost: 172.16.0.42 pgport: 11016 nclients: 32 duration: 600 dbName: postgres

transaction type: insert\_t\_uuid.sql

scaling factor: 1

query mode: prepared

number of clients: 32

number of threads: 1

duration: 600 s

number of transactions actually processed: 531667

latency average = 36.166 ms

tps = 884.807291 (including connections establishing)

tps = 884.813103 (excluding connections establishing)

script statistics:

- statement latencies in milliseconds:

0.013 \set id random(1, 10000000)

36.094 insert into t\_uuid(f2) values(:id::text) ;

* serial pgbench结果

pghost: 172.16.0.42 pgport: 11016 nclients: 32 duration: 600 dbName: postgres

transaction type: insert\_t\_serial.sql

scaling factor: 1

query mode: prepared

number of clients: 32

number of threads: 1

duration: 600 s

number of transactions actually processed: 493799

latency average = 38.889 ms

tps = 822.864578 (including connections establishing)

tps = 822.869984 (excluding connections establishing)

script statistics:

- statement latencies in milliseconds:

0.013 \set id random(1, 10000000)

38.861 insert into t\_serial(f2) values(:id::text) ;

* 测试数据对比

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| uuid用时 | Serial用时 | 百分比 | Uuid pgbench | Serial pgbench | 百分比 |
| 4.869s | 3.684s | 75.66% | 884 | 822 | 92.98% |

12.2.7、占用空间对比

* UUID表

postgres=# \d+ t\_uuid

Table "public.t\_uuid"

Column | Type | Collation | Nullable | Default | Storage | Stats target | Description

--------+------------------------+-----------+----------+--------------------+----------+--------------+-------------

f1 | uuid | | not null | uuid\_generate\_v1() | plain | |

f2 | character varying(256) | | | | extended | |

Distribute By: SHARD(f2)

Location Nodes: ALL DATANODES

postgres=# select count(1),pg\_size\_pretty(pg\_table\_size('t\_uuid')) from t\_uuid;

count | pg\_size\_pretty

---------+----------------

1000000 | 73 MB

(1 row)

* Serial表

postgres=# \d+ t\_serial

Table "public.t\_serial"

Column | Type | Collation | Nullable | Default | Storage | Stats target | Description

--------+------------------------+-----------+----------+--------------------------------------+----------+--------------+-------------

id | integer | | not null | nextval('t\_serial\_id\_seq'::regclass) | plain | |

f2 | character varying(256) | | | | extended | |

Distribute By: SHARD(id)

Location Nodes: ALL DATANODES

postgres=# select count(1),pg\_size\_pretty(pg\_table\_size('t\_serial')) from t\_serial;

count | pg\_size\_pretty

---------+----------------

1000000 | 66 MB

(1 row)

Time: 186.737 ms

* 测试数据对比

|  |  |  |
| --- | --- | --- |
| Uuid/10000条记录 | Serial/10000条记录 | 占用比 |
| 73MB | 66MB | 1.1倍 |

12.2.8、使用uuid做为分布列的方案

目前uuid类型字段不能做为分布列，提示如下

postgres=# create table t\_uuid(f1 uuid );

ERROR: No appropriate column can be used as distribute key because of data type.

解决方案如下

postgres=# create table t\_uuid(f1 varchar(36) default uuid\_nil()::text );

CREATE TABLE

postgres=#

12.2.9、使用建议

性能上序列和uuid相关不大，但serial每次都需要与gtm通信，会加大gtm的通信压力，所以尽可能使用uuid来代替serial

## 12.3、插件pg\_stat\_statements使用

12.3.1、pg\_stat\_statements功能介绍及添加方法

pg\_stat\_statements模块提供一种方法追踪一个服务器所执行的所有 SQL 语句的执行统计信息。该模块必须通过在postgresql.conf的shared\_preload\_libraries中增加pg\_stat\_statements来载入，因为它需要额外的共享内存。增加或移除该模块需要服务器重启才能生效。当pg\_stat\_statements被载入时，它会跟踪该服务器 的所有数据库的统计信息。该模块提供了一个视图 pg\_stat\_statements以及函数pg\_stat\_statements\_reset 和pg\_stat\_statements用于访问和操纵这些统计信息。

给数据库添加pg\_stat\_statements插件

postgres=# create extension pg\_stat\_statements with schema tbase;

CREATE EXTENSION

12.3.2、获取执行次数最多的语句

postgres=# select

pg\_authid.rolname as rolname,

pg\_database.datname as datname,

pg\_stat\_statements.query,

pg\_stat\_statements.calls,

pg\_stat\_statements.total\_time,

round((pg\_stat\_statements.total\_time/pg\_stat\_statements.calls)::numeric,6) as runtime\_of\_once

from

pg\_stat\_statements

inner join pg\_authid on pg\_authid.oid=pg\_stat\_statements.userid

inner join pg\_database on pg\_database.oid= pg\_stat\_statements.dbid

order by

pg\_stat\_statements.calls desc

limit 10;

12.3.3、获取执行总时间最长的语句

postgres=# select

pg\_authid.rolname as rolname,

pg\_database.datname as datname,

pg\_stat\_statements.query,

pg\_stat\_statements.calls,

pg\_stat\_statements.total\_time,

pg\_stat\_statements.total\_time/pg\_stat\_statements.calls AS onestime

from

pg\_stat\_statements

inner join pg\_authid on pg\_authid.oid=pg\_stat\_statements.userid

inner join pg\_database on pg\_database.oid= pg\_stat\_statements.dbid

order by

pg\_stat\_statements.total\_time desc

limit 10;

12.3.4、获取每句平均执行时间最长的语句

postgres=# select

pg\_authid.rolname as rolname,

pg\_database.datname as datname,

pg\_stat\_statements.query,

pg\_stat\_statements.calls,

pg\_stat\_statements.total\_time,

pg\_stat\_statements.total\_time/pg\_stat\_statements.calls as onestime

from

pg\_stat\_statements

inner join pg\_authid on pg\_authid.oid=pg\_stat\_statements.userid

inner join pg\_database on pg\_database.oid= pg\_stat\_statements.dbid

order by

pg\_stat\_statements.total\_time/pg\_stat\_statements.calls desc

limit 10;

12.3.5、获取buffer读最多的语句

postgres=# select

pg\_authid.rolname as rolname,

pg\_database.datname as datname,

pg\_stat\_statements.query,

pg\_stat\_statements.shared\_blks\_hit,

pg\_stat\_statements.shared\_blks\_read,

(pg\_stat\_statements.shared\_blks\_hit+pg\_stat\_statements.shared\_blks\_read) as all\_blks,

pg\_stat\_statements.calls,

pg\_stat\_statements.total\_time,

pg\_stat\_statements.total\_time/pg\_stat\_statements.calls

from

pg\_stat\_statements

inner join pg\_authid on pg\_authid.oid=pg\_stat\_statements.userid

inner join pg\_database on pg\_database.oid= pg\_stat\_statements.dbid

order by

pg\_stat\_statements.shared\_blks\_hit+pg\_stat\_statements.shared\_blks\_read desc

limit 10;

## 12.4、插件pg\_trgm使用

12.4.1、pg\_trgm功能介绍及添加方法

前模糊，后模糊，前后模糊，正则匹配都属于文本搜索领域常见的需求。TBase在文本搜索领域除了全文检索，还有TRGM。 对于前模糊和后模糊，TBase与其他数据库一样，可以使用btree来加速。 对于前后模糊和正则匹配，则可以使用TRGM，TRGM是一个非常强的插件，对这类文本搜索场景性能提升非常有效，100万左右的数据量，性能提升有100倍以上。

12.4.2、添加pg\_trgm 插件

create extension pg\_trgm;

12.4.3、测试环境准备

postgres=# create table t\_trgm (id int,trgm text,no\_trgm text) ;

NOTICE: Replica identity is needed for shard table, please add to this table through "alter table" command.

CREATE TABLE

postgres=# \timing

Timing is on.

postgres=# insert into t\_trgm select t,md5(t::text),md5(t::text) from generate\_series(1,1000000) as t;

INSERT 0 1000000

Time: 4239.598 ms (00:04.240)postgres=#

12.4.4、gist索引测试

#### 12.4.4.1、创建索引消耗时间

postgres=# create index t\_trgm\_trgm\_idx on t\_trgm using gist(trgm gist\_trgm\_ops);

CREATE INDEX

Time: 24974.600 ms (00:24.975)

postgres=#

postgres=# vacuum ANALYZE t\_trgm;

VACUUM

Time: 551.989 ms

#### 12.4.4.2、索引占用空间

postgres=# select pg\_size\_pretty(pg\_indexes\_size('t\_trgm'));

pg\_size\_pretty

----------------

178 MB

(1 row)

#### 12.4.4.3、模糊查询测试

* 返回记录数多

postgres=# explain analyze select \* from t\_trgm where trgm ilike '%67%';

QUERY PLAN

-------------------------------------------------------------------------------------------------------------------

Remote Fast Query Execution (cost=0.00..0.00 rows=0 width=0) (actual time=126.483..1172.196 rows=114475 loops=1)

Node/s: dn001, dn002

Planning time: 0.055 ms

Execution time: 1197.320 ms

(4 rows)

postgres=# explain analyze select \* from t\_trgm where no\_trgm ilike '%67%';

QUERY PLAN

----------------------------------------------------------------------------------------------------------------

Remote Fast Query Execution (cost=0.00..0.00 rows=0 width=0) (actual time=6.679..965.383 rows=114475 loops=1)

Node/s: dn001, dn002

Planning time: 0.056 ms

Execution time: 989.840 ms

(4 rows)

使用gist索引开销反而更大

* 返回记录比较少

postgres=# explain analyze select \* from t\_trgm where trgm ilike '%67a5%';

QUERY PLAN

---------------------------------------------------------------------------------------------------------------

Remote Fast Query Execution (cost=0.00..0.00 rows=0 width=0) (actual time=106.522..118.693 rows=481 loops=1)

Node/s: dn001, dn002

Planning time: 0.074 ms

Execution time: 118.831 ms

(4 rows)

postgres=# explain analyze select \* from t\_trgm where no\_trgm ilike '%67a5%';

QUERY PLAN

---------------------------------------------------------------------------------------------------------------

Remote Fast Query Execution (cost=0.00..0.00 rows=0 width=0) (actual time=190.056..502.639 rows=481 loops=1)

Node/s: dn001, dn002

Planning time: 0.063 ms

Execution time: 502.756 ms

(4 rows)

Time: 503.887 ms

过滤返回记录少，使用gist索引提高性能比较明显

#### 12.4.4.4、数据导入时间测试

postgres=# truncate table t\_trgm ;

TRUNCATE TABLE

Time: 78.705 ms

postgres=# insert into t\_trgm select t,md5(t::text),md5(t::text) from generate\_series(1,1000000) as t;

INSERT 0 1000000

Time: 29061.725 ms (00:29.062)

#### 12.4.4.5、pgbench并发查询测试

[tbase@VM\_0\_37\_centos pgbench]$ cat select\_t\_trgm.sql

\set id random(1, 1000000)

select \* from t\_trgm where trgm ilike '%'||md5(:id)||'%';

[tbase@VM\_0\_37\_centos pgbench]$ pgbench -h 127.0.0.1 -p 11008 -d postgres -U tbase -c 4 -j 1 -n -M prepared -T 60 -r -f select\_t\_trgm.sql > select\_t\_trgm.txt 2>&1

[tbase@VM\_0\_37\_centos pgbench]$ tail -20 select\_t\_trgm.txt

client 0 receiving

client 1 receiving

client 2 receiving

client 0 receiving

client 3 receiving

pghost: 127.0.0.1 pgport: 11008 nclients: 4 duration: 60 dbName: postgres

transaction type: select\_t\_trgm.sql

scaling factor: 1

query mode: prepared

number of clients: 4

number of threads: 1

duration: 60 s

number of transactions actually processed: 14150

latency average = 16.965 ms

tps = 235.773228 (including connections establishing)

tps = 235.794290 (excluding connections establishing)

script statistics:

- statement latencies in milliseconds:

0.013 \set id random(1, 1000000)

16.944 select \* from t\_trgm where trgm ilike '%'||md5(:id)||'%';

[tbase@VM\_0\_37\_centos pgbench]$ cat select\_t\_trgm.sql

\set id random(1, 1000000)

select \* from t\_trgm where trgm ilike '%'||substring(md5(:id) from 3 for 6)||'%';

[tbase@VM\_0\_37\_centos pgbench]$ pgbench -h 127.0.0.1 -p 11008 -d postgres -U tbase -c 4 -j 1 -n -M prepared -T 60 -r -f select\_t\_trgm.sql > select\_t\_trgm.txt 2>&1

[tbase@VM\_0\_37\_centos pgbench]$ tail -20 select\_t\_trgm.txt

client 3 receiving

client 2 receiving

client 1 receiving

client 0 receiving

client 3 receiving

pghost: 127.0.0.1 pgport: 11008 nclients: 4 duration: 60 dbName: postgres

transaction type: select\_t\_trgm.sql

scaling factor: 1

query mode: prepared

number of clients: 4

number of threads: 1

duration: 60 s

number of transactions actually processed: 2365

latency average = 101.642 ms

tps = 39.353883 (including connections establishing)

tps = 39.357547 (excluding connections establishing)

script statistics:

- statement latencies in milliseconds:

0.015 \set id random(1, 1000000)

101.528 select \* from t\_trgm where trgm ilike '%'||substring(md5(:id) from 3 for 6)||'%';

#### 12.4.4.6、pgbench并发写入测试

[tbase@VM\_0\_37\_centos pgbench]$ cat insert\_t\_trgm.sql

\set id random(1, 1000000)

insert into t\_trgm values(:id,:id,:id) ;

[tbase@VM\_0\_37\_centos pgbench]$ pgbench -h 127.0.0.1 -p 11008 -d postgres -U tbase -c 4 -j 1 -n -M prepared -T 60 -r -f insert\_t\_trgm.sql > insert\_t\_trgm.txt 2>&1

[tbase@VM\_0\_37\_centos pgbench]$ tail -20 insert\_t\_trgm.txt

client 0 receiving

client 1 receiving

client 3 receiving

client 2 receiving

client 0 receiving

pghost: 127.0.0.1 pgport: 11008 nclients: 4 duration: 60 dbName: postgres

transaction type: insert\_t\_trgm.sql

scaling factor: 1

query mode: prepared

number of clients: 4

number of threads: 1

duration: 60 s

number of transactions actually processed: 58884

latency average = 4.077 ms

tps = 981.188175 (including connections establishing)

tps = 981.285101 (excluding connections establishing)

script statistics:

- statement latencies in milliseconds:

0.011 \set id random(1, 1000000)

4.063 insert into t\_trgm values(:id,:id,:id) ;

12.4.5、gin索引测试

#### 12.4.5.1、创建索引消耗时间

postgres=# drop index t\_trgm\_trgm\_idx;

DROP INDEX

Time: 55.954 ms

postgres=# create index t\_trgm\_trgm\_idx on t\_trgm using gin(trgm gin\_trgm\_ops);

CREATE INDEX

Time: 16648.549 ms (00:16.649)

postgres=#

#### 12.4.5.2、索引占用空间

postgres=# select pg\_size\_pretty(pg\_indexes\_size('t\_trgm'));

pg\_size\_pretty

----------------

74 MB

(1 row)

#### 12.4.5.3、模糊查询测试

* 返回记录数多

postgres=# explain analyze select \* from t\_trgm where trgm ilike '%67%';

QUERY PLAN

----------------------------------------------------------------------------------------------------------------

Remote Fast Query Execution (cost=0.00..0.00 rows=0 width=0) (actual time=3.603..958.097 rows=114475 loops=1)

Node/s: dn001, dn002

Planning time: 0.061 ms

Execution time: 985.647 ms

(4 rows)

Time: 986.631 ms

postgres=# explain analyze select \* from t\_trgm where no\_trgm ilike '%67%';

QUERY PLAN

----------------------------------------------------------------------------------------------------------------

Remote Fast Query Execution (cost=0.00..0.00 rows=0 width=0) (actual time=2.890..947.736 rows=114475 loops=1)

Node/s: dn001, dn002

Planning time: 0.066 ms

Execution time: 973.220 ms

(4 rows)

Time: 974.374 ms

使用gin索引与不使用性能不相上下

* 返回记录比较少

postgres=# explain analyze select \* from t\_trgm where trgm ilike '%67a5%';

QUERY PLAN

-----------------------------------------------------------------------------------------------------------

Remote Fast Query Execution (cost=0.00..0.00 rows=0 width=0) (actual time=3.001..4.178 rows=481 loops=1)

Node/s: dn001, dn002

Planning time: 0.067 ms

Execution time: 4.300 ms

(4 rows)

Time: 5.212 ms

postgres=# explain analyze select \* from t\_trgm where no\_trgm ilike '%67a5%';

QUERY PLAN

---------------------------------------------------------------------------------------------------------------

Remote Fast Query Execution (cost=0.00..0.00 rows=0 width=0) (actual time=174.435..524.049 rows=481 loops=1)

Node/s: dn001, dn002

Planning time: 0.069 ms

Execution time: 524.207 ms

(4 rows)

Time: 525.226 ms

过滤返回记录少，使用gin索引提高性能100倍，效果非常的好

#### 12.4.5.4、数据导入时间测试

postgres=# truncate table t\_trgm ;

TRUNCATE TABLE

Time: 43.750 ms

postgres=# insert into t\_trgm select t,md5(t::text),md5(t::text) from generate\_series(1,1000000) as t;

INSERT 0 1000000

Time: 36371.813 ms (00:36.372)

#### 12.4.5.5、pgbench并发查询测试

[tbase@VM\_0\_37\_centos pgbench]$ cat select\_t\_trgm.sql

\set id random(1, 1000000)

select \* from t\_trgm where trgm ilike '%'||md5(:id)||'%';

[tbase@VM\_0\_37\_centos pgbench]$ pgbench -h 127.0.0.1 -p 11008 -d postgres -U tbase -c 4 -j 1 -n -M prepared -T 60 -r -f select\_t\_trgm.sql > select\_t\_trgm.txt 2>&1

[tbase@VM\_0\_37\_centos pgbench]$ tail -20 select\_t\_trgm.txt

client 2 receiving

client 1 receiving

client 3 receiving

client 0 receiving

client 2 receiving

pghost: 127.0.0.1 pgport: 11008 nclients: 4 duration: 60 dbName: postgres

transaction type: select\_t\_trgm.sql

scaling factor: 1

query mode: prepared

number of clients: 4

number of threads: 1

duration: 60 s

number of transactions actually processed: 13352

latency average = 17.982 ms

tps = 222.445724 (including connections establishing)

tps = 222.468993 (excluding connections establishing)

script statistics:

- statement latencies in milliseconds:

0.012 \set id random(1, 1000000)

17.959 select \* from t\_trgm where trgm ilike '%'||md5(:id)||'%';

[tbase@VM\_0\_37\_centos pgbench]$ cat select\_t\_trgm.sql

\set id random(1, 1000000)

select \* from t\_trgm where trgm ilike '%'||substring(md5(:id) from 3 for 6)||'%';

[tbase@VM\_0\_37\_centos pgbench]$ pgbench -h 127.0.0.1 -p 11008 -d postgres -U tbase -c 4 -j 1 -n -M prepared -T 60 -r -f select\_t\_trgm.sql > select\_t\_trgm.txt 2>&1

[tbase@VM\_0\_37\_centos pgbench]$ tail -20 select\_t\_trgm.txt

client 1 receiving

client 0 receiving

client 3 receiving

client 1 receiving

client 2 receiving

pghost: 127.0.0.1 pgport: 11008 nclients: 4 duration: 60 dbName: postgres

transaction type: select\_t\_trgm.sql

scaling factor: 1

query mode: prepared

number of clients: 4

number of threads: 1

duration: 60 s

number of transactions actually processed: 57280

latency average = 4.190 ms

tps = 954.570838 (including connections establishing)

tps = 954.650960 (excluding connections establishing)

script statistics:

- statement latencies in milliseconds:

0.011 \set id random(1, 1000000)

4.177 select \* from t\_trgm where trgm ilike '%'||substring(md5(:id) from 3 for 6)||'%';

#### 12.4.5.6、pgbench并发写入测试

[tbase@VM\_0\_37\_centos pgbench]$ cat insert\_t\_trgm.sql

\set id random(1, 1000000)

insert into t\_trgm values(:id,:id,:id) ;

[tbase@VM\_0\_37\_centos pgbench]$ pgbench -h 127.0.0.1 -p 11008 -d postgres -U tbase -c 4 -j 1 -n -M prepared -T 60 -r -f insert\_t\_trgm.sql > insert\_t\_trgm.txt 2>&1

[tbase@VM\_0\_37\_centos pgbench]$ tail -20 insert\_t\_trgm.txt

client 0 receiving

client 1 receiving

client 0 receiving

client 2 receiving

client 3 receiving

pghost: 127.0.0.1 pgport: 11008 nclients: 4 duration: 60 dbName: postgres

transaction type: insert\_t\_trgm.sql

scaling factor: 1

query mode: prepared

number of clients: 4

number of threads: 1

duration: 60 s

number of transactions actually processed: 55701

latency average = 4.309 ms

tps = 928.235504 (including connections establishing)

tps = 928.323672 (excluding connections establishing)

script statistics:

- statement latencies in milliseconds:

0.011 \set id random(1, 1000000)

4.296 insert into t\_trgm values(:id,:id,:id) ;

12.4.6、无索引字段测试

#### 12.4.6.1、pgbench并发查询测试

[tbase@VM\_0\_37\_centos pgbench]$ cat select\_t\_trgm.sql

\set id random(1, 1000000)

select \* from t\_trgm where no\_trgm ilike '%'||substring(md5(:id) from 3 for 6)||'%';

[tbase@VM\_0\_37\_centos pgbench]$ pgbench -h 127.0.0.1 -p 11008 -d postgres -U tbase -c 4 -j 1 -n -M prepared -T 60 -r -f select\_t\_trgm.sql > select\_t\_trgm.txt 2>&1

[tbase@VM\_0\_37\_centos pgbench]$ tail -20 select\_t\_trgm.txt

client 3 receiving

client 2 receiving

client 1 receiving

client 3 receiving

client 0 receiving

pghost: 127.0.0.1 pgport: 11008 nclients: 4 duration: 60 dbName: postgres

transaction type: select\_t\_trgm.sql

scaling factor: 1

query mode: prepared

number of clients: 4

number of threads: 1

duration: 60 s

number of transactions actually processed: 212

latency average = 1147.939 ms

tps = 3.484507 (including connections establishing)

tps = 3.484812 (excluding connections establishing)

script statistics:

- statement latencies in milliseconds:

0.016 \set id random(1, 1000000)

1144.441 select \* from t\_trgm where no\_trgm ilike '%'||substring(md5(:id) from 3 for 6)||'%';

[tbase@VM\_0\_37\_centos pgbench]$ cat select\_t\_trgm.sql

\set id random(1, 1000000)

select \* from t\_trgm where no\_trgm ilike '%'||md5(:id)||'%';

[tbase@VM\_0\_37\_centos pgbench]$ pgbench -h 127.0.0.1 -p 11008 -d postgres -U tbase -c 4 -j 1 -n -M prepared -T 60 -r -f select\_t\_trgm.sql > select\_t\_trgm.txt 2>&1

[tbase@VM\_0\_37\_centos pgbench]$ tail -20 select\_t\_trgm.txt

client 3 receiving

client 1 receiving

client 0 receiving

client 2 receiving

client 3 receiving

pghost: 127.0.0.1 pgport: 11008 nclients: 4 duration: 60 dbName: postgres

transaction type: select\_t\_trgm.sql

scaling factor: 1

query mode: prepared

number of clients: 4

number of threads: 1

duration: 60 s

number of transactions actually processed: 175

latency average = 1384.472 ms

tps = 2.889189 (including connections establishing)

tps = 2.889430 (excluding connections establishing)

script statistics:

- statement latencies in milliseconds:

0.015 \set id random(1, 1000000)

1379.714 select \* from t\_trgm where no\_trgm ilike '%'||md5(:id)||'%';

#### 12.4.6.1、pgbench并发写入测试

[tbase@VM\_0\_37\_centos pgbench]$ cat insert\_t\_trgm.sql

\set id random(1, 1000000)

insert into t\_trgm values(:id,:id,:id) ;

[tbase@VM\_0\_37\_centos pgbench]$ pgbench -h 127.0.0.1 -p 11008 -d postgres -U tbase -c 4 -j 1 -n -M prepared -T 60 -r -f insert\_t\_trgm.sql > insert\_t\_trgm.txt 2>&1

[tbase@VM\_0\_37\_centos pgbench]$ tail -20 insert\_t\_trgm.txt

client 0 receiving

client 2 receiving

client 3 receiving

client 0 receiving

client 1 receiving

pghost: 127.0.0.1 pgport: 11008 nclients: 4 duration: 60 dbName: postgres

transaction type: insert\_t\_trgm.sql

scaling factor: 1

query mode: prepared

number of clients: 4

number of threads: 1

duration: 60 s

number of transactions actually processed: 57719

latency average = 4.159 ms

tps = 961.874383 (including connections establishing)

tps = 961.961361 (excluding connections establishing)

script statistics:

- statement latencies in milliseconds:

0.011 \set id random(1, 1000000)

4.145 insert into t\_trgm values(:id,:id,:id) ;

12.4.7、数据对比总结及例外

#### 12.4.7.1、数据对比

|  |  |  |  |
| --- | --- | --- | --- |
| 项目 | GIST | GIN | 无索引 |
| 创建索引消耗时间 | 24.9s | 16.6s |  |
| 索引占用空间 | 178M | 74M |  |
| 数据导入时间测试 | 29s | 36s | 4.2s |
| 极短字符串查询 | 1197.320 ms | 986.631 ms | 989.840 ms |
| 中字符串查询 | 118.831 ms | 4.300 ms | 502.756 ms |
| pgbench长字符串查询，返回单一记录 | 235tps | 222tps | 2.8tps |
| pgbench短字符串查询，返回多条记录 | 39tps | 954tps | 3.4tps |
| pgbench并发写入测试 | 981tps | 928tps | 961tpc |

一般情况下使用gin索引即可

#### 12.4.7.2、pg\_trgm使用限制

* pg\_trgm不支持小于3个字的匹配条件

pg\_trgm不支持小于3个字的匹配条件，pg\_trgm的工作原理是把字符串切成N个3元组，然后对这些3元组做匹配，所以如果作为查询条件的字符串小于3个字符它就罢工了

* 数据库的LC\_CTYPE需要设置为中文区域

可能会发现pg\_trgm不支持中文，中文字符都被截掉了

show\_trgm()值返回空

postgres=# select show\_trgm('腾讯数据库');

show\_trgm

---------------------------------------------------------

{}

(1 row)

--正确返回如下

postgres=# select show\_trgm('腾讯数据库');

show\_trgm

---------------------------------------------------------

{0xa10449,0x077ed5,0x161aeb,0x2d07b4,0x41fde2,0x61dd3b}

(1 row)

这是因为，pg\_trgm调用了系统的isalpha()函数判断字符，而isalpha()依赖于LC\_CTYPE，如果数据库的LC\_CTYPE是C，isalpha()就不能识别中文。所以需要把数据库的LC\_CTYPE设成zh\_CN

# 13、Tbase应用程序样例

## 13.1、java

13.1.1、创建数据表

**import** java.sql.Connection;

**import** java.sql.DriverManager;

**import** java.sql.Statement;

**public** **class** createtable {

**public** static void main( **String** args[] )

{

**Connection** c = **null**;

**Statement** stmt = **null**;

**try** {

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

c = **DriverManager**.getConnection("jdbc:postgresql://127.0.0.1:15432/postgres?currentSchema=public&binaryTransfer=false","tbase", "tbase");

**System**.out.println("Opened database successfully");

stmt = c.createStatement();

**String** sql = "create table tbase(id int,nickname text) distribute by shard(id) to group default\_group" ;

stmt.executeUpdate(sql);

stmt.close();

c.close();

} **catch** ( **Exception** e ) {

**System**.err.println( e.getClass().getName()+": "+ e.getMessage() );

**System**.exit(0);

}

**System**.out.println("Table created successfully");

}

}

### 13.1.2、插入数据

**import** java.sql.Connection;

**import** java.sql.DriverManager;

**import** java.sql.Statement;

**public** **class** insert {

**public** static void main(**String** args[]) {

**Connection** c = **null**;

**Statement** stmt = **null**;

**try** {

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

c = **DriverManager**.getConnection("jdbc:postgresql://127.0.0.1:15432/postgres?currentSchema=public&binaryTransfer=false","tbase", "tbase");

c.setAutoCommit(**false**);

**System**.out.println("Opened database successfully");

stmt = c.createStatement();

**String** sql = "INSERT INTO tbase (id,nickname) "

+ "VALUES (1,'tbase');";

stmt.executeUpdate(sql);

sql = "INSERT INTO tbase (id,nickname) "

+ "VALUES (2, 'pgxz' ),(3,'pgxc');";

stmt.executeUpdate(sql);

stmt.close();

c.commit();

c.close();

} **catch** (**Exception** e) {

**System**.err.println( e.getClass().getName()+": "+ e.getMessage() );

**System**.exit(0);

}

**System**.out.println("Records created successfully");

}

}

### 13.1.4、扩展协议插入数据

**import** java.sql.Connection;

**import** java.sql.DriverManager;

**import** java.sql.\*;

**import** java.util.Random;

**public** **class** insert\_prepared {

**public** static void main(**String** args[]) {

**Connection** c = **null**;

**PreparedStatement** stmt;

**try** {

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

c = **DriverManager**.getConnection("jdbc:postgresql://127.0.0.1:15432/postgres?currentSchema=public&binaryTransfer=false","tbase", "tbase");

c.setAutoCommit(**false**);

**System**.out.println("Opened database successfully");

//插入数据

**String** sql = "INSERT INTO tbase (id,nickname) VALUES (?,?)";

stmt = c.prepareStatement(sql);

stmt.setInt(1, 9999);

stmt.setString(2, "tbase\_prepared");

stmt.executeUpdate();

//插入更新

sql = "INSERT INTO tbase (id,nickname) VALUES (?,?) ON CONFLICT(id) DO UPDATE SET nickname=?";

stmt = c.prepareStatement(sql);

stmt.setInt(1, 9999);

stmt.setString(2, "tbase\_prepared");

stmt.setString(3, "tbase\_prepared\_update");

stmt.executeUpdate();

stmt.close();

c.commit();

c.close();

} **catch** (**Exception** e) {

**System**.err.println( e.getClass().getName()+": "+ e.getMessage() );

**System**.exit(0);

}

**System**.out.println("Records created successfully");

}

}

### 13.1.3、copy from 入库

**import** java.sql.Connection;

**import** java.sql.DriverManager;

**import** java.sql.Statement;

**import** org.postgresql.copy.CopyManager;

**import** org.postgresql.core.BaseConnection;

**import** java.io.\*;

**public** **class** copyfrom {

**public** static void main( **String** args[] )

{

**Connection** c = **null**;

**Statement** stmt = **null**;

**FileInputStream** fs = **null**;

**try** {

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

c = **DriverManager**.getConnection("jdbc:postgresql://127.0.0.1:15432/postgres?currentSchema=public&binaryTransfer=false","tbase", "tbase");

**System**.out.println("Opened database successfully");

CopyManager cm = **new** CopyManager((BaseConnection) c);

fs = **new** **FileInputStream**("/data/tbase/tbase.csv");

**String** sql = "COPY tbase FROM STDIN DELIMITER AS ','";

cm.copyIn(sql, fs);

c.close();

fs.close();

} **catch** ( **Exception** e ) {

**System**.err.println( e.getClass().getName()+": "+ e.getMessage() );

**System**.exit(0);

}

**System**.out.println("Copy data successfully");

}

}

### 13.1.4、copy to出库

**import** java.sql.Connection;

**import** java.sql.DriverManager;

**import** java.sql.Statement;

**import** org.postgresql.copy.CopyManager;

**import** org.postgresql.core.BaseConnection;

**import** java.io.\*;

**public** **class** copyto {

**public** static void main( **String** args[] )

{

**Connection** c = **null**;

**Statement** stmt = **null**;

**FileOutputStream** fs = **null**;

**try** {

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

c = **DriverManager**.getConnection("jdbc:postgresql://127.0.0.1:15432/postgres?currentSchema=public&binaryTransfer=false","tbase", "tbase");

**System**.out.println("Opened database successfully");

CopyManager cm = **new** CopyManager((BaseConnection) c);

fs = **new** **FileOutputStream**("/data/tbase/tbase.csv");

**String** sql = "COPY tbase TO STDOUT DELIMITER AS ','";

cm.copyOut(sql, fs);

c.close();

fs.close();

} **catch** ( **Exception** e ) {

**System**.err.println( e.getClass().getName()+": "+ e.getMessage() );

**System**.exit(0);

}

**System**.out.println("Copy data successfully");

}

}

### 13.1.5、jdbc下载

<https://jdbc.postgresql.org/download.html>

## 13.2、C程序

### 13.2.1、连接服务

#include <stdio.h>

#include <stdlib.h>

#include "libpq-fe.h"

int

main(int argc, char \*\*argv){

**const** char \*conninfo;

PGconn \*conn;

**if** (argc > 1){

        conninfo = argv[1];

}**else**{

        conninfo = "dbname = postgres";

}

conn = PQconnectdb(conninfo);

**if** (PQstatus(conn) != CONNECTION\_OK){

fprintf(stderr, "连接数据库失败: %s",PQerrorMessage(conn));

}**else**{

printf("连接数据库成功！\n");

}

PQfinish(conn);

**return** 0;

}

编译

gcc -c -I /usr/local/install/tbase\_pgxz/include/ conn.c

gcc -o conn conn.o -L /usr/local/install/tbase\_pgxz/lib/ -lpq

运行

./conn "host=172.16.0.3 dbname=postgres port=11000"

连接数据库成功！

./conn "host=172.16.0.3 dbname=postgres port=15432 user=tbase"

连接数据库成功！

### 13.2.2、建立数据表

#include <stdio.h>

#include <stdlib.h>

#include "libpq-fe.h"

int

main(int argc, char \*\*argv){

**const** char \*conninfo;

PGconn \*conn;

PGresult \*res;

**const** char \*sql = "create table tbase(id int,nickname text) distribute by shard(id) to group default\_group";

**if** (argc > 1){

conninfo = argv[1];

}**else**{

conninfo = "dbname = postgres";

}

conn = PQconnectdb(conninfo);

**if** (PQstatus(conn) != CONNECTION\_OK){

fprintf(stderr, "连接数据库失败: %s",PQerrorMessage(conn));

}**else**{

printf("连接数据库成功！\n");

}

res = PQexec(conn,sql);

**if**(PQresultStatus(res) != PGRES\_COMMAND\_OK){

fprintf(stderr, "建立数据表失败: %s",PQresultErrorMessage(res));

}**else**{

printf("建立数据表成功！\n");

}

PQclear(res);

PQfinish(conn);

**return** 0;

}

编译

gcc -c -I /usr/local/install/tbase\_pgxz/include/ createtable.c

gcc -o createtable createtable.o -L /usr/local/install/tbase\_pgxz/lib/ -lpq

运行

./createtable "port=11000 dbname=postgres"

连接数据库成功！

建立数据表成功！

### 13.2.3、插入数据

#include <stdio.h>

#include <stdlib.h>

#include "libpq-fe.h"

int

main(int argc, char \*\*argv){

**const** char \*conninfo;

PGconn \*conn;

PGresult \*res;

**const** char \*sql = "INSERT INTO tbase (id,nickname) values(1,'tbase'),(2,'pgxz')";

**if** (argc > 1){

conninfo = argv[1];

}**else**{

conninfo = "dbname = postgres";

}

conn = PQconnectdb(conninfo);

**if** (PQstatus(conn) != CONNECTION\_OK){

fprintf(stderr, "连接数据库失败: %s",PQerrorMessage(conn));

}**else**{

printf("连接数据库成功！\n");

}

res = PQexec(conn,sql);

**if**(PQresultStatus(res) != PGRES\_COMMAND\_OK){

fprintf(stderr, "插入数据失败: %s",PQresultErrorMessage(res));

}**else**{

printf("插入数据成功！\n");

}

PQclear(res);

PQfinish(conn);

**return** 0;

}

编译

gcc -c -I /usr/local/install/tbase\_pgxz/include/ insert.c

gcc -o insert insert.o -L /usr/local/install/tbase\_pgxz/lib/ -lpq

运行

./insert "dbname=postgres port=15432"

连接数据库成功！

插入数据成功！

### 13.2.4、查询数据

#include <stdio.h>

#include <stdlib.h>

#include "libpq-fe.h"

int

main(int argc, char \*\*argv){

**const** char \*conninfo;

PGconn \*conn;

PGresult \*res;

**const** char \*sql = "select \* from tbase";

**if** (argc > 1){

conninfo = argv[1];

}**else**{

conninfo = "dbname = postgres";

}

conn = PQconnectdb(conninfo);

**if** (PQstatus(conn) != CONNECTION\_OK){

fprintf(stderr, "连接数据库失败: %s",PQerrorMessage(conn));

}**else**{

printf("连接数据库成功！\n");

}

res = PQexec(conn,sql);

**if**(PQresultStatus(res) != PGRES\_TUPLES\_OK){

fprintf(stderr, "插入数据失败: %s",PQresultErrorMessage(res));

}**else**{

printf("查询数据成功！\n");

int rownum = PQntuples(res) ;

int colnum = PQnfields(res);

**for**(int j = 0;j< colnum; ++j){

printf("%s\t",PQfname(res,j));

}

printf("\n");

**for**(int i = 0;i< rownum; ++i){

**for**(int j = 0;j< colnum; ++j){

printf("%s\t",PQgetvalue(res,i,j));

}

printf("\n");

}

}

PQclear(res);

PQfinish(conn);

**return** 0;

}

编译

gcc -std=c99 -c -I /usr/local/install/tbase\_pgxz/include/ select.c

gcc -o select select.o -L /usr/local/install/tbase\_pgxz/lib/ -lpq

运行

./select "dbname=postgres port=15432"

连接数据库成功！

查询数据成功！

id nickname

1 tbase

2 pgxz

### 13.2.5、copy入库

#include <string.h>

#include <stdio.h>

#include <stdlib.h>

#include "libpq-fe.h"

int

main(int argc, char \*\*argv){

**const** char \*conninfo;

PGconn \*conn;

PGresult \*res;

**const** char \*buffer = "1,tbase\n2,pgxz\n3,Tbase牛";

**if** (argc > 1){

conninfo = argv[1];

}**else**{

conninfo = "dbname = postgres";

}

conn = PQconnectdb(conninfo);

**if** (PQstatus(conn) != CONNECTION\_OK){

fprintf(stderr, "连接数据库失败: %s",PQerrorMessage(conn));

}**else**{

printf("连接数据库成功！\n");

}

res=PQexec(conn,"COPY tbase FROM STDIN DELIMITER ',';");

**if**(PQresultStatus(res) != PGRES\_COPY\_IN){

fprintf(stderr, "copy数据出错1: %s",PQresultErrorMessage(res));

}**else**{

int len = strlen(buffer);

**if**(PQputCopyData(conn,buffer,len) == 1){

**if**(PQputCopyEnd(conn,NULL) == 1){

res = PQgetResult(conn);

**if**(PQresultStatus(res) == PGRES\_COMMAND\_OK){

printf("copy数据成功！\n");

}**else**{

fprintf(stderr, "copy数据出错2: %s",PQerrorMessage(conn));

}

}**else**{

fprintf(stderr, "copy数据出错3: %s",PQerrorMessage(conn));

}

}**else**{

fprintf(stderr, "copy数据出错4: %s",PQerrorMessage(conn));

}

}

PQclear(res);

PQfinish(conn);

**return** 0;

}

编译

gcc -c -I /usr/local/install/tbase\_pgxz/include/ copy.c

gcc -o copy copy.o -L /usr/local/install/tbase\_pgxz/lib/ -lpq

执行

./copy "dbname=postgres port=15432"

连接数据库成功！

copy数据成功！

## 13.3、shell程序

#!/bin/sh

**if** [ $# -ne 0 ]

then

echo "usage: $0 exec\_sql"

exit 1

fi

exec\_sql=$1

masters=`psql -h 172.16.0.29 -d postgres -p 15432 -t -c "select string\_agg(node\_host, ' ') from (select \* from pgxc\_node where node\_type = 'D' order by node\_name) t"`

port\_list=`psql -h 172.16.0.29 -d postgres -p 15432 -t -c "select string\_agg(node\_port::text, ' ') from (select \* from pgxc\_node where node\_type = 'D' order by node\_name) t"`

node\_cnt=`psql -h 172.16.0.29 -d postgres -p 15432 -t -c "select count(\*) from pgxc\_node where node\_type = 'D'"`

masters=($masters)

ports=($port\_list)

echo $node\_cnt

flag=0

**for**((i=0;i<$node\_cnt;i++));

do

seq=$(($i+1))

master=${masters[$i]}

port=${ports[$i]}

echo $master

echo $port

psql -h $master -p $port postgres -c "$exec\_sql"

done

## 13.4、python程序

### 13.4.1、安装psycopg2模块

[root@VM\_0\_29\_centos ~]# yum install python-psycopg2

### 13.4.2、连接服务

#coding=utf-8

#!/usr/bin/python

**import** psycopg2

try:

conn = psycopg2.connect(database="postgres", user="tbase", password="", host="172.16.0.29", port="15432")

print "连接数据库成功"

conn.close()

except psycopg2.Error,msg:

print "连接数据库出错，错误详细信息： %s" %(msg.args[0])

运行

[tbase@VM\_0\_29\_centos python]$ python conn.py

连接数据库成功

### 13.4.3、创建数据表

#coding=utf-8

#!/usr/bin/python

**import** psycopg2

try:

conn = psycopg2.connect(database="postgres", user="tbase", password="", host="172.16.0.29", port="15432")

print "连接数据库成功"

cur = conn.cursor()

sql = """

create table tbase

(

id int,

nickname varchar(100)

)distribute by shard(id) to group default\_group

"""

cur.execute(sql)

conn.commit()

print "建立数据表成功"

conn.close()

except psycopg2.Error,msg:

print "TBase Error %s" %(msg.args[0])

运行

[tbase@VM\_0\_29\_centos python]$ python createtable.py

连接数据库成功

建立数据表成功

### 13.4.4、新增数据

#coding=utf-8

#!/usr/bin/python

**import** psycopg2

try:

conn = psycopg2.connect(database="postgres", user="tbase", password="", host="172.16.0.29", port="15432")

print "连接数据库成功"

cur = conn.cursor()

sql = "insert into tbase values(1,'tbase'),(2,'tbase');"

cur.execute(sql)

sql = "insert into tbase values(%s,%s)"

cur.execute(sql,(3,'pg'))

conn.commit()

print "插入数据成功"

conn.close()

except psycopg2.Error,msg:

print "操作数据库出库 %s" %(msg.args[0])

运行

[tbase@VM\_0\_29\_centos python]$ python insert.py

连接数据库成功

插入数据成功

### 13.4.5、查询数据

#coding=utf-8

#!/usr/bin/python

**import** psycopg2

try:

conn = psycopg2.connect(database="postgres", user="tbase", password="", host="172.16.0.29", port="15432")

print "连接数据库成功"

cur = conn.cursor()

sql = "select \* from tbase"

cur.execute(sql)

rows = cur.fetchall()

**for** row in rows:

print "ID = ", row[0]

print "NICKNAME = ", row[1],"\n"

conn.close()

except psycopg2.Error,msg:

print "操作数据库出库 %s" %(msg.args[0])

运行

[tbase@VM\_0\_29\_centos python]$ python select.py

连接数据库成功

ID = 1

NICKNAME = tbase

ID = 2

NICKNAME = pgxz

ID = 3

NICKNAME = pg

### 13.4.6、copy from方法

#coding=utf-8

#!/usr/bin/python

**import** psycopg2

try:

conn = psycopg2.connect(database="postgres", user="tbase", password="", host="172.16.0.29", port="15432")

print "连接数据库成功"

cur = conn.cursor()

filename = "/data/tbase/tbase.txt"

cols = ('id','nickname')

tablename="public.tbase"

cur.copy\_from(file=open(filename),table=tablename,columns=cols,sep=',')

conn.commit()

print "导入数据成功"

conn.close()

except psycopg2.Error,msg:

print "操作数据库出库 %s" %(msg.args[0])

执行

[tbase@VM\_0\_29\_centos python]$ python copy\_from.py

连接数据库成功

导入数据成功

## 13.5、php程序

### 13.5.1、连接服务

<?php

$host="172.16.0.29";

$port="15432";

$dbname="postgres";

$user="tbase" ;

$password="";

//连接数据库

$conn=@pg\_connect("host=$host port=$port dbname=$dbname user=$user password=$password");

**if** (!$conn){

$error\_msg=@pg\_errormessage($conn);

echo "连接数据库出错，详情：".$error\_msg."\n<BR>"; ;

exit;

}**else**{

echo "连接数据库成功"."\n<BR>";

}

//关闭连接

pg\_close($conn);

?>

执行

[root@VM\_0\_47\_centos test]# curl http://127.0.0.1:8080/dbsta/test/conn.php

连接数据库成功

### 13.5.2、创建数据表

<?php

$host="172.16.0.29";

$port="15432";

$dbname="postgres";

$user="tbase" ;

$password="";

//连接数据库

$conn=@pg\_connect("host=$host port=$port dbname=$dbname user=$user password=$password");

**if** (!$conn){

$error\_msg=@pg\_errormessage($conn);

echo "连接数据库出错，详情：".$error\_msg."\n"; ;

exit;

}**else**{

echo "连接数据库成功"."\n";

}

//建立数据表

$sql="create table public.tbase(id integer,nickname varchar(100)) distribute by shard(id) to group default\_group;";

$result = @pg\_exec($conn,$sql) ;

**if** (!$result){

$error\_msg=@pg\_errormessage($conn);

echo "创建数据表出错，详情：".$error\_msg."\n"; ;

exit;

}**else**{

echo "创建数据表成功"."\n";

}

//关闭连接

pg\_close($conn);

?>

执行

[root@VM\_0\_47\_centos test]# curl http://127.0.0.1:8080/dbsta/test/createtable.php

连接数据库成功

创建数据表成功

### 13.5.3、插入数据

<?php

$host="172.16.0.29";

$port="15432";

$dbname="postgres";

$user="tbase" ;

$password="";

//连接数据库

$conn=@pg\_connect("host=$host port=$port dbname=$dbname user=$user password=$password");

**if** (!$conn){

$error\_msg=@pg\_errormessage($conn);

echo "连接数据库出错，详情：".$error\_msg."\n"; ;

exit;

}**else**{

echo "连接数据库成功"."\n";

}

//插入数据

$sql="insert into public.tbase values(1,'tbase'),(2,'pgxz');";

$result = @pg\_exec($conn,$sql) ;

**if** (!$result){

$error\_msg=@pg\_errormessage($conn);

echo "插入数据出错，详情：".$error\_msg."\n";

exit;

}**else**{

echo "插入数据成功"."\n";

}

//关闭连接

pg\_close($conn);

?>

执行

[tbase@VM\_0\_47\_centos test]$ curl http://127.0.0.1:8080/dbsta/test/insert.php

连接数据库成功

插入数据成功

### 13.5.4、查询记录

<?php

$host="172.16.0.29";

$port="15432";

$dbname="postgres";

$user="tbase" ;

$password="";

//连接数据库

$conn=@pg\_connect("host=$host port=$port dbname=$dbname user=$user password=$password");

**if** (!$conn){

$error\_msg=@pg\_errormessage($conn);

echo "连接数据库出错，详情：".$error\_msg."\n"; ;

exit;

}**else**{

echo "连接数据库成功"."\n";

}

//查询数据

$sql="select id,nickname from public.tbase";

$result = @pg\_exec($conn,$sql) ;

**if** (!$result){

$error\_msg=@pg\_errormessage($conn);

echo "查询数据出错，详情：".$error\_msg."\n";

exit;

}**else**{

echo "插入数据成功"."\n";

}

$record\_num = pg\_numrows($result);

echo "返回记录数".$record\_num."\n";

$rec=pg\_fetch\_all($result);

**for**($i=0;$i<$record\_num;$i++){

echo "记录数#".strval($i+1)."\n";

echo "id：".$rec[$i]["id"]."\n";

echo "nickname：".$rec[$i]["nickname"]."\n\n";

}

//关闭连接

pg\_close($conn);

?>

调用方法

[root@VM\_0\_47\_centos ~]# curl http://127.0.0.1:8080/dbsta/test/select.php

连接数据库成功

插入数据成功

返回记录数2

记录数#1

id：1

nickname：tbase

记录数#2

id：2

nickname：pgxz

### 13.5.5、copy from 方法

把一个php数组导入到数据表中

<?php

$host="172.16.0.29";

$port="15432";

$dbname="postgres";

$user="tbase" ;

$password="";

//连接数据库

$conn=@pg\_connect("host=$host port=$port dbname=$dbname user=$user password=$password");

**if** (!$conn){

$error\_msg=@pg\_errormessage($conn);

echo "连接数据库出错，详情：".$error\_msg."\n"; ;

exit;

}**else**{

echo "连接数据库成功"."\n";

}

$row=ARRAY("1,TBase","2,pgxz");

$flag=pg\_copy\_from($conn,"public.tbase",$row,",");

**if** (!$flag){

$error\_msg=@pg\_errormessage($conn);

echo "copy出错，详情：".$error\_msg."\n";

}**else**{

echo "copy成功"."\n";

}

//关闭连接

pg\_close($conn);

?>

调用方法

curl http://127.0.0.1/dbsta/cron/php\_copy\_from.php

连接数据库成功

copy成功

### 13.5.6、copy to 方法

将一个表的记录复制到一个php数据中

<?php

$host="172.16.0.29";

$port="15432";

$dbname="postgres";

$user="tbase" ;

$password="";

//连接数据库

$conn=@pg\_connect("host=$host port=$port dbname=$dbname user=$user password=$password");

**if** (!$conn){

$error\_msg=@pg\_errormessage($conn);

echo "连接数据库出错，详情：".$error\_msg."\n"; ;

exit;

}**else**{

echo "连接数据库成功"."\n";

}

$row=pg\_copy\_to($conn,"public.tbase",",");

**if** (!$row){

$error\_msg=@pg\_errormessage($conn);

echo "copy出错，详情：".$error\_msg."\n";

}**else**{

print\_r($row);

}

//关闭连接

pg\_close($conn);

?>

调用方法

curl http://127.0.0.1/dbsta/cron/php\_copy\_to.php

连接数据库成功

Array

(

[0] => 1,TBase

[1] => 2,pgxz

)

### 13.5.7、入库去重方法

<?php

error\_reporting(E\_ALL && ~E\_NOTICE);

ini\_set('display\_errors', '1');

set\_time\_limit(0);

$host="172.16.0.29";

$port="15432";

$dbname="postgres";

$user="tbase" ;

$password="";

/\*

CREATE TABLE mydata (

mpid text NOT NULL,

datatype text NOT NULL,

datatime timestamp without time zone NOT NULL,

datavalue text,

inputtime timestamp without time zone

)

DISTRIBUTE BY SHARD (mpid);

CREATE unique INDEX mydata\_uidx ON mydata USING btree (mpid, datatype, datatime);

\*/

//连接数据库

$conn=@pg\_connect("host=$host port=$port dbname=$dbname user=$user password=$password");

**if** (!$conn){

$error\_msg=@pg\_errormessage($conn);

echo "连接数据库出错，详情：".$error\_msg."\n"; ;

exit;

}**else**{

echo "连接数据库成功"."\n";

}

//建立临时表

$tmp\_table\_name="mydata\_tmp\_".strval(rand(100000000, 999999999));

$k=0;

do {

$sql="

CREATE TABLE ".$tmp\_table\_name." (

mpid text NOT NULL,

datatype text NOT NULL,

datatime timestamp without time zone NOT NULL,

datavalue text,

inputtime timestamp without time zone

)

WITH OIDS DISTRIBUTE BY SHARD (mpid);

";

$result = @pg\_exec($conn,$sql) ;

**if** ($result){

//建议临时数据表成功,退出

$sql="CREATE INDEX ".$tmp\_table\_name."\_idx ON ".$tmp\_table\_name." USING btree (mpid, datatype, datatime); ";

$result = @pg\_exec($conn,$sql) ;

**if** (!$result){

$error\_msg=@pg\_errormessage($conn);

echo "建立临时表索引失败，详情：".$error\_msg."\n<BR>";

exit;

}

**break**;

}**else**{

$k++;

**if**($k>10){

$error\_msg=@pg\_errormessage($conn);

echo "多次建立数据表失败，详情：".$error\_msg."\n<BR>";

exit;

}

}

}while (true);

$sql="";

$mydata\_sql="

INSERT INTO mydata (mpid, datatype,datatime,datavalue,inputtime) VALUES

";

$mydata\_tmp\_sql="

INSERT INTO ".$tmp\_table\_name." (mpid, datatype,datatime,datavalue,inputtime) VALUES

";

**for** ($i=0;$i<100;$i++){

$insertnum = 250;

**for** ($j=0;$j<$insertnum;$j++){

$sql=$sql."

('".strval(rand(100000000, 999999999))."','10129f14','2018-03-15 02:00:00','004390.44','2018-03-15 11:05:55')

";

**if** (($j+1)!=$insertnum){

$sql=$sql.",";

}

}

$execsql=$mydata\_sql.$sql;

$result = @pg\_exec($conn,$execsql) ;

**if** (!$result){

ECHO "执行失败\n";

//将数据导入到临时表

$execsql=$mydata\_tmp\_sql.$sql;

$result = @pg\_exec($conn,$execsql) ;

**if** (!$result){

$error\_msg=@pg\_errormessage($conn);

echo "数据插入临时表失败，详情：".$error\_msg."\n<BR>";

exit;

}

//删除临时表中重复的数据,这一步最好在应用程序中去重，减少数据的负担

$execsql="DELETE FROM ".$tmp\_table\_name." WHERE oid NOT IN (select min(oid) from ".$tmp\_table\_name." group by mpid, datatype, datatime)";

$result = @pg\_exec($conn,$execsql) ;

**if** (!$result){

$error\_msg=@pg\_errormessage($conn);

echo "删除临时表重复数据失败，详情：".$error\_msg."\n";

exit;

}

$k=0;

do {

//删除重复数据

$execsql="DELETE FROM ".$tmp\_table\_name." USING mydata WHERE mydata.mpid=".$tmp\_table\_name.".mpid AND mydata.datatype=".$tmp\_table\_name.".datatype AND mydata.datatime=".$tmp\_table\_name.".datatime" ;

//将数据导入到正式表

$execsql=$execsql.";INSERT INTO mydata SELECT \* FROM ".$tmp\_table\_name;

$result = @pg\_exec($conn,$execsql) ;

**if** ($result){

//直到操作成功退出

//退出前清理数据

$execsql="truncate table ".$tmp\_table\_name;

$result = @pg\_exec($conn,$execsql) ;

**if** (!$result){

$error\_msg=@pg\_errormessage($conn);

echo "truncate 临时表数据出错，详情：".$error\_msg."\n";

exit;

}

ECHO "内层去重成功\n";

**break**;

}**else**{

$k++;

**if**($k>10){

$error\_msg=@pg\_errormessage($conn);

echo "多次删除重复数据失败，详情：".$error\_msg."\n";

exit;

}

}

} while (true);

}**else**{

ECHO "执行成功\n";

}

$sql="";

}

//退出前删除临时表

$execsql="drop table ".$tmp\_table\_name;

$result = @pg\_exec($conn,$execsql) ;

**if** (!$result){

$error\_msg=@pg\_errormessage($conn);

echo "删除临时表数据出错，详情：".$error\_msg."\n";

exit;

}

//关闭连接

pg\_close($conn); ;

?>

## 13.6、golang程序

### 13.6.1、连接服务

**package** main

**import** (

    "fmt"

    "time"

    "github.com/jackc/pgx"

)

**func** main() {

**var** error\_msg string

    //连接数据库

    conn, err := db\_connect()

**if** err != nil {

        error\_msg = "连接数据库失败，详情：" + err.Error()

        write\_log("Error", error\_msg)

**return**

    }

    //程序运行结束时关闭连接

**defer** conn.Close()

    write\_log("Log", "连接数据库成功")

}

/\*

功能描述：写入日志处理

参数说明：

log\_level -- 日志级别，只能是是Error或Log

error\_msg -- 日志内容

返回值说明：无

\*/

**func** write\_log(log\_level string, error\_msg string) {

    //打印错误信息

    fmt.Println("访问时间：", time.Now().Format("2006-01-02 15:04:05"))

    fmt.Println("日志级别：", log\_level)

    fmt.Println("详细信息：", error\_msg)

}

/\*

功能描述：连接数据库

参数说明：无

返回值说明：

conn \*pgx.Conn -- 连接信息

err error --错误信息

\*/

**func** db\_connect() (conn \*pgx.Conn, err error) {

**var** config pgx.ConnConfig

    config.Host = "127.0.0.1" //数据库主机host或ip

    config.User = "tbase" //连接用户

    config.Password = "pgsql" //用户密码

    config.Database = "postgres" //连接数据库名

    config.Port = 15432 //端口号

    conn, err = pgx.Connect(config)

**return** conn, err

}

[root@VM\_0\_29\_centos tbase]# go run conn.go

访问时间： 2018-04-03 20:40:28

日志级别： Log

详细信息： 连接数据库成功

编译后运行

[root@VM\_0\_29\_centos tbase]# go build conn.go

[root@VM\_0\_29\_centos tbase]# ./conn

访问时间： 2018-04-03 20:40:48

日志级别： Log

详细信息： 连接数据库成功

### 13.6.2、建立数据表

**package** main

**import** (

    "fmt"

    "time"

    "github.com/jackc/pgx"

)

**func** main() {

**var** error\_msg string

**var** sql string

    //连接数据库

    conn, err := db\_connect()

**if** err != nil {

        error\_msg = "连接数据库失败，详情：" + err.Error()

        write\_log("Error", error\_msg)

**return**

    }

    //程序运行结束时关闭连接

**defer** conn.Close()

    write\_log("Log", "连接数据库成功")

    //建立数据表

    sql = "create table public.tbase(id varchar(20),nickname varchar(100)) distribute by shard(id) to group default\_group;"

    \_, err = conn.Exec(sql)

**if** err != nil {

        error\_msg = "创建数据表失败,详情：" + err.Error()

        write\_log("Error", error\_msg)

**return**

    } **else** {

        write\_log("Log", "创建数据表成功")

    }

}

/\*

功能描述：写入日志处理

参数说明：

log\_level -- 日志级别，只能是是Error或Log

error\_msg -- 日志内容

返回值说明：无

\*/

**func** write\_log(log\_level string, error\_msg string) {

    //打印错误信息

    fmt.Println("访问时间：", time.Now().Format("2006-01-02 15:04:05"))

    fmt.Println("日志级别：", log\_level)

    fmt.Println("详细信息：", error\_msg)

}

/\*

功能描述：连接数据库

参数说明：无

返回值说明：

conn \*pgx.Conn -- 连接信息

err error --错误信息

\*/

**func** db\_connect() (conn \*pgx.Conn, err error) {

**var** config pgx.ConnConfig

    config.Host = "127.0.0.1" //数据库主机host或ip

    config.User = "tbase" //连接用户

    config.Password = "pgsql" //用户密码

    config.Database = "postgres" //连接数据库名

    config.Port = 15432 //端口号

    conn, err = pgx.Connect(config)

**return** conn, err

}

[root@VM\_0\_29\_centos tbase]# go run createtable.go

访问时间： 2018-04-03 20:50:24

日志级别： Log

详细信息： 连接数据库成功

访问时间： 2018-04-03 20:50:24

日志级别： Log

详细信息： 创建数据表成功

### 13.6.3、插入数据

**package** main

**import** (

    "fmt"

    "strings"

    "time"

    "github.com/jackc/pgx"

)

**func** main() {

**var** error\_msg string

**var** sql string

**var** nickname string

    //连接数据库

    conn, err := db\_connect()

**if** err != nil {

        error\_msg = "连接数据库失败，详情：" + err.Error()

        write\_log("Error", error\_msg)

**return**

    }

    //程序运行结束时关闭连接

**defer** conn.Close()

    write\_log("Log", "连接数据库成功")

    //插入数据

    sql = "insert into public.tbase values('1','tbase'),('2','pgxz');"

    \_, err = conn.Exec(sql)

**if** err != nil {

        error\_msg = "插入数据失败,详情：" + err.Error()

        write\_log("Error", error\_msg)

**return**

    } **else** {

        write\_log("Log", "插入数据成功")

    }

    //绑定变量插入数据,不需要做防注入处理

    sql = "insert into public.tbase values($1,$2),($1,$3);"

    \_, err = conn.Exec(sql, "3", "postgresql", "postgres")

**if** err != nil {

        error\_msg = "插入数据失败,详情：" + err.Error()

        write\_log("Error", error\_msg)

**return**

    } **else** {

        write\_log("Log", "插入数据成功")

    }

    //拼接sql语句插入数据,需要做防注入处理

    nickname = "TBase is ' good!"

    sql = "insert into public.tbase values('1','" + sql\_data\_encode(nickname) + "')"

    \_, err = conn.Exec(sql)

**if** err != nil {

        error\_msg = "插入数据失败,详情：" + err.Error()

        write\_log("Error", error\_msg)

**return**

    } **else** {

        write\_log("Log", "插入数据成功")

    }

}

/\*

功能描述：sql查询拼接字符串编码

参数说明：

str -- 要编码的字符串

返回值说明：

返回编码过的字符串

\*/

**func** sql\_data\_encode(str string) string {

**return** strings.Replace(str, "'", "''", -1)

}

/\*

功能描述：写入日志处理

参数说明：

log\_level -- 日志级别，只能是是Error或Log

error\_msg -- 日志内容

返回值说明：无

\*/

**func** write\_log(log\_level string, error\_msg string) {

    //打印错误信息

    fmt.Println("访问时间：", time.Now().Format("2006-01-02 15:04:05"))

    fmt.Println("日志级别：", log\_level)

    fmt.Println("详细信息：", error\_msg)

}

/\*

功能描述：连接数据库

参数说明：无

返回值说明：

conn \*pgx.Conn -- 连接信息

err error --错误信息

\*/

**func** db\_connect() (conn \*pgx.Conn, err error) {

**var** config pgx.ConnConfig

    config.Host = "127.0.0.1" //数据库主机host或ip

    config.User = "tbase" //连接用户

    config.Password = "pgsql" //用户密码

    config.Database = "postgres" //连接数据库名

    config.Port = 15432 //端口号

    conn, err = pgx.Connect(config)

**return** conn, err

}

[root@VM\_0\_29\_centos tbase]# go run insert.go

访问时间： 2018-04-03 21:05:51

日志级别： Log

详细信息： 连接数据库成功

访问时间： 2018-04-03 21:05:51

日志级别： Log

详细信息： 插入数据成功

访问时间： 2018-04-03 21:05:51

日志级别： Log

详细信息： 插入数据成功

访问时间： 2018-04-03 21:05:51

日志级别： Log

详细信息： 插入数据成功

### 13.6.4、查询数据

**package** main

**import** (

    "fmt"

    "strings"

    "time"

    "github.com/jackc/pgx"

)

**func** main() {

**var** error\_msg string

**var** sql string

    //连接数据库

    conn, err := db\_connect()

**if** err != nil {

        error\_msg = "连接数据库失败，详情：" + err.Error()

        write\_log("Error", error\_msg)

**return**

    }

    //程序运行结束时关闭连接

**defer** conn.Close()

    write\_log("Log", "连接数据库成功")

    sql = "SELECT id,nickname FROM public.tbase LIMIT 2"

    rows, err := conn.Query(sql)

**if** err != nil {

        error\_msg = "查询数据失败,详情：" + err.Error()

        write\_log("Error", error\_msg)

**return**

    } **else** {

        write\_log("Log", "查询数据成功")

    }

**var** nickname string

**var** id string

**for** rows.Next() {

        err = rows.Scan(&id, &nickname)

**if** err != nil {

            error\_msg = "执行查询失败，详情：" + err.Error()

            write\_log("Error", error\_msg)

**return**

        }

        error\_msg = fmt.Sprintf("id：%s nickname：%s", id, nickname)

        write\_log("Log", error\_msg)

    }

    rows.Close()

    nickname = "tbase"

    sql = "SELECT id,nickname FROM public.tbase WHERE nickname ='" + sql\_data\_encode(nickname) + "' "

    rows, err = conn.Query(sql)

**if** err != nil {

        error\_msg = "查询数据失败,详情：" + err.Error()

        write\_log("Error", error\_msg)

**return**

    } **else** {

        write\_log("Log", "查询数据成功")

    }

**defer** rows.Close()

**for** rows.Next() {

        err = rows.Scan(&id, &nickname)

**if** err != nil {

            error\_msg = "执行查询失败，详情：" + err.Error()

            write\_log("Error", error\_msg)

**return**

        }

        error\_msg = fmt.Sprintf("id：%s nickname：%s", id, nickname)

        write\_log("Log", error\_msg)

    }

}

/\*

功能描述：sql查询拼接字符串编码

参数说明：

str -- 要编码的字符串

返回值说明：

返回编码过的字符串

\*/

**func** sql\_data\_encode(str string) string {

**return** strings.Replace(str, "'", "''", -1)

}

/\*

功能描述：写入日志处理

参数说明：

log\_level -- 日志级别，只能是是Error或Log

error\_msg -- 日志内容

返回值说明：无

\*/

**func** write\_log(log\_level string, error\_msg string) {

    //打印错误信息

    fmt.Println("访问时间：", time.Now().Format("2006-01-02 15:04:05"))

    fmt.Println("日志级别：", log\_level)

    fmt.Println("详细信息：", error\_msg)

}

/\*

功能描述：连接数据库

参数说明：无

返回值说明：

conn \*pgx.Conn -- 连接信息

err error --错误信息

\*/

**func** db\_connect() (conn \*pgx.Conn, err error) {

**var** config pgx.ConnConfig

    config.Host = "127.0.0.1" //数据库主机host或ip

    config.User = "tbase" //连接用户

    config.Password = "pgsql" //用户密码

    config.Database = "postgres" //连接数据库名

    config.Port = 15432 //端口号

    conn, err = pgx.Connect(config)

**return** conn, err

}

[root@VM\_0\_29\_centos tbase]# go run select.go

访问时间： 2018-04-09 10:35:50

日志级别： Log

详细信息： 连接数据库成功

访问时间： 2018-04-09 10:35:50

日志级别： Log

详细信息： 查询数据成功

访问时间： 2018-04-09 10:35:50

日志级别： Log

详细信息： id：2 nickname：tbase

访问时间： 2018-04-09 10:35:50

日志级别： Log

详细信息： id：3 nickname：postgresql

访问时间： 2018-04-09 10:35:50

日志级别： Log

详细信息： 查询数据成功

访问时间： 2018-04-09 10:35:50

日志级别： Log

详细信息： id：1 nickname：tbase

### 13.6.5、copy from 方法

**package** main

**import** (

    "fmt"

    "math/rand"

    "time"

    "github.com/jackc/pgx"

)

**func** main() {

**var** error\_msg string

    //连接数据库

    conn, err := db\_connect()

**if** err != nil {

        error\_msg = "连接数据库失败，详情：" + err.Error()

        write\_log("Error", error\_msg)

**return**

    }

    //程序运行结束时关闭连接

**defer** conn.Close()

    write\_log("Log", "连接数据库成功")

    //构造5000行数据

    inputRows := [][]**interface**{}{}

**var** id string

**var** nickname string

**for** i := 0; i < 5000; i++ {

        id = fmt.Sprintf("%d", rand.Intn(10000))

        nickname = fmt.Sprintf("%d", rand.Intn(10000))

        inputRows = append(inputRows, []**interface**{}{id, nickname})

    }

    copyCount, err := conn.CopyFrom(pgx.Identifier{"tbase"}, []string{"id", "nickname"}, pgx.CopyFromRows(inputRows))

**if** err != nil {

        error\_msg = "执行copyFrom失败,详情：" + err.Error()

        write\_log("Error", error\_msg)

**return**

    }

**if** copyCount != len(inputRows) {

        error\_msg = fmt.Sprintf("执行copyFrom失败，copy行数：%d 返回行数为：%d", len(inputRows), copyCount)

        write\_log("Error", error\_msg)

**return**

    } **else** {

        error\_msg = "Copy 记录成功"

        write\_log("Log", error\_msg)

    }

}

/\*

功能描述：写入日志处理

参数说明：

log\_level -- 日志级别，只能是是Error或Log

error\_msg -- 日志内容

返回值说明：无

\*/

**func** write\_log(log\_level string, error\_msg string) {

    //打印错误信息

    fmt.Println("访问时间：", time.Now().Format("2006-01-02 15:04:05"))

    fmt.Println("日志级别：", log\_level)

    fmt.Println("详细信息：", error\_msg)

}

/\*

功能描述：连接数据库

参数说明：无

返回值说明：

conn \*pgx.Conn -- 连接信息

err error --错误信息

\*/

**func** db\_connect() (conn \*pgx.Conn, err error) {

**var** config pgx.ConnConfig

    config.Host = "127.0.0.1" //数据库主机host或ip

    config.User = "tbase" //连接用户

    config.Password = "pgsql" //用户密码

    config.Database = "postgres" //连接数据库名

    config.Port = 15432 //端口号

    conn, err = pgx.Connect(config)

**return** conn, err

}

[root@VM\_0\_29\_centos tbase]# go run copy\_from.go

访问时间： 2018-04-09 10:36:40

日志级别： Log

详细信息： 连接数据库成功

访问时间： 2018-04-09 10:36:40

日志级别： Log

详细信息： Copy 记录成功

### 13.6.6、go相关资源包

需要git的资源包

<https://github.com/jackc/pgx>

<https://github.com/pkg/errors>