TDSQL PostgreSQL版

开发指南

腾讯云

文档目录

开发指南
数据库基础开发
数据类型

最近更新时间：2022-09-02 15:35:32

数字类型

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

字符类型

名字	描述
`character varying(*n*), varchar(*n*)`	有限制的变长
`character(*n*), char(*n*)`	定长，空格填充

text	1GB

二进制数据类型

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

日期类型

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

布尔类型

名字	存储字节	描述
boolean	1字节	状态为真或假

更多数据类型介绍

请参见 数据类型说明 。

异构数据库类型对照表

最近更新时间：2023-07-20 10:49:01

与 Oracle 对照表（Oracle 兼容版特性）

Oracle	TDSQL PostgreSQL版（Oracle 兼容版）
Number	可以对应 TDSQL PostgreSQL版（Oracle 兼容版）的 smallint, integer, bigint, numeric(p,s) 等多种数据类型。由于 smallint，Integer，bigint 的算术运算效率比 numeric 高得多，所以要视业务需要转换成对应的 smallint, integer, bigint，无法转换时才转换成 numeric(p,s)
float	float（实际按照 double precision 或 real 存储）
binary_float	binary_float（实际按照 real 存储）
binary_double	binary_double（实际按照 double precision 存储）
char	char
nchar	char
varchar2	varchar2
nvarchar2	nvarchar2
rowid	rowid
urowid	urowid
long	long
clob	clob
nclob	nclob
blob	blob
bfile	bfile
Long raw	Long raw
raw	raw
date	date

timestamp	Timestamp
Interval	interval

与 MySQL 对照表

MySQL	TDSQL PostgreSQL版
int	int
smallint	smallint
bigint	bigint
int AUTO_INCREMENT	serial
smallint AUTO_INCREMENT	smallserial
bigint AUTO_INCREMENT	bigserial
bit	bit
tinyint	smallint
float	real
double	double precision
decimal	numeric
char	char
varchar	varchar
text	text
date	date
time	time
datetime	timestamp
longblob	bytea
Longtext	text

ENUMCREATE TABLE TYPE022(COL1 ENUM('S','M','L','XL','XXL','XXXL') ,COL2 INT PRIMARY KEY);	自定义类型CREATE TYPE mood AS ENUM ('S','M','L','XL','XXL','XXXL');CREATE TABLE TYPE022(COL1 mood ,COL2 INT PRIMARY KEY)
SET类型CREATE TABLE TYPE023(COL1 SET('A','B', 'C','D') ,COL2 INT PRIMARY KEY)	CREATE TABLE TYPE023(COL1 VARCHAR check(regexp_split_to_array(col1,',') <@ array['A','B','C','D']) ,COL2 INT PRIMARY KEY);

与 SQL Server 对照表

SQL Server	TDSQL PostgreSQL版
smallint	smallint
int	int
bigint	bigint
tinyint	smallint
real	real
float	double precision
numeric	numeric
bit	bit
char	char
nchar	char
varchar	varchar
nvarchar	varchar
text	text
ntext	text
date	date
time	time
datetime	timestamp
datetime2	timestamp

smalldatetime	timestamp
datetimeoffset	Timestamp
timestamp	money
uniqueidentifier	uuid
image	bytea
binary	bytea
varbinary	bytea

与 informix 对照表

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

库模式表索引视图物化视图等 DDL 操作
数据库管理

最近更新时间：2024-09-13 15:36:01

创建数据库

要创建一个数据库，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 您可以创建一个干净的数据库，它将只包含的 TDSQL PostgreSQL 版所预定义的标准对象。

- **默认参数创建数据库**

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

- **指定克隆库**

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

- **指定所有者**

```
postgres=# create role pgxz with login;
CREATE ROLE
postgres=#  create database tdsql_pg_db_owner owner pgxz;
CREATE DATABASE
postgres=# \l+ tdsql_pg_db_owner
                                  List of databases
  Name     | Owner | Encoding |  Collate   |  Ctype    | Access
privileges | Size   | Tablespace | Description
 ----------------+-------+----------+------------+------------+-------
------------+-------+------------+-------------
tdsql_pg_db_owner | pgxz | UTF8     | en_US.utf8 | en_US.utf8 |
| 18 MB | pg_default |
(1 row)
```

- **指定编码**

```
postgres=# create database tdsql_pg_db_encoding ENCODING UTF8;
CREATE DATABASE
postgres=# \l+  tdsql_pg_db_encoding
                                List of databases
   Name      | Owner | Encoding |  Collate   |  Ctype    | Access
privileges | Size   | Tablespace | Description
 -------------------+-------+----------+------------+------------+----
---------------+-------+------------+-------------
tdsql_pg_db_encoding | dbadmin | UTF8     | en_US.utf8 | en_US.utf8 |
| 18 MB | pg_default |
(1 row)
```

- **创建 gbk 编码**

> **说明：**
> TDSQL PostgreSQL 版（Oracle 兼容版）支持。

```
postgres=# CREATE DATABASE db_gbk template template0 encoding = gbk
LC_COLLATE = 'zh_CN.gbk' LC_CTYPE = 'zh_CN.gbk';
CREATE DATABASE
postgres=# \l+
                                 List of databases
Name    | Owner | Encoding |  Collate   |  Ctype    | Access privileges |
Size   | Tablespace |          Description
-----------+-------+----------+------------+------------+-------------
------+-------+------------+---------------------------------------------
--
 db_gbk    | dbadmin | GBK      | zh_CN.gbk  | zh_CN.gbk  |           | 19
MB | pg_default |
 postgres  | dbadmin | UTF8     | zh_CN.utf8 | zh_CN.utf8 |           |
26 MB | pg_default | default administrative connection database
 template0 | dbadmin | UTF8     | zh_CN.utf8 | zh_CN.utf8 | =c/dbadmin
+| 19 MB | pg_default | unmodifiable empty database
     |     |       |        |        | dbadmin=CTc/dbadmin  |     |        |
 template1 | dbadmin | UTF8     | zh_CN.utf8 | zh_CN.utf8 | =c/dbadmin
+| 24 MB | pg_default | default template for new databases
     |     |       |        |        | dbadmin=CTc/dbadmin  |     |        |
(4 rows)
```

- **指定排序规则**

```
postgres=# create database tdsql_pg_db_lc_collate lc_collate 'C';
CREATE DATABASE
postgres=# \l+ tdsql_pg_db_lc_collate
                                List of databases
   Name      | Owner | Encoding | Collate |  Ctype    | Access
privileges | Size   | Tablespace | Description
 ----------------------+-------+----------+---------+------------+-----
--------------+-------+------------+-------------
tdsql_pg_db_lc_collate | dbadmin | UTF8    | C     | en_US.utf8 |
| 18 MB | pg_default |
(1 row)
```

- **指定分组规则**

```
postgres=# create database tdsql_pg_db_lc_ctype LC_CTYPE 'C' ;
CREATE DATABASE
postgres=# \l+  tdsql_pg_db_lc_ctype
                              List of databases
   Name      | Owner | Encoding |  Collate   | Ctype | Access privileges
| Size   | Tablespace | Description
-------------------+-------+----------+------------+-------+----------
---------+-------+------------+-------------
tdsql_pg_db_lc_ctype | dbadmin | UTF8    | en_US.utf8 | C    |
| 18 MB | pg_default |
(1 row)
```

- **配置数据可连接**

```
postgres=# create database tdsql_pg_db_allow_connections
ALLOW_CONNECTIONS true;
CREATE DATABASE
postgres=# select datallowconn from pg_database where
datname='tdsql_pg_db_allow_connections';
  datallowconn
--------------
  t
(1 row)
```

- **配置连接数**

```
postgres=# create database tdsql_pg_db_connlimit CONNECTION LIMIT 100;
```

```
CREATE DATABASE
postgres=# select datconnlimit  from pg_database where
datname='tdsql_pg_db_connlimit';
  datconnlimit
--------------
       100
(1 row)
```

- **配置数据库可以被复制**

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

- **多个参数一起配置**

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

修改数据库配置

- **修改数据库名称**

```
postgres=# alter database tdsql_pg_db rename to tdsql_pg_db_new;
ALTER DATABASE
```

- **修改连接数**

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

- **修改数据库所有者**

```
postgres=# alter database tdsql_pg_db_new owner to tdsql_pg;
```

```
ALTER DATABASE
```

- **配置数据默认运行参行**

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

- **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

删除数据库

```
postgres=# drop database tdsql_pg_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 tdsql_pg_db_template;
ERROR:  database "tdsql_pg_db_template" is being accessed by other
usersDETAIL:  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='tdsql_pg_db_template';
 pg_terminate_backend
----------------------
(1 rows)
postgres=# drop database tdsql_pg_db_template;
DROP DATABASE
```

模式管理

最近更新时间：2022-09-02 16:23:03

模式本质上是一个名字空间，Oracle 里一般叫用户，SQL Server 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"。

创建模式

- **标准语句**

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

- **扩展语法，不存在时才创建**

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

- **指定所属用户**

```
 postgres=# create schema tdsql_pg_pgxz AUTHORIZATION pgxz;
 CREATE SCHEMA
 postgres=# \dn tdsql_pg_pgxz
List of schemas
 Name    | Owner
 ------------+-------
tdsql_pg_pgxz | pgxz
 (1 row)
```

修改模式属性

- **修改模式名**

```
 postgres=# alter schema tdsql_pg rename to tdsql_pg_new;
 ALTER SCHEMA
```

- **修改所有者**

```
postgres=# alter schema tdsql_pg_pgxz owner to tdsql_pg;
ALTER SCHEMA
```

删除模式

```
postgres=# drop schema tdsql_pg_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 tdsql_pg_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 tdsql_pg_pgxz;
 ERROR:  cannot drop schema tdsql_pg_pgxz because other objects depend
on it
 DETAIL:  table tdsql_pg_pgxz.t depends on schema tdsql_pg_pgxz
 HINT:  Use DROP ... CASCADE to drop the dependent objects too.
```

参考如下强制删除。

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

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 dbadmin
 psql (PostgreSQL 10.0 tdsql_pg V2)
 Type "help" for help.

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

授权用户可以访问 tdsql_pg.t 表。

```
postgres=# grant select on tdsql_pg.t to pgxz;
GRANT
postgres=# \q
```

pgxz 用户登录。

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

在没授权用户可以使用 tdsql_pg 模式前，还是无法访问。

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

给 schema 授权。

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

postgres=# grant USAGE on SCHEMA tdsql_pg to pgxz;
GRANT
postgres=# \q
```

pgxz 用户再次访问表。

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

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

```
postgres=>
```

配置访问模式的顺序

TDSQL PostgreSQL版 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
 --------------
  tdsql_pg
 (1 row)
```

总共三个模式。

```
postgres=# \dn
  List of schemas
   Name    | Owner
--------------+-------
 public     | dbadmin
 tdsql_pg     | dbadmin
 tdsql_pg_schema | dbadmin
(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“tdsql_pg”。

```
 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
 --------+------+-------+-------
  tdsql_pg | t    | table | dbadmin
 (1 row)
```

指定表位于某个模式下，不同模式下表名可以相同。

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

```
postgres=# create table tdsql_pg_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 tdsql_pg_schema.t1;
 id | mc
----+----
(0 rows)
```

上面出错是因为模式 tdsql_pg_schema 没有配置在 search_path 搜索路径中。

创建和删除数据表

最近更新时间：2024-09-13 15:46:12

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作为表的 shard key。

```
postgres=# create table t_first_col_share(id serial not null,nickname
text);
CREATE TABLE
postgres=# \d+ t_first_col_share
                        Table "public.t_first_col_share"
 Column   |  Type   |              Modifiers               | Storage  |
Stats target | Description
----------+---------+--------------------------------------------------
-------------+-----------+--------------+-------------
 id       | integer | not null default
nextval('t_first_col_share_id_seq'::regclass) | plain    |          |
 nickname | text    |                                      | extended |
|
Has OIDs: no
Distribute By SHARD(id)
    Location Nodes: ALL DATANODES
```

分布键选择原则：

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

指定 shard key 建表方式

```
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
```

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);
CREATE TABLE
postgres=# insert into t_range(f1,f3) values(1,1),(2,50),(3,100),
(2,110);
INSERT 0 4
```

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)
CREATE TABLE

postgres=# \d+ t_time_range
                Table "public.t_time_range"
 Column |        Type        | Modifiers | Storage | Stats target |
Description
--------+----------------------------+-----------+---------+-----------
---+-------------
 f1     | bigint                     |       | plain   |         |
 f2     | timestamp without time zone |       | plain   |         |
 f3     | bigint                     |       | plain   |         |
Has OIDs: no
Distribute By SHARD(f1)
```

```
    Location Nodes: ALL DATANODES
Partition By: RANGE(f2)
  \# Of Partitions: 12
  Start With: 2017-09-01
  Interval Of Partition: 1 MONTH
```

字段说明：

- partition by range (x) 用于指定分区键，支持 timesamp,int 类型，数据分布于那个子表就是根据这个字段值来计算分区。
- begin(xxx) 指定开始分区的时间点。
- step(xxx) 指定分区有周期。
- partions(xx) 初始化时建立分区子表个数。
- 增加分区子表的方法 ALTER TABLE public.t1_pt ADD PARTITIONS 2;

逻辑分区表

range 分区表

范围分区的表被分区到由列定义的范围中，不同分区的值范围之间没有重叠。通常可以按时间进行分区，或者有特定对象的范围进行分区。其操作如下：

- 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);
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 "tdsql_pg.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
```

- **创建 default 分区**

> **说明:**
> TDSQL PostgreSQL 版（Oracle 兼容版）支持。

不创建 default 分区，插入范围越界出错。

```
postgres=# insert into t_native_range values(1,'2016-09-01',1);
ERROR:  node:dn001, backend_pid:19011,
nodename:dn001,backend_pid:19011,message:no partition of relation
"t_native_range" found for row
DETAIL:  Partition key of the failing row contains (f2) = (2016-09-01
00:00:00).
```

创建 default 分区后能正常插入数据。

```
 postgres=# CREATE TABLE t_native_range_default PARTITION OF
t_native_range DEFAULT;
```

```
CREATE TABLE
postgres=# insert into t_native_range values(1,'2016-09-01',1);
INSERT 0 1
```

list 分区表

表格通过明确的键值进行分区。

- **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);

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 "tdsql_pg.t_native_list"
Column |  Type   | Collation | Nullable |           Default           |
Storage  | Stats target | Description
--------+---------+-----------+----------+-----------------------------
---------------+----------+--------------+-------------
f1     | bigint  |           | not null |
nextval('t_native_list_f1_seq'::regclass) | plain    |              |
f2     | text    |           |          |                           | extended |
|
```

```
f3    | integer |       |       |                          | plain    |
|
f4    | date    |       |       |                          | plain    |
|
Partition key: LIST (f2)
Partitions: t_list_bj FOR VALUES IN ('北京'),
    t_list_gd FOR VALUES IN ('广东')
Distribute By: SHARD(f1)
Location Nodes: ALL DATANODES

postgres=#
```

- **创建 default 分区**

> **说明**
> TDSQL PostgreSQL 版（Oracle 兼容版）支持。

没有 default 分区情况下会出错，插入会出错。

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

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

创建后就能正常插入。

```
postgres=# insert into t_native_list values(1,'上海',1,current_date);
INSERT 0 1
postgres=#
```

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);
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
```

TDSQL PostgreSQL 版支持存在1级和2级分区混用，大家不需要都平级。
更多分区表的内容可参考 如下文档 。

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;
 CREATE TABLE
 postgres=# insert into t_rep values(1,'tdsql_pg'),(2,'pgxz');
 INSERT 0 2
```

```
postgres=# EXECUTE DIRECT ON (dn001) 'select * from t_rep';
 id |  mc
----+-------
 1 | tdsql_pg
 2 | pgxz
(2 rows)
postgres=# EXECUTE DIRECT ON (dn002) 'select * from t_rep';
 id |  mc
----+-------
 1 | tdsql_pg
 2 | pgxz
(2 rows)
```

可以看到所有节点都保存了一份相同的数据。

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=#
```

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
```

使用将查询结果创建数据表

创建 t 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,'tdsql_pg');
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 | tdsql_pg
(1 row)
```

查看 t 表的表结构

```
 postgres=# \d+ t
                    Table "tdsql_pg.t"
  Column |  Type   | Collation | Nullable | Default | Storage  | Stats
target | Description
 --------+---------+-----------+----------+---------+----------+--------
------+-------------
  id    | integer |           |          |         | plain    |
  mc    | text    |           |          |         | extended |
 Distribute By: SHARD(mc)
 Location Nodes: ALL DATANODES
```

查看 t_as 表的表结构

```
 postgres=# \d+ t_as
                   Table "tdsql_pg.t_as"
  Column |  Type   | Collation | Nullable | Default | Storage  | Stats
target | Description
 --------+---------+-----------+----------+---------+----------+--------
------+-------------
  id    | integer |           |          |         | plain    |
  mc    | text    |           |          |         | extended |
 Distribute By: SHARD(id)
 Location Nodes: ALL DATANODES

 postgres=#
```

删除数据表

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

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

添加分区子表

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

```
      Interval Of Partition: 1 MONTH

postgres=# ALTER TABLE t1_pt ADD PARTITIONS 2;

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

创建和删除索引

最近更新时间：2023-07-20 10:49:01

普通索引

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

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.
```

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)
```

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)
```

索引对于条件在性别为男的情况下无效。

```
 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 节点查看索引占用空间大，而且度数也高。

```
 [tbase@VM_0_37_centos shell]$ psql -p 11010
 psql (PostgreSQL 10.0 tdsql_pg V2)
 Type "help" for help.

 postgres=# \di+
                    List of relations
  Schema |    Name     | Type  | Owner |  Table   | Size  | Allocated Size
| Description
 --------+-------------+-------+-------+---------+-------+------------
----+-------------
  tdsql_pg | t_sex_sex_idx | index | dbadmin | t_sex  | 14 MB | 14 MB
|
  tdsql_pg | t_upper_mc    | index | dbadmin | t_upper | 14 MB | 14 MB
|
```

```
 (2 rows)

 postgres=# \q

 [tbase@VM_0_37_centos shell]$ psql
 psql (PostgreSQL 10.0 tdsql_pg 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 tdsql_pg V2)
 Type "help" for help.

 postgres=# \di+
                   List of relations
  Schema |    Name     | Type  | Owner |  Table   | Size  | Allocated Size
| Description
```

```
 --------+---------------+-------+-------+---------+-------+------------
----+-------------
  tdsql_pg | t_sex_sex_idx | index | dbadmin | t_sex   | 16 kB | 16 kB
|
  tdsql_pg | t_upper_mc    | index | dbadmin | t_upper | 14 MB | 14 MB
|
 (2 rows)

 postgres=#
```

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
```

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=#
```

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
```

删除索引

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

修改表结构

最近更新时间：2022-09-02 16:25:35

修改表名

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

给表或字段添加注释

```
postgres=# comment on table tdsql_pg is ' TDSQL PostgreSQL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 | tdsql_pg        | table | pgxz  | 24 kB |  TDSQL PostgreSQL分布
式关系型数据库系统
(3 rows)
postgres=# comment on column tdsql_pg.nickname is ' TDSQL PostgreSQL昵称
是大象';
COMMENT
postgres=# \d+ tdsql_pg
                       Table "public.tdsql_pg"
 Column   |  Type   |          Modifiers           | Storage  | Stats
target |  Description
----------+---------+------------------------------------------------------+--
--------+--------------+-----------------
 id     | integer | not null default nextval('t_id_seq'::regclass) |
plain    |         |
 nickname | text    |                              | extended |         |
TDSQL PG版呢称是大象
Has OIDs: no
Distribute By SHARD(id)
    Location Nodes: ALL DATANODES
```

```
postgres=#
```

给表增加字段

```
postgres=# alter table tdsql_pg add column age integer;
ALTER TABLE
postgres=# \d+ tdsql_pg
                        Table "public.tdsql_pg"
 Column   |  Type   |          Modifiers           | Storage  | Stats
target |  Description
----------+---------+------------------------------------------------+--
--------+--------------+-----------------
 id       | integer | not null default nextval('t_id_seq'::regclass) |
plain    |         |
 nickname | text    |                                | extended |         |
TDSQL PG版呢称是大象
 age      | integer |                                | plain    |         |
Has OIDs: no
Distribute By SHARD(id)
    Location Nodes: ALL DATANODES
```

修改字段类型

```
postgres=# alter table tdsql_pg alter column age type float8;
ALTER TABLE
postgres=#  \d+ tdsql_pg
                           Table "public.tdsql_pg"
 Column   |     Type     |          Modifiers           | Storage  | Stats
target |  Description
----------+------------------+--------------------------------------------
------+----------+--------------+-----------------
 id       | integer          | not null default nextval('t_id_seq'::regclass) |
plain    |         |
 nickname | text             |                              | extended |         |
TDSQL PG版呢称是大象
 age      | double precision |                              | plain    |
|
Has OIDs: no
Distribute By SHARD(id)
    Location Nodes: ALL DATANODES
```

```
postgres=#
```

修改字段默认值

```
postgres=# alter table tdsql_pg alter column age set default 0.0;
ALTER TABLE
postgres=#  \d+ tdsql_pg
                          Table "public.tdsql_pg"
 Column  |     Type     |          Modifiers           | Storage  | Stats
target |  Description
----------+------------------+------------------------------------------
------+----------+--------------+-----------------
 id      | integer       | not null default nextval('t_id_seq'::regclass) |
plain    |         |
 nickname | text         |                          | extended |         |
TDSQL PG版呢称是大象
 age     | double precision | default 0.0                  | plain    |
|
Has OIDs: no
Distribute By SHARD(id)
    Location Nodes: ALL DATANODES
```

删除字段

```
postgres=# alter table tdsql_pg drop column age;
ALTER TABLE
postgres=#  \d+ tdsql_pg
                       Table "public.tdsql_pg"
 Column  |  Type  |          Modifiers           | Storage  | Stats
target |  Description
----------+---------+-----------------------------------------------+--
--------+--------------+-----------------
 id      | integer | not null default nextval('t_id_seq'::regclass) |
plain    |         |
 nickname | text    |                          | extended |         |
TDSQL PG版呢称是大象
Has OIDs: no
Distribute By SHARD(id)
    Location Nodes: ALL DATANODES
```

添加主键

```
postgres=# ALTER TABLE t ADD CONSTRAINT t_id_pkey PRIMARY KEY (id) ;
ALTER TABLE
postgres=# \d+ t
                    Table "tdsql_pg.t"
 Column |  Type   | Collation | Nullable | Default | Storage  | Stats
target | Description
--------+---------+-----------+----------+---------+----------+---------
-----+-------------
 id     | integer |           | not null |         | plain    |         |
 mc     | text    |           |          |         | extended |         |
Indexes:
  "t_id_pkey" PRIMARY KEY, btree (id)
Distribute By: SHARD(id)
Location Nodes: ALL DATANODES
```

删除主键

```
postgres=# ALTER TABLE t DROP CONSTRAINT t_id_pkey ;
ALTER TABLE
postgres=# \d+ t
                    Table "tdsql_pg.t"
 Column |  Type   | Collation | Nullable | Default | Storage  | Stats
target | Description
--------+---------+-----------+----------+---------+----------+---------
-----+-------------
 id     | integer |           | not null |         | plain    |         |
 mc     | text    |           |          |         | extended |         |
Distribute By: SHARD(id)
Location Nodes: ALL DATANODES
```

如果是分区表，则删除主键要加上 cascade，强制删除关联的子表主键。

重建主键

```
postgres=# \d t
          Table "public.t"
 Column |  Type   | Collation | Nullable | Default
--------+---------+-----------+----------+---------
 id     | integer |           | not null |
 mc     | text    |           |          |
Indexes:
```

```
"t_pkey" PRIMARY KEY, btree (id)

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

postgres=#

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

postgres=#
```

添加外键

```
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。

删除外键

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

修改表所属模式

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

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

postgres=# \dt t
    List of relations
 Schema | Name | Type  | Owner
--------+------+-------+-------
 public | t    | table | dbadmin
```

```
(1 row)
```

修改表所属用户

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

postgres=# alter table tdsql_pg owner to pgxz;
ALTER TABLE

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

修改字段名

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

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

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

```
----------+-----------------------+----------+----------+---------+----
------+--------------+-------------
 id       | integer               |          | not null |         | plain    |             |
 cityname | character varying(50) |          |          |         | extended |
|
Distribute By: SHARD(id)
Location Nodes: dn01
```

创建和删除视图

最近更新时间：2022-09-02 16:31:22

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)
```

TDSQL PostgreSQL版 支持视图引用表或字段改名联动，不受影响。

```
postgres=# \d+ t_view
              View "tdsql_pg.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 "tdsql_pg.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;
```

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
```

物化视图使用

创建物化视图

```
postgres=# CREATE MATERIALIZED VIEW t_range_mv AS select f1,f2::date
from t_range;
SELECT 5
```

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)
```

增量数据刷新

```
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)

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。

truncate 操作

最近更新时间：2023-04-18 10:37:35

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。

truncate 普通表

使用语法：`truncate table xx yy zz`；具体例子如下所示：

```
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=#
```

truncate 分区表

- truncate 一个时间分区表。

使用语法：`truncate xx partition for(x)`，具体例子如下所示：

- 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 | tdsql_pg
 2 | 2017-10-01 00:00:00 | pgxz
(2 rows)

postgres=# truncate t_time_range  partition for  ('2017-09-01'
::timestamp without time zone);
TRUNCATE TABLE
postgres=# select * from t_time_range;
 f1 |      f2       |  f3
----+---------------------+------
 2 | 2017-10-01 00:00:00 | pgxz
(1 row)

postgres=#
```

- **truncate 一个数字分区表。**

```
postgres=# \d+ t_range
                Table "pgxz.t_range"
 Column |       Type        |  Modifiers   | Storage | 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)
```

创建和删除外键

最近更新时间：2023-06-05 18:24:41

创建外键

```
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须都是表的分布键，否则由于数据分布于不同的节点内会导致更新失败。

删除外键

```
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

postgres=# ALTER TABLE t_f drop constraint t_f_f1_fkey;
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   |
Distribute By: SHARD(f1)
Location Nodes: ALL DATANODES
```

自定义复合类型

最近更新时间：2022-09-02 16:40:04

增加一个复合类型

```
postgres=# create type my_type as (f1 int,f2 varchar(10));
CREATE TYPE
```

数据表使用复合类型

```
postgres=# create table t_my_type (f1 int,f_my_type my_type);
CREATE TABLE
postgres=# insert into t_my_type values(1,row(1,'tdsql_pg'));
INSERT 0 1
postgres=# select f1,(f_my_type).f2 from t_my_type;
 f1 |   f2
----+--------
  1 | tdsql_pg
(1 row)
```

DML 语法
SELECT 语法

最近更新时间：2023-05-29 18:23:16

访问函数

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

数据排序

- **按某一列排序**

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

- **按第一列排序**

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

- **按 ID 升级排序，再按 nickname 降序排序**

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

效果与上面的语句一样。

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

- **随机排序**

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

- **计算排序**

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

排序也能用子查询。

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

- **null 值排序结果处理**

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

null 值记录排在最前面。

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

null 值记录排在最后。

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

- **按拼音排序**

```
postgres=# select * from (values ('张三'), ('李四'),('陈五')) t(myname)
order by myname;
```

```
 myname
--------
 张三
 李四
 陈五
(3 rows)
```

如果不加处理，则按汉字的 UTF-8 编码进行排序，不符合中国人使用习惯。

- 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(myname::bytea,'UTF-8','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 convert_to(myname,'GBK');
 myname
--------
 陈五
 李四
 张三
(3 rows)
```

- 通过指定排序规则 collact 来实现汉字按拼音进行排序。

```
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)
```

where 条件使用

- **单条件查询**

```
postgres=# select * from tdsql_pg where id=1;
 id |       nickname
----+------------------------------
 1 | hello tdsql_pg
 1 | tdsql_pg分布式数据库的时代来了
```

- **多条件 and**

```
postgres=# select * from tdsql_pg where id=1 and nickname like '%h%';
 id |  nickname
----+-------------
 1 | hello tdsql_pg
(1 row)
```

- **多条件 or**

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

- **ilike 不区分大小写匹配**

```
postgres=# create table t_ilike(id int,mc text);
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,'tdsql_pg'),(2,'tdsql_pg');
INSERT 0 2
postgres=# select * from t_ilike where mc ilike '%td%';
 id |  mc
----+-------
 1 | tdsql_pg
 2 | tdsql_pg
```

```
(2 rows)
```

- **where 条件也能支持子查询**

```
postgres=# select * from tdsql_pg where id=(select
(random()*2)::integer from tdsql_pg order by random() limit 1);
 id | nickname
----+----------
(0 rows)

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

- **null 值查询方法**

```
postgres=# select * from tdsql_pg where nickname is null;
 id | nickname
----+----------
 4 |
(1 row)

postgres=# select * from tdsql_pg where nickname is not null;
 id |       nickname
----+----------------------------
 1 | hello tdsql_pg
 2 | tdsql_pg好
 1 | tdsql_pg分布式数据库的时代来了
(3 rows)
```

- **exists 只要有记录返回就为真**

```
postgres=#  create table t_exists1(id int,mc text);
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,'tdsql_pg'),(2,'tdsql_pg');
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,'tdsql_pg'),(1,'tdsql_pg');
INSERT 0 2

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

- **exists 等价写法**

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

分页查询

默认从第一条开始，返回一条记录。

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

使用 offset 指定从第几条开始，0表示第一条开始，返回1条记录。

```
postgres=# select * from tdsql_pg limit 1 offset 0;
 id |  nickname
----+-------------
```

```
 1 | hello tdsql_pg
(1 row)
```

从第3条开始，返回二条记录。

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

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

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

合并多个查询结果

- 不过滤重复的记录。

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

- 过滤重复的记录。

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

```
(3 rows)
```

- 每个子查询分布在合并结果中的使用。

```
postgres=# select * from ( select * from tdsql_pg limit 1) as t union
all select * from (select * from t_appoint_col limit 1) as t ;
 id |  nickname
----+-------------
 1 | hello tdsql_pg
 1 | hello tdsql_pg
(2 rows)
```

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,'tdsql_pg'),(2,'tdsql_pg');
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,'tdsql_pg'),(3,'tdsql_pg');
INSERT 0 2

postgres=# select * from t_intersect1 INTERSECT select * from
t_intersect2;
 id |  mc
----+-------
 1 | tdsql_pg
(1 row)
```

返回两个结果的差集

```
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,'tdsql_pg'),(2,'tdsql_pg');
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,'tdsql_pg'),(3,'tdsql_pg');
INSERT 0 2

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

any 用法

只需要大于其中一个值即为真。

```
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,'tdsql_pg'),(2,'tdsql_pg');
INSERT 0 2

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

all 用法

需要大于所有值才为真。

```
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,'tdsql_pg'),(3,'tdsql_pg');
INSERT 0 2

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

聚集查询

- 统计记录数。

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

- 统计不重复值的记录表。

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

- 求和。

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

- 求最大值。

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

- **求最小值。**

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

- **求平均值。**

```
postgres=# select avg(id) from tdsql_pg;
        avg
--------------------
 1.3333333333333333
(1 row)
```

多表关联

- **内连接**

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

- **左外连接**

```
 postgres=# select * from tdsql_pg left join t_appoint_col on
tdsql_pg.id=t_appoint_col.id;
 id |          nickname          | id |  nickname
----+----------------------------+----+-------------
```

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

- **右外连接**

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

- **全连接**

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

聚合函数并发计算

- **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=#
```

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,'tdsql_pg'),(2,'pgxz');
INSERT 0 2
postgres=# select * from t_not_in where id not in (3,5);
 id |  mc
----+-------
```

```
 1 | tdsql_pg
 2 | pgxz
(2 rows)

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

只查某个 DN 的数据

```
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,'tdsql_pg'),(3,'pgxz');
INSERT 0 2
postgres=# EXECUTE DIRECT ON (dn001) 'select * from t_direct';
 id |  mc
----+-------
 1 | tdsql_pg
(1 row)

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

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

postgres=#
```

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,'tdsql_pg'),
(2,'tdsql_pg');
INSERT 0 2

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

- **一列变成多行**

```
postgres=# create table t_col_to_mulrow(id int,mc text);
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,'tdsql_pg,tdsql_pg');
INSERT 0 1

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

regexp_split_to_table
-----------------------
tdsql_pg
tdsql_pg
(2 rows)
```

查询记录所在 DN

```
postgres=# select xc_node_id,* from t1;
 xc_node_id | f1 | f2
------------+----+----
 2142761564 |  1 |  3
```

```
 2142761564 |  1 |  3
(2 rows)

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

postgres=#
```

grouping sets/rollup/cube 用法

group by 用法

- 销售明细表。

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

- 按 dep、product 两级汇总分数。

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

```
 业务3部 | 电脑   |  80
 业务3部 | 手机   | 160
```

使用 grouping sets

> **说明：**
> grouping sets 的每个子列表可以指定零个或多个列或表达式，并且与其直接在 GROUP BY 子句中的解释方式相同。一个空的分组集合意味着所有的行都被聚合到一个组中。

如按 name、class 单级分别汇总，再计算一个总分。

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

使用 grouping sets 代替 group by 。

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

使用 rollup

rollup((a),(b)) 等价于 grouping sets((a,b),(a),())。

```
postgres=# select dep,product,sum(num) from t_grouping group by
rollup((dep),(product)) order by dep,product;
```

```
  dep   | product | sum
--------+---------+-----
 业务1部 | 电脑    |  80
 业务1部 | 手机    | 160
 业务1部 |         | 240
 业务2部 | 电脑    | 120
 业务2部 | 手机    |  50
 业务2部 |         | 170
 业务3部 | 电脑    |  80
 业务3部 | 手机    | 160
 业务3部 |         | 240
         |         | 650
```

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

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

使用 cube

cube((a),(b)) 等价于 grouping sets((a,b),(a),(b),()) 。

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

```
 业务2部 | 手机    |  50
 业务2部 |       | 170
 业务3部 | 电脑    |  80
 业务3部 | 手机    | 160
 业务3部 |       | 240
      | 电脑    | 280
      | 手机    | 370
      |       | 650
```

该功能等价于 grouping sets((name,class),(name),(class),())。

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

INSERT 语法

最近更新时间：2023-06-08 11:17:31

插入单条记录

指定所有字段。

```
postgres=# insert into tdsql_pg(id,nickname) values(1,'hello tdsql_pg');
INSERT 0 1
```

指定某些字段，不指定时，如果该字段有默认值则会带上默认值。

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

不指定字段，则默认为所有字段与建表时一致。

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

字段顺序可以任意排列。

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

使用 default 关键字，即值为建表时指定的默认值方式。

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

上面五次插入记录后产生的数据。

```
postgres=# select * from tdsql_pg;
 id |   nickname
----+---------------
 1 | hello tdsql_pg
```

```
 2 | tdsql_pg好
 5 | tdsql_pg swap
 3 | tdsql_pg好
 4 | tdsql_pg default
(5 rows)
```

插入多数记录

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

使用子查询插入数据

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

从另外一个表取数据进行批量插入

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

```
(3 rows)
```

大批量的生成数据

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

返回插入数据，轻松获取插入记录的 serial 值

```
postgres=# insert into tdsql_pg(nickname) values('tdsql_pg好') returning
*;
 id | nickname
----+-----------
 7 | tdsql_pg好
(1 row)
INSERT 0 1

postgres=# insert into tdsql_pg(nickname) values('hello tdsql_pg')
returning id;
 id
\----
 8
(1 row)
```

指定返回的字段。

insert..update 更新

使用 ON CONFLICT

```
postgres=# \d+ t
                Table "public.t"
 Column |  Type   | Modifiers | Storage  | Stats target | Description
--------+---------+-----------+----------+--------------+-------------
 id     | integer |           | plain    |              |
```

```
 nc    | text    |       | extended |         |
Indexes:
  "t_id_udx" UNIQUE, btree (id)
Has OIDs: no
Distribute By SHARD(id)
    Location Nodes: ALL DATANODES

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

postgres=# insert into t values(1,'pgxz') ON CONFLICT (id)  DO UPDATE
SET nc = 'tdsql_pg';
INSERT 0 1
postgres=# select * from t;
 id |  nc
----+-------
 1 | tdsql_pg
(1 row)
```

UPDATE 语法

最近更新时间：2022-09-02 16:50:02

单表更新

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

null 条件的表达方法。

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

多表关联更新

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

返回更新的数据

```
postgres=# update tdsql_pg set nickname = nickname where id =
(random()*2)::integer returning *;
 id | nickname
----+-----------
```

```
 2 | tdsql_pg好
(1 row)
```

上面的语句随机更新了一些数据，然后返回更新过的记录，returning 机制旨在降低应用的复杂度。

多列匹配更新

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

shard key 禁止更新操作

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

DELETE 语法

最近更新时间：2022-09-02 16:50:26

带条件删除

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

postgres=# delete from tdsql_pg where id=4;
DELETE 1
```

null 条件的表达方式。

```
postgres=# delete from tdsql_pg where nickname is null;
DELETE 1
postgres=# select * from tdsql_pg;
 id |  nickname
----+-------------
 2 | tdsql_pg好
 1 | Hello tdsql_pg
(2 rows)
```

多表关联删除数据

```
postgres=# select * from tdsql_pg;
 id |  nickname
----+-------------
 2 | tdsql_pg好
 1 | Hello tdsql_pg
(2 rows)

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

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

返回删除数据

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

returning 特性可以返回 DML（insert、update、delete）修改的数据，降低应用复杂度。

删除所有数据

```
postgres=# insert into tdsql_pg select t,random()::text from
generate_series(1,100000) as t;
postgres=# \timing
Timing is on.
postgres=# delete from tdsql_pg ;
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 tdsql_pg select t,random()::text from
generate_series(1,100000) as t;
INSERT 0 100000
Time: 13178.429 ms
postgres=# truncate table tdsql_pg;
TRUNCATE TABLE
Time: 24.242 ms
```

序列的使用

最近更新时间：2024-09-27 15:59:51

序列的创建和访问

建立序列

示例：

```
[root@VM-0-14-tencentos ~]# psql -h 10.0.0.34 -p5432 -Udbadmin -
dpostgres
Password for user dbadmin:
psql (16.0, server 10.0 @ TBase_v5.06.5.1 (commit: b0f2c3390))
Type "help" for help.
postgres=# create sequence tdsql_pg_seq;
CREATE SEQUENCE
```

建立序列，不存在时才创建

示例：

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

查看序列当前的使用状况

示例：

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

获取序列的下一个值

示例：

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

获取序列的当前值，这个需要在访问 nextval() 后才能使用

示例：

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

也可以用下面的方式来获取序列当前使用的值。

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

设置序列当前值

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

序列在 DML 中使用

示例：

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

```
 2 | tdsql_pg好
(2 rows)
```

序列作为字段的默认值使用

示例：

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

序列作为字段类型使用

示例：

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

删除序列

示例：

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

删除序列，不存在时跳过

示例：

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

游标的使用

最近更新时间：2022-09-02 16:54:15

定义一个游标

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

游标需要放在一个事务中使用。

提取下一行数据

```
postgres=# DECLARE tdsql_pg_cur SCROLL CURSOR FOR SELECT * from tdsql_pg
ORDER BY id;
DECLARE CURSOR
postgres=# FETCH NEXT from tdsql_pg_cur ;
 id |  nickname
----+-------------
 1 | hello tdsql_pg
(1 row)

postgres=# FETCH NEXT from tdsql_pg_cur ;
 id | nickname
----+-----------
 2 | tdsql_pg好
(1 row)
```

提取前一行数据

```
postgres=# FETCH PRIOR from tdsql_pg_cur ;
 id |  nickname
----+-------------
 1 | hello tdsql_pg
(1 row)

postgres=# FETCH PRIOR from tdsql_pg_cur ;
```

```
 id | nickname
----+----------
(0 rows)
```

提取最后一行

```
postgres=# FETCH LAST from tdsql_pg_cur ;
 id |  nickname
----+------------
 5 | tdsql_pg swap
(1 row)
```

提取第一行

```
postgres=# FETCH FIRST from tdsql_pg_cur ;
 id |  nickname
----+-------------
 1 | hello tdsql_pg
(1 row)
```

提取该查询的第 x 行

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

postgres=# FETCH ABSOLUTE -1 from tdsql_pg_cur ;
 id |  nickname
----+------------
 5 | tdsql_pg swap
(1 row)

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

x 为负数时则从尾部向上提。

提取当前位置后的第 x 行

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

postgres=# FETCH RELATIVE 2 from tdsql_pg_cur ;
 id | nickname
----+-----------
 3 | tdsql_pg好
(1 row)

postgres=# FETCH RELATIVE 2 from tdsql_pg_cur ;
 id |  nickname
----+------------
 5 | tdsql_pg swap
(1 row)
```

每提取一次数据，游标的位置都是会前行。

向前提取 x 行数据

```
postgres=# FETCH FORWARD 2 from tdsql_pg_cur ;
 id |  nickname
----+-------------
 1 | hello tdsql_pg
 2 | tdsql_pg好
(2 rows)

postgres=# FETCH FORWARD 2 from tdsql_pg_cur ;
 id |  nickname
----+---------------
 3 | tdsql_pg好
 4 | tdsql_pg default
(2 rows)
```

向前提取剩下的所有数据

```
postgres=# FETCH FORWARD 2 from tdsql_pg_cur ;
 id |  nickname
----+-------------
 1 | hello tdsql_pg
 2 | tdsql_pg好
(2 rows)

postgres=# FETCH FORWARD all from tdsql_pg_cur ;
 id |  nickname
----+---------------
 3 | tdsql_pg好
 4 | tdsql_pg default
 5 | tdsql_pg swap
(3 rows)
```

向后提取 x 行数据

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

向后提取剩下的所有数据

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

copy 的使用

最近更新时间：2023-08-09 15:04:31

COPY 用于 TDSQL PostgreSQL版 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。
如果指定了一个列清单，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清单中，COPY FROM 将会为那些列插入默认值。
使用 COPY 时 TDSQL PostgreSQL版 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 TDSQL PostgreSQL版 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。

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
```

数据测试过程中再录入修改。

```
postgres=# select * from t;
 f1 |   f2   |         f3         | f4
----+--------+----------------------------+----
  3 | pgxz   | 2017-10-28 18:24:05.645691 |
  1 | tdsql_pg |                          |  7
  2 |        | 2017-10-28 18:24:05.643102 |  3
(3 rows)
```

copy to 复制数据到文件中

导出所有列

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

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

导出部分列

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

只导出 f1 和 f2 列。

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
tdsql_pg  \N
postgres=#
```

查询可以是任何复杂查询。

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,tdsql_pg,,7
```

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

- **生成二进制格式**

```
postgres=# copy public.t to '/data/pgxz/t.txt' with binary;
COPY 3
postgres=# \1
postgres=# \! cat  /data/pgxz/t.txt
PGCOPY
 tdsql_pg
```

默认为 TEXT 格式。

使用 delimiter 指定列与列之间的分隔符

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

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

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

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

NULL 值的处理

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

postgres=# copy public.t to '/data/pgxz/t.txt' with CSV NULL 'NULL';
COPY 3
postgres=# \! cat /data/pgxz/t.txt
1,tdsql_pg,NULL,7
2,pgxc,2017-10-28 18:24:05.643102,3
3,pgxz,2017-10-28 18:24:05.645691,NULL

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

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

生成列标题名

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

只有使用 CSV 格式时才允许这个选项。

导出 oids 系统列

```
postgres=# drop table t;
DROP TABLE
postgres=# CREATE TABLE t (
postgres(#    f1 integer NOT NULL,
postgres(#    f2 text NOT NULL,
```

```
postgres(#    f3 timestamp without time zone,
postgres(#    f4 integer
postgres(# )
postgres-# with oids DISTRIBUTE BY SHARD (f1);
CREATE TABLE
postgres=# copy t from '/data/pgxz/t.txt' with csv ;
COPY 3
postgres=# select * from t;
 f1 |    f2     |        f3        | f4
----+-----------+--------------------------+----
  1 | tdsql_pg      |                  |  7
  2 | pg'",     xc | 2017-10-28 18:24:05.643102 |  3
  3 | pgxz       | 2017-10-28 18:24:05.645691 |
(3 rows)

postgres=# copy t to  '/data/pgxz/t.txt' with oids ;
COPY 3
postgres=# \! cat /data/pgxz/t.txt
35055  1    tdsql_pg  \N    7
35056  2    pg'",     xc  2017-10-28 18:24:05.643102    3
35177  3    pgxz   2017-10-28 18:24:05.645691    \N
```

创建表时使用了 with oids 才能使用 oids 选项。

使用 quote 自定义引用字符

```
postgres=# copy t to  '/data/pgxz/t.txt' with csv;
COPY 3
postgres=#  \! cat /data/pgxz/t.txt
1,tdsql_pg,,7
2,"pg'"",     xc%",2017-10-28 18:24:05.643102,3
3,pgxz,2017-10-28 18:24:05.645691,
```

默认引用字符为“双引号”。

```
postgres=# copy t to  '/data/pgxz/t.txt' with quote '%' csv;
COPY 3
postgres=# \! cat /data/pgxz/t.txt
1,tdsql_pg,,7
2,%pg'",     xc%%%,2017-10-28 18:24:05.643102,3
3,pgxz,2017-10-28 18:24:05.645691,
```

上面指定了引用字符为百分号，系统自动把字段值为%的字符替换为双个%。

```
postgres=# copy t to  '/data/pgxz/t.txt' with quote '%';
ERROR:  COPY quote available only in CSV mode
```

只有使用 CSV 格式时才允许这个选项。

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

引用字符必须是一个单一的单字节字符，即汉字是不支持的。

使用 escape 自定义逃逸符

```
postgres=# copy t to  '/data/pgxz/t.txt' with quote '%' csv;
COPY 3
postgres=# \! cat /data/pgxz/t.txt
1,tdsql_pg,,7
2,%pg'",    xc%%%,2017-10-28 18:24:05.643102,3
3,pgxz,2017-10-28 18:24:05.645691,
```

不指定 escape 逃逸符，默认和 QUOTE 值一样。

```
postgres=# copy t to  '/data/pgxz/t.txt' with quote '%' escape '@' csv;
COPY 3
postgres=# \! cat /data/pgxz/t.txt
1,tdsql_pg,,7
2,%pg'",    xc@%%,2017-10-28 18:24:05.643102,3
3,pgxz,2017-10-28 18:24:05.645691,
```

指定逃逸符为'@'。

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

这必须是一个单一的单字节字符。

```
postgres=# copy t to  '/data/pgxz/t.txt' with quote '%' escape '@';
```

```
ERROR:  COPY quote available only in CSV mode
```

只有使用 CSV 格式时才允许这个选项。

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","tdsql_pg",,7
"2","pg'"",    xc%",2017-10-28 18:24:05.643102,3
"3","pgxz",2017-10-28 18:24:05.645691,
```

指定2列强制添加引用字符。

```
postgres=# copy t to  '/data/pgxz/t.txt' (format 'csv',force_quote
(f1,f4,f3,f2));
COPY 3
postgres=# \! cat /data/pgxz/t.txt
"1","tdsql_pg",,"7"
"2","pg'"",    xc%","2017-10-28 18:24:05.643102","3"
"3","pgxz","2017-10-28 18:24:05.645691",
```

指定4列强制添加引用字符，字段的顺序可以任意排列。

```
postgres=# copy t to  '/data/pgxz/t.txt' (format 'text',force_quote
(f1,f2,f3,f4));
ERROR:  COPY force quote available only in CSV mode
postgres=#
```

只有使用 CSV 格式时才允许这个选项。

使用 encoding 指定导出文件内容编码

```
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=#
```

copy from 复制文件内容到数据表中

导入所有列

```
postgres=# \! cat  /data/pgxz/t.txt
1    tdsql_pg  \N    7
2    pg'",    xc%  2017-10-28 18:24:05.643102    3
3    pgxz   2017-10-28 18:24:05.645691    \N
postgres=# truncate table t;
TRUNCATE TABLE
postgres=# copy t from '/data/pgxz/t.txt';
COPY 3
postgres=# select * from t;
 f1 |    f2    |        f3        | f4
----+----------+--------------------------+----
  1 | tdsql_pg     |                  |  7
  2 | pg'",    xc% | 2017-10-28 18:24:05.643102 |  3
  3 | pgxz       | 2017-10-28 18:24:05.645691 |
(3 rows)
```

导入部分指定列

```
postgres=#  copy t(f1,f2) to '/data/pgxz/t.txt';
postgres=#  \! cat /data/pgxz/t.txt
1    tdsql_pg
2    pg'",    xc%
3    pgxz
postgres=# truncate table t;
TRUNCATE TABLE
```

```
postgres=# copy t(f1,f2) from  '/data/pgxz/t.txt';
COPY 3
postgres=# select * from t;
 f1 |    f2     |        f3        | f4
----+-----------+--------------------------+----
  1 | tdsql_pg      | 2017-10-30 11:54:16.559579 |
  2 | pg'",     xc% | 2017-10-30 11:54:16.559579 |
  3 | pgxz      | 2017-10-30 11:54:16.560283 |
(3 rows)
```

有默认值的字段在没有导入时，会自动的将默认值赋上。

```
postgres=# \! cat /data/pgxz/t.txt
1    \N    tdsql_pg
2    2017-10-28 18:24:05.643102    pg'",    xc%
3    2017-10-28 18:24:05.645691    pgxz
postgres=# truncate table t;
TRUNCATE TABLE
postgres=# copy t(f1,f3,f2) from '/data/pgxz/t.txt';
COPY 3
postgres=# select * from t;
 f1 |    f2     |        f3        | f4
----+-----------+--------------------------+----
  1 | tdsql_pg      |                  |
  2 | pg'",     xc% | 2017-10-28 18:24:05.643102 |
  3 | pgxz      | 2017-10-28 18:24:05.645691 |
(3 rows)
```

字段的顺序可以任意调整，但需要与导放文件的存放顺序一致。

```
postgres=# \! cat /data/pgxz/t.txt;
1    tdsql_pg  \N    7
2    pg'",    xc%  2017-10-28 18:24:05.643102    3
3    pgxz   2017-10-28 18:24:05.645691    \N
postgres=# copy t (f1,f2) from  '/data/pgxz/t.txt';
ERROR:  extra data after last expected column
CONTEXT:  COPY t, line 1: "1   tdsql_pg  \N    7"
```

数据文件的列表不能多于要导入的列数，否则会出错。

指定导入文件格式

```
postgres=# \! cat /data/pgxz/t.txt
1    tdsql_pg  \N    7
2    pg'",    xc%  2017-10-28 18:24:05.643102    3
3    pgxz   2017-10-28 18:24:05.645691    \N
postgres=# copy t from '/data/pgxz/t.txt' (format 'text');
COPY 3

TRUNCATE TABLE
postgres=# \! cat /data/pgxz/t.csv
1,tdsql_pg,,7
2,"pg'"",    xc%",2017-10-28 18:24:05.643102,3
3,pgxz,2017-10-28 18:24:05.645691,
postgres=#  copy t from '/data/pgxz/t.csv' (format 'csv');
COPY 3

postgres=# truncate table t;
TRUNCATE TABLE
postgres=# \! od -c /data/pgxz/t.bin
0000000  P  G  C  O  P  Y  \n 377  \r  \n  \0  \0  \0  \0  \0  \0
0000020  \0  \0  \0  \0 004  \0  \0  \0 004  \0  \0  \0 001  \0  \0  \0
0000040 005  T  b  a  s  e 377 377 377 377  \0  \0  \0 004  \0  \0
0000060  \0  \a  \0 004  \0  \0  \0 004  \0  \0  \0 002  \0  \0  \0 016
0000100  p  g  '  "  ,              x  c  %  \0  \0
0000120  \0  \b  \0 001 377 236  G  w 213  ^  \0  \0  \0 004  \0  \0
0000140  \0 003  \0 004  \0  \0  \0 004  \0  \0  \0 003  \0  \0  \0 004
0000160  p  g  x  z  \0  \0  \0  \b  \0 001 377 236  G  w 225  {
0000200 377 377 377 377 377 377
0000206
postgres=# copy t from '/data/pgxz/t.bin' (format 'binary');
COPY 3
postgres=# select * from t;
 f1 |    f2     |          f3          | f4
----+-----------+----------------------+----
  1 | tdsql_pg  |                      |  7
  2 | pg'",    xc% | 2017-10-28 18:24:05.643102 |  3
  3 | pgxz      | 2017-10-28 18:24:05.645691 |
(3 rows)
```

使用 delimiter 指定列与列之间的分隔符

```
postgres=#  \! cat /data/pgxz/t.txt
1%tdsql_pg%\N%7
```

```
2%pg'",    xc\%%2017-10-28 18:24:05.643102%3
3%pgxz%2017-10-28 18:24:05.645691%\N
postgres=# copy t from  '/data/pgxz/t.txt' (format 'text',delimiter
'%');
COPY 3
postgres=# select * from t;
 f1 |    f2    |        f3        | f4
----+----------+--------------------------+----
  1 | tdsql_pg     |                  |  7
  2 | pg'",    xc% | 2017-10-28 18:24:05.643102 |  3
  3 | pgxz       | 2017-10-28 18:24:05.645691 |
(3 rows)

postgres=# \! cat /data/pgxz/t.csv
1%tdsql_pg%%7
2%"pg'"",    xc%"%2017-10-28 18:24:05.643102%3
3%pgxz%2017-10-28 18:24:05.645691%
postgres=# truncate table t;
TRUNCATE TABLE
postgres=# copy t from  '/data/pgxz/t.csv' (format 'csv',delimiter '%');
COPY 3
postgres=# select * from t;
 f1 |    f2    |        f3        | f4
----+----------+--------------------------+----
  1 | tdsql_pg     |                  |  7
  2 | pg'",    xc% | 2017-10-28 18:24:05.643102 |  3
  3 | pgxz       | 2017-10-28 18:24:05.645691 |
(3 rows)
```

NULL 值处理

```
postgres=#  \! cat /data/pgxz/t.txt
1    tdsql_pg  NULL   7
2    pg'",    xc%  2017-10-28 18:24:05.643102    3
3    pgxz   2017-10-28 18:24:05.645691    NULL
postgres=# copy t from '/data/pgxz/t.txt' (null 'NULL');
COPY 3
postgres=# select * from t;
 f1 |    f2    |        f3        | f4
----+----------+--------------------------+----
  1 | tdsql_pg     |                  |  7
  2 | pg'",    xc% | 2017-10-28 18:24:05.643102 |  3
  3 | pgxz       | 2017-10-28 18:24:05.645691 |
```

```
(3 rows)
```

将文件中的 NULL 字符串当成 NULL 值处理，SQL Server 导出来的文件中把 NULL 值替换成字符串 NULL，所以入库时可以这样处理一下，注意字符串是区分大小写，如下面语句导入数据就会出错。

```
postgres=# copy t from '/data/pgxz/t.txt' (null 'null');
ERROR:  invalid input syntax for type timestamp: "NULL"
CONTEXT:  COPY t, line 1, column f3: "NULL"
```

自定义 quote 字符

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

如果不配置 quote 字符则无法导入文件。

```
postgres=# copy t from '/data/pgxz/t.csv' (format 'csv');
ERROR:  unterminated CSV quoted field
CONTEXT:  COPY t, line 4: "2,%pg'",    xc%%%,2017-10-28
18:24:05.643102,3
3,pgxz,2017-10-28 18:24:05.645691,
"
postgres=# copy t from '/data/pgxz/t.csv' (format 'csv',quote '%');
COPY 3
postgres=#

postgres=# copy t from '/data/pgxz/t.csv' (format 'text',quote '%');
ERROR:  COPY quote available only in CSV mode
```

只有 csv 格式导入时才能配置 quote 字符。

自定义 escape 字符

```
postgres=#  \! cat  /data/pgxz/t.csv
1,tdsql_pg,,7
2,"pg'@",    xc%",2017-10-28 18:24:05.643102,3
3,pgxz,2017-10-28 18:24:05.645691,
postgres=#  copy t from '/data/pgxz/t.csv' (format 'csv');
```

```
ERROR:  unterminated CSV quoted field
CONTEXT:  COPY t, line 4: "2,"pg'@",    xc%",2017-10-28
18:24:05.643102,3
3,pgxz,2017-10-28 18:24:05.645691,
"
```

不指定 escape 字符时，系统默认就是双写的 quote 字符，如双倍的“双引号”。

```
postgres=#  copy t from '/data/pgxz/t.csv' (format 'csv',escape '@');
COPY 3
postgres=# select * from t;
 f1 |    f2     |        f3        | f4
----+-----------+--------------------------+----
  1 | tdsql_pg     |                  |  7
  2 | pg'",    xc% | 2017-10-28 18:24:05.643102 |  3
  3 | pgxz       | 2017-10-28 18:24:05.645691 |
(3 rows)

postgres=#
```

csv header 忽略首行

```
postgres=# \! cat /data/pgxz/t.csv;
f1,f2,f3,f4
1,tdsql_pg,,7
2,"pg'"",    xc%",2017-10-28 18:24:05.643102,3
3,pgxz,2017-10-28 18:24:05.645691,
postgres=# copy t from '/data/pgxz/t.csv' (format 'csv');
ERROR:  invalid input syntax for integer: "f1"
CONTEXT:  COPY t, line 1, column f1: "f1"
postgres=# copy t from '/data/pgxz/t.csv' (format 'csv',header true);
COPY 3
postgres=# select * from t;
 f1 |    f2     |        f3        | f4
----+-----------+--------------------------+----
  1 | tdsql_pg     |                  |  7
  2 | pg'",    xc% | 2017-10-28 18:24:05.643102 |  3
  3 | pgxz       | 2017-10-28 18:24:05.645691 |
(3 rows)
```

如果不忽略首行，则系统会把首行当成数据，造成导入失败。

导入 oid 列值

```
postgres=# \! cat /data/pgxz/t.txt
35242  1    tdsql_pg  \N    7
35243  2    pg'",    xc%  2017-10-28 18:24:05.643102    3
35340  3    pgxz   2017-10-28 18:24:05.645691    \N
postgres=# truncate table t;
TRUNCATE TABLE
postgres=# copy t from '/data/pgxz/t.txt' (oids true);
COPY 3
postgres=# select oid,* from t;
 oid   | f1 |    f2    |        f3        | f4
-------+----+----------------+----------------------------+----
 35242 |  1 | tdsql_pg     |               |  7
 35243 |  2 | pg'",    xc% | 2017-10-28 18:24:05.643102 |  3
 35340 |  3 | pgxz       | 2017-10-28 18:24:05.645691 |
(3 rows)
```

使用 FORCE_NOT_NULL 把某列中空值变成长度为0的字符串，而不是 NULL 值

```
postgres=# truncate table t;
TRUNCATE TABLE
postgres=# \! cat '/data/pgxz/t.csv' ;
1,tdsql_pg,,7
2,"pg'"",    xc%",2017-10-28 18:24:05.643102,3
3,pgxz,2017-10-28 18:24:05.645691,
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。

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 | tdsql_pg       |                  |  7
  2 | pg'",     xc% | 2017-10-28 18:24:05.643102 |  3
  3 | pgxz       | 2017-10-28 18:24:05.645691 |
  4 |          | 2017-10-30 16:41:09.157612 |  4
(4 rows)
```

不指定导入文件的编码格式，则无法正确导入中文字符。

```
 postgres=# truncate table t;
 TRUNCATE TABLE
 postgres=# copy t from '/data/pgxz/t.txt' (encoding gbk) ;
 COPY 4
 postgres=# select * from t;
  f1 |     f2     |        f3        | f4
 ----+----------------+--------------------------+----
   1 | tdsql_pg       |                  |  7
   2 | pg'",     xc% | 2017-10-28 18:24:05.643102 |  3
   3 | pgxz       | 2017-10-28 18:24:05.645691 |
   4 | 腾讯       | 2017-10-30 16:41:09.157612 |  4
 (4 rows)
```

使用 encoding gbk 后便可以正确导入文件的内容，您也可以使用下面的方式转换导入文件的编码后再导入数据。

```
postgres=# truncate table t;
TRUNCATE TABLE
postgres=# \! enconv -L zh_CN -x UTF-8 /data/pgxz/t.txt
postgres=# copy t from '/data/pgxz/t.txt';
COPY 4
postgres=# select * from t;
 f1 |    f2     |        f3        | f4
----+-----------+--------------------------+----
  1 | tdsql_pg  |                  |  7
  2 | pg'",  xc% | 2017-10-28 18:24:05.643102 |  3
  3 | pgxz      | 2017-10-28 18:24:05.645691 |
  4 | 腾讯      | 2017-10-30 16:41:09.157612 |  4
(4 rows)
```

json 和 jsonb 的使用

最近更新时间：2022-09-02 16:52:48

TDSQL PostgreSQL版 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 数据类型用来存储 JSON（JavaScript Object Notation）数据。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，这也是其优势。

json 应用

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
```

插入数据

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

通过键获得 json 对象域

```
postgres=# select f_json ->'col2' as col2 ,f_json -> 'col3' as col3 from
t_json;
 col2   |   col3
```

```
---------+--------
 "tdsql_pg" |
 "tdsql_pg" | "pgxz"
(2 rows)
```

以文本形式获取对象值

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

postgres=# select f_json ->>'col2' as col2 ,f_json ->> 'col3' as col3
from t_json where f_json ->> 'col3' is not null;
 col2   | col3
-------+------
 tdsql_pg | pgxz
(1 row)
```

jsonb 应用

创建 jsonb 类型字段表

```
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=#
```

插入数据

```
postgres=# insert into t_jsonb values(1,'{"col1":1,"col2":"tdsql_pg"}');
INSERT 0 1
postgres=# insert into t_jsonb
values(2,'{"col1":1,"col2":"tdsql_pg","col3":"pgxz"}');
INSERT 0 1

postgres=# select * from t_jsonb;
```

```
 id |           f_jsonb
----+---------------------------------------------
  1 | {"col1": 1, "col2": "tdsql_pg"}
  2 | {"col1": 1, "col2": "tdsql_pg", "col3": "pgxz"}
(2 rows)
```

jsonb 插入时会移除重复的键，如下所示。

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

更新数据

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":"tdsql_pg"}'::jsonb where id=3;
UPDATE 1

postgres=# select * from t_jsonb;
 id |           f_jsonb
----+---------------------------------------------
  2 | {"col1": 1, "col2": "tdsql_pg", "col3": "pgxz"}
  1 | {"col1": 1, "col2": "tdsql_pg", "col3": "pgxz"}
  3 | {"col1": 1, "col2": "tdsql_pg"}
(3 rows)
```

删除某个键。

```
postgres=# update t_jsonb set f_jsonb = f_jsonb - 'col3';
UPDATE 3

postgres=# select * from t_jsonb;
 id |       f_jsonb
----+------------------------------
 2 | {"col1": 1, "col2": "tdsql_pg"}
 1 | {"col1": 1, "col2": "tdsql_pg"}
 3 | {"col1": 1, "col2": "tdsql_pg"}
(3 rows)
```

jsonb_set() 函数更新数据

```
jsonb_set(target jsonb, path text[], new_value jsonb, [create_missing boolean])
```

> **说明**
> target 指要更新的数据源，path 指路径，new_value 指更新后的键值，create_missing 值为 true 表示如果键不存在则添加，create_missing 值为 false 表示如果键不存在则不添加。

```
postgres=# update t_jsonb set f_jsonb = jsonb_set( f_jsonb , '{col}' ,
'"pgxz"' , true ) where id=1;
UPDATE 1
postgres=# update t_jsonb set f_jsonb = jsonb_set( f_jsonb , '{col}' ,
'"pgxz"' , false ) where id=2;
UPDATE 1
postgres=# update t_jsonb set f_jsonb = jsonb_set( f_jsonb , '{col2}' ,
'"pgxz"' , false ) where id=3;
UPDATE 1
postgres=# select * from t_jsonb;
 id |           f_jsonb
----+-----------------------------------------
 1 | {"col": "pgxz", "col1": 1, "col2": "tdsql_pg"}
 2 | {"col1": 1, "col2": "tdsql_pg"}
 3 | {"col1": 1, "col2": "pgxz"}
(3 rows)
```

jsonb 函数应用

jsonb_each() 将 json 对象转变键和值

```
postgres=# select  f_jsonb  from t_jsonb where id=1;
          f_jsonb
-----------------------------------------------
 {"col": "pgxz", "col1": 1, "col2": "tdsql_pg"}
(1 row)

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

jsonb_each_text() 将 json 对象转变文本类型的键和值

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

row_to_json() 将一行记录变成一个 json 对象

```
postgres=# \d+ tdsql_pg
                  Table "public.tdsql_pg"
 Column  |  Type   | Collation | Nullable | Default | Storage  | Stats
target | Description
---------+---------+-----------+----------+---------+----------+-------
-------+-------------
 id      | integer |           | not null |         | plain    |
 nickname | text   |           |          |         | extended |
Indexes:
  "tdsql_pg_pkey" PRIMARY KEY, btree (id)
Distribute By: SHARD(id)
Location Nodes: ALL DATANODES
```

```
postgres=# select * from tdsql_pg;
 id | nickname
----+----------
  1 | tdsql_pg
  2 | pgxz
(2 rows)

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

json_object_keys() 返回一个对象中所有的键

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

jsonb 索引使用

tdsql_pg 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。

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
```

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
```

进阶开发
事务控制

最近更新时间：2024-10-08 15:27:51

事务概述

事务是一组数据库操作的集合，这些操作作为一个整体被执行，要么全部成功，要么全部失败，数据库会从一个一致性状态转换到下一个一致性状态。即使在系统故障的情况下，也能保证数据的一致性。
如果事务中的某个操作失败，那么整个事务将被回滚，数据库状态将恢复到事务开始之前的状态。
TDSQL PG 事务系统通过使用多版本并发控制（MVCC）和可串行化快照隔离（SSI）等技术，提供了高效的并发控制和严格的事务隔离，确保并发执行的事务不会互相干扰。在 MVCC 中查询数据获取的锁不会与写入数据获取的锁发生冲突，因此读取不会阻塞写入，写入也不会阻塞读取，可以为用户提供高性能并发读写能力。

事务隔离级别

TDSQL PG 支持四种事务隔离级别，这些级别定义了一个事务可能会看到其他并发事务所做更改的程度。这些级别从低到高分别是：

- Read Uncommitted（读未提交）：这是最低的隔离级别，一个事务可能会看到其他事务未提交的更改。
- Read Committed（读已提交）：这是 TDSQL PG 默认隔离级别。在这个级别，一个事务能看到在该事务开始之前已经提交事务所做的更改，后续的查询也会看到其他事务所提交新的更改。
- Repeatable Read（可重复读）：在这个级别，一个事务在其整个过程中看到的是一个一致的快照。也就是说，它只能看到在该事务开始时已经提交的事务所做的更改，而不能看到在该事务进行期间其他事务所提交的更改。
- Serializable（可串行化）：这是最高的隔离级别。它提供了最严格的隔离级别，确保事务序列化执行，即使在并发执行时也能得到相同的结果。这个级别可以防止所有类型的并发问题。

锁

TDSQL PG 还提供了表级和行级锁定，为应用提供显式管理互斥场景的能力；TDSQL PG 支持咨询锁，提供了跨事务的锁机制；整体上为应用提供了更高的灵活度。

表级锁

TDSQL PG 提供了多种锁模式来控制对表中数据的并发访问。在多版本并发控制（MVCC）无法提供所需行为的情况下，这些模式可以用于应用程序显式执行锁定动作。注意，TDSQL PG 支持的命令会自动获取合适的锁，确保在命令执行期间操作的表不会被删除等情况发生（例如 TRUNCATE 不能与其他命令同时操作一张表，所以 TRUNCATE 需要拿到 ACCESS EXCLUSIVE 模式锁）。
Conflicting Lock Modes

Requested Lock/Existing Lock	ACCESS SHARE	ROW SHARE	ROW EXCL.	SHARE UPDATE EXCL.	SHARE	SHARE ROW EXCL.	EXCL.	ACCESS EXCL.
`ACCESS SHARE`								X
`ROW SHARE`							X	X
`ROW EXCL.`					X	X	X	X
`SHARE UPDATE EXCL.`				X	X	X	X	X
`SHARE`			X	X		X	X	X
`SHARE ROW EXCL.`			X	X	X	X	X	X
`EXCL.`		X	X	X	X	X	X	X
`ACCESS EXCL.`	X	X	X	X	X	X	X	X

行级锁

TDSQL PG 的行级锁主要用于控制对单数据行的并发访问。行级锁通常在事务中使用，以确保在事务执行期间，其他事务不能修改被锁定的行。

TDSQL PG 提供了以下几种行级锁模式：

1. FOR UPDATE：这种锁定模式在读取数据行时获取一个排他锁。一旦一个事务获取了这种锁，其他事务就不能获取同一行的 FOR UPDATE、FOR NO KEY UPDATE、FOR SHARE 或 FOR KEY SHARE 锁，直到第一个事务完成。
2. FOR NO KEY UPDATE：这种锁定模式类似于 FOR UPDATE，但它允许其他事务获取 FOR KEY SHARE 锁。这对于需要防止其他事务修改数据行，但允许它们并发读取数据行的情况很有用。
3. FOR SHARE：这种锁定模式在读取数据行时获取一个共享锁。这意味着其他事务可以获取同一行的 FOR SHARE 或 FOR KEY SHARE 锁，但不能获取 FOR UPDATE 或 FOR NO KEY UPDATE 锁。
4. FOR KEY SHARE：这是最轻量级的行级锁定模式。它允许其他事务获取 FOR NO KEY UPDATE、FOR SHARE 或 FOR KEY SHARE 锁。

咨询锁

咨询锁（Advisory Locks）是 TDSQL PG 提供的一种特殊类型的锁，它们不由系统自动管理，完全提供给应用程序使用。这种锁的主要用途是在应用程序级别实现复杂的业务规则或协调多个事务的行为。
Advisory Lock 有两种级别：会话级别和事务级别。

1. 会话级别的 Advisory Lock：这种锁在当前会话结束时自动释放，或者可以通过调用 unlock 函数手动释放。如果在一个会话中获取了一个 Advisory Lock，那么只有这个会话可以看到这个锁，其他会话无法看到。
2. 事务级别的 Advisory Lock：这种锁在当前事务结束时自动释放，无论事务是提交还是回滚。这种锁只在当前事务中可见。

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
```

提交事务

进程#1访问。

```
postgres=# begin;
BEGIN
postgres=# delete from tdsql_pg where id=5;
DELETE 1
postgres=#
postgres=# select * from tdsql_pg order by id;
 id |   nickname
----+---------------
  1 | hello tdsql_pg
  2 | tdsql_pg好
  3 | tdsql_pg好
  4 | tdsql_pg default
```

TDSQL PostgreSQL 版完全支持 ACID 特性，没提交前开启另一个连接查询，会看到是5条记录，这是 TDSQL PostgreSQL 版隔离性和多版本视图的实现，如下所示。
进程#2访问。

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

进程#1提交数据。

```
postgres=# commit;
COMMIT
postgres=#
```

进程#2再查询数据，这时已经能看到提交的数据，这个级别叫“读已提交”。

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

回滚事务

```
postgres=# begin;
BEGIN
postgres=# delete from tdsql_pg where id in (3,4);
DELETE 2
postgres=# select * from tdsql_pg;
 id |   nickname
----+-------------
  1 | hello tdsql_pg
```

```
 2 | tdsql_pg好
(2 rows)

postgres=# rollback;
ROLLBACK
```

Rollback 后数据又回来了。

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

事务读一致性 REPEATABLE READ

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,'tdsql_pg');
 INSERT 0 1
 postgres=# begin isolation level repeatable read ;
 BEGIN
 postgres=# select * from t_repeatable_read ;
  id |  mc
 ----+-------
  1 | tdsql_pg
 (1 row)
```

#session2

```
postgres=# insert into t_repeatable_read values(1,'pgxz');
INSERT 0 1
postgres=# select * from t_repeatable_read;
```

```
 id |  mc
----+-------
 1 | tdsql_pg
 1 | pgxz
(2 rows)
```

#session1

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

postgres=#
```

行锁在事务中的运用

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,'tdsql_pg'),(2,'pgxz');
INSERT 0 2
postgres=# select * from t_row_lock;
 id |  mc
----+-------
 1 | tdsql_pg
 2 | pgxz
(2 rows)
```

直接 update 获取

#session1

```
postgres=# begin;
BEGIN
postgres=# set lock_timeout to 1;
```

```
SET
postgres=# update t_row_lock set mc='postgres' where mc='pgxz';
UPDATE 1
postgres=#
```

#session2

```
 postgres=# begin;
 BEGIN
 postgres=# set lock_timeout to 1;
 SET
 postgres=#  update t_row_lock set mc='postgresql' where mc='tdsql_pg';
 UPDATE 1
 postgres=#
```

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

select...for update 获取

#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='pgxz' for update;
 id |  mc
----+------
 2 | pgxz
```

```
(1 row)
```

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

与 MySQL 获取行级锁的区别

```
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='tdsql_pg' for update;
ERROR 1205 (HY000): Lock wait timeout exceeded; try restarting
transaction
mysql>
```

这是因为 MySQL 要使用行级锁需要有索引来配合使用，如下所示，使用 ID 主键来获取行锁。

#session1

```
mysql> begin;
Query OK, 0 rows affected (0.00 sec)

mysql> select * from t_row_lock where id=1 for update;
```

```
+----+-------+
| id | mc    |
+----+-------+
|  1 | tdsql_pg |
+----+-------+
1 row in set (0.00 sec)
```

#session2

```
mysql> begin;
Query OK, 0 rows affected (0.00 sec)

mysql> select * from t_row_lock where id=2 for update;
+----+------+
| id | mc   |
+----+------+
|  2 | pgxz |
+----+------+
1 row in set (0.00 sec)
```

自治事务

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

示例：

```
create table t3(f1 int);

do
$$
begin
     insert into t3 values(1);
     insert into t3 values(2);
     commit;
     insert into t3 values(3);
     rollback;
end;
$$ ;
create or replace procedure demo11(a_varchar in varchar)
as
$$
begin
```

```
    insert into t3 values(1);
    insert into t3 values(2);
    commit;
    insert into t3 values(3);
    rollback;
end;
$$
LANGUAGE plpgsql ;
call demo11(null);
select * from t3;
```

开发规范

最近更新时间：2023-08-09 15:02:01

命名规范

1. DB object：database， schema， table， column， view， index， sequence， function， trigger 等名称：
   - 建议使用小写字母、数字、下划线的组合。
   - 建议不使用双引号即"包围，除非必须包含大写字母或空格等特殊字符。
   - 长度不能超过63个字符。
   - 不建议以 `pg_` 开头或者 `pgxc_`（避免与系统 DB object 混淆），不建议以数字开头。
   - 禁止使用 SQL 关键字，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`，唯一索引为 `表名_列名_uidx`，如 student_name_idx，student_id_uidx。

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。

Constraints 设计

1. 建议每个 table 都使用 shard key 作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。

Index 设计

1. TDSQL PostgreSQL版 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，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）且选择时数据占比低的 query，建议带 where 的 Partial Indexes。

```
select * from test where status=1 and col=?; -- 其中status=1为固定的条件

create index on test (col) where status=1;
```

7. 建议对经常使用表达式作为查询条件的 query，可以使用表达式或函数索引加速 query。

```
select * from test where exp(xxx);

create index on test ( exp(xxx) );
```

8. 建议不要建过多 index，一般不要超过6个，核心 table（产品，订单）可适当增加 index 个数。

关于 NULL

1. NULL 的判断：IS NULL，IS NOT NULL。
2. 注意 boolean 类型取值 true，false，NULL。
3. 注意 NOT IN 集合中带有 NULL 元素。

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

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

postgres=# select id, nickname||null from tdsql_pg limit 1;
 id | ?column?
----+----------
  1 |
(1 row)
postgres=# select id, nickname||coalesce(null, '') from tdsql_pg
limit 1;
 id |  ?column?
----+-------------
  1 | hello tdsql_pg
(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 tdsql_pg ;
 id |  nickname
----+---------------
  1 | hello tdsql_pg
  2 | tdsql_pg好
  5 |
  3 | tdsql_pg好
  4 | tdsql_pg default
(5 rows)

postgres=# select count(1) from tdsql_pg;
 count
```

```
\-------
      5
(1 row)

postgres=# select count(*) from tdsql_pg;
 count
\-------
      5
(1 row)

postgres=# select count(nickname) from tdsql_pg;
 count
\-------
      4
(1 row)

postgres=# select count((id, nickname)) from tdsql_pg;
 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 tdsql_pg;
 count
\-------
      3
(1 row)

postgres=# select count(distinct (id, nickname)) from tdsql_pg;
 count
\-------
      5
(1 row)
```

7. 两个 NULL 的对比方法。

```
postgres=# select null is not  distinct from null as tdsql_pgnull;
 tdsql_pgnull
\-----------
 t
(1 row)
```

开发相关规范

- **建议对 DB object 尤其是 COLUMN 加 COMMENT，便于后续了解业务及维护**
  注释前后的数据表可读性对比，有注释的一看就明白。

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

postgres=# comment on column tdsql_pg_main.id is 'id号';
COMMENT
postgres=# comment on column tdsql_pg_main.mc is '产品名称';
COMMENT
postgres=# \d+ tdsql_pg_main
           Table "public.tdsql_pg_main"
 Column |  Type   | Modifiers | Storage  | Stats target | Description
--------+---------+-----------+----------+--------------+-------------
 id     | integer |           | plain    |              | id号
 mc     | text    |           | extended |              | 产品名称
Indexes:
  "tdsql_pg_main_id_uidx" UNIQUE, btree (id)
Has OIDs: no
Distribute By SHARD(id)
    Location Nodes: ALL DATANODES
```

- **建议非必须时避免 `select *`，只取所需字段，以减少包括不限于网络带宽消耗**

```
 postgres=#  explain (verbose) select * from tdsql_pg_main where id=1;
                        QUERY PLAN
 ------------------------------------------------------------------------
------------------------
  Index Scan using tdsql_pg_main_id_uidx on public.tdsql_pg_main
(cost=0.15..8.17 rows=1 width=36)
   Output: id,  mc
   Index Cond: (tdsql_pg_main.id = 1)
 (3 rows)

 postgres=#  explain (verbose) select tableoid from tdsql_pg_main
where id=1;
                        QUERY PLAN
 ------------------------------------------------------------------------
----------------------
  Index Scan using tdsql_pg_main_id_uidx on public.tdsql_pg_main
(cost=0.15..8.17 rows=1 width=4)
   Output: tableoid
   Index Cond: (tdsql_pg_main.id = 1)
 (3 rows)
```

`*` 是返回36个字符，而另一个一条记录只能4个字段的长度。

- **建议 update 时尽量做 <> 判断，如 update table_a set column_b = c where column_b <> c;**

```
 postgres=# update tdsql_pg_main set mc='tdsql_pg' ;
 UPDATE 1
 postgres=# select xmin, * from tdsql_pg_main;
  xmin | id |  mc
 ------+----+-------
  2562 |  1 | tdsql_pg
 (1 row)

 postgres=# update tdsql_pg_main set mc='tdsql_pg' ;
 UPDATE 1
 postgres=# select xmin, * from tdsql_pg_main;
  xmin | id |  mc
 ------+----+-------
  2564 |  1 | tdsql_pg
 (1 row)

 postgres=# update tdsql_pg_main set mc='tdsql_pg' where
mc!='tdsql_pg';
```

```
 UPDATE 0
postgres=# select xmin, * from tdsql_pg_main;
 xmin | id |  mc
------+----+-------
 2564 |  1 | tdsql_pg
(1 row)
```

上面的效果是一样的，但带条件的更新不会产生一个新的版本记录，不需要系统执行 vacuum 回收垃圾数据。

- **建议将单个事务的多条 SQL 操作，分解、拆分，或者不放在一个事务里，让每个事务的粒度尽可能小，尽量 lock 少的资源，避免lock 、dead lock 的产生**

\#session1 把所有数据都更新而不提交，一下锁了2000千万条记录。

```
postgres=# begin;
BEGIN
postgres=# update tdsql_pg_main set mc='tdsql_pg_1.3';
UPDATE 200000000
```

\#session2 等待

```
postgres=# update tdsql_pg_main set mc='tdsql_pg_1.4'  where id=1;
```

\#session3 等待

```
postgres=# update tdsql_pg_main set mc='tdsql_pg_1.5'  where id=2;
```

如果 #session1 分批更新的话，如下所示。

```
postgres=# begin;
BEGIN
postgres=# update tdsql_pg_main set mc='tdsql_pg_1.3' where id>0 and
id <=100000;
UPDATE 100000
postgres=#COMMIT;

postgres=# begin;
BEGIN
postgres=# update tdsql_pg_main set mc='tdsql_pg_1.3' where id>100000
and id <=200000;
UPDATE 100000
postgres=#COMMIT;
```

则 session2 和 session3 中就能部分提前完成，这样可以避免大量的锁等待和出现大量的 session 占用系统资源，在做全表更新时请使用这种方法来执行。

- **建议大批量的数据入库时，使用 copy，不建议使用 insert，以提高写入速度**

```
 postgres=# insert into tdsql_pg_main select
t, 'xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx' from generate_series(1,
100000) as t;
 INSERT 0 100000
 Time: 9511.755 ms

 postgres=# copy  tdsql_pg_main to '/data/pgxz/tdsql_pg_main.txt';
 COPY 100002
 Time: 179.428 ms

 postgres=# copy  tdsql_pg_main from  '/data/pgxz/tdsql_pg_main.txt';
 COPY 100002
 Time: 1625.803 ms
 postgres=#
```

性能相差5倍。

- **建议对报表类的或生成基础数据的查询，使用物化视图 (MATERIALIZED VIEW) 定期固化数据快照，避免对多表（尤其是读写频繁的表）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 tdsql_pg_main 的总记录数，可参考如下操作。

```
postgres=# select count(1) from tdsql_pg_main;
 count
--------
 200004
(1 row)

Time: 27.948 ms

postgres=# create MATERIALIZED VIEW tdsql_pg_main_count as select
count(1) as num from tdsql_pg_main;
SELECT 1
Time: 322.372 ms
postgres=# select num from  tdsql_pg_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  tdsql_pg_main from  '/data/pgxz/tdsql_pg_main.txt';
COPY 100002
Time: 1201.774 ms
postgres=# select count(1) from tdsql_pg_main;
 count
--------
 300006
(1 row)

Time: 23.164 ms
postgres=# REFRESH MATERIALIZED VIEW tdsql_pg_main_count;
REFRESH MATERIALIZED VIEW
Time: 49.486 ms
postgres=# select num from tdsql_pg_main_count ;
 num
--------
 300006
(1 row)

Time: 0.301 ms
```

- **建议复杂的统计查询可以尝试窗口函数**

  请参考 高级 SQL 语句编写 中窗口函数的用法。

- **两表 join 时尽量的使用分布 key 进行 join**

  在建立业务的主表，明细表时，就需要使用他们的关联键来做分布键，如下所示：

```
 [pgxz@VM_0_29_centos pgxz]$ psql -p 15001
 psql (PostgreSQL 10 (tdsql_pg 2.01))
 Type "help" for help.

 postgres=# create table tdsql_pg_main(id integer, mc text) distribute
by shard(id);
 CREATE TABLE
 postgres=# create table tdsql_pg_detail(id integer, tdsql_pg_main_id
integer, mc text) distribute by shard(tdsql_pg_main_id);
 CREATE TABLE
```

```
 postgres=# explain select tdsql_pg_detail.* from tdsql_pg_main,
tdsql_pg_detail where
tdsql_pg_main.id=tdsql_pg_detail.tdsql_pg_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 tdsql_pg_detail.* from
tdsql_pg_main, tdsql_pg_detail where
tdsql_pg_main.id=tdsql_pg_detail.tdsql_pg_main_id;
                                            QUERY PLAN
 ------------------------------------------------------------------------
------------------------------------------------------------------------
----------------------------------------
   Data Node Scan on "__REMOTE_FQS_QUERY__"  (cost=0.00..0.00 rows=0
width=0)
    Output: tdsql_pg_detail.id,  tdsql_pg_detail.tdsql_pg_main_id,
tdsql_pg_detail.mc
    Node/s: dn001,  dn002
    Remote query: SELECT tdsql_pg_detail.id,
tdsql_pg_detail.tdsql_pg_main_id,  tdsql_pg_detail.mc FROM
public.tdsql_pg_main,  public.tdsql_pg_detail WHERE (tdsql_pg_main.id
= tdsql_pg_detail.tdsql_pg_main_id)
 (4 rows)

 postgres=#
```

- **分布键用唯一索引代替主键**

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

因为唯一索引后期的维护成本比主键要低很多。

- **分布键无法建立唯一索引则要建立普通索引，提高查询的效率**

```
 postgres=# create index tdsql_pg_detail_tdsql_pg_main_id_idx on
tdsql_pg_detail using btree(tdsql_pg_main_id);
```

```
CREATE INDEX
```

这样两表在 join 查询时返回少量数据时的效率才会高。

- **不要对字段建立外键**
  目前 TDSQL PostgreSQL版 还不支持多 DN 外键约束。

高级 SQL 语句编写

最近更新时间：2025-03-18 15:34:12

窗口函数的使用

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));
```

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 | 5 0铲车，后八轮翻斗车   | 河南省    | 三门峡市  | 2015-10-
05 07:53:13 |  1030.9
          3 |  4 | 普货                    | 福建省    | 三明市    | 2015-
10-05 15:19:17 | 1316.27
          4 |  1 | 衣服                    | 海南省    | 三亚市    | 2015-
10-05 09:32:01 | 1915.86
          5 |  2 | 建筑设备                | 福建省    | 三明市    | 2015-10-
05 07:21:22 | 2022.31
          6 |  7 | 鲜香菇2000斤            | 河南省    | 三门峡市  | 2015-10-
05 10:38:29 | 4182.68
          7 |  8 | 旋挖附件38吨            | 河南省    | 三门峡市  | 2015-10-
05 10:48:38 | 5365.04
          8 | 10 | 旋挖附件39吨            | 河南省    | 三门峡市  | 2015-10-
05 11:38:38 |  8290.5
          9 |  3 | 设备                    | 福建省    | 三明市    | 2015-
10-05 11:21:54 | 8771.11
         10 |  9 | 旋挖附件35吨            | 河南省    | 三门峡市  | 2015-10-
05 10:48:38 | 9621.37
         11 |  5 | 普货40吨需13米半挂一辆  | 上海市    | 上海市    | 2015-10-
05 08:13:59 | 9886.15
(11 rows)
```

row_number() 返回行号，按 begincity 分组，pubtime 排序

```
postgres=# select row_number() over(partition by begincity order by
pubtime),* from bills;
 row_number | id |        goodsdesc         | beginunit | begincity |
pubtime        | amount
------------+----+--------------------------+-----------+-----------+-----
---------------+---------
          1 |  1 | 衣服                     | 海南省    | 三亚市    | 2015-
10-05 09:32:01 | 1915.86
          1 |  2 | 建筑设备                 | 福建省    | 三明市    | 2015-10-
05 07:21:22 | 2022.31
          2 |  3 | 设备                     | 福建省    | 三明市    | 2015-
10-05 11:21:54 | 8771.11
          3 |  4 | 普货                     | 福建省    | 三明市    | 2015-
10-05 15:19:17 | 1316.27
          1 |  6 | 5 0铲车，后八轮翻斗车    | 河南省    | 三门峡市  | 2015-10-
05 07:53:13 |  1030.9
          2 |  7 | 鲜香菇2000斤             | 河南省    | 三门峡市  | 2015-10-
05 10:38:29 | 4182.68
          3 |  8 | 旋挖附件38吨             | 河南省    | 三门峡市  | 2015-10-
05 10:48:38 | 5365.04
          4 |  9 | 旋挖附件35吨             | 河南省    | 三门峡市  | 2015-10-
05 10:48:38 | 9621.37
          5 | 10 | 旋挖附件39吨             | 河南省    | 三门峡市  | 2015-10-
05 11:38:38 |  8290.5
          1 | 11 | 设备                     | 上海市    | 上海市    | 2015-
10-05 07:59:35 |  971.54
          2 |  5 | 普货40吨需13米半挂一辆   | 上海市    | 上海市    | 2015-10-
05 08:13:59 | 9886.15
(11 rows)
```

rank() 返回行号，对比值重复时行号重复并间断，即返回1,2,2,4...

```
postgres=# select rank() over(partition by begincity order by pubtime),*
from bills;
 rank | id |        goodsdesc         | beginunit | begincity |
pubtime        | amount
------+----+--------------------------+-----------+-----------+-----------
----------+---------
    1 |  1 | 衣服                     | 海南省    | 三亚市    | 2015-10-05
09:32:01 | 1915.86
    1 |  2 | 建筑设备                 | 福建省    | 三明市    | 2015-10-05
07:21:22 | 2022.31
```

```
    2 |  3 | 设备                        | 福建省     | 三明市     | 2015-10-05
11:21:54 | 8771.11
    3 |  4 | 普货                        | 福建省     | 三明市     | 2015-10-05
15:19:17 | 1316.27
    1 |  6 | 5 0铲车，后八轮翻斗车       | 河南省     | 三门峡市   | 2015-10-05
07:53:13 |  1030.9
    2 |  7 | 鲜香菇2000斤                | 河南省     | 三门峡市   | 2015-10-05
10:38:29 | 4182.68
    3 |  8 | 旋挖附件38吨                | 河南省     | 三门峡市   | 2015-10-05
10:48:38 | 5365.04
    3 |  9 | 旋挖附件35吨                | 河南省     | 三门峡市   | 2015-10-05
10:48:38 | 9621.37
    5 | 10 | 旋挖附件39吨                | 河南省     | 三门峡市   | 2015-10-05
11:38:38 |  8290.5
    1 | 11 | 设备                        | 上海市     | 上海市     | 2015-10-05
07:59:35 |  971.54
    2 |  5 | 普货40吨需13米半挂一辆      | 上海市     | 上海市     | 2015-10-05
08:13:59 | 9886.15
(11 rows)
```

dense_rank() 返回行号，对比值重复时行号重复但不间断，即返回1,2,2,3...

```
postgres=# select dense_rank() over(partition by begincity order by
pubtime),* from bills;
 dense_rank | id |        goodsdesc         | beginunit | begincity |
pubtime        | amount
------------+----+--------------------------+-----------+-----------+-----
---------------+---------
          1 |  1 | 衣服                     | 海南省    | 三亚市    | 2015-
10-05 09:32:01 | 1915.86
          1 |  2 | 建筑设备                 | 福建省    | 三明市    | 2015-10-
05 07:21:22 | 2022.31
          2 |  3 | 设备                     | 福建省    | 三明市    | 2015-
10-05 11:21:54 | 8771.11
          3 |  4 | 普货                     | 福建省    | 三明市    | 2015-
10-05 15:19:17 | 1316.27
          1 |  6 | 5 0铲车，后八轮翻斗车    | 河南省    | 三门峡市  | 2015-10-
05 07:53:13 |  1030.9
          2 |  7 | 鲜香菇2000斤             | 河南省    | 三门峡市  | 2015-10-
05 10:38:29 | 4182.68
          3 |  8 | 旋挖附件38吨             | 河南省    | 三门峡市  | 2015-10-
05 10:48:38 | 5365.04
```

```
          3 |  9 | 旋挖附件35吨              | 河南省     | 三门峡市   | 2015-10-
05 10:48:38 | 9621.37
          4 | 10 | 旋挖附件39吨              | 河南省     | 三门峡市   | 2015-10-
05 11:38:38 |  8290.5
          1 | 11 | 设备                      | 上海市     | 上海市     | 2015-
10-05 07:59:35 |  971.54
          2 |  5 | 普货40吨需13米半挂一辆    | 上海市     | 上海市     | 2015-10-
05 08:13:59 | 9886.15
(11 rows)
```

percent_rank() 从当前开始，计算在分组中的比例 (行号 – 1) * (1 / (总记录数 – 1))

```
postgres=# select percent_rank() over(partition by begincity order by
id),* from bills;
 percent_rank | id |        goodsdesc         | beginunit | begincity |
pubtime        | amount
--------------+----+--------------------------+-----------+-----------+---
-----------------+---------
            0 |  1 | 衣服                     | 海南省    | 三亚市    | 2015-
10-05 09:32:01 | 1915.86
            0 |  2 | 建筑设备                 | 福建省    | 三明市    | 2015-
10-05 07:21:22 | 2022.31
          0.5 |  3 | 设备                     | 福建省    | 三明市    | 2015-
10-05 11:21:54 | 8771.11
            1 |  4 | 普货                     | 福建省    | 三明市    | 2015-
10-05 15:19:17 | 1316.27
            0 |  6 | 5 0铲车，后八轮翻斗车    | 河南省    | 三门峡市  | 2015-
10-05 07:53:13 |  1030.9
         0.25 |  7 | 鲜香菇2000斤             | 河南省    | 三门峡市  | 2015-
10-05 10:38:29 | 4182.68
          0.5 |  8 | 旋挖附件38吨             | 河南省    | 三门峡市  | 2015-
10-05 10:48:38 | 5365.04
         0.75 |  9 | 旋挖附件35吨             | 河南省    | 三门峡市  | 2015-
10-05 10:48:38 | 9621.37
            1 | 10 | 旋挖附件39吨             | 河南省    | 三门峡市  | 2015-
10-05 11:38:38 |  8290.5
            0 |  5 | 普货40吨需13米半挂一辆   | 上海市    | 上海市    | 2015-
10-05 08:13:59 | 9886.15
            1 | 11 | 设备                     | 上海市    | 上海市    | 2015-
10-05 07:59:35 |  971.54
(11 rows)
```

cume_dist() 返回行数除以记录数值

```
postgres=# select ROUND((cume_dist() over(partition by begincity order
by id))::NUMERIC,2) AS cume_dist,* from bills;
 cume_dist | id |       goodsdesc         | beginunit | begincity |
pubtime        | amount
-----------+----+-------------------------+-----------+-----------+------
---------------+---------
      1.00 |  1 | 衣服                    | 海南省    | 三亚市    | 2015-10-
05 09:32:01 | 1915.86
      0.33 |  2 | 建筑设备                | 福建省    | 三明市    | 2015-10-
05 07:21:22 | 2022.31
      0.67 |  3 | 设备                    | 福建省    | 三明市    | 2015-10-
05 11:21:54 | 8771.11
      1.00 |  4 | 普货                    | 福建省    | 三明市    | 2015-10-
05 15:19:17 | 1316.27
      0.20 |  6 | 5 0铲车，后八轮翻斗车   | 河南省    | 三门峡市  | 2015-10-05
07:53:13 |  1030.9
      0.40 |  7 | 鲜香菇2000斤            | 河南省    | 三门峡市  | 2015-10-
05 10:38:29 | 4182.68
      0.60 |  8 | 旋挖附件38吨            | 河南省    | 三门峡市  | 2015-10-
05 10:48:38 | 5365.04
      0.80 |  9 | 旋挖附件35吨            | 河南省    | 三门峡市  | 2015-10-
05 10:48:38 | 9621.37
      1.00 | 10 | 旋挖附件39吨            | 河南省    | 三门峡市  | 2015-10-
05 11:38:38 |  8290.5
      0.50 |  5 | 普货40吨需13米半挂一辆  | 上海市    | 上海市    | 2015-10-
05 08:13:59 | 9886.15
      1.00 | 11 | 设备                    | 上海市    | 上海市    | 2015-10-
05 07:59:35 |  971.54
(11 rows)
```

ntile(分组数量) 让所有记录尽可以的均匀分布

```
postgres=# select ntile(2) over(partition by begincity order by id),*
from bills;
 ntile | id |       goodsdesc         | beginunit | begincity |
pubtime        | amount
-------+----+-------------------------+-----------+-----------+----------
-----------+---------
     1 |  1 | 衣服                    | 海南省    | 三亚市    | 2015-10-05
09:32:01 | 1915.86
```

```
     1 |  2 | 建筑设备                  | 福建省     | 三明市     | 2015-10-05
07:21:22 | 2022.31
     1 |  3 | 设备                      | 福建省     | 三明市     | 2015-10-05
11:21:54 | 8771.11
     2 |  4 | 普货                      | 福建省     | 三明市     | 2015-10-05
15:19:17 | 1316.27
     1 |  6 | 5 0铲车，后八轮翻斗车     | 河南省     | 三门峡市   | 2015-10-05
07:53:13 |  1030.9
     1 |  7 | 鲜香菇2000斤              | 河南省     | 三门峡市   | 2015-10-05
10:38:29 | 4182.68
     1 |  8 | 旋挖附件38吨              | 河南省     | 三门峡市   | 2015-10-05
10:48:38 | 5365.04
     2 |  9 | 旋挖附件35吨              | 河南省     | 三门峡市   | 2015-10-05
10:48:38 | 9621.37
     2 | 10 | 旋挖附件39吨              | 河南省     | 三门峡市   | 2015-10-05
11:38:38 |  8290.5
     1 |  5 | 普货40吨需13米半挂一辆    | 上海市     | 上海市     | 2015-10-05
08:13:59 | 9886.15
     2 | 11 | 设备                      | 上海市     | 上海市     | 2015-10-05
07:59:35 |  971.54
(11 rows)

postgres=# select ntile(3) over(partition by begincity order by id),*
from bills;
 ntile | id |        goodsdesc          | beginunit | begincity |
pubtime        | amount
-------+----+---------------------------+-----------+-----------+----------
-----------+---------
     1 |  1 | 衣服                      | 海南省     | 三亚市     | 2015-10-05
09:32:01 | 1915.86
     1 |  2 | 建筑设备                  | 福建省     | 三明市     | 2015-10-05
07:21:22 | 2022.31
     2 |  3 | 设备                      | 福建省     | 三明市     | 2015-10-05
11:21:54 | 8771.11
     3 |  4 | 普货                      | 福建省     | 三明市     | 2015-10-05
15:19:17 | 1316.27
     1 |  6 | 5 0铲车，后八轮翻斗车     | 河南省     | 三门峡市   | 2015-10-05
07:53:13 |  1030.9
     1 |  7 | 鲜香菇2000斤              | 河南省     | 三门峡市   | 2015-10-05
10:38:29 | 4182.68
     2 |  8 | 旋挖附件38吨              | 河南省     | 三门峡市   | 2015-10-05
10:48:38 | 5365.04
```

```
     2 |  9 | 旋挖附件35吨              | 河南省     | 三门峡市   | 2015-10-05
10:48:38 | 9621.37
     3 | 10 | 旋挖附件39吨              | 河南省     | 三门峡市   | 2015-10-05
11:38:38 |  8290.5
     1 |  5 | 普货40吨需13米半挂一辆    | 上海市     | 上海市     | 2015-10-05
08:13:59 | 9886.15
     2 | 11 | 设备                      | 上海市     | 上海市     | 2015-10-05
07:59:35 |  971.54
(11 rows)
```

lag(value any [, offset integer [, default any]]) 返回偏移量值

offset integer 是偏移值，正数时取前值，负数时取后值，没有取到值时用 default 代替。

```
postgres=# select lag(amount,1,null) over(partition by begincity order
by id),* from bills;
   lag    | id |       goodsdesc         | beginunit | begincity |
pubtime        | amount
----------+----+-------------------------+-----------+-----------+--------
-------------+---------
          |  1 | 衣服                    | 海南省     | 三亚市     | 2015-10-05
09:32:01 | 1915.86
          |  2 | 建筑设备                | 福建省     | 三明市     | 2015-10-05
07:21:22 | 2022.31
 2022.31 |  3 | 设备                    | 福建省     | 三明市     | 2015-10-05
11:21:54 | 8771.11
 8771.11 |  4 | 普货                    | 福建省     | 三明市     | 2015-10-05
15:19:17 | 1316.27
          |  6 | 5 0铲车，后八轮翻斗车   | 河南省     | 三门峡市   | 2015-10-05
07:53:13 |  1030.9
  1030.9 |  7 | 鲜香菇2000斤            | 河南省     | 三门峡市   | 2015-10-05
10:38:29 | 4182.68
 4182.68 |  8 | 旋挖附件38吨            | 河南省     | 三门峡市   | 2015-10-05
10:48:38 | 5365.04
 5365.04 |  9 | 旋挖附件35吨            | 河南省     | 三门峡市   | 2015-10-05
10:48:38 | 9621.37
 9621.37 | 10 | 旋挖附件39吨            | 河南省     | 三门峡市   | 2015-10-05
11:38:38 |  8290.5
          |  5 | 普货40吨需13米半挂一辆  | 上海市     | 上海市     | 2015-10-05
08:13:59 | 9886.15
 9886.15 | 11 | 设备                    | 上海市     | 上海市     | 2015-10-05
07:59:35 |  971.54
(11 rows)
```

```
postgres=# select lag(amount,2,0::float8) over(partition by begincity
order by id),* from bills;
   lag    | id |        goodsdesc         | beginunit | begincity |
pubtime        | amount
----------+----+--------------------------+-----------+-----------+--------
-------------+---------
        0 |  1 | 衣服                     | 海南省    | 三亚市    | 2015-10-05
09:32:01 | 1915.86
        0 |  2 | 建筑设备                 | 福建省    | 三明市    | 2015-10-05
07:21:22 | 2022.31
        0 |  3 | 设备                     | 福建省    | 三明市    | 2015-10-05
11:21:54 | 8771.11
 2022.31 |  4 | 普货                     | 福建省    | 三明市    | 2015-10-05
15:19:17 | 1316.27
        0 |  6 | 5 0铲车，后八轮翻斗车    | 河南省    | 三门峡市  | 2015-10-05
07:53:13 |  1030.9
        0 |  7 | 鲜香菇2000斤             | 河南省    | 三门峡市  | 2015-10-05
10:38:29 | 4182.68
  1030.9 |  8 | 旋挖附件38吨             | 河南省    | 三门峡市  | 2015-10-05
10:48:38 | 5365.04
 4182.68 |  9 | 旋挖附件35吨             | 河南省    | 三门峡市  | 2015-10-05
10:48:38 | 9621.37
 5365.04 | 10 | 旋挖附件39吨             | 河南省    | 三门峡市  | 2015-10-05
11:38:38 |  8290.5
        0 |  5 | 普货40吨需13米半挂一辆   | 上海市    | 上海市    | 2015-10-05
08:13:59 | 9886.15
        0 | 11 | 设备                     | 上海市    | 上海市    | 2015-10-05
07:59:35 |  971.54
(11 rows)

postgres=# select lag(amount,-2,0::float8) over(partition by begincity
order by id),* from bills;
   lag    | id |        goodsdesc         | beginunit | begincity |
pubtime        | amount
----------+----+--------------------------+-----------+-----------+--------
-------------+---------
        0 |  1 | 衣服                     | 海南省    | 三亚市    | 2015-10-05
09:32:01 | 1915.86
 1316.27 |  2 | 建筑设备                 | 福建省    | 三明市    | 2015-10-05
07:21:22 | 2022.31
        0 |  3 | 设备                     | 福建省    | 三明市    | 2015-10-05
11:21:54 | 8771.11
```

```
        0 |  4 | 普货                          | 福建省      | 三明市      | 2015-10-05
15:19:17 | 1316.27
 5365.04 |  6 | 5 0铲车，后八轮翻斗车     | 河南省      | 三门峡市    | 2015-10-05
07:53:13 |  1030.9
 9621.37 |  7 | 鲜香菇2000斤                  | 河南省      | 三门峡市    | 2015-10-05
10:38:29 | 4182.68
  8290.5 |  8 | 旋挖附件38吨                  | 河南省      | 三门峡市    | 2015-10-05
10:48:38 | 5365.04
        0 |  9 | 旋挖附件35吨                  | 河南省      | 三门峡市    | 2015-10-05
10:48:38 | 9621.37
        0 | 10 | 旋挖附件39吨                  | 河南省      | 三门峡市    | 2015-10-05
11:38:38 |  8290.5
        0 |  5 | 普货40吨需13米半挂一辆    | 上海市      | 上海市      | 2015-10-05
08:13:59 | 9886.15
        0 | 11 | 设备                          | 上海市      | 上海市      | 2015-10-05
07:59:35 |  971.54
(11 rows)
```

lead(value any [,offset integer [, default any]]) 返回偏移量值

offset integer 是偏移值，正数时取后值，负数时取前值，没有取到值时用 default 代替。

```
postgres=# select lead(amount,2,null) over(partition by begincity order
by id),* from bills;
  lead   | id |       goodsdesc         | beginunit | begincity |
pubtime       | amount
---------+----+-------------------------+-----------+-----------+--------
-------------+---------
         |  1 | 衣服                          | 海南省      | 三亚市      | 2015-10-05
09:32:01 | 1915.86
 1316.27 |  2 | 建筑设备                      | 福建省      | 三明市      | 2015-10-05
07:21:22 | 2022.31
         |  3 | 设备                          | 福建省      | 三明市      | 2015-10-05
11:21:54 | 8771.11
         |  4 | 普货                          | 福建省      | 三明市      | 2015-10-05
15:19:17 | 1316.27
 5365.04 |  6 | 5 0铲车，后八轮翻斗车     | 河南省      | 三门峡市    | 2015-10-05
07:53:13 |  1030.9
 9621.37 |  7 | 鲜香菇2000斤                  | 河南省      | 三门峡市    | 2015-10-05
10:38:29 | 4182.68
  8290.5 |  8 | 旋挖附件38吨                  | 河南省      | 三门峡市    | 2015-10-05
10:48:38 | 5365.04
```

```
          |  9 | 旋挖附件35吨              | 河南省      | 三门峡市   | 2015-10-05
10:48:38 | 9621.37
          | 10 | 旋挖附件39吨              | 河南省      | 三门峡市   | 2015-10-05
11:38:38 |  8290.5
          |  5 | 普货40吨需13米半挂一辆    | 上海市      | 上海市     | 2015-10-05
08:13:59 | 9886.15
          | 11 | 设备                      | 上海市      | 上海市     | 2015-10-05
07:59:35 |  971.54
(11 rows)

postgres=# select lead(amount,-2,null) over(partition by begincity order
by id),* from bills;
  lead    | id |        goodsdesc         | beginunit | begincity |
pubtime        | amount
----------+----+--------------------------+-----------+-----------+--------
-------------+---------
          |  1 | 衣服                     | 海南省      | 三亚市     | 2015-10-05
09:32:01 | 1915.86
          |  2 | 建筑设备                 | 福建省      | 三明市     | 2015-10-05
07:21:22 | 2022.31
          |  3 | 设备                     | 福建省      | 三明市     | 2015-10-05
11:21:54 | 8771.11
 2022.31 |  4 | 普货                     | 福建省      | 三明市     | 2015-10-05
15:19:17 | 1316.27
          |  6 | 5 0铲车，后八轮翻斗车    | 河南省      | 三门峡市   | 2015-10-05
07:53:13 |  1030.9
          |  7 | 鲜香菇2000斤             | 河南省      | 三门峡市   | 2015-10-05
10:38:29 | 4182.68
  1030.9 |  8 | 旋挖附件38吨             | 河南省      | 三门峡市   | 2015-10-05
10:48:38 | 5365.04
 4182.68 |  9 | 旋挖附件35吨             | 河南省      | 三门峡市   | 2015-10-05
10:48:38 | 9621.37
 5365.04 | 10 | 旋挖附件39吨             | 河南省      | 三门峡市   | 2015-10-05
11:38:38 |  8290.5
          |  5 | 普货40吨需13米半挂一辆   | 上海市      | 上海市     | 2015-10-05
08:13:59 | 9886.15
          | 11 | 设备                     | 上海市      | 上海市     | 2015-10-05
07:59:35 |  971.54
(11 rows)
```

first_value(value any) 返回第一值

```
postgres=# select first_value(amount) over(partition by begincity order
by  id),* from bills;
 first_value | id |       goodsdesc         | beginunit | begincity |
pubtime        | amount
-------------+----+-------------------------+-----------+-----------+----
----------------+---------
     1915.86 |  1 | 衣服                    | 海南省    | 三亚市    | 2015-
10-05 09:32:01 | 1915.86
     2022.31 |  2 | 建筑设备                | 福建省    | 三明市    | 2015-
10-05 07:21:22 | 2022.31
     2022.31 |  3 | 设备                    | 福建省    | 三明市    | 2015-
10-05 11:21:54 | 8771.11
     2022.31 |  4 | 普货                    | 福建省    | 三明市    | 2015-
10-05 15:19:17 | 1316.27
      1030.9 |  6 | 5 0铲车，后八轮翻斗车   | 河南省    | 三门峡市  | 2015-10-
05 07:53:13 |  1030.9
      1030.9 |  7 | 鲜香菇2000斤            | 河南省    | 三门峡市  | 2015-
10-05 10:38:29 | 4182.68
      1030.9 |  8 | 旋挖附件38吨            | 河南省    | 三门峡市  | 2015-
10-05 10:48:38 | 5365.04
      1030.9 |  9 | 旋挖附件35吨            | 河南省    | 三门峡市  | 2015-
10-05 10:48:38 | 9621.37
      1030.9 | 10 | 旋挖附件39吨            | 河南省    | 三门峡市  | 2015-
10-05 11:38:38 |  8290.5
     9886.15 |  5 | 普货40吨需13米半挂一辆  | 上海市    | 上海市    | 2015-
10-05 08:13:59 | 9886.15
     9886.15 | 11 | 设备                    | 上海市    | 上海市    | 2015-
10-05 07:59:35 |  971.54
(11 rows)
```

last_value(value any) 返回最后值

```
postgres=# select last_value(amount) over(partition by begincity order
by pubtime),* FROM bills;
 last_value | id |       goodsdesc         | beginunit | begincity |
pubtime        | amount
------------+----+-------------------------+-----------+-----------+-----
----------------+---------
    1915.86 |  1 | 衣服                    | 海南省    | 三亚市    | 2015-
10-05 09:32:01 | 1915.86
    2022.31 |  2 | 建筑设备                | 福建省    | 三明市    | 2015-10-
05 07:21:22 | 2022.31
```

```
    8771.11 |  3 | 设备                    | 福建省     | 三明市     | 2015-
10-05 11:21:54 | 8771.11
    1316.27 |  4 | 普货                    | 福建省     | 三明市     | 2015-
10-05 15:19:17 | 1316.27
     1030.9 |  6 | 5 0铲车，后八轮翻斗车   | 河南省     | 三门峡市   | 2015-10-
05 07:53:13 |  1030.9
    4182.68 |  7 | 鲜香菇2000斤            | 河南省     | 三门峡市   | 2015-10-
05 10:38:29 | 4182.68
    9621.37 |  8 | 旋挖附件38吨            | 河南省     | 三门峡市   | 2015-10-
05 10:48:38 | 5365.04
    9621.37 |  9 | 旋挖附件35吨            | 河南省     | 三门峡市   | 2015-10-
05 10:48:38 | 9621.37
     8290.5 | 10 | 旋挖附件39吨            | 河南省     | 三门峡市   | 2015-10-
05 11:38:38 |  8290.5
     971.54 | 11 | 设备                    | 上海市     | 上海市     | 2015-
10-05 07:59:35 |  971.54
    9886.15 |  5 | 普货40吨需13米半挂一辆  | 上海市     | 上海市     | 2015-10-
05 08:13:59 | 9886.15
(11 rows)

postgres=# select last_value(amount) over(partition by begincity),* FROM
bills;
 last_value | id |       goodsdesc         | beginunit | begincity |
pubtime        | amount
------------+----+-------------------------+-----------+-----------+-----
----------------+---------
    1915.86 |  1 | 衣服                    | 海南省     | 三亚市     | 2015-
10-05 09:32:01 | 1915.86
    1316.27 |  2 | 建筑设备                | 福建省     | 三明市     | 2015-10-
05 07:21:22 | 2022.31
    1316.27 |  3 | 设备                    | 福建省     | 三明市     | 2015-
10-05 11:21:54 | 8771.11
    1316.27 |  4 | 普货                    | 福建省     | 三明市     | 2015-
10-05 15:19:17 | 1316.27
    9621.37 |  7 | 鲜香菇2000斤            | 河南省     | 三门峡市   | 2015-10-
05 10:38:29 | 4182.68
    9621.37 | 10 | 旋挖附件39吨            | 河南省     | 三门峡市   | 2015-10-
05 11:38:38 |  8290.5
    9621.37 |  6 | 5 0铲车，后八轮翻斗车   | 河南省     | 三门峡市   | 2015-10-
05 07:53:13 |  1030.9
    9621.37 |  8 | 旋挖附件38吨            | 河南省     | 三门峡市   | 2015-10-
05 10:48:38 | 5365.04
```

```
    9621.37 |  9 | 旋挖附件35吨              | 河南省     | 三门峡市   | 2015-10-
05 10:48:38 | 9621.37
     971.54 |  5 | 普货40吨需13米半挂一辆   | 上海市     | 上海市     | 2015-10-
05 08:13:59 | 9886.15
     971.54 | 11 | 设备                     | 上海市     | 上海市     | 2015-
10-05 07:59:35 |  971.54
(11 rows)
```

注意不要加上 order by id，默认情况下，带了 order by 参数会从分组的起始值开始一直叠加，直到当前值（不是当前记录）不同为止，当忽略 order by 参数则是整个分组。
下面通过修改分组的统计范围就可以实现 order by 参数取最后值。

```
postgres=# select last_value(amount) over(partition by begincity order
by id range between unbounded preceding and unbounded following),* FROM
bills;
 last_value | id |       goodsdesc         | beginunit | begincity |
pubtime        | amount
------------+----+-------------------------+-----------+-----------+-----
----------------+---------
    1915.86 |  1 | 衣服                    | 海南省     | 三亚市     | 2015-
10-05 09:32:01 | 1915.86
    1316.27 |  2 | 建筑设备                | 福建省     | 三明市     | 2015-10-
05 07:21:22 | 2022.31
    1316.27 |  3 | 设备                    | 福建省     | 三明市     | 2015-
10-05 11:21:54 | 8771.11
    1316.27 |  4 | 普货                    | 福建省     | 三明市     | 2015-
10-05 15:19:17 | 1316.27
     8290.5 |  6 | 5 0铲车，后八轮翻斗车   | 河南省     | 三门峡市   | 2015-10-
05 07:53:13 |  1030.9
     8290.5 |  7 | 鲜香菇2000斤            | 河南省     | 三门峡市   | 2015-10-
05 10:38:29 | 4182.68
     8290.5 |  8 | 旋挖附件38吨            | 河南省     | 三门峡市   | 2015-10-
05 10:48:38 | 5365.04
     8290.5 |  9 | 旋挖附件35吨            | 河南省     | 三门峡市   | 2015-10-
05 10:48:38 | 9621.37
     8290.5 | 10 | 旋挖附件39吨            | 河南省     | 三门峡市   | 2015-10-
05 11:38:38 |  8290.5
     971.54 |  5 | 普货40吨需13米半挂一辆   | 上海市     | 上海市     | 2015-10-
05 08:13:59 | 9886.15
     971.54 | 11 | 设备                     | 上海市     | 上海市     | 2015-
10-05 07:59:35 |  971.54
(11 rows)
```

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)
```

统计各个城市的总运费及平均每单的运费

```
postgres=# select sum(amount) over(partition by begincity),avg(amount)
over(partition by begincity),begincity,amount from bills;
   sum    |        avg         | begincity | amount
----------+--------------------+-----------+---------
  1915.86 |            1915.86 | 三亚市    | 1915.86
 12109.69 | 4036.56333333333   | 三明市    | 2022.31
 12109.69 | 4036.56333333333   | 三明市    | 8771.11
 12109.69 | 4036.56333333333   | 三明市    | 1316.27
```

```
 28490.49 |          5698.098 | 三门峡市   | 4182.68
 28490.49 |          5698.098 | 三门峡市   |  8290.5
 28490.49 |          5698.098 | 三门峡市   |  1030.9
 28490.49 |          5698.098 | 三门峡市   | 5365.04
 28490.49 |          5698.098 | 三门峡市   | 9621.37
 10857.69 |          5428.845 | 上海市     | 9886.15
 10857.69 |          5428.845 | 上海市     |  971.54
(11 rows)
```

窗口函数别名使用

```
postgres=# select sum(amount) over w,avg(amount) over w,begincity,amount
from bills window w as (partition by begincity);
   sum    |        avg         | begincity | amount
----------+--------------------+-----------+---------
  1915.86 |            1915.86 | 三亚市     | 1915.86
 12109.69 | 4036.56333333333 | 三明市     | 2022.31
 12109.69 | 4036.56333333333 | 三明市     | 8771.11
 12109.69 | 4036.56333333333 | 三明市     | 1316.27
 28490.49 |          5698.098 | 三门峡市   | 4182.68
 28490.49 |          5698.098 | 三门峡市   |  8290.5
 28490.49 |          5698.098 | 三门峡市   |  1030.9
 28490.49 |          5698.098 | 三门峡市   | 5365.04
 28490.49 |          5698.098 | 三门峡市   | 9621.37
 10857.69 |          5428.845 | 上海市     | 9886.15
 10857.69 |          5428.845 | 上海市     |  971.54
(11 rows)
```

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)
```

PL:pgsql 开发
应用程序语法介绍

最近更新时间：2023-08-09 15:02:01

建立函数语法

```
CREATE [OR REPLACE] FUNCTION [模式名.]函数名 ([参数模式 [参数名] 数据类型
[default 默认值] [,…]]) RETRUNS [SETOF] 数据类型 AS
[标签]
[DECLARE
    --变量定义]
BEGIN
    --注释
    /*注释*/
    --语句执行
END;
[标签]
LANGUAGE PLPGSQL;
```

[OR REPLACE] 更新函数介绍

OR REPLACE 的作用为函数存在时则替换，建立 PL/pgsql 函数时如果不带 OR REPLACE 关键字， 则遇到函数已经存在，系统会报错，如下所示：

```
postgres=# select prosrc from pg_proc where proname='f';
          prosrc
--------------------------
                         +
 BEGIN                   +
     RAISE NOTICE 'tdsql_pg';+
 END;                    +

(1 行记录)

postgres=# CREATE FUNCTION f() RETURNS VOID AS
postgres-# $$
postgres$# BEGIN
postgres$#     RAISE NOTICE 'Hello ,tdsql_pg';
postgres$# END;
postgres$# $$
```

```
postgres-# LANGUAGE PLPGSQL;
ERROR:  function "f" already exists with same argument types
postgres=# CREATE OR REPLACE FUNCTION f() RETURNS VOID AS
postgres-# $$
postgres$# BEGIN
postgres$#     RAISE NOTICE 'Hello ,tdsql_pg';
postgres$# END;
postgres$# $$
postgres-# LANGUAGE PLPGSQL;
CREATE FUNCTION
postgres=# select prosrc from pg_proc where proname='f';
             prosrc
---------------------------------
                                 +
 BEGIN                           +
     RAISE NOTICE 'Hello ,tdsql_pg';+
 END;                            +

(1 行记录)

postgres=# SELECT f();
NOTICE:  Hello ,tdsql_pg
 f
---

(1 行记录)
```

[模式名.]函数名介绍

建立函数名称，模式名可以指定，也可以不指定，不指存则存放在当前模式下，如上面例子就没有指定模式名，则就存放在当前模式下，如下所示：

```
postgres=# select * from pg_namespace;
      nspname        | nspowner |                nspacl
---------------------+----------+---------------------------------------
 pg_toast            |       10 |
 pg_temp_1           |       10 |
 pg_toast_temp_1     |       10 |
 pg_catalog          |       10 | {postgres=UC/postgres,=U/postgres}
 public              |       10 | {postgres=UC/postgres,=UC/postgres}
 information_schema  |       10 | {postgres=UC/postgres,=U/postgres}
(6 行记录)
```

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

postgres=# select pg_namespace.nspname,pg_proc.prosrc from
pg_proc,pg_namespace where
pg_proc.pronamespace=pg_namespace.oid and pg_proc.proname='f';
 nspname |             prosrc
---------+---------------------------------
 public  |                                 +
         | BEGIN                           +
         |     RAISE NOTICE 'Hello ,tdsql_pg';+
         | END;                            +
         |
(1 行记录)
```

因为 $user 模式不存在，所以存在 public 模式下。

参数详细介绍

最近更新时间：2022-09-02 15:30:29

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)，下面介绍这几个参数模式的用法。

IN 模式

IN 模式指的是执行函数时需要输入参数值，如下所示。

```
postgres=# CREATE OR REPLACE FUNCTION f1(IN a_xm text) RETURNS TEXT AS
postgres-# $$
postgres$# BEGIN
postgres$#     RETURN a_xm;
postgres$# END;
postgres$# $$
postgres-# LANGUAGE PLPGSQL;
CREATE FUNCTION
postgres=#
postgres=# SELECT f1('tdsql_pg');
  f1
------
 tdsql_pg
(1 行记录)
postgres=# CREATE OR REPLACE FUNCTION f1(a_xm text) RETURNS TEXT AS
postgres-# $$
postgres$# BEGIN
postgres$#     RETURN a_xm;
postgres$# END;
postgres$# $$
postgres-# LANGUAGE PLPGSQL;
CREATE FUNCTION
postgres=#
postgres=# SELECT * FROM f1('tdsql_pg');
  f1
------
 tdsql_pg
(1 行记录)
```

上面两种方式定义的参数效果是一样的。

OUT 模式

OUT 模式参数是指定了函数执行时返回的字段名及类型。

```
postgres=# CREATE OR REPLACE FUNCTION f1(OUT a_xm TEXT) RETURNS TEXT AS
postgres-# $$
postgres$# BEGIN
postgres$#     a_xm:='tdsql_pg';
postgres$# END;
postgres$# $$
postgres-# LANGUAGE PLPGSQL;
CREATE FUNCTION
postgres=# SELECT * FROM f1();
 a_xm
------
 tdsql_pg
(1 行记录)
```

采用 OUT 模式参数不能用 RETURN 返回，而是要对返回的 OUT 参数直接赋值。返回值类型与参数的数据类型必需一致。参数名就是返回的字段名。

INOUT 模式

INOUT 模式是指参数即传入，同时又指定了返回值的字段名和类型。

```
postgres=# CREATE OR REPLACE FUNCTION f1(INOUT a_xm text) RETURNS TEXT
AS
postgres-# $$
postgres$# BEGIN
postgres$# END;
postgres$# $$
postgres-# LANGUAGE PLPGSQL;
CREATE FUNCTION
postgres=#
postgres=# SELECT * FROM f1('tdsql_pg');
 a_xm
------
 tdsql_pg
(1 行记录)
```

值得注意的是，上面的函数跟下面的函数是相同的，即重新定义会覆盖掉。

```
postgres=# CREATE OR REPLACE FUNCTION f1(IN a_xm text) RETURNS TEXT AS
```

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

VARIADIC 模式

VARIADIC 模式是参数个数可变模式，系统用一个数组对传入的参数进行处理，VARIADIC 参数必需是所有最后一个声明，如下所示。

```
postgres=# CREATE OR REPLACE FUNCTION f1(VARIADIC a_int integer[])
RETURNS void AS
postgres-# $$
postgres$# BEGIN
postgres$#     RAISE NOTICE 'a_int = %',a_int;
postgres$#     RAISE NOTICE 'a_int[1] = %',a_int[1];
postgres$# END;
postgres$# $$
postgres-# LANGUAGE PLPGSQL;
CREATE FUNCTION
postgres=#
postgres=# SELECT f1(1);
NOTICE:  a_int = {1}
NOTICE:  a_int[1] = 1
 f1
----

(1 行记录)

postgres=# SELECT f1(1,2);
NOTICE:  a_int = {1,2}
NOTICE:  a_int[1] = 1
 f1
----
```

```
(1 行记录)

postgres=# CREATE OR REPLACE FUNCTION f1(a_xm TEXT,VARIADIC a_int
integer[])  RETURNS void AS
postgres-# $$
postgres$# BEGIN
postgres$#     RAISE NOTICE 'a_int = %',a_int;
postgres$#     RAISE NOTICE 'a_int[1] = %',a_int[1];
postgres$#     RAISE NOTICE 'a_xm = %',a_xm;
postgres$# END;
postgres$# $$
postgres-# LANGUAGE PLPGSQL;
CREATE FUNCTION
postgres=#
postgres=# SELECT f1('tdsql_pg',1,2);
NOTICE:  a_int = {1,2}
NOTICE:  a_int[1] = 1
NOTICE:  a_xm = tdsql_pga
 f1
----

(1 行记录)
```

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。

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('tdsql_pg');
  f2
------
 tdsql_pg
```

```
(1 行记录)
```

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('tdsql_pg');
  f2
------
 tdsql_pg
(1 行记录)
```

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

参数数据类型

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。

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,'tdsql_pg');
NOTICE:  a_int = 1 ; a_str = tdsql_pg
 f3
----

(1 行记录)

postgres=# CREATE OR REPLACE FUNCTION f3 (a_int integer[],a_str text[])
RETURNS VOID AS
postgres-# $$
postgres$# BEGIN
postgres$#     RAISE NOTICE 'a_int = % ; a_str = %',a_int,a_str;
postgres$# END;
postgres$# $$
postgres-# LANGUAGE PLPGSQL;
CREATE FUNCTION
postgres=# SELECT f3(ARRAY[1,2,3],ARRAY['tdsql_pg','pgxz']);
NOTICE:  a_int = {1,2,3} ; a_str = {tdsql_pg,pgxz}
 f3
----

(1 行记录)
```

复合类型

```
postgres=# CREATE TYPE t_per AS
postgres-# (
```

```
postgres(#     id integer,
postgres(#     mc text
postgres(# );
ERROR:  type "t_per" already exists
postgres=# CREATE OR REPLACE FUNCTION f3 (a_row public.t_per) RETURNS
VOID AS
postgres-# $$
postgres$# BEGIN
postgres$#     RAISE NOTICE 'id = % ; mc = %',a_row.id,a_row.mc;
postgres$# END;
postgres$# $$
postgres-# LANGUAGE PLPGSQL;
CREATE FUNCTION
postgres=#
postgres=# SELECT f3(ROW(1,'tdsql_pg')::public.t_per);
NOTICE:  id = 1 ; mc = tdsql_pg
 f3
----

(1 行记录)

postgres=# CREATE OR REPLACE FUNCTION f3 (a_rec public.t_per[]) RETURNS
VOID AS
postgres-# $$
postgres$# BEGIN
postgres$#     RAISE NOTICE 'a_rec = %',a_rec;
postgres$#     RAISE NOTICE 'a_rec[1].id = %',a_rec[1].id;
postgres$# END;
postgres$# $$
postgres-# LANGUAGE PLPGSQL;
CREATE FUNCTION
postgres=#
postgres=# SELECT
f3(ARRAY[ROW(1,'tdsql_pg'),ROW(1,'pgxz')]::public.t_per[]);
NOTICE:  a_rec = {"(1,tdsql_pg)","(1,pgxz)"}
NOTICE:  a_rec[1].id = 1
 f3
----

(1 行记录)
```

行类型

```
postgres=# \d t
    资料表 "public.t"
 栏位 |  型别   | 修饰词
------+---------+--------
 id   | integer |
 mc   | text    |

postgres=# CREATE OR REPLACE FUNCTION f3 (a_row public.t) RETURNS VOID
AS
postgres-# $$
postgres$# BEGIN
postgres$#     RAISE NOTICE 'id = % ; mc = %',a_row.id,a_row.mc;
postgres$# END;
postgres$# $$
postgres-# LANGUAGE PLPGSQL;
CREATE FUNCTION
postgres=#
postgres=# SELECT f3(ROW(1,'tdsql_pg'));
NOTICE:  id = 1 ; mc = tdsql_pg
 f3
----

(1 行记录)

postgres=# SELECT f3(t.*) FROM t LIMIT 1;
NOTICE:  id = 1 ; mc = tdsql_pg
 f3
----

(1 行记录)

postgres=# CREATE OR REPLACE FUNCTION f3 (a_rec public.t[]) RETURNS VOID
AS
postgres-# $$
postgres$# BEGIN
postgres$#     RAISE NOTICE 'a_rec = %',a_rec;
postgres$#     RAISE NOTICE 'a_rec[1].id = %',a_rec[1].id;
postgres$# END;
postgres$# $$
postgres-# LANGUAGE PLPGSQL;
CREATE FUNCTION
postgres=#
```

```
postgres=# SELECT
f3(array[row(1,'tdsql_pg'),row(1,'pgxz')]::public.t[]);
NOTICE:  a_rec = {"(1,tdsql_pg)","(1,pgxz)"}
NOTICE:  a_rec[1].id = 1
 f3
----

(1 行记录)

postgres=# SELECT f3(array[t.*,t.*]::public.t[]) FROM t LIMIT 2;
NOTICE:  a_rec = {"(1,tdsql_pg)","(1,tdsql_pg)"}
NOTICE:  a_rec[1].id = 1
NOTICE:  a_rec = {"(2,pgxz)","(2,pgxz)"}
NOTICE:  a_rec[1].id = 2
 f3
----

(2 行记录)
```

域类型

```
postgres=# CREATE DOMAIN xb AS TEXT CHECK
postgres-# (
postgres(#     VALUE = '男'
postgres(#     OR VALUE ='女'
postgres(#     OR VALUE = ''
postgres(# );
CREATE DOMAIN
postgres=#
postgres=# CREATE OR REPLACE FUNCTION f4 (a_xb public.xb) RETURNS VOID
AS
postgres-# $$
postgres$# BEGIN
postgres$#     RAISE NOTICE 'a_xb = %',a_xb;
postgres$# END;
postgres$# $$
postgres-# LANGUAGE PLPGSQL;
CREATE FUNCTION
postgres=#
postgres=# SELECT * FROM f4('男');
NOTICE:  a_xb = 男
 f4
----
```

```
(1 行记录)

postgres=# SELECT * FROM f4('她');
ERROR:  value for domain xb violates check constraint "xb_check"
postgres=#
```

域类型输入参数值时会检查是否违反规则。

游标类型

```
postgres=# CREATE OR REPLACE FUNCTION f5 (a_ref refcursor) RETURNS void
AS
postgres-# $$
postgres$# DECLARE
postgres$#     v_rec record;
postgres$# BEGIN
postgres$#     OPEN a_ref FOR SELECT * FROM t LIMIT 1;
postgres$#     FETCH a_ref INTO v_rec;
postgres$#     RAISE NOTICE 'v_rec = % ',v_rec;
postgres$# END;
postgres$# $$
postgres-# LANGUAGE PLPGSQL;
CREATE FUNCTION
postgres=#
postgres=# SELECT * FROM f5('a');
NOTICE:  v_rec = (1,tdsql_pg)
 f5
----

(1 行记录)
postgres=# CREATE OR REPLACE FUNCTION f6 (a_ref refcursor) RETURNS
refcursor AS
postgres-# $$
postgres$# BEGIN
postgres$#     OPEN a_ref FOR SELECT * FROM t LIMIT 1;
postgres$#     RETURN a_ref;
postgres$# END;
postgres$# $$
postgres-# LANGUAGE PLPGSQL;
CREATE FUNCTION
postgres=#
#注意这里需要开启一个事务
```

```
postgres=# BEGIN;
BEGIN
postgres=# SELECT * FROM f6('a');
 f6
----
 a
(1 行记录)
postgres=# FETCH ALL FROM a;
 id |  mc
----+------
  1 | tdsql_pg
(1 行记录)
```

多态类型

```
postgres=# CREATE OR REPLACE FUNCTION f_any(a_arg anyelement) RETURNS
VOID AS
postgres-# $$
postgres$# BEGIN
postgres$#     RAISE NOTICE '%',a_arg;
postgres$# END;
postgres$# $$
postgres-# LANGUAGE PLPGSQL;
CREATE FUNCTION
postgres=#
postgres=# SELECT f_any(1::integer);
NOTICE:  1
 f_any
-------

(1 行记录)

postgres=# SELECT f_any('tdsql_pg'::TEXT);
NOTICE:  tdsql_pg
 f_any
-------

(1 行记录)

postgres=# SELECT f_any(ROW(1,'tdsql_pg')::public.t_rec);
NOTICE:  (1,tdsql_pg)
 f_any
-------
```

```
(1 行记录)

postgres=# SELECT f_any(ARRAY[1,2]::INTEGER[]);
NOTICE:  {1,2}
 f_any
-------

(1 行记录)

postgres=# SELECT f_any(ARRAY[[1,2],[3,4],[5,6]]::INTEGER[][][]);
NOTICE:  {{1,2},{3,4},{5,6}}
 f_any
-------

(1 行记录)
```

注意多态类型参数函数调用时最好直接声明参数类型，否则有可能出错。

```
postgres=# CREATE OR REPLACE FUNCTION fanyarray(aarg anyarray) RETURNS
VOID AS
postgres-# $$
postgres$# BEGIN
postgres$#     RAISE NOTICE '%',a_arg;
postgres$# END;
postgres$# $$
postgres-# LANGUAGE PLPGSQL;
CREATE FUNCTION
postgres=#
postgres=# SELECT f_any_array(ARRAY['tdsql_pg','pgxz']::TEXT[]);
NOTICE:  {tdsql_pg,pgxz}
 f_any_array
-------------

(1 行记录)

postgres=# SELECT
f_any_array(ARRAY[ARRAY['tdsql_pg','pgxz'],ARRAY['tdsql_pg','Tencent']]:
:TEXT[][]);
NOTICE:  {{tdsql_pg,pgxz},{tdsql_pg,Tencent}}
 f_any_array
-------------
```

```
(1 行记录)
```

Anyelement 参数如果写成数组，其意义就跟 anyarray 参数一致，所以 f_any(a_arg anyelement) 与 f_any(a_arg anyarray) 在调用 f_any(ARRAY[1,2]) 时就会出现函数不是唯一化的错误 (ERROR: function f_any(…) is not unique) 提示。

参数默认值

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。

变量使用

最近更新时间：2022-09-02 15:31:12

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。

变量使用实例

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 :=。

定义一个普通变量

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

```
 f25
-----

(1 row)

postgres=#
```

定义 CONSTANT 变量

```
postgres=# CREATE OR REPLACE FUNCTION f25() RETURNS VOID AS
postgres-# $$
postgres$# DECLARE
postgres$#     v_int CONSTANT integer := 1;
postgres$# BEGIN
postgres$#     RAISE NOTICE 'v_int = %',v_int;
postgres$# END;
postgres$# $$
postgres-# LANGUAGE plpgsql;
CREATE FUNCTION
postgres=# select f25();
NOTICE:  v_int = 1
 f25
-----

(1 row)
```

CONSTANT 不能再次赋值。

```
postgres=# CREATE OR REPLACE FUNCTION f25() RETURNS VOID AS
postgres-# $$
postgres$# DECLARE
postgres$#     v_int CONSTANT integer := 1;
postgres$# BEGIN
postgres$#     RAISE NOTICE 'v_int = %',v_int;
postgres$#     v_int = 10;
postgres$#     RAISE NOTICE 'v_int = %',v_int;
postgres$# END;
postgres$# $$
postgres-# LANGUAGE plpgsql;
ERROR:  "v_int" is declared CONSTANT
```

定义 NOT NULL 变量

```
postgres=# CREATE OR REPLACE FUNCTION f25() RETURNS VOID AS
postgres-# $$
postgres$# DECLARE
postgres$#     v_int integer NOT NULL := 1;
postgres$# BEGIN
postgres$#     RAISE NOTICE 'v_int = %',v_int;
postgres$#     SELECT NULL INTO v_int;
postgres$#     RAISE NOTICE 'v_int = %',v_int;
postgres$# END;
postgres$# $$
postgres-# LANGUAGE plpgsql;
CREATE FUNCTION
postgres=# SELECT f25();
NOTICE:  v_int = 1
ERROR:  null value cannot be assigned to variable "v_int" declared NOT
NULL
CONTEXT:  PL/pgSQL function f25() line 6 at SQL statement
postgres=#
```

定义为 NOT NULL 变量，则该变量受 NOT NULL 约束。

定义 COLLATE 变量

按 unicode 值对比大小。

```
postgres=# CREATE OR REPLACE FUNCTION f25() RETURNS VOID AS
postgres-# $$
postgres$# DECLARE
postgres$#     v_txt1 TEXT COLLATE "C" := '严';
postgres$#     v_txt2 TEXT COLLATE "C" := '丰';
postgres$# BEGIN
postgres$#     IF v_txt1 > v_txt2 THEN
postgres$#         RAISE NOTICE ' % -> % ',v_txt1,v_txt2;
postgres$#     ELSE
postgres$#         RAISE NOTICE ' % -> % ',v_txt2,v_txt1;
postgres$#     END IF;
postgres$# END;
postgres$# $$
postgres-# LANGUAGE plpgsql;
CREATE FUNCTION
postgres=# SELECT f25();
```

```
NOTICE:  丰 -> 严
 f25
-----

(1 row)

postgres=# select '严'::bytea;
  bytea
----------
 \xe4b8a5
(1 row)

postgres=# select '丰'::bytea;
  bytea
----------
 \xe4b8b0
(1 row)
```

按汉字的拼音对比大小。

```
postgres=# CREATE OR REPLACE FUNCTION f25() RETURNS VOID AS
postgres-# $$
postgres$# DECLARE
postgres$#     v_txt1 TEXT COLLATE "zh_CN.utf8" := '严';
postgres$#     v_txt2 TEXT COLLATE "zh_CN.utf8" := '丰';
postgres$# BEGIN
postgres$#     IF v_txt1 > v_txt2 THEN
postgres$#         RAISE NOTICE ' % -> % ',v_txt1,v_txt2;
postgres$#     ELSE
postgres$#         RAISE NOTICE ' % -> % ',v_txt2,v_txt1;
postgres$#     END IF;
postgres$# END;
postgres$# $$
postgres-# LANGUAGE plpgsql;
CREATE FUNCTION
postgres=# SELECT f25();
NOTICE:  严 -> 丰
 f25
-----

(1 row)
```

变量赋值

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

(1 row)
```

控制结构

最近更新时间：2022-09-02 15:31:49

判断语句

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=#
```

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=#
```

IF...THEN...ELSIF...THEN...ELSE...END IF

```
postgres=# CREATE OR REPLACE FUNCTION f26() RETURNS VOID AS
postgres-# $$
postgres$# DECLARE
postgres$#     v_float8 float8 := random();
postgres$# BEGIN
postgres$#     IF v_float8>0.99 THEN
postgres$#         RAISE NOTICE '随机数大于0.99';
postgres$#     ELSIF v_float8>0.5 THEN
postgres$#         RAISE NOTICE '随机数大于0.50';
postgres$#     ELSIF v_float8>0.25 THEN
postgres$#         RAISE NOTICE '随机数大于0.25';
postgres$#     ELSE
postgres$#         RAISE NOTICE '随机数小于或等于0.25';
postgres$#     END IF;
postgres$# END;
postgres$# $$
postgres-# LANGUAGE plpgsql;
CREATE FUNCTION
postgres=# SELECT f26();
NOTICE:  随机数大于0.50
 f26
-----

(1 row)
```

CASE 语句

```
postgres=# CREATE OR REPLACE FUNCTION f26() RETURNS VOID AS
postgres-# $$
postgres$# DECLARE
postgres$#     v_float8 float8 := random();
postgres$# BEGIN
```

```
postgres$#     CASE
postgres$#     WHEN v_float8>0.99 THEN
postgres$#         RAISE NOTICE '随机数大于0.99';
postgres$#     WHEN v_float8>0.5 THEN
postgres$#         RAISE NOTICE '随机数大于0.50';
postgres$#     WHEN v_float8>0.25 THEN
postgres$#         RAISE NOTICE '随机数大于0.25';
postgres$#     ELSE
postgres$#         RAISE NOTICE '随机数小于或等于0.25';
postgres$#     END CASE;
postgres$# END;
postgres$# $$
postgres-# LANGUAGE plpgsql;
CREATE FUNCTION
postgres=# SELECT f26();
NOTICE:  随机数大于0.50
 f26
-----


(1 row)
```

循环语句

LOOP 循环

```
postgres=# CREATE OR REPLACE FUNCTION f27() RETURNS VOID AS
postgres-# $$
postgres$# DECLARE
postgres$#      v_id INTEGER := 1;
postgres$# BEGIN
postgres$#     LOOP
postgres$#         RAISE NOTICE '%',v_id;
postgres$#         EXIT WHEN random()>0.8;
postgres$#         v_id := v_id + 1;
postgres$#     END LOOP ;
postgres$# END;
postgres$# $$
postgres-# LANGUAGE plpgsql;
CREATE FUNCTION
postgres=# SELECT f27();
NOTICE:  1
NOTICE:  2
 f27
```

```
-----

(1 row)
```

使用 EXIT 退出循环。

```
postgres=# CREATE OR REPLACE FUNCTION f27() RETURNS VOID AS
postgres-# $$
postgres$# DECLARE
postgres$#     v_id INTEGER := 1;
postgres$#     v_random float8 ;
postgres$# BEGIN
postgres$#     LOOP
postgres$#        RAISE NOTICE '%',v_id;
postgres$#        v_id := v_id + 1;
postgres$#        v_random := random();
postgres$#        IF v_random > 0.8 THEN
postgres$#            RETURN;
postgres$#        END IF;
postgres$#     END LOOP ;
postgres$# END;
postgres$# $$
postgres-# LANGUAGE plpgsql;
CREATE FUNCTION
postgres=# SELECT f27();
NOTICE:  1
NOTICE:  2
NOTICE:  3
NOTICE:  4
NOTICE:  5
 f27
-----


(1 row)

postgres=#
```

使用 RETURN 退出循环返回。

WHILE 循环

```
postgres=# CREATE OR REPLACE FUNCTION f27() RETURNS VOID AS
```

```
postgres-# $$
postgres$# DECLARE
postgres$#     v_id INTEGER := 1;
postgres$#     v_random float8 := random() ;
postgres$# BEGIN
postgres$#     WHILE v_random > 0.8 LOOP
postgres$#         RAISE NOTICE '%',v_id;
postgres$#         v_id := v_id + 1;
postgres$#         v_random = random();
postgres$#     END LOOP;
postgres$# END;
postgres$# $$
postgres-# LANGUAGE plpgsql;
CREATE FUNCTION
postgres=# SELECT f27();
NOTICE:  1
 f27
-----

(1 row)
```

FOR 循环

```
postgres=# CREATE OR REPLACE FUNCTION f27() RETURNS VOID AS
postgres-# $$
postgres$# BEGIN
postgres$#     FOR i IN 1..3 LOOP
postgres$#         RAISE NOTICE 'i = %',i;
postgres$#     END LOOP;
postgres$# END;
postgres$# $$
postgres-# LANGUAGE plpgsql;
CREATE FUNCTION
postgres=# SELECT f27();
NOTICE:  i = 1
NOTICE:  i = 2
NOTICE:  i = 3
 f27
-----

(1 row)

postgres=# CREATE OR REPLACE FUNCTION f27() RETURNS VOID AS
```

```
postgres-# $$
postgres$# BEGIN
postgres$#     FOR i IN REVERSE 3..1 LOOP
postgres$#         RAISE NOTICE 'i = %',i;
postgres$#     END LOOP;
postgres$# END;
postgres$# $$
postgres-# LANGUAGE plpgsql;
CREATE FUNCTION
postgres=# SELECT f27();
NOTICE:  i = 3
NOTICE:  i = 2
NOTICE:  i = 1
 f27
-----

(1 row)
```

使用 REVERSE 递减。

```
postgres=# CREATE OR REPLACE FUNCTION f27() RETURNS VOID AS
postgres-# $$
postgres$# BEGIN
postgres$#     FOR i IN 1..8 BY 2 LOOP
postgres$#         RAISE NOTICE 'i = %',i;
postgres$#     END LOOP;
postgres$# END;
postgres$# $$
postgres-# LANGUAGE plpgsql;
CREATE FUNCTION
postgres=# SELECT f27();
NOTICE:  i = 1
NOTICE:  i = 3
NOTICE:  i = 5
NOTICE:  i = 7
 f27
-----

(1 row)
```

使用 BY 设置步长。

FOR 循环查询结果

```
postgres=# CREATE OR REPLACE FUNCTION f27() RETURNS VOID AS
postgres-# $$
postgres$# DECLARE
postgres$#     v_rec RECORD;
postgres$# BEGIN
postgres$#     FOR v_rec IN SELECT * FROM public.t LOOP
postgres$#         RAISE NOTICE '%',v_rec;
postgres$#     END LOOP;
postgres$# END;
postgres$# $$
postgres-# LANGUAGE plpgsql;
CREATE FUNCTION
postgres=# SELECT f27();
NOTICE:  (1,tdsql_pg)
NOTICE:  (2,pgxz)
 f27
-----


(1 row)
```

FOREACH 循环一个数组

```
postgres=# CREATE OR REPLACE FUNCTION f27() RETURNS VOID AS
postgres-# $$
postgres$# DECLARE
postgres$#     v_random_arr float8[]:=ARRAY[random(),random()];
postgres$#     v_random float8;
postgres$# BEGIN
postgres$#     FOREACH v_random IN ARRAY v_random_arr LOOP
postgres$#         RAISE NOTICE '%',v_random ;
postgres$#     END LOOP;
postgres$# END;
postgres$# $$
postgres-# LANGUAGE plpgsql;
CREATE FUNCTION
postgres=# SELECT f27();
NOTICE:  0.452758576720953
NOTICE:  0.975814974401146
 f27
-----
```

```
(1 row)

postgres=# CREATE OR REPLACE FUNCTION f27() RETURNS VOID AS
postgres-# $$
postgres$# DECLARE
postgres$#     v_random_arr float8[]
[]:=ARRAY[ARRAY[random(),random()],ARRAY[random(),random()]];
postgres$#     v_random float8;
postgres$# BEGIN
postgres$#     FOREACH v_random SLICE 0 IN ARRAY v_random_arr LOOP
postgres$#         RAISE NOTICE '%',v_random ;
postgres$#     END LOOP;
postgres$# END;
postgres$# $$
postgres-# LANGUAGE plpgsql;
CREATE FUNCTION
postgres=# SELECT f27();
NOTICE:  0.0588191924616694
NOTICE:  0.368828620761633
NOTICE:  0.813376842066646
NOTICE:  0.415377039927989
 f27
-----

(1 row)
```

循环会通过计算 expression 得到的数组的个体元素进行迭代。

```
postgres=# CREATE OR REPLACE FUNCTION f27() RETURNS VOID AS
postgres-# $$
postgres$# DECLARE
postgres$#     v_random_arr float8[]
[]:=ARRAY[ARRAY[random(),random()],ARRAY[random(),random()]];
postgres$#     v_random float8[];
postgres$# BEGIN
postgres$#     FOREACH v_random SLICE 1 IN ARRAY v_random_arr LOOP
postgres$#         RAISE NOTICE '%',v_random ;
postgres$#     END LOOP;
postgres$# END;
postgres$# $$
postgres-# LANGUAGE plpgsql;
CREATE FUNCTION
```

```
postgres=# SELECT f27();
NOTICE:  {0.578366641886532,0.78098024148494}
NOTICE:  {0.783956411294639,0.450278480071574}
 f27
-----

(1 row)
```

通过一个正 SLICE 值，FOREACH 通过数组的切片而不是单一元素迭代。

其它控制语句

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
-------
 tdsql_pg
(1 row)
```

动态执行就是拼 SQL 语句，然后使用 EXECUTE 命令执行。

执行一个没有结果的命令

```
postgres=# CREATE OR REPLACE FUNCTION f27() RETURNS void AS
postgres-# $$
postgres$# BEGIN
postgres$#     perform f27(1);
postgres$# END;
postgres$# $$
```

```
postgres-# LANGUAGE plpgsql;
CREATE FUNCTION
postgres=# SELECT f27();
 f27
-----

(1 row)

postgres=#
```

获取执行结果

```
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tdsql_pg
 f27
-----

(1 row)

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

(1 row)
```

获取影响行数

```
postgres=# CREATE OR REPLACE FUNCTION f27(a_id INTEGER) RETURNS VOID AS
postgres-# $$
postgres$# DECLARE
postgres$#     v_mc TEXT;
postgres$#     v_row_count BIGINT;
postgres$# BEGIN
postgres$#     SELECT mc INTO v_mc FROM t WHERE id=a_id;
postgres$#     GET DIAGNOSTICS v_row_count = ROW_COUNT;
postgres$#     RAISE NOTICE '查询到的记录数为 % ',v_row_count;
postgres$# END;
postgres$# $$
postgres-# LANGUAGE plpgsql;
CREATE FUNCTION

postgres=# SELECT f27(1);
NOTICE:  查询到的记录数为 1
 f27
-----

(1 row)
postgres=# SELECT f27(3);
NOTICE:  查询到的记录数为 0
 f27
-----

(1 row)
```

俘获错误

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,'tdsql_pg');
 f27
-----


(1 row)

postgres=# SELECT f27(1,'tdsql_pg');
   f27
----------
 执行出错
(1 row)
```

获取错误相关信息

```
postgres=# CREATE OR REPLACE FUNCTION f27(a_id integer,a_nc text)
RETURNS TEXT AS
postgres-# $$
postgres$# DECLARE
postgres$#      v_sqlstate text;
postgres$#      v_context text;
postgres$#      v_message_text text;
postgres$# BEGIN
postgres$#      INSERT INTO t_exception VALUES(a_id,a_nc);
postgres$#      RETURN '';
postgres$#      EXCEPTION WHEN OTHERS THEN
postgres$#      GET STACKED DIAGNOSTICS v_sqlstate = RETURNED_SQLSTATE,
postgres$#                              v_message_text =
MESSAGE_TEXT,
postgres$#                              v_context =
PG_EXCEPTION_CONTEXT;
postgres$#      RAISE NOTICE '错误代码 : %',v_sqlstate;
postgres$#      RAISE NOTICE '出错信息 : %',v_message_text;
postgres$#      RAISE NOTICE '发生异常语句 : %',v_context;
postgres$#      RETURN '错误代码 : '||v_sqlstate || '\n出错信息 :
'||v_message_text|| '发生异常语句 : '||v_context;
```

```
postgres$# END;
postgres$# $$
postgres-# LANGUAGE plpgsql;
CREATE FUNCTION

postgres=# SELECT f27(1,'tdsql_pg');
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)
```

触发器函数

最近更新时间：2022-09-02 15:32:09

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-# $$
postgres$# BEGIN
postgres$#     IF NEW.nc = '' THEN
postgres$#         NEW.nc = 'tdsql_pg_' || random()::text;
postgres$#     END IF;
postgres$#     RETURN NEW;
postgres$# END;
postgres$# $$
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 | tdsql_pg_0.426093454472721
(1 row)
```

注意使用 BEFORE，不能使用 AFTER，否则重写失效。

UPDATE 事件触发器函数

不准许更新 t_trigger.nc 字段值为 tdsql_pg。

```
postgres=# CREATE OR REPLACE FUNCTION t_trigger_update_trigger_func()
RETURNS trigger AS
postgres-# $$
postgres$# BEGIN
postgres$#     --不准许t_trigger.nc值为 tdsql_pg
postgres$#     IF NEW.nc = 'tdsql_pg' THEN
postgres$#         NEW.nc = OLD.nc ;
postgres$#     END IF;
postgres$#     RETURN NEW;
postgres$# END;
postgres$# $$
postgres-# LANGUAGE plpgsql;
CREATE FUNCTION
postgres=# CREATE TRIGGER t_trigger_update_trigger BEFORE UPDATE ON
t_trigger FOR EACH ROW EXECUTE PROCEDURE
t_trigger_update_trigger_func();
CREATE TRIGGER
postgres=# UPDATE t_trigger SET nc='tdsql_pg' WHERE id=1;
UPDATE 1
postgres=# SELECT * FROM t_trigger ;
 id |           nc
----+------------------------
  1 | tdsql_pg_0.426093454472721
(1 row)

postgres=#
```

DELETE 事件触发器函数

限制 tdsql_pg 记录不能被删除。

```
postgres=# CREATE OR REPLACE FUNCTION t_trigger_delete_trigger_func()
RETURNS trigger AS
postgres-# $$
postgres$# BEGIN
postgres$#     --不准许t_trigger.nc值为 tdsql_pg
postgres$#     IF OLD.nc = 'tdsql_pg' THEN
postgres$#         RETURN NULL;
postgres$#         --RAISE EXCEPTION 'tdsql_pg不能被删除';
postgres$#     END IF;
postgres$#     RETURN OLD;
postgres$# END;
```

```
postgres$# $$
postgres-# LANGUAGE plpgsql;
CREATE FUNCTION
postgres=# CREATE TRIGGER t_trigger_delete_trigger BEFORE DELETE ON
t_trigger FOR EACH ROW EXECUTE PROCEDURE
t_trigger_delete_trigger_func();
CREATE TRIGGER
postgres=# INSERT INTO t_trigger VALUES(2,'tdsql_pg');
INSERT 0 1
postgres=# SELECT * t_trigg

postgres=# SELECT * FROM t_trigger ;
 id |           nc
----+-------------------------
  1 | tdsql_pg_0.426093454472721
  2 | tdsql_pg
(2 rows)
postgres=# DELETE FROM t_trigger WHERE id=2;
DELETE 0
postgres=# SELECT * FROM t_trigger ;
 id |           nc
----+-------------------------
  1 | tdsql_pg_0.426093454472721
  2 | tdsql_pg
(2 rows)
```

删除触发器

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

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。

更多的触发器函数使用

更多的触发器函数使用请参见 文档 。

消息及异常输出

最近更新时间：2023-07-20 10:49:02

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)。

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，则事务会终止(abort)。

RAISE EXCEPTION 自定义 ERRCODE

```
postgres=# CREATE OR REPLACE FUNCTION f28() RETURNS VOID AS
postgres-# $$
postgres$# DECLARE
postgres$#     v_int INTEGER := 1;
postgres$# BEGIN
postgres$#     RAISE EXCEPTION ' 程序EXCEPTION ' USING ERRCODE = '23505';
postgres$# END;
postgres$# $$
postgres-# LANGUAGE plpgsql;
CREATE FUNCTION
postgres=# SELECT f28();
ERROR:    程序EXCEPTION
```

日志中会记录这个 ERRCODE。

```
2017-10-03 18:40:16.710 CST,"pgxz","postgres",15072,"
[local]",59d33b65.3ae0,225,"idle",2017-10-03 15:25:25
CST,4/367159,0,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"
```

Oracle 兼容特性
Oracle 兼容语法参数配置

最近更新时间：2025-03-17 15:32:02

本节主要介绍 TDSQL PG 数据库的 Oracle 模式中需要配置参数。
TDSQL PG 的参数配置主要有以下几种参数级别：`session 级`，`server 级`，`internal 级`。

- session 级：
  - 普通用户配置：用户在当前会话设置，只在当前会话生效，不会影响其他 session。使用方法：`set parameter = value` 。
  - **session 级（superuser）**配置：session 级的特殊情况，相关参数只能由 superuser 设置，普通用户设置时会报错。
- server 级：
  - 方法1：superuser 通过 `alter system` 关键字命令设置，使用方法：`alter system set parameter = value` ，修改后需要重新启动数据库服务器才能够生效。

    > **说明：**
    > alter system 修改后只是将该参数写到 `postgresql.auto.conf` 文件里，故需要重启才可以生效。

  - 方法2：手动配置到 `postgresql.conf` 里，修改后重启数据库生效。

    > **说明：**
    > 由于 `postgresql.auto.conf` 文件里的内容，在执行 `alter system reset all;` 会全部清除，故若需要手动配置，最好不要放在 `postgresql.auto.conf` 里。

- internal 级：内部级别，不允许用户手动进行设置。

TDSQL PG 数据库的 Oracle 模式中可配置参数参考下表：

参数名称	参数类型	参数级别	取值范围	参数说明
allow_limit_ident	布尔	session 级	on/off	是否允许 limit 做标识符，默认 off。
allow_unmatched_block_comments	布尔	session 级	on/off	是否允许 c 风格注释 /* */ 不配对出现，默认 off。

case_as_alias	布尔	session 级	on/off	是否允许用 case 做别名，默认 off。
convert_sysview_to_sysclass	布尔	session 级	on/off	在 oracle 模式下是否支持 regclass 函数把 PG_CLASS 这种系统视图转化为 pg_class 系统表，默认 on。
dblink_types	枚举	session 级	oracle_x86 oracle_arm tdsql_pg/postgresql	DBLink 连接的是 oracle 还是 postgresql，需要在 CREATE DATABASE LINK 之前设置，默认 oracle_x86。
default_sql_mode	枚举	session 级	oracle/postgresql	CREATE DATABASE 不指定 mode 和 template 时的默认模式，默认 postgresql。
default_with_rowid	布尔	session 级	on/off	创建的表是否带有 ROWID 列，默认值 off。
different_func_argret	布尔	session 级	on/off	函数的 out 参数行为是否兼容 oracle，默认 off。
disable_empty_to_null	布尔	session 级	on/off	是否禁止空串('')转换为 null，默认 off。
do_as_alias	布尔	session 级	on/off	是否允许用 do 做别名，默认 off。
enable_create_package_rec_typ	布尔	session 级	on/off	在包头中用户定义 record 类型是否允许定义到 pg_type 中，这样包外也可以使用该 record 类型，默认 off。
enable_distinct_order_by	布尔	session 级	on/off	检查 order by 的列与 distinct 的列是否匹配，默认 off。
enable_function_overload	布尔	session 级	on/off	是否允许函数重载，默认 on。

enable_inline_target_function	布尔	session 级(superuser)	on/off	是否允许将 targetlist 中对应的 sql function 转化为子连接，默认值 off。
enable_nestedtable_check	布尔	session 级	on/off	SQL 中引用包函数，而该函数包含嵌套类型，将导致 CachePlan 失败，在循环中执行将比较耗时，通过该配置规避 cacheplan 问题，默认 on。
enable_oracle_column_name	布尔	session 级	on/off	查询返回的列名是否完全兼容 oracle，默认 off。
enable_oracle_dist_udaf	布尔	session 级	on/off	是否允许在分布式环境创建 oracle 自定义聚合函数，默认 off。
enable_oracle_implicit_coercion	布尔	session 级	on/off	是否使用 oracle 隐式转换，默认 on。
enable_oracle_pkg_nested_table	布尔	session 级	on/off	包中嵌套表类型是否在 pg_type 中记录一条类型信息，默认 off。
enable_plsql_analysis	布尔	session 级	on/off	是否在创建函数或者存储过程时支持 PLSQL 静态检查，默认 off。
enable_secfunc_xact	布尔	session 级(superuser)	on/off	是否允许在 security definer 函数中使用 commit/rollback，默认 on。
enable_type_priority_in_ora_mode	布尔	session 级	on/off	是否在 oracle 模式下支持类型优先级判断，默认 on。
enable_typobj_func_verify	布尔	session 级	on/off	检查 type object 的 spec 和 body 的函数是否匹配，默认 on。
function_xact_control	布尔	session 级	on/off	函数内是否支持 COMMIT/ROLLBACK，默认 on。
keywords_as_ident	布尔	session 级	on/off	是否允许关键字做标识符，默认 on。
max_autonomous_transactions	整型	server 级	0~536870911	支持的自治事务最大数，默认100。

nls_date_format	字符串	session 级	-	date 的输入输出格式，默认 YYYY-MM-DD HH24:MI:SS。
nls_sort_locale	字符串	session 级	SCHINESE_PINYIN_M SCHINESE_STROKE_M SCHINESE_RADICAL_M	nlssort 函数使用的排序规则，默认空。
nls_timestamp_format	字符串	session 级	-	timestamp 的输入输出格式，默认 YYYY-MM-DD HH24:MI:SSXFF。
nls_timestamp_tz_format	字符串	session 级	-	timestamptz 的输入输出格式，默认 YYYY-MM-DD HH24:MI:SSXFF TZH:TZM。
only_rename_typobj_metainfo	布尔	session 级	on/off	alter type object rename 的时候，是否只改名字，不改其它信息。调试使用，正常情况下不要使用该参数， 默认 off。
plpgsql_hold_cursor	布尔	session 级	on/off	事务提交的时候是否保持游标，默认 on。
plsql_emptystring_to_blank	布尔	session 级	on/off	PLSQL 中空串仍然是空串，不会转换成 null，默认 on。
profile_password_limit	布尔	session 级 (superuser)	on/off	是否启用密码限制，默认 off。

profile_resource_limit	布尔	session 级 (superuser)	on/off	是否启用资源限制，默认 off。
skip_check_same_relname	布尔	session 级	on/off	是否跳过检测 sql 中存在相同的别名表，默认 on。
sql_mode	枚举	internal 级	oracle/postgresql	该参数仅用于显示当前连接的数据库的模式，不允许修改，无默认值，依赖于连接的数据库类型。

Oracle 兼容特性概述

最近更新时间：2024-11-01 15:20:42

本节介绍 TDSQL PG 数据库在 Oracle 兼容方面的特性。
TDSQL PG 数据库基本兼容了 Oracle 的数据类型、SQL 功能、数据库对象（如：同义词、包）。在 PL（过程化语言）方面也做了很多细致兼容工作，用户应用迁移基本不需要改动业务代码。
TDSQL PG 对如下功能做了兼容：

- 数据类型
- 内建函数
- SQL 语法
- PL/SQL
- 系统视图
- 系统包
- 数据库对象管理
- Oracle 配置参数

使用 Oracle 兼容模式

TDSQL PG 内核引擎会根据数据库类型来区分 Oracle 兼容模式与 PG 模式。
在创建数据库的时候使用 `sql mode` *`db_mode`* 关键字即可创建一个指定模式的数据库，db_mode 可选 oracle 或 postgresql。不指定即默认 PostgreSQL 模式。
示例：

```
tdsql=# create database ora sql mode oracle;
CREATE DATABASE

tdsql=# \l
                                    List of databases
     Name       | Owner | Encoding | Sqlmode  |  Collate   |   Ctype    |
Access privileges
---------------+-------+----------+----------+------------+------------
+-------------------
 ora            | tbase | UTF8     | oracle   | zh_CN.utf8 | zh_CN.utf8 |
 postgres       | tbase | UTF8     | postgres | zh_CN.utf8 | zh_CN.utf8 |
 tbase_db       | tbase | UTF8     | postgres | zh_CN.utf8 | zh_CN.utf8 |
 tdsql          | tbase | UTF8     | postgres | zh_CN.utf8 | zh_CN.utf8 |
 template0      | tbase | UTF8     | postgres | zh_CN.utf8 | zh_CN.utf8 |
=c/tbase          +
```

```
                |          |          |          |             |             |
tbase=CTc/tbase
 template0_ora  | tbase    | UTF8     | oracle   | zh_CN.utf8  | zh_CN.utf8  |
=c/tbase        +
                |          |          |          |             |             |
tbase=CTc/tbase
 template1      | tbase    | UTF8     | postgres | zh_CN.utf8  | zh_CN.utf8  |
=c/tbase        +
                |          |          |          |             |             |
tbase=CTc/tbase
 template1_ora  | tbase    | UTF8     | oracle   | zh_CN.utf8  | zh_CN.utf8  |
=c/tbase        +
                |          |          |          |             |             |
tbase=CTc/tbase
 user1          | user1    | UTF8     | postgres | zh_CN.utf8  | zh_CN.utf8  |
(9 rows)
```

可以看到 `ora` 数据库的 `Sqlmode` 列显示 `oracle`，即表示 `ora` 数据库当前为 Oracle 模式。
而 `tdsql` 数据库的 `Sqlmode` 列显示 `postgres`，即表示 `tdsql` 数据库当前为 PostgreSQL 模式。

SQL 数据类型

TDSQL PG 兼容了 Oracle 所有常用数据类型。详情参见 SQL 数据类型 。

内建函数

TDSQL PG 兼容了大部分 Oracle 内建函数。详情参见 内建函数 。

SQL 语法

TDSQL PG 支持语法说明：

序号	Oracle 数据库	TDSQL PG 数据库	说明
1	SELECT	支持	• 支持单表和多表连接；支持多种 JOIN 连接方式（内连接、外连接）；支持子查询；支持层次查询；支持多种窗口函数等。 • 支持多种集合操作：UNION、UNION ALL、INTERSECT、EXCEPT、MINUS 等。 • 支持 EXPLAIN 打印计划。 • 支持 PIVOT、UNPIVOT
2	INSERT	支持	• 支持 `VALUES` 插入一行或者多行；

			• 支持分区表插入； • 支持 `INSERT ... SELECT` 插入数据 • 支持同时插入多个目标表（INSERT ALL/FIRST）
3	UPDATE	支持	• SET 支持一次更新多列数据
4	DELETE	支持	• 支持删除表数据
5	TRUNCATE	支持	• 支持清空表数据
6	并行查询	支持	• 支持并行 INSERT 和 UPDATE
7	Hint	支持	• 支持 Hint

PL/SQL

TDSQL PG 支持 PLSQL 语法说明：

序号	Oracle 数据库	TDSQL PG 数据库	说明
1	数据类型	支持	• 支持 sql 类型数据类型 • 支持集合和记录类型 • 支持 object 类型 • 支持 oracle 兼容的 pls_integer 和 binary_integer
2	流程控制	支持	• 支持循环控制逻辑 • 支持条件控制逻辑 • 支持顺序执行逻辑
3	集合类型与记录类型	支持	• 支持嵌套表 • 支持关联数组 • 支持可变数组 • 支持记录类型
4	静态 SQL	支持	• 支持静态 SQL
5	动态 SQL	支持	• 支持动态 SQL
6	子过程	支持	• 支持包内子过程 • 支持 OBJECT 类型子过程

			• 支持创建 FUNCTION 和P ROCEDURE
7	触发器	支持	• 支持 insert/update/delete/truncate4种类型 DML 触发器 • 支持 alter/drop/create 且出发目标是 database/schema 的 ddl 触发器 • 支持 befor/after/instead of 三种触发时机的触发器
8	异常处理	支持	• PL/SQL 中支持异常块，可以便捷地检查和处理错误
9	PACKAGE 对象	支持	• 支持定义包对象，用来封装一组成员变量和成员函数
10	自定义类型	支持	• 支持使用 CREATE TYPE 创建自定义类型
11	游标	支持	• 支持隐式游标 • 支持显式游标 • 支持自定义游标类型 • 支持 SYS_REFCURSOR
12	函数嵌套声明	支持	• 支持在函数/存储过程/匿名块中嵌套定义函数或者存储过程
13	PL 系统包	支持	• 包括 DBMS_OUTPUT、DBMS_LOB、DBMS_ASSERT、DBMS_ALERT、DBMS_LOCK、DBMS_METADATA、DBMS_RANDOM、DBMS_SESSION 等

⚠ 注意：

- NOCOPY 仅仅语法兼容 Oracle，内核没有实现 OUT 参数按照 By Reference 来传递。
- 不支持条件编译、PL 的编译优化级别（PLSQL_OPTIMIZE_LEVEL）。

系统视图

支持多个 Oracle 系统视图。请参见 系统视图 。

系统包

支持多个 Oracle 系统包。请参见 系统包 。

数据库对象管理

序号	数据库对象	TDSQL PG 数据库	说明

1	表	支持	• 支持创建表，包括临时表和全局临时表。创建表可以指定约束、索引等信息。 • 支持修改表定义。通过 ALTER TABLE 来添加列，修改列类型，增加或者删除默认值、约束等。 • 支持删除表定义。
2	约束	支持	• 支持 CHECK、UNIQUE、NOT NULL 约束。 • 支持 PRIMARY KEY 约束。 • 支持 FOREIGN KEY 约束。 • 支持 ALTER TABLE 对约束进行启用、禁用。

3	分区表	支持	• 支持 TDSQL PG 语法创建一级和多级分区；不支持用 Oracle 语法方式创建分区表
			• 支持 RANGE、LIST、HASH 分区
			• 支持分区表的局部索引
			• 支持分区表的基本维护:
			  ○ 分区表添加
			  ○ 分区表删除
			  ○ 分区表分裂、合并

4	索引	支持	• 支持 BTREE、HASH、BRIN 索引。支持 gist、gin、spgist 索引。 • 支持索引的创建和删除
5	视图	支持	• 支持创建和删除视图 • 支持 FORCE VIEW • 支持简单视图的 DML 操作。不支持多表视图的 DML
6	序列	支持	• 支持序列的创建和删除 • 支持 NOCACHE、NOORDER 序列
7	同义词	支持	• 支持同义词的创建和删除 • 支持对表、视图、同义词等对象创建同义词
8	DBLINK	支持	• TDSQL PG 支持创建、删除 DBLINK • 支持用 DBLINK 读和写 Oracle 数据库

SQL 数据类型

最近更新时间：2024-11-01 15:20:42

本节主要介绍 TDSQL PG 数据库的 Oracle 兼容模式与原生 Oracle 数据库中 SQL 数据类型的兼容对比信息。我们把 Oracle 数据库的数据类型大体分为几类，以下简单说明与 TDSQL PG 中数据类型的差异：

- 字符类型，包括：

Oracle 数据类型	TDSQL PG 数据库是否支持	说明
CHAR [(size [BYTE | CHAR])]	支持	固定字符长度的字符类型，TDSQL PG 目前仅支持指定 size 为 CHAR（字符长度）。
NCHAR[(size)]	支持	固定字符长度的字符类型，TDSQL PG 目前不区分 UNICODE 字符和非 UNICODE 字符。
VARCHAR2(size [BYTE | CHAR])	支持	可变字节长度的字符类型，TDSQL PG 目前仅支持指定 size 为 BYTE（字节长度）。
NVARCHAR2(size)	支持	可变字符长度的字符类型，TDSQL PG 目前不区分 UNICODE 字符和非 UNICODE 字符。
LONG	支持	可变长度的字符类型，oracle 已废弃，仅供兼容历史版本使用。

- 数字类型，包括：

Oracle 数据类型	TDSQL PG 数据库是否支持	说明
NUMBER[(p[,s])]	支持	精度为 p 并且数值范围为 s 的数字类型，底层为 TDSQL PG 中的 NUMERIC
FLOAT[(p)]	支持	精度为 p 的浮点型，底层为 TDSQL PG 中的 NUMERIC
BINARY_FLOAT	支持	浮点型，底层为 TDSQL PG 中的 NUMERIC

BINARY_DOUBLE	支持	浮点型，底层为 TDSQL PG 中的 NUMERIC

- 时间类型，包括：

Oracle 数据类型	TDSQL PG 数据库是否支持	说明
DATE	支持	时间类型，显示格式取决于 NLS_DATE_FORMAT 格式
TIMESTAMP [(fractional_seconds_precision)]	支持	时间戳类型，fractional_seconds_precision 表示其中秒的小数位数
TIMESTAMP [(fractional_seconds_precision)] WITH TIME ZONE	支持	带时区的时间戳类型，fractional_seconds_precision 同 TIMESTAMP
TIMESTAMP [(fractional_seconds_precision)] WITH LOCAL TIME ZONE	支持	带本地时区的时间戳，fractional_seconds_precision 同 TIMESTAMP
INTERVAL YEAR [(year_precision)] TO MONTH	支持	包含年、月的时间间隔类型，year_precision 是年的数字位数
INTERVAL DAY [(day_precision)] TO SECOND [(fractional_seconds_precision)]	支持	包含天、小时、分钟、秒的时间间隔类型，day_precision 表示天的数字位数，fractional_seconds_precision 同 TIMESTAMP

- RAW 和 LONG RAW 类型

Oracle 数据类型	TDSQL PG 数据库是否支持	说明
RAW(size)	支持	可变字节长度的二进制数据类型
LONG RAW(size)	支持	可变字节长度的二进制数据类型

- 大对象类型，包括：

Oracle 数据类型	TDSQL PG 数据库是否支持	说明
BFILE	支持	外部二进制文件类型

BLOB	支持	二进制大对象
CLOB	支持	字符大对象
NCLOB	支持	同 CLOB，TDSQL PG 目前不区分 UNICODE 字符和非 UNICODE 字符。

- ROWID 和 UROWID 类型

Oracle 数据类型	TDSQL PG 数据库是否支持	说明
ROWID	支持	行标识符，可以唯一标识表中的一行。
UROWID	支持	同 ROWID

- XMLTYPE 类型

Oracle 数据类型	TDSQL PG 数据库是否支持	说明
XMLTYPE	支持	用于存储 XML 数据格式的类型

目前上述的 Oracle 数据类型，除了说明中标出的不同的地方，其他的地方 TDSQL PG 和 Oracle 完全相同的使用方法，更加详细的说明可以参考 Oracle 的官方文档。

SQL 表达式

最近更新时间：2024-11-01 15:20:42

TDSQL PG 兼容绝大部分 Oracle 的 SQL 表达式，具体如下：

序号	Oracle 数据库	TDSQL PG 数据库	说明
1	字面量	支持	支持字符串，数值，日期，时间戳和间隔字面量
2	简单表达式	支持	支持常见的简单表达式，包括 ROWNUM，ROWID 伪列，序列.CURRVAL/NEXTVAL
3	函数表达式	支持	支持函数作为表达式或者表达式的一个元素
4	复合表达式	支持	支持多种表达式组成的复合表达式，其中支持 PRIOR 层次查询
5	CASE 表达式	支持	支持 CASE WHEN ELSE END 表达式
6	列表达式	支持	支持表，视图的列作为表达式或者表达式的一个元素
7	时间戳表达式	支持	支持时间戳表达式，支持带时区
8	时间间隔表达式	支持	支持时间间隔表达式
9	标量子查询表达式	支持	支持标量子查询作为表达式或者表达式的一个元素
10	类型构造函数表达式	支持	支持对象类型的构造函数作为表达式或者表达式的一个元素

运算符

最近更新时间：2025-02-28 10:59:33

TDSQL PG 兼容绝大多数的 Oracle 的运算符，包括算术运算符，连接运算符，集合运算符，其他运算符，具体如下。

算术运算符

序号	Oracle 数据库运算符	TDSQL PG 数据库是否支持	描述
1	正负一元运算符	支持	正负运算
2	加减乘除二元运算符	支持	加减乘除运算

连接运算符

序号	Oracle 数据库运算符	TDSQL PG 数据库是否支持	描述
1	||	支持	连接运算

集合运算符

序号	Oracle 数据库运算符	TDSQL PG 数据库是否支持	描述
1	UNION	支持	集合求并集，结果集去重。
2	UNION ALL	支持	集合求并集，结果集不去重。
3	INTERSECT	支持	集合求交集，结果集去重。
4	MINUS	支持	集合求差集，结果集去重。

多集合运算符

序号	Oracle 数据库运算符	TDSQL PG 数据库是否支持	描述
1	MULTISET UNION ALL	支持	集合类型求并集，结果不去重。
2	MULTISET UNION DISTINCT	支持	集合类型求并集，结果去重。

3	MULTISET INTERSECT ALL	支持	集合类型求交集，结果不去重。
4	MULTISET INTERSECT DISTINCT	支持	集合类型求交集，结果去重。
5	MULTISET EXCEPT ALL	支持	集合类型求差集，如果有重复的元素，只移出第一个匹配项。
6	MULTISET EXCEPT DISTINCT	支持	集合类型求差集，结果去重。

> **注意：**
> 多集合运算符左右两侧必须是元素类型为 SQL 基础类型且不超过二维的 SCHEMA 对象。

比较操作符

序号	Oracle 数据库运算符	TDSQL PG 数据库是否支持	描述
1	=、<>、>、<、>=、<=	支持	相等、不相等、大于、小于、大于等于、小于等于
2	< >、> =、< = 、! =	支持	带空格的不相等、大于等于、小于等于、不相等

NULL 判断操作符

序号	Oracle 数据库运算符	TDSQL PG 数据库是否支持	描述
1	IS NULL	支持	是否为 NULL
2	IS NOT NULL	支持	是否非 NULL

用户自定义操作符

序号	Oracle 数据库	TDSQL PG 数据库是否支持	描述
1	CREATE OPERATOR	支持	创建用户自定义操作符

内建函数

最近更新时间：2025-02-28 10:59:33

本节主要介绍 TDSQL PG 的 Oracle 模式中与原生 Oracle 内建函数的详细兼容对比信息。

数字函数

序号	Oracle 数据库函数	TDSQL PG 数据库是否支持	描述
1	ABS(n)	支持	返回指定参数 n 的绝对值。
2	ACOS(n)	支持	返回指定参数 n 的反余弦值。
3	ASIN(n)	支持	返回指定参数 n 的反正弦值。
4	ATAN(n)	支持	返回指定参数 n 的反正切值。
5	ATAN2(n1, n2)	支持	返回指定的参数 n1 和参数 n2 的反正切。
6	BITAND(expr1, expr2)	支持	返回两个参数 expr1, expr2 按位与运算后的结果。
7	CEIL(n)	支持	返回大于等于指定参数 n 的最小整数。
8	COS(n)	支持	返回指定参数 n 的余弦。
9	COSH(n)	支持	返回指定参数 n 的双曲余弦。
10	EXP(n)	支持	返回自然常数 e 的指定参数 n 的次幂。
11	FLOOR(n)	支持	返回小于或等于指定参数 n 的最大整数值。
12	LN(n)	支持	返回指定参数 n 的自然对数。
13	LOG(n2, n1)	支持	返回参数 n2 基于 参数 n1 的对数。
14	MOD(n2, n1)	支持	返回参数 n2 除以 参数 n1 的余数。
15	NANVL(n2, n1)	支持	如参数 n2 为 NaN 则返回 参数 n1，

			否则返回参数 n2。
16	POWER(n2, n1)	支持	返回参数 n2 的参数 n1 次幂。
17	REMAINDER(n2, n1)	支持	返回参数 n2 除以参数 n1 的余数。
18	ROUND(n, integer)	支持	将给定的数字 n 四舍五入到给定的小数位数 integer。
19	SIGN(n)	支持	返回指定参数 n 的符号，如-1，0或1，表示负数，零或正数。
20	SIN(n)	支持	返回指定弧度值 n 的正弦。
21	SINH(n)	支持	返回指定数值 n 的双曲正弦。
22	SQRT(n)	支持	返回指定参数 n 的平方根。
23	TAN(n)	支持	返回指定弧度 n 的正切。
24	TANH(n)	支持	返回指定数值 n 的双曲正切。
25	TRUNC(n1, n2)	支持	将指定的数字 n1 截断到指定的精度 n2（默认为0）并返回结果。
26	WIDTH_BUCKET(numeric <expr>, numeric <min_value>, numeric <max_value>, int <num_buckets>)	支持	返回指定值 expr 位于指定范围 [min_value, max_value]和个数 num_buckets 的桶中的位置。

字符串函数

序号	Oracle 数据库的函数	TDSQL PG 数据库是否支持	描述
1	• CHR(n) • CHR(n USING NCHAR_CS)	支持	返回指定参数 n 对应的字符，如指定 `USING NCHAR_CS`，则返回国家字符集中的值。

2	CONCAT(char1, char2)	支持	返回一个字符串，由两个参数值 char1 和 char2 拼接结果。
3	INITCAP(char)	支持	返回一个字符串，它将参数指定的字符串中的每个单词的首字母转为大写，其他字母转为小写。
4	LOWER(char)	支持	返回给定字符串的小写形式。
5	LPAD(expr1, n, expr2)	支持	返回一个字符串，它使用给定的字符序列填充到原字符串的左侧，使其具有指定的长度。
6	LTRIM(char, set)	支持	返回一个删除任何指定的前导字符（默认为空格）的字符串。
7	NCHAR(number)	支持	返回一个字符，是给定的整数参数对应的 NVARCHAR2 类型的字符
8	NLS_INITCAP(char, 'nlsparam')	支持	返回一个字符串，它将参数指定的字符串中的每个单词的首字母转为大写，其他字母转为小写。
9	NLS_LOWER(char, 'nlsparam')	支持	返回给定字符串的小写形式。
10	NLS_UPPER(char, 'nlsparam')	支持	返回给定字符串的大写形式。
11	NLSSORT(char, 'nlsparam')	支持	返回给定字符串的排序规则键以及显式或隐式指定的排序规则。
12	• REGEXP_REPLACE(source_char, pattern) • REGEXP_REPLACE(source_char, pattern, replace_string) • REGEXP_REPLACE(source_char, pattern, replace_string, position) • REGEXP_REPLACE(source_char, pattern, replace_string,	支持	在一个字符串中 source 使用新内容 replace_string 替换一个和指定的正则表达式 pattern 匹配的内容，返回替换后的内容。 • 可通过 position 指定开始搜索的起始位置。默认为 1。 • 可指定 occurrence 返回第几次出现结果。默认为 0，即全部替换。 • 可指定 match_param 执行采用的匹配模式,如 'i' 指定不区分大小写的匹配，即使确定的条件排序规则区分大小写。

	position, occurrence) • REGEXP_REPLACE(source_char, pattern, replace_string, position, occurrence, match_param)		
13	• REGEXP_SUBSTR(source_char, pattern) • REGEXP_SUBSTR(source_char, pattern, position) • REGEXP_SUBSTR(source_char, pattern, position, occurrence) • REGEXP_SUBSTR(source_char, pattern, position, occurrence, match_param) • REGEXP_SUBSTR(source_char, pattern, position, occurrence, match_param, subexpr)	支持	从一个给定的源字符串 source_char 中搜索并返回一个与给定的正则表达式 pattern 匹配的字符串。 • 可指定 position 表示开始搜索的起始位置。默认为1。 • 可指定 occurrence 返回第几次出现的结果。默认为1。 • 可指定 match_param 执行采用的匹配模式,如 'i' 指定不区分大小写的匹配，即使确定的条件排序规则区分大小写。
14	REPLACE(char, search_string [, replacement_string])	支持	返回一个字符串，它将源字符串中的所有的 search_string 用 replacement_string 替换掉。
15	RPAD(str, len [, padstr])	支持	返回一个字符串，把指定的字符序列 padstr （默认为空格）填充到原字符串 str 的右侧，使其具有指定的长度 len。
16	RTRIM(str [, set])	支持	返回一个字符串，其为删除 str 指定的尾随字符 set （默认为空格）的字符串。
17	SOUNDEX(str)	支持	将普通字符串 str 转换为 SOUNDEX 字符串，Soundex 是一种语音算法。如果两个单词听起来相

			同，则它们应具有相同的 Soundex 字符串。
18	SUBSTR(char, position [, substring_length])	支持	从一个字符串 char 中返回一个从指定位置 position 开始的指定长度substring_length（如果不指定，则提取到原字符串的结尾）的子字符串。
19	TRANSLATE(string, from_string, to_string)	支持	返回一个翻译后的字符串，将 string 中的所有在 from_string 指定的字符翻译成在 to_string 中对应的字符。
20	TRIM(str) TRIM(trim_character FROM str) TRIM({BOTH / LEADING / TRAILING} trim_character FROM str)	支持	返回一个删除任何指定的前导和尾随字符（默认为空格）的字符串。 • 如果不指定 `BOTH, LEADING,TRAILING`，则默认值为 `BOTH`。 • 如果不指定 `trim_character`，则默认值为一个空格。
21	UPPER(str)	支持	返回指定字符串 str 的大写形式。
22	ASCII(character)	支持	根据数据库字符集返回字符型数值 character 第一个字符的 ascii 码。
23	INSTR(string , substring [, position [, occurrence]])	支持	返回一个整数，子字符串 substring 在字符串 string 中的索引位置。
24	LENGTH(str)	支持	返回指定字符串 str 的长度。

25	REGEXP_COUNT(source_char, pattern)
	REGEXP_COUNT(source_char, pattern, position)
	REGEXP_COUNT(source_char, pattern, position, match_param)
		支持	返回一个整数，指定的正则表达式 pattern 模式在一个源字符串 source_char 中出现的次数。
			• position 可选的, 是一个整数，表示开始搜索的起始位置。默认为 `1`。
			• match_param 可选的, 表示执行匹配采用的模式，包括:
			  ○ `'i'` 指定不区分大小写的匹配，即使确定的条件排序规则区分大小写。
			  ○ `'c'` 指定区分大小写和区分重音的匹配，即使确定的条件排序规则不区分大小写或不区分重音。
			  ○ `'n'` 允许句点（`.`）（匹配任意字符）匹配换行符。如果省略此参数，则句点与换行符不匹配。
			  ○ `'m'` 将源字符串视为多行。

			○ 'x' 忽略空白字符。
26	REGEXP_INSTR(source_char, pattern) REGEXP_INSTR(source_char, pattern, position) REGEXP_INSTR(source_char, pattern, position, occurrence) REGEXP_INSTR(source_char, pattern, position, occurrence, return_opt) REGEXP_INSTR(source_char, pattern, position, occurrence, return_opt, match_param) REGEXP_INSTR(source_char, pattern, position, occurrence, return_opt, match_param, subexpr)	支持	返回一个整数，其为指定的正则表达式 pattern 模式在一个源字符串 source_char 中匹配的子串的索引。 • position 可选的。它是一个整数，指示开始搜索的起始位置。默认为 1。 • occurrence 可选的。它是一个整数，指示要返回的是第几次出现。默认为1。 • return_opt 可选的。指定返回哪一种位置索引。如果为0，返回匹配的子串的第一个字符的位置索引；如果为1，返回匹配的子串的后面的位置索引。默认为0。 • subexpr 可选的, 从0到9， 0 表示整个正则表达式， 1-9 表示正则表达式中的分组。默认值为 0 。

日期函数

序号	Oracle 数据库的函数	TDSQL PG 数据库是否支持	描述
1	ADD_MONTHS(date, months)	支持	在指定的日期 date 上增加或减少指定月数 months。
2	CURRENT_DATE	支持	返回当前会话时区对应的当前日期。
3	CURRENT_TIMESTAMP	支持	返回当前会话时区中的当前日期和时间。
4	DBTIMEZONE	支持	返回数据库的时区。

5	EXTRACT(field FROM expr)	支持	从日期时间或间隔表达式 expr 中提取并返回指定的日期字段 field 的值。field包括 YEAR, MONTH, DAY, HOUR, MINUTE, SECOND。
6	FROM_TZ(timestamp_value, time_zone_value)	支持	将一个时间戳值 timestamp_value 和时区值 time_zone_value 转换为一个带有时区的时间戳值，如： `FROM_TZ(TIMESTAMP '2024-03-20 13:14:52', '+05:00')` => `2024-03-20 13:14:52.000000000 +05:00`
7	LAST_DAY(date)	支持	返回指定日期 date 所在月份的最后一天。
8	LOCALTIMESTAMP LOCALTIMESTAMP(precision)	支持	返回当前会话时区中的当前日期和时间。precision 可选，表示返回时间值的小数秒精度。
9	MONTHS_BETWEEN(date1, date2)	支持	返回两个给定日期之间的月数。
10	NEXT_DAY(date, char)	支持	返回晚于指定日期的指定第一个工作日的日期， 如 `NEXT_DAY('2024-03-20', 'Tuesday')` => `2024-03-26 00:00:00`
11	NUMTODSINTERVAL(n, 'interval_unit')	支持	将指定的数字 n 转为 `INTERVAL DAY TO SECOND` 文本。`interval_unit` 包括 `DAY`，`HOUR`，`MINUTE`，`SECOND` 等。
12	NUMTOYMINTERVAL(n, 'interval_unit')	支持	将指定的数字转为 `INTERVAL YEAR TO MONTH` 文本。`interval_unit` 包括 `'YEAR'` 和 `'MONTH'`。
13	ROUND(date) ROUND(date, fmt)	支持	将指定的日期 date 四舍五入到指定的单位。`fmt` 可选，指定四舍五入的单位，如未指定此参数，`date` 将被四舍五入到最近的天， 如 `YYYY`，`YEAR`，`MONTH` 等。
14	SESSIONTIMEZONE	支持	返回当前会话的时区的值，返回类型是时区偏移量（格式 `[+|-]TZH:TZM` ）或时区区域名称。
15	SYS_EXTRACT_UTC(datetime_with_timezone)	支持	从具有时区偏移量或者时区名的日期时间值 datetime_with_timezone 中提取 UTC。

16	SYSDATE	支持	返回数据库服务器所在的操作系统设置的当前日期和时间。
17	SYSTIMESTAMP	支持	返回数据库服务器所在的操作系统设置的当前日期和时间，包括小数秒和时区。
18	TO_CHAR(expr [, fmt [, 'nlsparam']])	支持	将指定的日期时间或者间隔表达式 expr 根据指定的格式 fmt 转化为字符串。
19	TO_DSINTERVAL(str, [DEFAULT return_value ON CONVERSION ERROR])	支持	将指定的字符串 str 参数转为一个 `INTERVAL DAY TO SECOND` 类型的值。
20	TO_TIMESTAMP(str [DEFAULT return_value ON CONVERSION ERROR] [, fmt [, 'nlsparam']])	支持	将指定的字符串 str 转为一个 `TIMESTAMP` 类型的值。
21	TO_TIMESTAMP_TZ(str [DEFAULT return_value ON CONVERSION ERROR] [, fmt [, 'nlsparam']])	支持	将指定的字符串 str 参数转为一个 `TIMESTAMP WITH TIME ZONE` 类型的值。
22	TO_YMINTERVAL(str, [DEFAULT return_value ON CONVERSION ERROR])	支持	将指定的字符串 str 转为一个 `INTERVAL MONTH TO YEAR` 类型的值。
23	TRUNC(date) TRUNC(date, fmt)	支持	将指定的日期截断到指定的单位。
24	TZ_OFFSET(tz)	支持	根据语句的执行日期返回与参数对应的时区偏移量。

通用比较函数

序号	Oracle 数据库的函数	TDSQL PG 数据库是否支持	描述

1	GREATEST(expr [, expr]...)	支持	返回给定的参数列表中的最大值。
2	LEAST(expr [, expr]...)	支持	返回给定的参数列表中的最小值。

转换函数

序号	Oracle 数据库的函数	TDSQL PG 数据库是否支持	描述
1	ASCIISTR(str)	支持	以数据库字符集返回给定字符串的 ASCII 版本。
2	BIN_TO_NUM(expr [, expr]...)	支持	返回由参数指定的位向量转换为的数字。
3	CAST({ expr | MULTISET (subquery) } AS type_name [DEFAULT return_value ON CONVERSION ERROR] [, fmt [, 'nlsparam']])	支持	将指定的参数 expr 从一种类型转换为另一种类型 type_name。
4	CHARTOROWID(rowid_str)	支持	将指定的字符串值 rowid_str 转换为 `ROWID` 数据类型。
5	CONVERT(char, dest_char_set[, source_char_set])	支持	将指定的字符串 char 从一个字符集 source_char_set(默认值为数据库字符集)转为另一个字符集 dest_char_set。
6	HEXTORAW(str)	支持	将指定参数 str 的十六进制值转为原始值。
7	RAWTOHEX(raw)	支持	将原始值转换为一个包含其十六进制表示的字符值。
8	ROWIDTOCHAR(rowid)	支持	将给定的 rowid 值转为 `VARCHAR2` 类型。
9	TO_BINARY_DOUBLE (expr [DEFAULT return_value ON	支持	将给定的表达式转换为双精度浮点数。

	CONVERSION ERROR] [, fmt [, 'nlsparam']])		
10	TO_BINARY_FLOAT(expr [DEFAULT return_value ON CONVERSION ERROR] [, fmt [, 'nlsparam']])	支持	将给定的表达式转换为浮点数。
11	TO_CHAR (character)	支持	将 `NCHAR`、`NVARCHAR2`、`CLOB` 或 `NCLOB` 数据转换为数据库字符集。
12	TO_CHAR (datetime)	支持	将给定的日期时间或者间隔值根据指定的格式转化为字符串。
13	TO_CHAR (number)	支持	使用可选的格式参数将给定的数值转换为 `VARCHAR2` 数据类型的值。
14	TO_DATE	支持	根据可选的格式将给定的字符串日期转为 `DATE` 数据类型的值。
15	TO_DSINTERVAL(str, [DEFAULT return_value ON CONVERSION ERROR])	支持	将给定的字符串参数转为一个 `INTERVAL DAY TO SECOND` 类型的值。
16	TO_NCLOB(lob_column | char)	支持	将 LOB 列或其他字符字符串中的 `CLOB` 值转换为 `NCLOB` 值。
17	TO_MULTI_BYTE(char)	支持	将给定的参数中的所有单字节字符转换为相应的多字节字符，并返回与参数相同数据类型的值。
18	TO_NUMBER(expr [DEFAULT return_value ON CONVERSION ERROR] [, fmt [, 'nlsparam']])	支持	将给定的参数转换为 `NUMBER` 数据类型的值。
19	TO_SINGLE_BYTE(char)	支持	将给定的参数中的所有多字节字符转换为相应的单字节字符，并返回与参数相同数据类型的值。
20	TO_TIMESTAMP(str [DEFAULT return_value ON	支持	将给定的字符串参数转为一个 `TIMESTAMP` 类型的值。

	CONVERSION ERROR] [, fmt [, 'nlsparam']])		
21	TO_TIMESTAMP_TZ(str [DEFAULT return_value ON CONVERSION ERROR] [, fmt [, 'nlsparam']])	支持	将给定的字符串参数转为一个 `TIMESTAMP WITH TIME ZONE` 类型的值。
22	TO_YMINTERVAL(str, [DEFAULT return_value ON CONVERSION ERROR])	支持	将给定的字符串参数转为一个 `INTERVAL MONTH TO YEAR` 类型的值。

编码解码函数

序号	Oracle 数据库的函数	TDSQL PG 数据库是否支持	描述
1	DUMP(expr[, return_fmt [, start_position [, length]]])	支持	返回一个 VARCHAR2 值，其中包含 expr 的数据类型代码、字节长度和内部表示。
2	STANDARD_HASH(expr [, 'method'])	支持	使用由美国国家标准与技术研究所定义和标准化的多个哈希算法之一，为给定的表达式计算哈希值。method 有效算法为 `SHA1` 、 `SHA256` 、 `SHA384` 、 `SHA512` 和 `MD5` 。如果省略此参数，则使用 `SHA1` 。
3	VSIZE(expr)	支持	返回给定表达式的内部表示中的字节数。
4	DECODE(expr, search, result [, search, result]... [, default])	支持	根据一个或多个给定的映射关系将给定的参数解码并返回解码后的值。
5	ORA_HASH(expr [, max_bucket [, seed_value]])	支持	计算给定表达式的哈希值，确定哈希函数返回的最大桶值。可指定0到 4294967295之间的任何值。默认值为4294967295。

环境和标识符函数

序号	Oracle 数据库的函数	TDSQL PG 数据库是否支持	描述
1	UID	支持	返回当前会话的用户的唯一标识。
2	USER	支持	返回当前会话的用户的名称。
3	USERENV	支持	返回用户环境信息。

空值相关的函数

序号	Oracle 数据库的函数	TDSQL PG 数据库是否支持	描述
1	COALESCE(expr1[, expr2 ...])	支持	返回参数列表中的第一个非 NULL 值。
2	LNNVL(condition)	支持	如果条件为 `TRUE` 就返回 `FALSE` 。如果条件为 `FALSE` 或者 `UNKNOWN` 就返回 `TRUE` 。
3	NANVL(num1, num2)	支持	如果 `num1` 是 `NaN` ， 返回 `num2` ，否则返回 `num1` 。
4	NULLIF(expr1, expr2)	支持	比较两个参数，如果两个参数相等就返回 `NULL` ，否则返回第一个参数。
5	NVL(expr1, expr2)	支持	如果 `expr1` 为 `NULL` ，则返回 `expr2` 。如果 `expr1` 不为 `NULL` ，则 `NVL` 返回 `expr1` 。
6	NVL2(expr1, expr2, expr3)	支持	如果 `expr1` 不为 `NULL` ， 则返回 `expr2` ，否则返回 `expr3` 。

JSON 函数

序号	Oracle 数据库的函数	TDSQL PG 数据库	描述

		是否支持	
1	JSON_OBJECT	支持	返回一个 JSON 对象，其成员为参数指定的键值对。

XML 函数

序号	Oracle 数据库的函数	TDSQL PG 数据库是否支持	描述
1	EXTRACT (XMLType_instance, XPath_string, namespace_string)	支持	按 XPath_string 提取 XMLType_instance 内容，返回类型是 XMLtype。
2	EXTRACTVALUE (XMLType_instance, XPath_string, namespace_string)	支持	按 XPath_string 提取 XMLType_instance 中的 XML 值。
3	PATH （correlation_integer ）	支持	返回指向父条件中指定的资源的相对路径。
4	XMLAGG	支持	XMLAgg 是一个聚合函数， 采用 XML 片段的集合并返回聚合的 XML 文档。
5	XMLEXISTS	支持	检查给定的 XQuery 表达式是否返回非空 XQuery 序列。 如果是，返回 TRUE； 否则，返回 FALSE。

聚合函数 && 分析函数

序号	Oracle 数据库的函数	TDSQL PG 数据库是否支持	描述
1	AVG	支持	求指定表达式的平均值。
2	COUNT	支持	统计查询到的行数。

3	LISTAGG	支持	对于查询中的每一组，LISTAGG 依据 ORDER BY 表达式　排序该组的列，然后将这些值串联成单一字符串。
4	MAX	支持	求最大值。
5	MIN	支持	求最小值。
6	SUM	支持	求和。

系统视图

最近更新时间：2024-11-01 15:20:42

本节主要介绍 TDSQL PG 数据库与原生 Oracle 数据库中静态视图的详细兼容列表。

静态视图

TDSQL PG 数据库兼容 Oracle 数据库的静态视图列表说明如下：

序号	Oracle 数据库	TDSQL PG 数据库	说明
1	ALL_VIEWS	支持	描述当前用户可以访问的所有视图。
2	ALL_USERS	支持	当前用户可访问的所有数据库用户。
3	ALL_TABLES	支持	描述当前用户可访问的关系表。
4	ALL_TAB_COMMENTS	支持	当前用户可访问的表或视图的所有列注释
5	ALL_TAB_COLUMNS	支持	当前用户可访问的表，视图列，过滤系统隐藏列和虚拟列
6	ALL_SYNONYMS	支持	描述当前用户可访问的所有同义词。
7	ALL_SEQUENCES	支持	描述当前用户可访问的所有序列。
8	ALL_INDEXES	支持	描述当前用户可访问的所有表上的索引信息。
9	ALL_IND_COLUMNS	支持	查看用户可访问的所有表的索引的索引列信息。
10	ALL_CONSTRAINTS	支持	显示用户可访问的所有表约束定义
11	ALL_CONS_COLUMNS	支持	查看用户可访问的所有表的约束中的列信息。
12	ALL_COL_COMMENTS	支持	当前用户可访问的表或视图的列的注释
13	ALL_ARGUMENTS	支持	当前用户可访问的函数和过程的参数列表
1	DBA_VIEWS	支持	描述数据库中的所有视图。

4			
15	DBA_USERS	支持	描述数据库的所有用户，并且比 ALL_USERS 包含更多的列。
16	DBA_TABLES	支持	描述数据库中的所有关系表。
17	DBA_TAB_COMMENTS	支持	显示数据库中所有表和视图的注释。
18	DBA_TAB_COLUMNS	支持	描述数据库中所有表，视图的列，其列与“ALL_TAB_COLUMNS”中的列相同。
19	DBA_SYNONYMS	支持	描述数据库中的所有同义词。
20	DBA_SEQUENCES	支持	描述数据库中的所有序列。
21	DBA_INDEXES	支持	描述数据库中的所有索引。
22	DBA_IND_COLUMNS	支持	描述数据库中所有表上的所有索引列信息。
23	DBA_CONSTRAINTS	支持	显示当前数据库上所有约束
24	USER_VIEWS	支持	描述当前用户拥有的视图，不显示“OWNER”列。
25	USER_USERS	支持	描述当前用户，并且包含的列多于 ALL_USERS。
26	USER_TABLES	支持	描述了当前用户拥有的关系表。
27	USER_TAB_COMMENTS	支持	当前用户拥有的表和视图上列的注释，不显示“OWNER”列。
28	USER_TAB_COLUMNS	支持	描述当前用户拥有的表，视图列，不显示“OWNER”列。
29	USER_SYNONYMS	支持	描述当前用户拥有的同义词，不显示 OWNER 列。
30	USER_SEQUENCES	支持	描述了当前用户拥有的所有序列，不显示 SEQUENCE_OWNER 列。

31	USER_INDEXES	支持	描述当前用户拥有的所有索引。
32	USER_IND_COLUMNS	支持	描述当前用户拥有的索引的列以及当前用户拥有的表的索引的列，不显示 INDEX_OWNER 或 TABLE_OWNER 列。
33	USER_CONSTRAINTS	支持	显示当前用户所拥有的所有约束信息
34	USER_CONS_COLUMNS	支持	显示用户约束列信息
35	USER_COL_COMMENTS	支持	当前用户拥有的表或视图的列的注释，不显示 OWNER 字段

系统包

最近更新时间：2024-11-06 17:25:02

系统包指的是 TDSQL PG 数据库提供的兼容 Oracle 功能的一系列内置 PL/SQL 程序包，包含程序、过程、函数、类型和变量，用于执行各种数据库级别的任务，如数据访问、数据处理、安全性、性能监控和数据库管理等。举例常见的 TDSQL PG 系统包如 DBMS_OUTPUT，其用于在 PL/SQL 代码块或存储过程中向用户显示信息。

使用方法如下：

```
---在支持的客户端工具中输出 "Hello, TDSQL PG!" 字符串。
BEGIN
  DBMS_OUTPUT.PUT_LINE('Hello, TDSQL PG!');
END;
/
```

TDSQL PG 目前兼容的 oracle 系统包列表如下表所示。

序号	Oracle 数据库	TDSQL PG 数据库	说明
1	DBMS_APPLICATION_INFO	支持	为执行上下文设置特定的模名和信息，方便后续的问题定位和性能跟踪。
2	DBMS_SQL	部分支持	提供了一套类似 OCI 的接口，允许用户使用该包模拟 OCI 的方式来操作动态 SQL。
3	DBMS_JOB	部分支持	使用该包来执行一些定时任务。
4	DBMS_OUTPUT	支持	使用该包来进行信息输出，方便调试和问题定位。
5	DBMS_ASSERT	支持	该包主要用来验证特定字符串是否是合乎语法语义规则，不同的字函数有不同的验证规则。
6	DBMS_ALERT	支持	该包用来订阅和发送消息，结合 TRIGGER 等使用，可以感知数据库发生的数据变化。
7	DBMS_PIPE	支持	使用该包来创建一个消息管道用来进行消息传递。

8	DBMS_RANDOM	支持	使用该包来生成一串符合正态分布的随机数，并能基于该随机数生成一些随机字符串。
9	DBMD_UTILITY	部分支持	该包提供了一些工具函数，如 FORMAT STACK 等，具体需要见该包的成员函数。
10	DBMS_LOB	部分支持	该包提供了一些工具函数用来操作 LOB 数据类型。注意由于 TDSQL PG 和 ORACLE 底层实现的不同，某些函数可能未实现或行为和 ORACLE 不一致。
11	DBMS_LOCK	部分支持	该包提供了一些工具函数，可以用来向 lock manager 获取锁，帮忙在 PL 中写一些需要锁支持的逻辑. 注意目前只支持 sleep 函数。
12	DBMS_SESSION	部分支持	搞包提供工具函数用来操纵 SESSION 内的部分数据。注意 TDSQL PG 目前只支持UNIQUE_SESSION_ID，RESET_PACKAGE，FREE_UNUSED_USER_MEMORY 三个函数。
13	DBMS_STATS	部分支持	该包提供工具函数用来进行统计信息搜集。
14	DBMS_METADATA	部分支持	该包提供一些工具函数，用来获取系统内部的一些 SCHEMA 信息。
15	DBMS_ROWID	部分支持	该包提供了一些工具函数，来创建和解析 rowid。由于 TDSQL PG 和 oracle 底层实现的不同，某些函数行为不一定相同。
16	UTL_FILE	部分支持	改包提供一些工具函数，用来读写操作系统的文件。
17	UTL_RAW	部分支持	该包提供一些工具函数，用来操作 RAW 数据类型。
18	XMLDOM	部分支持	该包提供一些 XML DOCUMENT 处理函数，目前支持 getElementsByTagName，getLength，item，getChildNodes，getFirstChild，getNodeValue，getNodeName，FREEDOCUMENT 等。
19	XMLPARSER	部分支持	该包提供一些 XML 解析函数，目前支持 parseClob，getDocument，freeParser 等。

存储过程开发
存储过程语法介绍

最近更新时间：2024-12-13 14:30:42

建立存储语法

```
CREATE [OR REPLACE] PROCEDURE [模式名.]存储过程 ([参数模式 [参数名] 数据类型
[default 默认值] [,…]])  AS
[标签]
[DECLARE
    --变量定义]
BEGIN
--注释
/*注释*/
    --语句执行
END;
[标签]
LANGUAGE PLPGSQL;
```

[OR REPLACE] 更新存储介绍

带 OR REPLACE 的作用，在建立存储过程时若存在则替换，建立存储时不带 OR REPLACE 关键字，则遇到函数已经存系统则会报错，如下所示。

```
postgres=#  select prosrc from pg_proc where proname='proc_1';
            prosrc
-------------------------------
                               +
 begin                         +
    raise notice 'Hello tdsql_pg';+
 end;                          +

(1 row)

postgres=# CREATE OR REPLACE PROCEDURE proc_1() AS
$$
begin
   raise notice 'Hello, tdsql_pg';
end;
$$
```

```
LANGUAGE PLPGSQL;
CREATE PROCEDURE
postgres=# select prosrc from pg_proc where proname='proc_1';
             prosrc
---------------------------------
                                 +
 begin                           +
    raise notice 'Hello, tdsql_pg';+
 end;                            +

(1 row)

postgres=#

postgres=# call proc_1();
NOTICE:  Hello, tdsql_pg
CALL
postgres=#
```

[模式名.]存储过程名介绍

建立存储过程，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;
      nspname        | nspowner |                  nspacl
---------------------+----------+-------------------------------------
 pg_toast            |       10 |
 pg_temp_1           |       10 |
 pg_toast_temp_1     |       10 |
 pg_catalog          |       10 | {postgres=UC/postgres,=U/postgres}
 public              |       10 | {postgres=UC/postgres,=UC/postgres}
 information_schema  |       10 | {postgres=UC/postgres,=U/postgres}
(6 行记录)

postgres=# show search_path;
  search_path
---------------
 "$user",public
(1 行记录)

postgres=# select pg_namespace.nspname,pg_proc.prosrc from
pg_proc,pg_namespace where
```

```
pg_proc.pronamespace=pg_namespace.oid and pg_proc.proname='proc_1';
 nspname |              prosrc
---------+---------------------------------
 public  |                                 +
         | begin                           +
         |    raise notice 'Hello, tdsql_pg';+
         | end;                            +
         |
(1 row)
```

因为 $user 模式不存在，所以存在 public 模式下。

存储过程与函数不能同名

如创建一个与函数同名的存储过程会提示 function xxx already exists with same argument types。

```
postgres=# CREATE OR REPLACE FUNCTION proc_1() RETURNS void AS
$$
begin
   raise notice 'Hello tdsql_pg';
end;
$$
LANGUAGE PLPGSQL;
CREATE FUNCTION

postgres=# CREATE  PROCEDURE proc_1() AS
$$
begin
   raise notice 'Hello tdsql_pg';
end;
$$
LANGUAGE PLPGSQL;
ERROR:  function "proc_1" already exists with same argument types
postgres=#
```

如果要替换，则提示。

```
postgres=# CREATE OR REPLACE PROCEDURE proc_1() AS
$$
begin
   raise notice 'Hello tdsql_pg';
end;
$$
```

```
LANGUAGE PLPGSQL;
ERROR:  cannot change routine kind
DETAIL:  "proc_1" is a function.
postgres=#
```

删除存储过程

删除不带参数的存储过程

```
postgres=# CREATE OR REPLACE PROCEDURE proc_1() AS
$$
begin
   raise notice 'Hello tdsql_pg';
end;
$$
LANGUAGE PLPGSQL;
CREATE PROCEDURE

postgres=# drop procedure proc_1 ( );
DROP PROCEDURE
postgres=#
```

删除带参数的存储过程

```
postgres=# CREATE OR REPLACE PROCEDURE proc_1(a_int int) AS
$$
begin
   raise notice '%',a_int;
end;
$$
LANGUAGE PLPGSQL;
CREATE PROCEDURE

postgres=# drop procedure proc_1 ( a_int int);
DROP PROCEDURE
postgres=#
```

也可以只指定参数的类型即可。

```
postgres=# drop procedure proc_1 (int);
DROP PROCEDURE
```

```
postgres=#
```

存储过程修改名称

修改不带参数的存储过程名称

```
postgres=# CREATE OR REPLACE PROCEDURE proc_1() AS
$$
begin
   raise notice 'Hello tdsql_pg';
end;
$$
LANGUAGE PLPGSQL;
CREATE PROCEDURE

postgres=# alter procedure proc_1() rename to proc_1_1;
ALTER PROCEDURE
postgres=#
```

修改带参数的存储过程名称

```
postgres=# CREATE OR REPLACE PROCEDURE proc_1(a_int int) AS
$$
begin
   raise notice '%',a_int;
end;
$$
LANGUAGE PLPGSQL;
CREATE PROCEDURE

postgres=# alter procedure proc_1 (a_int int) rename to proc_1_1;
ALTER PROCEDURE
```

修改存储过程所属 schema

修改不带参数的存储过程 schema

```
postgres=# CREATE OR REPLACE PROCEDURE public.proc_1() AS
$$
begin
   raise notice 'Hello tdsql_pg';
```

```
end;
$$
LANGUAGE PLPGSQL;
CREATE PROCEDURE
postgres=#

postgres=# alter procedure public.proc_1() set schema myche;
ALTER PROCEDURE
postgres=#
```

修改带参数的存储过程 schema

```
postgres=# CREATE OR REPLACE PROCEDURE public.proc_1(a_int int) AS
$$
begin
   raise notice '%',a_int;
end;
$$
LANGUAGE PLPGSQL;
CREATE PROCEDURE

postgres=# alter procedure public.proc_1 (int) set schema myche;
ALTER PROCEDURE
postgres=#
```

修改存储过程所属用户

修改不带参数的存储过程所属用户

```
postgres=# CREATE OR REPLACE PROCEDURE public.proc_1() AS
$$
begin
   raise notice 'Hello tdsql_pg';
end;
$$
LANGUAGE PLPGSQL;
CREATE PROCEDURE

postgres=# alter procedure proc_1() owner to tdsql_pg_01_admin;
ALTER PROCEDURE
postgres=#
```

修改不带参数的存储过程所属用户

```
postgres=# CREATE OR REPLACE PROCEDURE proc_1(a_int int) AS
$$
begin
   raise notice '%',a_int;
end;
$$
LANGUAGE PLPGSQL;
CREATE PROCEDURE

postgres=# alter procedure proc_1 (int) owner to tdsql_pg_01_admin;
ALTER PROCEDURE
postgres=#
```

存储过程执行

不带参数

```
postgres=# create or replace procedure p_no_para() as
$$
begin
    raise notice 'no_para procedure';
end;
$$
language plpgsql;
CREATE PROCEDURE
postgres=# call p_no_para();
NOTICE:  no_para procedure
CALL
postgres=#
```

带参数

```
postgres=# create or replace procedure p_para(a_int integer) as
$$
begin
    raise notice 'a_int = %',a_int;
end;
$$
language plpgsql;
CREATE PROCEDURE
```

```
postgres=# call p_para(1);
NOTICE:  a_int = 1
CALL
postgres=#
```

动态 SQL

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

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

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


create or replace procedure p_sql is
```

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

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

```
create table ascii_t(id int, c1 varchar2(100), c2 char(100), c3
nchar(100), c4 nvarchar2(100), c5 clob, c6 nclob, c7 number, c8
smallint, c9 date, c10 timestamp);
insert into ascii_t values(1, 'abc', 'eqw', chr(20)||'da',
chr(30)||'==', '而我却', chr(465)||'饿我去的', 465.89, -100, '2021-07-06
10:15:12', to_timestamp('2021-07-06 14:38:48.680297', 'yyyy-mm-dd
HH24:mi:ss.ff'));
insert into ascii_t values(2, '中国', '大苏打', chr(125)||'ddsaa',
chr(120)||'==', 'daewq', chr(879)||'大苏打ew', 1.89, 128,'2020-08-06
10:15:12', to_timestamp('2021-07-06 14:38:48.680297', 'yyyy-mm-dd
HH24:mi:ss.ff'));
insert into ascii_t values(3, '四届', 'eqw', chr(458)||'打算',
chr(654)||'==', '打算而我却', chr(135)||'而我打算eqw', 0.89, 0, '2020-08-
06 10:15:12', to_timestamp('2021-07-08 14:38:48', 'yyyy-mm-dd
HH24:mi:ss'));
insert into ascii_t values(4, 'abc', '而且我给v的', chr(20)||'da',
chr(445)||'==', '和梵蒂冈', chr(135)||'aeq4556大', -895.89, 11, '2021-
08-06 10:15:12', to_timestamp('2021-08-08 14:38:48', 'yyyy-mm-dd
HH24:mi:ss'));
insert into ascii_t values(5, 'abewqe', 'eqw', chr(34)||'恶趣味',
chr(102)||'==', '日期', chr(798)||'321', -7895.89, 22, '2020-09-06
10:15:12', to_timestamp('2021-07-09 14:38:48', 'yyyy-mm-dd
HH24:mi:ss'));
insert into ascii_t values(6, '打算', 'dasdas', chr(38)||'大师的人情味',
chr(128)||'==', '发', chr(4565)||'', 7851.89, -56, '2020-10-06
```

```
10:15:12', to_timestamp('2021-10-08 14:38:48', 'yyyy-mm-dd
HH24:mi:ss'));
-- 插入空值
insert into ascii_t values(7, '', 'dasdas','', chr(128)||'==', '发',
chr(4565)||'', '', 127, '', '');
insert into ascii_t values(8, '打算', '', chr(38)||'大师的人情味', ''
,'发', '', 7895.89, '', SYSDATE, '');
insert into ascii_t values(9, '', 'dasdas', '', '', '', chr(4565)||'',
7895.89, '', '', to_timestamp('2021-10-08 14:38:48', 'yyyy-mm-dd
HH24:mi:ss'));

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

3、动态执行 CREATE TABLE。

```
create table emp1 as select * from emp;
create table dept_tmp (deptno number(2),dname varchar2(14),loc
varchar2(13));
create or replace package pkg1 is
procedure raise_salary(v1 number,v2 number);
end;
/
create or replace package body pkg1 is
procedure raise_salary(v1 number,v2 number) is
begin
    update emp1 set sal=sal+v2 where empno=v1;
end;
end;
```

```
/
DECLARE
    sql_stmt     VARCHAR2(200);
    plsql_block VARCHAR2(500);
    emp_id       NUMBER(4) := 7566;
    salary       NUMBER(7,2);
    dept_id      NUMBER(2) := 50;
    dept_name    VARCHAR2(14) := 'PERSONNEL';
    location     VARCHAR2(13) := 'DALLAS';
    emp_rec      emp%ROWTYPE;
BEGIN
    EXECUTE IMMEDIATE 'CREATE TABLE using_t (id NUMBER, amt
NUMBER)';
    sql_stmt := 'INSERT INTO dept_tmp VALUES (:1, :2, :3)';
    EXECUTE IMMEDIATE sql_stmt USING dept_id, dept_name, location;
    sql_stmt := 'SELECT * FROM emp WHERE empno = :id';
    EXECUTE IMMEDIATE sql_stmt INTO emp_rec USING emp_id;
   -- plsql_block := 'BEGIN pkg1.raise_salary(:id, :amt); END;';
    plsql_block:='call pkg1.raise_salary(:id, :amt)';
    EXECUTE IMMEDIATE plsql_block USING 7788, 500;
   -- execute immediate v_sql using in vin, in vret;
--    sql_stmt := 'UPDATE emp SET sal = 2000 WHERE empno = :1
--       RETURNING sal INTO :2';
--    EXECUTE IMMEDIATE sql_stmt USING emp_id RETURNING INTO salary;
    EXECUTE IMMEDIATE 'DELETE FROM dept_tmp WHERE deptno = :num'
       USING dept_id;
    EXECUTE IMMEDIATE 'ALTER table dept_tmp add  c1 number';
END;
/
```

参数详细介绍

最近更新时间：2024-10-17 15:26:50

参数模式

存储过程中，参数模式当前只支持 IN（传入参数，这个是默认方式）。

参数引用

无命名参数

```
postgres=# CREATE OR REPLACE PROCEDURE p_unname(text) AS
$$
BEGIN
     raise notice '$1=%',$1;
END;
$$
LANGUAGE PLPGSQL;
CREATE PROCEDURE
postgres=# call ps_unname('tdsql_pg');
NOTICE:  $1=tdsql_pg
CALL
postgres=#
```

给标识符指定别名

```
postgres=# CREATE OR REPLACE PROCEDURE p_specify_name(text) AS
$$
DECLARE
     a_xm ALIAS FOR $1; --a_xm是$1的别名
BEGIN
     raise notice '$1=%',a_xm;
END;
$$
LANGUAGE PLPGSQL;
CREATE PROCEDURE
postgres=# CALL p_specify_name('tdsql_pg');
NOTICE:  $1=tdsql_pg
CALL
postgres=#
```

命名参数

```
postgres=# CREATE OR REPLACE PROCEDURE p_name(a_xm text) AS
$$
BEGIN
    raise notice '$1=%',a_xm;
END;
$$
LANGUAGE PLPGSQL;
CREATE PROCEDURE
postgres=# call p_name('tdsql_pg');
NOTICE:  $1=tdsql_pg
CALL
postgres=#
```

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。

基本类型

```
postgres=# CREATE OR REPLACE PROCEDURE p_base_para (a_int integer,a_str
text) AS
$$
BEGIN
    RAISE NOTICE 'a_int = % ; a_str = %',a_int,a_str;
END;
$$
LANGUAGE PLPGSQL;
CREATE PROCEDURE
postgres=#
CALL  p_base_para(1,'tdsql_pg');
NOTICE:  a_int = 1 ; a_str = tdsql_pg
CALL
postgres=#

postgres=# CREATE OR REPLACE PROCEDURE p_base_array (a_int
integer[],a_str text[]) AS
$$
BEGIN
    RAISE NOTICE 'a_int = % ; a_str = %',a_int,a_str;
END;
```

```
$$
LANGUAGE PLPGSQL;
CREATE PROCEDURE
postgres=# CALL p_base_array(ARRAY[1,2,3],ARRAY['tdsql_pg','pgxz']);
NOTICE:  a_int = {1,2,3} ; a_str = {tdsql_pg,pgxz}
CALL
postgres=#
```

复合类型

```
postgres=# CREATE TYPE public.t_per AS
(
    id integer,
    mc text
);
CREATE TYPE
postgres=# CREATE OR REPLACE PROCEDURE p_type (a_row public.t_per) AS
$$
BEGIN
RAISE NOTICE 'id = % ; mc = %',a_row.id,a_row.mc;
END;
$$
LANGUAGE PLPGSQL;
CREATE PROCEDURE
postgres=# CALL p_type(ROW(1,'tdsql_pg')::public.t_per);
NOTICE:  id = 1 ; mc = tdsql_pg
CALL
postgres=#
```

复合数组

```
postgres=# CREATE OR REPLACE PROCEDURE p_type_array (a_rec
public.t_per[]) AS
$$
BEGIN
    RAISE NOTICE 'a_rec = %',a_rec;
    RAISE NOTICE 'a_rec[1].id = %',a_rec[1].id;
END;
$$
LANGUAGE PLPGSQL;
CREATE PROCEDURE
```

```
postgres=# CALL p_type_array
(ARRAY[ROW(1,'tdsql_pg'),ROW(1,'pgxz')]::public.t_per[]);
NOTICE:  a_rec = {"(1,tdsql_pg)","(1,pgxz)"}
NOTICE:  a_rec[1].id = 1
CALL
postgres=#
```

行类型

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

postgres=# CREATE OR REPLACE PROCEDURE p_row (a_row public.t)  AS
$$
BEGIN
RAISE NOTICE 'id = % ; mc = %',a_row.id,a_row.mc;
END;
$$
LANGUAGE PLPGSQL;
CREATE PROCEDURE
postgres=#CALL p_row(ROW(1,'tdsql_pg'));
NOTICE:  id = 1 ; mc = tdsql_pg
CALL
postgres=#
```

行数组

```
postgres=# CREATE OR REPLACE PROCEDURE p_row_array (a_rec public.t[]) AS
$$
BEGIN
    RAISE NOTICE 'a_rec = %',a_rec;
    RAISE NOTICE 'a_rec[1].id = %',a_rec[1].id;
END;
$$
LANGUAGE PLPGSQL;
CREATE PROCEDURE
postgres=# CALL
p_row_array(array[row(1,'tdsql_pg'),row(1,'pgxz')]::public.t[]);
NOTICE:  a_rec = {"(1,tdsql_pg)","(1,pgxz)"}
NOTICE:  a_rec[1].id = 1
CALL
```

```
postgres=#
```

游标类型

```
postgres=# CREATE OR REPLACE PROCEDURE p_refcursor (a_ref refcursor) AS
$$
DECLARE    v_rec record;
BEGIN
    OPEN a_ref FOR SELECT * FROM t LIMIT 1;
    FETCH a_ref INTO v_rec;
    RAISE NOTICE 'v_rec = % ',v_rec;
END;
$$
LANGUAGE PLPGSQL;
CREATE PROCEDURE
postgres=# CALL p_refcursor('a');
NOTICE:  v_rec = (1,tdsql_pg)
CALL
postgres=#
```

多态类型

```
postgres=# CREATE OR REPLACE PROCEDURE f_any(a_arg anyelement) AS
$$
BEGIN
RAISE NOTICE '%',a_arg;
END;
$$
LANGUAGE PLPGSQL;
CREATE PROCEDURE

postgres=# CALL f_any(1);
NOTICE:  1
CALL

postgres=# CALL f_any('tdsql_pg'::varchar);
NOTICE:  tdsql_pg
CALL
postgres=#

postgres=# SELECT f_any('tdsql_pg'::TEXT);
NOTICE:  tdsql_pg
```

```
f_any
-------

(1 行记录)

postgres=# CALL f_any(ROW(1,'tdsql_pg')::public.t);
NOTICE:  (1,tdsql_pg)
CALL
postgres=#

postgres=# CALL f_any(ARRAY[1,2]::INTEGER[]);
NOTICE:  {1,2}
CALL
postgres=#

postgres=# CALL  f_any(ARRAY[[1,2],[3,4],[5,6]]::INTEGER[][][]);
NOTICE:  {{1,2},{3,4},{5,6}}
CALL
postgres=#
```

注意多态类型参数调用时最好直接声明参数类型，否则有可能出错。

```
postgres=# CREATE OR REPLACE PROCEDURE f_any(a_arg anyarray) AS
$$
BEGIN
    RAISE NOTICE '%',a_arg;
END;
$$
LANGUAGE PLPGSQL;
CREATE PROCEDURE
postgres=# call f_any(ARRAY['tdsql_pg','pgxz']::TEXT[]);
ERROR:  procedure f_any(text[]) is not unique
LINE 1: call f_any(ARRAY['tdsql_pg','pgxz']::TEXT[]);
              ^
HINT:  Could not choose a best candidate procedure. You might need to
add explicit type casts.
postgres=#
```

注意

Anyelement 参数如果写成数组，其意义就跟 anyarray 参数一致，所以 f_any(a_arg anyelement) 与 f_any(a_arg anyarray) 在调用 f_any(ARRAY[1,2]) 时就会出现函数不是唯一化

的错误 (ERROR: function f_any(…) is not unique) 提示。

参数默认值

```
postgres=# CREATE OR REPLACE PROCEDURE p_default_value (a_int INTEGER
DEFAULT 1) AS
$$
BEGIN
    RAISE NOTICE 'a_int = %',a_int;
END;
$$
LANGUAGE PLPGSQL;
CREATE PROCEDURE
postgres=# CALL p_default_value(2);
NOTICE:  a_int = 2
CALL
postgres=# CALL p_default_value();
NOTICE:  a_int = 1
CALL
postgres=#
```

说明

如果原来存在一个 p_default_value () 这样的存储过程，则上面的执行就会出错，因为系统无法清楚到底要执行哪个函数，如下所示。

```
postgres=# CREATE OR REPLACE PROCEDURE p_default_value() AS
$$
BEGIN
    RAISE NOTICE '无参数';
END;
$$
LANGUAGE plpgsql ;
CREATE PROCEDURE
postgres=# CALL p_default_value();
ERROR:  procedure p_default_value() is not unique
LINE 1: CALL p_default_value();
             ^
HINT:  Could not choose a best candidate procedure. You might need to
add explicit type casts.
postgres=#
```

出错提示，p_default_value () 存储过程不是唯一的，这是使用上一个需要特别注意的地方。

变量使用

最近更新时间：2024-10-17 15:26:50

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。

变量使用实例

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。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 :=。

定义一个普通变量

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

```
CALL
postgres=#
```

定义 CONSTANT 变量

```
postgres=# CREATE OR REPLACE PROCEDURE p_constant()  AS
$$
DECLARE
    v_int CONSTANT integer := 1;
BEGIN
    RAISE NOTICE 'v_int = %',v_int;
END;
$$
LANGUAGE plpgsql;
CREATE PROCEDURE
postgres=# CALL p_constant();
NOTICE:  v_int = 1
CALL
```

CONSTANT 不能再次赋值

```
postgres=# CREATE OR REPLACE PROCEDURE p_constant()  AS
$$
DECLARE
    v_int CONSTANT integer := 1;
BEGIN
    RAISE NOTICE 'v_int = %',v_int;
v_int = 10;
END;
$$
LANGUAGE plpgsql;
ERROR:  "v_int" is declared CONSTANT
LINE 7: v_int = 10;
        ^
postgres=#
```

定义 NOT NULL 变量

```
postgres=# CREATE OR REPLACE PROCEDURE p_not_null_var() AS
$$
DECLARE
```

```
    v_int integer NOT NULL := 1;
BEGIN
    RAISE NOTICE 'v_int = %',v_int;
    SELECT NULL INTO v_int;
    RAISE NOTICE 'v_int = %',v_int;
END;
$$
LANGUAGE plpgsql;
CREATE PROCEDURE
postgres=# CALL p_not_null_var();
NOTICE:  v_int = 1
ERROR:  null value cannot be assigned to variable "v_int" declared NOT
NULL
CONTEXT:  PL/pgSQL function p_not_null_var() line 6 at SQL statement
postgres=#
```

定义为NOT NULL变量，则该变量受NOT NULL约束

定义 COLLATE 变量

按 unicode 值对比大小。

```
postgres=# CREATE OR REPLACE PROCEDURE p_collate_unicode() AS
$$
DECLARE
    v_txt1 TEXT COLLATE "C" := '严';
    v_txt2 TEXT COLLATE "C" := '丰';
BEGIN
IF v_txt1 > v_txt2 THEN
    RAISE NOTICE ' % > % ',v_txt1,v_txt2;
ELSE
    RAISE NOTICE ' % > % ',v_txt2,v_txt1;
END IF;
END;
$$
LANGUAGE plpgsql;
CREATE PROCEDURE
postgres=# CALL p_collate_unicode();
NOTICE:   丰 > 严
CALL
postgres=#

postgres=# select '严'::bytea;
```

```
  bytea
----------
 \xe4b8a5
(1 row)

postgres=# select '丰'::bytea;
  bytea
----------
 \xe4b8b0
(1 row)
```

按汉字的拼音对比大小。

```
postgres=# CREATE OR REPLACE PROCEDURE p_collate_pinyin() AS
$$
DECLARE
    v_txt1 TEXT COLLATE "zh_CN.utf8" := '严';
    v_txt2 TEXT COLLATE "zh_CN.utf8" := '丰';
BEGIN
    IF v_txt1 > v_txt2 THEN
    RAISE NOTICE ' % -> % ',v_txt1,v_txt2;
    ELSE
    RAISE NOTICE ' % -> % ',v_txt2,v_txt1;
    END IF;
END;
$$
LANGUAGE plpgsql;
CREATE PROCEDURE
postgres=# CALL p_collate_pinyin();
NOTICE:   严 -> 丰
CALL
postgres=#
```

变量赋值

```
postgres=# CREATE OR REPLACE PROCEDURE p_setval()  AS
$$
DECLARE
    --定义时赋值
    v_int1 integer = 1;
    --使用 :=兼容于plsql
    v_int2 integer := 1;
```

```
    v_txt1 text;
    v_float float8;
    --使用查询赋值
    v_relname text = (select relname FROM pg_class LIMIT 1);
    v_relpages integer;
    v_rec RECORD;
BEGIN
    --在函数体中赋值
    v_txt1 = 'tdsql_pg';
    v_float = random();
    --使用查询赋值的另一种方式
    SELECT relname,relpages INTO v_relname,v_relpages FROM  pg_class
ORDER BY random() LIMIT 1;
    RAISE NOTICE 'v_relname = % , relpages = %',v_relname,v_relpages;
END;
$$
LANGUAGE plpgsql;
CREATE PROCEDURE
postgres=# CALL p_setval();
NOTICE:  v_relname = pg_toast_17220_index , relpages = 1
CALL
postgres=#
```

BULK COLLECT

BULK COLLECT 将一个查询结果集保存起来。

示例1

```
postgres=# create table t5(f1 integer,f2 varchar(10));
CREATE TABLE
postgres=# insert into t5 values(1,'tdsql_pg1');
INSERT 0 1
postgres=# insert into t5 values(2,'tdsql_pg2');
INSERT 0 1

postgres=# create or replace procedure p_bulk_collect()
AS
$$
declare
    TYPE t5list IS TABLE OF t5.f2%TYPE;
    t5s t5list;
BEGIN
    SELECT f2 BULK COLLECT INTO t5s FROM t5;
```

```
    FOR i IN t5s.FIRST .. t5s.LAST
    LOOP
        raise notice '%',t5s[i];
    END LOOP;
END
$$language plpgsql;
NOTICE:  type reference t5.f2%TYPE converted to character varying
CREATE PROCEDURE
postgres=# CALL p_bulk_collect();
NOTICE:  tdsql_pg1
NOTICE:  tdsql_pg2
CALL
```

示例2

```
postgres=# create table tbl_person(id integer, name text, tdd int);
CREATE TABLE
postgres=# insert into tbl_person values(1,'tdsql_pg',1);
insert into tbl_person values(2,'pgxz',1);
INSERT 0 1
postgres=# insert into tbl_person values(2,'pgxz',1);
INSERT 0 1
postgres=# insert into tbl_person values(3,'pg',2);
INSERT 0 1

postgres=# create or replace procedure p_bulkcollect_select_into(a_tdd
integer) AS
$$
declare
   type TAPersonlist is table of tbl_person%rowtype;
   vpa TAPersonlist;
   rp tbl_person%rowtype;
begin
   select * bulk collect into vpa from tbl_person where tdd = a_tdd;
   raise notice 'count=%', vpa.count;
   for i in vpa.first..vpa.last loop
       rp = vpa[i];
       raise notice 'loop=%', i;
       raise notice 'vname=%', rp.name;
   end loop;
   raise notice 'vpa.count=%',vpa.count;
end;
$$language plpgsql;
```

```
CREATE PROCEDURE
postgres=# CALL p_bulkcollect_select_into(1);
NOTICE:  count=2
NOTICE:  loop=1
NOTICE:  vname=tdsql_pg
NOTICE:  loop=2
NOTICE:  vname=pgxz
NOTICE:  vpa.count=2
CALL
postgres=#
```

控制结构

最近更新时间：2024-10-17 15:26:50

判断语句

IF...THEN...END IF

```
postgres=# CREATE OR REPLACE PROCEDURE p_if() AS
$$
BEGIN
    IF random()>0.5 THEN
        RAISE NOTICE '随机数大于0.5';
    END IF;
END;
$$
LANGUAGE plpgsql;
CREATE PROCEDURE
postgres=# CALL p_if();
NOTICE:  随机数大于0.5
CALL
postgres=#
```

IF...THEN...ELSE...END IF

```
postgres=# CREATE OR REPLACE PROCEDURE p_if_else() AS
$$
BEGIN
    IF random()>0.99 THEN
        RAISE NOTICE '随机数大于0.99';
    ELSE
        RAISE NOTICE '随机数小于或等于0.99';
    END IF;
END;
$$
LANGUAGE plpgsql;
CREATE PROCEDURE
postgres=# CALL p_if_else();
NOTICE:  随机数小于或等于0.99
CALL
postgres=#
```

IF...THEN...ELSIF...THEN...ELSE...END IF

```
postgres=# CREATE OR REPLACE PROCEDURE p_if_elsif()  AS
$$
DECLARE
    v_float8 float8 := random();
BEGIN
    IF v_float8>0.99 THEN
        RAISE NOTICE '随机数大于0.99';
    ELSIF v_float8>0.5 THEN
        RAISE NOTICE '随机数大于0.50';
    ELSIF v_float8>0.25 THEN
        RAISE NOTICE '随机数大于0.25';
    ELSE
        RAISE NOTICE '随机数小于或等于0.25';
    END IF;
END;
$$
LANGUAGE plpgsql;
CREATE PROCEDURE
postgres=# CALL p_if_elsif();
NOTICE:  随机数大于0.50
CALL
```

CASE 语句

```
postgres=# CREATE OR REPLACE PROCEDURE p_case()  AS
$$
DECLARE
    v_float8 float8 := random();
BEGIN
    CASE
    WHEN v_float8>0.99 THEN
        RAISE NOTICE '随机数大于0.99';
    WHEN v_float8>0.5 THEN
        RAISE NOTICE '随机数大于0.50';
    WHEN v_float8>0.25 THEN
        RAISE NOTICE '随机数大于0.25';
    ELSE
        RAISE NOTICE '随机数小于或等于0.25';
    END CASE;
END;
```

```
$$
LANGUAGE plpgsql;
CREATE PROCEDURE
postgres=# CALL p_case();
NOTICE:  随机数小于或等于0.25
CALL
postgres=#
```

循环语句

LOOP 循环

```
postgres=# CREATE OR REPLACE PROCEDURE p_loop()  AS
$$
DECLARE
     v_id INTEGER := 1;
BEGIN
    LOOP
        RAISE NOTICE '%',v_id;
        EXIT WHEN random()>0.8;
        v_id := v_id + 1;
    END LOOP ;
END;
$$
LANGUAGE plpgsql;
CREATE PROCEDURE

postgres=# CALL p_loop();
NOTICE:  1
NOTICE:  2
NOTICE:  3
CALL
postgres=#
```

WHILE 循环

```
postgres=# CREATE OR REPLACE PROCEDURE p_while()  AS
$$
DECLARE
    v_id INTEGER := 1;
    v_random float8 ;
BEGIN
```

```
    LOOP
        RAISE NOTICE '%',v_id;
        v_id := v_id + 1;
        v_random := random();
        IF v_random > 0.8 THEN
            RETURN;
        END IF;
    END LOOP ;
END;
$$
LANGUAGE plpgsql;
CREATE PROCEDURE
postgres=# CALL p_while();
NOTICE:  1
CALL
```

FOR 循环

```
postgres=# CREATE OR REPLACE PROCEDURE p_for()  AS
$$
BEGIN
    FOR i IN 1..3 LOOP
        RAISE NOTICE 'i = %',i;
    END LOOP;
END;
$$
LANGUAGE plpgsql;
CREATE PROCEDURE
postgres=# CALL p_for();
NOTICE:  i = 1
NOTICE:  i = 2
NOTICE:  i = 3
CALL

postgres=# CREATE OR REPLACE PROCEDURE p_for_reverse()  AS
$$
BEGIN
    FOR i IN REVERSE 3..1 LOOP
        RAISE NOTICE 'i = %',i;
    END LOOP;
END;
$$
LANGUAGE plpgsql;
```

```
CREATE PROCEDURE
postgres=# CALL p_for_reverse();
NOTICE:  i = 3
NOTICE:  i = 2
NOTICE:  i = 1
CALL
```

使用 REVERSE 递减。

```
postgres=# CREATE OR REPLACE PROCEDURE p_for_by()  AS
$$
BEGIN
    FOR i IN 1..8 BY 2 LOOP
        RAISE NOTICE 'i = %',i;
    END LOOP;
END;
$$
LANGUAGE plpgsql;
CREATE PROCEDURE
postgres=# CALL p_for_by();
NOTICE:  i = 1
NOTICE:  i = 3
NOTICE:  i = 5
NOTICE:  i = 7
CALL
postgres=#
```

使用 BY 设置步长。

FOR 循环查询结果

```
postgres=# CREATE OR REPLACE PROCEDURE p_for_record()  AS
$$
DECLARE
    v_rec RECORD;
BEGIN
    FOR v_rec IN SELECT relname,relkind FROM pg_class limit 2 LOOP
        RAISE NOTICE '%',v_rec;
    END LOOP;
END;
$$
LANGUAGE plpgsql;
```

```
CREATE PROCEDURE
postgres=# CALL p_for_record();
NOTICE:  (pg_stat_statements,v)
NOTICE:  (pg_proc,v)
CALL
postgres=#
```

FOREACH 循环一个数组

```
postgres=# CREATE OR REPLACE PROCEDURE p_foreach()  AS
$$
DECLARE
    v_random_arr float8[]:=ARRAY[random(),random()];
    v_random float8;
BEGIN
    FOREACH v_random IN ARRAY v_random_arr LOOP
        RAISE NOTICE '%',v_random ;
    END LOOP;
END;
$$
LANGUAGE plpgsql;
CREATE PROCEDURE
postgres=# CALL p_foreach();
NOTICE:  0.744417542591691
NOTICE:  0.804096563253552
CALL
postgres=#

postgres=# CREATE OR REPLACE PROCEDURE p_foreach_slice()  AS
$$
DECLARE
    v_random_arr float8[]
[]:=ARRAY[ARRAY[random(),random()],ARRAY[random(),random()]];
    v_random float8;
BEGIN
    FOREACH v_random SLICE 0 IN ARRAY v_random_arr LOOP
        RAISE NOTICE '%',v_random ;
    END LOOP;
END;
$$
LANGUAGE plpgsql;
CREATE PROCEDURE
postgres=# CALL p_foreach_slice();
```

```
NOTICE:  0.0220407997258008
NOTICE:  0.898449067492038
NOTICE:  0.190678883343935
NOTICE:  0.103653562255204
CALL
postgres=#
```

循环会通过计算 expression 得到的数组的个体元素进行迭代。

```
postgres=# CREATE OR REPLACE PROCEDURE p_foreach_slice_1()  AS
$$
DECLARE
    v_random_arr float8[]
[]:=ARRAY[ARRAY[random(),random()],ARRAY[random(),random()]];
    v_random float8[];
BEGIN
    FOREACH v_random SLICE 1 IN ARRAY v_random_arr LOOP
        RAISE NOTICE '%',v_random ;
    END LOOP;
END;
$$
LANGUAGE plpgsql;
CREATE PROCEDURE
postgres=# CALL p_foreach_slice_1();
NOTICE:  {0.248282201588154,0.757913041394204}
NOTICE:  {0.0194511725567281,0.43799454299733}
CALL
```

通过一个正 SLICE 值，FOREACH 通过数组的切片而不是单一元素迭代。

其它控制语句

动态执行

```
postgres=# CREATE OR REPLACE PROCEDURE p_execute() AS
$$
DECLARE
    v_sql TEXT;
    v_relname TEXT;
BEGIN
    v_sql := 'SELECT relname FROM pg_class limit 1';
    EXECUTE v_sql INTO v_relname;
```

```
    RAISE NOTICE 'relname = %',v_relname;
END;
$$
LANGUAGE plpgsql;
CREATE PROCEDURE
postgres=# CALL p_execute();
NOTICE:  relname = pg_stat_statements
CALL
postgres=#
```

也可以使用 immediate。

```
postgres=# CREATE OR REPLACE PROCEDURE p_execute() AS
$$
DECLARE
    v_sql TEXT;
    v_relname TEXT;
BEGIN
    v_sql := 'SELECT relname FROM pg_class limit 1';
    EXECUTE immediate v_sql INTO v_relname;
    RAISE NOTICE 'relname = %',v_relname;
END;
$$
LANGUAGE plpgsql;
CREATE PROCEDURE
postgres=# CALL p_execute();
NOTICE:  relname = s1
CALL
postgres=#
```

动态执行就是拼 SQL 语句，然后使用 EXECUTE 命令执行。

执行一个没有结果的命令

```
postgres=# CREATE OR REPLACE PROCEDURE p_perform()  AS
$$
BEGIN
    perform md5(random()::text);
END;
$$
LANGUAGE plpgsql;
CREATE PROCEDURE
```

```
postgres=# call p_perform();
CALL
postgres=#
```

获取执行结果

```
postgres=# CREATE OR REPLACE PROCEDURE p_found()  AS
$$
DECLARE
    v_relname TEXT;
BEGIN
    SELECT relname INTO v_relname FROM pg_class limit 1;
    IF FOUND THEN
        RAISE NOTICE '查询到记录，值为%',v_relname;
    ELSE
        RAISE NOTICE '查不到记录' ;
    END IF;
END;
$$
LANGUAGE plpgsql;
CREATE PROCEDURE
postgres=# CALL p_found();
NOTICE:  查询到记录，值为pg_stat_statements
CALL
```

获取影响行数

```
postgres=# CREATE OR REPLACE PROCEDURE p_row_count()  AS
$$
DECLARE
    v_row_count BIGINT;
BEGIN
    delete from t1;
    GET DIAGNOSTICS v_row_count = ROW_COUNT;
    RAISE NOTICE '查询到的记录数为 % ',v_row_count;
END;
$$
LANGUAGE plpgsql;
CREATE PROCEDURE
postgres=# call p_row_count();
NOTICE:  查询到的记录数为 3
CALL
```

```
postgres=#
```

GOTO

```
postgres=# create or replace procedure p_goto(v_maxnum integer) as
$$
declare
    maxnum integer;
begin
    maxnum := v_maxnum;
for i in 1..maxnum loop
    if i=3 then
        goto label;
    end if;
    raise notice 'i=%',i;
end loop;
<<label>>
    raise notice 'goto end';
end;
$$
language plpgsql;
CREATE PROCEDURE
postgres=# call p_goto(5);
NOTICE:  i=1
NOTICE:  i=2
NOTICE:  goto end
CALL
postgres=#
```

go 用于跳转到某个标签下。

俘获错误

错误俘获处理

```
postgres=# CREATE OR REPLACE PROCEDURE p_exception(a_id integer,a_nc
text) AS
$$
BEGIN
    INSERT INTO t_exception VALUES(a_id,a_nc);
    RETURN ;
    EXCEPTION WHEN OTHERS THEN
```

```
    RAISE NOTICE '执行出错';
END;
$$
LANGUAGE plpgsql;
CREATE PROCEDURE
postgres=#

postgres=# CALL p_exception(1,'Tdsql_pg');
CALL
postgres=# CALL p_exception(1,'Tdsql_pg');
NOTICE:  执行出错
CALL
```

获取错误相关信息

```
postgres=# CREATE OR REPLACE PROCEDURE p_exception_error(a_id
integer,a_nc text) AS
$$
DECLARE
    v_sqlstate text;
    v_context text;
    v_message_text text;
BEGIN
    INSERT INTO t_exception VALUES(a_id,a_nc);
    RETURN ;
    EXCEPTION WHEN OTHERS THEN
    GET STACKED DIAGNOSTICS v_sqlstate = RETURNED_SQLSTATE,
                            v_message_text = MESSAGE_TEXT,
                            v_context = PG_EXCEPTION_CONTEXT;
    RAISE NOTICE '错误代码 : %',v_sqlstate;
    RAISE NOTICE '出错信息 : %',v_message_text;
    RAISE NOTICE '发生异常语句 : %',v_context;
    raise notice '错误代码 : % \n出错信息 : % 发生异常语句 : %',v_sqlstate
,v_message_text,v_context;
END;
$$
LANGUAGE plpgsql;
CREATE PROCEDURE
postgres=# CALL p_exception_error(2,'Tdsql_pg');
CALL
postgres=# CALL p_exception_error(2,'Tdsql_pg');
NOTICE:  错误代码 : 23505
```

```
NOTICE:  出错信息 : node:dn001, backend_pid:16204,
nodename:dn001,backend_pid:16204,message:duplicate key value 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 p_exception_error(integer,text) line 7 at SQL
statement
NOTICE:  错误代码 : 23505 \n出错信息 : node:dn001, backend_pid:16204,
nodename:dn001,backend_pid:16204,message: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 p_exception_error(integer,text) line 7 at SQL
statement
CALL
postgres=#
```

连接 TDSQL PG
使用 psql 连接 TDSQL PG

最近更新时间：2024-11-01 15:20:42

本节主要介绍如何使用 psql 连接 TDSQL PG 。

前置条件

> **说明：**
> PostgreSQL 的 psql 客户端不支持分布键显示等功能。

通过 psql 连接 TDSQL PG 前，需要确保：

- 本地已正确安装 TDSQL PG 的 psql 客户端。
- 环境变量配置正确。

连接方式

psql 可以通过四种方式连接实例，分别是：使用参数连接、使用 conninfo 字符串或者 URI 连接、配置环境变量后快捷连接。

使用参数连接

使用 psql 工具通过参数连接 TDSQL 数据库，通常需要指定主机名 `-h` ，端口号 `-p` ，连接数据库的用户名 `-U` 和连接的数据库 `-d` ，如果不指定参数值，则取其默认值。
通过 psql 使用参数连接数据库的命令示例如下：

```
psql -h <数据库连接地址> -p <端口号> -U <用户名> -d <数据库名>
```

参数说明：

- `-h` 数据库IP，通常是 CN 的 IP。
- `-p` 数据库端口
- `-U` 数据库用户
- `-d` 需要访问的数据库名称

示例

指定数据库为 tdsql，用户为 tbase ，以 tbase 身份登录至 tdsql 数据库。

```
psql -h 10.211.55.39 -p 11345 -U tbase -d tdsql
```

使用 conninfo 字符串或者 URI 连接

使用 psql 通过 conninfo 字符串或者 URI 连接数据库，同样需要在 conninfo 和 URI 指定主机名称等参数，如果不指定，则取默认的参数值。

```
conninfo: psql "host=<数据库连接地址> port=<端口号> dbname=<数据库名> user=用户名"

URI: postgresql://[user[:password]@][netloc][:port][,...][/dbname][?param1=value1&...]
```

示例

示例1: 通过 conninfo 连接数据库

```
psql "host=10.211.55.39 port=11345 dbname=tdsql user=tbase"
```

示例2: 通过 URI 连接数据库

```
psql postgresql://tbase@10.211.55.39:11345/tdsql
```

配置环境变量后快捷连接

TDSQL PG 支持通过在环境变量中指定主机名称，端口地址，数据库以及数据库用户后通过命令 psql 即可快速连接。相关环境变量分别为 `PGHOST` ， `PGPORT` ， `PGDATABASE` ， `PGUSER` 。

示例

```
export PGUSER=tbase
export PGHOST=10.211.55.39
export PGDATABASE=tdsql
export PGPORT=11345

psql
```

psql 常用命令

命令	功能	命令	功能
\password	设置密码	\dt[+]	显示表格列表或表格的详细信息

\q \quit	退出 psql 程序	\dv[+]	显示视图列表或视图的详细信息
\conninfo	输出有关当前数据库连接的信息	\df[+]	显示函数/过程列表或函数/过程的详细信息
\timing	显式 SQL 语句打开和关闭的时间	\ds[+]	显示序列列表或序列的详细信息
\i	从指定文件中读取输入，并执行	\do[+]	显示操作符列表或操作符的详细信息
\o	指定结果输出文件	\du[+]	显示用户列表或用户的详细信息
\l	列出可用的数据库	\dn[+]	显示模式列表或模式的详细信息
\copy	执行一次前段拷贝	\di[+]	显示索引列表或索引的详细信息
\x	切换扩展表格格式化模式	\dx[+]	显示插件列表或插件的详细信息

更多信息

如果在通过 psql 客户端连接 TDSQL PG 数据库的过程中遇到本文档未涉及的问题，您可以查阅 PostgreSQL 客户端官方的使用文档 。

使用 Java 框架连接 TDSQL PG
使用 JDBC 连接 TDSQL PG

最近更新时间：2025-04-09 16:17:22

本节主要介绍如何使用 TDSQL PG JDBC 连接 TDSQL PG 数据库，实现数据增删查改等基本操作。

简介

The Java Database Connectivity (JDBC) API 是 Java 编程语言与各种数据库、SQL 数据库和其他表格数据源（如电子表格或平面文件）之间独立于数据库的连接的行业标准。JDBC API 提供了基于 SQL 数据库访问的调用级 API。TDSQL PG Connector 驱动程序是腾讯云独立更新和维护的、商业化的、纯 Java 编写的 JDBC 驱动程序；使用 PostgreSQL 原生网络协议进行通信，并特别针对 TDSQL PG 数据库的使用语法和功能特性进行了开发和适配。

连接数据库

准备 TDSQL PG JDBC 驱动包

TDSQL PG JDBC 驱动包请单击 驱动下载 获取，可根据开发环境选择合适的驱动包。

- `tdsql-pg-connector-java8-x.x.x.jar` 对应 JDK8 及以上
- `tdsql-pg-connector-java7-x.x.x.jar` 对应 JDK7
- `tdsql-pg-connector-java6-x.x.x.jar` 对应 JDK6

配置包名和连接串

TDSQL PG JDBC 驱动名为 `com.tencentcloud.tdsql.pg.jdbc.Driver` 。用于连接 TDSQL PG 服务器的 JDBC URL 的一般格式如下，方括号（[]）中的选项为可选内容：
`jdbc:tdsql-pg://host[:port]/][database][?property1=value1[&property2=value2]...]`，
其中：

- jdbc:tdsql-pg: (固定)：TDSQL PG Connector 驱动程序固定使用的 URL 前缀。
- host (可选)：服务器地址。可以是 DNS 或 IP 地址，也可以是本地计算机的 localhost 或127.0.0.1。默认值为 localhost。
- port (可选)：服务器上监听的端口号。默认值为5432。
- database (可选)：数据库名。默认为连接中使用的用户名。
- propertyX (可选)：是一个或多个连接参数。

> **说明：**
> 详细的配置说明以及连接参数等可参考 TDSQL PG JDBC 驱动说明 。

本文所有示例需在启用 Oracle 兼容内核的情况下执行，打开内核 Oracle 兼容模式请参考 Oracle 兼容特性概述。

连接示例

示例程序： `testConnect.java`

```
import java.sql.Connection;
import java.sql.DriverManager;

public class testConnect {
    public static void main(String args[]) {
        try {
            Class.forName("com.tencentcloud.tdsql.pg.jdbc.Driver");
            // host、port、database、user、password 需要替换为数据库管理员提供的数据库相关信息
            Connection conn = DriverManager.getConnection("jdbc:tdsql-pg://host:port/database?oracle_compile=true","user", "password");
            if (conn != null) {
                System.out.println("连接数据库成功！ ");
                conn.close();
            }
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

> ⚠ **注意：**
>
> `host` 、 `port` 、 `database` 、 `user` 、 `password` **需要改为实际使用的参数。**

运行结果

```
连接数据库成功！
```

开发示例

基础增删查改示例

示例程序： `testInsert.java`

```
import java.sql.Connection;
```

```
import java.sql.DriverManager;
import java.sql.ResultSet;
import java.sql.Statement;

public class testInsert {
    public static void main(String args[]) {
        try {
            Class.forName("com.tencentcloud.tdsql.pg.jdbc.Driver");
            // host、port、database、user、password 需要替换为数据库管理员提供
的数据库相关信息
            Connection conn = DriverManager.getConnection("jdbc:tdsql-
pg://host:port/database?oracle_compile=true","user", "password");
            Statement stmt = conn.createStatement();

            // 创建表
            System.out.println("------创建表------");
            stmt.execute("drop table if exists studentTab");
            stmt.execute("create table studentTab(id int, name
varchar)");
            System.out.println("创建studentTab表");

            // 插入数据
            System.out.println("------插入数据------");
            stmt.execute("insert into studentTab values (1,'Alex'),
(2,'Bob'),(3,'John')");
            System.out.println("插入3条数据");

            // 查询数据
            System.out.println("------查询数据------");
            ResultSet rs1 = stmt.executeQuery("select * from studentTab
order by id");
            while (rs1.next()) {
                System.out.println(rs1.getInt(1) + " " +
rs1.getString(2));
            }

            // 更改数据
            System.out.println("------更改数据------");
            stmt.execute("update studentTab set name = 'Alice' where id
= 1");
            System.out.println("更改id为1的数据");

            // 查询数据
```

```
            System.out.println("------查询数据------");
            ResultSet rs2 = stmt.executeQuery("select * from studentTab
order by id");
            while (rs2.next()) {
                System.out.println(rs2.getInt(1) + " " +
rs2.getString(2));
            }

            // 删除数据
            System.out.println("------删除数据------");
            stmt.execute("delete from studentTab where id = 2");
            System.out.println("删除id为2的数据");

            // 查询数据
            System.out.println("------查询数据------");
            ResultSet rs3 = stmt.executeQuery("select * from studentTab
order by id");
            while (rs3.next()) {
                System.out.println(rs3.getInt(1) + " " +
rs3.getString(2));
            }

            // 关闭对象
            rs1.close();
            rs2.close();
            rs3.close();
            stmt.close();
            conn.close();
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

运行结果

```
------创建表------
创建studentTab表
------插入数据------
插入3条数据
------查询数据------
1 Alex
2 Bob
```

```
3 John
------更改数据------
更改id为1的数据
------查询数据------
1 Alice
2 Bob
3 John
------删除数据------
删除id为2的数据
------查询数据------
1 Alice
3 John
```

prepare 语法示例

示例程序： `testPrepare.java`

```
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.Statement;

public class testPrepare {
    public static void main(String args[]) {
        try {
            Class.forName("com.tencentcloud.tdsql.pg.jdbc.Driver");
            // host、port、database、user、password 需要替换为数据库管理员提供
的数据库相关信息
            Connection conn = DriverManager.getConnection("jdbc:tdsql-
pg://host:port/database?oracle_compile=true","user", "password");
            Statement stmt = conn.createStatement();

            // 复用增删查改示例中的studentTab表
            // prepare插入数据
            System.out.println("------prepare插入数据------");
            PreparedStatement pstmt = conn.prepareStatement("insert into
studentTab values (?, ?)");
            pstmt.setInt(1, 4);
            pstmt.setString(2, "Luna");
            pstmt.execute();
            System.out.println("插入1条数据");
```

```
            // 查询数据
            System.out.println("------查询数据------");
            ResultSet rs = stmt.executeQuery("select * from studentTab
order by id");
            while (rs.next()) {
                System.out.println(rs.getInt(1) + " " +
rs.getString(2));
            }

            // 关闭对象
            rs.close();
            pstmt.close();
            stmt.close();
            conn.close();
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

> ⚠ **注意：**
> `host`、`port`、`database`、`user`、`password` 需要改为实际使用的参数。

运行结果

```
------prepare插入数据------
插入1条数据
------查询数据------
1 Alice
3 John
4 Luna
```

调用存储过程和函数示例

示例程序： `testCall.java`

```
import java.sql.CallableStatement;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.Statement;
import java.sql.Types;
```

```
public class testCall {
    public static void main(String args[]) {
        try {
            Class.forName("com.tencentcloud.tdsql.pg.jdbc.Driver");
            // host、port、database、user、password 需要替换为数据库管理员提供
的数据库相关信息
            // 需要添加escapeSyntaxCallMode=callIfNoReturn参数，以确保驱动程
序正确调用存储过程或者函数
            Connection conn = DriverManager.getConnection(
                    "jdbc:tdsql-pg://host:port/database?
oracle_compile=true&escapeSyntaxCallMode=callIfNoReturn","user",
"password");
            Statement stmt = conn.createStatement();

            // 创建存储过程
            stmt.execute("create or replace procedure testProc(p1 in
int, p2 out int) is\n"
                    + "begin\n"
                    + "  p2 := p1;\n"
                    + "end;");

            // 调用存储过程
            System.out.println("------调用存储过程------");
            CallableStatement proc = conn.prepareCall("{call testProc(?,
?)}");
            proc.setInt(1, 100);
            proc.registerOutParameter(2, Types.INTEGER);
            proc.execute();
            System.out.println("调用结果: " + proc.getInt(2));
            proc.close();

            // 创建函数
            stmt.execute("create or replace function testFunc(p1
varchar) return varchar\n"
                    + "is\n"
                    + "begin\n"
                    + "  return p1;\n"
                    + "end;");

            // 调用函数
            System.out.println("------调用函数------");
```

```
            CallableStatement func = conn.prepareCall("{? = call
testFunc(?)}");
            func.registerOutParameter(1, Types.VARCHAR);
            func.setString(2, "test");
            func.execute();
            System.out.println("调用结果: " + func.getString(1));
            func.close();

            // 关闭对象
            stmt.close();
            conn.close();
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

注意：

`host`、`port`、`database`、`user`、`password` 需要改为实际使用的参数。
需要添加 escapeSyntaxCallMode=callIfNoReturn 参数，以确保驱动程序正确调用存储过程或者函数。

运行结果

```
------调用存储过程------
调用结果: 100
------调用函数------
调用结果: test
```

使用 Hibemate 框架连接 TDSQL PG

最近更新时间：2025-08-12 14:33:12

本节主要介绍如何使用 Hibernate 框架连接 TDSQL PG 数据库，实现数据的增删查改等基本操作。

简介

Hibernate 是一个开放源代码的对象关系映射框架，它对 JDBC 进行了非常轻量级的对象封装，它将 POJO 与数据库表建立映射关系，是一个全自动的 ORM 框架，Hibernate 可以自动生成 SQL 语句，自动执行，使得 Java 程序员可以随心所欲的使用对象编程思维来操纵数据库。

引入和配置 Hibernate 框架

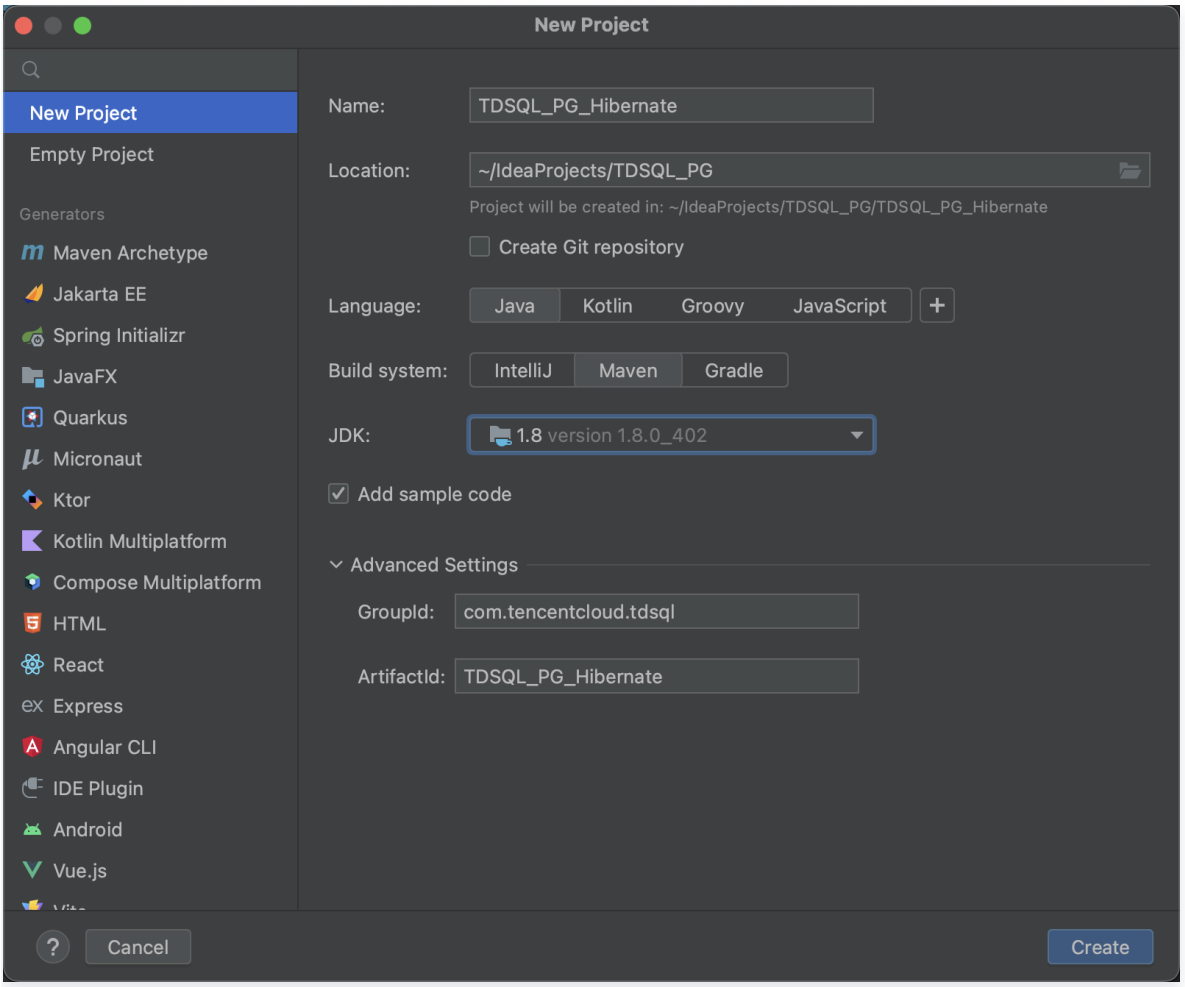
前置依赖

在配置之前需要保证你的机器上已安装对应版本的 JDK，本例是 JDK 8。
内核 Oracle 模式下建议使用腾讯自研驱动，下载地址请参考 驱动下载，联系数据库管理员获取数据库相关信息：包括数据库 IP、端口、实例名、用户、密码等。驱动名和连接 URL 如下：

- 驱动名 `com.tencentcloud.tdsql.pg.jdbc.Driver`
- 连接 URL（Oracle 模式下需打开 oracle_compile 参数）
  `jdbc:tdsql-pg://host:port/database?oracle_compile=true`

创建 Maven 项目

创建 Maven 项目，如下：



> **注意：**
> 本文档示例项目使用的 IDE 是 IntelliJ IDEA 2022.3 (Ultimate Edition)，也可根据喜好选择合适的 IDE。

配置 pom.xml 文件，导入项目依赖

配置文件 `pom.xml` 如下，配置 dependencies 部分。

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>

    <groupId>com.tencentcloud.tdsql</groupId>
    <artifactId>TDSQL_PG_Hibernate</artifactId>
    <version>1.0-SNAPSHOT</version>
```

```
    <properties>
        <maven.compiler.source>8</maven.compiler.source>
        <maven.compiler.target>8</maven.compiler.target>
    </properties>

    <dependencies>
        <!-- orm框架Hibernate -->
        <dependency>
            <groupId>org.hibernate</groupId>
            <artifactId>hibernate-core</artifactId>
            <version>5.6.15.Final</version>
        </dependency>
        <!-- 测试框架Junit5 -->
        <dependency>
            <groupId>org.junit.jupiter</groupId>
            <artifactId>junit-jupiter-engine</artifactId>
            <version>5.10.0</version>
            <scope>test</scope>
        </dependency>
        <!-- TDSQL-PG JDBC驱动 -->
        <dependency>
            <groupId>com.tencentcloud.tdsql</groupId>
            <artifactId>tdsql-pg-connector-java8</artifactId>
            <version>1.1.0</version>
            <scope>system</scope>
            <systemPath>${project.basedir}/lib/tdsql-pg-connector-java8-
1.1.0.jar</systemPath>
        </dependency>
    </dependencies>

</project>
```

通过 `<dependencies>` 定义项目所依赖的组件，主要包括：

1. Hibernate 依赖包，填写好后会从公共库自动下载。
2. JUnit5 测试框架依赖包，填写好后会从公共库自动下载。
3. 下载 TDSQL-PG JDBC 驱动包，并在 `TDSQL_PG_Hibernate` 目录下新建 `lib` 目录，将驱动包拷贝到 `lib` 目录下。

> **注意：**
>
> - 驱动包下载地址请参考 驱动下载 。

- 本文所有示例需在启用 Oracle 兼容内核的情况下执行。打开内核 Oracle 兼容模式请参考 Oracle 兼容特性概述 。

配置 Hibernate

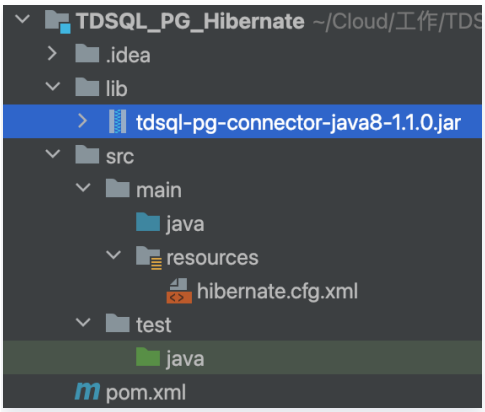
在 `TDSQL_PG_Hibernate/src/main/resources` 目录下新建 `hibernate.cfg.xml` 文件，用于配置 Hibernate，`hibernate.cfg.xml` 配置如下：

```
<?xml version='1.0' encoding='utf-8'?>
<!DOCTYPE hibernate-configuration PUBLIC
        "-//Hibernate/Hibernate Configuration DTD//EN"
        "http://www.hibernate.org/dtd/hibernate-configuration-3.0.dtd">
<hibernate-configuration>
    <session-factory>
        <!-- 配置JDBC连接信息：host、port、database、user、password 需要替换为数据库管理员提供的数据库相关信息。 -->
        <property name="hibernate.connection.driver_class">com.tencentcloud.tdsql.pg.jdbc.Driver</property>
        <property name="hibernate.connection.url">jdbc:tdsql-pg://host:port/database?oracle_compile=true</property>
        <property name="hibernate.connection.username">user</property>
        <property name="hibernate.connection.password">password</property>

        <!-- 配置数据库方言 -->
        <property name="hibernate.dialect">org.hibernate.dialect.PostgreSQL10Dialect</property>
    </session-factory>
</hibernate-configuration>
```

其中，需要配置 JDBC 连接信息和数据库方言：

- `hibernate.connection.driver_class`：指定驱动包名为 `com.tencentcloud.tdsql.pg.jdbc.Driver` 。
- `hibernate.connection.url`：指定连接 URL，包括数据库地址、端口、数据库名、连接属性等信息。
- `hibernate.connection.username`：指定用户名。
- `hibernate.connection.password`：指定相应用户的密码。
- `hibernate.dialect`：指定数据库方言为 PostgreSQL 的数据库方言，使 Hibernate 生成适当的 SQL。

至此，已完成 Hibernate 项目的基本配置，完整的项目目录结构如下图所示：



开发示例

添加数据表对应的实体类

首先，开发使用的数据表为 `STUDENT` 表（`CREATE TABLE STUDENT(ID INT PRIMARY KEY, NAME VARCHAR(50))`），对应地添加实体类 `Student`，先在 `TDSQL_PG_Hibernate/src/main/java/` 下创建 Package: `com.tencentcloud.tdsql.pojo` 再在 `TDSQL_PG_Hibernate/src/main/java/com/tencentcloud/tdsql/pojo` 目录下新建 `Student.java` 文件：

```
package com.tencentcloud.tdsql.pojo;

public class Student {
    private int id;
    private String name;

    public Student() {
    }

    public Student(int id, String name) {
        this.id = id;
        this.name = name;
    }

    public int getId() {
        return id;
    }

    public void setId(int id) {
        this.id = id;
    }
```

```
    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }

    @Override
    public String toString() {
        return "Student [id=" + id + ", name=" + name + "]";
    }
}
```

配置实体类与数据表之间的映射

其次，需要将 Java 对象 `Student` 映射到数据库中的表 `STUDENT`，先在 `TDSQL_PG_Hibernate/src/main/resources/` 下创建文件夹：`com/tencentcloud/tdsql/pojo` 在 `TDSQL_PG_Hibernate/src/main/resources/com/tencentcloud/tdsql/pojo` 下新建 `Student.hbm.xml` 映射文件，如下：

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE hibernate-mapping PUBLIC
        "-//Hibernate/Hibernate Mapping DTD 3.0//EN"
        "http://www.hibernate.org/dtd/hibernate-mapping-3.0.dtd">
<hibernate-mapping package="com.tencentcloud.tdsql.pojo">
    <!-- 配置表和属性 -->
    <class name="Student" table="STUDENT">
        <id name="id" column="ID" type="java.lang.Integer">
            <generator class="org.hibernate.id.Assigned" />
        </id>
        <property name="name" column="NAME" length="50"
type="java.lang.String"/>
    </class>
</hibernate-mapping>
```

同时，需要将该映射文件的路径写入到 Hibernate 配置文件 `hibernate.cfg.xml` 中，如下：

```
<?xml version='1.0' encoding='utf-8'?>
<!DOCTYPE hibernate-configuration PUBLIC
        "-//Hibernate/Hibernate Configuration DTD//EN"
        "http://www.hibernate.org/dtd/hibernate-configuration-3.0.dtd">
```

```
<hibernate-configuration>
    <session-factory>
        <!-- 配置数据库信息：host、port、database、user、password 需要替换为数
据库管理员提供的数据库相关信息。 -->
        <property
name="hibernate.connection.driver_class">com.tencentcloud.tdsql.pg.jdbc.
Driver</property>
        <property name="hibernate.connection.url">jdbc:tdsql-
pg://host:port/database?oracle_compile=true</property>
        <property name="hibernate.connection.username">user</property>
        <property
name="hibernate.connection.password">password</property>

        <!-- 配置数据库方言 -->
        <property
name="hibernate.dialect">org.hibernate.dialect.PostgreSQL10Dialect</prop
erty>

        <!-- 新增映射文件路径 -->
        <mapping
resource="com/tencentcloud/tdsql/pojo/Student.hbm.xml"/>
    </session-factory>
</hibernate-configuration>
```

封装数据库表的增删查改操作

然后，在 `TDSQL_PG_Hibernate/src/main/java/` 下创建 Package `com.tencentcloud.tdsql.dao`，在 `TDSQL_PG_Hibernate/src/main/java/com/tencentcloud/tdsql/dao` 下新建 `StudentDao` 类，定义与数据库交互相关的方法，以实现数据的增删查改，如下：

```
package com.tencentcloud.tdsql.dao;

import com.tencentcloud.tdsql.pojo.Student;
import java.util.List;
import org.hibernate.Session;
import org.hibernate.SessionFactory;
import org.hibernate.cfg.Configuration;
import org.hibernate.query.Query;

public class StudentDao {
    private Session session = null;
    {
        // 读取 hibernate.cfg.xml
```

```
        Configuration cfg = new Configuration().configure();
        SessionFactory sessionFactory = cfg.buildSessionFactory();
        // 打开一个 session
        session = sessionFactory.openSession();
    }

    // 插入
    public int insertStudent(Student student) {
        session.beginTransaction();
        session.save(student);
        session.getTransaction().commit();
        return 1;
    }

    // 查询
    public List<Student> selectAllStudent() {
        String sql = "from Student order by id";
        Query query = session.createQuery(sql);
        List<Student> students = query.list();
        return students;
    }

    // 更新
    public int updateStudentById(Student student) {
        session.beginTransaction();
        Student student1 = session.get(Student.class, student.getId());
        session.merge(student);
        session.getTransaction().commit();
        return 1;
    }

    // 删除
    public int deleteStudentById(int id) {
        session.beginTransaction();
        Student student = session.get(Student.class, id);
        session.delete(student);
        session.getTransaction().commit();
        return 1;
    }
}
```

更多 Hibernate 操作，请查询 Hibernate 官网。

测试

最后，在 `TDSQL_PG_Hibernate/src/test/java` 下添加测试类 `TestStudent` ，展示了基本的增删查改功能，如下：

```
import com.tencentcloud.tdsql.dao.StudentDao;
import com.tencentcloud.tdsql.pojo.Student;

import java.util.List;
import org.junit.jupiter.api.Test;

public class TestStudent {
    @Test
    public void test() {
        StudentDao studentDao = new StudentDao();
        System.out.println("------插入数据------");
        studentDao.insertStudent(new Student(1, "Alex"));
        studentDao.insertStudent(new Student(2, "Bob"));
        studentDao.insertStudent(new Student(3, "John"));
        System.out.println("插入3条数据");

        System.out.println("------查询数据------");
        List<Student> students1 = studentDao.selectAllStudent();
        for (Student student: students1) {
            System.out.println(student);
        }

        System.out.println("------更改数据------");
        studentDao.updateStudentById(new Student(1, "Alice"));
        System.out.println("更改id为1的数据");

        System.out.println("------查询数据------");
        List<Student> students2 = studentDao.selectAllStudent();
        for (Student student: students2) {
            System.out.println(student);
        }

        System.out.println("------删除数据------");
        studentDao.deleteStudentById(2);
        System.out.println("删除id为2的数据");

        System.out.println("------查询数据------");
        List<Student> students3 = studentDao.selectAllStudent();
```

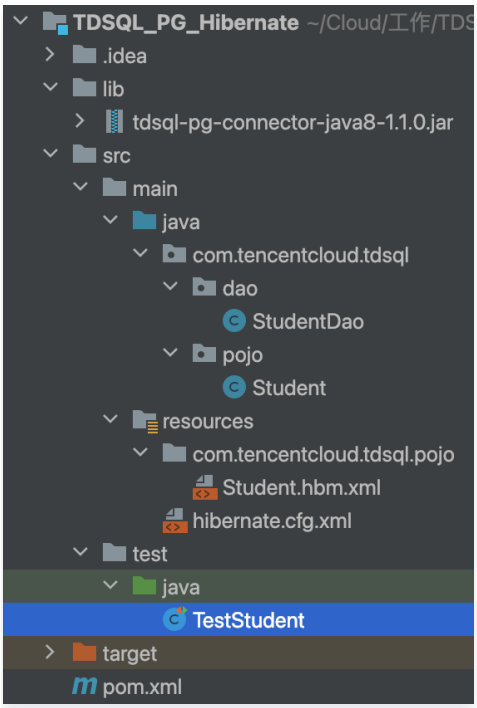
```
        for (Student student: students3) {
            System.out.println(student);
        }
    }
}
```

运行结果如下：

```
------插入数据------
插入3条数据
------查询数据------
Student [id=1, name=Alex]
Student [id=2, name=Bob]
Student [id=3, name=John]
------更改数据------
更改id为1的数据
------查询数据------
Student [id=1, name=Alice]
Student [id=2, name=Bob]
Student [id=3, name=John]
------删除数据------
删除id为2的数据
------查询数据------
Student [id=1, name=Alice]
Student [id=3, name=John]
```

附：项目结构

至此，完成了 Hibernate 项目的配置和开发示例，项目的整体结构如下图所示：


TDSQL_PG_Hibernate ~/Cloud/工作/TDS
.idea
lib
tdsql-pg-connector-java8-1.1.0.jar
src
main
java
com.tencentcloud.tdsql
dao
StudentDao
pojo
Student
resources
com.tencentcloud.tdsql.pojo
Student.hbm.xml
hibernate.cfg.xml
test
java
TestStudent
target
pom.xml

使用 MyBatis 框架连接 TDSQL PG

最近更新时间：2025-04-28 16:59:42

本节主要介绍如何使用 MyBatis 框架连接 TDSQL PG 数据库，实现数据的增删查改等基本操作。

简介

MyBatis 是一款优秀的持久层框架，它支持自定义 SQL、存储过程以及高级映射。MyBatis 免除了几乎所有的 JDBC 代码以及设置参数和获取结果集的工作。MyBatis 可以通过简单的 XML 或注解来配置和映射原始类型，将接口和 Java POJO（Plain Old Java Objects，普通的 Java 对象）映射成数据库中的记录。

引入和配置 MyBatis 框架

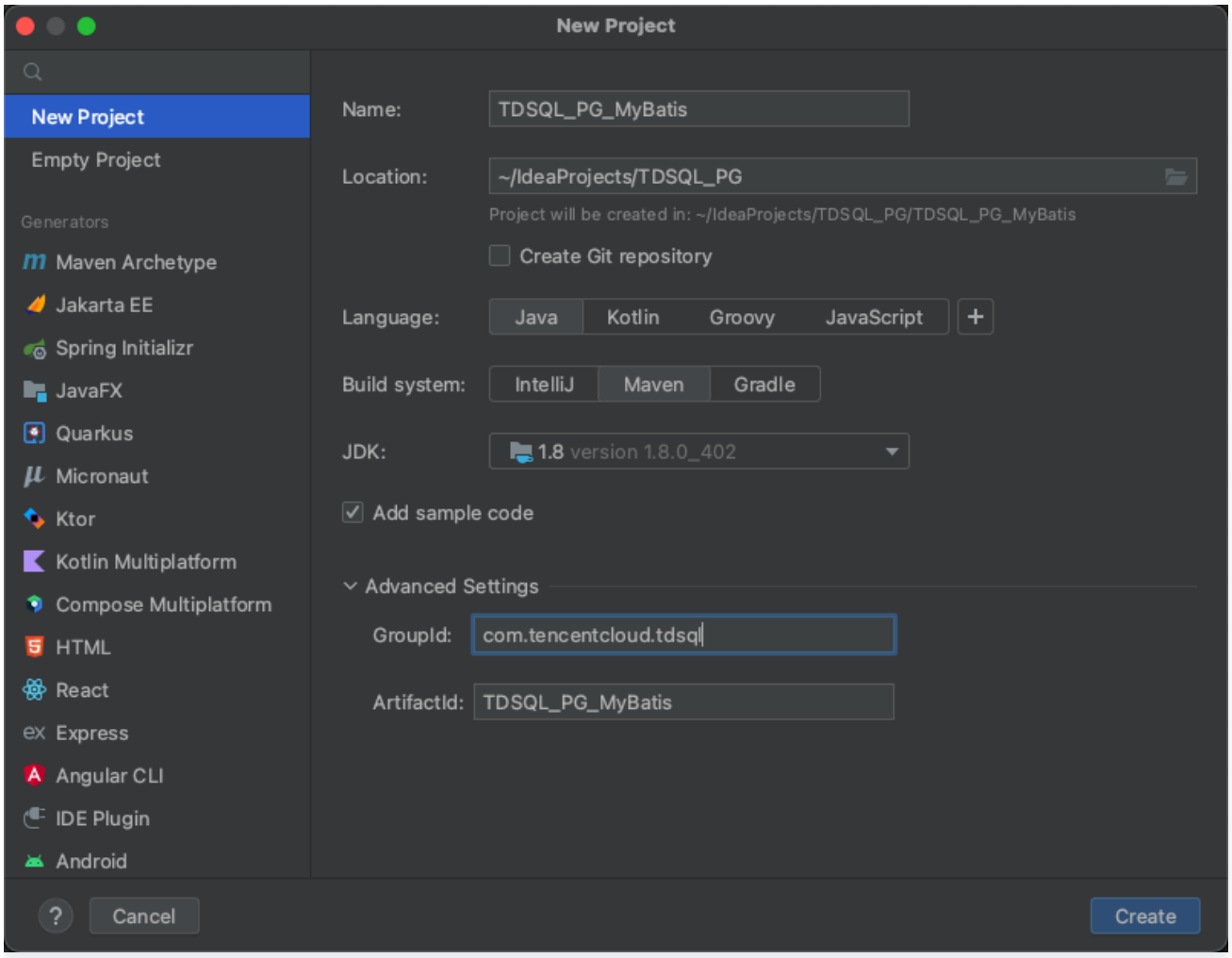
前置依赖

在配置之前需要保证你的机器上已安装对应版本的 JDK，本例是 JDK 8。内核 Oracle 模式下建议使用腾讯自研驱动，请单击 驱动下载 。
联系数据库管理员获取数据库相关信息：包括数据库 IP、端口、实例名、用户、密码等。驱动名和连接 URL 如下：

- 驱动名 `com.tencentcloud.tdsql.pg.jdbc.Driver`
- 连接 URL（Oracle 模式下需打开 oracle_compile 参数）
  `jdbc:tdsql-pg://host:port/database?oracle_compile=true`

创建 Maven 项目

创建 Maven 项目，如下：



> **注意：**
> 本文档示例项目使用的 IDE 是 IntelliJ IDEA 2022.3 (Ultimate Edition)，也可根据喜好选择合适的 IDE。

配置 pom.xml 文件，导入项目依赖

配置文件 `pom.xml` 如下：

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>

    <groupId>com.tencentcloud.tdsql</groupId>
    <artifactId>TDSQL_PG_MyBatis</artifactId>
    <version>1.0-SNAPSHOT</version>
```

```
    <properties>
        <maven.compiler.source>8</maven.compiler.source>
        <maven.compiler.target>8</maven.compiler.target>
    </properties>

    <dependencies>
        <!-- MyBatis -->
        <dependency>
            <groupId>org.mybatis</groupId>
            <artifactId>mybatis</artifactId>
            <version>3.5.11</version>
        </dependency>
        <!-- 测试框架Junit5 -->
        <dependency>
            <groupId>org.junit.jupiter</groupId>
            <artifactId>junit-jupiter-engine</artifactId>
            <version>5.10.0</version>
            <scope>test</scope>
        </dependency>
        <!-- TDSQL-PG JDBC驱动 -->
        <dependency>
            <groupId>com.tencentcloud.tdsql</groupId>
            <artifactId>tdsql-pg-connector-java8</artifactId>
            <version>1.1.0</version>
            <scope>system</scope>
            <systemPath>${project.basedir}/lib/tdsql-pg-connector-java8-
1.1.0.jar</systemPath>
        </dependency>
    </dependencies>

</project>
```

通过 `<dependencies>` 定义项目所依赖的组件，主要包括：

1. MyBatis 依赖包，填写好后会从公共库自动下载。
2. Junit5 测试框架依赖包，填写好后会从公共库自动下载。
3. 下载 TDSQL PG JDBC 驱动包，并在 `TDSQL_PG_MyBatis` 目录下新建 `lib` 目录，将驱动包拷贝到 `lib` 目录下。

> **注意：**
>
> - 驱动包下载请单击 **驱动下载** 。

- 本文所有示例需在启用 Oracle 兼容内核的情况下执行。打开内核 Oracle 兼容模式请参考 Oracle 兼容特性概述 。

配置 MyBatis

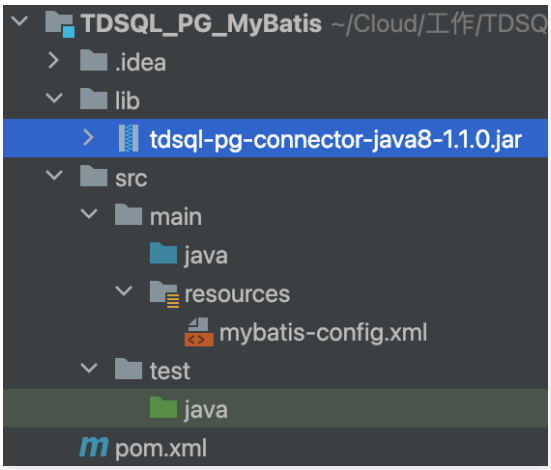
在 `TDSQL_PG_MyBatis/src/main/resources` 目录下新建 `mybatis-config.xml` 文件，用于配置 MyBatis，`mybatis-config.xml` 配置如下：

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE configuration
        PUBLIC "-//mybatis.org//DTD Config 3.0//EN"
        "https://mybatis.org/dtd/mybatis-3-config.dtd">
<configuration>
    <!-- 配置MyBatis运行环境 -->
    <environments default="development">
        <environment id="development">
            <transactionManager type="JDBC"/>
            <dataSource type="POOLED">
                <!-- 配置JDBC连接信息：host、port、database、user、password
需要替换为数据库管理员提供的数据库相关信息。 -->
                <property name="driver"
value="com.tencentcloud.tdsql.pg.jdbc.Driver"/>
                <property name="url" value="jdbc:tdsql-
pg://host:port/database?oracle_compile=true"/>
                <property name="username" value="user"/>
                <property name="password" value="password"/>
            </dataSource>
        </environment>
    </environments>
</configuration>
```

其中，需要配置 JDBC 连接信息，仅修改 `value` 的值：

- `driver`：指定驱动包名为 `com.tencentcloud.tdsql.pg.jdbc.Driver` 。
- `url`：指定连接 URL，包括数据库地址、端口、数据库名、连接属性等信息。
- `username`：指定用户名。
- `password`：指定相应用户的密码。

至此，已完成 MyBatis 项目的基本配置，完整的项目目录结构如下图所示：



开发示例

添加数据表对应的实体类

首先，开发使用的数据表为 `STUDENT` 表（`CREATE TABLE STUDENT(ID INT PRIMARY KEY, NAME VARCHAR(50))`），对应地，添加实体类 `Student`，先在 `TDSQL_PG_MyBatis/src/main/java/` 下创建 Package: `com.tencentcloud.tdsql.pojo` 再在 `TDSQL_PG_MyBatis/src/main/java/com/tencentcloud/tdsql/pojo` 下新建 `Student.java` 文件：

```
package com.tencentcloud.tdsql.pojo;

public class Student {
    private int id;
    private String name;

    public Student() {
    }

    public Student(int id, String name) {
        this.id = id;
        this.name = name;
    }

    public int getId() {
        return id;
    }

    public void setId(int id) {
        this.id = id;
    }
```

```
    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }

    @Override
    public String toString() {
        return "Student [id=" + id + ", name=" + name + "]";
    }
}
```

定义数据库操作接口

其次，在 `TDSQL_PG_MyBatis/src/main/java/` 下创建 Package: `com.tencentcloud.tdsql.mapper` 再在 `TDSQL_PG_MyBatis/src/main/java/com/tencentcloud/tdsql/mapper` 下新建 `StudentMapper.java` 文件，定义数据库的增删查改接口，如下：

```
package com.tencentcloud.tdsql.mapper;

import com.tencentcloud.tdsql.pojo.Student;
import java.util.List;

public interface StudentMapper {
    // 插入
    Integer insertStudent(Student student);

    // 查询
    List<Student> selectAllStudent();

    // 更新
    Integer updateStudentById(Student student);

    // 删除
    Integer deleteStudentById(Integer id);
}
```

配置操作接口与对应 SQL 语句的映射关系

然后，需要将数据库的增删查改接口映射到对应的 SQL 语句，在 `TDSQL_PG_MyBatis/src/main/resources/` 下创建目录 `com/tencentcloud/tdsql/mapper`，再在 `TDSQL_PG_MyBatis/src/main/resources/com/tencentcloud/tdsql/mapper` 下新建 `StudentMapper.xml` 映射文件，如下：

```
<!DOCTYPE mapper
        PUBLIC "-//mybatis.org//DTD Mapper 3.0//EN"
        "https://mybatis.org/dtd/mybatis-3-mapper.dtd">
<!-- 定义命名空间，对应接口类StudentMapper -->
<mapper namespace="com.tencentcloud.tdsql.mapper.StudentMapper">
    <insert id="insertStudent"
parameterType="com.tencentcloud.tdsql.pojo.Student">
        insert into STUDENT (ID, NAME) values (#{id}, #{name})
    </insert>
    <select id="selectAllStudent"
resultType="com.tencentcloud.tdsql.pojo.Student">
        select * from STUDENT order by ID
    </select>
    <update id="updateStudentById"
parameterType="com.tencentcloud.tdsql.pojo.Student">
        update STUDENT set NAME = #{name} where ID = #{id}
    </update>
    <delete id="deleteStudentById" parameterType="java.lang.Integer">
        delete from STUDENT where ID = #{id};
    </delete>
</mapper>
```

其中，定义四个 SQL 语句，分别对应了 `StudentMapper` 接口类的增删查改接口：

- `insertStudent`：向 `STUDENT` 表中插入一条记录，包括 `ID` 和 `NAME` 两个字段，值分别为 `#{id}` 和 `#{name}`，传入的参数类型则为 `com.tencentcloud.tdsql.pojo.Student`。
- `selectAllStudent`：从 `STUDENT` 表中查询所有记录，返回的结果类型为 `com.tencentcloud.tdsql.pojo.Student`。
- `updateStudentById`：根据 `ID` 字段去更新相应记录的 `NAME` 字段，值分别为 `#{id}` 和 `#{name}`，传入的参数类型为 `com.tencentcloud.tdsql.pojo.Student`。
- `deleteStudentById`：根据 `ID` 字段删除相应的记录，值为 `#{id}`，传入的参数类型为 `java.lang.Integer`。

同时，需要将 `StudentMapper.xml` 映射文件的路径写入到 MyBatis 配置文件 `mybatis-config.xml` 中，如下：

```
<?xml version="1.0" encoding="UTF-8" ?>
```

```
<!DOCTYPE configuration
        PUBLIC "-//mybatis.org//DTD Config 3.0//EN"
        "https://mybatis.org/dtd/mybatis-3-config.dtd">
<configuration>
    <!-- 配置MyBatis运行环境 -->
    <environments default="development">
        <environment id="development">
            <transactionManager type="JDBC"/>
            <dataSource type="POOLED">
                <!-- 配置JDBC连接信息：host、port、database、user、password
需要替换为数据库管理员提供的数据库相关信息。 -->
                <property name="driver"
value="com.tencentcloud.tdsql.pg.jdbc.Driver"/>
                <property name="url" value="jdbc:tdsql-
pg://host:port/database?oracle_compile=true"/>
                <property name="username" value="user"/>
                <property name="password" value="password"/>
            </dataSource>
        </environment>
    </environments>
    <!-- 增加Mapper映射文件 -->
    <mappers>
        <mapper
resource="com/tencentcloud/tdsql/mapper/StudentMapper.xml"/>
    </mappers>
</configuration>
```

测试

最后，在`TDSQL_PG_MyBatis/src/test/java`下添加测试类`TestStudent`，展示了基本的增删查改功能，如下：

```
import com.tencentcloud.tdsql.mapper.StudentMapper;
import com.tencentcloud.tdsql.pojo.Student;

import java.io.IOException;
import java.io.InputStream;
import java.util.List;

import org.apache.ibatis.io.Resources;
import org.apache.ibatis.session.SqlSession;
import org.apache.ibatis.session.SqlSessionFactory;
import org.apache.ibatis.session.SqlSessionFactoryBuilder;
```

```
import org.junit.jupiter.api.Test;

public class TestStudent {
    @Test
    public void test() throws IOException {
        // 加载MyBatis配置
        InputStream inputStream =
Resources.getResourceAsStream("mybatis-config.xml");
        // 创建factory实例，用于管理SqlSession
        SqlSessionFactory factory = new
SqlSessionFactoryBuilder().build(inputStream);
        // 创建SqlSession实例，可以执行SQL语句、提交或回滚等操作
        SqlSession session = factory.openSession();
        // 获取StudentMapper接口的实例
        StudentMapper studentMapper =
session.getMapper(StudentMapper.class);
        System.out.println("------插入数据------");
        studentMapper.insertStudent(new Student(1, "Alex"));
        studentMapper.insertStudent(new Student(2, "Bob"));
        studentMapper.insertStudent(new Student(3, "John"));
        // commit
        session.commit();
        System.out.println("插入3条数据");

        System.out.println("------查询数据------");
        List<Student> students1 = studentMapper.selectAllStudent();
        for (Student student: students1) {
            System.out.println(student);
        }

        System.out.println("------更改数据------");
        studentMapper.updateStudentById(new Student(1, "Alice"));
        // commit
        session.commit();
        System.out.println("更改id为1的数据");

        System.out.println("------查询数据------");
        List<Student> students2 = studentMapper.selectAllStudent();
        for (Student student: students2) {
            System.out.println(student);
        }

        System.out.println("------删除数据------");
```

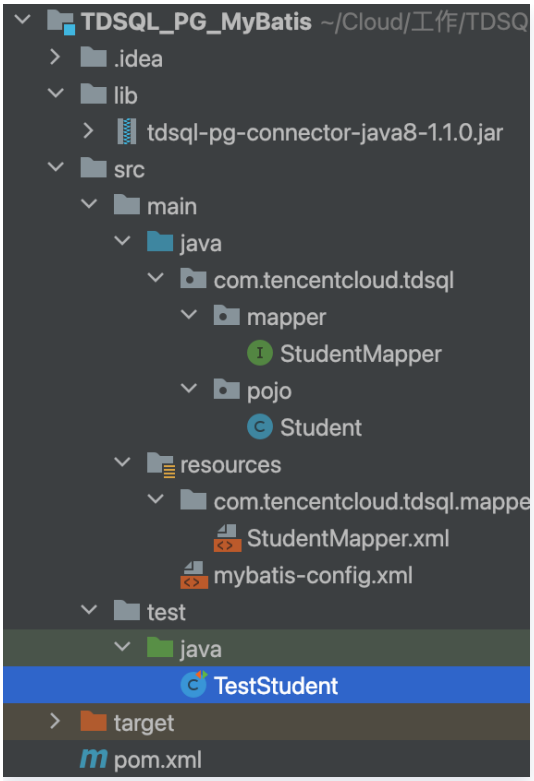
```
        studentMapper.deleteStudentById(2);
        // commit
        session.commit();
        System.out.println("删除id为2的数据");

        System.out.println("------查询数据------");
        List<Student> students3 = studentMapper.selectAllStudent();
        for (Student student: students3) {
            System.out.println(student);
        }

        session.close();
    }
}
```

运行结果如下：

```
------插入数据------
插入3条数据
------查询数据------
Student [id=1, name=Alex]
Student [id=2, name=Bob]
Student [id=3, name=John]
------更改数据------
更改id为1的数据
------查询数据------
Student [id=1, name=Alice]
Student [id=2, name=Bob]
Student [id=3, name=John]
------删除数据------
删除id为2的数据
------查询数据------
Student [id=1, name=Alice]
Student [id=3, name=John]
```

附：项目结构

至此，完成了 MyBatis 项目的配置和开发示例，项目的整体结构如下图所示：


TDSQL_PG_MyBatis ~/Cloud/工作/TDSQ
.idea
lib
tdsql-pg-connector-java8-1.1.0.jar
src
main
java
com.tencentcloud.tdsql
mapper
StudentMapper
pojo
Student
resources
com.tencentcloud.tdsql.mappe
StudentMapper.xml
mybatis-config.xml
test
java
TestStudent
target
pom.xml

用 Spring Boot 框架连接 TDSQL PG

最近更新时间：2025-04-25 16:26:42

本节主要介绍如何使用 Spring Boot 框架连接 TDSQL PG 数据库，实现数据的增删查改等基本操作。

简介

Spring Boot 是一个基于 Spring 的套件，可以简化 Spring 应用的创建及部署。它提供了丰富的 Spring 模块化支持，可以帮助开发者更轻松快捷地构建出企业级应用。Spring Boot 通过自动配置功能，降低了复杂性，同时支持基于 JVM 的多种开源框架，可以缩短开发时间，使开发更加简单和高效。

引入和配置 Spring Boot 框架

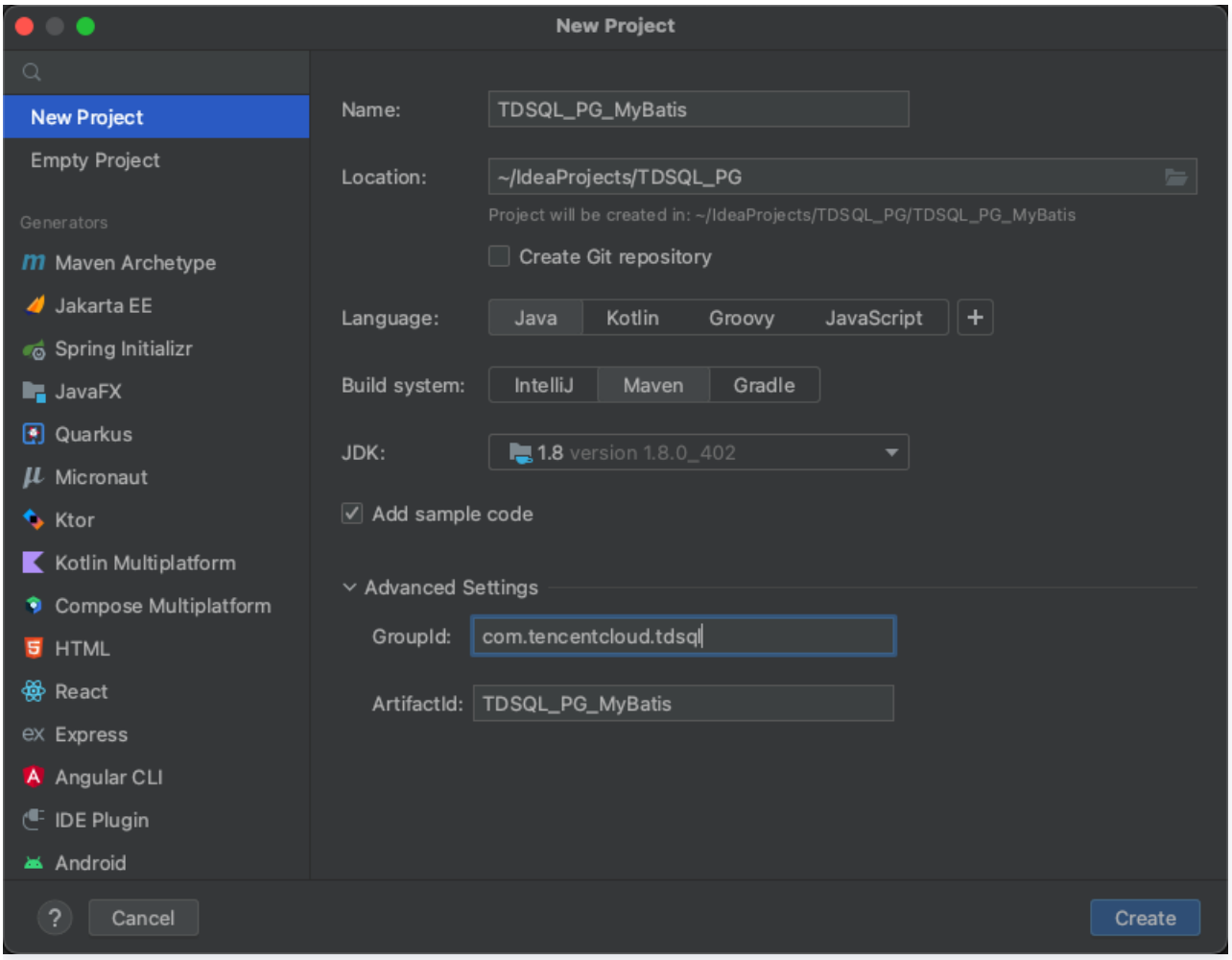
前置依赖

在配置之前需要保证你的机器上已安装对应版本的 JDK，本例是 JDK 8。内核 Oracle 模式下建议使用腾讯自研驱动，请单击 驱动下载 。联系数据库管理员获取数据库相关信息：包括数据库 IP、端口、实例名、用户、密码等。驱动名和连接 URL 如下：

- 驱动名 `com.tencentcloud.tdsql.pg.jdbc.Driver`
- 连接 URL（Oracle 模式下需打开 oracle_compile 参数）
  `jdbc:tdsql-pg://host:port/database?oracle_compile=true`

创建 Maven 项目

创建 Maven 项目，如下：



> **注意：**
> 本文档示例项目使用的 IDE 是 IntelliJ IDEA 2022.3 (Ultimate Edition)，也可根据喜好选择合适的 IDE。

配置 pom.xml 文件，导入项目依赖

配置文件 `pom.xml` 如下：

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>

    <!-- 配置父项依赖 -->
    <parent>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-parent</artifactId>
```

```
        <version>2.7.18</version>
        <relativePath/>
    </parent>

    <groupId>com.tencentcloud.tdsql</groupId>
    <artifactId>TDSQL_PG_SpringBoot</artifactId>
    <version>1.0-SNAPSHOT</version>

    <properties>
        <java.version>1.8</java.version>
        <maven.compiler.source>8</maven.compiler.source>
        <maven.compiler.target>8</maven.compiler.target>
    </properties>

    <dependencies>
        <!-- Spring Boot -->
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter</artifactId>
        </dependency>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-jdbc</artifactId>
        </dependency>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-test</artifactId>
            <scope>test</scope>
        </dependency>
        <!-- TDSQL-PG JDBC驱动 -->
        <dependency>
            <groupId>com.tencentcloud.tdsql</groupId>
            <artifactId>tdsql-pg-connector-java8</artifactId>
            <version>1.1.0</version>
            <scope>system</scope>
            <systemPath>${project.basedir}/lib/tdsql-pg-connector-java8-
1.1.0.jar</systemPath>
        </dependency>
    </dependencies>

    <!-- 配置Maven插件 -->
    <build>
        <plugins>
```

```
            <plugin>
                <groupId>org.springframework.boot</groupId>
                <artifactId>spring-boot-maven-plugin</artifactId>
            </plugin>
        </plugins>
    </build>

</project>
```

1. 通过 `<parent>` 配置父项依赖：配置父项目依赖 `spring-boot-starter-parent`，引入 Spring Boot 的预置配置。
2. 通过 `<dependencies>` 定义项目所依赖的组件，主要包括：
   - `spring-boot-starter`：Spring Boot 的核心启动器。
   - `spring-boot-starter-jdbc`：Spring JDBC 依赖包。
   - `spring-boot-starter-test`：Spring Boot 单元测试依赖包。
   - spring-boot 填写好后会从公共库自动下载。
   - 下载 TDSQL PG JDBC 驱动包，并在 `TDSQL_PG_SpringBoot` 目录下新建 `lib` 目录，将驱动包拷贝到 `lib` 目录下。
3. 配置 Maven 插件 `spring-boot-maven-plugin`，配合 `spring-boot-starter-parent` 可以把 Spring Boot 应用打包成 JAR 来直接运行。

> ⚠ **注意：**
> - 驱动包下载请单击 驱动下载 。
> - 本文所有示例需在启用 Oracle 兼容内核的情况下执行。打开内核 Oracle 兼容模式请参考 Oracle 兼容特性概述 。

配置 Spring Boot

在 `TDSQL_PG_SpringBoot/src/main/resources` 目录下新建 `application.properties` 文件，用于配置 Spring Boot，`application.properties` 配置如下：

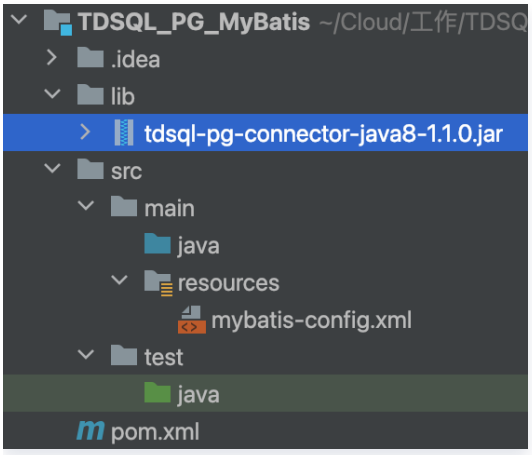
```
# 配置JDBC连接信息：host、port、database、user、password 需要替换为数据库管理员提供的数据库相关信息。
spring.datasource.driverClassName=com.tencentcloud.tdsql.pg.jdbc.Driver
spring.datasource.url=jdbc:tdsql-pg://host:port/database?oracle_compile=true
spring.datasource.username=user
spring.datasource.password=password
# 设置日志打印级别为WARN
```

```
logging.level.root=WARN
```

其中，需要配置 JDBC 连接信息：

- `spring.datasource.driverClassName`：指定驱动包名为 `com.tencentcloud.tdsql.pg.jdbc.Driver`。
- `spring.datasource.url`：指定连接 URL，包括数据库地址、端口、数据库名、连接属性等信息。
- `spring.datasource.username`：指定用户名。
- `spring.datasource.password`：指定相应用户的密码。

至此，已完成 Spring Boot 项目的基本配置，完整的项目目录结构如下图所示：



开发示例

添加数据表对应的实体类

首先，开发使用的数据表为 `STUDENT` 表（`CREATE TABLE STUDENT(ID INT PRIMARY KEY, NAME VARCHAR(50))`），对应地，添加实体类 `Student`，先在 `TDSQL_PG_SpringBoot/src/main/java/` 下创建 Package: `com.tencentcloud.tdsql.pojo` 再，在 `TDSQL_PG_SpringBoot/src/main/java/com/tencentcloud/tdsql/pojo` 下新建 `Student.java` 文件：

```
package com.tencentcloud.tdsql.pojo;

public class Student {
    private int id;
    private String name;

    public Student() {
    }

    public Student(int id, String name) {
```

```
        this.id = id;
        this.name = name;
    }

    public int getId() {
        return id;
    }

    public void setId(int id) {
        this.id = id;
    }

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }

    @Override
    public String toString() {
        return "Student [id=" + id + ", name=" + name + "]";
    }
}
```

定义数据库的增删查改操作

其次，在 `TDSQL_PG_SpringBoot/src/main/java/` **下创建 Package** `com.tencentcloud.tdsql.dao` **，在** `TDSQL_PG_SpringBoot/src/main/java/com/tencentcloud/tdsql/dao` **下新建** `StudentDao.java` **文件，定义数据库的增删查改操作，如下：**

```
package com.tencentcloud.tdsql.dao;

import com.tencentcloud.tdsql.pojo.Student;
import java.util.List;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.jdbc.core.BeanPropertyRowMapper;
import org.springframework.jdbc.core.JdbcTemplate;
import org.springframework.stereotype.Repository;

@Repository
public class StudentDao {
```

```
    @Autowired
    private JdbcTemplate jdbcTemplate;

    // 插入
    public Integer insertStudent(Student student) {
        return jdbcTemplate.update(
                "insert into STUDENT values(?, ?)",
                new Object[]{student.getId(), student.getName()});
    }

    // 查询
    public List<Student> selectAllStudent() {
        return jdbcTemplate.query(
                "select * from STUDENT order by ID",
                new BeanPropertyRowMapper<>(Student.class));
    }

    // 更新
    public Integer updateStudentById(Student student) {
        return jdbcTemplate.update(
                "update STUDENT set NAME = ? where ID = ?",
                new Object[]{student.getName(), student.getId()});
    }

    // 删除
    public Integer deleteStudentById(Integer id) {
        return jdbcTemplate.update("delete from STUDENT where ID = ?",
id);
    }
}
```

`StudentDao` 类通过 `JdbcTemplate` 对象执行 SQL 语句，实现了对 `STUDENT` 表的插入、查询、更新和删除操作。

构建启动类

然后，在 `TDSQL_PG_SpringBoot/src/main/java/com/tencentcloud/tdsql` 下新建 `Application.java` 文件，用于启动 Spring Boot 应用程序，如下：

```
package com.tencentcloud.tdsql;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
```

```
@SpringBootApplication
public class Application {

    public static void main(String[] args) {
        SpringApplication.run(Application.class, args);
    }
}
```

测试

最后，在`TDSQL_PG_SpringBoot/src/test/java/com/tencentcloud/tdsql`下添加测试类`TestStudent`，展示了基本的增删查改功能，如下：

```
package com.tencentcloud.tdsql;

import com.tencentcloud.tdsql.dao.StudentDao;
import com.tencentcloud.tdsql.pojo.Student;
import java.util.List;
import org.junit.jupiter.api.Test;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.boot.test.context.SpringBootTest;

@SpringBootTest
public class TestStudent {
    @Autowired
    private StudentDao studentDao;

    @Test
    void contextLoads() {
        System.out.println("------插入数据------");
        studentDao.insertStudent(new Student(1, "Alex"));
        studentDao.insertStudent(new Student(2, "Bob"));
        studentDao.insertStudent(new Student(3, "John"));
        System.out.println("插入3条数据");

        System.out.println("------查询数据------");
        List<Student> students1 = studentDao.selectAllStudent();
        for (Student student: students1) {
            System.out.println(student);
        }

        System.out.println("------更改数据------");
```

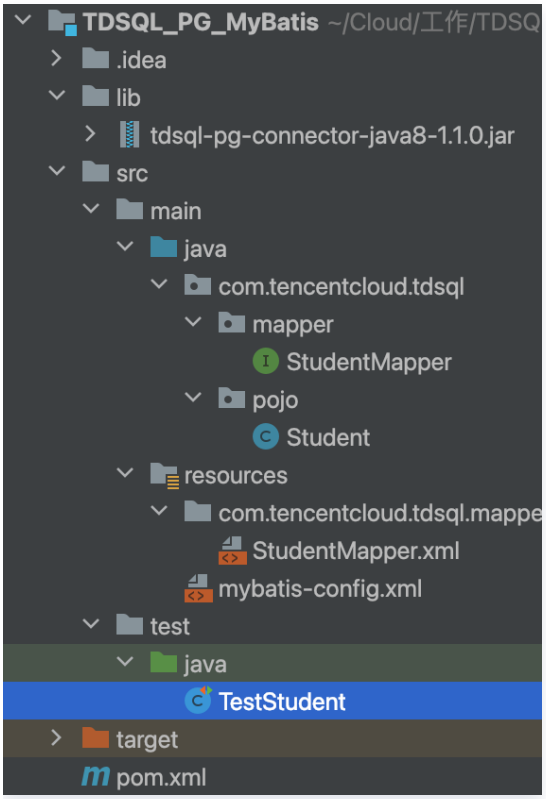
```
        studentDao.updateStudentById(new Student(1, "Alice"));
        System.out.println("更改id为1的数据");

        System.out.println("------查询数据------");
        List<Student> students2 = studentDao.selectAllStudent();
        for (Student student: students2) {
            System.out.println(student);
        }

        System.out.println("------删除数据------");
        studentDao.deleteStudentById(2);
        System.out.println("删除id为2的数据");

        System.out.println("------查询数据------");
        List<Student> students3 = studentDao.selectAllStudent();
        for (Student student: students3) {
            System.out.println(student);
        }
    }
}
```

运行结果如下：

```
------插入数据------
插入3条数据
------查询数据------
Student [id=1, name=Alex]
Student [id=2, name=Bob]
Student [id=3, name=John]
------更改数据------
更改id为1的数据
------查询数据------
Student [id=1, name=Alice]
Student [id=2, name=Bob]
Student [id=3, name=John]
------删除数据------
删除id为2的数据
------查询数据------
Student [id=1, name=Alice]
Student [id=3, name=John]
```

附：项目结构

至此，完成了 Spring Boot 项目的配置和开发示例，项目的整体结构如下图所示：



使用 C/C++ 连接 TDSQL PG
使用 ODBC 连接 TDSQL PG

最近更新时间：2025-02-28 10:59:33

本文介绍如何使用 ODBC 示例程序连接及操作数据库。

获取数据库连接信息

联系数据库管理员获取数据库连接命令：

```
psql -h 10.211.55.30 -p 11345 -U tbase -d tdsql
```

参数说明：

- `-h` 数据库 IP，通常是 CN 的 IP。
- `-p` 数据库端口。
- `-U` 数据库用户。
- `-d` 需要访问的数据库名称。

安装驱动依赖

操作系统上使用 ODBC 驱动依赖于 UnixODBC 管理，需要先在系统中安装 UnixODBC。

UnixODBC 安装

假设安装路径为 UNIXODBC_PATH。

> ⚠ **注意：**
> UnixODBC 版本必须为2.3.7，否则会导致无法连接，可参考源码包安装。

1. 下载 unixODBC

   官网地址unixODBC：https://www.unixodbc.org/unixODBC-2.3.7.tar.gz

2. 安装

   2.1 将 tar 包上传到服务器目录，例如：/usr/local，进入该目录解压安装包：

   ```
   tar -xvf unixODBC*.tar.gz
   ```

   2.2 编译（进入到解压后的目录，以下命令依次执行）

```
./configure --prefix=$UNIXODBC_PATH
make
make install
```

2.3 配置环境变量，编辑 ~/.bashrc。

```
export PATH=$UNIXODBC_PATH/bin:$PATH
export LD_LIBRARY_PATH=$UNIXODBC_PATH/lib:$LD_LIBRARY_PATH
```

2.4 环境变量生效

```
source ~/.bashrc
```

2.5 测试

```
odbcinst -j

    unixODBC 2.3.7
    DRIVERS............: /etc/odbcinst.ini
    SYSTEM DATA SOURCES: /etc/odbc.ini
    FILE DATA SOURCES..: /etc/ODBCDataSources
    USER DATA SOURCES..: /etc/odbc.ini
    SQLULEN Size.......: 8
    SQLLEN Size........: 8
    SQLSETPOSIROW Size.: 8
```

ODBC 驱动安装和配置

1. 下载驱动：请单击 ODBC 驱动包下载 。
2. 安装

2.1 将 tar 包上传到服务器解压：

```
tar -xvf tdsql_psqlodbc*.tar.gz
```

解压 TDSQL ODBC 驱动包，假设解压路径为 `$PG_ODBC_PATH`

2.2 配置环境变量，编辑 ~/.bashrc。

```
export LD_LIBRARY_PATH=$PG_ODBC_PATH:$LD_LIBRARY_PATH
```

2.3 环境变量生效

```
source ~/.bashrc
```

3. 手动配置 ODBC 数据源

3.1 编辑 /etc/odbcinst.ini，输入如下内容：

```
[TDSQL_PG]
Description     = ODBC for TDSQL PostgreSQL
Driver          = $PG_ODBC_PATH/psqlodbcw.so
FileUsage       = 1
```

> **注意：**
> /etc/odbcinst.ini 是通过 odbcinst –j 命令获取的路径，以实际查询配置文件为准。

3.2 编辑 /etc/odbc.ini，输入如下内容：

```
[PG_CON]
Driver=TDSQL_PG
Servername=10.211.55.30
Port=11345
Username=tbase
Password=1234abcd
Database=tdsql
```

参数说明：

- `Driver` 驱动名称，在文件 /etc/odbcinst.ini 中声明。
- `Servername` 服务器名称 /IP。
- `Port` 数据库端口。
- `Username` 数据库用户。
- `Password` 用户密码。
- `Database` 需要访问的数据库名称。

> **注意：**
> 请按照实际情况进行字段值的修改，驱动动态库路径依据实际路径填写。

校验驱动库已经包含依赖：

```
ldd $PG_ODBC_PATH/psqlodbcw.so
```

链接不出现 not found 则为成功。

4. 测试连接

可以使用 unixODBC 自带的 isql 工具进行检测，执行：

```
isql PG_CON -v

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

编写 ODBC 测试程序

示例程序：`test_odbc.c`

```
#include <stdio.h>
#include <stdlib.h>
#include <sql.h>
#include <sqlext.h>

// 定义连接字符串
#define DSN "DSN=PG_CON"

void
print_diag(char *msg, SQLSMALLINT htype, SQLHANDLE handle)
{
    char        sqlstate[32];
    char        message[1000];
    SQLINTEGER  nativeerror;
    SQLSMALLINT textlen;
    SQLRETURN   ret;
    SQLSMALLINT recno = 0;

    if (msg)
```

```
        printf("%s\n", msg);

    do
    {
        recno++;
        ret = SQLGetDiagRec(htype, handle, recno, sqlstate,
&nativeerror,
                            message, sizeof(message), &textlen);
        if (ret == SQL_INVALID_HANDLE)
            printf("Invalid handle\n");
        else if (SQL_SUCCEEDED(ret))
            printf("%s=%s\n", sqlstate, message);
    } while (ret == SQL_SUCCESS);

    if (ret == SQL_NO_DATA && recno == 1)
        printf("No error information\n");
}

// 创建表
void create_table(SQLHDBC dbc) {
    SQLHSTMT stmt;
    SQLRETURN ret;

    ret = SQLAllocHandle(SQL_HANDLE_STMT, dbc, &stmt);
    if (ret != SQL_SUCCESS) {
        fprintf(stderr, "Failed to allocate statement handle\n");
        return;
    }

    ret = SQLExecDirect(stmt, (SQLCHAR*)"CREATE TABLE IF NOT EXISTS
test_table (id SERIAL PRIMARY KEY, name VARCHAR(50), age INTEGER);",
SQL_NTS);
    if (ret != SQL_SUCCESS) {
        fprintf(stderr, "Failed to execute create table statement\n");
    }

    SQLFreeHandle(SQL_HANDLE_STMT, stmt);
}

// drop表
void drop_table(SQLHDBC dbc) {
    SQLHSTMT stmt;
    SQLRETURN ret;
```

```
    ret = SQLAllocHandle(SQL_HANDLE_STMT, dbc, &stmt);
    if (ret != SQL_SUCCESS) {
        fprintf(stderr, "Failed to allocate statement handle\n");
        return;
    }

    ret = SQLExecDirect(stmt, (SQLCHAR*)"DROP TABLE test_table;",
SQL_NTS);
    if (ret != SQL_SUCCESS) {
        fprintf(stderr, "Failed to execute drop table statement\n");
    }

    SQLFreeHandle(SQL_HANDLE_STMT, stmt);
}

// 插入数据
void insert_data(SQLHDBC dbc, const char* name, int age) {
    SQLHSTMT stmt;
    SQLRETURN ret;

    ret = SQLAllocHandle(SQL_HANDLE_STMT, dbc, &stmt);
    if (ret != SQL_SUCCESS) {
        fprintf(stderr, "Failed to allocate statement handle\n");
        return;
    }

    ret = SQLPrepare(stmt, (SQLCHAR*)"INSERT INTO test_table (name, age)
VALUES (?, ?);", SQL_NTS);
    if (ret != SQL_SUCCESS) {
        fprintf(stderr, "Failed to prepare insert statement\n");
        SQLFreeHandle(SQL_HANDLE_STMT, stmt);
        return;
    }

    ret = SQLBindParameter(stmt, 1, SQL_PARAM_INPUT, SQL_C_CHAR,
SQL_VARCHAR, 0, 0, (SQLPOINTER)name, 0, NULL);
    if (ret != SQL_SUCCESS) {
        fprintf(stderr, "Failed to bind parameter 1\n");
        SQLFreeHandle(SQL_HANDLE_STMT, stmt);
        return;
    }
```

```
        ret = SQLBindParameter(stmt, 2, SQL_PARAM_INPUT, SQL_C_LONG,
SQL_INTEGER, 0, 0, (SQLPOINTER)&age, 0, NULL);
    if (ret != SQL_SUCCESS) {
        fprintf(stderr, "Failed to bind parameter 2\n");
        SQLFreeHandle(SQL_HANDLE_STMT, stmt);
        return;
    }

    ret = SQLExecute(stmt);
    if (ret != SQL_SUCCESS) {
        fprintf(stderr, "Failed to execute insert statement\n");
    }

    SQLFreeHandle(SQL_HANDLE_STMT, stmt);
}

// 更新数据
void update_data(SQLHDBC dbc, int id, const char* name, int age) {
    SQLHSTMT stmt;
    SQLRETURN ret;

    ret = SQLAllocHandle(SQL_HANDLE_STMT, dbc, &stmt);
    if (ret != SQL_SUCCESS) {
        fprintf(stderr, "Failed to allocate statement handle\n");
        return;
    }

    ret = SQLPrepare(stmt, (SQLCHAR*)"UPDATE test_table SET name = ?,
age = ? WHERE id = ?;", SQL_NTS);
    if (ret != SQL_SUCCESS) {
        fprintf(stderr, "Failed to prepare update statement\n");
        SQLFreeHandle(SQL_HANDLE_STMT, stmt);
        return;
    }

    ret = SQLBindParameter(stmt, 1, SQL_PARAM_INPUT, SQL_C_CHAR,
SQL_VARCHAR, 0, 0, (SQLPOINTER)name, 0, NULL);
    if (ret != SQL_SUCCESS) {
        fprintf(stderr, "Failed to bind parameter 1\n");
        SQLFreeHandle(SQL_HANDLE_STMT, stmt);
        return;
    }
```

```
    ret = SQLBindParameter(stmt, 2, SQL_PARAM_INPUT, SQL_C_LONG,
SQL_INTEGER, 0, 0, (SQLPOINTER)&age, 0, NULL);
    if (ret != SQL_SUCCESS) {
        fprintf(stderr, "Failed to bind parameter 2\n");
        SQLFreeHandle(SQL_HANDLE_STMT, stmt);
        return;
    }

    ret = SQLBindParameter(stmt, 3, SQL_PARAM_INPUT, SQL_C_LONG,
SQL_INTEGER, 0, 0, (SQLPOINTER)&id, 0, NULL);
    if (ret != SQL_SUCCESS) {
        fprintf(stderr, "Failed to bind parameter 3\n");
        SQLFreeHandle(SQL_HANDLE_STMT, stmt);
        return;
    }

    ret = SQLExecute(stmt);
    if (ret != SQL_SUCCESS) {
        fprintf(stderr, "Failed to execute update statement\n");
    }

    SQLFreeHandle(SQL_HANDLE_STMT, stmt);
}

// 删除数据
void delete_data(SQLHDBC dbc, int id) {
    SQLHSTMT stmt;
    SQLRETURN ret;

    ret = SQLAllocHandle(SQL_HANDLE_STMT, dbc, &stmt);
    if (ret != SQL_SUCCESS) {
        fprintf(stderr, "Failed to allocate statement handle\n");
        return;
    }

    ret = SQLPrepare(stmt, (SQLCHAR*)"DELETE FROM test_table WHERE id =
?;", SQL_NTS);
    if (ret != SQL_SUCCESS) {
        fprintf(stderr, "Failed to prepare delete statement\n");
        SQLFreeHandle(SQL_HANDLE_STMT, stmt);
        return;
    }
```

```
    ret = SQLBindParameter(stmt, 1, SQL_PARAM_INPUT, SQL_C_LONG,
SQL_INTEGER, 0, 0, (SQLPOINTER)&id, 0, NULL);
    if (ret != SQL_SUCCESS) {
        fprintf(stderr, "Failed to bind parameter\n");
        SQLFreeHandle(SQL_HANDLE_STMT, stmt);
        return;
    }

    ret = SQLExecute(stmt);
    if (ret != SQL_SUCCESS) {
        fprintf(stderr, "Failed to execute delete statement\n");
    }

    SQLFreeHandle(SQL_HANDLE_STMT, stmt);
}

// 查询数据
void select_data(SQLHDBC dbc) {
    SQLHSTMT stmt;
    SQLRETURN ret;

    ret = SQLAllocHandle(SQL_HANDLE_STMT, dbc, &stmt);
    if (ret != SQL_SUCCESS) {
        fprintf(stderr, "Failed to allocate statement handle\n");
        return;
    }

    ret = SQLExecDirect(stmt, (SQLCHAR*)"SELECT * FROM test_table;",
SQL_NTS);
    if (ret != SQL_SUCCESS) {
        fprintf(stderr, "Failed to execute select statement\n");
        SQLFreeHandle(SQL_HANDLE_STMT, stmt);
        return;
    }

    SQLLEN id, age;
    SQLCHAR name[50];
    while (SQLFetch(stmt) == SQL_SUCCESS) {
        SQLGetData(stmt, 1, SQL_C_LONG, &id, 0, NULL);
        SQLGetData(stmt, 2, SQL_C_CHAR, name, sizeof(name), NULL);
        SQLGetData(stmt, 3, SQL_C_LONG, &age, 0, NULL);
        printf("ID: %ld, Name: %s, Age: %ld\n", id, name, age);
    }
```

```
    SQLFreeHandle(SQL_HANDLE_STMT, stmt);
}

int main() {
    SQLHENV env;
    SQLHDBC dbc;
    SQLRETURN ret;

    // 初始化环境句柄
    ret = SQLAllocHandle(SQL_HANDLE_ENV, SQL_NULL_HANDLE, &env);
    if (ret != SQL_SUCCESS) {
        fprintf(stderr, "Failed to allocate environment handle\n");
        return 1;
    }

    // 设置环境属性
    ret = SQLSetEnvAttr(env, SQL_ATTR_ODBC_VERSION,
(SQLPOINTER)SQL_OV_ODBC3, 0);
    if (ret != SQL_SUCCESS) {
        fprintf(stderr, "Failed to set environment attribute\n");
        SQLFreeHandle(SQL_HANDLE_ENV, env);
        return 1;
    }

    // 分配连接句柄
    ret = SQLAllocHandle(SQL_HANDLE_DBC, env, &dbc);
    if (ret != SQL_SUCCESS) {
        fprintf(stderr, "Failed to allocate connection handle\n");
        SQLFreeHandle(SQL_HANDLE_ENV, env);
        return 1;
    }

    // 建立连接
    ret = SQLDriverConnect(dbc, NULL, (SQLCHAR*)DSN, SQL_NTS, NULL, 0,
NULL, SQL_DRIVER_COMPLETE);
    if (ret != SQL_SUCCESS) {
        print_diag("SQLDriverConnect failed.", SQL_HANDLE_DBC, dbc);
        SQLFreeHandle(SQL_HANDLE_DBC, dbc);
        SQLFreeHandle(SQL_HANDLE_ENV, env);
        return 1;
    }
```

```
    // 创建表
    create_table(dbc);

    // 插入数据
    insert_data(dbc, "John", 25);
    insert_data(dbc, "Alice", 30);

    // 查询数据
    select_data(dbc);

    // 更新数据
    update_data(dbc, 1, "John Doe", 26);

    // 删除数据
    delete_data(dbc, 2);

    // 查询数据
    select_data(dbc);

    // drop表
    drop_table(dbc);

    // 断开连接
    SQLDisconnect(dbc);

    // 释放连接句柄和环境句柄
    SQLFreeHandle(SQL_HANDLE_DBC, dbc);
    SQLFreeHandle(SQL_HANDLE_ENV, env);

    return 0;
}
```

编译：

```
gcc -o test_odbc -g test_odbc.c -I$UNIXODBC_PATH/include -
L$UNIXODBC_PATH/lib -lodbc
```

运行：

```
./test_odbc
```

结果：

```
ID: 1, Name: John, Age: 25
ID: 2, Name: Alice, Age: 30
ID: 1, Name: John Doe, Age: 8589934618
```

使用 OCI 连接 TDSQL PG

最近更新时间：2025-02-28 10:59:33

本文介绍如何使用 OCI 示例程序连接及操作数据库。

获取数据库连接信息

联系数据库管理员获取数据库连接命令：

```
psql -h 10.211.55.30 -p 11345 -U tbase -d tdsql
```

参数说明：

- `-h` 数据库 IP，通常是 CN 的 IP。
- `-p` 数据库端口。
- `-U` 数据库用户。
- `-d` 需要访问的数据库名称。

安装驱动依赖

1. 下载驱动：请单击 TDSQL-PG OCI 驱动包下载 。
2. 安装

   ```
   tar -xvf tdsql_psqloci_tlinux4_x86_64.tar.gz
   ```

   解压 TDSQL OCI 驱动包，假设解压路径为 `$PG_OCI_PATH`
   安装包文件说明：
   - `include` 依赖的头文件，OCI 函数声明。
   - `lib` 依赖库，OCI 函数实现。
   - `oci_test.c` demo，提供参考。
   - `pg_service.conf` 数据库连接信息配置（数据源设置）
   - `README` demo 操作说明
   - `run.sh` demo 执行脚本
3. 配置相关环境信息

   3.1 配置连接文件，编辑 pg_service.conf：

   ```
   [db_test] #数据源名称
   #数据库服务IP
   hostaddr=10.211.55.30
   ```

```
#数据库服务端口号
port=11345
#需要连接的数据库
dbname=tdsql
```

> **注意：**
> OCI 应用程序连接数据库进行 dbname 设置时，使用的是 `pg_service.conf` 中的数据源名称而不是实际访问的数据库名称。在 OCI 应用程序中配置数据源、用户名和密码。在 `pg_service.conf` 配置数据库 IP、端口、实际访问的数据库名。

3.2 配置环境变量，编辑 ~/.bashrc。

```
export LD_LIBRARY_PATH=$PG_OCI_PATH/lib:$LD_LIBRARY_PATH
export PGSERVICEFILE=$PG_OCI_PATH/pg_service.conf
```

> **说明：**
> PGSERVICEFILE：pg_services.conf 文件环境变量，依据实际路径填写。

3.3 环境变量生效

```
source ~/.bashrc
```

校验驱动库依赖完整性：

```
ldd $PG_OCI_PATH/lib/liboci.so
```

链接不出现 not found 则为成功。

编写 OCI 测试程序

示例程序：`oci_test.c`，已提供在 `$PG_OCI_PATH` 路径中，修改用户名、密码、数据源信息即可。

```
#include "oci.h"
#include "oratypes.h"
#include "xa.h"
#include <unistd.h>
#include <stdbool.h>
#include <stdio.h>
#include <stdlib.h>
```

```
#include <string.h>
#include <sys/signal.h>
#include <unistd.h>
#include <pthread.h>

#define MAX_SIZE 1024
#define MAX_STR_LENTH 256
#define MAX_COUNT 3

typedef struct __OCI_GLOBAL_CONTEXT_TYPE
{
    OCIEnv*     envhp;
    OCIServer*  srvhp;
    OCISvcCtx*  svchp;
    OCIError*   errhp;
    OCISession* authp;
    OCIStmt*    stmthp;
} OCI_GLOBAL_CONTEXT;

//准备阶段测试
sword oci_alloc_connect(bool useLogon, OCI_GLOBAL_CONTEXT* ctxptr, char*
uname, char* passwd, char* dbname);
sword oci_free_disconnect(bool useLogon, OCI_GLOBAL_CONTEXT* ctxptr);
void oci_checkerr(OCIError* errhp, sword status);
sword oci_exec_sample_sql(OCI_GLOBAL_CONTEXT* ctxptr, char* sql, ub4
mode);

sword case0_create(OCI_GLOBAL_CONTEXT *ctxptr);
sword case1_select(OCI_GLOBAL_CONTEXT *ctxptr);
sword case2_update(OCI_GLOBAL_CONTEXT *ctxptr);
sword case3_insert(OCI_GLOBAL_CONTEXT *ctxptr);
sword case4_delete(OCI_GLOBAL_CONTEXT *ctxptr);
sword case5_vacuum(OCI_GLOBAL_CONTEXT *ctxptr);

//根据位置绑定参数和定义结果集测试
sword fetch_by_define(void* ptr, char* sql);
void test();

sword oci_free_handle(OCI_GLOBAL_CONTEXT* ctxptr, char* sql, ub4 mode);
sword oci_alloc_handle(OCI_GLOBAL_CONTEXT* ctxptr, char* sql, ub4 mode);

/*
```

```
测试函数：OCIInitialize()、OCIEnvInit()、OCIAttrSet()、OCIStmtPrepare()、
OCIStmtExecute()、OCITransCommit()、OCIDescriptorAlloc()、
OCIDefineByPos()、OCIStmtFetch()、OCIErrorGet()
    1、OCIInitialize()初始化 OCI 应用环境
    2、OCIEnvInit()初始化环境句柄
    3、OCIDescriptorAlloc()分配句柄
    4、OCIAttrSet()设置会话句柄用户名和密码
    5、OCIAttrSet()设置环境句柄属性
    6、创建并开始一个用户两个会话
    7、断开一个会话
    8、OCIErrorGet()读取错误信息
    9、创建表并执行以下操作
    a.  OCIStmtPrepare()准备 sql 语句
    b.  OCIStmtExecute()执行 sql 语句
    c.  OCITransCommit ()提交事务
    d.  OCIDefineByPos ()定义输出变量
    e.  OCIStmtFetch()获取数据
    10、结束释放资源，断开数据库连接，释放句柄
*/

/* 需要修改用户名、密码及在 pg_service.conf 中配置的数据源名称，这3个变量均不能为空
*/
char*       uname     = "tbase";
char*       passwd    = "1234@abcd";
char*       dbname    = "db_test";

int main(int argc, char* argv[])
{
    test();
    return 0;

}

void test()
{

    OCI_GLOBAL_CONTEXT ctxptr = {NULL};
    sword              err    = OCI_SUCCESS;

    do {
        err = oci_alloc_connect(true, &ctxptr, uname, passwd, dbname);
        if (err != OCI_SUCCESS) {
            break;
```

```
        }

        err = case0_create(&ctxptr);
        if (err != OCI_SUCCESS) {
            break;
        }

        err = case3_insert(&ctxptr);
        if (err != OCI_SUCCESS) {
            break;
        }

        err = case1_select(&ctxptr);
        if (err != OCI_SUCCESS) {
            break;
        }

        oci_alloc_handle(&ctxptr, NULL, OCI_COMMIT_ON_SUCCESS);
        err = case2_update(&ctxptr);
        if (err != OCI_SUCCESS) {
            break;
        }
        err = case5_vacuum(&ctxptr);
        if (err != OCI_SUCCESS) {
            break;
        }

        err = case4_delete(&ctxptr);
        if (err != OCI_SUCCESS) {
            break;
        }
        oci_free_handle(&ctxptr, NULL, OCI_COMMIT_ON_SUCCESS);

        oci_free_disconnect(false, &ctxptr);

    } while (0);
}

sword case0_create(OCI_GLOBAL_CONTEXT *ctxptr)
{
    char*    sql = NULL;
    sword    err = OCI_ERROR;
```

```
    do {
        // 删除表重建
        sql = "drop table if exists case1_bindbypos_define;";
        err = oci_exec_sample_sql(ctxptr, sql, OCI_COMMIT_ON_SUCCESS);

        sql = "create table case1_bindbypos_define(id int, name
character varying(255), created character varying(12), attr text);";
        err = oci_exec_sample_sql(ctxptr, sql, OCI_COMMIT_ON_SUCCESS);

    } while (0);
    return err;
}

sword case1_select(OCI_GLOBAL_CONTEXT *ctxptr)
{
    char*    sql = NULL;
    sword    err = OCI_ERROR;

    do {

        //获取结果集
        sql = "select * from case1_bindbypos_define where id >10";
        err = fetch_by_define((void*)ctxptr, sql);
        if (err != OCI_SUCCESS) {
            break;
        }

    } while (0);
    return err;
}

sword case2_update(OCI_GLOBAL_CONTEXT *ctxptr)
{
    char*    sql = NULL;
    sword    err = OCI_ERROR;

    do {

        // 更新数据
        sql = "update case1_bindbypos_define set name='YYYYYYYY' where
id=100;";
        err = oci_exec_sample_sql(ctxptr, sql, OCI_COMMIT_ON_SUCCESS);
```

```
    } while (0);
    return err;
}

sword case3_insert(OCI_GLOBAL_CONTEXT *ctxptr)
{
    char*    sql = NULL;
    sword    err = OCI_ERROR;

    do {
        // 插入数据
        sql = "insert into case1_bindbypos_define
values(generate_series(1, 10000), 'AAAAAAAAAAAAAAA', 'BBBBBBBBB',
'CCCCCCCCCCCCC');";
        err = oci_exec_sample_sql(ctxptr, sql, OCI_COMMIT_ON_SUCCESS);

    } while (0);
    return err;
}

sword case4_delete(OCI_GLOBAL_CONTEXT *ctxptr)
{
    char*    sql = NULL;
    sword    err = OCI_ERROR;

    do {

        //删除数据
        sql = "delete from case1_bindbypos_define where id < 10001;";
        err = oci_exec_sample_sql(ctxptr, sql, OCI_COMMIT_ON_SUCCESS);

    } while (0);
    return err;
}

sword case5_vacuum(OCI_GLOBAL_CONTEXT *ctxptr)
{
    char*    sql = NULL;
    sword    err = OCI_ERROR;

    do {
        sql = "vacuum full case1_bindbypos_define";
        err = oci_exec_sample_sql(ctxptr, sql, OCI_COMMIT_ON_SUCCESS);
```

```
    } while (0);
    return err;
}

sword fetch_by_define(void* ptr, char* sql)
{
#define max_one_size 5
    sword                status   = OCI_SUCCESS;
    OCI_GLOBAL_CONTEXT* ctxptr   = (OCI_GLOBAL_CONTEXT*)ptr;
    OCIDefine*           dnfp[4] = {NULL};

    int ret = -1;
    int j = 0;

    do {
        int  id[max_one_size]        = {0};
        char name[max_one_size][15] = {{'\0'}};
        char date[max_one_size][12] = {{'\0'}};
        char attr[max_one_size][20] = {{'\0'}};

        // 1 申请执行语句
        if (ctxptr->stmthp == NULL) {
            if ((status = OCIHandleAlloc((dvoid*)ctxptr->envhp,
(dvoid**)&ctxptr->stmthp, (ub4)OCI_HTYPE_STMT,
                                         (CONST size_t)0, (dvoid**)0))
!= OCI_SUCCESS) {
                oci_checkerr(ctxptr->errhp, status);
                break;
            }
        }

        // 3 query sql
        status = OCIStmtPrepare(ctxptr->stmthp, ctxptr->errhp,
(text*)sql, (ub4)strlen((char*)sql), OCI_NTV_SYNTAX,
                                OCI_DEFAULT);
        if (status != OCI_SUCCESS) {
            oci_checkerr(ctxptr->errhp, status);
            break;
        }
        //将结果集内存按查询位置绑定到对应列
        if (OCIDefineByPos(ctxptr->stmthp, &dnfp[0], ctxptr->errhp, 1,
(void*)&id[0], sizeof(int), SQLT_INT, (void*)0,
```

```
                                (ub2*)0, (ub2*)0, OCI_DEFAULT) !=
OCI_SUCCESS) {
            break;
        }
        printf("SUCCESS: OCIDefineByPos ()定义输出变量\n");

        if (OCIDefineByPos(ctxptr->stmthp, &dnfp[1], ctxptr->errhp, 2,
(void*)name[0], 15, SQLT_STR, (void*)0, (ub2*)0,
                                (ub2*)0, OCI_DEFAULT) != OCI_SUCCESS) {
            break;
        }

        if (OCIDefineByPos(ctxptr->stmthp, &dnfp[2], ctxptr->errhp, 3,
(void*)date[0], 12, SQLT_STR, (void*)0, (ub2*)0,
                                (ub2*)0, OCI_DEFAULT) != OCI_SUCCESS) {
            break;
        }

        if (OCIDefineByPos(ctxptr->stmthp, &dnfp[3], ctxptr->errhp, 4,
(void*)attr[0], 20, SQLT_STR, (void*)0, (ub2*)0,
                                (ub2*)0, OCI_DEFAULT) != OCI_SUCCESS) {
            break;
        }
        //因为结果集中的整个列都在一块内存，需要告知分割大小
        if (OCIDefineArrayOfStruct(dnfp[0], ctxptr->errhp, sizeof(int),
0, 0, 0) != OCI_SUCCESS) {
            break;
        }
        if (OCIDefineArrayOfStruct(dnfp[1], ctxptr->errhp, 15, 0, 0, 0)
!= OCI_SUCCESS) {
            break;
        }
        if (OCIDefineArrayOfStruct(dnfp[2], ctxptr->errhp, 12, 0, 0, 0)
!= OCI_SUCCESS) {
            break;
        }
        if (OCIDefineArrayOfStruct(dnfp[3], ctxptr->errhp, 20, 0, 0, 0)
!= OCI_SUCCESS) {
            break;
        }
        //SQL执行
        status = OCIStmtExecute(ctxptr->svchp, ctxptr->stmthp, ctxptr-
>errhp, (ub4)max_one_size, (ub4)0, NULL, NULL,
```

```
                                    OCI_DEFAULT);
        if (status != OCI_SUCCESS) {
            oci_checkerr(ctxptr->errhp, status);
            break;
        }

        // fetch data
        status = OCIStmtFetch(ctxptr->stmthp, ctxptr->errhp,
(ub4)max_one_size, (ub4)OCI_FETCH_NEXT, (ub4)OCI_DEFAULT);
        printf("SUCCESS: OCIStmtFetch()获取数据\n");
        if (status == OCI_SUCCESS) {
            for (j = 0; j < max_one_size; j++) {
                printf("%d %d %s %s %s\n", status, id[j], name[j],
date[j], attr[j]);
            }
        }
        else if (status == OCI_NO_DATA) {
            ub4 ret_val = 0, ret_size = 0;

            //获取属性值之后会通过循环打印变量值校验
            if ((OCIAttrGet((void*)ctxptr->stmthp, (ub4)OCI_HTYPE_STMT,
(void*)&ret_val, (ub4*)&ret_size,
                            (ub4)OCI_ATTR_ROW_COUNT, ctxptr->errhp)) !=
OCI_SUCCESS) {
                printf("DCIAttrGet fail\n");
                break;
            }
            printf("SUCCESS: OCIAttrGet()\n");
            for (j = 0; j < ret_val; j++) {
                printf("%d %d %s %s %s\n", status, id[j], name[j],
date[j], attr[j]);
            }
        }


        ret = 0;
    } while (0);
    //释放stmt句柄
    if (ctxptr->stmthp != NULL) {
        status = OCIHandleFree((dvoid*)ctxptr->stmthp,
(ub4)OCI_HTYPE_STMT);
        if (status != OCI_SUCCESS) {
            printf("OCIHandleFree OCI_HTYPE_STMT failed\n");
```

```
            return OCI_ERROR;
        }
        ctxptr->stmthp = NULL;
    }

    if (ret == -1) {
        return OCI_ERROR;
    }
    return OCI_SUCCESS;
}

sword oci_alloc_connect(bool useLogon, OCI_GLOBAL_CONTEXT* ctxptr, char*
uname, char* passwd, char* dbname)
{
    sword err = OCI_SUCCESS;
    do{
        //初始化，属于准备过程，无实际运行结果
        err = OCIInitialize((ub4)OCI_DEFAULT, (dvoid*)0, (dvoid * (*)
(dvoid*, size_t))0,
                            (dvoid * (*)(dvoid*, dvoid*, size_t))0,
(void (*)(dvoid*, dvoid*))0);
        if (err != OCI_SUCCESS) {
            printf("FAILED: OCIInitialize()\n");
            break;
        }

        printf("SUCCESS: OCIInitialize()初始化OCI应用环境\n");

        //申请env句柄，属于准备过程，无实际运行结果
        err = OCIEnvInit((OCIEnv**)&ctxptr->envhp, OCI_DEFAULT, 0,
NULL);
        if (err != OCI_SUCCESS) {
            printf("FAILED: OCIEnvInit()\n");
            break;
        }
        printf("SUCCESS: OCIEnvInit()初始化环境句柄\n");


        //申请错误句柄，属于准备过程，无实际运行结果
        if (OCIHandleAlloc((dvoid*)ctxptr->envhp, (dvoid**)&ctxptr-
>errhp, (ub4)OCI_HTYPE_ERROR, (size_t)0,
                           (dvoid**)0)) {
            printf("FAILED: OCIHandleAlloc() on ctxptr->errhp\n");
```

```
            break;
        }

        //申请 srv 句柄，属于准备过程，无实际运行结果
        if (OCIHandleAlloc((dvoid*)ctxptr->envhp, (dvoid**)&ctxptr-
>srvhp, (ub4)OCI_HTYPE_SERVER, (size_t)0,
                           (dvoid**)0)) {
            printf("FAILED: OCIHandleAlloc() on ctxptr->srvhp\n");
            break;
        }

        //申请 svch 句柄，属于准备过程，无实际运行结果
        if (OCIHandleAlloc((dvoid*)ctxptr->envhp, (dvoid**)&ctxptr-
>svchp, (ub4)OCI_HTYPE_SVCCTX, (size_t)0,
                           (dvoid**)0)) {
            printf("FAILED: OCIHandleAlloc() on ctxptr->svchp\n");
            break;
        }

        //申请 auth 句柄，属于准备过程，无实际运行结果
        if (OCIHandleAlloc((dvoid*)ctxptr->envhp, (dvoid**)&ctxptr-
>authp, (ub4)OCI_HTYPE_SESSION, (size_t)0,
                           (dvoid**)0)) {
            printf("FAILED: OCIHandleAlloc() on ctxptr->authp\n");
            break;
        }
        printf("SUCCESS: OCIHandleAlloc()分配句柄\n");

        //申请执行语句
        if (OCIHandleAlloc((dvoid*)ctxptr->envhp, (dvoid**)&ctxptr-
>stmthp, (ub4)OCI_HTYPE_STMT,
                                   (CONST size_t)0, (dvoid**)0)) {
            printf("FAILED: OCIHandleAlloc() on ctxptr->stmthp\n");
            break;
        }


        //非测试接口，属于辅助接口
        if (OCIServerAttach(ctxptr->srvhp, ctxptr->errhp, (text*)dbname,
(sb4)strlen((char*)dbname),
                           (ub4)OCI_DEFAULT)) {
            printf("FAILED: OCIServerAttach()\n");
            break;
```

```
        }

        //将 srvh 属性设置到 svch 句柄，属于准备过程，无实际运行结果
        if (OCIAttrSet((dvoid*)ctxptr->svchp, (ub4)OCI_HTYPE_SVCCTX,
(dvoid*)ctxptr->srvhp, (ub4)0,
                        (ub4)OCI_ATTR_SERVER, ctxptr->errhp)) {
            printf("FAILED: OCIAttrSet() server attribute\n");
            break;
        }
        printf("SUCCESS: OCIAttrSet()设置环境句柄属性\n"); //注意，
OCIHandleAlloc申请其他句柄过程中已经将其保存到环境句柄，OCIAttrSet将srvhp句柄属性
保存到svchp过程相当于也同时保存到env

        //将 uname 设置到 auth 句柄，属于准备过程，无实际运行结果
        if (OCIAttrSet((dvoid*)ctxptr->authp, (ub4)OCI_HTYPE_SESSION,
(dvoid*)uname, (ub4)strlen((char*)uname),
                        (ub4)OCI_ATTR_USERNAME, ctxptr->errhp)) {
            printf("FAILED: OCIAttrSet() userid\n");
            break;
        }
        printf("SUCCESS: OCIAttrSet()设置会话句柄用户名\n");

        //将 passwd 设置到 auth 句柄，属于准备过程，无实际运行结果
        if (OCIAttrSet((dvoid*)ctxptr->authp, (ub4)OCI_HTYPE_SESSION,
(dvoid*)passwd, (ub4)strlen((char*)passwd),
                        (ub4)OCI_ATTR_PASSWORD, ctxptr->errhp)) {
            printf("FAILED: OCIAttrSet() passwd\n");
            break;
        }
        printf("SUCCESS: OCIAttrSet()设置会话句柄密码\n");

        //开启一个会话，后续代码中有数据库操作，可以证明该接口生效
        if (OCISessionBegin((dvoid*)ctxptr->svchp, ctxptr->errhp,
ctxptr->authp, (ub4)OCI_CRED_RDBMS,
                            (ub4)OCI_DEFAULT)) {
            text sqlstate[6] = "", msg[256] = "";
            sb4  errcode = 0;
            OCIErrorGet(ctxptr->errhp, 1, sqlstate, &errcode, msg, 256,
OCI_HTYPE_ERROR);
            printf("SUCCESS: OCIErrorGet()读取错误信息\n");
            printf("initialize failed: %d %s\n", errcode, msg);
            break;
        }
```

```
        printf("SUCCESS: 创建并开始一个用户会话\n");

        //将 auth 设置到 svchp 句柄，属于准备过程，无实际运行结果
        if (OCIAttrSet((dvoid*)ctxptr->svchp, (ub4)OCI_HTYPE_SVCCTX,
(dvoid*)ctxptr->authp, (ub4)0,
                        (ub4)OCI_ATTR_SESSION, ctxptr->errhp)) {
            printf("FAILED: OCIAttrSet() session\n");
            break;
        }

        return OCI_SUCCESS;
    } while (0);
    return OCI_ERROR;
}

void oci_checkerr(OCIError* errhp, sword status)
{
    text errbuf[512] = {'\0'};
    sb4  errcode;

    if (status == OCI_SUCCESS) {
        return;
    }

    switch (status) {
        case OCI_SUCCESS_WITH_INFO: {
            printf("Error - OCI_SUCCESS_WITH_INFO\n");
            OCIErrorGet((void*)errhp, (ub4)1, (text*)NULL, &errcode,
errbuf, (ub4)sizeof(errbuf), (ub4)OCI_HTYPE_ERROR);
            printf("Error %d - %s\n", errcode, errbuf);
            break;
        }
        case OCI_NEED_DATA: {
            printf("Error - OCI_NEED_DATA\n");
            break;
        }
        case OCI_NO_DATA: {
            printf("Error - OCI_NO_DATA\n");
            break;
        }
        case OCI_ERROR: {
            OCIErrorGet((void*)errhp, (ub4)1, (text*)NULL, &errcode,
errbuf, (ub4)sizeof(errbuf), (ub4)OCI_HTYPE_ERROR);
```

```
            printf("OCI_ERROR %d - %s\n", errcode, errbuf);
            break;
        }
        case OCI_INVALID_HANDLE: {
            printf("Error - OCI_INVALID_HANDLE\n");
            break;
        }
        case OCI_STILL_EXECUTING: {
            printf("Error - OCI_STILL_EXECUTING\n");
            break;
        }
        case OCI_CONTINUE: {
            printf("Error - OCI_CONTINUE\n");
            break;
        }
        default: {
            printf("Error - %d\n", status);
            break;
        }
    }
}
//退出，并释放对应的句柄资源
sword oci_free_disconnect(bool useLogon, OCI_GLOBAL_CONTEXT* ctxptr)
{
    sword err = OCI_SUCCESS;

    printf("SUCCESS: 结束释放资源，断开数据库连接，释放句柄\n");
    //结束会话
    err = OCILogoff(ctxptr->svchp, ctxptr->errhp);
    if (err != OCI_SUCCESS) {
        printf("FAILED: OCILogoff()\n");
    }

    //句柄释放
    if (ctxptr->svchp != NULL) {
        err = OCIHandleFree((dvoid*)ctxptr->svchp,
(ub4)OCI_HTYPE_SVCCTX);
        if (err != OCI_SUCCESS) {
            printf("FAILED: OCIHandleFree() on ctxptr->svchp\n");
        }
        ctxptr->svchp = NULL;
    }
    if (ctxptr->errhp) {
```

```
        err            = OCIHandleFree((dvoid*)ctxptr->errhp,
(ub4)OCI_HTYPE_ERROR);
        ctxptr->errhp = NULL;
    }

    if (ctxptr->envhp) {
        err            = OCIHandleFree((dvoid*)ctxptr->envhp,
(ub4)OCI_HTYPE_ENV);
        ctxptr->envhp = NULL;
    }

    return err;
}

sword oci_exec_sample_sql(OCI_GLOBAL_CONTEXT* ctxptr, char* sql, ub4
mode)
{
    sword err      = OCI_SUCCESS;
    bool  new_stmt = false;

    do {
        //SQL 准备，无实际运行结果
        err = OCIStmtPrepare(ctxptr->stmthp, ctxptr->errhp, (const
text*)sql, (ub4)strlen((char*)sql),
                             (ub4)OCI_NTV_SYNTAX, (ub4)OCI_DEFAULT);
        if (err != OCI_SUCCESS) {
            text sqlstate[6] = "", msg[256] = "";
            sb4  errcode = 0;

            OCIErrorGet(ctxptr->errhp, 1, sqlstate, &errcode, msg, 256,
OCI_HTYPE_ERROR);
            printf("OCIStmtPrepare failed: %d %s\n", errcode, msg);
            break;
        }

        //SQL 执行，并未提交，需要执行 OCITransCommit 后方可验证
        err = OCIStmtExecute(ctxptr->svchp, ctxptr->stmthp, ctxptr-
>errhp, 1, 0, (OCISnapshot*)0, (OCISnapshot*)0, mode);
        if (err != OCI_SUCCESS) {
            text sqlstate[6] = "", msg[256] = "";
            sb4  errcode = 0;
```

```
            OCIErrorGet(ctxptr->errhp, 1, sqlstate, &errcode, msg, 256,
OCI_HTYPE_ERROR);
            printf("OCIStmtExecute failed: %d %s\n", errcode, msg);
            break;
        }

        //提交
        //具体，SQL 命令执行：\d case1_bindbypos_define，可查看表已经存在
        if (mode == OCI_DEFAULT) {
            err = OCITransCommit(ctxptr->svchp, ctxptr->errhp,
(ub4)OCI_DEFAULT);
            if (err != OCI_SUCCESS) {
                printf("OCITransCommit failed\n");
                break;
            }
        }
    } while (0);

    if (new_stmt && ctxptr->stmthp != NULL) {
        //释放句柄，无实际运行结果
        err = OCIHandleFree((dvoid*)ctxptr->stmthp,
(ub4)OCI_HTYPE_STMT);
        if (err != OCI_SUCCESS) {
            printf("OCIHandleFree failed\n");
        }
        ctxptr->stmthp = NULL;
    }

    return err;
}

sword oci_alloc_handle(OCI_GLOBAL_CONTEXT* ctxptr, char* sql, ub4 mode)
{
    sword err      = OCI_SUCCESS;
    bool  new_stmt = false;

    do {
        //申请执行语句
        if (ctxptr->stmthp == NULL) {
            if (OCIHandleAlloc((dvoid*)ctxptr->envhp, (dvoid**)&ctxptr-
>stmthp, (ub4)OCI_HTYPE_STMT, (CONST size_t)0,
                               (dvoid**)0)) {
                printf("FAILED: alloc statement handle\n");
```

```
                break;
            }
            else {
                new_stmt = true;
            }
        }
    } while (0);

    return err;
}

sword oci_free_handle(OCI_GLOBAL_CONTEXT* ctxptr, char* sql, ub4 mode)
{
    sword err       = OCI_SUCCESS;

    if ( ctxptr->stmthp != NULL) {
        //释放句柄，无实际运行结果
        err = OCIHandleFree((dvoid*)ctxptr->stmthp,
(ub4)OCI_HTYPE_STMT);
        if (err != OCI_SUCCESS) {
            printf("OCIHandleFree failed\n");
        }
        ctxptr->stmthp = NULL;
    }

    return err;
}
```

1. 修改示例代码中 uname、passwd、dbname 三个变量为用户名、密码、在 pg_service.conf 中配置的数据源名称（username,password,db_name）。
2. 运行：

```
cd $PG_OCI_PATH
./run.sh
```

结果：

```
SUCCESS: OCIInitialize()初始化OCI应用环境
SUCCESS: OCIEnvInit()初始化环境句柄
SUCCESS: OCIHandleAlloc()分配句柄
SUCCESS: OCIAttrSet()设置环境句柄属性
```

```
SUCCESS: OCIAttrSet()设置会话句柄用户名
SUCCESS: OCIAttrSet()设置会话句柄密码
SUCCESS: 创建并开始一个用户会话
SUCCESS: OCIDefineByPos ()定义输出变量
SUCCESS: OCIStmtFetch()获取数据
0 11 AAAAAAAAAAAAAA BBBBBBBBB CCCCCCCCCCCCCC
0 12 AAAAAAAAAAAAAA BBBBBBBBB CCCCCCCCCCCCCC
0 13 AAAAAAAAAAAAAA BBBBBBBBB CCCCCCCCCCCCCC
0 14 AAAAAAAAAAAAAA BBBBBBBBB CCCCCCCCCCCCCC
0 15 AAAAAAAAAAAAAA BBBBBBBBB CCCCCCCCCCCCCC
SUCCESS: 结束释放资源，断开数据库连接，释放句柄
```

使用 Pro*C 连接 TDSQL PG

最近更新时间：2025-02-28 10:59:33

本文介绍如何使用 Pro * C 示例程序连接及操作数据库。

获取数据库连接信息

联系数据库管理员获取数据库连接命令：

```
psql -h 10.211.55.30 -p 11345 -U tbase -d tdsql
```

参数说明：

- `-h` 数据库 IP，通常是 CN 的 IP。
- `-p` 数据库端口。
- `-U` 数据库用户。
- `-d` 需要访问的数据库名称。

安装驱动依赖

1. 下载驱动：请单击 TDSQL-PG Pro/C 驱动包下载 。
2. 安装

   ```
   rpm -ivh tdsql_proc-1.1.0-1.x86_64.rpm
   ```

   软件包会默认安装在 /usr/local/tdsql_proc_pg 目录下，目录中有以下文件：
   - 预编译程序 `/usr/local/tdsql_proc_pg/bin/tdsql_proc`
   - 头文件 `/usr/local/tdsql_proc_pg/include`
   - 运行库 `/usr/local/tdsql_proc_pg/lib/libtdsqlproc.so` 包含依赖库 libssl.so.10和 libcrypto.so.10
   - 连接配置模板 `/usr/local/tdsql_proc_pg/tdsql_proc.cnf`
3. 设置环境变量，在操作用户目录添加环境变量，编辑 ~/.bashrc。

   ```
   export PATH=/usr/local/tdsql_proc_pg/bin:$PATH
   export LD_LIBRARY_PATH=/usr/local/tdsql_proc_pg/lib:$LD_LIBRARY_PATH
   ```
4. 环境变量生效

```
source ~/.bashrc
```

5. 连接配置文件

   /usr/local/tdsql_proc_pg/tdsql_proc.cnf 文件中保存了数据库连接信息，输入如下内容：

```
[db_test]
host=10.211.55.30
port=11345
db=tdsql
```

   - `db_test` CONNECT 命令中指定的服务名
   - `host` tdsql 数据库的 ip 地址
   - `port` tdsql 数据库的端口号
   - `db` tdsql 数据库的数据库名

编写 Pro/C 测试程序

示例程序：`demo.pc`

```
/*---------------------------------------------------------------------
 * demo.pc is an example Pro*C program
*---------------------------------------------------------------------
*/

#include <stdio.h>
#include <sqlca.h>
#include <string.h>
#include <setjmp.h>
#include <stdlib.h>
#include <sqlcpr.h>
#include <sqlca.h>

#define MAX_VAR_LEN 255
#define MAX_NAME_LEN 31

void sqlerror();

/* 变量声明 */
EXEC SQL BEGIN DECLARE SECTION;
char *username = "tbase";
char *password = "1234abcd";
```

```
char *service = "db_test";
EXEC SQL END DECLARE SECTION;
EXEC SQL WHENEVER SQLERROR DO sqlerror();

/* 宿主变量类型 */

void variable_test()
{
    EXEC SQL BEGIN DECLARE SECTION;
    int a;
    unsigned int b;
    char c = 'a';
    unsigned char d = 'b';
    char *e = "hello";
    char f[20] = "world";
    long g = 3;
    unsigned long h = 4;
    float i = 3.14;
    double j = 5.4321;
    varchar k[100];
    EXEC SQL END DECLARE SECTION;
}

/* 创建表 */
void create_table_demo_test()
{
    EXEC SQL BEGIN DECLARE SECTION;
    char *sql1 = "DROP TABLE IF EXISTS demo_test";
    char *sql2 = "CREATE TABLE demo_test (id int, name varchar(100),
address varchar(100))";
    EXEC SQL END DECLARE SECTION;

    printf("***********create_table_demo_test start***********\n");
    EXEC SQL PREPARE stat FROM :sql1;
    EXEC SQL EXECUTE stat;
    EXEC SQL PREPARE stat FROM :sql2;
    EXEC SQL EXECUTE stat;
    EXEC SQL COMMIT;
    printf("***********create_table_demo_test success***********\n");
    printf ("\n");
}

/* SELECT */
```

```
void select_table_demo_test()
{
    EXEC SQL BEGIN DECLARE SECTION;
    long id;
    int test_id = 2;
    varchar name[100];
    char address[100];
    short ind;
    EXEC SQL END DECLARE SECTION;

    printf("***********select_table_demo_test start**********\n");
    EXEC SQL SELECT id,name,address INTO :id, :name, :address:ind from
demo_test where id = :test_id;
    name.arr[name.len] = '\0';
    printf("id:%d, name:%s, address:%s\n", id, name.arr, ind != -1 ?
address : "NULL");
    printf("***********select_table_demo_test success**********\n");
    printf ("\n");
}

/* INSERT */
void insert_table_demo_test()
{
    EXEC SQL BEGIN DECLARE SECTION;
    int test_id = 2;
    char name[100] = "FirstMan";
    char address[100] = "EARTH";
    EXEC SQL END DECLARE SECTION;

    printf("***********insert_table_demo_test start**********\n");
    EXEC SQL INSERT INTO demo_test values(:test_id, :name, :address);
    EXEC SQL COMMIT;
    printf("insert demo_test values(%d,%s,%s)\n", test_id, name,
address);
    printf("***********insert_table_demo_test success**********\n");
    printf ("\n");
}

/* UPDATE */
void update_table_demo_test()
{
    EXEC SQL BEGIN DECLARE SECTION;
    int test_id = 2;
```

```
    char address[100] = "AAAAA";
    short ind = -1;
    EXEC SQL END DECLARE SECTION;
    printf("***********update_table_demo_test start**********\n");
    EXEC SQL UPDATE demo_test SET address = :address where id =
:test_id;
    EXEC SQL COMMIT;
    printf("update demo_test address to %s\n",address);
    printf("***********update_table_demo_test success**********\n");
    printf ("\n");
}

/* DELETE */
void delete_table_demo_test()
{
    EXEC SQL BEGIN DECLARE SECTION;
    int test_id = 2;
    EXEC SQL END DECLARE SECTION;
    printf("***********delete_table_demo_test start**********\n");
    EXEC SQL DELETE FROM demo_test where id = :test_id;
    EXEC SQL COMMIT;
    printf("***********delete_table_demo_test success**********\n");
    printf ("\n");
}

/* COMMIT 已经包含在其他用例，ROLLBACK */
void rollback_demo_test()
{
    EXEC SQL BEGIN DECLARE SECTION;
    int test_id = 2;
    char name[100] = "ROLLBACK";
    char address[100] = "ROLLBACK";
    EXEC SQL END DECLARE SECTION;
    printf("***********rollback_demo_test start**********\n");
    printf("insert demo_test values(%d, %s, %s) rollback\n",test_id,
name, address);
    EXEC SQL INSERT INTO demo_test values(:test_id, :name, :address);
    EXEC SQL ROLLBACK;
    printf("***********rollback_demo_test success**********\n");
    printf ("\n");
}

/* 使用游标，将结果集保存在变量中 */
```

```
void select_table_with_cursor_demo_test()
{
    EXEC SQL BEGIN DECLARE SECTION;
    int id;
    varchar name[50];
    char address[50];
    EXEC SQL END DECLARE SECTION;

printf("**********select_table_with_cursor_demo_test
start**********\n");
    EXEC SQL DECLARE cur CURSOR for select id,name,address from
demo_test;
    EXEC SQL OPEN cur;
    EXEC SQL FETCH cur INTO :id, :name, :address;
    name.arr[name.len] = '\0';
    printf("id:%d, name:%s, address:%s\n", id, name.arr, address);
    EXEC SQL CLOSE cur;
printf("**********select_table_with_cursor_demo_test
success**********\n");
    printf ("\n");
}

/* 简单动态 SQL */
void simple_dynamic_SQL_test()
{
    EXEC SQL BEGIN DECLARE SECTION;
    char * sql = "insert into simple_dynamic_table(c1,c2) values (1,
2)";
    char *sql1 = "DROP TABLE IF EXISTS simple_dynamic_table";
    char *sql2 = "CREATE TABLE simple_dynamic_table (c1 int,c2 int)";
    EXEC SQL END DECLARE SECTION;

    EXEC SQL PREPARE stat FROM :sql1;
    EXEC SQL EXECUTE stat;
    EXEC SQL PREPARE stat FROM :sql2;
    EXEC SQL EXECUTE stat;

    printf("**********simple_dynamic_SQL_test start**********\n");
    printf("execute sql is %s\n", sql);
    EXEC SQL EXECUTE IMMEDIATE :sql;
    EXEC SQL COMMIT;
    printf("**********simple_dynamic_SQL_test success**********\n");
    printf ("\n");
```

```
}

/* ANSI 动态 SQL，使用游标将结果集保存在 DESCRIPTOR 中 */
void ANSI_dynamic_SQL_test()
{
    int i;
    EXEC SQL BEGIN DECLARE SECTION;
    char *sql1 = "DROP TABLE IF EXISTS ANSI_dynamic_table";
    char *sql2 = "CREATE TABLE ANSI_dynamic_table (c1 int, c2 int, c3
int)";
    int c1val = 1;
    int type = 3;
    int len = sizeof(int);
    int input_count, output_count, occurs;
    char name[31];
    int out_data;
    char out_name[MAX_NAME_LEN] = {0};
    short indi;
    char dyn_statement[1024] = "select c2,c3 from ANSI_dynamic_table
where c1=:c1val";
    EXEC SQL END DECLARE SECTION;

    EXEC SQL PREPARE stat FROM :sql1;
    EXEC SQL EXECUTE stat;
    EXEC SQL PREPARE stat FROM :sql2;
    EXEC SQL EXECUTE stat;

    printf("***********ANSI_dynamic_SQL_test start***********\n");
    EXEC SQL ALLOCATE DESCRIPTOR 'in_desc';
    EXEC SQL ALLOCATE DESCRIPTOR 'out_desc';
    EXEC SQL INSERT INTO ANSI_dynamic_table VALUES ( 1, 2, 3);
    EXEC SQL INSERT INTO ANSI_dynamic_table VALUES ( 1, 2, 2);
    EXEC SQL INSERT INTO ANSI_dynamic_table VALUES ( 2, 1, 1);
    EXEC SQL INSERT INTO ANSI_dynamic_table(c1) VALUES (1);
    EXEC SQL COMMIT;

    /* 通过 DESCRIPTOR 给 SQL 传入变量（c1val） */
    EXEC SQL PREPARE S from :dyn_statement;
    EXEC SQL DECLARE C CURSOR for S;
    EXEC SQL DESCRIBE INPUT S USING SQL DESCRIPTOR 'in_desc';
    EXEC SQL GET DESCRIPTOR 'in_desc' :input_count = COUNT;
    for (i=0; i < input_count; i++)
    {
```

```
        occurs = i +1;     /* occurence is 1 based */
        EXEC SQL SET DESCRIPTOR 'in_desc' VALUE :occurs TYPE = :type,
LENGTH = :len, DATA = :c1val;
    }
    EXEC SQL OPEN C USING DESCRIPTOR 'in_desc';

    /* 通过 DESCRIPTOR 设定 SQL 输出变量属性 */
    EXEC SQL DESCRIBE OUTPUT S USING DESCRIPTOR 'out_desc';
    EXEC SQL GET DESCRIPTOR 'out_desc' :output_count = COUNT;
    for (i = 0; i < output_count; i++)
    {
        occurs = i + 1;
        EXEC SQL GET DESCRIPTOR 'out_desc' VALUE :occurs
        :out_name = NAME;
        EXEC SQL SET DESCRIPTOR 'out_desc' VALUE :occurs
        TYPE = :type, LENGTH = :len;
        printf("%-*.*s", 9,9, out_name);
    }
    printf("\n");
    /* 通过 DESCRIPTOR 获取 SQL 输出变量值 */
    EXEC SQL WHENEVER NOT FOUND GOTO end_select_loop;

    for (;;)
    {
       EXEC SQL FETCH C INTO DESCRIPTOR 'out_desc';
       for (i=0; i < output_count; i++)
    {
        occurs = i + 1;
        EXEC SQL GET DESCRIPTOR 'out_desc' VALUE :occurs
        :out_data = DATA, :indi = INDICATOR;
        if (indi == -1)
            printf("%-*.*s", 9,9, "NULL");
        else
            printf("%-*d", 9, out_data);
    }
    printf ("\n");
    }
end_select_loop:
    EXEC SQL CLOSE C;
    EXEC SQL DEALLOCATE DESCRIPTOR 'in_desc';
    EXEC SQL DEALLOCATE DESCRIPTOR 'out_desc';
    printf("**********ANSI_dynamic_SQL_test success**********\n");
    printf ("\n");
```

```
    return;
}

void sqlerror() {
    printf("Stop Error:\t%25i, %s\n", sqlca.sqlcode,
sqlca.sqlerrm.sqlerrmc);
    EXEC SQL WHENEVER SQLERROR CONTINUE;
    EXEC SQL ROLLBACK WORK RELEASE;
    exit(-1);
}

void main() {
    /* 建立连接 */
    EXEC SQL CONNECT :username identified by :password using :service;

    printf("connect success\n");
    printf ("\n");
    /* 测试 */
    variable_test();
    create_table_demo_test();
    insert_table_demo_test();
    select_table_demo_test();
    update_table_demo_test();
    select_table_demo_test();
    rollback_demo_test();
    select_table_with_cursor_demo_test();
    delete_table_demo_test();
    simple_dynamic_SQL_test();
    ANSI_dynamic_SQL_test();
    return;
}
```

1. 修改示例代码中 username、password、service 三个变量为用户名、密码、在 tdsql_proc.cnf 配置的数据源名称。
2. 预编译

```
/usr/local/tdsql_proc_pg/bin/tdsql_proc iname=demo.pc oname=demo.c
conn_config=/usr/local/tdsql_proc_pg/tdsql_proc.cnf
```

3. 编译

```
gcc demo.c -o test -I /usr/local/tdsql_proc_pg/include -L
/usr/local/tdsql_proc_pg/lib -ltdsqlproc
```

4. 执行

```
./test
```

5. 结果

```
connect success

**********create_table_demo_test start**********
**********create_table_demo_test success**********

**********insert_table_demo_test start**********
insert demo_test values(2,FirstMan,EARTH)
**********insert_table_demo_test success**********

**********select_table_demo_test start**********
id:2, name:FirstMan, address:EARTH
**********select_table_demo_test success**********

**********update_table_demo_test start**********
update demo_test address to AAAAA
**********update_table_demo_test success**********

**********select_table_demo_test start**********
id:2, name:FirstMan, address:AAAAA
**********select_table_demo_test success**********

**********rollback_demo_test start**********
insert demo_test values(2, ROLLBACK, ROLLBACK) rollback
**********rollback_demo_test success**********

**********select_table_with_cursor_demo_test start**********
id:2, name:FirstMan, address:AAAAA
**********select_table_with_cursor_demo_test success**********

**********delete_table_demo_test start**********
**********delete_table_demo_test success**********
```

```
**********simple_dynamic_SQL_test start**********
execute sql is insert into simple_dynamic_table(c1,c2) values (1, 2)
**********simple_dynamic_SQL_test success**********

**********ANSI_dynamic_SQL_test start**********
c2        c3
27        37
27        27
NULL      NULL
**********ANSI_dynamic_SQL_test success**********
```