TDSQL PostgreSQL 版

产品文档

文档目录

- DBMS_SQL
- DBMS_STATS
- DBMS_UTILITY
- UTL_FILE
- UTL_RAW

系统视图

- all_arguments
- all_users
- all_tab_columns
- all_col_comments
- all_constraints
- all_synonyms
- all_indexes
- all_ind_columns
- all_cons_columns
- dba_ind_columns
- dba_constraints
- dba_indexes
- dba_synonyms
- dba_tab_columns
- dba_users
- user_indexes
- user_synonyms
- user_cons_columns
- user_tab_columns
- user_ind_columns
- user_constraints
- user_col_comments
- user_users

函数

- 单行函数
  - 数值函数
  - 返回字符值的字符函数
    - 部分返回字符值的字符函数示例
      - regexp_substr
      - nlssort
      - REGEXP_INSTR
      - regexp_replace
      - nchr
      - nls_upper
  - 返回数值的字符函数
    - 部分返回数值的字符函数示例
      - ASCIISTR
      - LENGTHB
      - length
      - instr

产品简介
产品架构

最近更新时间: 2024-06-12 15:06:00

TDSQL PostgreSQL 版架构简介

TDSQL PostgreSQL 版总体架构如图

- 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，DN互为主备节点不能部署在同一台物理机上。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。 TDSQL for PostgreSQL 每个CN和DN节点可以说是一个PostgreSQL的实例，从实现上来讲，TDSQL for PostgreSQL 就是在PG的代码上加了集群的相关功能，从而做成了一个MPP的数据库集群。

SQL执行过程

1. 业务发送请求到Coord节点，Coord节点向GTM请求事务信息；

2. Coord发送SQL语句和事务信息到Datanode；

3. Datanode执行完SQL后返回结果给CN；

4. CN收集DN的结果并汇总会后返回给业务。

产品优势

最近更新时间: 2024-06-12 15:06:00

SQL兼容度高

TDSQL PostgreSQL 版不仅兼容绝大多数的PostgreSQL语法，包括复杂查询、外键、触发器、视图、存储过程等等，满足大部分企业用户的诉求。同时还兼容大部分的Oracle类型、函数。

分布式事务全局一致性

TDSQL PostgreSQL 版引入全局事务管理节点（GTM，Global Transaction Manager）来专门处理分布式事务一致性，通过拥有自主专利的分布式事务一致性技术，包括两阶段提交（Two Phase Commit）以及全局时钟（Global Timestamp）的策略来保证在全分布式环境下的事务一致性。TDSQL PostgreSQL 版单实例的TPCC事务处理能力得到大幅度的提升，而且处理能力会随着实例规模准线性提升。

在线扩容能力

为了迎接业务的快速增长，系统不可避免的需要进行扩容，传统的分布式数据库所采用分库分表模式的实现使得扩容成本高昂，需要对业务进行长时间的中断。TDSQL PostgreSQL 版在新加节点时，只需要把一些shardmap中的shardid映射到新加的节点，并把对应的数据进行搬迁。扩容对业务的中断仅仅是在切换shardmap中映射关系的时刻，时间大大缩短。

HTAP事务及分析双引擎方案

Hybrid Transactional/Analytical Processing，即事务和分析混合处理技术，这个技术要求本来资源诉求矛盾的两种业务类型在同一个数据库中完成处理。传统的数据库因为各方面的限制，偏向于OLTP或OLAP的场景，两者很难兼得。TDSQL PostgreSQL 版 V2经过专门的设计完美的做到了HTAP，同时具备了高效的OLTP能力和海量的OLAP处理能力，降低业务复杂度和业务成本。

多级容灾能力保证

TDSQL PostgreSQL 版在多个维度保证集群的容灾能力。其中采用强同步复制来保证主从数据完全一致，保障主节点故障时数据无丢失；提供基于任意时间点的恢复特性来防止误操作带来的数据丢失。

资源隔离能力-读写多平面

TDSQL PostgreSQL 版作为一款HTAP数据库，天然具备一写多读的能力。一个实例中每个数据节点(DN)有多个备机，同时协调节点(CN)也有多个备机，这样在具备高可靠的同时，可以充分利用备机的计算能力，在备机上执行分析型的业务。即主节点的所组成的主平面提供读写业务(OLTP),而每个备平面提供一种只读业务(OLAP)，从而可以实现业务的读写分离。

卓越的数据安全保障能力

TDSQL PostgreSQL 版提出三权分立的体系，将传统数据库系统DBA的角色分解为三个相互独立的角色，安全管理员，审计管理员，数据管理员；基于此基础提出自己的安全策略，主要细分为三个部分，分别为数据加密，数据脱敏访问，强制访问控制，三者组合提供多个层级的数据安全保障能力。 TDSQL PostgreSQL 版提供两种数据加密方式，一是业务侧加密，业务调用TDSQL PostgreSQL 版内置的加密函数，将加密后的结果写入数据库，正常读取的也是加密后的数据，在应用里面执行解密。二是使用TDSQL PostgreSQL 版的内置加密功能，相对前一种对业务侧来是透明。 TDSQL PostgreSQL 版从多个维度来提供全方位的审计能力，同时TDSQL PostgreSQL 版审计通过旁路检测方式，对数据库运行效率的影响极小。

高效的数据治理能力

TDSQL PostgreSQL 版团队设计了专门的方案来解决数据倾斜问题。为了有效的降低业务的资源消耗成本，TDSQL PostgreSQL 版开发了冷热数据分离的功能，内核原生态支持数据的冷热分离，业务无需感知底层存储介质的不同，对外提供一个统一的数据库视图。

全面的运维能力

OSS系统是TDSQL PostgreSQL 版系统配套使用的平台管理系统，集租户管理、服务器资源管理、项目管理、实例监控运维管理于一体。其中租户管理，服务器资源管理，项目管理是实现多租户管理配套组件。监控运维系统设计的主要目的是监控和维护TDSQL PostgreSQL 版数据库，包括指标实时监控、告警，部分故障自动修复，在线扩容，数据搬迁等功能，与TDSQL PostgreSQL 版数据库组成一套完整的体系，组成一个高效、稳定、可靠的分布式数据库系统。

产品概述

最近更新时间: 2024-06-12 15:06:00

TDSQL PostgreSQL 版产品简介

TDSQL PostgreSQL 版是腾讯云金融专区 自主研发的分布式数据库系统，具备完整的HTAP处理能力。内置的分布式处理逻辑对业务屏蔽底层数据分布细节，提供完整的分布式事务处理能力，同时支持SQL2003，业务可以像使用单机数据库一样来设计自己的业务逻辑，对于已有的业务系统，可以轻松的完成分布式改造，减少对原有系统的侵入性。

TDSQL PostgreSQL 版不但具有分布式数据库具备的在线扩容，在线热升级等功能外，还提供完整的高可用架构保证用户数据库安全，且内置完整的MLS（multiple level security）安全体系，防止用户数据发生泄漏，是目前安全能力最强的数据库产品。

通过TDSQL PostgreSQL 版的OSS运维操作系统，可以对TDSQL PostgreSQL 版进行完整的运维监控操作，大大简化用户的运维操作成本。

目前TDSQL PostgreSQL 版已经服务政务，银行，保险，互联网金融，能源，烟草，医疗等众多行业。

TDSQL PostgreSQL 版技术特点

透明数据分布

TDSQL PostgreSQL 版支持shard表，replication表，业务只需建表时制定分表逻辑，TDSQL PostgreSQL 版自动完成底层的数据分布和分库分表逻辑，底层数据存储细节对业务透明。

完整分布式事务支持

提供完整的分布式事务保证，支持分布式事务ACID，提供分布式一致性读能力。业务可以像使用单机数据库一样来进行事务处理，而不用感知分布式事务处理的底层细节。

完整的SQL兼容性

TDSQL PostgreSQL 版完整兼容SQL2003语法，兼容PostgreSQL语法，同时兼容Oracle语法；在这个基础上，TDSQL PostgreSQL 版内置的高效执行器可以高效的处理各种复杂查询，包括分布式join，聚合，窗口函数，子查询等，业务逻辑无需感知底层数据分布逻辑。用户可以像使用普通数据库一样使用TDSQL PostgreSQL 版数据库。

行列混合HTAP（Hybrid Transactional/Analytical Processing）能力

单集群千万QPS分布式事务处理能力，高效处理OLTP类型的业务；同时支持行存储和列存储，行表和列表可以根据业务需要进行互操作，提升业务处理效率；具备全并行OLAP处理能力，硬件加速进行向量化计算，充分提升OLAP执行效率。

业界最安全的数据库

基于三权分立，构建了业界最高级别的数据安全体系。支持行列矩阵式权限控制，做到对数据的完整权限用户控制;提供特有的透明加密，透明脱敏处理功能，防止文件泄漏引起的数据泄漏；同时提供完整的安全审计能力，具备强大的时候追溯能力。

多租户能力支持

TDSQL PostgreSQL 版提供多种纬度的多租户解决方案，协助用户进行数据共享和资源隔离：物理资源隔离的多租户，用户数据层多租户。

在线扩容缩容能力

TDSQL PostgreSQL 版提供业务无感知的在线扩容缩容能力，后台自动完成扩容缩容任务，业务无须感知扩容缩容逻辑。

PostgreSQL扩展能力

支持PostgreSQL的常用插件，业务可以像使用单机的PostgreSQL一样使用TDSQL PostgreSQL 版。比如fdw，PostGIS，pipelinedb等。同时也提供完整兼容PostgreSQL的数据类型和函数。

多平面读写分离能力

TDSQL PostgreSQL 版提供数据多平面读写分离能力，可以使用集群的备机提供读取服务，在保障分布式一致性的前提下提升系统的资源利用率和系统吞吐量。

冷热数据分离能力

TDSQL PostgreSQL 版针对长尾业务数据量大的问题提出了完整的冷热数据分离处理方案，业务无须感知，TDSQL PostgreSQL 版自动完成业务数据的冷热分离，帮助用户自动完成数据的冷热切分。

内核版本说明

TDSQL PostgreSQL 版

TDSQL PostgreSQL 版提供了三种版本供您选择，包括PostgreSQL兼容版、Oracle兼容版和融合版。

- PostgreSQL兼容版：该版本专为分布式数据库而设计，其内核与 PostgreSQL 完全兼容，保证了无缝的使用体验。
- Oracle 兼容版：该版本既支持创建分布式架构的数据库实例，也支持集中式架构的数据库实例。Oracle 兼容版兼容了通用场景下的绝大部分 Oracle 语法，是您寻求商业数据库替换的理想选择。
- 融合版：这是 TDSQL PG 团队精心打造的 HTAP 国产精品。融合版既支持创建分布式架构的数据库实例，也支持集中式架构的数据库实例。该版本支持 Oracle 和 PostgreSQL 的双模式，采用统一引擎的行列混合存储，能实现同引擎自适应的TP/AP分流。相比其他版本，融合版的性能更为出色。无论是兼容 PostgreSQL 的场景还是兼容 Oracle 的场景，融合版都能提供优秀的服务。

PostgreSQL 版

PostgreSQL 版提供两个版本供您选择，分别是兼容社区的10.5.5版本和 TDSQL PostgreSQL 单机版。其中，TDSQL PostgreSQL 单机版完全兼容社区 PostgreSQL 15.6，确保了无缝的使用体验，让您在使用过程中感到轻松自如。

购买指南
计费概述

最近更新时间: 2024-06-12 15:06:00

计费概述

总费用 = (节点总内存数 × 内存价格 + 节点总磁盘数 × 磁盘价格) × 时长

流量费用目前不计费。

> 说明：
>
> - 节点的总内存数为实例中所有 CN 节点和 DN 节点的内存数量之和，包含所有的备节点。
> - 节点的总磁盘数为实例中所有 CN 节点和 DN 节点的磁盘数量之和，包含所有的备节点。
> - 实例的 GTM 节点当前不收费。

计费示例 【按量计费示例】：在北京地域下，购买1个按量计费 TDSQL PostgreSQL版，其 CN 节点内存为4GB、硬盘为100GB，CN 节点1主1备，3组；DN 节点内存为12GB，硬盘为200GB，DN 节点1主1备，2组，使用时长为96个小时。

则所需支付的费用计算如下：

实例费用 = ((CN 节点总内存数 + DN 节点总内存数) × 内存价格 + (CN 节点总磁盘数 + DN 节点总磁盘数) × 磁盘价格) × 时长 = ((4GB × 2 × 3 + 12GB × 2 × 2) × XX元/GB/小时 + (100GB × 2 × 3 + 200GB × 2 × 2) × XX元/GB/小时) × 96小时 = XX元

购买指引

最近更新时间: 2024-06-12 15:06:00

购买方式

1. 登录 TDSQL PostgreSQL 版购买页，根据实际需求选择各项配置信息，确认无误后，单击【立即购买】。

- 地域：选择您业务需要部署 TDSQL PostgreSQL版 的地域。

> 建议您选择与云服务器同一个地域，不同地域的云产品内网不通，购买后不能更换。

- 网络：TDSQL PostgreSQL版 所属网络。

> 建议您选择与云服务器同一个地域下的同一私有网络，否则无法通过内网连接云服务器和数据库，缺省设置为"Default-VPC（默认）"。

- 安全组：安全组创建与管理请参见 配置安全组。
- 购买数量：每个用户在每个可用区可购买按量计费实例的总数量为30个，可购买的包年包月实例总数量为30个。

2. 支付完成后，返回实例列表，会看到实例显示"发货中"（大概需要3min - 5min中，请耐心等待），待实例状态变为"运行中"，即可对实例进行操作。

查看账单

最近更新时间: 2024-06-12 15:06:00

操作步骤

1. 登录控制台，单击右上角【费用】，进入【计费管理】页面。
2. 在左侧导航栏中，选择【账单管理】>【账单明细】>【资源 ID 账单】。

开发指南
概述

最近更新时间: 2024-06-12 15:06:00

本文档涵盖TDSQL PostgreSQL连接方式，SQL语言开发编写，Oracle兼容特性等内容，目的是指导应用开发。 本文档适用于使用TDSQL PostgreSQL的应用开发人员、数据库开发设计人员、数据库管理员等。 本文档适用于TDSQL PostgreSQL V5.06.4及以上版本。

连接到数据库
使用TSudio
TStudio登录

最近更新时间: 2024-06-12 15:06:00

地址：http://imgcache.finance.cloud.tencent.com:80172.16.0.29:5050/ 用户名：postgres@postgres.com 密码：postgres ⚠实际使用时，应将172.16.0.29替换为安装了TStudio服务的机器ip，一般为center master的ip；用户名密码为部署服务时配置。 可以通过**主界面-添加新的服务器**进行实例连接，对象管理等操作。 可以通**过配置tstudio-首选项**界面，进行工具配置。

通过TStudio连接数据库实例

最近更新时间: 2024-06-12 15:06:00

通过“添加新的服务器”，可以连接一个TDSQL PostgreSQL实例进行对象操作与管理。 名称（必填）：连接实例的自定义名称 服务器组：

连接实例的组别，方便归类管理 注释（选填）：连接实例的描述 主机名称：连接实例的IP 端口：预先配置好的连接端口 维护数据库：登入数据库 用户名：登入数据库用户名 密码：登入数据库密码 SSL模式：SSL安全连接模式，默认即可。 之后即可通过左侧控制台页面，查看实例对象。

TStudio常用操作
表管理

最近更新时间: 2024-06-12 15:06:00

通过控制台创建表

1 ） Servers实例名称登入数据库名模式名 2 ）右键"模式名" - 〉创建 - 〉表 3) 在"创建-表"菜单中，根据提示填写表相关信息，保存即可

创建表。 ）注释：相当于comment on table。 约束页签：可添加主键、外键、检查（ check ）约束、唯一约束、排除约束。 SQL页签：配置完成之后，可以在该页面查看建表语句。

通过控制台创建表索引

1 ）右键新创建的表，菜单中选择创建-索引 2 ）根据页面提示填写创建索引信息，包括索引名称、表空间、注释、访问方法（ 索引类型 ）以及索引列等，保存即可创建索引。

运行脚本

Tstudio也拥有像主流数据库管理工具（ 例如PL/SQL Developer ）一样直接执行SQL命令的功能。 可以通过顶部菜单栏，"工具—查询工具"进入，也可以右键想要执行SQL脚本的数据库对象，选择"查询工具"进入。 点击编辑区的执行按钮或F5，即可执行SQL，下方数据输出

展示执行结果。

运行脚本

最近更新时间: 2024-06-12 15:06:00

Tstudio也拥有像主流数据库管理工具（例如PL/SQL Developer）一样直接执行SQL命令的功能。 可以通过顶部菜单栏，“工具—查询工具”进入，也可以右键想要执行SQL脚本的数据库对象，选择“查询工具”进入。 点击编辑区的执行按钮或F5，即可执行SQL，下方数据输出展示执行结果。

数据导入导出

最近更新时间: 2024-06-12 15:06:00

Tstudio也拥有像主流数据库管理工具（例如PL/SQL Developer）一样通过控制台导入/导出数据的功能。 选择要操作的表，可以通过顶部菜单栏，“工具—导入/导出”进入，也可以右键想要导入/导出的表，选择“导入/导出”进入。 根据页面提示操作，需要选择导入还是导出功能，文件信息包括文件名、格式，要导入/导出的列。 默认导出数据文件存放于目录/usr/local/install/tstudio/storage/postgres/

使用shell
使用shell连接数据库实例

最近更新时间: 2024-06-12 15:06:00

Shell连接TDSQL PostgreSQL实例首先需要连接集群的Center节点。 例：ssh root@XX.XX.XX.XX。XX为center节点的IP。 可以通过四种方式连接实例。 1）使用参数连接 psql -h 172.16.0.29 -p 15432 -U tbase -d postgres -h：host -p：port -U：数据库user -d：database 2）使用conninfo字符串或者一个URI psql postgresql://tbase@172.16.0.29:15432/postgres 3）配置证书连接 psql 'host=172.16.0.29 port=15432 dbname=postgres user=tbase sslmode=verify-ca sslcert=postgresql.crt sslkey=postgresql.key sslrootcert=root.crt' sslcert说明 4）配置环境变量后快捷连接 PGUSER=tbase PGHOST=127.0.0.1 PGDATABASE=postgres PGPORT=15432 export PGHOST PGUSER PGDATABASE PGPORT 环境变量生效后，命令psql即可快速连接实例

常用操作命令

最近更新时间: 2024-06-12 15:06:00

TDSQL PostgreSQL常用的命令如下：

\password	设置密码
\q	退出
\l	列出所有数据库
\c [database_name]	连接其他数据库，无参数显示当前连接数据库以及用户
\d	列出当前数据库的所有表
\d [table_name]	列出某一张表格的结构
\du	列出所有用户
\e	打开文本编辑器
\conninfo	列出当前数据库和连接的信息
\timing on/off	打开/关闭显示sql语句执行的时间
\i [file_name]	执行sql文件命令
! [command]	执行外部命令
\o [file_name]	将执行结果保存至文件

⚠注：？可查看所有psql命令列表及用法 \h 可查看所有sql命令解释及用法。例\h select 可以查看select的用法。

使用ems postgresql manager

最近更新时间: 2024-06-12 15:06:00

EMS PostgreSQL Manager，这是一款商业的postgresql的客户端管理工具，可以用来创建对象，执行sql等。 配置页面配置好Host name，Username，Password后，即可连接。其余操作与主流数据库管理工具类似。

关于DML与事务
DML语法
关于SELECT

最近更新时间: 2024-06-12 15:06:00

select用于从一张表或视图中获取数据。 使用select ...from...命令查询数据 示例说明： 示例： 1）分组查询，排序 查询销售明细表，按dep，product汇总，排序展示

```
create table t_grouping(id int,dep varchar(20),product varchar(20),num int);
insert into t_grouping values(1,'业务1部','手机',90);
insert into t_grouping values(2,'业务1部','电脑',80);
insert into t_grouping values(3,'业务1部','手机',70);
insert into t_grouping values(4,'业务2部','电脑',60);
insert into t_grouping values(5,'业务2部','手机',50);
insert into t_grouping values(6,'业务2部','电脑',60);
insert into t_grouping values(7,'业务3部','手机',70);
insert into t_grouping values(8,'业务3部','电脑',80);
insert into t_grouping values(9,'业务3部','手机',90);
select dep,product,sum(num) from t_grouping group by dep,product order by dep,product;
```

2）（左连接）表连接（类型内连接、左连接、右连接、全连接）

```
select * from tbase left join t_appoint_col on tbase.id=t_appoint_col.id;
```

3）子查询，排序，限制条数

```
select * from tbase order by (select id from tbase order by random() limit 1);
```

4）返回结果集的差值

```
select * from t_except1 except select * from t_except2;
```

5） 返回结果集的交集

```
create table t_intersect1(id int,mc text);
insert into t_intersect1 values(1,'tbase'),(2,'tbase');
create table t_intersect2(id int,mc text);
insert into t_intersect2 values(1,'tbase'),(3,'tbase');
select * from t_intersect1 INTERSECT select * from t_intersect2;
```

6） 合并多个查询结果

```
--不过虑重复的记录
select * from tbase union all select * from t_appoint_col;
--过虑重复的记录select * from tbase union select * from t_appoint_col;
```

7） 查询指定DN上的数据

```
create table t_direct(id int,mc text);
insert into t_direct values(1,'tbase'),(3,'pgxz');
```

```
EXECUTE DIRECT ON (dn001) 'select * from t_direct;';
```

8) 查询记录所在DN

```
select xc_node_id,* from t1;
select t1.xc_node_id,pgxc_node.node_name,t1.* from t1,pgxc_node where t1.xc_node_id=pgxc_node.node_id;
```

SELECT使用场景
with子查询及递归的使用
with 子查询

最近更新时间: 2024-06-12 15:06:00

```
#实验数据
create table t_area(id int,pid int,area varchar);
insert into t_area values(1,null,'广东省');
insert into t_area values(2,1,'深圳市');
insert into t_area values(3,1,'东莞市');
insert into t_area values(4,2,'南山区');
insert into t_area values(5,2,'福田区');
insert into t_area values(6,2,'罗湖区');
insert into t_area values(7,3,'南城区');
insert into t_area values(8,3,'东城区');
```

```
with 子查询
postgres=# with t_gd_city AS (
select * from t_area where pid=1
)
select * FROM t_gd_city;
id | pid | area
----+-----+--------
2 | 1 | 深圳市
3 | 1 | 东莞市
(2 rows)
```

with递归

最近更新时间: 2024-06-12 15:06:00

```
postgres=# WITH RECURSIVE t_city AS
(
SELECT id, area FROM t_area WHERE id = 1
UNION ALL
SELECT
t_area.id,
t_city.area || ' > ' || t_area.area
FROM
t_city
INNER JOIN t_area ON t_area.pid = t_city.id
)
SELECT id,area FROM t_city order by area;
id | area
----+--------------------------
1 | 广东省
3 | 广东省 > 东莞市
8 | 广东省 > 东莞市 > 东城区
7 | 广东省 > 东莞市 > 南城区
2 | 广东省 > 深圳市
5 | 广东省 > 深圳市 > 福田区
6 | 广东省 > 深圳市 > 罗湖区
4 | 广东省 > 深圳市 > 南山区
```

执行non-recursive term，其结果作为recursive term中对t_city的引用，同时将这部分结果放入临时的working table中，重复执行如下步骤，直到working table为空，来看看具体过程：

- 执行

```
SELECT id, area FROM t_area WHERE id = 1
结果集和working table为
1 | 广东省
```

- 执行

```
SELECT
t_area.id,
t_city.area || ' > ' || t_area.area
FROM
t_city
INNER JOIN t_area ON t_area.pid = t_city.id
结果集和working table为
3 | 广东省 > 东莞市
2 | 广东省 > 深圳市
```

- 再次执行recursive query,结果集和working table为：

```
8 | 广东省 > 东莞市 > 东城区
7 | 广东省 > 东莞市 > 南城区
5 | 广东省 > 深圳市 > 福田区
6 | 广东省 > 深圳市 > 罗湖区
4 | 广东省 > 深圳市 > 南山区
```

- 继续执行recursive query，结果集和working table为空
- 结束递归，将执行过程结果合并，返回结果集。

访问函数

最近更新时间: 2024-06-12 15:06:00

```
postgres=# select md5(random()::text);
md5
----------------------------------
3eb6c0c8f8355f0b0f0cad7a8f0f7491
```

数据排序

最近更新时间: 2024-06-12 15:06:00

- 按某一列排序。

```
postgres=# INSERT into tbase (id,nickname) VALUES(1,'hello TDSQL PG');
INSERT 0 1
postgres=# INSERT into tbase (id,nickname) VALUES(2,'TDSQL PG好');
INSERT 0 1
postgres=# INSERT into tbase (id,nickname) VALUES(1,'TDSQL PG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 TDSQL PG
1 | TDSQL PG分布式数据库的时代来了
2 | TDSQL PG好
(3 rows)
```

- 按第一列排序。

```
postgres=# select * from tbase order by 1;
id | nickname
----+----------------------------
1 | hello TDSQL PG
1 | TDSQL PG分布式数据库的时代来了
2 | TDSQL PG好
(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 | TDSQL PG分布式数据库的时代来了
1 | hello TDSQL PG
2 | TDSQL PG好
(3 rows)
```

- 效果与上面的语句一样。

```
postgres=# select * from tbase order by 1,2 desc;
id | nickname
----+----------------------------
1 | TDSQL PG分布式数据库的时代来了
1 | hello TDSQL PG
2 | TDSQL PG好
(3 rows)
```

- 随机排序。

```
postgres=# select * from tbase order by random();
id | nickname
----+-----------------------------
1 | TDSQL PG分布式数据库的时代来了
2 | TDSQL PG好
1 | hello TDSQL PG
(3 rows)
```

- 计算排序。

```
postgres=# select * from tbase order by md5(nickname);
id | nickname
----+-----------------------------
2 | TDSQL PG好
1 | TDSQL PG分布式数据库的时代来了
1 | hello TDSQL PG
(3 rows)
```

- 排序子查询。

```
postgres=# select * from tbase order by (select id from tbase order by random() limit 1);
id | nickname
----+-----------------------------
1 | hello TDSQL PG
2 | TDSQL PG好
1 | TDSQL PG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 TDSQL PG
1 | TDSQL PG分布式数据库的时代来了
2 | TDSQL PG好
(4 rows)
null值记录排在最后
postgres=# select * from tbase order by nickname nulls last;
id | nickname
----+-----------------------------
1 | hello TDSQL PG
1 | TDSQL PG分布式数据库的时代来了
2 | TDSQL PG好
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。

where条件使用

最近更新时间: 2024-06-12 15:06:00

- 单条件查询。

```
postgres=# select * from tbase where id=1;
id | nickname
----+--------------------------
1 | hello TDSQL PG
1 | TDSQL PG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 TDSQL PG
(1 row)
```

- 多条件or。

```
postgres=# select * from tbase where id=2 or nickname like '%h%';
id | nickname
----+-------------
1 | hello TDSQL PG
2 | TDSQL PG好
(2 rows)
```

- ilike不区分大小写匹配。

```
postgres=# create table t_ilike(id int,mc text);
CREATE TABLE
postgres=# insert into t_ilike values(1,'TDSQL PG'),(2,'TDSQL PG');
INSERT 0 2
postgres=# select * from t_ilike where mc ilike '%tb%';
id | mc
----+-------
1 | TDSQL PG
2 | TDSQL PG
(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 TDSQL PG
1 | TDSQL PG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 TDSQL PG
2 | TDSQL PG好
1 | TDSQL PG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);
CREATE TABLE

postgres=# insert into t_exists1 values(1,'TDSQL PG'),(2,'TDSQL PG');
INSERT 0 2

postgres=# create table t_exists2(id int,mc text);
CREATE TABLE

postgres=# insert into t_exists2 values(1,'TDSQL PG'),(1,'TDSQL PG');
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 | TDSQL PG
(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 | TDSQL PG
(1 row)
```

分页查询

最近更新时间: 2024-06-12 15:06:00

- 默认从第一条开始，返回一条记录。

```
postgres=# select * from tbase limit 1;
id | nickname
----+-------------
1 | hello TDSQL PG
(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 TDSQL PG
(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 | TDSQL PG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 | TDSQL PG好
(1 row)
```

合并多个查询结果

最近更新时间: 2024-06-12 15:06:00

- 不过滤重复的记录。

```
postgres=# select * from tbase union all select * from t_appoint_col;
id | nickname
----+-----------------------------
1 | hello TDSQL PG
2 | TDSQL PG好
1 | TDSQL PG分布式数据库的时代来了
1 | hello TDSQL PG
(4 rows)
```

- 过滤重复的记录。

```
postgres=# select * from tbase union select * from t_appoint_col;
id | nickname
----+-----------------------------
1 | TDSQL PG分布式数据库的时代来了
1 | hello TDSQL PG
2 | TDSQL PG好
(3 rows)
```

返回两个结果的交集

最近更新时间: 2024-06-12 15:06:00

```
postgres=# create table t_intersect1(id int,mc text);
CREATE TABLE
postgres=# insert into t_intersect1 values(1,'TDSQL PG'),(2,'TDSQL PG');
INSERT 0 2
postgres=# create table t_intersect2(id int,mc text);
CREATE TABLE
postgres=# insert into t_intersect2 values(1,'TDSQL PG'),(3,'TDSQL PG');
INSERT 0 2
postgres=# select * from t_intersect1 INTERSECT select * from t_intersect2;
id | mc
----+-------
1 | TDSQL PG
(1 row)
```

返回两个结果的差集

最近更新时间: 2024-06-12 15:06:00

```
postgres=# create table t_except1(id int,mc text);
CREATE TABLE
postgres=# insert into t_except1 values(1,'TDSQL PG'),(2,'TDSQL PG');
INSERT 0 2
postgres=# create table t_except2(id int,mc text);
CREATE TABLE
postgres=# insert into t_except2 values(1,'TDSQL PG'),(3,'TDSQL PG');
INSERT 0 2
postgres=# select * from t_except1 except select * from t_except2;
id | mc
----+-------
2 | TDSQL PG
(1 row)
```

any用法

最近更新时间: 2024-06-12 15:06:00

示例1、

```
postgres=# create table t_any(id int,mc text);
CREATE TABLE
postgres=# insert into t_any values(1,'TDSQL PG'),(2,'TDSQL PG');
INSERT 0 2
postgres=# select * from t_any where id>any (select 1 union select 3);
id | mc
----+-------
2 | TDSQL PG
(1 row)
```

只需要大于其中一个值即为真。 示例2、

```
drop table if exists t_any_dml_20220718_1;

create table t_any_dml_20220718_1(v int, w int);

insert into t_any_dml_20220718_1 values(1, 1);

insert into t_any_dml_20220718_1 values(2, 2);

update t_any_dml_20220718_1 set w = 3 where w > any(0, 1);

delete t_any_dml_20220718_1 where w > any(0, 1);

insert into t_any_dml_20220718_1 values(1, 1);

insert into t_any_dml_20220718_1 select 2, 2 from t_any_dml_20220718_1 where w <> any(0, 1);

with cte_any_update as (select v from t_any_dml_20220718_1 where v > any(0, 1)) update t_any_dml_20220718_1 set w =
(select v from cte_any_update order by v limit 1) where w > all(0, 1);

select * from t_any_dml_20220718_1 order by v;

with cte_any_delete as (select v from t_any_dml_20220718_1 where v > any(0, 1)) delete t_any_dml_20220718_1 where w =
any(select v from cte_any_delete order by v limit 1);

select * from t_any_dml_20220718_1 order by v;

insert into t_any_dml_20220718_1 values(1, 1);

with cte_any_insert as (select v, w from t_any_dml_20220718_1 where v > any(0, 1)) insert into t_any_dml_20220718_1 sele
ct v, w from cte_any_insert;

select * from t_any_dml_20220718_1 order by v;
```

all用法

最近更新时间: 2024-06-12 15:06:00

示例1、

```
postgres=# create table t_all(id int,mc text);
CREATE TABLE
postgres=# insert into t_all values(2,'TDSQL PG'),(3,'TDSQL PG');
INSERT 0 2

postgres=# select * from t_all where id>all (select 1 union select 2);
id | mc
----+-------
3 | TDSQL PG
(1 row)
```

需要大于所有值才为真。 示例2、

```
drop table if exists t_all_dml_20220718_1;

create table t_all_dml_20220718_1(v int, w int);

insert into t_all_dml_20220718_1 values(1, 1);

insert into t_all_dml_20220718_1 values(2, 2);

update t_all_dml_20220718_1 set w = 3 where w > all(0, 1);

delete t_all_dml_20220718_1 where w > all(0, 1);

select * from t_all_dml_20220718_1 order by v;

insert into t_all_dml_20220718_1 select 2, 2 from t_all_dml_20220718_1 where w >= all(0, 1);

with cte_all_update as (select v from t_all_dml_20220718_1 where v > all(0, 1)) update t_all_dml_20220718_1 set w = (selec
t v from cte_all_update order by v limit 1) where w > all(0, 1);

select * from t_all_dml_20220718_1 order by v;

with cte_all_delete as (select v from t_all_dml_20220718_1 where v > all(0, 1)) delete t_all_dml_20220718_1 where w = all(s
elect v from cte_all_delete order by v limit 1);

select * from t_all_dml_20220718_1 order by v;

with cte_all_insert as (select v, w from t_all_dml_20220718_1 where v >= all(0, 1)) insert into t_all_dml_20220718_1 select v,
w from cte_all_insert;

select * from t_all_dml_20220718_1 order by v;
```

some用法

最近更新时间: 2024-06-12 15:06:00

```
drop table if exists t_some_dml_20220718_1;

create table t_some_dml_20220718_1(v int, w int);
insert into t_some_dml_20220718_1 values(1, 1);
insert into t_some_dml_20220718_1 values(2, 2);

update t_some_dml_20220718_1 set w = 3 where w > some(0, 1);
delete t_some_dml_20220718_1 where w > some(0, 1);
insert into t_some_dml_20220718_1 values(1, 1);

insert into t_some_dml_20220718_1 select 2, 2 from t_some_dml_20220718_1 where w <> some(0, 1);

with cte_some_update as (select v from t_some_dml_20220718_1 where v > some(0, 1)) update t_some_dml_20220718_1 s
et w = (select v from cte_some_update order by v limit 1) where w > all(0, 1);

select * from t_some_dml_20220718_1 order by v;

with cte_some_delete as (select v from t_some_dml_20220718_1 where v > some(0, 1)) delete t_some_dml_20220718_1 wh
ere w = some(select v from cte_some_delete order by v limit 1);

select * from t_some_dml_20220718_1 order by v;

insert into t_some_dml_20220718_1 values(1, 1);

with cte_some_insert as (select v, w from t_some_dml_20220718_1 where v > some(0, 1)) insert into t_some_dml_20220718
_1 select v, w from cte_some_insert;

select * from t_some_dml_20220718_1 order by v;
```

聚集查询

最近更新时间: 2024-06-12 15:06:00

- 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)
```

多表关联

最近更新时间: 2024-06-12 15:06:00

- 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 TDSQL PG | 1 | hello TDSQL PG
1 | TDSQL PG分布式数据库的时代来了 | 1 | hello TDSQL PG
(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 TDSQL PG | 1 | hello TDSQL PG
2 | TDSQL PG好 | |
1 | TDSQL PG分布式数据库的时代来了 | 1 | hello TDSQL PG
(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 | TDSQL PG分布式数据库的时代来了 | 1 | hello TDSQL PG
1 | hello TDSQL PG | 1 | hello TDSQL PG
| | 5 | Power TDSQL PG
(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 TDSQL PG | 1 | hello TDSQL PG
2 | TDSQL PG好 | |
1 | TDSQL PG分布式数据库的时代来了 | 1 | hello TDSQL PG
| | 5 | Power TDSQL PG
(4 rows)
```

聚合函数并发计算

最近更新时间: 2024-06-12 15:06:00

- 单核计算。

```
postgres=#\timing
Timing ison.
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=#
```

not in中包含了null，结果全为真

最近更新时间: 2024-06-12 15:06:00

```
postgres=# create table t_not_in(id int,mc text);
CREATE TABLE
postgres=# insert into t_not_in values(1,'TDSQL PG'),(2,'pgxz');
INSERT 0 2
postgres=# select * from t_not_in where id not in (3,5);
id | mc
----+-------
1 | TDSQL PG
2 | pgxz
(2 rows)

postgres=# select * from t_not_in where id not in (3,5,null);
id | mc
----+----
(0 rows)
```

只查某个dn的数据

最近更新时间: 2024-06-12 15:06:00

```
postgres=# create table t_direct(id int,mc text);
CREATE TABLE
postgres=# insert into t_direct values(1,'TDSQL PG'),(3,'pgxz');
INSERT 0 2
postgres=# EXECUTE DIRECT ON (dn001) 'select * from t_direct';
id | mc
----+-------
1 | TDSQL PG
(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 | TDSQL PG
3 | pgxz
(2 rows)
```

特殊应用

最近更新时间: 2024-06-12 15:06:00

- 多行变成单行。

```
postgres=# create table t_mulcol_tosimplecol(id int,mc text);
CREATE TABLE

postgres=# insert into t_mulcol_tosimplecol values(1,'TDSQL PG'),(2,'TDSQL PG');
INSERT 0 2

postgres=# select array_to_string(array(select mc from t_mulcol_tosimplecol),',');
array_to_string
-----------------
TDSQL PG,TDSQL PG
(1 row)
```

- 一列变成多行。

```
postgres=# create table t_col_to_mulrow(id int,mc text);
CREATE TABLE

postgres=# insert into t_col_to_mulrow values(1,'TDSQL PG,TDSQL PG');
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
-----------------------
TDSQL PG
TDSQL PG
(2 rows)
```

查询记录所在dn

最近更新时间: 2024-06-12 15:06:00

```
postgres=# select xc_node_id,* from t1;
xc_node_id | f1 | f2
------------+----+----
2142761564 | 1 | 3
2142761564 | 1 | 3
(2 rows)
postgres=# select t1.xc_node_id,pgxc_node.node_name,t1.* from t1,pgxc_node where t1.xc_node_id=pgxc_node.node_id;
xc_node_id | node_name | f1 | f2
------------+-----------+----+----
2142761564 | dn001 | 1 | 3
2142761564 | dn001 | 1 | 3
(2 rows)
postgres=#
```

grouping sets/rollup/cube用法
group by 用法

最近更新时间: 2024-06-12 15:06:00

#销售明细表

```
create table t_grouping(id int,dep varchar(20),product varchar(20),num int);
insert into t_grouping values(1,'业务1部','手机',90);
insert into t_grouping values(2,'业务1部','电脑',80);
insert into t_grouping values(3,'业务1部','手机',70);
insert into t_grouping values(4,'业务2部','电脑',60);
insert into t_grouping values(5,'业务2部','手机',50);
insert into t_grouping values(6,'业务2部','电脑',60);
insert into t_grouping values(7,'业务3部','手机',70);
insert into t_grouping values(8,'业务3部','电脑',80);
insert into t_grouping values(9,'业务3部','手机',90);

postgres=# select dep,product,sum(num) from t_grouping group by dep,product order by dep,product;
dep | product | sum
---------+---------+-----
业务1部 | 电脑 | 80
业务1部 | 手机 | 160
业务2部 | 电脑 | 120
业务2部 | 手机 | 50
业务3部 | 电脑 | 80
业务3部 | 手机 | 160
```

按dep,product两级汇总分数

使用grouping sets

最近更新时间: 2024-06-12 15:06:00

grouping sets说明：GROUPING SETS的每个子列表可以指定零个或多个列或表达式，并且与其直接在GROUPBY子句中的解释方式相同。 一个空的分组集合意味着所有的行都被聚合到一个组中 如按name,class单级分别汇总，再计算一个总分

```
postgres=# select dep,product,sum(num) from t_grouping group by grouping sets((dep),(product),()) order by dep,produc
t;
dep | product | sum
---------+---------+-----
业务1部 | | 240
业务2部 | | 170
业务3部 | | 240
| 电脑 | 280
| 手机 | 370
| | 650
```

使用grouping sets代替group by

```
postgres=# select dep,product,sum(num) from t_grouping group by grouping sets((dep,product)) order by dep,product;
dep | product | sum
---------+---------+-----
业务1部 | 电脑 | 80
业务1部 | 手机 | 160
业务2部 | 电脑 | 120
业务2部 | 手机 | 50
业务3部 | 电脑 | 80
业务3部 | 手机 | 160
```

使用rollup

最近更新时间: 2024-06-12 15:06:00

使用rollup
* rollup((a),(b))等价于grouping sets((a,b),(a),()) *

```
postgres=# select dep,product,sum(num) from t_grouping group by rollup((dep),(product)) order by dep,product;
dep | product | sum
---------+---------+-----
业务1部 | 电脑 | 80
业务1部 | 手机 | 160
业务1部 | | 240
业务2部 | 电脑 | 120
业务2部 | 手机 | 50
业务2部 | | 170
业务3部 | 电脑 | 80
业务3部 | 手机 | 160
业务3部 | | 240
| | 650
```

该功能等价于grouping sets((dep, product),(dep),())

```
postgres=# select dep,product,sum(num) from t_grouping group by grouping sets((dep, product),( dep),()) order by dep,p
roduct;
dep | product | sum
---------+---------+-----
业务1部 | 电脑 | 80
业务1部 | 手机 | 160
业务1部 | | 240
业务2部 | 电脑 | 120
业务2部 | 手机 | 50
业务2部 | | 170
业务3部 | 电脑 | 80
业务3部 | 手机 | 160
业务3部 | | 240
| | 650
```

使用cube

最近更新时间: 2024-06-12 15:06:00

#*cube((a),(b))等价于grouping sets((a,b),(a),(b),()) *

postgres=#select dep,product,sum(num) from t_grouping group by cube((dep),(product))order by dep,product;
dep | product | sum
---------+---------+-----
业务1部 | 电脑 | 80
业务1部 | 手机 | 160
业务1部 | | 240
业务2部 | 电脑 | 120
业务2部 | 手机 | 50
业务2部 | | 170
业务3部 | 电脑 | 80
业务3部 | 手机 | 160
业务3部 | | 240
| 电脑 | 280
| 手机 | 370
| | 650

#该功能等价于groupingsets((name,class),(name),(class),())

postgres=#select dep,product,sum(num) from t_grouping group by grouping sets((dep,product),(dep),(product),())order b
y dep,product;
dep | product | sum
---------+---------+-----
业务1部 | 电脑 | 80
业务1部 | 手机 | 160
业务1部 | | 240
业务2部 | 电脑 | 120
业务2部 | 手机 | 50
业务2部 | | 170
业务3部 | 电脑 | 80
业务3部 | 手机 | 160
业务3部 | | 240
| 电脑 | 280
| 手机 | 370
| | 650

PREPARE预备使用

创建一个预备

最近更新时间: 2024-06-12 15:06:00

创建一个预备

```
postgres=# create table t1(f1 int,f2 int);
CREATE TABLE
postgres=# insert into t1 values(1,1),(2,2);
COPY 2
postgres=# PREPARE usrrptplan (int) AS SELECT * FROM t1 WHERE f1=$1;
PREPARE
postgres=# EXECUTE usrrptplan(1);
f1 | f2
----+----
1 | 1
(1 row)
```

释放一个预备

最近更新时间: 2024-06-12 15:06:00

```
postgres=# DEALLOCATE usrrptplan;
DEALLOCATE
--释放后再执行就会出错
postgres=# EXECUTE usrrptplan(1);
ERROR: prepared statement "usrrptplan" does not exist
```

关于INSERT

最近更新时间: 2024-06-12 15:06:00

insert用于向一张表中插入数据，期望插入的数据可以是一条，多条或者是一个select查询的结果集。 使用INSERT INTO...命令插入数据
示例说明：

1. 使用default关键字，即值为建表时指定的默认值方式

```
insert into tbase(id,nickname) values(default,'TDSQL PG default');
```

2. 子查询插入

```
insert into tbase(id,nickname) values(1,(select relname from pg_class limit 1));
```

3. 返回插入插入数据

```
insert into tbase(nickname) values('TDSQL PG好') returning *;
```

4. insert update使用

```
insert into tbase values(1,'pgxz') ON CONFLICT (id) DO UPDATE SET nc = 'TDSQL PG';
```

INSERT使用场景
插入单条记录

最近更新时间: 2024-06-12 15:06:00

- 指定所有字段。

```
postgres=# insert into tbase(id,nickname) values(1,'hello TDSQL PG');
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('TDSQL PG好');
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),'TDSQL PG好');
INSERT 0 1
```

- 字段顺序可以任意排列。

```
postgres=# insert into tbase(nickname,id) values('TDSQL PG 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,'TDSQL PG 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 TDSQL PG
2 | TDSQL PG好
5 | TDSQL PG swap
3 | TDSQL PG好
4 | TDSQL PG default
(5 rows)
```

插入多数记录

最近更新时间: 2024-06-12 15:06:00

```
postgres=# insert into tbase(id,nickname) values(1,'hello TDSQL PG'),(2,'TDSQL PG好');
INSERT 0 2
postgres=# select * from tbase;
id | nickname
----+-------------
1 | hello TDSQL PG
2 | TDSQL PG好
(2 rows)
```

使用子查询插入数据

最近更新时间: 2024-06-12 15:06:00

```
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)
```

从另外一个表取数据进行批量插入

最近更新时间: 2024-06-12 15:06:00

```
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)
```

大批量的生成数据

最近更新时间: 2024-06-12 15:06:00

```
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)
```

返回插入数据，轻松获取插入记录的serial值

最近更新时间: 2024-06-12 15:06:00

```
postgres=# insert into tbase(nickname) values('TDSQL PG好') returning *;
id | nickname
----+-----------
7 | TDSQL PG好
(1 row)
```

- 指定返回的字段。

```
INSERT 0 1
postgres=# insert into tbase(nickname) values('hello TDSQL PG') returning id;
id
----
8
(1 row)
```

insert..update更新

最近更新时间: 2024-06-12 15:06:00

- 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 = 'TDSQL PG';
INSERT 0 1
postgres=# select * from t;
id | nc
----+-------
1 | TDSQL PG
(1 row)
```

insert all

最近更新时间: 2024-06-12 15:06:00

```
postgres=# create table t5(f1 int,f2 int);create table t6(f1 int,f2 int);
CREATE TABLE
CREATE TABLE
postgres=# insert all into t5 values(f1,f2) into t6 values(f1,f2) select 1 as f1,1 as f2;
INSERT 0 2
postgres=# select * from t5;
f1 | f2
----+----
1 | 1
(1 row)
postgres=# select * from t6;
f1 | f2
----+----
1 | 1
(1 row)
```

关于UPDATE

最近更新时间: 2024-06-12 15:06:00

update用于更新表数据，可以更新一个字段或者多个字段。 使用update...命令更新表 示例： 1）多表关联更新

```
update tbase set nickname ='Good TDSQL PG' from t_appoint_col where t_appoint_col.id=tbase.id;
```

2）返回更新的数据

```
update tbase set nickname = nickname where id = (random()*2)::integer returning *;
```

3）多列匹配更新

```
update tbase set ( nickname , age ) = ( 'TDSQL PG好' , (random()*2)::integer );
```

UPDATE使用场景

单表更新

最近更新时间: 2024-06-12 15:06:00

```
postgres=# update tbase set nickname ='Hello TDSQL PG' where id=1;
UPDATE 1
```

- null条件的表达方法。

```
postgres=# update tbase set nickname = 'Good TDSQL PG' where nickname is null;
UPDATE 1
postgres=# select * from tbase;
id | nickname
----+-------------
2 | TDSQL PG好
1 | Hello TDSQL PG
3 | Good TDSQL PG
(3 rows)
```

多表关联更新

最近更新时间: 2024-06-12 15:06:00

```
postgres=# update tbase set nickname ='Good TDSQL PG' from t_appoint_col where t_appoint_col.id=tbase.id;
UPDATE 1
postgres=# select * from tbase;
id | nickname
----+------------
2 | TDSQL PG好
1 | Good TDSQL PG
(2 rows)
```

返回更新的数据

最近更新时间: 2024-06-12 15:06:00

```
postgres=# update tbase set nickname = nickname where id = (random()*2)::integer returning *;
id | nickname
----+-----------
2 | TDSQL PG好
(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。

多列匹配更新

最近更新时间: 2024-06-12 15:06:00

```
postgres=# update tbase set ( nickname , age ) = ( 'TDSQL PG好' , (random()*2)::integer );
UPDATE 2
postgres=# select * from tbase;
id | nickname | age
----+-----------+-----
1 | TDSQL PG好 | 2
2 | TDSQL PG好 | 0
(2 rows)
```

shard key不准许更新

最近更新时间: 2024-06-12 15:06:00

```
postgres=# update tbase set id=8 where id=1;
ERROR: Distribute column or partition column can't be updated in current version
```

关于DELETE

最近更新时间: 2024-06-12 15:06:00

delete用于删除表数据，可以全部删除或者部分删除。 示例说明： 1）多表关联删除 使用using关键字，支持字段前表名

```
delete from tbase using t_appoint_col where tbase.id=t_appoint_col.id;
```

2）返回删除数据

```
delete from tbase returning *;
```

3) 支持省略from关键字

```
delete tbase where id=3;
```

DELETE使用场景

带条件删除

最近更新时间: 2024-06-12 15:06:00

```
postgres=# select * from tbase;
id | nickname
----+-------------
2 | TDSQL PG好
1 | Hello TDSQL PG
3 |
4 | TDSQL PG 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 | TDSQL PG好
1 | Hello TDSQL PG
(2 rows)
```

多表关联删除数据

最近更新时间: 2024-06-12 15:06:00

```
postgres=# select * from tbase;
id | nickname
----+-------------
2 | TDSQL PG好
1 | Hello TDSQL PG
(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 | TDSQL PG好
(1 row)
```

返回删除数据

最近更新时间: 2024-06-12 15:06:00

```
postgres=# delete from tbase returning *;
id | nickname
----+-----------
2 | TDSQL PG好
(1 row)
```

returning机制大在的降低的应用的复杂度。

删除所有数据

最近更新时间: 2024-06-12 15:06:00

```
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
```

部分DML操作示例
select支持别名不用as修饰

最近更新时间: 2024-06-12 15:06:00

```
postgres=# select * from student as st where st.f1=1;
f1 | f2
----+----
1 | 2
(1 row)
postgres=# select * from student st where st.f1=1;
f1 | f2
----+----
1 | 2
```

update支持别名

最近更新时间: 2024-06-12 15:06:00

```
postgres=# create table student(f1 int,f2 int);
CREATE TABLE
postgres=# insert into student values(1,1);
INSERT 0 1
postgres=# update student st set st.f2=2 where f1=1;
UPDATE 1
postgres=# select * from student;
f1 | f2
----+----
1 | 2
```

关于事务
事务控制

最近更新时间: 2024-06-12 15:06:00

使用下面的命令来控制事务： BEGIN：开始一个事务。 SAVEPOINT：事务保存点。一个事务可以有多个保存点。 COMMIT：事务提交。 ROLLBACK：事务回滚。可回滚整个当前事务，或与savepoint一起使用，回滚到保存点。 事务控制命令只与 INSERT、UPDATE 和 DELETE 一起使用。他们不能在创建表或删除表时使用，因为这些操作在数据库中是自动提交的。 事务可以使用 BEGINTRANSACTION 命令或简单的 BEGIN 命令来启动。此类事务通常会持续执行下去，直到遇到下一个 COMMIT 或 ROLLBACK 命令。不过在数据库关闭或发生错误时，事务处理也会回滚。以下是启动一个事务的简单语法： BEGIN; 或者 BEGINTRANSACTION;

事务提交

最近更新时间: 2024-06-12 15:06:00

事务提交用于保存对数据库的更改命令。 命令如下： COMMIT; 或者COMMIT TRANSACTION; 例：

```
BEGIN;
delete from tbase where id=5;
COMMIT;
```

⚠在COMMIT;执行之前tbase表中id为5的记录会被另一连接进程查询到。

事务回滚

最近更新时间: 2024-06-12 15:06:00

事务回滚用于撤消尚未保存到数据库的命令。 命令语法如下： ROLLBACK; 或者 ROLLBACK to savepoint_name; 例1：

```
BEGIN;
delete from tbase where id in (3,4);
select * from tbase;
ROLLBACK;
select * from tbase;
```

⚠在ROLLBACK;执行之前同一连接进程查询不到id为3和4的数据。回滚执行之后，数据可查询到。 例2:

```
BEGIN;
delete from tbase where id =3;
select * from tbase;
savepoint A;
delete from tbase where id =4;
select * from tbase;
ROLLBACK to A;
COMMIT;
```

⚠在ROLLBACK to A;执行之前同一进程查询不到id为3和4的数据。回滚到保存点之后，id为3数据可查询到。ROLLBACK to savepoint_name;不会结束事务，需使用COMMIT;提交事务。

事务一致性

最近更新时间: 2024-06-12 15:06:00

事务一致性指事务自始至终读取的数据都是一致的。 示例：

```
#session 1
create table t_repeatable_read (id int,mc text);
insert into t_repeatable_read values(1,'TDSQL PG');
begin isolation level repeatable read ;
select * from t_repeatable_read ;
#session 2
insert into t_repeatable_read values(1,'TDSQL PG');
select * from t_repeatable_read;
#session 1
select * from t_repeatable_read ;
```

⚠session1查询到不到session2插入的数据，事务中读一致。

事务使用场景
开始一个事务

最近更新时间: 2024-06-12 15:06:00

```
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
```

提交事务

最近更新时间: 2024-06-12 15:06:00

- 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 TDSQL PG
2 |TDSQL PG好
3 |TDSQL PG好
4 |TDSQL PG default
```

TDSQL PG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TDSQL PG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 TDSQL PG
2 |TDSQL PG好
3 |TDSQL PG好
4 |TDSQL PG default
5 |TDSQL PG 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“已读提交”。

```
postgres=# select * from tbase order by id;
id| nickname
----+---------------
1 |hello TDSQL PG
2 |TDSQL PG好
3 |TDSQL PG好
4 |TDSQL PG default
(4 rows)
```

回滚事务

最近更新时间: 2024-06-12 15:06:00

```
postgres=# begin;
BEGIN
postgres=# delete from tbase where id in (3,4);
DELETE 2
postgres=# select * from tbase;
id | nickname
----+-------------
1 | hello TDSQL PG
2 | TDSQL PG好
(2 rows)
postgres=# rollback;
ROLLBACK
#Rollback后数据又回来了
postgres=# select * from tbase;
id | nickname
----+---------------
1 | hello TDSQL PG
2 | TDSQL PG好
3 | TDSQL PG好
4 | TDSQL PG default
(4 rows)
```

事务读一致性REPEATABLE READ

最近更新时间: 2024-06-12 15:06:00

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);
CREATE TABLE
postgres=# insert into t_repeatable_read values(1,'TDSQL PG');
INSERT 0 1
postgres=# begin isolation level repeatable read ;
BEGIN
postgres=# select * from t_repeatable_read ;
id | mc
----+-------
1 | TDSQL PG
(1 row)

#session2
postgres=# insert into t_repeatable_read values(1,'pgxz');
INSERT 0 1
postgres=# select * from t_repeatable_read;
id | mc
----+-------
1 | TDSQL PG
1 | pgxz
(2 rows)
#session1
postgres=# select * from t_repeatable_read ;
id | mc
----+-------
1 | TDSQL PG
(1 row)
postgres=#
```

行锁在事务中的运用

数据表准备

最近更新时间: 2024-06-12 15:06:00

```
postgres=# create table t_row_lock(id int,mc text,primary key (id));
CREATE TABLE
postgres=#
postgres=# insert into t_row_lock values(1,'TDSQL PG'),(2,'pgxz');
INSERT 0 2
postgres=# select * from t_row_lock;
id | mc
----+-------
1 | TDSQL PG
2 | pgxz
(2 rows)
```

直接update获取

最近更新时间: 2024-06-12 15:06:00

```
#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='TDSQL PG';
UPDATE 1
postgres=#
```

上面session1与session2分别持有mc=pgxz行和mc=TDSQL PG的行锁。

select...for update获取

最近更新时间: 2024-06-12 15:06:00

```
#session1
postgres=# begin;
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='TDSQL PG' for update;
id | mc
----+------
2 | pgxz
(1 row)
```

上面session1与session2分别持有mc=pgxz行和mc=TDSQL PG的行锁。

与mysql获取行级锁的区别

最近更新时间: 2024-06-12 15:06:00

```
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='TDSQL PG' 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 | TDSQL PG |
+----+-------+
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)
```

执行计划
查看执行计划

最近更新时间: 2024-06-12 15:06:00

TDSQL PostgreSQL会为每一个收到的查询产生一个查询计划，可以使用**EXPLAIN**命令察看生成的查询计划。 **EXPLAIN**语法如下：
`EXPLAIN [ ( 参数 [, ...] ) ] SQL语句` `EXPLAIN [ ANALYZE ] [ VERBOSE ] SQL语句` 这里 参数可以是：
*ANALYZE [boolean] * *VERBOSE [boolean] * *COSTS [boolean] * *SETTINGS [boolean] BUFFERS [boolean] * *WAL [boolean] * *TIMING [boolean] * *SUMMARY [boolean] * *FORMAT { TEXT | XML | JSON | YAML }*

参数说明： **ANALYZE** 执行命令并且显示实际的运行时间和其他统计信息。这个参数默认被设置为FALSE。

VERBOSE 显示关于执行计划的附加信息。具体包括计划树中每个结点的输出列列表、模式限定的表和函数名称、在表达式中使用范围表别名标记变量，并输出显示统计信息的每个触发器的名称。这个参数默认被设置为FALSE。

COSTS 每一个计划结点的估计启动和总代价，以及估计的行数和每行的宽度。这个参数默认被设置为TRUE。

SETTINGS 有关配置参数的信息。具体包括影响查询计划的选项，其值与内置默认值不同。此参数默认为FALSE。

BUFFERS 缓冲区使用的信息。具体包括共享块命中、读取、标记为脏和写入的次数、本地块命中、读取、标记为脏和写入的次数、以及临时块读取和写入的次数。一次命中表示避免了一次读取，因为需要的块已经在缓存中找到了。共享块包含着来自于常规表和索引的数据，本地块包含着来自于临时表和索引的数据，而临时块包含着在排序、哈希、物化计划结点和类似情况中使用的短期工作数据。脏块的数量表示被这个查询改变的之前未被修改块的数量，而写入块的数量表示这个后台在查询处理期间从缓存中替换出去的脏块的数量。为一个较高层结点显示的块数包括它的所有子结点所用到的块数。在文本格式中，只会打印非零值。只有当ANALYZE也被启用时，这个参数才能使用。它的默认被设置为FALSE。

WAL 有关WAL记录生成的信息。 具体包括记录数、整页图像数（fpi）和生成的WAL字节数量。 在文本格式中，仅打印非零值。 此参数只能在同时启用ANALYZE时使用。 它默认为FALSE。

TIMING 在输出中包括实际启动时间以及在每个结点中花掉的时间。反复读取系统时钟的负荷在某些系统上会显著地拖慢查询，因此在只需要实际的行计数而不是实际时间时，把这个参数设置为FALSE可能会有用。即便用这个选项关闭结点层的计时，整个语句的运行时间也总是会被度量。只有当ANALYZE也被启用时，这个参数才能使用。它的默认被设置为TRUE。

SUMMARY 在查询计划之后包含摘要信息（例如，总计的时间信息）。当使用ANALYZE 时默认包含摘要信息。不使用ANALYZE时可以使用此选项仅启用摘要信息。 使用EXPLAIN EXECUTE中的计划时间包括从缓存中获取计划所需的时间 以及重新计划所需的时间（如有必要）。

FORMAT 指定输出格式，可以是 TEXT、XML、JSON 或者 YAML。非文本输出包含和文本输出格式相同的信息。这个参数默认被设置为TEXT。

> **说明：**
>
> 为了保证TDSQL PostgreSQL优化器在优化查询时能做出合理的决策，查询中用到所有表的pg_statistic数据应该能保持为最新。通常这个工作会由autovacuum daemon自动完成。但是如果一个表最近发生大量的数据改变，那么需要手动做一次ANALYZE而不是等待 autovacuum 来修改记录。

解读执行计划

最近更新时间: 2024-06-12 15:06:00

使用**EXPLAIN**命令可以查看TDSQL PostgreSQL优化器为每个查询生成的具体执行计划。EXPLAIN的输出是一个节点树。 其中每一行对应一个数据库执行算子， 显示算子的节点类型和优化器为执行这个节点预估的开销。
通过以下示例来解读执行计划

```
---涉及到两个表的DDL
create table t3(f1 int, f2 int) distribute by hash(f1);
create table t4(f1 int, f2 int) distrinute by hash(f1);
```

> **说明：**
>
> distribute by hash 表示对t3，t4数据按f1 哈希值进行分布到不同节点dn上。

查看关联SQL的执行计划 `explain select * from t3, t4 where t3.f1=t4.f2;` 可以得到如下输出结果

```
QUERY PLAN
------------------------------------------------------------------------------
Remote Subquery Scan on all (datanode_1,datanode_2) (cost=120.19..222.56 rows=4556 width=16)
-> Hash Join (cost=120.19..222.56 rows=4556 width=16)
Hash Cond:(t4.f2 = t3.f1)
-> Remote Subquery Scan on all (datanode_1,datanode_2) (cost=100.00..120.53 rows=675 width=8)
Distribute results by H: f2
-> Seq Scan on t4 (cost=0.00..11.75 rows=675 width=8)
-> Hash (cost=11.75..11.75 rows=675 width=8)
-> Seq Scan on t3 (cost=0.00..11.75 rows=675 width=8)
(8rows)
```

这个执行计划可以简化成以下执行计划树：

执行计划树里的每个节点代笔一个执行算子（在"执行计划节点类型"中详述）。每个节点的EXPLAIN输出（ 如下所示）格式为 算子名称（算子开销预估）。 `-> Hash Join(cost=120.19..222.56 rows=4556 width=16)` `Hash Cond:(t4.f2 = t3.f1)` 括号中的数字从左到右依次是:

- 启动代价: 这是该算子在读取第一条元组前的开销预估。比方说索引扫描算子的启动代价就是读取目标表的索引页， 读取到第一个元组的开销。以上示例中的哈希关联(Hash Join)的启动代价是120.19。
- 算子总代价：这个改算子从执行到结束的总代价。以上示例中的哈希关联(Hash Join)的总代价是222.56。
- 算子输出的总行数。 以上示例中的哈希关联(Hash Join)的预估输出行数是4556。
- 算子输出行的行宽（字节数）。 以上示例中的哈希关联(Hash Join)的预估行宽是16字节。

有些算子在EXPLAIN的输出里除了上述的基本通用信息外， 还有算子的一些特定信息。 比方说上面的哈希关联算子的EXAPLIN输出还打印了关联条件。 在SQL执行计划输出结果中， 通常

- 最底层节点是表扫描节点， 扫描节点返回表中的原数据行。 不同的表有不同的扫描节点类型:顺序扫描，索引扫描和位图索引扫描。最底层的扫描节点也可能是也有非表列源，如VALUES子句并设置FROM返回，它们有自己的扫描类型。

- 如果查询需要关联，聚合，排序或其他操作，会在扫描节点之上增加节点执行这些操作。这些操作通常都有多种方法，因此在这些位置也有可能出现不同的执行节点类型。

- 分布式计划中通常会有个特殊的算子（“Remote Subquery Scan”）。这个算子实现了分布式架构的核心数据shuffle的功能。 这个算子有几种不同的形态， 分别对应分布式架构下不同的数据shuffle 功能:

  （1）数据收集形态 - 作用是CN从DN收集数据。

  （2）数据重分布形态 - 作用是CN根据选定的列按照特定的规则把数据重分布到所有的DN。

  数据重分布的规则有:

  - Redistribute by Hash - 把指定的列按照hash redistribute。 这种情况下， 执行计划的输出里会在“Remote Subquery Scan”的那一行的下面一行打印 - “Distribute results by H： 列名”。 如下所示。 其中"Remote Subquery Scan on all" 后面括号里显示的是该算子在哪些数据节点上执行。

    ```
    -> Remote Subquery Scan on all (datanode_1,datanode_2) (cost=100.00..120.53 rows=675 width=8)

    Distribute results by H: f2

    -> Seq Scan on t4 (cost=0.00..11.75 rows=675 width=8)
    ```

  - Redistribute by Shard - 把指定的列按照shard redistribute。 这种情况下， 执行计划的输出里会在“Remote Subquery Scan”的那一行的下面一行打印 - “Distribute results by S： 列名”

  - Broadcast - 把指定的表广播到所有的数据节点上。 这种情况下， 执行计划输出里在“Remote Subquery Scan”的那一行下面没有额外的重分布信息。

- 在分布式架构中， 当SQL可以完全下推到各个DN上计算，中间计算结果不需要重新分布到其他DN上的时候， 执行计划中会有个分布式快速执行算子 - “Remote Fast Query Execution”， 如下示例。在下面这个例子里，两表关联的列都是分布健。这种情况下，各个DN上各自独立执行SQL，CN只需要把DN返回的结果直接返回即可，无需额外的计算。

```
postgres@coord1=# explain select * from t3, t4 where t3.f1=t4.f1;
QUERY PLAN
------------------------------------------------------------------
Remote Fast Query Execution (cost=0.00..0.00 rows=0 width=0)
Node/s:datanode_1, datanode_2
->Merge Join (cost=187.38..330.81 rows=9112 width=16)
Merge Cond:(t3.f1=t4.f1)
-> Sort (cost=93.69..97.07 rows=1350 width=8)
Sort Key: t3.f1
-> Seq Scan on t3 (cost=0.00..23.50 rows=1350 width=8)
-> Sort (cost=93.69..97.07 rows=1350 width=8)
Sort Key: t4.f1
-> Seq Scan on t4 (cost=0.00..23.50 rows=1350 width=8)
(10rows)
```

在了解了这些特点之后，我们会对EXPLAIN 命令的输出有一个整体结构的概念。其实EXPLAIN 输出的就是一个用户可视化的查询计划树，可以告诉我们执行了哪些节点（操作），并且每个节点（操作）的代价预估是什么样的。

执行计划节点类型

最近更新时间: 2024-06-12 15:06:00

执行计划节点分为扫描算子、控制算子、物化算子、链接算子等。

扫描算子

扫描节点负责从底层数据来源抽取数据，数据来源可能是来自文件系统，也可能来自网络（分布式查询）。一般而言扫描节点都位于执行树的叶子节点，作为执行树PlanTree的数据输入来源。 关键特征：输入数据、叶子节点、表达式过滤

算子类型	EXPLAIN显示	含义	出现场景描述
SeqScan	Seq Scan	顺序扫描行存数据	用于扫描行存物理表（没有索引辅助的情况下）
EstoreSeqScan	Estore Seq Scan	扫描列存数据	基础的列存引擎扫描算子， 用于扫描列存物理表（没有索引的情况下）
VtsCacheSeqScan	VtsCache Seq Scan	向量化计算缓存结果集扫描	列存引擎下向量化计算缓存
BitmapIndexScan BitmapHeapScan	Bitmap Index Scan Bitmap Heap Scan	利用Bitmap获取元组	BitmapIndexScan 一次性将满足条件的索引项全部取出，然后交给bitmapHeapScan节点，并在内存中进行排序, 根据取出的索引项访问表数据。
TidScan	Tid Scan	通过Tid获取元组	通过对表的ctid字段进行过滤和查找
SampleScan	Sample Scan	数据抽样	用于数据抽样， SELECT ... FROM table_name TABLESAMPLE sampling_method
IndexScan	Index Scan	索引扫描	利用索引进行快速定位符合查询条件的元组
IndexOnlyScan	Index Only Scan	直接从索引返回元组	与IndexScan的差别是无需再次访问基表
SubqueryScan	Subquery Scan	子查询扫描	已一个子查询的结果集作为当前的输入
FunctionScan	Function Scan	函数扫描	将函数的结果集看成一个结果集， 进行后续的计算， 比如 FROM function_name
ValuesScan	Values Scan	扫描Values列表	对 values（ ）子句的元组集合进行扫描

算子类型	EXPLAIN 显示	含义	出现场景描述
TableFuncScan	Table Function Scan	处理TableFunc相关的扫描	xml table/函数
CteScan	CTE Scan	扫描Common Table Expression	将CTE的输出看成一个集合，进行后续的关系运算， 比如 with子语句定义的CTE子查询
NamedTuplestoreScan	Named Tuplestore Scan	用于某些命名的结果集的扫描	
WorkTableScan	Worktable Scan	扫描中间结果集	扫描查询过程中spillout的结果集
ForeignScan	Foreign Scan	外部表扫描	扫描基于外部数据源的外部表（FDW）
CustomScan	Custom Scan	自定义扫描	

控制算子

控制算子一般不映射代数运算符，通常是为了执行器完成一些特殊的流程引入的算子。 关键特征：用于控制数据流程。

算子类型	EXPLAIN 显示	含义	出现场景描述
Result	Row Result	行存储结果	处理仅需要一次计算的条件表达式或insert中的value子句
VecResult	Vector Result	列存储结果	向量化结果
ModifyTable	取决于具体的操作，可能的显示 - Insert - Update - Delete	Insert/Update/Delete操作的算子	增，删， 改
Append	Append	多个关系集合的追加操作	UNION， UNION-ALL
MergeAppend	Merge Append	多个有序关系集合的追加操作	UNION, 继承表
RecursiveUnion	Recursive Union	执行recursive subquery	with recursive递归子查询
Limit	Limit	控制数据流的返回数据量	处理带limit的语句

物化算子

物化算子一般指算法要求，在做算子逻辑处理的时候，要求把下层的数据进行缓存处理，因为对于下层算子返回的数据量不可提前预知，因此需要在算法上考虑数据无法全部放置到内存的情况。 关键特征：需要扫描所有数据之后才返回。

算子类型	EXPLAIN 显示	含义	使用场景
Materialzie	Materialize	物化	缓存结果集以方便后续重复扫描
Sort	Sort	对下层数据进行排序	Order By, MergeJoin, SortAgg, MergeAppend, 配合Unique去重等
Group	Group	对下层已经排序的数据进行分组	处理group-by分组操作
Agg	取决于优化器选择的聚合策略， 有多种可能的显示- - Aggregate - GroupAggregate - HashAggregate - MixedAggregate	对下层数据进行分组或者聚合；可操作有序或无序数据	count/sum/avg/max/min等聚合函数；distinct子句，union去重， group-by子句
Unique	Unique	对下层数据进行去重操作	Distinct. union去重
Hash	Hash	对下层数据进行缓存，存储到一个哈希表里	作为HashJoin算子的输入， 构造HashTable
SetOp	取决于优化器选择的策略，有多种可能的显示- - SetOp - HashSetOp	对下层数据进行缓存，用于处理集合操作	intersect/intersect-all, except/except-all 等操作
WindowAgg	WindowAgg	窗口函数	包含窗口函数的语句
LockRows	LockRows	处理行级锁	SELECT..FOR SHARE/UPDATE

链接算子

这类算子是为了应对数据库中最常见的关联操作。

算子类型	EXPLAIN 显示	含义	场景描述
Nestloop	Nested Loop	循环嵌套链接操作	Inner, Left-outer, Semi-Join, Anti-Join
MergeJoin	Merge Join	归并链接操作	Inner, left-outer, right-outer, full-outer, semi-join, anti-join
HashJoin	Hash Join	哈希链接操作	Inner, left-outer, right-outer, full-outer, semi-join, anti-join

分布式特有的算子

还有一些分布式架构下特有的算子类型

算子类型	EXPLAIN 显示	含义	场景描述
RemoteSubPlan	Remote Subquery Scan	数据重分布	分布式下需要数据重分布情况
RemoteQuery	Remote Fast Query Execution	SQL下推算子	分布式下SQL可以完全下推到数据节点

分布式执行计划示例

最近更新时间: 2024-06-12 15:06:00

下面是分布式架构下， 不同类型的关联（join)的执行计划的示例。

Local Join

```
--关联的两张表
create table t1 (
c1 integer,
c2 integer,
c3 integer,
c4 integer,
c5 integer
) distribute by shard(c1);

create table t2 (
c1 integer,
c2 integer,
c3 integer,
c4 integer,
c5 integer
) distribute by shard(c1);


tbase=# explain select t1.c3, t2.c3 from t1, t2 where t1.c1=t2.c1;
```

结果为

```
QUERY PLAN
------------------------------------------------------------------------------
Remote Subquery Scan on all (datanode_1, datanode_2, datanode_3) (cost=17.71..85.28 rows=3192 width=8)
-> Hash Join (cost=17.71..85.28 rows=3192 width=8)
Hash Cond: (t1.c1 = t2.c1)
-> Seq Scan on t1 (cost=0.00..10.65 rows=565 width=8)
-> Hash (cost=10.65..10.65 rows=565 width=8)
-> Seq Scan on t2 (cost=0.00..10.65 rows=565 width=8)
```

EXPLAIN对应的执行计划流程

Distributed Join（1）

下面这个例子中只有Join的一端（t2表）需要重新分布

```
tbase=# explain select t1.c3, t2.c3 from t1, t2 where t1.c1=t2.c2;
```

结果为

```
QUERY PLAN
------------------------------------------------------------------------------
Remote Subquery Scan on all (datanode_1, datanode_2, datanode_3) (cost=3285.00..7395.00 rows=100000 width=8)
-> Hash Join (cost=3285.00..7395.00 rows=100000 width=8)
Hash Cond: (t2.c2 = t1.c1)
-> Remote Subquery Scan on all (datanode_1, datanode_2, datanode_3) (cost=100.00..3335.00 rows=100000 width=8)
Distribute results by S: c2
-> Seq Scan on t2 (cost=0.00..1935.00 rows=100000 width=8)
-> Hash (cost=1935.00..1935.00 rows=100000 width=8)
-> Seq Scan on t1 (cost=0.00..1935.00 rows=100000 width=8)
```

EXPLAIN对应的执行流程： 其中t2表的扫描结果会按照c2列值重分布。

Distributed Join（2）

下面这个例子中Join的两个表（t1和t2）都需要重新分布， 因为关联条件都不是分布列。

```
tbase =# explain select t1.c3, t2.c3 from t1, t2 where t1.c2=t2.c2;
```

结果为

```
QUERY PLAN
------------------------------------------------------------------------------
Remote Subquery Scan on all (datanode_1, datanode_2, datanode_3) (cost=4685.00..8795.00 rows=100000 width=8)
-> Hash Join (cost=4685.00..8795.00 rows=100000 width=8)
Hash Cond: (t1.c2 = t2.c2)
-> Remote Subquery Scan on all (datanode_1, datanode_2, datanode_3) (cost=100.00..3335.00 rows=100000 width=8)
Distribute results by S: c2
-> Seq Scan on t1 (cost=0.00..1935.00 rows=100000 width=8)
-> Hash (cost=3335.00..3335.00 rows=100000 width=8)
-> Remote Subquery Scan on all (datanode_1, datanode_2, datanode_3) (cost=100.00..3335.00 rows=100000 width=8)
Distribute results by S: c2
-> Seq Scan on t2 (cost=0.00..1935.00 rows=100000 width=8)
```

EXPLAIN对应的执行流程： 其中t1表和t2表的扫描结果会按照各自c2列值重分布。

Broadcast

下面这个例子中，当表t1是大表， t2是小表的时候， 虽然两张表的关联条件都不是分布列， 优化器可能选择broadcast join, 把小表的扫描结果发送到各个数据节点。

```
tbase =# explain select t1.c3, t2.c3 from t1, t2 where t1.c2=t2.c2;
```

结果为

```
QUERY PLAN
------------------------------------------------------------------------------
Remote Subquery Scan on all (datanode_1, datanode_2, datanode_3) (cost=6635.00..10070.00 rows=100000 width=8)
-> Hash Join (cost=6635.00..10070.00 rows=100000 width=8)
Hash Cond: (t1.c2 = t2.c2)
-> Seq Scan on t1 (cost=0.00..1935.00 rows=100000 width=8)
-> Hash (cost=4135.00..4135.00 rows=200000 width=8)
-> Remote Subquery Scan on all (datanode_1, datanode_2, datanode_3) (cost=100.00..4135.00 rows=200000 width=8)
-> Seq Scan on t2 (cost=0.00..1935.00 rows=100000 width=8)
```

EXPLAIN对应的执行流程：

执行计划分类
顺序扫描

最近更新时间: 2024-06-12 15:06:00

```
postgres=# explain select * from t1;
QUERY PLAN
------------------------------------------------------------------
Remote Fast Query Execution (cost=0.00..0.00 rows=0 width=0)
Node/s: dn001, dn002
-> Seq Scan on t1 (cost=0.00..10694.64 rows=500564 width=37)
(3 rows)
postgres=#
```

index scan扫描

最近更新时间: 2024-06-12 15:06:00

```
postgres=# explain select * from t1 where f1=1;
QUERY PLAN
------------------------------------------------------------------------------
Remote Fast Query Execution (cost=0.00..0.00 rows=0 width=0)
Node/s: dn001
-> Index Scan using tf_f1_idx on t1 (cost=0.42..4.44 rows=1 width=37)
Index Cond: (f1 = 1)
(4 rows)
```

index only scan扫描

最近更新时间: 2024-06-12 15:06:00

```
postgres=# explain select f1 from t1 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 tf_f1_idx on t1 (cost=0.42..4.44 rows=1 width=4)
Index Cond: (f1 = 1)
(4 rows)
```

位图扫描bitmap index scan

最近更新时间: 2024-06-12 15:06:00

```
postgres=# explain select * from t1 where f1 =1 or f1=2;
QUERY PLAN
------------------------------------------------------------------------------------
Remote Fast Query Execution (cost=0.00..0.00 rows=0 width=0)
Node/s: dn001, dn002
-> Bitmap Heap Scan on t1 (cost=4.86..8.85 rows=2 width=37)
Recheck Cond: ((f1 = 1) OR (f1 = 2))
-> BitmapOr (cost=4.86..4.86 rows=2 width=0)
-> Bitmap Index Scan on tf_f1_idx (cost=0.00..2.43 rows=1 width=0)
Index Cond: (f1 = 1)
-> Bitmap Index Scan on tf_f1_idx (cost=0.00..2.43 rows=1 width=0)
Index Cond: (f1 = 2)
(9 rows)
```

join分类
nest loop

最近更新时间: 2024-06-12 15:06:00

```
postgres=# explain select * from t1,t2 where t1.f1=t2.f1 and t2.f2=1;
QUERY PLAN
------------------------------------------------------------------------------
Remote Fast Query Execution (cost=0.00..0.00 rows=0 width=0)
Node/s: dn001, dn002
-> Nested Loop (cost=0.70..8.75 rows=1 width=45)
-> Index Scan using t2_f2_idx on t2 (cost=0.28..4.30 rows=1 width=8)
Index Cond: (f2 = 1)
-> Index Scan using tf_f1_idx on t1 (cost=0.42..4.44 rows=1 width=37)
Index Cond: (f1 = t2.f1)
(7 rows)
postgres=#
```

> **注意：**
>
> 驱动表要小表。

merge join

最近更新时间: 2024-06-12 15:06:00

```
postgres=# explain select * from t1,t2 where t1.f1=t2.f1;
QUERY PLAN
------------------------------------------------------------------------------------------
Remote Fast Query Execution (cost=0.00..0.00 rows=0 width=0)
Node/s: dn001, dn002
-> Merge Join (cost=1.91..380.65 rows=5039 width=45)
Merge Cond: (t1.f1 = t2.f1)
-> Index Scan using tf_f1_idx on t1 (cost=0.42..15956.88 rows=500564 width=37)
-> Index Scan using t2_f1_idx on t2 (cost=0.28..146.87 rows=5039 width=8)
(6 rows)
```

hash join

最近更新时间: 2024-06-12 15:06:00

```
postgres=# explain select * from t3,t4 where t3.f1=t4.f1;
QUERY PLAN
------------------------------------------------------------------------
Remote Fast Query Execution (cost=0.00..0.00 rows=0 width=0)
Node/s: dn001, dn002
-> Hash Join (cost=76.62..209.52 rows=2561 width=16)
Hash Cond: (t4.f1 = t3.f1)
-> Seq Scan on t4 (cost=0.00..88.39 rows=5039 width=8)
-> Hash (cost=44.61..44.61 rows=2561 width=8)
-> Seq Scan on t3 (cost=0.00..44.61 rows=2561 width=8)
(7 rows)
```

数据重分布

最近更新时间: 2024-06-12 15:06:00

```
postgres=# explain select * from t3,t4 where t3.f1=t4.f2 and t4.f1=1;
QUERY PLAN
------------------------------------------------------------------------------------------------
Remote Subquery Scan on all (dn001,dn002) (cost=200.53..253.91 rows=1 width=16)
-> Hash Join (cost=200.53..253.91 rows=1 width=16)
Hash Cond: (t3.f1 = t4.f2)
-> Seq Scan on t3 (cost=0.00..44.00 rows=2500 width=8)
-> Hash (cost=200.51..200.51 rows=1 width=8)
-> Remote Subquery Scan on all (dn001) (cost=100.00..200.51 rows=1 width=8)
Distribute results by S: f2
-> Seq Scan on t4 (cost=0.00..100.50 rows=1 width=8)
Filter: (f1 = 1)
(9 rows)
postgres=# explain select * from t3,t4 where t3.f2=t4.f2 and t4.f1=1;
QUERY PLAN
------------------------------------------------------------------------------------------------
Remote Subquery Scan on all (dn001,dn002) (cost=300.53..386.41 rows=1 width=16)
-> Hash Join (cost=300.53..386.41 rows=1 width=16)
Hash Cond: (t3.f2 = t4.f2)
-> Remote Subquery Scan on all (dn001,dn002) (cost=100.00..176.50 rows=2500 width=8)
Distribute results by S: f2
-> Seq Scan on t3 (cost=0.00..44.00 rows=2500 width=8)
-> Hash (cost=200.51..200.51 rows=1 width=8)
-> Remote Subquery Scan on all (dn001) (cost=100.00..200.51 rows=1 width=8)
Distribute results by S: f2
-> Seq Scan on t4 (cost=0.00..100.50 rows=1 width=8)
Filter: (f1 = 1)
(11 rows)
```

并行扫描

最近更新时间: 2024-06-12 15:06:00

```
postgres=# set max_parallel_workers_per_gather to 2;
postgres=# explain select count(1) from t1;
QUERY PLAN
------------------------------------------------------------------------------------------------
Parallel Finalize Aggregate (cost=9387.28..9387.29 rows=1 width=8)
-> Parallel Remote Subquery Scan on all (dn001,dn002) (cost=9387.17..9387.28 rows=1 width=0)
-> Gather (cost=9287.17..9287.28 rows=1 width=8)
Workers Planned: 2
-> Partial Aggregate (cost=8287.17..8287.18 rows=1 width=8)
-> Parallel Seq Scan on t1 (cost=0.00..7766.33 rows=208333 width=0)
(6 rows)
```

创建和管理数据库对象（DDL）
创建GROUP
创建数据表默认的default group

最近更新时间: 2024-06-12 15:06:00

TDSQL PG作为做为分布式数据库系统，使用前必需配置实例的默认存储组（也称group），推荐一个实例只创建一个group，该存储组取名为default_group，并且指定该存储组为默认存储组。跨group数据访问会有一些限制，非特殊情况不允许一个实例创建多个group。group中需要添加DN节点，每个group会均匀分配总共4096个shardkey。

下面演示如何创建一个default group：

1. 切换为tbase用户 su tbase。

2. 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
```

3. 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
-----------+-----------+-----------+-------------+----------------+------------------+-------------+-------------------
dn001 | D | 23001 | 172.16.0.29 | f | f | 1485981022 | tbase_cluster
dn002 | D | 23002 | 172.16.0.47| f | f | -1300059100 | tbase_cluster
(2 rows)
```

4. 建立数据表默认使用的group。

```
postgres=#create default node group default_group with(dn001, dn002);
```

为default group创建shardmap

最近更新时间: 2024-06-12 15:06:00

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TDSQL PG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TDSQL PG集群了。

更多group的使用方法
创建扩展group

最近更新时间: 2024-06-12 15:06:00

- 建立group。

```
postgres=# create node group ext_group with(dn003,dn004);
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=#
```

删除group

最近更新时间: 2024-06-12 15:06:00

```
postgres=# drop sharding in group ext_group;
DROP SHARDING GROUP
postgres=# drop node group ext_group ;
DROP NODE GROUP
```

查询集群中group的数量

最近更新时间: 2024-06-12 15:06:00

```
postgres=# select * from pgxc_group;
group_name | default_group | group_members
---------------+---------------+---------------
default_group | 1 | 16386 16387
ext_group | 0 | 16388 16389
(2 rows)
```

创建和管理模式
模式管理
创建模式

最近更新时间: 2024-06-12 15:06:00

- 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
```

- 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)
```

修改模式属性

最近更新时间: 2024-06-12 15:06:00

- 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
```

删除模式

最近更新时间: 2024-06-12 15:06:00

```
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
```

配置用户访问模式权限

最近更新时间: 2024-06-12 15:06:00

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);

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)
```

配置访问模式的顺序

最近更新时间: 2024-06-12 15:06:00

TDSQL PG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：

1. 当前连接用户。

```
postgres=# select current_user;
current_user
--------------
tbase
(1 row)
```

2. 总共三个模式。

```
postgres=# \dn
List of schemas
Name | Owner
--------------+-------
public | tbase
tbase | tbase
tbase_schema | tbase
(3 rows)
```

3. 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)
```

4. 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);
CREATE TABLE
postgres=# \dt
List of relations
Schema | Name | Type | Owner
--------+------+-------+-------
tbase | t | table | tbase
(1 row)
```

5. 指定表位于某个模式下，不同模式下表名可以相同。

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

6. 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。

创建和管理表
创建表

最近更新时间: 2024-06-12 15:06:00

使用命令create table创建表 1 ）不用指定shard key建表，系统默认使用第一个字段做为表的shard key

```
create table t_first_col_share(id serial not null,nickname text);
```

2 ）指定shard key建表

```
create table t_appoint_col(id serial not null,nickname text) distribute by shard(nickname);
```

⚠分布键选择原则

- 分布键只能选择一个字段
- 如果有主键，则选择主键做分布键
- 如果主键是复合字段组合，则选择字段值选择性多的字段做分布键
- 也可以把复合字段拼接成一个新的字段来做分布键
- 没有主键的可以使用UUID来做分布键
- 让数据尽可能的分布得足够散

3 ）表不存在时才创建

```
create table IF NOT EXISTS t(id int,mc text);
```

数据类型
数字类型

最近更新时间: 2024-06-12 15:06:00

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

字符类型

最近更新时间: 2024-06-12 15:06:00

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	1GB

⚠对于类型CHAR来说，如果不指定n的值，n就会默认为1。 如果要存储的字符串长度小于n，那么类型CHAR的值将会不用空格补齐，与Oracle略有不同。

二进制数据类型

最近更新时间: 2024-06-12 15:06:00

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

日期类型

最近更新时间: 2024-06-12 15:06:00

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（无时区）	0:00:00	24:00:00	1微秒 / 14位
time [(*p*)] [with time zone]	12字节	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

布尔类型

最近更新时间: 2024-06-12 15:06:00

名字	存储字节	描述
boolean	1字节	状态为真或假

异构数据库类型对照表
与ORACLE数据类型对比

最近更新时间: 2024-06-12 15:06:00

Oracle	TDSQL PostgreSQL版（Oracle兼容版）
Number	可以对应 TDSQL PostgreSQL版（Oracle兼容版）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 numeric 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	float（实际按照 double precision 或 real 存储）
binary_float	binary_float（实际按照 real 存储）
binary_double	binary_double（实际按照 double preciso pion 存储）
char	char
nchar	char
varchar2	varchar2
nvarchar2	nvarchar2
rowid	rowid
urowid	urowid
long	long
clob	clob（将clob转换成text类型，最大可存储1GB）
nclob	nclob
blob	blob（将blob转换成bytea类型，最大可存储1GB）
bfile	bfile
Long raw	Long raw
raw	raw
date	date
timestamp	Timestamp
Interval	interval

与MYSQL数据类型对照表

最近更新时间: 2024-06-12 15:06:00

与Mysql对照表

Mysql	TDSQL PG
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);

与SQLSERVER数据类型对照表

最近更新时间: 2024-06-12 15:06:00

与SQLSERVER对照表

SQLSERVER	TDSQL PG
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

与INFORMIX数据类型对照表

最近更新时间: 2024-06-12 15:06:00

与informix对照表

informix	TDSQL PG
smallint	smallint
integer	integer
float	double precision
smallfloat	double precision
DECIMAL	numeric
MONEY	numeric
char	char
varchar	varchar
lvarchar	varchar
text	text
date	date
datetime	timestamp
interval	interval
BYTE	bytea

json/jsonb类型
json应用
创建json类型字段表

最近更新时间: 2024-06-12 15:06:00

```
postgres=# create table t_json(id int,f_json json);
CREATE TABLE
```

插入数据

最近更新时间: 2024-06-12 15:06:00

```
postgres=# insert into t_json values(1,'{"col1":1,"col2":"TDSQL PG"}');
INSERT 0 1
postgres=# insert into t_json values(2,'{"col1":1,"col2":"TDSQL PG","col3":"pgxz"}');
INSERT 0 1
postgres=# select * from t_json;
id | f_json
----+----------------------------------------
1 | {"col1":1,"col2":"TDSQL PG"}
2 | {"col1":1,"col2":"TDSQL PG","col3":"pgxz"}
(2 rows)
```

通过键获得 JSON 对象域

最近更新时间: 2024-06-12 15:06:00

```
postgres=# select f_json ->'col2' as col2 ,f_json -> 'col3' as col3 from t_json;
col2 | col3
---------+--------
"TDSQL PG" |
"TDSQL PG" | "pgxz"
(2 rows)
```

以文本形式获取对象值

最近更新时间: 2024-06-12 15:06:00

```
postgres=# select f_json ->>'col2' as col2 ,f_json ->> 'col3' as col3 from t_json;
col2 | col3
-------+------
TDSQL PG |
TDSQL PG | 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
-------+------
TDSQL PG | pgxz
(1 row)
```

jsonb应用
创建jsonb类型字段表

最近更新时间: 2024-06-12 15:06:00

```
postgres=# create table t_jsonb(id int,f_jsonb jsonb);
CREATE TABLE
```

插入数据

最近更新时间: 2024-06-12 15:06:00

```
postgres=# insert into t_jsonb values(1,'{"col1":1,"col2":"TDSQL PG"}');
INSERT 0 1
postgres=# insert into t_jsonb values(2,'{"col1":1,"col2":"TDSQL PG","col3":"pgxz"}');
INSERT 0 1
postgres=# select * from t_jsonb;
id | f_jsonb
----+-----------------------------------------------
1 | {"col1": 1, "col2": "TDSQL PG"}
2 | {"col1": 1, "col2": "TDSQL PG", "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":"TDSQL PG","col2":"pgxz"}');
INSERT 0 1
postgres=# select * from t_jsonb;
id | f_jsonb
----+-----------------------------------------------
1 | {"col1": 1, "col2": "TDSQL PG"}
3 | {"col1": 1, "col2": "pgxz"}
2 | {"col1": 1, "col2": "TDSQL PG", "col3": "pgxz"}
(3 rows)
```

更新数据

最近更新时间: 2024-06-12 15:06:00

1. 增加元素。

```
postgres=# update t_jsonb set f_jsonb = f_jsonb || '{"col3":"pgxz"}'::jsonb where id=1;
UPDATE 1
```

2. 更新原来的元素。

```
postgres=# update t_jsonb set f_jsonb = f_jsonb || '{"col2":"TDSQL PG"}'::jsonb where id=3;
UPDATE 1
postgres=# select * from t_jsonb;
id | f_jsonb
----+-----------------------------------------------
2 | {"col1": 1, "col2": "TDSQL PG", "col3": "pgxz"}
1 | {"col1": 1, "col2": "TDSQL PG", "col3": "pgxz"}
3 | {"col1": 1, "col2": "TDSQL PG"}
(3 rows)
```

3. 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": "TDSQL PG"}
1 | {"col1": 1, "col2": "TDSQL PG"}
3 | {"col1": 1, "col2": "TDSQL PG"}
(3 rows)
```

jsonb_set()函数更新数据

最近更新时间: 2024-06-12 15:06:00

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": "TDSQL PG"}
2 | {"col1": 1, "col2": "TDSQL PG"}
3 | {"col1": 1, "col2": "pgxz"}
(3 rows)
```

jsonb函数应用
jsonb_each()将json对象转变键和值

最近更新时间: 2024-06-12 15:06:00

```
postgres=# select f_jsonb from t_jsonb where id=1;
f_jsonb
-----------------------------------------------
{"col": "pgxz", "col1": 1, "col2": "TDSQL PG"}
(1 row)
postgres=# select * from jsonb_each((select f_jsonb from t_jsonb where id=1));
key | value
------+---------
col | "pgxz"
col1 | 1
col2 | "TDSQL PG"
(3 rows)
```

jsonb_each_text()将json对象转变文本类型的键和值

最近更新时间: 2024-06-12 15:06:00

```
postgres=# select * from jsonb_each_text((select f_jsonb from t_jsonb where id=1));
key | value
------+-------
col | pgxz
col1 | 1
col2 | TDSQL PG
(3 rows)
```

row_to_json()将一行记录变成一个json对象

最近更新时间: 2024-06-12 15:06:00

```
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 | TDSQL PG
2 | pgxz
(2 rows)
postgres=# select row_to_json(tbase) from tbase;
row_to_json
------------------------------
{"id":1,"nickname":"TDSQL PG"}
{"id":2,"nickname":"pgxz"}
(2 rows)
```

json_object_keys()返回一个对象中所有的键

最近更新时间: 2024-06-12 15:06:00

```
postgres=# select * from json_object_keys((select f_jsonb from t_jsonb where id=1)::json);
json_object_keys
------------------
col
col1
col2
(3 rows)
```

jsonb索引使用
创建立jsonb索引

最近更新时间: 2024-06-12 15:06:00

```
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
```

测试查询的性能

最近更新时间: 2024-06-12 15:06:00

```
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
```

部分数据类型示例

varchar2

最近更新时间: 2024-06-12 15:06:00

```
postgres=# create table t_varchar2(f1 varchar2,f2 int);
CREATE TABLE
postgres=# \d+ t_varchar2
Table "public.t_varchar2"
Column | Type | Collation | Nullable | Default | Storage | Stats target | Description
--------+----------+-----------+----------+---------+----------+--------------+-------------
f1 | varchar2 | | not null | | extended | |
f2 | integer | | | | plain | |
```

number

最近更新时间: 2024-06-12 15:06:00

```
postgres=# create table t_number(f1 number,f2 number(10),f3 number(10,2));
CREATE TABLE
postgres=# \d t_number
Table "public.t_number"
Column | Type | Collation | Nullable | Default
--------+---------------+-----------+----------+---------
f1 | numeric | | not null |
f2 | numeric(10,0) | | |
f3 | numeric(10,2) | | |
```

系统转换成numeric。

blob

最近更新时间: 2024-06-12 15:06:00

```
postgres=# create table t_blob(f1 int,f2 Blob);
CREATE TABLE
postgres=# \d t_blob
Table "public.t_blob"
Column | Type | Collation | Nullable | Default
--------+---------+-----------+----------+---------
f1 | integer | | not null |
f2 | blob | | |
```

TDSQL PG的blob类型支持最大长度为1G。

clob

最近更新时间: 2024-06-12 15:06:00

```
postgres=# create table t_clob(f1 int,f2 clob);
CREATE TABLE
postgres=# \d t_clob
Table "public.t_clob"
Column | Type | Collation | Nullable | Default
--------+---------+-----------+----------+---------
f1 | integer | | not null |
f2 | clob | | |
```

TDSQL PG的clob类型支持最大长度为1G。

创建和管理索引
创建和删除索引

最近更新时间: 2024-06-12 15:06:00

创建索引

使用命令create index创建索引 示例： 1) 创建唯一索引

```
create table t_first_col_share(id serial not null,nickname text);
create unique index t_first_col_share_id_uidx on t_first_col_share using btree(id);
```

⚠非shard key字段不能建立唯一索引 2) 创建多字段索引

```
create table t_mul_idx (f1 int,f2 int,f3 int,f4 int);
create index t_mul_idx_idx on t_mul_idx(f1,f2,f3);
```

⚠or查询条件bitmap scan最多支持两个不同字段条件

删除索引

使用命令DROP INDEX修改索引

```
drop index t_first_col_share_id_uidx;
```

索引类型
全局索引

最近更新时间: 2024-06-12 15:06:00

TDSQL PG是一个Share Nothing的MPP分布式数据，表数据通过Shard表的分布key散落到多个DN，DN间数据没有冗余。当业务查询或者DML操作有分布key条件时，CN上可以直接路由到目标DN。 全局索引存储了对应索引数据的存储目标节点信息，在业务查询缺少分布key条件时，也可以精确地定位对应索引key的数据在全局中的位置，不必扫描所有DN节点来获取数据，可以减少对资源的消耗，从而提高吞吐量（故全局索引一般创建在非分布key列）。 TDSQL PG全局索引支持的表类型：shard表，分区表。暂不支持复制表。 数据类型支持：能够用于shard列的类型，smallint, integer, bigint, real, double precision, binary_float, binary_double, smallserial, serial, bigserial, OID, boolean, char, name, varchar, text, varchar2, nvchar, oidvector, abstime, reltime, cash, bpchar, raw, byte, date, time, oracledate, timestamp, timestamptz, interval, numeric, jsonb, rid （bfile，xml，bytea不支持）。 索引类型默认Btree（暂仅支持该类型）。

语法： `create global [unique] index [if not exists] index_name on table_name [using method] (column_name);`

示例： 1、创建全局索引

```
postgres=#
create table gindex_test(a int,b int,c varchar,d numeric,e float);
postgres=#
create global index on gindex_test using btree (b);
```

---查询创建结果

```
postgres=#
\d+ gindex_test
Table "public.gindex_test"
Column | Type | Collation | Nullable | Default | Storage | Stats target | Description
--------+------------------+-----------+----------+---------+----------+--------------+-------------
a | integer | | not null | | plain | |
b | integer | | | | plain | |
c | character varying | | | | extended | |
d | numeric | | | | main | |
e | double precision | | | | plain | |
Indexes:
"gindex_test_b_idx" GLOBAL, btree (b)
Distribute By: SHARD(a)
Location Nodes: ALL DATANODES
```

2、创建全局唯一索引

```
postgres=#
create global unique index on gindex_test(c);
```

---查询创建结果

```
postgres=#
\d+ gindex_test
Table "public.gindex_test"
Column | Type | Collation | Nullable | Default | Storage | Stats target | Description
--------+------------------+-----------+----------+---------+----------+--------------+-------------
```

```
a | integer | | not null | | plain | |
b | integer | | | | plain | |
c | character varying | | | | extended | |
d | numeric | | | | main | |
e | double precision | | | | plain | |
Indexes:
"gindex_test_c_idx" UNIQUE, GLOBAL, btree (c)
"gindex_test_b_idx" GLOBAL, btree (b)
Distribute By: SHARD(a)
Location Nodes: ALL DATANODES
```

3、全局索引列的关联查询 ---创建表并插入数据

```
postgres=#
create table gi_insert1(a int,f1 int,f2 varchar2(20),f3 numeric,f4 float,f5 char(30),f6 nchar(20),f7 raw(20),f8 clob,f9 blob,f10 ti
mestamp with local time zone,f11 binary_double,f12 binary_float,f13 real,f14 nclob,f15 date,f16 timestamp,f17 interval day
to second, f18 interval year to month,f19 xmltype,f20 long) ;

postgres=#
insert into gi_insert1 select t,t,t,t,t,to_char(t),to_char(t),hextoraw(t),to_char(t),to_char(t)::blob, timestamp '2021-06-22 16:02:
07.067029'+t*interval '1' day, sin(t),t,t,to_clob(to_char(t)),date '2021-06-22 16:02:07'+ t*interval '1' day,timestamp '2021-06
-22 16:02:07.067029'+t*interval '1' day, t*interval '1' day,interval '3-2' year to month,'<ROW>XMLType2</ROW>' ,to_char
(t) from generate_series(1,10)t;


postgres=#
create table gi_insert(a int,f1 int,f2 varchar2(20),f3 numeric,f4 float,f5 char(30),f6 nchar(20),f7 raw(20),f8 clob,f9 blob,f10 ti
mestamp with local time zone,f11 binary_double,f12 binary_float,f13 real,f14 nclob,f15 date,f16 timestamp,f17 interval day
to second, f18 interval year to month,f19 xmltype,f20 long) ;

postgres=#
insert into gi_insert select t,t,t,t,t,to_char(t),to_char(t),hextoraw(t),to_char(t),to_char(t)::blob, timestamp '2021-06-22 16:02:0
7.067029'+t*interval '1' day, sin(t),t,t,to_clob(to_char(t)),date '2021-06-22 16:02:07'+ t*interval '1' day,timestamp '2021-06-
22 16:02:07.067029'+t*interval '1' day, t*interval '1' day,interval '3-2' year to month,'<ROW>XMLType2</ROW>' ,to_char
(t) from generate_series(1,10)t;
```

---创建全局索引

```
create global index on gi_insert1 using btree (f1);
create global index on gi_insert1 using btree (f2);
create global unique index on gi_insert1 (f3);
create global index on gi_insert1(f4);
create global index if not exists index5 on gi_insert1 using btree (f13);


create global index on gi_insert using btree (f1);
create global index on gi_insert using btree (f2);
create global unique index on gi_insert (f3);
create global index on gi_insert(f4);
create global index if not exists index51 on gi_insert using btree (f13);
```

---连接查询

```
select a.f1
from gi_insert a
join gi_insert1 b
```

```
on a.f1=b.f2
where a.f3=(select floor(avg(f)) from (select max(decode(f4,4,f4*f4,f4,f4+1,f4+10)) f
from gi_insert
where f19='<ROW>XMLType2</ROW>'
group by f4
having f4>=any (select (case f3 when 1 then 2 when 2 then 3 else 5 end)
from gi_insert1
where f1 in (select f1
from gi_insert
where f13>=any(select t1.f13 from gi_insert t1
left join gi_insert1 t2
on t1.f13=t2.f13
group by t1.f13
)
group by f1,f13
having f1<=(select max(f3)
from (select t1.f3 f3
from gi_insert t1
left join gi_insert1 t2
on t1.f1=t2.f1
group by t1.f3
) a
where f3 >=1
)
)
)
))
group by a.f1
having a.f1>=1;
```

--查询执行结果

```
f1
----
8
(1 row)
```

4、支持分区表创建全局唯一索引

```
--创建表
create table t_order_range_20220721_76(id bigserial not null,userid integer,product text, createdate date not null) partition by range ( createdate ) distribute by shard(id);

--创建分区
create table t_order_range_20220721_76_202204 partition of t_order_range_20220721_76(id primary key,userid,product, createdate) for values from ('2022-04-01') to ('2022-05-01');

create table t_order_range_20220721_76_202205 partition of t_order_range_20220721_76(id primary key,userid,product, createdate) for values from ('2022-05-01') to ('2022-06-01');

create table t_order_range_20220721_76_202206 partition of t_order_range_20220721_76(id primary key,userid,product, createdate) for values from ('2022-06-01') to ('2022-07-01');

create table t_order_range_20220721_76_202207 partition of t_order_range_20220721_76(id primary key,userid,product, createdate) for values from ('2022-07-01') to ('2022-08-01');
```

```
--插入数据
insert into t_order_range_20220721_76 values(1,7,'123','2022-05-13');
insert into t_order_range_20220721_76 values(2,8,'234','2022-06-13');
insert into t_order_range_20220721_76 values(3,9,'345','2022-07-13');
insert into t_order_range_20220721_76 values(6,10,'456','2022-07-03');
insert into t_order_range_20220721_76 values(4,11,'567','2022-07-11');
insert into t_order_range_20220721_76 values(5,12,'678','2022-05-08');



--create global index

create global index on t_order_range_20220721_76(userid);
select a.* from t_order_range_20220721_76 a, t_order_range_20220721_76_userid_idx_idt b where a.userid = b.indexkey an
d a.ctid = b.location and a.shardid = b.shard and a.tableoid = b.subtableoid order by id;
```

--查询结果

```
id | userid | product | createdate
----+--------+---------+---------------------
1 | 7 | 123 | 2022-05-13 00:00:00
2 | 8 | 234 | 2022-06-13 00:00:00
3 | 9 | 345 | 2022-07-13 00:00:00
4 | 11 | 567 | 2022-07-11 00:00:00
5 | 12 | 678 | 2022-05-08 00:00:00
6 | 10 | 456 | 2022-07-03 00:00:00
```

普通索引

最近更新时间: 2024-06-12 15:06:00

```
postgres=# create index t_appoint_id_idx on t_appoint_col using btree(id);
CREATE INDEX
```

唯一索引

最近更新时间: 2024-06-12 15:06:00

- 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.
```

表达式索引

最近更新时间: 2024-06-12 15:06:00

```
postgres=# create table t_upper(id int,mc text);
postgres=# create index t_upper_mc on t_upper(mc);
postgres=# insert into t_upper select t,md5(t::text) from generate_series(1,10000) as t;
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)
```

条件索引

最近更新时间: 2024-06-12 15:06:00

```
postgres=# create table t_sex(id int,sex text) ;
postgres=# create index t_sex_sex_idx on t_sex (sex);
postgres=# insert into t_sex select t,'男' from generate_series(1,1000000) as t;
postgres=# insert into t_sex select t,'女' from generate_series(1,100) as t;
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

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

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=# \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)
```

gist索引

最近更新时间: 2024-06-12 15:06:00

```
postgres=# create table t_trgm (id int,trgm text,no_trgm text) ;
postgres=# create index t_trgm_trgm_idx on t_trgm using gist(trgm gist_trgm_ops);
```

⚠使用gist索引需要预先安装插件。

```
create extension pg_trgm;
```

gin索引

最近更新时间: 2024-06-12 15:06:00

- 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);
```

- jsonb索引。

```
postgres=# create table t_jsonb(id int,f_jsonb jsonb);
postgres=# create index t_jsonb_f_jsonb_idx on t_jsonb using gin(f_jsonb);
```

- 数组索引。

```
postgres=# create table t_array(id int, mc text[]);
postgres=# insert into t_array select t,('{'||md5(t::text)||'}')::text[] from generate_series(1,1000000) as t;
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);
postgres=# insert into gin_mul select random()*5000, random()6000, now()+((30000-60000random())||' sec')::interval ,
md5(random()::text), round((random()*100000)::numeric,2), md5(random()::text) from generate_series(1,1000000);
postgres=# create extension btree_gin;
postgres=# create index gin_mul_gin_idx on gin_mul using gin(f1,f2,f3,f4,f5,f6);
#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 = 'fa627dc16c2b
  d026150afa0453a0991d'::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 = 'fa627dc16c2bd0
  26150afa0453a0991d'::text))
```

(6 rows) postgres=#

多字段索引

最近更新时间: 2024-06-12 15:06:00

```
postgres=# create table t_mul_idx (f1 int,f2 int,f3 int,f4 int);
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bitmapscan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;
QUERYPLAN
-------------------------------------------------------------------------------------------
Remote Fast Query Execution (cost=0.00..0.00 rows=0 width=0)
Node/s: dn001, dn002
-> Bitmap Heap Scan ont_mul_idx (cost=7617.08..7621.07 rows=2width=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_idxwhere f1=1 or f2=2 or f1=3;
QUERYPLAN
-------------------------------------------------------------------------------------------
Remote Fast Query Execution (cost=0.00..0.00 rows=0 width=0)
Node/s: dn001, dn002
-> Bitmap Heap Scan ont_mul_idx (cost=7619.51..7625.49 rows=3width=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_idxwhere f1=1 or f2=2 or f3=3;
QUERYPLAN
```

```
------------------------------------------------------------------
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;
QUERYPLAN
-------------------------------------------------------------------------------------------
Remote Fast Query Execution (cost=0.00..0.00 rows=0 width=0)
Node/s: dn001
-> Index Only Scan usingt_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
```

删除索引

最近更新时间: 2024-06-12 15:06:00

```
postgres=# drop index t_appoint_id_idx;
DROP INDEX
```

修改表
修改表结构场景
修改表名

最近更新时间: 2024-06-12 15:06:00

```
postgres=# alter table t rename to tbase;
ALTER TABLE
```

给表或字段添加注释

最近更新时间: 2024-06-12 15:06:00

```
postgres=# comment on table tbase is 'TDSQL PG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 | TDSQLPG分布式关系型数据库系统
(3 rows)
postgres=# comment on column tbase.nickname is 'TDSQL PG昵称是大象';
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 | | TDSQL PG昵称是大象
Has OIDs: no
Distribute By SHARD(id)
Location Nodes: ALL DATANODES

postgres=#
```

给表增加字段

最近更新时间: 2024-06-12 15:06:00

```
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 | | TDSQL PG昵称是大象
age | integer | | plain | |
Has OIDs: no
Distribute By SHARD(id)
Location Nodes: ALL DATANODES
```

修改字段类型

最近更新时间: 2024-06-12 15:06:00

```
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 | | TDSQL PG昵称是大象
age | double precision | | plain | |
Has OIDs: no
Distribute By SHARD(id)
Location Nodes: ALL DATANODES
```

修改字段默认值

最近更新时间: 2024-06-12 15:06:00

```
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 | | TDSQL PG昵称是大象
age | double precision | default 0.0 | plain | |
Has OIDs: no
Distribute By SHARD(id)
Location Nodes: ALL DATANODES
```

删除字段

最近更新时间: 2024-06-12 15:06:00

```
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 | | TDSQL PG昵称是大象
Has OIDs: no
Distribute By SHARD(id)
Location Nodes: ALL DATANODES
```

添加主键

最近更新时间: 2024-06-12 15:06:00

```
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
```

删除主键

最近更新时间: 2024-06-12 15:06:00

```
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
```

如果是分区表，则删除主键要加上cascade，强制删除关联的子表主键。

重建主键

最近更新时间: 2024-06-12 15:06:00

```
postgres=# \d t
Table "public.t"
Column | Type | Collation | Nullable | Default
--------+---------+-----------+----------+---------
id | integer | | not null |
mc | text | | |
Indexes:
"t_pkey" PRIMARY KEY, btree (id)

postgres=# CREATE UNIQUE INDEX CONCURRENTLY t_id_temp_idx ON t (id);
CREATE INDEX
postgres=#
postgres=# \d t
Table "public.t"
Column | Type | Collation | Nullable | Default
--------+---------+-----------+----------+---------
id | integer | | not null |
mc | text | | |
Indexes:
"t_pkey" PRIMARY KEY, btree (id)
"t_id_temp_idx" UNIQUE, btree (id)

postgres=#

postgres=# ALTER TABLE t DROP CONSTRAINT t_pkey, ADD CONSTRAINT t_pkey PRIMARY KEY USING INDEX t_id_temp_i
dx;
NOTICE: ALTER TABLE / ADD CONSTRAINT USING INDEX will rename index "t_id_temp_idx" to "t_pkey"
ALTER TABLE
postgres=#
postgres=# \d t
Table "public.t"
Column | Type | Collation | Nullable | Default
--------+---------+-----------+----------+---------
id | integer | | not null |
mc | text | | |
Indexes:
"t_pkey" PRIMARY KEY, btree (id)

postgres=#
```

添加外键

最近更新时间: 2024-06-12 15:06:00

```
postgres=# create table t_p(f1 int not null,f2 int ,primary key(f1));
postgres=# create table t_f(f1 int not null,f2 int );
postgres=# ALTER TABLE t_f ADD CONSTRAINT t_f_f1_fkey FOREIGN KEY (f1) REFERENCES t_p (f1);
ALTER TABLE
postgres=# \d+ t_f
Table "public.t_f"
Column | Type | Collation | Nullable | Default | Storage | Stats target | Description
--------+---------+-----------+----------+---------+---------+--------------+-------------
f1 | integer | | not null | | plain | |
f2 | integer | | | | plain | |
Foreign-key constraints:
"t_f_f1_fkey" FOREIGN KEY (f1) REFERENCES t_p(f1)
Distribute By: SHARD(f1)
Location Nodes: ALL DATANODES
```

外键使用限制：

- 外键只是同一个节点内约束有效果，所以外键字段和对应主键字段必需都是表的分布键，否则由于数据分布于不同的节点内会导致更新失败。
- 分区表和冷热分区表也不支持外键，数据分区后位于不同的物理文件中，无法约束。

删除外键

最近更新时间: 2024-06-12 15:06:00

```
postgres=# ALTER TABLE t_f DROP CONSTRAINT t_f_f1_fkey;
ALTER TABLE
```

修改表所属模式

最近更新时间: 2024-06-12 15:06:00

```
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)
```

修改表所属用户

最近更新时间: 2024-06-12 15:06:00

```
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)
```

修改字段名

最近更新时间: 2024-06-12 15:06:00

```
postgres=# \d+ tbase
Table "public.tbase"
Column | Type | Collation | Nullable | Default | Storage | Stats target | Description
--------+----------------------+-----------+----------+---------+----------+--------------+-------------
id | integer | | not null | | plain | |
city | character varying(50) | | | | extended | |
Distribute By: SHARD(id)
Location Nodes: dn01

postgres=# alter table tbase rename city to cityname;
ALTER TABLE

postgres=# \d+ tbase
Table "public.tbase"
Column | Type | Collation | Nullable | Default | Storage | Stats target | Description
----------+----------------------+-----------+----------+---------+----------+--------------+-------------
id | integer | | not null | | plain | |
cityname | character varying(50) | | | | extended | |
Distribute By: SHARD(id)
Location Nodes: dn01
```

修改表的填充率

最近更新时间: 2024-06-12 15:06:00

一个表的填充率(fillfactor)默认值是100。INSERT 操作仅按照填充率指定的百分率填充表页。每个页上的剩余空间将用于在该页上的行更新，这就使得 UPDATE 有机会在同一页上放置同一条记录的新版本，这比把新版本放置在其它页上更有效。对于一个从不更新的表将填充因子设为 100 效率最高，但是对于频繁更新的表，较小的填充率则更加有效，但小的填充率会占用更大有空间。 在PostgreSQL中，当更新一行数据时，实际上旧行并没有删除，只是插入了一行新数据。如果这个表其他列上有索引，而更新的列上没有索引，因为新行的物理位置发生变化，因此需要更新索引，这将导致性能下降。为了解决这个问题，PostgreSQL引入了Heap Only Tuple（HOT）技术，如果更新后的新行和旧行位于同一个数据块内，则旧行会有一个指针指向新行，这样就不用更新索引了，通过索引访问到旧行数据，进而访问到新行数据。

```
postgres=# \d+ t1
Table "tbase_pg_proc.t1"
Column | Type | Collation | Nullable | Default | Storage | Stats target | Description
--------+---------+-----------+----------+---------+---------+--------------+-------------
f1 | integer | | not null | | plain | |
f2 | integer | | | | plain | |
Indexes:
"t1_f1_idx" btree (f1)
Distribute By: SHARD(f1)
Location Nodes: dn002, dn001

postgres=# alter table t1 set(fillfactor=80);
ALTER TABLE
postgres=# \d+ t1
Table "tbase_pg_proc.t1"
Column | Type | Collation | Nullable | Default | Storage | Stats target | Description
--------+---------+-----------+----------+---------+---------+--------------+-------------
f1 | integer | | not null | | plain | |
f2 | integer | | | | plain | |
Indexes:
"t1_f1_idx" btree (f1)
Distribute By: SHARD(f1)
Location Nodes: dn002, dn001
Options: fillfactor=80

postgres=#
```

添加触发器

最近更新时间: 2024-06-12 15:06:00

- INSERT触发器。

```
postgres=# create table t_trigger(f1 int,f2 int);
CREATE TABLE
postgres=# CREATE OR REPLACE FUNCTION t_trigger_insert_trigger_func () RETURNS trigger AS
$body$
BEGIN
if NEW.f2
NEW.f2=0;
endif;
RETURN NEW;
END;
$body$
LANGUAGE plpgsql;
CREATE FUNCTION
postgres=# CREATE TRIGGER t_trigger_insert_trigger BEFORE INSERT ON t_trigger FOR EACH ROW EXECUTE PROCEDU
RE t_trigger_insert_trigger_func();
CREATE TRIGGER
上面触发器的作用是在插入记录时，如果f2字段值小于0时，则将f2的值修改成0，效果如下

postgres=# insert into t_trigger values(1,-1);
INSERT 0 1
postgres=# select * from t_trigger;
f1 |f2
----+----
1| 0
```

- UPDATE触发器。

```
postgres=# CREATE OR REPLACE FUNCTION t_trigger_update_trigger_func () RETURNS trigger AS
$body$
BEGIN
if NEW.f2
NEW.f2=OLD.f2;
endif;
RETURN NEW;
END;
$body$
LANGUAGE plpgsql;
CREATE FUNCTION
postgres=# CREATE TRIGGER t_trigger_update_trigger BEFORE UPDATE ON t_trigger FOR EACH ROW EXECUTE PROCED
URE t_trigger_update_trigger_func();
CREATE TRIGGER
postgres=#

上面触发器的作用是在修改记录时，如果f2字段值小于0时，则将f2的值修改成原值，效果如下

postgres=# update t_trigger set f2=-1 where f1=1;
UPDATE 1
postgres=# select * from t_trigger;
```

```
f1 |f2
----+----
1| 0

postgres=# update t_trigger set f2=1 where f1=1;
UPDATE 1
postgres=# select * from t_trigger;
f1 |f2
----+----
1| 1
```

- DELETE触发器。

```
postgres=# CREATE OR REPLACE FUNCTION t_trigger_delete_trigger_func () RETURNS trigger AS
$body$
BEGIN
if OLD.f2=0 then
RETURN NULL;
endif;
RETURN OLD;
END;
$body$
LANGUAGE plpgsql;
CREATE FUNCTION
postgres=# CREATE TRIGGER t_trigger_delete_trigger BEFORE DELETE ON t_trigger FOR EACH ROW EXECUTE PROCEDU
RE t_trigger_delete_trigger_func();
CREATE TRIGGER

上面触发器的作用是在删除记录时，如果f2字段值等于0时，则不删除记录，效果如下

postgres=# delete from t_trigger where f2=0;
DELETE 0
postgres=# select * from t_trigger;
f1 |f2
----+----
1| 0

postgres=# update t_trigger set f2=10;
UPDATE 1
postgres=# select * from t_trigger;
f1 |f2
----+----
1 |10
(1 row)

postgres=# delete from t_trigger where f1=1;
DELETE 1
postgres=# select * from t_trigger;
f1 |f2
----+----
(0 rows)
```

- 多个事件。

```
postgres=# create table t_trigger_mulevent(f1 int,f2 int);
CREATE TABLE
CREATE OR REPLACE FUNCTION t_trigger_mulevent_func () RETURNS trigger AS
$body$
BEGIN
if NEW.f2
NEW.f2=0;
endif;
RETURN NEW;
END;
$body$
LANGUAGE plpgsql;
CREATE FUNCTION
postgres=#

postgres=# CREATE TRIGGER t_trigger_insert_update_trigger BEFORE INSERT OR UPDATE ON t_trigger_mulevent FOR E
ACH ROW EXECUTE PROCEDURE t_trigger_mulevent_func();
CREATE TRIGGER
```

上面触发器的作用是在插入和更新记录时，如果f2字段值小于0时，则f2值重置为0，效果如下

```
postgres=# insert into t_trigger_mulevent values(1,-10);
INSERT 0 1
postgres=# select * from t_trigger_mulevent;
f1 |f2
----+----
1| 0
(1 row)

postgres=# update t_trigger_mulevent setf2=-10;
UPDATE 1
postgres=# select * from t_trigger_mulevent;
f1 |f2
----+----
1| 0
(1 row)
```

删除触发器

最近更新时间: 2024-06-12 15:06:00

```
#删除触发器
postgres=# DROP TRIGGER t_trigger_insert_update_trigger ON t_trigger_mulevent;
DROP TRIGGER
postgres=#
#删除触发器函数
postgres=# drop function t_trigger_mulevent_func();
DROP FUNCTION
postgres=#
```

truncate普通表

最近更新时间: 2024-06-12 15:06:00

```
postgres=# truncate table t1;
TRUNCATE TABLE
#也可以一次truncate多个数据表
postgres=# truncate table t1,t2;
TRUNCATE TABLE
```

删除表

最近更新时间: 2024-06-12 15:06:00

使用命令drop table删除表 示例： 当表存在时删除

```
drop table if exists tbase;
```

表管理场景
不用指定shard key 建表方式

最近更新时间: 2024-06-12 15:06:00

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
```

分布键选择原则：

- 分布键只能选择一个字段。
- 如果有主键，则选择主键做分布键。
- 如果主键是复合字段组合，则选择字段值选择性多的字段做分布键。
- 也可以把复合字段拼接成一个新的字段来做分布键。
- 没有主键的可以使用UUID来做分布键。
- 总之一定要让数据尽可能的分布得足够散。

指定shard key 建表方式

最近更新时间: 2024-06-12 15:06:00

```
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
```

指定group 建表方式

最近更新时间: 2024-06-12 15:06:00

```
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
```

复制表

最近更新时间: 2024-06-12 15:06:00

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。

列存表管理
创建列存表

最近更新时间: 2024-06-12 15:06:00

```
postgres=# create table t_col_test(f1 int,f2 varchar(32),f3 date) with(orientation='column');
CREATE TABLE
postgres=# \d+ t_col_test
Table "public.t_col_test"
Column | Type | Collation | Nullable | Default | Storage | Stats target | Description
--------+----------------------+-----------+----------+---------+----------+--------------+-------------
f1 | integer | | | | plain | |
f2 | character varying(32) | | | | extended | |
f3 | date | | | | plain | |
Options: orientation=column
```

指定压缩类型

最近更新时间: 2024-06-12 15:06:00

```
postgres=# create table t_col_compress
(
f1 int encoding(compress_method='delta'),
f2 varchar(32)ENCODING(compress_method='zlib',compress_level=9) ,
f3 date ENCODING(compress_method='zlib',compress_level=9)
) with(orientation='column');
CREATE TABLE
postgres=# \d+ t_col_compress
Table"public.t_col_compress"
Column | Type | Collation | Nullable | Default |Storage | Stats target | Description
--------+----------------------+-----------+----------+---------+----------+--------------+-------------
f1 |integer | | | | plain | |
f2 |character varying(32) | | | | extended | |
f3 |date | | | | plain | |
Options: orientation=column

postgres=#
```

指定表列字段的压缩方式及压缩等级创建表 目前支持压缩方式compress_method包括（delta、zstd、zlib、rle、bitpack），compress_level表示缩级别；压缩级别越大，压缩比越大，压缩时间越长，压缩级别越高越消耗cpu。 轻量压缩为自研实现算法，包括rle、delta、bitpack，他们均只有一种压缩级别，且只能压缩数值类型。 其中rle主要针对大量重复的数据，比如11111122222333111111进行压缩。delta主要针对递增或者递减的数据，比如固定的时间类型(今天明天后天)，或者数值类型12345678进行压缩。bitpack针对数值比较小的数据，比如创建表的时候是integer，但实际存储的数值只是1-100。 创建表的时候指定压缩类型，但是实际存储的时候有没有用到指定的压缩类型是会自动适应调整的，有写场景可能导致最终存储没有使用压缩，如：压缩算法使用不当，比如针对text类型使用delta。 约束数据对比，100万行记录。

```
postgres=# \dt+
List ofrelations
Schema | Name | Type | Owner | Size | Description
--------+----------------------------+-------+-------+------------+-------------
public | t_col_compress | table | tbase | 104 MB |
public | t_col_test | table | tbase | 119 MB |
public | t_row_test | table | tbase | 97 MB |
```

下面对比一下重复率比较高的，测试表如下：

- 表定义

  表定义

行存	列存	列存（压缩）

行存	列存	列存（压缩）
create table t_row_normal (f1 int, f2 text , f3 text , f4 text , f5 text , f6 text , f7 text , f8 text , f9 text , f10 text) ;	create table t_col_normal (f1 int, f2 text , f3 text , f4 text , f5 text , f6 text , f7 text , f8 text , f9 text , f10 text) with(orientation='column');	create table t_col_compress (f1 int, f2 text ENCODING(compress_method='zstd',compress_level=1) , f3 text ENCODING(compress_method='zstd',compress_level=1) , f4 text ENCODING(compress_method='zstd',compress_level=1) , f5 text ENCODING(compress_method='zstd',compress_level=1) , f6 text ENCODING(compress_method='zstd',compress_level=1) , f7 text ENCODING(compress_method='zstd',compress_level=1) , f8 text ENCODING(compress_method='zstd',compress_level=1) , f9 text ENCODING(compress_method='zstd',compress_level=1) , f10 text ENCODING(compress_method='zstd',compress_level=1)) with(orientation='column');

- 测试脚本

```
insert into t_row_normalselectt,repeat(random()::text,3),repeat(random()::text,3),repeat(random()::text,3),repeat(random()::text,3),repeat(random()::text,3),repeat(random()::text,3),repeat(random()::text,3),repeat(random()::text,3),repeat(random()::text,3)from generate_series(1,1000000) as t;

insert into t_col_normalselectt,repeat(random()::text,3),repeat(random()::text,3),repeat(random()::text,3),repeat(random()::text,3),repeat(random()::text,3),repeat(random()::text,3),repeat(random()::text,3),repeat(random()::text,3),repeat(random()::text,3)from generate_series(1,1000000) as t;

insert into t_col_compressselectt,repeat(random()::text,3),repeat(random()::text,3),repeat(random()::text,3),repeat(random()::text,3),repeat(random()::text,3),repeat(random()::text,3),repeat(random()::text,3),repeat(random()::text,3),repeat(random()::text,3),repeat(random()::text,3)from generate_series(1,1000000) as t;
```

- 用时及容量对比

对比

项目	行存	列存	列存（带压缩）
大小	521MB	896MB	480MB
生成数据时间	8.029	12.801	11.818

下面对比一下重复率非常高的，测试表如下：

- 测试脚本

```
insert into t_row_normalselectt,repeat(random()::text,3),repeat(random()::text,3),repeat(random()::text,3),repeat(random()::text,3),repeat(random()::text,3),repeat(random()::text,3),repeat(random()::text,3),repeat(random()::text,3),repeat(random()::text,3)from generate_series(1,1000000) as t;

insert into t_col_normalselectt,repeat(random()::text,3),repeat(random()::text,3),repeat(random()::text,3),repeat(random()::text,3),repeat(random()::text,3),repeat(random()::text,3),repeat(random()::text,3),repeat(random()::text,3),repeat(random()::text,3)from generate_series(1,1000000) as t;

insert into t_col_compressselect t,repeat(random()::text,3),repeat(random()::text,3),repeat(random()::text,3),repeat(random()::text,3),repeat(random()::text,3),repeat(random()::text,3),repeat(random()::text,3),repeat(random()::text,3),repeat(random()::text,3)from generate_series(1,1000000) as t;
```

- 用时及容量对比

对比

项目	行存	列存	列存（带压缩）
大小	1898MB	1976MB	502MB
生成数据时间	12.680	26.019	19.261

tdx外表管理
tdx服务

最近更新时间: 2024-06-12 15:06:00

- 部署tdx服务

```
[tbase@master ~]$ mkdir -p/data/tbase/tdx/tbase_pgxz
[tbase@master ~]$ mkdir -p/data/tbase/tdx/data
[tbase@master ~]$
#复制一份TDSQL PG二进制程序到/data/tbase/tdx/tbase_pgxz,然后配置环境变量如下
export PGXZ_HOME=/data/tbase/tdx/tbase_pgxz
export PATH=$PGXZ_HOME/bin:$PATH
exportLD_LIBRARY_PATH=$PGXZ_HOME/lib:${LD_LIBRARY_PATH}
```

- 启动服务

`tdx -d /data/tbase/tdx/data -p 8088 -l/data/tbase/tdx/tdx.log`

`nohup tdx -d /data/tbase/tdx/data -p 8088-l /data/tbase/tdx/tdx.log & #` 放后面运行

参数说明

参数	说明
-d	工作目录，后面的数据文件存放地方
-p	服务port
-l	日志文件名称

- 停止服务

```
[tbase@master ~]$ ps -ewf |grep tdx |grep -v color
tbase 20279 13747 0 14:55 pts/1 00:00:00 tdx -d /data/tbase/tdx/data -p8088 -l /data/tbase/tdx/tdx.log
[tbase@master ~]$ kill 20279
```

创建exttable_fdw

最近更新时间: 2024-06-12 15:06:00

```
postgres=# create extension exttable_fdw;
CREATE EXTENSION
```

创建外表

最近更新时间: 2024-06-12 15:06:00

- Csv格式

```
postgres=# CREATE EXTERNAL TABLE t_table_csv
(
f1 int,
f2 text ,
f3 text ,
f4 text ,
f5 text ,
f6 text ,
f7 text ,
f8 text ,
f9 text ,
f10 text
)
LOCATION ('tdx://172.16.0.23:8088/t_table.csv')
FORMAT 'csv';
```

> **说明：**
>
> (header DELIMITER '|') -- csv文件第一行为列的名称，列分隔符为竖线(默认为逗号)。

- text格式

```
postgres=# CREATE EXTERNAL TABLE t_table_text
(
f1 int,
f2 text ,
f3 text ,
f4 text ,
f5 text ,
f6 text ,
f7 text ,
f8 text ,
f9 text ,
f10 text
)
LOCATION ('tdx://172.16.0.23:8088/t_table.txt')
FORMAT 'text' (delimiter E'\x0E' null '' eol E'\x0F\n');
```

> **说明：**
>
> delimiter E'\x0E' 表示列分隔符为音乐符，空串”转换为null，行分隔符为太阳符加回车。

删除外表

最近更新时间: 2024-06-12 15:06:00

```
postgres=# drop EXTERNAL table t_table_csv;
DROP FOREIGN TABLE
```

将外表数据导入到物理表中

最近更新时间: 2024-06-12 15:06:00

```
postgres=# insert into t_col_compress select * from t_table_text;
INSERT 0 1000000
Time: 19241.301 ms (00:19.241)
```

导入数据对比，导入文件大小8.6G，行数1700万 对比项目

对比项目	行存	列存	列存（带压缩）
从tdx外表导入	100s	158s	137s
copy	129s	198	169s
大小	9.9G	12G	2.3G

指定模式创建表

最近更新时间: 2024-06-12 15:06:00

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

使用将查询结果创建数据表

最近更新时间: 2024-06-12 15:06:00

```
postgres=# create table t(id int,mc text) distribute by shard(mc);
CREATE TABLE
postgres=# insert into t values(1,'TDSQL PG');
INSERT 0 1
postgres=# create table t_as as select * from t;
INSERT 0 1
postgres=# select * from t_as;
id | mc
----+-------
1 | TDSQL PG
(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=#
```

删除数据表

最近更新时间: 2024-06-12 15:06:00

- 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=#
```

copy的使用
实验表结构及数据

最近更新时间: 2024-06-12 15:06:00

```
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 | TDSQL PG | | 7
2 | | 2017-10-28 18:24:05.643102 | 3
(3 rows)
```

copy to用法详解--复制数据到文件中
导出所有列

最近更新时间: 2024-06-12 15:06:00

```
postgres=# copy public.t to '/data/pgxz/t.txt';
COPY 3
postgres=# \! cat /data/pgxz/t.txt
1 TDSQL PG \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。

导出部分列

最近更新时间: 2024-06-12 15:06:00

```
postgres=# copy public.t(f1,f2) to '/data/pgxz/t.txt';
COPY 3
postgres=# \! cat /data/pgxz/t.txt
1 TDSQL PG
2
3 pgxz

#只导出f1和f2列
```

导出查询结果

最近更新时间: 2024-06-12 15:06:00

```
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
```

指定生成文件格式

最近更新时间: 2024-06-12 15:06:00

- 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,TDSQL PG,,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
TDSQL PG
```

默认为TEXT格式。

使用delimiter指定列与列之间的分隔符

最近更新时间: 2024-06-12 15:06:00

```
postgres=# copy public.t to '/data/pgxz/t.txt' with delimiter '@';
COPY 3
postgres=# \! cat /data/pgxz/t.txt
1@TDSQL PG@\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@TDSQL PG@@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。

NULL值的处理

最近更新时间: 2024-06-12 15:06:00

```
postgres=# copy public.t to '/data/pgxz/t.txt' with NULL 'NULL';
COPY 3
postgres=# \! cat /data/pgxz/t.txt
1 TDSQL PG 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,TDSQL PG,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。

生成列标题名

最近更新时间: 2024-06-12 15:06:00

```
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,TDSQL PG,,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。

导出oids系统列

最近更新时间: 2024-06-12 15:06:00

```
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 | TDSQL PG | | 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 TDSQL PG \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。

使用quote自定义引用字符

最近更新时间: 2024-06-12 15:06:00

```
postgres=# copy t to '/data/pgxz/t.txt' with csv;
COPY 3
postgres=# \! cat /data/pgxz/t.txt
1,TDSQL PG,,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,TDSQL PG,,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。
```

使用escape自定义逃逸符

最近更新时间: 2024-06-12 15:06:00

```
postgres=# copy t to '/data/pgxz/t.txt' with quote '%' csv;
COPY 3
postgres=# \! cat /data/pgxz/t.txt
1,TDSQL PG,,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,TDSQL PG,,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
```

强制给某个列添加引用字符

最近更新时间: 2024-06-12 15:06:00

```
postgres=# copy t to '/data/pgxz/t.txt' (format 'csv',force_quote (f1,f2));
COPY 3
postgres=# \! cat /data/pgxz/t.txt
"1","TDSQL PG",,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","TDSQL PG",,"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。
```

使用encoding指定导出文件内容编码

最近更新时间: 2024-06-12 15:06:00

```
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
```

copy from用法详解--复制文件内容到数据表中导入所有列

最近更新时间: 2024-06-12 15:06:00

```
postgres=# \! cat /data/pgxz/t.txt
1 TDSQL PG \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 | TDSQL PG | | 7
2 | pg'", xc% | 2017-10-28 18:24:05.643102 | 3
3 | pgxz | 2017-10-28 18:24:05.645691 |
(3 rows)
```

导入部分指定列

最近更新时间: 2024-06-12 15:06:00

```
postgres=# copy t(f1,f2) to '/data/pgxz/t.txt';
postgres=# \! cat /data/pgxz/t.txt
1 TDSQL PG
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 | TDSQL PG | 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 |
(3 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 TDSQL PG
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 | TDSQL PG | |
2 | pg'", xc% | 2017-10-28 18:24:05.643102 |
3 | pgxz | 2017-10-28 18:24:05.645691 |
(3 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 TDSQL PG \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 TDSQL PG \N 7"
```

数据文件的列表不能多于要导入的列数，否则会出错。

指定导入文件格式

最近更新时间: 2024-06-12 15:06:00

```
postgres=# \! cat /data/pgxz/t.txt
1 TDSQL PG \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,TDSQL PG,,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 | TDSQL PG | | 7
2 | pg'", xc% | 2017-10-28 18:24:05.643102 | 3
3 | pgxz | 2017-10-28 18:24:05.645691 |
(3 rows)
```

使用delimiter指定列与列之间的分隔符

最近更新时间: 2024-06-12 15:06:00

```
postgres=# \! cat /data/pgxz/t.txt
1%TDSQL PG%\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 | TDSQL PG | | 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%TDSQL PG%%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 | TDSQL PG | | 7
2 | pg'", xc% | 2017-10-28 18:24:05.643102 | 3
3 | pgxz | 2017-10-28 18:24:05.645691 |
(3 rows)
```

NULL值处理

最近更新时间: 2024-06-12 15:06:00

```
postgres=# \! cat /data/pgxz/t.txt
1 TDSQL PG 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 | TDSQL PG | | 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"
```

自定义quote字符

最近更新时间: 2024-06-12 15:06:00

```
postgres=# \! cat /data/pgxz/t.csv
1,TDSQL PG,,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
```

自定义escape字符

最近更新时间: 2024-06-12 15:06:00

```
postgres=# \! cat /data/pgxz/t.csv
1,TDSQL PG,,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 | TDSQL PG | | 7
2 | pg'", xc% | 2017-10-28 18:24:05.643102 | 3
3 | pgxz | 2017-10-28 18:24:05.645691 |
(3 rows)
postgres=#
```

csv header忽略首行

最近更新时间: 2024-06-12 15:06:00

```
postgres=# \! cat /data/pgxz/t.csv;
f1,f2,f3,f4
1,TDSQL PG,,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 | TDSQL PG | | 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。

导入oid列值

最近更新时间: 2024-06-12 15:06:00

```
postgres=# \! cat /data/pgxz/t.txt
35242 1 TDSQL PG \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 | TDSQL PG | | 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)
```

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

最近更新时间: 2024-06-12 15:06:00

```
postgres=#truncate table t;
TRUNCATETABLE
postgres=#\! cat '/data/pgxz/t.csv' ;
1,TDSQL PG,,7
2,"pg'"", xc%",2017-10-28 18:24:05.643102,3
3,pgxz,2017-10-2818:24:05.645691,
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, 2017-10-3016:14:14.954213, 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;
TRUNCATETABLE
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
row)
```

使用FORCE_NOT_NULL处理就变成长度为0的字符串。

encoding指定导入文件的编码

最近更新时间: 2024-06-12 15:06:00

```
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 | TDSQL PG | | 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 | TDSQL PG | | 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)
#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 | TDSQL PG | | 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=#
```

position指定字段长度导入

最近更新时间: 2024-06-12 15:06:00

```
postgres=# \! cat /data/tbase/t_position_copy.txt
123阿弟2021
postgres=#
postgres=# copy t_position_copy (f1 position(1:3),f2 position(4:9) ,f3 position(10:13)) from '/data/tbase/t_position_copy.tx
t';
COPY 1
postgres=#
```

position指定字段长度+函数处理导入

最近更新时间: 2024-06-12 15:06:00

```
postgres=# \! cat /data/tbase/t_position_copy.txt
123阿弟2020010120200101121212
postgres=# copy t_position_copy (f1 position(1:3),f2 position(4:9) ,f3 position(10:17) to_date($3,'yyyymmdd'),f4 position(1
8:31) to_timestamp($4,'yyyymmddhh24miss')) from '/data/tbase/t_position_copy.txt';
COPY 1
```

copy支持多字节分隔符

最近更新时间: 2024-06-12 15:06:00

TDSQL PG支持多字节分隔符入库，限制10字节以内，支持中文分隔符，支持多字符集utf8以及gbk。 示例： 1、csv格式入库 --创建表

```
postgres=#
create table test_copy_base_1(c0 int,c1 macaddr,c2 point,c3 text,c4 numeric,c5 char,c6 oidvector,c7 float4,c8 abstime,c9 rel
time,c10 polygon,c11 bool,c12 tinterval,c13 money,c14 tid,c15 circle,c16 macaddr8,c17 inet,c18 varbit(5),c19 varchar,c20 ti
mestamp,c21 oid,c22 line,c23 int4,c24 int2,c25 pg_oracle.date,c26 path,c27 bit varying(5),c28 int8,c29 box,c30 lseg,c31 na
me,c32 interval,c33 timestamptz,c34 bpchar,c35 time,c36 float8,c37 timetz,c38 bytea,c39 varchar2,c40 number,c41 serial,c
42 bigserial) distribute by shard(c0);
```

---插入数据

```
create or REPLACE procedure insert_data_start(starts int, table_num int, table_name varchar2)
as
v_sql varchar2;
region varchar[];
BEGIN
region := array['深圳','北京','上海','广州','重启','杭州','合肥','成都'];
v_sql := 'insert into '||table_name||' select i, concat_ws(''-'',to_hex((i%192)+1),to_hex((i%168)+1),to_hex((i%200)+1),to_hex
((i%100)+1),to_hex((i%98)+1),to_hex((i%255)+1))::macaddr,point(ceil((i%68)+1),ceil((i%77)+1)),''南山''||(i%100)||''号'',round
((0.356*500*i)%876,3)::numeric,(i%10)::char,(i*987)::varchar::oidvector,round(i*5.301,3)::float4,abstime(''2022-06-07 12:12:1
2''::timestamp+(i%256)*interval ''1 day''),reltime(i*542),polygon (box(point(i%550,i%360),point(i%50,i%22))),bool(i%2),tinte
rval(abstime(''2022-06-07 12:12:12''::timestamp+(i%300)*interval ''1 day''),abstime(''2022-06-07 12:12:12''::timestamp+i*in
terval ''1 hour'')),((i*3.156)%9999)::money,(i%798,i%666)::varchar::tid,circle(point(i%250,i%160),i%76),concat_ws(''-'',to_hex
((i%192)+16),to_hex((i%168)+16),to_hex((i%200)+16),to_hex((i%100)+16),to_hex((i%98)+16),to_hex((i%239)+16))::macaddr
8,concat(concat_ws(''.'',(i%100)+100,(i%100)+1,(i%100)+1,(i%100)+100),''/25'')::inet,((i+1)%98799)::int::bit(5)::varbit(5),(''开
发''||(i%20)+1||''组'')::varchar,(date ''2022-06-07 12:12:12''+i * interval ''1 day''+(i%24) * interval ''1 hour''+(i%60) * interval
''1 minute''+(i%60) * interval ''1 second'')::timestamp,(i%999)::bigint::oid,line(point(i%93+1,i%120),point(i%450+2,i%36)),in
t4(i%1000),int2(i%150),date ''2022-06-07 12:12:12''+(i%378)*interval ''1 day'',path (polygon(box(point(i%192,i%120),point
(i%913,i%120)))),int4(i%10000)::bit(5),int8(i*999),box(point(i%292+1,i%20),point(i%92+2,i%24)),lseg(point(i%92+1,i%220),
point(i%238+2,i%120)),name(''腾讯''||(i%12)),i * interval ''1 day'',timestamptz(date(date ''2022-06-07 12:12:12'' + (i%777) *
interval ''1 day''),(timestamp ''2022-06-07 12:12:12''+(i%100) * interval ''1 hour'')::time without time zone),bpchar((''你好''||
(i%98))::text),(i%300)*interval ''1 hour 1 minute 1 second'',float8((i*3.122333)%998),timetz(timestamptz ''2022-06-07 10:56:
00+08'' + (i%200) * interval ''1 hour''),repeat(''深圳''||(i%1000),3)::bytea,(''tbase in 深圳''||(i%900))::varchar2,((i*3.1123)%99
8)::number(10,2) from generate_series('||starts||','||table_num||') i';
-- RAISE notice '%', v_sql;
execute v_sql;
end;
/

postgres=#
call insert_data_start(1, 5000, 'test_copy_base_1');
postgres=#
analyze test_copy_base_1;
```

--出库

```
COPY (select * from test_copy_base_1 order by c0) to '/tmp/test_copy_base_1.data' with CSV delimiter '$^$';
```

--入库

```
COPY test_copy_base_1 from '/tmp/test_copy_base_1.data' with CSV delimiter '$^$';
```

--查询入库结果

```
postgres=# select count(1) from test_copy_base_1;
count
-------
5000
(1 row)

postgres=# select c0,c22,c25 from test_copy_base_1 order by c0,c25 desc nulls last limit 5;
c0 | c22 | c25
----+----------+---------------------
1 | {0,-1,1} | 2022-06-08 12:12:12
2 | {0,-1,2} | 2022-06-09 12:12:12
3 | {0,-1,3} | 2022-06-10 12:12:12
4 | {0,-1,4} | 2022-06-11 12:12:12
5 | {0,-1,5} | 2022-06-12 12:12:12
(5 rows)
```

2、支持中文分隔符 --出库

```
postgres=#
COPY (select * from test_copy_base_1 order by c0) to '/tmp/test_copy_base_2.data' (format TEXT, delimiter '你好');
postgres=#
delete from test_copy_base_1 where 1=1;
```

--入库

```
postgres=#
-----使用错误分隔符会报错
COPY test_copy_base_1 from '/tmp/test_copy_base_2.data' with (format TEXT,delimiter '//');
ERROR: invalid input syntax for type numeric: "1你好02:02:02:02:02:02你好(2,2)你好南山1号你好178你好1你好987你好5.301
你好2022-06-08 12:12:12 +08:00你好00:09:02你好((1,1),(1,1),(1,1),(1,1))你好t你好["2022-06-07 13:12:12 +08:00" "2022-06-0
8 12:12:12 +08:00"]你好￥3.16你好(1,1)你好
CONTEXT: COPY test_copy_base_1, line 1, column c0: "1你好02:02:02:02:02:02你好(2,2)你好南山1号你好178你好1你好987你
好5.301你好2022-06...", nodetype:1(1:cn,0:dn)
postgres=#
COPY test_copy_base_1 from '/tmp/test_copy_base_2.data' with (format TEXT,delimiter '你好');
```

--查询入库结果

```
postgres=# select count(1) from test_copy_base_1;
count
-------
5000
(1 row)
```

创建和管理分区表
范围分区
创建时间范围分区表

最近更新时间: 2024-06-12 15:06:00

```
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
```

范围分区类型

最近更新时间: 2024-06-12 15:06:00

- 创建主分区

```
postgres=# create table t_native_range (f1 bigint,f2 timestamp default now(), f3 integer) partition by range ( f2 )distribu
te by shard(f1) to group default_group;
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');
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');
CREATE TABLE
```

- 默认分区表

``` cpp

--没有默认分区表时插入会出错
postgres=# insert into t_native_range values(2,'2017-08-01',2);
ERROR: node:dn001, backend_pid:32123, nodename:dn001,backend_pid:32123,message:no partition of relation "t_native_r
ange" found for row
DETAIL: Partition key of the failing row contains (f2) = (2017-08-01 00:00:00).

--添加默认分区表

postgres=# CREATE TABLE t_native_range_default PARTITION OF t_native_range DEFAULT;
CREATE TABLE
postgres=# insert into t_native_range values(2,'2017-08-01',2);
INSERT 0 1
```

- MAXVALUE分区

```
postgres=# CREATE TABLE t_native_range_maxvalue PARTITION OF t_native_range for values from ('2017-11-01') to (ma
xvalue);
CREATE TABLE
postgres=# insert into t_native_range values(1,'2017-11-01',1);
INSERT 0 1
postgres=# select * from t_native_range where f2='2017-11-01';
f1 | f2 | f3
----+---------------------+----
1 | 2017-11-01 00:00:00 | 1
```

```
(1 row)

postgres=# explain select * from t_native_range where f2='2017-11-01';
QUERY PLAN
------------------------------------------------------------------------------------
Remote Fast Query Execution (cost=0.00..0.00 rows=0 width=0)
Node/s: dn001, dn002
-> Append (cost=0.00..24.12 rows=6 width=20)
-> Seq Scan on t_native_range_maxvalue (cost=0.00..24.12 rows=6 width=20)
Filter: (f2 = '2017-11-01 00:00:00'::timestamp without time zone)
(5 rows)

postgres=# insert into t_native_range values(1,'2018-1-01',1);
INSERT 0 1
postgres=# explain select * from t_native_range where f2='2018-1-01';
QUERY PLAN
------------------------------------------------------------------------------------
Remote Fast Query Execution (cost=0.00..0.00 rows=0 width=0)
Node/s: dn001, dn002
-> Append (cost=0.00..24.12 rows=6 width=20)
-> Seq Scan on t_native_range_maxvalue (cost=0.00..24.12 rows=6 width=20)
Filter: (f2 = '2018-01-01 00:00:00'::timestamp without time zone)
(5 rows)

postgres=#
```

所有比2017-11-1大的数据都存储到子表t_native_range_maxvalue

- MINVALUE分区

```
postgres=# CREATE TABLE t_native_range_minvalue PARTITION OF t_native_range for values from (minvalue) to ('2017-09-01');
CREATE TABLE
postgres=# insert into t_native_range values(1,'2017-08-01',1);
INSERT 0 1
postgres=# explain SELECT * FROM t_native_range where f2='2017-08-01';
QUERY PLAN
------------------------------------------------------------------------------------
Remote Fast Query Execution (cost=0.00..0.00 rows=0 width=0)
Node/s: dn001, dn002
-> Append (cost=0.00..24.12 rows=6 width=20)
-> Seq Scan on t_native_range_minvalue (cost=0.00..24.12 rows=6 width=20)
Filter: (f2 = '2017-08-01 00:00:00'::timestamp without time zone)
(5 rows)
```

- 查看表结构

```
postgres=# \d+ t_native_range
Table"tbase_pg_proc.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 FORVALUES 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_maxvalue FOR VALUES FROM ('2017-11-01 00:00:00') TO(MAXVALUE),
t_native_range_minvalue FOR VALUES FROM (MINVALUE) TO ('2017-09-0100:00:00'),
t_native_range_default DEFAULT
Distribute By: SHARD(f1)
Location Nodes: ALL DATANODES创建主分区

postgres=# create table t_native_range (f1 bigint,f2 timestamp default now(), f3 integer) partition by range ( f2 )distribu
te by shard(f1) to group default_group;
CREATE TABLE
```

范围分区表使用

最近更新时间: 2024-06-12 15:06:00

```
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

#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=#
```

列表分区
列表分区使用

最近更新时间: 2024-06-12 15:06:00

以下示例说明列表分区使用

- 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 sha
rd(f1) to group default_group;
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');
CREATE TABLE

postgres=# create table t_list_bj partition of t_native_list(f1 ,f2 , f3,f4) for values in ('北京');
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=#
```

- 创建default分区

```
#没有default分区情况下会出错，插入会出错

postgres=# insert into t_native_list values(1,'上海',1,current_date);
ERROR: node:dn001, backend_pid:31664, nodename:dn001,backend_pid:31664,message:no partition of relation "t_nativ
e_list" found for row
DETAIL: Partition key of the failing row contains (f2) = (上海).

postgres=# CREATE TABLE t_native_list_default PARTITION OF t_native_list DEFAULT;
```

```
CREATE TABLE

#创建后就能正常插入
postgres=# insert into t_native_list values(1,'上海',1,current_date);
INSERT 0 1
postgres=#
```

散列分区
散列分区使用

最近更新时间: 2024-06-12 15:06:00

以下示例说明散列分区使用

```
postgres=# create table t_hash_partition(f1 int,f2 int) partition by hash(f2);
create table t_hash_partition_1 partition of t_hash_partition FOR VALUES WITH(MODULUS 4, REMAINDER 0);
create table t_hash_partition_2 partition of t_hash_partition FOR VALUES WITH(MODULUS 4, REMAINDER 1);
create table t_hash_partition_3 partition of t_hash_partition FOR VALUES WITH(MODULUS 4, REMAINDER 2);
create table t_hash_partition_4 partition of t_hash_partition FOR VALUES WITH(MODULUS 4, REMAINDER 3);

#上面创建4个子分区的hash表，hash分区表需要指定分区的个数，因为使用分区数做为算子来计算每条数据所在分区，所以目前hash分区不支持删除添加和删除。

postgres=# insert into t_hash_partition values(1,1),(2,2),(3,3);
COPY 3
postgres=# select * from t_hash_partition;
f1 | f2
----+----
1 | 1
3 | 3
2 | 2
(3 rows)

#TDSQL pg自动根据分区值进行剪枝

postgres=# explain select * from t_hash_partition where f2=2;
QUERY PLAN
------------------------------------------------------------------------------
Remote Fast Query Execution (cost=0.00..0.00 rows=0 width=0)
Node/s: dn001, dn002
-> Append (cost=0.00..26.88 rows=7 width=8)
-> Seq Scan on t_hash_partition_3 (cost=0.00..26.88 rows=7 width=8)
Filter: (f2 = 2)
(5 rows)
```

多级分区
多级分区表使用

最近更新时间: 2024-06-12 15:06:00

- 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 ) distr
ibute 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 ('201
7-01-01') 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 ('201
7-02-01') 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
```

TDSQL PG支持存在1级和2级分区混用，大家不需要都平级。

添加分区子表

最近更新时间: 2024-06-12 15:06:00

```
postgres=# \d+ t1_pt
Table "public.t1_pt"
Column | Type | Collation | Nullable | Default | Storage | Stats target | Description
--------+-----------------------------+-----------+----------+---------+----------+--------------+-------------
f1 | integer | | not null | | plain | |
f2 | timestamp without time zone | | not null | | plain | |
f3 | character varying(20) | | | | extended | |
Indexes:
"t1_pt_pkey" PRIMARY KEY, btree (f1)
Distribute By: SHARD(f1)
Location Nodes: dn01
Partition By: RANGE(f2)
# Of Partitions: 3
Start With: 2019-01-01
Interval Of Partition: 1 MONTH

postgres=# ALTER TABLE t1_pt ADD PARTITIONS 2;

ALTER TABLE
postgres=# \d+ t1_pt
Table "public.t1_pt"
Column | Type | Collation | Nullable | Default | Storage | Stats target | Description
--------+-----------------------------+-----------+----------+---------+----------+--------------+-------------
f1 | integer | | not null | | plain | |
f2 | timestamp without time zone | | not null | | plain | |
f3 | character varying(20) | | | | extended | |
Indexes:
"t1_pt_pkey" PRIMARY KEY, btree (f1)
Distribute By: SHARD(f1)
Location Nodes: dn01
Partition By: RANGE(f2)
# Of Partitions: 5
Start With: 2019-01-01
Interval Of Partition: 1 MONTH
```

访问子分区

最近更新时间: 2024-06-12 15:06:00

使用parent_table PARTITION (part_name) 直接访问子分区 示例： 访问范围分区

```
select * from part_range_t2partition(part_3) where rownum<=3;
```

访问列表分区

```
select * from part_list_t1 partition(part_3) where rownum<=3;
```

truncate分区

最近更新时间: 2024-06-12 15:06:00

使用 alter table ... truncate...删除分区 示例：

```
alter table part_range_t2 truncate partition(part_3) update global indexes;
```

查询数据

```
select * from part_range_t2 partition (part_3);
```

自研分区

最近更新时间: 2024-06-12 15:06:00

在多分区表情况下，自研分区表性能更好。自研分区表随着分区的增多，性能不能不会下降，在2048个分区的情况下，自研分区表的剪技访问性能是普通分区（范围、列表、散列）表的30倍。 示例： 创建自研分区表（2048个分区）：

```
create table t_time_range
(
f1 bigint not null, f2 timestamp ,f3 bigint
)
partition by range (f2) begin (timestamp without time zone '2017-09-01 0:0:0')
step (interval '1 month')
partitions (2048) distribute by shard(f1)
to group default_group;
```

向分区表中插入数据

```
select 'insert into t_time_range values(1,'''||startyf||''',1);' from (select ('2017-09-01'::date + (t::text||' months')::interval)::dat
e::text as startyf from generate_series(0,2047) as t) as t;
```

冷热分区表

最近更新时间: 2024-06-12 15:06:00

以下说明冷热分区的场景及使用。

```
postgres=# create table t_cold_hot_table(f1 bigint, f2 timestamp ,f3 bigint) partition by range (f2) begin (timestamp witho
ut time zone '2017-09-01 0:0:0') step (interval '1 month') partitions (12) distribute by shard(f1,f2) to group default_group e
xt_group;
CREATE TABLE

postgres=# \d+ t_cold_hot_table
Table "pgxz.t_cold_hot_table"
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,f2)
Hotnodes:dn02, dn01 Coldnodes:dn03, dn04
Partition By: RANGE(f2)
# Of Partitions: 12
Start With: 2017-09-01
Interval Of Partition: 1 MONTH
```

> **注意：**
>
> 创建时间范围冷热分区表需要有两个group，冷数据的ext_group对应的节点dn03和dn04需要标识为冷节点，如下所示：

```
postgres=# select pg_set_node_cold_access();
pg_set_node_cold_access
-------------------------
success
(1 row)

postgres=# \q
[pgxz@VM_0_29_centos install]$ psql -p 23004
psql (PGXC , based on PG 9.4beta1)
Type "help" for help.

postgres=# select pg_set_node_cold_access();
pg_set_node_cold_access
-------------------------
success
(1 row)

postgres=#
```

> **注意：**
>
> 使用冷热分区表需要在postgresql.conf中配置冷热分区时间参数，如下所示：

```
manual_hot_date = '2017-12-01
```

truncate分区表

最近更新时间: 2024-06-12 15:06:00

- truncate一个时间分区表

```
postgres=# \d+ t_time_range

Table"pgxz.t_time_range"
Column | Type | Modifiers | Storage | Statstarget | 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 | TDSQL PG

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 | Statstarget | 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=# truncatet_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=#
```

部分分区表示例
partition访问子分区

最近更新时间: 2024-06-12 15:06:00

```
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;
CREATE TABLE
postgres=# create table t_native_range_201709 partition of t_native_range for values from ('2017-09-01') to ('2017-10-01');
CREATE TABLE
postgres=# create table t_native_range_201710 partition of t_native_range for values from ('2017-10-01') to ('2017-11-01');
CREATE TABLE
postgres=# insert into t_native_range values(1,'2017-09-01',1);
INSERT 0 1
postgres=# insert into t_native_range values(2,'2017-09-01',2);
INSERT 0 1
postgres=# select * from t_native_range partition(t_native_range_201709);
f1 | f2 | f3
----+---------------------+----
1 | 2017-09-01 00:00:00 | 1
2 | 2017-09-01 00:00:00 | 2
(2 rows)
postgres=#
```

支持创建分区表时将默认分区中属于新分区的数据转移到新的分区表中

最近更新时间: 2024-06-12 15:06:00

```
postgres=# CREATE TABLE t_native_range_default PARTITION OF t_native_range DEFAULT;
CREATE TABLE
postgres=# insert into t_native_range values(2,'2017-08-01',2);
INSERT 0 1
postgres=# select * from t_native_range_default;
f1 | f2 | f3
----+---------------------+----
2 | 2017-08-01 00:00:00 | 2
(1 row)
postgres=# create table t_native_range_201708 partition of t_native_range for values from ('2017-08-01') to ('2017-09-0
1');
CREATE TABLE
postgres=# select * from t_native_range_201708;
f1 | f2 | f3
----+---------------------+----
2 | 2017-08-01 00:00:00 | 2
(1 row)
postgres=# select * from t_native_range_default;
f1 | f2 | f3
----+----+----
(0 rows)
postgres=#
```

TDSQ了PG实例自动转换。

truncate partition子分区

最近更新时间: 2024-06-12 15:06:00

```
postgres=# alter table t_native_range truncate partition (t_native_range_201708) update indexes;
TRUNCATE TABLE
postgres=# select * from t_native_range_201708;
f1 | f2 | f3
----+----+----
(0 rows)
```

此语法支持自研分区表、社区分区表（range , list, hash ）。

分区拆分

最近更新时间: 2024-06-12 15:06:00

```
postgres=# \d+ t_native_range
Table "tbase_pg_proc.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_201708 FOR VALUES FROM ('2017-08-01 00:00:00') TO ('2017-09-01 00:00:00'),
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_minvalue FOR VALUES FROM (MINVALUE) TO ('2017-08-01 00:00:00'),
t_native_range_default DEFAULT
Distribute By: SHARD(f1)
Location Nodes: ALL DATANODES
将t_native_range_minvalue拆分成
(MINVALUE) TO ('2017-07-01 00:00:00')
('2017-07-01 00:00:00') TO ('2017-08-01 00:00:00')
postgres=# alter table t_native_range split partition t_native_range_minvalue at ('2017-07-01') into (partition t_native_rang
e_minvalue_201706, partition t_native_range_201707);
ALTER TABLE
postgres=# \d+ t_native_range
Table "tbase_pg_proc.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_201707 FOR VALUES FROM ('2017-07-01 00:00:00') TO ('2017-08-01 00:00:00'),
t_native_range_201708 FOR VALUES FROM ('2017-08-01 00:00:00') TO ('2017-09-01 00:00:00'),
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_minvalue_201706 FOR VALUES FROM (MINVALUE) TO ('2017-07-01 00:00:00'),
t_native_range_default DEFAULT
Distribute By: SHARD(f1)
Location Nodes: ALL DATANODES
```

分区合并

最近更新时间: 2024-06-12 15:06:00

```
postgres=# alter table t_native_range merge partitions (t_native_range_minvalue_201706, t_native_range_201707) into par
tition t_native_range_minvalue_201707;
ALTER TABLE
postgres=# \d+ t_native_range
Table "tbase_pg_proc.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_201708 FOR VALUES FROM ('2017-08-01 00:00:00') TO ('2017-09-01 00:00:00'),
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_minvalue_201707 FOR VALUES FROM (MINVALUE) TO ('2017-08-01 00:00:00'),
t_native_range_default DEFAULT
Distribute By: SHARD(f1)
Location Nodes: ALL DATANODES
```

支持不同父表下相同的子表名称

最近更新时间: 2024-06-12 15:06:00

- 分区表1。

```
postgres=# create table t_same_child_partition_name_1 (f1 bigint,f2 timestamp default now(), f3 integer) partition by ra
nge ( f2 ) distribute by shard(f1) to group default_group;
CREATE TABLE
postgres=# create table t_same_child_partition_name_1_201709 partition of t_same_child_partition_name_1 for values fr
om ('2017-09-01') to ('2017-10-01') as part_1;
CREATE TABLE
postgres=# insert into t_same_child_partition_name_1 values(1,'2017-09-01',1);
INSERT 0 1
```

- 分区表2。

```
postgres=# create table t_same_child_partition_name_2 (f1 bigint,f2 timestamp default now(), f3 integer) partition by ra
nge ( f2 ) distribute by shard(f1) to group default_group;
CREATE TABLE
postgres=#
postgres=# create table t_same_child_partition_name_2_201709 partition of t_same_child_partition_name_2 for values fr
om ('2017-09-01') to ('2017-10-01') as part_1;
CREATE TABLE
postgres=#
postgres=# insert into t_same_child_partition_name_2 values(1,'2017-09-01',1);
INSERT 0 1
postgres=#
```

- 查询。

```
postgres=# select * from t_same_child_partition_name_1 partition(part_1);
f1 | f2 | f3
----+---------------------+----
1 | 2017-09-01 00:00:00 | 1
(1 row)

postgres=# select * from t_same_child_partition_name_2 partition(part_1);
f1 | f2 | f3
----+---------------------+----
1 | 2017-09-01 00:00:00 | 1
(1 row)
postgres=#
```

创建和管理视图
创建视图
创建视图示例

最近更新时间: 2024-06-12 15:06:00

```
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 tomc_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;
```

修改视图

最近更新时间: 2024-06-12 15:06:00

通过create or replace view命令修改视图。 示例： 1）修改原指向t_range的视图指向t_list

```
create or replace view t_range_view as select * from t_list;
```

删除视图

删除视图示例

最近更新时间: 2024-06-12 15:06:00

```
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
```

物化视图
创建物化视图

最近更新时间: 2024-06-12 15:06:00

物化视图是一种特殊的物理表，“物化”(Materialized)视图是相对普通视图而言的。普通视图是虚拟表，应用的局限性大，任何对视图的查询，都实际上转换为视图SQL语句的查询。这样对整体查询性能的提高，并没有实质上的好处。而物化视图是一张实际存在的表，是占有数据库磁盘空间的。 物化视图并不像普通视图那样，只有在使用的时候才去读取数据，而是预先计算并保存表连接或者聚集等比较耗时操作的结果，这样大大提高了读取的速度，特别适合抽取大数据量表的某些信息。 通过create MATERIALIZED VIEW创建物化视图 示例：

```
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)
```

访问物化视图

最近更新时间: 2024-06-12 15:06:00

```
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)
```

增量数据刷新

最近更新时间: 2024-06-12 15:06:00

```
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。

视图触发器创建与删除
创建视图触发器

最近更新时间: 2024-06-12 15:06:00

```
#创建视图
postgres=# create or replace view t1_view as select * from t1;
CREATE VIEW
postgres=#

#创建触发器函数
postgres=# CREATE OR REPLACE FUNCTION t1_view_insert_trigger_func () RETURNS trigger AS
$body$
BEGIN
INSERT INTO t1 values(NEW.ID,NEW.nickname);
RETURN NEW;
END;
$body$
LANGUAGE 'plpgsql';
CREATE FUNCTION
postgres=#
#创建触发器
postgres=# CREATE TRIGGER t1_view_insert_trigger INSTEAD OF INSERT ON t1_view FOR EACH ROW EXECUTE PROCEDU
RE t1_view_insert_trigger_func();
CREATE TRIGGER
#验证
postgres=# INSERT INTO t1_view values(2,'TDSQL PG');
INSERT 0 1
postgres=# select * from t1;
id | nickname
----+----------
1 | TDSQL PG
2 | TDSQL PG
(2 rows)
```

删除视图触发器

最近更新时间: 2024-06-12 15:06:00

```
postgres=# drop trigger t1_view_insert_trigger on t1_view ;
DROP TRIGGER
postgres=#
```

创建和管理序列
自增列与序列的用法
序列的创建和访问

最近更新时间: 2024-06-12 15:06:00

1. 建立序列。

```
postgres=# create sequence tbase_seq;
CREATE SEQUENCE
```

2. 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
```

3. 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
```

4. 获取序列的下一个值。

```
postgres=# select nextval('tbase_seq');
nextval
---------
1
```

5. 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
```

6. 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
```

7. 设置序列当前值。

```
postgres=# select setval('tbase_seq',1);
setval
--------
1
```

序列在DML中使用

最近更新时间: 2024-06-12 15:06:00

```
postgres=# create table t(id int ,nickname varchar(15));
postgres=# INSERT INTO t (id,nickname) VALUES(nextval('tbase_seq'),'TBase好');
postgres=# select * from t;
id | nickname
----+-----------
1 | 腾讯TDSQL PG
2 | TDSQL PG好
(2 rows)
```

序列做为字段的默认值使用

最近更新时间: 2024-06-12 15:06:00

```
postgres=# alter table t alter column id set default nextval('tbase_seq');
postgres=# INSERT INTO t (nickname) VALUES('hello TDSQL PG');
INSERT 0 1
postgres=# select * from t;
id | nickname
----+-------------
3 | hello TDSQL PG
1 | 腾讯TDSQL PG
2 | TDSQL PG好
(3 rows)
```

序列做为字段类型使用

最近更新时间: 2024-06-12 15:06:00

```
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 TDSQL PG');
INSERT 0 1
postgres=# select * from t;
id | nickname
----+-------------
1 | hello TDSQL PG
(1 row)
```

删除序列

最近更新时间: 2024-06-12 15:06:00

```
postgres=# drop sequence tbase_seq;
DROP SEQUENCE
```

1. 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
```

查询gtm中的序列对象列表

最近更新时间: 2024-06-12 15:06:00

```
postgres=# select * from pg_list_storage_sequence();
```

创建和管理同义词
部分同义词示例
创建同义词

最近更新时间: 2024-06-12 15:06:00

```
postgres=# create table public.syn(f1 int,f2 text);
CREATE TABLE
postgres=# insert into public.syn values(1,'TDSQL PG');
INSERT 0 1
postgres=# create synonym syn_copy for public.syn;
CREATE SYNONYM
```

同义词访问

最近更新时间: 2024-06-12 15:06:00

```
postgres=# select * from syn_copy;
f1 | f2
----+-------
1 | TDSQL PG
(1 row)
postgres=# insert into syn_copy values(2,'pg');
INSERT 0 1
postgres=# update syn_copy set f2='postgres' where f1=2;
UPDATE 1
postgres=# delete from syn_copy where f1=2;
DELETE 1
```

同义词系统表

最近更新时间: 2024-06-12 15:06:00

```
postgres=# select * from pg_synonym ;
synname | synnamespace | objnamespace | objname | objdblink | synowner
----------+--------------+--------------+---------+-----------+----------
syn_copy | 19509 | public | syn | | 10
(1 row)
postgres=#
```

关于PL/SQL
PL/SQL介绍

最近更新时间: 2024-06-12 15:06:00

PL / SQL是SQL的过程扩展，是一种可移植的高性能事务处理语言，PL/SQL把数据操作和查询语句组织在过程化代码中，通过逻辑判断、循环等操作实现高级复杂功能。 PL/SQL具有高性能、可移植、可扩展、便于管理等优点。

PL/SQL控制语句
条件选择语句

最近更新时间: 2024-06-12 15:06:00

以下几个示例，说明条件选择语句的类型以及使用方法。

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)
```

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)
```

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)
```

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)
```

循环语句

最近更新时间: 2024-06-12 15:06:00

以下几个示例，说明循环语句的类型以及使用方法。

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

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)
```

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

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,TDSQL PG)
NOTICE: (2,pgxz)
f27
-----

(1 row)
```

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

其他控制语句

最近更新时间: 2024-06-12 15:06:00

以下几个示例，说明其他控制语句的类型以及使用方法。

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)
```

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TDSQLPG
f27
-----
```

```
(1 row)

postgres=# SELECT f27(3);
NOTICE: 查不到记录
f27
-----

(1 row)
```

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)
```

PL/SQL集合和记录
集合概述

最近更新时间: 2024-06-12 15:06:00

集合是存储过程复合变量，可以按顺序存储多个类型相同的元素，类似于一维数组。整个集合可以作为子程序的参数进行传递（如果发送或者接收的子程序都不是独立的子程序）。 集合中的元素有唯一的下标描述它在集合中的位置。访问集合的元素，可以使用下标命名方式：集合名(下标)。 集合方法是内建的存储过程，可以返回集合的信息或者对集合进行操作。需要使用"."表示调用集合，格式为：集合名.方法名。例如，集合名.COUNT 方法用于返回该集合元素的数量。 TDSQL PG支持三种集合类型：关联数组，可变数组和嵌套表。 目前，关联数组，嵌套表和可变数组只能在plpgsql定义的function、procdure、package、block中使用。支持select bulk collect into 到对应的变量中。

关联数组
关联数组定义和初始化

最近更新时间: 2024-06-12 15:06:00

示例： 由不同类型索引的关联数组类型

```
set enable_oracle_compatible=on;
drop table dept cascade;
CREATE TABLE DEPT
(DEPTNO NUMBER(2) CONSTRAINT PK_DEPT PRIMARY KEY,
DNAME VARCHAR2(14) ,
LOC VARCHAR2(13) ) ;
drop table emp cascade;
CREATE TABLE EMP
(EMPNO NUMBER(4) CONSTRAINT PK_EMP PRIMARY KEY,
ENAME VARCHAR2(10),
JOB VARCHAR2(9),
MGR NUMBER(4),
HIREDATE DATE,
SAL NUMBER(7,2),
COMM NUMBER(7,2),
DEPTNO NUMBER(2));
INSERT INTO DEPT VALUES
(10,'ACCOUNTING','NEW YORK');
INSERT INTO DEPT VALUES (20,'RESEARCH','DALLAS');
INSERT INTO DEPT VALUES
(30,'SALES','CHICAGO');
INSERT INTO DEPT VALUES
(40,'OPERATIONS','BOSTON');
INSERT INTO EMP VALUES
(7369,'SMITH','CLERK',7902,to_date('17-12-1980','dd-mm-yyyy'),800,NULL,20);
INSERT INTO EMP VALUES
(7499,'ALLEN','SALESMAN',7698,to_date('20-2-1981','dd-mm-yyyy'),1600,300,30);
INSERT INTO EMP VALUES
(7521,'WARD','SALESMAN',7698,to_date('22-2-1981','dd-mm-yyyy'),1250,500,30);
INSERT INTO EMP VALUES
(7566,'JONES','MANAGER',7839,to_date('2-4-1981','dd-mm-yyyy'),2975,NULL,20);
INSERT INTO EMP VALUES
(7654,'MARTIN','SALESMAN',7698,to_date('28-9-1981','dd-mm-yyyy'),1250,1400,30);
INSERT INTO EMP VALUES
(7698,'BLAKE','MANAGER',7839,to_date('1-5-1981','dd-mm-yyyy'),2850,NULL,30);
INSERT INTO EMP VALUES
(7782,'CLARK','MANAGER',7839,to_date('9-6-1981','dd-mm-yyyy'),2450,NULL,10);
INSERT INTO EMP VALUES
(7788,'SCOTT','ANALYST',7566,to_date('19-04-1987','dd-mm-yyyy'),3000,NULL,20);
INSERT INTO EMP VALUES
(7839,'KING','PRESIDENT',NULL,to_date('17-11-1981','dd-mm-yyyy'),5000,NULL,10);
INSERT INTO EMP VALUES
(7844,'TURNER','SALESMAN',7698,to_date('8-9-1981','dd-mm-yyyy'),1500,0,30);
INSERT INTO EMP VALUES
(7876,'ADAMS','CLERK',7788,to_date('23-05-1987', 'dd-mm-yyyy'),1100,NULL,20);
INSERT INTO EMP VALUES
(7900,'JAMES','CLERK',7698,to_date('3-12-1981','dd-mm-yyyy'),950,NULL,30);
```

```
INSERT INTO EMP VALUES
(7902,'FORD','ANALYST',7566,to_date('3-12-1981','dd-mm-yyyy'),3000,NULL,20);
INSERT INTO EMP VALUES
(7934,'MILLER','CLERK',7782,to_date('23-1-1982','dd-mm-yyyy'),1300,NULL,10);
drop table bonus cascade;
CREATE TABLE BONUS
(
ENAME VARCHAR2(10) ,
JOB VARCHAR2(9) ,
SAL NUMBER,
COMM NUMBER
) ;
drop table salgrade cascade;
CREATE TABLE SALGRADE
( GRADE NUMBER,
LOSAL NUMBER,
HISAL NUMBER );
INSERT INTO SALGRADE VALUES (1,700,1200);
INSERT INTO SALGRADE VALUES (2,1201,1400);
INSERT INTO SALGRADE VALUES (3,1401,2000);
INSERT INTO SALGRADE VALUES (4,2001,3000);
INSERT INTO SALGRADE VALUES (5,3001,9999);

-- TestPoint : Associative array with different index type

DECLARE
TYPE aa_type IS TABLE OF INTEGER INDEX BY PLS_INTEGER;
TYPE bb_type IS TABLE OF number(6,2) INDEX BY VARCHAR2(2);
TYPE cc_type IS TABLE OF char(5) INDEX BY VARCHAR(2);
TYPE dd_type IS TABLE OF date INDEX BY long;

aa aa_type; -- associative array
bb bb_type;
cc cc_type;
dd dd_type;
BEGIN
aa(1):=3;
aa(2):=6;
perform dbms_output.serveroutput('t');
--raise notice '%,%',aa.count,aa.limit;
perform DBMS_OUTPUT.PUT('aa.COUNT = ');
perform DBMS_OUTPUT.PUT_LINE(NVL(TO_CHAR(aa.COUNT), 'NULL'));

perform DBMS_OUTPUT.PUT('aa.LIMIT = ');
perform DBMS_OUTPUT.PUT_LINE(NVL(to_char(aa.LIMIT), 'NULL'));

bb('a'):=1.1;
bb('b'):=2.222;
raise notice '%',bb.count;
raise notice '%,%',bb('a'),bb('b');
cc('a'):='chqin';
cc('b'):='tbase';
raise notice '%',bb.count;
raise notice '%,%',cc('a'),cc('b');
dd('a'):=to_date('2021-09-23 15:12:55','yyyy-mm-dd hh24:mi:ss');
dd('b'):=to_date('2021-09-24 15:12:55','yyyy-mm-dd hh24:mi:ss');
raise notice '%',bb.count;
```

```
raise notice '%,%',dd('a'),dd('b');

end;
/

-- TestPoint : Associative Array index type long

declare
TYPE population IS TABLE OF long INDEX BY long;
myvar population; -- Associative array variable
i long; -- Scalar variable

BEGIN
raise notice 'Begin Count=%', myvar.COUNT;
myvar('v1') := 'v1';
myvar('v2') := 'v2';
raise notice 'Initialed Count=%', myvar.COUNT;

raise notice 'Print first to last';
i := myvar.FIRST();
WHILE i IS NOT NULL LOOP
raise notice 'Loop[%] is %',i, myvar(i);
i := myvar.NEXT(i);
END LOOP;
end;
/

-- TestPoint : Associative Array index type varchar

declare
TYPE population IS TABLE OF long raw INDEX BY varchar(20);
myvar population; -- Associative array variable
i varchar(20); -- Scalar variable
BEGIN
raise notice 'Begin Count=%', myvar.COUNT;

myvar('1') := '2000';
myvar('2') := '750000';
myvar('3') := '1000000';
myvar('4') := '2001';
myvar('5') := '20210617';
myvar('10') := '20210618';

raise notice 'Initialed Count=%', myvar.COUNT;

raise notice 'Print first to last';
i := myvar.FIRST();
WHILE i IS NOT NULL LOOP
raise notice 'Loop[%] is %',i, myvar(i);
i := myvar.NEXT(i);
END LOOP;

IF myvar.EXISTS('12') THEN
raise notice 'myvar(''12'') is exist';
ELSE
raise notice 'myvar(''12'') is not exist';
END IF;
```

```
myvar('12') := '1001';
raise notice 'INSERT (''12'') = 1001, COUNT=%', myvar.COUNT;

IF myvar.EXISTS('12') THEN
raise notice 'myvar(''12'') is exist';
ELSE
raise notice 'myvar(''12'') is not exist';
END IF;

myvar('20') := '30006';

raise notice 'Print last to first';
i := myvar.LAST;
WHILE i IS NOT NULL LOOP
raise notice 'Loop[%] is %',i, myvar(i);
i := myvar.PRIOR(i);
END LOOP;

raise notice 'End Count=%', myvar.COUNT;
END;
/


-- TestPoint : Associative Array : nclob type index of PLS_INTEGER

declare
TYPE population IS TABLE OF nclob INDEX BY PLS_INTEGER;
myvar population; -- Associative array variable
i PLS_INTEGER; -- Scalar variable
BEGIN
raise notice 'Begin Count=%', myvar.COUNT;

myvar(1) := '2000';
myvar(2) := '750000';
myvar(3) := '1000000';
myvar(4) := '2001';
myvar(5) := '20210617';
myvar(10) := '20210618';

raise notice 'Initialed Count=%', myvar.COUNT;

raise notice 'Print first to last';
i := myvar.FIRST();
WHILE i IS NOT NULL LOOP
raise notice 'Loop[%] is %',i, myvar(i);
i := myvar.NEXT(i);
END LOOP;

IF myvar.EXISTS(12) THEN
raise notice 'myvar(''12'') is exist';
ELSE
raise notice 'myvar(''12'') is not exist';
END IF;

myvar(12) := '1001';
raise notice 'INSERT (''12'') = 1001, COUNT=%', myvar.COUNT;
```

```
IF myvar.EXISTS(12) THEN
raise notice 'myvar(''12'') is exist';
ELSE
raise notice 'myvar(''12'') is not exist';
END IF;

myvar(20) := '30006';

raise notice 'Print last to first';
i := myvar.LAST;
WHILE i IS NOT NULL LOOP
raise notice 'Loop[%] is %',i, myvar(i);
i := myvar.PRIOR(i);
END LOOP;

raise notice 'End Count=%', myvar.COUNT;
END;
/
declare
v1 string(20);
begin
v1:='v';
end;
/


-- TestPoint : Associative Array : emp.ename%type type index of PLS_INTEGER

declare
TYPE population IS TABLE OF emp.ename%type INDEX BY PLS_INTEGER;
myvar population; -- Associative array variable
i PLS_INTEGER; -- Scalar variable
v1 emp.ename%type;
BEGIN
select ename into v1 from emp where empno=7788;
myvar(1):=v1;
select ename into v1 from emp where empno=7521;
myvar(2):=v1;
raise notice '%',myvar(1);
i := myvar.FIRST();
WHILE i IS NOT NULL LOOP
raise notice 'Loop[%] is %',i, myvar(i);
i := myvar.NEXT(i);
END LOOP;

end;
/

-- TestPoint : Associative Array : emp.hiredate%type type index of varchar2(20)
-- ERROR: input value length is 40; too long for type varchar2(10)

declare
TYPE population IS TABLE OF emp.hiredate%type INDEX BY varchar2(10);
myvar population; -- Associative array variable
-- i varchar2(10); -- Scalar variable
i emp.ename%type;
```

```
v1 emp.ename%type;
v2 emp.hiredate%type;
BEGIN
raise notice 'Begin Count=%', myvar.COUNT;
select ename,hiredate into v1,v2 from emp where empno=7788;
myvar(v1) := v2;
select ename,hiredate into v1,v2 from emp where empno=7521;
myvar(v1) := v2;
raise notice 'Initialed Count=%', myvar.COUNT;

raise notice 'Print first to last';
i := myvar.FIRST();
WHILE i IS NOT NULL LOOP
raise notice 'Loop[%] is %',i, myvar(i);
i := myvar.NEXT(i);
END LOOP;

IF myvar.EXISTS('12') THEN
raise notice 'myvar(''12'') is exist';
ELSE
raise notice 'myvar(''12'') is not exist';
END IF;

raise notice 'Print last to first';
i := myvar.LAST;
WHILE i IS NOT NULL LOOP
raise notice 'Loop[%] is %',i, myvar(i);
i := myvar.PRIOR(i);
END LOOP;

END;
/


-- TestPoint : Associative Array with table columns: varchar2 index by varchar2

drop table if exists chqin;
create table chqin (f1 varchar2(10), f2 varchar2(20));
insert into chqin values('1','tbase1');
insert into chqin values('2','tbase2');

declare
TYPE population IS TABLE OF chqin.f2%type INDEX BY varchar2(20);
myvar population; -- Associative array variable
i varchar2(10); -- Scalar variable
v1 chqin.f1%type;
v2 chqin.f2%type;
BEGIN
raise notice 'Begin Count=%', myvar.COUNT;
select f1,f2 into v1,v2 from chqin where f1='1';
myvar(v1) := v2;
select f1,f2 into v1,v2 from chqin where f1='2';
myvar(v1) := v2;
raise notice 'Initialed Count=%', myvar.COUNT;

raise notice 'Print first to last';
i := myvar.FIRST();
```

```
WHILE i IS NOT NULL LOOP
raise notice 'Loop[%] is %',i, myvar(i);
i := myvar.NEXT(i);
END LOOP;

IF myvar.EXISTS('12') THEN
raise notice 'myvar(''12'') is exist';
ELSE
raise notice 'myvar(''12'') is not exist';
END IF;

raise notice 'Print last to first';
i := myvar.LAST;
WHILE i IS NOT NULL LOOP
raise notice 'Loop[%] is %',i, myvar(i);
i := myvar.PRIOR(i);
END LOOP;

END;
/
drop table if exists chqin;


-- TestPoint : Associative Array with table columns

drop table if exists chqin;
create table chqin (f1 varchar2(10), f2 long,f3 nclob,f4 interval year to month,f5 timestamp(6) with local time zone,f6 raw
(8),f7 nchar(6));
insert into chqin values('1','tbase1','tbase1',interval '3-2' year to month, '2021-06-22 16:02:07.067',utl_raw.cast_to_raw('raw
1'),'tbase1');
insert into chqin values('2','tbase2','tbase2',interval '4-2' year to month, '2021-07-22 16:02:07.067',utl_raw.cast_to_raw('raw
2'),'tbase2');


-- TestPoint : Associative Array: varchar2 index by long --core dump
declare
TYPE population IS TABLE OF chqin.f1%type INDEX BY long;
myvar population; -- Associative array variable
i long; -- Scalar variable
v1 chqin.f1%type;
v2 chqin.f2%type;
BEGIN
raise notice 'Begin Count=%', myvar.COUNT;
select f1,f2 into v1,v2 from chqin where f1='1';
myvar(v2) := v1;
select f1,f2 into v1,v2 from chqin where f1='2';
myvar(v2) := v1;
raise notice 'Initialed Count=%', myvar.COUNT;

raise notice 'Print first to last';
i := myvar.FIRST();
WHILE i IS NOT NULL LOOP
raise notice 'Loop[%] is %',i, myvar(i);
i := myvar.NEXT(i);
END LOOP;
end;
/
```

```
-- TestPoint : Associative Array: nclob index by long --core

declare
TYPE population IS TABLE OF chqin.f3%type INDEX BY long;
myvar population; -- Associative array variable
i long; -- Scalar variable
v1 chqin.f1%type;
v2 chqin.f2%type;
BEGIN
raise notice 'Begin Count=%', myvar.COUNT;
select f3,f2 into v1,v2 from chqin where f1='1';
myvar(v2) := v1;
select f3,f2 into v1,v2 from chqin where f1='2';
myvar(v2) := v1;
raise notice 'Initialed Count=%', myvar.COUNT;

raise notice 'Print first to last';
i := myvar.FIRST();
WHILE i IS NOT NULL LOOP
raise notice 'Loop[%] is %',i, myvar(i);
i := myvar.NEXT(i);
END LOOP;
end;
/
```

关联数组的使用

最近更新时间: 2024-06-12 15:06:00

示例： 关联数组在包中的使用

```
drop package My_Types;
CREATE OR REPLACE PACKAGE My_Types AUTHID CURRENT_USER IS
TYPE My_AA IS TABLE OF VARCHAR2(20) INDEX BY PLS_INTEGER;
FUNCTION Init_My_AA RETURN My_AA;
END My_Types;
/
CREATE OR REPLACE PACKAGE BODY My_Types IS
FUNCTION Init_My_AA RETURN My_AA IS
Ret My_AA;
BEGIN
Ret(-10) := '-ten';
Ret(0) := 'zero';
Ret(1) := 'one';
Ret(2) := 'two';
Ret(3) := 'three';
Ret(4) := 'four';
Ret(9) := 'nine';
RETURN Ret;
END Init_My_AA;
END My_Types;
/

DECLARE
v CONSTANT My_Types.My_AA := My_Types.Init_My_AA();
BEGIN
DECLARE
Idx PLS_INTEGER := v.FIRST();
BEGIN
WHILE Idx IS NOT NULL LOOP
raise notice '%,%',Idx,v(Idx);

-- DBMS_OUTPUT.PUT_LINE(TO_CHAR(Idx, '999')||LPAD(v(Idx), 7));
Idx := v.NEXT(Idx);
END LOOP;
END;
END;
/
```

可变数组

最近更新时间: 2024-06-12 15:06:00

可变数组是一个其元素数可以从零（空）到声明的最大大小变化的数组。当创建可变数组时，必须指定可变数组的最大长度。 要访问varray变量的元素，请使用语法variable_name(index)。index的下限是1;上限是当前的元素数。 示例：

```
set enable_oracle_compatible=on;
--匿名块中使用varray，使用dbms_output 输出array中的数据
declare
type v_arr is varray(10) of varchar2(30);
my_arr v_arr:= v_arr('1','2','3');
begin
for i in 1..my_arr.count
loop
perform dbms_output.serveroutput('t');
perform dbms_output.put_line(my_arr[i]);
end loop;
end;
/

--初始化1个字符后，输出溢出值和正常值
declare
type v_arr is varray(3) of varchar2(30);
my_arr v_arr:= v_arr(null);
begin
my_arr[1]:='插入';
perform dbms_output.serveroutput('t');
perform DBMS_OUTPUT.PUT_LINE(my_arr[1]);
perform dbms_output.serveroutput(my_arr[2]);
exception WHEN others THEN
BEGIN
perform dbms_output.serveroutput('t');
perform DBMS_OUTPUT.PUT_LINE('error ...');
my_arr.extend;
perform DBMS_OUTPUT.PUT_LINE(my_arr[1]||'--'||my_arr[2]);
end;
end;
/
--输出溢出值
declare
type v_arr is varray(3) of varchar2(30);
my_arr v_arr:= v_arr(null);
begin
my_arr[4]:='插入';
perform DBMS_OUTPUT.PUT_LINE(my_arr[4]);
end;
/

declare
type v_arr is varray(3) of varchar2(30);
my_arr v_arr:= v_arr(null);
begin
my_arr[2]:='插入';
perform DBMS_OUTPUT.PUT_LINE(my_arr[2]);
```

```
end;
/

--验证nesttable 类型
declare
type dateArray is table of varchar2(10);
date_val dateArray:=dateArray('20201231','20200930');
begin
for i in 1 .. date_val.count loop
perform dbms_output.serveroutput('t');
perform dbms_output.put_line('结果：'||date_val[i]);
end loop;
end;
/
```

嵌套表

最近更新时间: 2024-06-12 15:06:00

嵌套表是一种列类型，它以无特定顺序存储未指定数量的行。当一个嵌套表值从数据库检索到一个PL/SQL 嵌套表变量中时，PL/SQL会为这些行提供从 1 开始的连续索引。使用这些索引，可以访问嵌套表变量的各个行。语法是variable_name(index)。 支持的方法如下：

方法	描述
exists（index）	索引处的元素是否存在
count	当前集合中的元素总个数
limit	集合元素索引的最大值，返回null
first	返回集合第一个元素索引
last	返回集合最后一个元素索引
prior	当前元素的前一个
next	当前元素的后一个
extend	扩展集合的容量
trim	从集合的尾部删除元素
delete	按索引删除集合元素

目前TDSQL PG不支持使用外部定义的嵌套表类型，只能使用select bulk collect into 示例： 嵌套表读取

```
drop table if exists emp;
CREATE TABLE EMP(EMPNO NUMBER(4) CONSTRAINT PK_EMP PRIMARY KEY,ENAME VARCHAR2(20),JOB VARCHAR2(9),M
GR NUMBER(4), HIREDATE DATE,SAL NUMBER(7,2),COMM NUMBER(7,2), DEPTNO NUMBER(4));
INSERT INTO EMP VALUES(7369,'SMITH','CLERK',7902,to_date('17-12-1980','dd-mm-yyyy'),800,NULL,200);
INSERT INTO EMP VALUES(7499,'ALLEN','SALESMAN',7698,to_date('20-2-1981','dd-mm-yyyy'),1600,300,300);
INSERT INTO EMP VALUES(7521,'WARD','SALESMAN',7698,to_date('22-2-1981','dd-mm-yyyy'),1250,500,300);
INSERT INTO EMP VALUES(7566,'JONES','MANAGER',7839,to_date('2-4-1981','dd-mm-yyyy'),2975,NULL,200);
INSERT INTO EMP VALUES(7654,'MARTIN','SALESMAN',7698,to_date('28-9-1981','dd-mm-yyyy'),1250,1400,300);
INSERT INTO EMP VALUES(7698,'BLAKE','MANAGER',7839,to_date('1-5-1981','dd-mm-yyyy'),2850,NULL,300);
INSERT INTO EMP VALUES(7782,'CLARK','MANAGER',7839,to_date('9-6-1981','dd-mm-yyyy'),2450,NULL,100);
INSERT INTO EMP VALUES(7788,'SCOTT','ANALYST',7566,to_date('19-04-87','dd-mm-rr'),3000,NULL,200);
INSERT INTO EMP VALUES(7839,'KING','PRESIDENT',NULL,to_date('17-11-1981','dd-mm-yyyy'),5000,NULL,100);
INSERT INTO EMP VALUES(7844,'TURNER','SALESMAN',7698,to_date('8-9-1981','dd-mm-yyyy'),1500,0,300);
INSERT INTO EMP VALUES(7876,'ADAMS','CLERK',7788,to_date('23-05-87', 'dd-mm-rr'),1100,NULL,200);
INSERT INTO EMP VALUES(7900,'JAMES','CLERK',7698,to_date('3-12-1981','dd-mm-yyyy'),950,NULL,300);
INSERT INTO EMP VALUES(7902,'FORD','ANALYST',7566,to_date('3-12-1981','dd-mm-yyyy'),3000,NULL,200);
INSERT INTO EMP VALUES(7934,'MILLER','CLERK',7782,to_date('23-1-1982','dd-mm-yyyy'),1300,NULL,100);

drop table IF EXISTS dept;
create table dept (DEPTNO NUMBER(4), DNAME VARCHAR2(14), LOC VARCHAR2(13));
insert into dept VALUES(100, 'ACCOUNTING', 'NEW YORK');
insert into dept VALUES(200, 'RESEARCH', 'DALLAS');
insert into dept VALUES(300, 'SALES', 'CHICAGO');
insert into dept VALUES(300, 'SALES', 'NEW YORK');
```

```
insert into dept VALUES(200, 'ACCOUNTING', 'NEW YORK');
insert into dept VALUES(200, 'RESEARCH', 'CHICAGO');
insert into dept VALUES(100, 'ACCOUNTING', 'BOSTON');
insert into dept VALUES(400, 'OPERATIONS', 'BOSTON');

-- TestPoint：存储过程里使用动态数组的相关函数extend/first/count/last/exists/prior/limit/next
create or replace procedure nested_pro
as
declare
cursor emp_cursor is select ename from emp;
type emp_table is table of emp.ename%type;
empTable emp_table:=emp_table();
keyValue integer:=0;
begin
empTable.extend;
perform dbms_output.serveroutput('y');
for tmp in emp_cursor
loop
keyValue := keyValue+1;
--EXTEND:追加1个空元素到集合
-- bug,已经有一个空间了，还是报数据越界
empTable.extend;
empTable[keyValue] := tmp.ename;
end loop;

--FIRST：返回在使用整数下标集合的第一个（最小的）索引号
keyValue := empTable.FIRST;

raise notice '%',keyValue;
raise notice '%',empTable.last;
raise notice '%',empTable.count;

WHILE keyValue IS NOT null
LOOP
perform dbms_output.put_line('Emp name: ' || TO_CHAR(empTable[keyValue] || ' The emp index: ' || keyValue));
--返回索引keyValue的下一个索引号
keyValue := empTable.NEXT(keyValue);
END LOOP;

--LAST：返回在使用整数下标集合的最后一个（最大的）索引号
-- bug,first=0
keyValue:= empTable.LAST;
WHILE keyValue IS NOT null
LOOP
perform dbms_output.put_line('Emp name: ' || TO_CHAR(empTable[keyValue] || ' The emp index: ' || keyValue));
--返回索引keyValue的上一个索引号
keyValue := empTable.prior(keyValue);
END LOOP;

-- 打印最大值
-- bug
-- perform dbms_output.put_line('limit values:'||empTable.limit);
raise notice 'limit values %', empTable.limit;

-- 判断元素是否存在
raise notice 'exists:%', empTable.exists(2);
end;
```

```
/
call nested_pro();
```

记录类型

最近更新时间: 2024-06-12 15:06:00

要创建记录变量，可以先定义RECORD类型，然后创建该类型的变量，也可以使用%ROWTYPE或%TYPE声明变量类型。 示例：record类型使用binary名

```
declare
type empinfo is record (empno number,ename varchar2(50));
v_one_row empinfo;
begin
select eno,ename into v_one_row from emp where rownum<=1; DBMS_OUTPUT.PUT_LINE(v_one_row.empno||' '||v_one_row.ename);
end;
/
```

PL/SQL静态SQL
静态SQL概述

最近更新时间: 2024-06-12 15:06:00

静态SQL在编译时是已知的。一般情况下静态 SQL 语句与相应的 SQL 语句具有相同的语法。

- SELECT（查询语句）
- 数据操纵语言（DML）语句：INSERT、UPDATE、DELETE 和 MERGE。
- 事务控制语言（TCL）语句：COMMIT、ROLLBACK、SAVEPOINT和SET TRANSACTION。

游标

显示游标

最近更新时间: 2024-06-12 15:06:00

用户构造和管理的游标是显式游标 显示游标支持以下属性

%FOUND	只有在DML语句影响一行或者多行时，此属性才会返回true
%NOTFOUND	与%FOUND作用相反，如果DML语句没有影响任何行，则此属性返回true
%ROWCOUNT	返回DML语句影响的行数。如果DML语句没有影响任何行，则返回0
%ISOPEN	游标打开时返回true，游标关闭时返回false。隐式游标，此属性一直返回false

示例：显示游标属性

```
DECLARE
v_empno EMPLOYEES.EMPLOYEE_ID%TYPE;
v_sal EMPLOYEES.Salary%TYPE;
CURSOR c_cursor IS SELECT EMPLOYEE_ID, Salary FROM EMPLOYEES;
BEGIN
OPEN c_cursor;
LOOP
FETCH c_cursor INTO v_empno, v_sal;
EXIT WHEN c_cursor%NOTFOUND;
IF v_sal<=2000 THEN
UPDATE EMPLOYEES SET Salary=Salary+50 WHERE EMPLOYEE_ID=v_empno;
DBMS_OUTPUT.PUT_LINE('编码为'||v_empno||'工资已更新!');
END IF;
DBMS_OUTPUT.PUT_LINE('记录数:'|| c_cursor%ROWCOUNT);
END LOOP;
DBMS_OUTPUT.PUT_LINE('总记录数:'|| c_cursor%ROWCOUNT);
CLOSE c_cursor;
if c_cursor%ISOPEN then
DBMS_OUTPUT.PUT_LINE('关闭游标失败');
else
DBMS_OUTPUT.PUT_LINE('关闭游标成功');
end if;
END;
/
```

隐式游标

最近更新时间: 2024-06-12 15:06:00

由数据库系统隐含创建游标是隐式游标 隐示游标支持以下属性

属性	值	SELECT	INSERT	UPDATE	DELETE
SQL%ISOPEN		FALSE	FALSE	FALSE	FALSE
SQL%FOUND	TRUE	有结果	成功	成功	失败
SQL%FOUND	FALSE	无结果	失败	失败	失败
SQL%NOTFOUND	TRUE	无结果	失败	失败	失败
SQL%NOTFOUND	FALSE	有结果	成功	成功	成功
SQL%ROWCOUNT		返回行数	插入行数	修改行数	删除行数

示例：隐式游标属性

```
DECLARE
V_deptno departments.department_id%TYPE :=1;
BEGIN
DELETE FROM departments WHERE department_id=v_deptno;
IF SQL%NOTFOUND THEN
DBMS_OUTPUT.PUT_LINE('没有发现 department_i='||v_deptno);
insert into departments values('a'||v_deptno, v_deptno*10, v_deptno);
DBMS_OUTPUT.PUT_LINE('插入数据行数='||SQL%ROWCOUNT);
elsif SQL%FOUND then
DBMS_OUTPUT.PUT_LINE('删除数据行数='||SQL%ROWCOUNT);
END IF;

if SQL%ISOPEN then
DBMS_OUTPUT.PUT_LINE('ISOPEN 为TRUE！！！有问题');
end if;
END;
/
```

使用for循环遍历游标

最近更新时间: 2024-06-12 15:06:00

示例：

```
---创建表
drop table if exists employees;
create table employees(last_name varchar2(20), employee_id int, job_id varchar2(30), salary number(10, 3));
insert into employees values('a', 1, 'ACADFIMKSA_MANGR', 10005.45);
insert into employees values('b', 2, 'b_M_MANGR', 47956);
insert into employees values('c', 3, 'c_cursor', 1457.79);
insert into employees values('d', 4, 'deptcurtyp', 1458.256);192ye
insert into employees values('e', 5, 'SHT_CLERK', 1458.256);
insert into employees values('f', 6, 'SH_CLERK', 1458.256);
insert into employees values('g', 7, 'SA_REP', 1458.256);
insert into employees values('h', 8, 'AD_REP', 1458.256);
---定义游标，使用for循环遍历游标
DECLARE
cursor c_cursor is select job_id||'--'||last_name||'--'||to_char(salary) as l from employees where rownum<=10 order by last_
name;
v_ename employees.last_name%type;
begin
perform dbms_output.serveroutput('t');
for v_ename in c_cursor
loop
perform DBMS_OUTPUT.PUT_LINE(c_cursor%ROWCOUNT||'---'||v_ename.l);
end loop;
end;
/
```

游标的使用

定义一个游标

最近更新时间: 2024-06-12 15:06:00

```
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。

提取下一行数据

最近更新时间: 2024-06-12 15:06:00

```
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 TDSQL PG
(1 row)
postgres=# FETCH NEXT from tbase_cur ;
id | nickname
----+-----------
2 | TDSQL PG好
(1 row)
```

提取前一行数据

最近更新时间: 2024-06-12 15:06:00

```
postgres=# FETCH PRIOR from tbase_cur ;
id | nickname
----+-------------
1 | hello TDSQL PG
```

提取最后一行

最近更新时间: 2024-06-12 15:06:00

```
postgres=# FETCH LAST from tbase_cur ;
id | nickname
----+------------
5 | TDSQL PG swap
(1 row)
```

提取第一行

最近更新时间: 2024-06-12 15:06:00

```
postgres=# FETCH FIRST from tbase_cur ;
id | nickname
----+-------------
1 | hello TDSQL PG
(1 row)
```

提取该查询的第x行

最近更新时间: 2024-06-12 15:06:00

```
postgres=# FETCH ABSOLUTE 2 from tbase_cur ;
id | nickname
----+-----------
2 | TDSQL PG好
(1 row)
postgres=# FETCH ABSOLUTE -1 from tbase_cur ;
id | nickname
----+------------
5 | TDSQL PG swap
(1 row)
postgres=# FETCH ABSOLUTE -2 from tbase_cur ;
id | nickname
----+---------------
4 | TDSQL PG default
(1 row)
```

X为负数时则从尾部向上提。

提取当前位置后的第x行

最近更新时间: 2024-06-12 15:06:00

```
postgres=# FETCH ABSOLUTE 1 from tbase_cur ;
id | nickname
----+-------------
1 | hello TDSQL PG
(1 row)
postgres=# FETCH RELATIVE 2 from tbase_cur ;
id | nickname
----+------------
3 | TDSQL PG好
(1 row)
postgres=# FETCH RELATIVE 2 from tbase_cur ;
id | nickname
----+------------
5 | TDSQL PG swap
(1 row)
```

每提取一次数据，游标的位置都是会前行。

向前提取x行数据

最近更新时间: 2024-06-12 15:06:00

```
postgres=# FETCH FORWARD 2 from tbase_cur ;
id | nickname
----+-------------
1 | hello TDSQL PG
2 | TDSQL PG好
(2 rows)
postgres=# FETCH FORWARD 2 from tbase_cur ;
id | nickname
----+---------------
3 | TDSQL PG好
4 | TDSQL PG default
(2 rows)
```

向前提取剩下的所有数据

最近更新时间: 2024-06-12 15:06:00

```
postgres=# FETCH FORWARD 2 from tbase_cur ;
id | nickname
----+-------------
1 | hello TDSQL PG
2 | TDSQL PG好
(2 rows)
postgres=# FETCH FORWARD all from tbase_cur ;
id | nickname
----+---------------
3 | TDSQL PG好
4 | TDSQL PG default
5 | TDSQL PG swap
(3 rows)
```

向后提取x行数据

最近更新时间: 2024-06-12 15:06:00

```
postgres=# FETCH BACKWARD 2 from tbase_cur ;
id | nickname
----+---------------
5 | TDSQL PG swap
4 | TDSQL PG default
(2 rows)
```

向后提取剩下的所有数据

最近更新时间: 2024-06-12 15:06:00

```
postgres=# FETCH BACKWARD all from tbase_cur ;
id | nickname
----+-------------
3 | TDSQL PG好
2 | TDSQL PG好
1 | hello TDSQL PG
(3 rows)
```

事务处理与控制

最近更新时间: 2024-06-12 15:06:00

TDSQL PG支持事务处理，允许多个用户同时处理数据库，并确保每个用户看到一致的数据版本，并确保以正确的顺序应用所有更改。 事务是Oracle数据库作为一个单元处理的一个或多个SQL语句的序列：要么执行所有语句，要么都不执行。 详细见章节 **关于事务**

自治事务

最近更新时间: 2024-06-12 15:06:00

自治事务是由另一个事务（主事务）启动的独立事务。 自治事务执行SQL操作并提交或回滚，而无需提交或回滚主事务。 支持匿名块或者存储过程中使用子事务。 示例：

```
create table t3(f1 int);

do
$$
begin
insert into t3 values(1);
insert into t3 values(2);
commit;
insert into t3 values(3);
rollback;
end;
$$ ;
create or replace procedure demo11(a_varchar in varchar)
as
$$
begin
insert into t3 values(1);
insert into t3 values(2);
commit;
insert into t3 values(3);
rollback;
end;
$$
LANGUAGE plpgsql ;
call demo11(null);
select * from t3;
```

PL/SQL动态SQL

最近更新时间: 2024-06-12 15:06:00

动态SQL是一种在PL/SQL运行时生成和执行SQL语句的编程方法，为PL/SQL编程提供了很大的灵活性。主要应用于一些静态SQL无法支持的操作场景，例如DDL，又或者是一些需要传入参数的SQL语句。 动态 SQL 使用 EXECUTE IMMEDIATE 语句处理大多数动态 SQL 语句。如果动态 SQL 语句是返回多行的 SELECT 语句，将 EXECUTE IMMEDIATE 语句与 BULK COLLECT INTO 子句一起使用。动态SQL传入参数使用USING子句。 示例： 1、使用USING字句给动态SQL传入参数

```
CREATE TABLE EMP
(EMPNO NUMBER(4) CONSTRAINT PK_EMP PRIMARY KEY,
ENAME VARCHAR2(10),
JOB VARCHAR2(9),
MGR NUMBER(4),
HIREDATE DATE,
SAL NUMBER(7,2),
COMM NUMBER(7,2),
DEPTNO NUMBER(2));

INSERT INTO EMP VALUES
(7369,'SMITH','CLERK',7902,to_date('17-12-1980','dd-mm-yyyy'),800,NULL,20);
INSERT INTO EMP VALUES
(7499,'ALLEN','SALESMAN',7698,to_date('20-2-1981','dd-mm-yyyy'),1600,300,30);
INSERT INTO EMP VALUES
(7521,'WARD','SALESMAN',7698,to_date('22-2-1981','dd-mm-yyyy'),1250,500,30);
INSERT INTO EMP VALUES
(7566,'JONES','MANAGER',7839,to_date('2-4-1981','dd-mm-yyyy'),2975,NULL,20);


create or replace procedure p_sql is
v_sql varchar(500);
vret number;
vin number:=1111;
begin
perform dbms_output.serveroutput('t');
v_sql:='select sal from emp where empno=:id';
execute immediate v_sql into vret using 7521;
perform dbms_output.put_line(vret);
-- raise notice 'vret=%',vret;
end;
/
call p_sql();
```

2、SELECT返回多条记录时，使用 BULK COLLECT INTO 字句。

```
create table ascii_t(id int, c1 varchar2(100), c2 char(100), c3 nchar(100), c4 nvarchar2(100), c5 clob, c6 nclob, c7 number, c8 smallint, c9 date, c10 timestamp);
insert into ascii_t values(1, 'abc', 'eqw', chr(20)||'da', chr(30)||'==', '而我却', chr(465)||'饿我去的', 465.89, -100, '2021-07-06 10:15:12', to_timestamp('2021-07-06 14:38:48.680297', 'yyyy-mm-dd HH24:mi:ss.ff'));
insert into ascii_t values(2, '中国', '大苏打', chr(125)||'ddsaa', chr(120)||'==', 'daewq', chr(879)||'大苏打ew', 1.89, 128,'2020-08-06 10:15:12', to_timestamp('2021-07-06 14:38:48.680297', 'yyyy-mm-dd HH24:mi:ss.ff'));
insert into ascii_t values(3, '四届', 'eqw', chr(458)||'打算', chr(654)||'==', '打算而我却', chr(135)||'而我打算eqw', 0.89, 0, '2020-08-06 10:15:12', to_timestamp('2021-07-08 14:38:48', 'yyyy-mm-dd HH24:mi:ss'));
insert into ascii_t values(4, 'abc', '而且我给v的', chr(20)||'da', chr(445)||'==', '和梵蒂冈', chr(135)||'aeq4556大', -895.89, 11, '20
```

```
21-08-06 10:15:12', to_timestamp('2021-08-08 14:38:48', 'yyyy-mm-dd HH24:mi:ss'));
insert into ascii_t values(5, 'abewqe', 'eqw', chr(34)||'恶趣味', chr(102)||'==', '日期', chr(798)||'321', -7895.89, 22, '2020-09-06
10:15:12', to_timestamp('2021-07-09 14:38:48', 'yyyy-mm-dd HH24:mi:ss'));
insert into ascii_t values(6, '打算', 'dasdas', chr(38)||'大师的人情味', chr(128)||'==', '发', chr(4565)||'', 7851.89, -56, '2020-10-06
10:15:12', to_timestamp('2021-10-08 14:38:48', 'yyyy-mm-dd HH24:mi:ss'));
-- 插入空值
insert into ascii_t values(7, '', 'dasdas','', chr(128)||'==', '发', chr(4565)||'', '', 127, '', '');
insert into ascii_t values(8, '打算', '', chr(38)||'大师的人情味', '' ,'发', '', 7895.89, '', SYSDATE, '');
insert into ascii_t values(9, '', 'dasdas', '', '', '', chr(4565)||'', 7895.89, '', '', to_timestamp('2021-10-08 14:38:48', 'yyyy-mm-dd
HH24:mi:ss'));

create or replace procedure ascii_pro(col varchar) is
type ascii_into is table of number;
var_c1 ascii_into;
v_sql varchar2(200);
begin
v_sql := 'select ascii('||col||') from ascii_t order by id;';
execute immediate v_sql bulk collect into var_c1;
/*输出雇员信息*/
for v_index in var_c1.first .. var_c1.last loop
raise notice '%','ascii:'||var_c1[v_index];
end loop;
end;
/
call ascii_pro('c2');
```

3、动态执行CREATE TABLE

```
create table emp1 as select * from emp;
create table dept_tmp (deptno number(2),dname varchar2(14),loc varchar2(13));
create or replace package pkg1 is
procedure raise_salary(v1 number,v2 number);
end;
/
create or replace package body pkg1 is
procedure raise_salary(v1 number,v2 number) is
begin
update emp1 set sal=sal+v2 where empno=v1;
end;
end;
/
DECLARE
sql_stmt VARCHAR2(200);
plsql_block VARCHAR2(500);
emp_id NUMBER(4) := 7566;
salary NUMBER(7,2);
dept_id NUMBER(2) := 50;
dept_name VARCHAR2(14) := 'PERSONNEL';
location VARCHAR2(13) := 'DALLAS';
emp_rec emp%ROWTYPE;
BEGIN
EXECUTE IMMEDIATE 'CREATE TABLE using_t (id NUMBER, amt NUMBER)';
sql_stmt := 'INSERT INTO dept_tmp VALUES (:1, :2, :3)';
EXECUTE IMMEDIATE sql_stmt USING dept_id, dept_name, location;
sql_stmt := 'SELECT * FROM emp WHERE empno = :id';
EXECUTE IMMEDIATE sql_stmt INTO emp_rec USING emp_id;
-- plsql_block := 'BEGIN pkg1.raise_salary(:id, :amt); END;';
```

```
plsql_block:='call pkg1.raise_salary(:id, :amt)';
EXECUTE IMMEDIATE plsql_block USING 7788, 500;
-- execute immediate v_sql using in vin, in vret;
-- sql_stmt := 'UPDATE emp SET sal = 2000 WHERE empno = :1
-- RETURNING sal INTO :2';
-- EXECUTE IMMEDIATE sql_stmt USING emp_id RETURNING INTO salary;
EXECUTE IMMEDIATE 'DELETE FROM dept_tmp WHERE deptno = :num'
USING dept_id;
EXECUTE IMMEDIATE 'ALTER table dept_tmp add c1 number';
END;
/
```

PL/SQL子程序
过程

最近更新时间: 2024-06-12 15:06:00

使用CREATEPROCEDURE 创建过程 示例：支持using out用法

```
set enable_oracle_compatible to on;

begin
PERFORM DBMS_OUTPUT.ENABLE();
PERFORM DBMS_OUTPUT.SERVEROUTPUT ('t');
end;
/

create or replace procedure using_out_p1(a_ret out number) is
begin
a_ret:=10;
end;
/

create or replace procedure using_out_p2(a_ret out number, b_ret out varchar) is
begin
a_ret:=10;
b_ret:='ok';
end;
/

create or replace procedure using_out_p3(a_ret out number, b_ret out varchar, c_out in out numeric) is
begin
a_ret:=10;
b_ret:='ok';
c_out := c_out+10;
end;
/


-- TestPoint : 一个out参数
create or replace procedure p_1 is
vsql varchar(500);
v_ret number;
begin
vsql:='call using_out_p1(:1)';
execute immediate vsql using out v_ret;
perform dbms_output.put_line(v_ret);
end;
/

call p_1();
```

函数

最近更新时间: 2024-06-12 15:06:00

使用CREATE[OR REPLACE] FUNCTION创建函数 示例： 有无参数的函数与有参数的函数，从block调用函数，从函数调用函数

```
create or replace function get_log_id return number is
Result number;
begin
select LOG_ID.nextval into Result from dual;
return(Result);
end get_log_id;
/

create or replace function get_log_id(v1 number) return number is
Result number;
begin
Result:=v1;
return(Result);
end get_log_id;
/

declare
v number;
begin
v:= get_log_id;
raise notice '%',v;
exception
when others then
raise;
end;
/

declare
v number;
begin
v:= get_log_id + get_log_id(100);
raise notice '%',v;
exception
when others then
raise;
end;
/

drop function if exists f1;
create or replace function f1 return number is
v number;
begin
v:= get_log_id + get_log_id(100);
raise notice '%',v;
return v;
exception
when others then
return null;
end;
/
```

```
select f1 from dual;
```

⚠TDSQL支持包内子程序重载

PL/SQL触发器

最近更新时间: 2024-06-12 15:06:00

触发器是一个定义好的的PL/ SQL单元，它存储在数据库中并运行以响应数据库中发生的事件。 您可以指定事件，触发器在事件之前或之后触发，以及触发器是针对每个事件还是针对受事件影响的每一行运行。例如，您可以创建一个每次INSERT语句影响EMPLOYEES表时运行的触发器。 详细见章节**触发器**

PL/SQL错误处理

最近更新时间: 2024-06-12 15:06:00

PL / SQL使检测和处理错误变得容易。 发生错误时，PL/ SQL会引发异常。正常执行停止并控制传输到PL/ SQL块的异常处理部分。通过以下几个示例，说明错误处理类型。

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
```

结果为

```
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)

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)

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,4/36715
9,0,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,4/36
7159,0,ERROR,23505," 程序EXCEPTION ",,,,,,,,,"psql"
```

参数详细介绍
参数模式
IN模式

最近更新时间: 2024-06-12 15:06:00

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('TDSQL PG');
f1
------
TDSQL PG
(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('TDSQL PG');
f1
------
TDSQL PG
(1 行记录)
#上面两种方式定义的参数效果是一样的
```

OUT模式

最近更新时间: 2024-06-12 15:06:00

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-# $$
postgres$# BEGIN
postgres$# a_xm:='TDSQL PG';
postgres$# END;
postgres$# $$
postgres-# LANGUAGE PLPGSQL;
CREATE FUNCTION
postgres=# SELECT * FROM f1();
a_xm
------
TDSQL PG
(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
```

INOUT模式

最近更新时间: 2024-06-12 15:06:00

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('TDSQL PG');
a_xm
------
TDSQL PG
(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-# $$
postgres$# BEGIN
postgres$# RETURN 'TDSQL PG';
postgres$# END;
postgres$# $$
postgres-# LANGUAGE PLPGSQL;
CREATE FUNCTION
postgres=# SELECT * FROM f1('TDSQL PG');
f1
------
TDSQL PG
(1 行记录)
```

VARIADIC模式

最近更新时间: 2024-06-12 15:06:00

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('TDSQL PG',1,2);
NOTICE: a_int = {1,2}
NOTICE: a_int[1] = 1
NOTICE: a_xm = TDSQL PG
f1
----
(1 行记录)
```

参数引用
无命名参数

最近更新时间: 2024-06-12 15:06:00

```
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('TDSQL PG');
f2
------
TDSQL PG
(1 行记录)
```

给标识符指定别名

最近更新时间: 2024-06-12 15:06:00

```
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('TDSQL PG');
f2
------
TDSQL PG
(1 行记录)
```

命名参数

最近更新时间: 2024-06-12 15:06:00

```
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('TDSQL PG');
NOTICE: a_xm = TDSQL PG ; v_xm = TDSQL PG ; $1 = TDSQL PG
f2
------
TDSQL PG
(1 行记录)
```

参数数据类型
基本类型

最近更新时间: 2024-06-12 15:06:00

```
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,'TDSQL PG');
NOTICE: a_int = 1 ; a_str = TDSQL PG
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['TDSQL PG','pgxz']);
NOTICE: a_int = {1,2,3} ; a_str = {TDSQL PG,pgxz}
f3
----
(1 行记录)
```

复合类型

最近更新时间: 2024-06-12 15:06:00

```
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,'TDSQL PG')::public.t_per);
NOTICE: id = 1 ; mc = TDSQL PG
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,'TDSQL PG'),ROW(1,'pgxz')]::public.t_per[]);
NOTICE: a_rec = {"(1,TDSQL PG)","(1,pgxz)"}
NOTICE: a_rec[1].id = 1
f3
----
(1 行记录)
```

行类型

最近更新时间: 2024-06-12 15:06:00

```
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,'TDSQL PG'));
NOTICE: id = 1 ; mc = TDSQL PG
f3
----
(1 行记录)
postgres=# SELECT f3(t.*) FROM t LIMIT 1;
NOTICE: id = 1 ; mc = TDSQL PG
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,'TDSQL PG'),row(1,'pgxz')]::public.t[]);
NOTICE: a_rec = {"(1,TDSQL PG)","(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,TDSQL PG)","(1,TDSQL PG)"}
NOTICE: a_rec[1].id = 1
NOTICE: a_rec = {"(2,pgxz)","(2,pgxz)"}
NOTICE: a_rec[1].id = 2
f3
----
(2 行记录)
```

域类型

最近更新时间: 2024-06-12 15:06:00

```
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
```

游标类型

最近更新时间: 2024-06-12 15:06:00

```
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,TDSQL PG)
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 | TDSQL PG
(1 行记录)
```

多态类型

最近更新时间: 2024-06-12 15:06:00

```
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('TDSQL PG'::TEXT);
NOTICE: TDSQL PG
f_any
-------
(1 行记录)
postgres=# SELECT f_any(ROW(1,'TDSQL PG')::public.t_rec);
NOTICE: (1,TDSQL PG)
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['TDSQL PG','pgxz']::TEXT[]);
NOTICE: {TDSQL PG,pgxz}
f_any_array
-------------
(1 行记录)
postgres=# SELECT f_any_array(ARRAY[ARRAY['TDSQL PG','pgxz'],ARRAY['TDSQL PG','Tencent']]::TEXT[][]);
NOTICE: {{TDSQL PG,pgxz},{TDSQL PG,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。

参数默认值

最近更新时间: 2024-06-12 15:06:00

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。

返回值详细介绍
返回值介绍

最近更新时间: 2024-06-12 15:06:00

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。

返回值类型介绍
没有返回值

最近更新时间: 2024-06-12 15:06:00

```
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)
```

返回简单类型

最近更新时间: 2024-06-12 15:06:00

```
postgres=# CREATE OR REPLACE FUNCTION f9() RETURNS TEXT AS
postgres-# $$
postgres$# BEGIN
postgres$# RETURN 'TDSQL PG';
postgres$# END;
postgres$# $$
postgres-# LANGUAGE plpgsql;
CREATE FUNCTION
postgres=#
postgres=# SELECT * FROM f9() t(a_xm);
a_xm
------
TDSQL PG
(1 行记录)
postgres=#
postgres=# CREATE OR REPLACE FUNCTION f9(OUT a_xm TEXT) RETURNS TEXT AS
postgres-# $$
postgres$# BEGIN
postgres$# a_xm:='TDSQL PG';
postgres$# END;
postgres$# $$
postgres-# LANGUAGE plpgsql;
CREATE FUNCTION
postgres=#
postgres=# SELECT * FROM f9();
a_xm
------
TDSQL PG
(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-# $$
postgres$# BEGIN
postgres$# RETURN ARRAY['TDSQL PG','pgxz'];
postgres$# END;
postgres$# $$
postgres-# LANGUAGE plpgsql;
CREATE FUNCTION
postgres=#
postgres=# SELECT * FROM f10();
f10
---------------
{TDSQL PG,pgxz}
(1 行记录)
```

返回一个复合类型

最近更新时间: 2024-06-12 15:06:00

```
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-# $$
postgres$# DECLARE
postgres$# v_rec public.t_rec;
postgres$# BEGIN
postgres$# v_rec.id:=1;
postgres$# v_rec.mc='TDSQL PG';
postgres$# RETURN v_rec;
postgres$# END;
postgres$# $$
postgres-# LANGUAGE plpgsql;
CREATE FUNCTION
postgres=#
postgres=# SELECT * FROM f11();
id | mc
----+------
1 | TDSQL PG
(1 行记录)
postgres=# CREATE OR REPLACE FUNCTION f12() RETURNS t_rec[] AS
postgres-# $$
postgres$# BEGIN
postgres$# RETURN ARRAY[ROW(1,'TDSQL PG'),ROW(1,'pgxz')]::t_rec[];
postgres$# END;
postgres$# $$
postgres-# LANGUAGE plpgsql;
CREATE FUNCTION
postgres=#
postgres=# SELECT * FROM f12();
f12
---------------------------
{"(1,TDSQL PG)","(1,pgxz)"}
(1 行记录)
```

返回行类型

最近更新时间: 2024-06-12 15:06:00

```
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 | TDSQL PG
(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,TDSQL PG)","(1,TDSQL PG)"}
(1 行记录)
```

返回TABLE类型

最近更新时间: 2024-06-12 15:06:00

```
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,TDSQL PG,pgxz)
(1 行记录)
postgres=# SELECT * FROM f15() t(id integer,xm text,xl text);
id | xm | xl
----+------+--------
1 | TDSQL PG | pgxz
(1 行记录)
```

返回RECORD类型

最近更新时间: 2024-06-12 15:06:00

```
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,TDSQL PG,pgxz)
(1 行记录)
postgres=# SELECT * FROM f15() t(id integer,xm text,xl text);
id | xm | xl
----+------+--------
1 | TDSQL PG | pgxz
(1 行记录)
```

返回一个游标

最近更新时间: 2024-06-12 15:06:00

```
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 | TDSQL PG
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 | TDSQL PG
2 | pgxz
(2 行记录)
```

```
postgres=# END;
COMMIT
```

返回记录集

最近更新时间: 2024-06-12 15:06:00

```
postgres=# CREATE OR REPLACE FUNCTION f17() RETURNS SETOF TEXT AS
postgres-# $$
postgres$# BEGIN
postgres$# RETURN NEXT 'TDSQLPG::text;
postgres$# RETURN NEXT 'pgxz'::text;
postgres$# RETURN ; #最后的RETURN可以加，也可以不加上去
postgres$# END;
postgres$# $$
postgres-# LANGUAGE PLPGSQL;
CREATE FUNCTION
postgres=#
postgres=# SELECT * FROM f17();
f17
--------
TDSQLPG
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 | TDSQLPG
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 | TDSQLPG
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 | TDSQLPG
2 | pgxc
3 | pgxz
(3 行记录)
postgres=# SELECT * FROM f20(3) t(id integer,yhm text,nc text);
id | yhm | nc
----+--------+--------
1 | TDSQLPG | TDSQLPG
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 | TDSQLPG
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 | TDSQLPG
2 | pgxz
(2 行记录)
postgres=# FETCH ALL FROM "<unnamed portal 14>";
id | yhm | nc | mm
----+--------+--------+---------------------------------
```

```
1 | TDSQLPG | TDSQLPG | 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 | TDSQLPG
2 | pgxz
(2 行记录)
postgres=# FETCH ALL FROM "b";
id | yhm | nc | mm
----+--------+--------+----------------------------------
1 | TDSQLPG | TDSQLPG | 202cb962ac59075b964b07152d234b70
2 | pgxc | pgxc | 202cb962ac59075b964b07152d234b70
3 | pgxz | pgxz | 202cb962ac59075b964b07152d234b70
(3 行记录)
postgres=# COMMIT;
COMMIT
```

返回多态类型

最近更新时间: 2024-06-12 15:06:00

```
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('TDSQLPG'::text);
f23
------
TDSQLPG
(1 行记录)
postgres=# SELECT * FROM f23(1::integer);
f23
-----
1
(1 行记录)

postgres=# SELECT * FROM f23(ARRAY['TDSQLPG','pgxz']);
f23
---------------
{TDSQLPG,pgxz}
(1 行记录)
postgres=# SELECT * FROM f23(ROW(1,'TDSQLPG')::public.t_rec);
id | mc
----+------
1 | TDSQLPG
(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,TDSQLPG,TDSQLPG,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。

变量使用
变量使用介绍

最近更新时间: 2024-06-12 15:06:00

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。

变量使用实例
变量声明语法

最近更新时间: 2024-06-12 15:06:00

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NOTNULL，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 :=。

定义一个普通变量

最近更新时间: 2024-06-12 15:06:00

```
postgres=# CREATE OR REPLACE FUNCTION f25() RETURNS VOID AS
postgres-# $$
postgres$# DECLARE
postgres$# #所有变量的声明都要放在这里,建议变量以v_开头,参数以a_开头
postgres$# v_int integer := 1;
postgres$# v_text text;
postgres$# BEGIN
postgres$# v_text = 'TDSQL PG';
postgres$# RAISE NOTICE 'v_int = %',v_int;
postgres$# RAISE NOTICE 'v_text = %',v_text;
postgres$# END;
postgres$# $$
postgres-# LANGUAGE plpgsql;
CREATE FUNCTION
postgres=# SELECT f25();
NOTICE: v_int = 1
NOTICE: v_text = TDSQL PG
f25
-----
(1 row)
postgres=#
```

定义CONSTANT变量

最近更新时间: 2024-06-12 15:06:00

```
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
```

定义NOT NULL变量

最近更新时间: 2024-06-12 15:06:00

```
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。

定义COLLATE变量

最近更新时间: 2024-06-12 15:06:00

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)
```

变量赋值

最近更新时间: 2024-06-12 15:06:00

```
postgres=# CREATE OR REPLACE FUNCTION f25() RETURNS VOID AS
postgres-# $$
postgres$# DECLARE
postgres$# #定义时赋值
postgres$# v_int1 integer = 1;
postgres$# #使用 :=兼容于plsql
postgres$# v_int2 integer := 1;
postgres$# v_txt1 text;
postgres$# v_float float8;
postgres$# #使用查询赋值
postgres$# v_relname text = (select relname FROM pg_class LIMIT 1);
postgres$# v_relpages integer;
postgres$# v_rec RECORD;
postgres$# BEGIN
postgres$# #在函数体中赋值
postgres$# v_txt1 = 'TDSQL PG';
postgres$# v_float = random();
postgres$# #使用查询赋值的另一种方式
postgres$# SELECT relname,relpages INTO v_relname,v_relpages FROM pg_class ORDER BY random() LIMIT 1;
postgres$# RAISE NOTICE 'v_relname = % , relpages = %',v_relname,v_relpages;
postgres$# END;
postgres$# $$
postgres-# LANGUAGE plpgsql;
CREATE FUNCTION
postgres=# SELECT * FROM f25();
NOTICE: v_relname = pg_ts_parser , relpages = 1
f25
-----
(1 row)
```

PL/SQL函数实战
批量设置表owner的函数

最近更新时间: 2024-06-12 15:06:00

```
CREATE OR REPLACE FUNCTION public.alter_owner(a_schema_name varchar,a_role_name varchar) RETURNS TEXT AS
$$
DECLARE
-- a_schema_name:指定某个模式下，不对定是对数据库的所有表
-- a_role_name：表所有者
v_rec RECORD;
v_sql TEXT;
BEGIN
IF a_schema_name != '' THEN --如果指模式，检查模式是否存
PERFORM 1 FROM pg_namespace WHERE nspname = a_schema_name;
IF NOT FOUND THEN
RETURN '指定的模式 ' || a_schema_name || ' 不存在!';
END IF;
END IF;
PERFORM 1 FROM pg_roles WHERE rolname = a_role_name ;
IF NOT FOUND THEN --检查用户是否存在
RETURN '指定的用户 ' || a_role_name || ' 不存在!';
END IF;
IF a_schema_name != '' THEN --指定了模式
v_sql:='SELECT schemaname,tablename FROM pg_tables WHERE schemaname=''' || a_sche || '''';
ELSE
v_sql:='SELECT schemaname,tablename FROM pg_tables WHERE schemaname!=''pg_catalog'' AND schemaname!=''infor
mation_schema'' ';
END IF;
FOR v_rec IN EXECUTE v_sql LOOP
EXECUTE 'ALTER TABLE "'||v_rec.schemaname||'"."'||v_rec.tablename||'" OWNER TO '||a_role_name;
END LOOP;
RETURN 'ok';
END;
$$
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';
```

批量设置表的加密规则函数

最近更新时间: 2024-06-12 15:06:00

```
create or replace function MLS_TRANSPARENT_CRYPT_ALGORITHM_BIND_ALL_TABLE(a_schema text,a_algo_id int) returns
text as
$$
declare
v_rec record;
v_algorithm_name text;
v_raise_notice text;
v_sqlstate text;
v_context text;
v_message_text text;
begin
perform 1 from pg_namespace where nspname=a_schema ;
if not found then
return '模式['||a_schema||']不存在';
end if;
--显示使用的加密算法
select algorithm_name INTO v_algorithm_name from pg_transparent_crypt_policy_algorithm where algorithm_id=a_algo_i
d;
if not found then
return '加密算法id['||a_algo_id::text||']不存在';
else
raise notice '你使用的密码算法为--%',v_algorithm_name;
end if;
for v_rec in select pg_tables.schemaname,pg_tables.tablename,pg_transparent_crypt_policy_map.tblname from pg_tables l
eft outer join pg_transparent_crypt_policy_map on pg_tables.schemaname=pg_transparent_crypt_policy_map.nspname an
d pg_tables.tablename=pg_transparent_crypt_policy_map.tblname where pg_tables.schemaname=a_schema and pg_trans
parent_crypt_policy_map.tblname is null loop
begin
PERFORM MLS_TRANSPARENT_CRYPT_ALGORITHM_BIND_TABLE(v_rec.schemaname,v_rec.tablename, a_algo_id);
EXCEPTION WHEN OTHERS THEN
GET STACKED DIAGNOSTICS v_sqlstate = RETURNED_SQLSTATE,
v_message_text = MESSAGE_TEXT,
v_context = PG_EXCEPTION_CONTEXT;
RAISE NOTICE '出错信息 : %',v_message_text;
end;
end loop;
return '配置表加密完成';
end;
$$
language plpgsql;
```

oracle to_date函数的实现

最近更新时间: 2024-06-12 15:06:00

TDSQL PG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)
```

关于包(PACKAGE)
语法说明
包结构

最近更新时间: 2024-06-12 15:06:00

包有两个组成部分：包说明（Specification）与包体（Body）。 包说明由公共变量、游标、对象、过程、函数和异常的声明组成，是一个应用程序接口 (API)，包含客户端程序调用其子程序所需的所有信息，但没有关于其实现的信息。 包体定义了对已声明游标的查询，以及在包说明中声明的子程序的代码（因此，既没有声明游标也没有子程序的包不需要主体）。包体还可以定义局部子程序，这些子程序在包说明中没有声明，只能被包中的其他子程序调用（内部调用）。

创建包

最近更新时间: 2024-06-12 15:06:00

创建包说明

```
CREATE [OR REPLACE] PACKAGE 包名IS

#定义数据类型
type xxx is 数据类型。

#定义函数
function 函数名(参数 参数类型...) return 返回类型;

#定义存储过程
procedure 存储过程(参数 参数类型...) ;

END
/
```

创建包体

```
CREATE [OR REPLACE] PACKAGE BODY 包名IS
#函数实现
function 函数名(参数 参数类型...) return 返回类型 is

#定义变量
begin
----
end

#存储过程实现
procedure 存储过程(参数 参数类型...) is

begin
----
end;

END
/
```

删除包

最近更新时间: 2024-06-12 15:06:00

---删除包体 DROP PACKAGE BODY 包名; ---删除包 DROP PACKAGE 包名;

包的使用
函数在包中的用法

最近更新时间: 2024-06-12 15:06:00

```
#定义
postgres=#CREATE OR REPLACE package b_1 IS
#定义函数
function addnum(a_1 number,a_2 number) return number;
end;
/
CREATE OR REPLACE package body b_1 is
#实现方法
function addnum(a_1 number,a_2 number) return number is
num number;
begin
num:=a_1+a_2;
return num;
end;
end;
/
#调用
postgres=# select b_1.addnum(1,1);
addnum
--------
2
(1 row)
```

存储过程在包中的用法

最近更新时间: 2024-06-12 15:06:00

```
#定义
postgres=#
drop table t1;
create table t1(f1 int ,f2 varchar);
CREATE OR REPLACE package b_2 IS
#定义过程
procedure insert_record(a_f1 int,a_f2 varchar);
end;
/
CREATE OR REPLACE package body b_2 is
#实现过程
procedure insert_record(a_f1 int,a_f2 varchar) is
begin
insert into t1 values(a_f1,a_f2);
end;
end;
/
#调用
postgres=# call b_2.insert_record(1,'tdsql pg');
CALL
postgres=# select * from t1;
f1 | f2
----+-------
1 | tdsql pg
(1 row)
```

函数与存储过程一起使用

最近更新时间: 2024-06-12 15:06:00

```
postgres=#
drop table t1;
create table t1(f1 int ,f2 varchar);
CREATE OR REPLACE package b_3 IS
#定义函数
function addnum(a_1 number,a_2 number) return number;
#定义过程
procedure insert_record(a_f1 int,a_f2 varchar);
end;
/
CREATE OR REPLACE package body b_3 is
#实现方法
function addnum(a_1 number,a_2 number) return number is
num number;
begin
num:=a_1+a_2;
return num;
end;

#实现过程
procedure insert_record(a_f1 int,a_f2 varchar) is
begin
insert into t1 values(a_f1,a_f2);
end;
end;
/
#调用
postgres=# select b_3.addnum(1,1);
addnum
--------
2
(1 row)
postgres=# call b_3.insert_record(1,'tdsql pg');
CALL
postgres=# select * from t1;
f1 | f2
----+-------
1 | tdsql pg
(1 row)
```

变量使用方法

最近更新时间: 2024-06-12 15:06:00

```
postgres=# CREATE OR REPLACE package b_4 IS
#定义变量
v_1 int default 10;
#定义结构体
type v_struct is record(
f1 int,
f2 varchar
);
v_struct_1 v_struct;
#定义游标,但不能定义游标变量
type v_ref is ref cursor;

#定义存储过程
procedure setval();
procedure echoval();
end;
/
CREATE OR REPLACE package body b_4 is
#实现过程
procedure setval() is
begin
v_1:=random()*10;
v_struct_1.f1:=random()*10;
v_struct_1.f2:=md5(random()::text);
end;

procedure echoval() is
v_rec record;
v_ref_1 v_ref;
type myvarchar is table of varchar;
v_myvarchar_1 myvarchar:='{"tbase","pgxz"}';
begin

raise notice 'v_1 = %',v_1;
raise notice 'v_struct_1 = %',v_struct_1;
raise notice 'v_myvarchar = %',v_myvarchar_1;
open v_ref_1 for select schemaname,tablename from pg_tables limit 2;
loop
fetch v_ref_1 into v_rec;
exit when v_ref_1%notfound;
raise notice 'v_rec = %',v_rec;
end loop;
end;
end;
/
#调用
postgres=# call b_4.setval();
CALL
postgres=# call b_4.echoval();
NOTICE: v_1 = 8
NOTICE: v_struct_1 = (9,618ad5bdc38b67af927e46d5ba2fca60)
```

```
NOTICE: v_myvarchar = {tbase,pgxz}
NOTICE: v_rec = (public,t1)
NOTICE: v_rec = (squeeze,tables_internal)
CALL
postgres=#
postgres=# declare
v_1 int:=b_4.v_1; #定义包中变量
begin
raise notice 'v_1 = %',v_1;
b_4.v_1:=11;
raise notice 'b_4.v_1 = %',b_4.v_1;
end;
/
NOTICE: v_1 = 11
NOTICE: b_4.v_1 = 11
DO
```

其他常见使用

最近更新时间: 2024-06-12 15:06:00

- 查看已经存在的package。

```
postgres=# select pkgname from pg_package;
```

- 查看已经存在的公共变量和函数列表。

```
postgres=# select pkgheadersrc from pg_package;
```

- 查看函数列表。

```
postgres=# \df b_3.*
```

- 查看函数内容。

```
postgres=# \df+ b_3.addnum
postgres=# \df+ b_3.insert_record
```

关于触发器
创建触发器

最近更新时间: 2024-06-12 15:06:00

使用命令 CREATE TRIGGER... ON ... EXECUTE PROCEDURE ... 创建触发器 触发事件可以是INSERT，UPDATE [OF column_name [, ...]]，DELETE或者TRUNCATE 示例：触发事件，插入记录时，如果f2字段值小于0时，则将f2的值修改成0 创建表

```
create table t_trigger(f1 int,f2 int);
```

创建函数

```
CREATE OR REPLACE FUNCTION t_trigger_insert_trigger_func () RETURNS trigger AS
$body$
BEGIN
if NEW.f2
NEW.f2=0;
end if;
RETURN NEW;
END;
$body$
LANGUAGE plpgsql;
```

创建触发器

```
CREATE TRIGGER t_trigger_insert_trigger BEFORE INSERT ON t_trigger FOR EACH ROW EXECUTE PROCEDURE t_trigger_ins
ert_trigger_func();
```

插入数据

```
insert into t_trigger values(1,-1);
```

查询数据

```
select * from t_trigger;
```

⚠查询结果f2为0，插入数据事件触发了触发器，直接将f2的值修改为0

删除触发器

最近更新时间: 2024-06-12 15:06:00

使用DROP TRIGGER 删除触发器 示例：

```
DROP TRIGGER t_trigger_insert_trigger ON t_trigger;
```

触发器函数
INSERT事件触发器函数

最近更新时间: 2024-06-12 15:06:00

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-# $$
postgres$# BEGIN
postgres$# IF NEW.nc = '' THEN
postgres$# NEW.nc = 'TDSQLPG_' || random()::text;
postgres$# END IF;
postgres$# RETURN NEW;
postgres$# END;
postgres$# $$
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 | TDSQLPG_0.426093454472721
(1 row)
```

> **注意：**
>
> 使用BEFORE，不能使用AFTER，否则重写失效。

UPDATE事件触发器函数

最近更新时间: 2024-06-12 15:06:00

不准许更新t_trigger.nc字段值为 TDSQL PG。

```
postgres=# CREATE OR REPLACE FUNCTION t_trigger_update_trigger_func() RETURNS trigger AS
postgres-# $$
postgres$# BEGIN
postgres$# --不准许t_trigger.nc值为 TBase
postgres$# IF NEW.nc = 'TDSQL PG' THEN
postgres$# NEW.nc = OLD.nc ;
postgres$# END IF;
postgres$# RETURN NEW;
postgres$# END;
postgres$# $$
postgres-# LANGUAGE plpgsql;
CREATE FUNCTION
postgres=# CREATE TRIGGER t_trigger_update_trigger BEFORE UPDATE ON t_trigger FOR EACH ROW EXECUTE PROCEDU
RE t_trigger_update_trigger_func();
CREATE TRIGGER
postgres=# UPDATE t_trigger SET nc='TDSQL PG' WHERE id=1;
UPDATE 1
postgres=# SELECT * FROM t_trigger ;
id | nc
----+-------------------------
1 | TDSQLPG_0.426093454472721
(1 row)
postgres=#
```

DELETE事件触发器函数

最近更新时间: 2024-06-12 15:06:00

限制TDSQL PG记录不能被删除。

```
postgres=# CREATE OR REPLACE FUNCTION t_trigger_delete_trigger_func() RETURNS trigger AS
postgres-# $$
postgres$# BEGIN
postgres$# #不准许t_trigger.nc值为 TDSQL PG
postgres$# IF OLD.nc = 'TDSQL PG' THEN
postgres$# RETURN NULL;
postgres$# --RAISE EXCEPTION 'TDSQL PG不能被删除';
postgres$# END IF;
postgres$# RETURN OLD;
postgres$# END;
postgres$# $$
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,'TDSQL PG');
INSERT 0 1
postgres=# SELECT * t_trigg
postgres=# SELECT * FROM t_trigger ;
id | nc
----+-------------------------
1 | TDSQLPG_0.426093454472721
2 | TDSQL PG
(2 rows)
postgres=# DELETE FROM t_trigger WHERE id=2;
DELETE 0
postgres=# SELECT * FROM t_trigger ;
id | nc
----+-------------------------
1 | TDSQLPG_0.426093454472721
2 | TDSQL PG
(2 rows)
```

删除触发器

最近更新时间: 2024-06-12 15:06:00

```
postgres=# drop TRIGGER t_trigger_insert_trigger on t_trigger;
DROP TRIGGER
```

触发器使用限制

最近更新时间: 2024-06-12 15:06:00

分区表，冷热分区表和复制表不支持使用触发器。

ORACLE兼容性特性与SQL参考
启用Oracle兼容性

最近更新时间: 2024-06-12 15:06:00

启用Oracle兼容特性，需要进行参数 `enable_oracle_compatible` 设置。

> **注意：**
>
> TDSQL PG Oracle兼容版v5.0x版本该参数目前不可关闭。

- session中生效

  ```
  set enable_oracle_compatible to on;
  ```

- 配置某个库默认生效

  ```
  alter database postgres set enable_oracle_compatible to on;
  ```

- 配置某个账号生效

  ```
  alter role tbase_user set enable_oracle_compatible to on;
  ```

系统package
ALL_ARGUMENTS

最近更新时间: 2024-06-12 15:06:00

列出了当前用户可以访问的函数和过程的参数。

名称	类型	说明
owner	varchar2(128)	对象所有者
object_name	varchar2(128)	过程或函数名称
package_name	varchar2(128)	包名称
object_id	number	对象的对象编号
overload	varchar2(40)	无效
subprogram_id	number	无效
argument_name	varchar2(128)	参数名。空参数名称用于表示函数返回
position	number	表示在参数列表中的位置，或 0 表示函数返回值
sequence	number	定义参数的顺序。 参数序列从 1 开始。返回类型在前，然后是每个参数
data_level	number	复合类型参数的嵌套深度
data_type	varchar2(30)	参数数据类型
defaulted	varchar2(1)	指定参数是否默认
default_value	long	保留
default_length	number	保留
in_out	varchar2(9)	参数方向:IN/OUT/INOUT
data_length	number	无效
data_precision	number	无效
data_scale	number	无效
radix	number	无效
character_set_name	varchar2(44)	无效
type_owner	varchar2(128)	无效
type_name	varchar2(128)	无效
type_subname	varchar2(128)	无效
type_link	varchar2(128)	无效

名称	类型	说明
type_object_type	varchar2(7)	无效
pls_type	varchar2(128)	无效
char_length	number	无效
char_used	varchar2(1)	无效
origin_con_id	varchar2(256)	无效

示例：

```
create or replace FUNCTION f3f_all_argument_20221111_1 (a int, b out number, c INOUT varchar2) return int is
begin
return a;
end;
/
create or replace procedure p3p_all_argument_20221111_1(e int, f out number, g INOUT varchar2) is
aa int;
begin
aa := e;
end;
/
create or replace function f1f_all_argument_20221111_2(a int, b number default 123) return int is
begin
return a;
end;
/
create or replace procedure p5p_all_argument_20221111_2(a int, b varchar2 default 'abc', c number default 123, d number
default NULL) is
e int;
begin
e := a;
end;
/
create or replace procedure pDp_all_argument_20221111_3(a int) is
begin
null;
end;
/
create or replace type obj_all_argument_20221111_1 as object(a int);
/
CREATE OR REPLACE TYPE nt_all_argument_20221111_2 AS TABLE OF VARCHAR2(30);
/
create table tbl_all_arguments_20221111_1 (f1 int, f2 varchar2(60));
create or replace function argTYPes_all_argument_20221111_3(a int[], obj obj_all_argument_20221111_1, nt nt_all_argume
nt_20221111_2, rt tbl_all_arguments_20221111_1%rowtype) return int is
begin
return 2;
end;
/
select owner, object_name, package_name, overload,
SUBPROGRAM_ID, ARGUMENT_NAME, position, sequence,
```

```
DEFAULTED, DEFAULT_VALUE, IN_OUT, data_length, radix
from all_arguments where object_name = upper('argTYPes_all_argument_20221111_3') order by object_name, position;
```

DBMS_JOB

最近更新时间: 2024-06-12 15:06:00

DBMS_JOB主要用来对在JOB QUEUE中的JOB做管理。该包已经被DBMS_SCHEDULER 包所取代. 特别是,如果你正在管理JOB以控制系统负载，那么你应该考虑通过收回用户在该 包上的执行权限，以禁和DBMS_JOB包。

示例：

```
declare
jobno number;
begin dbms_job.submit(
job=>jobno,
what=>'proc_job(2);',
next_date=>sysdate,
interval=>'sysdate+1/1440');
COMMIT;
dbms_output.put_line(jobno);
end;
```

DBMS_ASSERT

最近更新时间: 2024-06-12 15:06:00

DBMS_ASSERT包提供了一个验证输入值属性的接口。

存储过程/函数	描述
ENQUOTE_LITERAL	引用字符串文字
ENQUOTE_NAME	确保字符串用引号括起来，然后检查结果是否为有效的SQL标识符。
QUALIFIED_SQL_NAME	验证输入字符串是否为合格的SQL名称
SCHEMA_NAME	验证输入字符串是否为现有模式名称
SIMPLE_SQL_NAME	验证输入字符串是否为简单的SQL名称
SQL_OBJECT_NAME	验证输入参数字符串是现有SQL对象的限定SQL标识符

示例：

```
select dbms_assert.enquote_literal('tbase''') from dual;
ERROR: numeric or value error
select dbms_assert.enquote_name('TBase') from dual;
select dbms_assert.qualified_sql_name('create table') from dual;
ERROR: string is not qualified SQL name
select dbms_assert.schema_name('TBASE') from dual;
select dbms_assert.simple_sql_name('2tb') from dual;
ERROR: string is not simple SQL name
select dbms_assert.sql_object_name('T1111') from dual;
ERROR: invalid object name
```

DBMS_LOB

最近更新时间: 2024-06-12 15:06:00

DBMS_LOB用于在大对象上进行操作。DBMS_LOB包提供了子程序可以在BLOB、CLOB、NCLOB、BFILE和临时LOB上操作的子程序。使用DBMS_LOB可以访问和处理LOB的特定部分或全部

存储过程/函数	描述
APPEND(dest_lob IN OUT,src_lob)	Appends one large object to another.
COMPARE(lob_1, lob_2 [, amount[, offset_1 [, offset_2]]])	Compares two large objects.
CONVERTOBLOB(dest_lob IN OUT,src_clob, amount, dest_offsetIN OUT, src_offset IN OUT,blob_csid, lang_context IN OUT,warning OUT)	Converts character data to binary.
CONVERTTOCLOB(dest_lob IN OUT,src_blob, amount, dest_offsetIN OUT, src_offset IN OUT,blob_csid, lang_context IN OUT,warning OUT)	Converts binary data to character.
COPY(dest_lob IN OUT, src_lob,amount [, dest_offset [,src_offset]])	Copies one large object to another.
ERASE(lob_loc IN OUT, amount IN OUT [, offset])	Erase a large object.
GET_STORAGE_LIMIT(lob_loc)	Get the storage limit for large objects.
GETLENGTH(lob_loc)	Get the length of the large object.
INSTR(lob_loc, pattern [,offset [, nth]])	Get the position of the nth occurrence of a pattern in the large object starting atoffset.
READ(lob_loc, amount IN OUT,offset, buffer OUT)	Read a large object.
SUBSTR(lob_loc [, amount [,offset]])	Get part of a large object.
TRIM(lob_loc IN OUT, newlen)	Trim a large object to the specified length.
WRITE(lob_loc IN OUT, amount,offset, buffer)	Write data to a large object.
WRITEAPPEND(lob_loc IN OUT,amount, buffer)	Write data from the buffer to the end of a large object.

示例：

```
DBMS_LOB.WRITEAPPEND (
lob_loc IN OUT NOCOPY BLOB,
amount IN INTEGER,
buffer IN RAW);

DBMS_LOB.WRITEAPPEND (
lob_loc IN OUT NOCOPY CLOB CHARACTER SET ANY_CS,
amount IN INTEGER,
buffer IN VARCHAR2 CHARACTER SET lob_loc%CHARSET);


declare
v_clob1 clob;
```

```
begin
v_clob1:=to_clob('123456');
dbms_output.put_line(v_clob1);
DBMS_LOB.WRITEAPPEND(v_clob1,3,'789');
dbms_output.put_line(v_clob1);
end;
```

DBMS_OUTPUT

最近更新时间: 2024-06-12 15:06:00

DBMS_OUTPUT用于向消息缓冲区发送消息（文本行的形式出现），或者从消息缓冲区中获取消息。

存储过程/函数	描述
DISABLE	Disable the capability to send and receive messages.
ENABLE(buffer size)	Enable the capability to send and receive messages.
GET LINE(line OUT, status OUT)	Get a line from the message buffer.
GET LINES(lines OUT, numlines IN OUT)	Get multiple lines from the message buffer.
NEW LINE	Puts an end-of-line character sequence.
PUT(item)	Puts a partial line without an end-of-line character sequence.
PUT LINE(item)	Puts a complete line with an end-of-line character sequence.
SERVEROUTPUT(stdout)	Direct messages from PUT, PUT LINE, or NEW_LINE to either standard output or the message buffer.

示例：

```
\set ECHO none
SET client_min_messages = warning;
SET DATESTYLE TO ISO;
SET client_encoding = utf8;
\pset null '<NULL>'
\set ECHO all

DROP FUNCTION dbms_output_test();
DROP TABLE dbms_output_test;

-- DBMS_OUTPUT.DISABLE [0]
CREATE TABLE dbms_output_test (id serial,buff VARCHAR(20), status INTEGER);
CREATE FUNCTION dbms_output_test() RETURNS VOID AS $$
DECLARE
buff VARCHAR(20);
stts INTEGER;
BEGIN
PERFORM DBMS_OUTPUT.PUT_LINE ('ORAFCE TEST 1');
SELECT INTO buff,stts line,status FROM DBMS_OUTPUT.GET_LINE();
INSERT INTO dbms_output_test(buff,status) VALUES (buff, stts);
END;
$$ LANGUAGE plpgsql;
SELECT dbms_output_test();
DROP TABLE dbms_output_test;
DROP FUNCTION dbms_output_test();

-- DBMS_OUTPUT.PUT_LINE [1]
```

```
CREATE FUNCTION dbms_output_test() RETURNS VOID AS $$
DECLARE
buff1 VARCHAR(20) := 'orafce';
BEGIN
PERFORM DBMS_OUTPUT.DISABLE();
PERFORM DBMS_OUTPUT.ENABLE();
PERFORM DBMS_OUTPUT.SERVEROUTPUT ('t');
PERFORM DBMS_OUTPUT.PUT_LINE ('ORAFCE');
PERFORM DBMS_OUTPUT.PUT_LINE (buff1);
PERFORM DBMS_OUTPUT.PUT ('ABC');
PERFORM DBMS_OUTPUT.PUT_LINE ('');
END;
$$ LANGUAGE plpgsql;
SELECT dbms_output_test();
DROP FUNCTION dbms_output_test();

-- DBMS_OUTPUT.PUT_LINE [2]
CREATE FUNCTION dbms_output_test() RETURNS VOID AS $$
BEGIN
PERFORM DBMS_OUTPUT.DISABLE();
PERFORM DBMS_OUTPUT.ENABLE();
PERFORM DBMS_OUTPUT.SERVEROUTPUT ('t');
PERFORM DBMS_OUTPUT.PUT_LINE ('ORA
F
CE');
END;
$$ LANGUAGE plpgsql;
SELECT dbms_output_test();
DROP FUNCTION dbms_output_test();

-- DBMS_OUTPUT.PUT [1]
CREATE FUNCTION dbms_output_test() RETURNS VOID AS $$
DECLARE
buff1 VARCHAR(20) := 'ora';
buff2 VARCHAR(20) := 'f';
buff3 VARCHAR(20) := 'ce';
BEGIN
PERFORM DBMS_OUTPUT.DISABLE();
PERFORM DBMS_OUTPUT.ENABLE();
PERFORM DBMS_OUTPUT.SERVEROUTPUT ('t');
PERFORM DBMS_OUTPUT.PUT ('ORA');
PERFORM DBMS_OUTPUT.PUT ('F');
PERFORM DBMS_OUTPUT.PUT ('CE');
PERFORM DBMS_OUTPUT.PUT_LINE ('');
PERFORM DBMS_OUTPUT.PUT ('ABC');
PERFORM DBMS_OUTPUT.PUT_LINE ('');
END;
$$ LANGUAGE plpgsql;
SELECT dbms_output_test();
DROP FUNCTION dbms_output_test();

-- DBMS_OUTPUT.PUT [2]
CREATE FUNCTION dbms_output_test() RETURNS VOID AS $$
BEGIN
PERFORM DBMS_OUTPUT.DISABLE();
PERFORM DBMS_OUTPUT.ENABLE();
PERFORM DBMS_OUTPUT.SERVEROUTPUT ('t');
```

```
PERFORM DBMS_OUTPUT.PUT ('ORA
F
CE');
PERFORM DBMS_OUTPUT.PUT_LINE ('');
END;
$$ LANGUAGE plpgsql;
SELECT dbms_output_test();
DROP FUNCTION dbms_output_test();

-- DBMS_OUTPUT.GET_LINE [1]
CREATE TABLE dbms_output_test (id serial,buff VARCHAR(20), status INTEGER);
CREATE FUNCTION dbms_output_test() RETURNS VOID AS $$
DECLARE
buff VARCHAR(20);
stts INTEGER;
BEGIN
PERFORM DBMS_OUTPUT.DISABLE();
PERFORM DBMS_OUTPUT.ENABLE();
PERFORM DBMS_OUTPUT.SERVEROUTPUT ('f');
PERFORM DBMS_OUTPUT.PUT_LINE ('ORAFCE TEST 1');
PERFORM DBMS_OUTPUT.PUT_LINE ('ORAFCE TEST 2');
SELECT INTO buff,stts line,status FROM DBMS_OUTPUT.GET_LINE();
INSERT INTO dbms_output_test(buff,status) VALUES (buff, stts);
SELECT INTO buff,stts line,status FROM DBMS_OUTPUT.GET_LINE();
INSERT INTO dbms_output_test(buff,status) VALUES (buff, stts);
PERFORM DBMS_OUTPUT.DISABLE();
PERFORM DBMS_OUTPUT.ENABLE();
END;
$$ LANGUAGE plpgsql;
SELECT dbms_output_test();
SELECT * FROM dbms_output_test order by buff;
DROP TABLE dbms_output_test;
DROP FUNCTION dbms_output_test();

-- DBMS_OUTPUT.GET_LINE [2]
CREATE TABLE dbms_output_test (id serial,buff VARCHAR(20), status INTEGER);
CREATE FUNCTION dbms_output_test() RETURNS VOID AS $$
DECLARE
buff VARCHAR(20);
stts INTEGER;
BEGIN
PERFORM DBMS_OUTPUT.DISABLE();
PERFORM DBMS_OUTPUT.ENABLE();
PERFORM DBMS_OUTPUT.SERVEROUTPUT ('f');
PERFORM DBMS_OUTPUT.PUT_LINE ('ORAFCE TEST 1');
PERFORM DBMS_OUTPUT.PUT_LINE ('ORAFCE TEST 2');
SELECT INTO buff,stts line,status FROM DBMS_OUTPUT.GET_LINE();
INSERT INTO dbms_output_test(buff,status) VALUES (buff, stts);
PERFORM DBMS_OUTPUT.PUT_LINE ('ORAFCE TEST 3');
SELECT INTO buff,stts line,status FROM DBMS_OUTPUT.GET_LINE();
INSERT INTO dbms_output_test(buff,status) VALUES (buff, stts);
SELECT INTO buff,stts line,status FROM DBMS_OUTPUT.GET_LINE();
INSERT INTO dbms_output_test(buff,status) VALUES (buff, stts);
SELECT INTO buff,stts line,status FROM DBMS_OUTPUT.GET_LINE();
INSERT INTO dbms_output_test(buff,status) VALUES (buff, stts);
PERFORM DBMS_OUTPUT.DISABLE();
PERFORM DBMS_OUTPUT.ENABLE();
```

```
END;
$$ LANGUAGE plpgsql;
SELECT dbms_output_test();
DROP TABLE dbms_output_test;
DROP FUNCTION dbms_output_test();

-- DBMS_OUTPUT.GET_LINE [3]
CREATE TABLE dbms_output_test (id serial,buff VARCHAR(20), status INTEGER);
CREATE FUNCTION dbms_output_test() RETURNS VOID AS $$
DECLARE
buff VARCHAR(20);
stts INTEGER;
BEGIN
PERFORM DBMS_OUTPUT.DISABLE();
PERFORM DBMS_OUTPUT.ENABLE();
PERFORM DBMS_OUTPUT.SERVEROUTPUT ('f');
PERFORM DBMS_OUTPUT.PUT_LINE ('ORAFCE TEST 1');
PERFORM DBMS_OUTPUT.PUT_LINE ('ORAFCE TEST 2');
SELECT INTO buff,stts line,status FROM DBMS_OUTPUT.GET_LINE();
INSERT INTO dbms_output_test(buff,status) VALUES (buff, stts);
PERFORM DBMS_OUTPUT.PUT ('ORA');
SELECT INTO buff,stts line,status FROM DBMS_OUTPUT.GET_LINE();
INSERT INTO dbms_output_test(buff,status) VALUES (buff, stts);
SELECT INTO buff,stts line,status FROM DBMS_OUTPUT.GET_LINE();
INSERT INTO dbms_output_test(buff,status) VALUES (buff, stts);
PERFORM DBMS_OUTPUT.DISABLE();
PERFORM DBMS_OUTPUT.ENABLE();
END;
$$ LANGUAGE plpgsql;
SELECT dbms_output_test();
DROP TABLE dbms_output_test;
DROP FUNCTION dbms_output_test();

-- DBMS_OUTPUT.GET_LINE [4]
CREATE TABLE dbms_output_test (id serial,buff VARCHAR(20), status INTEGER);
CREATE FUNCTION dbms_output_test() RETURNS VOID AS $$
DECLARE
buff VARCHAR(20);
stts INTEGER;
BEGIN
PERFORM DBMS_OUTPUT.DISABLE();
PERFORM DBMS_OUTPUT.ENABLE();
PERFORM DBMS_OUTPUT.SERVEROUTPUT ('f');
PERFORM DBMS_OUTPUT.PUT_LINE ('ORAFCE TEST 1');
PERFORM DBMS_OUTPUT.PUT_LINE ('ORAFCE TEST 2');
SELECT INTO buff,stts line,status FROM DBMS_OUTPUT.GET_LINE();
INSERT INTO dbms_output_test(buff,status) VALUES (buff, stts);
PERFORM DBMS_OUTPUT.NEW_LINE();
SELECT INTO buff,stts line,status FROM DBMS_OUTPUT.GET_LINE();
INSERT INTO dbms_output_test(buff,status) VALUES (buff, stts);
SELECT INTO buff,stts line,status FROM DBMS_OUTPUT.GET_LINE();
INSERT INTO dbms_output_test(buff,status) VALUES (buff, stts);
PERFORM DBMS_OUTPUT.DISABLE();
PERFORM DBMS_OUTPUT.ENABLE();
END;
$$ LANGUAGE plpgsql;
SELECT dbms_output_test();
```

```
DROP TABLE dbms_output_test;
DROP FUNCTION dbms_output_test();

-- DBMS_OUTPUT.GET_LINE [5]
CREATE TABLE dbms_output_test (id serial,buff VARCHAR(20), status INTEGER);
CREATE FUNCTION dbms_output_test() RETURNS VOID AS $$
DECLARE
buff VARCHAR(20);
stts INTEGER;
BEGIN
PERFORM DBMS_OUTPUT.DISABLE();
PERFORM DBMS_OUTPUT.ENABLE();
PERFORM DBMS_OUTPUT.SERVEROUTPUT ('f');
PERFORM DBMS_OUTPUT.PUT_LINE ('ORAFCE TEST 1
');
PERFORM DBMS_OUTPUT.PUT_LINE ('ORAFCE TEST 2');
SELECT INTO buff,stts line,status FROM DBMS_OUTPUT.GET_LINE();
INSERT INTO dbms_output_test(buff,status) VALUES (buff, stts);
SELECT INTO buff,stts line,status FROM DBMS_OUTPUT.GET_LINE();
INSERT INTO dbms_output_test(buff,status) VALUES (buff, stts);
PERFORM DBMS_OUTPUT.DISABLE();
PERFORM DBMS_OUTPUT.ENABLE();
END;
$$ LANGUAGE plpgsql;
SELECT dbms_output_test();
DROP TABLE dbms_output_test;
DROP FUNCTION dbms_output_test();

-- DBMS_OUTPUT.GET_LINE [6]
CREATE TABLE dbms_output_test (id serial,buff VARCHAR(20), status INTEGER);
CREATE FUNCTION dbms_output_test() RETURNS VOID AS $$
DECLARE
buff VARCHAR(20);
stts INTEGER;
BEGIN
PERFORM DBMS_OUTPUT.DISABLE();
PERFORM DBMS_OUTPUT.ENABLE();
PERFORM DBMS_OUTPUT.SERVEROUTPUT ('f');
PERFORM DBMS_OUTPUT.PUT_LINE ('ORA
F
CE');
SELECT INTO buff,stts line,status FROM DBMS_OUTPUT.GET_LINE();
INSERT INTO dbms_output_test(buff,status) VALUES (buff, stts);
PERFORM DBMS_OUTPUT.DISABLE();
PERFORM DBMS_OUTPUT.ENABLE();
END;
$$ LANGUAGE plpgsql;
SELECT dbms_output_test();
DROP TABLE dbms_output_test;
DROP FUNCTION dbms_output_test();

-- DBMS_OUTPUT.GET_LINES [1]
CREATE TABLE dbms_output_test (id serial,buff VARCHAR(20), status INTEGER);
CREATE FUNCTION dbms_output_test() RETURNS VOID AS $$
DECLARE
buff VARCHAR(20);
buff1 VARCHAR(20);
```

```
buff2 VARCHAR(20);
buff3 VARCHAR(20);
stts INTEGER := 10;
BEGIN
PERFORM DBMS_OUTPUT.DISABLE();
PERFORM DBMS_OUTPUT.ENABLE();
PERFORM DBMS_OUTPUT.SERVEROUTPUT ('f');
PERFORM DBMS_OUTPUT.PUT_LINE ('ORAFCE TEST 1');
PERFORM DBMS_OUTPUT.PUT_LINE ('ORAFCE TEST 2');
PERFORM DBMS_OUTPUT.PUT_LINE ('ORAFCE TEST 3');
SELECT INTO buff1,buff2,buff3,stts lines[1],lines[2],lines[3],numlines FROM DBMS_OUTPUT.GET_LINES(stts);
INSERT INTO dbms_output_test(buff,status) VALUES (buff1, stts);
INSERT INTO dbms_output_test(buff,status) VALUES (buff2, stts);
INSERT INTO dbms_output_test(buff,status) VALUES (buff3, stts);
SELECT INTO buff,stts lines[1],numlines FROM DBMS_OUTPUT.GET_LINES(stts);
INSERT INTO dbms_output_test(buff,status) VALUES (buff, stts);
PERFORM DBMS_OUTPUT.DISABLE();
PERFORM DBMS_OUTPUT.ENABLE();
END;
$$ LANGUAGE plpgsql;
SELECT dbms_output_test();
DROP TABLE dbms_output_test;
DROP FUNCTION dbms_output_test();

-- DBMS_OUTPUT.GET_LINES [2]
CREATE TABLE dbms_output_test (id serial,buff VARCHAR(20), status INTEGER);
CREATE FUNCTION dbms_output_test() RETURNS VOID AS $$
DECLARE
buff VARCHAR(20);
buff1 VARCHAR(20);
buff2 VARCHAR(20);
stts INTEGER := 2;
BEGIN
PERFORM DBMS_OUTPUT.DISABLE();
PERFORM DBMS_OUTPUT.ENABLE();
PERFORM DBMS_OUTPUT.SERVEROUTPUT ('f');
PERFORM DBMS_OUTPUT.PUT_LINE ('ORAFCE TEST 1');
PERFORM DBMS_OUTPUT.PUT_LINE ('ORAFCE TEST 2');
PERFORM DBMS_OUTPUT.PUT_LINE ('ORAFCE TEST 3');
SELECT INTO buff1,buff2,stts lines[1],lines[2],numlines FROM DBMS_OUTPUT.GET_LINES(stts);
INSERT INTO dbms_output_test(buff,status) VALUES (buff1, stts);
INSERT INTO dbms_output_test(buff,status) VALUES (buff2, stts);
SELECT INTO buff,stts lines[1],numlines FROM DBMS_OUTPUT.GET_LINES(stts);
INSERT INTO dbms_output_test(buff,status) VALUES (buff, stts);
PERFORM DBMS_OUTPUT.DISABLE();
PERFORM DBMS_OUTPUT.ENABLE();
END;
$$ LANGUAGE plpgsql;
SELECT dbms_output_test();
DROP TABLE dbms_output_test;
DROP FUNCTION dbms_output_test();

-- DBMS_OUTPUT.GET_LINES [3]
CREATE TABLE dbms_output_test (id serial,buff VARCHAR(20), status INTEGER);
CREATE FUNCTION dbms_output_test() RETURNS VOID AS $$
DECLARE
buff VARCHAR(20);
```

```
stts INTEGER := 1;
BEGIN
PERFORM DBMS_OUTPUT.DISABLE();
PERFORM DBMS_OUTPUT.ENABLE();
PERFORM DBMS_OUTPUT.SERVEROUTPUT ('f');
PERFORM DBMS_OUTPUT.PUT_LINE ('ORAFCE TEST 1');
PERFORM DBMS_OUTPUT.PUT_LINE ('ORAFCE TEST 2');
SELECT INTO buff,stts lines[1],numlines FROM DBMS_OUTPUT.GET_LINES(stts);
INSERT INTO dbms_output_test(buff,status) VALUES (buff, stts);
PERFORM DBMS_OUTPUT.PUT_LINE ('ORAFCE TEST 3');
SELECT INTO buff,stts lines[1],numlines FROM DBMS_OUTPUT.GET_LINES(stts);
INSERT INTO dbms_output_test(buff,status) VALUES (buff, stts);
SELECT INTO buff,stts lines[1],numlines FROM DBMS_OUTPUT.GET_LINES(stts);
INSERT INTO dbms_output_test(buff,status) VALUES (buff, stts);
PERFORM DBMS_OUTPUT.DISABLE();
PERFORM DBMS_OUTPUT.ENABLE();
END;
$$ LANGUAGE plpgsql;
SELECT dbms_output_test();
DROP TABLE dbms_output_test;
DROP FUNCTION dbms_output_test();

-- DBMS_OUTPUT.GET_LINES [4]
CREATE TABLE dbms_output_test (id serial,buff VARCHAR(20), status INTEGER);
CREATE FUNCTION dbms_output_test() RETURNS VOID AS $$
DECLARE
buff VARCHAR(20);
stts INTEGER := 1;
BEGIN
PERFORM DBMS_OUTPUT.DISABLE();
PERFORM DBMS_OUTPUT.ENABLE();
PERFORM DBMS_OUTPUT.SERVEROUTPUT ('f');
PERFORM DBMS_OUTPUT.PUT_LINE ('ORAFCE TEST 1');
PERFORM DBMS_OUTPUT.PUT_LINE ('ORAFCE TEST 2');
SELECT INTO buff,stts lines[1],numlines FROM DBMS_OUTPUT.GET_LINES(stts);
INSERT INTO dbms_output_test(buff,status) VALUES (buff, stts);
PERFORM DBMS_OUTPUT.PUT ('ORA');
SELECT INTO buff,stts lines[1],numlines FROM DBMS_OUTPUT.GET_LINES(stts);
INSERT INTO dbms_output_test(buff,status) VALUES (buff, stts);
SELECT INTO buff,stts lines[1],numlines FROM DBMS_OUTPUT.GET_LINES(stts);
INSERT INTO dbms_output_test(buff,status) VALUES (buff, stts);
PERFORM DBMS_OUTPUT.DISABLE();
PERFORM DBMS_OUTPUT.ENABLE();
END;
$$ LANGUAGE plpgsql;
SELECT dbms_output_test();
DROP TABLE dbms_output_test;
DROP FUNCTION dbms_output_test();

-- DBMS_OUTPUT.GET_LINES [5]
CREATE TABLE dbms_output_test (id serial,buff VARCHAR(20), status INTEGER);
CREATE FUNCTION dbms_output_test() RETURNS VOID AS $$
DECLARE
buff VARCHAR(20);
stts INTEGER := 1;
BEGIN
PERFORM DBMS_OUTPUT.DISABLE();
```

```
PERFORM DBMS_OUTPUT.ENABLE();
PERFORM DBMS_OUTPUT.SERVEROUTPUT ('f');
PERFORM DBMS_OUTPUT.PUT_LINE ('ORAFCE TEST 1');
PERFORM DBMS_OUTPUT.PUT_LINE ('ORAFCE TEST 2');
SELECT INTO buff,stts lines[1],numlines FROM DBMS_OUTPUT.GET_LINES(stts);
INSERT INTO dbms_output_test(buff,status) VALUES (buff, stts);
PERFORM DBMS_OUTPUT.NEW_LINE();
SELECT INTO buff,stts lines[1],numlines FROM DBMS_OUTPUT.GET_LINES(stts);
INSERT INTO dbms_output_test(buff,status) VALUES (buff, stts);
SELECT INTO buff,stts lines[1],numlines FROM DBMS_OUTPUT.GET_LINES(stts);
INSERT INTO dbms_output_test(buff,status) VALUES (buff, stts);
PERFORM DBMS_OUTPUT.DISABLE();
PERFORM DBMS_OUTPUT.ENABLE();
END;
$$ LANGUAGE plpgsql;
SELECT dbms_output_test();
DROP TABLE dbms_output_test;
DROP FUNCTION dbms_output_test();

-- DBMS_OUTPUT.GET_LINES [6]
CREATE TABLE dbms_output_test (id serial,buff VARCHAR(20), status INTEGER);
CREATE FUNCTION dbms_output_test() RETURNS VOID AS $$
DECLARE
buff VARCHAR(20);
stts INTEGER := 1;
BEGIN
PERFORM DBMS_OUTPUT.DISABLE();
PERFORM DBMS_OUTPUT.ENABLE();
PERFORM DBMS_OUTPUT.SERVEROUTPUT ('f');
PERFORM DBMS_OUTPUT.PUT_LINE ('ORA
F
CE');
SELECT INTO buff,stts lines[1],numlines FROM DBMS_OUTPUT.GET_LINES(stts);
INSERT INTO dbms_output_test(buff,status) VALUES (buff, stts);
PERFORM DBMS_OUTPUT.DISABLE();
PERFORM DBMS_OUTPUT.ENABLE();
END;
$$ LANGUAGE plpgsql;
SELECT dbms_output_test();
DROP TABLE dbms_output_test;
DROP FUNCTION dbms_output_test();

-- DBMS_OUTPUT.NEW_LINE [1]
CREATE TABLE dbms_output_test (id serial,buff VARCHAR(20), status INTEGER);
CREATE FUNCTION dbms_output_test() RETURNS VOID AS $$
DECLARE
buff1 VARCHAR(20);
buff2 VARCHAR(20);
stts INTEGER := 10;
BEGIN
PERFORM DBMS_OUTPUT.DISABLE();
PERFORM DBMS_OUTPUT.ENABLE();
PERFORM DBMS_OUTPUT.SERVEROUTPUT ('f');
PERFORM DBMS_OUTPUT.PUT ('ORA');
PERFORM DBMS_OUTPUT.NEW_LINE();
PERFORM DBMS_OUTPUT.PUT ('FCE');
PERFORM DBMS_OUTPUT.NEW_LINE();
```

```
SELECT INTO buff1,buff2,stts lines[1],lines[2],numlines FROM DBMS_OUTPUT.GET_LINES(stts);
INSERT INTO dbms_output_test(buff,status) VALUES (buff1, stts);
INSERT INTO dbms_output_test(buff,status) VALUES (buff2, stts);
PERFORM DBMS_OUTPUT.DISABLE();
PERFORM DBMS_OUTPUT.ENABLE();
END;
$$ LANGUAGE plpgsql;
SELECT dbms_output_test();
DROP TABLE dbms_output_test;
DROP FUNCTION dbms_output_test();

-- DBMS_OUTPUT.NEW_LINE [2]
CREATE TABLE dbms_output_test (id serial,buff VARCHAR(3000), status INTEGER);
CREATE FUNCTION dbms_output_test() RETURNS VOID AS $$
DECLARE
buff1 VARCHAR(3000);
stts INTEGER := 10;
BEGIN
PERFORM DBMS_OUTPUT.DISABLE();
PERFORM DBMS_OUTPUT.ENABLE();
PERFORM DBMS_OUTPUT.SERVEROUTPUT ('f');
PERFORM DBMS_OUTPUT.ENABLE(2000);
FOR j IN 1..1999 LOOP
PERFORM DBMS_OUTPUT.PUT ('A');
END LOOP;
PERFORM DBMS_OUTPUT.NEW_LINE();
SELECT INTO buff1,stts lines[1],numlines FROM DBMS_OUTPUT.GET_LINES(stts);
INSERT INTO dbms_output_test(buff,status) VALUES (buff1, stts);
PERFORM DBMS_OUTPUT.DISABLE();
PERFORM DBMS_OUTPUT.ENABLE();
END;
$$ LANGUAGE plpgsql;
SELECT dbms_output_test();
DROP TABLE dbms_output_test;
DROP FUNCTION dbms_output_test();

-- DBMS_OUTPUT.DISABLE [1]
CREATE TABLE dbms_output_test (id serial,buff VARCHAR(20), status INTEGER);
CREATE FUNCTION dbms_output_test() RETURNS VOID AS $$
DECLARE
buff VARCHAR(20);
stts INTEGER;
BEGIN
PERFORM DBMS_OUTPUT.DISABLE();
PERFORM DBMS_OUTPUT.ENABLE();
PERFORM DBMS_OUTPUT.SERVEROUTPUT ('f');
PERFORM DBMS_OUTPUT.PUT_LINE ('ORAFCE TEST 1');
PERFORM DBMS_OUTPUT.DISABLE();
PERFORM DBMS_OUTPUT.ENABLE();
SELECT INTO buff,stts line,status FROM DBMS_OUTPUT.GET_LINE();
INSERT INTO dbms_output_test(buff,status) VALUES (buff, stts);

PERFORM DBMS_OUTPUT.DISABLE();
PERFORM DBMS_OUTPUT.ENABLE();
PERFORM DBMS_OUTPUT.DISABLE();
PERFORM DBMS_OUTPUT.PUT_LINE ('ORAFCE TEST 2');
PERFORM DBMS_OUTPUT.ENABLE();
```

```
SELECT INTO buff,stts line,status FROM DBMS_OUTPUT.GET_LINE();
INSERT INTO dbms_output_test(buff,status) VALUES (buff, stts);

PERFORM DBMS_OUTPUT.DISABLE();
PERFORM DBMS_OUTPUT.ENABLE();
PERFORM DBMS_OUTPUT.PUT_LINE ('ORAFCE TEST 3');
PERFORM DBMS_OUTPUT.DISABLE();
SELECT INTO buff,stts line,status FROM DBMS_OUTPUT.GET_LINE();
INSERT INTO dbms_output_test(buff,status) VALUES (buff, stts);
PERFORM DBMS_OUTPUT.ENABLE();

PERFORM DBMS_OUTPUT.DISABLE();
PERFORM DBMS_OUTPUT.ENABLE();
PERFORM DBMS_OUTPUT.DISABLE();
PERFORM DBMS_OUTPUT.PUT ('ORAFCE TEST 4');
PERFORM DBMS_OUTPUT.ENABLE();
PERFORM DBMS_OUTPUT.NEW_LINE();
SELECT INTO buff,stts line,status FROM DBMS_OUTPUT.GET_LINE();
INSERT INTO dbms_output_test(buff,status) VALUES (buff, stts);

PERFORM DBMS_OUTPUT.DISABLE();
PERFORM DBMS_OUTPUT.ENABLE();
PERFORM DBMS_OUTPUT.PUT ('ORAFCE TEST 5');
PERFORM DBMS_OUTPUT.DISABLE();
PERFORM DBMS_OUTPUT.NEW_LINE();
PERFORM DBMS_OUTPUT.ENABLE();
SELECT INTO buff,stts line,status FROM DBMS_OUTPUT.GET_LINE();
INSERT INTO dbms_output_test(buff,status) VALUES (buff, stts);

PERFORM DBMS_OUTPUT.DISABLE();
PERFORM DBMS_OUTPUT.ENABLE();
END;
$$ LANGUAGE plpgsql;
SELECT dbms_output_test();
DROP TABLE dbms_output_test;
DROP FUNCTION dbms_output_test();

-- DBMS_OUTPUT.DISABLE [2]
CREATE TABLE dbms_output_test (id serial,buff VARCHAR(20), status INTEGER);
CREATE FUNCTION dbms_output_test() RETURNS VOID AS $$
DECLARE
buff VARCHAR(20);
stts INTEGER := 10;
BEGIN
PERFORM DBMS_OUTPUT.DISABLE();
PERFORM DBMS_OUTPUT.ENABLE();
PERFORM DBMS_OUTPUT.SERVEROUTPUT ('f');
PERFORM DBMS_OUTPUT.DISABLE();
PERFORM DBMS_OUTPUT.PUT_LINE ('ORAFCE TEST 1');
SELECT INTO buff,stts lines[1],numlines FROM DBMS_OUTPUT.GET_LINES(stts);
INSERT INTO dbms_output_test(buff,status) VALUES (buff, stts);
PERFORM DBMS_OUTPUT.ENABLE();
END;
$$ LANGUAGE plpgsql;
SELECT dbms_output_test();
DROP TABLE dbms_output_test;
DROP FUNCTION dbms_output_test();
```

```
-- DBMS_OUTPUT.ENABLE [1]
CREATE TABLE dbms_output_test (id serial,buff VARCHAR(20), status INTEGER);
CREATE FUNCTION dbms_output_test() RETURNS VOID AS $$
DECLARE
buff VARCHAR(20);
status INTEGER;
num INTEGER := 2000;
BEGIN
PERFORM DBMS_OUTPUT.DISABLE();
PERFORM DBMS_OUTPUT.ENABLE();
PERFORM DBMS_OUTPUT.SERVEROUTPUT ('t');
PERFORM DBMS_OUTPUT.ENABLE(2000);
PERFORM DBMS_OUTPUT.PUT ('ORAFCE TEST 1');
PERFORM DBMS_OUTPUT.NEW_LINE();
PERFORM DBMS_OUTPUT.ENABLE();
END;
$$ LANGUAGE plpgsql;
SELECT dbms_output_test();
DROP TABLE dbms_output_test;
DROP FUNCTION dbms_output_test();

-- DBMS_OUTPUT.ENABLE [2]
CREATE TABLE dbms_output_test (id serial,buff VARCHAR(20), status INTEGER);
CREATE FUNCTION dbms_output_test() RETURNS VOID AS $$
DECLARE
buff VARCHAR(20);
stts INTEGER;
BEGIN
PERFORM DBMS_OUTPUT.DISABLE();
PERFORM DBMS_OUTPUT.ENABLE();
PERFORM DBMS_OUTPUT.SERVEROUTPUT ('f');
PERFORM DBMS_OUTPUT.DISABLE();
PERFORM DBMS_OUTPUT.ENABLE();
PERFORM DBMS_OUTPUT.PUT_LINE ('ORAFCE TEST 1');
SELECT INTO buff,stts line,status FROM DBMS_OUTPUT.GET_LINE();
INSERT INTO dbms_output_test(buff,status) VALUES (buff, stts);
PERFORM DBMS_OUTPUT.PUT ('ORAFCE TEST 2');
PERFORM DBMS_OUTPUT.NEW_LINE();
SELECT INTO buff,stts line,status FROM DBMS_OUTPUT.GET_LINE();
INSERT INTO dbms_output_test(buff,status) VALUES (buff, stts);
PERFORM DBMS_OUTPUT.ENABLE();
END;
$$ LANGUAGE plpgsql;
SELECT dbms_output_test();
DROP TABLE dbms_output_test;
DROP FUNCTION dbms_output_test();

-- DBMS_OUTPUT.ENABLE [3]
CREATE TABLE dbms_output_test (id serial,buff VARCHAR(20), status INTEGER);
CREATE FUNCTION dbms_output_test() RETURNS VOID AS $$
DECLARE
buff VARCHAR(20);
stts INTEGER := 10;
BEGIN
PERFORM DBMS_OUTPUT.DISABLE();
PERFORM DBMS_OUTPUT.ENABLE();
```

```
PERFORM DBMS_OUTPUT.SERVEROUTPUT ('f');
PERFORM DBMS_OUTPUT.DISABLE();
PERFORM DBMS_OUTPUT.ENABLE();
PERFORM DBMS_OUTPUT.PUT_LINE ('ORAFCE TEST 1');
SELECT INTO buff,stts lines[1],numlines FROM DBMS_OUTPUT.GET_LINES(stts);
INSERT INTO dbms_output_test(buff,status) VALUES (buff, stts);

END;
$$ LANGUAGE plpgsql;
SELECT dbms_output_test();
DROP TABLE dbms_output_test;
DROP FUNCTION dbms_output_test();

-- DBMS_OUTPUT.ENABLE [4]
CREATE FUNCTION dbms_output_test() RETURNS VOID AS $$
BEGIN
PERFORM DBMS_OUTPUT.DISABLE();
PERFORM DBMS_OUTPUT.ENABLE();
PERFORM DBMS_OUTPUT.SERVEROUTPUT ('t');
PERFORM DBMS_OUTPUT.DISABLE();
PERFORM DBMS_OUTPUT.ENABLE();
FOR j IN 1..2000 LOOP
PERFORM DBMS_OUTPUT.PUT ('A');
END LOOP;
PERFORM DBMS_OUTPUT.NEW_LINE();
END;
$$ LANGUAGE plpgsql;
SELECT dbms_output_test();
DROP FUNCTION dbms_output_test();

-- DBMS_OUTPUT.ENABLE [5]
CREATE TABLE dbms_output_test (id serial,buff VARCHAR(20), status INTEGER);
CREATE FUNCTION dbms_output_test() RETURNS VOID AS $$
DECLARE
buff VARCHAR(20);
stts INTEGER;
BEGIN
PERFORM DBMS_OUTPUT.DISABLE();
PERFORM DBMS_OUTPUT.ENABLE();
PERFORM DBMS_OUTPUT.SERVEROUTPUT ('f');
PERFORM DBMS_OUTPUT.DISABLE();
PERFORM DBMS_OUTPUT.ENABLE(NULL);
PERFORM DBMS_OUTPUT.PUT_LINE ('ORAFCE TEST 1');
SELECT INTO buff,stts line,status FROM DBMS_OUTPUT.GET_LINE();
INSERT INTO dbms_output_test(buff,status) VALUES (buff, stts);
PERFORM DBMS_OUTPUT.DISABLE();
PERFORM DBMS_OUTPUT.ENABLE();
END;
$$ LANGUAGE plpgsql;
SELECT dbms_output_test();
DROP TABLE dbms_output_test;
DROP FUNCTION dbms_output_test();

-- DBMS_OUTPUT.ENABLE [6]
CREATE TABLE dbms_output_test (id serial,buff VARCHAR(20), status INTEGER);
CREATE FUNCTION dbms_output_test() RETURNS VOID AS $$
DECLARE
```

```
buff VARCHAR(20);
stts INTEGER;
BEGIN
PERFORM DBMS_OUTPUT.DISABLE();
PERFORM DBMS_OUTPUT.ENABLE();
PERFORM DBMS_OUTPUT.SERVEROUTPUT ('f');
PERFORM DBMS_OUTPUT.PUT_LINE ('ORAFCE TEST 1');
PERFORM DBMS_OUTPUT.ENABLE();
SELECT INTO buff,stts line,status FROM DBMS_OUTPUT.GET_LINE();
INSERT INTO dbms_output_test(buff,status) VALUES (buff, stts);
PERFORM DBMS_OUTPUT.DISABLE();
PERFORM DBMS_OUTPUT.ENABLE();
END;
$$ LANGUAGE plpgsql;
SELECT dbms_output_test();
DROP TABLE dbms_output_test;
DROP FUNCTION dbms_output_test();

-- SERVEROUTPUT [1]
CREATE FUNCTION dbms_output_test() RETURNS VOID AS $$
BEGIN
PERFORM DBMS_OUTPUT.DISABLE();
PERFORM DBMS_OUTPUT.ENABLE();
PERFORM DBMS_OUTPUT.SERVEROUTPUT ('f');
PERFORM DBMS_OUTPUT.PUT_LINE ('ORAFCE TEST 1');
END;
$$ LANGUAGE plpgsql;
SELECT dbms_output_test();
DROP FUNCTION dbms_output_test();

CREATE FUNCTION dbms_output_test() RETURNS VOID AS $$
BEGIn
PERFORM DBMS_OUTPUT.DISABLE();
PERFORM DBMS_OUTPUT.ENABLE();
PERFORM DBMS_OUTPUT.SERVEROUTPUT ('t');
PERFORM DBMS_OUTPUT.PUT_LINE ('ORAFCE TEST 2');
END;
$$ LANGUAGE plpgsql;
SELECT dbms_output_test();
DROP FUNCTION dbms_output_test();

-- SERVEROUTPUT [2]
CREATE FUNCTION dbms_output_test() RETURNS VOID AS $$
BEGIN
PERFORM DBMS_OUTPUT.DISABLE();
PERFORM DBMS_OUTPUT.ENABLE();
PERFORM DBMS_OUTPUT.SERVEROUTPUT ('f');
PERFORM DBMS_OUTPUT.PUT ('ORAFCE TEST 1');
PERFORM DBMS_OUTPUT.NEW_LINE();
END;
$$ LANGUAGE plpgsql;
SELECT dbms_output_test();
DROP FUNCTION dbms_output_test();

CREATE FUNCTION dbms_output_test() RETURNS VOID AS $$
BEGIN
PERFORM DBMS_OUTPUT.DISABLE();
```

```
PERFORM DBMS_OUTPUT.ENABLE();
PERFORM DBMS_OUTPUT.SERVEROUTPUT ('t');
PERFORM DBMS_OUTPUT.PUT ('ORAFCE TEST 2');
PERFORM DBMS_OUTPUT.NEW_LINE();
END;
$$ LANGUAGE plpgsql;
SELECT dbms_output_test();
DROP FUNCTION dbms_output_test();

-- SERVEROUTPUT [3]
CREATE FUNCTION dbms_output_test() RETURNS VOID AS $$
BEGIN
PERFORM DBMS_OUTPUT.DISABLE();
PERFORM DBMS_OUTPUT.ENABLE();
PERFORM DBMS_OUTPUT.SERVEROUTPUT ('f');
PERFORM DBMS_OUTPUT.DISABLE();
END;
$$ LANGUAGE plpgsql;
SELECT dbms_output_test();
DROP FUNCTION dbms_output_test();

CREATE FUNCTION dbms_output_test() RETURNS VOID AS $$
BEGIN
PERFORM DBMS_OUTPUT.DISABLE();
PERFORM DBMS_OUTPUT.ENABLE();
PERFORM DBMS_OUTPUT.SERVEROUTPUT ('t');
PERFORM DBMS_OUTPUT.PUT_LINE ('ORAFCE TEST 1');
END;
$$ LANGUAGE plpgsql;
SELECT dbms_output_test();
DROP FUNCTION dbms_output_test();
```

DBMS_PIPE

最近更新时间: 2024-06-12 15:06:00

DBMS_PIPE包用于在同一实例的不同会话之间进行通信；注意，如果用户要执行包dbms_pipe中的过程和函数,则必须要为用户授权。
DBMS_PIPE包含以下接口：

接口	描述
*CREATE_PIPE	用于建立公用管道或私有管道。如果将参数private设置为TRUE，则建立私有管道；如果设置为FALSE，则建立公用管道。
*PACK_MESSAGE	用于将消息写入到本地消息缓冲区，包含类型number，bytea，date，string，timestamp, record
*SEND_MESSAGE	用于将本地消息缓冲区中的内容发送到管道
*RECEIVE_MESSAGE	用于接收管道消息
NEXT_ITEM_TYPE	用于确定本地消息缓冲区下一项的数据类型。 如果该函数返回0,则表示管道没有任何消息
*UNPACK_MESSAGE	用于将消息缓冲区的内容写入到变量中
*REMOVE_PIPE	用于删除已经建立的管道
PUGER	用于清除管道中的内容
RESET_BUFFER	用于复位管道缓冲区
UNIQUE_SESSION_NAME	用于为特定会话返回惟一的名称,并且名称的最长度为30字节

示例：

```
create or replace procedure dbmspipe_crtpipe_pro(in_cno int) as
declare
v_no int;
v_bigint bigint;
v_bytea bytea;
v_date date;
v_int int;
v_num numeric;
v_text text;
v_tmptz timestamp with time zone;
begin
select c,c_bigint,c_bytea,c_date,c_int,c_num,c_text,c_tmptz
from dbmspipe_tbl
where c=in_cno
into v_no,v_bigint,v_bytea,v_date,v_int,v_num,v_text,v_tmptz;
perform dbms_pipe.create_pipe('bigint_pipe1'||v_no);
perform dbms_pipe.pack_message(v_bigint);
perform dbms_pipe.send_message('bigint_pipe1'||v_no);
perform dbms_pipe.create_pipe('bigint_pipe2'||v_no,50);
perform dbms_pipe.pack_message(v_bigint);
perform dbms_pipe.send_message('bigint_pipe2'||v_no);
perform dbms_pipe.create_pipe('bigint_pipe3'||v_no,100,true);
```

```
perform dbms_pipe.pack_message(v_bigint);
perform dbms_pipe.send_message('bigint_pipe3'||v_no);
perform dbms_pipe.create_pipe('bigint_pipe4'||v_no,200,false);
perform dbms_pipe.pack_message(v_bigint);
perform dbms_pipe.send_message('bigint_pipe4'||v_no);

perform dbms_pipe.create_pipe('bytea_pipe1'||v_no);
perform dbms_pipe.pack_message(v_bytea);
perform dbms_pipe.send_message('bytea_pipe1'||v_no);
perform dbms_pipe.create_pipe('bytea_pipe2'||v_no,50);
perform dbms_pipe.pack_message(v_bytea);
perform dbms_pipe.send_message('bytea_pipe2'||v_no);
perform dbms_pipe.create_pipe('bytea_pipe3'||v_no,100,true);
perform dbms_pipe.pack_message(v_bytea);
perform dbms_pipe.send_message('bytea_pipe3'||v_no);
perform dbms_pipe.create_pipe('bytea_pipe4'||v_no,200,false);
perform dbms_pipe.pack_message(v_bytea);
perform dbms_pipe.send_message('bytea_pipe4'||v_no);

perform dbms_pipe.create_pipe('date_pipe1'||v_no);
perform dbms_pipe.pack_message(v_date);
perform dbms_pipe.send_message('date_pipe1'||v_no);
perform dbms_pipe.create_pipe('date_pipe2'||v_no,50);
perform dbms_pipe.pack_message(v_date);
perform dbms_pipe.send_message('date_pipe2'||v_no);
perform dbms_pipe.create_pipe('date_pipe3'||v_no,100,true);
perform dbms_pipe.pack_message(v_date);
perform dbms_pipe.send_message('date_pipe3'||v_no);
perform dbms_pipe.create_pipe('date_pipe4'||v_no,200,false);
perform dbms_pipe.pack_message(v_date);
perform dbms_pipe.send_message('date_pipe4'||v_no);

perform dbms_pipe.create_pipe('int_pipe1'||v_no);
perform dbms_pipe.pack_message(v_int);
perform dbms_pipe.send_message('int_pipe1'||v_no);
perform dbms_pipe.create_pipe('int_pipe2'||v_no,50);
perform dbms_pipe.pack_message(v_int);
perform dbms_pipe.send_message('int_pipe2'||v_no);
perform dbms_pipe.create_pipe('int_pipe3'||v_no,100,true);
perform dbms_pipe.pack_message(v_int);
perform dbms_pipe.send_message('int_pipe3'||v_no);
perform dbms_pipe.create_pipe('int_pipe4'||v_no,200,false);
perform dbms_pipe.pack_message(v_int);
perform dbms_pipe.send_message('int_pipe4'||v_no);

perform dbms_pipe.create_pipe('num_pipe1'||v_no);
perform dbms_pipe.pack_message(v_num);
perform dbms_pipe.send_message('num_pipe1'||v_no);
perform dbms_pipe.create_pipe('num_pipe2'||v_no,50);
perform dbms_pipe.pack_message(v_num);
perform dbms_pipe.send_message('num_pipe2'||v_no);
perform dbms_pipe.create_pipe('num_pipe3'||v_no,100,true);
perform dbms_pipe.pack_message(v_num);
perform dbms_pipe.send_message('num_pipe3'||v_no);
perform dbms_pipe.create_pipe('num_pipe4'||v_no,200,false);
perform dbms_pipe.pack_message(v_num);
perform dbms_pipe.send_message('num_pipe4'||v_no);
```

```
perform dbms_pipe.create_pipe('text_pipe1'||v_no);
perform dbms_pipe.pack_message(v_text);
perform dbms_pipe.send_message('text_pipe1'||v_no);
perform dbms_pipe.create_pipe('text_pipe2'||v_no,50);
perform dbms_pipe.pack_message(v_text);
perform dbms_pipe.send_message('text_pipe2'||v_no);
perform dbms_pipe.create_pipe('text_pipe3'||v_no,100,true);
perform dbms_pipe.pack_message(v_text);
perform dbms_pipe.send_message('text_pipe3'||v_no);
perform dbms_pipe.create_pipe('text_pipe4'||v_no,200,false);
perform dbms_pipe.pack_message(v_text);
perform dbms_pipe.send_message('text_pipe4'||v_no);

perform dbms_pipe.create_pipe('tmptz_pipe1'||v_no);
perform dbms_pipe.pack_message(v_tmptz);
perform dbms_pipe.send_message('tmptz_pipe1'||v_no);
perform dbms_pipe.create_pipe('tmptz_pipe2'||v_no,50);
perform dbms_pipe.pack_message(v_tmptz);
perform dbms_pipe.send_message('tmptz_pipe2'||v_no);
perform dbms_pipe.create_pipe('tmptz_pipe3'||v_no,100,true);
perform dbms_pipe.pack_message(v_tmptz);
perform dbms_pipe.send_message('tmptz_pipe3'||v_no);
perform dbms_pipe.create_pipe('tmptz_pipe4'||v_no,200,false);
perform dbms_pipe.pack_message(v_tmptz);
perform dbms_pipe.send_message('tmptz_pipe4'||v_no);
end;
/

--创建接收pipe message，打印message子存储过程
create or replace procedure rec_subpro(in_pipename varchar) as
declare
v_num numeric;
v_bytea bytea;
v_date date;
v_str varchar;
v_tmptz timestamp with time zone;
begin
perform dbms_output.disable();
perform dbms_output.enable();
perform dbms_output.serveroutput ('t');
perform dbms_pipe.receive_message(in_pipename);
if in_pipename like '%int%' or in_pipename like 'num' then
v_num := dbms_pipe.unpack_message_number();
perform dbms_output.put_line(in_pipename || ' message: '||v_num);
elsif in_pipename like '%bytea%' then
v_bytea := dbms_pipe.unpack_message_bytea();
perform dbms_output.put_line(in_pipename || ' message: '||v_bytea);
elsif in_pipename like '%date%' then
v_date := dbms_pipe.unpack_message_date();
perform dbms_output.put_line(in_pipename || ' message: '||v_date::text);
elsif in_pipename like '%text%' then
v_str := dbms_pipe.unpack_message_text();
perform dbms_output.put_line(in_pipename || ' message: '||v_str);
elsif in_pipename like '%tmptz%' then
v_tmptz := dbms_pipe.unpack_message_timestamp();
perform dbms_output.put_line(in_pipename || ' message: '||v_tmptz::text);
```

```
end if;
perform dbms_pipe.remove_pipe(in_pipename);
end;
/
--unpack_message_*测试：接收pipe中各种数据类型的message（record除外），接收后remove_pipe，打印出message
create or replace procedure dbmspipe_rec_pro(in_cno int) as
begin
call rec_subpro('bigint_pipe1'||in_cno);
call rec_subpro('bigint_pipe2'||in_cno);
call rec_subpro('bigint_pipe3'||in_cno);
call rec_subpro('bigint_pipe4'||in_cno);

call rec_subpro('bytea_pipe1'||in_cno);
call rec_subpro('bytea_pipe2'||in_cno);
call rec_subpro('bytea_pipe3'||in_cno);
call rec_subpro('bytea_pipe4'||in_cno);

call rec_subpro('date_pipe1'||in_cno);
call rec_subpro('date_pipe2'||in_cno);
call rec_subpro('date_pipe3'||in_cno);
call rec_subpro('date_pipe4'||in_cno);

call rec_subpro('int_pipe1'||in_cno);
call rec_subpro('int_pipe2'||in_cno);
call rec_subpro('int_pipe3'||in_cno);
call rec_subpro('int_pipe4'||in_cno);

call rec_subpro('num_pipe1'||in_cno);
call rec_subpro('num_pipe2'||in_cno);
call rec_subpro('num_pipe3'||in_cno);
call rec_subpro('num_pipe4'||in_cno);

call rec_subpro('text_pipe1'||in_cno);
call rec_subpro('text_pipe2'||in_cno);
call rec_subpro('text_pipe3'||in_cno);
call rec_subpro('text_pipe4'||in_cno);

call rec_subpro('tmptz_pipe1'||in_cno);
call rec_subpro('tmptz_pipe2'||in_cno);
call rec_subpro('tmptz_pipe3'||in_cno);
call rec_subpro('tmptz_pipe4'||in_cno);
end;
/
--sessionA:
call dbmspipe_crtpipe_pro(1);
--sessionB:
call dbmspipe_rec_pro(1);
--sessionA:
call dbmspipe_crtpipe_pro(2);
--sessionB:
call dbmspipe_rec_pro(2);
--sessionA:
call dbmspipe_crtpipe_pro(3);
--sessionB:
call dbmspipe_rec_pro(3);
--sessionA:
call dbmspipe_crtpipe_pro(4);
```

```
--sessionB:
call dbmspipe_rec_pro(4);
--sessionA:
call dbmspipe_crtpipe_pro(5);
--sessionB:
call dbmspipe_rec_pro(5);
```

DBMS_RANDOM value()函数示例

最近更新时间: 2024-06-12 15:06:00

返回两值之间的随机数。

```
postgres=#select dbms_random.value(1,100);
value
------------------
40.3055268009193
```

DBMS_SQL

最近更新时间: 2024-06-12 15:06:00

DBMS_SQL可以在应用的运行时间构建查询和其它的命令。DBMS_SQL中可以使用的存储过程及函数如下表所示：

存储过程/函数	描述
BIND_VARIABLE(c, name, value [, out_value_size])	Bind a value to a variable
BIND_VARIABLE_CHAR(c, name, value [, out_value_size])	Bind a CHAR value to a variable
BIND_VARIABLE_RAW(c, name, value [, out_value_size])	Bind a RAW value to a variable
CLOSE_CURSOR(c IN OUT)	Close a cursor
COLUMN_VALUE(c, position, value OUT [, column_error OUT [, actual_length OUT]])	Return a column value into a variable.
COLUMN_VALUE_CHAR(c, position, value OUT [, column_error OUT [, actual_length OUT]])	Return a CHAR column value into a variable.
COLUMN_VALUE_RAW(c, position, value OUT [, column_error OUT [, actual_length OUT]])	Return a RAW column value into a variable.
DEFINE_COLUMN(c, position, column [, column_size])	Define a column in the SELECT list.
DEFINE_COLUMN_CHAR(c, position, column, column_size)	Define a CHAR column in the SELECT list.
DEFINE_COLUMN_RAW(c, position, column, column_size)	Define a RAW column in the SELECT list.
DESCRIBE_COLUMNS	Defines columns to hold a cursor result set.
EXECUTE(c)	Execute a cursor.
EXECUTE_AND_FETCH(c [, exact])	Execute a cursor and fetch a single row.
FETCH_ROWS(c)	Fetch rows from the cursor.
IS_OPEN(c)	Check if a cursor is open.
LAST_ROW_COUNT	Return cumulative number of rows fetched.
OPEN_CURSOR	Open a cursor.
PARSE(c, statement, language_flag)	Parse a statement.

示例：

```
set client_min_messages TO error;
CREATE EXTENSION IF NOT EXISTS dbms_sql;
```

```
set client_min_messages TO default;

do
$$
declare
c int;
strval varchar;
intval int;
nrows int default 30;
begin
c := dbms_sql.open_cursor();
call dbms_sql.parse(c, 'select ''ahoj'' || i, i from generate_series(1, :nrows) g(i)');
call dbms_sql.bind_variable(c, 'nrows', nrows);
call dbms_sql.define_column(c, 1, strval);
call dbms_sql.define_column(c, 2, intval);
perform dbms_sql.execute(c);
while dbms_sql.fetch_rows(c) > 0
loop
call dbms_sql.column_value(c, 1, strval);
call dbms_sql.column_value(c, 2, intval);
raise notice 'c1: %, c2: %', strval, intval;
end loop;
call dbms_sql.close_cursor(c);
end;
$$;

do
$$
declare
c int;
strval varchar;
intval int;
nrows int default 30;
begin
c := dbms_sql.open_cursor();
call dbms_sql.parse(c, 'select ''ahoj'' || i, i from generate_series(1, :nrows) g(i)');
call dbms_sql.bind_variable(c, 'nrows', nrows);
call dbms_sql.define_column(c, 1, strval);
call dbms_sql.define_column(c, 2, intval);
perform dbms_sql.execute(c);
while dbms_sql.fetch_rows(c) > 0
loop
strval := dbms_sql.column_value_f(c, 1, strval);
intval := dbms_sql.column_value_f(c, 2, intval);
raise notice 'c1: %, c2: %', strval, intval;
end loop;
call dbms_sql.close_cursor(c);
end;
$$;

create table foo
(
a int,
b varchar,
c numeric
);
```

```
do
$$
declare
c int;
begin
c := dbms_sql.open_cursor();
call dbms_sql.parse(c, 'insert into foo values(:a, :b, :c)');
for i in 1..100
loop
call dbms_sql.bind_variable(c, 'a', i);
call dbms_sql.bind_variable(c, 'b', 'Ahoj ' || i);
call dbms_sql.bind_variable(c, 'c', i + 0.033);
perform dbms_sql.execute(c);
end loop;
end;
$$;

select *
from foo
order by a;
truncate foo;

do
$$
declare
c int;
begin
c := dbms_sql.open_cursor();
call dbms_sql.parse(c, 'insert into foo values(:a, :b, :c)');
for i in 1..100
loop
perform dbms_sql.bind_variable_f(c, 'a', i);
perform dbms_sql.bind_variable_f(c, 'b', 'Ahoj ' || i);
perform dbms_sql.bind_variable_f(c, 'c', i + 0.033);
perform dbms_sql.execute(c);
end loop;
end;
$$;

select *
from foo
order by a;
truncate foo;


do
$$
declare
c int;
a int[];
b varchar[];
ca numeric[];
begin
c := dbms_sql.open_cursor();
call dbms_sql.parse(c, 'insert into foo values(:a, :b, :c)');
a := ARRAY [1, 2, 3, 4, 5];
b := ARRAY ['Ahoj', 'Nazdar', 'Bazar'];
```

```
ca := ARRAY [3.14, 2.22, 3.8, 4];

call dbms_sql.bind_array(c, 'a', a);
call dbms_sql.bind_array(c, 'b', b);
call dbms_sql.bind_array(c, 'c', ca);
raise notice 'inserted rows %', dbms_sql.execute(c);
end;
$$;

select *
from foo
order by a;
truncate foo;

do
$$
declare
c int;
a int[];
b varchar[];
ca numeric[];
begin
c := dbms_sql.open_cursor();
call dbms_sql.parse(c, 'insert into foo values(:a, :b, :c)');
a := ARRAY [1, 2, 3, 4, 5];
b := ARRAY ['Ahoj', 'Nazdar', 'Bazar'];
ca := ARRAY [3.14, 2.22, 3.8, 4];

call dbms_sql.bind_array(c, 'a', a, 2, 3);
call dbms_sql.bind_array(c, 'b', b, 3, 4);
call dbms_sql.bind_array(c, 'c', ca);
raise notice 'inserted rows %', dbms_sql.execute(c);
end;
$$;

select *
from foo
order by a;
truncate foo;

do
$$
declare
c int;
a int[];
b varchar[];
ca numeric[];
begin
c := dbms_sql.open_cursor();
call dbms_sql.parse(c, 'select i, ''Ahoj'' || i, i + 0.003 from generate_series(1, 35) g(i)');
call dbms_sql.define_array(c, 1, a, 10, 1);
call dbms_sql.define_array(c, 2, b, 10, 1);
call dbms_sql.define_array(c, 3, ca, 10, 1);

perform dbms_sql.execute(c);
while dbms_sql.fetch_rows(c) > 0
loop
```

```
call dbms_sql.column_value(c, 1, a);
call dbms_sql.column_value(c, 2, b);
call dbms_sql.column_value(c, 3, ca);
raise notice 'a = %', a;
raise notice 'b = %', b;
raise notice 'c = %', ca;
end loop;
call dbms_sql.close_cursor(c);
end;
$$;
```

DBMS_STATS

最近更新时间: 2024-06-12 15:06:00

DBMS_STATS能良好地估计统计数据(尤其是针对较大的分区表)，并能获得更好的统计结果,最终制定出速度更快的SQL执行计划。包含以下接口：

接口	描述
GATHER_DATABASE_STATS	分析数据库，包括所有用户对象和系统对象
GATHER_TABLE_STATS	分析表
GET_COLUMN_STATS	取得列的统计信息
GET_INDEX_STATS	取得索引的统计信息
GET_TABLE_STATS	取得表的统计信息

示例：

```
CREATE EXTENSION IF NOT EXISTS tbase_oracle_package_function;

create user godlike_dbms_stats superuser;
create user joe;
create user no_privilege;

\c postgres godlike_dbms_stats
grant usage on schema dbms_stats to no_privilege;
grant usage on schema dbms_stats to joe;

\c postgres joe
-- table joe_t
create table joe_t (id integer not null PRIMARY KEY, test integer);
create index joe_t_test_idx on joe_t(test);
insert into joe_t SELECT generate_series(1,1000) as key, (random()*(10^3))::integer;

\c postgres joe
exec dbms_stats.gather_table_stats(ownname => 'joe',tabname => 'joe_t');
exec dbms_stats.get_table_stats(ownname => 'joe',tabname => 'joe_t');
exec dbms_stats.get_column_stats('joe', 'joe_t', 'test');
exec dbms_stats.get_index_stats('joe', 'joe_t_test_idx');

-- table joe_t_p
create table joe_t_p (id integer not null PRIMARY KEY, test integer) partition by range (id) begin (1) step (5) partitions (200) distribute by shard(id);
create index joe_t_p_test_idx on joe_t_p(test);
insert into joe_t_p SELECT generate_series(1,1000) as key, (random()*(10^3))::integer;

exec dbms_stats.gather_database_stats();
exec dbms_stats.get_table_stats(ownname => 'joe',tabname => 'joe_t_p');
exec dbms_stats.get_column_stats('joe', 'joe_t_p', 'test');
exec dbms_stats.get_index_stats('joe', 'joe_t_p_test_idx');
exec dbms_stats.get_table_stats('joe', 'joe_t_p', 'joe_t_p_part_0');
exec dbms_stats.get_index_stats('joe', 'joe_t_p_test_idx', 'joe_t_p_test_idx_part_0');
```

```
-- clean
\c postgres godlike_dbms_stats
drop table joe_t;
drop table joe_t_p;
REVOKE usage ON schema dbms_stats FROM joe;
REVOKE usage ON schema dbms_stats FROM no_privilege;
drop user joe;
drop user no_privilege;
```

DBMS_UTILITY

最近更新时间: 2024-06-12 15:06:00

** FORMAT_CALL_STACK**

这个内置函数返回一个格式化的字符串，它显示了执行调用堆栈，直至此函数的调用点处的所有过程或者函数的调用顺序。 示例：

```
create or replace function dbms_uti_func() returns text as
declare
v_str1 text;
v_str2 text;
v_str3 text;
v_str4 text;
begin
v_str1 := dbms_utility.format_call_stack();
select regexp_replace(v_str1,'[ 0-9a-fA-F]{4}[0-9a-fA-F]{4}',' 0','g') into v_str1;
select regexp_replace(v_str1,'[45()]','','g') into v_str1;
v_str2 := dbms_utility.format_call_stack('o');
select regexp_replace(v_str2,'[ 0-9a-fA-F]{4}[0-9a-fA-F]{4}',' 0','g') into v_str2;
select regexp_replace(v_str2,'[45()]','','g') into v_str2;
v_str3 := dbms_utility.format_call_stack('p');
select regexp_replace(v_str3,'[ 0-9a-fA-F]{4}[0-9a-fA-F]{4}',' 0','g') into v_str3;
select regexp_replace(v_str3,'[45()]','','g') into v_str3;
v_str4 := dbms_utility.format_call_stack('s');
select regexp_replace(v_str4,'[ 0-9a-fA-F]{4}[0-9a-fA-F]{4}',' 0','g') into v_str4;
select regexp_replace(v_str4,'[45()]','','g') into v_str4;
return 'v_str1:
'||v_str1 || '
v_str2:
'|| v_str2|| '
v_str3:
'|| v_str3|| '
v_str4:
'|| v_str4;
end;
/
select dbms_uti_func();
```

GET_HASH_VALUE

这个内置函数用于计算给定字符串的散列值。 **使用方法**：DBMS_UTILITY.GET_HASH_VALUE (name VARCHAR2, base NUMBER, hash_size NUMBER) RETURN NUMBER;

参数	描述
name	需要计算其散列值的字符串。
base	需要生成散列值的起始值。

参数	描述
hash_size	所需散列表的散列值的数量。

返回值：计算出来的哈希值 例如，要获取哈希值应在1000到3047之间的字符串上的哈希值，请使用1000作为基值，2048作为hash_size值。 **使用场景：**对于指定的字符串，返回范围在[base, base+hase_size-1]的散列值 示例：

```
postgres=# select dbms_utility.get_hash_value('hello tdsql pg',1,10000) from dual;


get_hash_value
----------------
9930
(1 行记录)
```

> **注意：**
>
> - name为null值无法计算哈希，约定name参数为null时且其他参数正常，则直接返回base值
> - base不可为null，可为负值。返回值的最小值为base；返回hash值的最大值bash+hash_size-1。
> - hash_size不可为null或者0，若hash_size为负值则返回的哈希值不在哈希范围内，哈希值可能为正，也可能为负（内部运算时会把hash_size转成正值）。
> - base和hash_size的最大最小值兼容oracle，因哈希算法不同，因此哈希值不兼容oracle的哈希值。

UTL_FILE

最近更新时间: 2024-06-12 15:06:00

UTL_FILE包提供文本文件输入和输出功能。 UTL_FILE包含以下接口：

接口	描述
*FOPEN	用于打开文件
IS_OPEN	用于确定文件是否已经被打开
FCLOSE	用于关闭已经打开的文件
FCLOSE_ALL	该过程用于关闭当前打开的所有文件
GET_LINE	用于从已经打开的文件中读取行内容，行内容会被读取到输出缓冲区
GET_NEXTLINE	用于从已经打开的文件中读取下一条行内容
*PUT	用于将缓冲区内容写入到文件中。当使用PUT过程的时候，文件必须以写方式打开，在写入缓冲区之后，如果要结束行，那么可以使用NEW_LINE过程
NEW_LINE	该过程用于为文件增加行终止符
*PUT_LINE	该过程用于将文本缓冲区内容写入到文件中。当使用该过程为文件追加内容时，会自动在文件的尾部追加行终止符。
FFLUSH	用于将数据强制性写入到文件中，正常情况下，当给文件写入数据的时候，数据会被暂时的放到缓存中。过程FFLUSH用于强制将数据写入到文件中。
FREMOVE	用于删除磁盘文件
*FCOPY	用于将源文件的全部或者部分内容复制到目标文件中。
*FRENAME	该过程用于修改已经存在的文件名字，其作用于UNIX的MV命令完全相同，在修改文件名字的时候，通过指定overwrite参数可以覆盖已经存在的文件
FGETATTR	读取磁盘上的文件并返回文件的属性
FRENAME	将一个存在的文件重命名
PUTF	写入格式化的内容到文件中

示例：

```
create orreplace procedure utlfile_open_prowa() as
declare
v_count int;
v_fileintinteger;
begin
--open_node=w
v_fileint :=utl_file.fopen('/data1/tbasev5_autotest/tbaseTest/TbaseV5/pro_package/data/utl_file_dir','file_w.txt','w');
perform utl_file.put_line(v_fileint,'writefile test.');
if utl_file.is_open(v_fileint) then
perform utl_file.fclose(v_fileint);
```

```
end if;
v_fileint :=utl_file.fopen('/data1/tbasev5_autotest/tbaseTest/TbaseV5/pro_package/data/utl_file_dir','file_w.txt','w',50);
perform utl_file.new_line(v_fileint);
perform utl_file.put_line(v_fileint,'一二三四五六七八九十');
if utl_file.is_open(v_fileint) then
perform utl_file.fclose(v_fileint);
end if;
v_fileint :=utl_file.fopen('/data1/tbasev5_autotest/tbaseTest/TbaseV5/pro_package/data/utl_file_dir','file_w.txt','w',18,'SQL_
ASCII');
perform utl_file.new_line(v_fileint);
perform utl_file.put_line(v_fileint,'nice tomeet you');
if utl_file.is_open(v_fileint) then
perform utl_file.fclose(v_fileint);
end if;
v_fileint :=utl_file.fopen('/data1/tbasev5_autotest/tbaseTest/TbaseV5/pro_package/data/utl_file_dir','file_w.txt','w',31,'UTF-
8');
perform utl_file.new_line(v_fileint);
perform utl_file.put_line(v_fileint,'abcdefghijklmn opq rst uvw xyz');
if utl_file.is_open(v_fileint) then
perform utl_file.fclose(v_fileint);
end if;
v_fileint :=utl_file.fopen('/data1/tbasev5_autotest/tbaseTest/TbaseV5/pro_package/data/utl_file_dir','file_w.txt','w',88,'GB
K');
perform utl_file.new_line(v_fileint);
perform utl_file.put_line(v_fileint,'hello');
if utl_file.is_open(v_fileint) then
perform utl_file.fclose(v_fileint);
end if;
--open_mode=a
v_fileint := utl_file.fopen('/data1/tbasev5_autotest/tbaseTest/TbaseV5/pro_package/data/utl_file_dir','file_w.txt','a');
perform utl_file.put_line(v_fileint,'writefile test.');
if utl_file.is_open(v_fileint) then
perform utl_file.fclose(v_fileint);
end if;
v_fileint :=utl_file.fopen('/data1/tbasev5_autotest/tbaseTest/TbaseV5/pro_package/data/utl_file_dir','file_w.txt','a',50);
perform utl_file.new_line(v_fileint);
perform utl_file.put_line(v_fileint,'一二三四五六七八九十');
if utl_file.is_open(v_fileint) then
perform utl_file.fclose(v_fileint);
end if;
v_fileint :=utl_file.fopen('/data1/tbasev5_autotest/tbaseTest/TbaseV5/pro_package/data/utl_file_dir','file_w.txt','a',18,'SQL_A
SCII');
perform utl_file.new_line(v_fileint);
perform utl_file.put_line(v_fileint,'nice tomeet you');
if utl_file.is_open(v_fileint) then
perform utl_file.fclose(v_fileint);
end if;
v_fileint :=utl_file.fopen('/data1/tbasev5_autotest/tbaseTest/TbaseV5/pro_package/data/utl_file_dir','file_w.txt','a',31,'UTF-
8');
perform utl_file.new_line(v_fileint);
perform utl_file.put_line(v_fileint,'abcdefghijklmn opq rst uvw xyz');
if utl_file.is_open(v_fileint) then
perform utl_file.fclose(v_fileint);
end if;
v_fileint := utl_file.fopen('/data1/tbasev5_autotest/tbaseTest/TbaseV5/pro_package/data/utl_file_dir','file_w.txt','a',88,'GB
K');
perform utl_file.new_line(v_fileint);
```

```
perform utl_file.put_line(v_fileint,'hello');
if utl_file.is_open(v_fileint) then
perform utl_file.fclose(v_fileint);
end if;
exception
when others then
raise notice 'EXP: something wrong.';
end;
/
callutlfile_open_prowa();
```

UTL_RAW

最近更新时间: 2024-06-12 15:06:00

UTL_RAW 系统包提供用于操作 RAW 数据类型的 SQL 函数。普通的 SQL 函数不在RAW 数据上操作，而 PL 不允许在 RAW 和 CHAR 数据类型之间重载。

存储过程/函数	描述
BIT_AND	对 RAW r1 和 RAW r2 中的值执行按位逻辑"与"，并返回原始的"与"结果。
BIT_COMPLEMENT	对 RAW r 中的值执行按位逻辑"补码"，并返回"补码"结果 RAW。
BIT_OR	对 RAW r1 和 RAW r2 中的值执行按位逻辑"或"，并返回原始的"或"结果。
BIT_XOR	对 RAW r1 和 RAW r2 中的值执行逐位逻辑"异或"，并返回原始的"异或"结果。
CAST_TO_RAW	将 VARCHAR2 值转换为 RAW 值。
CAST_TO_VARCHAR2	将RAW 值转换为 VARCHAR2 值。
COPIES	返回串联在一起的 r 的 n 个副本。
COMPARE	比较 RAW r1 和 RAW r2。
CONCAT	将最多 12 个 RAW 串连到一个 RAW 中。
LENGTH	返回 RAW r 的字节长度。
REVERSE	从端到端反转 RAW r 中的字节序列。
SUBSTR	返回 len 字节数，从 RAW r 的 pos 位开始计算。

示例：

```
select utl_raw.bit_and('ABC', 'ABC') fromdual;
select utl_raw.bit_and('ABC', '') fromdual;
select utl_raw.bit_and('', 'ABC') fromdual;
select utl_raw.bit_and('', '') from dual;
select utl_raw.bit_and(null, '') from dual;
select utl_raw.bit_and(null, null) fromdual;
select utl_raw.bit_and('f000', 'EE10') fromdual;
select utl_raw.bit_and('000', '111') fromdual;
select utl_raw.bit_and('1111', '111') fromdual;

select utl_raw.bit_or('ABC', 'ABC') fromdual;
select utl_raw.bit_or('ABC', '') from dual;
select utl_raw.bit_or('', 'ABC') from dual;
select utl_raw.bit_or('', '') from dual;
select utl_raw.bit_or(null, '') from dual;
select utl_raw.bit_or(null, null) fromdual;
select utl_raw.bit_or('f000', 'EE10') fromdual;
select utl_raw.bit_or('000', '111') fromdual;
select utl_raw.bit_or('1111', '111') fromdual;

select utl_raw.bit_xor('ABC', 'ABC') fromdual;
```

```
select utl_raw.bit_xor('ABC', '') fromdual;
select utl_raw.bit_xor('', 'ABC') fromdual;
select utl_raw.bit_xor('', '') from dual;
select utl_raw.bit_xor(null, '') from dual;
select utl_raw.bit_xor(null, null) fromdual;
select utl_raw.bit_xor('f000', 'EE10') fromdual;
select utl_raw.bit_xor('000', '111') fromdual;
select utl_raw.bit_xor('1111', '111') fromdual;

SELECT utl_raw.bit_complement('0102F3')FROM dual;
SELECTutl_raw.bit_complement('000000000000000000000000000') FROM dual;
SELECTutl_raw.bit_complement('FFFFFFFFFFFFFFFFFFFFFFFFFFFffffff') FROM dual;
SELECT utl_raw.bit_complement('0246813579')FROM dual;
SELECTutl_raw.bit_complement('ABCDEF13579') FROM dual;
SELECT utl_raw.bit_complement('') FROMdual;
SELECT utl_raw.bit_complement(null) FROMdual;


select utl_raw.substr('1234567810', -4, 3)from dual;
select utl_raw.substr('1234567810', -5, 5)from dual;
select utl_raw.substr('1234567810', -5)from dual;
select utl_raw.substr('1', 0) from dual;
select utl_raw.substr('1', '1') from dual;
select utl_raw.substr('ABCD', '1E0') fromdual;
select utl_raw.substr('ABCD', '1E-1') fromdual;
select utl_raw.substr('ABCD', '0.2') fromdual;

select utl_raw.length(repeat('AB', 32768))from dual;
select utl_raw.length('') from dual;
select utl_raw.length(1E10||'') from dual;
select utl_raw.length(null) from dual;


select utl_raw.reverse('ABC123') from dual;
select utl_raw.reverse('ABCD') from dual;
select utl_raw.reverse('A') from dual;
select utl_raw.reverse(123E3||'') fromdual;

select utl_raw.concat('ABC') from dual;
select utl_raw.concat('ABC', '') from dual;
select utl_raw.concat('ABC', 99E2||'') fromdual;
select utl_raw.concat('', 'ABC') from dual;
select utl_raw.concat('', '') from dual;
select utl_raw.concat('') from dual;
select utl_raw.concat() from dual;
select utl_raw.concat('', null) from dual;
select utl_raw.concat(null, null) fromdual;
select utl_raw.concat('', '') from dual;
select utl_raw.concat(repeat('ABCD', 2000),repeat('ABCD', 2000)) from dual;

select utl_raw.compare('912', '') fromdual;
select utl_raw.compare('912', null) fromdual;
select utl_raw.compare('', null) from dual;
select utl_raw.compare(null, null) from dual;
select utl_raw.compare('912', '91245') fromdual;
select utl_raw.compare('ABC', 'ABCDEFF')from dual;
select utl_raw.compare('ABCD', 'ABCDEFF')from dual;
```

```
select utl_raw.COPIES('AB', 10) from dual;
select utl_raw.COPIES('AB123', 10) fromdual;


select utl_raw.cast_to_varchar2('1') fromdual;
select utl_raw.cast_to_varchar2('') fromdual;
select utl_raw.cast_to_varchar2(null) fromdual;
select utl_raw.cast_to_varchar2('6666')from dual;
selectutl_raw.cast_to_varchar2('E4BDA0E5A5BD') from dual;

select utl_raw.CAST_TO_RAW('ff') from dual;
select utl_raw.CAST_TO_RAW(' ') from dual;
select utl_raw.CAST_TO_RAW('') from dual;
select utl_raw.CAST_TO_RAW(null) from dual;
select utl_raw.CAST_TO_RAW(123) from dual;
select utl_raw.CAST_TO_RAW(123E4) fromdual;
select utl_raw.CAST_TO_RAW('嗯嗯ABC123.5') from dual;
```

系统视图
all_arguments

最近更新时间: 2024-06-12 15:06:00

列出了当前用户可以访问的函数和过程的参数。

名称	类型	说明
owner	varchar2(128)	对象所有者
object_name	varchar2(128)	过程或函数名称
package_name	varchar2(128)	包名称
object_id	number	对象的对象编号
overload	varchar2(40)	无效
subprogram_id	number	无效
argument_name	varchar2(128)	参数名。空参数名称用于表示函数返回
position	number	表示在参数列表中的位置，或 0 表示函数返回值
sequence	number	定义参数的顺序。 参数序列从 1 开始。返回类型在前，然后是每个参数
data_level	number	复合类型参数的嵌套深度
data_type	varchar2(30)	参数数据类型
defaulted	varchar2(1)	指定参数是否默认
default_value	long	保留
default_length	number	保留
in_out	varchar2(9)	参数方向 - IN - OUT - INOUT
data_length	number	无效
data_precision	number	无效
data_scale	number	无效
radix	number	无效
character_set_name	varchar2(44)	无效
type_owner	varchar2(128)	无效
type_name	varchar2(128)	无效

名称	类型	说明
type_subname	varchar2(128)	无效
type_link	varchar2(128)	无效
type_object_type	varchar2(7)	无效
pls_type	varchar2(128)	无效
char_length	number	无效
char_used	varchar2(1)	无效
origin_con_id	varchar2(256)	无效

all_users

最近更新时间: 2024-06-12 15:06:00

列出了对当前用户可见的数据库的所有用户。

名称	类型	说明
username	name	用户名
user_id	oid	用户 ID
created	text	创建用户的时间

all_tab_columns

最近更新时间: 2024-06-12 15:06:00

所有表和视图中的所有列的信息

名称	类型	说明
owner	name	所属用户
table_schema	name	表或视图所在 schema 的名称。
table_name	name	列所在的表或视图的名称。
column_name	name	列的名称。
data_type	text	列的数据类型。
data_type_owner	name	数据类型所有者
data_length	integer	文本列的长度。
data_precision	integer	NUMBER 列的精度（位数）。
data_scale	integer	NUMBER 列的小数位数。
nullable	text	列是否可为空，可能值包括： Y：列可为空。 N：列不可为空。
column_id	numeric	表中列的相对位置。
data_default	character varying	分配给列的默认值。
num_distinct	numeric(38,0)	表格每列中不同值的数量
low_value	text	对于超过三行的表，第二低和第二高的值
high_value	text	N/A
density	text	N/A
num_nulls	real	列中的空值数
num_buckets	integer	列的直方图中的桶数
last_analyzed	text	最近分析此列的日期
sample_size	text	用于分析此列的样本量
character_set_name	text	字符集的名称
char_col_decl_length	integer	字符集的长度
global_stats	text	N/A
user_stats	text	N/A

名称	类型	说明
avg_col_len	integer	列的平均长度（以字节为单位）
char_length	integer	以字符显示列的长度
char_used	text	N/A
v80_fmt_image	text	N/A
data_upgraded	text	N/A
histogram	text	N/A

all_col_comments

最近更新时间: 2024-06-12 15:06:00

ALL_COL_COMMENTS所有表的字段注释 。 all_col_comments名称说明

名称	类型	说明
OWNER	NAME	所有者的名称。
SCHEMA_NAME	NAME	模式名称。
TABLE_NAME	NAME	对象名。
COLUMN_NAME	NAME	列名。
COMMENT	TEXT	列注释。

all_constraints

最近更新时间: 2024-06-12 15:06:00

所有表中所有约束信息。 all_constraints名称说明

名称	类型	说明
owner	TEXT	约束所有者的名称。
constraint_schema	TEXT	约束所属 schema 的名称。
constraint_name	TEXT	约束的名称。
constraint_type	TEXT	约束类型。可能值包括： - C：检查约束 - F：外键约束 - P：主键约束 - U：唯一键约束 - R：引用完整性约束 - V：视图上的约束 - O：具有只读属性，在视图上
table_schema	NAME	约束所属表所在schema。
table_name	TEXT	约束所属表的名称。
search_condition	TEXT	应用于检查约束的搜索条件。
r_owner	TEXT	引用约束引用的表的所有者。
r_constraint_schema	-	引用表的约束定义的模式名称。
r_constraint_name	TEXT	引用表的约束定义的名称。
delete_rule	TEXT	引用约束的删除规则。可能值包括： - C：级联 - R：限制 - N：无操作
is_deferrable	BOOLEAN	指定了约束是否可延迟（T 或 F）。
deferred	BOOLEAN	指定约束是否已延迟（T 或 F）。

all_synonyms

最近更新时间: 2024-06-12 15:06:00

描述了当前用户可以访问的同义词。

名称	类型	说明
owner	name	同义词的拥有者
synonym_name	name	同义词名字
table_owner	text	同义词引用对象的名字
table_name	name	同义词引用的对象名字
db_link	name	如果引用的是 DB Link，则表示 DB Link 的名字

all_indexes

最近更新时间: 2024-06-12 15:06:00

描述了当前用户可访问的表上的索引

名称	类型	说明
owner	name	所有者
index_nsp	name	
index_name	name	索引名称
index_type	name	索引类型
table_owner	name	索引所属表的所有者
table_nsp	name	
table_name	name	表名
table_type	text	索引所属表名称
compression	text	索引的压缩类型
prefix_length	text	索引前缀长度
tablespace_name	name	索引的表空间的名称
ini_trans	text	初始交易数量
max_trans	text	最大交易次数
initial_extent	text	初始范围的大小
next_extent	text	辅助范围的大小
min_extents	text	段中允许的最小区数
max_extents	text	段中允许的最大区数
pct_increase	text	范围大小的百分比增加
pct_threshold	integer	每个索引条目允许的块空间的阈值百分比
include_column	text	要包含在索引组织表主键（非溢出）索引中的最后一列的列ID
freelists	text	分配给此段的进程空闲列表的数量
freelist_groups	text	分配给此段的空闲列表组的数量
pct_free	text	块中可用空间的最小百分比
logging	text	表示对索引的修改是否被记录了日志
blevel	integer	表示 B*-Tree 的深度

名称	类型	说明
leaf_blocks	integer	索引中的叶块数
distinct_keys	integer	不同索引值的数量
avg_leaf_blocks_per_key	integer	索引中每个不同值出现的平均叶块数
avg_data_blocks_per_key	integer	表中由索引中的不同值指向的平均数据块数舍入为最接近的整数
clustering_factor	integer	表示基于索引值的表中行的顺序数
status	text	状态
num_rows	real	索引中的行数
sample_size	integer	索引的样本的大小
last_analyzed	text	最近分析此索引的日期
degree	text	扫描索引的每个实例的线程数
instances	text	要扫描索引的实例数
partitioned	text	指示索引是否已分区
temporary	text	指示索引是否在临时表上
generated	text	指示索引的名称是否是系统生成的
secondary	text	指示索引是否是辅助对象
buffer_pool	text	用于索引块的缓冲池
user_stats	text	统计信息是否由用户直接输入
duration	text	指示临时表的持续时间
pct_direct_access	integer	对于索引组织表上的二级索引
ityp_owner	text	对于域索引，indextype的所有者
ityp_name	text	对于域索引，indextype的名称
parameters	text	对于域索引，参数字符串
global_stats	text	对于分区索引，指定是通过整体分析索引收集统计信息，还是从基础索引分区和子分区的统计信息中估算统计信息
domidx_status	text	域索引的状态
domidx_opstatus	text	域索引上的操作状态
funcidx_status	text	基于函数的索引的状态
join_index	text	指示索引是否为连接索引
iot_redundant_pkey_elim	text	指示是否从索引组织表上的二级索引中删除冗余主键列

名称	类型	说明
dropped	text	指示索引是否已被删除并且是否在回收站中

all_ind_columns

最近更新时间: 2024-06-12 15:06:00

描述当前用户可访问的所有表的索引列

名称	类型	说明
index_owner	name	索引列的所有者
Index_nsp	name	
index_name	name	索引名称
table_owner	name	表的所有者
table_nsp	name	
table_name	text	表名
column_name	text	列名
column_position	smallint	列的编号
column_length	smallint	列长
char_length	integer	列的数据类型如果为char，表示字节数。否则为0
descend	text	默认DESC。指示列是按降序（DESC）还是按升序（ASC）排序

all_cons_columns

最近更新时间: 2024-06-12 15:06:00

提供所有表中，约束包含的所有列的信息。 all_cons_columns名称说明

名称	类型	说明
owner	TEXT	约束所有者的用户名。
schema_name	TEXT	约束所属 schema 的名称。
constraint_name	TEXT	约束的名称。
table_name	TEXT	约束所属表的名称。
column_name	TEXT	约束中引用的列的名称。
position	SMALLINT	列在对象定义中的位置。

dba_ind_columns

最近更新时间: 2024-06-12 15:06:00

描述了数据库中所有表和集群的所有索引的列。它的列与 ALL_IND_COLUMNS 中的列相同。

名称	类型	说明
index_owner	name	索引列的所有者
index_nsp	name	
index_name	name	索引名称
table_owner	name	表的所有者
table_nsp	name	
table_name	name	表名
column_name	name	列名
column_position	smallint	列的编号
column_length	smallint	列长
char_length	integer	列的数据类型如果为char，表示字节数。否则为0
descend	text	默认DESC。指示列是按降序（DESC）还是按升序（ASC）排序

dba_constraints

最近更新时间: 2024-06-12 15:06:00

描述了数据库中所有表的所有约束定义。它的列与 ALL_CONSTRAINTS 中的列相同。

名称	类型	说明
owner	name	约束的拥有者
constraint_schema	name	
constraint_name	name	约束名字
constraint_type	text	索引类型
table_schema	name	
table_name	name	约束所在的表的名字
r_owner	name	被引用约束的拥有者
r_constraint_schema	name	
r_constraint_name	name	被引用约束的名字
delete_rule	text	外键约束的级联删除规则
is_deferrable	boolean	表示约束是 DEFERRABLE 还是 NOT DEFERRABLE
deferred	boolean	表示约束是 DEFERRED 还是 IMMEDIATE

dba_indexes

最近更新时间: 2024-06-12 15:06:00

DBA_INDEXES描述了数据库中的所有索引。要收集此视图的统计信息，请使用 DBMS_STATS 包。此视图支持并行分区索引扫描。它的列与 ALL_INDEXES中的列相同。

名称	类型	说明
owner	name	所有者
index_nsp	name	
index_name	name	索引名称
index_type	name	索引类型
table_owner	name	索引所属表的所有者
table_nsp	name	
table_name	name	表名
table_type	text	索引所属表名称
compression	text	索引的压缩类型
prefix_length	text	索引前缀长度
tablespace_name	name	索引的表空间的名称
ini_trans	text	初始交易数量
max_trans	text	最大交易次数
initial_extent	text	初始范围的大小
next_extent	text	辅助范围的大小
min_extents	text	段中允许的最小区数
max_extents	text	段中允许的最大区数
pct_increase	text	范围大小的百分比增加
pct_threshold	integer	每个索引条目允许的块空间的阈值百分比
include_column	text	要包含在索引组织表主键（非溢出）索引中的最后一列的列ID
freelists	text	分配给此段的进程空闲列表的数量
freelist_groups	text	分配给此段的空闲列表组的数量
pct_free	text	块中可用空间的最小百分比
logging	text	表示对索引的修改是否被记录了日志
blevel	integer	表示 B*-Tree 的深度

名称	类型	说明
leaf_blocks	integer	索引中的叶块数
distinct_keys	integer	不同索引值的数量
avg_leaf_blocks_per_key	integer	索引中每个不同值出现的平均叶块数
avg_data_blocks_per_key	integer	表中由索引中的不同值指向的平均数据块数舍入为最接近的整数
clustering_factor	integer	表示基于索引值的表中行的顺序数
status	text	状态
num_rows	real	索引中的行数
sample_size	integer	索引的样本的大小
last_analyzed	text	最近分析此索引的日期
degree	text	扫描索引的每个实例的线程数
instances	text	要扫描索引的实例数
partitioned	text	指示索引是否已分区
temporary	text	指示索引是否在临时表上
generated	text	指示索引的名称是否是系统生成的
secondary	text	指示索引是否是辅助对象
buffer_pool	text	用于索引块的缓冲池
user_stats	text	统计信息是否由用户直接输入
duration	text	指示临时表的持续时间
pct_direct_access	integer	对于索引组织表上的二级索引
ityp_owner	text	对于域索引，indextype的所有者
ityp_name	text	对于域索引，indextype的名称
parameters	text	对于域索引，参数字符串
global_stats	text	对于分区索引，指定是通过整体分析索引收集统计信息，还是从基础索引分区和子分区的统计信息中估算统计信息
domidx_status	text	域索引的状态
domidx_opstatus	text	域索引上的操作状态
funcidx_status	text	基于函数的索引的状态
join_index	text	指示索引是否为连接索引
iot_redundant_pkey_elim	text	指示是否从索引组织表上的二级索引中删除冗余主键列

名称	类型	说明
dropped	text	指示索引是否已被删除并且是否在回收站中

dba_synonyms

最近更新时间: 2024-06-12 15:06:00

描述了数据库中的所有同义词。它的列与 ALL_SYNONYMS 中的列相同

名称	类型	说明
owner	name	同义词的拥有者
synonym_name	name	同义词名字
table_owner	text	同义词引用对象的名字
table_name	name	同义词引用的对象名字
db_link	name	如果引用的是 DB Link，则表示 DB Link 的名字

dba_tab_columns

最近更新时间: 2024-06-12 15:06:00

DBA_TAB_COLUMNS描述了数据库中所有表、视图和集群的列。

名称	类型	说明
owner	name	所属用户
table_schema	name	表或视图所在 schema 的名称。
table_name	name	列所在的表或视图的名称。
column_name	name	列的名称。
data_type	text	列的数据类型。
data_type_owner	name	数据类型所有者
data_length	integer	文本列的长度。
data_precision	integer	NUMBER 列的精度（位数）。
data_scale	integer	NUMBER 列的小数位数。
nullable	text	列是否可为空，可能值包括： Y：列可为空。 N：列不可为空。
column_id	smallint	表中列的相对位置。
data_default	text	分配给列的默认值。
num_distinct	numeric(38,0)	表格每列中不同值的数量
low_value	text	对于超过三行的表，第二低和第二高的值
high_value	text	N/A
density	text	N/A
num_nulls	real	列中的空值数
num_buckets	integer	列的直方图中的桶数
last_analyzed	text	最近分析此列的日期
sample_size	text	用于分析此列的样本量
character_set_name	text	字符集的名称
char_col_decl_length	integer	字符集的长度
global_stats	text	N/A
user_stats	text	N/A

名称	类型	说明
avg_col_len	integer	列的平均长度（以字节为单位）
char_length	integer	以字符显示列的长度
char_used	text	N/A
v80_fmt_image	text	N/A
data_upgraded	text	N/A
histogram	text	N/A

dba_users

最近更新时间: 2024-06-12 15:06:00

DBA_USERS描述了数据库的所有用户。

名称	类型	说明
username	name	用户名
user_id	oid	用户的身份证号
password	text	此列不建议,使用 AUTHENTICATION_TYPE 列
account_status	text	帐户状态
lock_date	timestamp with time zone	如果帐户状态被锁定，则锁定帐户的日期
expiry_date	timestamp with time zone	账户到期日期
default_tablespace	text	数据的默认表空间
temporary_tablespace	text	临时表的默认表空间的名称或表空间组的名称
created	text	用户创建日期
profile	name	用户资源配置文件名称
initial_rsrc_consumer_group	text	用户的初始资源使用者组
external_name	text	用户外部名称

user_indexes

最近更新时间: 2024-06-12 15:06:00

USER_INDEXES描述当前用户拥有的索引。要收集此视图的统计信息，请使用 DBMS_STATS 包。此视图支持并行分区索引扫描。它的列（除了 OWNER）与 ALL_INDEXES 中的列相同。

名称	类型	说明
index_nsp	name	
index_name	name	索引名称
index_type	name	索引类型
table_owner	name	索引所属表的所有者
table_nsp	name	
table_name	name	表名
table_type	text	索引所属表名称
compression	text	索引的压缩类型
prefix_length	text	索引前缀长度
tablespace_name	name	索引的表空间的名称
ini_trans	text	初始交易数量
max_trans	text	最大交易次数
initial_extent	text	初始范围的大小
next_extent	text	辅助范围的大小
min_extents	text	段中允许的最小区数
max_extents	text	段中允许的最大区数
pct_increase	text	范围大小的百分比增加
pct_threshold	integer	每个索引条目允许的块空间的阈值百分比
include_column	text	要包含在索引组织表主键（非溢出）索引中的最后一列的列ID
freelists	text	分配给此段的进程空闲列表的数量
freelist_groups	text	分配给此段的空闲列表组的数量
pct_free	text	块中可用空间的最小百分比
logging	text	表示对索引的修改是否被记录了日志
blevel	integer	表示 B*-Tree 的深度
leaf_blocks	integer	索引中的叶块数

名称	类型	说明
distinct_keys	integer	不同索引值的数量
avg_leaf_blocks_per_key	integer	索引中每个不同值出现的平均叶块数
avg_data_blocks_per_key	integer	表中由索引中的不同值指向的平均数据块数舍入为最接近的整数
clustering_factor	integer	表示基于索引值的表中行的顺序数
status	text	状态
num_rows	real	索引中的行数
sample_size	integer	索引的样本的大小
last_analyzed	text	最近分析此索引的日期
degree	text	扫描索引的每个实例的线程数
instances	text	要扫描索引的实例数
partitioned	text	指示索引是否已分区
temporary	text	指示索引是否在临时表上
generated	text	指示索引的名称是否是系统生成的
secondary	text	指示索引是否是辅助对象
buffer_pool	text	用于索引块的缓冲池
user_stats	text	统计信息是否由用户直接输入
duration	text	指示临时表的持续时间
pct_direct_access	integer	对于索引组织表上的二级索引
ityp_owner	text	对于域索引，indextype的所有者
ityp_name	text	对于域索引，indextype的名称
parameters	text	对于域索引，参数字符串
global_stats	text	对于分区索引，指定是通过整体分析索引收集统计信息，还是从基础索引分区和子分区的统计信息中估算统计信息
domidx_status	text	域索引的状态
domidx_opstatus	text	域索引上的操作状态
funcidx_status	text	基于函数的索引的状态
join_index	text	指示索引是否为连接索引
iot_redundant_pkey_elim	text	指示是否从索引组织表上的二级索引中删除冗余主键列
dropped	text	指示索引是否已被删除并且是否在回收站

user_synonyms

最近更新时间: 2024-06-12 15:06:00

描述了私有同义词（当前用户拥有的同义词）。它的列（除了 OWNER）与 ALL_SYNONYMS 中的列相同。

名称	类型	说明
synonym_name	name	同义词的拥有者
table_owner	text	同义词名字
table_name	name	同义词引用对象的名字
db_link	name	同义词引用的对象名字

user_cons_columns

最近更新时间: 2024-06-12 15:06:00

提供当前用户拥有的表中，约束包含的所有列的信息。

名称	类型	说明
constraint _schema	name	约束所属 schema 的名称。
constraint_name	name	约束的名称。
table_schema	name	约束所属表的模式名称
table_name	name	约束所属表的名称。
column_name	name	约束中引用的列的名称。
position	smallint	列在对象定义中的位置。

user_tab_columns

最近更新时间: 2024-06-12 15:06:00

提供当前用户拥有的表和视图中的所有列的信息。

名称	类型	说明
table_schema	name	表或视图所在 schema 的名称。
table_name	name	列所在的表或视图的名称。
column_name	name	列的名称。
data_type	text	列的数据类型。
data_type_owner	name	数据类型所有者
data_length	integer	文本列的长度。
data_precision	integer	NUMBER 列的精度（位数）。
data_scale	integer	NUMBER 列的小数位数。
nullable	text	列是否可为空，可能值包括： Y：列可为空。 N：列不可为空。
column_id	smallint	表中列的相对位置。
data_default	text	分配给列的默认值。
num_distinct	numeric(38,0)	表格每列中不同值的数量
low_value	text	对于超过三行的表，第二低和第二高的值
high_value	text	N/A
density	text	N/A
num_nulls	real	列中的空值数
num_buckets	integer	列的直方图中的桶数
last_analyzed	text	最近分析此列的日期
sample_size	text	用于分析此列的样本量
character_set_name	text	字符集的名称
char_col_decl_length	integer	字符集的长度
global_stats	text	N/A
user_stats	text	N/A
avg_col_len	integer	列的平均长度（以字节为单位）

名称	类型	说明
char_length	integer	以字符显示列的长度
char_used	text	N/A
v80_fmt_image	text	N/A
data_upgraded	text	N/A
histogram	text	N/A

user_ind_columns

最近更新时间: 2024-06-12 15:06:00

描述了当前用户拥有的索引的列以及当前用户拥有的表的索引列。

名称	类型	说明
index_owner	name	索引列的所有者
index_nsp	name	
index_name	name	索引名称
table_owner	name	表的所有者
table_nsp	name	
table_name	name	表名
column_name	name	列名
column_position	smallint	列的编号
column_length	smallint	列长
char_length	integer	列的数据类型如果为char，表示字节数。否则为0
descend	text	默认DESC。指示列是按降序（DESC）还是按升序（ASC）排序

user_constraints

最近更新时间: 2024-06-12 15:06:00

用户表约束提供当前用户拥有的表中放置的所有约束的信息。。 user_constraints名称说明

名称	类型	说明
constraint_schema	TEXT	约束所属 schema 的名称。
constraint_name	TEXT	约束的名称。
constraint_type	TEXT	约束类型。可能值包括： - C：检查约束 - F：外键约束 - P：主键约束 - U：唯一键约束 - R：引用完整性约束 - V：视图上的约束 - O：具有只读属性，在视图上
table_schema	NAME	约束所属表所在schema。
table_name	TEXT	约束所属表的名称。
search_condition	TEXT	应用于检查约束的搜索条件。
r_owner	TEXT	引用约束引用的表的所有者。
r_constraint_schema	-	引用表的约束定义的模式名称。
r_constraint_name	TEXT	引用表的约束定义的名称。
delete_rule	TEXT	引用约束的删除规则。可能值包括： - C：级联 - R：限制 - N：无操作
is_deferrable	BOOLEAN	指定了约束是否可延迟（T 或 F）。
Deferred	BOOLEAN	指定约束是否已延迟（T 或 F）。

user_col_comments

最近更新时间: 2024-06-12 15:06:00

用户表的字在当前用户拥有的表和视图的列上显示注释。其列（OWNER除外）与ALL_COL_COMMENTS中的列相同。注释。

user_col_comments名称说明

名称	类型	说明
SCHEMA_NAME	NAME	模式名称。
TABLE_NAME	NAME	对象名。
COLUMN_NAME	NAME	列名。
COMMENTS	TEXT	列注释。

user_users

最近更新时间: 2024-06-12 15:06:00

描述当前用户信息

名称	类型	说明
username	name	用户名
user_id	oid	用户身份证号
account_status	text	帐户状态
lock_date	Timestamp with time zone	如果帐户状态为锁定，则帐户被锁定的日期
expiry_date	Timestamp with time zone	账户到期日期
default_tablespace	text	数据的默认表空间
temporary_tablespace	text	临时表的默认表空间的名称或表空间组的名称
created	text	用户创建日期
initial_rsrc_consumer_group	text	用户的初始资源使用者组
external_name	text	用户外部名称

函数
单行函数
数值函数

最近更新时间: 2024-06-12 15:06:00

函数名	功能描述
ABS	返回指定数值表达式的绝对值（正值）的数学函数。
ACOS	返回以弧度表示的角，其余弦为指定 NUMBER表达式，也称为反余弦。
ASIN	求反正弦值。
ATAN	用于求反正切值。
ATAN2	返回 y 和 x 的反正切值，即返回的是原点至点 (x,y) 与 x 轴的夹角。
BITAND	运算符按位进行“与”操作。输入和输出类型均为 INT 整型，且类型一致。
CEIL	返回值大于等于数值 numeric_expression 的最小整数。
COS	用于计算参数角度的余弦值。
EXP	返回 e 的 numeric_expression 次幂。
FLOOR	返回小于等于数值 numeric_expression 的最大整数。
MOD	返回 x 除以 y 的余数。
NANVL	用于判断输入值参数 n1 是不是 NaN（表示非数字），并返回结果。
POWER	返回 x 的 y 次幂。
REMAINDER	返回 x 除以 y 的余数。
ROUND	返回 numberic 四舍五入后的值。
SIGN	返回数字 n 的符号，大于 0 返回 1，小于 0 返回 -1 ，等于 0 返回 0。
SIN	返回参数角度的正弦值。
SQRT	返回 n 的平方根。
TAN	返回角度的正切值。
TANH	返回数值参数的双曲正切值。
TRUNC	返回 numeric 按精度 precision 截取后的值。

返回字符值的字符函数
部分返回字符值的字符函数示例
regexp_substr

最近更新时间: 2024-06-12 15:06:00

- string：需要进行正则处理的字符串 。
- pattern：进行匹配的正则表达式 。
- position：起始位置，从字符串的第几个字符开始正则表达式匹配（默认为1） 注意：字符串最初的位置是1而不是0。
- occurrence：获取第几个分割出来的组（分割后最初的字符串会按分割的顺序排列成组）。
- modifier：模式（‘i’不区分大小写进行检索；‘c’区分大小写进行检索。默认为‘c’）针对的是正则表达式里字符大小写的匹配。

```
postgres=# SELECT REGEXP_SUBSTR('17,20,23','[^,]+',1,1,'i') AS STR FROM DUAL;
str
-----
17
(1 row)
postgres=#
```

nlssort

最近更新时间: 2024-06-12 15:06:00

指定排序规则。

```
postgres=# create table t_nlssort(f1 integer,f2 varchar2(10));
CREATE TABLE
postgres=#
postgres=# insert into t_nlssort values(1,'腾讯');
INSERT 0 1
postgres=#
postgres=# insert into t_nlssort values(2,'深圳');
INSERT 0 1
postgres=#
postgres=# insert into t_nlssort values(3,'中国');
INSERT 0 1
postgres=# SELECT * FROM t_nlssort ORDER BY NLSSORT(f2,'NLS_SORT = SCHINESE_PINYIN_M');
f1 | f2
----+------
2 | 深圳
1 | 腾讯
3 | 中国
(1 rows)
```

目前tbase只能支持按拼音。

REGEXP_INSTR

最近更新时间: 2024-06-12 15:06:00

寻找字符第一个出现位置。

```
create or replace function regexp_instr(str text,str2 text,str3 INTEGER) returns text as
$$
declare
res text;
begin
res = str3 - 1 + position(substring(substring(str, str3) from str2) in str);
return res;
end;
$$
language plpgsql;
create or replace function regexp_instr(str text,str2 text) returns text as $$
declare
res text;
begin
res = position(substring(str from str2) in str);
return res;
end;
$$
language plpgsql;
```

调用结果

```
#搜索第一次出现的位置
postgres=# select regexp_instr('abc123abc156', '[1-2]+') from dual;
regexp_instr
--------------
4
(1 row)
#搜索第一次出现的位置
postgres=# select regexp_instr('hello Tbase','e');
regexp_instr
--------------
2
(1 row)
#从第7个开始搜索
postgres=# select regexp_instr('abc123abc156', '[1-2]+', 7) from dual;
regexp_instr
--------------
10
(1 row)
```

regexp_replace

最近更新时间: 2024-06-12 15:06:00

regexp_replace(1,2,3,4,5,6)
语法说明： 1：字段
2：替换的字段
3：替换成什么
4：起始位置（默认从1开始）
5：替换的次数（0是无限次）
6：不区分大小写。

```
postgres=# select regexp_replace('tbase_Tbase','s','ee',1,1) from dual;
regexp_replace
----------------
tbaeee_Tbase
(1 row)
postgres=#
```

nchr

最近更新时间: 2024-06-12 15:06:00

给出一个数字代码，返回其对应字符。

```
postgres=# select NCHR(116) from dual;
nchr
------
t
(1 row)
```

nls_upper

最近更新时间: 2024-06-12 15:06:00

将字符转换为大写。

```
postgres=# select NLS_UPPER('tbase','nls_sort= SCHINESE_PINYIN_M') from dual;
nls_upper
-----------
TBASE
(1 row)
```

返回数值的字符函数
部分返回数值的字符函数示例
ASCIISTR

最近更新时间: 2024-06-12 15:06:00

ASCIISTR函数，参数是一个字符串，如果这个字符在ASCII码表中有，则转成ASCII表中的字符。如果没有，则转成\xxxx格式，xxxx是UTF8的编码。

```
create or replace function asciistr(str text) returns text as
$$
declare
mid text;
res text:='';
begin
foreach mid in array regexp_split_to_array(str, '')
loop
if ascii(mid)<256 then
res := res || mid;
else
res := res || convert_to(mid,'UTF8')::text;
end if;
end loop;
return res;
end;
$$
language plpgsql strict;
```

LENGTHB

最近更新时间: 2024-06-12 15:06:00

返回字符长度。

```
postgres=# select LENGTHB('测试') from dual;
lengthb
---------
6
(1 row)
postgres=# select LENGTH('测试') from dual;
length
--------
2
(1 row)
```

length

最近更新时间: 2024-06-12 15:06:00

获取字符长度。

```
postgres=# select length(1);
length
--------
1
(1 row)
postgres=# select length('tbase');
length
--------
5
(1 row)
postgres=# select length('阿弟');
length
--------
2
(1 row)
postgres=# select length(12.12::numeric(10,2));
length
--------
5
(1 row)
```

instr

最近更新时间: 2024-06-12 15:06:00

instr函数返回要截取的字符串在源字符串中的位置。

```
postgres=# select instr('helloworld','l') from dual;
instr
-------
3
(1 row)
postgres=#
```

regexp_count

最近更新时间: 2024-06-12 15:06:00

REGEXP_COUNT 返回pattern 在source_char 串中出现的次数。

```
postgres=# select REGEXP_COUNT('tbase_TBase','se') from DUAL;
regexp_count
--------------
2
(1 row)
```

日期时间函数
部分日期时间函数示例
NUMTODSINTERVAL

最近更新时间: 2024-06-12 15:06:00

numtodsinterval(

```
postgres=# select sysdate,sysdate+numtodsinterval(2,'hour') as res from dual;
orcl_sysdate | res
---------------------+---------------------
2020-08-02 21:55:35 | 2020-08-02 23:55:35
(1 row)
postgres=#
```

DBTIMEZONE

最近更新时间: 2024-06-12 15:06:00

```
postgres=# select DBTIMEZONE from dual;
dbtimezone
------------
08:00:00
(1 row)
postgres=#
```

MONTHS_BETWEEN

最近更新时间: 2024-06-12 15:06:00

MONTHS_BETWEEN(DATE1,DATE2) #函数返回两个日期之间的月份数
postgres=# select months_between(to_date('20210331', 'yyyymmdd'), to_date('20200131', 'yyyymmdd')) as months from dual;
months

14
(1 row)
postgres=#

LAST_DAY

最近更新时间: 2024-06-12 15:06:00

LAST_DAY函数返回指定日期对应月份的最后一天。

```
postgres=# SELECT last_day('2020-05-01') FROM dual;
last_day
------------------------
2020-05-31 00:00:00+08
(1 row)
```

ADD_MONTHS

最近更新时间: 2024-06-12 15:06:00

ADD_MONTHS(x,y) x值为日期，y值为数量, 用于计算某个日期向前或者向后y个月后的时间。

```
postgres=# select add_months(sysdate,1) from dual;
add_months
---------------------
2020-09-04 16:08:11
(1 row)
postgres=# select add_months(sysdate,-1) from dual;
add_months
---------------------
2020-07-04 16:08:15
(1 row)
postgres=#
```

比较函数

最近更新时间: 2024-06-12 15:06:00

函数名	功能描述
REGEXP_LIKE	比较字符串和正则表达式，返回t or f。
GREATEST	返回一个或多个表达式列表中的最大值。
LEAST	返回一个或多个表达式列表中的最小值。

转换函数
部分转换函数示例
to_number

最近更新时间: 2024-06-12 15:06:00

将 expr 转换为数值数据类型的值。 示例： 1、to_number(?,'xxxx')：将指定的16进制数转化成10进制数显示

```
SELECT to_number(substr(dbms_session.unique_session_id,1,4),'xxxx') FROM dual;
```

> **注意：**
>
> 支持字符串类型的16进制数转10进制数，不支持数字类型的16进制数转10进制数。

to_clob

最近更新时间: 2024-06-12 15:06:00

转换字符为clob类型。

```
postgres=# select to_clob('tbase') from dual;
to_clob
---------
tbase
(1 row)
```

ROWIDTOCHAR

最近更新时间: 2024-06-12 15:06:00

转换rowid值为varchar2类型。

```
postgres=# \d+ t_rowid
Table "public.t_rowid"
Column | Type | Collation | Nullable | Default | Storage | Stats target | Description
--------+---------+-----------+----------+---------+---------+--------------+-------------
f1 | integer | | not null | | plain | |
f2 | integer | | | | plain | |
Distribute By: SHARD(f1)
Location Nodes: ALL DATANODES
postgres=# SELECT ROWIDTOCHAR(rowid),rowid from t_rowid;
rowidtochar | rowid
---------------------+----------------------
XPK3fw==AAAAAA==AQA= | XPK3fw==AAAAAA==AQA=
(1 row)
postgres=#
```

CHARTOROWID

最近更新时间: 2024-06-12 15:06:00

CHARTOROWID(c1) 转换varchar2类型为rowid值,c1,字符串，长度为20的字符串，字符串必须符合rowid格式，ROWID。

```
postgres=# select CHARTOROWID('AAAFd1AAFAAAABSACCAA') a1 from dual;
a1
----------------------
AAAFdw==FAAAAA==CCA=
(1 row)
postgres=#
```

XML函数
部分XML函数示例
extractvalue

最近更新时间: 2024-06-12 15:06:00

xml文本解释

```
postgres=# SELECT extractvalue(XMLTYPE('
<AAA>
<BBB>
<BBB>
<BBB>
</AAA>
'),'/AAA/BBB[2]') FROM dual;
extractvalue
--------------
好转
(1 row)
```

extract

最近更新时间: 2024-06-12 15:06:00

extract(xmltype类型,节点)
create table xmlexample(ID varchar(100),name varchar(20),data xmltype);
insert into xmlexample(id,name,data) values('xxxxxxxxxxxxxxx','my document','<?xml version="1.0" encoding="UTF-8" ?>
<collection xmlns="">
<record>
<leader>-----nam0-22-----^^^450-</leader>
<datafield tag="200" ind1="1" ind2=" ">

</datafield>
<datafield tag="209" ind1=" " ind2=" ">

<subfield code="c">10001</subfield>
<subfield code="d">2005-07-09</subfield>
</datafield>
<datafield tag="610" ind1="0" ind2=" ">

</datafield>
</record>
</collection>'::xmltype);

postgres=# select id,name, extract(x.data,'/collection/record/datafield/subfield') as A from xmlexample x;
id | name |
a

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

----
xxxxxxxxxxxxxxx | my document |
d code="b">计算机
de="a">笔记本
(1 row)

编码和解码函数
部分编码和解码函数示例
vsize

最近更新时间: 2024-06-12 15:06:00

返回传入参数的字节数。

```
postgres=# select vsize('TBase合理');
vsize
-------
11
(1 row)
#一个汉字三个字节
postgres=# select vsize(1);
vsize
-------
4
(1 row)
postgres=# select vsize(10000);
vsize
-------
4
(1 row)
#整形是4个节点
```

NULL相关函数
部分NULL相关函数示例
lnnvl

最近更新时间: 2024-06-12 15:06:00

传入表达式为true 返回false；传入为false 返回true。

```
postgres=# create table t_lnnvl(f1 integer, f2 integer);
CREATE TABLE
postgres=#
postgres=# insert into t_lnnvl values(1,1);
INSERT 0 1
postgres=#
postgres=# insert into t_lnnvl values(1,2);
INSERT 0 1
postgres=#
postgres=# insert into t_lnnvl values(1,3);
INSERT 0 1
postgres=#
postgres=# insert into t_lnnvl values(1,4);
INSERT 0 1
postgres=#
postgres=# insert into t_lnnvl values(1,null);
INSERT 0 1
postgres=# select * from t_lnnvl where lnnvl(f2>2);
f1 | f2
----+----
1 | 1
1 | 2
1 |
(1 rows)
```

nvl2

最近更新时间: 2024-06-12 15:06:00

NVL2(E1, E2, E3) 如果E1为NULL，则函数返回E3，若E1不为null，则返回E2。

```
postgres=# select NVL2('tbase', 'tbase1'::text, 'tbase2'::text) from dual;
nvl2
--------
tbase1
(1 row)
postgres=#
postgres=# select NVL2(NULL, 'tbase1'::text, 'tbase2'::text) from dual;
nvl2
--------
tbase2
(1 row)
```

环境和标识符函数
说明以及示例

最近更新时间: 2024-06-12 15:06:00

函数XS_SYS_CONTEXT

参数名	返回值
CREATED_BY	创建当前应用程序会话的所有者。
CREATE_TIME	当前应用程序会话的创建时间。
COOKIE	作为参数传递的安全会话 cookie，可用于在以后的调用中识别新创建的 Real Application Security 应用程序会话，直到更改 cookie 值或会话被销毁。
CURRENT_XS_USER	权限当前处于活动状态的 Real Application Security 会话应用程序用户的名称。
CURRENT_XS_USER_GUID	权限当前处于活动状态的 Real Application Security 会话应用程序用户的标识符。
INACTIVITY_TIMEOUT	当前应用程序会话的指定非活动超时值（以分钟为单位）。
LAST_ACCESS_TIME	会话应用程序用户最后一次访问会话的时间。
LAST_AUTHENTICATION_TIME	上次对会话应用程序用户进行身份验证的时间。
LAST_UPDATED_BY	上次更新应用程序会话的时间。
PROXY_GUID	代表 SESSION_XS_USER 打开当前会话的 Real Application Security 会话应用程序用户的标识符。
SESSION_ID	应用程序会话的会话标识符。
SESSION_SIZE	应用程序会话的大小（以字节为单位）。
SESSION_XS_USER	登录时 Real Application Security 会话应用程序用户的名称。
SESSION_XS_USER_GUID	登录时 Real Application Security 会话应用程序用户的标识符。
USERNAME	会话应用程序用户名。
USER_ID	会话应用程序用户的标识符。

示例：

```
SELECT DECODE(USER, 'XS$NULL', XS_SYS_CONTEXT('XS$SESSION','USERNAME'), USER) FROM DUAL ;
```

函数USERENV

参数名	返回值
SCHEMAID	返回当前默认的 Schema ID。
LANG	返回 ISO 缩写的语言名称，比 LANGUAGE 参数更短的格式。
LANGUAGE	返回当前会话的语言、地域和字符集。
SID	返回会话 ID。

> **注意：**
>
> language参数返回值，Oracle为全称组合，TDSQL PG是缩写组合

示例：

```
select 'USER_LANGUAGE' as Lang,userenv('language') from dual;
```

函数UID

示例

```
SELECT USER, UID FROM DUAL;
```

聚合函数

最近更新时间: 2024-06-12 15:06:00

函数名称	功能描述
APPROX_COUNT_DISTINCT	返回包含expr的不同值的近似行数为COUNT (DISTINCT expr)函数的替代方法
AVG	返回expr平均值
COUNT	返回查询返回的行数，可以用作聚合或分析函数
KEEP	以某种规则排序后，取同一个分组下的第一个或最后一个的值
LISTAGG	根据查询条件中的一个或多个表达式将查询结果集划分为组，可用于列转行
MAX	返回expr最大值。可用作聚合或分析函数
MIN	返回expr最小值。可用作聚合或分析函数
ROLLUP	用于分组统计，返回小组合计以及总计
STDDEV	返回expr的样本标准差，即一组数字。可用作聚合和分析函数
SUM	返回expr值的总和。可用作聚合或分析函数。
VARIANCE	返回expr的方差。可用作聚合或分析函数。
WMSYS.WM_CONCAT/WM_CONCAT	可以把列值以'，'分割开来，并显示成一行，用于列转行

分析函数

最近更新时间: 2024-06-12 15:06:00

函数名称	功能描述
AVG	返回expr平均值
COUNT	返回查询返回的行数，可以用作聚合或分析函数
KEEP	以某种规则排序后，取同一个分组下的第一个或最后一个的值
SUM	返回expr值的总和。可用作聚合或分析函数。
MAX	返回expr最大值。可用作聚合或分析函数
MIN	返回expr最小值。可用作聚合或分析函数
LISTAGG	根据查询条件中的一个或多个表达式将查询结果集划分为组，可用于列转行
STDDEV	返回expr的样本标准差，即一组数字。可用作聚合和分析函数
VARIANCE	返回expr的方差。可用作聚合或分析函数。
RANK	计算一组值中的值的等级。返回类型为NUMBER
LEAD	在查询中取出同一字段的后 offset 行的数据作为独立的列存在表中。可以代替表的自联接
LAG	查询中取出同一字段的前 offset 行的数据作为独立的列存在表中。可以代替表的自联接
FIRST_VALUE	返回有序值集中的第一个值。如果零，则返回NULL ，除非指定IGNORE NULLS
LAST_VALUE	返回有序值集中的最后一个值
NTH_VALUE	返回analytic_clause定义的窗口中第n行的measure_expr值
CUME_DIST	计算一组值中值的累积分布。返回的值范围> 0到
DENSE_RANK	计算有序行组中行的排序位置
NTILE	将有序数据集划分为若干个组，并为每一行分配适当的组号。组编号为 1 到 expr。
PERCENT_RANK	类似于CUME_DIST （累积分布）函数。 PERCENT_RANK返回的值范围是0到1（包括0和1）
RATIO_TO_REPORT	计算值与一组值之和的比率
ROW_NUMBER	为每一行（分区中的每一行或查询返回的每一行）分配一个唯一的编号
WMSYS.WM_CONCAT/WM_CONCAT	可以把列值以'，'分割开来，并显示成一行，用于列转行

二进制操作函数
empty_clob

最近更新时间: 2024-06-12 15:06:00

初始化CLOB字段。

```
postgres=# select empty_clob();
empty_clob
------------
(1 row)
postgres=# create table t1 (f1 int,f2 clob);
CREATE TABLE
postgres=# insert into t1(f1,f2) values (1,empty_clob());
INSERT 0 1
postgres=#
```

统计函数
listagg

最近更新时间: 2024-06-12 15:06:00

```
listagg (filedname,',') WITHIN GROUP (ORDER BY filedname)
```

行转列函数

```
postgres=# create table person
(
deptno varchar2(10),
ename varchar(20)
);
CREATE TABLE
postgres=# insert into person values('20','aaa');
INSERT 0 1
postgres=#
postgres=# insert into person values('20','bbb');
INSERT 0 1
postgres=#
postgres=# insert into person values('20','ccc');
INSERT 0 1
postgres=#
postgres=# insert into person values('21','ddd');
INSERT 0 1
postgres=#
postgres=# insert into person values('21','eee');
INSERT 0 1
postgres=# select
postgres-# deptno,
postgres-# listagg (ename,',') WITHIN GROUP (ORDER BY ENAME)
postgres-# from
postgres-# person
postgres-# group by
postgres-# deptno ;
deptno | listagg
--------+-------------
20 | aaa,bbb,ccc
21 | ddd,eee
(2 rows)
```

其他函数的使用

sys_guid()

最近更新时间: 2024-06-12 15:06:00

sys_guid 生成并返回一个由 16 个字节组成的全局唯一标识符（RAW 值）。 生成的标识符由主机标识符、调用函数的进程或线程的进程或线程标识符以及该进程或线程的非重复值（字节序列）组成。

```
postgres=# select sys_guid();


sys_guid

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

B975096BCDCCBA209F74A69DDA83DE7B

(1 行记录)
```

数据表准备

最近更新时间: 2024-06-12 15:06:00

```
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));
```

row_number() --返回行号，不分组

最近更新时间: 2024-06-12 15:06:00

```
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)
```

row_number() --返回行号，按amount排序

最近更新时间: 2024-06-12 15:06:00

```
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)
```

row_number() --返回行号，按begincity分组，pubtime排序

最近更新时间: 2024-06-12 15:06:00

```
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)
```

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...

最近更新时间: 2024-06-12 15:06:00

```
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)
```

dense_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...

最近更新时间: 2024-06-12 15:06:00

```
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)
```

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))

最近更新时间: 2024-06-12 15:06:00

```
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)
```

cume_dist() --返回行数除以记录数值

最近更新时间: 2024-06-12 15:06:00

```
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)
```

ntile(分组数量)--让所有记录尽可以的均匀分布

最近更新时间: 2024-06-12 15:06:00

```
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)
```

lag(value any [, offset integer [, default any]])--返回偏移量值

最近更新时间: 2024-06-12 15:06:00

```
#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)
```

lead(value any [,offset integer [, default any]])--返回偏移量值

最近更新时间: 2024-06-12 15:06:00

```
#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)
```

first_value(value any)返回第一值

最近更新时间: 2024-06-12 15:06:00

```
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)
```

last_value(value any)返回最后值

最近更新时间: 2024-06-12 15:06:00

```
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 un
bounded 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)
```

nth_value(value any, nth integer)：返回窗口框架中的指定值

最近更新时间: 2024-06-12 15:06:00

```
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)
```

统计各个城市的总运费及平均每单的运费

最近更新时间: 2024-06-12 15:06:00

```
postgres=# select sum(amount) over(partition by begincity),avg(amount) over(partition by begincity),begincity,amount fr
om 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)
```

窗口函数别名使用

最近更新时间: 2024-06-12 15:06:00

```
postgres=# select sum(amount) over w,avg(amount) over w,begincity,amount from bills window w as (partition by beginci
ty);
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)
```

获取每个城市运费前两名订单

最近更新时间: 2024-06-12 15:06:00

```
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)
```

连续百分率：返回一个对应于排序中指定分数的值

最近更新时间: 2024-06-12 15:06:00

```
postgres=# select * from bills where begincity='三明市';
id | goodsdesc | beginunit | begincity | pubtime | amount
----+-----------+-----------+-----------+---------------------+---------
12 | 建筑设备 | 福建省 | 三明市 | 2015-10-05 07:21:22 | 2022.31
13 | 设备 | 福建省 | 三明市 | 2015-10-05 11:21:54 | 8771.11
14 | 普货 | 福建省 | 三明市 | 2015-10-05 15:19:17 | 1316.27
(3 rows)
postgres=# select
min(amount),
PERCENTILE_CONT(0) WITHIN GROUP(order by amount) AS RATE_0 ,
PERCENTILE_CONT(0.25) WITHIN GROUP(order by amount) AS RATE_25,
PERCENTILE_CONT(0.5) WITHIN GROUP(order by amount) AS RATE_50,
PERCENTILE_CONT(0.75) WITHIN GROUP(order by amount) AS RATE_75,
PERCENTILE_CONT(1) WITHIN GROUP(order by amount) AS RATE_100
from bills
where
begincity='三明市'
;
min | rate_0 | rate_25 | rate_50 | rate_75 | rate_100
---------+---------+---------+---------+---------+----------
1316.27 | 1316.27 | 1669.29 | 2022.31 | 5396.71 | 8771.11
(1 row)
postgres=#
```

伪列
ROWNUM

最近更新时间: 2024-06-12 15:06:00

ROWNUM是Oracle中对结果集加的一个伪列，即先查到结果集之后再加上去的一个列 (强调：先要有结果集)。简单的说 ROWNUM是对符合条件结果的序列号。它总是从1开始排起的。 TDSQL PG支持ROWNUM伪列用法。

使用限制 使用ROWNUM时需要注意事项：

- ROWNUM是数据库从数据文件或缓冲区中读取数据的顺序，所以rownum>n（n>=1）、rownum between n(n>1) and m是得不到数据的。
- 不能使用ROWNUM做为字段名称。

语法 `SELECT [ROWNUM,]column1,... FROM table1...WHERE ROWNUM [< |<= ] n [ and ...];`

示例

```
postgres=# create table t1(f1 int,f2 varchar2(16));
CREATE TABLE
postgres=# insert into t1 values (1,'ONE');
INSERT 0 1
postgres=# insert into t1 values (2,'TWO');
INSERT 0 1
postgres=# insert into t1 values (3,'THREE');
INSERT 0 1
postgres=# insert into t1 values (4,'FOUR');
INSERT 0 1
postgres=# insert into t1 values (5,'FIVE');
INSERT 0 1
postgres=# insert into t1 values (6,'SIX');
INSERT 0 1
postgres=# insert into t1 values (7,'SEVEN');
INSERT 0 1
postgres=# select * from t1 order by f1;
f1 | f2
----+-------
1 | ONE
2 | TWO
3 | THREE
4 | FOUR
5 | FIVE
6 | SIX
7 | SEVEN
(7 rows)

postgres=# select rownum,f1,f2 from t1 where f1>5;
rownum | f1 | f2
--------+----+-------
1 | 6 | SIX
2 | 7 | SEVEN
(2 rows)
```

ROWNUM从1开始。

```
postgres=# select rownum,f1,f2 from t1 where rownum<2;
rownum | f1 | f2
--------+----+-----
1 | 1 | ONE
(1 row)
```

只显示一行数据。

```
postgres=# select rownum,f1,f2 from t1 where rownum>=1;
rownum | f1 | f2
--------+----+-------
1 | 1 | ONE
2 | 2 | TWO
3 | 5 | FIVE
4 | 6 | SIX
5 | 3 | THREE
6 | 4 | FOUR
7 | 7 | SEVEN
(7 rows)
```

显示全部数据。

```
postgres=# select rownum,f1,f2 from t1 where rownum between 1 and 2;
rownum | f1 | f2
--------+----+-----
1 | 1 | ONE
2 | 2 | TWO
(2 rows)

postgres=# select rownum,f1,f2 from t1 where rownum between 2 and 3;
rownum | f1 | f2
--------+----+----
(0 rows)
```

rownumbetween 如果大于1，查询结果为空，因为ROWNUM总是从1开始。

ROWID

最近更新时间: 2024-06-12 15:06:00

Oracle数据库的表中的每一行数据都有一个唯一的标识符，或者称为ROWID，在Oracle内部通常就是使用它来访问数据的。ROWID需要10个字节的存储空间，并用18个字符来显示。该值表明了该行在Oracle数据库中的物理具体位置。可以在一个查询中使用ROWID来表明查询结果中包含该值。 保存ROWID需要10个字节或者是80个位二进制位。这80个二进制位分别是：

1. 数据对象编号，表明此行所属的数据库对象的编号，每个数据对象在数据库建立的时候都被唯一分配一个编号，并且此编号唯一。数据对象编号占用大约32位。

2. 对应文件编号，表明该行所在文件的编号，表空间的每一个文件标号都是唯一的。文件编号所占用的位置是10位。

3. 块编号，表明改行所在文件的块的位置块编号需要22位。

4. 行编号，表明该行在行目录中的具体位置行编号需要16位。

   这样加起来就是80位。

Oracle的物理扩展ROWID有18位，每位采用64位编码，分别用A~~Z、a~~z、0~9、+、/共64个字符表示。A表示0，B表示1，……Z表示25，a表示26，……z表示51，0表示52，……，9表示61，+表示62，/表示63。

TDSQL PG 兼容Oracle的ROWID，由

使用限制 关于使用ROWID的一些注意事项：

- 参数default_with_rowid可以设置默认建表时是否带ROWID，默认该值为off；可以根据需要修改该参数。
- default_with_rowid=OFF时，也可以通过create table 时加上with (rowids)来让表带ROWID列。
- ROWID列默认没有索引，所以基于该列的sql可能存在性能问题，可能通过创建索引来提高sql效率，建索引语法与普通字段建索引语法相同。
- 不支持Oracle的DBMS_ROWID包调用。
- cluster table、vacuum full table重写表数据后，ROWID会变，与Oracle一致。

语法 default_with_rowid=off，建表时带ROWID语法，示例：

```
CREATE TABLE[ IF NOT EXISTS ] table_name
( [
{ column_name data_type [ COLLATE collation] [ column_constraint [ ... ] ]
| table_constraint
| LIKE source_table [ like_option ... ] }
[, ... ]
] )
[ INHERITS ( parent_table [, ... ] ) ]
[ PARTITION BY { RANGE | LIST | HASH } ( {column_name | ( expression ) } [ COLLATE collation ] [ opclass ] [, ... ] ) ]
```

```
[ WITH ( storage_parameter [= value] [, ... ]) | WITH OIDS | WITHOUT OIDS ]
WITH ( ROWIDS ) ;
```

示例：

```
postgres=#create table t_rowid(f1 int,f2 varchar2(32)) with (rowids);
CREATE TABLE
postgres=#insert into t_rowid(f1,f2) values(1,'ONE');
INSERT 0 1
postgres=#insert into t_rowid(f1,f2) values(2,'TWO');
INSERT 0 1
postgres=#select rowid,* from t_rowid;
rowid | f1 | f2
---------------------+----+-----
XPK3fw==AQAAAAAAAAA= | 1 | ONE
XPK3fw==AgAAAAAAAAA= | 2 | TWO

postgres=#select rowid,* from t_rowid where rowid='XPK3fw==AgAAAAAAAAA=';
rowid | f1 | f2
---------------------+----+-----
XPK3fw==AgAAAAAAAAA= | 2 | TWO
(1 row)

postgres=#update t_rowid set f2='TWO1' where f1=2;
UPDATE 1
postgres=#select rowid,* from t_rowid where rowid='XPK3fw==AgAAAAAAAAA=';
rowid | f1 | f2
---------------------+----+------
XPK3fw==AgAAAAAAAAA= | 2 | TWO1
(1 row)

#数据更新，ROWID不变。

postgres=#create index idx_t_rowid_rowid on t_rowid(rowid);
CREATE INDEX
postgres=#explain select rowid,* from t_rowid where rowid='XPK3fw==AgAAAAAAAAA=';
QUERY PLAN
-------------------------------------------------------------------------------------------
Remote Fast Query Execution (cost=0.00..0.00 rows=0 width=0)
Node/s: dn001, dn002
-> Index Scan using idx_t_rowid_rowid on t_rowid (cost=0.13..4.15 rows=1 width=98)
Index Cond: (rowid ='XPK3fw==AgAAAAAAAAA='::rowid)
(4 rows)
```

支持ROWID字段上创建索引。

hint

加载插件

最近更新时间: 2024-06-12 15:06:00

```
postgres=# create extension pg_hint_plan ;
CREATE EXTENSION
postgres=# alter role all set session_preload_libraries='pg_hint_plan';
ALTER ROLE
postgres=# \c
```

部分hint用法示例
full全表扫描

最近更新时间: 2024-06-12 15:06:00

示例：/*+full(table_name) */强制选择全表扫描 创建表并插入数据，对比使用hint全表扫描前后执行计划差异。

```
postgres=# create table hint_t1(f1 integer,f2 integer) ;
CREATE TABLE
postgres=# create index hint_t1_f1_idx on hint_t1(f1);
CREATE INDEX
postgres=# insert into hint_t1 select t as f1,t as f2 from generate_series(1,10000) as t;
INSERT 0 10000
postgres=# explain select /*+SeqScan(hint_t1) */ * from hint_t1 where f1=1;
QUERY PLAN
------------------------------------------------------------------------
Remote Subquery Scan on all (dn001) (cost=0.00..205.00 rows=1 width=8)
-> Seq Scan on hint_t1 (cost=0.00..205.00 rows=1 width=8)
Filter: (f1 = 1)
(4 rows)
postgres=# explain select * from hint_t1 where f1=1;
QUERY PLAN
-------------------------------------------------------------------------------------
Remote Subquery Scan on all (dn001) (cost=0.16..4.18 rows=1 width=8)
-> Index Scan using hint_t1_f1_idx on hint_t1 (cost=0.16..4.18 rows=1 width=8)
Index Cond: (f1 = 1)
(3 rows)
postgres=# explain select /*+full(hint_t1) */ * from hint_t1 where f1=1;
QUERY PLAN
------------------------------------------------------------------------
Remote Subquery Scan on all (dn001) (cost=0.00..205.00 rows=1 width=8)
-> Seq Scan on hint_t1 (cost=0.00..205.00 rows=1 width=8)
Filter: (f1 = 1)
(3 rows)
```

index全表扫描

最近更新时间: 2024-06-12 15:06:00

示例：*/+index(table_name)*/对于指定表的查询强制选择索引

```
postgres=# create table hint_t2(f1 integer,f2 integer) ;
CREATE TABLE
postgres=# create index hint_t2_f1_idx on hint_t2(f1);
CREATE INDEX
postgres=# insert into hint_t2 select t as f1,t as f2 from generate_series(1,1000) as t;
INSERT 0 1000
postgres=# vacuum ANALYZE hint_t2;
VACUUM
postgres=#
postgres=# explain select /*+index(hint_t2) */ * from hint_t2 where f1>1;
QUERY PLAN
------------------------------------------------------------------------------------------
Remote Subquery Scan on all (dn001) (cost=0.15..28.65 rows=1000 width=8)
-> Index Scan using hint_t2_f1_idx on hint_t2 (cost=0.15..28.65 rows=1000 width=8)
Index Cond: (f1 > 1)
(3 rows)
postgres=# explain select /*+index(hint_t2 hint_t2_f1_idx1) */ * from hint_t2 where f1>1;
QUERY PLAN
-------------------------------------------------------------------------------------------------
Remote Subquery Scan on all (dn001) (cost=10000000000.00..10000000020.50 rows=1000 width=8)
-> Seq Scan on hint_t2 (cost=10000000000.00..10000000020.50 rows=1000 width=8)
Filter: (f1 > 1)
(3 rows)
```

no_index全表扫描

最近更新时间: 2024-06-12 15:06:00

示例：*/+no_index(table_name)/*强制SQL不走特定索引的方法

```
postgres=# create table hint_t7(f1 integer,f2 integer) ;
CREATE TABLE
postgres=# insert into hint_t7 select t as f1,t as f2 from generate_series(1,10000) as t;
INSERT 0 10000
postgres=# create index hint_t7_f1_idx on hint_t7(f1);
CREATE INDEX
postgres=# vacuum ANALYZE hint_t7;
VACUUM
postgres=# explain select /*+no_index(hint_t7) */ * from hint_t7 where f1=1 ;
QUERY PLAN
------------------------------------------------------------------------
Remote Subquery Scan on all (dn001) (cost=0.00..200.00 rows=1 width=8)
-> Seq Scan on hint_t7 (cost=0.00..200.00 rows=1 width=8)
Filter: (f1 = 1)
(3 rows)
postgres=# explain select * from hint_t7 where f1=1 ;
QUERY PLAN
-------------------------------------------------------------------------------------
Remote Subquery Scan on all (dn001) (cost=0.16..4.18 rows=1 width=8)
-> Index Scan using hint_t7_f1_idx on hint_t7 (cost=0.16..4.18 rows=1 width=8)
Index Cond: (f1 = 1)
(3 rows)
postgres=#
```

别名使用

最近更新时间: 2024-06-12 15:06:00

```
postgres=# create table hint_t6(f1 integer,f2 integer) ;
CREATE TABLE
postgres=# insert into hint_t6 select t as f1,t as f2 from generate_series(1,10000) as t;
INSERT 0 10000
postgres=# create index hint_t6_f1_idx on hint_t6(f1);
CREATE INDEX
postgres=# vacuum ANALYZE hint_t6;
VACUUM
postgres=# select /*+full(hint_t6) */ * from hint_t6 a where f1=1;
f1 | f2
----+----
1 | 1
(1 row)
postgres=# explain select /*+full(hint_t6) */ * from hint_t6 a where f1=1;;
QUERY PLAN
-------------------------------------------------------------------------------------
Remote Subquery Scan on all (dn001) (cost=0.16..4.18 rows=1 width=8)
-> Index Scan using hint_t6_f1_idx on hint_t6 a (cost=0.16..4.18 rows=1 width=8)
Index Cond: (f1 = 1)
(3 rows)
postgres=# explain select /*+full(a) */ * from hint_t6 a where f1=1;
QUERY PLAN
-----------------------------------------------------------------------
Remote Subquery Scan on all (dn001) (cost=0.00..200.00 rows=1 width=8)
-> Seq Scan on hint_t6 a (cost=0.00..200.00 rows=1 width=8)
Filter: (f1 = 1)
(3 rows)
```

强制走nest loop

最近更新时间: 2024-06-12 15:06:00

示例：/*+use_nl(table1,table2) */强制选择嵌套循环连接

```
postgres=# create table hint_t6(f1 integer,f2 integer) ;
CREATE TABLE
postgres=# insert into hint_t6 select t as f1,t as f2 from generate_series(1,10000) as t;
INSERT 0 10000
postgres=# vacuum ANALYZE hint_t6;
VACUUM
postgres=#
postgres=# create table hint_t7(f1 integer,f2 integer) ;
CREATE TABLE
postgres=# insert into hint_t7 select t as f1,t as f2 from generate_series(1,10000) as t;
INSERT 0 10000
postgres=# vacuum ANALYZE hint_t7;
VACUUM
postgres=#
postgres=# explain select * from hint_t6 t,hint_t7 t1 where t.f1=t1.f1 and t1.f1>9999 ;
QUERY PLAN
------------------------------------------------------------------------------
Remote Subquery Scan on all (dn001) (cost=200.01..412.52 rows=1 width=16)
-> Hash Join (cost=200.01..412.52 rows=1 width=16)
Hash Cond: (t.f1 = t1.f1)
-> Seq Scan on hint_t6 t (cost=0.00..175.00 rows=10000 width=8)
-> Hash (cost=200.00..200.00 rows=1 width=8)
-> Seq Scan on hint_t7 t1 (cost=0.00..200.00 rows=1 width=8)
Filter: (f1 > 9999)
(7 rows)
postgres=# explain select /*+use_nl(t,t1) */ * from hint_t6 t,hint_t7 t1 where t.f1=t1.f1 and t.f1>999 ;
QUERY PLAN
------------------------------------------------------------------------------
Remote Subquery Scan on all (dn001) (cost=0.00..1350547.50 rows=9001 width=16)
-> Nested Loop (cost=0.00..1350547.50 rows=9001 width=16)
Join Filter: (t.f1 = t1.f1)
-> Seq Scan on hint_t7 t1 (cost=0.00..175.00 rows=10000 width=8)
-> Materialize (cost=0.00..245.00 rows=9001 width=8)
-> Seq Scan on hint_t6 t (cost=0.00..200.00 rows=9001 width=8)
Filter: (f1 > 999)
(7 rows)
postgres=#
```

强制走merge join

最近更新时间: 2024-06-12 15:06:00

示例：/*+use_merge(table1,table2) */强制选择合并连接

```
postgres=# create table hint_t6(f1 integer,f2 integer) ;
CREATE TABLE
postgres=# insert into hint_t6 select t as f1,t as f2 from generate_series(1,10000) as t;
INSERT 0 10000
postgres=# create index hint_t6_f1_idx on hint_t6(f1);
CREATE INDEX
postgres=# vacuum ANALYZE hint_t6;
VACUUM
postgres=# create table hint_t7(f1 integer,f2 integer) ;
CREATE TABLE
postgres=# insert into hint_t7 select t as f1,t as f2 from generate_series(1,10000) as t;
INSERT 0 10000
postgres=# create index hint_t7_f1_idx on hint_t7(f1);
CREATE INDEX
postgres=# vacuum ANALYZE hint_t7;
VACUUM
postgres=#
postgres=# explain select * from hint_t6 t,hint_t7 t1 where t.f1=t1.f1 and t.f1>9999 ;
QUERY PLAN
------------------------------------------------------------------------------------------
Remote Subquery Scan on all (dn001) (cost=0.32..8.37 rows=1 width=16)
-> Nested Loop (cost=0.32..8.37 rows=1 width=16)
-> Index Scan using hint_t6_f1_idx on hint_t6 t (cost=0.16..4.18 rows=1 width=8)
Index Cond: (f1 > 9999)
-> Index Scan using hint_t7_f1_idx on hint_t7 t1 (cost=0.16..4.18 rows=1 width=8)
Index Cond: (f1 = t.f1)
(6 rows)
postgres=# explain select /*+use_merge(t,t1) */ * from hint_t6 t,hint_t7 t1 where t.f1=t1.f1 and t.f1>9999 ;
QUERY PLAN
---------------------------------------------------------------------------------------------
Remote Subquery Scan on all (dn001) (cost=0.32..257.35 rows=1 width=16)
-> Merge Join (cost=0.32..257.35 rows=1 width=16)
Merge Cond: (t.f1 = t1.f1)
-> Index Scan using hint_t6_f1_idx on hint_t6 t (cost=0.16..4.18 rows=1 width=8)
Index Cond: (f1 > 9999)
-> Index Scan using hint_t7_f1_idx on hint_t7 t1 (cost=0.16..228.16 rows=10000 width=8)
(6 rows)
postgres=#
```

强制走hash join

最近更新时间: 2024-06-12 15:06:00

示例：/*+use_hash(table1,table2) */强制选择哈希连接

```
postgres=# create table hint_t6(f1 integer,f2 integer) ;
CREATE TABLE
postgres=# insert into hint_t6 select t as f1,t as f2 from generate_series(1,10000) as t;
INSERT 0 10000
postgres=# create index hint_t6_f1_idx on hint_t6(f1);
CREATE INDEX
postgres=# vacuum ANALYZE hint_t6;
VACUUM
postgres=# create table hint_t7(f1 integer,f2 integer) ;
CREATE TABLE
postgres=# insert into hint_t7 select t as f1,t as f2 from generate_series(1,10000) as t;
INSERT 0 10000
postgres=# create index hint_t7_f1_idx on hint_t7(f1);
CREATE INDEX
postgres=# vacuum ANALYZE hint_t7;
VACUUM
postgres=#
postgres=# explain select * from hint_t6 t,hint_t7 t1 where t.f1=t1.f1 and t.f1>9999 ;
QUERY PLAN
------------------------------------------------------------------------------------------
Remote Subquery Scan on all (dn001) (cost=0.32..8.37 rows=1 width=16)
-> Nested Loop (cost=0.32..8.37 rows=1 width=16)
-> Index Scan using hint_t6_f1_idx on hint_t6 t (cost=0.16..4.18 rows=1 width=8)
Index Cond: (f1 > 9999)
-> Index Scan using hint_t7_f1_idx on hint_t7 t1 (cost=0.16..4.18 rows=1 width=8)
Index Cond: (f1 = t.f1)
(6 rows)
postgres=# explain select /*+use_hash(t,t1) */ * from hint_t6 t,hint_t7 t1 where t.f1=t1.f1 and t.f1>9999 ;
QUERY PLAN
-------------------------------------------------------------------------------------------------
Remote Subquery Scan on all (dn001) (cost=4.19..216.70 rows=1 width=16)
-> Hash Join (cost=4.19..216.70 rows=1 width=16)
Hash Cond: (t1.f1 = t.f1)
-> Seq Scan on hint_t7 t1 (cost=0.00..175.00 rows=10000 width=8)
-> Hash (cost=4.18..4.18 rows=1 width=8)
-> Index Scan using hint_t6_f1_idx on hint_t6 t (cost=0.16..4.18 rows=1 width=8)
Index Cond: (f1 > 9999)
(7 rows)
```

并行执行

最近更新时间: 2024-06-12 15:06:00

示例：parallel(n)指定SQL的并发数

```
postgres=# create table hint_t3(f1 integer,f2 integer) ;
CREATE TABLE
postgres=# insert into hint_t3 select t as f1,t as f2 from generate_series(1,1000000) as t;
INSERT 0 1000000
postgres=# vacuum ANALYZE hint_t3;
VACUUM
postgres=#
postgres=# explain select count(1) from hint_t3;
QUERY PLAN
------------------------------------------------------------------------------------
Finalize Aggregate (cost=20020.01..20020.02 rows=1 width=8)
-> Remote Subquery Scan on all (dn001) (cost=20020.00..20020.01 rows=1 width=0)
-> Partial Aggregate (cost=19920.00..19920.01 rows=1 width=8)
-> Seq Scan on hint_t3 (cost=0.00..17420.00 rows=1000000 width=0)
(4 rows)

postgres=# explain select /*+parallel(hint_t3 2) */ count(1) from hint_t3;
QUERY PLAN
-----------------------------------------------------------------------------------------------
Parallel Finalize Aggregate (cost=13728.35..13728.36 rows=1 width=8)
-> Parallel Remote Subquery Scan on all (dn001) (cost=13728.33..13728.35 rows=1 width=0)
-> Gather (cost=13628.33..13628.34 rows=1 width=8)
Workers Planned: 2
-> Partial Aggregate (cost=12628.33..12628.34 rows=1 width=8)
-> Parallel Seq Scan on hint_t3 (cost=0.00..11586.67 rows=416667 width=0)
(6 rows)
postgres=#
```

GOTO

最近更新时间: 2024-06-12 15:06:00

语法： `GOTO lable`

示例：

```
create tablegoto_test(id int,name varchar);

create orreplace function goto_func(v_maxnum integer) returns void as
declare
maxnuminteger;
begin
maxnum :=v_maxnum;
for i in1..maxnum loop
if i=3 then
goto label;
end if;
insert intogoto_test values (i,'hasai');
end loop;
<<label>>
updategoto_test set name = 'user1' where id = 2;
end;
/

select goto_func(5);


select *from goto_test;
```

关于插件
插件查看，添加和删除
查看数据库加载了那些插件

最近更新时间: 2024-06-12 15:06:00

```
postgres=# SELECT e.extname AS "Name", e.extversion AS "Version", n.nspname AS "Schema", c.description AS "Descripti
on" FROM pg_catalog.pg_extension e LEFT JOIN pg_catalog.pg_namespace n ON n.oid = e.extnamespace LEFT JOIN pg_c
atalog.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)
```

添加插件

最近更新时间: 2024-06-12 15:06:00

```
postgres=#create extension "uuid-ossp" with schema tbase;
CREATEEXTENSION
```

上面的语句把"uuid-ossp"创建到模式tbase下。

删除插件

最近更新时间: 2024-06-12 15:06:00

```
postgres=# drop extension "uuid-ossp" ;
DROP EXTENSION
```

插件uuid-ossp使用
uuid-ossp功能介绍及添加方法

最近更新时间: 2024-06-12 15:06:00

功能介绍

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。

操作步骤

1. 给数据库添加uuid-ossp插件。

```
postgres=# create extension "uuid-ossp" with schema tbase;
CREATE EXTENSION
```

uuid常量函数
uuid_nil()

最近更新时间: 2024-06-12 15:06:00

这是一个"nil" UUID 常量。

```
postgres=# select uuid_nil();
uuid_nil
--------------------------------------
00000000-0000-0000-0000-000000000000
(1 row)
```

uuid_ns_dns()

最近更新时间: 2024-06-12 15:06:00

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)
```

uuid_ns_url()

最近更新时间: 2024-06-12 15:06:00

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)
```

uuid_ns_oid()

最近更新时间: 2024-06-12 15:06:00

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)
```

uuid_ns_x500()

最近更新时间: 2024-06-12 15:06:00

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)
```

uuid生成函数
uuid_generate_v1()

最近更新时间: 2024-06-12 15:06:00

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)
```

uuid_generate_v1mc()

最近更新时间: 2024-06-12 15:06:00

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)
```

uuid_generate_v3(namespace uuid, name text)

最近更新时间: 2024-06-12 15:06:00

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://imgcache.finance.cloud.tencent.com:80tbase.qq.com');
uuid_generate_v3
--------------------------------------
af6371f5-bcd5-300b-8a35-7186d54009d5
(1 row)

postgres=# SELECT uuid_generate_v3(uuid_ns_dns(), 'http://imgcache.finance.cloud.tencent.com:80tbase.qq.com');
uuid_generate_v3
--------------------------------------
246a9a9c-cff2-3b34-9ddd-80cc6e30a222
(1 row)
```

uuid_generate_v4()

最近更新时间: 2024-06-12 15:06:00

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)
```

uuid_generate_v5(namespace uuid, name text)

最近更新时间: 2024-06-12 15:06:00

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://imgcache.finance.cloud.tencent.com:80tbase.qq.com');
uuid_generate_v5
--------------------------------------
d8381c8e-6899-5e43-ab26-18d3660d7db1
(1 row)

postgres=# SELECT uuid_generate_v5(uuid_ns_dns(), 'http://imgcache.finance.cloud.tencent.com:80tbase.qq.com');
uuid_generate_v5
--------------------------------------
d5051eb0-2051-53be-8d36-967851c3946d
(1 row)
```

在数据表中使用uuid默认值

最近更新时间: 2024-06-12 15:06:00

```
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)
```

uuid各函数性能对比

最近更新时间: 2024-06-12 15:06:00

- 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://imgcache.finance.cloud.tencent.com:80tbase.qq.com'),uuid_generate_v3(uuid_ns_dns(), 'http://imgcache.finance.cloud.tencent.com:80tbase.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://imgcache.finance.cloud.tencent.com:80tbase.qq.com'),uuid_generate_v5(uuid_ns_dns(), 'http://imgcache.finance.cloud.tencent.com:80tbase.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。

uuid与serial 在TDSQL PG中性能对比

最近更新时间: 2024-06-12 15:06:00

- 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) ;
```

- erial pgbench结果。

```
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%

占用空间对比

最近更新时间: 2024-06-12 15:06:00

- 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

使用uuid做为分布列的方案

最近更新时间: 2024-06-12 15:06:00

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=#
```

使用建议

最近更新时间: 2024-06-12 15:06:00

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。

插件pg_stat_statements使用
pg_stat_statements功能介绍及添加方法

最近更新时间: 2024-06-12 15:06:00

功能介绍

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。

操作步骤

1. 给数据库添加pg_stat_statements插件。

```
postgres=#create extension pg_stat_statements with schema tbase;
CREATE EXTENSION
```

获取执行次数最多的语句

最近更新时间: 2024-06-12 15:06:00

```
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;
```

获取执行总时间最长的语句

最近更新时间: 2024-06-12 15:06:00

```
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;
```

获取每句平均执行时间最长的语句

最近更新时间: 2024-06-12 15:06:00

```
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;
```

获取buffer读最多的语句

最近更新时间: 2024-06-12 15:06:00

```
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;
```

插件pg_trgm使用
pg_trgm功能介绍及添加方法

最近更新时间: 2024-06-12 15:06:00

功能介绍

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。TDSQL PG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，TDSQL PG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。

添加方法

```
create extension pg_trgm;
```

测试环境准备

最近更新时间: 2024-06-12 15:06:00

```
postgres=# create table t_trgm (id int,trgm text,no_trgm text) ;
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=#
```

gist索引测试
创建索引消耗时间

最近更新时间: 2024-06-12 15:06:00

```
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
```

索引占用空间

最近更新时间: 2024-06-12 15:06:00

```
postgres=# select pg_size_pretty(pg_indexes_size('t_trgm'));
pg_size_pretty
----------------
178 MB
(1 row)
```

模糊查询测试

最近更新时间: 2024-06-12 15:06:00

- 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。

数据导入时间测试

最近更新时间: 2024-06-12 15:06:00

```
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)
```

pgbench并发查询测试

最近更新时间: 2024-06-12 15:06:00

```
[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)||'%';
```

pgbench并发写入测试

最近更新时间: 2024-06-12 15:06:00

```
[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) ;
```

gin索引测试
创建索引消耗时间

最近更新时间: 2024-06-12 15:06:00

```
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=#
```

索引占用空间

最近更新时间: 2024-06-12 15:06:00

```
postgres=# select pg_size_pretty(pg_indexes_size('t_trgm'));
pg_size_pretty
----------------
74 MB
(1 row)
```

模糊查询测试

最近更新时间: 2024-06-12 15:06:00

- 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。

数据导入时间测试

最近更新时间: 2024-06-12 15:06:00

```
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)
```

pgbench并发查询测试

最近更新时间: 2024-06-12 15:06:00

```
[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)||'%';
```

pgbench并发写入测试

最近更新时间: 2024-06-12 15:06:00

```
[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)||'%';
```

无索引字段测试
pgbench并发查询测试

最近更新时间: 2024-06-12 15:06:00

```
[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)||'%';
```

pgbench并发写入测试

最近更新时间: 2024-06-12 15:06:00

```
[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) ;
```

数据对比总结及例外
数据对比

最近更新时间: 2024-06-12 15:06:00

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。

插件postgis使用
postgis添加方法

最近更新时间: 2024-06-12 15:06:00

连接cn节点。

```
postgres=# set enable_sampling_analyze to off;
postgres=# create extension postgis;
postgres=# set enable_sampling_analyze to on;
```

关于存储过程
存储过程语法介绍
建立存储语法

最近更新时间: 2024-06-12 15:06:00

语法概述

```
CREATE [ ORREPLACE ] PROCEDURE [ schema. ]procedure
[ (argument [ { IN | OUT | IN OUT } ]
[ NOCOPY ]
datatype [ DEFAULT expr ]
[, argument [ { IN | OUT | IN OUT } ]
[ NOCOPY ]
datatype [ DEFAULT expr ]
]...
)
]
[ invoker_rights_clause ]
{ IS | AS }
{ pl/sql_subprogram_body | call_spec } ;
```

示例

```
create orreplace procedure exe_imed_pro() as
begin
perform dbms_output.put_line('tbase');
end;
```

callexe_imed_pro();

- IN：指定IN表示在调用过程时必须为参数提供一个值。
- OUT：指定OUT以指示该过程在执行后将该参数的值传递回其调用环境。
- IN OUT：指定IN OUT表示在调用过程时必须为参数提供一个值，并且该过程在执行后将值传递回其调用环境。
- 如果省略IN，OUT和IN OUT，则参数默认为IN。

示例：

```
create or replace procedure demo4(a_int in integer)
as
begin
a_int:=1;
end;
/
```

[OR REPLACE] 更新存储介绍

最近更新时间: 2024-06-12 15:06:00

带OR REPLACE的作用就存储过程存在时则替换的功能，建立存储时不带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='proc_1';
prosrc
--------------------------------
+
begin +
raise notice 'Hello TBase';+
end; +

(1 row)
postgres=# CREATE OR REPLACE PROCEDURE proc_1() AS
$$
begin
raise notice 'Hello，TBase';
end;
$$
LANGUAGE PLPGSQL;
CREATE PROCEDURE
postgres=# select prosrc from pg_proc where proname='proc_1';
prosrc
--------------------------------
+
begin +
raise notice 'Hello，TBase';+
end; +
(1 row)
postgres=#
postgres=# call proc_1();
NOTICE: Hello，TBase
CALL
postgres=#
```

另外也可以写成这样：

```
postgres=# CREATE OR REPLACE PROCEDURE proc_1() AS
begin
raise notice 'Hello，TBase';
end;
/
```

即不要$$开始和结束符号，最后使用/结束，这个语法更接近于oralce。

[模式名.]存储过程名介绍

最近更新时间: 2024-06-12 15:06:00

建立存储过程，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='proc_1';
nspname | prosrc
---------+---------------------------------
public | +
| begin +
| raise notice 'Hello，TBase';+
| end; +
|
(1 row)
```

因为$user模式不存在，所以存在public模式下。

存储过程与函数不能同名

最近更新时间: 2024-06-12 15:06:00

如创建一个与函数同名的存储过程会提示function xxx already exists with same argument types。

```
postgres=# CREATE OR REPLACE FUNCTION proc_1() RETURNS void AS
$$
begin
raise notice 'Hello TBase';
end;
$$
LANGUAGE PLPGSQL;
CREATE FUNCTION
postgres=# CREATE PROCEDURE proc_1() AS
$$
begin
raise notice 'Hello TBase';
end;
$$
LANGUAGE PLPGSQL;
ERROR: function "proc_1" already exists with same argument types
postgres=#
#如果你要替换，则提示
postgres=# CREATE OR REPLACE PROCEDURE proc_1() AS
$$
begin
raise notice 'Hello TBase';
end;
$$
LANGUAGE PLPGSQL;
ERROR: cannot change routine kind
DETAIL: "proc_1" is a function.
postgres=#
```

删除存储过程

删除不带参数的存储过程

最近更新时间: 2024-06-12 15:06:00

```
postgres=# CREATE OR REPLACE PROCEDURE proc_1() AS
$$
begin
raise notice 'Hello TBase';
end;
$$
LANGUAGE PLPGSQL;
CREATE PROCEDURE
postgres=# drop procedure proc_1 ( );
DROP PROCEDURE
postgres=#
```

删除带参数的存储过程

最近更新时间: 2024-06-12 15:06:00

```
postgres=# CREATE OR REPLACE PROCEDURE proc_1(a_int int) AS
$$
begin
raise notice '%',a_int;
end;
$$
LANGUAGE PLPGSQL;
CREATE PROCEDURE
postgres=# drop procedure proc_1 ( a_int int);
DROP PROCEDURE
postgres=#
#也可以只指定参数的类型即可
postgres=# drop procedure proc_1 (int);
DROP PROCEDURE
postgres=#
```

存储过程修改名称
修改不带参数的存储过程名称

最近更新时间: 2024-06-12 15:06:00

```
postgres=# CREATE OR REPLACE PROCEDURE proc_1() AS
$$
begin
raise notice 'Hello TBase';
end;
$$
LANGUAGE PLPGSQL;
CREATE PROCEDURE
postgres=# alter procedure proc_1() rename to proc_1_1;
ALTER PROCEDURE
postgres=#
```

修改带参数的存储过程名称

最近更新时间: 2024-06-12 15:06:00

```
postgres=# CREATE OR REPLACE PROCEDURE proc_1(a_int int) AS
$$
begin
raise notice '%',a_int;
end;
$$
LANGUAGE PLPGSQL;
CREATE PROCEDURE
postgres=# alter procedure proc_1 (a_int int) rename to proc_1_1;
ALTER PROCEDURE
```

修改存储过程所属schema
修改不带参数的存储过程schema

最近更新时间: 2024-06-12 15:06:00

```
postgres=# CREATE OR REPLACE PROCEDURE public.proc_1() AS
$$
begin
raise notice 'Hello TBase';
end;
$$
LANGUAGE PLPGSQL;
CREATE PROCEDURE
postgres=#
postgres=# alter procedure public.proc_1() set schema myche;
ALTER PROCEDURE
postgres=#
```

修改带参数的存储过程schema

最近更新时间: 2024-06-12 15:06:00

```
postgres=# CREATE OR REPLACE PROCEDURE public.proc_1(a_int int) AS
$$
begin
raise notice '%',a_int;
end;
$$
LANGUAGE PLPGSQL;
CREATE PROCEDURE
postgres=# alter procedure public.proc_1 (int) set schema myche;
ALTER PROCEDURE
postgres=#
```

修改存储过程所属用户
修改不带参数的存储过程所属用户

最近更新时间: 2024-06-12 15:06:00

```
postgres=# CREATE OR REPLACE PROCEDURE public.proc_1() AS
$$
begin
raise notice 'Hello TBase';
end;
$$
LANGUAGE PLPGSQL;
CREATE PROCEDURE
postgres=# alter procedure proc_1() owner to tbase_01_admin;
ALTER PROCEDURE
postgres=#
```

修改带参数的存储过程所属用户

最近更新时间: 2024-06-12 15:06:00

```
postgres=# CREATE OR REPLACE PROCEDURE proc_1(a_int int) AS
$$
begin
raise notice '%',a_int;
end;
$$
LANGUAGE PLPGSQL;
CREATE PROCEDURE
postgres=# alter procedure proc_1 (int) owner to tbase_01_admin;
ALTER PROCEDURE
postgres=#
```

存储过程执行
不带参数

最近更新时间: 2024-06-12 15:06:00

```
postgres=# create or replace procedure p_no_para() as
$$
begin
raise notice 'no_para procedure';
end;
$$
language plpgsql;
CREATE PROCEDURE
postgres=# call p_no_para();
NOTICE: no_para procedure
CALL
postgres=#
```

带参数

最近更新时间: 2024-06-12 15:06:00

```
postgres=# create or replace procedure p_para(a_int integer) as
$$
begin
raise notice 'a_int = %',a_int;
end;
$$
language plpgsql;
CREATE PROCEDURE
postgres=# call p_para(1);
NOTICE: a_int = 1
CALL
postgres=#
```

中途return返回

最近更新时间: 2024-06-12 15:06:00

过程没有返回值。因此，过程的结束可以不用RETURN语句。 如果想用一个RETURN语句提前退出代码，只需写一个没有表达式的RETURN。

```
postgres=# CREATE OR REPLACE PROCEDURE p_return(a_return integer) AS
$$
BEGIN
if a_return > 1 then
raise notice '提前返回';
return;
end if;
raise notice '结束返回';
END;
$$
LANGUAGE plpgsql;
CREATE PROCEDURE
postgres=# CALL p_return(2);
NOTICE: 提前返回
CALL
postgres=#
```

参数详细介绍
参数模式
IN模式

最近更新时间: 2024-06-12 15:06:00

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 PROCEDURE p_in(IN a_xm text) AS
$$
BEGIN
RAISE NOTICE 'a_xm=%',a_xm;
END;
$$
LANGUAGE PLPGSQL;
CREATE PROCEDURE
postgres=# CALL p_in('Tbase');
NOTICE: a_xm=Tbase
CALL
postgres=#

postgres=# CREATE OR REPLACE PROCEDURE p_in_default(a_xm text) AS
$$
BEGIN
RAISE NOTICE 'a_xm=%',a_xm;
END;
$$
LANGUAGE PLPGSQL;
CREATE PROCEDURE
postgres=# CALL p_in_default('Tbase');
NOTICE: a_xm=Tbase
CALL
```

上面两种方式定义的参数效果是一样的。

INOUT模式

最近更新时间: 2024-06-12 15:06:00

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 PROCEDURE p_inout(INOUT a_xm text) AS
$$
BEGIN
RAISE NOTICE 'a_xm=%',a_xm;
END;
$$
LANGUAGE PLPGSQL;
CREATE PROCEDURE
postgres=# CALL p_inout('Tbase');
NOTICE: a_xm=Tbase
a_xm
-------
Tbase
(1 row)
```

参数引用
无命名参数

最近更新时间: 2024-06-12 15:06:00

```
postgres=# CREATE OR REPLACE PROCEDURE p_unname(text) AS
$$
BEGIN
raise notice '$1=%',$1;
END;
$$
LANGUAGE PLPGSQL;
CREATE PROCEDURE
postgres=# call ps_unname('Tbase');
NOTICE: $1=Tbase
CALL
postgres=#
```

给标识符指定别名

最近更新时间: 2024-06-12 15:06:00

```
postgres=# CREATE OR REPLACE PROCEDURE p_specify_name(text) AS
$$
DECLARE
a_xm ALIAS FOR $1; #a_xm是$1的别名
BEGIN
raise notice '$1=%',a_xm;
END;
$$
LANGUAGE PLPGSQL;
CREATE PROCEDURE
postgres=# CALL p_specify_name('Tbase');
NOTICE: $1=Tbase
CALL
postgres=#
```

命名参数

最近更新时间: 2024-06-12 15:06:00

```
postgres=# CREATE OR REPLACE PROCEDURE p_name(a_xm text) AS
$$
BEGIN
raise notice '$1=%',a_xm;
END;
$$
LANGUAGE PLPGSQL;
CREATE PROCEDURE
postgres=# call p_name('Tbase');
NOTICE: $1=Tbase
CALL
postgres=#
```

参数数据类型
基本类型

最近更新时间: 2024-06-12 15:06:00

```
postgres=# CREATE OR REPLACE PROCEDURE p_base_para (a_int integer,a_str text) AS
$$
BEGIN
RAISE NOTICE 'a_int = % ; a_str = %',a_int,a_str;
END;
$$
LANGUAGE PLPGSQL;
CREATE PROCEDURE
postgres=# CALL p_base_para(1,'Tbase');
NOTICE: a_int = 1 ; a_str = Tbase
CALL
postgres=#

postgres=# CREATE OR REPLACE PROCEDURE p_base_array (a_int integer[],a_str text[]) AS
$$
BEGIN
RAISE NOTICE 'a_int = % ; a_str = %',a_int,a_str;
END;
$$
LANGUAGE PLPGSQL;
CREATE PROCEDURE
postgres=# CALL p_base_array(ARRAY[1,2,3],ARRAY['TBase','pgxz']);
NOTICE: a_int = {1,2,3} ; a_str = {TBase,pgxz}
CALL
postgres=#
```

复合类型

最近更新时间: 2024-06-12 15:06:00

```
postgres=# CREATE TYPE public.t_per AS
(
id integer,
mc text
);
CREATE TYPE
postgres=# CREATE OR REPLACE PROCEDURE p_type (a_row public.t_per) AS
$$
BEGIN
RAISE NOTICE 'id = % ; mc = %',a_row.id,a_row.mc;
END;
$$
LANGUAGE PLPGSQL;
CREATE PROCEDURE
postgres=# CALL p_type(ROW(1,'TBase')::public.t_per);
NOTICE: id = 1 ; mc = TBase
CALL
postgres=#
```

- 复合数组。

```
postgres=# CREATE OR REPLACE PROCEDURE p_type_array (a_rec public.t_per[]) AS
$$
BEGIN
RAISE NOTICE 'a_rec = %',a_rec;
RAISE NOTICE 'a_rec[1].id = %',a_rec[1].id;
END;
$$
LANGUAGE PLPGSQL;
CREATE PROCEDURE
postgres=# CALL p_type_array (ARRAY[ROW(1,'TBase'),ROW(1,'pgxz')]::public.t_per[]);
NOTICE: a_rec = {"(1,TBase)","(1,pgxz)"}
NOTICE: a_rec[1].id = 1
CALL
postgres=#
```

行类型

最近更新时间: 2024-06-12 15:06:00

```
postgres=# create table public.t(id int,mc text);
CREATE TABLE
postgres=#
postgres=# CREATE OR REPLACE PROCEDURE p_row (a_row public.t) AS
$$
BEGIN
RAISE NOTICE 'id = % ; mc = %',a_row.id,a_row.mc;
END;
$$
LANGUAGE PLPGSQL;
CREATE PROCEDURE
postgres=# CALL p_row(ROW(1,'TBase'));
NOTICE: id = 1 ; mc = TBase
CALL
postgres=#
```

- 行数组。

```
postgres=# CREATE OR REPLACE PROCEDURE p_row_array (a_rec public.t[]) AS
$$
BEGIN
RAISE NOTICE 'a_rec = %',a_rec;
RAISE NOTICE 'a_rec[1].id = %',a_rec[1].id;
END;
$$
LANGUAGE PLPGSQL;
CREATE PROCEDURE
postgres=# CALL p_row_array(array[row(1,'TBase'),row(1,'pgxz')]::public.t[]);
NOTICE: a_rec = {"(1,TBase)","(1,pgxz)"}
NOTICE: a_rec[1].id = 1
CALL
postgres=#
```

游标类型

最近更新时间: 2024-06-12 15:06:00

```
postgres=# CREATE OR REPLACE PROCEDURE p_refcursor (a_ref refcursor) AS
$$
DECLARE
v_rec record;
BEGIN
OPEN a_ref FOR SELECT * FROM t LIMIT 1;
FETCH a_ref INTO v_rec;
RAISE NOTICE 'v_rec = % ',v_rec;
END;
$$
LANGUAGE PLPGSQL;
CREATE PROCEDURE
postgres=# CALL p_refcursor('a');
NOTICE: v_rec = (1,Tbase)
CALL
postgres=#
```

多态类型

最近更新时间: 2024-06-12 15:06:00

```
postgres=# CREATE OR REPLACE PROCEDURE f_any(a_arg anyelement) AS
$$
BEGIN
RAISE NOTICE '%',a_arg;
END;
$$
LANGUAGE PLPGSQL;
CREATE PROCEDURE
postgres=# CALL f_any(1);
NOTICE: 1
CALL
postgres=# CALL f_any('Tbase'::varchar);
NOTICE: Tbase
CALL
postgres=#
postgres=# SELECT f_any('TBase'::TEXT);
NOTICE: TBase
f_any
-------
(1 行记录)
postgres=# CALL f_any(ROW(1,'TBase')::public.t);
NOTICE: (1,TBase)
CALL
postgres=#
postgres=# CALL f_any(ARRAY[1,2]::INTEGER[]);
NOTICE: {1,2}
CALL
postgres=#
postgres=# CALL f_any(ARRAY[[1,2],[3,4],[5,6]]::INTEGER[][][]);
NOTICE: {{1,2},{3,4},{5,6}}
CALL
postgres=#
#注意多态类型参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 PROCEDURE f_any(a_arg anyarray) AS
$$
BEGIN
RAISE NOTICE '%',a_arg;
END;
$$
LANGUAGE PLPGSQL;
CREATE PROCEDURE
postgres=# call f_any(ARRAY['TBase','pgxz']::TEXT[]);
ERROR: procedure f_any(text[]) is not unique
LINE 1: call f_any(ARRAY['TBase','pgxz']::TEXT[]);
^
HINT: Could not choose a best candidate procedure. You might need to add explicit type casts.
postgres=#
```

> **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。

参数默认值

最近更新时间: 2024-06-12 15:06:00

```
postgres=# CREATE OR REPLACE PROCEDURE p_default_value (a_int INTEGER DEFAULT 1) AS
$$
BEGIN
RAISE NOTICE 'a_int = %',a_int;
END;
$$
LANGUAGE PLPGSQL;
CREATE PROCEDURE
postgres=# CALL p_default_value(2);
NOTICE: a_int = 2
CALL
postgres=# CALL p_default_value();
NOTICE: a_int = 1
CALL
postgres=#
#备注：如果原来存在一个p_default_value ()这样的存储过程，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 PROCEDURE p_default_value() AS
$$
BEGIN
RAISE NOTICE '无参数';
END;
$$
LANGUAGE plpgsql ;
CREATE PROCEDURE
postgres=# CALL p_default_value();
ERROR: procedure p_default_value() is not unique
LINE 1: CALL p_default_value();
^
HINT: Could not choose a best candidate procedure. You might need to add explicit type casts.
postgres=#
```

出错提示，p_default_value ()存储过程不是唯一的，这是使用上一个需要特别注意的地方。

变量使用
变量使用介绍
%TYPE

最近更新时间: 2024-06-12 15:06:00

语法：

```
{ collection_name
| cursor_variable_name
| object name
| record_name
| record_name . field_name
| db_table_name . column_name
| variable_name
}
%TYPE
```

示例：

```
create table t4 (f1 integer,f2 varchar2(10));
insert into t4 values(1,'tbase');
commit;
DECLARE
v_t4_f2 t4.f2%type;
begin
select f2 into v_t4_f2 from t4 where f1=1;
perform dbms_output.put_line('v_t4_f2='||v_t4_f2);
end;
/
```

%ROWTYPE

最近更新时间: 2024-06-12 15:06:00

示例：%rowtype当作参数和返回值

```
-- 9 rowtype as parameter
set enable_oracle_compatible to on;

drop table if exists t_rowtype_20221111_1;
create table t_rowtype_20221111_1(a int, b int);
insert into t_rowtype_20221111_1 values(generate_series(1, 10), generate_series(1, 10));

create table t_rowtype_20221111_2(a int);
insert into t_rowtype_20221111_2 values(generate_series(1, 10));

create table t_rowtype_20221111_3(a text, b text);
insert into t_rowtype_20221111_3 values(generate_series(1, 10), generate_series(1, 10));

drop table if exists t_rowtype_20221111_4;
drop view if exists tv_rowtype_20221111_1;
drop table if exists t_rowtype_20221111_5;
create table t_rowtype_20221111_4(a int, b varchar2(50), c number);
create or replace view tv_rowtype_20221111_1 as select a, b from t_rowtype_20221111_4;
insert into t_rowtype_20221111_4 values(generate_series(1, 10), chr(generate_series(1,10) + 96), generate_series(1, 10));
create table t_rowtype_20221111_5(key1 int, des1 varchar2(50));

drop function if exists f2f_rowtype_20221111_1;
create or replace function f2f_rowtype_20221111_1(e int, b t_rowtype_20221111_4%rowtype) return int is
res t_rowtype_20221111_4%rowtype;
begin
res := b;
select * into res from t_rowtype_20221111_4 where t_rowtype_20221111_4.a = e;
insert into t_rowtype_20221111_5 values(res.a, res.b);
return b.b;
end;
/

drop function if exists p2p_rowtype_20221111_1;
create or replace procedure p2p_rowtype_20221111_1(e int, b t_rowtype_20221111_4%rowtype) is
res t_rowtype_20221111_4%rowtype;
begin
res := b;
select * into res from t_rowtype_20221111_4 where t_rowtype_20221111_4.a = e;
insert into t_rowtype_20221111_5 values(res.a, res.b);
end;
/

drop function if exists p4p;
create or replace procedure p4p(e1 int, e2 int, f out t_rowtype_20221111_4%rowtype, g IN OUT t_rowtype_20221111_4%r
owtype) is
res t_rowtype_20221111_4%rowtype;
begin
res := g;
insert into t_rowtype_20221111_5 values(res.a, 'inout :' || res.b);
select * into res from t_rowtype_20221111_4 where t_rowtype_20221111_4.a = e1;
```

```
f := res;
select * into res from t_rowtype_20221111_4 where t_rowtype_20221111_4.a = e2;
g := res;
end;
/

create or replace package pkg_rowtype_20221111_1 is
dft_rt t_rowtype_20221111_4%rowtype;
end;
/

create or replace package pkg_rowtype_20221111_2 is
dft_rt t_rowtype_20221111_4%rowtype;
CURSOR c1 is select * from t_rowtype_20221111_4;
dft_rt1 c1%rowtype;
function pkg_f_rowtype_20221111_3(a int, b c1%rowtype) return c1%rowtype;
end;
/

create or replace package body pkg_rowtype_20221111_2 is
function pkg_f_rowtype_20221111_3(a int, b c1%rowtype) return c1%rowtype is
val c1%rowtype;
begin
val := b;
select * into val from t_rowtype_20221111_4;
return val;
end;
end;
/

begin
select * into pkg_rowtype_20221111_2.dft_rt1 from t_rowtype_20221111_4;
end;
/

insert into t_rowtype_20221111_4 values(1, 'pkg_f', 2);

declare
v1 pkg_rowtype_20221111_2.c1%rowtype;
v2 pkg_rowtype_20221111_2.c1%rowtype;
a int := 3;
begin
v1 := pkg_rowtype_20221111_2.pkg_f_rowtype_20221111_3(a, v2);
end;
/

drop function if exists p3p_rowtype_20221111_2;
-- pl/sql not support using package var as default parameter
-- create or replace procedure p3p_rowtype_20221111_2(e int, a t_rowtype_20221111_4%rowtype default pkg_rowtype_20
221111_1.dft_rt, b t_rowtype_20221111_4%rowtype default null) is
create or replace procedure p3p_rowtype_20221111_2(e int, b t_rowtype_20221111_4%rowtype default null) is
buf t_rowtype_20221111_4%rowtype;
begin
if b is null then
select * into buf from t_rowtype_20221111_4 where t_rowtype_20221111_4.a = e;
insert into t_rowtype_20221111_5 values(buf.a, 'null test, defualt val:' || buf.b);
else
```

```
insert into t_rowtype_20221111_5 values(-1, 'not support null test of record type');
end if;
end;
/

declare
p1 t_rowtype_20221111_4%rowtype;
p2 t_rowtype_20221111_4%rowtype;
res int;
begin
res := f2f_rowtype_20221111_1(5, p1);
call p2p_rowtype_20221111_1(3, p1);
p2.a := 12;
p2.b := 'z';
p2.c := 12;
call p4p(6,7,p1,p2);
insert into t_rowtype_20221111_5 values(p1.a, 'OUT: ' || p1.b);
insert into t_rowtype_20221111_5 values(p2.a, 'OUT: ' || p2.b);
pkg_rowtype_20221111_1.dft_rt.a := 13;
pkg_rowtype_20221111_1.dft_rt.b := 'x';
pkg_rowtype_20221111_1.dft_rt.c := 13;
call p3p_rowtype_20221111_2(8);
end;
/
select * from t_rowtype_20221111_5 order by key1, des1;

-- 10 rowtype as return
drop function if exists f3f_rowtype_20221111_2;
create or replace function f3f_rowtype_20221111_2(e int, b int, c int) return t_rowtype_20221111_4%rowtype is
res t_rowtype_20221111_4%rowtype;
begin
select * into res from t_rowtype_20221111_4 where t_rowtype_20221111_4.a = e;
return res;
end;
/
declare
p1 t_rowtype_20221111_4%rowtype;
begin
p1 := f3f_rowtype_20221111_2(6, 2, 2);
insert into t_rowtype_20221111_5 values(p1.a, p1.b);
end;

/
select * from t_rowtype_20221111_5 order by key1, des1;
```

> **注意：**
>
> 暂不支持游标使用%rowtype。

记录类型

最近更新时间: 2024-06-12 15:06:00

语法 `TYPE rec_type IS RECORD ( fields )` 示例：

```
CREATE SCHEMA IF NOT EXISTS test_oracle;
set search_path to test_oracle;
create table tbl_city(id int, population int, nation varchar(32), city varchar(32), gdp int, others text);

insert into tbl_city(id, population, nation, city, gdp) values(1,500, 'China','Guangzhou', 23000);
insert into tbl_city(id, population, nation, city, gdp) values(2,1500, 'China', 'Shanghai', 29000);
insert into tbl_city(id, population, nation, city, gdp) values(3,1500, 'China', 'Beijing', 25000);
insert into tbl_city(id, population, nation, city, gdp) values(4,1000, 'China', 'Shenzhen', 24000);
insert into tbl_city(id, population, nation, city, gdp) values(5,1000,'USA','New York', 35000);
insert into tbl_city(id, population, nation, city, gdp) values(6,500, 'USA', 'Bostom', 15000);
insert into tbl_city(id, population, nation, city, gdp) values(7,500, 'Japan','Tokyo', 40000);
insert into tbl_city(id, population, nation, city, gdp) values(8,800, 'China', 'Hongkong', 23500);
insert into tbl_city(id, population, nation, city, gdp) values(9,800, 'China', 'Hangzhou', 15500);
insert into tbl_city(id, population, nation, city, gdp) values(10,100, 'USA', 'Los Angele', 15500);


set search_path to test_oracle;
DROP FUNCTION catcity;
CREATE FUNCTION catcity(nation_name text) RETURNS text AS $$
DECLARE
TYPE mycitytype IS RECORD(city text);
trow tbl_city%ROWTYPE;
rlt text;
lt mycitytype;
BEGIN
rlt = ':' || nation_name;
FOR trow IN SELECT * FROM tbl_city WHERE nation = nation_name
LOOP

lt.city = trow.city;
rlt = lt.city ||'+' || rlt;
END LOOP;
return rlt;
END;
$$ LANGUAGE plpgsql;

select catcity('China');
```

变量使用示例
变量声明语法

最近更新时间: 2024-06-12 15:06:00

`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NOTNULL，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 :=。

定义一个普通变量

最近更新时间: 2024-06-12 15:06:00

```
postgres=# CREATE OR REPLACE PROCEDURE ordinary_var() AS
$$
DECLARE
#所有变量的声明都要放在这里,建议变量以v_开头,参数以a_开头
v_int integer := 1;
v_text text;
BEGIN
v_text = 'TBase';
RAISE NOTICE 'v_int = %',v_int;
RAISE NOTICE 'v_text = %',v_text;
END;
$$
LANGUAGE plpgsql;
CREATE PROCEDURE
postgres=# CALL ordinary_var();
NOTICE: v_int = 1
NOTICE: v_text = TBase
CALL
postgres=#
```

定义CONSTANT变量

最近更新时间: 2024-06-12 15:06:00

```
postgres=# CREATE OR REPLACE PROCEDURE p_constant() AS
$$
DECLARE
v_int CONSTANT integer := 1;
BEGIN
RAISE NOTICE 'v_int = %',v_int;
END;
$$
LANGUAGE plpgsql;
CREATE PROCEDURE
postgres=# CALL p_constant();
NOTICE: v_int = 1
CALL
#CONSTANT 不能再次赋值
postgres=# CREATE OR REPLACE PROCEDURE p_constant() AS
$$
DECLARE
v_int CONSTANT integer := 1;
BEGIN
RAISE NOTICE 'v_int = %',v_int;
v_int = 10;
END;
$$
LANGUAGE plpgsql;
ERROR: "v_int" is declared CONSTANT
LINE 7: v_int = 10;
^
postgres=#
```

定义NOT NULL变量

最近更新时间: 2024-06-12 15:06:00

```
postgres=# CREATE OR REPLACE PROCEDURE p_not_null_var() AS
$$
DECLARE
v_int integer NOT NULL := 1;
BEGIN
RAISE NOTICE 'v_int = %',v_int;
SELECT NULL INTO v_int;
RAISE NOTICE 'v_int = %',v_int;
END;
$$
LANGUAGE plpgsql;
CREATE PROCEDURE
postgres=# CALL p_not_null_var();
NOTICE: v_int = 1
ERROR: null value cannot be assigned to variable "v_int" declared NOT NULL
CONTEXT: PL/pgSQL function p_not_null_var() line 6 at SQL statement
postgres=#
```

定义为NOT NULL变量，则该变量受NOT NULL约束。

定义COLLATE变量

最近更新时间: 2024-06-12 15:06:00

- 按unicode值对比大小。

```
postgres=# CREATE OR REPLACE PROCEDURE p_collate_unicode() AS
$$
DECLARE
v_txt1 TEXT COLLATE "C" := '严';
v_txt2 TEXT COLLATE "C" := '丰';
BEGIN
IF v_txt1 > v_txt2 THEN
RAISE NOTICE ' % > % ',v_txt1,v_txt2;
ELSE
RAISE NOTICE ' % > % ',v_txt2,v_txt1;
END IF;
END;
$$
LANGUAGE plpgsql;
CREATE PROCEDURE
postgres=# CALL p_collate_unicode();
NOTICE: 丰 > 严
CALL
postgres=#

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 PROCEDURE p_collate_pinyin() AS
$$
DECLARE
v_txt1 TEXT COLLATE "zh_CN.utf8" := '严';
v_txt2 TEXT COLLATE "zh_CN.utf8" := '丰';
BEGIN
IF v_txt1 > v_txt2 THEN
RAISE NOTICE ' % -> % ',v_txt1,v_txt2;
ELSE
RAISE NOTICE ' % -> % ',v_txt2,v_txt1;
END IF;
END;
$$
LANGUAGE plpgsql;
```

```
CREATE PROCEDURE
postgres=# CALL p_collate_pinyin();
NOTICE: 严 -> 丰
CALL
postgres=#
```

变量赋值

最近更新时间: 2024-06-12 15:06:00

```
postgres=# CREATE OR REPLACE PROCEDURE p_setval() AS
$$
DECLARE
#定义时赋值
v_int1 integer = 1;
--使用 :=兼容于plsql
v_int2 integer := 1;
v_txt1 text;
v_float float8;
--使用查询赋值
v_relname text = (select relname FROM pg_class LIMIT 1);
v_relpages integer;
v_rec RECORD;
BEGIN
#在函数体中赋值
v_txt1 = 'TBase';
v_float = random();
#使用查询赋值的另一种方式
SELECT relname,relpages INTO v_relname,v_relpages FROM pg_class ORDER BY random() LIMIT 1;
RAISE NOTICE 'v_relname = % , relpages = %',v_relname,v_relpages;
END;
$$
LANGUAGE plpgsql;
CREATE PROCEDURE
postgres=# CALL p_setval();
NOTICE: v_relname = pg_toast_17220_index , relpages = 1
CALL
postgres=#
```

BULK COLLECT

最近更新时间: 2024-06-12 15:06:00

BULK COLLECT将一个查询结果集保存起来。

- 用例1。

```
postgres=# create table t5(f1 integer,f2 varchar(10));
CREATE TABLE
postgres=# insert into t5 values(1,'tbase1');
INSERT 0 1
postgres=# insert into t5 values(2,'tbase2');
INSERT 0 1

postgres=# create or replace procedure p_bulk_collect()
AS
$$
declare
TYPE t5list IS TABLE OF t5.f2%TYPE;
t5s t5list;
BEGIN
SELECT f2 BULK COLLECT INTO t5s FROM t5;
FOR i IN t5s.FIRST .. t5s.LAST
LOOP
raise notice '%',t5s[i];
END LOOP;
END
$$language plpgsql;
NOTICE: type reference t5.f2%TYPE converted to character varying
CREATE PROCEDURE
postgres=# CALL p_bulk_collect();
NOTICE: tbase1
NOTICE: tbase2
CALL
```

- 用例2。

```
postgres=# create table tbl_person(id integer, name text, tdd int);
CREATE TABLE
postgres=# insert into tbl_person values(1,'tbase',1);
insert into tbl_person values(2,'pgxz',1);
INSERT 0 1
postgres=# insert into tbl_person values(2,'pgxz',1);
INSERT 0 1
postgres=# insert into tbl_person values(3,'pg',2);
INSERT 0 1

postgres=# create or replace procedure p_bulkcollect_select_into(a_tdd integer) AS
$$
declare
type TAPersonlist is table of tbl_person%rowtype;
vpa TAPersonlist;
rp tbl_person%rowtype;
```

```
begin
select * bulk collect into vpa from tbl_person where tdd = a_tdd;
raise notice 'count=%', vpa.count;
for i in vpa.first..vpa.last loop
rp = vpa[i];
raise notice 'loop=%', i;
raise notice 'vname=%', rp.name;
end loop;
raise notice 'vpa.count=%',vpa.count;
end;
$$language plpgsql;
CREATE PROCEDURE
postgres=# CALL p_bulkcollect_select_into(1);
NOTICE: count=2
NOTICE: loop=1
NOTICE: vname=tbase
NOTICE: loop=2
NOTICE: vname=pgxz
NOTICE: vpa.count=2
CALL
```

控制结构
判断语句
IF...THEN...END IF

最近更新时间: 2024-06-12 15:06:00

```
postgres=# CREATE OR REPLACE PROCEDURE p_if() AS
$$
BEGIN
IF random()>0.5 THEN
RAISE NOTICE '随机数大于0.5';
END IF;
END;
$$
LANGUAGE plpgsql;
CREATE PROCEDURE
postgres=# CALL p_if();
NOTICE: 随机数大于0.5
CALL
postgres=#
```

IF...THEN...ELSE...END IF

最近更新时间: 2024-06-12 15:06:00

```
postgres=# CREATE OR REPLACE PROCEDURE p_if_else() AS
$$
BEGIN
IF random()>0.99 THEN
RAISE NOTICE '随机数大于0.99';
ELSE
RAISE NOTICE '随机数小于或等于0.99';
END IF;
END;
$$
LANGUAGE plpgsql;
CREATE PROCEDURE
postgres=# CALL p_if_else();
NOTICE: 随机数小于或等于0.99
CALL
postgres=#
```

IF...THEN...ELSIF...THEN...ELSE...END IF

最近更新时间: 2024-06-12 15:06:00

```
postgres=# CREATE OR REPLACE PROCEDURE p_if_elsif() AS
$$
DECLARE
v_float8 float8 := random();
BEGIN
IF v_float8>0.99 THEN
RAISE NOTICE '随机数大于0.99';
ELSIF v_float8>0.5 THEN
RAISE NOTICE '随机数大于0.50';
ELSIF v_float8>0.25 THEN
RAISE NOTICE '随机数大于0.25';
ELSE
RAISE NOTICE '随机数小于或等于0.25';
END IF;
END;
$$
LANGUAGE plpgsql;
CREATE PROCEDURE
postgres=# CALL p_if_elsif();
NOTICE: 随机数大于0.50
CALL
```

CASE语句

最近更新时间: 2024-06-12 15:06:00

```
postgres=# CREATE OR REPLACE PROCEDURE p_case() AS
$$
DECLARE
v_float8 float8 := random();
BEGIN
CASE
WHEN v_float8>0.99 THEN
RAISE NOTICE '随机数大于0.99';
WHEN v_float8>0.5 THEN
RAISE NOTICE '随机数大于0.50';
WHEN v_float8>0.25 THEN
RAISE NOTICE '随机数大于0.25';
ELSE
RAISE NOTICE '随机数小于或等于0.25';
END CASE;
END;
$$
LANGUAGE plpgsql;
CREATE PROCEDURE
postgres=# CALL p_case();
NOTICE: 随机数小于或等于0.25
CALL
postgres=#
```

循环语句
LOOP循环

最近更新时间: 2024-06-12 15:06:00

```
postgres=# CREATE OR REPLACE PROCEDURE p_loop() AS
$$
DECLARE
v_id INTEGER := 1;
BEGIN
LOOP
RAISE NOTICE '%',v_id;
EXIT WHEN random()>0.8;
v_id := v_id + 1;
END LOOP ;
END;
$$
LANGUAGE plpgsql;
CREATE PROCEDURE
postgres=# CALL p_loop();
NOTICE: 1
NOTICE: 2
NOTICE: 3
CALL
postgres=#
```

WHILE循环

最近更新时间: 2024-06-12 15:06:00

```
postgres=# CREATE OR REPLACE PROCEDURE p_while() AS
$$
DECLARE
v_id INTEGER := 1;
v_random float8 ;
BEGIN
LOOP
RAISE NOTICE '%',v_id;
v_id := v_id + 1;
v_random := random();
IF v_random > 0.8 THEN
RETURN;
END IF;
END LOOP ;
END;
$$
LANGUAGE plpgsql;
CREATE PROCEDURE
postgres=# CALL p_while();
NOTICE: 1
CALL
```

FOR循环

最近更新时间: 2024-06-12 15:06:00

```
postgres=# CREATE OR REPLACE PROCEDURE p_for() AS
$$
BEGIN
FOR i IN 1..3 LOOP
RAISE NOTICE 'i = %',i;
END LOOP;
END;
$$
LANGUAGE plpgsql;
CREATE PROCEDURE
postgres=# CALL p_for();
NOTICE: i = 1
NOTICE: i = 2
NOTICE: i = 3
CALL
postgres=# CREATE OR REPLACE PROCEDURE p_for_reverse() AS
$$
BEGIN
FOR i IN REVERSE 3..1 LOOP
RAISE NOTICE 'i = %',i;
END LOOP;
END;
$$
LANGUAGE plpgsql;
CREATE PROCEDURE
postgres=# CALL p_for_reverse();
NOTICE: i = 3
NOTICE: i = 2
NOTICE: i = 1
CALL
使用REVERSE递减
postgres=# CREATE OR REPLACE PROCEDURE p_for_by() AS
$$
BEGIN
FOR i IN 1..8 BY 2 LOOP
RAISE NOTICE 'i = %',i;
END LOOP;
END;
$$
LANGUAGE plpgsql;
CREATE PROCEDURE
postgres=# CALL p_for_by();
NOTICE: i = 1
NOTICE: i = 3
NOTICE: i = 5
NOTICE: i = 7
CALL
postgres=#
#使用BY设置步长
```

FOR循环查询结果

最近更新时间: 2024-06-12 15:06:00

```
postgres=# CREATE OR REPLACE PROCEDURE p_for_record() AS
$$
DECLARE
v_rec RECORD;
BEGIN
FOR v_rec IN SELECT relname,relkind FROM pg_class limit 2 LOOP
RAISE NOTICE '%',v_rec;
END LOOP;
END;
$$
LANGUAGE plpgsql;
CREATE PROCEDURE
postgres=# CALL p_for_record();
NOTICE: (pg_stat_statements,v)
NOTICE: (pg_proc,v)
CALL
postgres=#
```

FOREACH循环一个数组

最近更新时间: 2024-06-12 15:06:00

```
postgres=# CREATE OR REPLACE PROCEDURE p_foreach() AS
$$
DECLARE
v_random_arr float8[]:=ARRAY[random(),random()];
v_random float8;
BEGIN
FOREACH v_random IN ARRAY v_random_arr LOOP
RAISE NOTICE '%',v_random ;
END LOOP;
END;
$$
LANGUAGE plpgsql;
CREATE PROCEDURE
postgres=# CALL p_foreach();
NOTICE: 0.744417542591691
NOTICE: 0.804096563253552
CALL
postgres=#
postgres=# CREATE OR REPLACE PROCEDURE p_foreach_slice() AS
$$
DECLARE
v_random_arr float8[][]:=ARRAY[ARRAY[random(),random()],ARRAY[random(),random()]];
v_random float8;
BEGIN
FOREACH v_random SLICE 0 IN ARRAY v_random_arr LOOP
RAISE NOTICE '%',v_random ;
END LOOP;
END;
$$
LANGUAGE plpgsql;
CREATE PROCEDURE
postgres=# CALL p_foreach_slice();
NOTICE: 0.0220407997258008
NOTICE: 0.898449067492038
NOTICE: 0.190678883343935
NOTICE: 0.103653562255204
CALL
postgres=#
#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 PROCEDURE p_foreach_slice_1() AS
$$
DECLARE
v_random_arr float8[][]:=ARRAY[ARRAY[random(),random()],ARRAY[random(),random()]];
v_random float8[];
BEGIN
FOREACH v_random SLICE 1 IN ARRAY v_random_arr LOOP
RAISE NOTICE '%',v_random ;
END LOOP;
END;
$$
LANGUAGE plpgsql;
```

```
CREATE PROCEDURE
postgres=# CALL p_foreach_slice_1();
NOTICE: {0.248282201588154,0.757913041394204}
NOTICE: {0.0194511725567281,0.43799454299733}
CALL
#通过一个正SLICE值，FOREACH通过数组的切片而不是单一元素迭代
```

其它控制语句
动态执行

最近更新时间: 2024-06-12 15:06:00

```
postgres=# CREATE OR REPLACE PROCEDURE p_execute() AS
$$
DECLARE
v_sql TEXT;
v_relname TEXT;
BEGIN
v_sql := 'SELECT relname FROM pg_class limit 1';
EXECUTE v_sql INTO v_relname;
RAISE NOTICE 'relname = %',v_relname;
END;
$$
LANGUAGE plpgsql;
CREATE PROCEDURE
postgres=# CALL p_execute();
NOTICE: relname = pg_stat_statements
CALL
postgres=#
#也可以使用immediate
postgres=# CREATE OR REPLACE PROCEDURE p_execute() AS
$$
DECLARE
v_sql TEXT;
v_relname TEXT;
BEGIN
v_sql := 'SELECT relname FROM pg_class limit 1';
EXECUTE immediate v_sql INTO v_relname;
RAISE NOTICE 'relname = %',v_relname;
END;
$$
LANGUAGE plpgsql;
CREATE PROCEDURE
postgres=# CALL p_execute();
NOTICE: relname = s1
CALL
postgres=#
#动态执行就是拼sql语句,然后使用EXECUTE命令执行
```

执行一个没有结果的命令

最近更新时间: 2024-06-12 15:06:00

```
postgres=# CREATE OR REPLACE PROCEDURE p_perform() AS
$$
BEGIN
perform md5(random()::text);
END;
$$
LANGUAGE plpgsql;
CREATE PROCEDURE
postgres=# call p_perform();
CALL
postgres=#
```

获取执行结果

最近更新时间: 2024-06-12 15:06:00

```
postgres=# CREATE OR REPLACE PROCEDURE p_found() AS
$$
DECLARE
v_relname TEXT;
BEGIN
SELECT relname INTO v_relname FROM pg_class limit 1;
IF FOUND THEN
RAISE NOTICE '查询到记录，值为%',v_relname;
ELSE
RAISE NOTICE '查不到记录' ;
END IF;
END;
$$
LANGUAGE plpgsql;
CREATE PROCEDURE
postgres=# CALL p_found();
NOTICE: 查询到记录，值为pg_stat_statements
CALL
```

获取影响行数

最近更新时间: 2024-06-12 15:06:00

```
postgres=# CREATE OR REPLACE PROCEDURE p_row_count() AS
$$
DECLARE
v_row_count BIGINT;
BEGIN
delete from t1;
GET DIAGNOSTICS v_row_count = ROW_COUNT;
RAISE NOTICE '查询到的记录数为 % ',v_row_count;
END;
$$
LANGUAGE plpgsql;
CREATE PROCEDURE
postgres=# call p_row_count();
NOTICE: 查询到的记录数为 3
CALL
postgres=#
```

GOTO

最近更新时间: 2024-06-12 15:06:00

```
postgres=# create or replace procedure p_goto(v_maxnum integer) as
$$
declare
maxnum integer;
begin
maxnum := v_maxnum;
for i in 1..maxnum loop
if i=3 then
goto label;
end if;
raise notice 'i=%',i;
end loop;
<<label>>
raise notice 'goto end';
end;
$$
language plpgsql;
CREATE PROCEDURE
postgres=# call p_goto(5);
NOTICE: i=1
NOTICE: i=2
NOTICE: goto end
CALL
postgres=#
```

go用于跳转到某个标签下。

俘获错误
错误俘获处理

最近更新时间: 2024-06-12 15:06:00

```
postgres=# CREATE OR REPLACE PROCEDURE p_exception(a_id integer,a_nc text) AS
$$
BEGIN
INSERT INTO t_exception VALUES(a_id,a_nc);
RETURN ;
EXCEPTION WHEN OTHERS THEN
RAISE NOTICE '执行出错';
END;
$$
LANGUAGE plpgsql;
CREATE PROCEDURE
postgres=#
postgres=# CALL p_exception(1,'Tbase');
CALL
postgres=# CALL p_exception(1,'Tbase');
NOTICE: 执行出错
CALL
```

获取错误相关信息

最近更新时间: 2024-06-12 15:06:00

```
postgres=# CREATE OR REPLACE PROCEDURE p_exception_error(a_id integer,a_nc text) AS
$$
DECLARE
v_sqlstate text;
v_context text;
v_message_text text;
BEGIN
INSERT INTO t_exception VALUES(a_id,a_nc);
RETURN ;
EXCEPTION WHEN OTHERS THEN
GET STACKED DIAGNOSTICS v_sqlstate = RETURNED_SQLSTATE,
v_message_text = MESSAGE_TEXT,
v_context = PG_EXCEPTION_CONTEXT;
RAISE NOTICE '错误代码 : %',v_sqlstate;
RAISE NOTICE '出错信息 : %',v_message_text;
RAISE NOTICE '发生异常语句 : %',v_context;
raise notice '错误代码 : % \n出错信息 : % 发生异常语句 : %',v_sqlstate ,v_message_text,v_context;
END;
$$
LANGUAGE plpgsql;
CREATE PROCEDURE
postgres=# CALL p_exception_error(2,'Tbase');
CALL
postgres=# CALL p_exception_error(2,'Tbase');
NOTICE: 错误代码 : 23505
NOTICE: 出错信息 : node:dn001, backend_pid:16204, nodename:dn001,backend_pid:16204,message:duplicate key value vio
lates 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 p_exception_error(integer,text) line 7 at SQL statement
NOTICE: 错误代码 : 23505 \n出错信息 : node:dn001, backend_pid:16204, nodename:dn001,backend_pid:16204,message:dup
licate key value violates unique constraint "t_exception_id_uidx" 发生异常语句 : SQL statement "INSERT INTO t_exception V
ALUES(a_id,a_nc)"
PL/pgSQL function p_exception_error(integer,text) line 7 at SQL statement
CALL
postgres=#
```

自治事务
在存储过程中commit和rollback

最近更新时间: 2024-06-12 15:06:00

```
postgres=# create table t_autonomous_transaction( f1 int,f2 int);
CREATE TABLE
postgres=# create or replace procedure t_autonomous_transaction_isnert() as
$$
begin
insert into t_autonomous_transaction values(1,1);
insert into t_autonomous_transaction values(2,2);
commit;
insert into t_autonomous_transaction values(3,3);
rollback;
end;
$$
language plpgsql;
CREATE PROCEDURE
postgres=# call t_autonomous_transaction_isnert();
CALL
postgres=# select * from t_autonomous_transaction ;
f1 | f2
----+----
1 | 1
2 | 2
(2 rows)
#在同一个存储过程中，记录1，2 commit，记录3 rollback
#另外，如果存储过程中使用了自治事务，则这个存储过程就不能放在一个事务中调用，如下
postgres=# begin;
BEGIN
postgres=# call t_autonomous_transaction_isnert();
ERROR: invalid transaction termination
CONTEXT: PL/pgSQL function t_autonomous_transaction_isnert() line 5 at COMMIT
postgres=# rollback;
#用例，每插入10条记录提交一次
postgres=# create or replace procedure t_autonomous_transaction_isnert() as
$$
begin
for i in 1 .. 100 loop
insert into t_autonomous_transaction values(i,i);
if mod(i,10)=0 then
commit;
end if;
insert into t_autonomous_transaction values(i,i);
end loop;
end;
$$
language plpgsql;
```

游标使用限制

最近更新时间: 2024-06-12 15:06:00

```
postgres=# create table t_ref(f1 int,f2 int);
CREATE TABLE
postgres=# insert into t_ref values(1,1),(2,2),(3,3);
COPY 3
postgres=# CREATE OR REPLACE procedure p_refcursor() AS
$$
DECLARE
v_ref refcursor;
v_rec record;
BEGIN
OPEN v_ref FOR SELECT * FROM t_ref;
fetch next from v_ref into v_rec ;
commit;
fetch next from v_ref into v_rec ;
END;
$$
LANGUAGE plpgsql;
CREATE PROCEDURE
postgres=# call p_refcursor();
ERROR: cursor "<unnamed portal 5>" does not exist
CONTEXT: PL/pgSQL function p_refcursor() line 9 at FETCH
```

在存储过程中如果使用了游标，则游标在commit后不可使用。

自冶事务与exception的限制

最近更新时间: 2024-06-12 15:06:00

```
postgres=# CREATE OR REPLACE PROCEDURE p_exception_error(a_id integer,a_nc text) AS
$$
DECLARE
v_sqlstate text;
v_context text;
v_message_text text;
BEGIN
INSERT INTO t_exception VALUES(a_id,a_nc);
COMMIT;
EXCEPTION WHEN OTHERS THEN
GET STACKED DIAGNOSTICS v_sqlstate = RETURNED_SQLSTATE,
v_message_text = MESSAGE_TEXT,
v_context = PG_EXCEPTION_CONTEXT;
RAISE NOTICE '错误代码 : %',v_sqlstate;
RAISE NOTICE '出错信息 : %',v_message_text;
RAISE NOTICE '发生异常语句 : %',v_context;
raise notice '错误代码 : % \n出错信息 : % 发生异常语句 : %',v_sqlstate ,v_message_text,v_context;
END;
$$
LANGUAGE plpgsql;
CREATE PROCEDURE
postgres=# call p_exception_error(1,'tbase');
NOTICE: 错误代码 : 2D000
NOTICE: 出错信息 : cannot commit while a subtransaction is active
NOTICE: 发生异常语句 : PL/pgSQL function p_exception_error(integer,text) line 8 at COMMIT
NOTICE: 错误代码 : 2D000 \n出错信息 : cannot commit while a subtransaction is active 发生异常语句 : PL/pgSQL function p_
exception_error(integer,text) line 8 at COMMIT
CALL
postgres=#
```

如果存储过程中使用了exception，就不能再使用自冶事务，在5.06版本中兼容这个问题。

消息及异常处理
RAISE NOTICE

最近更新时间: 2024-06-12 15:06:00

```
postgres=# CREATE OR REPLACE PROCEDURE p_raise() AS
$$
DECLARE
v_int INTEGER := 1;
BEGIN
RAISE NOTICE 'v_int = %, 随机数 = %',v_int,random();
END;
$$
LANGUAGE plpgsql;
CREATE PROCEDURE
postgres=# CALL p_raise();
NOTICE: v_int = 1, 随机数 = 0.668172102887183
CALL
postgres=#
```

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)。

RAISE EXCEPTION

最近更新时间: 2024-06-12 15:06:00

```
postgres=# CREATE OR REPLACE PROCEDURE p_exception() AS
$$
DECLARE
v_int INTEGER := 1;
BEGIN
RAISE EXCEPTION '程序EXCEPTION ';
--下面的语句不会再执行
RAISE NOTICE 'v_int = %, 随机数 = %',v_int,random();
END;
$$
LANGUAGE plpgsql;
CREATE PROCEDURE
postgres=# CALL p_exception();
ERROR: 程序EXCEPTION
CONTEXT: PL/pgSQL function p_exception() line 5 at RAISE
postgres=#
```

如果在事务中执行这个存储过程,则事务会中止(abort)。

RAISE EXCEPTION 自定义ERRCODE

最近更新时间: 2024-06-12 15:06:00

```
postgres=# CREATE OR REPLACE PROCEDURE p_exception_errcode() AS
$$
DECLARE
v_int INTEGER := 1;
BEGIN
RAISE EXCEPTION ' 程序EXCEPTION ' USING ERRCODE = '23505';
END;
$$
LANGUAGE plpgsql;
CREATE PROCEDURE
postgres=# CALL p_exception_errcode();
ERROR: 程序EXCEPTION
CONTEXT: PL/pgSQL function p_exception_errcode() line 5 at RAISE
postgres=#
#日志中会记录这个ERRCODE
2021-04-25 11:25:25.958 CST,"tbase","postgres",30486,coord(30486,135876),"172.16.64.8:62252",6084c50a.7716,coord(30
486,135876),91,"CALL",2021-04-25 09:25:30 CST,11/135876,0,ERROR,23505," 程序EXCEPTION ",,,,,"PL/pgSQL function p_ex
ception_errcode() line 5 at RAISE","CALL p_exception_errcode();",,,"psql"
```

oracle存储过程兼容性
不带参数的存储过程不需要括号

最近更新时间: 2024-06-12 15:06:00

```
postgres=# set enable_oracle_compatible to on;
SET
```

- 带括号创建存储过程。

```
postgres=# create or replace procedure no_para() is
begin
raise notice 'no_para';
end;
/
CREATE PROCEDURE
```

- 不带括号创建存储过程。

```
postgres=# create or replace procedure no_para is
begin
raise notice 'no_para';
end;
/
CREATE PROCEDURE
postgres=# call no_para();
NOTICE: no_para
CALL
```

支持使用is语法

最近更新时间: 2024-06-12 15:06:00

- as语法。

```
postgres=# create or replace procedure p_is() as
begin
raise notice '兼容 is 语法';
end;
/
CREATE PROCEDURE
```

- is 语法。

```
postgres=# create or replace procedure p_is() is
begin
raise notice '兼容 is 语法';
end;
/
CREATE PROCEDURE
postgres=# call p_is();
NOTICE: 兼容 is 语法
CALL
postgres=#
```

支持不使用$$语法

最近更新时间: 2024-06-12 15:06:00

- 使用$$符号。

```
postgres=# create or replace procedure p_dollor() is
$$
begin
raise notice '使用美元符号';
end;
$$
/
CREATE PROCEDURE
```

- 不使用$$符号。

```
postgres=# create or replace procedure p_dollor() is
begin
raise notice '不使用美元符号';
end;
/
CREATE PROCEDURE
postgres=# call p_dollor();
NOTICE: 不使用美元符号
CALL
postgres=#
```

支持使用”end存储过程名称“结束

最近更新时间: 2024-06-12 15:06:00

```
postgres=# create or replace procedure p_end_procedure() is
begin
raise notice '使用 "end 存储过程名称" 结束 ';
end p_end_procedure;
/
CREATE PROCEDURE
postgres=# call p_end_procedure();
NOTICE: 使用 "end 存储过程名称" 结束
CALL
postgres=#
```

使用“/“结束函数定义

最近更新时间: 2024-06-12 15:06:00

- 使用language plpgsql。

```
postgres=# create or replace procedure p_end_line() is
$$
begin
raise notice '使用斜线结束定义';
end ;
$$
language plpgsql;
CREATE PROCEDURE
postgres=#
```

- 使用斜线结束。

```
postgres=# create or replace procedure p_end_line() is
begin
raise notice '使用斜线结束定义';
end ;
/
CREATE PROCEDURE
postgres=# call p_end_line();
NOTICE: 使用斜线结束定义
CALL
postgres=#
```

定义变量无需要declare

最近更新时间: 2024-06-12 15:06:00

- declare用法。

```
postgres=# create or replace procedure p_declare() is
declare
v_int int;
begin
v_int:=1;
raise notice 'v_int = %',v_int;
end ;
/
CREATE PROCEDURE
```

- 无declare用法。

```
postgres=# create or replace procedure p_declare() is
v_int int;
begin
v_int:=1;
raise notice 'v_int = %',v_int;
end ;
/
CREATE PROCEDURE
postgres=# call p_declare();
NOTICE: v_int = 1
CALL
postgres=#
```

调用存储过程不需要call

最近更新时间: 2024-06-12 15:06:00

```
postgres=# create or replace procedure p_call_test() is
begin
raise notice '存储过程调用';
end ;
/
CREATE PROCEDURE
postgres=# create or replace procedure p_call() is
begin
call p_call_test();
p_call_test();
end ;
/
CREATE PROCEDURE
postgres=#
postgres=# call p_call();
NOTICE: 存储过程调用
NOTICE: 存储过程调用
CALL
postgres=#
```

支持存储过程out返回值

最近更新时间: 2024-06-12 15:06:00

```
postgres=# create or replace procedure triple(x out number) is
begin
x:=10;
end ;
/
CREATE PROCEDURE
postgres=# declare
test number:=1;
begin
triple(test);
raise notice 'test = %',test;
end;
/
NOTICE: test = 10
DO
postgres=#
postgres=# call triple(1);
x
----
10
(1 row)
```

oracle语法改写
forall改写

最近更新时间: 2024-06-12 15:06:00

- Oracle语法。

```
create table t5(f1 int,f2 int);
insert into t5 values(1,1);
insert into t5 values(2,2);
commit;

create table t6(f1 int,f2 int);

create or replace procedure p_forall is
TYPE t5list IS TABLE OF t5.f2%TYPE;
t5s t5list;
BEGIN
SELECT f2 BULK COLLECT INTO t5s FROM t5;
FORALL i IN t5s.FIRST .. t5s.LAST
insert into t6 values(t5s(i),t5s(i));
commit;
END;
/
```

- TDSQL-PG改写。

```
create or replace procedure p_forall is
TYPE t5list IS TABLE OF t5.f2%TYPE;
t5s t5list;
BEGIN
SELECT f2 BULK COLLECT INTO t5s FROM t5;
FOR i IN t5s.FIRST .. t5s.LAST
LOOP
insert into t6 values(t5s[i],t5s[i]);
END LOOP;
commit;
END;
/
```

table函数

最近更新时间: 2024-06-12 15:06:00

- Oracle用法。

```
create table t_table(f1 varchar2(36),f2 varchar2(36),f3 varchar2(36));
insert into t_table values('tbase','tbase','tbase');
insert into t_table values('Tbase','Tbase','Tbase');
commit;
create or replace type ty_row as object
(
col1 varchar2(36),
col2 varchar2(36),
col3 varchar2(36)
);
/
create or replace type ty_table as table of ty_row;
create or replace function f_table return ty_table as
v_ty_table ty_table;
begin
select ty_row(f1,f2,f3) bulk collect into v_ty_table from t_table;
return v_ty_table;
end;
/
#调用，使用table返回一个表格
select * from table(f_table());
```

- TDSQL-PG改写。

```
create table t_table(f1 varchar2(36),f2 varchar2(36),f3 varchar2(36));
insert into t_table values('tbase','tbase','tbase');
insert into t_table values('Tbase','Tbase','Tbase');
create or replace function f_table returns setof t_table as
v_rec record;
begin
for v_rec in select * from t_table loop
return next v_rec;
end loop;
return;
end;
/
#调用
select * from f_table();
```

应用程序样例
java
创建数据表

最近更新时间: 2024-06-12 15:06:00

```
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?binaryTransfer=false","tbase", "tbase");
System.out.println("Opened database successfully");
stmt = c.createStatement();
String sql = "create table public.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");
}
}
```

编译

```
javac createtable.java
```

执行

```
java createtable
```

> **注意：**
>
> 连接url中禁止使用currentSchema=xxxx来指定当前搜索路径。

插入数据

最近更新时间: 2024-06-12 15:06:00

```
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?binaryTransfer=false","tbase", "tbase");
c.setAutoCommit(false);
System.out.println("Opened database successfully");
stmt = c.createStatement();
String sql = "INSERT INTO public.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");
}
}
```

扩展协议插入数据

最近更新时间: 2024-06-12 15:06:00

```
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.*;
import java.util.Random;
public class insert_prepared_return {
public static void main(String args[]) {
Connection c = null;
PreparedStatement stmt;
try {
Class.forName("org.postgresql.Driver");
c = DriverManager.getConnection("jdbc:postgresql://172.16.0.23:11345/postgres","adichen", "tbase");
c.setAutoCommit(true);
System.out.println("Opened database successfully");
//插入数据
String sql = "INSERT INTO public.t1 (f1,f2) VALUES (?,?) returning *";
stmt = c.prepareStatement(sql,Statement.RETURN_GENERATED_KEYS);
stmt.setInt(1, 1);
stmt.setString(2, "111");
stmt.executeUpdate();

ResultSet generatedKeys = stmt.getGeneratedKeys();
if (generatedKeys.next()) {
long f1 = generatedKeys.getLong(1);
String f2 = generatedKeys.getString(2);
System.out.println(f1);
System.out.println(f2);
}
generatedKeys.close();
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");
}
}
```

扩展协议插入返回数据

最近更新时间: 2024-06-12 15:06:00

```
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.*;
import java.util.Random;
public class insert_prepared_return {
public static void main(String args[]) {
Connection c = null;
PreparedStatement stmt;
try {
Class.forName("org.postgresql.Driver");
c = DriverManager.getConnection("jdbc:postgresql://172.16.0.23:11345/postgres","adichen", "tbase");
c.setAutoCommit(true);
System.out.println("Opened database successfully");
//插入数据
String sql = "INSERT INTO public.t1 (f1,f2) VALUES (?,?) returning *";
stmt = c.prepareStatement(sql,Statement.RETURN_GENERATED_KEYS);
stmt.setInt(1, 1);
stmt.setString(2, "111");
stmt.executeUpdate();

ResultSet generatedKeys = stmt.getGeneratedKeys();
if (generatedKeys.next()) {
long f1 = generatedKeys.getLong(1);
String f2 = generatedKeys.getString(2);
System.out.println(f1);
System.out.println(f2);
}
generatedKeys.close();
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");
}
}
```

查询数据

最近更新时间: 2024-06-12 15:06:00

```
/*
表及数据
create table select_normal(f1 int not null ,f2 date not null,f2 varchar(10));
insert into select_normal values(1,'2020-01-01','tbase'),(1,'2020-02-01','tbase');
*/
import java.sql.*;

public class select_bind {
public static void main(String args[]) {
Connection c = null;
Statement stmt = null;
try {
Class.forName("org.postgresql.Driver");
c = DriverManager.getConnection("jdbc:postgresql://172.16.0.30:11345/postgres","tbase", "tbase");
c.setAutoCommit(false);
System.out.println("Opened database successfully");
long start_time = System.currentTimeMillis();
PreparedStatement preparedStatement = null;
try {
stmt = c.createStatement();
ResultSet resultSet = stmt.executeQuery("select * from public.select_normal where f2 >= '2020-01-01' and f2< '2020-02-0
1'");
while(resultSet.next()){
System.out.println(resultSet.getString(1));
System.out.println(resultSet.getString(2));
System.out.println(resultSet.getString(3));
}
resultSet.close();
stmt.close();
c.close();
} catch (SQLException e) {
e.printStackTrace();
}
} catch (Exception e) {
System.err.println( e.getClass().getName()+": "+ e.getMessage() );
System.exit(0);
}
System.out.println("Records created successfully");
}
}
```

扩展协议查询数据

最近更新时间: 2024-06-12 15:06:00

```
import java.sql.*;
import java.text.SimpleDateFormat;
public class select_normal {
public static void main(String args[]) {
Connection c = null;
Statement stmt = null;
try {
Class.forName("org.postgresql.Driver");
c = DriverManager.getConnection("jdbc:postgresql://172.16.0.30:11345/postgres","tbase", "tbase");
c.setAutoCommit(false);
System.out.println("Opened database successfully");
long start_time = System.currentTimeMillis();
PreparedStatement preparedStatement = null;
try {
String sql = "select * from public.select_normal where f2 >= ? and f2 < ? " ;
preparedStatement = c.prepareStatement(sql);
String deskdate_start = "2020-01-01";
String deskdate_end = "2020-02-01";
Date deskdate_start_date = new java.sql.Date(new SimpleDateFormat("yyyy-MM-dd").parse(deskdate_start).getTime());
Date deskdate_end_date = new java.sql.Date(new SimpleDateFormat("yyyy-MM-dd").parse(deskdate_end).getTime());
preparedStatement.setDate(1,deskdate_start_date);
preparedStatement.setDate(2,deskdate_end_date);
ResultSet resultSet = preparedStatement.executeQuery();
System.out.println("执行查询耗时："+(System.currentTimeMillis()-start_time)+"ms");
while(resultSet.next()){
System.out.println(resultSet.getString(1));
System.out.println(resultSet.getString(2));
System.out.println(resultSet.getString(3));
}

} catch (SQLException e) {
e.printStackTrace();
}

} catch (Exception e) {
System.err.println( e.getClass().getName()+": "+ e.getMessage() );
System.exit(0);
}
System.out.println("Records created successfully");
}
}
```

copy from 入库

最近更新时间: 2024-06-12 15:06:00

```
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?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 public.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");
}
}
```

copy from 数据流入库

最近更新时间: 2024-06-12 15:06:00

```
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 copy_from_buff {
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://127.0.0.1:11379/postgres","tbase", "tbase");
System.out.println("Opened database successfully");
CopyManager cm = new CopyManager((BaseConnection) c);
String delimiter = new String(";");
StringBuilder buf = new StringBuilder();
//create table t_buff(f1 int,f2 varchar(10));
for (int i=0; i<10000; i++){
buf.append(i + delimiter+ "tbase" + "\r\n");
}
StringReader reader = new StringReader(buf.toString());
cm.copyIn("COPY public.t_buff FROM STDIN WITH DELIMITER ';'", reader);
c.close();
} catch ( Exception e ) {
System.err.println( e.getClass().getName()+": "+ e.getMessage() );
System.exit(0);
}
System.out.println("Copy data successfully");
}
}
```

copy to出库

最近更新时间: 2024-06-12 15:06:00

```
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?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 public.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");
}
}
```

兼容oracle字段大写

最近更新时间: 2024-06-12 15:06:00

```
import java.sql.*;
public class oracle_test {
public static void main(String args[]) {
Connection c = null;
Statement stmt = null;
try {
Class.forName("org.postgresql.Driver");
c = DriverManager.getConnection("jdbc:postgresql://172.16.0.30:11345/postgres?oracle_compile=true","tbase", "tbase");
c.setAutoCommit(false);
System.out.println("Opened database successfully");

PreparedStatement preparedStatement = null;
try {
preparedStatement = c.prepareStatement("select * from public.select_normal");
ResultSetMetaData metaData = preparedStatement.getMetaData();
for (int i = 0; i < metaData.getColumnCount(); i++) {
// resultSet数据下标从1开始
String columnName = metaData.getColumnName(i + 1);
int type = metaData.getColumnType(i + 1);
System.out.println("列名："+columnName);
}
} catch (SQLException e) {
e.printStackTrace();
}
} catch (Exception e) {
System.err.println( e.getClass().getName()+": "+ e.getMessage() );
System.exit(0);
}
System.out.println("Records created successfully");
}
}
```

配置多个cn负载

最近更新时间: 2024-06-12 15:06:00

```
import java.sql.*;
public class select_bind {
public static void main(String args[]) {
Connection c = null;
Statement stmt = null;
try {
Class.forName("org.postgresql.Driver");
c = DriverManager.getConnection("jdbc:postgresql://172.16.0.61:11345,172.16.0.30:11347/postgres?loadBalanceHosts=tru
e","tbase", "tbase");
c.setAutoCommit(false);
System.out.println("Opened database successfully");
long start_time = System.currentTimeMillis();
PreparedStatement preparedStatement = null;
try {
stmt = c.createStatement();
ResultSet resultSet = stmt.executeQuery("select * from public.select_normal where f2 >= '2020-01-01' and f2< '2020-02-0
1'");
while(resultSet.next()){
System.out.println(resultSet.getString(1));
System.out.println(resultSet.getString(2));
System.out.println(resultSet.getString(3));
}
resultSet.close();
stmt.close();
c.close();
} catch (SQLException e) {
e.printStackTrace();
}
} catch (Exception e) {
System.err.println( e.getClass().getName()+": "+ e.getMessage() );
System.exit(0);
}
System.out.println("Records created successfully");
}
}
```

合并多条insert

最近更新时间: 2024-06-12 15:06:00

```
import java.sql.*;
import java.util.Random;
import java.util.UUID;
public class batch {
public static void main(String args[]) {
Connection conn = null;
Statement stmt = null;
try {
Class.forName("org.postgresql.Driver");
conn = DriverManager.getConnection("jdbc:postgresql://172.16.0.30:11345/postgres?reWriteBatchedInserts=true","tbase",
"tbase");
conn.setAutoCommit(true);
System.out.println("Opened database successfully");
//开始执行批量插入
String insert_sql = "INSERT INTO t1 VALUES (?,?)";
try {
assert conn != null;
PreparedStatement preparedStatement = conn.prepareStatement(insert_sql);
for (int i = 0; i < 100000; i++) {
preparedStatement.setInt(1, new Random().nextInt(1000));
preparedStatement.setString(2, UUID.randomUUID().toString());
preparedStatement.addBatch();
if (i % 500 == 0) {
preparedStatement.executeBatch();
preparedStatement.clearBatch();
}
}
preparedStatement.executeBatch();
preparedStatement.clearBatch();
} catch (SQLException e) {
e.printStackTrace();
}

} catch (Exception e) {
System.err.println( e.getClass().getName()+": "+ e.getMessage() );
System.exit(0);
}
System.out.println("Records created successfully");
}
}
```

jdbc驱动包

最近更新时间: 2024-06-12 15:06:00

参见JDBC驱动包下载

C程序

连接服务

最近更新时间: 2024-06-12 15:06:00

```
#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！
```

建立数据表

最近更新时间: 2024-06-12 15:06:00

```
#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！
```

插入数据

最近更新时间: 2024-06-12 15:06:00

```
#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！
```

查询数据

最近更新时间: 2024-06-12 15:06:00

```
#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
```

copy入库

最近更新时间: 2024-06-12 15:06:00

```
#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
```

运行

```
./copy "dbname=postgres port=15432"
#连接数据库成功！
copy数据成功！
```

shell程序

最近更新时间: 2024-06-12 15:06:00

```
#!/bin/sh
if [ $# -ne 0 ]
then
echo "usage: $0 exec_sql"
exit 1
fi
exec_sql=$1
masters=`psql -h 172.16.0.29 -d postgres -p 15432 -t -c "select string_agg(node_host, ' ') from (select * from pgxc_node w
here node_type = 'D' order by node_name) t"`
port_list=`psql -h 172.16.0.29 -d postgres -p 15432 -t -c "select string_agg(node_port::text, ' ') from (select * from pgxc_no
de where node_type = 'D' order by node_name) t"`
node_cnt=`psql -h 172.16.0.29 -d postgres -p 15432 -t -c "select count(*) from pgxc_node where node_type = 'D'"`
masters=($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
```

python程序
安装psycopg2模块

最近更新时间: 2024-06-12 15:06:00

```
[root@VM_0_29_centos ~]# yum install python-psycopg2
```

连接服务

最近更新时间: 2024-06-12 15:06:00

```
#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
```

创建数据表

最近更新时间: 2024-06-12 15:06:00

```
#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
```

新增数据

最近更新时间: 2024-06-12 15:06:00

```
#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
```

查询数据

最近更新时间: 2024-06-12 15:06:00

```
#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
```

copy from方法

最近更新时间: 2024-06-12 15:06:00

```
#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])
```

运行

```
[tbase@VM_0_29_centos python]$ python copy_from.py
#连接数据库成功
#导入数据成功
```

php程序
连接服务

最近更新时间: 2024-06-12 15:06:00

```
<?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：".$error_msg."\n
exit;
}else{
echo "连接数据库成功"."\n
}
//关闭连接
pg_close($conn);
?>
```

执行

```
[root@VM_0_47_centos test]# curl http://imgcache.finance.cloud.tencent.com:80127.0.0.1:8080/dbsta/test/conn.php
#连接数据库成功
```

创建数据表

最近更新时间: 2024-06-12 15:06:00

```
<?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://imgcache.finance.cloud.tencent.com:80127.0.0.1:8080/dbsta/test/createtable.ph
p
#连接数据库成功
#创建数据表成功
```

插入数据

最近更新时间: 2024-06-12 15:06:00

```
<?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://imgcache.finance.cloud.tencent.com:80127.0.0.1:8080/dbsta/test/insert.php
#连接数据库成功
#插入数据成功
```

查询记录

最近更新时间: 2024-06-12 15:06:00

```
<?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://imgcache.finance.cloud.tencent.com:80127.0.0.1:8080/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
```

copy from 方法

最近更新时间: 2024-06-12 15:06:00

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://imgcache.finance.cloud.tencent.com:80127.0.0.1/dbsta/cron/php_copy_from.php
#连接数据库成功
#copy成功
```

copy to 方法

最近更新时间: 2024-06-12 15:06:00

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://imgcache.finance.cloud.tencent.com:80127.0.0.1/dbsta/cron/php_copy_to.php
#连接数据库成功
Array
(
[0] => 1,TBase
[1] => 2,pgxz
)
```

入库去重方法

最近更新时间: 2024-06-12 15:06:00

```
<?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
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
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
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 m
pid, 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=".$tmp_table_name.".mpid AND myd
ata.datatype=".$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); ;
?>
```

golang程序
连接服务

最近更新时间: 2024-06-12 15:06:00

```
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
```

建立数据表

最近更新时间: 2024-06-12 15:06:00

```
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
```

插入数据

最近更新时间: 2024-06-12 15:06:00

```
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
```

查询数据

最近更新时间: 2024-06-12 15:06:00

```
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

copy from 方法

最近更新时间: 2024-06-12 15:06:00

```
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
```

go相关资源包

最近更新时间: 2024-06-12 15:06:00

需要git的资源包： jackc/pgx pkg/errors

关于运维
数据库管理
创建数据库

最近更新时间: 2024-06-12 15:06:00

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TDSQL PG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)
```

- 创建gbk编码。

```
postgres=# CREATE DATABASE db_gbk template template0 encoding = gbk LC_COLLATE = 'zh_CN.gbk' LC_CTYPE = 'zh
_CN.gbk';
CREATE DATABASE
postgres=# \l+
List of databases
Name | Owner | Encoding| Collate | Ctype | Access privileges |Size | Tablespace | Description
-----------+-------+----------+------------+------------+-------------------+-------+------------+-------------------------
----------------
db_gbk | tbase | GBK | zh_CN.gbk | zh_CN.gbk | | 19 MB | pg_default |
postgres | tbase | UTF8 | zh_CN.utf8 | zh_CN.utf8 | | 26 MB | pg_default | default administrative connection database
template0 | tbase | UTF8 | zh_CN.utf8 | zh_CN.utf8 | =c/tbase +| 19 MB | pg_default | unmodifiable empty database
| | | | | tbase=CTc/tbase | | |
template1 | tbase | UTF8 | zh_CN.utf8 | zh_CN.utf8 | =c/tbase +| 24 MB | pg_default | default template for new databases
| | | | | tbase=CTc/tbase | | |
(4 rows)
```

- 创建gb18030编码。

```
postgres=# create database db_gb18030 template template0 encoding=gb18030 LC_COLLATE = 'zh_CN.gb18030' LC_C
TYPE = 'zh_CN.gb18030';
CREATE DATABASE
postgres=# \l
List of databases
Name | Owner | Encoding | Collate | Ctype | Access privileges
------------+-------+----------+---------------+---------------+-------------------
db_gb18030 | tbase | GB18030 | zh_CN.gb18030 | zh_CN.gb18030 |
db_gbk | tbase | GBK | zh_CN.gbk | zh_CN.gbk |
postgres | tbase | UTF8 | zh_CN.utf8 | zh_CN.utf8 |
template0 | tbase | UTF8 | zh_CN.utf8 | zh_CN.utf8 | =c/tbase +
| | | | | tbase=CTc/tbase
template1 | tbase | UTF8 | zh_CN.utf8 | zh_CN.utf8 | =c/tbase +
| | | | | tbase=CTc/tbase
test | tbase | UTF8 | zh_CN.utf8 | zh_CN.utf8 |
(6 rows)

postgres=#
```

- 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
```

- 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_c
ollate 'C';
CREATE DATABASE
```

修改数据库配置

最近更新时间: 2024-06-12 15:06:00

- 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
```

- Alter database不支持的项目。

  不支持项目

项目	备注
encoding	编码
lc_collate	排序规则
lc_ctype	分组规则

删除数据库

最近更新时间: 2024-06-12 15:06:00

```
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
```

会话及锁管理
查看当前会话的PID

最近更新时间: 2024-06-12 15:06:00

```
postgres=# Select pg_backend_pid();
```

查看当前节点有那些会话

最近更新时间: 2024-06-12 15:06:00

```
select * from pg_stat_activity;
```

- 查看活跃超时会话。

```
select pid,client_addr,state_change,query_start,state_change,EXTRACT(EPOCH FROM(now()-query_start)),query,state,use
name,application_name from pg_stat_activity where EXTRACT(EPOCH FROM (now()-query_start))>60 and state!='idle';
```

- 按用户统计活跃连接数。

```
select count(1),usename from pg_stat_activity where EXTRACT(EPOCH FROM (now()-query_start))>60 and state!='idle'
group by usename;
```

- 按客户端统计活跃连接数。

```
select count(1),client_addr from pg_stat_activity where EXTRACT(EPOCH FROM (now()-query_start))>60 and state!='idl
e' group by client_addr;
```

杀掉连接

最近更新时间: 2024-06-12 15:06:00

```
postgres=# select pg_terminate_backend(pid);
```

⚠pid为进程号

查看会话持锁情况

最近更新时间: 2024-06-12 15:06:00

- 排它锁阻塞。

```
postgres=# select pid,pg_blocking_pids(pid),EXTRACT(EPOCH FROM(now()-query_start)),wait_event_type,query from pg_stat_activity where wait_event_type = 'Lock' and pid!=pg_backend_pid();
```

- 同步阻塞。

```
postgres=# select query,wait_event_type,wait_event from pg_stat_activity where wait_event='SyncRep';
```

配置会话锁超时

最近更新时间: 2024-06-12 15:06:00

- 当前会话。

```
postgres=# set lock_timeout to '3s';
```

- 数据库默认锁超时。

```
postgres=# alter database postgres set lock_timeout to '3s';
```

- 用户默认锁超时。

```
postgres=# alter role tbase set lock_timeout to '3s';
```

问题定位及性能优化
访问日志管理
配置日志

最近更新时间: 2024-06-12 15:06:00

TDSQL PG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
```

日志格式说明

最近更新时间: 2024-06-12 15:06:00

执行正确的语句产生的日志。

```
2017-10-11 16:23:55.178 CST,"pgxz","postgres",11499,"127.0.0.1:2329",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"
```

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。

对日志进行分析
创建日志表

最近更新时间: 2024-06-12 15:06:00

```
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
);
```

导入日志数据

最近更新时间: 2024-06-12 15:06:00

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
```

统计日志数据

最近更新时间: 2024-06-12 15:06:00

- 按照session连接及操作时间排序

```
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,datab
ase_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
(10 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_na
me,database_name order by count(1) desc limit 10;
count | user_name | database_name
-------+-----------+---------------
1390 | pgxz | postgres
(1 row)
```

配置只收集慢的sql语句

最近更新时间: 2024-06-12 15:06:00

```
#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,2/91749387678543872,0,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。

如何查询数据是否倾斜

最近更新时间: 2024-06-12 15:06:00

连接上不同的 dn 节点，分别查询数据表的容量大小，如果大小偏差较大就可以判断存在数据倾斜。

```
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)
```

如何优化有问题的Sql语句
查看是否为分布键查询

最近更新时间: 2024-06-12 15:06:00

```
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。

查看是否使用上索引

最近更新时间: 2024-06-12 15:06:00

```
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。

查看是否为分布key join

最近更新时间: 2024-06-12 15:06:00

```
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。

查看join发生的节点

最近更新时间: 2024-06-12 15:06:00

```
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。

查看并行的worker数

最近更新时间: 2024-06-12 15:06:00

```
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。

检查各个节点的执行计划是否一致

最近更新时间: 2024-06-12 15:06:00

```
./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，可以配置在系统空闲时执行全库analyze和vacuum。

优化实例
count(distinct xx)优化

最近更新时间: 2024-06-12 15:06:00

```
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 t
ext,f12 text) distribute by shard(f1);
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;
postgres=# analyze t1;
ANALYZE
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 ro
ws=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 ro
ws=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。

增大work_mem减少io访问

最近更新时间: 2024-06-12 15:06:00

```
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 t
ext,f12 text) distribute by shard(f1);

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 t
ext,f12 text) distribute by shard(f1);
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=# 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=5000
0 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=5000
0 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。

not in改写为anti join

最近更新时间: 2024-06-12 15:06:00

增大work_mem减少io访问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 wi
dth=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=100
0 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=5
0000 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 wi
dth=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=100
0 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=50
000 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=#
```

分布key jon+limit优化

最近更新时间: 2024-06-12 15:06:00

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 t
ext,f12 text) distribute by shard(f1);
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 t
ext,f12 text) distribute by shard(f1);
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=1
000000 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=1
000000 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。

非分布key join使用hash join性能一般最好

最近更新时间: 2024-06-12 15:06:00

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 tex
t) distribute by shard(f1);
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 tex
t) distribute by shard(f1);
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::t
ext),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::t
ext),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..6555.34
9 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..5608.34
9 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..10
83.699 rows=10 loops=1)
Planning time: 0.530 ms
Execution time: 1108.830 ms
(4 rows)
Time: 1117.713 ms (00:01.118)
mydb=#
```

exists的优化

最近更新时间: 2024-06-12 15:06:00

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。

重新聚簇表

最近更新时间: 2024-06-12 15:06:00

重新聚簇表可以减少扫描走某个索引值扫描表的页数。

```
psql -h 172.16.0.61 -p 11002 -d postgres -U tbase -c "explain (analyze,buffers) select count(1) from t1 where f2=1;"
QUERY PLAN
------------------------------------------------------------------------------------------------------------------------------------------------
Aggregate (cost=7208.99..7209.00 rows=1 width=8) (actual time=25.505..25.505 rows=1 loops=1)
Buffers: shared hit=5869
-> Bitmap Heap Scan on t1 (cost=124.87..7185.62 rows=9348 width=0) (actual time=3.567..23.002 rows=10051 loops=1)
Recheck Cond: (f2 = 1)
Heap Blocks: exact=5838
Buffers: shared hit=5869
-> Bitmap Index Scan on t1_f2_idx (cost=0.00..122.53 rows=9348 width=0) (actual time=2.405..2.405 rows=10051 loops=
1)
Index Cond: (f2 = 1)
Buffers: shared hit=31
Planning time: 0.626 ms
Execution time: 25.659 ms
(11 rows)
postgres=# CLUSTER t1 USING t1_f2_idx;
CLUSTER
postgres=#
dn001 --- psql -h 172.16.0.61 -p 11002 -d postgres -U tbase -c "explain (analyze,buffers) select count(1) from t1 where f2
=1;"
QUERY PLAN
------------------------------------------------------------------------------------------------------------------------------------------------
Aggregate (cost=7201.23..7201.24 rows=1 width=8) (actual time=9.808..9.808 rows=1 loops=1)
Buffers: shared hit=116
-> Bitmap Heap Scan on t1 (cost=124.87..7177.86 rows=9348 width=0) (actual time=1.312..7.348 rows=10051 loops=1)
Recheck Cond: (f2 = 1)
Heap Blocks: exact=85
Buffers: shared hit=116
-> Bitmap Index Scan on t1_f2_idx (cost=0.00..122.53 rows=9348 width=0) (actual time=1.219..1.219 rows=10051 loops=
1)
Index Cond: (f2 = 1)
Buffers: shared hit=31
Planning time: 0.696 ms
Execution time: 9.969 ms
(11 rows)
```

聚簇前要扫描的block数为5869，聚簇后为116，性能提高2倍。

jdbc访问数据库优化

最近更新时间: 2024-06-12 15:06:00

- 正确的数据类型绑定。

```
import java.sql.*;
import java.text.SimpleDateFormat;
public class select_normal {
public static void main(String args[]) {
Connection c = null;
Statement stmt = null;
try {
Class.forName("org.postgresql.Driver");
c = DriverManager.getConnection("jdbc:postgresql://172.16.0.30:11345/postgres","tbase", "tbase");
c.setAutoCommit(false);
System.out.println("Opened database successfully");
long start_time = System.currentTimeMillis();
PreparedStatement preparedStatement = null;
try {
String sql = "select * from public.select_normal where f2 >= ? and f2 < ? " ;
preparedStatement = c.prepareStatement(sql);
String deskdate_start = "2020-01-01";
String deskdate_end = "2020-02-01";
Date deskdate_start_date = new java.sql.Date(new SimpleDateFormat("yyyy-MM-dd").parse(deskdate_start).getTime());
Date deskdate_end_date = new java.sql.Date(new SimpleDateFormat("yyyy-MM-dd").parse(deskdate_end).getTime());
preparedStatement.setDate(1,deskdate_start_date);
preparedStatement.setDate(2,deskdate_end_date);
ResultSet resultSet = preparedStatement.executeQuery();
System.out.println("执行查询耗时："+(System.currentTimeMillis()-start_time)+"ms");
while(resultSet.next()){
System.out.println(resultSet.getString(1));
System.out.println(resultSet.getString(2));
System.out.println(resultSet.getString(3));
}
} catch (SQLException e) {
e.printStackTrace();
}
} catch (Exception e) {
System.err.println( e.getClass().getName()+": "+ e.getMessage() );
System.exit(0);
}
System.out.println("Records created successfully");
}
}
```

- 错误的数据类型绑定。

```
import java.sql.*;
import java.text.SimpleDateFormat;
public class select_normal {
public static void main(String args[]) {
Connection c = null;
Statement stmt = null;
```

```
try {
Class.forName("org.postgresql.Driver");
c = DriverManager.getConnection("jdbc:postgresql://172.16.0.30:11345/postgres","tbase", "tbase");
c.setAutoCommit(false);
System.out.println("Opened database successfully");
long start_time = System.currentTimeMillis();
PreparedStatement preparedStatement = null;
try {
String sql = "select * from public.select_normal where f2 >= ? and f2 < ? " ;
preparedStatement = c.prepareStatement(sql);
String deskdate_start = "2020-01-01";
String deskdate_end = "2020-02-01";
preparedStatement.setDate(1,deskdate_start);
preparedStatement.setDate(2,deskdate_end);
ResultSet resultSet = preparedStatement.executeQuery();
System.out.println("执行查询耗时："+(System.currentTimeMillis()-start_time)+"ms");
while(resultSet.next()){
System.out.println(resultSet.getString(1));
System.out.println(resultSet.getString(2));
System.out.println(resultSet.getString(3));
}
} catch (SQLException e) {
e.printStackTrace();
}
} catch (Exception e) {
System.err.println( e.getClass().getName()+": "+ e.getMessage() );
System.exit(0);
}
System.out.println("Records created successfully");
}
}
```

日期类型如果使用String类型绑定变量，则相当

```
select * from select_normal from f2::text>'2020-01-10'::text
```

这样导致查询无法走索引。

分区表now()剪枝问题

最近更新时间: 2024-06-12 15:06:00

```
postgres=# create table t_time_range
(f1 bigint, f2 timestamp ,f3 bigint)
partition by range (f2) begin (timestamp without time zone '2021-06-01 0:0:0')
step (interval '1 month')
partitions (12) distribute by shard(f1)
to group default_group;
CREATE TABLE
```

- 使用原生的now()函数是剪不了枝。

```
postgres=# explain select * from t_time_range where f2<now();
QUERY PLAN
------------------------------------------------------------------------------------------------------------------------
Remote Fast Query Execution (cost=0.00..0.00 rows=0 width=0)
Node/s: dn001, dn002, dn003
-> Append (cost=0.00..0.00 rows=0 width=0)
-> Seq Scan on t_time_range (partition sequence: 0, name: t_time_range_part_0) (cost=0.00..2.17 rows=30 width=24)
Filter: (f2 < now())
-> Seq Scan on t_time_range (partition sequence: 1, name: t_time_range_part_1) (cost=0.00..2.17 rows=30 width=24)
Filter: (f2 < now())
-> Seq Scan on t_time_range (partition sequence: 2, name: t_time_range_part_2) (cost=0.00..2.17 rows=30 width=24)
Filter: (f2 < now())
-> Seq Scan on t_time_range (partition sequence: 3, name: t_time_range_part_3) (cost=0.00..2.17 rows=30 width=24)
Filter: (f2 < now())
-> Seq Scan on t_time_range (partition sequence: 4, name: t_time_range_part_4) (cost=0.00..2.17 rows=30 width=24)
Filter: (f2 < now())
-> Seq Scan on t_time_range (partition sequence: 5, name: t_time_range_part_5) (cost=0.00..2.17 rows=30 width=24)
Filter: (f2 < now())
-> Seq Scan on t_time_range (partition sequence: 6, name: t_time_range_part_6) (cost=0.00..2.17 rows=30 width=24)
Filter: (f2 < now())
-> Seq Scan on t_time_range (partition sequence: 7, name: t_time_range_part_7) (cost=0.00..2.17 rows=30 width=24)
Filter: (f2 < now())
-> Seq Scan on t_time_range (partition sequence: 8, name: t_time_range_part_8) (cost=0.00..2.17 rows=30 width=24)
Filter: (f2 < now())
-> Seq Scan on t_time_range (partition sequence: 9, name: t_time_range_part_9) (cost=0.00..2.17 rows=30 width=24)
Filter: (f2 < now())
-> Seq Scan on t_time_range (partition sequence: 10, name: t_time_range_part_10) (cost=0.00..2.17 rows=30 width=24)
Filter: (f2 < now())
-> Seq Scan on t_time_range (partition sequence: 11, name: t_time_range_part_11) (cost=0.00..2.17 rows=30 width=24)
Filter: (f2 < now())
(27 rows)
```

- 定义一个稳定函数。

```
postgres=# create or replace function get_current_timestamp() returns timestamp as
$$
begin
return current_timestamp;
```

```
end;
$$
language plpgsql IMMUTABLE;
CREATE FUNCTION
postgres=# explain select * from t_time_range where f2<get_current_timestamp();
QUERY PLAN
------------------------------------------------------------------------------------------------------------------------------------------
Remote Fast Query Execution (cost=0.00..0.00 rows=0 width=0)
Node/s: dn001, dn002, dn003
-> Append (cost=0.00..0.00 rows=0 width=0)
-> Seq Scan on t_time_range (partition sequence: 0, name: t_time_range_part_0) (cost=0.00..11.69 rows=178 width=24)
Filter: (f2 < '2021-07-12 11:31:33.639119'::timestamp without time zone)
-> Seq Scan on t_time_range (partition sequence: 1, name: t_time_range_part_1) (cost=0.00..11.69 rows=178 width=24)
Filter: (f2 < '2021-07-12 11:31:33.639119'::timestamp without time zone)
(7 rows)
postgres=#
```

这样优化后剪枝就正常了。

使用TDSQL PG自研分区表
插入数据性能测试对比
自研分区表

最近更新时间: 2024-06-12 15:06:00

```
#创建测试表2048个子表
create table t_time_range
(
f1 bigint not null, f2 timestamp ,f3 bigint
)
partition by range (f2) begin (timestamp without time zone '2017-09-01 0:0:0')
step (interval '1 month')
partitions (2048) distribute by shard(f1)
to group default_group;

#pgbench sql脚本

[tbase@eg-9-117-182-250 pgbench]$ cat insert_t_time_range.sql
\set f1 random(1, 10000000)
\set f2 random(1, 50000)
insert into t_time_range values(:f1,('2017-09-01'::date+:f2::integer)::timestamp,:f1);

#测试方法

[tbase@eg-9-117-182-250 pgbench]$ pgbench -h 9.117.183.12 -p 15432 -d pgbench -U tbase -c 64 -j 1 -n -M prepared -
T 60 -r -f insert_t_time_range.sql > insert_t_time_range.log 2>&1
#测试结果
scaling factor: 1
query mode: prepared
number of clients: 64
number of threads: 1
duration: 60 s
number of transactions actually processed: 1039672
latency average = 3.695 ms
tps = 17319.573296 (including connections establishing)
tps = 17320.178973 (excluding connections establishing)
script statistics:
- statement latencies in milliseconds:
0.007 \set f1 random(1, 10000000)
0.004 \set f2 random(1, 50000)
3.673 insert into t_time_range values(:f1,('2017-09-01'::date+:f2::integer)::timestamp,:f1);
```

PG社区分区表

最近更新时间: 2024-06-12 15:06:00

```
#创建测试表2048个子表
[tbase@TENCENT64 shell]$ psql -d pgbench
psql (PostgreSQL 10.0 TBase V2)
Type "help" for help.
pgbench=#create table t_native_range(f1 bigint,f2 timestamp default now(),f3 integer) partition by range ( f2 );
\q
[tbase@TENCENT64 shell]$ psql -At -c "select 'create table t_native_range_'||num::text||' partition of t_native_range(f1,f2,f
3) for values from ('''||startyf||''') to ('''||endyf||'''); ' from (select t::text as num, ('2017-09-01'::date + (t::text||' months')::inter
val)::date::text as startyf,('2017-10-01'::date + (t::text||' months')::interval)::date::text as endyf from generate_series(0,2047)
as t) as t;"|psql -d pgbench
#pgbench sql脚本
[tbase@eg-9-117-182-250 pgbench]$ cat insert_t_native_range.sql
\set f1 random(1, 10000000)
\set f2 random(1, 50000)
insert into t_native_range values(:f1,('2017-09-01'::date+:f2::integer)::timestamp,:f1);
#测试方法
[tbase@eg-9-117-182-250 pgbench]$ pgbench -h 9.117.183.12 -p 15432 -d pgbench -U tbase -c 64 -j 1 -n -M prepared -
T 60 -r -f insert_t_native_range.sql > insert_t_native_range.log 2>&1
#测试结果
scaling factor: 1
query mode: prepared
number of clients: 64
number of threads: 1
duration: 60 s
number of transactions actually processed: 781613
latency average = 4.913 ms
tps = 13025.700700 (including connections establishing)
tps = 13026.161416 (excluding connections establishing)
script statistics:
- statement latencies in milliseconds:
0.007 \set f1 random(1, 10000000)
0.004 \set f2 random(1, 50000)
4.850 insert into t_native_range values(:f1,('2017-09-01'::date+:f2::integer)::timestamp,:f1);
```

查询数据性能测试对比
自研分区表

最近更新时间: 2024-06-12 15:06:00

```
#创建测试表2048个子表
drop table t_time_range;
create table t_time_range
(
f1 bigint not null, f2 timestamp ,f3 bigint
)
partition by range (f2) begin (timestamp without time zone '2017-09-01 0:0:0')
step (interval '1 month')
partitions (2048) distribute by shard(f1)
to group default_group;
#每个分区插入一条数据
psql -At -c "select 'insert into t_time_range values(1,'''||startyf||''',1);' from (select ('2017-09-01'::date + (t::text||' months')::in
terval)::date::text as startyf from generate_series(0,2047) as t) as t"|psql -d pgbench
#编写测试存储过程
create or replace function get_t_time_range() returns t_time_range as
$$
declare
v_start timestamp;
v_end timestamp;
v_random integer;
v_rec t_time_range%rowtype;
begin
v_random:=(random()*50000)::integer;
v_start:=(to_char('2017-09-01'::date+v_random,'YYYY-MM')||'-01')::timestamp;
v_end:=(to_char('2017-09-01'::date+v_random,'YYYY-MM')||'-01')::DATE+'1 months'::interval;
select * into v_rec from t_time_range where f2>=v_start and f2<v_end;
return v_rec;
end;
$$
LANGUAGE plpgsql;
#pgbench sql脚本
[tbase@eg-9-117-182-250 pgbench]$ cat select_t_time_range.sql
select * from get_t_time_range();
#pgbench方法
pgbench -h 9.117.183.12 -p 15432 -d pgbench -U tbase -c 64 -j 1 -n -M prepared -T 60 -r -f select_t_time_range.sql > sele
ct_t_time_range.log 2>&1
#测试结果
scaling factor: 1
query mode: prepared
number of clients: 64
number of threads: 1
duration: 60 s
number of transactions actually processed: 1002843
latency average = 3.829 ms
tps = 16712.957914 (including connections establishing)
tps = 16713.547319 (excluding connections establishing)
script statistics:
- statement latencies in milliseconds:
3.813 select * from get_t_time_range();
```

PG社区分区表

最近更新时间: 2024-06-12 15:06:00

```
#创建测试表2048个子表
[tbase@TENCENT64 shell]$ psql -d pgbench
psql (PostgreSQL 10.0 TBase V2)
Type "help" for help.
pgbench=#drop table t_native_range; create table t_native_range(f1 bigint,f2 timestamp default now(),f3 integer) partition by range ( f2 );
\q
[tbase@TENCENT64 shell]$ psql -At -c "select 'create table t_native_range_'||num::text||' partition of t_native_range(f1,f2,f3) for values from ('''||startyf||''') to ('''||endyf||'''); ' from (select t::text as num, ('2017-09-01'::date + (t::text||' months')::interval)::date::text as startyf,('2017-10-01'::date + (t::text||' months')::interval)::date::text as endyf from generate_series(0,2047) as t) as t;"|psql -d pgbench
#每个分区插入一条数据
psql -At -c "select 'insert into t_native_range values(1,'''||startyf||''',1);' from (select ('2017-09-01'::date + (t::text||' months')::interval)::date::text as startyf from generate_series(0,2047) as t) as t"|psql -d pgbench
#编写测试存储过程
create or replace function get_t_native_range() returns t_native_range as
$$
declare
v_start timestamp;
v_end timestamp;
v_random integer;
v_rec t_native_range%rowtype;
begin
v_random:=(random()*50000)::integer;
v_start:=(to_char('2017-09-01'::date+v_random,'YYYY-MM')||'-01')::timestamp;
v_end:=(to_char('2017-09-01'::date+v_random,'YYYY-MM')||'-01')::DATE+'1 months'::interval;
select * into v_rec from t_native_range where f2>=v_start and f2<v_end;
return v_rec;
end;
$$
LANGUAGE plpgsql;
#pgbench sql脚本
[tbase@eg-9-117-182-250 pgbench]$ cat select_t_native_range.sql
select * from get_t_native_range();
#pgbench方法
pgbench -h 9.117.183.12 -p 15432 -d pgbench -U tbase -c 64 -j 1 -n -M prepared -T 60 -r -f select_t_native_range.sql > select_t_native_range.log 2>&1
#测试结果
scaling factor: 1
query mode: prepared
number of clients: 64
number of threads: 1
duration: 60 s
number of transactions actually processed: 35978
latency average = 106.832 ms
tps = 599.068955 (including connections establishing)
tps = 599.090129 (excluding connections establishing)
script statistics:
- statement latencies in milliseconds:
105.706 select * from get_t_native_range();
```

数据库开发规范
分布键设计规范
分布键约束规则

最近更新时间: 2024-06-12 15:06:00

- 分布键字段无法更新，更新分布键需要先删除记录，再插入新的记录。
- 分布键字段类型不能修改。
- 分布键字段的长度不能修改。
- 分布键数据类型只能支持char,varchar,varchar2,text,date,timestamp,int,bigint,float8,number,numberic。
- 分布键只能选择一个字段。

分布键选择规范

最近更新时间: 2024-06-12 15:06:00

- 分布键关系到数据分布是否均衡，原则就是不能由于各个分布键值数据不均产生数据倾斜，出现木桶效应。
- 如果有主键，则选择主键做分布键。
- 如果是复合主键，则可选择数据重复率低的字段来做分布键。
- 没有主键的可以使用JAVA生成UUID来做分布键。
- 如果涉及数据表JOIN,则选择JOIN字段来做分布键。
- 也可以按业务类型，地区或者分公司来做分布键。

分布键对其它约束影响

最近更新时间: 2024-06-12 15:06:00

- 主键必需包含分布键字段。
- 唯一索引必需包含分布键字段。
- 外键必需是分布键字段。

命名规范

最近更新时间: 2024-06-12 15:06:00

1. DB object: database, schema, table, column, view, index, sequence,function, trigger等名称：

- 建议使用小写字母、数字、下划线的组合。
- 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。
- 长度不能超过63个字符。
- 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。
- 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。

COLUMN设计

最近更新时间: 2024-06-12 15:06:00

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。

Constraints 设计

最近更新时间: 2024-06-12 15:06:00

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。

> **注意：**
>
> 非shard key不可以建立primary key或者 unique index。

Index 设计

最近更新时间: 2024-06-12 15:06:00

1. TDSQL PG提供的index类型: B-tree, Hash, GiST (GeneralizedSearch Tree), SP-GiST (space-partitioned GiST), GIN (Generalized InvertedIndex), 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 , dropindex 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。

关于NULL

最近更新时间: 2024-06-12 15:06:00

1. NULL 的判断：IS NULL ，IS NOT NULL。

> **注意：**
>
> boolean 类型取值 true，false， NULL；

2. 小心NOT IN 集合中带有NULL元素。

```
postgres=# select * from tbase;
id | nickname
----+---------------
1 | hello TDSQL PG
2 | TDSQL PG好
3 | TDSQL PG好
4 | TDSQL PG default
(4 rows)

postgres=# select * from tbase where id not in (null);
id | nickname
----+----------
(0 rows)
```

3. 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 TDSQL PG
(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 TDSQL PG
(1 row)
```

4. 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 TDSQL PG
2 | TDSQL PG好
5 |
3 | TDSQL PG好
4 | TDSQL PG 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)
```

5. 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)
```

6. 两个null的对比方法。

```
postgres=# select null is not distinct from null as TBasenull;
TBasenull
-----------
t
(1 row)
```

开发相关规范

最近更新时间: 2024-06-12 15:06:00

- 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
```

- 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。

- 建议update 时尽量做

```
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 | TDSQL PG
(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 | TDSQL PG
(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。

- 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。

```
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：

- 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,'xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx' 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。

- 建议对报表类的或生成基础数据的查询，使用物化视图(MATERIALIZEDVIEW)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
```

- 建议复杂的统计查询可以尝试窗口函数 。
- 两表join时尽量的使用分布key进行join。

所以在建立业务的主表，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_i
d;
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.TBa
se_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.tba
```

```
se_detail WHERE (TBase_main.id = TBase_detail.TBase_main_id)
(4 rows)

postgres=#
```

- 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。

- 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。

- 不要对字段建立外键。

目前TDSQL PG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。

TDSQL-PG的进阶开发
高级sql语句编写

最近更新时间: 2024-06-12 15:06:00

数据库开发基础
TDSQL-PG的数据类型
数字类型

最近更新时间: 2024-06-12 15:06:00

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

字符类型

最近更新时间: 2024-06-12 15:06:00

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

二进制数据类型

最近更新时间: 2024-06-12 15:06:00

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

日期类型

最近更新时间: 2024-06-12 15:06:00

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

布尔类型

最近更新时间: 2024-06-12 15:06:00

布尔类型

名字	存储字节	描述
boolean	1字节	状态为真或假

更多的数据类型介绍

最近更新时间: 2024-06-12 15:06:00

请单击 postgresql官网数据类型说明 查看更多的数据类型说明。

消息及异常输出
RAISE NOTICE

最近更新时间: 2024-06-12 15:06:00

```
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)。

RAISE EXCEPTION

最近更新时间: 2024-06-12 15:06:00

```
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)。

RAISE EXCEPTION 自定义ERRCODE

最近更新时间: 2024-06-12 15:06:00

```
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,4/36715
9,0,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,4/36
7159,0,ERROR,23505," 程序EXCEPTION ",,,,,,,,,"psql"
```

控制结构
判断语句
IF...THEN...END IF

最近更新时间: 2024-06-12 15:06:00

```
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=#
```

IF...THEN...ELSE...END IF

最近更新时间: 2024-06-12 15:06:00

```
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=#
```

IF...THEN...ELSIF...THEN...ELSE...END IF

最近更新时间: 2024-06-12 15:06:00

```
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)
```

CASE语句

最近更新时间: 2024-06-12 15:06:00

```
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)
```

循环语句
LOOP循环

最近更新时间: 2024-06-12 15:06:00

```
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
```

WHILE循环

最近更新时间: 2024-06-12 15:06:00

```
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)
```

FOR循环

最近更新时间: 2024-06-12 15:06:00

```
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
```

FOR循环查询结果

最近更新时间: 2024-06-12 15:06:00

```
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)
```

FOREACH循环一个数组

最近更新时间: 2024-06-12 15:06:00

```
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
```

其它控制语句
动态执行

最近更新时间: 2024-06-12 15:06:00

```
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。

执行一个没有结果的命令

最近更新时间: 2024-06-12 15:06:00

```
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=#
```

获取执行结果

最近更新时间: 2024-06-12 15:06:00

```
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)
```

获取影响行数

最近更新时间: 2024-06-12 15:06:00

```
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)
```

俘获错误
错误俘获处理

最近更新时间: 2024-06-12 15:06:00

```
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)
```

获取错误相关信息

最近更新时间: 2024-06-12 15:06:00

```
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)
```

应用程序语法介绍
建立函数语法

最近更新时间: 2024-06-12 15:06:00

```
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;
```

[OR REPLACE] 更新函数介绍

最近更新时间: 2024-06-12 15:06:00

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)
```

[模式名.]函数名介绍

最近更新时间: 2024-06-12 15:06:00

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

连接到数据库（replace）
可视化开发工具TStudio
如何打开TStudio

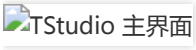
最近更新时间: 2024-06-12 15:06:00

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://imgcache.finance.cloud.tencent.com:80172.16.0.29:5050/

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

TStudio主界面说明

最近更新时间: 2024-06-12 15:06:00



TStudio 主界面

如何添加要管理的节点

最近更新时间: 2024-06-12 15:06:00

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

如何创建数据表、索引

创建数据表

最近更新时间: 2024-06-12 15:06:00

1. 单击菜单项目 **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：

4. 编辑完第一页和第二页框信息后单击**保存**即可建立表名为"t"的数据表。

给表建立索引

最近更新时间: 2024-06-12 15:06:00

1. 单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：

4. 编辑完第一页和第二页框信息后单击**保存**按钮即可为表"t"建立索引"t_nickname_idx"。

运行手工编写脚本

最近更新时间: 2024-06-12 15:06:00

1. 选择左边菜单项目 postgres库。

2. 单击顶部菜单工具 > 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。

表数据导入导出数据

最近更新时间: 2024-06-12 15:06:00

1. 选择左边菜单项目要导出或导入数据的表。

2. 单击顶部**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]#
```

shell交互客户端psql 连接到一个数据库

最近更新时间: 2024-06-12 15:06:00

- 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://&#x74;&#x62;&#97;&#x73;&#x65;&#x40;&#49;&#55;&#x32;&#46;&#x31;&#54;&#46;&#48;&#46;&#x32;&#x39;: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]$ 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
```

建立一个新连接

最近更新时间: 2024-06-12 15:06:00

- 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 172.16.0.4715432
You are now connected to database"postgres" as user "tbase" on host "172.16.0.47"at port "15432".
```

显示和设置该连接当前运行参数

最近更新时间: 2024-06-12 15:06:00

- 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

``` cpp
postgres=# \pset format html
Output format is html.
postgres=# \d tbase
<table border="1">
<caption>Table "public.tbase"</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">
</tr>
<tr valign="top">
<td align="left">mc</td>
<td align="left">text</td>
<td align="left">
```

</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

``` cpp

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

``` cpp
postgres=# \encoding
UTF8
#配置客户端编码为SQL_ASCII
postgres=# \encoding sql_ascii
postgres=# \encoding
SQL_ASCII
```
```

退出连接

最近更新时间: 2024-06-12 15:06:00

```
postgres=# \q
```

psql执行一个sql命令

最近更新时间: 2024-06-12 15:06:00

- 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
```

psql执行一个sql文件中所有命令

最近更新时间: 2024-06-12 15:06:00

- 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)
```

调用编辑器编写sql脚本

最近更新时间: 2024-06-12 15:06:00

```
postgres=# \e
"/tmp/psql.edit.5532.sql" 2L, 35Cwritten
```

编辑完成后保证退出执行。

调用外部命令

最近更新时间: 2024-06-12 15:06:00

```
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=#
```

将执行的结果保存到文件

最近更新时间: 2024-06-12 15:06:00

```
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)
```

改变当前的工作目录

最近更新时间: 2024-06-12 15:06:00

```
postgres=# \! ls
backup data install java log.txt pgbench tbase tbaseconf shell tbase.sql
postgres=# \cd /data
postgres=# \! ls
package pgsql tbase tbaseConfdb
postgres=#
```

插件管理

最近更新时间: 2024-06-12 15:06:00

- 查看当前库加载了那些插件。

```
postgres=# \dx
List of installedextensions
Name | Version | Schema | Description
--------------------+---------+------------+-------------------------------------------------------------
pg_stat_statements | 1.1 | public | track execution statistics of all SQLstatements executed
plpgsql | 1.0 | pg_catalog | PL/pgSQL procedurallanguage
(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
```

数据库相关操作

最近更新时间: 2024-06-12 15:06:00

- \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 | unmodifiableempty database
| | | | | tbase=CTc/tbase | | |
template1 | tbase | UTF8 | en_US.utf8 | en_US.utf8 | =c/tbase +| 14 MB | pg_default | defaulttemplate for new databases
| | | | | tbase=CTc/tbase | | |
(3 rows)
```

- 创建一个新库。

```
postgres=# create database mydb;
CREATE DATABASE
postgres=#
```

模式相关操作

最近更新时间: 2024-06-12 15:06:00

- \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 +| standardpublic schema
| | =UC/tbase |
(2 rows)
```

- 创建一个新模式。

```
postgres=# create schema mysche;
CREATE SCHEMA
```

用户相关操作

最近更新时间: 2024-06-12 15:06:00

- \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, CreateDB | {}
```

- \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, CreateDB | {} |
```

- 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:
```

表相关操作

最近更新时间: 2024-06-12 15:06:00

- 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)
```

- \det 列出外部表 。

```
postgres=# \det
List of foreign tables
Schema | Table | Server
--------+-------------+-----------------
public | t_table_csv | exttable_server
(1 row)
```

视图相关操作

最近更新时间: 2024-06-12 15:06:00

- 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 | 0bytes |
```

```
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 | 0bytes |
(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 | 0bytes |
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)
```

物化视图相关操作

最近更新时间: 2024-06-12 15:06:00

- 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)
```

序列相关操作

最近更新时间: 2024-06-12 15:06:00

- 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 | 8192bytes |
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 | 8192bytes |
(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)
```

索引相关操作

最近更新时间: 2024-06-12 15:06:00

- 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)
```

函数相关操作

最近更新时间: 2024-06-12 15:06:00

- 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+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
```

自定义数据类型相关操作

最近更新时间: 2024-06-12 15:06:00

- 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 ofdata 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 ofdata 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 ofdata 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 ofdata 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 ofdata types
Schema | Name | Internal name | Size |Elements | Owner | Access privileges | Description
--------+------------+---------------+------+----------+-------+-------------------+-------------
public | bug_status | bug_status | 4 | new +| tbase | |
| | | | open +| | |
| | | | closed | | |
```

存储过程语句相关操作

最近更新时间: 2024-06-12 15:06:00

- 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)
```

列出表、视图和序列和它们相关的访问权限

最近更新时间: 2024-06-12 15:06:00

- 显示所有对象的访问权限。

```
postgres=# \dp
Accessprivileges
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
Accessprivileges
Schema | Name | Type | Access privileges | Column privileges | Policies
--------+-------+-------+-------------------+-------------------+----------
public | tbase | table | tbase=arwdDxt/tbase+| |
| | | pgxc=r/tbase | |
(1 row)

postgres=# \dp public.*
Accessprivileges
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)
```

列出库或用户定义的配置

最近更新时间: 2024-06-12 15:06:00

```
postgres=# \drds
List of settings
Role | Database | Settings
------+----------+---------------------------------
pgxc | | log_statement=none
| postgres | search_path="$user", public, pgxc
(2 rows)
```

copy命令的使用

最近更新时间: 2024-06-12 15:06:00

这是一个针对客户端文件操作的copy应用，服务器端copy使用，详见章节copy的使用。

- \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';
```

copy FROM stdin使用方法

最近更新时间: 2024-06-12 15:06:00

```
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)
```

注：1 和tbase 之间不是空格，而是tab符

打印当前查询缓冲区到标准输出

最近更新时间: 2024-06-12 15:06:00

```
postgres=# select * from tbase;
id | mc
----+-------
1 | tbase
(1 row)

postgres=# \p
select * from tbase;
```

自定义显示格式

最近更新时间: 2024-06-12 15:06:00

- 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
-- ------
1tbase
2(null)
(2 rows)
```

- 只显示记录。

```
postgres=# \pset tuples_only
Showing only tuples.
postgres=# select * from tbase;
1tbase
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)
```

显示psql内部操作

最近更新时间: 2024-06-12 15:06:00

```
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' W
HEN '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
```

重复执行上一条语句

最近更新时间: 2024-06-12 15:06:00

```
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。

sql命令帮助查看

最近更新时间: 2024-06-12 15:06:00

```
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=#
```

ems postgresql manager
ems postgresql manage介绍

最近更新时间: 2024-06-12 15:06:00

这是一款商业的postgresql的客户端管理工具，可以用来创建对象，执行sql，ems postgresql manage分为收费版本和免费的lite版本，使用上lite版本基本足够使用。

连接到tbase服务

最近更新时间: 2024-06-12 15:06:00

查询表结构

最近更新时间: 2024-06-12 15:06:00

双击数据表即可显示表结构，也可增减字段。

执行SQL语句

最近更新时间: 2024-06-12 15:06:00

单击菜单**tools** > **query data**，输入要执行的sql，然后单击**execute**即可。

连接tbase-v5出错处理

最近更新时间: 2024-06-12 15:06:00

这是因为tbase merge了一些postgresql11存储过程的一些特性，我们可以通过定义一个新的pg_proc视图来达到兼容访问，如下所示。

- 连接到数据库后创建一个兼容的专用schema。

```
postgres=# create schema tbase_pg_proc;
CREATE SCHEMA

postgres=# CREATE OR REPLACE VIEW tbase_pg_proc.pg_proc as
select
*,
case when prokind='a' then true else false end as proisagg,
case when prokind='w' then true else false end as proiswindow,
oid as oid,xmin as xmin,tableoid tableoid
from
pg_catalog.pg_proc;
CREATE VIEW
postgres=#
```

- 配置用户的搜索路径。

```
postgres=# alter role tbase set search_path to tbase_pg_proc,pg_catalog,"$user", public;
ALTER ROLE
postgres=#
```

再次断开，连接就可以正常访问function了。

- 适配function和procedure分开。

```
postgres=# create or replace function tbase_pg_proc.version() returns text as
$$
begin
return 'PostgreSQL 11.0 TBase V5 on x86_64-pc-linux-gnu, compiled by gcc (GCC) 4.8.5 20150623 (Red Hat 4.8.5-39), 64
-bit';
end;
$$
language plpgsql;
```

返回postgresql11版本号就能将procedure和function 分别返回。

快速入门
客户端psql的使用

最近更新时间: 2024-06-12 15:06:00

psql简介

psql是一个基于终端的 TDSQL PostgreSQL 版客户端工具，类似Oracle中的命令行工具sqlplus，但比sqlplus更加强大。它让你能交互式地键入查询，把它们发送给 TDSQL PostgreSQL 版，并且查看查询结果。或者，输入可以来自于一个文件或者命令行参数。此外，psql还提供一些元命令和多种类似shell的特性来为编写脚本和自动化多种任务提供便利。psql还支持命令补全，历史命令回找。

psql使用环境配置

TDSQL PostgreSQL 版-v2是基于多租户设计，安装后所有机器都不会配置 TDSQL PostgreSQL 版的客户端默认使用环境，所以使用psql需要配置其环境变量。切换到数据库用户tbase，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，因为测试只初始化了一个实例，所以我们可以将它配置到 ~/.bashrc 中。

创建 TDSQL PostgreSQL 版默认的分布式使用环境

TDSQL PostgreSQL 版作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 node group。

切换到 TDSQL PostgreSQL 版用户：su - tbase

- 连接数据库

> **注意：此处指连接到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 -p port -d database -U user -h
```

host（psql连接参数就是这些，超级管理用户就是数据库用户tbase，默认数据库为postgres，host如果是在本机连接可省略，默认使用本机ip）

示例：

```
[tbase@node1 ~]$ psql -p 11000 -d postgres

psql (PostgreSQL 10.0 TBase V2)

Type "help" for help.
```

- 查询集群节点配置

```
postgres=# select * from pgxc_node;
node_name | node_type | node_port | node_host | nodeis_primary |
nodeis_preferred | node_id | node_cluster_name
-----------+-----------+-----------+-------------+---------------+-----------------+-------------+------------------
agtm_0 | G | 11004 | 172.16.0.12 | t | f | 475343005 | tbase_cluster
cn001 | C | 11000 | 172.16.0.12 | f | f | -264077367 | tbase_cluster
cn002 | C | 11000 | 172.16.0.5 | f | f | -674870440 | tbase_cluster
dn001 | D | 11002 | 172.16.0.12 | f | f | 2142761564 | tbase_cluster
dn002 | D | 11002 | 172.16.0.5 | f | f | -17499968 | tbase_cluster
dn003 | D | 11006 | 172.16.0.12 | f | f | -1956435056 | tbase_cluster
(6 rows)
```

- 查询当前数据节点（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

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

dn001 | D | 11002 | 172.16.0.12 | f | f | 2142761564 | tbase_cluster

dn002 | D | 11002 | 172.16.0.5 | f | f | -17499968 | tbase_cluster

dn003 | D | 11006 | 172.16.0.12 | f| f | -1956435056 | tbase_cluster
```

- 创建数据表默认使用的group

```
postgres=# create default node group default_group with(dn001, dn002);
```

- 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 TDSQL PostgreSQL 版各个哈希值与DN的对照表，下面演示如何创建一个shardmap给default_group

执行以下指令，即可像单机一样使用 TDSQL PostgreSQL 版集群：

```
postgres=# create sharding group to group default_group;

postgres=# clean sharding;
```

更多group的使用方法

创建扩展group

- 建立group

```
postgres=# create node group ext_group with(dn003);

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=#
```

删除group

```
postgres=# drop sharding in group ext_group;

DROP SHARDING GROUP

postgres=# drop node group ext_group ;

DROP NODE GROUP
```

查看集群group

```
postgres=#select * from pgxc_group;
```

psql常用命令使用

最近更新时间: 2024-06-12 15:06:00

连接到一个数据库实例

- 使用参数连接

```
[pgxz@VM_0_3_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

```
[pgxz@VM_0_3_centos root]$ psql
postgresql://pgxz@172.16.0.29:15432/postgres
psql (PostgreSQL 10 (TBase 2.01))
Type "help" for help.

postgres=#
```

建立一个新连接

- 查看当前连接会话pid

```
postgres=# select pg_backend_pid();
pg_backend_pid
--
2408
(1 row)
```

- 连接到当前库

```
postgres=# \c
You are now connected to database "postgres" as user "tbase".
postgres=# select pg_backend_pid();
pg_backend_pid
--
2426
(1 row)
```

- 连接到其他库

```
postgres=# \c template1
You are now connected to database "template1" as user "tbase".
template1=#
```

- 连接到其他用户

```
postgres=# \c - postgres
You are now connected to database "postgres" as user "postgres".
```

- 连接到其他服务器上的库

```
postgres=# \c postgres tbase 172.16.0.5 11000

You are now connected to database "postgres" as user "tbase" on host
"172.16.0.5" at port "11000".
```

显示和设置该连接当前运行参数

- 显示当前连接用户

```
postgres=# SELECT CURRENT_USER;
current_user
--------------
pgxz
(1 row)
```

- 显示当前连接的schema

```
postgres=# show search_path ;
search_path
----------------
"$user",public
(1 row)
```

- 显示节点参数值

```
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)
```

- 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
```

退出连接

postgres=# \q

psql执行一个sql命令

- 显示标题

```
[pgxz@VM_0_3_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

```
[pgxz@VM_0_3_centos root]$ psql -h 172.16.0.29 -p 15432 -U tbase -d
postgres -t -c "select count(1) from pg_class"
317
```

psql执行一个sql文件中所有命令

- 在外部执行

```
[pgxz@VM_0_3_centos ~]$ cat /data/tbase/tbase.sql
set search_path = public;
insert into tbase values(1,2);
select count(1) from tbase;
[pgxz@VM_0_3_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

```
[pgxz@VM_0_3_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)
```

数据库相关操作

- 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）

```
postgres=# \l+
List of databases
Name | Owner | Encoding | Collate | Ctype | Access privileges | Size
| Tablespace | Description
-----------+-------+----------+------------+------------+-------------------+-------+------------+-----------------------------
-------------
postgres | tbase | UTF8 | en_US.utf8 | en_US.utf8 | | 31 MB |
pg_default | default administrative connection database

template0 | tbase | UTF8 | en_US.utf8 | en_US.utf8 | =c/tbase +| 27 MB
| pg_default | unmodifiable empty database

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

template1 | tbase | UTF8 | en_US.utf8 | en_US.utf8 | =c/tbase +| 27 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
```

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
```

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
postgres=# create user pgxz ;（使用create user，那么用户自动拥有login权限）
```

- 配置用户密码

```
postgres=# \password pgxc
Enter new password:
Enter it again:
```

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)
```

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
```

sql帮助命令

```
postgres=# \h
postgres=# \h create table;
\h 可以查看一些sql命令的使用方法。
```

操作指南
数据库使用说明

最近更新时间: 2024-06-12 15:06:00

psql是一个基于终端的 TDSQL PostgreSQL 版客户端工具，类似Oracle中的命令行工具sqlplus，但比sqlplus强大多了。它让你能交互式地键入查询，把它们发送给 TDSQL PostgreSQL 版，并且查看查询结果。或者，输入可以来自于一个文件或者命令行参数。此外，psql还提供一些元命令和多种类似 shell 的特性来为编写脚本和自动化多种任务提供便利。psql还支持命令补全，历史命令回找。

连接数据库参数介绍

```
-d dbname

--dbname=dbname
```

指定要连接到的数据库名，不指定该参数时默认是从PGDATABASE环境变量中取得（如果被设置）。

```
-h host

--host=host
```

指定要连接服务器机器的主机名或者ip地址。不指定该参数时默认是从PGHOST环境变量中取得（如果被设置）。

```
-p port

--port=port
```

指定要连接服务器监听的 TCP端口，不指定该参数时默认是从PGPORT环境变量中取得（如果被设置），否则使用编译在程序中的默认值。

```
-U username

--username=username
```

指定要连接服务的用户名，不指定该参数时默认是从PGUSER环境变量中取得（如果被设置），否则使用当前操作系统用户名做为默认值。

```
-w

--no-password
```

从不发出一个口令提示。如果服务器要求口令认证并且没有其他方式提供口令（例如一个.pgpass文件），那么连接尝试将失败。这个选项对于批处理任务和脚本有用，因为在其中没有一个用户来输入口令。

```
-W

--password
```

强制pg_dump在连接到一个数据库之前提示要求一个口令。

连接到一个数据库

- 使用参数连接

```
[pgxz@VM_0_3_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

```
[pgxz@VM_0_3_centos root]$ psql
postgresql://pgxz@172.16.0.29:15432/postgres

psql (PostgreSQL 10 (TBase 2.01))

Type "help" for help.

postgres=#
```

- 配置证书连接

```
[pgxz@VM_0_3_centos ~]$ psql 'host=172.16.0.29 port=15432 dbname=postgres
user=tbase sslmode=verify-ca sslcert=postgresql.crt sslkey=postgresql.key
sslroo腾讯云金融专区rt=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

```
[pgxz@VM_0_3_centos root]$ PGUSER=tbase

[pgxz@VM_0_3_centos root]$ PGHOST=127.0.0.1

[pgxz@VM_0_3_centos root]$ PGDATABASE=postgres

[pgxz@VM_0_3_centos root]$ PGPORT=15432

[pgxz@VM_0_3_centos root]$ export PGHOST PGUSER PGDATABASE PGPORT

[pgxz@VM_0_3_centos root]$ psql
```

```
psql (PostgreSQL 10 (TBase 2.01))
Type "help" for help.
postgres=#
```

变些环境变量可以配置在

```
[pgxz@VM_0_3_centos root]$ cat ~/.bashrc
# .bashrc
PGUSER=tbase
PGHOST=127.0.0.1
PGDATABASE=postgres
PGPORT=15432
export PGHOST PGUSER PGDATABASE PGPORT
```

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 "pgxz".
postgres=# select pg_backend_pid();
pg_backend_pid
----------------
2426
(1 row)
postgres=# \c template1
You are now connected to database "template1" as user "pgxz".
```

```
template1=#
```

- 连接到外一台服务

```
postgres=# \c postgres pgxz 172.16.0.47 15432

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

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

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

pgxz

(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 | pgxz
public | tbase | table | pgxz
(2 rows)
postgres=# \set ECHO_HIDDEN off
postgres=# \dt
List of relations
Schema | Name | Type | Owner
--------+--------------+-------+-------
public | t_time_range | table | pgxz
public | tbase | table | pgxz
(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 \&amp;quot;public.tbase&amp;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">\&amp;nbsp; </td>
</tr>
<tr valign="top">
<td align="left">mc</td>
<td align="left">text</td>
<td align="left">\&amp;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
```

退出连接

postgres=# \q

psql执行一个sql命令

- 显示标题

```
[pgxz@VM_0_3_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

```
[pgxz@VM_0_3_centos root]$ psql -h 172.16.0.29 -p 15432 -U tbase -d
postgres -t -c "select count(1) from pg_class"

317
```

实例管理

最近更新时间: 2024-06-12 15:06:00

实例创建

1. 登陆管理控制台选择TDSQL PostgreSQL 版菜单，进入TDSQL PostgreSQL 版的控制台，可查看实例列表。

   TDSQL PostgreSQL 版支持两种内核实例的创建，分别为TDSQL PostgreSQL 版和PostgreSQL 版。

2. 点击【新建】按钮，创建实例，如下图所示。

3. 新建实例时需指定登陆用户的密码，实例所属地域和可用区，实例所属私有网络，以及协调节点和数据节点的规格。具体如下图所示：

实例销毁

进入TDSQL PostgreSQL 版管理控制台，在实例列表的操作栏点击【更多】-》【销毁】来销毁一个实例。注意：实例销毁后数据将被清空，请谨慎操作。

修改实例名

进入控制台实例列表页面，点击实例名称旁边的图标可修改实例名称。具体如下图所示。 ）

查看实例详情

在控制台实例列表点击实例ID或者【操作】-》【管理】可查看实例详情。如下图所示。

重置密码

在TDSQL PostgreSQL 版管理控制台实例列表页面，点击【管理】-》【管理】-》【重置密码】来重置实例生产时指定的密码。具体如下图所示。

节点管理

- 此界面功能有节点起停、添加节点、主从切换、重做备机、删除备节点、创建节点组等功能。节点起停即控制单个或者多个节点启动、停止服务；
- 添加节点即给集群添加新的节点，所有类型节点都能添加，只要服务器系统资源足够；
- 主从切换就是主节点与备节点就行倒换，倒换后原来的备节点变成主节点，原来的主节点变成备节点，节点组是使用TDSQL PostgreSQL 版分布式环境的前提，如果没有节点组，就无法建表、写入数据等；
- 重做备机是指重新初始化一个备机，只有在主机正常而备机故障且无法恢复的情况下才能执行此操作。

在线扩容

TDSQL PostgreSQL 支持三种扩容方式，分别为纵向调整节点配置、横向添加节点以及快速扩容，满足您灵活的配置调整需求。分别详述如下：

- 调整节点配置

  调整节点配置为为调整CN或者DN节点的配置，实例整体的架构不变（节点的组数不变）。其操作如下图所示：

- 添加节点

  添加节点为横向变更实例的架构（节点的组数会进一步随之改变）。当单节点的配置无法再扩展之后，建议您使用该方式扩容。该扩容方式是分布式相对于集中式的一个突出的优势。

- 快速扩容

  在分布式数据库节点较多，数据较大，通常扩容涉及搬迁数据，因此快速扩容是分布式数据库的必不可少的功能。快速扩容会让您选择某几组DN节点，迅速的将这些DN节点一扩为二，并且能指定切换时间，极大的提高了大型分布式数据库实例的扩容速度。

参数配置

TDSQL PostgreSQL 数据库支持CN节点以及DN节点的参数的自助修改。并能够记录参数的修改行为。

备份管理

实例创建后自动开启实例备份，并支持自动备份的设置。系统能记录一段时间内的备份记录。TDSQL PostgreSQL版 支持**自动备份**和**手动备份**两种方式来备份数据库。

> **说明：**
>
> 自动备份为物理备份，物理备份期间系统会对每个平面 CN 执行加锁，业务无法执行 DDL 语句。

您可以自助设置备份的开始时间。备份开始时间默认时间为系统自动分配的备份发起时间。可自定义选择时间区间，建议设置为业务低峰期。备份发起时间只是备份开始启动的时间，并不代表备份结束的时间。例如，选择：02:00 - 04:00点，系统会在02:00 - 04:00时间范围

内的某一个时间点发起备份，具体的发起时间点取决于服务端备份策略和备份系统状况。 如您由于变更需要手动备份，系统提供了该功能。

资源池

最近更新时间: 2024-06-12 15:06:00

TDSQL PostgreSQL 版及PostgreSQL支持将机器资源分成多个资源池。资源池便于设备和资源的有效管控。例如，业务当前有三个项目，相互独立计算成本和管控资源，则可以将机器划分为三个资源池，分别给三个业务使用。实例不能跨资源池部署。

新增资源池

默认所有的机器上架时都在系统默认的公共资源池。如果您需要新增资源池，则需要在运维端进行操作。

新增资源池操作如下图所示：

变更资源池与租户的关系

资源池默认是不能被租户在租户端的购买页看到的，一个新的资源池需要变更跟租户的关系后才能被相应的租户所看到。具体如下图所示：

隔离方式说明

最近更新时间: 2024-06-12 15:06:00

TDSQL PostgreSQL 版和PostgreSQL当前支持两种隔离方式：实例级和节点级。分别说明如下：

实例级

实例级隔离指定的是当前实例在每台机器上所能使用的CPU和内存资源的上限。例如，某个实例指定的是实例级隔离方式，指定的隔离资源是10核CPU 20GB内存，实例最终在3台机器上部署，则该实例最终可使用的资源是30核CPU 60GB内存。

节点级

节点级隔离指定的是当前实例的每一个节点所能使用的CPU和内存资源的上限。例如，某实例指定的是节点级隔离方式，指定的隔离资源是10核CPU 20GB内存，实例一共有6个节点，则实例最终可使用的资源是60核120GB内存。

客户端psql的使用

最近更新时间: 2024-06-12 15:06:00

psql简介

psql是一个基于终端的 TDSQL PostgreSQL 版客户端工具，类似Oracle中的命令行工具sqlplus，但比sqlplus更加强大。它让你能交互式地键入查询，把它们发送给 TDSQL PostgreSQL 版，并且查看查询结果。或者，输入可以来自于一个文件或者命令行参数。此外，psql还提供一些元命令和多种类似shell的特性来为编写脚本和自动化多种任务提供便利。psql还支持命令补全，历史命令回找。

psql使用环境配置

Tbase-v2是基于多租户设计，安装后所有机器都不会配置 TDSQL PostgreSQL 版的客户端默认使用环境，所以使用psql需要配置其环境变量。切换到数据库用户 TDSQL PostgreSQL 版，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，因为测试只初始化了一个实例，所以我们可以将它配置到~/.bashrc 中。

创建 TDSQL PostgreSQL 版默认的分布式使用环境

TDSQL PostgreSQL 版作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 node group。

切换到tbase用户：su - tbase

- 连接数据库

> **注意：此处指连接到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 -p port -d database -U user -h
```

host（psql连接参数就是这些，超级管理用户就是数据库用户tbase，默认数据库为postgres，host如果是在本机连接可省略，默认使用本机ip）

示例：

```
[tbase@node1 ~]$ psql -p 11000 -d postgres

psql (PostgreSQL 10.0 TBase V2)
```

```
Type "help" for help.
```

- 查询集群节点配置

```
postgres=# select * from pgxc_node;
node_name | node_type | node_port | node_host | nodeis_primary |
nodeis_preferred | node_id | node_cluster_name
-----------+-----------+-----------+-------------+---------------+-----------------+-------------+------------------
agtm_0 | G | 11004 | 172.16.0.12 | t | f | 475343005 | tbase_cluster
cn001 | C | 11000 | 172.16.0.12 | f | f | -264077367 | tbase_cluster
cn002 | C | 11000 | 172.16.0.5 | f | f | -674870440 | tbase_cluster
dn001 | D | 11002 | 172.16.0.12 | f | f | 2142761564 | tbase_cluster
dn002 | D | 11002 | 172.16.0.5 | f | f | -17499968 | tbase_cluster
dn003 | D | 11006 | 172.16.0.12 | f | f | -1956435056 | tbase_cluster
(6 rows)
```

- 查询当前数据节点（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

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

dn001 | D | 11002 | 172.16.0.12 | f | f | 2142761564 | tbase_cluster

dn002 | D | 11002 | 172.16.0.5 | f | f | -17499968 | tbase_cluster

dn003 | D | 11006 | 172.16.0.12 | f| f | -1956435056 | tbase_cluster
```

- 创建数据表默认使用的group

```
postgres=# create default node group default_group with(dn001, dn002);
```

- 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 TDSQL PostgreSQL 版各个哈希值与DN的对照表，下面演示如何创建一个shardmap给default_group

执行以下指令，即可像单机一样使用 TDSQL PostgreSQL 版集群：

```
postgres=# create sharding group to group default_group;

postgres=# clean sharding;
```

更多group的使用方法

创建扩展group

- 建立group

```
postgres=# create node group ext_group with(dn003);

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=#
```

删除group

```
postgres=# drop sharding in group ext_group;

DROP SHARDING GROUP

postgres=# drop node group ext_group ;

DROP NODE GROUP
```

查看集群group

```
postgres=#select * from pgxc_group;
```

psql常用命令使用

最近更新时间: 2024-06-12 15:06:00

连接到一个数据库实例

- 使用参数连接

```
[pgxz@VM_0_3_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

```
[pgxz@VM_0_3_centos root]$ psql
postgresql://pgxz@172.16.0.29:15432/postgres
psql (PostgreSQL 10 (TBase 2.01))
Type "help" for help.

postgres=#
```

建立一个新连接

- 查看当前连接会话pid

```
postgres=# select pg_backend_pid();
pg_backend_pid
--
2408
(1 row)
```

- 连接到当前库

```
postgres=# \c
You are now connected to database "postgres" as user "tbase".
postgres=# select pg_backend_pid();
pg_backend_pid
--
2426
(1 row)
```

- 连接到其他库

```
postgres=# \c template1
You are now connected to database "template1" as user "tbase".
template1=#
```

- 连接到其他用户

```
postgres=# \c - postgres
You are now connected to database "postgres" as user "postgres".
```

- 连接到其他服务器上的库

```
postgres=# \c postgres tbase 172.16.0.5 11000

You are now connected to database "postgres" as user "tbase" on host
"172.16.0.5" at port "11000".
```

显示和设置该连接当前运行参数

- 显示当前连接用户

```
postgres=# SELECT CURRENT_USER;
current_user
--------------
pgxz
(1 row)
```

- 显示当前连接的schema

```
postgres=# show search_path ;
search_path
----------------
"$user",public
(1 row)
```

- 显示节点参数值

```
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)
```

- 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
```

退出连接

postgres=# \q

psql执行一个sql命令

- 显示标题

```
[pgxz@VM_0_3_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

```
[pgxz@VM_0_3_centos root]$ psql -h 172.16.0.29 -p 15432 -U tbase -d
postgres -t -c "select count(1) from pg_class"
317
```

psql执行一个sql文件中所有命令

- 在外部执行

```
[pgxz@VM_0_3_centos ~]$ cat /data/tbase/tbase.sql
set search_path = public;
insert into tbase values(1,2);
select count(1) from tbase;
[pgxz@VM_0_3_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

```
[pgxz@VM_0_3_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)
```

数据库相关操作

- 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）

```
postgres=# \l+
List of databases
Name | Owner | Encoding | Collate | Ctype | Access privileges | Size
| Tablespace | Description
-----------+-------+----------+------------+------------+-------------------+-------+------------+-----------------------------
-------------
postgres | tbase | UTF8 | en_US.utf8 | en_US.utf8 | | 31 MB |
pg_default | default administrative connection database

template0 | tbase | UTF8 | en_US.utf8 | en_US.utf8 | =c/tbase +| 27 MB
| pg_default | unmodifiable empty database

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

template1 | tbase | UTF8 | en_US.utf8 | en_US.utf8 | =c/tbase +| 27 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
```

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
```

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
postgres=# create user pgxz ;（使用create user，那么用户自动拥有login权限）
```

- 配置用户密码

```
postgres=# \password pgxc
Enter new password:
Enter it again:
```

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)
```

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
```

sql帮助命令

```
postgres=# \h
postgres=# \h create table;
\h 可以查看一些sql命令的使用方法。
```

最佳实践
设计开发规范

最近更新时间: 2024-06-12 15:06:00

设计准则

命名准则

数据库对象名命名准则

- 数据库对象: 模式名, 表名,视图名, 字段名，索引名等名称：
- 使用小写字母、数字、下划线的组合。
- 禁用双引号即"包围，除非必须包含大写字母。
- 注意TDSQL PostgreSQL 版的对象名称长度不能超过63个字符。
- 不要以pg_开头或者pgxc_（避免与系统表，系统视图名称混淆），禁用以数字开头。
- 禁止使用 SQL 关键字例如 type, order 等，所有关键字见TDSQL PostgreSQL 版关键字列表。
- 尽量用英文单词，这样使用者根据英文单词也可以大概知道数据库对象的用途，不用再去查找文档；
- 由于数据库对象名不能够超过63个字符，所以当超过63个字符时，英文单词应该缩写。

表名命名规则

同一业务的表名前缀保持一致： 保险业务相关可以写成：policy_basic，policy_xxxx。

视图命名规则

同一业务的表名前缀保持一致，视图带上对象类型为view如： 保险业务相关可以写成：policy_basic_view。

物化视图命名规则

同一业务的物化视图名前缀保持一致，物化视图的对象类型为mv如： 保险业务相关可以写成：policy_basic_mv。

序列命名规则

序列命名为【表名_字段名_seq】：如表policy_basic的id字段使用的序列

```
policy_basic_id_seq
```

主键命名规则

主键命名规则为【表名_字段名_pkey】，如果是多个字段组合成主键，则每个字段使用下划画隔开，如

```
policy_basic_id_pkey
```

唯一索引命名规则

唯一索引命名规则为【表名_字段名_uidx】，如果是多个字段组合成的唯一索引，则每个字段使用下划画隔开，如

```
policy_basic_id_uidx
```

普通索引命名规则

普通索引命名规则为【表名_字段名_idx】，如果是多个字段组合成的索引，则每个字段使用下划画隔开，如

```
policy_basic_id_idx
```

临时表命名规则

临时表命名规则为【表名_tmp】，如

```
policy_basic_tmp
```

转储表命名规则

转储表命名规则为【表名_his】，如

```
policy_basic_his
```

字段命名规则

字段名称必须用字母开头，采用有特征含义的单词或缩写，不能用双引号包含。如

```
inserttime,updatetime
```

使用准则

shard（分片表）数据表设计准则

这里的数据表包括普通表，普通分区表，冷热分区表。

- 所有数据表都必需有主键。
- 业务表都必需有插入时间截和最后更新时间截字段，并且在业务发生时保存对应时间截。
- 单个表大小建议不要超过32G，有效的控制表大小方法，增加集群节点数可以降低单个表的大小，使用分区表可以降低单数据表的大小。
- 对表名增加COMMENT中文注释，便于后续新人了解业务及维护，其它对象也需要遵守这个规则。

分区表使用准则

分区表带来的好处就是表大小得到有效控制，索引的高度低，更新性能更好，坏处就是全表扫描更慢，系统元数据更大，DDL操作开销更大，使用分区表准则如下。

- 带有时间截字段（如插入时间）的业务数据表。
- 对该表进行查询时，业务大多数是限制在一定时间范围内进行。
- 对于高并发的查询业务，需要指定到某个分区范围执行查询才能发挥分区表的带来的好处。
- 删除记录和更新记录时必须限定在某个分区表内，否则更新效率低。

冷热分区表使用准则

冷热分区表使用除了具有分区表的特性外，还有这一点要特别注意，运维成本大一些，冷热数据搬迁需要占用磁盘IO，网络带宽，热group的数据无法与冷group的数据进行跨DN库join。

- 冷热分区表继承分区表的使用情景。
- 历史数据查询少。
- 历史数据存储容量大。
- 热数据存储介质成本大，冷GROUP可以使用便宜的SAS盘，解决存储成本的问题。
- 禁止热group和冷group join操作。

  满足上面的场景的话就可以考虑使用冷热分区表。

全局表使用准则

- 全局表是所有节点都有全量数据，对于大数据量的数据表不适合。
- 全局表更新性能较低，控制不好容易产生死锁，对于经常更新的业务不适合使用全局表。
- 数据经常要跨库JOIN的小数据量表可以考虑使用全局表。

序列使用准则

- 无特别要求，禁止在系统中使用序列，主要是序列需要与GTM频繁的通信，增加了GTM负载，降低了系统TPS值
- 使用 TDSQL PostgreSQL 版自带的UUID函数或应用程序的UUID代替。

外键使用准则

- 无特别要求，禁止使用外键，外键对应程序和运维带来极高要求，同时也增加了数据库性能开销。
- 可以使用事务来替代外键达到数据一致性要求。

触发器使用准则

- 无特别要求，禁止使用触发器，触发器需要把数据拉回接入层CN计算，数据路由复杂，性能低，开销大。
- 触发器不利于应用程序和数据库功能解偶，排错复杂。

存储过程使用准则

无特别要求，禁止在存储过程业务中使用

- 存储过程使用增加数据库服务器的额外负担。
- 存储过程也不利于应用程序和数据库功能解偶，业务还是放到业务代码中。

物化视图使用准则

无特殊要求，禁止使用物化视图。

- 物化视图生成数据存放于CN节点，如果需要使用物化视图，需要确保CN节点有足够的空间和良好的IO能力。
- 物化视图与其它表JOIN时，数据需要拉回协调节点CN进行计算，不适合于大量数据JOIN业务使用。

索引使用准则

- 能使用Btree 索引的就不要使用GIN索引。
- 有唯一约束要求的字段必需使用UNIQUE来约束。
- 索引越多，更新越慢。
- 索引也需要占用物理空间。
- 在线建立索引请使用CONCURRENTLY方式。
- 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，创建带where的索引；

```
select * from test where status=1 and col=?; -- 其中status=1为固定的条件

create index test_col_idx on test (col) where status=1;
```

- 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 test_xxx_idx on test ( exp(xxx) );
```

- 建议不要建过多index，一般不要超过6个，核心表（保订单）可适当增加index个数。

- 通过查询系统视图pg_stat_all_indexes可以评估索引使用情况 。

字段设计准则

- 能用数值类型的，就不用字符类型。
- 能用varchar(N)就不用char(N),以利于节省存储空间，而且varchar拉长时不需要更新全表数据。
- 能用varchar(N) 就不用text,varchar。
- 使用default NULL,而不用default '',以节省存储空间。
- 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
- 逻辑复制表禁止bytea大对象。

用户及权限设计准则

- 不准许在业务操作中使用DBA角色用户。
- 每个独立的应用都需要有独立的数据库用户。

开发准则

GROUP拆分原则

TDSQL PostgreSQL 版可以把集群中的部分数据节点定义成一个GROUP，然后在建立数据表时指定数据存放到这个GROUP中，实现不同业务表的物理资源使用隔离。如可以按不同的地区来进行拆分，这样拆分的好处

- 各个地区的数据完全隔离，互不影响。
- 各个地区可以独立扩容，包括该地区的冷热节点。
- 各个地区可以独立做灰度升级，互不影响。
- 某个数据节点DN故障倒换只会影响到该个地区的业务。
- 每个地区上业务时只需要评估该地区的资源要求。
- 对于非分布键的查询，更新，删除不会占用到其它地区服务器资源，而且访问的节点更少，性能及可靠性更高。

数据库拆分原则

TDSQL PostgreSQL 版的逻辑隔离有两个级别，分别是database（数据库）和schema（模式）级别，请使用使用schema级别，这样拆分后的好处

- 用户访问不同schema下的资源需要授权后才能访问，所以schema级别来说已经足够安全。

- schema拆分只是访问权限上的隔离，放开权限后，同一个数据源就可以访问不同schema下的数据表。
- 不同schema下的数据表可以在接入层CN进行JOIN查询，而database级别是无法这样操作。
- 从 TDSQL PostgreSQL 版的设计上来看，不同数据库连接增加了协调节点CN到数据节点DN这间连接池的维护工作，即随着数据库的增加，要维护的连接数会更多，占用资源更大
- 参考其它DB的使用习惯来看，oracle使用的是一用户对应一个schema。

数据库设计原则

数据库编码

数据库编码统一使用UTF8

- 建立UTF8编码数据库方法

```
create database postgres ENCODING 'utf8';
```

- 查询现在数据库编码方法

```
postgres=# select datname,pg_encoding_to_char(encoding) from pg_database ;
```

datname | pg_encoding_to_char

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

template1 | UTF8

template0 | UTF8

postgres | UTF8

(3 rows)

注意：数据库编码建立后就无法再修改。

数据库默认排序分组规则

数据库排序分组规则，使用zh_CN.utf8或者C，但推荐使用zh_CN.utf8，主要是因为zh_CN.utf8支持模糊查询和全文搜索，还有汉字排序时使用拼音首字母。

- 指定数据库排序分组规则

```bash
create database postgres ENCODING 'utf8' lc_collate 'zh_CN.utf8' LC_CTYPE

'zh_CN.utf8';
```

- 查询数据库排序分组规则

```
postgres=# select datname,datcollate,datctype from pg_database;
datname | datcollate | datctype
-----------+------------+-----------
template1 | zh_CN.utf8 | zh_CN.utf8
template0 | zh_CN.utf8 | zh_CN.utf8
postgres | zh_CN.utf8 | zh_CN.utf8
(3 rows)
```

schema设计技巧

在 TDSQL PostgreSQL 版中schema与用户不存在必然的联系，但建议为每个用户建立一个相同名称的schema，并且将该schema分配给其同名的用户，如

```
create schema picc_gd AUTHORIZATION picc_gd;
```

这样做好处就是使用picc_gd这个用户连接上来后，访问数据表时最先搜索的路径为picc_gd这个schema，这处简单的设计就能实现同一套程序匹配多套库的功能。

数据库用户设计原则

- 绝对不准许在业务中使用DBA角色用户。
- 不同应用请使用不同的用户来隔离数据表的访问。
- 用户停止使用后要禁用

建立普通用户的方法

```
create role picc_gd with login password 'picc_gd@123';
```

禁用用户方法

```
alter role picc_gd with nolgin;
```

分布键的选择原则

- 分布键只能选择一个字段。
- 如果有主键，则选择主键做分布键。
- 如果主键是复合字段组合，则选择字段值选择性多的字段做分布键。

- 也可以把复合字段拼接成一个新的字段来做分布键。
- 没有主键的可以使用UUID来做分布键。
- 总之一定要让数据尽可能的分布得足够散。

索引的设计原则

btree索引

创建方法

```
create table policy_basis(id int not null,cardid varchar(20) not null,primary
key(id));
create index policy_basis_cardid_idx on policy_base using btree(cardid);
```

什么情况下会使用到索引

- 单表=,>,>=,<=,between x and x,in操作，如

```
Select * from policy_basis where cardid=xxx;
Select * from policy_basis where cardid>xxx;
Select * from policy_basi where cardid>=xxx;
Select * from policy_basis where cardid>xxx and cardid<xxx;
Select * from policy_basis where cardid between xx and xxx;
Select * from policy_basis where cardid in('xx','xx');
```

- like左匹配模糊查询用法

  要使用like左匹配模糊查询，需要指定排序方法为 collate "C"

```
create table t_like (id varchar(32),cardid varchar(20),primary key(id) );
create index t_like_cardid_idx on t_like using btree(cardid collate "C");
```

  然后这样的语句就能走索引

  select * from t_like where cardid like '99999%';

  但其它操作符需要这样写才能使用索引

```
select * from t_like where cardid COLLATE "C" = '99999';
```

```
select * from t_like where cardid COLLATE "C" > '99999';
```

- order by + limit x 之类返回部分数据，如

``` bash
create table t_order_by(id int not null,cardid varchar(20) not null,primary

key(id));

insert into t_order_by select t,t from generate_series(1,1000000) as t;

select * from t_order_by order by id limit 10;

select * from t_order_by order by cardid limit 10;
```

上面的查询不管是分布键还是非分布键都要对排序的字段建立索引

```
create index t_order_by_id_idx on t_order_by(id);

create index t_order_by_cardid_idx on t_order_by(cardid);
```

如果是全表排序，则无需要建立索引。

- join连接查询

“分布键与分布键”，“分布键与非分布键”，“非分布键与非分布键”，只要是join连接扫描的数据量占比量少，建立索引都是能大大提高查询速度

```
ccreate table t_join_main(id int not null,cardid int,primary key(id));

create table t_join_detail(id int not null,cardid int,phone varchar(20),primary

key(id));

insert into t_join_main select t,t from generate_series(1,1000000) as t;

insert into t_join_detail select t,t from generate_series(1,1000000) as t;

select * from t_join_main,t_join_detail where t_join_main.id=t_join_detail.id

and t_join_main.id=1;

select * from t_join_main,t_join_detail where

t_join_main.id=t_join_detail.cardid and t_join_main.id=1;

select * from t_join_main,t_join_detail where

t_join_main.cardid=t_join_detail.cardid and t_join_detail.cardid=2;
```

上面几个join查询对应的字段都要建立索引

```
create index t_join_detail_cardid_idx on t_join_detail(cardid);

create index t_join_main_cardid_idx on t_join_main(cardid);
```

数组gin索引

创建方法

```
create table t_array(id int not null, cardid varchar(32)[],primary key(id));

insert into t_array select t,('{'||md5(t::text)||'}')::text[] from
generate_series(1,1000000) as t;

insert into t_array values(1000001,'{1,2,3}');
insert into t_array values(1000002,'{1,2,3,4,5}');
create index t_array_cardid_idx on t_array using gin(cardid);
```

使用方法

```
select * from t_array where cardid @>
('{c4ca4238a0b923820dcc509a6f75849b}')::text[];

select * from t_array where cardid @> ('{1,2}')::text[];

select * from t_array where cardid @> ('{1,2,5}')::text[];
```

jsonb使用索引

创建方法

```
create table t_jsonb(id int not null,f_jsonb jsonb,primary key(id));

create index t_jsonb_f_jsonb_idx on t_jsonb using gin(f_jsonb);
```

使用方法

```
select * from t_jsonb where f_jsonb @> '{"phone":"81838898"}';
```

pg_trgm全模糊走索引

创建方法

```
create extension pg_trgm;

create table t_trgm(id int not null,cardid varchar(500),primary key(id));

create index t_trgm_cardid_idx on t_trgm using gin(cardid gin_trgm_ops);
```

使用方法

```
select * from t_trgm where cardid like '%440521%';
```

使用限制

- 数据库排序规则需要使用lc_collate 'zh_CN.utf8';
- 查询字符不能少于3个
- 存储的内容不能大于8k

任意字段都可使用索引

创建方法

```
create table gin_mul(f1 int, f2 int, f3 timestamp, f4 text, f5 numeric, f6
text);
create extension btree_gin;
create index gin_mul_gin_idx on gin_mul using gin(f1,f2,f3,f4,f5,f6);
```

使用方法

这样建立索引后，这个表的任意字段就能组合使用索引查询，如

```
select * from gin_mul where f1=xx;
select * from gin_mul where f3=xx;
select * from gin_mul where f1=xx and f2=xx;
select * from gin_mul where f1=xx and f2=xx and f5=xxx;
```

使用注意

btree_gin索引的容量一般是数据表2倍左右，索引的维护成本非常大，适合于更新少，需要任意字段组合查询使用场景。

如何检查查询是否使用了索引

```
postgres=# explain select * from t1 where f1=1;
QUERY PLAN
------------------------------------------------------------------------
Remote Fast Query Execution (cost=0.00..0.00 rows=0 width=0)
Node/s: dn001
-> Index Scan using t1_f1_idx on t1 (cost=0.42..4.44 rows=1 width=8)
Index Cond: (f1 = 1)
```

```
(4 rows)

postgres=#

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=31.40..3174.64 rows=2503 width=61)

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

-> Bitmap Index Scan on t_array_mc_idx (cost=0.00..30.78 rows=2503 width=0)

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

(6 rows)


postgres=# explain select * from t_join_1,t_join_2 where

t_join_1.f1=t_join_2.f2 and t_join_1.f1=1;

QUERY PLAN

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

Nested Loop (cost=0.25..8.29 rows=1 width=16)

-> Remote Subquery Scan on all (dn001) (cost=100.12..104.16 rows=1 width=8)

-> Index Scan using t_join_1_f1_idx on t_join_1 (cost=0.12..4.14 rows=1

width=8)

Index Cond: (f1 = 1)

-> Materialize (cost=100.12..104.16 rows=1 width=8)

-> Remote Subquery Scan on all (dn001,dn002) (cost=100.12..104.16 rows=1

width=8)

-> Index Scan using t_join_2_f2_idx on t_join_2 (cost=0.12..4.14 rows=1

width=8)
```

```
Index Cond: (f2 = 1)
(8 rows)
postgres=# explain select * from t_join_1,t_join_2 where
t_join_1.f1=t_join_2.f1 and t_join_1.f1=1;
QUERY PLAN
------------------------------------------------------------------------------------------
Remote Fast Query Execution (cost=0.00..0.00 rows=0 width=0)
Node/s: dn001, dn002
-> Nested Loop (cost=14.51..1359.15 rows=989 width=16)
-> Index Scan using t_join_1_f1_idx on t_join_1 (cost=0.42..4.44 rows=1
width=8)
Index Cond: (f1 = 1)
-> Bitmap Heap Scan on t_join_2 (cost=14.09..1344.82 rows=989 width=8)
Recheck Cond: (f1 = 1)
-> Bitmap Index Scan on t_join_2_f1_idx (cost=0.00..13.84 rows=989 width=0)
Index Cond: (f1 = 1)
(9 rows)
```

增删改查使用原则

insert

一次插入多条记录

```
insert into tbase(id,nickname) values(1,'hello TBase'),(2,'TBase hello');
```

备注：一次性提交效率更高

插入记录存在时更新

```
insert into tbase values(1,'pgxz') ON CONFLICT (id) DO UPDATE SET nickname =
'tbase';
```

插入记录，记录存在时变成更新，使用ON CONFLICT

2.8.1.3 插入记录后，查询返回新增的记录

```
insert into tbase(nickname) values(1,'TBase') returning *;
```

使用returning相当insert+select操作

delete

分布键做为删除条件效率高，扩展性好

不使用分布键 ，则删除语句发往所有节点。

删除分区表数据

删除分区表数据除了带分布键做为删除条件外，还要带上分区键，否则会扫描所有分区表。

删除也要考虑使用索引

带索引的字段做为过滤条件，往往执行效率更高。

使用returning需要返回数据

```
postgres=# delete from tbase where id=2 returning *;
id | nickname
----+-----------
2 | TBase
(1 row)
```

两表关联删除只支持分布键关联

```
delete from tbase using t_appoint_col where tbase.id=t_appoint_col.id;
```

这里 t_appoint_col.id和tbase.id 必需都是分布键

删除所有数据请使用truncate

```
truncate table tbase;
truncate
```

效率高，而且能直接释放空间，但对于做主主复制的系统，需要两个集群各自操作一次。另外还要考虑数据同步到kafka的问题。

update

分布键不支持更新

```
update tbase set id=8 where id=1;
ERROR: Distribute column or partition column can't be updated in current version
```

分区表的分区键不支持更新

```
update t_time_range set inserttime='2017-09-02';
ERROR: Distributed column or partition column "inserttime" can't be updated in
current version
```

更新并返回记录请使用returning

```
update tbase set nickname = nickname where id = (random()*2)::integer returning
*;
id | nickname
----+-----------
2 | TBase
(1 row)
```

两表关联更新只支持分布键关联

```
update tbase set nickname ='Good TBase' from t_appoint_col where
t_appoint_col.id=tbase.id;
```

这里 t_appoint_col.id和tbase.id 必需都是分布键

分布键做为更新条件效率高，扩展性好

不使用分布键 ，则更新语句发往所有节点。

分区表数据更新

更新分区表数据除了带分布键做为更新条件外，还要带上分区键，否则会扫描所有分区表。

更新也要考虑使用索引

带索引的字段做为过滤条件，往往执行效率更高。

select

分布键查询效率高，扩展性好

如

```
select * from tbase where nickname='tbase'; #扩展性差
select * from tbase where nickname='tbase' and id=1; #效率高，具有良好扩展性
```

分区表查询需要带上分区时间

如 select * from t_time_range where id=1;#搜索所有分区，效率低，60个分区表，无数据，查询一次12ms select * from t_time_range where id=1 and inserttime>='2019-01-01' and inserttime<'2019-02-01';#搜索单一分区，效率高，查询一次2ms

not in 中包含了null，结果全为真

```
select * from tbase where id not in (3,5,null);
```

这样的查询返回为空，一条记录都查询不出来

尽量分页提取数据

```
select * from policy_basic order by id limit 10 offset 0;

select * from policy_basic order by id limit 10 offset 9;
```

offset 0表示第一条开始，另外 offset x只是不提出数据，但还要查询出来。

大数据抽取请使用游标

游标抽取数据效率更高。

分布式事务使用原则

tbase支持全局acid级别分布式事务，但应用设计时尽可能减少，分布式事务在最后提交时开销更大。

字段类型使用原则

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
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
double precision	8字节	可变精度，不精确	15位十进制精度

- 整数smallint,integer,bigint请按最大可能数进行选择，这样可以在存入超出范围的数字时由数据库做最后的检查保证。
- 小数字可以减少不必需的存储空间。
- 小数字减少查询时扫描的数据块。
- numeric用于需要精度准确的业务，如存储金额。
- double precision用于精度要求不高的业务，其性能要好于numeric。

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

- 如果字段将来存在变更长度的可能，则禁止使用char。
- varchar类型要指定长度使用，不指定长度的varchar与text一样可以保持1G数据。

- 禁止使用text来存储副件图片之类的数据。

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，最大存储1G数据 |

TDSQL PostgreSQL 版支持二进制数据存储，但访问效率远低于对象存储系统，无特别要求禁止使用。

日期类型

| **名字** | **存储尺寸** | **描述** | **范围** |

|--------------|--------------|----------------|-------------|

| timestamp | 8字节 | 包括日期和时间 | 精确到1微秒 |

| timestamp(0) | 8字节 | 包括日期和时间 | 精确到1秒 |

| date | 4字节 | 只包括日期 | |

| time | 8字节 | 一天中的时间 | 精确到1微秒 |

| Time(0) | 8字节 | 一天中的时间 | 精确到1秒 |

- 如果存入的数据只想精确到秒，则使用timestamp(0),time(0)

- 注意oracle转换过来的应用，oracle的date是精确到秒，对应的tbase类型为timestamp(0)

null值拼接字段串需要处理

不处理返回为null值

```
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)
```

count函数使用

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));
- 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
5 |
3 | TBase
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)
```

只取所需字段

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

过滤不必要更新

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=# update tbase_main set mc='TBase' where mc!='TBase';
UPDATE 0
```

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。

大数据量更新尽量折分

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，尽量少锁定资源，避免死锁的产生；
#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
```

禁止在业务代码中使用DDL

TDSQL PostgreSQL 版在扩容，冷热数据搬迁时DDL都会禁止执行。

跨库访问规范

所有不按分布键的查询都是属于跨库访问，需要按下面的规则来限制

- 如果查询需要将数据拉回协调节点(CN)计算，只要涉及计算数据量比较少，这样在协调节点(CN)上面的聚集也是高性能的。
- 针对于单表聚集统计之类的查询，由于可以实现查询下推，只是汇集最后的结果，这样的查询只要DN的处理能力足够，使用上也没什么问题。
- 如果需要拉回大量数据在协调节点(CN)进行计算，就需要考虑使用提前计算好的中间表，宽表，或者按业务维度分布的数据表，减少协调节点(CN)和数据节点(DN)之间的数据量传输，如果做不到，需要禁止这样的查询使用。

高可用数据源接入方式

TDSQL PostgreSQL 版的每一个cn都可以接入，看到的信息都是全局一致的，可以使用下面的方法实现高可用

- 应用程序配置多个IP+PORT数据源，轮循选择接入。

- 前置CLB，F5或者LVS，应该程序直接访问前置的VIP

防SQL注入

为了防止SQL注入，需要把传入的变量值做处理，处理方法如下，如

```
select * from tbase where nickname='$1';
```

上面的$1 参数如果传入成这样 123';delete from tbase ;select ' 就会变成这样的语句

```
select * from tbase where nickname='123';delete from tbase ;select '';
```

这样就会把数据表tbase的数据给删除掉 所以在程序中需要对传入的参数进行处理（使用扩展协议不需要处理这个问题），把参数值中的单引号'变成两个''，最终变成这样

```
select * from tbase where nickname='123'';delete from tbase ;select ''';
```

这样就变成只查询一个字符串，如下所示

```
postgres=# select * from tbase where nickname='123'';delete from tbase ;select
''';
id | nickname
----+----------
(0 rows)
postgres=# select '123'';delete from tbase ;select ''';
?column?
----------------------------------
123';delete from tbase ;select '
(1 row)
```

应用程序升级规范

概述

执行工具

TDSQL PostgreSQL 版之客户端工具psql

连接环境

如无特别约定，连接到 TDSQL PostgreSQL 版集群之任意一个CN都可以

操作方法

DDL更新

操作前环境配置

DDL需要锁表，所以执行DDL操作时都要配置拿到表锁的最大时长，特别是在线服务项目，下在配置某个节点在3秒内如果无法获取表锁就rollback

```
psql -h 172.16.0.29 -p 11000 -d postgres -U tbase
psql (PostgreSQL 10.0 TBase V2)
```

```
Type "help" for help.
postgres=# set lock_timeout to 3000;
SET
postgres=#
postgres=# alter table tbase add column phone varchar(12);
ALTER TABLE
postgres=#
```

更新超时提示

```
postgres=# alter table tbase add column phone varchar(12);

ERROR: canceling statement due to lock timeout
```

下面情况下禁止执行DDL

添加新节点，节点扩容，冷热数据搬迁，节点异常

主主复制表DDL变更准则

对于主主复制表DDL变更，两个集群DDL需要保持一致的操作顺序，每次更新后务必检查主主复制进程是否能正常同步数据。

DML更新

INSERT插入数据

直接在线执行即可。

UPDATE或者DELETE

如果是全表的UPDATE,DELETE，则需要使用分批更新法，如

UPDATE 表名 set 字段名=xxxx WHERE id>1 and id<=100 and 字段名!=xxxx ;

UPDATE 表名 set 字段名=xxxx WHERE id>100 and id<=200 and 字段名!=xxxx ;

上面语句好处就是减少行锁数，尽量做到对业务影响最小化

原子性批量更新

```
Begin;
Insert ....
Update...
...
Commit;
```

把所有更新语句合在一个事务中执行来保证更新的原子性，但要注意更新过程持有的锁时间会更长。

数据管理

数据复制

TDSQL PostgreSQL 版目前提供了三种不同级别的复制方案

流复制

流复制用于保证数据节点DN节点高可用和异地容灾使用。同步的级别分别同步和异步，同时支持配置多个备机同步。 部署建议

1. 建议配置最少一主一备同步，由于同步在备机异常时会卡住主节点写入，所以需要准备一个备机用于同步备机异常时做为后备接入，即一个IDC为1主2备。

2. 如果有同城IDC，要求网络的延迟在3ms以内，则同城的另一个IDC可以部署一个同步备机。

3. 异地IDC备节点只能使用异步同步方式。

   以上数据同步复制只支持同集群同构同版本。

逻辑复制

对于不同 TDSQL PostgreSQL 版集群之间数据复制， TDSQL PostgreSQL 版提供了表级逻辑复制能力，逻辑复制可以实现A集群的A表在数据发生变化时同步至B集群，使用逻辑复制注意点。

- 同步的数据表需要有主键。
- 逻辑复制在数据约束冲突时会导致复制服务停止。
- 逻辑复制不支持强同步复制。
- 修改字段名，删除字段的DDL操作需要确保无增量数据需要同步，否则会导致增量数据无法同步。
- 逻辑复制的数据表做DDL维护时两边需要保持一致。
- 逻辑复制只支持INSERT,DELETE,UPDATE操作的数据变更，对于COPY,TRUNCATE是无法进行逻辑解释。

逻辑解释至kafka

TDSQL PostgreSQL 版支持将增加的数据逻辑解释同步到kafka集群，其它需要同步数据的服务再通过订阅kafka的方式来实现数据同步。

冷热数据分离

TDSQL PostgreSQL 版支持集群内使用异构机型，机型之间配置差异化，满足高性能的同时，节约业务存储成本和冷热数据访问隔离。TDSQL PostgreSQL 版目前支持按月度划分数据分区，从而在业务规划时，可以将频繁访问月份的数据存储在高IO性能机型（例如SSD磁盘），将非频繁访问数据存储在大存储容量机型（例如SAS磁盘）。 冷热分区表使用注意事项： 冷热分区表需要在建立数据表时就定义好，普通表无法修改成冷热分区表。

故障处理

最近更新时间: 2024-06-12 15:06:00

1.节点服务启动或者停止失败排查方法

从监控平台上或者脚本操作节点服务失败的排查方法。

1.1 检查磁盘空间是否足够

上面的Avail表示还有多少空间可用，如果空间不够的话会影响tbase服务的运行

1.2 查看故障节点目录中pg_log目录最后一个日志文件日志内容

输入 ll -rt | tail -n 10，最示在最后一个文件就是最新的日志文件，使用vim打开，按shift + g移到文件内容最后面，按日志提示出错解决问题即可。

2.取消掉某些查询方法

有时应用写错了sql语句，或者程序bug引发大量不需要的查询并发执行，占用了大量系统资源，这时需要dba使用工具取消掉这些正在执行的查询。

2.1 查询要取消进程pid

```
postgres=# select * from pg_stat_activity where datid is not null and
pid!=pg_backend_pid() limit 1;
-[ RECORD 1 ]----+------------------------------
datid | 13325
datname | postgres
pid | 3981
usesysid | 10
usename | tbase
application_name | psql
client_addr | 127.0.0.1
client_hostname |
client_port | 56254
backend_start | 2018-08-14 19:33:12.235117+08
xact_start |
query_start | 2018-08-14 19:33:26.458043+08
state_change | 2018-08-14 19:33:26.60002+08
wait_event_type | Client
wait_event | ClientRead
state | idle
backend_xid |
backend_xmin |
query | select count(1) from t;
backend_type | client backend
```

2.2 取消某个节点上面的查询

```
postgres=# select pg_cancel_backend(3981)
```

2.3 取消所有cn节点上面的查询

```
[tbase@VM_0_37_centos shell]$./tbase_run_sql_cn.sh "select
pg_cancel_backend(pid) from pg_stat_activity where query like '%update t set
mc%' and pid!=pg_backend_pid()"
```

你可以先执行

```
./tbase_run_sql_cn.sh "select pid,query from pg_stat_activity where query like
'%update t set mc%' and pid!=pg_backend_pid()"
```

确认要取消进程是否正确

2.4 取消所有dn节点上面的查询

```
[tbase@VM_0_37_centos shell]$./tbase_run_sql_dn_master.sh "select
pg_cancel_backend(pid) from pg_stat_activity where query like '%update t set
mc%' and pid!=pg_backend_pid()"
```

你可以先执行

```
./tbase_run_sql_dn_master.sh "select pid,query from pg_stat_activity where query
like '%update t set mc%' and pid!=pg_backend_pid()"
```

确认要取消进程是否正确

使用执行pg_cancel_backend()有时进程不会响应，这时需要使用pg_terminate_backend()来杀死某个进程

3 kill掉某些进程方法

3.1 查到要kill掉进程pid

```
postgres=# select * from pg_stat_activity where datid is not null and
pid!=pg_backend_pid() limit 1;
-[ RECORD 1 ]----+------------------------------
datid | 13325
datname | postgres
pid | 3981
usesysid | 10
usename | tbase
application_name | psql
client_addr | 127.0.0.1
client_hostname |
client_port | 56254
```

```
backend_start | 2018-08-14 19:33:12.235117+08
xact_start |
query_start | 2018-08-14 19:33:26.458043+08
state_change | 2018-08-14 19:33:26.60002+08
wait_event_type | Client
wait_event | ClientRead
state | idle
backend_xid |
backend_xmin |
query | select count(1) from t;
backend_type | client backend
```

3.2 kill某个节点上面的查询

```
postgres=# select pg_terminate_backend(3981)
```

3.3 kill所有cn节点上面的查询

```
[tbase@VM_0_37_centos shell]$./tbase_run_sql_cn.sh "select

pg_terminate_backend(pid) from pg_stat_activity where query like '%update t set

mc%' and pid!=pg_backend_pid()"
```

你可以先执行

```
[tbase@VM_0_37_centos shell]$./tbase_run_sql_cn.sh "select pid,query from

pg_stat_activity where query like '%update t set mc%' and pid!=pg_backend_pid()"
```

确认要kill进程是否正确

3.4 kill所有dn节点上面的查询

```
[tbase@VM_0_37_centos shell]$./tbase_run_sql_dn_master.sh "select

pg_terminate_backend(pid) from pg_stat_activity where query like '%update t set

mc%' and pid!=pg_backend_pid()"
```

你可以先执行

```
[tbase@VM_0_37_centos shell]$./tbase_run_sql_dn_master.sh "select pid,query

from pg_stat_activity where query like '%update t set mc%' and

pid!=pg_backend_pid()"
```

确认要kill进程是否正确

4.执行ddl或者更新数据卡住排查

4.1 检查dn主备节点的复制同步级别

使用下面语句检查主备同步模式

```
[tbase@VM_0_37_centos shell]$./tbase_run_sql_dn_master.sh "select
current_setting('synchronous_commit') as
synchronous_commit,current_setting('synchronous_standby_names') as
synchronous_standby_names,array((select sync_state from pg_stat_replication ))
as sync_state"

synchronous_commit | synchronous_standby_names | sync_state
-------------------+---------------------------+------------
local | | {async}
(1 row)
```

如果返回值为synchronous_commit = on、synchronous_standby_names

不为空则表示主备为同步复制模式，这时sync_state值必需为sync，否则无法执行ddl或者更新的dml操作

解决问题方法

如果 sync_state值为空则表示备机服务没拉起业，把备机服务拉起来即可，或者

1. 按8.1.2配置主备复制为某智能模式
2. 把synchronous_commit 值配置为local
3. roload dn主服务

4.2 检查cn或者dn节点是否有2pc残留

使用下面语句检查cn上是否有2pc存在

```
[tbase@VM_0_37_centos shell]$./tbase_run_sql_cn.sh "select gid from
pg_catalog.pg_prepared_xacts"
psql -h 172.16.0.37 -p 15432 -d postgres -U pgxz -c "select gid from
pg_catalog.pg_prepared_xacts"
gid
-----------
tbase_2pc
(1 row)
psql -h 172.16.0.42 -p 15432 -d postgres -U pgxz -c "select gid from
pg_catalog.pg_prepared_xacts"
gid
-----
(0 rows)
```

使用下面语句检查dn上是否有2pc存在

```
[tbase@VM_0_37_centos shell]$./tbase_run_sql_dn_master.sh "select gid from
pg_catalog.pg_prepared_xacts"
psql -h 172.16.0.37 -p 23001 -d postgres -U pgxz -c "select gid from
pg_catalog.pg_prepared_xacts"
gid
-----
(0 rows)
psql -h 172.16.0.42 -p 23002 -d postgres -U pgxz -c "select gid from
pg_catalog.pg_prepared_xacts"
gid
```

```
-----------
tbase_2pc
(1 row)
```

解决问题方法

commit或者rollback 影响的2pc残留，清理cn 2pc残留方法

```
rollback cn 2pc ：./tbase_cn_2pc_rollback.sh
commit cn 2pc ：./tbase_cn_2pc_commit.sh
rollback dn 2pc ：./tbase_dn_2pc_rollback.sh
commit dn 2pc ：./tbase_dn_2pc_commit.sh
```

4.3 排它锁影响

使用下面语句查询各个cn或者dn节点上面是否运行相关锁语句

检查cn的语句

```
[tbase@VM_0_37_centos shell]$./tbase_run_sql_cn.sh "select * from

pg_stat_activity where query ilike '%t_time_range%' and pid!=pg_backend_pid()"
```

检查dn的语句

```
[tbase@VM_0_37_centos shell]$ ./tbase_run_sql_dn_master.sh "select * from

pg_stat_activity where query ilike '%t_time_range%' and pid!=pg_backend_pid()"
```

上面的t_time_range为资源关键字，可以是表名，索引名。。。

解决问题方法

- **使用pg_cancel_backend函数把查询取消掉**

  在cn上执行

  ```
  [tbase@VM_0_37_centos shell]$./tbase_run_sql_cn.sh "select

  pg_cancel_backend(pid) from pg_stat_activity where query like '%t_time_range%'

  and pid!=pg_backend_pid()"
  ```

  在dn上面执行

  ```
  [tbase@VM_0_37_centos shell]$./tbase_run_sql_dn_master.sh "select

  pg_cancel_backend(pid) from pg_stat_activity where query like '%t_time_range%'

  and pid!=pg_backend_pid()"
  ```

- **使用pg_terminate_backend函数把进程kill掉**

在cn上执行

```
[tbase@VM_0_37_centos shell]$./tbase_run_sql_cn.sh "select
pg_terminate_backend(pid) from pg_stat_activity where query like
'%t_time_range%' and pid!=pg_backend_pid()"
```

在dn上面执行

```
[tbase@VM_0_37_centos shell]$./tbase_run_sql_dn_master.sh "select
pg_terminate_backend(pid) from pg_stat_activity where query like
'%t_time_range%' and pid!=pg_backend_pid()"
```

API文档
TBase部署中心（tbase）
版本（2019-01-07）
API概览

最近更新时间: 2024-06-18 14:31:25

API版本

V3

TBase部署中心

接口名称	接口功能
DescribeAvailableRegion	查询售卖地域信息

其他接口

接口名称	接口功能
ModifyDBInstanceSecurityGroups	修改云数据库安全组

调用方式
接口签名v1

最近更新时间: 2024-06-18 14:31:25

tcecloud API 会对每个访问请求进行身份验证，即每个请求都需要在公共请求参数中包含签名信息（Signature）以验证请求者身份。 签名信息由安全凭证生成，安全凭证包括 SecretId 和 SecretKey；若用户还没有安全凭证，请前往云API密钥页面申请，否则无法调用云API接口。

1. 申请安全凭证

在第一次使用云API之前，请前往云API密钥页面申请安全凭证。 安全凭证包括 SecretId 和 SecretKey：

- SecretId 用于标识 API 调用者身份
- SecretKey 用于加密签名字符串和服务器端验证签名字符串的密钥。
- 用户必须严格保管安全凭证，避免泄露。

申请安全凭证的具体步骤如下：

1. 登录tcecloud管理中心控制台。
2. 前往云API密钥的控制台页面
3. 在云API密钥页面，点击【新建】即可以创建一对SecretId/SecretKey

注意：开发商帐号最多可以拥有两对 SecretId / SecretKey。

2. 生成签名串

有了安全凭证SecretId 和 SecretKey后，就可以生成签名串了。以下是生成签名串的详细过程：

假设用户的 SecretId 和 SecretKey 分别是：

- SecretId: AKIDz8krbsJ5yKBZQpn74WFkmLPx3EXAMPLE
- SecretKey: Gu5t9xGARNpq86cd98joQYCN3EXAMPLE

注意：这里只是示例，请根据用户实际申请的 SecretId 和 SecretKey 进行后续操作！

以云服务器查看实例列表(DescribeInstances)请求为例，当用户调用这一接口时，其请求参数可能如下:

参数名称	中文	参数值
Action	方法名	DescribeInstances
SecretId	密钥Id	AKIDz8krbsJ5yKBZQpn74WFkmLPx3EXAMPLE
Timestamp	当前时间戳	1465185768
Nonce	随机正整数	11886
Region	实例所在区域	ap-guangzhou

参数名称	中文	参数值
InstanceIds.0	待查询的实例ID	ins-09dx96dg
Offset	偏移量	0
Limit	最大允许输出	20
Version	接口版本号	2017-03-12

2.1. 对参数排序

首先对所有请求参数按参数名的字典序（ ASCII 码）升序排序。注意：1）只按参数名进行排序，参数值保持对应即可，不参与比大小；2）按 ASCII 码比大小，如 InstanceIds.2 要排在 InstanceIds.12 后面，不是按字母表，也不是按数值。用户可以借助编程语言中的相关排序函数来实现这一功能，如 php 中的 ksort 函数。上述示例参数的排序结果如下:

```
{
'Action' : 'DescribeInstances',
'InstanceIds.0' : 'ins-09dx96dg',
'Limit' : 20,
'Nonce' : 11886,
'Offset' : 0,
'Region' : 'ap-guangzhou',
'SecretId' : 'AKIDz8krbsJ5yKBZQpn74WFkmLPx3EXAMPLE',
'Timestamp' : 1465185768,
'Version': '2017-03-12',
}
```

使用其它程序设计语言开发时，可对上面示例中的参数进行排序，得到的结果一致即可。

2.2. 拼接请求字符串

此步骤生成请求字符串。 将把上一步排序好的请求参数格式化成"参数名称"="参数值"的形式，如对 Action 参数，其参数名称为 "Action" ，参数值为 "DescribeInstances" ，因此格式化后就为 Action=DescribeInstances 。 **注意：“参数值”为原始值而非url编码后的值。**

然后将格式化后的各个参数用"&"拼接在一起，最终生成的请求字符串为:

```
Action=DescribeInstances&InstanceIds.0=ins-09dx96dg&Limit=20&Nonce=11886&Offset=0&Region=ap-guangzhou&SecretId=AKIDz8krbsJ5yKBZQpn74WFkmLPx3EXAMPLE&Timestamp=1465185768&Version=2017-03-12
```

2.3. 拼接签名原文字符串

此步骤生成签名原文字符串。 签名原文字符串由以下几个参数构成:

1. 请求方法: 支持 POST 和 GET 方式，这里使用 GET 请求，注意方法为全大写。
2. 请求主机:查看实例列表(DescribeInstances)的请求域名为：cvm.finance.cloud.tencent.com。实际的请求域名根据接口所属模块的不同而不同，详见各接口说明。
3. 请求路径: 当前版本云API的请求路径固定为 / 。
4. 请求字符串: 即上一步生成的请求字符串。

签名原文串的拼接规则为: 请求方法 + 请求主机 +请求路径 + ? + 请求字符串

示例的拼接结果为：

```
GETcvm.finance.cloud.tencent.com/?Action=DescribeInstances&InstanceIds.0=ins-09dx96dg&Limit=20&Nonce=11886&Offset=0&Region=ap-guangzhou&SecretId=AKIDz8krbsJ5yKBZQpn74WFkmLPx3EXAMPLE&Timestamp=1465185768&Version=2017-03-12
```

2.4. 生成签名串

此步骤生成签名串。 首先使用 HMAC-SHA1 算法对上一步中获得的**签名原文字符串**进行签名，然后将生成的签名串使用 Base64 进行编码，即可获得最终的签名串。

具体代码如下，以 PHP 语言为例:

```
$secretKey = 'Gu5t9xGARNpq86cd98joQYCN3EXAMPLE';
$srcStr = 'GETcvm.finance.cloud.tencent.com/?Action=DescribeInstances&InstanceIds.0=ins-09dx96dg&Limit=20&Nonce=11886&Offset=0&Region=ap-guangzhou&SecretId=AKIDz8krbsJ5yKBZQpn74WFkmLPx3EXAMPLE&Timestamp=1465185768&Version=2017-03-12';
$signStr = base64_encode(hash_hmac('sha1', $srcStr, $secretKey, true));
echo $signStr;
```

最终得到的签名串为:

```
EliP9YW3pW28FpsEdkXt/+WcGeI=
```

使用其它程序设计语言开发时，可用上面示例中的原文进行签名验证，得到的签名串与例子中的一致即可。

3. 签名串编码

生成的签名串并不能直接作为请求参数，需要对其进行 URL 编码。

如上一步生成的签名串为 EliP9YW3pW28FpsEdkXt/+WcGeI= ，最终得到的签名串请求参数（Signature）为：EliP9YW3pW28FpsEdkXt%2f%2bWcGeI%3d，它将用于生成最终的请求 URL。

注意：如果用户的请求方法是 GET，或者请求方法为 POST 同时 Content-Type 为 application/x-www-form-urlencoded，则发送请求时所有请求参数的值均需要做 URL 编码，参数键和=符号不需要编码。非 ASCII 字符在 URL 编码前需要先以 UTF-8 进行编码。

注意：有些编程语言的 http 库会自动为所有参数进行 urlencode，在这种情况下，就不需要对签名串进行 URL 编码了，否则两次 URL 编码会导致签名失败。

注意：其他参数值也需要进行编码，编码采用 RFC 3986。使用 %XY 对特殊字符例如汉字进行百分比编码，其中"X"和"Y"为十六进制字符（0-9 和大写字母 A-F），使用小写将引发错误。

4. 签名失败

根据实际情况，存在以下签名失败的错误码，请根据实际情况处理

错误代码	错误描述
AuthFailure.SignatureExpire	签名过期
AuthFailure.SecretIdNotFound	密钥不存在
AuthFailure.SignatureFailure	签名错误

错误代码	错误描述
AuthFailure.TokenFailure	token 错误
AuthFailure.InvalidSecretId	密钥非法（不是云 API 密钥类型）

5. 签名演示

在实际调用 API 3.0 时，推荐使用配套的tcecloud SDK 3.0 ，SDK 封装了签名的过程，开发时只关注产品提供的具体接口即可。详细信息参见 SDK 中心。当前支持的编程语言有：

- Python
- Java
- PHP
- Go
- JavaScript
- .NET

为了更清楚的解释签名过程，下面以实际编程语言为例，将上述的签名过程具体实现。请求的域名、调用的接口和参数的取值都以上述签名过程为准，代码只为解释签名过程，并不具备通用性，实际开发请尽量使用 SDK 。

最终输出的 url 可能为： `http://imgcache.finance.cloud.tencent.com:80cvm.finance.cloud.tencent.com/?Action=DescribeInstances&InstanceIds.0=ins-09dx96dg&Limit=20&Nonce=11886&Offset=0&Region=ap-guangzhou&SecretId=AKIDz8krbsJ5yKBZQpn74WFkmLPx3EXAMPLE&Signature=EliP9YW3pW28FpsEdkXt%2F%2BWcGeI%3D&Timestamp=1465185768&Version=2017-03-12`

注意：由于示例中的密钥是虚构的，时间戳也不是系统当前时间，因此如果将此 url 在浏览器中打开或者用 curl 等命令调用时会返回鉴权错误：签名过期。为了得到一个可以正常返回的 url ，需要修改示例中的 SecretId 和 SecretKey 为真实的密钥，并使用系统当前时间戳作为 Timestamp 。

注意：在下面的示例中，不同编程语言，甚至同一语言每次执行得到的 url 可能都有所不同，表现为参数的顺序不同，但这并不影响正确性。只要所有参数都在，且签名计算正确即可。

注意：以下代码仅适用于 API 3.0，不能直接用于其他的签名流程，即使是旧版的 API ，由于存在细节差异也会导致签名计算错误，请以对应的实际文档为准。

Java

```
import java.io.UnsupportedEncodingException;
import java.net.URLEncoder;
import java.util.Random;
import java.util.TreeMap;
import javax.crypto.Mac;
import javax.crypto.spec.SecretKeySpec;
import javax.xml.bind.DatatypeConverter;

public class TceCloudAPIDemo {
private final static String CHARSET = "UTF-8";

public static String sign(String s, String key, String method) throws Exception {
Mac mac = Mac.getInstance(method);
```

```
SecretKeySpec secretKeySpec = new SecretKeySpec(key.getBytes(CHARSET), mac.getAlgorithm());
mac.init(secretKeySpec);
byte[] hash = mac.doFinal(s.getBytes(CHARSET));
return DatatypeConverter.printBase64Binary(hash);
}

public static String getStringToSign(TreeMap<String, Object> params) {
StringBuilder s2s = new StringBuilder("GETcvm.finance.cloud.tencent.com/?");
// 签名时要求对参数进行字典排序，此处用TreeMap保证顺序
for (String k : params.keySet()) {
s2s.append(k).append("=").append(params.get(k).toString()).append("&");
}
return s2s.toString().substring(0, s2s.length() - 1);
}

public static String getUrl(TreeMap<String, Object> params) throws UnsupportedEncodingException {
StringBuilder url = new StringBuilder("http://imgcache.finance.cloud.tencent.com:80cvm.finance.cloud.tencent.com/?");
// 实际请求的url中对参数顺序没有要求
for (String k : params.keySet()) {
// 需要对请求串进行urlencode，由于key都是英文字母，故此处仅对其value进行urlencode
url.append(k).append("=").append(URLEncoder.encode(params.get(k).toString(), CHARSET)).append("&");
}
return url.toString().substring(0, url.length() - 1);
}

public static void main(String[] args) throws Exception {
TreeMap<String, Object> params = new TreeMap<String, Object>(); // TreeMap可以自动排序
// 实际调用时应当使用随机数，例如：params.put("Nonce", new Random().nextInt(java.lang.Integer.MAX_VALUE));
params.put("Nonce", 11886); // 公共参数
// 实际调用时应当使用系统当前时间，例如： params.put("Timestamp", System.currentTimeMillis() / 1000);
params.put("Timestamp", 1465185768); // 公共参数
params.put("SecretId", "AKIDz8krbsJ5yKBZQpn74WFkmLPx3EXAMPLE"); // 公共参数
params.put("Action", "DescribeInstances"); // 公共参数
params.put("Version", "2017-03-12"); // 公共参数
params.put("Region", "ap-guangzhou"); // 公共参数
params.put("Limit", 20); // 业务参数
params.put("Offset", 0); // 业务参数
params.put("InstanceIds.0", "ins-09dx96dg"); // 业务参数
params.put("Signature", sign(getStringToSign(params), "Gu5t9xGARNpq86cd98joQYCN3EXAMPLE", "HmacSHA1")); // 公共
参数
System.out.println(getUrl(params));
}
}
```

Python

注意：如果是在 Python 2 环境中运行，需要先安装 requests 依赖包： `pip install requests` 。

```
# -*- coding: utf8 -*-
import base64
import hashlib
import hmac
import time

import requests
```

```
secret_id = "AKIDz8krbsJ5yKBZQpn74WFkmLPx3EXAMPLE"
secret_key = "Gu5t9xGARNpq86cd98joQYCN3EXAMPLE"

def get_string_to_sign(method, endpoint, params):
s = method + endpoint + "/?"
query_str = "&".join("%s=%s" % (k, params[k]) for k in sorted(params))
return s + query_str

def sign_str(key, s, method):
hmac_str = hmac.new(key.encode("utf8"), s.encode("utf8"), method).digest()
return base64.b64encode(hmac_str)

if __name__ == '__main__':
endpoint = "cvm.finance.cloud.tencent.com"
data = {
'Action' : 'DescribeInstances',
'InstanceIds.0' : 'ins-09dx96dg',
'Limit' : 20,
'Nonce' : 11886,
'Offset' : 0,
'Region' : 'ap-guangzhou',
'SecretId' : secret_id,
'Timestamp' : 1465185768, # int(time.time())
'Version': '2017-03-12'
}
s = get_string_to_sign("GET", endpoint, data)
data["Signature"] = sign_str(secret_key, s, hashlib.sha1)
print(data["Signature"])
# 此处会实际调用，成功后可能产生计费
# resp = requests.get("http://imgcache.finance.cloud.tencent.com:80" + endpoint, params=data)
# print(resp.url)
```

接口签名v3

最近更新时间: 2024-06-18 14:31:25

tcecloud API 会对每个访问请求进行身份验证，即每个请求都需要在公共请求参数中包含签名信息（Signature）以验证请求者身份。 签名信息由安全凭证生成，安全凭证包括 SecretId 和 SecretKey；若用户还没有安全凭证，请前往云API密钥页面申请，否则无法调用云API接口。

1. 申请安全凭证

在第一次使用云API之前，请前往云API密钥页面申请安全凭证。 安全凭证包括 SecretId 和 SecretKey：

- SecretId 用于标识 API 调用者身份
- SecretKey 用于加密签名字符串和服务器端验证签名字符串的密钥。
- 用户必须严格保管安全凭证，避免泄露。

申请安全凭证的具体步骤如下：

1. 登录tcecloud管理中心控制台。
2. 前往云API密钥的控制台页面
3. 在云API密钥页面，点击【新建】即可以创建一对SecretId/SecretKey

注意：开发商帐号最多可以拥有两对 SecretId / SecretKey。

2. TC3-HMAC-SHA256 签名方法

注意：对于GET方法，只支持 Content-Type: application/x-www-form-urlencoded 协议格式。对于POST方法，目前支持 Content-Type: application/json 以及 Content-Type: multipart/form-data 两种协议格式，json 格式默认所有业务接口均支持，multipart 格式只有特定业务接口支持，此时该接口不能使用 json 格式调用，参考具体业务接口文档说明。

下面以云服务器查询广州区实例列表作为例子，分步骤介绍签名的计算过程。我们仅用到了查询实例列表的两个参数：Limit 和 Offset，使用 GET 方法调用。

假设用户的 SecretId 和 SecretKey 分别是：AKIDz8krbsJ5yKBZQpn74WFkmLPx3EXAMPLE 和 Gu5t9xGARNpq86cd98joQYCN3EXAMPLE

2.1. 拼接规范请求串

按如下格式拼接规范请求串（CanonicalRequest）：

```
CanonicalRequest =
HTTPRequestMethod + '\n' +
CanonicalURI + '\n' +
CanonicalQueryString + '\n' +
CanonicalHeaders + '\n' +
SignedHeaders + '\n' +
HashedRequestPayload
```

- HTTPRequestMethod：HTTP 请求方法（GET、POST ），本示例中为 GET；

- CanonicalURI：URI 参数，API 3.0 固定为正斜杠（/）；
- CanonicalQueryString：发起 HTTP 请求 URL 中的查询字符串，对于 POST 请求，固定为空字符串，对于 GET 请求，则为 URL 中问号（?）后面的字符串内容，本示例取值为：Limit=10&Offset=0。注意：CanonicalQueryString 需要经过 URL 编码。
- CanonicalHeaders：参与签名的头部信息，至少包含 host 和 content-type 两个头部，也可加入自定义的头部参与签名以提高自身请求的唯一性和安全性。拼接规则：1）头部 key 和 value 统一转成小写，并去掉首尾空格，按照 key:value\n 格式拼接；2）多个头部，按照头部 key（小写）的字典排序进行拼接。此例中为：`content-type:application/x-www-form-urlencoded\nhost:cvm.finance.cloud.tencent.com\n`
- SignedHeaders：参与签名的头部信息，说明此次请求有哪些头部参与了签名，和 CanonicalHeaders 包含的头部内容是一一对应的。content-type 和 host 为必选头部。拼接规则：1）头部 key 统一转成小写；2）多个头部 key（小写）按照字典排序进行拼接，并且以分号（;）分隔。此例中为：`content-type;host`
- HashedRequestPayload：请求正文的哈希值，计算方法为 Lowercase(HexEncode(Hash.SHA256(RequestPayload)))，对 HTTP 请求整个正文 payload 做 SHA256 哈希，然后十六进制编码，最后编码串转换成小写字母。注意：对于 GET 请求，RequestPayload 固定为空字符串，对于 POST 请求，RequestPayload 即为 HTTP 请求正文 payload。

根据以上规则，示例中得到的规范请求串如下（为了展示清晰，\n 换行符通过另起打印新的一行替代）：

```
GET
/
Limit=10&Offset=0
content-type:application/x-www-form-urlencoded
host:cvm.finance.cloud.tencent.com

content-type;host
e3b0c44298fc1c149afbf4c8996fb92427ae41e4649b934ca495991b7852b855
```

2.2. 拼接待签名字符串

按如下格式拼接待签名字符串：

```
StringToSign =
Algorithm + \n +
RequestTimestamp + \n +
CredentialScope + \n +
HashedCanonicalRequest
```

- Algorithm：签名算法，目前固定为 TC3-HMAC-SHA256；
- RequestTimestamp：请求时间戳，即请求头部的 X-TC-Timestamp 取值，如上示例请求为 1539084154；
- CredentialScope：凭证范围，格式为 Date/service/tc3_request，包含日期、所请求的服务和终止字符串（tc3_request）。**Date 为 UTC 标准时间的日期，取值需要和公共参数 X-TC-Timestamp 换算的 UTC 标准时间日期一致**；service 为产品名，必须与调用的产品域名一致，例如 cvm。如上示例请求，取值为 2018-10-09/cvm/tc3_request；
- HashedCanonicalRequest：前述步骤拼接所得规范请求串的哈希值，计算方法为 Lowercase(HexEncode(Hash.SHA256(CanonicalRequest)))。

注意：

> 1. Date 必须从时间戳 X-TC-Timestamp 计算得到，且时区为 UTC+0。如果加入系统本地时区信息，例如东八区，将导致白天和晚上调用成功，但是凌晨时调用必定失败。假设时间戳为 1551113065，在东八区的时间是 2019-02-26 00:44:25，但是计算得到的 Date 取 UTC+0 的日期应为 2019-02-25，而不是 2019-02-26。

> 2. Timestamp 必须是当前系统时间，且需确保系统时间和标准时间是同步的，如果相差超过五分钟则必定失败。如果长时间不和标准时间同步，可能导致运行一段时间后，请求必定失败（返回签名过期错误）。

根据以上规则，示例中得到的待签名字符串如下（为了展示清晰，\n 换行符通过另起打印新的一行替代）：

```
TC3-HMAC-SHA256
1539084154
2018-10-09/cvm/tc3_request
91c9c192c14460df6c1ffc69e34e6c5e90708de2a6d282cccf957dbf1aa7f3a7
```

2.3. 计算签名

1）计算派生签名密钥，伪代码如下

```
SecretKey = "Gu5t9xGARNpq86cd98joQYCN3EXAMPLE"
SecretDate = HMAC_SHA256("TC3" + SecretKey, Date)
SecretService = HMAC_SHA256(SecretDate, Service)
SecretSigning = HMAC_SHA256(SecretService, "tc3_request")
```

- SecretKey：原始的 SecretKey；
- Date：即 Credential 中的 Date 字段信息，如上示例，为2018-10-09；
- Service：即 Credential 中的 Service 字段信息，如上示例，为 cvm；

2）计算签名，伪代码如下

```
Signature = HexEncode(HMAC_SHA256(SecretSigning, StringToSign))
```

- SecretSigning：即以上计算得到的派生签名密钥；
- StringToSign：即步骤2计算得到的待签名字符串；

2.4. 拼接 Authorization

按如下格式拼接 Authorization：

```
Authorization =
Algorithm + ' ' +
'Credential=' + SecretId + '/' + CredentialScope + ', ' +
'SignedHeaders=' + SignedHeaders + ', '
'Signature=' + Signature
```

- Algorithm：签名方法，固定为 TC3-HMAC-SHA256；
- SecretId：密钥对中的 SecretId；
- CredentialScope：见上文，凭证范围；
- SignedHeaders：见上文，参与签名的头部信息；
- Signature：签名值

根据以上规则，示例中得到的值为：

```
TC3-HMAC-SHA256 Credential=AKIDEXAMPLE/Date/service/tc3_request, SignedHeaders=content-type;host, Signature=5da7a33f6993f0614b047e5df4582db9e9bf4672ba50567dba16c6ccf174c474
```

最终完整的调用信息如下：

```
http://imgcache.finance.cloud.tencent.com:80cvm.finance.cloud.tencent.com/?Limit=10&Offset=0

Authorization: TC3-HMAC-SHA256 Credential=AKIDz8krbsJ5yKBZQpn74WFkmLPx3EXAMPLE/2018-10-09/cvm/tc3_request, SignedHeaders=content-type;host, Signature=5da7a33f6993f0614b047e5df4582db9e9bf4672ba50567dba16c6ccf174c474
Content-Type: application/x-www-form-urlencoded
Host: cvm.finance.cloud.tencent.com
X-TC-Action: DescribeInstances
X-TC-Version: 2017-03-12
X-TC-Timestamp: 1539084154
X-TC-Region: ap-guangzhou
```

3. 签名失败

根据实际情况，存在以下签名失败的错误码，请根据实际情况处理

错误代码	错误描述
AuthFailure.SignatureExpire	签名过期
AuthFailure.SecretIdNotFound	密钥不存在
AuthFailure.SignatureFailure	签名错误
AuthFailure.TokenFailure	token 错误
AuthFailure.InvalidSecretId	密钥非法（不是云 API 密钥类型）

4. 签名演示

Java

```
import java.io.BufferedReader;
import java.io.InputStream;
import java.io.InputStreamReader;
import java.net.URL;
import java.text.SimpleDateFormat;
import java.util.Date;
import java.util.Map;
import java.util.TimeZone;
import java.util.TreeMap;
import javax.crypto.Mac;
import javax.crypto.spec.SecretKeySpec;
import javax.net.ssl.HttpsURLConnection;
import javax.xml.bind.DatatypeConverter;

import org.apache.commons.codec.digest.DigestUtils;

public class TceCloudAPITC3Demo {
private final static String CHARSET = "UTF-8";
```

```
private final static String ENDPOINT = "cvm.finance.cloud.tencent.com";
private final static String PATH = "/";
private final static String SECRET_ID = "AKIDz8krbsJ5yKBZQpn74WFkmLPx3EXAMPLE";
private final static String SECRET_KEY = "Gu5t9xGARNpq86cd98joQYCN3EXAMPLE";
private final static String CT_X_WWW_FORM_URLENCODED = "application/x-www-form-urlencoded";
private final static String CT_JSON = "application/json";
private final static String CT_FORM_DATA = "multipart/form-data";

public static byte[] sign256(byte[] key, String msg) throws Exception {
Mac mac = Mac.getInstance("HmacSHA256");
SecretKeySpec secretKeySpec = new SecretKeySpec(key, mac.getAlgorithm());
mac.init(secretKeySpec);
return mac.doFinal(msg.getBytes(CHARSET));
}

public static void main(String[] args) throws Exception {
String service = "cvm";
String host = "cvm.finance.cloud.tencent.com";
String region = "ap-guangzhou";
String action = "DescribeInstances";
String version = "2017-03-12";
String algorithm = "TC3-HMAC-SHA256";
String timestamp = "1539084154";
//String timestamp = String.valueOf(System.currentTimeMillis() / 1000);
SimpleDateFormat sdf = new SimpleDateFormat("yyyy-MM-dd");
// 注意时区，否则容易出错
sdf.setTimeZone(TimeZone.getTimeZone("UTC"));
String date = sdf.format(new Date(Long.valueOf(timestamp + "000")));

// ************ 步骤 1：拼接规范请求串 ************
String httpRequestMethod = "GET";
String canonicalUri = "/";
String canonicalQueryString = "Limit=10&Offset=0";
String canonicalHeaders = "content-type:application/x-www-form-urlencoded\n" + "host:" + host + "\n";
String signedHeaders = "content-type;host";
String hashedRequestPayload = DigestUtils.sha256Hex("");
String canonicalRequest = httpRequestMethod + "\n" + canonicalUri + "\n" + canonicalQueryString + "\n"
+ canonicalHeaders + "\n" + signedHeaders + "\n" + hashedRequestPayload;
System.out.println(canonicalRequest);

// ************ 步骤 2：拼接待签名字符串 ************
String credentialScope = date + "/" + service + "/" + "tc3_request";
String hashedCanonicalRequest = DigestUtils.sha256Hex(canonicalRequest.getBytes(CHARSET));
String stringToSign = algorithm + "\n" + timestamp + "\n" + credentialScope + "\n" + hashedCanonicalRequest;
System.out.println(stringToSign);

// ************ 步骤 3：计算签名 ************
byte[] secretDate = sign256(("TC3" + SECRET_KEY).getBytes(CHARSET), date);
byte[] secretService = sign256(secretDate, service);
byte[] secretSigning = sign256(secretService, "tc3_request");
String signature = DatatypeConverter.printHexBinary(sign256(secretSigning, stringToSign)).toLowerCase();
System.out.println(signature);

// ************ 步骤 4：拼接 Authorization ************
String authorization = algorithm + " " + "Credential=" + SECRET_ID + "/" + credentialScope + ", "
+ "SignedHeaders=" + signedHeaders + ", " + "Signature=" + signature;
System.out.println(authorization);
```

```
TreeMap<String, String> headers = new TreeMap<String, String>();
headers.put("Authorization", authorization);
headers.put("Host", host);
headers.put("Content-Type", CT_X_WWW_FORM_URLENCODED);
headers.put("X-TC-Action", action);
headers.put("X-TC-Timestamp", timestamp);
headers.put("X-TC-Version", version);
headers.put("X-TC-Region", region);
}
}
```

Python

```
# -*- coding: utf-8 -*-
import hashlib, hmac, json, os, sys, time
from datetime import datetime

# 密钥参数
secret_id = "AKIDz8krbsJ5yKBZQpn74WFkmLPx3EXAMPLE"
secret_key = "Gu5t9xGARNpq86cd98joQYCN3EXAMPLE"

service = "cvm"
host = "cvm.finance.cloud.tencent.com"
endpoint = "http://imgcache.finance.cloud.tencent.com:80" + host
region = "ap-guangzhou"
action = "DescribeInstances"
version = "2017-03-12"
algorithm = "TC3-HMAC-SHA256"
timestamp = 1539084154
date = datetime.utcfromtimestamp(timestamp).strftime("%Y-%m-%d")
params = {"Limit": 10, "Offset": 0}

# ************* 步骤 1：拼接规范请求串 *************
http_request_method = "GET"
canonical_uri = "/"
canonical_querystring = "Limit=10&Offset=0"
ct = "x-www-form-urlencoded"
payload = ""
if http_request_method == "POST":
canonical_querystring = ""
ct = "json"
payload = json.dumps(params)
canonical_headers = "content-type:application/%s\nhost:%s\n" % (ct, host)
signed_headers = "content-type;host"
hashed_request_payload = hashlib.sha256(payload.encode("utf-8")).hexdigest()
canonical_request = (http_request_method + "\n" +
canonical_uri + "\n" +
canonical_querystring + "\n" +
canonical_headers + "\n" +
signed_headers + "\n" +
hashed_request_payload)
print(canonical_request)

# ************* 步骤 2：拼接待签名字符串 *************
credential_scope = date + "/" + service + "/" + "tc3_request"
```

```
hashed_canonical_request = hashlib.sha256(canonical_request.encode("utf-8")).hexdigest()
string_to_sign = (algorithm + "\n" +
str(timestamp) + "\n" +
credential_scope + "\n" +
hashed_canonical_request)
print(string_to_sign)


# ************* 步骤 3：计算签名 *************
# 计算签名摘要函数
def sign(key, msg):
return hmac.new(key, msg.encode("utf-8"), hashlib.sha256).digest()
secret_date = sign(("TC3" + secret_key).encode("utf-8"), date)
secret_service = sign(secret_date, service)
secret_signing = sign(secret_service, "tc3_request")
signature = hmac.new(secret_signing, string_to_sign.encode("utf-8"), hashlib.sha256).hexdigest()
print(signature)

# ************* 步骤 4：拼接 Authorization *************
authorization = (algorithm + " " +
"Credential=" + secret_id + "/" + credential_scope + ", " +
"SignedHeaders=" + signed_headers + ", " +
"Signature=" + signature)
print(authorization)

# 公共参数添加到请求头部
headers = {
"Authorization": authorization,
"Host": host,
"Content-Type": "application/%s" % ct,
"X-TC-Action": action,
"X-TC-Timestamp": str(timestamp),
"X-TC-Version": version,
"X-TC-Region": region,
}
```

请求结构

最近更新时间: 2024-06-18 14:31:25

1. 服务地址

地域（Region）是指物理的数据中心的地理区域。tcecloud交付验证不同地域之间完全隔离，保证不同地域间最大程度的稳定性和容错性。为了降低访问时延、提高下载速度，建议您选择最靠近您客户的地域。

您可以通过 API接口 **查询地域列表** 查看完成的地域列表。

2. 通信协议

tcecloud API 的所有接口均通过 HTTPS 进行通信，提供高安全性的通信通道。

3. 请求方法

支持的 HTTP 请求方法:

- POST（推荐）
- GET

POST 请求支持的 Content-Type 类型：

- application/json（推荐），必须使用 TC3-HMAC-SHA256 签名方法。
- application/x-www-form-urlencoded，必须使用 HmacSHA1 或 HmacSHA256 签名方法。
- multipart/form-data（仅部分接口支持），必须使用 TC3-HMAC-SHA256 签名方法。

GET 请求的请求包大小不得超过 32 KB。POST 请求使用签名方法为 HmacSHA1、HmacSHA256 时不得超过 1 MB。POST 请求使用签名方法为 TC3-HMAC-SHA256 时支持 10 MB。

4. 字符编码

均使用UTF-8编码。

返回结果

最近更新时间: 2024-06-18 14:31:25

正确返回结果

以云服务器的接口查看实例状态列表 (DescribeInstancesStatus) 2017-03-12 版本为例，若调用成功，其可能的返回如下为：

```
{
"Response": {
"TotalCount": 0,
"InstanceStatusSet": [],
"RequestId": "b5b41468-520d-4192-b42f-595cc34b6c1c"
}
}
```

- Response 及其内部的 RequestId 是固定的字段，无论请求成功与否，只要 API 处理了，则必定会返回。
- RequestId 用于一个 API 请求的唯一标识，如果 API 出现异常，可以联系我们，并提供该 ID 来解决问题。
- 除了固定的字段外，其余均为具体接口定义的字段，不同的接口所返回的字段参见接口文档中的定义。此例中的 TotalCount 和 InstanceStatusSet 均为 DescribeInstancesStatus 接口定义的字段，由于调用请求的用户暂时还没有云服务器实例，因此 TotalCount 在此情况下的返回值为 0， InstanceStatusSet 列表为空。

错误返回结果

若调用失败，其返回值示例如下为：

```
{
"Response": {
"Error": {
"Code": "AuthFailure.SignatureFailure",
"Message": "The provided credentials could not be validated. Please check your signature is correct."
},
"RequestId": "ed93f3cb-f35e-473f-b9f3-0d451b8b79c6"
}
}
```

- Error 的出现代表着该请求调用失败。Error 字段连同其内部的 Code 和 Message 字段在调用失败时是必定返回的。
- Code 表示具体出错的错误码，当请求出错时可以先根据该错误码在公共错误码和当前接口对应的错误码列表里面查找对应原因和解决方案。
- Message 显示出了这个错误发生的具体原因，随着业务发展或体验优化，此文本可能会经常保持变更或更新，用户不应依赖这个返回值。
- RequestId 用于一个 API 请求的唯一标识，如果 API 出现异常，可以联系我们，并提供该 ID 来解决问题。

公共错误码 (TODO: 重复信息, 是否真的需要?)

返回结果中如果存在 Error 字段，则表示调用 API 接口失败。 Error 中的 Code 字段表示错误码，所有业务都可能出现的错误码为公共错误码，下表列出了公共错误码。

错误码	错误描述
AuthFailure.InvalidSecretId	密钥非法（不是云 API 密钥类型）。
AuthFailure.MFAFailure	MFA 错误。
AuthFailure.SecretIdNotFound	密钥不存在。
AuthFailure.SignatureExpire	签名过期。
AuthFailure.SignatureFailure	签名错误。
AuthFailure.TokenFailure	token 错误。
AuthFailure.UnauthorizedOperation	请求未 CAM 授权。
DryRunOperation	DryRun 操作，代表请求将会是成功的，只是多传了 DryRun 参数。
FailedOperation	操作失败。
InternalError	内部错误。
InvalidAction	接口不存在。
InvalidParameter	参数错误。
InvalidParameterValue	参数取值错误。
LimitExceeded	超过配额限制。
MissingParameter	缺少参数错误。
NoSuchVersion	接口版本不存在。
RequestLimitExceeded	请求的次数超过了频率限制。
ResourceInUse	资源被占用。
ResourceInsufficient	资源不足。
ResourceNotFound	资源不存在。
ResourceUnavailable	资源不可用。
UnauthorizedOperation	未授权操作。
UnknownParameter	未知参数错误。
UnsupportedOperation	操作不支持。
UnsupportedProtocol	http(s)请求协议错误，只支持 GET 和 POST 请求。
UnsupportedRegion	接口不支持所传地域。

公共参数

最近更新时间: 2024-06-18 14:31:25

公共参数是用于标识用户和接口鉴权目的的参数，如非必要，在每个接口单独的接口文档中不再对这些参数进行说明，但每次请求均需要携带这些参数，才能正常发起请求。

签名方法 v3

使用 TC3-HMAC-SHA256 签名方法时，公共参数需要统一放到 HTTP Header 请求头部中，如下：

参数名称	类型	必选	描述
X-TC-Action	String	是	操作的接口名称。取值参考接口文档中输入参数公共参数 Action 的说明。例如云服务器的查询实例列表接口，取值为 DescribeInstances。
X-TC-Region	String	是	地域参数，用来标识希望操作哪个地域的数据。接口接受的地域取值参考接口文档中输入参数公共参数 Region 的说明。注意：某些接口不需要传递该参数，接口文档中会对此特别说明，此时即使传递该参数也不会生效。
X-TC-Timestamp	Integer	是	当前 UNIX 时间戳，可记录发起 API 请求的时间。例如 1529223702。注意：如果与服务器时间相差超过5分钟，会引起签名过期错误。
X-TC-Version	String	是	操作的 API 的版本。取值参考接口文档中入参公共参数 Version 的说明。例如云服务器的版本 2017-03-12。
Authorization	String	是	HTTP 标准身份认证头部字段，例如： TC3-HMAC-SHA256 Credential=AKIDEXAMPLE/Date/service/tc3_request, SignedHeaders=content-type;host, Signature=fe5f80f77d5fa3beca038a248ff027d0445342fe2855ddc963176630326f1024 其中， - TC3-HMAC-SHA256：签名方法，目前固定取该值； - Credential：签名凭证，AKIDEXAMPLE 是 SecretId；Date 是 UTC 标准时间的日期，取值需要和公共参数 X-TC-Timestamp 换算的 UTC 标准时间日期一致；service为产品名，必须与调用的产品域名一致，例如cvm； - SignedHeaders：参与签名计算的头部信息，content-type 和 host 为必选头部； - Signature：签名摘要。
X-TC-Token	String	否	临时证书所用的 Token ，需要结合临时密钥一起使用。临时密钥和 Token 需要到访问管理服务调用接口获取。长期密钥不需要 Token。

签名方法 v1

使用 HmacSHA1 和 HmacSHA256 签名方法时，公共参数需要统一放到请求串中，如下

参数名称	类型	必选	描述
Action	String	是	操作的接口名称。取值参考接口文档中输入参数公共参数 Action 的说明。例如云服务器的查询实例列表接口，取值为 DescribeInstances。

参数名称	类型	必选	描述
Region	String	是	地域参数，用来标识希望操作哪个地域的数据。接口接受的地域取值参考接口文档中输入参数公共参数 Region 的说明。注意：某些接口不需要传递该参数，接口文档中会对此特别说明，此时即使传递该参数也不会生效。
Timestamp	Integer	是	当前 UNIX 时间戳，可记录发起 API 请求的时间。例如1529223702，如果与当前时间相差过大，会引起签名过期错误。
Nonce	Integer	是	随机正整数，与 Timestamp 联合起来，用于防止重放攻击。
SecretId	String	是	在云API密钥上申请的标识身份的 SecretId，一个 SecretId 对应唯一的 SecretKey ，而 SecretKey 会用来生成请求签名 Signature。
Signature	String	是	请求签名，用来验证此次请求的合法性，需要用户根据实际的输入参数计算得出。具体计算方法参见接口鉴权文档。
Version	String	是	操作的 API 的版本。取值参考接口文档中入参公共参数 Version 的说明。例如云服务器的版本 2017-03-12。
SignatureMethod	String	否	签名方式，目前支持 HmacSHA256 和 HmacSHA1。只有指定此参数为 HmacSHA256 时，才使用 HmacSHA256 算法验证签名，其他情况均使用 HmacSHA1 验证签名。
Token	String	否	临时证书所用的 Token ，需要结合临时密钥一起使用。临时密钥和 Token 需要到访问管理服务调用接口获取。长期密钥不需要 Token 。

地域列表

地域（Region）是指物理的数据中心的地理区域。tcecloud交付验证不同地域之间完全隔离，保证不同地域间最大程度的稳定性和容错性。为了降低访问时延、提高下载速度，建议您选择最靠近您客户的地域。

您可以通过 API接口 **查询地域列表** 查看完成的地域列表。

TBase部署中心
查询售卖地域信息

最近更新时间: 2024-06-18 14:31:25

1. 接口描述

接口请求域名： tbase.api3.finance.cloud.tencent.com。

本接口(DescribeAvailableRegion)用于查询售卖地域信息

默认接口请求频率限制：20次/秒。

接口更新时间：2022-05-10 14:55:52。

接口只验签名不鉴权。

2. 输入参数

以下请求参数列表仅列出了接口请求参数和部分公共参数，完整公共参数列表见公共请求参数。

参数名称	必选	允许NULL	类型	描述
Action	是	否	String	公共参数，本接口取值：DescribeAvailableRegion
Version	是	否	String	公共参数，本接口取值：2019-01-07
Region	是	否	String	公共参数，详见产品支持的**地域列表**(TODO)
EngineType	是	否	Array of String	引擎类型，TbaseV2/TbaseV3/TbaseV5

3. 输出参数

参数名称	类型	描述
Items	RegionSaleInfo	可售卖地域详情
RequestId	String	唯一请求 ID，每次请求都会返回。定位问题时需要提供该次请求的 RequestId。

4. 错误码

以下仅列出了接口业务逻辑相关的错误码，其他错误码详见公共错误码。

错误码	描述
FailedOperation.DescribeAvailableRegion	

错误码	描述
InternalError.SystemError	
MissingParameter.InvalidParameterValue	
UnauthorizedOperation.PermissionDenied	

其他接口
修改云数据库安全组

最近更新时间: 2024-06-18 14:31:25

1. 接口描述

接口请求域名： tbase.api3.finance.cloud.tencent.com。

本接口（ModifyDBInstanceSecurityGroups）用于修改云数据库安全组

默认接口请求频率限制：20次/秒。

接口更新时间：2022-04-27 16:38:54。

接口只验签名不鉴权。

2. 输入参数

以下请求参数列表仅列出了接口请求参数和部分公共参数，完整公共参数列表见公共请求参数。

参数名称	必选	允许NULL	类型	描述
Action	是	否	String	公共参数，本接口取值：ModifyDBInstanceSecurityGroups
Version	是	否	String	公共参数，本接口取值：2019-01-07
Region	是	否	String	公共参数，详见产品支持的**地域列表**(TODO)
Product	是	否	String	数据库引擎名称：tbase 等。
SecurityGroupIds	是	否	Array of String	要修改的安全组ID列表，一个或者多个安全组Id组成的数组。
InstanceId	是	否	String	实例ID，格式如：tdpg-c1nl9rpv，与云数据库控制台页面中显示的实例ID相同

3. 输出参数

参数名称	类型	描述
RequestId	String	唯一请求 ID，每次请求都会返回。定位问题时需要提供该次请求的 RequestId。

4. 错误码

以下仅列出了接口业务逻辑相关的错误码，其他错误码详见公共错误码。

错误码	描述
FailedOperation.AssociateSecurityGroupsFailed	
FailedOperation.DisassociateSecurityGroupsFailed	

数据结构

最近更新时间: 2024-06-18 14:31:25

ParamConstraint

实例参数的类型取值范围。

被如下接口引用：

名称	必选	允许NULL	类型	描述
Type	是	否	String	用来描述取值范围类型。section: 数值范围, enum: 枚举, string: 字符串。
Range	是	否	Array of ConstraintRange	type取值为section时的数值范围。
Enum	是	否	Array of String	type取值为enum时的枚举值。
String	是	否	String	type取值为string时的数值。

ProductLicenseData

license 数据统计

被如下接口引用：DescribeResourceOverview

名称	必选	允许NULL	类型	描述
ProductId	否	否	String	磐石分配的产品 ID，固定为 87870
ProductIdDescribe	否	否	String	产品描述
UsageUnit	否	否	String	数量单位或节点类型
UsageValue	否	否	String	节点数量

PgNodeInfo

节点规格信息

被如下接口引用：CreatePGInstanceHour

名称	必选	允许NULL	类型	描述
SpecCode	是	否	String	节点规格代码
Storage	是	否	Int64	磁盘单个节点大小

RegionSaleDetail

售卖地域详细信息

被如下接口引用：DescribeAvailableRegion

名称	必选	允许NULL	类型	描述
Region	否	否	String	地域英文名
RegionName	否	否	String	地域中文名
Area	否	否	String	地域所属大区
IsSellOut	否	否	Bool	是否售罄

SecurityGroup

安全组详情

被如下接口引用：DescribeDBSecurityGroups

名称	必选	允许NULL	类型	描述
CreateTime	是	否	String	创建时间，时间格式：yyyy-mm-dd hh:mm:ss
Inbound	是	否	Array of Inbound	入站规则。
Outbound	是	否	Array of Outbound	出站规则。
ProjectId	是	否	Int64	项目ID。
SecurityGroupId	是	否	String	安全组ID。
SecurityGroupName	是	否	String	安全组名称。
SecurityGroupRemark	是	否	String	安全组备注。

ZoneSellConf

描述单个可用区的可选规格

被如下接口引用：DescribeAvailableZoneConfig

名称	必选	允许NULL	类型	描述
Zone	是	否	String	可用区英文名
ZoneName	是	否	String	可用区中文名
IsDefaultZone	是	否	Bool	是否为当前地域的默认可用区
AvailableSpec	是	否	Array of SellSpec	可选节点规格

AccountInfo

账号信息。

被如下接口引用：DescribeAccounts

名称	必选	允许NULL	类型	描述
InstanceId	是	否	String	实例Id。
Username	是	否	String	账号名。

Inbound

安全组入站规则

被如下接口引用：DescribeDBSecurityGroups

名称	必选	允许NULL	类型	描述
Action	是	否	String	策略，ACCEPT或者DROP。
AddressModule	是	否	String	地址组id代表的地址集合。
CidrIp	是	否	String	来源Ip或Ip段，例如192.168.0.0/16。
Desc	是	否	String	描述。
Id	是	否	String	安全组id代表的地址集合。
IpProtocol	是	否	String	网络协议，支持udp、tcp等。
PortRange	是	否	String	端口。
ServiceModule	是	否	String	服务组id代表的协议和端口集合。

Node

节点信息

被如下接口引用：DescribePGInstanceDetails

名称	必选	允许NULL	类型	描述
Role	是	否	String	节点角色
Region	是	否	String	地域信息
Zone	是	否	String	可用区信息
ZoneName	是	否	String	可用区名
Type	是	否	String	节点所在节点组中的节点类型

名称	必选	允许NULL	类型	描述
NodeName	是	否	String	节点名

Filter

描述键值对过滤器，用于条件过滤查询。例如过滤ID、名称、状态等

- 若存在多个Filter时，Filter间的关系为逻辑与（AND）关系。
- 若同一个Filter存在多个Values，同一Filter下Values间的关系为逻辑或（OR）关系。

被如下接口引用：DescribeInstances、DescribePGInstances、SecurityGroups

名称	必选	允许NULL	类型	描述
Name	是	否	String	需要过滤的字段。
Values	是	否	Array of String	字段的过滤值。

ConfigParam

实例参数配置。

被如下接口引用：ModifyDBParameters

名称	必选	允许NULL	类型	描述
ParameterName	是	否	String	参数名。
ParameterValue	是	否	String	参数值。

NodeConfigParams

节点参数配置。

被如下接口引用：ModifyDBParameters

名称	必选	允许NULL	类型	描述
NodeType	是	否	String	节点类型，目前支持：cn、dn。
ConfigParams	是	否	Array of ConfigParam	需要修改的参数配置。

ParamItem

各节点对应的参数。

被如下接口引用：

名称	必选	允许NULL	类型	描述
NodeType	是	否	String	节点类型。
NodeName	是	否	String	节点名。
Details	是	否	Array of ParamDetail	节点的各个参数信息。
TotalCount	是	否	Int64	参数个数。

VpcInfo

节点vpc信息

被如下接口引用：DescribePGInstanceDetails

名称	必选	允许NULL	类型	描述
VpcId	是	否	String	私有网络Id
SubnetId	是	否	String	私有网络子网Id
VpcName	是	否	String	私有网络名称
SubnetName	是	否	String	私有网络子网名称
Vip	是	否	String	vip
Vport	是	否	Int64	vport

Instance

描述实例的信息

被如下接口引用：DescribeInstances

名称	必选	允许NULL	类型	描述
InstanceId	是	否	String	实例ID
InstanceName	是	否	String	实例名称
Region	是	否	String	地域ID
RegionName	是	否	String	地域名称
Zone	是	否	String	可用区ID
ZoneName	是	否	String	可用区名称
VpcId	是	否	String	私有网络Id
VpcName	是	否	String	私有网络名称

名称	必选	允许NULL	类型	描述
SubnetId	是	否	String	私有网络子网ID
SubnetName	是	否	String	私有网络子网名称
CreatedAt	是	否	Datetime	创建时间
Status	是	否	String	状态
StatusDesc	是	否	String	状态描述
TaskType	是	否	String	任务类型
TaskDesc	是	否	String	任务描述
Vip	是	否	String	虚拟IP
Vport	是	否	Int64	虚拟端口
TradeType	是	否	String	付费类型，postpaid - 后付费，prepaid - 预付费
CNNodes	是	否	Array of InstanceNode	计算节点列表
DNNodes	是	否	Array of InstanceNode	数据节点列表
AdminUserName	是	否	String	管理员账号名
ProjectId	否	否	String	项目ID，为空表示没有所属项目
ProjectName	否	否	String	项目名，为空表示没有所属项目

TagInfo

Tag信息，包含TagKey和TagValue信息

被如下接口引用：CreateInstanceHour

名称	必选	允许NULL	类型	描述
TagKey	是	否	String	TagKey值
TagValue	否	是	String	TagValue值

RegionSaleInfo

售卖地域信息

被如下接口引用：DescribeAvailableRegion

名称	必选	允许NULL	类型	描述
TotalCount	是	否	Uint64	可售卖地域总数

名称	必选	允许NULL	类型	描述
RegionSaleDetail	是	否	Array of RegionSaleDetail	可售卖地域详情
EngineType	是	否	String	可售卖地域引擎类型TbaseV2/TbaseV3/TbaseV5

ConstraintRange

实例参数取值范围，标识最大最小值。

被如下接口引用：

名称	必选	允许NULL	类型	描述
Min	是	否	String	区间的最小值。
Max	是	否	String	区间的最大值。

RecoveryInfo

创建回档实例的必传参数

被如下接口引用：CreateInstanceHour

名称	必选	允许NULL	类型	描述
RecoveryTaskId	是	否	Uint64	回档任务ID，会在获取回档时间接口中返回
RecoveryTime	是	否	String	回档时间，会在获取回档时间接口中返回，需选择回档时间范围中的某一个时刻点
RecoveryInstanceId	是	否	String	要回档的实例ID

EngineVersion

描述可用的引擎版本

被如下接口引用：DescribeAvailableEngineVersions

名称	必选	允许NULL	类型	描述
Type	是	否	String	引擎类型，例如 TBaseV2,TBaseV3
Version	是	否	String	引擎版本，例如：1.0.1

ResourceUsage

资源使用统计

被如下接口引用：DescribeResourceOverview

名称	必选	允许NULL	类型	描述
Total	是	否	Int64	总量
Used	是	否	Int64	已使用量

Backup

实例备份信息

被如下接口引用：DescribeBackupLists

名称	必选	允许NULL	类型	描述
BackupUid	是	否	Int64	实例备份信息的唯一标识，查询备份详情时的入参。
StartTime	是	否	String	实例备份开始时间。
EndTime	是	否	String	实例备份结束时间。
StrategyType	是	否	String	备份策略类型 Auto(自动备份)
BackupType	是	否	String	备份类型 Physical(物理备份)
Status	是	否	Int64	备份状态。1(备份中)，2（备份成功），-1（备份失败）

SellSpec

描述单条规格信息

被如下接口引用：DescribeAvailableZoneConfig

名称	必选	允许NULL	类型	描述
SpecCode	是	否	String	规格码，唯一标识一个规格，创建实例时传该字段用于标识一个规格
NodeType	是	否	String	节点类型: cn（计算节点），dn（存储节点）
Cpu	是	否	Int64	cpu核心数，单位：核
Memory	是	否	Int64	内存大小，单位：GB
MinStorage	是	否	Int64	磁盘最小规格，单位：GB
MaxStorage	是	否	Int64	磁盘最大规格，单位：GB
StorageStep	是	否	Int64	磁盘调整步长，单位：GB
NodeCount	是	否	Int64	磁盘调整步长，单位：GB

PGZoneSellConf

描述Postgres实例单个可用区的可选规格

被如下接口引用：DescribePGAvailableZoneConfig

名称	必选	允许NULL	类型	描述
Zone	是	否	String	可用区英文名
ZoneName	是	否	String	可用区中文名
IsDefaultZone	是	否	Bool	是否为当前地域的默认可用区
AvailableSpec	是	否	Array of PGSellSpec	可选节点规格

PGInstanceSet

Postgres实例信息

被如下接口引用：DescribePGInstances

名称	必选	允许NULL	类型	描述
InstanceId	是	否	String	实例Id
InstanceName	是	否	String	实例名
Region	是	否	String	地域信息
RegionName	是	否	String	地域名称
Zone	是	否	String	可用区信息
ZoneName	是	否	String	可用区名称
VpcId	是	否	String	私有网络Id
VpcName	是	否	String	私有网络名称
SubnetId	是	否	String	私有网络子网ID
SubnetName	是	否	String	私有网络子网名称
CreatedAt	是	否	String	创建时间
Status	是	否	String	实例状态
StatusDesc	是	否	String	实例状态描述
TaskType	是	否	String	任务类型
TaskDesc	是	否	String	任务描述信息
Vip	是	否	String	实例ip

名称	必选	允许NULL	类型	描述
Vport	是	否	Int64	实例端口
Cpu	是	否	Int64	cpu核数
Memory	是	否	Int64	内存大小
Storage	是	否	Int64	磁盘大小
NodeCount	是	否	Int64	节点数量
TradeType	是	否	String	付费类型，(后付费-postpaid，预付费-prepaid)
AdminUserName	是	否	String	管理员账号名
RoVip	是	否	String	只读vip
RoVport	是	否	Int64	只读port
ProjectId	否	否	String	项目ID，为空表示没有所属项目
ProjectName	否	否	String	项目名，为空表示没有所属项目
IsolateOperator	否	否	String	实例隔离者
IsArrears	否	否	Int64	是否欠费，1 - 欠费， 0 - 不欠费

TimePoint

实例回档时可选择的时间

被如下接口引用：DescribeAvailableRecoveryTimePoint

名称	必选	允许NULL	类型	描述
Time	是	否	String	实例回档时可选择的时间

RegionSellConf

描述单个地域的可选规格

被如下接口引用：DescribeAvailableZoneConfig

名称	必选	允许NULL	类型	描述
Region	是	否	String	地域英文名
RegionName	是	否	String	地域中文名
Area	是	否	String	所属大区
IsDefaultRegion	是	否	Bool	是否为默认地域

名称	必选	允许NULL	类型	描述
AvailableZones	是	否	Array of ZoneSellConf	可用区售卖配置

InstanceOverviewDetail

资源概览详情

被如下接口引用：DescribeResourceOverview

名称	必选	允许NULL	类型	描述
Count	否	否	Int64	本产品实例数量
ProductName	否	否	String	产品名称
ProductType	否	否	String	产品类型

ResourcePool

资源池信息

被如下接口引用：DescribeResourcePools

名称	必选	允许NULL	类型	描述
PoolName	否	否	String	资源池名
PoolId	否	否	Int64	资源池id
PoolDesc	否	否	String	资源池描述信息

NodeInfo

创建tbase实例时的节点相关信息

被如下接口引用：CreateInstanceHour

名称	必选	允许NULL	类型	描述
NodeCount	是	否	Int64	要创建实例的某类型节点的总个数（节点组数与每组节点个数的乘积）。
SetCount	是	否	Int64	要创建实例的某类型节点的组数。
Storage	是	否	Int64	要创建实例的某类型节点的单节点磁盘大小。
SpecCode	是	否	String	要创建实例的某类型节点的规格码，可以通过DescribeAvailableZoneConfig接口获得。
SyncType	否	是	String	设置节点的复制类型，目前只有dn设置，可选参数为sync（同步），async（异步），sync2async（异步可退化），默认值为sync2async

ParamDetail

实例参数详细信息。

被如下接口引用：

名称	必选	允许NULL	类型	描述
ParameterName	是	否	String	参数名。
DefaultValue	是	否	String	参数默认值。
NeedRestart	是	否	Bool	修改此参数是否需要重启。
RunningValue	是	否	String	当前运行值。
ValueRange	是	否	Array of ParamConstraint	参数取值范围。
Unit	是	否	String	参数单位。
ShortDesc	是	否	String	参数英文简介。

PGSellSpec

描述Postgres售卖单条规格信息

被如下接口引用：DescribePGAvailableZoneConfig

名称	必选	允许NULL	类型	描述
SpecCode	是	否	String	规格码，唯一标识一个规格，创建实例时传该字段用于标识一个规格
NodeType	是	否	String	节点类型: pg_dn（postgres实例存储节点）
Cpu	是	否	Int64	cpu核心数，单位：核
Memory	是	否	Int64	内存大小，单位：GB
MinStorage	是	否	Int64	磁盘最小规格，单位：GB
MaxStorage	是	否	Int64	磁盘最大规格，单位：GB
StorageStep	是	否	Int64	磁盘调整步长，单位：GB
NodeCount	是	否	Int64	磁盘调整步长，单位：GB

InstanceNode

描述实例的节点信息

被如下接口引用：DescribeInstanceDetails、DescribeInstances

名称	必选	允许NULL	类型	描述

名称	必选	允许NULL	类型	描述
NodeType	是	否	String	节点类型
Cpu	是	否	Int64	节点cpu
Memory	是	否	Int64	节点内存
Storage	是	否	Int64	节点数据盘大小
NodeId	是	否	String	节点唯一ID
UsedStorage	是	否	Int64	已用数据存储空间
NodeName	是	否	String	节点名
Type	是	否	String	所在节点组中的节点类型，master-主节点
Zone	否	是	String	可用区
ZoneName	否	是	String	可用区名称

NormQueryItem

慢查询详细信息

被如下接口引用：DescribeSlowLog

名称	必选	允许NULL	类型	描述
DatabaseName	是	否	String	数据库名。
NormalQuery	是	否	String	脱敏后的sql语句。
MaxElapsedQuery	是	否	String	执行时间最长的语句。
Calls	是	否	Int64	调用次数。
CostTime	是	否	Float	花费总时间。
ClientIp	是	否	String	客户端ip。
UserName	是	否	String	用户名。
TotalCallsPercent	是	否	Float	在总的慢查询中的访问次数占比。
TotalCostTimePercent	是	否	Float	在总的慢查询中的访问时间占比。
MinCostTime	是	否	Float	花费的最小时间。
MaxCostTime	是	否	Float	花费的最长时间。
FirstTime	是	否	String	最早一条慢查询的sql时间。
LastTime	是	否	String	最近一条慢查询的sql时间。

名称	必选	允许NULL	类型	描述
SharedReadBlks	是	否	Int64	读共享内存块数。
SharedWriteBlks	是	否	Int64	写共享内存块数。
ReadCostTime	是	否	Float	读io总耗时。
WriteCostTime	是	否	Float	写io总耗时。

SyncTypeInfo

修改节点的复制类型。

被如下接口引用：ModifyInstanceSyncType

名称	必选	允许NULL	类型	描述
SourceSyncType	否	否	String	节点的原复制类型，可选项(sync:同步不可退化，async:异步，sync2async:同步可退化)
SlaveNodeId	是	否	String	备节点的节点id。
TargetSyncType	是	否	String	节点的目标复制类型，可选项(sync:同步不可退化，async:异步，sync2async:同步可退化)
MasterNodeId	是	否	String	主节点的节点id。

ConfigHistory

参数修改历史记录

被如下接口引用：DescribeDBConfigHistory

名称	必选	允许NULL	类型	描述
Id	是	否	Int64	参数修改历史记录唯一id。
InstanceId	是	否	String	实例id。
CreatedAt	是	否	String	参数修改的时间。
UpdatedAt	是	否	String	参数修改的结束时间。
NodeType	是	否	String	节点类型。
ParamName	是	否	String	参数名。
ParamNewValue	是	否	String	修改后的参数值。
ParamOldValue	是	否	String	修改前的参数值。
Status	是	否	String	参数修改状态：success, failed, doing

ErrorLogDetail

错误日志详细信息。

被如下接口引用：DescribeErrorLog、DescribePGErrorLog

名称	必选	允许NULL	类型	描述
UserName	是	否	String	用户名。
Database	是	否	String	数据库名。
ErrorTime	是	否	String	错误日志发生时间。
ErrorMessage	是	否	String	错误详细信息。

SlowLogDetail

慢查询日志详细信息。

被如下接口引用：DescribeSlowLog

名称	必选	允许NULL	类型	描述
TotalTime	是	否	Float	花费总时间。
TotalCalls	是	否	Int64	调用总次数。
NormalQuerys	是	否	Array of NormQueryItem	慢查询sql详细信息。

NodeBackupDetail

实例节点备份详情

被如下接口引用：DescribeBackupDetails

名称	必选	允许NULL	类型	描述
NodeName	是	否	String	节点名。
NodeType	是	否	String	节点类型。
LatestStatus	是	否	Int64	备份状态。1(备份中)，2(备份成功)，-1(备份失败)
LatestStartTime	是	否	String	最近备份开始时间。
LatestEndTime	是	否	String	最近备份结束时间。

PGRegionSellConf

描述单个地域的可选规格

被如下接口引用：DescribePGAvailableZoneConfig

名称	必选	允许NULL	类型	描述
Region	是	否	String	地域英文名
RegionName	是	否	String	地域中文名
Area	是	否	String	所属大区
IsDefaultRegion	是	否	Bool	是否为默认地域
AvailableZones	是	否	Array of PGZoneSellConf	可用区售卖配置

InstanceOverview

实例资源概览

被如下接口引用：DescribeResourceOverview

名称	必选	允许NULL	类型	描述
TotalCount	是	否	Int64	所有类型实例总数量
Detail	是	否	Array of InstanceOverviewDetail	实例资源详情

Outbound

安全组出站规则

被如下接口引用：DescribeDBSecurityGroups

名称	必选	允许NULL	类型	描述
Action	是	否	String	策略，ACCEPT或者DROP。
AddressMod	是	否	String	地址组id代表的地址集合。
CidrIp	是	否	String	来源Ip或Ip段，例如192.168.0.0/16。
Desc	是	否	String	描述。
Id	是	否	String	安全组id代表的地址集合。
IpProtocol	是	否	String	网络协议，支持udp、tcp等。
PortRange	是	否	String	端口。
ServiceModule	是	否	String	服务组id代表的协议和端口集合。

错误码

最近更新时间: 2024-06-18 14:31:26

功能说明

如果返回结果中存在 Error 字段，则表示调用 API 接口失败。例如：

```
{
"Response": {
"Error": {
"Code": "AuthFailure.SignatureFailure",
"Message": "The provided credentials could not be validated. Please check your signature is correct."
},
"RequestId": "ed93f3cb-f35e-473f-b9f3-0d451b8b79c6"
}
}
```

Error 中的 Code 表示错误码，Message 表示该错误的具体信息。

错误码列表

公共错误码

错误码	说明
AuthFailure.InvalidSecretId	密钥非法（不是云 API 密钥类型）。
AuthFailure.MFAFailure	MFA 错误。
AuthFailure.SecretIdNotFound	密钥不存在。 请在控制台检查密钥是否已被删除或者禁用，如状态正常，请检查密钥是否填写正确，注意前后不得有空格。
AuthFailure.SignatureExpire	签名过期。Timestamp 和服务器时间相差不得超过五分钟，请检查本地时间是否和标准时间同步。
AuthFailure.SignatureFailure	签名错误。 签名计算错误，请对照调用方式中的接口鉴权文档检查签名计算过程。
AuthFailure.TokenFailure	token 错误。
AuthFailure.UnauthorizedOperation	请求未 CAM 授权。
DryRunOperation	DryRun 操作，代表请求将会是成功的，只是多传了 DryRun 参数。
FailedOperation	操作失败。
InternalError	内部错误。
InvalidAction	接口不存在。
InvalidParameter	参数错误。
InvalidParameterValue	参数取值错误。

错误码	说明
LimitExceeded	超过配额限制。
MissingParameter	缺少参数错误。
NoSuchVersion	接口版本不存在。
RequestLimitExceeded	请求的次数超过了频率限制。
ResourceInUse	资源被占用。
ResourceInsufficient	资源不足。
ResourceNotFound	资源不存在。
ResourceUnavailable	资源不可用。
UnauthorizedOperation	未授权操作。
UnknownParameter	未知参数错误。
UnsupportedOperation	操作不支持。
UnsupportedProtocol	http(s)请求协议错误，只支持 GET 和 POST 请求。
UnsupportedRegion	接口不支持所传地域。

业务错误码

错误码	说明
FailedOperation.ModifyInstanceSyncTypeFailed	
FailedOperation.ListBackupsFailed	
FailedOperation.QueryCharsetFailed	
FailedOperation.QueryAvailableZoneConfigFailed	
FailedOperation.QueryInstanceFailed	
InvalidParameter	
InvalidParameter.PermissionDenied	
InvalidParameterValue.VpcIpUsed	
FailedOperation.AssociateSecurityGroupsFailed	
InvalidParameterValue.VpcIpInvalid	
InvalidParameterValue.CheckVipStatusFail	
FailedOperation.QueryErrLogFailed	
FailedOperation.OperateDatabaseFailed	

错误码	说明
FailedOperation.QueryInstanceNodesFailed	
InvalidParameter.CheckParamNotPass	
FailedOperation.QueryServerFailed	
UnauthorizedOperation.PermissionDenied	
MissingParameter.InvalidParameterValue	
FailedOperation.DisassociateSecurityGroupsFailed	
FailedOperation.QueryDBNameListFailed	
FailedOperation.IsolateInstanceFailed	
FailedOperation.GetBackupRulesFailed	
FailedOperation.QuerySupportedEngineFailed	
FailedOperation.PostpaidCreateInstanceHourFailed	
InvalidParameterValue.VpcCodeUnknown	
InternalError.SystemError	
FailedOperation.ActiveIsolateInstanceFailed	
FailedOperation.DescribeAvailableRegion	
InvalidParameterValue.TypeNotSupport	
FailedOperation.QueryInstanceInfoFailed	
FailedOperation.ModifyDBParametersFailed	
FailedOperation.SetBackupRulesFailed	
FailedOperation.ListAccountsFailed	
FailedOperation.GetBackupDetailsFailed	
ResourceUnavailable.UnexpectedInstanceStatus	
InvalidParameterValue.VpcIpNotInSubnet	
FailedOperation.GetInstanceNetworkFailed	
InvalidParameterValue.SpecNotMatch	
FailedOperation.QueryVpcAndSubnetFailed	
InvalidParameterValue.ZoneNotFound	
InvalidParameterValue.RegionNotFound	
FailedOperation.OssCreatedResource	

错误码	说明
FailedOperation.ActivateIsolateInstanceFailed	
FailedOperation.QuerySlowLogFailed	
InvalidParameterValue.InstanceNotFound	
InvalidParameterValue.GetSubnetInfoFail	
FailedOperation.CreateFlowFailed	
FailedOperation.QueryConfigHistoryFailed	
InvalidParameter.PasswordStrengthNotEnough	