


TBase 开发工程师教材



Tencent/大数据平台/TBase 研发组

版本号——V2.10

更新日期：2019-03-23



1、TBase 数据库概述

1.1、什么是 TBase

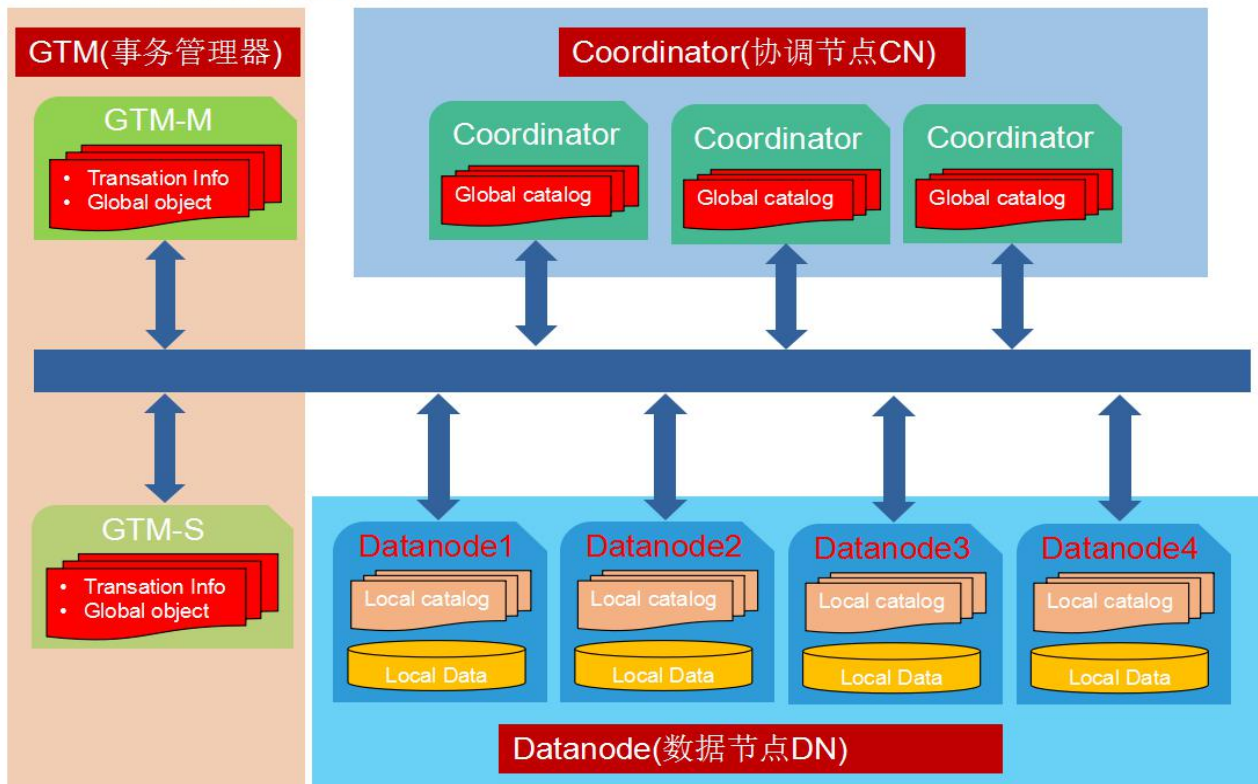
腾讯 TBase 分布式关系型数据库是一款面向海量在线实时分布式事务交易和 MPP 实时数据分析通用型高性能数据库系统。面对应用业务产生的不定性数据爆炸需求，不管是高并发的交易还是海量的实时数据分析，TBase 都有足够能力处理。TBase 产品原型起源于技术成熟，功能强大的 PostgreSQL，PostgreSQL 经历了 30 多年的发展，是目前世界上最好的企业级开源关系型数据库。在此基础上我们构造和发行了具有功能更丰富、稳定性更好、兼容性更广、安全性更高、性能更强、扩展性极好的分布式数据库 TBase 产品。腾讯公司对 TBase 完全自主可控，具有完全自主知识产权，具备在中高端市场可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

1.2、TBase 设计目标

- 1、提供全局实时分布式事务能力
- 2、弹性扩缩容和负载再平衡能力
- 3、冷热数据分离，数据倾斜治理能力
- 4、三权分立管理能力
- 5、支持两地三中心部署，支持数据同步，半同步，异步复制能力
- 6、可以提供多个主节点同时处理来自应用端发出的 SQL 语句，这些节点称为 coordinator（协调节点）。
- 7、任何一个协调节点都有全局数据库视图，也就是说当任一台协调节点接收更新数据并提交后，在另外的主节点可以迅速地看到变化的数据视图。
- 8、表以 distributed by shard 或者 replicated 方式分布式存储，并且这对应用来说是透明的。
- 9、不需要架构师在应用层再构建 sharding 应用组件，大大的降低应用程序的复杂度，提高开发的效率。

1.3、TBase 集群构架

TBase集群架构：



■ Coordinator

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

■ Datanode

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

■ GlobalTransactionManager

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

1.4、TBase 安装

1.4.1、运行环境硬件要求

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

1.4.2、操作系统要求

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

部署请使用最小化安装方式

Centos7.3minimal 发行版本，下载地址

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

1.4.3、如何获取 TBase 安装包

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

1.4.4、安装过程

参考 TBase 部署文档

1.5 、配置 tbase 的客户端使用环境

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

```
export PGXZ_HOME=/usr/local/install/tbase_pgxz
export PATH=$PGXZ_HOME/bin:$PATH
export LD_LIBRARY_PATH=$PGXZ_HOME/lib:${LD_LIBRARY_PATH}
```

```
PGUSER=tbase
PGHOST=127.0.0.1
PGDATABASE=postgres
```

PGPORT=11000

export PGHOST PGUSER PGDATABASE PGPORT

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

~/.bashrc 中

1.6、使用 TBase 对开发者的基本要求

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

1.7 、TBase 每个 cn/dn 节点的系统限制

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

2、创建 TBase 默认的分布式使用环境

2.1、创建数据表默认的 default group

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

--切换为 tbase 用户

su tbase

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

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

```
psql (PostgreSQL 10 (TBase 2.01))
```

```
Type "help" for help.
```

--查询当前什么数据节点 (DN), 这些 DN 节点就是上面初始化集群时建立的

```
postgres=# select * from pgxc_node where node_type='D';
```

node_name	node_type	node_port	node_host	nodeis_primary	nodeis_preferred	node_id	node_cluster_name
dn_0	D	23001	172.16.0.29	f	f	1485981022	tbase_cluster
dn_1	D	23002	172.16.0.47	f	f	-1300059100	tbase_cluster

```
(2 rows)
```

--建立数据表默认使用的 group

```
postgres=# create default node group default_group with(dn_0, dn_2);
```

2.2、为 default group 创建 shardmap

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

```
postgres=# create sharding group to group default_group;
```

```
postgres=# clean sharding;
```

至此我们就可以像单机一样使用 TBase 集群了。

2.3、更多 group 的使用方法

2.3.1、创建扩展 group

■ 建立 group

```
postgres=# create node group ext_group with(dn03,dn04);
```

```
CREATE NODE GROUP
```

■ 为 group 创建 shard

```
postgres=# create extension sharding group to group ext_group;
```

```
CREATE SHARDING GROUP
```

```
postgres=# clean sharding;
```

```
CLEAN SHARDING
```

```
postgres=#
```

2.3.2、删除 group

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

2.3.3、查询集群中 group 的数量

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

3、开发者工具介绍和使用

3.1、可视化开发工具 TStudio

3.1.1、如何打开 TStudio

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

<http://172.16.0.29:5050/>



TStudio Login

postgres@postgres.com

.....

登录

语言 Chinese (Simplified)

忘记你的 密码了吗?

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

3.1.2、TStudio 主界面说明



3.1.3、如何添加要管理的节点

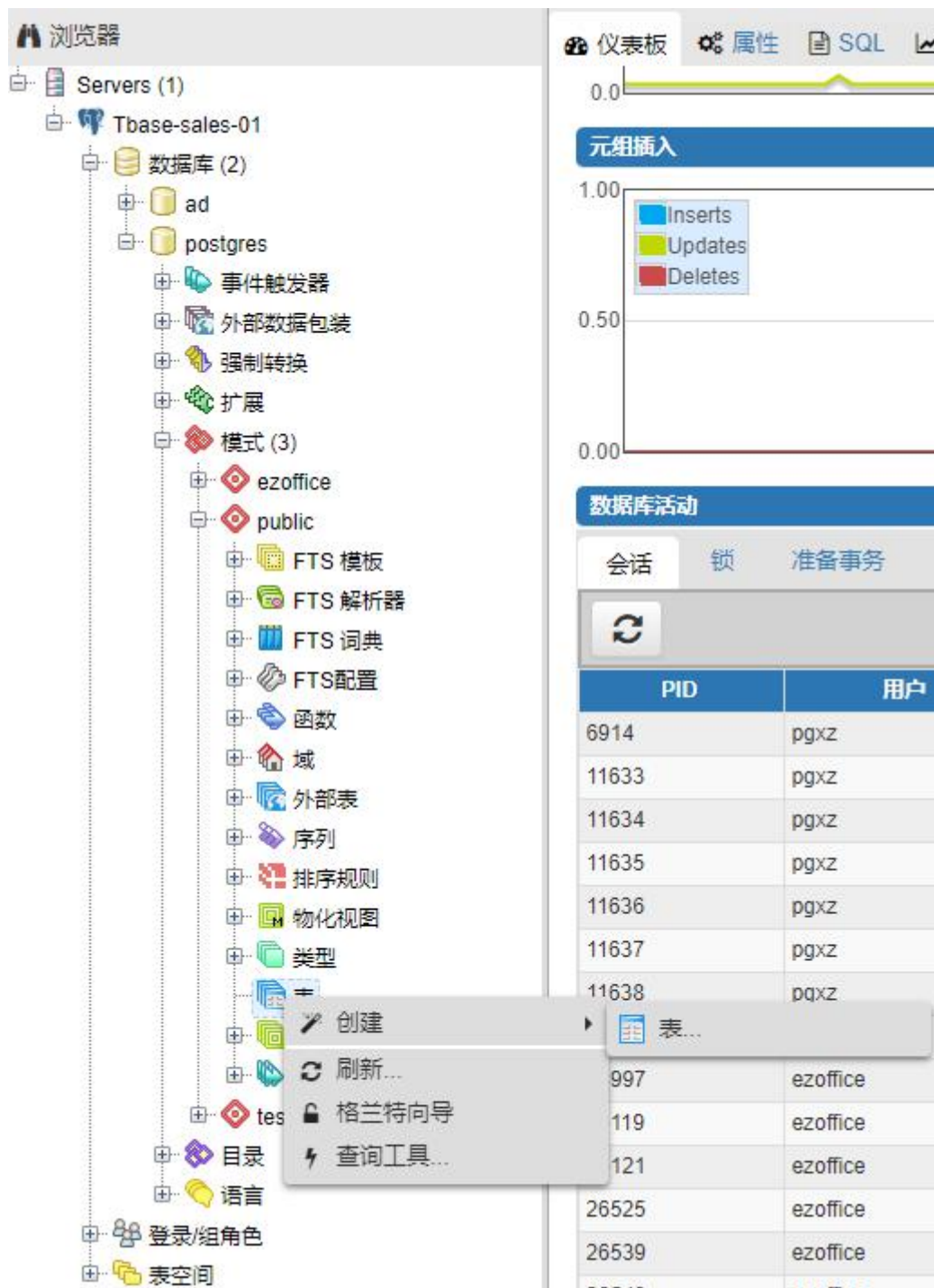
点击上图的“添加新的服务器”在弹出的操作窗口录入相应的信息后按“保存”即可添加一个管理节点，如下图所示。



3.1.4、如何创建数据表、索引

3.1.4.1、创建数据表

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



- 3、) 系统弹出创建表的对话框，如下所示

创建表

通常 列 约束 高级 参数 安全 SQL

名称: t 这里录入表名

所有者: pgxz 这里选择表的所有者

模式: public 这里选择表保存在那个模式下

表空间: S 这里选择表空间, 默认使用即可

注释: 这里可以为表录入注释, 相当于comment on table

保存 取消 重置

表-t

通常 列 约束 高级 参数 安全 SQL

继承自表: 选择要从其继承...

列	名称	数据类型	长度	精度	不为NULL	主键?
<input checked="" type="checkbox"/>	id	integer			No	否
<input checked="" type="checkbox"/>	nickname	character varying	50		No	否

保存 取消 重置

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

3.1.4.2、给表建立索引

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



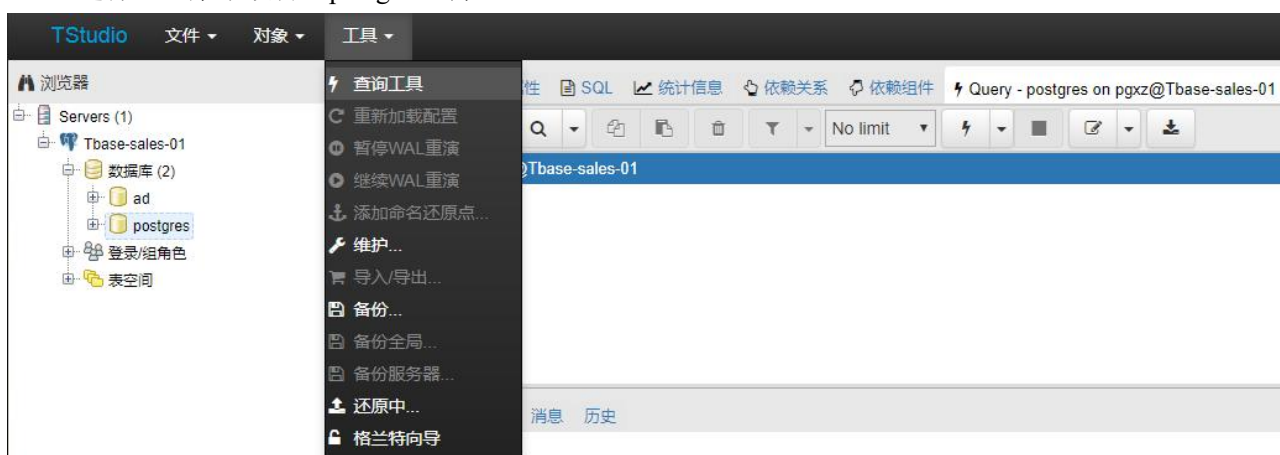
3、) 系统弹出创建索引的对话框, 如下所示



4)、编辑完第一页和第二页框信息后按“保存”按钮即可为表“t”建立索引“t_nickname_idx”

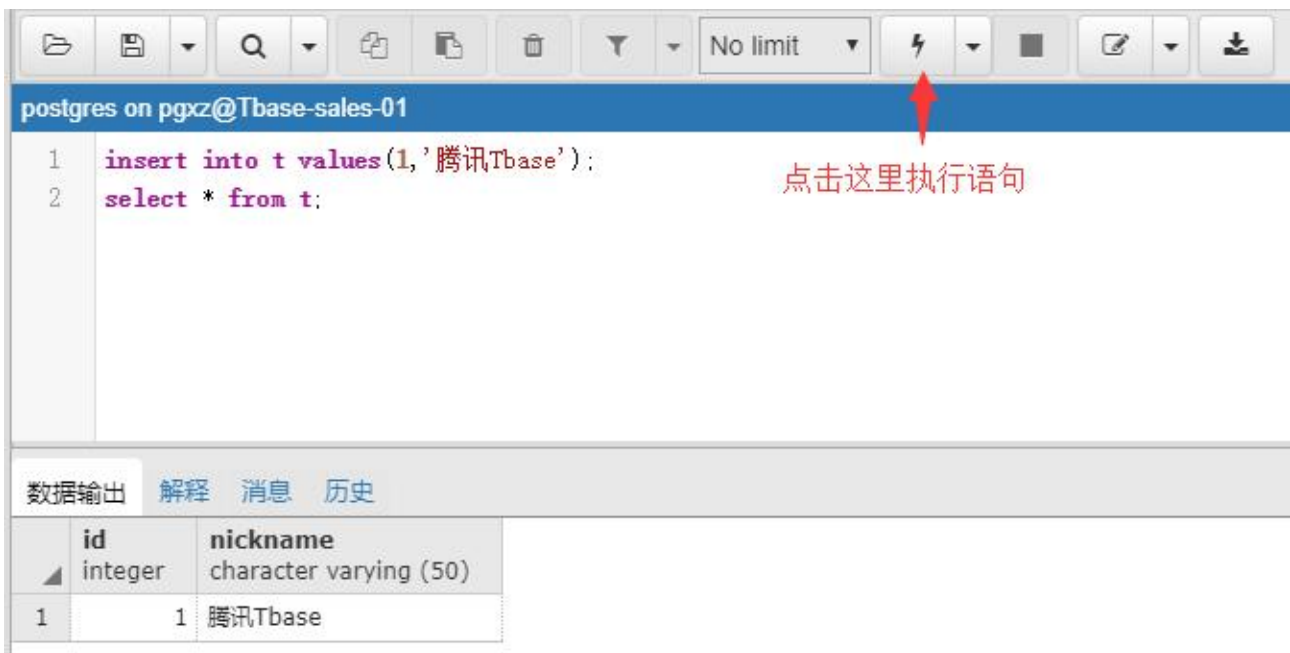
3.1.5、运行手工编写脚本

1)、选择左边菜单项目“postgres”库



2)、点击顶部菜单 工具—> 查询工具 即可在右边工作区域弹出查询窗口

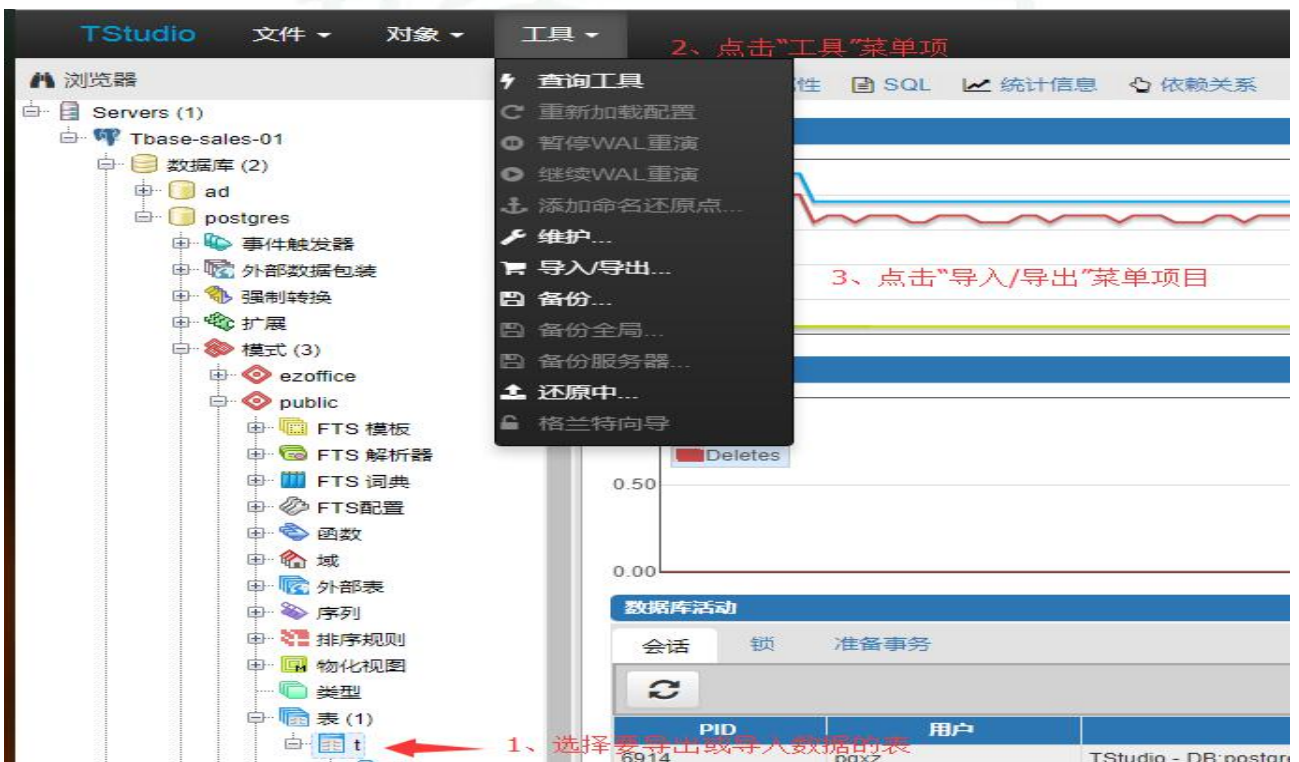
3)、录入要执行的 sql 语句后按窗口上面的“执行”图标执行 sql 脚本，如下图所示



上面语句先插入一条记录，再将插入的记录找出来

3.1.6、表数据导入导出数据

1)、选择左边菜单项目要导出或导入数据的表



2)、点击顶部菜单 工具—> 导入/导出 即可弹出操作对话框，如下图所示

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

```
[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]#
```

3.2、shell 交互客户端 psql

3.2.1、连接到一个数据库

■ 使用参数连接

```
[tbase@VM_0_29_centos root]$ psql -h 172.16.0.29 -p 15432 -U tbase -d postgres
psql (PostgreSQL 10 (TBase 2.01))
Type "help" for help.

postgres=#
```

■ 使用 conninfo 字符串或者一个 URI

```
[tbase@VM_0_29_centos root]$ psql postgresql://tbase@172.16.0.29:15432/postgres
psql (PostgreSQL 10 (TBase 2.01))
```


Type "help" for help.

```
postgres=#
```

■ 配置证书连接

```
[tbase@VM_0_29_centos ~]$ psql 'host=172.16.0.29 port=15432 dbname=postgres user=tbase
sslmode=verify-ca sslcert=postgresql.crt sslkey=postgresql.key sslrootcert=root.crt'
```

Password:

```
psql (PostgreSQL 10 (TBase 2.01))
```

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

Type "help" for help.

```
postgres=> \q
```

■ 配置环境变量后快捷连接

```
[tbase@VM_0_29_centos root]$ PGUSER=tbase
```

```
[tbase@VM_0_29_centos root]$ PGHOST=127.0.0.1
```

```
[tbase@VM_0_29_centos root]$ PGDATABASE=postgres
```

```
[tbase@VM_0_29_centos root]$ PGPORT=15432
```

```
[tbase@VM_0_29_centos root]$ export PGHOST PGUSER PGDATABASE PGPORT
```

```
[tbase@VM_0_29_centos root]$ psql
```

```
psql (PostgreSQL 10 (TBase 2.01))
```

Type "help" for help.

```
postgres=#
```

也可以配置在用户环境变量中

```
[tbase@VM_0_29_centos root]$ cat ~/.bashrc
```

```
# .bashrc
```

```
PGUSER=tbase
```

```
PGHOST=127.0.0.1
```

```
PGDATABASE=postgres
```

```
PGPORT=15432
```

```
export PGHOST PGUSER PGDATABASE PGPORT
```

3.2.2、建立一个新连接

■ 连接到另外一个库（也可以是当前库）

```
postgres=# select pg_backend_pid();
```

```
pg_backend_pid
```

```
-----
```

2408

(1 row)

postgres=# \c

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

postgres=# select pg_backend_pid();

pg_backend_pid

2426

(1 row)

postgres=# \c template1

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

template1=#

■ 连接到外一台服务

postgres=# \c postgres tbase 172.16.0.47 15432

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

3.2.3、显示和设置该连接当前运行参数

■ 显示当前连接的运行参数

postgres=# SELECT CURRENT_USER;

current_user

tbase

(1 row)

postgres=# show search_path ;

search_path

"\$user",public

(1 row)

postgres=# show work_mem ;

work_mem

4MB

(1 row)

■ 设置当前连接的运行参数

postgres=# set search_path = "\$user",public,pg_catalog;


```
SET
postgres=# set work_mem = '8MB';
SET
```

■ 打开和关闭显示每个 sql 语句执行的时间

```
postgres=# \timing on
Timing is on.
postgres=# select count(1) from tbase;
count
-----
10000
(1 row)
```

Time: 5.139 ms

```
postgres=# \timing off
Timing is off.
postgres=# select count(1) from tbase;
count
-----
10000
(1 row)
```

■ 打开和关闭显示每个快捷操作符实际运行的 sql 语句

```
postgres=# \set ECHO_HIDDEN on
postgres=# \dt
***** QUERY *****
SELECT n.nspname as "Schema",
       c.relname as "Name",
       CASE c.relkind WHEN 'r' THEN 'table' WHEN 'v' THEN 'view' WHEN 'm' THEN 'materialized view' WHEN
'i' THEN 'index' WHEN 'S' THEN 'sequence' WHEN 's' THEN 'special' WHEN 'f' THEN 'foreign table' END as
"Type",
       pg_catalog.pg_get_userbyid(c.relowner) as "Owner"
FROM pg_catalog.pg_class c
      LEFT JOIN pg_catalog.pg_namespace n ON n.oid = c.relnamespace
WHERE c.relkind IN ('r',"
      AND n.nspname <> 'pg_catalog'
      AND n.nspname <> 'information_schema'
      AND n.nspname !~ '^pg_toast'
      AND pg_catalog.pg_table_is_visible(c.oid)
ORDER BY 1,2;
*****
```

List of relations

Schema	Name	Type	Owner
--------	------	------	-------

```

-----+-----+-----+-----
public | t_time_range | table | tbase
public | tbase         | table | tbase
(2 rows)

```

```

postgres=# \set ECHO_HIDDEN off
postgres=# \dt

```

List of relations

```

Schema |      Name      | Type | Owner
-----+-----+-----+-----
public | t_time_range | table | tbase
public | tbase         | table | tbase
(2 rows)

```

■ 配置输出结果为 HTML 格式

```

postgres=# \pset format html
Output format is html.
postgres=# \d tbase
<table border="1">
  <caption>Table &quot;public.tbase&quot;</caption>
  <tr>
    <th align="center">Column</th>
    <th align="center">Type</th>
    <th align="center">Modifiers</th>
  </tr>
  <tr valign="top">
    <td align="left">id</td>
    <td align="left">integer</td>
    <td align="left">&nbsp;</td>
  </tr>
  <tr valign="top">
    <td align="left">mc</td>
    <td align="left">text</td>
    <td align="left">&nbsp;</td>
  </tr>
</table>

```

恢复为对齐模式

```

postgres=# \pset format aligned
Output format is aligned.
postgres=# \d tbase
      Table "public.tbase"
  Column | Type    | Modifiers
-----+-----+-----
 id      | integer |

```

```
mc      | text      |
```

■ 配置行列显示格式

```
postgres=# \x on
Expanded display is on.
postgres=# select * from tbase where id=1;
-[ RECORD 1 ]
id | 1
mc | 1
-[ RECORD 2 ]
id | 1
mc | 2
-[ RECORD 3 ]
id | 1
mc | 2
```

```
postgres=# \x off
Expanded display is off.
postgres=# select * from tbase where id=1;
 id | mc
----+----
  1 | 1
  1 | 2
  1 | 2
(3 rows)
```

■ 显示和配置客户端编码

```
postgres=# \encoding
UTF8
```

配置客户端编码为 SQL_ASCII

```
postgres=# \encoding sql_ascii
postgres=# \encoding
SQL_ASCII
```

3.2.4、退出连接

```
postgres=# \q
```

3.2.5、psql 执行一个 sql 命令

■ 显示标题

```
[tbase@VM_0_29_centos root]$ psql -h 172.16.0.29 -p 15432 -U tbase -d postgres -c "select count(1) from pg_class"
count
-----
    317
(1 row)
```

■ 不显示标题

```
[tbase@VM_0_29_centos root]$ psql -h 172.16.0.29 -p 15432 -U tbase -d postgres -t -c "select count(1) from pg_class"
317
```

3.2.6、psql 执行一个 sql 文件中所有命令

■ 在外部执行

```
[tbase@VM_0_29_centos ~]$ cat /data/tbase/tbase.sql
set search_path = public;
insert into tbase values(1,2);
select count(1) from tbase;

[tbase@VM_0_29_centos ~]$ psql -h 172.16.0.29 -p 15432 -U tbase -d postgres -f /data/tbase/tbase.sql
SET
INSERT 0 1
count
-----
 10001
(1 row)
```

■ 在内部执行

```
[tbase@VM_0_29_centos ~]$ psql -h 172.16.0.29 -p 15432 -U tbase -d postgres
psql (PostgreSQL 10 (TBase 2.01))
Type "help" for help.
```

```
postgres=# \i /data/tbase/tbase.sql
SET
INSERT 0 1
count
-----
 10002
(1 row)
```

3.2.7、调用编辑器编写 sql 脚本

```
postgres=# \e
"/tmp/psql.edit.5532.sql" 2L, 35C written
```

编辑完成后保证退出执行

3.2.8、调用外部命令

```
postgres=# \! cat /data/tbase/tbase.sql
set search_path = public;
insert into tbase values(1,2);
select count(1) from tbase;
postgres=# \! ls
data1 data2 data3 install
postgres=# \! ls -l
total 0
drwxrwxr-x 3 tbase tbase 28 Sep 18 11:05 data1
drwxrwxr-x 3 tbase tbase 28 Sep 18 11:05 data2
drwxrwxr-x 3 tbase tbase 28 Sep 18 11:17 data3
drwxr-xr-x 3 tbase tbase 23 Sep 18 10:55 install
postgres=#
```

3.2.9、将执行的结果保存到文件

```
postgres=# \o /data/tbase/log.txt
postgres=# select * from tbase where id=1;
postgres=# \! cat /data/tbase/log.txt
```

```
id | mc
```

```
----+----
```

```
1 | 1
```

```
1 | 2
```

```
1 | 2
```

```
(3 rows)
```

```
postgres=# \o
```

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

```
id | mc
```

```
----+----
```

```
1 | 1
```

```
1 | 2
```

```
1 | 2
```

```
(3 rows)
```

3.2.10、改变当前的工作目录

```
postgres=# \! ls
backup data install java log.txt pgbench tbase tbaseconf shell tbase.sql
postgres=# \cd /data
postgres=# \! ls
package pgsql tbase tbaseConfdb
postgres=#
```

3.2.11、插件管理

- 查看当前库加载了那些插件

```
postgres=# \dx

                                List of installed extensions
  Name          | Version | Schema  | Description
-----+-----+-----+-----
pg_stat_statements | 1.1     | public  | track execution statistics of all SQL statements executed
plpgsql         | 1.0     | pg_catalog | PL/pgSQL procedural language
(2 rows)
```

- 给当前库加载插件

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

- 删除当前库某个插件

```
postgres=# drop extension pg_stat_statements;
DROP EXTENSION
```

3.2.12、数据库相关操作

- \l 显示当前集群中所有数据库

```
postgres=# \l

                                List of databases
  Name          | Owner  | Encoding  | Collate  | Ctype    | Access privileges
-----+-----+-----+-----+-----+-----
postgres      | tbase  | UTF8      | en_US.utf8 | en_US.utf8 |
template0     | tbase  | UTF8      | en_US.utf8 | en_US.utf8 | =c/tbase
               |        |           |           |           | tbase=CTc/tbase
template1     | tbase  | UTF8      | en_US.utf8 | en_US.utf8 | =c/tbase
               |        |           |           |           | tbase=CTc/tbase
```

(3 rows)

■ \l+显示当前当前集群中所有数据库（包含库大小及注释）

注意，如果节点特别多，数据表特别多，使用\l+时统计比较耗时

```
postgres=# \l+
```

List of databases							
Name	Owner	Encoding	Collate	Ctype	Access privileges	Size	Tablespace
postgres	tbase	UTF8	en_US.utf8	en_US.utf8		17 MB	pg_default
template0	tbase	UTF8	en_US.utf8	en_US.utf8	=c/tbase	14 MB	pg_default
unmodifiable empty database							
						tbase=CTc/tbase	
template1	tbase	UTF8	en_US.utf8	en_US.utf8	=c/tbase	14 MB	pg_default
template for new databases							
						tbase=CTc/tbase	

(3 rows)

■ 创建一个新库

```
postgres=# create database mydb;
CREATE DATABASE
postgres=#
```

3.2.13、模式相关操作

■ \dn 显示当前库所有模式

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

■ \dn+显示当前库所有模式（包含注释）

```
postgres=# \dn+
List of schemas
```


Name	Owner	Access privileges	Description
pgxc	tbase		
public	tbase	tbase=UC/tbase =UC/tbase	standard public schema

(2 rows)

■ 创建一个新模式

```
postgres=# create schema mysche;
CREATE SCHEMA
```

3.2.14、用户相关操作

■ \du 显示当前集群中所有数据库用户

```
postgres=# \du
```

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+
```

Role name	Attributes	Member of	Description
audit_admin	No inheritance	{}	
mls_admin	No inheritance	{}	
tbase	Superuser, Create role, Create DB, Replication, Bypass RLS	{}	
tbase01_admin	Superuser, Create role, Create DB	{}	

■ 创建一个新的用户

```
postgres=# create role pgxc with login ;
CREATE ROLE
```

■ 配置用户密码

```
postgres=# \password pgxc
Enter new password:
Enter it again:
```

3.2.15、表相关操作

■ 建立数据表

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

■ \d 查看表结构，包括使用的触发器

```
postgres=# \d tbase
      Table "public.tbase"
  Column | Type      | Modifiers
-----+-----+-----
 id      | integer   |
 mc      | text      |
```

■ \d+ 查看表结构（包含注释），表类型，分布节点

```
postgres=# \d+ tbase
      Table "public.tbase"
  Column | Type      | Modifiers | Storage | Stats target | Description
-----+-----+-----+-----+-----+-----
 id      | integer   |           | plain   |              |
 mc      | text      |           | extended|              |
Has OIDs: no
Distribute By SHARD(id)
Location Nodes: ALL DATANODES
```

■ \dt 查看表列表

```
postgres=# \dt
      List of relations
 Schema | Name          | Type | Owner
-----+-----+-----+-----
 public | t_time_range | table | tbase
 public | tbase         | table | tbase
(2 rows)
```

■ \dt+ 查看表列表详细信息，包含表大小和注释

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

```
postgres=# \dt+
```

```

              List of relations
 Schema |      Name      | Type | Owner | Size | Description
-----+-----+-----+-----+-----+-----
 public | t_time_range | table | tbase | 0 bytes | 这是一个日期分区表
 public | tbase         | table | tbase | 576 kB |
(2 rows)

```

■ \dt+显示某个模式下的所有表

```
postgres=# \dt+ pgxc.*
```

```

              List of relations
 Schema |      Name      | Type | Owner | Size | Description
-----+-----+-----+-----+-----+-----
 pgxc   | order_main | table | tbase | 0 bytes |
(1 row)

```

■ \dt+表名 显示某个表的详细信息

```
postgres=# \dt+ tbase
```

```

              List of relations
 Schema | Name | Type | Owner | Size | Description
-----+-----+-----+-----+-----+-----
 public | tbase | table | tbase | 576 kB |
(1 row)

```

■ \dt+通配符列出适配的表

```
postgres=# \dt+ t*
```

```

              List of relations
 Schema |      Name      | Type | Owner | Size | Description
-----+-----+-----+-----+-----+-----
 public | t_time_range | table | tbase | 0 bytes | 这是一个日期分区表
 public | tbase         | table | tbase | 576 kB |
(2 rows)

```

```
postgres=# \dt+ t_*
```

```

              List of relations
 Schema |      Name      | Type | Owner | Size | Description
-----+-----+-----+-----+-----+-----
 public | t_time_range | table | tbase | 0 bytes | 这是一个日期分区表
(1 row)

```

3.2.16、视图相关操作

■ 建立视图

```
postgres=# create or replace view tbase_view as select * from tbase;
CREATE VIEW
```

■ \d 查视图结构

```
postgres=# \d tbase
          Table "public.tbase"
  Column | Type    | Modifiers
-----+-----+-----
 id      | integer |
 mc      | text    |
```

■ \d+查看视图结构（包含注释），包含创建视图的 sql 语句

```
postgres=# \d+ tbase_view
          View "public.tbase_view"
  Column | Type    | Modifiers | Storage | Description
-----+-----+-----+-----+-----
 id      | integer |           | plain   |
 mc      | text    |           | extended|
View definition:
SELECT tbase.id,
       tbase.mc
FROM tbase;
```

■ \dv 查看视图列表

```
postgres=# \dv
          List of relations
 Schema |          Name          | Type | Owner
-----+-----+-----+-----
 pgxc   | t_time_range_view     | view | tbase
 public | tbase_view            | view | tbase
(2 rows)
```

■ \dv+查看视图列表详细信息（包含注释）

```
postgres=# \dv+
          List of relations
 Schema |          Name          | Type | Owner | Size | Description
-----+-----+-----+-----+-----+-----
```

```
pgxc | t_time_range_view | view | tbase | 0 bytes |
public | tbase_view | view | tbase | 0 bytes | 我的视图
(2 rows)
```

■ \dv+显示某个模式下的所有视图

```
postgres=# \dv+ pgxc.*
```

```

                        List of relations
 Schema |      Name      | Type | Owner | Size  | Description
-----+-----+-----+-----+-----+-----
 pgxc   | t_time_range_view | view | tbase | 0 bytes |
(1 row)
```

■ \dv+视图名 显示某个视图的详细信息

```
postgres=# \dv+ tbase_view
```

```

                        List of relations
 Schema |      Name      | Type | Owner | Size  | Description
-----+-----+-----+-----+-----+-----
 public | tbase_view | view | tbase | 0 bytes | 我的视图
(1 row)
```

■ \dv+通配符列出适配的视图

```
postgres=# \dv+ t*
```

```

                        List of relations
 Schema |      Name      | Type | Owner | Size  | Description
-----+-----+-----+-----+-----+-----
 pgxc   | t_time_range_view | view | tbase | 0 bytes |
 public | tbase_view | view | tbase | 0 bytes | 我的视图
(2 rows)
```

```
postgres=# \dv+ tb*
```

```

                        List of relations
 Schema |      Name      | Type | Owner | Size  | Description
-----+-----+-----+-----+-----+-----
 public | tbase_view | view | tbase | 0 bytes | 我的视图
(1 row)
```

3.2.17、物化视图相关操作

■ 建立物化视图

```
postgres=# create MATERIALIZED VIEW tbase_count as select count(1) as num from tbase;
SELECT 1
```

```
postgres=# select * from tbase_count;
   num
-----
 10000
(1 row)
```

■ \d 查物化视图结构

```
postgres=# \d tbase_count
Materialized view "public.tbase_count"
 Column | Type | Modifiers
-----+-----+-----
  num   | bigint |
```

■ \d+ 查看物化视图结构（包含注释），包含创建物化视图的 sql 语句

```
postgres=# \d+ tbase_count
Materialized view "public.tbase_count"
 Column | Type | Modifiers | Storage | Stats target | Description
-----+-----+-----+-----+-----+-----
  num   | bigint |          | plain   |              |
View definition:
SELECT count(1) AS num
FROM tbase;
```

■ \dm 查看视图列表

```
postgres=# \dm
List of relations
 Schema | Name | Type | Owner
-----+-----+-----+-----
 pgxc   | tbase_sum | materialized view | tbase
 public | tbase_count | materialized view | tbase
(2 rows)
```

■ \dm+ 查看物化视图列表详细信息（包含注释），占用空间大小

```
postgres=# \dm+
List of relations
 Schema | Name | Type | Owner | Size | Description
-----+-----+-----+-----+-----+-----
 pgxc   | tbase_sum | materialized view | tbase | 8192 bytes |
 public | tbase_count | materialized view | tbase | 8192 bytes | tbase 总记录数
(2 rows)
```

■ \dm+ 显示某个模式下的所有物化视图

```
postgres=# \dm+ pgxc.*
```

List of relations					
Schema	Name	Type	Owner	Size	Description
pgxc	tbase_sum	materialized view	tbase	8192 bytes	

(1 row)

■ \dm+视图名 显示某个物化视图的详细信息

```
postgres=# \dm+ tbase_count
```

List of relations					
Schema	Name	Type	Owner	Size	Description
public	tbase_count	materialized view	tbase	8192 bytes	tbase 总记录数

(1 row)

■ \dm+通配符列出适配的物化视图

```
postgres=# \dm t*
```

List of relations			
Schema	Name	Type	Owner
pgxc	tbase_sum	materialized view	tbase
public	tbase_count	materialized view	tbase

(2 rows)

```
postgres=# \dm tbase_c*
```

List of relations			
Schema	Name	Type	Owner
public	tbase_count	materialized view	tbase

(1 row)

3.2.18、序列相关操作

■ 建立序列

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

■ \d 查看序列定义和使用情况


```
postgres=# \d tbase_seq
          Sequence "public.tbase_seq"
  Column  | Type  | Value
-----+-----+-----
last_value | bigint | 1
log_cnt   | bigint | 0
is_called | boolean | f
```

■ \ds 查看序列列表

```
postgres=# \ds
          List of relations
 Schema | Name      | Type  | Owner
-----+-----+-----+-----
 pgxc   | tbase_seq | sequence | tbase
 public | tbase_seq | sequence | tbase
(2 rows)
```

■ \ds+查看序列列表详细信息（包含注释），占用空间大小

```
postgres=# \ds+
          List of relations
 Schema | Name      | Type  | Owner | Size      | Description
-----+-----+-----+-----+-----+-----
 pgxc   | tbase_seq | sequence | tbase | 8192 bytes |
 public | tbase_seq | sequence | tbase | 8192 bytes | tbase 序列
(2 rows)
```

■ \ds+显示某个模式下的所有序列

```
postgres=# \ds+ pgxc.*
          List of relations
 Schema | Name      | Type  | Owner | Size      | Description
-----+-----+-----+-----+-----+-----
 pgxc   | tbase_seq | sequence | tbase | 8192 bytes |
(1 row)
```

■ \ds+序列名 显示某个序列的详细信息

```
postgres=# \ds+ tbase_seq
          List of relations
 Schema | Name      | Type  | Owner | Size      | Description
-----+-----+-----+-----+-----+-----
 public | tbase_seq | sequence | tbase | 8192 bytes | tbase 序列
(1 row)
```

■ \ds+通配符列出适配的序列

```
postgres=# \ds *_seq
          List of relations
 Schema |   Name   |   Type   | Owner
-----+-----+-----+-----
 pgxc   | tbase_seq | sequence | tbase
 public | tbase_seq | sequence | tbase
(2 rows)
```

```
postgres=# \ds t*_seq
          List of relations
 Schema |   Name   |   Type   | Owner
-----+-----+-----+-----
 public | tbase_seq | sequence | tbase
(1 row)
```

3.2.19、索引相关操作

■ 建立索引

```
postgres=# create unique index tbase_id_uidx on tbase(id);
CREATE INDEX
postgres=# create index tbase_mc_idx on tbase(mc);
CREATE INDEX
postgres=#
```

■ \di 查看索引列表

```
postgres=# \di
          List of relations
 Schema |      Name      | Type | Owner | Table
-----+-----+-----+-----+-----
 public | tbase_id_uidx | index | tbase | tbase
 public | tbase_mc_idx  | index | tbase | tbase
(2 rows)
```

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

```
postgres=# \di+
          List of relations
 Schema |      Name      | Type | Owner | Table |   Size   | Description
-----+-----+-----+-----+-----+-----+-----
 public | tbase_id_uidx | index | tbase | tbase | 8192 bytes | tbase 唯一索引
```

```
public | tbase_mc_idx | index | tbase | tbase | 8192 bytes |
(2 rows)
```

■ \di+显示某个模式下的所有索引--只能在 dn 上面查看索引大小

```
postgres=# \di+ public.*
```

```

                        List of relations
 Schema |      Name      | Type | Owner | Table |      Size      | Description
-----+-----+-----+-----+-----+-----+-----
 public | tbase_id_uidx | index | tbase | tbase | 8192 bytes | tbase 唯一索引
 public | tbase_mc_idx  | index | tbase | tbase | 8192 bytes |
(2 rows)
```

■ \di+索引名 显示某个索引的详细信息--只能在 dn 上面查看索引大小

```
postgres=# \di+ public.tbase_id_uidx
```

```

                        List of relations
 Schema |      Name      | Type | Owner | Table |      Size      | Description
-----+-----+-----+-----+-----+-----+-----
 public | tbase_id_uidx | index | tbase | tbase | 8192 bytes | tbase 唯一索引
(1 row)
```

■ \di+通配符列出适配的索引--只能在 dn 上面查看索引大小

```
postgres=# \di+ *idx
```

```

                        List of relations
 Schema |      Name      | Type | Owner | Table |      Size      | Description
-----+-----+-----+-----+-----+-----+-----
 pgxc   | order_main_id_idx | index | tbase | order_main | 8192 bytes |
 public | tbase_id_uidx    | index | tbase | tbase    | 8192 bytes | tbase 唯一索引
 public | tbase_mc_idx     | index | tbase | tbase    | 8192 bytes |
(3 rows)
```

```
postgres=# \di+ *uidx--只能在 dn 上面查看索引大小
```

```

                        List of relations
 Schema |      Name      | Type | Owner | Table |      Size      | Description
-----+-----+-----+-----+-----+-----+-----
 public | tbase_id_uidx | index | tbase | tbase | 8192 bytes | tbase 唯一索引
(1 row)
```

3.2.20、函数相关操作

■ 建立函数

```
postgres=# CREATE OR REPLACE FUNCTION tbase_fl(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.tbbase_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_fl
Result data type  | text
Argument data types | a_1 text
Type             | normal
Volatility        | volatile
Parallel          | unsafe
Owner             | tbase
Security          | invoker
Access privileges |
Language         | plpgsql
Source code       |          +
                  |      begin          +
                  |      return a_1;+
                  |      end;           +
                  |
Description      |

```

■ \df+显示某个模式下的所有索引

```

-[ RECORD 1 ]-----+-----
Schema           | public
Name             | tbase_fl
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 fl
```

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	
	beg
	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_fl
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+ *fl
```

```
List of functions
```

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

```

Schema              | public
Name                | tbase_fl
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         |

```

3.2.21、自定义数据类型相关操作

■ 建立数据类型

```

postgres=# CREATE TYPE bug_status AS ENUM ('new', 'open', 'closed');
CREATE TYPE

```


■ \dT 查看自定义数据类型列表

```
postgres=# \dT
```

List of data types		
Schema	Name	Description
-----+-----+-----		
pg_oracle	nvarchar2	oracle nvarchar2(length)
pg_oracle	varchar2	oracle varchar2(length)
public	bug_status	

■ \dT+查看自定义数据类型列表详细信息（包含 enum 类型的值）

```
postgres=# \dT+
```

List of data types						
Schema	Name	Internal name	Size	Elements	Access privileges	Description
-----+-----+-----+-----+-----+-----+-----						
pg_oracle	nvarchar2	nvarchar2	var			oracle nvarchar2(length)
pg_oracle	varchar2	varchar2	var			oracle varchar2(length)
public	bug_status	bug_status	4	new	+	
				open	+	
				closed		

■ \dT+显示某个模式下的所有自定义类型

```
postgres=# \dT+ public.*
```

List of data types						
Schema	Name	Internal name	Size	Elements	Access privileges	Description
-----+-----+-----+-----+-----+-----+-----						
public	bug_status	bug_status	4	new	+	
				open	+	
				closed		

■ \dT+自定义数据类型 显示某个数据类型的详细信息

```
postgres=# \dT+ bug_status
```

List of data types						
Schema	Name	Internal name	Size	Elements	Access privileges	Description
-----+-----+-----+-----+-----+-----+-----						
public	bug_status	bug_status	4	new	+	
				open	+	
				closed		

```
(1 row)
```

■ \dT+通配符列出适配的数据类型

```
postgres=# \dT+ bug_*
```

List of data types

Schema	Name	Internal name	Size	Elements	Owner	Access privileges	Description
public	bug_status	bug_status	4	new	+	tbase	
					open	+	
					closed		

3.2.22、存储过程语句相关操作

- 加载某个存储过程语言

```
postgres=# create language plpgsql;
CREATE LANGUAGE
```

- \dL 查看存储过程语言列表

```
postgres=# \dL
```

List of languages

Name	Owner	Trusted	Description
plpgsql	tbase	t	PL/pgSQL procedural language

(1 row)

3.2.23、列出表、视图和序列和它们相关的访问权限

- 显示所有对象的访问权限

```
postgres=# \dp
```

Access privileges

Schema	Name	Type	Access privileges	Column privileges	Policies
public	pg_stat_statements	view	tbase=arwdDxt/tbase+		
			=r/tbase		
public	tbase	table	tbase=arwdDxt/tbase+		
			pgxc=r/tbase		
public	tbase_seq	sequence	tbase=rwU/tbase +		
			pgxc=rwU/tbase		

(3 rows)

- \dp+某个对象名，只显示匹配的对象

```
postgres=# \dp tbase
```

Access privileges

Schema	Name	Type	Access privileges	Column privileges	Policies
--------	------	------	-------------------	-------------------	----------

```
public | tbase | table | tbase=arwdDxt/tbase+ |
      |      |      | pgxc=r/tbase      |
(1 row)
```

```
postgres=# \dp public.*
```

		Access privileges		
Schema	Name	Type	Access privileges	Column privileges Policies
public	pg_stat_statements	view	tbase=arwdDxt/tbase+ =r/tbase	
public	tbase	table	tbase=arwdDxt/tbase+ pgxc=r/tbase	
public	tbase_seq	sequence	tbase=rwU/tbase + pgxc=rwU/tbase	

(3 rows)

3.2.24、列出库或用户定义的配置

```
postgres=# \drds
```

List of settings	
Role	Database Settings
pgxc	log_statement=none
postgres	search_path="\$user", public, pgxc

(2 rows)

3.2.25、copy 命令的使用

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

- \copy to 将数据复制到本地文件中

```
postgres=# \copy tbase to '/data/tbase/tbase.txt';
postgres=# \! cat /data/tbase/tbase.txt
1      tbase
```

- \copy from 将本地文件复制到数据表中

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

3.2.26、copy FROM stdin 使用方法

```
postgres=# COPY tbase (id, mc) FROM stdin;
Enter data to be copied followed by a newline.
```

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

```
>> 1      tbase
>> 2      \N
>> 3      pgxc
>> \.

postgres=# select * from tbase;
 id | mc
----+-----
  1 | tbase
  2 |
  3 | pgxc
(3 rows)
```

3.2.27、打印当前查询缓冲区到标准输出

```
postgres=# select * from tbase;
 id | mc
----+-----
  1 | tbase
(1 row)
```

```
postgres=# \p
select * from tbase;
```

3.2.28、自定义显示格式

■ 配置 null 的显示代替字符串

```
postgres=# \pset null '(null)'
Null display is "(null)".
postgres=# insert into tbase values(2,null);
INSERT 0 1
postgres=# select * from tbase;
 id | mc
----+-----
  1 | tbase
  2 | (null)
(2 rows)
```

■ 不显示边框

```
postgres=# \pset border 0
Border style is 0.
postgres=# select * from tbase;
 id  mc
```

```
-- -----
1 tbase
2 (null)
(2 rows)
```

■ 只显示记录

```
postgres=# \set tuples_only
Showing only tuples.
postgres=# select * from tbase;
1 tbase
2 (null)
```

■ 列之间使用逗号分隔

```
postgres=# \set format unaligned
Output format is unaligned.
postgres=# \set fieldsep ,
Field separator is ",".
postgres=# select * from tbase;
1,tbase
2,(null)
```

3.2.29、显示 psql 内部操作

```
postgres=# \set ECHO_HIDDEN on
postgres=# \dt+ t
***** QUERY *****
SELECT n.nspname as "Schema",
       c.relname as "Name",
       CASE c.relkind WHEN 'r' THEN 'table' WHEN 'v' THEN 'view' WHEN 'm' THEN 'materialized view' WHEN
'i' THEN 'index' WHEN 'S' THEN 'sequence' WHEN 's' THEN 'special' WHEN 'f' THEN 'foreign table' WHEN 'p'
THEN 'table' END as "Type",
       pg_catalog.pg_get_userbyid(c.relowner) as "Owner",
       pg_catalog.pg_size_pretty(pg_catalog.pg_table_size(c.oid)) as "Size",
       pg_catalog.pg_size_pretty(pg_catalog.pg_allocated_table_size(c.oid)) as "Allocated Size",
       pg_catalog.obj_description(c.oid, 'pg_class') as "Description"
FROM pg_catalog.pg_class c
      LEFT JOIN pg_catalog.pg_namespace n ON n.oid = c.relnamespace
WHERE c.relkind IN ('r','p','s','')
      AND n.nspname !~ '^pg_toast'
      AND c.relname ~ '^(\d)$'
      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

3.2.30、重复执行上一条语句

```
postgres=# select count(1) from pg_stat_activity where state!='idle';
count
-----
      1
(1 row)
```

```
postgres=# \watch 1
Fri 28 Sep 2018 04:58:51 PM CST (every 1s)
```

```
count
-----
      1
(1 row)
```

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

```
count
-----
      1
(1 row)
```

上面的语句为每次自动查询活跃的进程数，相当于临时监控作用

3.2.31、sql 命令帮助查看

```
postgres=# \h
```

Available help:

ABORT	ALTER USER	CREATE ROLE
DROP INDEX	LOCK	CREATE RULE
ALTER AGGREGATE	ALTER USER MAPPING	
DROP LANGUAGE	MOVE	CREATE SCHEMA
ALTER COLLATION	ALTER VIEW	

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=#

4、TBase 数据库开发基础

4.1、TBase 的数据类型

4.1.1、数字类型

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

4.1.2、字符类型

名字	描述
character varying(<i>n</i>), varchar(<i>n</i>)	有限制的变长
character(<i>n</i>), char(<i>n</i>)	定长, 空格填充
text	1G

4.1.3、二进制数据类型

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

4.1.4、日期类型

名字	存储尺寸	描述	最小值	最大值	解析度
timestamp [(<i>p</i>)] [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 [(<i>p</i>)] 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 [(<i>p</i>)] [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 [(<i>p</i>)] 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 [<i>fields</i>] [(<i>p</i>)]	16 字节	时间间隔	-178000000 年	178000000 年	1 微秒 / 14 位

4.1.5、布尔类型

名字	存储字节	描述
boolean	1 字节	状态为真或假

4.1.6、更多的数据类型介绍

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

4.1.7、异构数据库类型对照表

4.1.7.1、与 oracle 对照表

Oracle	TBase
numeric	可以对应 TBase 的 smallint, integer, bigint, numeric(p,s)等多种数据类型。由于 smallint, Integer, bigint 的算术运算效率比 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	double precision
binary_float	real
binary_double	double precision
char	char
nchar	char
varchar2	varchar
nvarchar2	varchar
rowid	char(18)
urowid	varchar
long	text
clob	text
nclob	text
blob	bytea
bfile	bytea
Long raw	bytea
raw	bytea
date	Timestamp(0)
timestamp	Timestamp
Interval	interval

4.1.7.2、与 mysql 对照表

Mysql	TBase
-------	-------

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

4.1.7.3、与 sqlserver 对照表

SQLSERVER	TBase
smallint	smallint
int	int

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

4.2、自增列于序列的用法

4.2.1、序列的创建和访问

```
[tbase@VM_0_29_centos tbase_mgr]$ psql -p 15432 -U tbase -d postgres
psql (PostgreSQL 10 (TBase 2.01))
Type "help" for help.
```

--建立序列

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

--建立序列，不存在时才创建

```
postgres=# create sequence IF NOT EXISTS tbase_seq;
NOTICE:  relation "tbase_seq" already exists, skipping
CREATE SEQUENCE
```

--查看序列当前的使用状况

```
postgres=# \x
Expanded display is on.
postgres=# select * from tbase_seq ;
-[ RECORD 1 ]-
last_value | 1
log_cnt    | 0
is_called  | f
```

--获取序列的下一个值

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

--获取序列的当前值，这个需要在访问 nextval()后才能使用

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

你可以后下面的方式来获取序列当前使用到那一个值

```
postgres=# select last_value from tbase_seq ;
last_value
-----
3
```

--设置序列当前值

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

4.2.2、序列在 DML 中使用

```
postgres=# INSERT INTO t (id,nickname) VALUES(nextval('tbase_seq'),'TBase 好');
INSERT 0 1
postgres=# select * from t;
   id | nickname
-----+-----
    1 | 腾讯 TBase
    2 | TBase 好
(2 rows)
```

4.2.3、序列做为字段的默认值使用

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

4.2.4、序列做为字段类型使用

```
postgres=# drop table t;
DROP TABLE
postgres=# create table t (id serial not null,nickname text);
CREATE TABLE
postgres=# INSERT INTO t (nickname) VALUES('hello TBase');
INSERT 0 1
postgres=# select * from t;
   id |  nickname
-----+-----
    1 | hello TBase
(1 row)

postgres=#
```

4.2.5、删除序列

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

--删除序列，不存在时跳过

```
postgres=# drop sequence IF EXISTS tbase_seq;
NOTICE: sequence "tbase_seq" does not exist, skipping
DROP SEQUENCE
postgres=#
```

4.3、库/模式/表/索引/视图/物化视图等 DDL 操作

4.3.1、数据库管理

4.3.1.1、创建数据库

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

■ 默认参数创建数据库

```
postgres=# create database tbase_db;
CREATE DATABASE
```

■ 指定克隆库

```
postgres=# create database tbase_db_template TEMPLATE template0;
CREATE DATABASE
```

■ 指定所有者

```
postgres=# create role pgxz with login;
CREATE ROLE
postgres=# create database tbase_db_owner owner pgxz;
CREATE DATABASE
postgres=# \l+ tbase_db_owner
```

List of databases

Name	Owner	Encoding	Collate	Ctype	Access privileges	Size	Tablespace
Description							

```
-----+-----+-----+-----+-----+-----+-----+-----+
tbase_db_owner | pgxz | UTF8 | en_US.utf8 | en_US.utf8 | | 18 MB | pg_default |
(1 row)
```

■ 指定编码

```
postgres=# create database tbase_db_encoding ENCODING UTF8;
```

```
CREATE DATABASE
```

```
postgres=# \l+ tbase_db_encoding
```

```

                                List of databases
   Name   | Owner | Encoding | Collate | Ctype | Access privileges | Size | Tablespace |
Description
-----+-----+-----+-----+-----+-----+-----+-----+
tbase_db_encoding | tbase | UTF8 | en_US.utf8 | en_US.utf8 | | 18 MB | pg_default |
(1 row)
```

■ 指定排序规则

```
postgres=# create database tbase_db_lc_collate lc_collate 'C';
```

```
CREATE DATABASE
```

```
postgres=# \l+ tbase_db_lc_collate
```

```

                                List of databases
   Name   | Owner | Encoding | Collate | Ctype | Access privileges | Size | Tablespace |
Description
-----+-----+-----+-----+-----+-----+-----+-----+
tbase_db_lc_collate | tbase | UTF8 | C | en_US.utf8 | | 18 MB | pg_default |
(1 row)
```

■ 指定分组规则

```
postgres=# create database tbase_db_lc_ctype LC_CTYPE 'C';
```

```
CREATE DATABASE
```

```
postgres=# \l+ tbase_db_lc_ctype
```

```

                                List of databases
   Name   | Owner | Encoding | Collate | Ctype | Access privileges | Size | Tablespace |
Description
-----+-----+-----+-----+-----+-----+-----+-----+
tbase_db_lc_ctype | tbase | UTF8 | en_US.utf8 | C | | 18 MB | pg_default |
(1 row)
```

■ 配置数据可连接

```
postgres=# create database tbase_db_allow_connections ALLOW_CONNECTIONS true;
```

```
CREATE DATABASE
```

```
postgres=# select datallowconn from pg_database where datname='tbase_db_allow_connections';
datallowconn
-----
t
(1 row)
```

■ 配置连接数

```
postgres=# create database tbase_db_connlimit CONNECTION LIMIT 100;
CREATE DATABASE
postgres=# select datconlimit from pg_database where datname='tbase_db_connlimit';
datconlimit
-----
100
(1 row)
```

■ 配置数据库可以被复制

```
postgres=# create database tbase_db_istemplate is_template true;
CREATE DATABASE
postgres=# select datistemplate from pg_database where datname='tbase_db_istemplate';
datistemplate
-----
t
(1 row)
```

■ 多个参数一起配置

```
postgres=# create database tbase_db_mul owner pgxz CONNECTION LIMIT 50 template template0 encoding
'utf8' lc_collate 'C';
CREATE DATABASE
```

4.3.1.2、修改数据库配置

■ 修改数据库名称

```
postgres=# alter database tbase_db rename to tbase_db_new;
ALTER DATABASE
```

■ 修改连接数

```
postgres=# alter database tbase_db_new connection limit 50;
ALTER DATABASE
```

■ 修改数据库所有者


```
postgres=# alter database tbase_db_new owner to tbase;
ALTER DATABASE
```

■ 配置数据默认运行参行

```
postgres=# alter database tbase_db_new set search_path to public,pg_catalog,pg_oracle;
ALTER DATABASE
```

更多 set 的用法参考运维文档

■ Alter database 不支持的项目

项目	备注
encoding	编码
lc_collate	排序规则
lc_ctype	分组规则

4.3.1.3、删除数据库

```
postgres=# drop database tbase_db_new;
DROP DATABASE
```

删除数据库时，如果该数据库已经有 session 连接上来，则会提示如下错误

```
postgres=# drop database tbase_db_template;
ERROR:  database "tbase_db_template" is being accessed by other users
DETAIL:  There is 1 other session using the database.
```

使用下面方法可以把 session 断开，然后再删除

```
postgres=# select pg_terminate_backend(pid) from pg_stat_activity where datname='tbase_db_template';
pg_terminate_backend
-----
t
(1 row)
postgres=# drop database tbase_db_template;
DROP DATABASE
```

4.3.2、模式管理

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

4.3.2.1、创建模式

■ 标准语句

```
postgres=# create schema tbase;
CREATE SCHEMA
```

■ 扩展语法，不存在时才创建

```
postgres=# create schema if not exists tbase ;
NOTICE:  schema "tbase" already exists, skipping
CREATE SCHEMA
```

```
postgres=#
```

■ 指定所属用户

```
postgres=# create schema tbase_pgxz AUTHORIZATION pgxz;
CREATE SCHEMA
```

```
postgres=# \dn tbase_pgxz
```

List of schemas

Name	Owner
tbase_pgxz	pgxz

-----+-----

tbase_pgxz | pgxz

(1 row)

4.3.2.2、修改模式属性

■ 修改模式名

```
postgres=# alter schema tbase rename to tbase_new;
ALTER SCHEMA
```

■ 修改所有者

```
postgres=# alter schema tbase_pgxz owner to tbase;
ALTER SCHEMA
```

4.3.2.3、删除模式

```
postgres=# drop schema tbase_new;
DROP SCHEMA
```

当模式中存在对象时，则会删除失败，提示如下

```
postgres=# create table tbase_pgxz.t(id int);
NOTICE:  Replica identity is needed for shard table, please add to this table through "alter table" command.
CREATE TABLE
postgres=# drop schema tbase_pgxz;
ERROR:   cannot drop schema tbase_pgxz because other objects depend on it
DETAIL:  table tbase_pgxz.t depends on schema tbase_pgxz
HINT:   Use DROP ... CASCADE to drop the dependent objects too.
```

可以这样强制删除

```
postgres=# drop schema tbase_pgxz CASCADE;
NOTICE:  drop cascades to table tbase_pgxz.t
DROP SCHEMA
```

4.3.2.4、配置用户访问模式权限

普通用户对于某个模式下的对象访问除了访问对象要授权外，模式也需要授权

```
[tbase@VM_0_37_centos root]$ psql -U tbase
psql (PostgreSQL 10.0 TBase V2)
Type "help" for help.
```

```
postgres=# create table tbase.t(id int);
NOTICE:  Replica identity is needed for shard table, please add to this table through "alter table" command.
CREATE TABLE
```

--这里授权用户可以访问 tbase.t 表

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

```
postgres=# \q
[tbase@VM_0_37_centos root]$ psql -U pgxz
psql (PostgreSQL 10.0 TBase V2)
Type "help" for help.
```

--在没授权用户可以使用 tbase 模式前，还是访问不了

```
postgres=> select * from tbase.t;
ERROR:  permission denied for schema tbase
LINE 1: select * from tbase.t;
          ^
```

```
postgres=> \q
[tbase@VM_0_37_centos root]$ psql -U tbase
psql (PostgreSQL 10.0 TBase V2)
Type "help" for help.
```

postgres=# grant USAGE on SCHEMA tbase to pgxz;

GRANT

```
postgres=# \q
[tbase@VM_0_37_centos root]$ psql -U pgxz
```

```
psql (PostgreSQL 10.0 TBase V2)
Type "help" for help.
```

```
postgres=> select * from tbase.t;
 id
----
(0 rows)
```

```
postgres=>
```

4.3.2.5、配置访问模式的顺序

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

--当前连接用户

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

--总共三个模式

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

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

```
postgres=# show search_path ;
 search_path
-----
 "$user", public
(1 row)
```

--不指定模式创建数据表，则该表存放于第一个搜索模式下面

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

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

```
CREATE TABLE
```

```
postgres=# \dt
```

```
List of relations
```

```
Schema | Name | Type | Owner
```

```
-----+-----+-----+-----
```

```
tbase | t | table | tbase
```

```
(1 row)
```

--指定表位于某个模式下，不同模式下表名可以相同哦

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

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

```
CREATE TABLE
```

```
postgres=# \dt public.t
```

```
List of relations
```

```
Schema | Name | Type | Owner
```

```
-----+-----+-----+-----
```

```
public | t | table | tbase
```

```
(1 row)
```

```
postgres=# create table tbase_schema.t1(id int,mc text);
```

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

```
CREATE TABLE
```

```
postgres=#
```

--访问不在搜索路径对象时，需要写全路径

```
postgres=# select * from t1;
```

ERROR: relation "t1" does not exist

```
LINE 1: select * from t1;
```

```
^
```

```
postgres=# select * from tbase_schema.t1;
```

```
id | mc
```

```
----+----
```

```
(0 rows)
```

上面出错就是因为模式 tbase_schema 没有配置在 search_path 搜索路径中

4.3.3、创建和删除数据表

4.3.3.1、不用指定 shard key 建表方式

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

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

```
CREATE TABLE
```

```
postgres=# \d+ t_first_col_share
```

Table "public.t_first_col_share"

Column	Type	Modifiers	Storage
Stats target	Description		
id	integer	not null default nextval('t_first_col_share_id_seq'::regclass) plain	
nickname	text		extended
Has OIDs: no			
Distribute By SHARD(id)			
Location Nodes: ALL DATANODES			

4.3.3.2、指定 shard key 建表方式

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

```
CREATE TABLE
```

```
postgres=# \d+ t_appoint_col
```

Table "public.t_appoint_col"

Column	Type	Modifiers	Storage
Stats target	Description		
id	integer	not null default nextval('t_appoint_col_id_seq'::regclass) plain	
nickname	text		extended
Has OIDs: no			
Distribute By SHARD(nickname)			
Location Nodes: ALL DATANODES			

4.3.3.3、指定 group 建表方式

```
postgres=# create table t (id integer,nc text) distribute by shard (id) to group default_group;
```

```
CREATE TABLE
```

```
postgres=# \d+ t
```

Table "public.t"

Column	Type	Modifiers	Storage	Stats target	Description
id	integer		plain		
nc	text		extended		
Has OIDs: no					
Distribute By SHARD(id)					
Location Nodes: ALL DATANODES					

4.3.3.4、创建范围分区表

```
postgres=#create table t_range (f1 bigint,f2 timestamp default now(), f3 integer) partition by range (f3) begin (1)
step (50) partitions (3) distribute by shard(f1) to group default_group;
CREATE TABLE
postgres=# insert into t_range(f1,f3) values(1,1),(2,50),(3,100),(2,110);
INSERT 0 4
```

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

```
postgres=# insert into t_range(f1,f3) values(1,151);
```

这样的语句执行会出错,提示超出范围

```
postgres=# insert into t_range(f1,f3) values(1,151);
ERROR:  node:16385, error inserted value is not in range of partitioned table, please check the value of partition
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=#
```

4.3.3.5、创建时间范围分区表

```
postgres=# create table t_time_range(f1 bigint, f2 timestamp ,f3 bigint) partition by range (f2) begin (timestamp
without time zone '2017-09-01 0:0:0') step (interval '1 month') partitions (12) distribute by shard(f1) to group
default_group;
CREATE TABLE
postgres=# \d+ t_time_range
```

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

4.3.3.6、创建冷热分区表

```
postgres=# create table t_cold_hot_table(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,f2)
to group default_group ext_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

```
[pgxz@VM_0_29_centos install]$ psql -p 23003
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=# \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'
```

4.3.3.7、逻辑分区表

4.3.3.7.1、range 分区表

■ 创建主分区

```
postgres=# create table t_native_range (f1 bigint,f2 timestamp default now(), f3 integer) partition by range ( f2 )
distribute by shard(f1) to group default_group;
NOTICE: Replica identity is needed for shard table, please add to this table through "alter table" command.
CREATE TABLE
```

■ 建立两个子表

```
postgres=# create table t_native_range_201709 partition of t_native_range (f1 ,f2 , f3 ) for values from
('2017-09-01') to ('2017-10-01');
NOTICE: Replica identity is needed for shard table, please add to this table through "alter table" command.
CREATE TABLE
postgres=# create table t_native_range_201710 partition of t_native_range (f1 ,f2 , f3 ) for values from
('2017-10-01') to ('2017-11-01');
NOTICE: Replica identity is needed for shard table, please add to this table through "alter table" command.
CREATE TABLE
postgres=#
```

■ 查看表结构

```
postgres=# \d+ t_native_range
```

Table "tbase.t_native_range"							
Column	Type	Collation	Nullable	Default	Storage	Stats target	Description
f1	bigint				plain		
f2	timestamp without time zone			now()	plain		
f3	integer				plain		

Partition key: RANGE (f2)

Partitions: t_native_range_201709 FOR VALUES FROM ('2017-09-01 00:00:00') TO ('2017-10-01 00:00:00'),
t_native_range_201710 FOR VALUES FROM ('2017-10-01 00:00:00') TO ('2017-11-01 00:00:00')

Distribute By: SHARD(f1)

Location Nodes: ALL DATANODES

4.3.3.7.2、list 分区表

■ 创建主分区

```
postgres=# create table t_native_list(f1 bigserial not null,f2 text, f3 integer,f4 date) partition by list( f2 ) distribute
by shard(f1) to group default_group;
```

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

■ 建立两个子表,分别存入“广东”和“北京”

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

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

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

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

■ 查看表结构

```
postgres=# \d+ t_native_list
```

Table "tbase.t_native_list"						
Column	Type	Collation	Nullable	Default	Storage	Stats
target	Description					
f1	bigint		not null	nextval('t_native_list_f1_seq'::regclass)	plain	
f2	text				extended	
f3	integer				plain	
f4	date				plain	

Partition key: LIST (f2)

Partitions: t_list_bj FOR VALUES IN ('北京'),
t_list_gd FOR VALUES IN ('广东')

Distribute By: SHARD(f1)

Location Nodes: ALL DATANODES

```
postgres=#
```

4.3.3.7.3、多级分区表

■ 创建主表

```
postgres=# create table t_native_mul_list(f1 bigserial not null,f2 integer,f3 text,f4 text, f5 date) partition by list
(f3) distribute by shard(f1) to group default_group;
```

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

CREATE TABLE

■ 创建二级表

```
postgres=# create table t_native_mul_list_gd partition of t_native_mul_list for values in ('广东') partition by
range(f5);
```

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

CREATE TABLE

```
postgres=# create table t_native_mul_list_bj partition of t_native_mul_list for values in ('北京') partition by
range(f5);
```

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

CREATE TABLE

```
postgres=# create table t_native_mul_list_sh partition of t_native_mul_list for values in ('上海');
```

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

CREATE TABLE

■ 创建三级表

```
postgres=# create table t_native_mul_list_gd_201701 partition of t_native_mul_list_gd(f1,f2,f3,f4,f5) for values
from ('2017-01-01') to ('2017-02-01');
```

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

CREATE TABLE

```
postgres=# create table t_native_mul_list_gd_201702 partition of t_native_mul_list_gd(f1,f2,f3,f4,f5) for values
from ('2017-02-01') to ('2017-03-01');
```

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

CREATE TABLE

```
postgres=# create table t_native_mul_list_bj_201701 partition of t_native_mul_list_bj(f1,f2,f3,f4,f5) for values
from ('2017-01-01') to ('2017-02-01');
```

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

CREATE TABLE

```
postgres=# create table t_native_mul_list_bj_201702 partition of t_native_mul_list_bj(f1,f2,f3,f4,f5) for values
from ('2017-02-01') to ('2017-03-01');
```

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

CREATE TABLE

TBase 支持存在 1 级和 2 级分区混用，大家不需要都平级。

4.3.3.8、复制表

复制表是所有 dn 节点都存储一份相同的数据

```
postgres=# create table t_rep (id int,mc text) distribute by replication to group default_group;
```

```
CREATE TABLE
```

```
postgres=# insert into t_rep values(1,'TBase'),(2,'pgxz');
```

```
INSERT 0 2
```

```
postgres=# EXECUTE DIRECT ON (dn001) 'select * from t_rep';
```

```
id | mc
```

```
----+-----
```

```
1 | TBase
```

```
2 | pgxz
```

```
(2 rows)
```

```
postgres=# EXECUTE DIRECT ON (dn002) 'select * from t_rep';
```

```
id | mc
```

```
----+-----
```

```
1 | TBase
```

```
2 | pgxz
```

```
(2 rows)
```

我们可以看到所有节点都保存了一份相同的数据。

4.3.3.9、分区表性能测试

4.3.3.9.1、逻辑分区表 range 分区表

```
postgres=# \d+ t_native_range
```

Table "tbase.t_native_range"

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

```
-----+-----+-----+-----+-----+-----+-----+-----
```

```
---
```

```
f1 | bigint
```

```
f2 | timestamp without time zone
```

```
f3 | integer
```

```
Partition key: RANGE (f2)
```

```
Partitions: t_native_range_201709 FOR VALUES FROM ('2017-09-01 00:00:00') TO ('2017-10-01 00:00:00'),
```

```
t_native_range_201710 FOR VALUES FROM ('2017-10-01 00:00:00') TO ('2017-11-01 00:00:00'),
```

```
t_native_range_201711 FOR VALUES FROM ('2017-11-01 00:00:00') TO ('2017-12-01 00:00:00'),
```

```
t_native_range_201712 FOR VALUES FROM ('2017-12-01 00:00:00') TO ('2018-01-01 00:00:00'),
```

```
t_native_range_201801 FOR VALUES FROM ('2018-01-01 00:00:00') TO ('2018-02-01 00:00:00'),
```

```
t_native_range_201802 FOR VALUES FROM ('2018-02-01 00:00:00') TO ('2018-03-01 00:00:00'),
```

```
t_native_range_201803 FOR VALUES FROM ('2018-03-01 00:00:00') TO ('2018-04-01 00:00:00'),
```

```
t_native_range_201804 FOR VALUES FROM ('2018-04-01 00:00:00') TO ('2018-05-01 00:00:00'),
```

```
t_native_range_201805 FOR VALUES FROM ('2018-05-01 00:00:00') TO ('2018-06-01 00:00:00'),
t_native_range_201806 FOR VALUES FROM ('2018-06-01 00:00:00') TO ('2018-07-01 00:00:00'),
t_native_range_201807 FOR VALUES FROM ('2018-07-01 00:00:00') TO ('2018-08-01 00:00:00'),
t_native_range_201808 FOR VALUES FROM ('2018-08-01 00:00:00') TO ('2018-09-01 00:00:00')
```

Distribute By: SHARD(f1)

Location Nodes: ALL DATANODES

```
[tbase@VM_0_37_centos pgbench]$ pgbench -h 127.0.0.1 -p 11008 -d postgres -U tbase -c 4 -j 1 -n -M prepared -T
60 -r -f insert_t_native_range.sql > insert_t_native_range.log 2>&1
```

```
[tbase@VM_0_37_centos pgbench]$ tail insert_t_native_range.log
```

duration: 60 s

number of transactions actually processed: 13220

latency average = 18.162 ms

tps = 220.241817 (including connections establishing)

tps = 220.260935 (excluding connections establishing)

script statistics:

- statement latencies in milliseconds:

0.013 \set f1 random(1, 10000000)

0.006 \set f2 random(1, 364)

18.130 insert into t_native_range values(:f1, ('2017-09-01'::date+f2::integer)::timestamp, :f1);

4.3.3.9.2、时间范围分区表

```
postgres=# \d+ t_time_range
```

Table "tbase.t_time_range"							
Column	Type	Collation	Nullable	Default	Storage	Stats target	Description
f1	bigint				plain		
f2	timestamp without time zone				plain		
f3	bigint				plain		

Distribute By: SHARD(f1)

Location Nodes: ALL DATANODES

Partition By: RANGE(f2)

Of Partitions: 12

Start With: 2017-09-01

Interval Of Partition: 1 MONTH

```
postgres=#
```

```
[tbase@VM_0_37_centos pgbench]$ pgbench -h 127.0.0.1 -p 11008 -d postgres -U tbase -c 4 -j 1 -n -M prepared -T
60 -r -f insert_t_time_range.sql > insert_t_time_range.log 2>&1
```

```
[tbase@VM_0_37_centos pgbench]$ tail -20 insert_t_time_range.log
```

client 3 receiving

client 0 receiving

client 1 receiving

```

client 2 receiving
pghost: 127.0.0.1 pgport: 11008 nclients: 4 duration: 60 dbName: postgres
transaction type: insert_t_time_range.sql
scaling factor: 1
query mode: prepared
number of clients: 4
number of threads: 1
duration: 60 s
number of transactions actually processed: 55464
latency average = 4.328 ms
tps = 924.229841 (including connections establishing)
tps = 924.343708 (excluding connections establishing)
script statistics:
- statement latencies in milliseconds:
    0.011  \set f1 random(1, 10000000)
    0.006  \set f2 random(1, 364)
    4.307  insert into t_time_range values(:f1, ('2017-09-01'::date+f2::integer)::timestamp, :f1);

```

4.3.3.10、使用 IF NOT EXISTS

带 IF NOT EXISTS 关键字作用表示表不存在时才创建

```

postgres=# create table t(id int,mc text);
CREATE TABLE
postgres=# create table t(id int,mc text);
ERROR:  relation "t" already exists
postgres=# create table IF NOT EXISTS t(id int,mc text);
NOTICE:  relation "t" already exists, skipping
CREATE TABLE
postgres=#

```

4.3.3.11、指定模式创建表

```

postgres=# create table public.t(id int,mc text);
NOTICE:  Replica identity is needed for shard table, please add to this table through "alter table" command.
CREATE TABLE

```

4.3.3.12、使用将查询结果创建数据表

```

postgres=# create table t(id int,mc text) distribute by shard(mc);
NOTICE:  Replica identity is needed for shard table, please add to this table through "alter table" command.
CREATE TABLE
postgres=# insert into t values(1,'tbase');
INSERT 0 1
postgres=# create table t_as as select * from t;

```

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

INSERT 0 1

postgres=# select * from t_as;

```
id | mc
---+-----
 1 | tbase
(1 row)
```

postgres=# \d+ t

Table "tbase.t"

Column	Type	Collation	Nullable	Default	Storage	Stats target	Description
id	integer				plain		
mc	text				extended		

Distribute By: **SHARD(mc)**
Location Nodes: ALL DATANODES

postgres=# \d+ t_as

Table "tbase.t_as"

Column	Type	Collation	Nullable	Default	Storage	Stats target	Description
id	integer				plain		
mc	text				extended		

Distribute By: **SHARD(id)**
Location Nodes: ALL DATANODES

postgres=#

4.3.3.13、删除数据表

--删除当前模式下的数据表

```
postgres=# drop table t;
DROP TABLE
```

--删除某个模式下数据表

```
postgres=# drop table public.t;
DROP TABLE
```

--删除数据表，不存在时不执行，不报错

```
postgres=# drop table IF EXISTS t;
NOTICE: table "t" does not exist, skipping
DROP TABLE
```

--使用 **CASCADE** 无条件删除数据表

```
postgres=# create view tbase_schema.t1_view as select * from tbase_schema.t1 ;
CREATE VIEW
postgres=# drop table tbase_schema.t1 ;
ERROR:  cannot drop table tbase_schema.t1 because other objects depend on it
DETAIL:  view tbase_schema.t1_view depends on table tbase_schema.t1
HINT:   Use DROP ... CASCADE to drop the dependent objects too.
```

```
postgres=# drop table tbase_schema.t1 CASCADE;
NOTICE:  drop cascades to view tbase_schema.t1_view
DROP TABLE
postgres=#
```

4.3.4、创建和删除索引

4.3.4.1、普通索引

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

4.3.4.2、唯一索引

创建唯一索引

```
postgres=# create unique index t_first_col_share_id_uidx on t_first_col_share using btree(id);
CREATE INDEX
```

非 shard key 字段不能建立唯一索引

```
postgres=# create unique index t_first_col_share_nickname_uidx on t_first_col_share using btree(nickname);
ERROR:  Unique index of partitioned table must contain the hash/modulo distribution column.
```

4.3.4.3、表达式索引

```
postgres=# create table t_upper(id int,mc text);
NOTICE:  Replica identity is needed for shard table, please add to this table through "alter table" command.
CREATE TABLE
postgres=# create index t_upper_mc on t_upper(mc);
CREATE INDEX
```

```
postgres=# insert into t_upper select t,md5(t::text) from generate_series(1,10000) as t;
INSERT 0 10000
```



```
postgres=# analyze t_upper;
ANALYZE
```

```
postgres=# explain select * from t_upper where upper(mc)=md5('1');
QUERY PLAN
```

```
-----
Remote Fast Query Execution  (cost=0.00..0.00 rows=0 width=0)
Node/s: dn001, dn002
-> Seq Scan on t_upper  (cost=0.00..135.58 rows=25 width=37)
    Filter: (upper(mc) = 'c4ca4238a0b923820dcc509a6f75849b'::text)
(4 rows)
```

```
postgres=# create index t_upper_mc on t_upper(upper(mc));
CREATE INDEX
```

```
postgres=# explain select * from t_upper where upper(mc)=md5('1');
QUERY PLAN
```

```
-----
Remote Fast Query Execution  (cost=0.00..0.00 rows=0 width=0)
Node/s: dn001, dn002
-> Bitmap Heap Scan on t_upper  (cost=2.48..32.43 rows=25 width=37)
    Recheck Cond: (upper(mc) = 'c4ca4238a0b923820dcc509a6f75849b'::text)
-> Bitmap Index Scan on t_upper_mc  (cost=0.00..2.47 rows=25 width=0)
    Index Cond: (upper(mc) = 'c4ca4238a0b923820dcc509a6f75849b'::text)
(6 rows)
```

4.3.4.4、条件索引

```
postgres=# create table t_sex(id int,sex text) ;
NOTICE:  Replica identity is needed for shard table, please add to this table through "alter table" command.
CREATE TABLE
postgres=# create index t_sex_sex_idx on t_sex (sex);
CREATE INDEX
postgres=# insert into t_sex select t,'男' from generate_series(1,1000000) as t;
INSERT 0 1000000
postgres=# insert into t_sex select t,'女' from generate_series(1,100) as t;
INSERT 0 100
postgres=# analyze t_sex ;
ANALYZE
postgres=#
```

```
postgres=# explain select * from t_sex where sex ='女';
QUERY PLAN
```

```
-----
Remote Fast Query Execution  (cost=0.00..0.00 rows=0 width=0)
Node/s: dn001, dn002
```

```
-> Index Scan using t_sex_sex_idx on t_sex (cost=0.42..5.81 rows=67 width=8)
```

```
Index Cond: (sex = '女'::text)
```

```
(4 rows)
```

#索引对于条件为男的情况下无效

```
postgres=# explain select * from t_sex where sex = '男';
```

```
QUERY PLAN
```

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

```
Node/s: dn001, dn002
```

```
-> Seq Scan on t_sex (cost=0.00..9977.58 rows=500539 width=8)
```

```
Filter: (sex = '男'::text)
```

```
(4 rows)
```

#连接 dn 节点查看索引点用空间大，而且度数也高

```
[tbase@VM_0_37_centos shell]$ psql -p 11010
```

```
psql (PostgreSQL 10.0 TBase V2)
```

```
Type "help" for help.
```

```
postgres=# \di+
```

```
List of relations
```

Schema	Name	Type	Owner	Table	Size	Allocated Size	Description
tbase	t_sex_sex_idx	index	tbase	t_sex	14 MB	14 MB	
tbase	t_upper_mc	index	tbase	t_upper	14 MB	14 MB	

```
(2 rows)
```

```
postgres=# \q
```

```
[tbase@VM_0_37_centos shell]$ psql
```

```
psql (PostgreSQL 10.0 TBase V2)
```

```
Type "help" for help.
```

```
postgres=# drop index t_sex_sex_idx;
```

```
DROP INDEX
```

```
postgres=# create index t_sex_sex_idx on t_sex (sex) where sex = '女';
```

```
CREATE INDEX
```

```
postgres=# analyze t_sex;
```

```
ANALYZE
```

```
postgres=# explain select * from t_sex where sex = '女';
```

```
QUERY PLAN
```

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

```
Node/s: dn001, dn002
```

```
-> Index Scan using t_sex_sex_idx on t_sex (cost=0.14..6.69 rows=33 width=8)
(3 rows)
```

```
postgres=# explain select * from t_sex where sex='男';
          QUERY PLAN
```

```
-----
Remote Fast Query Execution (cost=0.00..0.00 rows=0 width=0)
Node/s: dn001, dn002
-> Seq Scan on t_sex (cost=0.00..9977.58 rows=500573 width=8)
    Filter: (sex = '男'::text)
(4 rows)
```

```
postgres=# \q
[tbase@VM_0_37_centos shell]$ psql -p 11010
psql (PostgreSQL 10.0 TBase V2)
Type "help" for help.
```

```
postgres=# \di+
                                List of relations
 Schema |      Name      | Type  | Owner | Table | Size  | Allocated Size | Description
-----+-----+-----+-----+-----+-----+-----+-----
 tbase  | t_sex_sex_idx  | index | tbase | t_sex  | 16 kB | 16 kB          |
 tbase  | t_upper_mc     | index | tbase | t_upper | 14 MB | 14 MB          |
(2 rows)
```

```
postgres=#
```

4.3.4.5、gist 索引

```
postgres=# create table t_trgm (id int,trgm text,no_trgm text);
NOTICE:  Replica identity is needed for shard table, please add to this table through "alter table" command.
CREATE TABLE
postgres=# create index t_trgm_trgm_idx on t_trgm using gist(trgm gist_trgm_ops);
CREATE INDEX
```

4.3.4.6、gin 索引

■ pg_trgm 索引

```
postgres=# drop index t_trgm_trgm_idx;
DROP INDEX
Time: 55.954 ms
postgres=# create index t_trgm_trgm_idx on t_trgm using gin(trgm gin_trgm_ops);
CREATE INDEX
```

■ jsonb 索引

```
postgres=# create table t_jsonb(id int,f_jsonb jsonb);
NOTICE:  Replica identity is needed for shard table, please add to this table through "alter table" command.
CREATE TABLE
postgres=# create index t_jsonb_f_jsonb_idx on t_jsonb using gin(f_jsonb);
CREATE INDEX
```

■ 数组索引

```
postgres=# create table t_array(id int, mc text[]);
NOTICE:  Replica identity is needed for shard table, please add to this table through "alter table" command.
CREATE TABLE
postgres=# insert into t_array select t,('{'||md5(t::text)||'}')::text[] from generate_series(1,1000000) as t;
INSERT 0 1000000
postgres=# analyze;
ANALYZE
postgres=# \timing
Timing is on.
```

```
postgres=# explain select * from t_array where mc @> ('{'||md5('1')||'}')::text[];
```

QUERY PLAN

```
-----
Remote Fast Query Execution  (cost=0.00..0.00 rows=0 width=0)
Node/s: dn001, dn002
->  Gather  (cost=1000.00..12060.25 rows=2503 width=61)
    Workers Planned: 2
    ->  Parallel Seq Scan on t_array  (cost=0.00..10809.95 rows=1043 width=61)
        Filter: (mc @> ('{c4ca4238a0b923820dcc509a6f75849b}'::cstring)::text[])
(6 rows)
```

Time: 4.105 ms

```
postgres=# select * from t_array where mc @> ('{'||md5('1')||'}')::text[];
```

```
id |          mc
---+-----
 1 | {c4ca4238a0b923820dcc509a6f75849b}
(1 row)
```

Time: 494.371 ms

```
postgres=# create index t_array_mc_idx on t_array using gin(mc);
CREATE INDEX
```

Time: 8195.387 ms (00:08.195)

```
postgres=# explain select * from t_array where mc @> ('{'||md5('1')||'}')::text[];
```

QUERY PLAN

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

Node/s: dn001, dn002

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

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

-> Bitmap Index Scan on t_array_mc_idx (cost=0.00..28.78 rows=2503 width=0)

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

(6 rows)

Time: 1.716 ms

postgres=# select * from t_array where mc @> ('{md5('1')}')::text[];

id | mc

----+-----

1 | {c4ca4238a0b923820dcc509a6f75849b}

(1 row)

Time: 2.980 ms

■ Btree_gin 任意字段索引

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

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

CREATE TABLE

postgres=#

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

INSERT 0 1000000

postgres=# create extension btree_gin;

CREATE EXTENSION

postgres=# create index gin_mul_gin_idx on gin_mul using gin(f1,f2,f3,f4,f5,f6);

CREATE INDEX

#单字段查询

postgres=# explain select * from gin_mul where f1=10;

QUERY PLAN

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

Node/s: dn002

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

Recheck Cond: (f1 = 10)

-> Bitmap Index Scan on gin_mul_gin_idx (cost=0.00..11.46 rows=194 width=0)

Index Cond: (f1 = 10)

(6 rows)

postgres=#

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

QUERY PLAN

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

Node/s: dn001, dn002

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

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

-> Bitmap Index Scan on gin_mul_gin_idx (cost=0.00..10.01 rows=1 width=0)

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

(6 rows)

postgres=# explain select * from gin_mul where f4='2364d9969c8b66402c9b7d17a6d5b7d3';

QUERY PLAN

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

Node/s: dn001, dn002

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

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

-> Bitmap Index Scan on gin_mul_gin_idx (cost=0.00..10.01 rows=1 width=0)

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

(6 rows)

postgres=# explain select * from gin_mul where f5=85375.30;

QUERY PLAN

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

Node/s: dn001, dn002

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

Recheck Cond: (f5 = 85375.30)

-> Bitmap Index Scan on gin_mul_gin_idx (cost=0.00..10.01 rows=1 width=0)

Index Cond: (f5 = 85375.30)

(6 rows)

#二个字段组合

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

QUERY PLAN

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

Node/s: dn001

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

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

```
-> Bitmap Index Scan on gin_mul_gin_idx (cost=0.00..18.00 rows=1 width=0)
      Index Cond: ((f1 = 2) AND (f3 = '2019-02-18 16:59:52.872523'::timestamp without time
zone))
(6 rows)
```

#三字段组合查询

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

QUERY PLAN

```
-----
Remote Fast Query Execution (cost=0.00..0.00 rows=0 width=0)
  Node/s: dn001
    -> Bitmap Heap Scan on gin_mul (cost=26.00..28.02 rows=1 width=90)
          Recheck Cond: ((f1 = 2) AND (f3 = '2019-02-18 16:59:52.872523'::timestamp without time zone)
AND (f6 = 'fa627dc16c2bd026150afa0453a0991d'::text))
          -> Bitmap Index Scan on gin_mul_gin_idx (cost=0.00..26.00 rows=1 width=0)
                Index Cond: ((f1 = 2) AND (f3 = '2019-02-18 16:59:52.872523'::timestamp without time zone)
AND (f6 = 'fa627dc16c2bd026150afa0453a0991d'::text))
(6 rows)
```

postgres=#

4.3.4.7、多字段索引

```
postgres=# create table t_mul_idx (f1 int,f2 int,f3 int,f4 int);
NOTICE:  Replica identity is needed for shard table, please add to this table through "alter table" command.
CREATE TABLE
Time: 308.109 ms
postgres=# create index t_mul_idx_idx on t_mul_idx(f1,f2,f3);
CREATE INDEX
Time: 108.734 ms
```

多字段使用注意事项

■ or 查询条件 bitmap scan 最多支持两个不同字段条件

```
postgres=# insert into t_mul_idx select t,t,t,t from generate_series(1,1000000) as t;
INSERT 0 1000000
postgres=# analyze ;
ANALYZE
```

```
postgres=# explain select * from t_mul_idx where f1=1 or f2=2 ;
```

QUERY PLAN

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

Node/s: dn001, dn002

-> Bitmap Heap Scan on t_mul_idx (cost=7617.08..7621.07 rows=2 width=16)

Recheck Cond: ((f1 = 1) OR (f2 = 2))

-> BitmapOr (cost=7617.08..7617.08 rows=2 width=0)

-> Bitmap Index Scan on t_mul_idx_idx (cost=0.00..2.43 rows=1 width=0)

Index Cond: (f1 = 1)

-> Bitmap Index Scan on t_mul_idx_idx (cost=0.00..7614.65 rows=1 width=0)

Index Cond: (f2 = 2)

(9 rows)

Time: 3.655 ms

postgres=# explain select * from t_mul_idx where f1=1 or f2=2 or f1=3 ;

QUERY PLAN

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

Node/s: dn001, dn002

-> Bitmap Heap Scan on t_mul_idx (cost=7619.51..7625.49 rows=3 width=16)

Recheck Cond: ((f1 = 1) OR (f2 = 2) OR (f1 = 3))

-> BitmapOr (cost=7619.51..7619.51 rows=3 width=0)

-> Bitmap Index Scan on t_mul_idx_idx (cost=0.00..2.43 rows=1 width=0)

Index Cond: (f1 = 1)

-> Bitmap Index Scan on t_mul_idx_idx (cost=0.00..7614.65 rows=1 width=0)

Index Cond: (f2 = 2)

-> Bitmap Index Scan on t_mul_idx_idx (cost=0.00..2.43 rows=1 width=0)

Index Cond: (f1 = 3)

(11 rows)

Time: 3.429 ms

postgres=# explain select * from t_mul_idx where f1=1 or f2=2 or f3=3 ;

QUERY PLAN

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

Node/s: dn001, dn002

-> Seq Scan on t_mul_idx (cost=0.00..12979.87 rows=3 width=16)

Filter: ((f1 = 1) OR (f2 = 2) OR (f3 = 3))

(4 rows)

Time: 1.679 ms

■ 如果返回字段全部在索引文件中，则只需要扫描索引，io 开销会更少

postgres=# explain select f1,f2,f3 from t_mul_idx where f1=1 ;

QUERY PLAN

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

Node/s: dn001

-> Index Only Scan using t_mul_idx_idx on t_mul_idx (cost=0.42..4.44 rows=1 width=12)

Index Cond: (f1 = 1)

(4 rows)

Time: 1.564 ms

■ 更新性能比单字段多索引文件要好

--多字段

postgres=# insert into t_simple_idx select t,t,t,t from generate_series(1,1000000) as t;

INSERT 0 1000000

Time: 7143.754 ms (00:07.144)

--单字段

postgres=# insert into t_mul_idx select t,t,t,t from generate_series(1,1000000) as t;

INSERT 0 1000000

Time: 4034.208 ms (00:04.034)

■ 多字段索引走非第一字段查询时性能比独立的单字段差

--多字段

postgres=# select * from t_mul_idx where f1=1;

f1 | f2 | f3 | f4

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

1 | 1 | 1 | 1

(1 row)

Time: 1.769 ms

postgres=# select * from t_mul_idx where f2=1;

f1 | f2 | f3 | f4

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

1 | 1 | 1 | 1

(1 row)

Time: 25.423 ms

postgres=# select * from t_mul_idx where f3=1;

f1 | f2 | f3 | f4

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

1 | 1 | 1 | 1

(1 row)

Time: 27.791 ms

--独立字段

```
postgres=# select * from t_simple_idx where f1=1;
 f1 | f2 | f3 | f4
----+----+---+---
  1 |  1 |  1 |  1
(1 row)
```

Time: 1.530 ms

```
postgres=# select * from t_simple_idx where f2=1;
 f1 | f2 | f3 | f4
----+----+----+---
   1 |   1 |   1 |   1
(1 row)
```

Time: 2.315 ms

```
postgres=# select * from t_simple_idx where f3=1;
 f1 | f2 | f3 | f4
----+----+----+---
   1 |   1 |   1 |   1
(1 row)
```

Time: 2.390 ms

4.3.4.8、删除索引

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

4.3.5、修改表结构

4.3.5.1、修改表名

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

4.3.5.2、给表或字段添加注释

```
postgres=# comment on table tbase is 'TBase 分布式关系型数据库系统';
COMMENT
postgres=# \dt+
```

List of relations					
Schema	Name	Type	Owner	Size	Description
+	+	+	+	+	

```
public | t_appoint_col | table | pgxz | 16 kB |
public | t_first_col_share | table | pgxz | 16 kB |
public | TBase | table | pgxz | 24 kB | TBase 分布式关系型数据库系统
(3 rows)
```

```
postgres=# comment on column tbase.nickname is 'TBase 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		TBase 昵称是大象

```
Has OIDs: no
```

```
Distribute By SHARD(id)
```

```
Location Nodes: ALL DATANODES
```

```
postgres=#
```

4.3.5.3、给表增加字段

```
postgres=# alter table tbase add column age integer;
```

```
ALTER TABLE
```

```
postgres=# \d+ tbase
```

Table "public.tbase"

Column	Type	Modifiers	Storage	Stats target	Description
id	integer	not null default nextval('t_id_seq'::regclass)	plain		
nickname	text		extended		TBase 昵称是大象
age	integer		plain		

```
Has OIDs: no
```

```
Distribute By SHARD(id)
```

```
Location Nodes: ALL DATANODES
```

4.3.5.4、修改字段类型

```
postgres=# alter table tbase alter column age type float8;
```

```
ALTER TABLE
```

```
postgres=# \d+ tbase
```

Table "public.tbase"

Column	Type	Modifiers	Storage	Stats target	Description
id	integer	not null default nextval('t_id_seq'::regclass)	plain		
nickname	text		extended		TBase 昵称是大象
age	double precision		plain		

```
Has OIDs: no
```

Distribute By SHARD(id)

Location Nodes: ALL DATANODES

postgres=#

4.3.5.5、修改字段默认值

postgres=# alter table tbase alter column age set default 0.0;

ALTER TABLE

postgres=# \d+ tbase

Table "public.tbase"						
Column	Type	Modifiers	Storage	Stats target	Description	
id	integer	not null default nextval('t_id_seq'::regclass)	plain			
nickname	text		extended			TBase 昵称是大象
age	double precision	default 0.0	plain			

Has OIDs: no

Distribute By SHARD(id)

Location Nodes: ALL DATANODES

4.3.5.6、删除字段

postgres=# alter table tbase drop column age;

ALTER TABLE

postgres=# \d+ tbase

Table "public.tbase"						
Column	Type	Modifiers	Storage	Stats target	Description	
id	integer	not null default nextval('t_id_seq'::regclass)	plain			
nickname	text		extended			TBase 昵称是大象

Has OIDs: no

Distribute By SHARD(id)

Location Nodes: ALL DATANODES

4.3.5.7、添加主键

postgres=# ALTER TABLE t ADD CONSTRAINT t_id_pkey PRIMARY KEY (id);

ALTER TABLE

postgres=# \d+ t

Table "tbase.t"						
Column	Type	Collation	Nullable	Default	Storage	Stats target Description
id	integer		not null		plain	
mc	text				extended	

Indexes:

"t_id_pkey" PRIMARY KEY, btree (id)

Distribute By: SHARD(id)

Location Nodes: ALL DATANODES

4.3.5.8、删除主键

```
postgres=# ALTER TABLE t DROP CONSTRAINT t_id_pkey ;
```

```
ALTER TABLE
```

```
postgres=# \d+ t
```

Table "tbase.t"

Column	Type	Collation	Nullable	Default	Storage	Stats target	Description
id	integer		not null		plain		
mc	text				extended		

Distribute By: SHARD(id)

Location Nodes: ALL DATANODES

4.3.5.9、添加外键

```
create table t_p(f1 int not null,f2 int ,primary key(f1));
```

```
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。

4.3.5.10、删除外键

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

```
ALTER TABLE
```

```
postgres=#
```

4.3.5.9、修改表所属模式

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

```
postgres=# alter table t set schema public;
ALTER TABLE
```

```
postgres=# \dt t
      List of relations
 Schema | Name | Type  | Owner
-----+-----+-----+-----
 public | t    | table | tbase
(1 row)
```

4.3.5.10、修改表所属用户

```
postgres=# \dt tbase
      List of relations
 Schema | Name  | Type  | Owner
-----+-----+-----+-----
 public | tbase | table | tbase
(1 row)
```

```
postgres=# alter table tbase owner to pgxz;
ALTER TABLE
```

```
postgres=# \dt tbase
      List of relations
 Schema | Name  | Type  | Owner
-----+-----+-----+-----
 public | tbase | table | pgxz
(1 row)
```

4.3.6、视图创建和删除

4.3.6.1、创建视图

```
postgres=# create view t_range_view as select * from t_range;
CREATE VIEW
```

```
postgres=# select * from t_range_view;
 f1 |          f2          | f3 | f4
----+-----+-----+----
  1 | 2017-09-27 23:17:39.674318 |   1 |
  2 | 2017-09-27 23:17:39.674318 |  50 |
  2 | 2017-09-27 23:17:39.674318 | 110 |
  1 | 2017-09-27 23:39:45.841093 | 151 |
  3 | 2017-09-27 23:17:39.674318 | 100 |
(5 rows)
```

数据类型重定义

```
postgres=# create view t_range_view as select f1,f2::date from t_range;
CREATE VIEW
postgres=# select * from t_range_view;
 f1 |      f2
----+-----
  1 | 2017-09-27
  2 | 2017-09-27
  2 | 2017-09-27
  1 | 2017-09-27
  3 | 2017-09-27
(5 rows)
```

数据类型重定义,以及取别名

```
postgres=# create view t_range_view as select f1,f2::date as mydate from t_range;
CREATE VIEW
postgres=# select * from t_range_view;
 f1 |    mydate
----+-----
  1 | 2017-09-27
  2 | 2017-09-27
  2 | 2017-09-27
  1 | 2017-09-27
  3 | 2017-09-27
(5 rows)
```

tbase 支持视图引用表或字段改名联动, 不受影响

```
postgres=# \d+ t_view

View "tbase.t_view"

Column | Type   | Collation | Nullable | Default | Storage | Description
-----+-----+-----+-----+-----+-----+-----
 id     | integer |           |          |          | plain   |
 mc     | text    |           |          |          | extended |
View definition:
SELECT t.id,
```

```
t.mc
FROM t;
```

```
postgres=# alter table t rename to t_new;
```

```
ALTER TABLE
```

```
Time: 62.875 ms
```

```
postgres=# alter table t_new rename mc to mc_new;
```

```
ALTER TABLE
```

```
Time: 22.081 ms
```

```
postgres=# \d+ t_view
```

```
View "tbase.t_view"
```

Column	Type	Collation	Nullable	Default	Storage	Description
id	integer				plain	
mc	text				extended	

```
View definition:
```

```
SELECT t_new.id,
       t_new.mc_new AS mc
FROM t_new;
```

4.3.6.2、删除视图

```
postgres=# drop table t;
```

```
DROP TABLE
```

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

```
CREATE TABLE
```

```
postgres=# create view t_view as select * from t;
```

```
CREATE VIEW
```

```
postgres=# create view t_view_1 as select * from t_view;
```

```
CREATE VIEW
```

```
postgres=# create view t_view_2 as select * from t_view;
```

```
CREATE VIEW
```

```
postgres=# drop view t_view_2;
```

```
DROP VIEW
```

```
#使用 cascade 强制删除依赖对象
```

```
postgres=# drop view t_view;
```

```
ERROR:  cannot drop view t_view because other objects depend on it
```

```
DETAIL:  view t_view_1 depends on view t_view
```

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

```
postgres=# drop view t_view cascade;
```

```
NOTICE:  drop cascades to view t_view_1
```

```
DROP VIEW
```


4.3.7、物化视图使用

4.3.7.1、创建物化视图

```
postgres=# CREATE MATERIALIZED VIEW t_range_mv AS select f1,f2::date from t_range;
SELECT 5
postgres=# select * from t_range_mv;
 f1 |      f2
----+-----
  1 | 2017-09-27
  2 | 2017-09-27
  2 | 2017-09-27
  1 | 2017-09-27
  3 | 2017-09-27
(5 rows)
```

4.3.7.2、访问物化视图

```
postgres=# select * from t_range_mv;
 f1 |      f2
----+-----
  1 | 2017-09-27
  2 | 2017-09-27
  2 | 2017-09-27
  1 | 2017-09-27
  3 | 2017-09-27
(5 rows)
```

```
postgres=# insert into t_range(f1,f3) values(5,10);
INSERT 0 1
```

```
postgres=# select * from t_range;
 f1 |      f2      | f3 | f4
----+-----+----+----
  1 | 2017-09-27 23:17:39.674318 |   1 |
  2 | 2017-09-27 23:17:39.674318 |  50 |
  5 | 2017-09-27 23:50:51.576173 |  10 |
  2 | 2017-09-27 23:17:39.674318 | 110 |
  1 | 2017-09-27 23:39:45.841093 | 151 |
  3 | 2017-09-27 23:17:39.674318 | 100 |
(6 rows)
```

4.3.7.3、增量数据刷新

```
postgres=# select * from t_range_mv ;
```

```
 f1 |      f2
----+-----
  1 | 2017-09-27
  2 | 2017-09-27
  2 | 2017-09-27
  1 | 2017-09-27
  3 | 2017-09-27
```

(5 rows)

```
postgres=# REFRESH MATERIALIZED VIEW t_range_mv;
```

```
REFRESH MATERIALIZED VIEW
```

```
postgres=# select * from t_range_mv ;
```

```
 f1 |      f2
----+-----
  1 | 2017-09-27
  2 | 2017-09-27
  5 | 2017-09-27
  2 | 2017-09-27
  1 | 2017-09-27
  3 | 2017-09-27
```

(6 rows)

注意：物化视图数据存储在 **cn** 节点上面，每个 **cn** 节点各有一份相同的数据

4.3.8、truncate 操作

truncate 功能用于对表数据进行快速清除，truncate 属于 ddl 级别，会给 truncate 表加上 ACCESS EXCLUSIVE 最高级别的锁

4.3.8.1、truncate 普通表

```
postgres=# truncate table t1;
```

```
TRUNCATE TABLE
```

#也可以一次 truncate 多个数据表

```
postgres=# truncate table t1,t2;
```

```
TRUNCATE TABLE
```

```
postgres=#
```

4.3.8.2、truncate 分区表

■ truncate 一个时间分区表

```
postgres=# \d+ t_time_range
```

Table "pgxz.t_time_range"

Column	Type	Modifiers	Storage	Stats target	Description
f1	bigint		plain		
f2	timestamp without time zone		plain		
f3	character varying(20)		extended		

Has OIDs: no

Distribute By SHARD(f1)

Location Nodes: dn001, dn002

Partition By: RANGE(f2)

Of Partitions: 12

Start With: 2017-09-01

Interval Of Partition: 1 MONTH

```
postgres=# select * from t_time_range;
```

```
f1 | f2 | f3
```

```
-----+-----+-----
```

```
1 | 2017-09-01 00:00:00 | tbase
```

```
2 | 2017-10-01 00:00:00 | pgxz
```

(2 rows)

```
postgres=# truncate t_time_range partition for ('2017-09-01'::timestamp without time zone);
```

TRUNCATE TABLE

```
postgres=# select * from t_time_range;
```

```
f1 | f2 | f3
```

```
-----+-----+-----
```

```
2 | 2017-10-01 00:00:00 | pgxz
```

(1 row)

```
postgres=#
```

■ truncate 一个数字分区表

```
postgres=# \d+ t_range
```

Table "pgxz.t_range"

Column	Type	Modifiers	Storage	Stats target	Description
f1	integer		plain		
f2	timestamp without time zone	default now()	plain		
f3	integer		plain		

Has OIDs: no

Distribute By SHARD(f1)

Location Nodes: dn01, dn02

Partition By: RANGE(f3)

```
# Of Partitions: 3
Start With: 1
Interval Of Partition: 50
```

```
postgres=# select * from t_range ;
 f1 |                f2                | f3
----+-----+-----
  1 | 2017-12-22 11:47:39.153234 |    1
  2 | 2017-12-22 11:47:39.153234 |   50
  2 | 2017-12-22 11:47:39.153234 |  110
  3 | 2017-12-22 11:47:39.153234 |  100
(4 rows)
```

```
postgres=# truncate t_range partition for (1);
```

```
TRUNCATE TABLE
```

```
postgres=# select * from t_range ;
 f1 |                f2                | f3
----+-----+-----
  2 | 2017-12-22 11:47:39.153234 |  110
  3 | 2017-12-22 11:47:39.153234 |  100
(2 rows)
```

```
postgres=#
```

4.3.9、外键创建与删除

4.3.9.1、外键创建

```
postgres=# create table t_p(f1 int not null,f2 int ,primary key(f1));
create table t_f(f1 int not null,f2 int );NOTICE:  Replica identity is needed for shard table, please add to this
table through "alter table" command.
CREATE TABLE
postgres=# create table t_f(f1 int not null,f2 int );
NOTICE:  Replica identity is needed for shard table, please add to this table through "alter table" command.
CREATE TABLE
postgres=# ALTER TABLE t_f ADD CONSTRAINT t_f_f1_fkey FOREIGN KEY (f1) REFERENCES t_p (f1);
ALTER TABLE
postgres=#
```

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。

4.3.9.2、删除外键

4.4、select 语句

4.4.1、访问函数

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

4.4.2、数据排序

--按某一列排序

```
postgres=# INSERT into tbase (nickname) VALUES('TBase 好');
INSERT 0 1
postgres=# INSERT into tbase (id,nickname) VALUES(1,'TBase 分布式数据库的时代来了');
INSERT 0 1
postgres=# select * from tbase order by id;
 id |          nickname
----+-----
  1 | hello TBase
  1 | TBase 分布式数据库的时代来了
  2 | TBase 好
(3 rows)
```

--按第一列排序

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

--按 id 升级排序，再按 nickname 降序排序

```
postgres=# select * from tbase order by id,nickname desc;
 id |          nickname
```

```
-----+-----
1 | TBase 分布式数据库的时代来了
1 | hello TBase
2 | TBase 好
(3 rows)
```

--效果与上面的语句一样

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

--随机排序

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

--计算排序

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

--排序都能用子查询，牛

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

--null 值排序结果处理

```
postgres=# insert into tbase values(4,null);
INSERT 0 1
```

null 值记录排在最前面

```
postgres=# select * from tbase order by nickname nulls first;
 id |      nickname
----+-----
  4 |
  1 | hello TBase
  1 | TBase 分布式数据库的时代来了
  2 | TBase 好
(4 rows)
```

null 值记录排在最后

```
postgres=# select * from tbase order by nickname nulls last;
 id |      nickname
----+-----
  1 | hello TBase
  1 | TBase 分布式数据库的时代来了
  2 | TBase 好
  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 collate 来实现汉字按拼音进行排序

4.4.3、where 条件使用

--单条件查询

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

```
id |      nickname
```

----+-----

1 | hello TBase

1 | TBase 分布式数据库的时代来了

--多条件 and

```
postgres=# select * from tbase where id=1 and nickname like '%h%';
```

```
id |      nickname
```

----+-----

1 | hello TBase

(1 row)

--多条件 or


```
postgres=# select * from tbase where id=2 or nickname like '%h%';
```

```
id |  nickname
```

```
----+-----
```

```
1 | hello TBase
```

```
2 | TBase 好
```

```
(2 rows)
```

--ilike 不区分大小写匹配

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

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

```
CREATE TABLE
```

```
postgres=# insert into t_ilike values(1,'tbase'),(2,'TBase');
```

```
INSERT 0 2
```

```
postgres=# select * from t_ilike where mc ilike '%tb%';
```

```
id |  mc
```

```
----+-----
```

```
1 | tbase
```

```
2 | TBase
```

```
(2 rows)
```

--where 条件也能支持子查询

```
postgres=# select * from tbase where id=(select (random()*2)::integer from tbase order by random() limit 1);
```

```
id | nickname
```

```
----+-----
```

```
(0 rows)
```

```
postgres=# select * from tbase where id=(select (random()*2)::integer from tbase order by random() limit 1);
```

```
id |          nickname
```

```
----+-----
```

```
1 | hello TBase
```

```
1 | TBase 分布式数据库的时代来了
```

```
(2 rows)
```

--null 值查询方法

```
postgres=# select * from tbase where nickname is null;
```

```
id | nickname
```

```
----+-----
```

```
4 |
```

```
(1 row)
```

```
postgres=# select * from tbase where nickname is not null;
```

```
id |          nickname
```

```
----+-----
```

```

1 | hello TBase
2 | TBase 好
1 | TBase 分布式数据库的时代来了
(3 rows)

```

--exists，只要有记录返回就为真

```

postgres=# create table t_exists1(id int,mc text);
NOTICE:  Replica identity is needed for shard table, please add to this table through "alter table" command.
CREATE TABLE

```

```

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

```

```

postgres=# create table t_exists2(id int,mc text);
NOTICE:  Replica identity is needed for shard table, please add to this table through "alter table" command.
CREATE TABLE

```

```

postgres=# insert into t_exists2 values(1,'tbase'),(1,'TBase');
INSERT 0 2

```

```

postgres=# select * from t_exists1 where exists(select 1 from t_exists2 where t_exists1.id=t_exists2.id);
 id | mc
----+-----
  1 | tbase
(1 row)

```

--exists 等价写法

```

postgres=# select t_exists1.* from t_exists1,(select distinct id from t_exists2) as t where t_exists1.id=t.id;;
 id | mc
----+-----
  1 | tbase
(1 row)

```

4.4.4、分页查询

--默认从第一条开始，返回一条记录

```

postgres=# select * from tbase limit 1;
 id | nickname
----+-----
  1 | hello TBase
(1 row)

```

--使用 **offset** 指定从第几条开始，0 表示第一条开始，返回 1 条记录

```
postgres=# select * from tbase limit 1 offset 0;
 id |  nickname
----+-----
  1 | hello TBase
(1 row)
```

--从第 3 条开始，返回二条记录

```
postgres=# select * from tbase limit 1 offset 2;
 id |  nickname
----+-----
  1 | TBase 分布式数据库的时代来了
(1 row)
```

--上面的语句没有使用排序，返回结果不可预知，使用 **order by** 可以获得一个有序的结果

```
postgres=# select * from tbase order by id limit 1 offset 2;
 id |  nickname
----+-----
  2 | TBase 好
(1 row)
```

4.4.5、合并多个查询结果

--不过虑重复的记录

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

--过虑重复的记录

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

(3 rows)

--每个子查询分布在合并结果中的使用

```
postgres=# select * from ( select * from tbase limit 1) as t union all select * from (select * from t_appoint_col limit 1) as t ;
```

```
id | nickname
```

```
----+-----
```

```
1 | hello TBase
```

```
1 | hello TBase
```

(2 rows)

4.4.6、返回两个结果的交集

```
postgres=# create table t_intersect1(id int,mc text);
```

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

CREATE TABLE

```
postgres=# insert into t_intersect1 values(1,'tbase'),(2,'tbase');
```

INSERT 0 2

```
postgres=# create table t_intersect2(id int,mc text);
```

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

CREATE TABLE

```
postgres=# insert into t_intersect2 values(1,'tbase'),(3,'tbase');
```

INSERT 0 2

```
postgres=# select * from t_intersect1 INTERSECT select * from t_intersect2;
```

```
id | mc
```

```
----+-----
```

```
1 | tbase
```

(1 row)

4.4.7、返回两个结果的差集

```
postgres=# create table t_except1(id int,mc text);
```

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

CREATE TABLE

```
postgres=# insert into t_except1 values(1,'tbase'),(2,'tbase');
```

INSERT 0 2

```
postgres=# create table t_except2(id int,mc text);
```

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

CREATE TABLE

```
postgres=# insert into t_except2 values(1,'tbase'),(3,'tbase');
INSERT 0 2
```

```
postgres=# select * from t_except1 except select * from t_except2;
 id | mc
----+-----
   2 | tbase
(1 row)
```

4.4.8、any 用法

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

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

CREATE TABLE

```
postgres=# insert into t_any values(1,'tbase'),(2,'TBase');
INSERT 0 2
```

```
postgres=# select * from t_any where id>any (select 1 union select 3);
 id | mc
----+-----
   2 | TBase
(1 row)
```

只需要大于其中一个值即为真

4.4.9、all 用法

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

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

CREATE TABLE

```
postgres=# insert into t_all values(2,'tbase'),(3,'TBase');
INSERT 0 2
```

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

需要大于所有值才为真

4.4.10、聚集查询

--统计记录数

```
postgres=# select count(1) from tbase;
count
-----
      3
(1 row)
```

--统计不重复值的记录表

```
postgres=# select count(distinct id) from tbase;
count
-----
      2
(1 row)
```

--求和

```
postgres=# select sum(id) from tbase;
sum
----
   4
(1 row)
```

--求最大值

```
postgres=# select max(id) from tbase;
max
----
   2
(1 row)
```

--求最小值

```
postgres=# select min(id) from tbase;
min
----
   1
(1 row)
```

--求平均值

```
postgres=# select avg(id) from tbase;
```

avg

```
-----
1.3333333333333333
(1 row)
```

4.4.11、多表关联

--内连接

```
postgres=# select * from tbase inner join t_appoint_col on tbase.id=t_appoint_col.id;
 id |          nickname          | id | nickname
----+-----+-----+-----
  1 | hello TBase                |  1 | hello TBase
  1 | TBase 分布式数据库的时代来了 |  1 | hello TBase
(2 rows)
```

--左外连接

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

--右外连接

```
postgres=# select * from tbase right join t_appoint_col on tbase.id=t_appoint_col.id;
 id |          nickname          | id | nickname
----+-----+-----+-----
  1 | TBase 分布式数据库的时代来了 |  1 | hello TBase
  1 | hello TBase                  |  1 | hello TBase
   |                              |  5 | Power TBase
(3 rows)
```

--全连接

```
postgres=# select * from tbase full join t_appoint_col on tbase.id=t_appoint_col.id;
 id |          nickname          | id | nickname
----+-----+-----+-----
  1 | hello TBase                |  1 | hello TBase
  2 | TBase 好                   |   | 
  1 | TBase 分布式数据库的时代来了 |  1 | hello TBase
   |                              |  5 | Power TBase
```

(4 rows)

4.4.12、聚合函数并发计算

■ 单核计算

```
postgres=# \timing
Timing is on.
postgres=# set max_parallel_workers_per_gather to 0;
SET
Time: 0.633 ms
postgres=# select count(1) from t_count;
 count
-----
20000000
(1 row)
```

Time: 3777.518 ms (00:03.778)

■ 二核并行

```
postgres=# set max_parallel_workers_per_gather to 2;
SET
Time: 0.478 ms
postgres=# select count(1) from t_count;
 count
-----
20000000
(1 row)
```

Time: 2166.481 ms (00:02.166)

■ 四核并行

```
postgres=# set max_parallel_workers_per_gather to 4;
SET
Time: 0.315 ms
postgres=# select count(1) from t_count;
 count
-----
20000000
(1 row)
```

Time: 1162.433 ms (00:01.162)

```
postgres=#
```


4.4.13、not in 中包含了 null，结果全为真

```
postgres=# create table t_not_in(id int,mc text);
NOTICE:  Replica identity is needed for shard table, please add to this table through "alter table" command.
CREATE TABLE
postgres=# insert into t_not_in values(1,'tbase'),(2,'pgxz');
INSERT 0 2
postgres=# select * from t_not_in where id not in (3,5);
 id | mc
----+-----
  1 | tbase
  2 | pgxz
(2 rows)
```

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

4.4.14、只查某个 dn 的数据

```
postgres=# create table t_direct(id int,mc text);
NOTICE:  Replica identity is needed for shard table, please add to this table through "alter table" command.
CREATE TABLE
postgres=# insert into t_direct values(1,'tbase'),(3,'pgxz');
INSERT 0 2
postgres=# EXECUTE DIRECT ON (dn001) 'select * from t_direct';
 id | mc
----+-----
  1 | tbase
(1 row)
```

```
postgres=# EXECUTE DIRECT ON (dn002) 'select * from t_direct';
 id | mc
----+-----
  3 | pgxz
(1 row)
```

```
postgres=# select * from t_direct ;
 id | mc
----+-----
  1 | tbase
  3 | pgxz
(2 rows)
```

```
postgres=#
```

4.4.15、特殊应用

--多行变成单行

```
postgres=# create table t_mulcol_tosimplecol(id int,mc text);
NOTICE:  Replica identity is needed for shard table, please add to this table through "alter table" command.
CREATE TABLE
```

```
postgres=# insert into t_mulcol_tosimplecol values(1,'tbase'),(2,'TBase');
INSERT 0 2
```

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

--一列变成多行

```
postgres=# create table t_col_to_mulrow(id int,mc text);
NOTICE:  Replica identity is needed for shard table, please add to this table through "alter table" command.
CREATE TABLE
```

```
postgres=# insert into t_col_to_mulrow values(1,'tbase,TBase');
INSERT 0 1
```

```
postgres=# select regexp_split_to_table((select mc from t_col_to_mulrow where id=1 limit 1),',');

 regexp_split_to_table
-----
 tbase
 TBase
(2 rows)
```

4.5、insert 语句

4.5.1、插入单条记录

指定所有字段

```
postgres=# insert into tbase(id,nickname) values(1,'hello TBase');
```

```
INSERT 0 1
```

指定某些字段，不指的有默认值的插入时带上默认值

```
postgres=# insert into tbase(nickname) values('TBase 好');
INSERT 0 1
```

不指定字段则默认为所有字段与建表时一致

```
postgres=# insert into tbase values(nextval('t_id_seq'::regclass),'TBase 好');
INSERT 0 1
```

字段顺序可以任意排列

```
postgres=# insert into tbase(nickname,id) values('TBase swap',5);
INSERT 0 1
```

使用 default 关键字，即值为建表时指定的默认值方式

```
postgres=# insert into tbase(id,nickname) values(default,'TBase default');
INSERT 0 1
```

上面五次插入记录后产生的数据

```
postgres=# select * from tbase;
 id |  nickname
----+-----
  1 | hello TBase
  2 | TBase 好
  5 | TBase swap
  3 | TBase 好
  4 | TBase default
(5 rows)
```

4.5.2、插入多数记录

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

4.5.3、使用子查询插入数据

```
postgres=# insert into tbase(id,nickname) values(1,(select relname from pg_class limit 1));
INSERT 0 1
postgres=# select * from tbase;
 id |      nickname
----+-----
  1 | pg_statistic
(1 row)
```

4.5.4、从另外一个表取数据进行批量插入

```
postgres=# insert into tbase(nickname) select relname from pg_class limit 3;
INSERT 0 3
postgres=# select * from tbase;
 id |      nickname
----+-----
  5 | pg_type
  6 | pg_toast_2619
  4 | pg_statistic
(3 rows)
```

4.5.5、大批量的生成数据

```
postgres=# insert into tbase select t,md5(random()::text) from generate_series(1,10000) as t;
INSERT 0 10000
postgres=# select count(1) from tbase;
 count
-----
 10000
(1 row)
```

4.5.6、返回插入数据，轻松获取插入记录的 serial 值

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

指定返回的字段

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

4.5.7、insert..update 更新

■ 使用 ON CONFLICT

```
postgres=# \d+ t
```

```

          Table "public.t"
  Column | Type   | Modifiers | Storage | Stats target | Description
-----+-----+-----+-----+-----+-----
 id      | integer |           | plain   |              |
 nc      | text    |           | extended |              |

```

Indexes:

"t_id_idx" UNIQUE, btree (id)

Has OIDs: no

Distribute By SHARD(id)

Location Nodes: ALL DATANODES

```
postgres=# select * from t;
 id | nc
----+-----
  1 | pgxz
(1 row)
```

```
postgres=# insert into t values(1,'pgxz') ON CONFLICT (id) DO UPDATE SET nc = 'tbase';
```

```
INSERT 0 1
```

```
postgres=# select * from t;
 id | nc
----+-----
  1 | tbase
(1 row)
```

4.6、update 语句

4.6.1、单表更新

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

--null 条件的表达方法

```
postgres=# update tbase set nickname = 'Good TBase' where nickname is null;
```

```
UPDATE 1
```

```
postgres=# select * from tbase;
```

```
id | nickname
----+-----
 2 | TBase 好
 1 | Hello TBase
 3 | Good TBase
```

```
(3 rows)
```

4.6.2、多表关联更新

```
postgres=# update tbase set nickname = 'Good TBase' from t_appoint_col where t_appoint_col.id=tbase.id;
```

```
UPDATE 1
```

```
postgres=# select * from tbase;
```

```
id | nickname
----+-----
 2 | TBase 好
 1 | Good TBase
```

```
(2 rows)
```

4.6.3、返回更新的数据

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

```
id | nickname
```

```
----+-----
 2 | TBase 好
```

```
(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

4.6.4、多列匹配更新

```
postgres=# update tbase set ( nickname , age ) = ( 'TBase 好' , (random()*2)::integer );
```

```
UPDATE 2
```

```
postgres=# select * from tbase;
```

```
id | nickname | age
----+-----+----
 1 | TBase 好 | 2
 2 | TBase 好 | 0
```

```
(2 rows)
```

4.6.5、shard key 不允许更新

```
postgres=# update tbase set id=8 where id=1;
```

```
ERROR:  Distribute column or partition column can't be updated in current version
```

4.7、delete 语句

4.7.1、带条件删除

```
postgres=# select * from tbase;
```

```
id |  nickname
----+-----
 2 | TBase 好
 1 | Hello TBase
 3 |
 4 | TBase good
(4 rows)
```

```
postgres=# delete from tbase where id=4;
```

```
DELETE 1
```

--null 条件的表达方式

```
postgres=# delete from tbase where nickname is null;
```

```
DELETE 1
```

```
postgres=# select * from tbase;
```

```
id |  nickname
----+-----
 2 | TBase 好
 1 | Hello TBase
(2 rows)
```

4.7.2、多表关联删除数据

```
postgres=# select * from tbase;
```

```
id |  nickname
----+-----
 2 | TBase 好
 1 | Hello TBase
(2 rows)
```

```
postgres=# set prefer_olap to on;
```

```
SET
```

```
postgres=# delete from tbase using t_appoint_col where tbase.id=t_appoint_col.id;
DELETE 1
postgres=# select * from tbase;
 id | nickname
----+-----
  2 | TBase 好
(1 row)
```

4.7.3、返回删除数据

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

returning 机制大在的降低的应用的复杂度

4.7.4、删除所有数据

```
postgres=# insert into tbase select t,random()::text from generate_series(1,100000) as t;
postgres=# \timing
Timing is on.
postgres=# delete from tbase ;
DELETE 100000
Time: 100.808 ms
```

使用 truncate 方法是全表删除更高效的方法

```
postgres=# insert into tbase select t,random()::text from generate_series(1,100000) as t;
INSERT 0 100000
Time: 13178.429 ms
postgres=# truncate table tbase;
TRUNCATE TABLE
Time: 24.242 ms
```

4.8、游标的使用

4.8.1、定义一个游标

```
postgres=# begin;
BEGIN
postgres=# DECLARE tbase_cur SCROLL CURSOR FOR SELECT * from tbase ORDER BY id;
DECLARE CURSOR
```


注意：游标需要放在一个事务中使用

4.8.2、提取下一行数据

```
postgres=# DECLARE tbase_cur SCROLL CURSOR FOR SELECT * from tbase ORDER BY id;
DECLARE CURSOR
```

```
postgres=# FETCH NEXT from tbase_cur ;
```

```
id |  nickname
```

```
----+-----
```

```
1 | hello TBase
```

```
(1 row)
```

```
postgres=# FETCH NEXT from tbase_cur ;
```

```
id |  nickname
```

```
----+-----
```

```
2 | TBase 好
```

```
(1 row)
```

4.8.3、提取前一行数据

```
postgres=# FETCH PRIOR from tbase_cur ;
```

```
id |  nickname
```

```
----+-----
```

```
1 | hello TBase
```

```
(1 row)
```

```
postgres=# FETCH PRIOR from tbase_cur ;
```

```
id |  nickname
```

```
----+-----
```

```
(0 rows)
```

4.8.4、提取最后一行

```
postgres=# FETCH LAST from tbase_cur ;
```

```
id |  nickname
```

```
----+-----
```

```
5 | TBase swap
```

```
(1 row)
```

4.8.5、提取第一行

```
postgres=# FETCH FIRST from tbase_cur ;
```

```
id | nickname
----+-----
 1 | hello TBase
(1 row)
```

4.8.6、提取该查询的第 x 行

```
postgres=# FETCH ABSOLUTE 2 from tbase_cur ;
id | nickname
----+-----
 2 | TBase 好
(1 row)
```

```
postgres=# FETCH ABSOLUTE -1 from tbase_cur ;
id | nickname
----+-----
 5 | TBase swap
(1 row)
```

```
postgres=# FETCH ABSOLUTE -2 from tbase_cur ;
id | nickname
----+-----
 4 | TBase default
(1 row)
```

X 为负数时则从尾部向上提

4.8.7、提取当前位置后的第 x 行

```
postgres=#  FETCH ABSOLUTE 1 from tbase_cur ;
id | nickname
----+-----
 1 | hello TBase
(1 row)
```

```
postgres=# FETCH RELATIVE 2 from tbase_cur ;
id | nickname
----+-----
 3 | TBase 好
(1 row)
```

```
postgres=# FETCH RELATIVE 2 from tbase_cur ;
id | nickname
----+-----
 5 | TBase swap
```

(1 row)

每提取一次数据，游标的位置都是会前行

4.8.8、向前提取 x 行数据

```
postgres=# FETCH FORWARD 2 from tbase_cur ;
```

```
id |  nickname
```

```
----+-----
```

```
1 | hello TBase
```

```
2 | TBase 好
```

(2 rows)

```
postgres=# FETCH FORWARD 2 from tbase_cur ;
```

```
id |  nickname
```

```
----+-----
```

```
3 | TBase 好
```

```
4 | TBase default
```

(2 rows)

4.8.9、向前提取剩下的所有数据

```
postgres=# FETCH FORWARD 2 from tbase_cur ;
```

```
id |  nickname
```

```
----+-----
```

```
1 | hello TBase
```

```
2 | TBase 好
```

(2 rows)

```
postgres=# FETCH FORWARD all from tbase_cur ;
```

```
id |  nickname
```

```
----+-----
```

```
3 | TBase 好
```

```
4 | TBase default
```

```
5 | TBase swap
```

(3 rows)

4.8.10、向后提取 x 行数据

```
postgres=# FETCH BACKWARD 2 from tbase_cur ;
```

```
id |  nickname
```

```
----+-----
```

```
5 | TBase swap
```

```
4 | TBase default
```

(2 rows)

4.8.11、向后提取剩下的所有数据

```
postgres=# FETCH BACKWARD all from tbase_cur;
```

```
id | nickname
```

```
----+-----
```

```
3 | TBase 好
```

```
2 | TBase 好
```

```
1 | hello TBase
```

(3 rows)

4.9、copy 的使用

COPY 用于 TBase 表和标准文件系统文件之间数据互相复制。COPY TO 可以把一个表的内容复制到一个文件，COPY FROM 可以从一个文件复制数据到一个表（数据以追加形式入库），COPY TO 也能复制一个 SELECT 查询的结果到一个文件。如果指定了一个列列表，COPY 将只把指定列的数据复制到文件或者从文件复制数据到指定列。如果表中有列不在列列表中，COPY FROM 将会为那些列插入默认值。使用 COPY 时 TBase 服务器直接从“本地”一个文件读取或者写入到一个文件。该文件必须是 TBase 用户（运行服务器的用户 ID）可访问的并且应该以服务器的视角来指定其名称。

4.9.1、实验表结构及数据

```
postgres=# \d+ t
```

Table "public.t"					
Column	Type	Modifiers	Storage	Stats target	Description
f1	integer	not null	plain		
f2	character varying(32)	not null	extended		
f3	timestamp without time zone	default now()	plain		
f4	integer		plain		

```
Has OIDs: yes
```

```
Distribute By SHARD(f1)
```

```
Location Nodes: ALL DATANODES
```

数据测试过程可以行再录入修改

```
postgres=# select * from t;
```

```
f1 | f2 | f3 | f4
```

```
----+-----+-----+-----
```

```
3 | pgxz | 2017-10-28 18:24:05.645691 |
```

```
1 | Tbase | | 7
```

```
2 | | 2017-10-28 18:24:05.643102 | 3
```

(3 rows)

4.9.2、copy to 用法详解--复制数据到文件中

4.9.2.1、导出所有列

```
postgres=# copy public.t to '/data/pgxz/t.txt';
COPY 3
postgres=# \! cat /data/pgxz/t.txt
1      Tbase   \N      7
2              2017-10-28 18:24:05.643102      3
3      pgxz    2017-10-28 18:24:05.645691      \N
```

默认生成的文件内容为表的所有列，列与列之间使用 tab 分隔开来。NULL 值生成的值为 \N

4.9.2.2、导出部分列

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

只导出 f1 和 f2 列

4.9.2.3、导出查询结果

```
postgres=# copy (select f2,f3 from public.t order by f3) to '/data/pgxz/t.txt';
COPY 3
postgres=# \! cat /data/pgxz/t.txt
              2017-10-28 18:24:05.643102
pgxz    2017-10-28 18:24:05.645691
Tbase   \N
postgres=#
```

查询可以是任何复杂查询

4.9.2.4、指定生成文件格式

■ 生成 csv 格式

```
postgres=# copy public.t to '/data/pgxz/t.txt' with csv;
COPY 3
postgres=# \! cat /data/pgxz/t.txt
```

```
1,Tbase,,7
2,pgxc,2017-10-28 18:24:05.643102,3
3,pgxz,2017-10-28 18:24:05.645691,
```

■ 生成二进制格式

```
postgres=# copy public.t to '/data/pgxz/t.txt' with binary;
COPY 3
postgres=# \l
postgres=# \! cat /data/pgxz/t.txt
PGCOPY
    Tbase
```

默认为 TEXT 格式

4.9.2.5、使用 delimiter 指定列与列之间的分隔符

```
postgres=# copy public.t to '/data/pgxz/t.txt' with delimiter '@';
COPY 3
postgres=# \! cat /data/pgxz/t.txt
1@Tbase@N@7
2@pgxc@2017-10-28 18:24:05.643102@3
3@pgxz@2017-10-28 18:24:05.645691@N
postgres=# copy public.t to '/data/pgxz/t.txt' with csv delimiter '@';
COPY 3
postgres=# \! cat /data/pgxz/t.txt
1@Tbase@@@7
2@pgxc@2017-10-28 18:24:05.643102@3
3@pgxz@2017-10-28 18:24:05.645691@
```

```
postgres=# copy public.t to '/data/pgxz/t.txt' with csv delimiter '@@';
ERROR: COPY delimiter must be a single one-byte character
```

```
postgres=# copy public.t to '/data/pgxz/t.txt' with binary delimiter '@';
ERROR: cannot specify DELIMITER in BINARY mode
```

指定分隔文件各列的字符。文本格式中默认是一个制表符，而 CSV 格式中默认是一个逗号。分隔符必须是一个单一的单字节字符，即汉字是不支持的。使用 binary 格式时不允许这个选项。

4.9.2.6、NULL 值的处理

```
postgres=# copy public.t to '/data/pgxz/t.txt' with NULL 'NULL';
COPY 3
postgres=# \! cat /data/pgxz/t.txt
1      Tbase      NULL      7
```

```

2      pgxc      2017-10-28 18:24:05.643102      3
3      pgxz      2017-10-28 18:24:05.645691      NULL

```

```

postgres=# copy public.t to '/data/pgxz/t.txt' with CSV NULL 'NULL';
COPY 3

```

```

postgres=# \! cat /data/pgxz/t.txt
1,Tbase,NULL,7
2,pgxc,2017-10-28 18:24:05.643102,3
3,pgxz,2017-10-28 18:24:05.645691,NULL

```

```

postgres=# copy public.t to '/data/pgxz/t.txt' with binary NULL 'NULL';
ERROR:  cannot specify NULL in BINARY mode
postgres=#

```

记录值为 NULL 时导出为 NULL 字符。使用 binary 格式时不允许这个选项。

4.9.2.7、生成列标题名

```

postgres=# copy public.t to '/data/pgxz/t.txt' with csv HEADER;
COPY 3
postgres=# \! cat /data/pgxz/t.txt
f1,f2,f3,f4
1,Tbase,,7
2,pgxc,2017-10-28 18:24:05.643102,3
3,pgxz,2017-10-28 18:24:05.645691,
postgres=# copy public.t to '/data/pgxz/t.txt' with  HEADER;
ERROR:  COPY HEADER available only in CSV mode

```

只有使用 CSV 格式时才允许这个选项。

4.9.2.8、导出 oids 系统列

```

postgres=# drop table t;
DROP TABLE
postgres=# CREATE TABLE t (
postgres(#      f1 integer NOT NULL,
postgres(#      f2 text NOT NULL,
postgres(#      f3 timestamp without time zone,
postgres(#      f4 integer
postgres(# )
postgres=# with oids DISTRIBUTE BY SHARD (f1);
CREATE TABLE
postgres=# copy t from '/data/pgxz/t.txt' with csv ;
COPY 3
postgres=# select * from t;

```

```

f1 |      f2      |      f3      | f4
---+-----+-----+---
 1 | Tbase      |              | 7
 2 | pg",      xc | 2017-10-28 18:24:05.643102 | 3
 3 | pgxz      | 2017-10-28 18:24:05.645691 |
(3 rows)

```

```
postgres=# copy t to '/data/pgxz/t.txt' with oids ;
```

```
COPY 3
```

```
postgres=# \! cat /data/pgxz/t.txt
```

```

35055  1      Tbase  \N      7
35056  2      pg",    xc    2017-10-28 18:24:05.643102      3
35177  3      pgxz    2017-10-28 18:24:05.645691      \N

```

创建表时使用了 with oids 才能使用 oids 选项

4.9.2.9、使用 quote 自定义引用字符

```
postgres=# copy t to '/data/pgxz/t.txt' with csv;
```

```
COPY 3
```

```
postgres=# \! cat /data/pgxz/t.txt
```

```

1,Tbase,,7
2,"pg"",    xc%",2017-10-28 18:24:05.643102,3
3,pgxz,2017-10-28 18:24:05.645691,

```

默认引用字符为“双引号”

```
postgres=# copy t to '/data/pgxz/t.txt' with quote '%' csv;
```

```
COPY 3
```

```
postgres=# \! cat /data/pgxz/t.txt
```

```

1,Tbase,,7
2,%pg",    xc%%%,2017-10-28 18:24:05.643102,3
3,pgxz,2017-10-28 18:24:05.645691,

```

上面指定了引用字符为百分号，系统自动把字段值为%的字符替换为双个%

```
postgres=# copy t to '/data/pgxz/t.txt' with quote '%';
```

```
ERROR: COPY quote available only in CSV mode
```

只有使用 CSV 格式时才允许这个选项。

```
postgres=# copy t to '/data/pgxz/t.txt' with quote '%%' csv;
```

```
ERROR: COPY quote must be a single one-byte character
```

```
postgres=#
```


引用字符必须是一个单一的单字节字符，即汉字是不支持的。

4.9.2.10、使用 escape 自定义逃逸符

```
postgres=# copy t to '/data/pgxz/t.txt' with quote '%' csv;
COPY 3
postgres=# \! cat /data/pgxz/t.txt
1,Tbase,,7
2,%pg",      xc%%,2017-10-28 18:24:05.643102,3
3,pgxz,2017-10-28 18:24:05.645691,
```

不指定 escape 逃逸符，默认和 QUOTE 值一样

```
postgres=# copy t to '/data/pgxz/t.txt' with quote '%' escape '@' csv;
COPY 3
postgres=# \! cat /data/pgxz/t.txt
1,Tbase,,7
2,%pg",      xc@%,2017-10-28 18:24:05.643102,3
3,pgxz,2017-10-28 18:24:05.645691,
```

指定逃逸符为 '@'

```
postgres=# copy t to '/data/pgxz/t.txt' with quote '%' escape '@@' csv;
ERROR: COPY escape must be a single one-byte character
```

这必须是一个单一的单字节字符。

```
postgres=# copy t to '/data/pgxz/t.txt' with quote '%' escape '@';
ERROR: COPY quote available only in CSV mode
```

只有使用 CSV 格式时才允许这个选项。

4.9.2.11、强制给某个列添加引用字符

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

指定 2 列强制添加引用字符

```
postgres=# copy t to '/data/pgxz/t.txt' (format 'csv',force_quote (f1,f4,f3,f2));
COPY 3
postgres=# \! cat /data/pgxz/t.txt
"1","Tbase",, "7"
"2","pg'",      xc%", "2017-10-28 18:24:05.643102", "3"
"3","pgxz", "2017-10-28 18:24:05.645691",
```

指定 4 列强制添加引用字符，字段的顺序可以任意排列

```
postgres=# copy t to '/data/pgxz/t.txt' (format 'text',force_quote (f1,f2,f3,f4));
ERROR: COPY force quote available only in CSV mode
postgres=#
```

只有使用 CSV 格式时才允许这个选项。

4.9.2.12、使用 encoding 指定导出文件内容编码

```
postgres=# copy t to '/data/pgxz/t.csv' (encoding utf8);
COPY 3
```

导出文件编码为 UTF8

```
postgres=# copy t to '/data/pgxz/t.csv' (encoding gbk);
COPY 3
postgres=#
```

导出文件编码为 gbk

使用 set_client_encoding to gbk;也可以将文件的内容设置为需要的编码，如下所示

```
postgres=# set client_encoding to gbk;
SET
postgres=# copy t to '/data/pgxz/t.csv' with csv ;
COPY 4
postgres=#
```

4.9.3、copy from 用法详解--复制文件内容到数据表中

4.9.3.1、导入所有列

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

```

postgres=# truncate table t;
TRUNCATE TABLE
postgres=# copy t from '/data/pgxz/t.txt';
COPY 3
postgres=# select * from t;
 f1 |          f2          |          f3          | f4
----+-----+-----+----
  1 | Tbase                |                      | 7
  2 | pg",          xc% | 2017-10-28 18:24:05.643102 | 3
  3 | pgxz                | 2017-10-28 18:24:05.645691 |
(3 rows)

```

4.9.3.2、导入部分指定列

```

postgres=# copy t(f1,f2) to '/data/pgxz/t.txt';
postgres=# \! cat /data/pgxz/t.txt
1      Tbase
2      pg",          xc%
3      pgxz
postgres=# truncate table t;
TRUNCATE TABLE
postgres=# copy t(f1,f2) from  '/data/pgxz/t.txt';
COPY 3
postgres=# select * from t;
 f1 |          f2          |          f3          | f4
----+-----+-----+----
  1 | Tbase                | 2017-10-30 11:54:16.559579 |
  2 | pg",          xc% | 2017-10-30 11:54:16.559579 |
  3 | pgxz                | 2017-10-30 11:54:16.560283 |
(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      Tbase
2      2017-10-28 18:24:05.643102      pg",          xc%
3      2017-10-28 18:24:05.645691      pgxz
postgres=# truncate table t;
TRUNCATE TABLE
postgres=# copy t(f1,f3,f2) from '/data/pgxz/t.txt';
COPY 3
postgres=# select * from t;
 f1 |          f2          |          f3          | f4
----+-----+-----+----
  1 | Tbase                |                      |
  2 | pg",          xc% | 2017-10-28 18:24:05.643102 |

```

```
3 | pgxz          | 2017-10-28 18:24:05.645691 |
(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      Tbase   \N      7
2      pg",      xc%    2017-10-28 18:24:05.643102      3
3      pgxz     2017-10-28 18:24:05.645691      \N
postgres=# copy t (f1,f2) from '/data/pgxz/t.txt';
ERROR:  extra data after last expected column
CONTEXT:  COPY t, line 1: "1      Tbase   \N      7"
```

数据文件的列表不能多于要导入的列数，否则会出错。

4.9.3.3、指定导入文件格式

```
postgres=# \! cat /data/pgxz/t.txt
1      Tbase   \N      7
2      pg",      xc%    2017-10-28 18:24:05.643102      3
3      pgxz     2017-10-28 18:24:05.645691      \N
postgres=# copy t from '/data/pgxz/t.txt' (format 'text');
COPY 3
```

TRUNCATE TABLE

```
postgres=# \! cat /data/pgxz/t.csv
1,Tbase,,7
2,"pg",,xc%,2017-10-28 18:24:05.643102,3
3,pgxz,2017-10-28 18:24:05.645691,
postgres=# copy t from '/data/pgxz/t.csv' (format 'csv');
COPY 3
```

postgres=# truncate table t;

TRUNCATE TABLE

```
postgres=# \! od -c /data/pgxz/t.bin
0000000 P G C O P Y \n 377 \r \n \0 \0 \0 \0 \0 \0
0000020 \0 \0 \0 \0 004 \0 \0 \0 004 \0 \0 \0 001 \0 \0 \0
0000040 005 T b a s e 377 377 377 377 \0 \0 \0 004 \0 \0
0000060 \0 \a \0 004 \0 \0 \0 004 \0 \0 \0 002 \0 \0 \0 016
0000100 p g ' " , x c % \0 \0
0000120 \0 \b \0 001 377 236 G w 213 ^ \0 \0 \0 004 \0 \0
0000140 \0 003 \0 004 \0 \0 \0 004 \0 \0 \0 003 \0 \0 \0 004
0000160 p g x z \0 \0 \0 \b \0 001 377 236 G w 225 {
0000200 377 377 377 377 377 377
0000206
postgres=# copy t from '/data/pgxz/t.bin' (format 'binary');
```

COPY 3

```
postgres=# select * from t;
```

f1	f2	f3	f4
1	Tbase		7
2	pg", xc%	2017-10-28 18:24:05.643102	3
3	pgxz	2017-10-28 18:24:05.645691	

(3 rows)

4.9.3.4、使用 delimiter 指定列与列之间的分隔符

```
postgres=# \! cat /data/pgxz/t.txt
```

```
1%Tbase%\N%7
```

```
2%pg", xc%\%2017-10-28 18:24:05.643102%3
```

```
3%pgxz%2017-10-28 18:24:05.645691%\N
```

```
postgres=# copy t from '/data/pgxz/t.txt' (format 'text',delimiter '%');
```

COPY 3

```
postgres=# select * from t;
```

f1	f2	f3	f4
1	Tbase		7
2	pg", xc%	2017-10-28 18:24:05.643102	3
3	pgxz	2017-10-28 18:24:05.645691	

(3 rows)

```
postgres=# \! cat /data/pgxz/t.csv
```

```
1%Tbase%%7
```

```
2%"pg", xc%"%2017-10-28 18:24:05.643102%3
```

```
3%pgxz%2017-10-28 18:24:05.645691%
```

```
postgres=# truncate table t;
```

```
TRUNCATE TABLE
```

```
postgres=# copy t from '/data/pgxz/t.csv' (format 'csv',delimiter '%');
```

COPY 3

```
postgres=# select * from t;
```

f1	f2	f3	f4
1	Tbase		7
2	pg", xc%	2017-10-28 18:24:05.643102	3
3	pgxz	2017-10-28 18:24:05.645691	

(3 rows)

4.9.3.5、NULL 值处理

```
postgres=# \! cat /data/pgxz/t.txt
```

```
1 Tbase NULL 7
```

```

2      pg",      xc%  2017-10-28 18:24:05.643102      3
3      pgxz      2017-10-28 18:24:05.645691      NULL
postgres=# copy t from '/data/pgxz/t.txt' (null 'NULL');
COPY 3
postgres=# select * from t;
 f1 |      f2      |      f3      | f4
----+-----+-----+----
  1 | Tbase      |              | 7
  2 | pg",      xc% | 2017-10-28 18:24:05.643102 | 3
  3 | pgxz      | 2017-10-28 18:24:05.645691 |
(3 rows)

```

将文件中的 NULL 字符串当成 NULL 值处理,SQL SERVER 导出来的文件中把 NULL 值替换成字符串 NULL,所以入库时可以这样处理一下,注意字符串是区分大小写,如下面语句导入数据就会出错

```

postgres=# copy t from '/data/pgxz/t.txt' (null 'null');
ERROR:  invalid input syntax for type timestamp: "NULL"
CONTEXT:  COPY t, line 1, column f3: "NULL"

```

4.9.3.6、自定义 quote 字符

```

postgres=# \! cat /data/pgxz/t.csv
1,Tbase,,7
2,%pg",      xc%%%,2017-10-28 18:24:05.643102,3
3,pgxz,2017-10-28 18:24:05.645691,

```

如果不配置 quote 字符则无法导入文件

```

postgres=# copy t from '/data/pgxz/t.csv' (format 'csv');
ERROR:  unterminated CSV quoted field
CONTEXT:  COPY t, line 4: "2,%pg",      xc%%%,2017-10-28 18:24:05.643102,3
3,pgxz,2017-10-28 18:24:05.645691,
"

```

```

postgres=# copy t from '/data/pgxz/t.csv' (format 'csv',quote '%');
COPY 3
postgres=#

```

```

postgres=# copy t from '/data/pgxz/t.csv' (format 'text',quote '%');
ERROR:  COPY quote available only in CSV mode

```

只有 csv 格式导入时才能配置 quote 字符

4.9.3.7、自定义 escape 字符

```

postgres=# \! cat /data/pgxz/t.csv

```

```

1,Tbase,,7
2,"pg'@",      xc%",2017-10-28 18:24:05.643102,3
3,pgxz,2017-10-28 18:24:05.645691,
postgres=# copy t from '/data/pgxz/t.csv' (format 'csv');
ERROR:  unterminated CSV quoted field
CONTEXT:  COPY t, line 4: "2,"pg'@",      xc%",2017-10-28 18:24:05.643102,3
3,pgxz,2017-10-28 18:24:05.645691,
"

```

不指定 `escape` 字符时，系统默认就是双写的 `quote` 字符，如双倍的“双引号”

```

postgres=# copy t from '/data/pgxz/t.csv' (format 'csv',escape '@');
COPY 3
postgres=# select * from t;
 f1 |          f2          |          f3          | f4
----+-----+-----+----
  1 | Tbase                |                      |  7
  2 | pg'",      xc%" | 2017-10-28 18:24:05.643102 |  3
  3 | pgxz              | 2017-10-28 18:24:05.645691 |
(3 rows)

postgres=#

```

4.9.3.8、csv header 忽略首行

```

postgres=# \! cat /data/pgxz/t.csv;
f1,f2,f3,f4
1,Tbase,,7
2,"pg'",      xc%",2017-10-28 18:24:05.643102,3
3,pgxz,2017-10-28 18:24:05.645691,
postgres=# copy t from '/data/pgxz/t.csv' (format 'csv');
ERROR:  invalid input syntax for integer: "f1"
CONTEXT:  COPY t, line 1, column f1: "f1"
postgres=# copy t from '/data/pgxz/t.csv' (format 'csv',header true);
COPY 3
postgres=# select * from t;
 f1 |          f2          |          f3          | f4
----+-----+-----+----
  1 | Tbase                |                      |  7
  2 | pg'",      xc%" | 2017-10-28 18:24:05.643102 |  3
  3 | pgxz              | 2017-10-28 18:24:05.645691 |
(3 rows)

```

如果不忽略首行，则系统会把首行当成数据，造成导入失败

4.9.3.9、导入 oid 列值

```
postgres=# \! cat /data/pgxz/t.txt
35242  1      Tbase  \N      7
35243  2      pg",    xc%    2017-10-28 18:24:05.643102      3
35340  3      pgxz    2017-10-28 18:24:05.645691      \N

postgres=# truncate table t;
TRUNCATE TABLE
postgres=# copy t from '/data/pgxz/t.txt' (oids true);
COPY 3
postgres=# select oid,* from t;
   oid | f1 |      f2      |      f3      | f4
-----+-----+-----+-----+-----
 35242 |  1 | Tbase         |              |  7
 35243 |  2 | pg",    xc% | 2017-10-28 18:24:05.643102 |  3
 35340 |  3 | pgxz        | 2017-10-28 18:24:05.645691 |
(3 rows)
```

4.9.3.10、使用 FORCE_NOT_NULL 把某列中空值变成长度为 0 的字符串，而不是 NULL 值

```
postgres=# truncate table t;
TRUNCATE TABLE
postgres=# \! cat '/data/pgxz/t.csv' ;
1,Tbase,,7
2,"pg",,    xc%",2017-10-28 18:24:05.643102,3
3,pgxz,2017-10-28 18:24:05.645691,
4,,2017-10-30 16:14:14.954213,4
postgres=# copy t from '/data/pgxz/t.csv' (format 'csv');
ERROR:  node:16386, error null value in column "f2" violates not-null constraint
DETAIL:  Failing row contains (4, null, 2017-10-30 16:14:14.954213, 4).
postgres=# select * from t where f2=";
 f1 | f2 | f3 | f4
----+----+----+----
(0 rows)
```

不使用 FORCE_NOT_NULL 处理的话就变成 NULL 值

```
postgres=# truncate table t;
TRUNCATE TABLE
postgres=# copy t from '/data/pgxz/t.csv' (format 'csv' ,FORCE_NOT_NULL (f2));
COPY 4
postgres=# select * from t where f2=";
 f1 | f2 |      f3      | f4
----+-----+-----+-----
  4 |    | 2017-10-30 16:14:14.954213 |  4
```


(1 row)

使用 FORCE_NOT_NULL 处理就变成长度为 0 的字符串

4.9.3.11、encoding 指定导入文件的编码

```
postgres=# \! enca -L zh_CN /data/pgxz/t.txt
Simplified Chinese National Standard; GB2312
```

```
postgres=# copy t from '/data/pgxz/t.txt';
COPY 4
postgres=# select * from t;
 f1 |      f2      |      f3      | f4
----+-----+-----+----
  1 | Tbase        |               | 7
  2 | pg",        xc% | 2017-10-28 18:24:05.643102 | 3
  3 | pgxz         | 2017-10-28 18:24:05.645691 | 
  4 |              | 2017-10-30 16:41:09.157612 | 4
```

(4 rows)

不指定导入文件的编码格式，则无法正确导入中文字符

```
postgres=# truncate table t;
TRUNCATE TABLE
postgres=# copy t from '/data/pgxz/t.txt' (encoding gbk);
COPY 4
postgres=# select * from t;
 f1 |      f2      |      f3      | f4
----+-----+-----+----
  1 | Tbase        |               | 7
  2 | pg",        xc% | 2017-10-28 18:24:05.643102 | 3
  3 | pgxz         | 2017-10-28 18:24:05.645691 | 
  4 | 腾讯         | 2017-10-30 16:41:09.157612 | 4
```

(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=# \! iconv -L zh_CN -x UTF-8 /data/pgxz/t.txt
postgres=# copy t from '/data/pgxz/t.txt';
COPY 4
postgres=# select * from t;
 f1 |      f2      |      f3      | f4
----+-----+-----+----
  1 | Tbase        |               | 7
```

```

2 | pg",          xc% | 2017-10-28 18:24:05.643102 | 3
3 | pgxz          | 2017-10-28 18:24:05.645691 |
4 | 腾讯          | 2017-10-30 16:41:09.157612 | 4
(4 rows)

```

```
postgres=#
```

4.10、json/jsonb 的使用

TBase 不只是一个分布式关系型数据库系统，同时它还支持非关系数据类型 json，JSON 数据类型是用来存储 JSON（JavaScript Object Notation）数据的。这种数据也可以被存储为 text，但是 JSON 数据类型的优势在于能强制要求每个被存储的值符合 JSON 规则。也有很多 JSON 相关的函数和操作符可以用于存储在这些数据类型中的数据。JSON 数据类型有 json 和 jsonb。它们接受完全相同的值集合作为输入。主要的实际区别是效率。json 数据类型存储输入文本的精准拷贝，处理函数必须在每次执行时必须重新解析该数据。而 jsonb 数据被存储在一种分解好的二进制格式中，它在输入时要稍慢一些，因为需要做附加的转换。但是 jsonb 在处理时要快很多，因为不需要解析。jsonb 也支持索引，这也是一个令人瞩目的优势。

4.10.1、json 应用

4.10.1.1、创建 json 类型字段表

```

postgres=# create table t_json(id int,f_json json);
NOTICE:  Replica identity is needed for shard table, please add to this table through "alter table" command.
CREATE TABLE

```

4.10.1.2、插入数据

```

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

```

4.10.1.3、通过键获得 JSON 对象域

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

4.10.1.4、以文本形式获取对象值

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

```
postgres=# select f_json ->>'col2' as col2 ,f_json ->> 'col3' as col3 from t_json where f_json ->> 'col3' is not null;
 col2  | col3
-----+-----
tbase | pgxz
(1 row)
```

4.10.2、jsonb 应用

4.10.2.1、创建 jsonb 类型字段表

```
postgres=# create table t_jsonb(id int,f_jsonb jsonb);
NOTICE:  Replica identity is needed for shard table, please add to this table through "alter table" command.
CREATE TABLE
postgres=#
```

4.10.2.2、插入数据

```
postgres=# insert into t_jsonb values(1,{'col1':1,'col2':"tbase"});
INSERT 0 1
postgres=# insert into t_jsonb values(2,{'col1':1,'col2':"tbase",'col3':"pgxz"});
INSERT 0 1
```

```
postgres=# select * from t_jsonb;
 id |          f_jsonb
----+-----
  1 | {"col1": 1, "col2": "tbase"}
```

```
2 | {"col1": 1, "col2": "tbase", "col3": "pgxz"}
(2 rows)
```

--jsonb 插入时会移除重复的键，如下所示

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

4.10.2.3、更新数据

--增加元素

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

--更新原来的元素

```
postgres=# update t_jsonb set f_jsonb = f_jsonb || '{"col2":"tbase"}'::jsonb where id=3;
UPDATE 1
```

```
postgres=# select * from t_jsonb;
 id |          f_jsonb
----+-----
  2 | {"col1": 1, "col2": "tbase", "col3": "pgxz"}
  1 | {"col1": 1, "col2": "tbase", "col3": "pgxz"}
  3 | {"col1": 1, "col2": "tbase"}
(3 rows)
```

--删除某个键

```
postgres=# update t_jsonb set f_jsonb = f_jsonb - 'col3';
UPDATE 3
```

```
postgres=# select * from t_jsonb;
 id |          f_jsonb
----+-----
  2 | {"col1": 1, "col2": "tbase"}
  1 | {"col1": 1, "col2": "tbase"}
  3 | {"col1": 1, "col2": "tbase"}
```

```
3 | {"col1": 1, "col2": "tbase"}
(3 rows)
```

4.10.2.4、jsonb_set()函数更新数据

jsonb_set(target jsonb, path text[], new_value jsonb, [create_missing boolean]) 说明: target 指要更新的数据源, path 指路径, new_value 指更新后的键值, create_missing 值为 true 表示如果键不存在则添加, create_missing 值为 false 表示如果键不存在则不添加。

```
postgres=# update t_jsonb set f_jsonb = jsonb_set( f_jsonb , '{col}' , '"pgxz"' , true ) where id=1;
UPDATE 1
postgres=# update t_jsonb set f_jsonb = jsonb_set( f_jsonb , '{col}' , '"pgxz"' , false ) where id=2;
UPDATE 1
postgres=# update t_jsonb set f_jsonb = jsonb_set( f_jsonb , '{col2}' , '"pgxz"' , false ) where id=3;
UPDATE 1
postgres=# select * from t_jsonb;
 id |          f_jsonb
----+-----
  1 | {"col": "pgxz", "col1": 1, "col2": "tbase"}
  2 | {"col1": 1, "col2": "tbase"}
  3 | {"col1": 1, "col2": "pgxz"}
(3 rows)
```

4.10.3、jsonb 函数应用

4.10.3.1、jsonb_each()将 json 对象转变键和值

```
postgres=# select  f_jsonb  from t_jsonb where id=1;
          f_jsonb
-----
 {"col": "pgxz", "col1": 1, "col2": "tbase"}
(1 row)

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

4.10.3.2、jsonb_each_text()将 json 对象转变文本类型的键和值

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

4.10.3.3、row_to_json()将一行记录变成一个 json 对象

```
postgres=# \d+ tbase
```

Table "public.tbase"

Column	Type	Collation	Nullable	Default	Storage	Stats target	Description
id	integer		not null		plain		
nickname	text				extended		

Indexes:

"tbase_pkey" PRIMARY KEY, btree (id)

Distribute By: SHARD(id)

Location Nodes: ALL DATANODES

```
postgres=# select * from tbase;
```

```
 id | nickname
```

```
----+-----
```

```
 1 | tbase
```

```
 2 | pgxz
```

```
(2 rows)
```

```
postgres=# select row_to_json(tbase) from tbase;
```

```
 row_to_json
```

```
-----
```

```
 {"id":1,"nickname":"tbase"}
```

```
 {"id":2,"nickname":"pgxz"}
```

```
(2 rows)
```

4.10.3.4、json_object_keys()返回一个对象中所有的键

```
postgres=# select * from json_object_keys((select f_jsonb from t_jsonb where id=1)::json);
```

```
 json_object_keys
```

```
-----
```

```
 col
```

```
 col1
```

```
 col2
```

(3 rows)

4.10.4、jsonb 索引使用

TBase 为文档 jsonb 提供了 GIN 索引,GIN 索引可以被用来有效地搜索在大量 jsonb 文档(数据)中出现的键或者键值对。

4.10.4.1、创建 jsonb 索引

```
postgres=# create index t_jsonb_f_jsonb_idx on t_jsonb using gin(f_jsonb);
CREATE INDEX
```

```
postgres=# \d+ t_jsonb
```

Table "public.t_jsonb"

Column	Type	Collation	Nullable	Default	Storage	Stats target	Description
id	integer				plain		
f_jsonb	jsonb				extended		

Indexes:

```
"t_jsonb_f_jsonb_idx" gin (f_jsonb)
```

Distribute By: SHARD(id)

Location Nodes: ALL DATANODES

4.10.4.2、测试查询的性能

```
postgres=# select count(1) from t_jsonb;
```

```
count
```

```
-----
10000000
```

(1 row)

```
postgres=# analyze t_jsonb;
```

```
ANALYZE
```

--没有索引开销

```
postgres=# select * from t_jsonb where f_jsonb @> '{"col1":9999}';
```

```
id | f_jsonb
```

```
-----+-----
9999 | {"col1": 9999, "col2": "9999"}
9999 | {"col1": 9999, "col2": "9999"}
9999 | {"col1": 9999, "col2": "9999"}
9999 | {"col1": 9999, "col2": "9999"}
9999 | {"col1": 9999, "col2": "9999"}
(5 rows)
```

Time: 2473.488 ms (00:02.473)

--有索引开销

```
postgres=# select * from t_jsonb where f_jsonb @> '{"col1":9999}';
```

id	f_jsonb
9999	{"col1": 9999, "col2": "9999"}
9999	{"col1": 9999, "col2": "9999"}
9999	{"col1": 9999, "col2": "9999"}
9999	{"col1": 9999, "col2": "9999"}
9999	{"col1": 9999, "col2": "9999"}

(5 rows)

Time: 217.968 ms

5、TBase 的进阶开发

5.1、TBase 事务控制

5.1.1、开始一个事务

```
postgres=# begin;
BEGIN
```

或者

```
postgres=# begin TRANSACTION ;
BEGIN
```

也可以定义事务的级别

```
postgres=# begin transaction isolation level read committed ;
BEGIN
```

5.1.2、提交事务

进程#1 访问

```
postgres=# begin;
BEGIN
```



```
postgres=# delete from tbase where id=5;
DELETE 1
postgres=#
postgres=# select * from tbase order by id;
 id |  nickname
----+-----
  1 | hello TBase
  2 | TBase 好
  3 | TBase 好
  4 | TBase default
```

TBase 也是完全支持 ACID 特性，没提交前开启另一个连接查询，你会看到是 5 条记录，这是 TBase 隔离性和多版本视图的实现，如下所示

进程#2 访问

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

进程#1 提交数据

```
postgres=# commit;
COMMIT
postgres=#
```

进程#2 再查询数据，这时能看到已经提交的数据了，这个级别叫“读已提交”

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

5.1.3、回滚事务

```
postgres=# begin;
BEGIN
postgres=# delete from tbase where id in (3,4);
DELETE 2
postgres=# select * from tbase;
 id |  nickname
----+-----
  1 | hello TBase
  2 | TBase 好
(2 rows)
```

```
postgres=# rollback;
ROLLBACK
```

Rollback 后数据又回来了

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

5.1.4、事务读一致性 REPEATABLE READ

这种事务级别表示事务自始至终读取的数据都是一致的，如下所示

```
#session1
```

```
postgres=# create table t_repeatable_read (id int,mc text);
NOTICE:  Replica identity is needed for shard table, please add to this table through "alter table" command.
CREATE TABLE
postgres=# insert into t_repeatable_read values(1,'tbase');
INSERT 0 1
```

```
postgres=# begin isolation level repeatable read ;
BEGIN
postgres=# select * from t_repeatable_read ;
 id |  mc
----+-----
  1 | tbase
```

(1 row)

#session2

```
postgres=# insert into t_repeatabl_read values(1,'pgxz');
```

```
INSERT 0 1
```

```
postgres=# select * from t_repeatabl_read;
```

```
id | mc
----+-----
 1 | tbase
 1 | pgxz
(2 rows)
```

#session1

```
postgres=# select * from t_repeatabl_read ;
```

```
id | mc
----+-----
 1 | tbase
(1 row)
```

```
postgres=#
```

5.1.5、行锁在事务中的运用

5.1.5.1、环境准备

```
postgres=# create table t_row_lock(id int,mc text,primary key (id));
```

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

```
CREATE TABLE
```

```
postgres=#
```

```
postgres=# insert into t_row_lock values(1,'tbase'),(2,'pgxz');
```

```
INSERT 0 2
```

```
postgres=# select * from t_row_lock;
```

```
id | mc
----+-----
 1 | tbase
 2 | pgxz
(2 rows)
```

5.1.5.2、直接 update 获取

#session1

```
postgres=# begin;
```

```

BEGIN
postgres=# set lock_timeout to 1;
SET
postgres=# update t_row_lock set mc='postgres' where mc='pgxz';
UPDATE 1
postgres=#

#session2

postgres=# begin;
BEGIN
postgres=# set lock_timeout to 1;
SET
postgres=# update t_row_lock set mc='postgresql' where mc='tbase';
UPDATE 1
postgres=#

```

上面 session1 与 session2 分别持有 mc=pgxz 行和 mc=tbase 的行锁

5.1.5.3、select...for update 获取

```

#session1

postgres=#
BEGIN
postgres=# set lock_timeout to 1;
SET
postgres=# select * from t_row_lock where mc='pgxz' for update;
 id | mc
----+-----
   2 | pgxz
(1 row)

#session2

postgres=# begin;
BEGIN
postgres=# set lock_timeout to 1;
SET
postgres=# select * from t_row_lock where mc='tbase' for update;
 id | mc
----+-----
   2 | pgxz
(1 row)

```

上面 session1 与 session2 分别持有 mc=pgxz 行和 mc=tbase 的行锁

5.1.5.4、与 mysql 获取行级锁的区别

```
mysql> select version();
```

```
+-----+
| version() |
+-----+
| 5.6.36    |
+-----+
```

```
1 row in set (0.00 sec)
```

```
#session1
```

```
mysql> begin;
```

```
Query OK, 0 rows affected (0.00 sec)
```

```
mysql> select * from t_row_lock where mc='pgxz' for update;
```

```
+---+-----+
| id | mc    |
+---+-----+
|  2 | pgxz  |
+---+-----+
```

```
1 row in set (0.00 sec)
```

```
#session2
```

```
mysql> select * from t_row_lock where mc='tbase' for update;
```

```
ERROR 1205 (HY000): Lock wait timeout exceeded; try restarting transaction
```

```
mysql>
```

这是因为 mysql 要使用行级锁需要有索引来配合使用，如下所示,使用 id 主键来获取行锁

```
#session1
```

```
mysql> begin;
```

```
Query OK, 0 rows affected (0.00 sec)
```

```
mysql> select * from t_row_lock where id=1 for update;
```

```
+---+-----+
| id | mc    |
+---+-----+
|  1 | tbase |
+---+-----+
```

```
1 row in set (0.00 sec)
```

```
#session2
```

```
mysql> begin;
Query OK, 0 rows affected (0.00 sec)

mysql> select * from t_row_lock where id=2 for update;
+----+-----+
| id | mc   |
+----+-----+
|  2 | pgxz |
+----+-----+
1 row in set (0.00 sec)
```

5.2、TBase 相关开发规范

5.2.1、命名规范

- 1)、DB object: database, schema, table, column, view, index, sequence, function, trigger 等名称:
 - (1) 建议使用小写字母、数字、下划线的组合。
 - (2) 建议不使用双引号即"包围，除非必须包含大写字母或空格等特殊字符。
 - (3) 长度不能超过 63 个字符。
 - (4) 不建议以 pg_ 开头或者 pgxc_（避免与系统 DB object 混淆），不建议以数字开头。
 - (5) 禁止使用 SQL 关键字例如 type, order 等。
- 2)、table 能包含的 column 数目,根据字段类型的不同,数目在 250 到 1600 之间;
- 3)、临时或备份的 DB object:table,view 等,建议加上日期,如 dba_ops.b2c_product_summay_2014_07_12 (其中 dba_ops 为 DBA 专用 schema);
- 4)、index 命名规则为: 普通索引为 表名_列名_idx, 唯一索引表名_列名_uidx, 如 student_name_idx, student_id_uidx。

5.2.2、COLUMN 设计

- 1)、建议能用数值类型的,就不用字符类型;
- 2)、建议能用 varchar(N) 就不用 char(N),以利于节省存储空间;
- 3)、建议能用 varchar(N) 就不用 text,varchar;
- 4)、建议使用 default NULL,而不用 default "",以节省存储空间;
- 5)、建议如有国际货业务的话,使用 timestamp with time zone(timestamptz),而不用 timestamp without time zone,避免时间函数在对于不同时区的时间点返回值不同,也为业务国际化扫清障碍;
- 6)、建议使用 NUMERIC(precision, scale)来存储货币金额和其它要求精确计算的数值,而不建议使用 real, double precision;

5.2.3、Constraints 设计

- 1)、建议每个 table 都使用 shard key 做为主键或者唯一索引。
- 2)、建议建表时一步到位把主键或者唯一索引也一起建立。

3)、注意, 非 shard key 不可以建立 primary key 或者 unique index。

5.2.4、Index 设计

- 1)、TBase 提供的 index 类型: B-tree, Hash, GiST (Generalized Search Tree), SP-GiST (space-partitioned GiST), GIN (Generalized Inverted Index), BRIN (Block Range Index), 目前不建议使用 Hash, 通常情况下使用 B-tree;
- 2)、建议 create 或 drop index 时, 加 CONCURRENTLY 参数, 这是个好习惯, 达到与写入数据并发的效果;
- 3)、建议对于频繁 update, delete 的包含于 index 定义中的 column 的 table, 用 create index CONCURRENTLY, drop index CONCURRENTLY 的方式进行维护其对应 index;
- 4)、建议用 unique index 代替 unique constraints, 便于后续维护;
- 5)、建议对 where 中带多个字段 and 条件的高频 query, 参考数据分布情况, 建多个字段的联合 index;

6)、建议对固定条件的 (一般有特定业务含义) 且选择比好 (数据占比低) 的 query, 建带 where 的 Partial Indexes;

```
select * from test where status=1 and col=?; -- 其中 status=1 为固定的条件
create index on test (col) where status=1;
```

7)、建议对经常使用表达式作为查询条件的 query, 可以使用表达式或函数索引加速 query;

```
select * from test where exp(xxx);
create index on test ( exp(xxx) );
```

8)、建议不要建过多 index, 一般不要超过 6 个, 核心 table (产品, 订单) 可适当增加 index 个数。

5.2.5、关于 NULL

- 1)、NULL 的判断: IS NULL, IS NOT NULL;
- 2)、注意 boolean 类型取值 true, false, NULL;
- 3)、小心 NOT IN 集合中带有 NULL 元素;

```
postgres=# select * from tbase;
```

```
id |  nickname
----+-----
 1 | hello TBase
 2 | TBase 好
 3 | TBase 好
 4 | TBase default
(4 rows)
```

```
postgres=# select * from tbase where id not in (null);
```

```
id | nickname
----+-----
(0 rows)
```

4)、建议对字符串型 NULL 值处理后, 进行 || 操作;

```
postgres=# select id,nickname from tbase limit 1;
```

```
id |  nickname
```

```
----+-----
 1 | hello TBase
(1 row)
```

```
postgres=# select id,nickname||null from tbase limit 1;
 id | ?column?
```

```
----+-----
 1 |
(1 row)
```

```
postgres=# select id,nickname||coalesce(null,"") from tbase limit 1;
 id | ?column?
```

```
----+-----
 1 | hello TBase
(1 row)
```

5)、建议使用 `count(1)` 或 `count(*)` 来统计行数，而不建议使用 `count(col)` 来统计行数，因为 `NULL` 值不会计入；

注意: `count(多列列名)`时，多列列名必须使用括号，例如 `count((col1,col2,col3))`；注意多列的 `count`，即使所有列都为 `NULL`，该行也被计数，所以效果与 `count(*)` 一致；

```
postgres=# select * from tbase ;
 id |  nickname
```

```
----+-----
 1 | hello TBase
 2 | TBase 好
 5 |
 3 | TBase 好
 4 | TBase default
(5 rows)
```

```
postgres=# select count(1) from tbase;
 count
```

```
-----
      5
(1 row)
```

```
postgres=# select count(*) from tbase;
 count
```

```
-----
      5
(1 row)
```

```
postgres=# select count(nickname) from tbase;
```



```
count
-----
      4
(1 row)
```

```
postgres=# select count((id,nickname)) from tbase;
count
-----
      5
(1 row)
```

6)、count(distinct col) 计算某列的非 NULL 不重复数量，NULL 不被计数；

注意: count(distinct (col1,col2,...)) 计算多列的唯一值时，NULL 会被计数，同时 NULL 与 NULL 会被认为是相同的；

```
postgres=# select count(distinct nickname) from tbase;
count
-----
      3
(1 row)
```

```
postgres=# select count(distinct (id,nickname)) from tbase;
count
-----
      5
(1 row)
```

7)、两个 null 的对比方法

```
postgres=# select null is not distinct from null as TBasenull;
TBasenull
-----
t
(1 row)
```

5.2.6、开发相关规范

1)、 建议对 DB object 尤其是 COLUMN 加 COMMENT，便于后续新人了解业务及维护；

注释前后的数据表可读性对比，有注释的一看就明白

```
postgres=# \d+ TBase_main
Table "public.tbase_main"
```

Column	Type	Modifiers	Storage	Stats target	Description
id	integer		plain		
mc	text		extended		

Indexes:

"TBase_main_id_uidx" UNIQUE, btree (id)

Has OIDs: no

Distribute By SHARD(id)

Location Nodes: ALL DATANODES

postgres=# comment on column TBase_main.id is 'id 号';

COMMENT

postgres=# comment on column TBase_main.mc is '产品名称';

COMMENT

postgres=# \d+ TBase_main

Table "public.tb主se_main"

Column	Type	Modifiers	Storage	Stats target	Description
id	integer		plain		id 号
mc	text		extended		产品名称

Indexes:

"TBase_main_id_uidx" UNIQUE, btree (id)

Has OIDs: no

Distribute By SHARD(id)

Location Nodes: ALL DATANODES

2)、建议非必须时避免 select *, 只取所需字段, 以减少包括不限于网络带宽消耗

postgres=# explain (verbose) select * from tb主se_main where id=1;

QUERY PLAN

```

Index Scan using TBase_main_id_uidx on public.tb主se_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 tb主se_main where id=1;

QUERY PLAN

```

Index Scan using TBase_main_id_uidx on public.tb主se_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

3)、建议 update 时尽量做 $\lt \gt$ 判断,比如 `update table_a set column_b = c where column_b $\lt \gt$ c;`

```
postgres=# update tbase_main set mc='TBase' ;
```

```
UPDATE 1
```

```
postgres=# select xmin,* from tbase_main;
```

```
xmin | id | mc
-----+-----+-----
2562 | 1 | TBase
(1 row)
```

```
postgres=# update tbase_main set mc='TBase' ;
```

```
UPDATE 1
```

```
postgres=# select xmin,* from tbase_main;
```

```
xmin | id | mc
-----+-----+-----
2564 | 1 | TBase
(1 row)
```

```
postgres=# update tbase_main set mc='TBase' where mc!='TBase';
```

```
UPDATE 0
```

```
postgres=# select xmin,* from tbase_main;
```

```
xmin | id | mc
-----+-----+-----
2564 | 1 | TBase
(1 row)
```

上面的效果是一样的,但带条件的更新不会产生一个新的版本记录,不需要系统执行 `vacuum` 回收垃圾数据。

4)、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;

#seseion1 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 2000000000
```

#seseion2 等待

```
postgres=# update tbase_main set mc='TBase_1.4' where id=1;
```

#seseion3 等待

```
postgres=# update tbase_main set mc='TBase_1.5' where id=2;
```

如果#seseion1 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

5)、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,'xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx' 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

6)、建议对报表类的或生成基础数据的查询，使用物化视图(MATERIALIZED VIEW)定期固化数据快照，避免对多表（尤其多写频繁的表）重复跑相同的查询，且物化视图支持 REFRESH MATERIALIZED VIEW CONCURRENTLY，支持并发更新；

如有一个程序需要不断查询 TBase_main 的总记录数，那么我们这样做

```

postgres=# select count(1) from tbase_main;
count
-----
200004
(1 row)

```

Time: 27.948 ms

```
postgres=# create MATERIALIZED VIEW TBase_main_count as select count(1) as num from tbase_main;
SELECT 1
```

Time: 322.372 ms

```
postgres=# select num from TBase_main_count ;
```

```
num
```

```
-----
```

```
200004
```

```
(1 row)
```

Time: 0.421 ms

性能提高上百倍

有数据变化时刷新方法

```
postgres=# copy TBase_main from '/data/pgxz/TBase_main.txt';
```

```
COPY 100002
```

Time: 1201.774 ms

```
postgres=# select count(1) from tbase_main;
```

```
count
```

```
-----
```

```
300006
```

```
(1 row)
```

Time: 23.164 ms

```
postgres=# REFRESH MATERIALIZED VIEW TBase_main_count;
```

```
REFRESH MATERIALIZED VIEW
```

Time: 49.486 ms

```
postgres=# select num from tbase_main_count ;
```

```
num
```

```
-----
```

```
300006
```

```
(1 row)
```

Time: 0.301 ms

7)、建议复杂的统计查询可以尝试窗口函数

请参考 5.3.2SQL-CET 用法

8)、两表 join 时尽量使用分布 key 进行 join

所似在建立业务的主表，明细表时，就需要使用他们的关联键来做分布键，如下所示

```
[pgxz@VM_0_29_centos pgzx]$ psql -p 15001
psql (PostgreSQL 10 (TBase 2.01))
Type "help" for help.
```

```
postgres=# create table tbase_main(id integer,mc text) distribute by shard(id);
CREATE TABLE
postgres=# create table tbase_detail(id integer,TBase_main_id integer,mc text) distribute by
shard(TBase_main_id);
CREATE TABLE
postgres=# explain select TBase_detail.* from tbase_main,TBase_detail where
TBase_main.id=TBase_detail.TBase_main_id;
QUERY PLAN
```

```
-----
Data Node Scan on "__REMOTE_FQS_QUERY__" (cost=0.00..0.00 rows=0 width=0)
Node/s: dn001, dn002
(2 rows)
```

```
postgres=# explain (verbose) select TBase_detail.* from tbase_main,TBase_detail where
TBase_main.id=TBase_detail.TBase_main_id;
QUERY
PLAN
```

```
-----
Data Node Scan on "__REMOTE_FQS_QUERY__" (cost=0.00..0.00 rows=0 width=0)
Output: TBase_detail.id, TBase_detail.TBase_main_id, TBase_detail.mc
Node/s: dn001, dn002
Remote query: SELECT TBase_detail.id, TBase_detail.TBase_main_id, TBase_detail.mc FROM
public.tbase_main, public.tbase_detail WHERE (TBase_main.id = TBase_detail.TBase_main_id)
(4 rows)
```

```
postgres=#
```

9)、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

10)、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。

11)、不要对字段建立外键

目前 TBase 还不支持多 dn 外键约束，除非你能确定数据关联键的数据全部落在同一个 dn 上面

5.3、高级 sql 语句编写

5.3.1、窗口函数的使用

5.3.1.1、环境准备

```
drop table if exists bills ;
```

```
create table bills
```

```
(
  id serial not null,
  goodsdesc text not null,
  beginunit text not null,
  begincity text not null,
  pubtime timestamp not null,
  amount float8 not null default 0,
  primary key (id)
);
```

```
COMMENT ON TABLE bills is '运单记录';
```

```
COMMENT ON COLUMN bills.id IS 'id 号';
```

```
COMMENT ON COLUMN bills.goodsdesc IS '货物名称';
```

```
COMMENT ON COLUMN bills.beginunit IS '启运省份';
```

```
COMMENT ON COLUMN bills.begincity IS '启运城市';
```

```
COMMENT ON COLUMN bills.pubtime IS '发布时间';
```

```
COMMENT ON COLUMN bills.amount IS '运费';
```

```
INSERT INTO bills(id,goodsdesc,beginunit,begincity,pubtime,amount)
```

```
VALUES(default,'衣服','海南省','三亚市','2015-10-05 09:32:01',ROUND((random()*10000)::NUMERIC,2));
```

```
INSERT INTO bills(id,goodsdesc,beginunit,begincity,pubtime,amount)
```

```
VALUES(default,' 建筑 设 备 ','福 建 省 ','三 明 市 ','2015-10-05 07:21:22',ROUND((random()*10000)::NUMERIC,2));
```

```
INSERT INTO bills(id,goodsdesc,beginunit,begincity,pubtime,amount)
```

```
VALUES(default,'设备','福建省','三明市','2015-10-05 11:21:54',ROUND((random()*10000)::NUMERIC,2));
```

```
INSERT INTO bills(id,goodsdesc,beginunit,begincity,pubtime,amount)
```

```
VALUES(default,'普货','福建省','三明市','2015-10-05 15:19:17',ROUND((random()*10000)::NUMERIC,2));
```

```
INSERT INTO bills(id,goodsdesc,beginunit,begincity,pubtime,amount)
VALUES(default,'5 0 铲车 , 后八轮翻斗车 ','河南省 ','三门峡市 ','2015-10-05
07:53:13',ROUND((random()*10000)::NUMERIC,2));
```

```
INSERT INTO bills(id,goodsdesc,beginunit,begincity,pubtime,amount)
VALUES(default,'鲜香菇 2000 斤 ','河南省 ','三门峡市 ','2015-10-05
10:38:29',ROUND((random()*10000)::NUMERIC,2));
```

```
INSERT INTO bills(id,goodsdesc,beginunit,begincity,pubtime,amount)
VALUES(default,'旋挖附件 38 吨 ','河南省 ','三门峡市 ','2015-10-05
10:48:38',ROUND((random()*10000)::NUMERIC,2));
```

```
INSERT INTO bills(id,goodsdesc,beginunit,begincity,pubtime,amount)
VALUES(default,'旋挖附件 35 吨 ','河南省 ','三门峡市 ','2015-10-05
10:48:38',ROUND((random()*10000)::NUMERIC,2));
```

```
INSERT INTO bills(id,goodsdesc,beginunit,begincity,pubtime,amount)
VALUES(default,'旋挖附件 39 吨 ','河南省 ','三门峡市 ','2015-10-05
11:38:38',ROUND((random()*10000)::NUMERIC,2));
```

```
INSERT INTO bills(id,goodsdesc,beginunit,begincity,pubtime,amount)
VALUES(default,'设备','上海市','上海市','2015-10-05 07:59:35',ROUND((random()*10000)::NUMERIC,2));
```

```
INSERT INTO bills(id,goodsdesc,beginunit,begincity,pubtime,amount)
VALUES(default,'普货 40 吨需 13 米半挂一辆 ','上海市 ','上海市 ','2015-10-05
08:13:59',ROUND((random()*10000)::NUMERIC,2));
```

5.3.1.2、row_number() --返回行号，不分组

```
postgres=# select row_number() over(),* from bills limit 2;
```

row_number	id	goodsdesc	beginunit	begincity	pubtime	amount
1	1	衣服	海南省	三亚市	2015-10-05 09:32:01	1915.86
2	2	建筑设备	福建省	三明市	2015-10-05 07:21:22	2022.31

(2 rows)

```
postgres=# select row_number() over(),* from bills limit 2 offset 2;
```

row_number	id	goodsdesc	beginunit	begincity	pubtime	amount
3	6	5 0 铲车, 后八轮翻斗车	河南省	三门峡市	2015-10-05 07:53:13	1030.9
4	8	旋挖附件 38 吨	河南省	三门峡市	2015-10-05 10:48:38	5365.04

(2 rows)

5.3.1.3、row_number() --返回行号，按 amount 排序

```
postgres=# select row_number() over(order by amount),* from bills;
```

row_number	id	goodsdesc	beginunit	begincity	pubtime	amount
1	11	设备	上海市	上海市	2015-10-05 07:59:35	971.54
2	6	50 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)

5.3.1.4、row_number() --返回行号，按 begincity 分组，pubtime 排序，注意绿色记录行号不间断

```
postgres=# select row_number() over(partition by begincity order by pubtime),* from bills;
```

row_number	id	goodsdesc	beginunit	begincity	pubtime	amount
1	1	衣服	海南省	三亚市	2015-10-05 09:32:01	1915.86
1	2	建筑设备	福建省	三明市	2015-10-05 07:21:22	2022.31
2	3	设备	福建省	三明市	2015-10-05 11:21:54	8771.11
3	4	普货	福建省	三明市	2015-10-05 15:19:17	1316.27
1	6	50 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)

5.3.1.5、rank()--返回行号,对比值重复时行号重复并间断，即返回 1,2,2,4...

```
postgres=# select rank() over(partition by begincity order by pubtime),* from bills;
```

rank	id	goodsdesc	beginunit	begincity	pubtime	amount
1	1	衣服	海南省	三亚市	2015-10-05 09:32:01	1915.86
1	2	建筑设备	福建省	三明市	2015-10-05 07:21:22	2022.31
2	3	设备	福建省	三明市	2015-10-05 11:21:54	8771.11
3	4	普货	福建省	三明市	2015-10-05 15:19:17	1316.27

```

1| 6| 50 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)

5.3.1.6、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...

```
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	50 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)

5.3.1.7、percent_rank()从当前开始, 计算在分组中的比例 (行号-1)/(总记录数-1)

```
postgres=# select percent_rank() over(partition by begincity order by id),* from bills;
```

percent_rank	id	goodsdesc	beginunit	begincity	pubtime	amount
0	1	衣服	海南省	三亚市	2015-10-05 09:32:01	1915.86
0	2	建筑设备	福建省	三明市	2015-10-05 07:21:22	2022.31
0.5	3	设备	福建省	三明市	2015-10-05 11:21:54	8771.11
1	4	普货	福建省	三明市	2015-10-05 15:19:17	1316.27
0	6	50 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)

5.3.1.8、cume_dist() --返回行数除以记录数值

```
postgres=# select ROUND((cume_dist() over(partition by begincity order by id))::NUMERIC,2) AS cume_dist,*
from bills;
```

cume_dist	id	goodsdesc	beginunit	begincity	pubtime	amount
1.00	1	衣服	海南省	三亚市	2015-10-05 09:32:01	1915.86
0.33	2	建筑设备	福建省	三明市	2015-10-05 07:21:22	2022.31
0.67	3	设备	福建省	三明市	2015-10-05 11:21:54	8771.11
1.00	4	普货	福建省	三明市	2015-10-05 15:19:17	1316.27
0.20	6	5 0 铲车, 后八轮翻斗车	河南省	三门峡市	2015-10-05 07:53:13	1030.9
0.40	7	鲜香菇 2000 斤	河南省	三门峡市	2015-10-05 10:38:29	4182.68
0.60	8	旋挖附件 38 吨	河南省	三门峡市	2015-10-05 10:48:38	5365.04
0.80	9	旋挖附件 35 吨	河南省	三门峡市	2015-10-05 10:48:38	9621.37
1.00	10	旋挖附件 39 吨	河南省	三门峡市	2015-10-05 11:38:38	8290.5
0.50	5	普货 40 吨需 13 米半挂一辆	上海市	上海市	2015-10-05 08:13:59	9886.15
1.00	11	设备	上海市	上海市	2015-10-05 07:59:35	971.54

(11 rows)

5.3.1.9、ntile(分组数量)--让所有记录尽可以的均匀分布

```
postgres=# select ntile(2) over(partition by begincity order by id),* from bills;
```

ntile	id	goodsdesc	beginunit	begincity	pubtime	amount
1	1	衣服	海南省	三亚市	2015-10-05 09:32:01	1915.86
1	2	建筑设备	福建省	三明市	2015-10-05 07:21:22	2022.31
1	3	设备	福建省	三明市	2015-10-05 11:21:54	8771.11
2	4	普货	福建省	三明市	2015-10-05 15:19:17	1316.27
1	6	5 0 铲车, 后八轮翻斗车	河南省	三门峡市	2015-10-05 07:53:13	1030.9
1	7	鲜香菇 2000 斤	河南省	三门峡市	2015-10-05 10:38:29	4182.68
1	8	旋挖附件 38 吨	河南省	三门峡市	2015-10-05 10:48:38	5365.04
2	9	旋挖附件 35 吨	河南省	三门峡市	2015-10-05 10:48:38	9621.37
2	10	旋挖附件 39 吨	河南省	三门峡市	2015-10-05 11:38:38	8290.5
1	5	普货 40 吨需 13 米半挂一辆	上海市	上海市	2015-10-05 08:13:59	9886.15
2	11	设备	上海市	上海市	2015-10-05 07:59:35	971.54

(11 rows)

```
postgres=# select ntile(3) over(partition by begincity order by id),* from bills;
```

ntile	id	goodsdesc	beginunit	begincity	pubtime	amount
1	1	衣服	海南省	三亚市	2015-10-05 09:32:01	1915.86
1	2	建筑设备	福建省	三明市	2015-10-05 07:21:22	2022.31
2	3	设备	福建省	三明市	2015-10-05 11:21:54	8771.11
3	4	普货	福建省	三明市	2015-10-05 15:19:17	1316.27
1	6	5 0 铲车, 后八轮翻斗车	河南省	三门峡市	2015-10-05 07:53:13	1030.9

1	7	鲜香菇 2000 斤	河南省	三门峡市	2015-10-05 10:38:29	4182.68
2	8	旋挖附件 38 吨	河南省	三门峡市	2015-10-05 10:48:38	5365.04
2	9	旋挖附件 35 吨	河南省	三门峡市	2015-10-05 10:48:38	9621.37
3	10	旋挖附件 39 吨	河南省	三门峡市	2015-10-05 11:38:38	8290.5
1	5	普货 40 吨需 13 米半挂一辆	上海市	上海市	2015-10-05 08:13:59	9886.15
2	11	设备	上海市	上海市	2015-10-05 07:59:35	971.54

(11 rows)

5.3.1.10、lag(value any [, offset integer [, default any]])--返回偏移量值

offset integer 是偏移值，正数时取前值，负数时取后值，没有取到值时用 default 代替

postgres=# select lag(amount,1,null) over(partition by begincity order by id),* from bills;

lag	id	goodsdesc	beginunit	begincity	pubtime	amount
	1	衣服	海南省	三亚市	2015-10-05 09:32:01	1915.86
	2	建筑设备	福建省	三明市	2015-10-05 07:21:22	2022.31
2022.31	3	设备	福建省	三明市	2015-10-05 11:21:54	8771.11
8771.11	4	普货	福建省	三明市	2015-10-05 15:19:17	1316.27
	6	50 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	50 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		50 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)						

5.3.1.11、lead(value any [,offset integer [, default any]])--返回偏移量值

offset integer 是偏移值, 正数时取后值, 负数时取前值, 没有取到值时用 default 代替

postgres=# select lead(amount,2,null) over(partition by begincity order by id),* from bills;

lead		id		goodsdesc		beginunit		begincity		pubtime		amount
-----+-----+-----+-----+-----+-----+-----												
		1		衣服		海南省		三亚市		2015-10-05 09:32:01		1915.86
1316.27		2		建筑设备		福建省		三明市		2015-10-05 07:21:22		2022.31
		3		设备		福建省		三明市		2015-10-05 11:21:54		8771.11
		4		普货		福建省		三明市		2015-10-05 15:19:17		1316.27
5365.04		6		50 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		50 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)

5.3.1.12、first_value(value any)返回第一值

```
postgres=# select first_value(amount) over(partition by begincity order by id),* from bills;
```

first_value	id	goodsdesc	beginunit	begincity	pubtime	amount
1915.86	1	衣服	海南省	三亚市	2015-10-05 09:32:01	1915.86
2022.31	2	建筑设备	福建省	三明市	2015-10-05 07:21:22	2022.31
2022.31	3	设备	福建省	三明市	2015-10-05 11:21:54	8771.11
2022.31	4	普货	福建省	三明市	2015-10-05 15:19:17	1316.27
1030.9	6	50 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)

5.3.1.13、last_value(value any)返回最后值

```
postgres=# select last_value(amount) over(partition by begincity order by pubtime),* FROM bills;
```

last_value	id	goodsdesc	beginunit	begincity	pubtime	amount
1915.86	1	衣服	海南省	三亚市	2015-10-05 09:32:01	1915.86
2022.31	2	建筑设备	福建省	三明市	2015-10-05 07:21:22	2022.31
8771.11	3	设备	福建省	三明市	2015-10-05 11:21:54	8771.11
1316.27	4	普货	福建省	三明市	2015-10-05 15:19:17	1316.27
1030.9	6	50 铲车, 后八轮翻斗车	河南省	三门峡市	2015-10-05 07:53:13	1030.9
4182.68	7	鲜香菇 2000 斤	河南省	三门峡市	2015-10-05 10:38:29	4182.68
9621.37	8	旋挖附件 38 吨	河南省	三门峡市	2015-10-05 10:48:38	5365.04
9621.37	9	旋挖附件 35 吨	河南省	三门峡市	2015-10-05 10:48:38	9621.37
8290.5	10	旋挖附件 39 吨	河南省	三门峡市	2015-10-05 11:38:38	8290.5
971.54	11	设备	上海市	上海市	2015-10-05 07:59:35	971.54
9886.15	5	普货 40 吨需 13 米半挂一辆	上海市	上海市	2015-10-05 08:13:59	9886.15

(11 rows)

```
postgres=# select last_value(amount) over(partition by begincity),* FROM bills;
```

last_value	id	goodsdesc	beginunit	begincity	pubtime	amount
1915.86	1	衣服	海南省	三亚市	2015-10-05 09:32:01	1915.86
1316.27	2	建筑设备	福建省	三明市	2015-10-05 07:21:22	2022.31

1316.27	3	设备	福建省	三明市	2015-10-05 11:21:54	8771.11
1316.27	4	普货	福建省	三明市	2015-10-05 15:19:17	1316.27
9621.37	7	鲜香菇 2000 斤	河南省	三门峡市	2015-10-05 10:38:29	4182.68
9621.37	10	旋挖附件 39 吨	河南省	三门峡市	2015-10-05 11:38:38	8290.5
9621.37	6	5 0 铲车, 后八轮翻斗车	河南省	三门峡市	2015-10-05 07:53:13	1030.9
9621.37	8	旋挖附件 38 吨	河南省	三门峡市	2015-10-05 10:48:38	5365.04
9621.37	9	旋挖附件 35 吨	河南省	三门峡市	2015-10-05 10:48:38	9621.37
971.54	5	普货 40 吨需 13 米半挂一辆	上海市	上海市	2015-10-05 08:13:59	9886.15
971.54	11	设备	上海市	上海市	2015-10-05 07:59:35	971.54

(11 rows)

注意不要加上 `order by id`, 默认情况下, 带了 `order by` 参数会从分组的起始值开始一直叠加, 直到当前值 (不是当前记录) 不同为止, 当忽略 `order by` 参数则是整个分组。下面通过修改分组的统计范围就可以实现 `order by` 参数取最后值

```
postgres=# select last_value(amount) over(partition by begincity order by id range between unbounded preceding
and unbounded following),* FROM bills;
```

last_value id	goodsdesc	beginunit begincity	pubtime	amount	
1915.86	1	衣服	海南省	三亚市	2015-10-05 09:32:01 1915.86
1316.27	2	建筑设备	福建省	三明市	2015-10-05 07:21:22 2022.31
1316.27	3	设备	福建省	三明市	2015-10-05 11:21:54 8771.11
1316.27	4	普货	福建省	三明市	2015-10-05 15:19:17 1316.27
8290.5	6	5 0 铲车, 后八轮翻斗车	河南省	三门峡市	2015-10-05 07:53:13 1030.9
8290.5	7	鲜香菇 2000 斤	河南省	三门峡市	2015-10-05 10:38:29 4182.68
8290.5	8	旋挖附件 38 吨	河南省	三门峡市	2015-10-05 10:48:38 5365.04
8290.5	9	旋挖附件 35 吨	河南省	三门峡市	2015-10-05 10:48:38 9621.37
8290.5	10	旋挖附件 39 吨	河南省	三门峡市	2015-10-05 11:38:38 8290.5
971.54	5	普货 40 吨需 13 米半挂一辆	上海市	上海市	2015-10-05 08:13:59 9886.15
971.54	11	设备	上海市	上海市	2015-10-05 07:59:35 971.54

(11 rows)

5.3.1.14、nth_value(value any, nth integer): 返回窗口框架中的指定值

```
postgres=# select nth_value(amount,2) over(partition by begincity order by id),* from bills;
```

nth_value id	goodsdesc	beginunit begincity	pubtime	amount	
	1	衣服	海南省	三亚市	2015-10-05 09:32:01 1915.86
	2	建筑设备	福建省	三明市	2015-10-05 07:21:22 2022.31
8771.11	3	设备	福建省	三明市	2015-10-05 11:21:54 8771.11
8771.11	4	普货	福建省	三明市	2015-10-05 15:19:17 1316.27
	6	5 0 铲车, 后八轮翻斗车	河南省	三门峡市	2015-10-05 07:53:13 1030.9
4182.68	7	鲜香菇 2000 斤	河南省	三门峡市	2015-10-05 10:38:29 4182.68
4182.68	8	旋挖附件 38 吨	河南省	三门峡市	2015-10-05 10:48:38 5365.04
4182.68	9	旋挖附件 35 吨	河南省	三门峡市	2015-10-05 10:48:38 9621.37

```
4182.68 | 10 | 旋挖附件 39 吨 | 河南省 | 三门峡市 | 2015-10-05 11:38:38 | 8290.5
| 5 | 普货 40 吨需 13 米半挂一辆 | 上海市 | 上海市 | 2015-10-05 08:13:59 | 9886.15
971.54 | 11 | 设备 | 上海市 | 上海市 | 2015-10-05 07:59:35 | 971.54
```

(11 rows)

5.3.1.15、统计各个城市的总运费及平均每单的运费

```
postgres=# select sum(amount) over(partition by begincity),avg(amount) over(partition by
begincity),begincity,amount from bills;
```

sum	avg	begincity	amount
1915.86	1915.86	三亚市	1915.86
12109.69	4036.563333333333	三明市	2022.31
12109.69	4036.563333333333	三明市	8771.11
12109.69	4036.563333333333	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)

5.3.1.16、窗口函数别名使用

```
postgres=# select sum(amount) over w,avg(amount) over w,begincity,amount from bills window w as (partition by
begincity);
```

sum	avg	begincity	amount
1915.86	1915.86	三亚市	1915.86
12109.69	4036.563333333333	三明市	2022.31
12109.69	4036.563333333333	三明市	8771.11
12109.69	4036.563333333333	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)

5.3.1.17、获取每个城市运费前两名订单

```
postgres=# select * from (select row_number() over(partition by begincity order by amount desc),* from bills)
where row_number<3;
```

row_number	id	goodsdesc	beginunit	begincity	pubtime	amount
1	1	衣服	海南省	三亚市	2015-10-05 09:32:01	1915.86
1	3	设备	福建省	三明市	2015-10-05 11:21:54	8771.11
2	2	建筑设备	福建省	三明市	2015-10-05 07:21:22	2022.31
1	9	旋挖附件 35 吨	河南省	三门峡市	2015-10-05 10:48:38	9621.37
2	10	旋挖附件 39 吨	河南省	三门峡市	2015-10-05 11:38:38	8290.5
1	5	普货 40 吨需 13 米半挂一辆	上海市	上海市	2015-10-05 08:13:59	9886.15
2	11	设备	上海市	上海市	2015-10-05 07:59:35	971.54

(7 rows)

6、TBase-PL/pgsql 开发

6.1、应用程序语法介绍

6.1.1、建立函数语法

```
CREATE [OR REPLACE] FUNCTION [模式名.]函数名 ([参数模式 [参数名] 数据类型 [default 默认值]
[,...]]) RETURNS [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;
```

6.1.2、[OR REPLACE] 更新函数介绍

带 OR REPLACE 的作用就函数存在时则替换的功能,建立 PL/pgsql 函数时不带 OR REPLACE 关键字,则遇到函数已经存系统则会报错,如下所示

```
postgres=# select prosrc from pg_proc where proname='f';
prosrc
-----
```

```

+
BEGIN
+
    RAISE NOTICE 'TBase';+
END;
+

```

(1 行记录)

```

postgres=# CREATE FUNCTION f() RETURNS VOID AS
postgres-# $$
postgres$# BEGIN
postgres$#     RAISE NOTICE 'Hello ,TBase';
postgres$# END;
postgres$# $$
postgres-# LANGUAGE PLPGSQL;
ERROR:  function "f" already exists with same argument types
postgres=# CREATE OR REPLACE FUNCTION f() RETURNS VOID AS
postgres-# $$
postgres$# BEGIN
postgres$#     RAISE NOTICE 'Hello ,TBase';
postgres$# END;
postgres$# $$
postgres-# LANGUAGE PLPGSQL;
CREATE FUNCTION
postgres=# select prosrc from pg_proc where proname='f';
           prosrc
-----
BEGIN
+
    RAISE NOTICE 'Hello ,TBase';+
END;
+

```

(1 行记录)

```

postgres=# SELECT f();
NOTICE:  Hello ,TBase
 f
---

```

(1 行记录)

6.1.3、[模式名.]函数名介绍

建立函数名称，模式名可以指定，也可以不指定，不指定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;
```

nsname	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	<pre> + BEGIN + RAISE NOTICE 'Hello ,TBase'; + END; + </pre>

(1 行记录)

因为\$用er 模式不存在，所以存在 public 模式下

6.2、参数详细介绍

6.2.1、参数模式

参数模式总共有三种，分别是 IN (传入参数，这个是默认方式)、OUT(返回值参数)、INOUT(传入返回值参数)，下面说说这几个参数模式的用法

6.2.1.1、IN 模式

IN 模式指的是执行函数时需要输入参数值，如下所示

```
postgres=# CREATE OR REPLACE FUNCTION fl(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 fl('TBase');
   fl
-----
 TBase
(1 行记录)

```

```

postgres=# CREATE OR REPLACE FUNCTION fl(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 fl('TBase');
   fl
-----
 TBase
(1 行记录)

```

上面两种方式定义参数效果是一样的

6.2.1.2、OUT 模式

OUT 模式参数是指定了函数执行时返回的字段名及类型

```

postgres=# CREATE OR REPLACE FUNCTION fl(OUT a_xm TEXT) RETURNS TEXT AS
postgres-# $$
postgres$# BEGIN
postgres$#      a_xm:='TBase';
postgres$# END;
postgres$# $$
postgres-# LANGUAGE PLPGSQL;
CREATE FUNCTION
postgres=# SELECT * FROM fl();
   a_xm
-----
 TBase
(1 行记录)

```

采用 OUT 模式参数不能用 RETURN 返回，而是要对返回的 OUT 参数直接付值。返回值类型与参数的数据类型必需一致。参数名就是返回的字段名

6.2.1.3、INOUT 模式

INOUT 模式是指参数即传入，同时又指定了返回值的字段名和类型

```
postgres=# CREATE OR REPLACE FUNCTION fl(INOUT a_xm text) RETURNS TEXT AS
postgres-# $$
postgres$# BEGIN
postgres$# END;
postgres$# $$
postgres-# LANGUAGE PLPGSQL;
CREATE FUNCTION
postgres=#
postgres=# SELECT * FROM fl('TBase');
 a_xm
-----
 TBase
(1 行记录)
```

值得注意的是，上面的函数跟下面的函数是相同的，即重新定义会覆盖掉

```
postgres=# CREATE OR REPLACE FUNCTION fl(IN a_xm text) RETURNS TEXT AS
postgres-# $$
postgres$# BEGIN
postgres$#     RETURN 'TBase';
postgres$# END;
postgres$# $$
postgres-# LANGUAGE PLPGSQL;
CREATE FUNCTION
postgres=# SELECT * FROM fl('TBase');
 fl
-----
 TBase
(1 行记录)
```

6.2.1.4、VARIADIC 模式

VARIADICS 模式是参数个数可变模式，系统用一个数组对传入的参数进行处理，VARIADIC 参数必需是所有最后一个声明，如下所示

```
postgres=# CREATE OR REPLACE FUNCTION fl(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 fl(1);
NOTICE:  a_int = {1}
NOTICE:  a_int[1] = 1
fl
----

```

(1 行记录)

```

postgres=# SELECT fl(1,2);
NOTICE:  a_int = {1,2}
NOTICE:  a_int[1] = 1
fl
----

```

(1 行记录)

```

postgres=# CREATE OR REPLACE FUNCTION fl(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 fl('TBase',1,2);
NOTICE:  a_int = {1,2}
NOTICE:  a_int[1] = 1
NOTICE:  a_xm = TBasea
fl
----

```

(1 行记录)

6.2.2、参数引用

PL/pgsql 函数的参数是以\$1,\$2 这样标识符来进行传递，也支持命名参数，所以参数的定义可以用下面的方式

6.2.2.1、无命名参数

```
postgres=# CREATE OR REPLACE FUNCTION f2(text) RETURNS TEXT AS
postgres-# $$
postgres$# BEGIN
postgres$#     RETURN $1;
postgres$# END;
postgres$# $$
postgres-# LANGUAGE PLPGSQL;
CREATE FUNCTION
postgres=#
postgres=# SELECT * FROM f2('TBase');
 f2
-----
TBase
(1 行记录)
```

6.2.2.2、给标识符指定别名

```
postgres=# CREATE OR REPLACE FUNCTION f2(text) RETURNS TEXT AS
postgres-# $$
postgres$# DECLARE
postgres$#     a_xm ALIAS FOR $1; --a_xm 是$1 的别名
postgres$# BEGIN
postgres$#     RETURN a_xm;
postgres$# END;
postgres$# $$
postgres-# LANGUAGE PLPGSQL;
CREATE FUNCTION
postgres=#
postgres=# SELECT * FROM f2('TBase');
 f2
-----
TBase
(1 行记录)
```

6.2.2.3、命名参数

```
postgres=# CREATE OR REPLACE FUNCTION f2(a_xm text) RETURNS TEXT AS
```

```

postgres=# $$
postgres## DECLARE
postgres##      v_xm ALIAS FOR $1;
postgres## BEGIN
postgres##      RAISE NOTICE 'a_xm = % ; v_xm = % ; $1 = %',a_xm,v_xm,$1;
postgres##      RETURN $1;
postgres## END;
postgres## $$
postgres=# LANGUAGE PLPGSQL;
CREATE FUNCTION
postgres=#
postgres=# SELECT * FROM f2('TBase');
NOTICE:  a_xm = TBase ; v_xm = TBase ; $1 = TBase
      f2
-----
      TBase
(1 行记录)

```

6.2.3、参数数据类型

数据类型(可以有模式修饰), 可以是基本类型, 复合类型、域类型、游标、或者可以引用一个现有表类型、字段类型(建立时转换为对应的类型)、还可以是多态类型 `anyelement`、`anyarray`, 也可以是各种数据类型的数组形式

6.2.3.1、基本类型

```

postgres=# CREATE OR REPLACE FUNCTION f3 (a_int integer,a_str text) RETURNS VOID AS
postgres=# $$
postgres## BEGIN
postgres##      RAISE NOTICE 'a_int = % ; a_str = %',a_int,a_str;
postgres## END;
postgres## $$
postgres=# LANGUAGE PLPGSQL;
CREATE FUNCTION
postgres=# SELECT * FROM f3(1,'TBase');
NOTICE:  a_int = 1 ; a_str = TBase
      f3
----

```

(1 行记录)

```

postgres=# CREATE OR REPLACE FUNCTION f3 (a_int integer[],a_str text[]) RETURNS VOID AS
postgres=# $$
postgres## BEGIN
postgres##      RAISE NOTICE 'a_int = % ; a_str = %',a_int,a_str;

```



```

postgres$# END;
postgres$# $$
postgres-# LANGUAGE PLPGSQL;
CREATE FUNCTION
postgres=# SELECT f3(ARRAY[1,2,3],ARRAY['TBase','pgxz']);
NOTICE:  a_int = {1,2,3} ; a_str = {TBase,pgxz}
f3
----

```

(1 行记录)

6.2.3.2、复合类型

```

postgres=# CREATE TYPE t_per AS
postgres-# (
postgres(#      id integer,
postgres(#      mc text
postgres(# );
ERROR:  type "t_per" already exists
postgres=# CREATE OR REPLACE FUNCTION f3 (a_row public.t_per) RETURNS VOID AS
postgres-# $$
postgres$# BEGIN
postgres$#      RAISE NOTICE 'id = % ; mc = %',a_row.id,a_row.mc;
postgres$# END;
postgres$# $$
postgres-# LANGUAGE PLPGSQL;
CREATE FUNCTION
postgres=#
postgres=# SELECT f3(ROW(1,'TBase')::public.t_per);
NOTICE:  id = 1 ; mc = TBase
f3
----

```

(1 行记录)

```

postgres=# CREATE OR REPLACE FUNCTION f3 (a_rec public.t_per[]) RETURNS VOID AS
postgres-# $$
postgres$# BEGIN
postgres$#      RAISE NOTICE 'a_rec = %',a_rec;
postgres$#      RAISE NOTICE 'a_rec[1].id = %',a_rec[1].id;
postgres$# END;
postgres$# $$
postgres-# LANGUAGE PLPGSQL;
CREATE FUNCTION
postgres=#
postgres=# SELECT f3(ARRAY[ROW(1,'TBase'),ROW(1,'pgxz')]::public.t_per[]);

```

```
NOTICE:  a_rec = {(1,TBase)","(1,pgxz)}
NOTICE:  a_rec[1].id = 1
f3
----
```

(1 行记录)

6.2.3.3、行类型

```
postgres=# \d t
      资料表 "public.t"
  栏位 | 型别      | 修饰词
-----+-----+-----
id    | integer   |
mc    | text      |
```

```
postgres=# CREATE OR REPLACE FUNCTION f3 (a_row public.t) RETURNS VOID AS
postgres-# $$
postgres$# BEGIN
postgres$#      RAISE NOTICE 'id = % ; mc = %',a_row.id,a_row.mc;
postgres$# END;
postgres$# $$
postgres-# LANGUAGE PLPGSQL;
CREATE FUNCTION
postgres=#
postgres=# SELECT f3(ROW(1,'TBase'));
NOTICE:  id = 1 ; mc = TBase
f3
----
```

(1 行记录)

```
postgres=# SELECT f3(t.*) FROM t LIMIT 1;
NOTICE:  id = 1 ; mc = TBase
f3
----
```

(1 行记录)

```
postgres=# CREATE OR REPLACE FUNCTION f3 (a_rec public.t[]) RETURNS VOID AS
postgres-# $$
postgres$# BEGIN
postgres$#      RAISE NOTICE 'a_rec = %',a_rec;
postgres$#      RAISE NOTICE 'a_rec[1].id = %',a_rec[1].id;
postgres$# END;
postgres$# $$
```

```

postgres=# LANGUAGE PLPGSQL;
CREATE FUNCTION
postgres=#
postgres=# SELECT f3(array[row(1,'TBase'),row(1,'pgxz')::public.t[]]);
NOTICE:  a_rec = {(1,TBase),"(1,pgxz)"}
NOTICE:  a_rec[1].id = 1
      f3
----

```

(1 行记录)

```

postgres=# SELECT f3(array[t.*,t.*::public.t[]] FROM t LIMIT 2;
NOTICE:  a_rec = {(1,TBase),"(1,TBase)"}
NOTICE:  a_rec[1].id = 1
NOTICE:  a_rec = {(2,pgxz),"(2,pgxz)"}
NOTICE:  a_rec[1].id = 2
      f3
----

```

(2 行记录)

6.2.3.4、域类型

```

postgres=# CREATE DOMAIN xb AS TEXT CHECK
postgres=# (
postgres(#      VALUE = '男'
postgres(#      OR VALUE = '女'
postgres(#      OR VALUE = "
postgres(# );
CREATE DOMAIN
postgres=#
postgres=# CREATE OR REPLACE FUNCTION f4 (a_xb public.xb) RETURNS VOID AS
postgres=# $$
postgres$# BEGIN
postgres$#      RAISE NOTICE 'a_xb = %',a_xb;
postgres$# END;
postgres$# $$
postgres=# LANGUAGE PLPGSQL;
CREATE FUNCTION
postgres=#
postgres=# SELECT * FROM f4('男');
NOTICE:  a_xb = 男
      f4
----

```

(1 行记录)

```
postgres=# SELECT * FROM f4('她');
ERROR:  value for domain xb violates check constraint "xb_check"
postgres=#
```

域类型输入参数值时会检查是否违反规则

6.2.3.5、游标类型

```
postgres=# CREATE OR REPLACE FUNCTION f5 (a_ref refcursor) RETURNS void AS
postgres-# $$
postgres$# DECLARE
postgres$#      v_rec record;
postgres$# BEGIN
postgres$#      OPEN a_ref FOR SELECT * FROM t LIMIT 1;
postgres$#      FETCH a_ref INTO v_rec;
postgres$#      RAISE NOTICE 'v_rec = % ',v_rec;
postgres$# END;
postgres$# $$
postgres-# LANGUAGE PLPGSQL;
CREATE FUNCTION
postgres=#
postgres=# SELECT * FROM f5('a');
NOTICE:  v_rec = (1,TBase)
 f5
----
```

(1 行记录)

```
postgres=# CREATE OR REPLACE FUNCTION f6 (a_ref refcursor) RETURNS refcursor AS
postgres-# $$
postgres$# BEGIN
postgres$#      OPEN a_ref FOR SELECT * FROM t LIMIT 1;
postgres$#      RETURN a_ref;
postgres$# END;
postgres$# $$
postgres-# LANGUAGE PLPGSQL;
CREATE FUNCTION
postgres=#
#注意这里需要开启一个事务
postgres=# BEGIN;
BEGIN
postgres=# SELECT * FROM f6('a');
 f6
----
 a
```

(1 行记录)

```
postgres=# FETCH ALL FROM a;
```

```
id | mc
```

```
----+-----
```

```
1 | TBase
```

(1 行记录)

6.2.3.6、多态类型

```
postgres=# CREATE OR REPLACE FUNCTION f_any(a_arg anyelement) RETURNS VOID AS
```

```
postgres=# $$
```

```
postgres$# BEGIN
```

```
postgres$#      RAISE NOTICE '%',a_arg;
```

```
postgres$# END;
```

```
postgres$# $$
```

```
postgres=# LANGUAGE PLPGSQL;
```

```
CREATE FUNCTION
```

```
postgres=#
```

```
postgres=# SELECT f_any(1::integer);
```

```
NOTICE:  1
```

```
  f_any
```

```
-----
```

(1 行记录)

```
postgres=# SELECT f_any('TBase'::TEXT);
```

```
NOTICE:  TBase
```

```
  f_any
```

```
-----
```

(1 行记录)

```
postgres=# SELECT f_any(ROW(1,'TBase')::public.t_rec);
```

```
NOTICE:  (1,TBase)
```

```
  f_any
```

```
-----
```

(1 行记录)

```
postgres=# SELECT f_any(ARRAY[1,2]::INTEGER[]);
```

```
NOTICE:  {1,2}
```

```
  f_any
```

```
-----
```

(1 行记录)

```
postgres=# SELECT f_any(ARRAY[[1,2],[3,4],[5,6]]::INTEGER[][]);
NOTICE:  {{1,2},{3,4},{5,6}}
 f_any
-----
```

(1 行记录)

注意多态类型参数函数调用时最好直接声明参数类型，否则有可能出错

```
postgres=# CREATE OR REPLACE FUNCTION f_any_array(a_arg anyarray) RETURNS VOID AS
postgres-# $$
postgres$# BEGIN
postgres$#     RAISE NOTICE '%',a_arg;
postgres$# END;
postgres$# $$
postgres-# LANGUAGE PLPGSQL;
CREATE FUNCTION
postgres=#
postgres=# SELECT f_any_array(ARRAY['TBase','pgxz']::TEXT[]);
NOTICE:  {TBase,pgxz}
 f_any_array
-----
```

(1 行记录)

```
postgres=# SELECT f_any_array(ARRAY[ARRAY['TBase','pgxz'],ARRAY['TBase','Tencent']]::TEXT[][]);
NOTICE:  {{TBase,pgxz},{TBase,Tencent}}
 f_any_array
-----
```

(1 行记录)

注意：Anyelement 参数如果写成数组，其意义就跟 anyarray 参数一致，所以 f_any(a_arg anyelement)与 f_any(a_arg anyarray)在调用 f_any(ARRAY[1,2])时就会出现函数不是唯一化的错误(ERROR: function f_any(...) is not unique)提示

6.2.3.7、参数默认值

PL/pgsql 扩展语言函数支持给参数设置默认值

```
postgres=# CREATE OR REPLACE FUNCTION f7 (a_int INTEGER DEFAULT 1) RETURNS VOID AS
postgres-# $$
postgres$# BEGIN
postgres$#     RAISE NOTICE 'a_int = %',a_int;
postgres$# END;
postgres$# $$
```

```

postgres=# LANGUAGE PLPGSQL;
CREATE FUNCTION
postgres=#
postgres=# SELECT * FROM f7();
NOTICE:  a_int = 1
 f7
----

```

(1 行记录)

备注：如果原来存在一个 f7() 这样的函数，则上面的执行就会出错，因为系统无法清楚到你到底要执行那个函数，如下所示

```

postgres=# CREATE OR REPLACE FUNCTION f7() RETURNS void AS
postgres=# $$
postgres$# BEGIN
postgres$#     RAISE NOTICE '无参数';
postgres$# END;
postgres$# $$
postgres=# LANGUAGE plpgsql ;
CREATE FUNCTION
postgres=#
postgres=# SELECT * FROM f7();
ERROR:  function f7() is not unique
第 1 行 SELECT * FROM f7();
      ^
提示:  Could not choose a best candidate function. You might need to add explicit type casts.
postgres=#

```

出错提示，f7() 函数不是唯一的，这是使用上一个需要特别注意的地方

6.3、返回值详细介绍

6.3.1、返回值介绍

返回值可以是一个简单数据类型、复合类型、RECORD、已经存在的表行类型、表字段类型、游标、另外还可以返回一个记录集、如果不需要返回值，则可以用 RETURN void。返回值的字段名及类型可以在参数在用 OUT, INOUT 模式中声明

6.3.2、返回值类型介绍

6.3.2.1、没有返回值

```
postgres=# CREATE OR REPLACE FUNCTION f8() RETURNS void AS
postgres-# $$
postgres$# BEGIN
postgres$#     RAISE NOTICE '不用返回值，函数体可以有或没有 return 语句';
postgres$#     RETURN ;--这一句可以有，也可以没有
postgres$# END;
postgres$# $$
postgres=# LANGUAGE plpgsql;
CREATE FUNCTION
postgres=#
postgres=# SELECT * FROM f8();
NOTICE: 不用返回值，函数体可以有或没有 return 语句
 f8
----
```

(1 行记录)

6.3.2.2、返回简单类型

```
postgres=# CREATE OR REPLACE FUNCTION f9() RETURNS TEXT AS
postgres-# $$
postgres$# BEGIN
postgres$#     RETURN 'TBase';
postgres$# END;
postgres$# $$
postgres=# LANGUAGE plpgsql;
CREATE FUNCTION
postgres=#
postgres=# SELECT * FROM f9() t(a_xm);
 a_xm
-----
```

TBase
(1 行记录)

```
postgres=#
postgres=# CREATE OR REPLACE FUNCTION f9(OUT a_xm TEXT) RETURNS TEXT AS
postgres-# $$
postgres$# BEGIN
postgres$#     a_xm:='TBase';
postgres$# END;
```



```

postgres=# $$
postgres=# LANGUAGE plpgsql;
CREATE FUNCTION
postgres=#
postgres=# SELECT * FROM f9();
 a_xm
-----
 TBase
(1 行记录)

```

上面两个函数其实就是同一个函数，建立时如果不加 **OR REPLACE** 则会提示已经存在

```

postgres=# CREATE OR REPLACE FUNCTION f10() RETURNS TEXT[] AS
postgres=# $$
postgres$# BEGIN
postgres$#     RETURN ARRAY['TBase','pgxz'];
postgres$# END;
postgres$# $$
postgres=# LANGUAGE plpgsql;
CREATE FUNCTION
postgres=#
postgres=# SELECT * FROM f10();
      f10
-----
 {TBase,pgxz}
(1 行记录)

```

6.3.2.3、返回一个复合类型

```

postgres=# CREATE TYPE t_rec AS
postgres=# (
postgres(#     id integer,
postgres(#     mc text
postgres(# );
CREATE TYPE
postgres=#
postgres=# CREATE OR REPLACE FUNCTION f11() RETURNS t_rec AS
postgres=# $$
postgres$# DECLARE
postgres$#     v_rec public.t_rec;
postgres$# BEGIN
postgres$#     v_rec.id:=1;
postgres$#     v_rec.mc='TBase';
postgres$#     RETURN v_rec;
postgres$# END;
postgres$# $$

```

```
postgres=# LANGUAGE plpgsql;
CREATE FUNCTION
postgres=#
postgres=# SELECT * FROM f11();
 id | mc
----+-----
  1 | TBase
(1 行记录)
```

```
postgres=# CREATE OR REPLACE FUNCTION f12() RETURNS t_rec[] AS
postgres=# $$
postgres$# BEGIN
postgres$#     RETURN ARRAY[ROW(1,'TBase'),ROW(1,'pgxz')::t_rec[]];
postgres$# END;
postgres$# $$
postgres=# LANGUAGE plpgsql;
CREATE FUNCTION
postgres=#
postgres=# SELECT * FROM f12();
          f12
-----
{"(1,TBase)","(1,pgxz)"}
(1 行记录)
```

6.3.2.4、返回行类型

```
postgres=# \d t
      资料表 "public.t"
  栏位 |  型别   | 修饰词
-----+-----+-----
 id    | integer |
 mc    | text    |

postgres=# CREATE OR REPLACE FUNCTION f13() RETURNS public.t AS
postgres=# $$
postgres$# DECLARE
postgres$#     v_rec public.t%ROWTYPE;
postgres$# BEGIN
postgres$#     SELECT * INTO v_rec FROM public.t LIMIT 1;
postgres$#     RETURN v_rec;
postgres$# END;
postgres$# $$
postgres=# LANGUAGE plpgsql;
CREATE FUNCTION
postgres=#
postgres=# SELECT * FROM f13();
```

```
id | mc
----+-----
 1 | TBase
(1 行记录)
```

```
postgres=# CREATE OR REPLACE FUNCTION f14() RETURNS public.t[] AS
postgres-# $$
postgres$# DECLARE
postgres$#      v_rec public.t[];
postgres$# BEGIN
postgres$#      SELECT ARRAY[ROW(t.*),ROW(t.*)]::public.t[] INTO v_rec FROM public.t LIMIT 1;
postgres$#      RETURN v_rec;
postgres$# END;
postgres$# $$
postgres-# LANGUAGE plpgsql;
CREATE FUNCTION
postgres=#
postgres=# SELECT * FROM f14();
          f14
-----
{"(1,TBase)","(1,TBase)"}
(1 行记录)
```

6.3.2.5、返回 TABLE 类型

```
postgres=# DROP FUNCTION f14();
DROP FUNCTION
postgres=# CREATE FUNCTION f14() RETURNS TABLE(a_id integer, a_nc text) AS
postgres-# $$
postgres$# BEGIN
postgres$#      RETURN QUERY SELECT 1::integer,'TBase'::Text;
postgres$# END;
postgres$# $$LANGUAGE plpgsql;
CREATE FUNCTION
postgres=# SELECT * FROM f14();
 a_id | a_nc
-----+-----
    1 | TBase
(1 row)
```

6.3.2.6、返回 RECORD 类型

```
postgres=# CREATE OR REPLACE FUNCTION f15() RETURNS RECORD AS
postgres-# $$
postgres$# DECLARE
```

```

postgres$#      v_rec RECORD;
postgres$# BEGIN
postgres$#      v_rec:=ROW(1::integer,'TBase'::text,'pgxz'::text);
postgres$#      RETURN v_rec;
postgres$# END;
postgres$# $$
postgres-# LANGUAGE plpgsql;
CREATE FUNCTION
postgres=# SELECT f15();
          f15
-----
(1,TBase,pgxz)
(1 行记录)

```

```

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

```

6.3.2.7、返回一个游标

```

postgres=# CREATE OR REPLACE FUNCTION f16() RETURNS refcursor AS
postgres-# $$
postgres$# DECLARE
postgres$#      v_ref refcursor;
postgres$# BEGIN
postgres$#      OPEN v_ref FOR SELECT * FROM public.t;
postgres$#      RETURN v_ref;
postgres$# END;
postgres$# $$
postgres-# LANGUAGE plpgsql;
CREATE FUNCTION
postgres=#
postgres=# BEGIN;
BEGIN
postgres=#
postgres=# SELECT * FROM f16();
          f15
-----
<unnamed portal 1>
(1 行记录)

```

```

postgres=# FETCH ALL FROM "<unnamed portal 1>";
 id |  mc
----+-----

```

```

1 | TBase
2 | pgxz
(2 行记录)
postgres=# END;

postgres=# CREATE OR REPLACE FUNCTION fl6(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 fl6('a');
 f15
-----
 a
(1 行记录)

postgres=# FETCH ALL FROM a;
 id |  mc
----+-----
  1 | TBase
  2 | pgxz
(2 行记录)

postgres=# END;
COMMIT

```

6.3.2.8、返回记录集

```

postgres=# CREATE OR REPLACE FUNCTION fl7() RETURNS SETOF TEXT AS
postgres-# $$
postgres$# BEGIN
postgres$#     RETURN NEXT 'TBase'::text;
postgres$#     RETURN NEXT 'pgxz'::text;
postgres$#     RETURN ;--最后的 RETURN 可以加，也可以不加上去
postgres$# END;
postgres$# $$
postgres-# LANGUAGE PLPGSQL;
CREATE FUNCTION
postgres=#

```

```
postgres=# SELECT * FROM f17();
 f17
```

```
-----
```

```
TBase
```

```
pgxz
```

```
(2 行记录)
```

```
postgres=#
```

```
postgres=#
```

```
postgres=# CREATE OR REPLACE FUNCTION f18() RETURNS SETOF public.t AS
```

```
postgres-# $$
```

```
postgres$# DECLARE
```

```
postgres$#      --使用行类型返回
```

```
postgres$#      v_rec public.t%ROWTYPE;
```

```
postgres$# BEGIN
```

```
postgres$#      FOR v_rec IN SELECT * FROM t ORDER BY id LOOP
```

```
postgres$#          RETURN NEXT v_rec;
```

```
postgres$#      END LOOP;
```

```
postgres$#      RETURN ;--最后的 RETURN 可以加，也可以不加上去
```

```
postgres$# END;
```

```
postgres$# $$
```

```
postgres-# LANGUAGE PLPGSQL;
```

```
CREATE FUNCTION
```

```
postgres=#
```

```
postgres=# SELECT * FROM f18();
```

```
id |  mc
```

```
---+-----
```

```
1 | TBase
```

```
2 | pgxz
```

```
(2 行记录)
```

```
postgres=# \d t1
```

```
资料表 "public.t1"
```

```
栏位 |      型别      | 修饰词
```

```
-----+-----+-----
```

```
id   | integer      | 非空
```

```
yhm  | text         | 非空
```

```
nc   | text         | 非空
```

```
mm   | character(32) | 非空
```

```
索引:
```

```
"t1_pkey" PRIMARY KEY, btree (id)
```

```
"t1_yhm_key" UNIQUE CONSTRAINT, btree (yhm)
```

```
postgres=# CREATE OR REPLACE FUNCTION f19() RETURNS SETOF public.t_rec AS
```

```
postgres-# $$
```

```
postgres$# DECLARE
```

```

postgres$#      --使用已经定义的结构类型返回
postgres$#      v_rec public.t_rec;
postgres$# BEGIN
postgres$#      FOR v_rec IN SELECT id,yhm FROM t1 ORDER BY id LOOP
postgres$#          RETURN NEXT v_rec;
postgres$#      END LOOP;
postgres$#      RETURN ;--最后的 RETURN 可以加，也可以不加上去
postgres$# END;
postgres$# $$
postgres-# LANGUAGE PLPGSQL;
CREATE FUNCTION

```

```
postgres=#
```

```
postgres=# SELECT * FROM f19();
```

```
id | mc
```

```
----+-----
```

```
1 | TBase
```

```
2 | pgxc
```

```
3 | pgxz
```

```
(3 行记录)
```

```
postgres=# CREATE OR REPLACE FUNCTION f20(a_int integer) RETURNS SETOF record AS
```

```
postgres-# $$
```

```
postgres$# DECLARE
```

```
postgres$#      --a_int 定义返回的字段数，实现动态列返回
```

```
postgres$#      v_rec record;
```

```
postgres$#      v_sql text;
```

```
postgres$# BEGIN
```

```
postgres$#      IF a_int = 2 THEN
```

```
postgres$#          v_sql:='SELECT id,yhm FROM t1 ORDER BY id';
```

```
postgres$#      ELSE
```

```
postgres$#          v_sql:='SELECT id,yhm,nc FROM t1 ORDER BY id';
```

```
postgres$#      END IF;
```

```
postgres$#      FOR v_rec IN EXECUTE v_sql LOOP
```

```
postgres$#          RETURN NEXT v_rec;
```

```
postgres$#      END LOOP;
```

```
postgres$#      RETURN ;--最后的 RETURN 可以加，也可以不加上去
```

```
postgres$# END;
```

```
postgres$# $$
```

```
postgres-# LANGUAGE PLPGSQL;
```

```
CREATE FUNCTION
```

```
postgres=# SELECT * FROM f20(2) t(id integer,yhm text);
```

```
id | yhm
```

```
----+-----
```

```
1 | TBase
```

```
2 | pgxc
```

```
3 | pgxz
```

(3 行记录)

```
postgres=# SELECT * FROM f20(3) t(id integer,yhm text,nc text);
```

```
 id | yhm | nc
----+-----+-----
```

```
  1 | TBase | TBase
```

```
  2 | pgxc | pgxc
```

```
  3 | pgxz | pgxz
```

(3 行记录)

```
postgres=# CREATE OR REPLACE FUNCTION f21(OUT a_id integer,OUT a_yhm TEXT) RETURNS SETOF
record AS
```

```
postgres-# $$
```

```
postgres$# DECLARE
```

```
postgres$#      --使用 out 返回
```

```
postgres$#      v_rec record;
```

```
postgres$# BEGIN
```

```
postgres$#      FOR v_rec IN SELECT id,yhm FROM t1 LOOP
```

```
postgres$#          a_id:=v_rec.id;
```

```
postgres$#          a_yhm:=v_rec.yhm;
```

```
postgres$#          RETURN NEXT;
```

```
postgres$#      END LOOP;
```

```
postgres$#      RETURN ;--最后的 RETURN 可以加，也可以不加上去
```

```
postgres$# END;
```

```
postgres$# $$
```

```
postgres-# LANGUAGE PLPGSQL;
```

```
CREATE FUNCTION
```

```
postgres=#
```

```
postgres=# SELECT * FROM f21();
```

```
 a_id | a_yhm
```

```
-----+-----
```

```
  1 | TBase
```

```
  2 | pgxc
```

```
  3 | pgxz
```

(3 行记录)

```
postgres=# CREATE OR REPLACE FUNCTION f22() RETURNS SETOF refcursor AS
```

```
postgres-# $$
```

```
postgres$# DECLARE
```

```
postgres$#      --返回游标集
```

```
postgres$#      v_ref1 REFCURSOR;
```

```
postgres$#      v_ref2 REFCURSOR;
```

```
postgres$# BEGIN
```

```
postgres$#      OPEN v_ref1 FOR SELECT * FROM t;
```

```
postgres$#      OPEN v_ref2 FOR SELECT * FROM t1;
```

```
postgres$#      RETURN NEXT v_ref1;
```



```

postgres$$      RETURN NEXT v_ref2;
postgres$$      RETURN ;--最后的 RETURN 可以加，也可以不加上去
postgres$$ END;
postgres$$ $$
postgres-# LANGUAGE PLPGSQL;
CREATE FUNCTION
postgres=#
postgres=# BEGIN;
BEGIN
postgres=# SELECT * FROM f22();
          f22
-----

```

<unnamed portal 13>

<unnamed portal 14>

(2 行记录)

```
postgres=# FETCH ALL FROM "<unnamed portal 13>";
```

```
id |   mc
```

```
----+-----
```

```
1 | TBase
```

```
2 | pgxz
```

(2 行记录)

```
postgres=# FETCH ALL FROM "<unnamed portal 14>";
```

```
id | yhm   | nc   | mm
```

```
----+-----+-----+-----
```

```
1 | TBase   | TBase   | 202cb962ac59075b964b07152d234b70
```

```
2 | pgxc    | pgxc    | 202cb962ac59075b964b07152d234b70
```

```
3 | pgxz    | pgxz    | 202cb962ac59075b964b07152d234b70
```

(3 行记录)

```
postgres=# COMMIT;
```

```
COMMIT
```

```
postgres=# CREATE OR REPLACE FUNCTION f22(a_ref1 refcursor,a_ref2 refcursor) RETURNS SETOF
refcursor AS
```

```
postgres-# $$
```

```
postgres$$ BEGIN
```

```
postgres$$      --指定游标名称
```

```
postgres$$      OPEN a_ref1 FOR SELECT * FROM t;
```

```
postgres$$      OPEN a_ref2 FOR SELECT * FROM t1;
```

```
postgres$$      RETURN NEXT a_ref1;
```

```
postgres$$      RETURN NEXT a_ref2;
```

```
postgres$$      RETURN ;--最后的 RETURN 可以加，也可以不加上去
```

```
postgres$$ END;
```

```
postgres$$ $$
```

```
postgres=# LANGUAGE PLPGSQL;
CREATE FUNCTION
postgres=#
postgres=# BEGIN;
BEGIN
postgres=#
postgres=# SELECT * FROM f22('a','b');
  f22
-----
 a
 b
(2 行记录)
```

```
postgres=# FETCH ALL FROM "a";
 id |  mc
----+-----
  1 | TBase
  2 | pgxz
(2 行记录)
```

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

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

6.3.2.9、返回多态类型

```
postgres=# CREATE OR REPLACE FUNCTION f23(a_arg anyelement) RETURNS anyelement AS
postgres=# $$
postgres$# BEGIN
postgres$#     RETURN a_arg;
postgres$# END;
postgres$# $$
postgres=# LANGUAGE PLPGSQL;
CREATE FUNCTION
postgres=#
postgres=# SELECT * FROM f23('TBase'::text);
```

```
f23
```

```
-----
```

```
TBase
```

```
(1 行记录)
```

```
postgres=# SELECT * FROM f23(1::integer);
```

```
f23
```

```
-----
```

```
1
```

```
(1 行记录)
```

```
postgres=# SELECT * FROM f23(ARRAY['TBase','pgxz']);
```

```
f23
```

```
-----
```

```
{TBase,pgxz}
```

```
(1 行记录)
```

```
postgres=# SELECT * FROM f23(ROW(1,'TBase')::public.t_rec);
```

```
id | mc
```

```
----+-----
```

```
1 | TBase
```

```
(1 行记录)
```

```
postgres=# CREATE OR REPLACE FUNCTION f24(a_arg ANYARRAY) RETURNS anyarray AS
```

```
postgres=# $$
```

```
postgres$# BEGIN
```

```
postgres$#     RETURN a_arg;
```

```
postgres$# END;
```

```
postgres$# $$
```

```
postgres=# LANGUAGE PLPGSQL;
```

```
CREATE FUNCTION
```

```
postgres=#
```

```
postgres=# SELECT * FROM f24(ARRAY[1,2]::INTEGER[]);
```

```
f24
```

```
-----
```

```
{1,2}
```

```
(1 行记录)
```

```
postgres=# SELECT f24(ARRAY[t1.*]) FROM t1;
```

```
f24
```

```
-----
```

```
{"(1,TBase,TBase,202cb962ac59075b964b07152d234b70)"}
```

```
{"(2,pgxc,pgxc,202cb962ac59075b964b07152d234b70)"}
```

```
{"(3,pgxz,pgxz,202cb962ac59075b964b07152d234b70)"}
```

```
(3 行记录)
```

返回数据类型如果是多态，则函数最少需要定义一个多态参数

6.4、变量使用

6.4.1、变量使用介绍

在一个块中使用的所有变量必须在该块的声明小节中事先进行声明,PL/pgSQL 变量可以是任意 SQL 数据类型，可以是一个简单数据类型、复合类型、RECORD、已经存在的表行类型、表字段类型、游标。

6.4.2、变量使用实例

6.4.2.1、变量声明语法

```
name [ CONSTANT ] type [ COLLATE collation_name ] [ NOT NULL ] [ { DEFAULT | := | = } expression ];
```

如果给定 DEFAULT 子句，它会指定进入该块时分配给该变量的初始值。如果没有给出 DEFAULT 子句，则该变量被初始化为 SQL 空值。CONSTANT 选项阻止该变量在初始化之后被赋值，这样它的值在块的持续期内保持不变。COLLATE 选项指定用于该变量的一个排序规则（见第 41.3.6 节）。如果指定了 NOT NULL，对该变量赋值为空值会导致一个运行时错误。所有被声明为 NOT NULL 的变量必须被指定一个非空默认值。等号 (=) 可以被用来代替 PL/SQL-兼容的 :=。

6.4.2.2、定义一个普通变量

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

(1 row)

```
postgres=#
```

6.4.2.3、定义 CONSTANT 变量

```
postgres=# CREATE OR REPLACE FUNCTION f25() RETURNS VOID AS
postgres-# $$
postgres$# DECLARE
postgres$#     v_int CONSTANT integer := 1;
postgres$# BEGIN
postgres$#     RAISE NOTICE 'v_int = %',v_int;
postgres$# END;
postgres$# $$
postgres-# LANGUAGE plpgsql;
CREATE FUNCTION
postgres=# select f25();
NOTICE:  v_int = 1
 f25
-----
```

(1 row)

CONSTANT 不能再次赋值

```
postgres=# CREATE OR REPLACE FUNCTION f25() RETURNS VOID AS
postgres-# $$
postgres$# DECLARE
postgres$#     v_int CONSTANT integer := 1;
postgres$# BEGIN
postgres$#     RAISE NOTICE 'v_int = %',v_int;
postgres$#     v_int = 10;
postgres$#     RAISE NOTICE 'v_int = %',v_int;
postgres$# END;
postgres$# $$
postgres-# LANGUAGE plpgsql;
ERROR:  "v_int" is declared CONSTANT
```

6.4.2.4、定义 NOT NULL 变量

```
postgres=# CREATE OR REPLACE FUNCTION f25() RETURNS VOID AS
postgres-# $$
postgres$# DECLARE
postgres$#     v_int integer NOT NULL := 1;
postgres$# BEGIN
```

```

postgres$#      RAISE NOTICE 'v_int = %',v_int;
postgres$#      SELECT NULL INTO v_int;
postgres$#      RAISE NOTICE 'v_int = %',v_int;
postgres$# END;
postgres$# $$
postgres-# LANGUAGE plpgsql;
CREATE FUNCTION
postgres=# SELECT f25();
NOTICE:   v_int = 1
ERROR:   null value cannot be assigned to variable "v_int" declared NOT NULL
CONTEXT:  PL/pgSQL function f25() line 6 at SQL statement
postgres=#

```

定义为 NOT NULL 变量，则该变量受 NOT NULL 约束

6.4.2.5、定义 COLLATE 变量

按 unicode 值对比大小

```

postgres=# CREATE OR REPLACE FUNCTION f25() RETURNS VOID AS
postgres-# $$
postgres$# DECLARE
postgres$#      v_txt1 TEXT COLLATE "C" := '严';
postgres$#      v_txt2 TEXT COLLATE "C" := '丰';
postgres$# BEGIN
postgres$#      IF v_txt1 > v_txt2 THEN
postgres$#          RAISE NOTICE '% -> %',v_txt1,v_txt2;
postgres$#      ELSE
postgres$#          RAISE NOTICE '% -> %',v_txt2,v_txt1;
postgres$#      END IF;
postgres$# END;
postgres$# $$
postgres-# LANGUAGE plpgsql;
CREATE FUNCTION
postgres=# SELECT f25();
NOTICE:   丰 -> 严
f25
-----

```

(1 row)

```

postgres=# select '严'::bytea;
bytea
-----
\x4b8a5
(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)
```

6.4.2.6、变量赋值

```
postgres=# CREATE OR REPLACE FUNCTION f25() RETURNS VOID AS
postgres-# $$
postgres$# DECLARE
postgres$#     --定义时赋值
postgres$#     v_int1 integer = 1;
postgres$#     --使用 :=兼容于 plsql
postgres$#     v_int2 integer := 1;
postgres$#     v_txt1 text;
postgres$#     v_float float8;
postgres$#     --使用查询赋值
postgres$#     v_relname text = (select relname FROM pg_class LIMIT 1);
postgres$#     v_relpages integer;
postgres$#     v_rec RECORD;
```

```

postgres## BEGIN
postgres##      --在函数体中赋值
postgres##      v_txt1 = 'TBase';
postgres##      v_float = random();
postgres##      --使用查询赋值的另一种方式
postgres##      SELECT relname,relpages INTO v_relname,v_relpages FROM   pg_class ORDER BY random()
LIMIT 1;
postgres##      RAISE NOTICE 'v_relname = % , relpages = %',v_relname,v_relpages;
postgres## END;
postgres## $$
postgres-# LANGUAGE plpgsql;
CREATE FUNCTION
postgres=# SELECT * FROM f25();
NOTICE:  v_relname = pg_ts_parser , relpages = 1
f25
-----

(1 row)

```

6.5、控制结构

6.5.1、判断语句

6.5.1.1、IF...THEN...END IF

```

postgres=# CREATE OR REPLACE FUNCTION f26() RETURNS VOID AS
postgres-# $$
postgres## BEGIN
postgres##      IF random()>0.5 THEN
postgres##          RAISE NOTICE '随机数大于 0.5';
postgres##      END IF;
postgres## END;
postgres## $$
postgres-# LANGUAGE plpgsql;
CREATE FUNCTION
postgres=# select f26();
NOTICE:  随机数大于 0.5
f26
-----

(1 row)

postgres=#

```


6.5.1.2、IF...THEN...ELSE...END IF

```

postgres=# CREATE OR REPLACE FUNCTION f26() RETURNS VOID AS
postgres-# $$
postgres$# BEGIN
postgres$#     IF random()>0.99 THEN
postgres$#         RAISE NOTICE '随机数大于 0.99';
postgres$#     ELSE
postgres$#         RAISE NOTICE '随机数小于或等于 0.99';
postgres$#     END IF;
postgres$# END;
postgres$# $$
postgres-# LANGUAGE plpgsql;
CREATE FUNCTION
postgres=# select f26();
NOTICE:  随机数小于或等于 0.99
 f26
-----
(1 row)

postgres=#

```

6.5.1.3、IF...THEN...ELSIF...THEN...ELSE...END IF

```

postgres=# CREATE OR REPLACE FUNCTION f26() RETURNS VOID AS
postgres-# $$
postgres$# DECLARE
postgres$#     v_float8 float8 := random();
postgres$# BEGIN
postgres$#     IF v_float8>0.99 THEN
postgres$#         RAISE NOTICE '随机数大于 0.99';
postgres$#     ELSIF v_float8>0.5 THEN
postgres$#         RAISE NOTICE '随机数大于 0.50';
postgres$#     ELSIF v_float8>0.25 THEN
postgres$#         RAISE NOTICE '随机数大于 0.25';
postgres$#     ELSE
postgres$#         RAISE NOTICE '随机数小于或等于 0.25';
postgres$#     END IF;
postgres$# END;
postgres$# $$
postgres-# LANGUAGE plpgsql;
CREATE FUNCTION
postgres=# SELECT f26();
NOTICE:  随机数大于 0.50

```

f26

(1 row)

6.5.1.4、CASE 语句

```
postgres=# CREATE OR REPLACE FUNCTION f26() RETURNS VOID AS
postgres-# $$
postgres$# DECLARE
postgres$#     v_float8 float8 := random();
postgres$# BEGIN
postgres$#     CASE
postgres$#     WHEN v_float8>0.99 THEN
postgres$#         RAISE NOTICE '随机数大于 0.99';
postgres$#     WHEN v_float8>0.5 THEN
postgres$#         RAISE NOTICE '随机数大于 0.50';
postgres$#     WHEN v_float8>0.25 THEN
postgres$#         RAISE NOTICE '随机数大于 0.25';
postgres$#     ELSE
postgres$#         RAISE NOTICE '随机数小于或等于 0.25';
postgres$#     END CASE;
postgres$# END;
postgres$# $$
postgres-# LANGUAGE plpgsql;
CREATE FUNCTION
postgres=# SELECT f26();
NOTICE: 随机数大于 0.50
f26
-----
```

(1 row)

6.5.2、循环语句

6.5.2.1、LOOP 循环

```
postgres=# CREATE OR REPLACE FUNCTION f27() RETURNS VOID AS
postgres-# $$
postgres$# DECLARE
postgres$#     v_id INTEGER := 1;
postgres$# BEGIN
postgres$#     LOOP
postgres$#         RAISE NOTICE '%',v_id;
postgres$#         EXIT WHEN random()>0.8;
```

```

postgres$#          v_id := v_id + 1;
postgres$#          END LOOP ;
postgres$# END;
postgres$# $$
postgres-# LANGUAGE plpgsql;
CREATE FUNCTION
postgres=# SELECT f27();
NOTICE:  1
NOTICE:  2
f27
-----

```

(1 row)

使用 EXIT 退出循环

```

postgres=# CREATE OR REPLACE FUNCTION f27() RETURNS VOID AS
postgres-# $$
postgres$# DECLARE
postgres$#      v_id INTEGER := 1;
postgres$#      v_random float8 ;
postgres$# BEGIN
postgres$#      LOOP
postgres$#          RAISE NOTICE '%',v_id;
postgres$#          v_id := v_id + 1;
postgres$#          v_random := random();
postgres$#          IF v_random > 0.8 THEN
postgres$#              RETURN;
postgres$#          END IF;
postgres$#      END LOOP ;
postgres$# END;
postgres$# $$
postgres-# LANGUAGE plpgsql;
CREATE FUNCTION
postgres=# SELECT f27();
NOTICE:  1
NOTICE:  2
NOTICE:  3
NOTICE:  4
NOTICE:  5
f27
-----

```

(1 row)

```

postgres=#

```

使用 RETURN 退出循环返回

6.5.2.2、WHILE 循环

```
postgres=# CREATE OR REPLACE FUNCTION f27() RETURNS VOID AS
postgres-# $$
postgres$# DECLARE
postgres$#     v_id INTEGER := 1;
postgres$#     v_random float8 := random() ;
postgres$# BEGIN
postgres$#     WHILE v_random > 0.8 LOOP
postgres$#         RAISE NOTICE '%',v_id;
postgres$#         v_id := v_id + 1;
postgres$#         v_random = random();
postgres$#     END LOOP;
postgres$# END;
postgres$# $$
postgres-# LANGUAGE plpgsql;
CREATE FUNCTION
postgres=# SELECT f27();
NOTICE:  1
f27
-----

(1 row)
```

6.5.2.3、FOR 循环

```
postgres=# CREATE OR REPLACE FUNCTION f27() RETURNS VOID AS
postgres-# $$
postgres$# BEGIN
postgres$#     FOR i IN 1..3 LOOP
postgres$#         RAISE NOTICE 'i = %',i;
postgres$#     END LOOP;
postgres$# END;
postgres$# $$
postgres-# LANGUAGE plpgsql;
CREATE FUNCTION
postgres=# SELECT f27();
NOTICE:  i = 1
NOTICE:  i = 2
NOTICE:  i = 3
f27
-----
```

(1 row)

```
postgres=# CREATE OR REPLACE FUNCTION f27() RETURNS VOID AS
postgres-# $$
postgres$# BEGIN
postgres$#     FOR i IN REVERSE 3..1 LOOP
postgres$#         RAISE NOTICE 'i = %',i;
postgres$#     END LOOP;
postgres$# END;
postgres$# $$
postgres-# LANGUAGE plpgsql;
CREATE FUNCTION
postgres=# SELECT f27();
NOTICE:  i = 3
NOTICE:  i = 2
NOTICE:  i = 1
f27
-----
```

(1 row)

使用 REVERSE 递减

```
postgres=# CREATE OR REPLACE FUNCTION f27() RETURNS VOID AS
postgres-# $$
postgres$# BEGIN
postgres$#     FOR i IN 1..8 BY 2 LOOP
postgres$#         RAISE NOTICE 'i = %',i;
postgres$#     END LOOP;
postgres$# END;
postgres$# $$
postgres-# LANGUAGE plpgsql;
CREATE FUNCTION
postgres=# SELECT f27();
NOTICE:  i = 1
NOTICE:  i = 3
NOTICE:  i = 5
NOTICE:  i = 7
f27
-----
```

(1 row)

使用 BY 设置步长

6.5.2.4、FOR 循环查询结果

```

postgres=# CREATE OR REPLACE FUNCTION f27() RETURNS VOID AS
postgres-# $$
postgres$$ DECLARE
postgres$$     v_rec RECORD;
postgres$$ BEGIN
postgres$$     FOR v_rec IN SELECT * FROM public.t LOOP
postgres$$         RAISE NOTICE '%',v_rec;
postgres$$     END LOOP;
postgres$$ END;
postgres$$ $$
postgres-# LANGUAGE plpgsql;
CREATE FUNCTION
postgres=# SELECT f27();
NOTICE:  (1,TBase)
NOTICE:  (2,pgxz)
  f27
-----

(1 row)

```

6.5.2.5、FOREACH 循环一个数组

```

postgres=# CREATE OR REPLACE FUNCTION f27() RETURNS VOID AS
postgres-# $$
postgres$$ DECLARE
postgres$$     v_random_arr float8[]:=ARRAY[random(),random()];
postgres$$     v_random float8;
postgres$$ BEGIN
postgres$$     FOREACH v_random IN ARRAY v_random_arr LOOP
postgres$$         RAISE NOTICE '%',v_random ;
postgres$$     END LOOP;
postgres$$ END;
postgres$$ $$
postgres-# LANGUAGE plpgsql;
CREATE FUNCTION
postgres=# SELECT f27();
NOTICE:  0.452758576720953
NOTICE:  0.975814974401146
  f27
-----

(1 row)

```

```

postgres=# CREATE OR REPLACE FUNCTION f27() RETURNS VOID AS
postgres-# $$
postgres$# DECLARE
postgres$#     v_random_arr float8[][]:=ARRAY[ARRAY[random(),random()],ARRAY[random(),random()]];
postgres$#     v_random float8;
postgres$# BEGIN
postgres$#     FOREACH v_random SLICE 0 IN ARRAY v_random_arr LOOP
postgres$#         RAISE NOTICE '%',v_random ;
postgres$#     END LOOP;
postgres$# END;
postgres$# $$
postgres-# LANGUAGE plpgsql;
CREATE FUNCTION
postgres=# SELECT f27();
NOTICE:  0.0588191924616694
NOTICE:  0.368828620761633
NOTICE:  0.813376842066646
NOTICE:  0.415377039927989
 f27
-----

(1 row)

```

循环会通过计算 `expression` 得到的数组的个体元素进行迭代

```

postgres=# CREATE OR REPLACE FUNCTION f27() RETURNS VOID AS
postgres-# $$
postgres$# DECLARE
postgres$#     v_random_arr float8[][]:=ARRAY[ARRAY[random(),random()],ARRAY[random(),random()]];
postgres$#     v_random float8[];
postgres$# BEGIN
postgres$#     FOREACH v_random SLICE 1 IN ARRAY v_random_arr LOOP
postgres$#         RAISE NOTICE '%',v_random ;
postgres$#     END LOOP;
postgres$# END;
postgres$# $$
postgres-# LANGUAGE plpgsql;
CREATE FUNCTION
postgres=# SELECT f27();
NOTICE:  {0.578366641886532,0.78098024148494}
NOTICE:  {0.783956411294639,0.450278480071574}
 f27
-----

(1 row)

```

通过一个正 SLICE 值，FOREACH 通过数组的切片而不是单一元素迭代

6.5.3、其它控制语句

6.5.3.1、动态执行

```
postgres=# CREATE OR REPLACE FUNCTION f27(a_id INTEGER) RETURNS text AS
postgres-# $$
postgres$# DECLARE
postgres$#     v_sql TEXT;
postgres$#     v_mc TEXT;
postgres$# BEGIN
postgres$#     v_sql := 'SELECT mc FROM t WHERE id='||a_id::TEXT;
postgres$#     EXECUTE v_sql INTO v_mc;
postgres$#     RETURN v_mc;
postgres$# END;
postgres$# $$
postgres-# LANGUAGE plpgsql;
CREATE FUNCTION
postgres=# SELECT f27(1);
 f27
-----
TBase
(1 row)
```

动态执行就是拼 sql 语句,然后使用 EXECUTE 命令执行

6.5.3.2、执行一个没有结果的命令

```
postgres=# CREATE OR REPLACE FUNCTION f27() RETURNS void AS
postgres-# $$
postgres$# BEGIN
postgres$#     perform f27(1);
postgres$# END;
postgres$# $$
postgres-# LANGUAGE plpgsql;
CREATE FUNCTION
postgres=# SELECT f27();
 f27
-----

(1 row)

postgres=#
```


6.5.3.3、获取执行结果

```

postgres=# DROP FUNCTION f27(INTEGER);
DROP FUNCTION
postgres=# CREATE OR REPLACE FUNCTION f27(a_id INTEGER) RETURNS VOID AS
postgres-# $$
postgres$# DECLARE
postgres$#     v_mc TEXT;
postgres$# BEGIN
postgres$#     SELECT mc INTO v_mc FROM t WHERE id=a_id;
postgres$#     IF FOUND THEN
postgres$#         RAISE NOTICE '查询到记录，值为%',v_mc;
postgres$#     ELSE
postgres$#         RAISE NOTICE '查不到记录';
postgres$#     END IF;
postgres$# END;
postgres$# $$
postgres-# LANGUAGE plpgsql;
CREATE FUNCTION
postgres=# SELECT f27(1);
NOTICE:  查询到记录，值为 TBase
f27
-----
(1 row)

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

```

6.5.3.4、获取影响行数

```

postgres=# CREATE OR REPLACE FUNCTION f27(a_id INTEGER) RETURNS VOID AS
postgres-# $$
postgres$# DECLARE
postgres$#     v_mc TEXT;
postgres$#     v_row_count BIGINT;
postgres$# BEGIN
postgres$#     SELECT mc INTO v_mc FROM t WHERE id=a_id;
postgres$#     GET DIAGNOSTICS v_row_count = ROW_COUNT;
postgres$#     RAISE NOTICE '查询到的记录数为 % ',v_row_count;

```

```
postgres$# END;
postgres$# $$
postgres-# LANGUAGE plpgsql;
CREATE FUNCTION
```

```
postgres=# SELECT f27(1);
NOTICE:  查询到的记录数为 1
 f27
-----
```

```
(1 row)
postgres=# SELECT f27(3);
NOTICE:  查询到的记录数为 0
 f27
-----
```

```
(1 row)
```

6.5.4、俘获错误

6.5.4.1、错误俘获处理

```
postgres=# CREATE TABLE t_exception (id integer not null,nc text);
CREATE TABLE
postgres=# create unique index t_exception_id_uidx on t_exception using btree(id);
CREATE INDEX
postgres=# CREATE OR REPLACE FUNCTION f27(a_id integer,a_nc text) RETURNS TEXT AS
postgres-# $$
postgres$# BEGIN
postgres$#     INSERT INTO t_exception VALUES(a_id,a_nc);
postgres$#     RETURN "";
postgres$#     EXCEPTION WHEN OTHERS THEN
postgres$#     RETURN '执行出错';
postgres$# END;
postgres$# $$
postgres-# LANGUAGE plpgsql;
CREATE FUNCTION
postgres=# SELECT f27(1,'TBase');
 f27
-----

(1 row)
```

```
postgres=# SELECT f27(1,'TBase');
```

f27

 执行出错
 (1 row)

6.5.4.2、获取错误相关信息

```
postgres=# CREATE OR REPLACE FUNCTION f27(a_id integer,a_nc text) RETURNS TEXT AS
postgres-# $$
postgres$# DECLARE
postgres$#     v_sqlstate text;
postgres$#     v_context text;
postgres$#     v_message_text text;
postgres$# BEGIN
postgres$#     INSERT INTO t_exception VALUES(a_id,a_nc);
postgres$#     RETURN ";
postgres$#     EXCEPTION WHEN OTHERS THEN
postgres$#     GET STACKED DIAGNOSTICS v_sqlstate = RETURNED_SQLSTATE,
postgres$#                                     v_message_text = MESSAGE_TEXT,
postgres$#                                     v_context = PG_EXCEPTION_CONTEXT;
postgres$#     RAISE NOTICE '错误代码 : %',v_sqlstate;
postgres$#     RAISE NOTICE '出错信息 : %',v_message_text;
postgres$#     RAISE NOTICE '发生异常语句 : %',v_context;
postgres$#     RETURN '错误代码 : '||v_sqlstate || '\n 出错信息 : '||v_message_text|| '发生异常语句 : '
postgres$#     ||v_context;
postgres$# END;
postgres$# $$
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)

6.6、触发器函数

6.7.1、INSERT 事件触发器函数

函数功能实现字段值 t_trigger.nc 值重写

```
postgres=# CREATE TABLE t_trigger
postgres=# (
postgres=#     id integer NOT NULL,
postgres=#     nc text NOT NULL
postgres=# );
CREATE TABLE
postgres=# CREATE OR REPLACE FUNCTION t_trigger_insert_trigger_func() RETURNS trigger AS
postgres=# $$
postgres$$ BEGIN
postgres$$     IF NEW.nc = " THEN
postgres$$         NEW.nc = 'TBase_' || random()::text;
postgres$$     END IF;
postgres$$     RETURN NEW;
postgres$$ END;
postgres$$ $$
postgres=# LANGUAGE plpgsql;
CREATE FUNCTION
postgres=# CREATE TRIGGER t_trigger_insert_trigger BEFORE INSERT ON t_trigger FOR EACH ROW
EXECUTE PROCEDURE t_trigger_insert_trigger_func();
CREATE TRIGGER
postgres=# INSERT INTO t_trigger values(1,");
INSERT 0 1
postgres=# SELECT * FROM t_trigger ;
 id |          nc
----+-----
  1 | TBase_0.426093454472721
(1 row)
```

注意使用 BEFORE,不能使用 AFTER,否则重写失效

6.7.2、UPDATE 事件触发器函数

不准许更新 t_trigger.nc 字段值为 TBase

```
postgres=# CREATE OR REPLACE FUNCTION t_trigger_update_trigger_func() RETURNS trigger AS
postgres=# $$
postgres$$ BEGIN
postgres$$     --不准许 t_trigger.nc 值为 TBase
```

```

postgres##      IF NEW.nc = 'TBase' THEN
postgres##          NEW.nc = OLD.nc ;
postgres##      END IF;
postgres##      RETURN NEW;
postgres## END;
postgres## $$
postgres-# LANGUAGE plpgsql;
CREATE FUNCTION
postgres=# CREATE TRIGGER t_trigger_update_trigger BEFORE UPDATE ON t_trigger FOR EACH ROW
EXECUTE PROCEDURE t_trigger_update_trigger_func();
CREATE TRIGGER
postgres=# UPDATE t_trigger SET nc='TBase' WHERE id=1;
UPDATE 1
postgres=# SELECT * FROM t_trigger ;
 id |      nc
----+-----
  1 | TBase_0.426093454472721
(1 row)

postgres=#

```

6.7.3、DELETE 事件触发器函数

限制 TBase 记录不能被删除

```

postgres=# CREATE OR REPLACE FUNCTION t_trigger_delete_trigger_func() RETURNS trigger AS
postgres-# $$
postgres## BEGIN
postgres##      --不准许 t_trigger.nc 值为 TBase
postgres##      IF OLD.nc = 'TBase' THEN
postgres##          RETURN NULL;
postgres##          --RAISE EXCEPTION 'TBase 不能被删除';
postgres##      END IF;
postgres##      RETURN OLD;
postgres## END;
postgres## $$
postgres-# LANGUAGE plpgsql;
CREATE FUNCTION
postgres=# CREATE TRIGGER t_trigger_delete_trigger BEFORE DELETE ON t_trigger FOR EACH ROW
EXECUTE PROCEDURE t_trigger_delete_trigger_func();
CREATE TRIGGER
postgres=# INSERT INTO t_trigger VALUES(2,'TBase');
INSERT 0 1
postgres=# SELECT * t_trigg

```

```
postgres=# SELECT * FROM t_trigger ;
 id |          nc
----+-----
  1 | TBase_0.426093454472721
  2 | TBase
(2 rows)
postgres=# DELETE FROM t_trigger WHERE id=2;
DELETE 0
postgres=# SELECT * FROM t_trigger ;
 id |          nc
----+-----
  1 | TBase_0.426093454472721
  2 | TBase
(2 rows)
```

6.7.4、删除触发器

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

6.7.5、触发器使用限制

分区表，冷热分区表和复制表不支持使用触发器

6.7.6、更多的触发器函数使用

更多的触发器函数使用见文档

<http://www.postgres.cn/docs/10/plpgsql-trigger.html>

6.7、消息及异常输出

6.7.1、RAISE NOTICE

```
postgres=# CREATE OR REPLACE FUNCTION f28() RETURNS VOID AS
postgres-# $$
postgres$# DECLARE
postgres$#     v_int INTEGER := 1;
postgres$# BEGIN
postgres$#     RAISE NOTICE 'v_int = %, 随机数 = %',v_int,random();
postgres$# END;
postgres$# $$
postgres-# LANGUAGE plpgsql;
```

```
CREATE FUNCTION
postgres=# SELECT f28();
NOTICE:  v_int = 1, 随机数 = 0.236714988015592
 f28
-----

(1 row)
```

使用 `raise notice` 向终端输出一个消息,也有可能写到日志中(需要调整日志的保存级别)

6.7.2、RAISE EXCEPTION

```
postgres=# CREATE OR REPLACE FUNCTION f28() RETURNS VOID AS
postgres-# $$
postgres$# DECLARE
postgres$#     v_int INTEGER := 1;
postgres$# BEGIN
postgres$#     RAISE EXCEPTION '程序 EXCEPTION';
postgres$#     --下面的语句不会再执行
postgres$#     RAISE NOTICE 'v_int = %, 随机数 = %',v_int,random();
postgres$# END;
postgres$# $$
postgres-# LANGUAGE plpgsql;
CREATE FUNCTION
postgres=# SELECT f28();
ERROR:  程序 EXCEPTION
```

如果在事务中执行这个函数,则事务会中止(`abort`)

6.7.3、RAISE EXCEPTION 自定义 ERRCODE

```
postgres=# CREATE OR REPLACE FUNCTION f28() RETURNS VOID AS
postgres-# $$
postgres$# DECLARE
postgres$#     v_int INTEGER := 1;
postgres$# BEGIN
postgres$#     RAISE EXCEPTION ' 程序 EXCEPTION ' USING ERRCODE = '23505';
postgres$# END;
postgres$# $$
postgres-# LANGUAGE plpgsql;
CREATE FUNCTION
postgres=# SELECT f28();
ERROR:  程序 EXCEPTION
```

日志中会记录这个 `ERRCODE`

```

2017-10-03 18:40:16.710 CST,"pgxz","postgres",15072,"[local]",59d33b65.3ae0,225,"idle",2017-10-03 15:25:25
CST,4/367159,0,LOG,00000,"statement: SELECT f28();" ,,,,,,,,,,"psql"
2017-10-03 18:40:16.710 CST,"pgxz","postgres",15072,"[local]",59d33b65.3ae0,226,"SELECT",2017-10-03
15:25:25 CST,4/367159,0,ERROR,23505," 程序 EXCEPTION " ,,,,,,,,,,"psql"

```

6.8、pg/pgsql 函数实战

6.8.1、批量设置表 owner 的函数

```

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

COMMENT ON FUNCTION public.alter_owner(a_schema_name varchar,a_role_name varchar) IS '批量设置表所有者';

```



```

CREATE OR REPLACE FUNCTION public.alter_owner(a_role_name varchar) RETURNS TEXT AS
$$
BEGIN
    RETURN public.alter_owner(",a_role_name);
END;
$$
LANGUAGE PLPGSQL;

COMMENT ON FUNCTION public.alter_owner(a_role_name varchar) IS '批量设置表所有者重载';

```

6.8.2、批量设置表的加密规则函数

```

create or replace function MLS_TRANSPARENT_CRYPT_ALGORITHM_BIND_ALL_TABLE(a_schema
text,a_algo_id int) returns text as
$$
declare
    v_rec record;
    v_algorithm_name text;
    v_raise_notice text;
    v_sqlstate text;
    v_context text;
    v_message_text text;
begin
    perform 1 from pg_namespace where nspname=a_schema ;
    if not found then
        return '模式["||a_schema||"]不存在';
    end if;

    --显示使用的加密算法
    select algorithm_name INTO v_algorithm_name from pg_transparent_crypt_policy_algorithm where
algorithm_id=a_algo_id;
    if not found then
        return '加密算法 id["||a_algo_id::text||"]不存在';
    else
        raise notice '你使用的密码算法为--%',v_algorithm_name;
    end if;

    for v_rec in select pg_tables.schemaname,pg_tables.tablename,pg_transparent_crypt_policy_map.tblname
from pg_tables left outer join pg_transparent_crypt_policy_map on
pg_tables.schemaname=pg_transparent_crypt_policy_map.nspname and
pg_tables.tablename=pg_transparent_crypt_policy_map.tblname where pg_tables.schemaname=a_schema and
pg_transparent_crypt_policy_map.tblname is null loop
    begin
        PERFORM
        MLS_TRANSPARENT_CRYPT_ALGORITHM_BIND_TABLE(v_rec.schemaname,v_rec.tablename,

```

```

a_algo_id);
    EXCEPTION WHEN OTHERS THEN
    GET STACKED DIAGNOSTICS v_sqlstate = RETURNED_SQLSTATE,
                           v_message_text = MESSAGE_TEXT,
                           v_context = PG_EXCEPTION_CONTEXT;
    RAISE NOTICE '出错信息 : %',v_message_text;
end;
end loop;
return '配置表加密完成';
end;
$$
language plpgsql;

```

6.8.3、oracle to_date 函数的实现

Oracle 的 to_date 函数可以精确处理到年-月-日 时:分:秒, TBase 只能处理 年-月-日

TBase 可以使用 to_timestamp 函数代替, 如果应用程序中已经大量的使用的 oracle to_date 函数并且是精细到时:分:秒, 那我们可以自定义一个 to_date 函数存放到一个优先访问的 schema 中, 函数的内容为 to_timestamp, 这样就可以兼容原来的 oracle 应用了, 如下所示

```

postgres=# create schema oracle ;
CREATE SCHEMA
postgres=# CREATE OR REPLACE FUNCTION oracle.to_date(a_date text,a_style text ) RETURNS timestamp
with time zone AS
postgres-# $$
postgres$# BEGIN
postgres$#     RETURN to_timestamp(a_date,a_style);
postgres$# END;
postgres$# $$
postgres-# LANGUAGE PLPGSQL;
CREATE FUNCTION
postgres=# set search_path = oracle ,"$user", public,pg_catalog;;
SET
postgres=# select to_date('2028-01-01 13:14:20','yyyy-MM-dd HH24:MI');
           to_date
-----
2028-01-01 13:14:00+08
(1 row)

```

7、问题定位及性能优化

7.1、访问日志管理

7.1.1、配置 TBase 日志

TBase 运行参数配置文件 `Postgresql.conf` 中涉及日志配置参数如下所示

#日志相关配置

#用户访问日志格式支持多种方法来记录服务器消息，包括 `stderr`、`csvlog`

#如果 `csvlog` 被包括在 `log_destination` 中，日志项会以"逗号分隔值"（CSV）格式被输出，这样可以很方便地把日志载入到程序中

`log_destination = 'csvlog'`

#启用用户访问日志收集器

`logging_collector = on`

#日志存放目录，可以是相对路径（相对了 `data` 目录）或绝对路径

`log_directory = 'pg_log'`

#设置日志文件的权限

`log_file_mode = 0600`

#按我先前经常是只有 `log_rotation_age` 这个值有效果

`log_truncate_on_rotation = 'on'`

#120 分钟切换一次

`log_rotation_age = 120`

#64MB 切换一次

`log_rotation_size = 64MB`

#会话的当前执行命令保留长度，用于在字段 `pg_stat_activity.query` 中显示

`track_activity_query_size = 4096`

#配置 sql 语句执行超过多少毫秒数时，语句将被记录,值为-1 时禁用，0 时记录所有语句

`log_min_duration_statement = 1000`

#是否记录 checkpoint

`log_checkpoints = on`

#是否记录连接

`log_connections = on`

#是否记录断开连接

log_disconnections = on

#是否记录 autovacuum 执行日志

log_autovacuum_min_duration = 0

#是否记录所有语句的执行时间，值为 on 时将单独记录所有语句的执行时间，这里不记录语句

log_duration = off

#stderr 配置日志记录的内容

log_line_prefix = '%h'

#控制那些类型的语句被记录，有效值是 none (off)、ddl、mod 和 all（所有语句）。

#如果 log_min_duration_statement 值为 0 时，则 log_statement 什么值的效果都一样

#如果 log_min_duration_statement 值大于 0，并且 log_statement 为 ddl 则 ddl 语句全部表被记录，为两条 log
#dml 超时才被记录，为一条记录

log_statement = 'none'

#设置在服务器日志中写入的时间戳的时区

log_timezone = 'PRC'

log_filename = 'postgresql-%A-%H.log'

#是否记录 autovacuum 执行日志

log_autovacuum_min_duration = 0

#控制被发送给客户端的消息级别

#其值有 debug5,debug4,debug3,debug2,debug1,log,notice,warning,error

client_min_messages = notice

#控制哪些消息级别被写入到服务器日

#其值有 debug5,debug4,debug3,debug2,debug1,log,notice,warning,error

log_min_messages = warning

#控制哪些导致一个错误情况的 SQL 语句被记录在服务器日志中

#默认值是 ERROR,它表示导致错误、日志消息、致命错误或恐慌错误的语句将被记录在日志中

log_min_error_statement = error

7.1.2、TBase 日志格式说明

执行正确的语句产生的日志

```
2017-10-11 16:23:55.178 CST,"pgxz","postgres",11499,"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"

```

向 TBase 执行错误的语句会产生两条日志，一条是执行语句，一条提示出错的原因

7.1.3、对日志进行分析

7.1.3.1、创建日志表

```

CREATE table tbase_log
(
    log_time timestamp without time zone,
    user_name text,
    database_name text,
    process_id integer,

```

```

connection_from text,
session_id text,
session_line_num bigint,
command_tag text,
session_start_time timestamp without time zone,
virtual_transaction_id text,
transaction_id bigint,
error_severity text,
sql_state_code text,
message text,
detail text,
hint text,
internal_query text,
internal_query_pos integer,
context text,
query text,
query_pos integer,
location text,
application_name text
);

```

7.1.3.2、导入日志数据

TBase 日志文件默认存储在“数据目录/pg_log”目录下

```

postgres=# COPY tbase_log FROM '/data/pgxz/data/pgxz/dn001/pg_log/postgresql-Tuesday-16.csv' WITH csv;
COPY 10790

```

7.1.3.3、统计日志数据

--按照 session 连接及操作时间排序

```

postgres=# select * from tbase_log order by process_id,log_time;

```

--查询错误日志

```

SELECT * FROM TBase_log WHERE error_severity='ERROR' limit 1;

```

--统计 session 操作数统计

```

postgres=# select count(1),process_id,user_name,database_name from tbase_log group by
process_id,user_name,database_name order by count(1) desc limit 10;

```

```

count | process_id | user_name | database_name
-----+-----+-----+-----

```

```

2770 | 48067 | pgxz | postgres

```

```

10 |      22143 | pgxz      | postgres
10 |      28778 | pgxz      | postgres
 9 |      28367 | pgxz      | postgres
 9 |      44280 | pgxz      | postgres
 8 |      32442 | pgxz      | postgres
 7 |      17911 | pgxz      | postgres
 7 |      21865 | pgxz      | postgres
 7 |      26159 | pgxz      | postgres
 7 |      45471 | pgxz      | postgres
(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_name,database_name order by count(1) desc limit 10;
count | user_name | database_name
-----+-----+-----
1390 | pgxz      | postgres
(1 row)

```

7.1.4、配置只收集慢的 sql 语句

#用户访问日志格式

```
log_destination = 'csvlog'
```

#启用用户访问日志收集器

```
logging_collector = on
```

#配置 sql 语句执行超过多少毫秒数时，语句将被记录,值为-1 时禁用，0 时记录所有语句

#下面配置只收集运行超过 1 秒的语句

```
log_min_duration_statement = 1000
```

```
#默认不记录任何日志
```

```
log_statement = 'none'
```

收集到的日志文件内容如下所示

```
2017-10-15 10:25:54.106
CST,"postgres","postgres",43799,"127.0.0.1:17899",59e2c65b.ab17,4,"SELECT",2017-10-15 10:22:19
CST,2/0,0,LOG,00000,"duration: 1338.366 ms statement: select * from t where id=20000 or
id=2000000;",,,,,,,,,,"psql"
```

系统记录运行的语句及运行时间

7.2、如何查询数据是否倾斜

连接上不同的 dn 节点，查询表的容量大小，如果大小偏差较大就可以判断存在数据倾斜

```
[pgxz@VM_0_29_centos pgxz]$ psql -p 15432 -h 172.16.0.47
psql (PostgreSQL 10 (TBase 2.01))
Type "help" for help.
```

```
postgres=# select pg_size_pretty(pg_table_size('TBase_1'));
pg_size_pretty
-----
2408 kB
(1 row)
```

```
postgres=# select pg_size_pretty(pg_table_size('TBase_2'));
pg_size_pretty
-----
896 kB
(1 row)
```

```
postgres=# \q
[pgxz@VM_0_29_centos pgxz]$ psql -p 5431 -h 172.16.0.47
psql (PostgreSQL 10 (TBase 2.01))
Type "help" for help.
```

```
postgres=# select pg_size_pretty(pg_table_size('TBase_1'));
pg_size_pretty
-----
2408 kB
(1 row)
```

```
postgres=# select pg_size_pretty(pg_table_size('TBase_2'));
```



```
pg_size_pretty
-----
464 kB
(1 row)
```

上面数据表“TBase_2”容量相差为一半，基本可以判定存在数据倾斜。更方便的办法请参考运维文档 7.5.3。

7.3、如何优化有问题的 Ssql 语句

7.3.1、查看是否为分布键查询

```
postgres=# explain select * from tbase_1 where f1=1;
               QUERY PLAN
-----
Remote Fast Query Execution  (cost=0.00..0.00 rows=0 width=0)
  Node/s: dn001, dn002
    -> Gather  (cost=1000.00..7827.20 rows=1 width=14)
          Workers Planned: 2
            -> Parallel Seq Scan on tbase_1  (cost=0.00..6827.10 rows=1 width=14)
                  Filter: (f1 = 1)
(6 rows)
```

```
postgres=# explain select * from tbase_1 where f2=1;
               QUERY PLAN
-----
Remote Fast Query Execution  (cost=0.00..0.00 rows=0 width=0)
  Node/s: dn001
    -> Gather  (cost=1000.00..7827.20 rows=1 width=14)
          Workers Planned: 2
            -> Parallel Seq Scan on tbase_1  (cost=0.00..6827.10 rows=1 width=14)
                  Filter: (f2 = 1)
(6 rows)
```

上面第一个查询为非分布键查询，需要发往所有节点，这样最慢的节点决定了整个业务的速度，需要保持所有节点的响应性能一致，**业务设计查询时尽可能带上分布键**

7.3.2、查看是否使用上索引

```
postgres=# create index tbase_2_f2_idx on tbase_2(f2);
CREATE INDEX
postgres=# explain select * from tbase_2 where f2=1;
               QUERY PLAN
-----
Remote Fast Query Execution  (cost=0.00..0.00 rows=0 width=0)
```

Node/s: dn001, dn002

-> Index Scan using tbase_2_f2_idx on tbase_2 (cost=0.42..4.44 rows=1 width=14)

Index Cond: (f2 = 1)

(4 rows)

postgres=# explain select * from tbase_2 where f3='1';

QUERY PLAN

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

Node/s: dn001, dn002

-> Gather (cost=1000.00..7827.20 rows=1 width=14)

Workers Planned: 2

-> Parallel Seq Scan on tbase_2 (cost=0.00..6827.10 rows=1 width=14)

Filter: (f3 = '1'::text)

(6 rows)

postgres=#

第一个查询使用了索引，第二个没有使用索引，通常情况下，使用索引可以加速查询速度，但要记住索引也会增加更新的开销

7.3.3、查看是否为分布 key join

postgres=# explain select tbase_1.* from tbase_1,tbase_2 where tbase_1.f1=tbase_2.f1 ;

QUERY PLAN

Remote Subquery Scan on all (dn001,dn002) (cost=29.80..186.32 rows=3872 width=40)

-> Hash Join (cost=29.80..186.32 rows=3872 width=40)

Hash Cond: (tbase_1.f1 = tbase_2.f1)

-> Remote Subquery Scan on all (dn001,dn002) (cost=100.00..158.40 rows=880 width=40)

Distribute results by S: f1

-> Seq Scan on tbase_1 (cost=0.00..18.80 rows=880 width=40)

-> Hash (cost=18.80..18.80 rows=880 width=4)

-> Seq Scan on tbase_2 (cost=0.00..18.80 rows=880 width=4)

(8 rows)

postgres=# explain select tbase_1.* from tbase_1,tbase_2 where tbase_1.f2=tbase_2.f1 ;

QUERY PLAN

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

Node/s: dn001, dn002

-> Hash Join (cost=18904.69..46257.08 rows=500564 width=14)

Hash Cond: (tbase_1.f2 = tbase_2.f1)

-> Seq Scan on tbase_1 (cost=0.00..9225.64 rows=500564 width=14)

-> Hash (cost=9225.64..9225.64 rows=500564 width=4)

```
-> Seq Scan on tbase_2 (cost=0.00..9225.64 rows=500564 width=4)
```

```
(7 rows)
```

第一查询需要数据重分布，而第二个是不需要，分布键 join 查询性能会更高。

7.3.4、查看 join 发生的节点

```
postgres=# explain select tbase_1.* from tbase_1,tbase_2 where tbase_1.fl=tbase_2.fl ;
```

```
QUERY PLAN
```

```
-----
Hash Join (cost=29.80..186.32 rows=3872 width=40)
```

```
Hash Cond: (tbase_1.fl = tbase_2.fl)
```

```
-> 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.fl=tbase_2.fl ;
```

```
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.fl = tbase_2.fl)
```

```
-> Remote Subquery Scan on all (dn001,dn002) (cost=100.00..158.40 rows=880 width=40)
```

```
Distribute results by S: fl
```

```
    -> 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 上进行少量数据 join 性能会更好

7.3.5、查看并行的 worker 数

```
postgres=# explain select count(1) from tbase_1;
```

```
QUERY PLAN
```

```
-----
Finalize Aggregate (cost=118.81..118.83 rows=1 width=8)
```

```
-> Remote Subquery Scan on all (dn001,dn002) (cost=118.80..118.81 rows=1 width=0)
```

```
    -> Partial Aggregate (cost=18.80..18.81 rows=1 width=8)
```

```
-> Seq Scan on tbase_1 (cost=0.00..18.80 rows=880 width=0)
```

```
(4 rows)
```

```
postgres=# analyze tbase_1;
```

```
ANALYZE
```

```
postgres=# explain select count(1) from tbase_1;
```

QUERY PLAN

```
-----
Parallel Finalize Aggregate (cost=14728.45..14728.46 rows=1 width=8)
```

```
-> Parallel Remote Subquery Scan on all (dn001,dn002) (cost=14728.33..14728.45 rows=1 width=0)
```

```
-> Gather (cost=14628.33..14628.44 rows=1 width=8)
```

```
Workers Planned: 2
```

```
-> Partial Aggregate (cost=13628.33..13628.34 rows=1 width=8)
```

```
-> Parallel Seq Scan on tbase_1 (cost=0.00..12586.67 rows=416667 width=0)
```

```
(6 rows)
```

上面第一个查询没走并行，analyze 后走并行才是正确的，建议大数据量更新再执行 analyze。

7.3.6、检查各个节点的执行计划是否一致

```
./tbase_run_sql_dn_master.sh "explain select * from tbase_2 where f2=1"
```

```
dn006 --- psql -h 172.16.0.13 -p 11227 -d postgres -U tbase -c "explain select * from tbase_2 where f2=1"
```

QUERY PLAN

```
-----
Bitmap Heap Scan on tbase_2 (cost=2.18..7.70 rows=4 width=40)
```

```
Recheck Cond: (f2 = 1)
```

```
-> Bitmap Index Scan on tbase_2_f2_idx (cost=0.00..2.18 rows=4 width=0)
```

```
Index Cond: (f2 = 1)
```

```
(4 rows)
```

```
dn002 --- psql -h 172.16.0.42 -p 11012 -d postgres -U tbase -c "explain select * from tbase_2 where f2=1"
```

QUERY PLAN

```
-----
Index Scan using tbase_2_f2_idx on tbase_2 (cost=0.42..4.44 rows=1 width=14)
```

```
Index Cond: (f2 = 1)
```

```
(2 rows)
```

这两个 dn 的执行计划不一致，最大可能可以是数据倾斜或者是执行计划给禁用。

如果有可能的话，DBA 可以配置在系统空闲时执行全库 analyze 和 vacuum。

7.4、优化实例

7.4.1、count(distinct xx)优化

```
postgres=# CREATE TABLE t1(f1 serial not null unique,f2 text,f3 text,f4 text,f5 text,f6 text,f7 text,f8 text,f9
text,f10 text,f11 text,f12 text) distribute by shard(f1);
```

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

```
CREATE TABLE
```

Time: 89.938 ms

```
postgres=#          insert          into          t1          select
t,md5(t::text),md5(t::text),md5(t::text),md5(t::text),md5(t::text),md5(t::text),md5(t::text),md5(t::text),
md5(t::text),md5(t::text) from generate_series(1,1000000) as t;
```

```
INSERT 0 1000000
```

Time: 14849.045 ms (00:14.849)

```
postgres=# analyze t1;
```

```
ANALYZE
```

Time: 1340.387 ms (00:01.340)

```
postgres=# explain (verbose) select count(distinct f2) from t1;
```

QUERY PLAN

```
-----
Aggregate  (cost=103320.00..103320.01 rows=1 width=8)
  Output: count(DISTINCT f2)
    -> Remote Subquery Scan on all (dn01,dn02,dn03,dn04,dn05,dn06,dn07,dn08,dn09,dn10)  (cost=100.00..100820.00 rows=1000000 width=33)
          Output: f2
            -> Seq Scan on public.t1  (cost=0.00..62720.00 rows=1000000 width=33)
                  Output: f2
```

(6 rows)

Time: 0.748 ms

```
postgres=# select count(distinct f2) from t1;
```

```
count
```

```
-----
1000000
(1 row)
```

Time: 6274.684 ms (00:06.275)

```
postgres=# select count(distinct f2) from t1 where f1 <10;
```

```
count
```

```
-----
9
(1 row)
```

Time: 19.261 ms

上面发现 `count(distinct f2)` 是发生在 `cn` 节点，对于 `TP` 类业务，需要操作的数据量少的情况下，性能开销是没有问题的，而且往往比下推执行的性能开销还要小。但如果一次要操作的数据量比较大的 `ap` 类业务，则网络传输就会成功瓶颈，下面看看改写后的执行计划

```
postgres=# explain (verbose) select count(1) from (select f2 from t1 group by f2) as t ;
```

QUERY PLAN

```
-----
Finalize Aggregate (cost=355600.70..355600.71 rows=1 width=8)
  Output: count(1)
  -> Remote Subquery Scan on all (dn01,dn02,dn03,dn04,dn05,dn06,dn07,dn08,dn09,dn10) (cost=355600.69..355600.70 rows=1 width=0)
        Output: PARTIAL count(1)
        -> Partial Aggregate (cost=355500.69..355500.70 rows=1 width=8)
              Output: PARTIAL count(1)
              -> Group (cost=340500.69..345500.69 rows=1000000 width=33)
                    Output: t1.f2
                    Group Key: t1.f2
                    -> Sort (cost=340500.69..343000.69 rows=1000000 width=0)
                          Output: t1.f2
                          Sort Key: t1.f2
                          -> Remote Subquery Scan on all (dn01,dn02,dn03,dn04,dn05,dn06,dn07,dn08,dn09,dn10) (cost=216192.84..226192.84 rows=1000000 width=0)
                                Output: t1.f2
                                Distribute results by S: f2
                                -> Group (cost=216092.84..221092.84 rows=1000000 width=33)
                                      Output: t1.f2
                                      Group Key: t1.f2
                                      -> Sort (cost=216092.84..218592.84 rows=1000000 width=33)
                                            Output: t1.f2
                                            Sort Key: t1.f2
                                            -> Seq Scan on public.t1 (cost=0.00..62720.00 rows=1000000 width=33)
                                                  Output: t1.f2

(23 rows)
```

改写后，并行推到 `dn` 去执行，现在看看执行的效果

```
postgres=# select count(1) from (select f2 from t1 group by f2) as t ;
```

count

```
-----
1000000
(1 row)
```

Time: 1328.431 ms (00:01.328)

```
postgres=# select count(1) from (select f2 from t1 where f1<10 group by f2) as t ;
```

count

```
-----
          9
(1 row)
```

Time: 24.991 ms

postgres=#

我们可以看到对于大量数据计算的 AP 类业务，性能提高了 5 倍

7.4.2、增大 work_mem 减少 io 访问

```
postgres=# CREATE TABLE t1(f1 serial not null unique,f2 text,f3 text,f4 text,f5 text,f6 text,f7 text,f8 text,f9
text,f10 text,f11 text,f12 text) distribute by shard(f1);
```

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

```
CREATE TABLE
```

Time: 70.545 ms

```
postgres=# CREATE TABLE t2(f1 serial not null unique,f2 text,f3 text,f4 text,f5 text,f6 text,f7 text,f8 text,f9
text,f10 text,f11 text,f12 text) distribute by shard(f1);
```

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

```
CREATE TABLE
```

Time: 61.913 ms

```
postgres=#          insert          into          t1          select
t,md5(t::text),md5(t::text),md5(t::text),md5(t::text),md5(t::text),md5(t::text),md5(t::text),md5(t::text),
md5(t::text),md5(t::text) 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) from generate_series(1,50000) as t;
```

```
INSERT 0 50000
```

Time: 792.858 ms

```
postgres=# analyze t1;
```

```
ANALYZE
```

Time: 175.946 ms

```
postgres=# analyze t2;
```

```
ANALYZE
```

Time: 318.802 ms

postgres=#

```
postgres=# explain  select * from t1 where f2 not in (select f2 from t2);
```

QUERY PLAN

```

-----
Remote Subquery Scan on all (dn01,dn02,dn03,dn04,dn05,dn06,dn07,dn08,dn09,dn10) (cost=0.00..2076712.50 rows=500 width=367)
-> Seq Scan on t1 (cost=0.00..2076712.50 rows=500 width=367)
    Filter: (NOT (SubPlan 1))
    SubPlan 1
        -> Materialize (cost=0.00..4028.00 rows=50000 width=33)
            -> Remote Subquery Scan on all (dn01,dn02,dn03,dn04,dn05,dn06,dn07,dn08,dn09,dn10) (cost=0.00..3240.00 rows=50000 width=33)
                -> Seq Scan on t2 (cost=0.00..3240.00 rows=50000 width=33)
(7 rows)

```

Time: 0.916 ms

postgres=# select * from t1 where f2 not in (select f2 from t2);

f1 | f2 | f3 | f4 | f5 | f6 | f7 | f8 | f9 | f10 | f11 | f12

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

(0 rows)

Time: 4226.825 ms (00:04.227)

postgres=# set work_mem to '8MB';

SET

Time: 0.289 ms

postgres=# explain select * from t1 where f2 not in (select f2 from t2);

QUERY PLAN

```

-----
Remote Subquery Scan on all (dn01,dn02,dn03,dn04,dn05,dn06,dn07,dn08,dn09,dn10) (cost=3365.00..3577.50 rows=500 width=367)
-> Seq Scan on t1 (cost=3365.00..3577.50 rows=500 width=367)
    Filter: (NOT (hashed SubPlan 1))
    SubPlan 1
        -> Remote Subquery Scan on all (dn01,dn02,dn03,dn04,dn05,dn06,dn07,dn08,dn09,dn10) (cost=0.00..3240.00 rows=50000 width=33)
            -> Seq Scan on t2 (cost=0.00..3240.00 rows=50000 width=33)
(6 rows)

```

Time: 0.890 ms

postgres=# select * from t1 where f2 not in (select f2 from t2);

f1 | f2 | f3 | f4 | f5 | f6 | f7 | f8 | f9 | f10 | f11 | f12

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

(0 rows)

Time: 105.249 ms

postgres=#

增大 work_mem 后，性能提高了 40 倍，因为 work_mem 足够放下 filter 的数据，不需要再做 Materialize 物化，filter 由原来的 subplan 变成了 hash subplan，直接在内存 hash 表中 filter，性能就上去了

上面 7.4.2 通过增大计算内存达到提高性能，但内存不可能无限扩大，下面通过改写语句也可以达到提高查询的性能。

QUERY PLAN

(10 rows)

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)

```
postgres=#
```

也可以修改 not exists

QUERY PLAN

```

Remote Subquery Scan on all (dn01,dn02,dn03,dn04,dn05,dn06,dn07,dn08,dn09,dn10) (cost=3865.00..4078.75 rows=1 width=367)
-> Hash Anti Join (cost=3865.00..4078.75 rows=1 width=367)
    Hash Cond: (t1.f2 = t2.f2)
    -> Remote Subquery Scan on all (dn01,dn02,dn03,dn04,dn05,dn06,dn07,dn08,dn09,dn10) (cost=100.00..682.00 rows=1000 width=367)
        Distribute results by S: f2
        -> Seq Scan on t1 (cost=0.00..210.00 rows=1000 width=367)
    -> Hash (cost=5240.00..5240.00 rows=50000 width=33)

```

```
-> Remote Subquery Scan on all (dn01,dn02,dn03,dn04,dn05,dn06,dn07,dn08,dn09,dn10) (cost=100.00..5240.00 rows=50000 width=33)
    Distribute results by S: f2
    -> Seq Scan on t2 (cost=0.00..3240.00 rows=50000 width=33)
```

(10 rows)

Time: 0.974 ms

```
postgres=# select * from t1 where not exists( select 1 from t2 where t1.f2=t2.f2);
```

```
 f1 | f2 | f3 | f4 | f5 | f6 | f7 | f8 | f9 | f10 | f11 | f12
```

```
-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----
```

(0 rows)

Time: 42.944 ms

```
postgres=#
```

7.4.4、分布 key join+limit 优化

--数据准备

```
postgres=# CREATE TABLE t1(f1 serial not null unique,f2 text,f3 text,f4 text,f5 text,f6 text,f7 text,f8 text,f9
text,f10 text,f11 text,f12 text) distribute by shard(f1);
```

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

```
CREATE TABLE
```

```
postgres=# CREATE TABLE t2(f1 serial not null unique,f2 text,f3 text,f4 text,f5 text,f6 text,f7 text,f8 text,f9
text,f10 text,f11 text,f12 text) distribute by shard(f1);
```

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

```
CREATE TABLE
```

```
postgres=#          insert          into          t1          select
t,md5(t::text),md5(t::text),md5(t::text),md5(t::text),md5(t::text),md5(t::text),md5(t::text),md5(t::text),md5(t::text),
md5(t::text),md5(t::text) from generate_series(1,1000000) as t;
```

```
INSERT 0 1000000
```

```
postgres=#          insert          into          t2          select
t,md5(t::text),md5(t::text),md5(t::text),md5(t::text),md5(t::text),md5(t::text),md5(t::text),md5(t::text),md5(t::text),
md5(t::text),md5(t::text) from generate_series(1,1000000) as t;
```

```
INSERT 0 1000000
```

```
postgres=# analyze t1;
```

```
ANALYZE
```

```
postgres=# analyze t2;
```

```
ANALYZE
```

```
postgres=#
```

```
postgres=# \timing
```

Timing is on.

```
postgres=# explain select t1.* from t1,t2 where t1.f1=t2.f1 limit 10;
```

QUERY PLAN

```

-----
Limit (cost=0.25..1.65 rows=10 width=367)
-> Merge Join (cost=0.25..140446.26 rows=1000000 width=367)
    Merge Cond: (t1.f1 = t2.f1)
    -> Remote Subquery Scan on all (dn01,dn02,dn03,dn04,dn05,dn06,dn07,dn08,dn09,dn10) (cost=100.12..434823.13 rows=1000000 width=367)
        -> Index Scan using t1_f1_key on t1 (cost=0.12..62723.13 rows=1000000 width=367)
    -> Remote Subquery Scan on all (dn01,dn02,dn03,dn04,dn05,dn06,dn07,dn08,dn09,dn10) (cost=100.12..71823.13 rows=1000000 width=4)
        -> Index Only Scan using t2_f1_key on t2 (cost=0.12..62723.13 rows=1000000 width=4)
(7 rows)

```

Time: 1.372 ms

postgres=# explain analyze select t1.* from t1,t2 where t1.f1=t2.f1 limit 10;

QUERY PLAN

```

-----
Limit (cost=0.25..1.65 rows=10 width=367) (actual time=2675.437..2948.199 rows=10 loops=1)
-> Merge Join (cost=0.25..140446.26 rows=1000000 width=367) (actual time=2675.431..2675.508 rows=10 loops=1)
    Merge Cond: (t1.f1 = t2.f1)
    -> Remote Subquery Scan on all (dn01,dn02,dn03,dn04,dn05,dn06,dn07,dn08,dn09,dn10) (cost=100.12..434823.13 rows=1000000 width=367)
(actual time=1.661..1.704 rows=10 loops=1)
    -> Remote Subquery Scan on all (dn01,dn02,dn03,dn04,dn05,dn06,dn07,dn08,dn09,dn10) (cost=100.12..71823.13 rows=1000000 width=4)
(actual time=2673.761..2673.783 rows=10 loops=1)
Planning time: 0.358 ms
Execution time: 2973.948 ms
(7 rows)

```

Time: 2976.008 ms (00:02.976)

postgres=#

看执行计划是在 cn 上面执行，merge join 需要把要 join 的数据拉回 cn 再排序，然后再 join，这里主切的开销在于网络，优化的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

7.4.5、非分布 key join 使用 hash join 性能一般最好

为了提高 tp 类业务查询的性能，我们经常需要对一些字段建立索引，使用有索引字段 join 时系统往往也会使用 Merge Cond 和 nestloop

mydb=# CREATE TABLE t1(f1 serial not null,f2 text,f3 text,f4 text,f5 text,f6 text,f7 text,f8 text,f9 text,f10 text,f11 text,f12 text) distribute by shard(f1);

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

CREATE TABLE

Time: 481.042 ms

mydb=# create index t1_f1_idx on t1(f2);

CREATE INDEX

Time: 85.521 ms

mydb=# CREATE TABLE t2(f1 serial not null,f2 text,f3 text,f4 text,f5 text,f6 text,f7 text,f8 text,f9 text,f10 text,f11 text,f12 text) distribute by shard(f1);

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

CREATE TABLE

Time: 75.973 ms

mydb=# create index t2_f1_idx on t2(f2);

CREATE INDEX

Time: 29.890 ms

mydb=#	insert	into	t1	select
--------	--------	------	----	--------

```
t,md5(t::text),md5(t::text),md5(t::text),md5(t::text),md5(t::text),md5(t::text),md5(t::text),md5(t::text),md5(t::text),
md5(t::text),md5(t::text) from generate_series(1,1000000) as t;
```

```
INSERT 0 1000000
```

```
Time: 16450.623 ms (00:16.451)
```

```
mydb=#          insert          into          t2          select
t,md5(t::text),md5(t::text),md5(t::text),md5(t::text),md5(t::text),md5(t::text),md5(t::text),md5(t::text),md5(t::text),
md5(t::text),md5(t::text) from generate_series(1,1000000) as t;
```

```
INSERT 0 1000000
```

```
Time: 17218.738 ms (00:17.219)
```

```
mydb=# analyze t1;
```

```
ANALYZE
```

```
Time: 2219.341 ms (00:02.219)
```

```
mydb=# analyze t2;
```

```
ANALYZE
```

```
Time: 1649.506 ms (00:01.650)
```

```
mydb=#
```

```
--merge join
```

```
mydb=# explain select t1.* from t1,t2 where t1.f2=t2.f2 limit 10;
```

QUERY PLAN

```
-----
Limit  (cost=100.25..102.78 rows=10 width=367)
```

```
  -> Remote Subquery Scan on all (dn001,dn002)  (cost=100.25..102.78 rows=10 width=367)
```

```
    -> Limit  (cost=0.25..2.73 rows=10 width=367)
```

```
      -> Merge Join  (cost=0.25..248056.80 rows=1000000 width=367)
```

```
        Merge Cond: (t1.f2 = t2.f2)
```

```
          -> Remote Subquery Scan on all (dn001,dn002)  (cost=100.12..487380.85 rows=1000000 width=367)
```

```
            Distribute results by S: f2
```

```
              -> Index Scan using t1_f1_idx on t1  (cost=0.12..115280.85 rows=1000000 width=367)
```

```
          -> Materialize  (cost=100.12..155875.95 rows=1000000 width=33)
```

```
            -> Remote Subquery Scan on all (dn001,dn002)  (cost=100.12..153375.95 rows=1000000 width=33)
```

```
              Distribute results by S: f2
```

```
                -> Index Only Scan using t2_f1_idx on t2  (cost=0.12..115275.95 rows=1000000 width=33)
```

```
(12 rows)
```

```
Time: 4.183 ms
```

```
mydb=# explain analyze select t1.* from t1,t2 where t1.f2=t2.f2 limit 10;
```

QUERY PLAN

```
-----
Limit  (cost=100.25..102.78 rows=10 width=367) (actual time=6555.346..6556.296 rows=10 loops=1)
```

-> Remote Subquery Scan on all (dn001,dn002) (cost=100.25..102.78 rows=10 width=367) (actual time=6555.343..6555.349 rows=10 loops=1)

Planning time: 0.473 ms

Execution time: 6569.828 ms

(4 rows)

Time: 6614.439 ms (00:06.614)

--nested loop

mydb=# set enable_mergejoin to off;

SET

Time: 0.422 ms

mydb=# explain select t1.* from t1,t2 where t1.f2=t2.f2 limit 10;

QUERY PLAN

Limit (cost=100.12..103.57 rows=10 width=367)

-> Remote Subquery Scan on all (dn001,dn002) (cost=100.12..103.57 rows=10 width=367)

-> Limit (cost=0.12..3.52 rows=10 width=367)

-> Nested Loop (cost=0.12..339232.00 rows=1000000 width=367)

-> Remote Subquery Scan on all (dn001,dn002) (cost=100.00..434740.00 rows=1000000 width=367)

Distribute results by S: f2

-> Seq Scan on t1 (cost=0.00..62640.00 rows=1000000 width=367)

-> Materialize (cost=100.12..100.31 rows=1 width=33)

-> Remote Subquery Scan on all (dn001,dn002) (cost=100.12..100.30 rows=1 width=33)

Distribute results by S: f2

-> Index Only Scan using t2_f1_idx on t2 (cost=0.12..0.27 rows=1 width=33)

Index Cond: (f2 = t1.f2)

(12 rows)

Time: 1.033 ms

mydb=# explain analyze select t1.* from t1,t2 where t1.f2=t2.f2 limit 10;

QUERY PLAN

--

Limit (cost=100.12..103.57 rows=10 width=367) (actual time=5608.326..5609.571 rows=10 loops=1)

-> Remote Subquery Scan on all (dn001,dn002) (cost=100.12..103.57 rows=10 width=367) (actual time=5608.323..5608.349 rows=10 loops=1)

Planning time: 0.347 ms

Execution time: 5669.901 ms

(4 rows)

Time: 5672.584 ms (00:05.673)

mydb=# set enable_nestloop to off;

SET

Time: 0.436 ms

mydb=# explain select t1.* from t1,t2 where t1.f2=t2.f2 limit 10;

QUERY PLAN

```

-----
Limit (cost=85983.00..85984.94 rows=10 width=367)
-> Remote Subquery Scan on all (dn001,dn002) (cost=85983.00..85984.94 rows=10 width=367)
    -> Limit (cost=85883.00..85884.89 rows=10 width=367)
        -> Hash Join (cost=85883.00..274580.00 rows=1000000 width=367)
            Hash Cond: (t1.f2 = t2.f2)
            -> Remote Subquery Scan on all (dn001,dn002) (cost=100.00..434740.00 rows=1000000 width=367)
                Distribute results by S: f2
                -> Seq Scan on t1 (cost=0.00..62640.00 rows=1000000 width=367)
            -> Hash (cost=100740.00..100740.00 rows=1000000 width=33)
                -> Remote Subquery Scan on all (dn001,dn002) (cost=100.00..100740.00 rows=1000000 width=33)
                    Distribute results by S: f2
                    -> Seq Scan on t2 (cost=0.00..62640.00 rows=1000000 width=33)

```

(12 rows)

Time: 1.141 ms

mydb=# explain analyze select t1.* from t1,t2 where t1.f2=t2.f2 limit 10;

QUERY PLAN

```

-----
Limit (cost=85983.00..85984.94 rows=10 width=367) (actual time=1083.691..1085.962 rows=10 loops=1)
-> Remote Subquery Scan on all (dn001,dn002) (cost=85983.00..85984.94 rows=10 width=367) (actual time=1083.688..1083.699 rows=10 loops=1)
Planning time: 0.530 ms
Execution time: 1108.830 ms
(4 rows)

```

Time: 1117.713 ms (00:01.118)

mydb=#

7.4.6、exists 的优化

exists 在数据量比较大情况下，一般使用的是 Semi Join，在 work_mem 足够大的情况下走的是 hash join，性能会更好

postgres=# show work_mem;

work_mem

4MB

(1 row)

Time: 0.298 ms

postgres=# explain select count(1) from t1 where exists(select 1 from t2 where t2.t1_f1=t1.f1);

QUERY PLAN

```

Finalize Aggregate (cost=242218.32..242218.33 rows=1 width=8)
-> Remote Subquery Scan on all (dn001,dn002) (cost=242218.30..242218.32 rows=1 width=0)
    -> Partial Aggregate (cost=242118.30..242118.31 rows=1 width=8)
        -> Hash Semi Join (cost=110248.00..242118.30 rows=505421 width=0)
            Hash Cond: (t1.f1 = t2.t1_f1)
                -> Seq Scan on t1 (cost=0.00..17420.00 rows=1000000 width=4)
                -> Hash (cost=79340.00..79340.00 rows=3000000 width=4)
                    -> Remote Subquery Scan on all (dn001,dn002) (cost=100.00..79340.00
rows=3000000 width=4)

Distribute results by S: t1_f1
-> Seq Scan on t2 (cost=0.00..52240.00 rows=3000000 width=4)

(10 rows)

```

Time: 1.091 ms

postgres=# select count(1) from t1 where exists(select 1 from t2 where t2.t1_f1=t1.f1);

```

count
-----
500000
(1 row)

```

Time: 3779.401 ms (00:03.779)

postgres=# set work_mem to '128MB';

SET

Time: 0.368 ms

postgres=# explain select count(1) from t1 where exists(select 1 from t2 where t2.t1_f1=t1.f1);

QUERY PLAN

```

-----
Finalize Aggregate (cost=101763.76..101763.77 rows=1 width=8)
-> Remote Subquery Scan on all (dn001,dn002) (cost=101763.75..101763.76 rows=1 width=0)
    -> Partial Aggregate (cost=101663.75..101663.76 rows=1 width=8)
        -> Hash Join (cost=89660.00..101663.75 rows=505421 width=0)
            Hash Cond: (t2.t1_f1 = t1.f1)
                -> Remote Subquery Scan on all (dn001,dn002) (cost=59840.00..69443.00
rows=505421 width=4)

Distribute results by S: t1_f1
-> HashAggregate (cost=59740.00..64794.21 rows=505421 width=4)
    Group Key: t2.t1_f1
        -> Seq Scan on t2 (cost=0.00..52240.00 rows=3000000 width=4)
    -> Hash (cost=17420.00..17420.00 rows=1000000 width=4)
        -> Seq Scan on t1 (cost=0.00..17420.00 rows=1000000 width=4)

(12 rows)

```

Time: 4.739 ms

postgres=# select count(1) from t1 where exists(select 1 from t2 where t2.t1_f1=t1.f1);

```

count
-----

```



```
500000
```

```
(1 row)
```

```
Time: 1942.037 ms (00:01.942)
```

```
postgres=#
```

大约有一倍性能的提升

8、TBase-插件使用

8.1、插件查看，添加和删除

8.1.1、查看数据库加载了那些插件

```
postgres=# SELECT e.extname AS "Name", e.extversion AS "Version", n.nspname AS "Schema", c.description
AS "Description" FROM pg_catalog.pg_extension e LEFT JOIN pg_catalog.pg_namespace n ON n.oid =
e.extnamespace LEFT JOIN pg_catalog.pg_description c ON c.objoid = e.oid AND c.classoid =
'pg_catalog.pg_extension':pg_catalog.regclass ORDER BY 1;
```

Name	Version	Schema	Description
pageinspect	1.1	public	inspect the contents of database pages at a low level
pg_errcode_stat	1.1	public	track error code of all processes
pg_stat_statements	1.1	public	track execution statistics of all SQL statements executed
plpgsql	1.0	pg_catalog	PL/pgSQL procedural language
shard_statistic	1.0	public	tools for get shard statistic

```
(5 rows)
```

8.1.2、添加插件

```
postgres=# create extension "uuid-oss" with schema tbase;
CREATE EXTENSION
```

上面的语句把"uuid-oss"创建到模式 tbase 下

8.1.3、删除插件

```
postgres=# drop extension "uuid-oss" ;
DROP EXTENSION
```

8.2、插件 uuid-oss 使用

8.2.1、uuid-oss 功能介绍及添加方法

uuid-oss 模块提供函数使用几种标准算法之一产生通用唯一标识符（UUID）。还提供产生某些特殊 UUID 常量的函数。该模块提供的功能函数可用于替代序列的，而且性能比序列更好。

给数据库添加 uuid-oss 插件

```
postgres=# create extension "uuid-oss" with schema tbase;
CREATE EXTENSION
```

8.2.2、uuid 常量函数

8.2.2.1、uuid_nil()

这是一个"nil" UUID 常量

```
postgres=# select uuid_nil();
          uuid_nil
-----
00000000-0000-0000-0000-000000000000
(1 row)
```

8.2.2.2、uuid_ns_dns()

为 UUID 指定 DNS 名字空间的常量

```
postgres=# select uuid_ns_dns();
          uuid_ns_dns
-----
6ba7b810-9dad-11d1-80b4-00c04fd430c8
(1 row)
```

8.2.2.3、uuid_ns_url()

为 UUID 指定 URL 名字空间的常量。

```
postgres=# select uuid_ns_url();
          uuid_ns_url
-----
6ba7b811-9dad-11d1-80b4-00c04fd430c8
```

(1 row)

8.2.2.4、uuid_ns_oid()

为 UUID 指定 ISO 对象标识符 (OID) 名字空间的常量

```
postgres=# select uuid_ns_oid();
          uuid_ns_oid
-----
6ba7b812-9dad-11d1-80b4-00c04fd430c8
(1 row)
```

8.2.2.5、uuid_ns_x500()

为 UUID 指定 X.500 可识别名 (DN) 名字空间的常量

```
postgres=# select uuid_ns_x500();
          uuid_ns_x500
-----
6ba7b814-9dad-11d1-80b4-00c04fd430c8
(1 row)
```

8.2.3、uuid 生成函数

8.2.3.1、uuid_generate_v1()

这个函数产生一个版本 1 的 UUID。这涉及到计算机的 MAC 地址和一个时间戳。

```
postgres=# select uuid_generate_v1(),clock_timestamp();
          uuid_generate_v1          |          clock_timestamp
-----+-----
e909c7f0-c36b-11e7-984d-525400efe0ca | 2017-11-07 11:29:49.67391+08
(1 row)
```

```
postgres=# select uuid_generate_v1(),clock_timestamp();
          uuid_generate_v1          |          clock_timestamp
-----+-----
e9f24458-c36b-11e7-9f81-525400efe0ca | 2017-11-07 11:29:51.197516+08
(1 row)
```

8.2.3.2、uuid_generate_v1mc()

这个函数产生一个版本 1 的 UUID，但是使用一个随机广播 MAC 地址而不是该计算机真实的

MAC 地址。

```
postgres=# select uuid_generate_v1mc();
          uuid_generate_v1mc
-----
1538d848-c36c-11e7-9043-5749d89bc820
(1 row)
```

8.2.3.3、uuid_generate_v3(namespace uuid, name text)

这个函数使用指定的输入名称在给定的名字空间中产生一个版本 3 的 UUID,名称参数将使用 MD5 进行哈希,因此从产生的 UUID 中得不到明文

```
postgres=# SELECT uuid_generate_v3(uuid_ns_url(), 'http://tbase.qq.com');
          uuid_generate_v3
-----
af6371f5-bcd5-300b-8a35-7186d54009d5
(1 row)
```

```
postgres=# SELECT uuid_generate_v3(uuid_ns_dns(), 'http://tbase.qq.com');
          uuid_generate_v3
-----
246a9a9c-cff2-3b34-9ddd-80cc6e30a222
(1 row)
```

8.2.3.4、uuid_generate_v4()

这个函数产生一个版本 4 的 UUID,它完全从随机数产生。

```
postgres=# select uuid_generate_v4();
          uuid_generate_v4
-----
dfc68e17-6b97-496e-a992-531aca74ef18
(1 row)
```

```
postgres=# select uuid_generate_v4();
          uuid_generate_v4
-----
e96b64c2-cf1e-423b-845c-541e9b3901a3
(1 row)
```

8.2.3.5、uuid_generate_v5(namespace uuid, name text)

这个函数产生一个版本 5 的 UUID,它和版本 3 的 UUID 相似,但是采用的是 SHA-1 作为哈希方法。版本 5 比版本 3 更好,因为 SHA-1 被认为比 MD5 更安全。

```
postgres=# SELECT uuid_generate_v5(uuid_ns_url(), 'http://tbase.qq.com');
          uuid_generate_v5
```

```
-----
d8381c8e-6899-5e43-ab26-18d3660d7db1
(1 row)
```

```
postgres=# SELECT uuid_generate_v5(uuid_ns_dns(), 'http://tbase.qq.com');
          uuid_generate_v5
```

```
-----
d5051eb0-2051-53be-8d36-967851c3946d
(1 row)
```

8.2.4、在数据表中使用 uuid 默认值

```
postgres=# create table t_uuid(f1 uuid not null default uuid_generate_v1(),f2 varchar(256));
```

```
CREATE TABLE
```

```
postgres=# \d+ t_uuid
```

```

                                Table "public.t_uuid"
  Column |          Type          | Modifiers | Storage | Stats target |
Description
-----+-----+-----+-----+-----+
 f1      | uuid                   | not null default uuid_generate_v1() | plain   |              |
 f2      | character varying(256) |          | extended |              |
Has OIDs: no
Distribute By SHARD(f2)
Location Nodes: ALL DATANODES
```

```
postgres=# insert into t_uuid (f2) values(uuid_generate_v4());
```

```
INSERT 0 1
```

```
postgres=# select * from t_uuid;
```

```

          f1              |          f2
-----+-----
75dfb6b0-c382-11e7-9f82-525400efe0ca | 2b99799c-d77e-4023-81db-3e5c6ad901b0
(1 row)
```

8.2.5、uuid 各函数性能对比

■ 数据表结构

```
postgres=# \d+ t_uuid
```

```

                                Table "public.t_uuid"
  Column |          Type          | Modifiers | Storage | Stats target |
Description
```

```

-----+-----+-----+-----+-----+-----+
f1      | uuid                                | not null default uuid_generate_v1() | plain      |          |
f2      | character varying(256)              |                                     | extended   |          |
Has OIDs: no
Distribute By SHARD(f2)
Location Nodes: ALL DATANODES

```

■ uuid_generate_v3 函数

```

postgres=# insert into t_uuid(f1,f2) select uuid_generate_v3(uuid_ns_dns(),
'http://tbase.qq.com'),uuid_generate_v3(uuid_ns_dns(), 'http://tbase.qq.com') from generate_series(1,10000) as t;
INSERT 0 10000
Time: 58.407 ms

```

■ uuid_generate_v5 函数

```

postgres=# insert into t_uuid(f1,f2) select uuid_generate_v5(uuid_ns_dns(),
'http://tbase.qq.com'),uuid_generate_v5(uuid_ns_dns(), 'http://tbase.qq.com') from generate_series(1,10000) as t;
INSERT 0 10000
Time: 47.049 ms

```

■ uuid_generate_v1 函数

```

postgres=# insert into t_uuid(f1,f2) select uuid_generate_v1(),uuid_generate_v1() from generate_series(1,10000)
as t;
INSERT 0 10000
Time: 111.204 ms

```

■ uuid_generate_v4 函数

```

postgres=# insert into t_uuid(f1,f2) select uuid_generate_v4(),uuid_generate_v4() from generate_series(1,10000)
as t;
INSERT 0 10000
Time: 137.062 ms

```

■ uuid_generate_v1mc 函数

```

postgres=# insert into t_uuid(f1,f2) select uuid_generate_v1mc(),uuid_generate_v1mc() from
generate_series(1,10000) as t;
INSERT 0 10000
Time: 125.663 ms

```

■ 测试数据对比

V1	v1mc	V4	V3	V5
----	------	----	----	----

111.204 ms	125.663 ms	137.062 ms	58.407 ms	47.049 ms
------------	------------	------------	-----------	-----------

可以看出 V5 的性能是最好的

8.2.6、uuid 与 serial 在 TBase 中性能对比

■ uuid 插入 1 万条数据

```
postgres=# \d+ t_uuid
```

Table "public.t_uuid"					
Column	Type	Modifiers	Storage	Stats target	Description
f1	uuid	not null default uuid_generate_v1()	plain		
f2	character varying(256)		extended		
Has OIDs: no					
Distribute By SHARD(f2)					
Location Nodes: ALL DATANODES					

```
postgres=# insert into t_uuid(f2) select t from generate_series(1,1000000) as t;
INSERT 0 1000000
Time: 4869.140 ms (00:04.869)
```

■ serial 插入 1 万条数据

```
postgres=# \d+ t_serial
```

Table "public.t_serial"					
Column	Type	Modifiers	Storage	Stats target	Description
f1	integer	not null default nextval('t_serial_f1_seq'::regclass)	plain		
f2	character varying(256)		extended		
Has OIDs: no					
Distribute By SHARD(f1)					
Location Nodes: ALL DATANODES					

```
postgres=# insert into t_serial(f2) select t from generate_series(1,1000000) as t;
INSERT 0 1000000
Time: 3683.533 ms (00:03.684)
```

■ uuid pgbench 结果

```
pghost: 172.16.0.42 pgport: 11016 nclients: 32 duration: 600 dbName: postgres
transaction type: insert_t_uuid.sql
scaling factor: 1
```

query mode: prepared
 number of clients: 32
 number of threads: 1
 duration: 600 s
 number of transactions actually processed: 531667
 latency average = 36.166 ms
 tps = 884.807291 (including connections establishing)
 tps = 884.813103 (excluding connections establishing)
 script statistics:
 - statement latencies in milliseconds:
 0.013 \set id random(1, 10000000)
 36.094 insert into t_uuid(f2) values(:id::text) ;

■ serial pgbench 结果

pghost: 172.16.0.42 pgport: 11016 nclients: 32 duration: 600 dbName: postgres
 transaction type: insert_t_serial.sql
 scaling factor: 1
 query mode: prepared
 number of clients: 32
 number of threads: 1
 duration: 600 s
 number of transactions actually processed: 493799
 latency average = 38.889 ms
 tps = 822.864578 (including connections establishing)
 tps = 822.869984 (excluding connections establishing)
 script statistics:
 - statement latencies in milliseconds:
 0.013 \set id random(1, 10000000)
 38.861 insert into t_serial(f2) values(:id::text) ;

■ 测试数据对比

uuid 用时	Serial 用时	百分比	Uuid pgbench	Serial pgbench	百分比
4.869s	3.684s	75.66%	884	822	92.98%

8.2.7、占用空间对比

■ UUID 表

postgres=# \d+ t_uuid

Table "public.t_uuid"						
Column	Type	Collation	Nullable	Default	Storage	Stats target Description
f1	uuid		not null	uuid_generate_v1()	plain	


```
f2 | character varying(256) | | | extended |
```

Distribute By: SHARD(f2)

Location Nodes: ALL DATANODES

```
postgres=# select count(1),pg_size_pretty(pg_table_size('t_uuid')) from t_uuid;
```

```
count | pg_size_pretty
```

```
-----+-----
```

```
1000000 | 73 MB
```

```
(1 row)
```

■ Serial 表

```
postgres=# \d+ t_serial
```

Table "public.t_serial"

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

Description

```
-----+-----+-----+-----+-----+-----+-----
```

```
id | integer | | not null | nextval('t_serial_id_seq'::regclass) | plain | |
```

```
f2 | character varying(256) | | | | | extended |
```

```
|
```

Distribute By: SHARD(id)

Location Nodes: ALL DATANODES

```
postgres=# select count(1),pg_size_pretty(pg_table_size('t_serial')) from t_serial;
```

```
count | pg_size_pretty
```

```
-----+-----
```

```
1000000 | 66 MB
```

```
(1 row)
```

Time: 186.737 ms

■ 测试数据对比

Uuid/10000 条记录	Serial/10000 条记录	占用比
73MB	66MB	1.1 倍

8.2.8、使用 uuid 做为分布列的方案

目前 uuid 类型字段不能做为分布列，提示如下

```
postgres=# create table t_uuid(f1 uuid);
```

```
ERROR: No appropriate column can be used as distribute key because of data type.
```

解决方案如下

```
postgres=# create table t_uuid(fl varchar(36) default uuid_nil()::text );
CREATE TABLE
postgres=#
```

8.2.9、使用建议

性能上序列和 uuid 相关不大，但 serial 每次都需要与 gtm 通信，会加大 gtm 的通信压力，所以尽可能使用 uuid 来代替 serial

8.3、插件 pg_stat_statements 使用

8.3.1、pg_stat_statements 功能介绍及添加方法

pg_stat_statements 模块提供一种方法追踪一个服务器所执行的所有 SQL 语句的执行统计信息。该模块必须通过在 postgresql.conf 的 shared_preload_libraries 中增加 pg_stat_statements 来载入，因为它需要额外的共享内存。增加或移除该模块需要服务器重启才能生效。当 pg_stat_statements 被载入时，它会跟踪该服务器的所有数据库的统计信息。该模块提供了一个视图 pg_stat_statements 以及函数 pg_stat_statements_reset 和 pg_stat_statements 用于访问和操纵这些统计信息。

给数据库添加 pg_stat_statements 插件

```
postgres=# create extension pg_stat_statements with schema tbase;
CREATE EXTENSION
```

8.3.2、获取执行次数最多的语句

```
postgres=# select
    pg_authid.rolname as rolname,
    pg_database.datname as datname,
    pg_stat_statements.query,
    pg_stat_statements.calls,
    pg_stat_statements.total_time,
    round((pg_stat_statements.total_time/pg_stat_statements.calls)::numeric,6) as runtime_of_once
from
    pg_stat_statements
    inner join pg_authid on pg_authid.oid=pg_stat_statements.userid
    inner join pg_database on pg_database.oid= pg_stat_statements.dbid
order by
    pg_stat_statements.calls desc
limit 10;
```

8.3.3、获取执行总时间最长的语句

```

postgres=# select
    pg_authid.rolname as rolname,
    pg_database.datname as datname,
    pg_stat_statements.query,
    pg_stat_statements.calls,
    pg_stat_statements.total_time,
    pg_stat_statements.total_time/pg_stat_statements.calls AS onestime
from
    pg_stat_statements
    inner join pg_authid on pg_authid.oid=pg_stat_statements.userid
    inner join pg_database on pg_database.oid= pg_stat_statements.dbid
order by
    pg_stat_statements.total_time desc
limit 10;

```

8.3.4、获取每句平均执行时间最长的语句

```

postgres=# select
    pg_authid.rolname as rolname,
    pg_database.datname as datname,
    pg_stat_statements.query,
    pg_stat_statements.calls,
    pg_stat_statements.total_time,
    pg_stat_statements.total_time/pg_stat_statements.calls as onestime
from
    pg_stat_statements
    inner join pg_authid on pg_authid.oid=pg_stat_statements.userid
    inner join pg_database on pg_database.oid= pg_stat_statements.dbid
order by
    pg_stat_statements.total_time/pg_stat_statements.calls desc
limit 10;

```

8.3.5、获取 buffer 读最多的语句

```

postgres=# select
    pg_authid.rolname as rolname,
    pg_database.datname as datname,
    pg_stat_statements.query,
    pg_stat_statements.shared_blks_hit,
    pg_stat_statements.shared_blks_read,
    (pg_stat_statements.shared_blks_hit+pg_stat_statements.shared_blks_read) as all_blks,
    pg_stat_statements.calls,
    pg_stat_statements.total_time,
    pg_stat_statements.total_time/pg_stat_statements.calls

```

```

from
  pg_stat_statements
  inner join pg_authid on pg_authid.oid=pg_stat_statements.userid
  inner join pg_database on pg_database.oid= pg_stat_statements.dbid
order by
  pg_stat_statements.shared_blks_hit+pg_stat_statements.shared_blks_read desc
limit 10;

```

8.4、插件 pg_trgm 使用

8.4.1、pg_trgm 功能介绍及添加方法

前模糊，后模糊，前后模糊，正则匹配都属于文本搜索领域常见的需求。TBase 在文本搜索领域除了全文检索，还有 TRGM。对于前模糊和后模糊，TBase 与其他数据库一样，可以使用 btree 来加速。对于前后模糊和正则匹配，则可以使用 TRGM，TRGM 是一个非常强的插件，对这类文本搜索场景性能提升非常有效，100 万左右的数据量，性能提升有 100 倍以上。

8.4.2、添加 pg_trgm 插件

```
create extension pg_trgm;
```

8.4.3、测试环境准备

```

postgres=# create table t_trgm (id int,trgm text,no_trgm text) ;
NOTICE:  Replica identity is needed for shard table, please add to this table through "alter table" command.
CREATE TABLE

```

```

postgres=# \timing
Timing is on.

```

```

postgres=# insert into t_trgm select t,md5(t::text),md5(t::text) from generate_series(1,1000000) as t;
INSERT 0 1000000
Time: 4239.598 ms (00:04.240)postgres=#

```

8.4.4、gist 索引测试

8.4.4.1、创建索引消耗时间

```

postgres=# create index t_trgm_trgm_idx on t_trgm using gist(trgm gist_trgm_ops);
CREATE INDEX
Time: 24974.600 ms (00:24.975)

```

```
postgres=#
postgres=# vacuum ANALYZE t_trgm;
VACUUM
Time: 551.989 ms
```

8.4.4.2、索引占用空间

```
postgres=# select pg_size_pretty(pg_indexes_size('t_trgm'));
pg_size_pretty
-----
178 MB
(1 row)
```

8.4.4.3、模糊查询测试

■ 返回记录数多

```
postgres=# explain analyze select * from t_trgm where trgm ilike '%67%';
                                QUERY PLAN
-----
Remote Fast Query Execution  (cost=0.00..0.00 rows=0 width=0) (actual time=126.483..1172.196
rows=114475 loops=1)
  Node/s: dn001, dn002
  Planning time: 0.055 ms
  Execution time: 1197.320 ms
(4 rows)
```

```
postgres=# explain analyze select * from t_trgm where no_trgm ilike '%67%';
                                QUERY PLAN
-----
Remote Fast Query Execution  (cost=0.00..0.00 rows=0 width=0) (actual time=6.679..965.383 rows=114475
loops=1)
  Node/s: dn001, dn002
  Planning time: 0.056 ms
  Execution time: 989.840 ms
(4 rows)
```

使用 gist 索引开销反而更大

■ 返回记录比较少

```
postgres=# explain analyze select * from t_trgm where trgm ilike '%67a5%';
                                QUERY PLAN
-----
Remote Fast Query Execution  (cost=0.00..0.00 rows=0 width=0) (actual time=106.522..118.693 rows=481
```

```
loops=1)
Node/s: dn001, dn002
Planning time: 0.074 ms
Execution time: 118.831 ms
(4 rows)
```

```
postgres=# explain analyze select * from t_trgm where no_trgm ilike '%67a5%';
```

QUERY PLAN

```
-----
Remote Fast Query Execution (cost=0.00..0.00 rows=0 width=0) (actual time=190.056..502.639 rows=481
loops=1)
Node/s: dn001, dn002
Planning time: 0.063 ms
Execution time: 502.756 ms
(4 rows)
```

Time: 503.887 ms

过滤返回记录少，使用 gist 索引提高性能比较明显

8.4.4.4、数据导入时间测试

```
postgres=# truncate table t_trgm ;
TRUNCATE TABLE
Time: 78.705 ms
postgres=# insert into t_trgm select t,md5(t::text),md5(t::text) from generate_series(1,1000000) as t;
INSERT 0 1000000
Time: 29061.725 ms (00:29.062)
```

8.4.4.5、pgbench 并发查询测试

```
[tbase@VM_0_37_centos pgbench]$ cat select_t_trgm.sql
\set id random(1, 1000000)
select * from t_trgm where trgm ilike '%||md5(:id)||%';
```

```
[tbase@VM_0_37_centos pgbench]$ pgbench -h 127.0.0.1 -p 11008 -d postgres -U tbase -c 4 -j 1 -n -M prepared
-T 60 -r -f select_t_trgm.sql > select_t_trgm.txt 2>&1
```

```
[tbase@VM_0_37_centos pgbench]$ tail -20 select_t_trgm.txt
client 0 receiving
client 1 receiving
client 2 receiving
client 0 receiving
client 3 receiving
pgghost: 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)||%';

```

8.4.4.6、pgbench 并发写入测试

```

[tbase@VM_0_37_centos pgbench]$ cat insert_t_trgm.sql
\set id random(1, 1000000)

```

```

insert into t_trgm values(:id,:id,:id) ;
[tbase@VM_0_37_centos pgbench]$ pgbench -h 127.0.0.1 -p 11008 -d postgres -U tbase -c 4 -j 1 -n -M prepared
-T 60 -r -f insert_t_trgm.sql > insert_t_trgm.txt 2>&1
[tbase@VM_0_37_centos pgbench]$ tail -20 insert_t_trgm.txt
client 0 receiving
client 1 receiving
client 3 receiving
client 2 receiving
client 0 receiving
pghost: 127.0.0.1 pgport: 11008 nclients: 4 duration: 60 dbName: postgres
transaction type: insert_t_trgm.sql
scaling factor: 1
query mode: prepared
number of clients: 4
number of threads: 1
duration: 60 s
number of transactions actually processed: 58884
latency average = 4.077 ms
tps = 981.188175 (including connections establishing)
tps = 981.285101 (excluding connections establishing)
script statistics:
- statement latencies in milliseconds:
    0.011  \set id random(1, 1000000)
    4.063  insert into t_trgm values(:id,:id,:id) ;

```

8.4.5、gin 索引测试

8.4.5.1、创建索引消耗时间

```

postgres=# drop index t_trgm_trgm_idx;
DROP INDEX
Time: 55.954 ms
postgres=# create index t_trgm_trgm_idx on t_trgm using gin(trgm gin_trgm_ops);
CREATE INDEX
Time: 16648.549 ms (00:16.649)
postgres=#

```

8.4.5.2、索引占用空间

```

postgres=# select pg_size_pretty(pg_indexes_size('t_trgm'));
pg_size_pretty
-----
74 MB
(1 row)

```


8.4.5.3、模糊查询测试

■ 返回记录数多

```
postgres=# explain analyze select * from t_trgm where trgm ilike '%67%';
```

QUERY PLAN

```
-----
Remote Fast Query Execution  (cost=0.00..0.00 rows=0 width=0) (actual time=3.603..958.097 rows=114475
loops=1)
```

Node/s: dn001, dn002

Planning time: 0.061 ms

Execution time: 985.647 ms

(4 rows)

Time: 986.631 ms

```
postgres=# explain analyze select * from t_trgm where no_trgm ilike '%67%';
```

QUERY PLAN

```
-----
Remote Fast Query Execution  (cost=0.00..0.00 rows=0 width=0) (actual time=2.890..947.736 rows=114475
loops=1)
```

Node/s: dn001, dn002

Planning time: 0.066 ms

Execution time: 973.220 ms

(4 rows)

Time: 974.374 ms

使用 gin 索引与不使用性能不相上下

■ 返回记录比较少

```
postgres=# explain analyze select * from t_trgm where trgm ilike '%67a5%';
```

QUERY PLAN

```
-----
Remote Fast Query Execution  (cost=0.00..0.00 rows=0 width=0) (actual time=3.001..4.178 rows=481
loops=1)
```

Node/s: dn001, dn002

Planning time: 0.067 ms

Execution time: 4.300 ms

(4 rows)

Time: 5.212 ms

```
postgres=# explain analyze select * from t_trgm where no_trgm ilike '%67a5%';
```

QUERY PLAN

Remote Fast Query Execution (cost=0.00..0.00 rows=0 width=0) (actual time=174.435..524.049 rows=481 loops=1)

Node/s: dn001, dn002

Planning time: 0.069 ms

Execution time: 524.207 ms

(4 rows)

Time: 525.226 ms

过滤返回记录少，使用 gin 索引提高性能 100 倍，效果非常的好

8.4.5.4、数据导入时间测试

```
postgres=# truncate table t_trgm ;
```

```
TRUNCATE TABLE
```

```
Time: 43.750 ms
```

```
postgres=# insert into t_trgm select t,md5(t::text),md5(t::text) from generate_series(1,1000000) as t;
```

```
INSERT 0 1000000
```

```
Time: 36371.813 ms (00:36.372)
```

8.4.5.5、pgbench 并发查询测试

```
[tbase@VM_0_37_centos pgbench]$ cat select_t_trgm.sql
```

```
\set id random(1, 1000000)
```

```
select * from t_trgm where trgm ilike '%'||md5(:id)||'%';
```

```
[tbase@VM_0_37_centos pgbench]$ pgbench -h 127.0.0.1 -p 11008 -d postgres -U tbase -c 4 -j 1 -n -M prepared  
-T 60 -r -f select_t_trgm.sql > select_t_trgm.txt 2>&1
```

```
[tbase@VM_0_37_centos pgbench]$ tail -20 select_t_trgm.txt
```

```
client 2 receiving
```

```
client 1 receiving
```

```
client 3 receiving
```

```
client 0 receiving
```

```
client 2 receiving
```

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

```
transaction type: select_t_trgm.sql
```

```
scaling factor: 1
```

```
query mode: prepared
```

```
number of clients: 4
```

```
number of threads: 1
```

```
duration: 60 s
```

```
number of transactions actually processed: 13352
```

```
latency average = 17.982 ms
tps = 222.445724 (including connections establishing)
tps = 222.468993 (excluding connections establishing)
script statistics:
- statement latencies in milliseconds:
    0.012  \set id random(1, 1000000)
    17.959  select * from t_trgm where trgm ilike '%'||md5(:id)||'%';
```

```
[tbase@VM_0_37_centos pgbench]$ cat select_t_trgm.sql
\set id random(1, 1000000)
select * from t_trgm where trgm ilike '%'||substring(md5(:id) from 3 for 6)||'%';
[tbase@VM_0_37_centos pgbench]$ pgbench -h 127.0.0.1 -p 11008 -d postgres -U tbase -c 4 -j 1 -n -M prepared
-T 60 -r -f select_t_trgm.sql > select_t_trgm.txt 2>&1
[tbase@VM_0_37_centos pgbench]$ tail -20 select_t_trgm.txt
client 1 receiving
client 0 receiving
client 3 receiving
client 1 receiving
client 2 receiving
pgghost: 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)||'%';
```

8.4.5.6、pgbench 并发写入测试

```
[tbase@VM_0_37_centos pgbench]$ cat insert_t_trgm.sql
\set id random(1, 1000000)
insert into t_trgm values(:id,:id,:id) ;

[tbase@VM_0_37_centos pgbench]$ pgbench -h 127.0.0.1 -p 11008 -d postgres -U tbase -c 4 -j 1 -n -M prepared
-T 60 -r -f insert_t_trgm.sql > insert_t_trgm.txt 2>&1

[tbase@VM_0_37_centos pgbench]$ tail -20 insert_t_trgm.txt
client 0 receiving
```

```

client 1 receiving
client 0 receiving
client 2 receiving
client 3 receiving
pghost: 127.0.0.1 pgport: 11008 nclients: 4 duration: 60 dbName: postgres
transaction type: insert_t_trgm.sql
scaling factor: 1
query mode: prepared
number of clients: 4
number of threads: 1
duration: 60 s
number of transactions actually processed: 55701
latency average = 4.309 ms
tps = 928.235504 (including connections establishing)
tps = 928.323672 (excluding connections establishing)
script statistics:
- statement latencies in milliseconds:
    0.011  \set id random(1, 1000000)
    4.296  insert into t_trgm values(:id,:id,:id) ;

```

8.4.6、无索引字段测试

8.4.6.1、pgbench 并发查询测试

```

[tbase@VM_0_37_centos pgbench]$ cat select_t_trgm.sql
\set id random(1, 1000000)
select * from t_trgm where no_trgm ilike '%'||substring(md5(:id) from 3 for 6)||'%';

[tbase@VM_0_37_centos pgbench]$ pgbench -h 127.0.0.1 -p 11008 -d postgres -U tbase -c 4 -j 1 -n -M prepared
-T 60 -r -f select_t_trgm.sql > select_t_trgm.txt 2>&1

[tbase@VM_0_37_centos pgbench]$ tail -20 select_t_trgm.txt
client 3 receiving
client 2 receiving
client 1 receiving
client 3 receiving
client 0 receiving
pghost: 127.0.0.1 pgport: 11008 nclients: 4 duration: 60 dbName: postgres
transaction type: select_t_trgm.sql
scaling factor: 1
query mode: prepared
number of clients: 4
number of threads: 1
duration: 60 s
number of transactions actually processed: 212

```

latency average = 1147.939 ms

tps = 3.484507 (including connections establishing)

tps = 3.484812 (excluding connections establishing)

script statistics:

- statement latencies in milliseconds:

```
0.016  \set id random(1, 1000000)
1144.441 select * from t_trgm where no_trgm ilike '%'||substring(md5(:id) from 3 for 6)||'%';
```

```
[tbase@VM_0_37_centos pgbench]$ cat select_t_trgm.sql
```

```
\set id random(1, 1000000)
```

```
select * from t_trgm where no_trgm ilike '%'||md5(:id)||'%';
```

```
[tbase@VM_0_37_centos pgbench]$ pgbench -h 127.0.0.1 -p 11008 -d postgres -U tbase -c 4 -j 1 -n -M prepared
```

```
-T 60 -r -f select_t_trgm.sql > select_t_trgm.txt 2>&1
```

```
[tbase@VM_0_37_centos pgbench]$ tail -20 select_t_trgm.txt
```

client 3 receiving

client 1 receiving

client 0 receiving

client 2 receiving

client 3 receiving

pgghost: 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)||'%';
```

8.4.6.1、pgbench 并发写入测试

```
[tbase@VM_0_37_centos pgbench]$ cat insert_t_trgm.sql
```

```
\set id random(1, 1000000)
```

```
insert into t_trgm values(:id,:id,:id) ;
```

```
[tbase@VM_0_37_centos pgbench]$ pgbench -h 127.0.0.1 -p 11008 -d postgres -U tbase -c 4 -j 1 -n -M prepared
```

```
-T 60 -r -f insert_t_trgm.sql > insert_t_trgm.txt 2>&1
```

```
[tbase@VM_0_37_centos pgbench]$ tail -20 insert_t_trgm.txt
```

client 0 receiving

client 2 receiving

client 3 receiving

client 0 receiving
 client 1 receiving
 pgghost: 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);

8.4.7、数据对比总结及例外

8.4.7.1、数据对比

项目	GIST	GIN	无索引
创建索引消耗时间	24.9s	16.6s	
索引占用空间	178M	74M	
数据导入时间测试	29s	36s	4.2s
极短字符串查询	1197.320 ms	986.631 ms	989.840 ms
中字符串查询	118.831 ms	4.300 ms	502.756 ms
pgbench 长字符串查询，返回单一记录	235tps	222tps	2.8tps
pgbench 短字符串查询，返回多条记录	39tps	954tps	3.4tps
pgbench 并发写入测试	981tps	928tps	961tpc

一般情况下使用 gin 索引即可

8.4.7.2、pg_trgm 使用限制

- pg_trgm 不支持小于 3 个字的匹配条件

pg_trgm 不支持小于 3 个字的匹配条件，pg_trgm 的工作原理是把字符串切成 N 个 3 元组，然后对这些 3 元组做匹配，所以如果作为查询条件的字符串小于 3 个字符它就罢工了

■ 数据库的 LC_CTYPE 需要设置为中文区域

可能会发现 pg_trgm 不支持中文，中文字符都被截掉了

show_trgm()值返回空

```
postgres=# select show_trgm('腾讯数据库');
               show_trgm
```

```
{}
(1 row)
```

--正确返回如下

```
postgres=# select show_trgm('腾讯数据库');
               show_trgm
-----
{0xa10449,0x077ed5,0x161aeb,0x2d07b4,0x41fde2,0x61dd3b}
(1 row)
```

这是因为，pg_trgm 调用了系统的 isalpha()函数判断字符，而 isalpha()依赖于 LC_CTYPE，如果数据库的 LC_CTYPE 是 C，isalpha()就不能识别中文。所以需要把数据库的 LC_CTYPE 设成 zh_CN

9、应用程序样例

9.1、java

9.1.1、创建数据表

```
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.Statement;

public class createtable {
    public static void main( String args[] )
    {
        Connection c = null;
```

```

Statement stmt = null;
try {
    Class.forName("org.postgresql.Driver");
    c =
DriverManager.getConnection("jdbc:postgresql://127.0.0.1:15432/postgres?currentS
chema=public&binaryTransfer=false","tbase", "tbase");
    System.out.println("Opened database successfully");
    stmt = c.createStatement();
    String sql = "create table tbase(id int,nickname text) distribute by shard(id)
to group default_group" ;
    stmt.executeUpdate(sql);
    stmt.close();
    c.close();
} catch ( Exception e ) {
    System.err.println( e.getClass().getName()+" : "+ e.getMessage() );
    System.exit(0);
}
System.out.println("Table created successfully");
}
}

```

9.1.2、插入数据

```

import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.Statement;

public class insert {
    public static void main(String args[]) {
        Connection c = null;
        Statement stmt = null;
        try {
            Class.forName("org.postgresql.Driver");
            c =
DriverManager.getConnection("jdbc:postgresql://127.0.0.1:15432/postgres?currentS
chema=public&binaryTransfer=false","tbase", "tbase");
            c.setAutoCommit(false);
            System.out.println("Opened database successfully");

            stmt = c.createStatement();
            String sql = "INSERT INTO tbase (id,nickname) "
                + "VALUES (1,'tbase');";
            stmt.executeUpdate(sql);

            sql = "INSERT INTO tbase (id,nickname) "

```



```

        + "VALUES (2, 'pgxz' ), (3, 'pgxc')";
    stmt.executeUpdate(sql);
    stmt.close();
    c.commit();
    c.close();
} catch (Exception e) {
    System.err.println( e.getClass().getName()+" : "+ e.getMessage() );
    System.exit(0);
}
System.out.println("Records created successfully");
}
}

```

9.1.4、扩展协议插入数据

```

import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.*;
import java.util.Random;

public class insert_prepared {
    public static void main(String args[]) {
        Connection c = null;
        PreparedStatement stmt;
        try {
            Class.forName("org.postgresql.Driver");
            c =
DriverManager.getConnection("jdbc:postgresql://127.0.0.1:15432/postgres?currentS
chema=public&binaryTransfer=false","tbase", "tbase");
            c.setAutoCommit(false);
            System.out.println("Opened database successfully");
            //插入数据
            String sql = "INSERT INTO tbase (id,nickname) VALUES (?,?)";
            stmt = c.prepareStatement(sql);
            stmt.setInt(1, 9999);
            stmt.setString(2, "tbase_prepared");
            stmt.executeUpdate();

            //插入更新
            sql = "INSERT INTO tbase (id,nickname) VALUES (?,?) ON CONFLICT(id) DO UPDATE
SET nickname=?";
            stmt = c.prepareStatement(sql);
            stmt.setInt(1, 9999);
            stmt.setString(2, "tbase_prepared");
            stmt.setString(3, "tbase_prepared_update");

```

```

stmt.executeUpdate();

stmt.close();
c.commit();
c.close();
} catch (Exception e) {
    System.err.println( e.getClass().getName()+" : "+ e.getMessage() );
    System.exit(0);
}
System.out.println("Records created successfully");
}
}

```

9.1.3、copy from 入库

```

import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.Statement;
import org.postgresql.copy.CopyManager;
import org.postgresql.core.BaseConnection;
import java.io.*;

public class copyfrom {
    public static void main( String args[] )
    {
        Connection c = null;
        Statement stmt = null;
        FileInputStream fs = null;
        try {
            Class.forName("org.postgresql.Driver");
            c =
DriverManager.getConnection("jdbc:postgresql://127.0.0.1:15432/postgres?currentS
chema=public&binaryTransfer=false","tbase", "tbase");
            System.out.println("Opened database successfully");
            CopyManager cm = new CopyManager((BaseConnection) c);
            fs = new FileInputStream("/data/tbase/tbase.csv");
            String sql = "COPY tbase FROM STDIN DELIMITER AS ','";
            cm.copyIn(sql, fs);
            c.close();
            fs.close();
        } catch ( Exception e ) {
            System.err.println( e.getClass().getName()+" : "+ e.getMessage() );
            System.exit(0);
        }
        System.out.println("Copy data successfully");
    }
}

```

```

    }
}

```

9.1.4、copy to 出库

```

import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.Statement;
import org.postgresql.copy.CopyManager;
import org.postgresql.core.BaseConnection;
import java.io.*;

public class copyto {
    public static void main( String args[] )
    {
        Connection c = null;
        Statement stmt = null;
        FileOutputStream fs = null;
        try {
            Class.forName("org.postgresql.Driver");
            c =
DriverManager.getConnection("jdbc:postgresql://127.0.0.1:15432/postgres?currentS
chema=public&binaryTransfer=false","tbase", "tbase");
            System.out.println("Opened database successfully");
            CopyManager cm = new CopyManager((BaseConnection) c);
            fs = new FileOutputStream("/data/tbase/tbase.csv");
            String sql = "COPY tbase TO STDOUT DELIMITER AS ','";
            cm.copyOut(sql, fs);
            c.close();
            fs.close();
        } catch ( Exception e ) {
            System.err.println( e.getClass().getName()+" : "+ e.getMessage() );
            System.exit(0);
        }
        System.out.println("Copy data successfully");
    }
}

```

9.1.5、jdbc 下载

<https://jdbc.postgresql.org/download.html>

9.2、C 程序

9.2.1、连接服务

```
#include <stdio.h>
#include <stdlib.h>
#include "libpq-fe.h"
int
main(int argc, char **argv){
    const char *conninfo;
    PGconn      *conn;
    if (argc > 1){
        conninfo = argv[1];
    }else{
        conninfo = "dbname = postgres";
    }
    conn = PQconnectdb(conninfo);
    if (PQstatus(conn) != CONNECTION_OK){
        fprintf(stderr, "连接数据库失败: %s", PQerrorMessage(conn));
    }else{
        printf("连接数据库成功! \n");
    }
    PQfinish(conn);
    return 0;
}
```

编译

```
gcc -c -I /usr/local/install/tbase_pgxz/include/ conn.c
gcc -o conn conn.o -L /usr/local/install/tbase_pgxz/lib/ -lpq
```

运行

```
./conn "host=172.16.0.3 dbname=postgres port=11000"
连接数据库成功!
./conn "host=172.16.0.3 dbname=postgres port=15432 user=tbase"
连接数据库成功!
```

9.2.2、建立数据表

```
#include <stdio.h>
#include <stdlib.h>
#include "libpq-fe.h"
int
```

```

main(int argc, char **argv){
    const char *conninfo;
    PGconn      *conn;
    PGresult     *res;
    const char *sql = "create table tbase(id int,nickname text) distribute by shard(id) to
group default_group";
    if (argc > 1){
        conninfo = argv[1];
    }else{
        conninfo = "dbname = postgres";
    }
    conn = PQconnectdb(conninfo);
    if (PQstatus(conn) != CONNECTION_OK){
        fprintf(stderr, "连接数据库失败: %s", PQerrorMessage(conn));
    }else{
        printf("连接数据库成功! \n");
    }
    res = PQexec(conn,sql);
    if(PQresultStatus(res) != PGRES_COMMAND_OK){
        fprintf(stderr, "建立数据表失败: %s", PQresultErrorMessage(res));
    }else{
        printf("建立数据表成功! \n");
    }
    PQclear(res);
    PQfinish(conn);
    return 0;
}

```

编译

```

gcc -c -I /usr/local/install/tbase_pgxz/include/ createtable.c
gcc -o createtable createtable.o -L /usr/local/install/tbase_pgxz/lib/ -lpq

```

运行

```

./createtable "port=11000 dbname=postgres"
连接数据库成功!
建立数据表成功!

```

9.2.3、插入数据

```

#include <stdio.h>
#include <stdlib.h>
#include "libpq-fe.h"
int

```

```

main(int argc, char **argv){
    const char *conninfo;
    PGconn      *conn;
    PGresult     *res;
    const char *sql = "INSERT INTO tbase (id,nickname) values(1,'tbase'),(2,'pgxz')";
    if (argc > 1){
        conninfo = argv[1];
    }else{
        conninfo = "dbname = postgres";
    }
    conn = PQconnectdb(conninfo);
    if (PQstatus(conn) != CONNECTION_OK){
        fprintf(stderr, "连接数据库失败: %s", PQerrorMessage(conn));
    }else{
        printf("连接数据库成功! \n");
    }
    res = PQexec(conn,sql);
    if(PQresultStatus(res) != PGRES_COMMAND_OK){
        fprintf(stderr, "插入数据失败: %s", PQresultErrorMessage(res));
    }else{
        printf("插入数据成功! \n");
    }
    PQclear(res);
    PQfinish(conn);
    return 0;
}

```

编译

```

gcc -c -I /usr/local/install/tbase_pgxz/include/ insert.c
gcc -o insert insert.o -L /usr/local/install/tbase_pgxz/lib/ -lpq

```

运行

```

./insert "dbname=postgres port=15432"
连接数据库成功!
插入数据成功!

```

9.2.4、查询数据

```

#include <stdio.h>
#include <stdlib.h>
#include "libpq-fe.h"
int
main(int argc, char **argv){
    const char *conninfo;

```

```

PGconn      *conn;
PGresult     *res;
const char *sql = "select * from tbase";
if (argc > 1){
    conninfo = argv[1];
}else{
    conninfo = "dbname = postgres";
}
conn = PQconnectdb(conninfo);
if (PQstatus(conn) != CONNECTION_OK){
    fprintf(stderr, "连接数据库失败: %s", PQerrorMessage(conn));
}else{
    printf("连接数据库成功! \n");
}
res = PQexec(conn, sql);
if(PQresultStatus(res) != PGRES_TUPLES_OK){
    fprintf(stderr, "插入数据失败: %s", PQresultErrorMessage(res));
}else{
    printf("查询数据成功! \n");
    int rownum = PQntuples(res) ;
    int colnum = PQnfields(res);
    for(int j = 0; j< colnum; ++j){
        printf("%s\t", PQfname(res, j));
    }
    printf("\n");
    for(int i = 0; i< rownum; ++i){
        for(int j = 0; j< colnum; ++j){
            printf("%s\t", PQgetvalue(res, i, j));
        }
        printf("\n");
    }
}
PQclear(res);
PQfinish(conn);
return 0;
}

```

编译

```

gcc -std=c99 -c -I /usr/local/install/tbase_pgxz/include/ select.c
gcc -o select select.o -L /usr/local/install/tbase_pgxz/lib/ -lpq

```

运行

```

./select "dbname=postgres port=15432"
连接数据库成功!

```

查询数据成功！

```
id      nickname
1       tbase
2       pgxz
```

9.2.5、copy 入库

```
#include <string.h>
#include <stdio.h>
#include <stdlib.h>
#include "libpq-fe.h"
int
main(int argc, char **argv){
    const char *conninfo;
    PGconn      *conn;
    PGresult     *res;
    const char *buffer = "1,tbase\n2,pgxz\n3,Tbase 牛";
    if (argc > 1){
        conninfo = argv[1];
    }else{
        conninfo = "dbname = postgres";
    }
    conn = PQconnectdb(conninfo);
    if (PQstatus(conn) != CONNECTION_OK){
        fprintf(stderr, "连接数据库失败: %s", PQerrorMessage(conn));
    }else{
        printf("连接数据库成功! \n");
    }
    res=PQexec(conn,"COPY tbase FROM STDIN DELIMITER ',';");
    if(PQresultStatus(res) != PGRES_COPY_IN){
        fprintf(stderr, "copy 数据出错 1: %s", PQresultErrorMessage(res));
    }else{
        int len = strlen(buffer);
        if(PQputCopyData(conn,buffer,len) == 1){
            if(PQputCopyEnd(conn,NULL) == 1){
                res = PQgetResult(conn);
                if(PQresultStatus(res) == PGRES_COMMAND_OK){
                    printf("copy 数据成功! \n");
                }else{
                    fprintf(stderr, "copy 数据出错 2: %s", PQerrorMessage(conn));
                }
            }else{
                fprintf(stderr, "copy 数据出错 3: %s", PQerrorMessage(conn));
            }
        }else{

```



```

        fprintf(stderr, "copy 数据出错 4: %s", PQerrorMessage(conn));
    }
}
PQclear(res);
PQfinish(conn);
return 0;
}

```

编译

```

gcc -c -I /usr/local/install/tbase_pgxz/include/ copy.c
gcc -o copy copy.o -L /usr/local/install/tbase_pgxz/lib/ -lpq

```

执行

```
./copy "dbname=postgres port=15432"
```

连接数据库成功!

copy 数据成功!

9.3、shell 程序

```
#!/bin/sh
```

```

if [ $# -ne 0 ]
then
    echo "usage: $0 exec_sql"
    exit 1
fi

```

```
exec_sql=$1
```

```

masters=`psql -h 172.16.0.29 -d postgres -p 15432 -t -c "select
string_agg(node_host, ' ') from (select * from pgxc_node where node_type = 'D'
order by node_name) t"`
port_list=`psql -h 172.16.0.29 -d postgres -p 15432 -t -c "select
string_agg(node_port::text, ' ') from (select * from pgxc_node where node_type
= 'D' order by node_name) t"`
node_cnt=`psql -h 172.16.0.29 -d postgres -p 15432 -t -c "select count(*) from
pgxc_node where node_type = 'D' "`
masters=($masters)
ports=($port_list)

echo $node_cnt

flag=0

```

```

for ((i=0;i<$node_cnt;i++));
do
    seq=$((i+1))
    master=${masters[$i]}
    port=${ports[$i]}
    echo $master
    echo $port

    psql -h $master -p $port postgres -c "$exec_sql"
done

```

9.4、python 程序

9.4.1、安装 psycopg2 模块

```
[root@VM_0_29_centos ~]# yum install python-psycopg2
```

9.4.2、连接服务

```

#coding=utf-8
#!/usr/bin/python
import psycopg2
try:
    conn = psycopg2.connect(database="postgres", user="tbase", password="",
host="172.16.0.29", port="15432")
    print "连接数据库成功"
    conn.close()
except psycopg2.Error,msg:
    print "连接数据库出错, 错误详细信息:  %s" %(msg.args[0])

```

运行

```
[tbase@VM_0_29_centos python]$ python conn.py
连接数据库成功
```

9.4.3、创建数据表

```

#coding=utf-8
#!/usr/bin/python
import psycopg2
try:
    conn = psycopg2.connect(database="postgres", user="tbase", password="", host="172.16.0.29", port="15432")

```

```

print "连接数据库成功"
cur = conn.cursor()
sql = """
    create table tbase
    (
        id int,
        nickname varchar(100)
    ) distribute by shard(id) to group default_group
    """
cur.execute(sql)
conn.commit()
print "建立数据表成功"
conn.close()
except psycopg2.Error,msg:
    print "TBase Error %s" %(msg.args[0])

```

运行

```

[tbase@VM_0_29_centos python]$ python createtable.py
连接数据库成功
建立数据表成功

```

9.4.4、新增数据

```

#coding=utf-8
#!/usr/bin/python
import psycopg2
try:
    conn = psycopg2.connect(database="postgres", user="tbase", password="",
host="172.16.0.29", port="15432")
    print "连接数据库成功"
    cur = conn.cursor()
    sql = "insert into tbase values(1,'tbase'),(2,'tbase');"
    cur.execute(sql)
    sql = "insert into tbase values(%s,%s)"
    cur.execute(sql,(3,'pg'))
    conn.commit()
    print "插入数据成功"
    conn.close()
except psycopg2.Error,msg:
    print "操作数据库出库 %s" %(msg.args[0])

```

运行

```

[tbase@VM_0_29_centos python]$ python insert.py

```

连接数据库成功

插入数据成功

9.4.5、查询数据

```
#coding=utf-8
#!/usr/bin/python
import psycopg2
try:
    conn = psycopg2.connect(database="postgres", user="tbase", password="",
host="172.16.0.29", port="15432")
    print "连接数据库成功"
    cur = conn.cursor()
    sql = "select * from tbase"
    cur.execute(sql)
    rows = cur.fetchall()
    for row in rows:
        print "ID = ", row[0]
        print "NICKNAME = ", row[1], "\n"
    conn.close()
except psycopg2.Error,msg:
    print "操作数据库出库 %s" %(msg.args[0])
```

运行

```
[tbase@VM_0_29_centos python]$ python select.py
```

连接数据库成功

ID = 1

NICKNAME = tbase

ID = 2

NICKNAME = pgxz

ID = 3

NICKNAME = pg

9.4.6、copy from 方法

```
#coding=utf-8
#!/usr/bin/python
import psycopg2
try:
    conn = psycopg2.connect(database="postgres", user="tbase", password="",
host="172.16.0.29", port="15432")
    print "连接数据库成功"
```

```

cur = conn.cursor()
filename = "/data/tbase/tbase.txt"
cols = ('id','nickname')
tablename="public.tbase"
cur.copy_from(file=open(filename),table=tablename,columns=cols,sep=',')
conn.commit()
print "导入数据成功"
conn.close()
except psycopg2.Error,msg:
    print "操作数据库出库 %s" %(msg.args[0])

```

执行

```
[tbase@VM_0_29_centos python]$ python copy_from.py
```

连接数据库成功

导入数据成功

9.5、php 程序

9.5.1、连接服务

```

<?php
$host="172.16.0.29";
$port="15432";
$dbname="postgres";
$user="tbase" ;
$password="";

//连接数据库
$conn=@pg_connect("host=$host port=$port dbname=$dbname user=$user
password=$password");
if (!$conn) {
    $error_msg=@pg_errormessage($conn);
    echo "连接数据库出错, 详情: ".$error_msg."\n<BR>"; ;
    exit;
}else{
    echo "连接数据库成功"."<BR>";
}
//关闭连接
pg_close($conn);
?>

```

执行

```
[root@VM_0_47_centos test]# curl http://127.0.0.1:8080/dbsta/test/conn.php
```

连接数据库成功

9.5.2、创建数据表

```
<?php
$host="172.16.0.29";
$port="15432";
$dbname="postgres";
$user="tbase" ;
$password="";

//连接数据库
$conn=@pg_connect("host=$host port=$port dbname=$dbname user=$user
password=$password");
if (!$conn){
    $error_msg=@pg_errormessage($conn);
    echo "连接数据库出错, 详情: ".$error_msg."\n"; ;
    exit;
}else{
    echo "连接数据库成功"."\\n";
}

//建立数据表
$sql="create table public.tbase(id integer,nickname varchar(100)) distribute by
shard(id) to group default_group;";
$result = @pg_exec($conn,$sql) ;
if (!$result){
    $error_msg=@pg_errormessage($conn);
    echo "创建数据表出错, 详情: ".$error_msg."\n"; ;
    exit;
}else{
    echo "创建数据表成功"."\\n";
}

//关闭连接
pg_close($conn);
?>
```

执行

```
[root@VM_0_47_centos test]# curl http://127.0.0.1:8080/dbsta/test/createtable.php
```

连接数据库成功
创建数据表成功

9.5.3、插入数据

```
<?php
$host="172.16.0.29";
$port="15432";
$dbname="postgres";
$user="tbase" ;
$password="";

//连接数据库
$conn=@pg_connect("host=$host port=$port dbname=$dbname user=$user
password=$password");
if (!$conn){
    $error_msg=@pg_errormessage($conn);
    echo "连接数据库出错, 详情: ".$error_msg."\n"; ;
    exit;
}else{
    echo "连接数据库成功". "\n";
}

//插入数据
$sql="insert into public.tbases values(1,'tbase'),(2,'pgxz');"
$result = @pg_exec($conn,$sql) ;
if (!$result){
    $error_msg=@pg_errormessage($conn);
    echo "插入数据出错, 详情: ".$error_msg."\n";
    exit;
}else{
    echo "插入数据成功". "\n";
}

//关闭连接
pg_close($conn);

?>
```

执行

```
[tbase@VM_0_47_centos test]$ curl http://127.0.0.1:8080/dbsta/test/insert.php
连接数据库成功
插入数据成功
```

9.5.4、查询记录

```
<?php
```

```

$host="172.16.0.29";
$port="15432";
$dbname="postgres";
$user="tbase" ;
$password="";

//连接数据库
$conn=@pg_connect("host=$host port=$port dbname=$dbname user=$user
password=$password");
if (! $conn) {
    $error_msg=@pg_errormessage($conn);
    echo "连接数据库出错, 详情: ".$error_msg."\n"; ;
    exit;
} else {
    echo "连接数据库成功". "\n";
}

//查询数据
$sql="select id,nickname from public.tbases";
$result = @pg_exec($conn,$sql) ;
if (! $result) {
    $error_msg=@pg_errormessage($conn);
    echo "查询数据出错, 详情: ".$error_msg."\n";
    exit;
} else {
    echo "插入数据成功". "\n";
}
$record_num = pg_numrows($result);
echo "返回记录数".$record_num."\n";
$rec=pg_fetch_all($result);
for($i=0;$i<$record_num;$i++) {
    echo "记录数#".strval($i+1)."\n";
    echo "id: ".$rec[$i]["id"]."\n";
    echo "nickname: ".$rec[$i]["nickname"]."\n\n";
}
//关闭连接
pg_close($conn);
?>

```

调用方法

```
[root@VM_0_47_centos ~]# curl http://127.0.0.1:8080/dbsta/test/select.php
```

连接数据库成功

插入数据成功

返回记录数 2

记录数#1


```
id: 1
nickname: tbase
```

记录数#2

```
id: 2
nickname: pgxz
```

9.5.5、copy from 方法

把一个 php 数组导入到数据表中

```
<?php

$host="172.16.0.29";
$port="15432";
$dbname="postgres";
$user="tbase" ;
$password="";

//连接数据库
$conn=@pg_connect("host=$host port=$port dbname=$dbname user=$user
password=$password");
if (!$conn){
    $error_msg=@pg_errormessage($conn);
    echo "连接数据库出错, 详情: ".$error_msg."\n"; ;
    exit;
}else{
    echo "连接数据库成功"."\n";
}
$row=ARRAY("1,TBase","2,pgxz");
$flag=pg_copy_from($conn,"public.tbase",$row,"");

if (!$flag){
    $error_msg=@pg_errormessage($conn);
    echo "copy 出错, 详情: ".$error_msg."\n";
}else{
    echo "copy 成功"."\n";
}

//关闭连接
pg_close($conn);

?>
```

调用方法

```
curl http://127.0.0.1/dbsta/cron/php_copy_from.php
```

连接数据库成功

copy 成功

9.5.6、copy to 方法

将一个表的记录复制到一个 php 数据中

```
<?php

$host="172.16.0.29";
$port="15432";
$dbname="postgres";
$user="tbase" ;
$password="";

//连接数据库
$conn=@pg_connect ("host=$host port=$port dbname=$dbname user=$user
password=$password");
if (!$conn) {
    $error_msg=@pg_errormessage($conn);
    echo "连接数据库出错, 详情: ".$error_msg."\n"; ;
    exit;
}else{
    echo "连接数据库成功"."\n";
}

$row=pg_copy_to($conn,"public.tbase","");
if (!$row) {
    $error_msg=@pg_errormessage($conn);
    echo "copy 出错, 详情: ".$error_msg."\n";
}else{
    print_r($row);
}

//关闭连接
pg_close($conn);
?>
```

调用方法

```
curl http://127.0.0.1/dbsta/cron/php_copy_to.php
```

连接数据库成功

Array

(

```
[0] => 1,TBase
```

```
[1] => 2,pgxz
```

```
)
```

9.5.7、入库去重方法

```
<?php
error_reporting(E_ALL && ~E_NOTICE);
ini_set('display_errors', '1');
set_time_limit(0);

$host="172.16.0.29";
$port="15432";
$dbname="postgres";
$user="tbase" ;
$password="";

/*
CREATE TABLE mydata (
    mpid text NOT NULL,
    datatype text NOT NULL,
    datatime timestamp without time zone NOT NULL,
    datavalue text,
    inputtime timestamp without time zone
)
DISTRIBUTE BY SHARD (mpid);

CREATE unique INDEX mydata_uidx ON mydata USING btree (mpid, datatype, datatime);
*/

//连接数据库
$conn=@pg_connect("host=$host port=$port dbname=$dbname user=$user
password=$password");
if (!$conn) {
    $error_msg=@pg_errormessage($conn);
    echo "连接数据库出错, 详情: ".$error_msg."\n"; ;
    exit;
}else{
    echo "连接数据库成功"."\\n";
}

//建立临时表
```

```

$tmp_table_name="mydata_tmp_".strval(rand(100000000, 999999999));
$k=0;
do {
    $sql="
CREATE TABLE ".$tmp_table_name." (
    mpid text NOT NULL,
    datatype text NOT NULL,
    datatime timestamp without time zone NOT NULL,
    datavalue text,
    inputtime timestamp without time zone
)
WITH OIDS DISTRIBUTE BY SHARD (mpid);
";
    $result = @pg_exec($conn,$sql) ;
    if ($result){
        //建议临时数据表成功,退出
        $sql="CREATE INDEX ".$tmp_table_name."_idx ON ".$tmp_table_name." USING
btree (mpid, datatype, datatime); ";
        $result = @pg_exec($conn,$sql) ;
        if (!$result){
            $error_msg=@pg_errormessage($conn);
            echo "建立临时表索引失败, 详情: ".$error_msg."\n<BR>";
            exit;
        }
        break;
    }else{
        $k++;
        if($k>10){
            $error_msg=@pg_errormessage($conn);
            echo "多次建立数据表失败, 详情: ".$error_msg."\n<BR>";
            exit;
        }
    }
}while (true);
$sql="";

$mydata_sql="
INSERT INTO mydata (mpid, datatype,datatime,datavalue,inputtime) VALUES
";
$mydata_tmp_sql="
INSERT INTO ".$tmp_table_name." (mpid, datatype,datatime,datavalue,inputtime)
VALUES
";

for ($i=0;$i<100;$i++){
    $insertnum = 250;

```

```

for ($j=0;$j<$insertnum;$j++){
    $sql=$sql."
    ('".strval(rand(100000000, 999999999))."', '10129f14', '2018-03-15
02:00:00', '004390.44', '2018-03-15 11:05:55')
    ";
    if (($j+1) != $insertnum) {
        $sql=$sql.", ";
    }
}
$execsql=$mydata_sql.$sql;
$result = @pg_exec($conn,$execsql) ;

if (!$result){
    ECHO "执行失败\n";
    //将数据导入到临时表
    $execsql=$mydata_tmp_sql.$sql;
    $result = @pg_exec($conn,$execsql) ;
    if (!$result){
        $error_msg=@pg_errormessage($conn);
        echo "数据插入临时表失败, 详情: ".$error_msg."\n<BR>";
        exit;
    }
    //删除临时表中重复的数据, 这一步最好在应用程序中去重, 减少数据的负担
    $execsql="DELETE FROM ".$tmp_table_name." WHERE oid NOT IN (select min(oid)
from ".$tmp_table_name." group by mpid, datatype, datetime)";
    $result = @pg_exec($conn,$execsql) ;
    if (!$result){
        $error_msg=@pg_errormessage($conn);
        echo "删除临时表重复数据失败, 详情: ".$error_msg."\n";
        exit;
    }
    $k=0;
    do {
        //删除重复数据
        $execsql="DELETE FROM ".$tmp_table_name." USING mydata WHERE
mydata.mpid=".$tmp_table_name.".mpid AND
mydata.datatype=".$tmp_table_name.".datatype AND
mydata.datetime=".$tmp_table_name.".datetime" ;
        //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); ;

?>

```

9.6、golang 程序

9.6.1、连接服务

```

package main

import (
    "fmt"

```

```

"time"

"github.com/jackc/pgx"
)

func main() {
    var error_msg string

    //连接数据库
    conn, err := db_connect()
    if err != nil {
        error_msg = "连接数据库失败, 详情: " + err.Error()
        write_log("Error", error_msg)
        return
    }
    //程序运行结束时关闭连接
    defer conn.Close()
    write_log("Log", "连接数据库成功")
}

/*
功能描述: 写入日志处理

参数说明:
log_level -- 日志级别, 只能是 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

9.6.2、建立数据表

```
package main
```

```

import (
    "fmt"
    "time"

    "github.com/jackc/pgx"
)

```

```

func main() {
    var error_msg string
    var sql string

    //连接数据库
    conn, err := db_connect()
    if err != nil {
        error_msg = "连接数据库失败, 详情: " + err.Error()
    }
}

```



```

    write_log("Error", error_msg)
    return
}
//程序运行结束时关闭连接
defer conn.Close()
write_log("Log", "连接数据库成功")

//建立数据表
sql = "create table public.tbbase(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 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
```

9.6.3、插入数据

```
package main
```

```
import (
    "fmt"
    "strings"
    "time"

    "github.com/jackc/pgx"
)
```

```
func main() {
    var error_msg string
    var sql string
    var nickname string

    //连接数据库
    conn, err := db_connect()
    if err != nil {
        error_msg = "连接数据库失败, 详情: " + err.Error()
        write_log("Error", error_msg)
        return
    }
}
```

```

//程序运行结束时关闭连接
defer conn.Close()
write_log("Log", "连接数据库成功")

//插入数据
sql = "insert into public.tbbase values('1','tbbase'),('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.tbbase 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.tbbase 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 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
```

9.6.4、查询数据

```
package main

import (
    "fmt"
    "strings"
    "time"

    "github.com/jackc/pgx"
)

func main() {
    var error_msg string
    var sql string

    //连接数据库
    conn, err := db_connect()
    if err != nil {
        error_msg = "连接数据库失败, 详情: " + err.Error()
        write_log("Error", error_msg)
        return
    }
    //程序运行结束时关闭连接
    defer conn.Close()
    write_log("Log", "连接数据库成功")

    sql = "SELECT id,nickname FROM public.tbase LIMIT 2"
    rows, err := conn.Query(sql)
    if err != nil {
```

```

    error_msg = "查询数据失败,详情: " + err.Error()
    write_log("Error", error_msg)
    return
} else {
    write_log("Log", "查询数据成功")
}

var nickname string
var id string

for rows.Next() {
    err = rows.Scan(&id, &nickname)
    if err != nil {
        error_msg = "执行查询失败,详情: " + err.Error()
        write_log("Error", error_msg)
        return
    }
    error_msg = fmt.Sprintf("id: %s nickname: %s", id, nickname)
    write_log("Log", error_msg)
}
rows.Close()

nickname = "tbase"

sql = "SELECT id,nickname FROM public.tbase WHERE nickname =' " +
sql_data_encode(nickname) + "' "
rows, err = conn.Query(sql)
if err != nil {
    error_msg = "查询数据失败,详情: " + err.Error()
    write_log("Error", error_msg)
    return
} else {
    write_log("Log", "查询数据成功")
}
defer rows.Close()

for rows.Next() {
    err = rows.Scan(&id, &nickname)
    if err != nil {
        error_msg = "执行查询失败,详情: " + err.Error()
        write_log("Error", error_msg)
        return
    }
    error_msg = fmt.Sprintf("id: %s nickname: %s", id, nickname)
    write_log("Log", error_msg)
}

```

```
}
```

```
/*
```

功能描述: sql 查询拼接字符串编码

参数说明:

str -- 要编码的字符串

返回值说明:

返回编码过的字符串

```
*/
```

```
func sql_data_encode(str string) string {
    return strings.Replace(str, "'", "''", -1)
}
```

```
/*
```

功能描述: 写入日志处理

参数说明:

log_level -- 日志级别, 只能是 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

9.6.5、copy from 方法

```
package main
```

```

import (
    "fmt"
    "math/rand"
    "time"

    "github.com/jackc/pgx"
)

```

```

func main() {
    var error_msg string

```



```

//连接数据库
conn, err := db_connect()
if err != nil {
    error_msg = "连接数据库失败, 详情: " + err.Error()
    write_log("Error", error_msg)
    return
}
//程序运行结束时关闭连接
defer conn.Close()
write_log("Log", "连接数据库成功")

//构造 5000 行数据
inputRows := [][]interface{}{}
var id string
var nickname string
for i := 0; i < 5000; i++ {
    id = fmt.Sprintf("%d", rand.Intn(10000))
    nickname = fmt.Sprintf("%d", rand.Intn(10000))
    inputRows = append(inputRows, []interface{}{id, nickname})
}
copyCount, err := conn.CopyFrom(pgx.Identifier{"tbase"}, []string{"id",
"nickname"}, pgx.CopyFromRows(inputRows))
if err != nil {
    error_msg = "执行 copyFrom 失败, 详情: " + err.Error()
    write_log("Error", error_msg)
    return
}
if copyCount != len(inputRows) {
    error_msg = fmt.Sprintf("执行 copyFrom 失败, copy 行数: %d 返回行数为: %d",
len(inputRows), copyCount)
    write_log("Error", error_msg)
    return
} else {
    error_msg = "Copy 记录成功"
    write_log("Log", error_msg)
}
}

```

/*

功能描述: 写入日志处理

参数说明:

log_level -- 日志级别, 只能是 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

9.6.6、go 相关资源包

需要 git 的资源包

<https://github.com/jackc/pgx>

<https://github.com/pkg/errors>

