

**腾讯云**

**TBase**产品文档

**【版权声明】**

本文档所涉及的软件著作权、版权和知识产权已依法进行了相关注册、登记，由原厂商及经合法注册并拥有合法所有权的厂商拥有，受《中华人民共和国著作权法》、《计算机软件保护条例》、《知识产权保护条例》和相关国际版权条约、法律、法规以及其它知识产权法律和条约的保护。未经授权许可，不得非法使用。

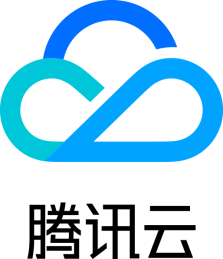
**【免责声明】**

本文档包含的版权信息由原厂商及经合法注册并拥有合法所有权的厂商拥有，受法律的保护，原厂商及经合法注册并拥有合法所有权的厂商对本文档可能涉及到的非原厂商及经合法注册并拥有合法所有权的厂商的信息不承担任何责任。在法律允许的范围内，您可以查阅，并仅能够在《中华人民共和国著作权法》规定的合法范围内复制和打印本文档。任何单位和个人未经原厂商书面授权许可，不得使用、修改、再发布本文档的任何部分和内容，否则将视为侵权，原厂商具有依法追究其责任的权利。

本文档中包含的信息如有更新，恕不另行通知。您对本文档的任何问题，可直接向原厂商告知或查询。

未经原厂商明确授予的任何权利均予保留。

**【商标声明】**



及其它腾讯云服务相关的商标均为腾讯云计算（北京）有限责任公司及其关联公司所有。本文档涉及的第三方主体的商标，依法由权利人所有。

# 数据类型

## 字符

### CHAR

当创建带有CHAR列的表时，您可以将列长度指定为大小（可选），对于类型CHAR来说，如果不指定n的值，n就会默认为1。 如果要存储的字符串长度小于n，那么类型CHAR的值将会不用空格补齐，目前与Oracle略有不同。

如果您要明确地将超出长度的值转换为类型CHAR（N）的长度，那么超出长度的值被插入将报错。

### VARCHAR、VARCHAR2

如果要赋值的字符串长度小于n，那么VARCHAR类型的字符串值将会按照实际的长度进行存储，超出部分不用空格补齐。VARCHAR2最大可存储1G。

### CLOB

可以在CLOB类型中存储长字符串。通常来说当字符串的最大长度不可知的时候，建议使用CLOB类型。TBase系统将CLOB转换成text类型。TBase的CLOB可存储1G长度。

## 数字

### NUMBER

NUMBER类型能够用于存储对于精度位数没有限制的数字，并且可以用于执行精确计算。当要求高精确度时，推荐使用这种类型来存储货币总量和其他类型的数量值。

NUMBER类型数值的精度和范围是可以配置的NUMBER(p [, s ] )。精度p是指整个数值包含的所有数字，范围s是小数点右边部分的小数位数。精度p必须为正数，范围可以是0或正数。将列指定为不带任意精度和范围的NUMBER类型，将不会把输入值强制为转换为任意特定的范围。如果列的数据类型是带有范围的NUMBER类型，那么超过返回将会报错。

例子：

**create table t4(f1 number(6,2));**

**insert into t4 values(1000.02);**

**以上插入正常**

**insert into t4 values(10000.02);**

**以上报错：**

**ERROR: numeric field overflow**

**SQL 状态:22003**

**细节:A field with precision 6, scale 2 must round to an absolute value less than 10^4.**

## 日期

### DATE

如CREATE TABLE或者ALTER TABLE命令中出现的时候，在把数据表定义存储在数据库时，会自动把它转换成timestamp(0) without time zone类型。

## 大对象

### BLOB

系统将BLOB转换成bytea类型，TBase BLOB最大可存储1G。

# 运算符

## 算术运算符

您可以使用算数运算符对TBase V5所支持数据类型的值进行操作。本文介绍TBase V5支持的算数运算符。

下面的表显示了允许使用的算数运算符。

|  |  |  |  |
| --- | --- | --- | --- |
| 运算符 | 描述 | 示例 | 结果 |
| + | 加法 | 4 + 4 | 8 |
| - | 减法 | 5 - 2 | 3 |
| \* | 乘法 | 4 \* 4 | 16 |
| / | 除法 | 16/4 | 4.0000000000000000 |

这里除法与Oracle运算结果精度稍有差异，TBase V5运算结果小数点后为16-20位，Oracle运算结果小数点后最多9位，自动省略末尾0，小于1时自动省略个位数。

示例：

**TBase V5：**

**postgres=# select 16/4 from dual;**

**?column?**

**--------------------**

**4.0000000000000000**

**(1 row)**

**postgres=# select 2/3 from dual;**

**?column?**

**------------------------**

**0.66666666666666666667**

**postgres=# select 4/4 from dual;**

**?column?**

**------------------------**

**1.00000000000000000000**

**(1 row)**

**Oracle：**

**SQL> select 16/4 from dual;**

**16/4**

**----------**

**4**

**SQL> select 2/3 from dual;**

**2/3**

**----------**

**.666666667**

**SQL> select 4/4 from dual;**

**4/4**

**----------**

**1**

**SQL> select 20/8 from dual;**

**20/8**

**----------**

**2.5**

## 连接运算符

TBase V5兼容Oracle的sql连接符”||”，支持字符与字符、字符与数字、数字与数字类型的连接运算。

|  |  |  |
| --- | --- | --- |
| 描述 | 示例 | 结果 |
| 字符与字符 | 'a'||'b' | ab |
| 字符与数字 | 'a'||1 | a1 |
| 数字与数字 | 1||2 | 12 |
| 1||2.1 | 12.1 |
| 1.1||2.1 | 1.12.1 |

示例：

**postgres=# select 'a'||'b' from dual;**

**?column?**

**----------**

**ab**

**(1 row)**

**postgres=# select 'a'||1 from dual;**

**?column?**

**----------**

**a1**

**(1 row)**

**postgres=# select 1||2 from dual;**

**?column?**

**----------**

**12**

**(1 row)**

**postgres=# select 1||2.1 from dual;**

**?column?**

**----------**

**12.1**

**(1 row)**

**postgres=# select 1.1||2.1 from dual;**

**?column?**

**----------**

**1.12.1**

**(1 row)**

## 集合运算符

TBase V5兼容Oracle的集合运算符UNION、UNION ALL、INTERSECT、MINUS。

|  |  |
| --- | --- |
| 运算符 | 描述 |
| UNION | 多个sql结果求并集，结果集去重 |
| UNION ALL | 多个sql结果求并集，结果集不去重 |
| INTERSECT | 多个sql结果求交集，结果集去重 |
| MINUS | 多个sql结果求差集，结果集去重 |

语法：

**SELECT STATEMENT**

**UNION | UNION ALL | INTERSECT | MINUS**

**SELECT STATEMENT**

**[**

**UNION | UNION ALL | INTERSECT | MINUS**

**SELECT STATEMENT**

**...**

**]**

**[ ORDER BY [ number [ ASC | DESC ] , ... ]**

示例：

**create table student**

**(**

**id int primary key,**

**name nvarchar2(50) not null,**

**score number not null**

**);**

**insert into student values(1,'Aaron',78);**

**insert into student values(2,'Bill',76);**

**insert into student values(3,'Cindy',89);**

**insert into student values(4,'Damon',90);**

**insert into student values(5,'Ella',73);**

**insert into student values(6,'Frado',61);**

**insert into student values(7,'Gill',99);**

**insert into student values(8,'Hellen',56);**

**insert into student values(9,'Ivan',93);**

**insert into student values(10,'Jay',90);**

UNION示例：

**postgres=# select \* from student where id < 4 union select \* from student where id > 2 and id < 6;**

**id | name | score**

**----+-------+-------**

**5 | Ella | 73**

**3 | Cindy | 89**

**1 | Aaron | 78**

**2 | Bill | 76**

**4 | Damon | 90**

**(5 rows)**

UNION ALL示例：

**postgres=# select \* from student where id < 4 union all select \* from student where id > 2 and id < 6;**

**id | name | score**

**----+-------+-------**

**1 | Aaron | 78**

**2 | Bill | 76**

**5 | Ella | 73**

**3 | Cindy | 89**

**3 | Cindy | 89**

**4 | Damon | 90**

**(6 rows)**

INTERSECT示例:

**postgres=# select \* from student where id < 4 intersect select \* from student where id > 2 and id < 6;**

**id | name | score**

**----+-------+-------**

**3 | Cindy | 89**

**(1 row)**

MINUS示例：

**postgres=# select \* from student where id < 4 minus select \* from student where id > 2 and id < 6;**

**id | name | score**

**----+-------+-------**

**1 | Aaron | 78**

**2 | Bill | 76**

**(2 rows)**

## 其它运算符

### (+)

TBase V5兼容Oracle的sql（左/右）外连接”(+)”写法，等同于left | right join。

使用限制

关于使用(+)的一些注意事项：

1. (+)操作符只能出现在where子句中，且不能与outer join语法同时使用。
2. 使用(+)操作符进行关联操作时，如果where子句中包含多个条件，则关联的两张表的其它所有条件上关联表的字段上必须都包含(+)操作符。
3. (+)操作符只能用于列，不能用于表达式。
4. (+)操作符不能与or和in操作符一起使用。

语法

左外连接（left join）：

**... WHERE table1.column1 =table2.column1(+) [ AND table1.column2 =table2.column2(+) ...]**

右外连接（right join）：

**... WHERE table1.column1(+) =table2.column1[ AND table1.column2(+) =table2.column2 ...]**

示例

**create table a(a1 number(12),a2 varchar2(50));**

**create table b(b1 number(12),b2 varchar2(50));**

**insert into a values (1,'a'),(2,'ba'),(3,'ca'),(4,'da');**

**insert into b values (1,'a'),(2,'bba'),(3,'ca'),(5,'dda'),(6,'eda');**

左外连接（left join）:

**postgres=# select a.a1,b.b1 from a,b where a.a1=b.b1(+) ;**

**a1 | b1**

**----+----**

**1 | 1**

**2 | 2**

**4 |**

**3 | 3**

**(4 rows)**

右外连接（right join）：

**postgres=# select a.a1,b.b1 from a,b where a.a1(+)=b.b1;**

**a1 | b1**

**----+----**

**1 | 1**

**3 | 3**

**2 | 2**

**| 5**

**| 6**

**(5 rows)**

### 空值与NULL

空值：即空字符串，两个单引号，中间无空格等任何内容。

NULL：SQL关键字，代表没有。

Oracle中的空值被当做NULL处理 ，而PostgreSQL/TBase V2版本中空值与NULL不同，TBase V5兼容版则保持与Oracle一致。

下表是空值与NULL在不同数据库中的对比情况。

|  |  |  |  |
| --- | --- | --- | --- |
| 描述 | Oracle | PostgreSQL/TBase V2 | TBase V5 |
| 空值与NULL是否一致 | 一致 | 不一致 | 一致 |
| NULL与数值运算 | NULL | NULL | NULL |
| 空值与数值运算 | NULL | 报错 | NULL |
| 空值排序 | 空值最大 | 空值最大 | 空值最大 |
| NULL排序 | NULL值最大 | NULL值最小 | NULL值最大 |

使用限制

1. 空值、NULL与任何数值运算，结果为NULL。
2. SQL中空值、NULL不能用=’’、!=’’来判断，应使用is null、is not null。
3. 对于有空值、NULL的情况，需要注意排序时的场景，如有需要可以用nvl转换为

示例

**TBase V5**

**postgres=# select nvl2('','1','2') from dual ;**

**nvl2**

**------**

**2**

**(1 row)**

**postgres=# select 1 from dual where '' is null ;**

**?column?**

**----------**

**1**

**(1 row)**

**postgres=# select null||'tbase' from dual;**

**?column?**

**----------**

**tbase**

**(1 row)**

**postgres=# select null+1 from dual;**

**?column?**

**----------**

**(1 row)**

**postgres=# drop table t\_null;**

**DROP TABLE**

**postgres=# create table t\_null(f1 int,f2 varchar2(10));**

**CREATE TABLE**

**postgres=# insert into t\_null values(1,null);**

**INSERT 0 1**

**postgres=# insert into t\_null values(2,'');**

**INSERT 0 1**

**postgres=# select \* from t\_null where f2 is null;**

**f1 | f2**

**----+----**

**1 |**

**2 |**

**(2 rows)**

**postgres=# select \* from t\_null where f2 ='' ;**

**f1 | f2**

**----+----**

**(0 rows)**

**postgres=# insert into t\_null values(3,'1');**

**INSERT 0 1**

**postgres=# insert into t\_null values(4,'2');**

**INSERT 0 1**

**postgres=# select \* from t\_null order by f2;**

**f1 | f2**

**----+----**

**3 | 1**

**4 | 2**

**1 |**

**2 |**

**(4 rows)**

# 数据库对象

## 序列

序列是数据库对象一种。多个用户能够通过序列生成连续的数字以此来实现主键字段的自己主动、唯一增长，而且一个序列可为多列、多表同一时候使用。

序列消除了串行化而且提高了应用程序一致性。

使用限制：

1. TBase序列当前为bigint类型，最大值为9223372036854775807，超过该值报错。
2. 兼容Oracle用法，TBase V5支持只读事务时获取序列下一个值。
3. Oracle中NOCYCLE为不循环生成，TBase V5中为NO CYCLE，写法稍有差异。
4. 暂不支持Oracle中的参数NOCACHE、ORDER、NOORDER。
5. 默认不循环生成，达到最大值时将报错；如需循环生成创建时可加参数CYCLE。

语法：

创建序列：

**CREATE [ TEMPORARY | TEMP ] SEQUENCE [ IF NOT EXISTS ] name**

**[ AS data\_type ]**

**[ INCREMENT [ BY ] increment ]**

**[ MINVALUE minvalue | NO MINVALUE ] [ MAXVALUE maxvalue | NO MAXVALUE ]**

**[ START [ WITH ] start ] [ CACHE cache ] [ [ NO ] CYCLE ]**

**[ OWNED BY { table\_name.column\_name | NONE } ]**

修改序列：

**ALTER SEQUENCE [ IF EXISTS ] name**

**[ AS data\_type ]**

**[ INCREMENT [ BY ] increment ]**

**[ MINVALUE minvalue | NO MINVALUE ] [ MAXVALUE maxvalue | NO MAXVALUE ]**

**[ START [ WITH ] start ]**

**[ RESTART [ [ WITH ] restart ] ]**

**[ CACHE cache ] [ [ NO ] CYCLE ]**

**[ OWNED BY { table\_name.column\_name | NONE } ]**

**ALTER SEQUENCE [ IF EXISTS ] name OWNER TO { new\_owner | CURRENT\_USER | SESSION\_USER }**

**ALTER SEQUENCE [ IF EXISTS ] name RENAME TO new\_name**

**ALTER SEQUENCE [ IF EXISTS ] name SET SCHEMA new\_schema**

删除序列：

**DROP SEQUENCE [ IF EXISTS ] name [, ...] [ CASCADE | RESTRICT ]**

序列函数：

获取当前值：

**select sequence\_name.currval [from dual];**

或

**select currval('sequence\_name') [from dual];**

获取下一个值：

**select nextval('sequence\_name') [from dual];**

或

**select sequence\_name.nextval [from dual];**

设置当前值：

**select setval('sequence\_name', value) [from dual];**

示例：

**postgres=# create sequence tbase\_seq;**

**CREATE SEQUENCE**

**postgres=# select tbase\_seq.currval;**

**currval**

**---------**

**1**

**(1 row)**

**postgres=# select tbase\_seq.nextval;**

**nextval**

**---------**

**2**

**(1 row)**

**postgres=# select tbase\_seq.currval;**

**currval**

**---------**

**2**

**(1 row)**

**postgres=# select setval('tbase\_seq', 1000000000);**

**setval**

**------------**

**1000000000**

**(1 row)**

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

**currval**

**------------**

**1000000000**

**(1 row)**

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

**nextval**

**------------**

**1000000001**

**(1 row)**

**postgres=# select setval('tbase\_seq', 9223372036854775807);**

**setval**

**---------------------**

**9223372036854775807**

**(1 row)**

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

**ERROR: nextval: reached maximum value of sequence "tbase\_seq" (9223372036854775806)**

**postgres=# drop sequence tbase\_seq;**

**DROP SEQUENCE**

**postgres=# create sequence tbase\_seq2 start with 9223372036854775806 CYCLE;**

**CREATE SEQUENCE**

**postgres=# select nextval('tbase\_seq2') from dual;**

**nextval**

**---------------------**

**9223372036854775806**

**(1 row)**

**postgres=# select nextval('tbase\_seq2') from dual;**

**nextval**

**---------**

**1**

**(1 row)**

## 同义词

同义词是Oracle中数据库对象的一种别名，支持的对象有：表、视图、序列、运算符、函数、存储过程、包、物化视图、java class模式对象、用户自定义对象类型以及其他同义词。

同义词分为公共同义词、私有同义词。

使用限制

1. 暂不支持同义词修改，如需修改，可先删除、再创建。
2. 暂需要超级用户创建、删除公有同义词，不支持普通用户操作。
3. 同义词名和被代表对象名不能重复。
4. 同义词不能循环解释。
5. 对象不存在时，仍能创建同义词，所以创建同义词后，须验证同义词是否可用。

语法：

创建同义词语法：

**CREATE [ OR REPLACE ] [ PUBLIC ] SYNONYM [ schema. ] synonym FOR [ schema. ] object [ @ dblink ] ;**

说明：

1） 权限

a) 创建私有同义词，需要超户或者具有对应模式的CREATE权限。（Oracle：需要CREATE ANY SYNONYM/CREATE SYNONYM权限才可以创建同义词，在TBase里将同义词看成普通对象，简化创建同义词的权限管理）。

b) 创建公共同义词，需要超级用户才可以创建同义词（Oracle需要CREATE PUBLIC SYNONYM）

2） OR REPLACE – 不存在即创建同义词；存在则替换定义

4） PUBLIC – 创建公共同义词，所有用户都能够使用这种同义词，但是需要有被代表对象的访问权限。

\* 当使用没有模式名限定的同义词时，并且后面没有@dblink，系统会将其按照公共同义词定义进行解析。

\* 当公共同义词和当前模式下的“表”同名时，优先使用“表”。

\* 创建与模式同名同义词，导致PLSQL失效。Oracle行为，TBase不支持。

不指定PUBLIC关键字，语句创建私有同义词。

\* 私有同义词在定义模式下名称唯一，并且也不能与其他对象（同义词支持的表、视图、函数等）重名。

\* 其他用户在私有同义词对应的对象如果有权限，那么是允许使用这些私有同义词的。

5） schema – 指定同义词存放的模式名。TBase行为：会按照search\_path值创建对象；如果search\_path中只有PUBLIC合法，则会创建公共同义词。（Oracle：如果不指定，存放到当前用户的模式下）

\* 指定PUBLIC关键字，就不能指定schema

6） synonym – 同义词名

8） FOR 子句 – 指定被代表的数据库对象，对象不能是包里的。支持如下对象：

\* 表、对象表（TBase: typed table， create table…of…）

\* 视图，（对象视图Oracle特性，Tbase不支持）

\* 序列

\* 存储过程、函数、包

\* 物化视图

\* Java class schema object（Java存储过程）

\* 用户自定义类型

\* 同义词

schema – 对象所在的模式，不指定Tbase使用search\_path来查找对象（Oracle如果不指定，默认为当前用户模式）。Oracle要求对远程函数存储过程定义同义词，必须指定模式，TBase这里不做这个限制。

dblink – 指定远程对象的dblink。

9） 其他特殊情况

\* 同义词名和被代表对象名不能重复

\* 同义词不能循环解释

\* 同义词代表的对象可以事先不存在

删除同义词语法：

**DROP [PUBLIC] SYNONYM [ schema. ] synonym;**

说明：

1） 如果模式属主是当前用户，则可以删除这个模式下的同义词，或者可以删除其他模式下，同义词属主与当前用户相同的那些同义词对象（Oracle行为：需要有DROP ANY SYNONYM权限。如果删除公共同义词，需要有DROP PUBLIC SYNONYM权限。）

2） PUBLIC – 指定PUBLIC可以删除公共同义词。PUBLIC和schema不能同时存在。

3） schema – 同义词所在的schema，如果不指定，在当前search\_path下查找同义词。（Oracle：删除当前用户模式下的同义词）

删除物化视图里面的表所对应的同义词，不会使物化视图失效。（Oracle：如果删除的同义词被物化视图引用，删除同义词会标识物化视图失效）

4） FORCE – TBase不支持，TBase会在Transform阶段将同义词替换为实际对象，后续不会再依赖同义词。（Oracle：如果物化视图被其他对象引用，会将对象标识为UNUSED）。

修改同义词语法：

暂不支持修改同义词。

示例

**create table company (**

**id int ,**

**name varchar (60),**

**age int ,**

**address varchar (30),**

**salary int**

**);**

**insert into company values(1,'Paul',32,'California',20000);**

**insert into company values(2,'Allen',25, 'Texas',15000);**

**insert into company values(3,'Teddy',23,'Norway',20000);**

**postgres=# create synonym s\_company for company;**

**NOTICE: create synonym as PUBLIC**

**postgres=# select \* from s\_company;**

**id | name | age | address | salary**

**----+-------+-----+------------+--------**

**1 | Paul | 32 | California | 20000**

**2 | Allen | 25 | Texas | 15000**

**3 | Teddy | 23 | Norway | 20000**

**(3 rows)**

**postgres=# select \* from pg\_synonym ;**

**synname | synnamespace | objnamespace | objname | objdblink | synowner**

**-----------+--------------+--------------+---------+-----------+----------**

**s\_company | 2200 | | company | | 10**

**(1 row)**

**postgres=# drop synonym s\_company;**

**DROP SYNONYM**

## 伪列

### ROWNUM

ROWNUM是Oracle中对结果集加的一个伪列，即先查到结果集之后再加上去的一个列 (强调：先要有结果集)。简单的说 ROWNUM是对符合条件结果的序列号。它总是从1开始排起的。

TBase V5支持ROWNUM伪列用法。

使用限制

使用ROWNUM时需要注意事项：

1. ROWNUM是数据库从数据文件或缓冲区中读取数据的顺序，所以rownum>n（n>=1）、rownum between n(n>1) and m是得不到数据的。
2. 不能使用ROWNUM做为字段名称。

语法

**SELECT [ROWNUM,]column1,... FROM table1... WHERE ROWNUM [< | <= ] n [ and ...];**

示例

**postgres=# create table t1(f1 int,f2 varchar2(16));**

**CREATE TABLE**

**postgres=# insert into t1 values (1,'ONE');**

**INSERT 0 1**

**postgres=# insert into t1 values (2,'TWO');**

**INSERT 0 1**

**postgres=# insert into t1 values (3,'THREE');**

**INSERT 0 1**

**postgres=# insert into t1 values (4,'FOUR');**

**INSERT 0 1**

**postgres=# insert into t1 values (5,'FIVE');**

**INSERT 0 1**

**postgres=# insert into t1 values (6,'SIX');**

**INSERT 0 1**

**postgres=# insert into t1 values (7,'SEVEN');**

**INSERT 0 1**

**postgres=# select \* from t1 order by f1;**

**f1 | f2**

**----+-------**

**1 | ONE**

**2 | TWO**

**3 | THREE**

**4 | FOUR**

**5 | FIVE**

**6 | SIX**

**7 | SEVEN**

**(7 rows)**

**postgres=# select rownum,f1,f2 from t1 where f1>5;**

**rownum | f1 | f2**

**--------+----+-------**

**1 | 6 | SIX**

**2 | 7 | SEVEN**

**(2 rows)**

ROWNUM从1开始。

**postgres=# select rownum,f1,f2 from t1 where rownum<2;**

**rownum | f1 | f2**

**--------+----+-----**

**1 | 1 | ONE**

**(1 row)**

只显示一行数据。

**postgres=# select rownum,f1,f2 from t1 where rownum>=1;**

**rownum | f1 | f2**

**--------+----+-------**

**1 | 1 | ONE**

**2 | 2 | TWO**

**3 | 5 | FIVE**

**4 | 6 | SIX**

**5 | 3 | THREE**

**6 | 4 | FOUR**

**7 | 7 | SEVEN**

**(7 rows)**

显示全部数据。

**postgres=# select rownum,f1,f2 from t1 where rownum between 1 and 2;**

**rownum | f1 | f2**

**--------+----+-----**

**1 | 1 | ONE**

**2 | 2 | TWO**

**(2 rows)**

**postgres=# select rownum,f1,f2 from t1 where rownum between 2 and 3;**

**rownum | f1 | f2**

**--------+----+----**

**(0 rows)**

rownum between 如果大于1，查询结果为空，因为ROWNUM总是从1开始。

### ROWID

Oracle数据库的表中的每一行数据都有一个唯一的标识符，或者称为ROWID，在Oracle内部通常就是使用它来访问数据的。ROWID需要 10个字节的存储空间，并用18个字符来显示。该值表明了该行在Oracle数据库中的物理具体位置。可以在一个查询中使用ROWID来表明查询结果中包含该值。

保存ROWID需要10个字节或者是80个位二进制位。这80个二进制位分别是：

1. 数据对象编号，表明此行所属的数据库对象的编号，每个数据对象在数据库建立的时候都被唯一分配一个编号，并且此编号唯一。数据对象编号占用大约32位。

2. 对应文件编号，表明该行所在文件的编号，表空间的每一个文件标号都是唯一的。文件编号所占用的位置是10位。

3. 块编号，表明改行所在文件的块的位置块编号需要22位。

4. 行编号，表明该行在行目录中的具体位置行编号需要16位。

这样加起来就是80位。

Oracle的物理扩展ROWID有18位，每位采用64位编码，分别用A~Z、a~z、0~9、+、/共64个字符表示。A表示0，B表示1，……Z表示25，a表示26，……z表示51，0表示52，……，9表示61，+表示62，/表示63。

TBase V5兼容Oracle的ROWID，由<xc\_node\_id, sequence\_id>10个字节组成，同样可以保证行更新后，ROWID值不变，但不支持Oracle的DBMS\_ROWID包调用。

使用限制

关于使用ROWID的一些注意事项：

1. 参数default\_with\_rowid可以设置默认建表时是否带ROWID，默认该值为off；可以根据需要修改该参数。
2. default\_with\_rowid=OFF时，也可以通过create table 时加上with (rowids)来让表带ROWID列。
3. ROWID列默认没有索引，所以基于该列的sql可能存在性能问题，可能通过创建索引来提高sql效率，建索引语法与普通字段建索引语法相同。
4. 不支持Oracle的DBMS\_ROWID包调用。
5. cluster table、vacuum full table重写表数据后，ROWID会变，与Oracle一致。

语法

default\_with\_rowid=off，建表时带ROWID语法示例：

**CREATE TABLE [ IF NOT EXISTS ] table\_name**

**( [**

**{ column\_name data\_type [ COLLATE collation ] [ column\_constraint [ ... ] ]**

**| table\_constraint**

**| LIKE source\_table [ like\_option ... ] }**

**[, ... ]**

**] )**

**[ INHERITS ( parent\_table [, ... ] ) ]**

**[ PARTITION BY { RANGE | LIST | HASH } ( { column\_name | ( expression ) } [ COLLATE collation ] [ opclass ] [, ... ] ) ]**

**[ WITH ( storage\_parameter [= value] [, ... ] ) | WITH OIDS | WITHOUT OIDS ]**

**WITH ( ROWIDS ) ;**

示例

**postgres=# create table t\_rowid(f1 int,f2 varchar2(32)) with (rowids);**

**CREATE TABLE**

**postgres=# insert into t\_rowid(f1,f2) values(1,'ONE');**

**INSERT 0 1**

**postgres=# insert into t\_rowid(f1,f2) values(2,'TWO');**

**INSERT 0 1**

**postgres=# select rowid,\* from t\_rowid;**

**rowid | f1 | f2**

**----------------------+----+-----**

**XPK3fw==AQAAAAAAAAA= | 1 | ONE**

**XPK3fw==AgAAAAAAAAA= | 2 | TWO**

**postgres=# select rowid,\* from t\_rowid where rowid='XPK3fw==AgAAAAAAAAA=';**

**rowid | f1 | f2**

**----------------------+----+-----**

**XPK3fw==AgAAAAAAAAA= | 2 | TWO**

**(1 row)**

**postgres=# update t\_rowid set f2='TWO1' where f1=2;**

**UPDATE 1**

**postgres=# select rowid,\* from t\_rowid where rowid='XPK3fw==AgAAAAAAAAA=';**

**rowid | f1 | f2**

**----------------------+----+------**

**XPK3fw==AgAAAAAAAAA= | 2 | TWO1**

**(1 row)**

数据更新，ROWID不变。

**postgres=# create index idx\_t\_rowid\_rowid on t\_rowid(rowid);**

**CREATE INDEX**

**postgres=# explain select rowid,\* from t\_rowid where rowid='XPK3fw==AgAAAAAAAAA=';**

**QUERY PLAN**

**----------------------------------------------------------------------------------------**

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

**Node/s: dn001, dn002**

**-> Index Scan using idx\_t\_rowid\_rowid on t\_rowid (cost=0.13..4.15 rows=1 width=98)**

**Index Cond: (rowid = 'XPK3fw==AgAAAAAAAAA='::rowid)**

**(4 rows)**

**postgres=#**

支持ROWID字段上创建索引。

# 分区表

## 分区表类型

TBase V5版本支持范围（RANGE）、列表（LIST）、散列（HASH）分区，同时支持基于范围（RANGE）、列表（LIST）、散列（HASH）的多级分区。

### 范围（RANGE）分区

使用限制

1. 社区分区表暂不支持在主表上创建主键或唯一索引，可在每个分区子表上创建。自研分区表支持在主表上创建主键或唯一索引。

语法

自研分区表建表语法：

**CREATE TABLE [ IF NOT EXISTS ] table\_name ( [**

**{ column\_name data\_type [ COLLATE collation ] [ column\_constraint [ ... ] ]**

**| table\_constraint**

**| LIKE source\_table [ like\_option ... ] }**

**[, ... ]**

**] )**

**[ PARTITION BY RANGE ( { column\_name | ( expression ) } [ COLLATE collation ] [ opclass ] [, ... ] ) ]**

**BEGIN ( { numeric\_literal | string\_literal | MINVALUE} ) STEP (numeric\_literal) PARTITIONS( numeric\_literal)**

**[ distribute by { replication | shard(column\_name) } ]**

**[ to group { default\_group | group\_name } ] ;**

新增分区子表语法：

**ALTER TABLE table\_name ADD PARTITIONS numeric\_literal ;**

自动生成的分区子表命名规则：

表名\_part\_序号（从0开始）

table\_name\_part\_number

如：分区主表名为tbl\_range\_test，则分区子表为tbl\_range\_test\_part\_0、tbl\_range\_test\_part\_1...

社区分区表建表语法：

分区主表：

**CREATE TABLE [ IF NOT EXISTS ] table\_name ( [**

**{ column\_name data\_type [ COLLATE collation ] [ column\_constraint [ ... ] ]**

**| table\_constraint**

**| LIKE source\_table [ like\_option ... ] }**

**[, ... ]**

**] )**

**[ PARTITION BY RANGE ( { column\_name | ( expression ) } [ COLLATE collation ] [ opclass ] [, ... ] ) ]**

**[ distribute by { replication | shard(column\_name) } ]**

**[ to group { default\_group | group\_name } ] ;**

分区子表：

**CREATE TABLE [ IF NOT EXISTS ] table\_name**

**PARTITION OF parent\_table [ (**

**{ column\_name [ WITH OPTIONS ] [ column\_constraint [ ... ] ]**

**| table\_constraint }**

**[, ... ]**

**) ] { FOR VALUES partition\_bound\_spec | DEFAULT }**

**[ PARTITION BY { RANGE | LIST | HASH } ( { column\_name | ( expression ) } [ COLLATE collation ] [ opclass ] [, ... ] ) ] ;**

**partition\_bound\_spec is:**

**FROM ( { numeric\_literal | string\_literal | MINVALUE | MAXVALUE } [, ...] )**

**TO ( { numeric\_literal | string\_literal | MINVALUE | MAXVALUE } [, ...] )**

示例

整型范围分区：

创建自研分区表：

**postgres=# create table part\_range\_t3 (f1 bigint,f2 timestamp default now(), f3 integer) partition by range (f3) begin (1) step (50) partitions (3) ;**

**CREATE TABLE**

插入分区范围内数据：

**postgres=# insert into part\_range\_t3(f1,f3) values(1,1),(2,50),(3,100),(2,110);**

**COPY 4**

插入超出分区范围数据：

**postgres=# insert into part\_range\_t3(f1,f3) values(1,151);**

**ERROR: node:dn001, backend\_pid:19944, nodename:dn001,backend\_pid:19944,message:value to inserted execeed range of partitioned table**

添加2个分区：

**postgres=# alter table part\_range\_t3 add partitions 2;**

**ALTER TABLE**

再次插入分区范围外数据：

**postgres=# insert into part\_range\_t3(f1,f3) values(1,151);**

**INSERT 0 1**

时间范围分区：

创建自研分区表：

**postgres=# create table part\_range\_t1**

**(**

**f1 int not null,**

**f2 timestamp not null default now(),**

**f3 varchar(50),**

**f4 integer,**

**primary key(f1)**

**)**

**partition by range (f2)**

**begin (timestamp without time zone '2020-06-01 0:0:0')**

**step (interval '1 month') partitions (3) ;**

**CREATE TABLE**

插入分区范围内数据：

**postgres=# insert into part\_range\_t1(f1,f2,f3,f4) values(1,'2020-06-01','one',2);**

**INSERT 0 1**

**postgres=# insert into part\_range\_t1(f1,f2,f3,f4) values(2,'2020-07-01','two',3);**

**INSERT 0 1**

**postgres=# insert into part\_range\_t1(f1,f2,f3,f4) values(3,'2020-08-01','three',4);**

**INSERT 0 1**

**postgres=# insert into part\_range\_t1(f1,f2,f3,f4) values(4,'2020-08-21','four',4);**

**INSERT 0 1**

插入分区范围外数据：

**postgres=# insert into part\_range\_t1(f1,f2,f3,f4) values(5,'2020-09-01','four',4);**

**ERROR: node:dn001, backend\_pid:30099, nodename:dn001,backend\_pid:30099,message:value to inserted execeed range of partitioned table**

**postgres=#**

增加1个分区：

**postgres=# ALTER TABLE part\_range\_t1 ADD PARTITIONS 1;**

**ALTER TABLE**

**postgres=#**

当前分区范围上限为2020-10-01。

再次插入分区范围外数据：

**postgres=# insert into part\_range\_t1(f1,f2,f3,f4) values(5,'2020-09-01','four',4);**

**INSERT 0 1**

**postgres=#**

创建社区分区表：

**postgres=# create table public.part\_range\_t2**

**(**

**f1 int not null,**

**f2 timestamp not null default now(),**

**f3 varchar(50),**

**f4 integer)**

**partition by range (f2);**

**CREATE TABLE**

创建子分区：

**postgres=# create table part\_range\_t2\_part\_0 partition of public.part\_range\_t2 for values from ('2020-06-01') to ('2020-07-01');**

**CREATE TABLE**

**postgres=# create table part\_range\_t2\_part\_1 partition of part\_range\_t2 for values from ('2020-07-01') to ('2020-08-01');**

**CREATE TABLE**

**postgres=# create table part\_range\_t2\_part\_2 partition of part\_range\_t2 for values from ('2020-08-01') to ('2020-09-01');**

**CREATE TABLE**

插入分区范围内数据：

**postgres=# insert into part\_range\_t2(f1,f2,f3,f4) values(1,'2020-06-01','one',2);**

**INSERT 0 1**

**postgres=# insert into part\_range\_t2(f1,f2,f3,f4) values(2,'2020-07-01','two',3);**

**INSERT 0 1**

**postgres=# insert into part\_range\_t2(f1,f2,f3,f4) values(3,'2020-08-01','three',4);**

**INSERT 0 1**

**postgres=# insert into part\_range\_t2(f1,f2,f3,f4) values(4,'2020-08-21','four',4);**

**INSERT 0 1**

### 列表（LIST）分区

使用限制

1、社区分区表暂不支持在主表上创建主键或唯一索引，可在每个分区子表上创建。

语法

分区主表建表语法：

**CREATE TABLE [ IF NOT EXISTS ] table\_name ( [**

**{ column\_name data\_type [ COLLATE collation ] [ column\_constraint [ ... ] ]**

**| table\_constraint**

**| LIKE source\_table [ like\_option ... ] }**

**[, ... ]**

**] )**

**[ PARTITION BY LIST ( { column\_name | ( expression ) } [ COLLATE collation ] [ opclass ] [, ... ] ) ]**

**[ distribute by { replication | shard(column\_name) } ]**

**[ to group { default\_group | group\_name } ] ;**

分区子表建表语法：

**CREATE TABLE [ IF NOT EXISTS ] table\_name**

**PARTITION OF parent\_table [ (**

**{ column\_name [ WITH OPTIONS ] [ column\_constraint [ ... ] ]**

**| table\_constraint }**

**[, ... ]**

**) ] FOR VALUES IN ( { numeric\_literal | string\_literal | NULL } )**

**[ PARTITION BY { RANGE | LIST | HASH } ( { column\_name | ( expression ) } [ COLLATE collation ] [ opclass ] [, ... ] ) ] ;**

示例

**postgres=# create table part\_list\_t1(f1 serial not null,f2 text, f3 integer,f4 date) partition by list( f2 );**

**CREATE TABLE**

**postgres=# create table part\_list\_t1\_gd partition of part\_list\_t1(f1 ,f2 , f3,f4) for values in ('广东');**

**CREATE TABLE**

**postgres=# create table part\_list\_t1\_bj partition of part\_list\_t1(f1 ,f2 , f3,f4) for values in ('北京');**

**CREATE TABLE**

**postgres=# create table part\_list\_t1\_sh partition of part\_list\_t1(f1 ,f2 , f3,f4) for values in ('上海');**

**CREATE TABLE**

**postgres=# insert into part\_list\_t1(f1,f2,f3,f4) values(1,'北京',1,now());**

**INSERT 0 1**

**postgres=# insert into part\_list\_t1(f1,f2,f3,f4) values(2,'北京',2,now());**

**INSERT 0 1**

**postgres=# insert into part\_list\_t1(f1,f2,f3,f4) values(3,'上海',3,now());**

**INSERT 0 1**

**postgres=# insert into part\_list\_t1(f1,f2,f3,f4) values(4,'上海',4,now());**

**INSERT 0 1**

**postgres=# insert into part\_list\_t1(f1,f2,f3,f4) values(5,'广东',5,now());**

**INSERT 0 1**

**postgres=# insert into part\_list\_t1(f1,f2,f3,f4) values(6,'广东',6,now());**

**INSERT 0 1**

### 散列（HASH）分区

使用限制

1、社区分区表暂不支持在主表上创建主键或唯一索引，可在每个分区子表上创建。

语法

分区主表建表语法：

**CREATE TABLE [ IF NOT EXISTS ] table\_name ( [**

**{ column\_name data\_type [ COLLATE collation ] [ column\_constraint [ ... ] ]**

**| table\_constraint**

**| LIKE source\_table [ like\_option ... ] }**

**[, ... ]**

**] )**

**[ PARTITION BY HASH ( { column\_name | ( expression ) } [ COLLATE collation ] [ opclass ] [, ... ] ) ]**

**[ distribute by { replication | shard(column\_name) } ]**

**[ to group { default\_group | group\_name } ] ;**

分区子表建表语法：

**CREATE TABLE [ IF NOT EXISTS ] table\_name**

**PARTITION OF parent\_table [ (**

**{ column\_name [ WITH OPTIONS ] [ column\_constraint [ ... ] ]**

**| table\_constraint }**

**[, ... ]**

**) ] FOR VALUES WITH ( MODULUS numeric\_literal, REMAINDER numeric\_literal )**

**[ PARTITION BY { RANGE | LIST | HASH } ( { column\_name | ( expression ) } [ COLLATE collation ] [ opclass ] [, ... ] ) ] ;**

**示例**

**postgres=# create table part\_hash\_t1(id integer, product integer) partition by hash (id);**

**CREATE TABLE**

**postgres=# create table part\_hash\_t1\_part\_0 partition of part\_hash\_t1 for values with (modulus 2,remainder 0);**

**CREATE TABLE**

**postgres=# create table part\_hash\_t1\_part\_1 partition of part\_hash\_t1 for values with (modulus 2,remainder 1);**

**CREATE TABLE**

**postgres=# insert into part\_hash\_t1 select generate\_series(1,1000),generate\_series(1,1000);**

**INSERT 0 1000**

**postgres=# select count(\*) from part\_hash\_t1\_part\_0;**

**count**

**-------**

**535**

**(1 row)**

**postgres=# select count(\*) from part\_hash\_t1\_part\_1;**

**count**

**-------**

**465**

**(1 row)**

### 多级分区

可支持一级分区为RANGE、LIST，二级分区为RANGE、LIST、HASH组合。

语法

一级分区主表建表语法：

**CREATE TABLE [ IF NOT EXISTS ] table\_name ( [**

**{ column\_name data\_type [ COLLATE collation ] [ column\_constraint [ ... ] ]**

**| table\_constraint**

**| LIKE source\_table [ like\_option ... ] }**

**[, ... ]**

**] )**

**[ PARTITION BY { RANGE | LIST } ( { column\_name | ( expression ) } [ COLLATE collation ] [ opclass ] [, ... ] ) ]**

**[ distribute by { replication | shard(column\_name) } ]**

**[ to group { default\_group | group\_name } ] ;**

一级分区子表（即二级分区主表）建表语法：

**CREATE TABLE [ IF NOT EXISTS ] table\_name**

**PARTITION OF parent\_table [ (**

**{ column\_name [ WITH OPTIONS ] [ column\_constraint [ ... ] ]**

**| table\_constraint }**

**[, ... ]**

**) ] FOR VALUES partition\_bound\_spec**

**[ PARTITION BY { RANGE | LIST | HASH } ( { column\_name | ( expression ) } [ COLLATE collation ] [ opclass ] [, ... ] ) ] ;**

**partition\_bound\_spec is:**

**IN ( { numeric\_literal | string\_literal | NULL } [, ...] ) |**

**FROM ( { numeric\_literal | string\_literal | MINVALUE | MAXVALUE } [, ...] )**

**TO ( { numeric\_literal | string\_literal | MINVALUE | MAXVALUE } [, ...] ) |**

**WITH ( MODULUS numeric\_literal, REMAINDER numeric\_literal )**

二级分区子表建表语法同二级分区主表建表语法。

示例

创建LIST-RANGE分区表：

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

**CREATE TABLE**

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

**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);**

**CREATE TABLE**

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

**CREATE TABLE**

**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');**

**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');**

**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');**

**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');**

**CREATE TABLE**

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

**CREATE TABLE**

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

**CREATE TABLE**

**postgres=#**

**postgres=# \d+ t\_native\_mul\_list**

**Table "public.t\_native\_mul\_list"**

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

**--------+--------------------------------+-----------+----------+-----------------------------------------------+----------+--------------+-------------**

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

**f2 | integer | | | | plain | |**

**f3 | text | | | | extended | |**

**f4 | text | | | | extended | |**

**f5 | timestamp(0) without time zone | | | | plain | |**

**Partition key: LIST (f3)**

**Partitions: t\_native\_mul\_list\_bj FOR VALUES IN ('北京'), PARTITIONED,**

**t\_native\_mul\_list\_gd FOR VALUES IN ('广东'), PARTITIONED,**

**t\_native\_mul\_list\_sh FOR VALUES IN ('上海'), PARTITIONED**

**Distribute By: SHARD(f1)**

**Location Nodes: ALL DATANODES**

**postgres=# \d+ t\_native\_mul\_list\_bj**

**Table "public.t\_native\_mul\_list\_bj"**

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

**--------+--------------------------------+-----------+----------+-----------------------------------------------+----------+--------------+-------------**

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

**f2 | integer | | | | plain | |**

**f3 | text | | | | extended | |**

**f4 | text | | | | extended | |**

**f5 | timestamp(0) without time zone | | | | plain | |**

**Partition of: t\_native\_mul\_list FOR VALUES IN ('北京')**

**Partition constraint: ((f3 IS NOT NULL) AND (f3 = ANY (ARRAY['北京'::text])))**

**Partition key: RANGE (f5)**

**Partitions: t\_native\_mul\_list\_bj\_201701 FOR VALUES FROM ('2017-01-01 00:00:00') TO ('2017-02-01 00:00:00'),**

**t\_native\_mul\_list\_bj\_201702 FOR VALUES FROM ('2017-02-01 00:00:00') TO ('2017-03-01 00:00:00')**

**Distribute By: SHARD(f1)**

**Location Nodes: ALL DATANODES**

插入数据：

**postgres=# insert into t\_native\_mul\_list(f2,f3,f4,f5) values(1,'广东',md5(random()::text),'2017-01-01');**

**INSERT 0 1**

**postgres=# insert into t\_native\_mul\_list(f2,f3,f4,f5) values(2,'广东',md5(random()::text),'2017-02-01');**

**INSERT 0 1**

**postgres=# insert into t\_native\_mul\_list(f2,f3,f4,f5) values(1,'北京',md5(random()::text),'2017-01-01');**

**INSERT 0 1**

**postgres=# insert into t\_native\_mul\_list(f2,f3,f4,f5) values(2,'北京',md5(random()::text),'2017-02-01');**

**INSERT 0 1**

**postgres=# insert into t\_native\_mul\_list(f2,f3,f4,f5) values(1,'上海',md5(random()::text),'2017-01-01');**

**INSERT 0 1**

**postgres=# insert into t\_native\_mul\_list(f2,f3,f4,f5) values(2,'上海',md5(random()::text),'2017-02-01');**

**INSERT 0 1**

**postgres=# explain select \* from t\_native\_mul\_list where f3='广东';**

**QUERY PLAN**

**-----------------------------------------------------------------------------------------**

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

**Node/s: dn001, dn002**

**-> Append (cost=0.00..34.75 rows=6 width=84)**

**-> Seq Scan on t\_native\_mul\_list\_gd\_201701 (cost=0.00..17.38 rows=3 width=84)**

**Filter: (f3 = '广东'::text)**

**-> Seq Scan on t\_native\_mul\_list\_gd\_201702 (cost=0.00..17.38 rows=3 width=84)**

**Filter: (f3 = '广东'::text)**

**(7 rows)**

**postgres=# explain select \* from t\_native\_mul\_list where f3='广东' and f5>='2017-02-01'**

**postgres-# ;**

**QUERY PLAN**

**------------------------------------------------------------------------------------------------------------**

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

**Node/s: dn001, dn002**

**-> Append (cost=0.00..18.85 rows=1 width=84)**

**-> Seq Scan on t\_native\_mul\_list\_gd\_201702 (cost=0.00..18.85 rows=1 width=84)**

**Filter: ((f5 >= '2017-02-01 00:00:00'::timestamp without time zone) AND (f3 = '广东'::text))**

**(5 rows)**

## 分区拆分

TBase V5兼容Oracle分区表分区拆分语法，支持RANGE、LIST的一级和多级分区拆分。

分区拆分通常用于将DEFAULT分区拆分到新的分区，或者将一个较大的分区拆分成两个分区。

使用限制

1. 暂不支持自研分区、HASH分区拆分。

语法

范围（RANGE）分区拆分：

**ALTER TABLE table\_name SPLIT PARTITION part\_3 AT ( { numeric\_literal | string\_literal } ) INTO (PARTITION part\_1, PARTITION part\_2);**

列表（LIST）分区拆分：

**ALTER TABLE table\_name SPLIT PARTITION part\_3 VALUES ( { numeric\_literal | string\_literal | NULL } , { numeric\_literal | string\_literal | NULL } ) INTO (PARTITION part\_1, PARTITION part\_2);**

示例

**postgres=# \d+ part\_range\_t2**

**Table "public.part\_range\_t2"**

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

**--------+-----------------------------+-----------+----------+---------+----------+--------------+-------------**

**f1 | integer | | not null | | plain | |**

**f2 | timestamp without time zone | | not null | now() | plain | |**

**f3 | character varying(50) | | | | extended | |**

**f4 | integer | | | | plain | |**

**Partition key: RANGE (f2)**

**Partitions: part\_range\_t2\_part\_0 FOR VALUES FROM ('2020-06-01 00:00:00') TO ('2020-07-01 00:00:00'),**

**part\_range\_t2\_part\_1 FOR VALUES FROM ('2020-07-01 00:00:00') TO ('2020-08-01 00:00:00'),**

**part\_range\_t2\_part\_2 FOR VALUES FROM ('2020-08-01 00:00:00') TO ('2020-09-01 00:00:00')**

**Distribute By: SHARD(f1)**

**Location Nodes: ALL DATANODES**

当前分区表有part\_range\_t2\_part\_0、part\_range\_t2\_part\_1、part\_range\_t2\_part\_2 3个分区。

下面的sql将part\_range\_t2\_part\_2中 ('2020-08-01 00:00:00') 到 ('2020-08-20 00:00:00')的数据拆分到part\_range\_t2\_part\_21。

**postgres=# ALTER TABLE part\_range\_t2 SPLIT PARTITION part\_range\_t2\_part\_2 AT ( '2020-08-20') INTO (PARTITION part\_range\_t2\_part\_21, PARTITION part\_range\_t2\_part\_2);**

**ALTER TABLE**

**postgres=# \d+ part\_range\_t2**

**Table "public.part\_range\_t2"**

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

**--------+-----------------------------+-----------+----------+---------+----------+--------------+-------------**

**f1 | integer | | not null | | plain | |**

**f2 | timestamp without time zone | | not null | now() | plain | |**

**f3 | character varying(50) | | | | extended | |**

**f4 | integer | | | | plain | |**

**Partition key: RANGE (f2)**

**Partitions: part\_range\_t2\_part\_0 FOR VALUES FROM ('2020-06-01 00:00:00') TO ('2020-07-01 00:00:00'),**

**part\_range\_t2\_part\_1 FOR VALUES FROM ('2020-07-01 00:00:00') TO ('2020-08-01 00:00:00'),**

**part\_range\_t2\_part\_2 FOR VALUES FROM ('2020-08-20 00:00:00') TO ('2020-09-01 00:00:00'),**

**part\_range\_t2\_part\_21 FOR VALUES FROM ('2020-08-01 00:00:00') TO ('2020-08-20 00:00:00')**

**Distribute By: SHARD(f1)**

**Location Nodes: ALL DATANODES**

确认分区已拆分。

**postgres=# select \* from part\_range\_t2\_part\_2;**

**f1 | f2 | f3 | f4**

**----+---------------------+------+----**

**4 | 2020-08-21 00:00:00 | four | 4**

**(1 row)**

**postgres=# select \* from part\_range\_t2\_part\_21;**

**f1 | f2 | f3 | f4**

**----+---------------------+-------+----**

**3 | 2020-08-01 00:00:00 | three | 4**

**(1 row)**

确认数据已拆分到新的分区子表中。

## 分区合并

TBase V5兼容Oracle分区表分区合并语法。通过分区合并，可以将两个分区的数据合并到另一个分区中。

分区合并通常用于将一些较小的列表（list）分区，或相邻的较小的范围分区合并，减少分区数量，提高效率。

使用限制

1. 不能用于散列（HASH）分区和子分区的合并。
2. 范围（RANGE）分区合并时，允许合并两个临近范围的分区到另外分区，不相邻分区无法合并。合并结果分区继承两个源分区的最大边界。
3. 暂不支持UPDATE [ GLOBAL ] INDEXES。

语法

**ALTER TABLE table\_name MERGE PARTITIONS ( part1, part2 ) INTO PARTITION part3 ;**

示例

当前分区有part\_range\_t2\_part\_0、part\_range\_t2\_part\_1、part\_range\_t2\_part\_2、part\_range\_t2\_part\_21 四个。

**postgres=# \d+ part\_range\_t2**

**Table "public.part\_range\_t2"**

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

**--------+-----------------------------+-----------+----------+---------+----------+--------------+-------------**

**f1 | integer | | not null | | plain | |**

**f2 | timestamp without time zone | | not null | now() | plain | |**

**f3 | character varying(50) | | | | extended | |**

**f4 | integer | | | | plain | |**

**Partition key: RANGE (f2)**

**Partitions: part\_range\_t2\_part\_0 FOR VALUES FROM ('2020-06-01 00:00:00') TO ('2020-07-01 00:00:00'),**

**part\_range\_t2\_part\_1 FOR VALUES FROM ('2020-07-01 00:00:00') TO ('2020-08-01 00:00:00'),**

**part\_range\_t2\_part\_2 FOR VALUES FROM ('2020-08-20 00:00:00') TO ('2020-09-01 00:00:00'),**

**part\_range\_t2\_part\_21 FOR VALUES FROM ('2020-08-01 00:00:00') TO ('2020-08-20 00:00:00')**

**Distribute By: SHARD(f1)**

**Location Nodes: ALL DATANODES**

**下面的sql将part\_range\_t2\_part\_2、part\_range\_t2\_part\_21两个分区的数据合并到part\_range\_t2\_part\_22中。**

**postgres=# ALTER TABLE part\_range\_t2 MERGE PARTITIONS ( part\_range\_t2\_part\_2, part\_range\_t2\_part\_21) INTO PARTITION part\_range\_t2\_part\_22;**

**ALTER TABLE**

**postgres=# \d+ part\_range\_t2**

**Table "public.part\_range\_t2"**

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

**--------+-----------------------------+-----------+----------+---------+----------+--------------+-------------**

**f1 | integer | | not null | | plain | |**

**f2 | timestamp without time zone | | not null | now() | plain | |**

**f3 | character varying(50) | | | | extended | |**

**f4 | integer | | | | plain | |**

**Partition key: RANGE (f2)**

**Partitions: part\_range\_t2\_part\_0 FOR VALUES FROM ('2020-06-01 00:00:00') TO ('2020-07-01 00:00:00'),**

**part\_range\_t2\_part\_1 FOR VALUES FROM ('2020-07-01 00:00:00') TO ('2020-08-01 00:00:00'),**

**part\_range\_t2\_part\_22 FOR VALUES FROM ('2020-08-01 00:00:00') TO ('2020-09-01 00:00:00')**

**Distribute By: SHARD(f1)**

**Location Nodes: ALL DATANODES**

**postgres=#**

确认分区合并后，分区从4个变为3个。

**postgres=# select \* from part\_range\_t2\_part\_22;**

**f1 | f2 | f3 | f4**

**----+---------------------+-------+----**

**4 | 2020-08-21 00:00:00 | four | 4**

**3 | 2020-08-01 00:00:00 | three | 4**

**(2 rows)**

确认数据迁移至合并后的分区。

## 支持不同父表下相同子表名称

Oracle父表不相同，子表名称可以相同，TBase V5兼容Oracle，支持不同父表下相同子表名称。

需要先在创建分区子表时，通过as 子表别名的语法指定别名。

语法

**table\_name PARTITIONS ( part\_name )**

示例

如在上文分区表中新增2个分区。

**postgres=# create table part\_range\_t2\_part\_3 partition of part\_range\_t2 for values from ('2020-09-01') to ('2020-10-01') as part\_3;**

**CREATE TABLE**

**postgres=# create table part\_list\_t1\_cd partition of part\_list\_t1(f1 ,f2 , f3,f4) for values in ('成都') as part\_3;**

**CREATE TABLE**

插入数据：

**postgres=# insert into part\_range\_t2(f1,f2,f3,f4) values(6,'2020-09-01','four',6);**

**INSERT 0 1**

**postgres=# insert into part\_list\_t1(f1,f2,f3,f4) values(7,'成都',7,now());**

**INSERT 0 1**

**postgres=# insert into part\_list\_t1(f1,f2,f3,f4) values(8,'成都',8,now());**

**INSERT 0 1**

**postgres=#**

分区子表同名（实为分区子表别名同名），查询数据正常。

**postgres=# select \* from part\_range\_t2 partition(part\_3) where rownum<=3;**

**f1 | f2 | f3 | f4**

**----+---------------------+------+----**

**6 | 2020-09-01 00:00:00 | four | 6**

**(1 row)**

**postgres=# select \* from part\_list\_t1 partition(part\_3) where rownum<=3;**

**f1 | f2 | f3 | f4**

**----+------+----+---------------------**

**8 | 成都 | 8 | 2021-03-18 18:00:01**

**7 | 成都 | 7 | 2021-03-18 18:00:00**

**(2 rows)**

## TRUNCATE分区子表

兼容Oracle语法，TBase V5支持分区子表TRUNCATE。

语法：

**ALTER TABLE parent\_table TRUNCATE PARTITION (part\_name) UPDATE [GLOBAL] INDEXES;**

示例：

**postgres=# alter table part\_range\_t2 truncate partition ( part\_3 ) update global indexes;**

**TRUNCATE TABLE**

**postgres=# select \* from part\_range\_t2 partition ( part\_3 ) ;**

**f1 | f2 | f3 | f4**

**----+----+----+----**

**(0 rows)**

**postgres=#**

## 分区子表直接访问

兼容Oracle语法，TBase V5支持分区子表直接访问。

语法：

**parent\_table PARTITION (part\_name)**

示例：

**postgres=# select \* from part\_range\_t2 partition(part\_3) where rownum<=3;**

**f1 | f2 | f3 | f4**

**----+---------------------+------+----**

**6 | 2020-09-01 00:00:00 | four | 6**

**(1 row)**

**postgres=# select \* from part\_list\_t1 partition(part\_3) where rownum<=3;**

**f1 | f2 | f3 | f4**

**----+------+----+---------------------**

**8 | 成都 | 8 | 2021-03-18 18:00:01**

**7 | 成都 | 7 | 2021-03-18 18:00:00**

**(2 rows)**

# 系统内置包

## DBMS\_RANDOM

DBMS\_RANDOM包用于产生随机值。DBMS\_RANDOM中可以使用的存储过程及函数如下表所示：

|  |  |
| --- | --- |
| **存储过程/函数** | **描述** |
| \*INITIALIZE | 用一个种子值来初始化DBMS\_RANDOM |
| NORMAL | 返回服从正态分布的一组数 |
| \*RANDOM | 返回值的范围为： [-2^31, 2^31)，返回的是整数 |
| \*SEED | 功能和INITIALIZE函数类似，实际上，INITIALIZE函数被淘汰，推荐的替代函数即是SEED函数。与INITIALIZE函数不同的是SEED函数同时支持数值和字符作为种子值，而INITIALIZE函数只支持数值 |
| \*STRING | 随机生成字符串 |
| \*TERMINATE | 在使用完DBMS\_RANDOM包后，用该函数进行终止 |
| \*VALUE | 随机返回数值 |

示例：

**Create table random\_t1(col1 int,col2 numeric,col3 numeric,col4 numeric,col5 numeric);**

**Create table random\_t2 as select \* from random\_t1;**

**begin**

**perform dbms\_random.initialize(100);**

**for i in 1..100 loop**

**insert into random\_t1 values(i,dbms\_random.normal(),dbms\_random.random(),dbms\_random.value(),dbms\_random.value(10,20));**

**end loop;**

**end;**

**/**

**begin**

**perform dbms\_random.initialize(100);**

**for i in 1..100 loop**

**insert into random\_t2 values(i,dbms\_random.normal(),dbms\_random.random(),dbms\_random.value(),dbms\_random.value(10,20));**

**end loop;**

**end;**

**/**

**select \* from random\_t1 minus select \* from random\_t2;**

**Drop table if exists random\_t1;**

**Drop table if exists random\_t2;**

## DBMS\_SQL

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

|  |  |
| --- | --- |
| **存储过程/函数** | **描述** |
| BIND\_VARIABLE(c, name, value [, out\_value\_size ]) | Bind a value to a variable |
| BIND\_VARIABLE\_CHAR(c, name, value [, out\_value\_size ]) | Bind a CHAR value to a variable |
| BIND\_VARIABLE\_RAW(c, name, value [, out\_value\_size ]) | Bind a RAW value to a variable |
| CLOSE\_CURSOR(c IN OUT) | Close a cursor |
| COLUMN\_VALUE(c, position, value OUT [, column\_error OUT [, actual\_length OUT ]]) | Return a column value into a variable. |
| COLUMN\_VALUE\_CHAR(c, position, value OUT [, column\_error OUT [, actual\_length OUT ]]) | Return a CHAR column value into a variable. |
| COLUMN\_VALUE\_RAW(c, position, value OUT [, column\_error OUT [, actual\_length OUT ]]) | Return a RAW column value into a variable. |
| DEFINE\_COLUMN(c, position, column [, column\_size ]) | Define a column in the SELECT list. |
| DEFINE\_COLUMN\_CHAR(c, position, column, column\_size) | Define a CHAR column in the SELECT list. |
| DEFINE\_COLUMN\_RAW(c, position, column, column\_size) | Define a RAW column in the SELECT list. |
| DESCRIBE\_COLUMNS | Defines columns to hold a cursor result set. |
| EXECUTE(c) | Execute a cursor. |
| EXECUTE\_AND\_FETCH(c [, exact ]) | Execute a cursor and fetch a single row. |
| FETCH\_ROWS(c) | Fetch rows from the cursor. |
| IS\_OPEN(c) | Check if a cursor is open. |
| LAST\_ROW\_COUNT | Return cumulative number of rows fetched. |
| OPEN\_CURSOR | Open a cursor. |
| PARSE(c, statement, language\_flag) | Parse a statement. |

示例：

**set client\_min\_messages TO error;**

**CREATE EXTENSION IF NOT EXISTS dbms\_sql;**

**set client\_min\_messages TO default;**

**do**

**$$**

**declare**

**c int;**

**strval varchar;**

**intval int;**

**nrows int default 30;**

**begin**

**c := dbms\_sql.open\_cursor();**

**call dbms\_sql.parse(c, 'select ''ahoj'' || i, i from generate\_series(1, :nrows) g(i)');**

**call dbms\_sql.bind\_variable(c, 'nrows', nrows);**

**call dbms\_sql.define\_column(c, 1, strval);**

**call dbms\_sql.define\_column(c, 2, intval);**

**perform dbms\_sql.execute(c);**

**while dbms\_sql.fetch\_rows(c) > 0**

**loop**

**call dbms\_sql.column\_value(c, 1, strval);**

**call dbms\_sql.column\_value(c, 2, intval);**

**raise notice 'c1: %, c2: %', strval, intval;**

**end loop;**

**call dbms\_sql.close\_cursor(c);**

**end;**

**$$;**

**do**

**$$**

**declare**

**c int;**

**strval varchar;**

**intval int;**

**nrows int default 30;**

**begin**

**c := dbms\_sql.open\_cursor();**

**call dbms\_sql.parse(c, 'select ''ahoj'' || i, i from generate\_series(1, :nrows) g(i)');**

**call dbms\_sql.bind\_variable(c, 'nrows', nrows);**

**call dbms\_sql.define\_column(c, 1, strval);**

**call dbms\_sql.define\_column(c, 2, intval);**

**perform dbms\_sql.execute(c);**

**while dbms\_sql.fetch\_rows(c) > 0**

**loop**

**strval := dbms\_sql.column\_value\_f(c, 1, strval);**

**intval := dbms\_sql.column\_value\_f(c, 2, intval);**

**raise notice 'c1: %, c2: %', strval, intval;**

**end loop;**

**call dbms\_sql.close\_cursor(c);**

**end;**

**$$;**

**create table foo**

**(**

**a int,**

**b varchar,**

**c numeric**

**);**

**do**

**$$**

**declare**

**c int;**

**begin**

**c := dbms\_sql.open\_cursor();**

**call dbms\_sql.parse(c, 'insert into foo values(:a, :b, :c)');**

**for i in 1..100**

**loop**

**call dbms\_sql.bind\_variable(c, 'a', i);**

**call dbms\_sql.bind\_variable(c, 'b', 'Ahoj ' || i);**

**call dbms\_sql.bind\_variable(c, 'c', i + 0.033);**

**perform dbms\_sql.execute(c);**

**end loop;**

**end;**

**$$;**

**select \***

**from foo**

**order by a;**

**truncate foo;**

**do**

**$$**

**declare**

**c int;**

**begin**

**c := dbms\_sql.open\_cursor();**

**call dbms\_sql.parse(c, 'insert into foo values(:a, :b, :c)');**

**for i in 1..100**

**loop**

**perform dbms\_sql.bind\_variable\_f(c, 'a', i);**

**perform dbms\_sql.bind\_variable\_f(c, 'b', 'Ahoj ' || i);**

**perform dbms\_sql.bind\_variable\_f(c, 'c', i + 0.033);**

**perform dbms\_sql.execute(c);**

**end loop;**

**end;**

**$$;**

**select \***

**from foo**

**order by a;**

**truncate foo;**

**do**

**$$**

**declare**

**c int;**

**a int[];**

**b varchar[];**

**ca numeric[];**

**begin**

**c := dbms\_sql.open\_cursor();**

**call dbms\_sql.parse(c, 'insert into foo values(:a, :b, :c)');**

**a := ARRAY [1, 2, 3, 4, 5];**

**b := ARRAY ['Ahoj', 'Nazdar', 'Bazar'];**

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

**call dbms\_sql.bind\_array(c, 'a', a);**

**call dbms\_sql.bind\_array(c, 'b', b);**

**call dbms\_sql.bind\_array(c, 'c', ca);**

**raise notice 'inserted rows %', dbms\_sql.execute(c);**

**end;**

**$$;**

**select \***

**from foo**

**order by a;**

**truncate foo;**

**do**

**$$**

**declare**

**c int;**

**a int[];**

**b varchar[];**

**ca numeric[];**

**begin**

**c := dbms\_sql.open\_cursor();**

**call dbms\_sql.parse(c, 'insert into foo values(:a, :b, :c)');**

**a := ARRAY [1, 2, 3, 4, 5];**

**b := ARRAY ['Ahoj', 'Nazdar', 'Bazar'];**

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

**call dbms\_sql.bind\_array(c, 'a', a, 2, 3);**

**call dbms\_sql.bind\_array(c, 'b', b, 3, 4);**

**call dbms\_sql.bind\_array(c, 'c', ca);**

**raise notice 'inserted rows %', dbms\_sql.execute(c);**

**end;**

**$$;**

**select \***

**from foo**

**order by a;**

**truncate foo;**

**do**

**$$**

**declare**

**c int;**

**a int[];**

**b varchar[];**

**ca numeric[];**

**begin**

**c := dbms\_sql.open\_cursor();**

**call dbms\_sql.parse(c, 'select i, ''Ahoj'' || i, i + 0.003 from generate\_series(1, 35) g(i)');**

**call dbms\_sql.define\_array(c, 1, a, 10, 1);**

**call dbms\_sql.define\_array(c, 2, b, 10, 1);**

**call dbms\_sql.define\_array(c, 3, ca, 10, 1);**

**perform dbms\_sql.execute(c);**

**while dbms\_sql.fetch\_rows(c) > 0**

**loop**

**call dbms\_sql.column\_value(c, 1, a);**

**call dbms\_sql.column\_value(c, 2, b);**

**call dbms\_sql.column\_value(c, 3, ca);**

**raise notice 'a = %', a;**

**raise notice 'b = %', b;**

**raise notice 'c = %', ca;**

**end loop;**

**call dbms\_sql.close\_cursor(c);**

**end;**

**$$;**

## DBMS\_STATS

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

|  |  |
| --- | --- |
| **接口** | **描述** |
| GATHER\_DATABASE\_STATS | 分析数据库，包括所有用户对象和系统对象 |
| GATHER\_TABLE\_STATS | 分析表 |
| GET\_COLUMN\_STATS | 取得列的统计信息 |
| GET\_INDEX\_STATS | 取得索引的统计信息 |
| GET\_TABLE\_STATS | 取得表的统计信息 |

示例：

**CREATE EXTENSION IF NOT EXISTS tbase\_oracle\_package\_function;**

**create user godlike\_dbms\_stats superuser;**

**create user joe;**

**create user no\_privilege;**

**\c postgres godlike\_dbms\_stats**

**grant usage on schema dbms\_stats to no\_privilege;**

**grant usage on schema dbms\_stats to joe;**

**\c postgres joe**

**-- table joe\_t**

**create table joe\_t (id integer not null PRIMARY KEY, test integer);**

**create index joe\_t\_test\_idx on joe\_t(test);**

**insert into joe\_t SELECT generate\_series(1,1000) as key, (random()\*(10^3))::integer;**

**\c postgres joe**

**exec dbms\_stats.gather\_table\_stats(ownname => 'joe',tabname => 'joe\_t');**

**exec dbms\_stats.get\_table\_stats(ownname => 'joe',tabname => 'joe\_t');**

**exec dbms\_stats.get\_column\_stats('joe', 'joe\_t', 'test');**

**exec dbms\_stats.get\_index\_stats('joe', 'joe\_t\_test\_idx');**

**-- table joe\_t\_p**

**create table joe\_t\_p (id integer not null PRIMARY KEY, test integer) partition by range (id) begin (1) step (5) partitions (200) distribute by shard(id);**

**create index joe\_t\_p\_test\_idx on joe\_t\_p(test);**

**insert into joe\_t\_p SELECT generate\_series(1,1000) as key, (random()\*(10^3))::integer;**

**exec dbms\_stats.gather\_database\_stats();**

**exec dbms\_stats.get\_table\_stats(ownname => 'joe',tabname => 'joe\_t\_p');**

**exec dbms\_stats.get\_column\_stats('joe', 'joe\_t\_p', 'test');**

**exec dbms\_stats.get\_index\_stats('joe', 'joe\_t\_p\_test\_idx');**

**exec dbms\_stats.get\_table\_stats('joe', 'joe\_t\_p', 'joe\_t\_p\_part\_0');**

**exec dbms\_stats.get\_index\_stats('joe', 'joe\_t\_p\_test\_idx', 'joe\_t\_p\_test\_idx\_part\_0');**

**-- clean**

**\c postgres godlike\_dbms\_stats**

**drop table joe\_t;**

**drop table joe\_t\_p;**

**REVOKE usage ON schema dbms\_stats FROM joe;**

**REVOKE usage ON schema dbms\_stats FROM no\_privilege;**

**drop user joe;**

**drop user no\_privilege;**

## DBMS\_LOB

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

| **存储过程/函数** | **描述** |
| --- | --- |
| **APPEND(dest\_lob IN OUT,src\_lob)** | **Appends one large object to another.** |
| **COMPARE(lob\_1, lob\_2 [, amount[, offset\_1 [, offset\_2 ]]])** | **Compares two large objects.** |
| **CONVERTOBLOB(dest\_lob IN OUT,src\_clob, amount, dest\_offsetIN OUT, src\_offset IN OUT,blob\_csid, lang\_context IN OUT,warning OUT)** | **Converts character data to binary.** |
| **CONVERTTOCLOB(dest\_lob IN OUT,src\_blob, amount, dest\_offsetIN OUT, src\_offset IN OUT,blob\_csid, lang\_context IN OUT,warning OUT)** | **Converts binary data to character.** |
| **COPY(dest\_lob IN OUT, src\_lob,amount [, dest\_offset [,src\_offset ]])** | **Copies one large object to another.** |
| **ERASE(lob\_loc IN OUT, amount IN OUT [, offset ])** | **Erase a large object.** |
| **GET\_STORAGE\_LIMIT(lob\_loc)** | **Get the storage limit for large objects.** |
| **GETLENGTH(lob\_loc)** | **Get the length of the large object.** |
| **INSTR(lob\_loc, pattern [,offset [, nth ]])** | **Get the position of the nth occurrence of a pattern in the large object starting atoffset.** |
| **READ(lob\_loc, amount IN OUT,offset, buffer OUT)** | **Read a large object.** |
| **SUBSTR(lob\_loc [, amount [,offset ]])** | **Get part of a large object.** |
| **TRIM(lob\_loc IN OUT, newlen)** | **Trim a large object to the specified length.** |
| **WRITE(lob\_loc IN OUT, amount,offset, buffer)** | **Write data to a large object.** |
| **WRITEAPPEND(lob\_loc IN OUT,amount, buffer)** | **Write data from the buffer to the end of a large object.** |

示例：

**DBMS\_LOB.WRITEAPPEND (**

**lob\_loc IN OUT NOCOPY BLOB,**

**amount  IN            INTEGER,**

**buffer  IN            RAW);**

**DBMS\_LOB.WRITEAPPEND (**

**lob\_loc IN OUT NOCOPY CLOB CHARACTER SET ANY\_CS,**

**amount  IN            INTEGER,**

**buffer  IN            VARCHAR2 CHARACTER SET lob\_loc%CHARSET);**

**declare**

**v\_clob1 clob;**

**begin**

**v\_clob1:=to\_clob('123456');**

**dbms\_output.put\_line(v\_clob1);**

**DBMS\_LOB.WRITEAPPEND(v\_clob1,3,'789');**

**dbms\_output.put\_line(v\_clob1);**

**end;**

**/**

## DBMS\_JOB

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

示例：

**declare**

**jobno number;**

**begin dbms\_job.submit(**

**job=>jobno,**

**what=>'proc\_job(2);',**

**next\_date=>sysdate,**

**interval=>'sysdate+1/1440');**

**COMMIT;**

**dbms\_output.put\_line(jobno);**

**end;**

**/**

## DBMS\_ASSERT

这个包来判定输入值的属性。

示例：

select dbms\_assert.enquote\_name('TBase') from dual;

## DBMS\_OUTPUT

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

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

示例：

**\set ECHO none**

**SET client\_min\_messages = warning;**

**SET DATESTYLE TO ISO;**

**SET client\_encoding = utf8;**

**\pset null '<NULL>'**

**\set ECHO all**

**DROP FUNCTION dbms\_output\_test();**

**DROP TABLE dbms\_output\_test;**

**-- DBMS\_OUTPUT.DISABLE [0]**

**CREATE TABLE dbms\_output\_test (id serial,buff VARCHAR(20), status INTEGER);**

**CREATE FUNCTION dbms\_output\_test() RETURNS VOID AS $$**

**DECLARE**

**buff VARCHAR(20);**

**stts INTEGER;**

**BEGIN**

**PERFORM DBMS\_OUTPUT.PUT\_LINE ('ORAFCE TEST 1');**

**SELECT INTO buff,stts line,status FROM DBMS\_OUTPUT.GET\_LINE();**

**INSERT INTO dbms\_output\_test(buff,status) VALUES (buff, stts);**

**END;**

**$$ LANGUAGE plpgsql;**

**SELECT dbms\_output\_test();**

**DROP TABLE dbms\_output\_test;**

**DROP FUNCTION dbms\_output\_test();**

**-- DBMS\_OUTPUT.PUT\_LINE [1]**

**CREATE FUNCTION dbms\_output\_test() RETURNS VOID AS $$**

**DECLARE**

**buff1 VARCHAR(20) := 'orafce';**

**BEGIN**

**PERFORM DBMS\_OUTPUT.DISABLE();**

**PERFORM DBMS\_OUTPUT.ENABLE();**

**PERFORM DBMS\_OUTPUT.SERVEROUTPUT ('t');**

**PERFORM DBMS\_OUTPUT.PUT\_LINE ('ORAFCE');**

**PERFORM DBMS\_OUTPUT.PUT\_LINE (buff1);**

**PERFORM DBMS\_OUTPUT.PUT ('ABC');**

**PERFORM DBMS\_OUTPUT.PUT\_LINE ('');**

**END;**

**$$ LANGUAGE plpgsql;**

**SELECT dbms\_output\_test();**

**DROP FUNCTION dbms\_output\_test();**

**-- DBMS\_OUTPUT.PUT\_LINE [2]**

**CREATE FUNCTION dbms\_output\_test() RETURNS VOID AS $$**

**BEGIN**

**PERFORM DBMS\_OUTPUT.DISABLE();**

**PERFORM DBMS\_OUTPUT.ENABLE();**

**PERFORM DBMS\_OUTPUT.SERVEROUTPUT ('t');**

**PERFORM DBMS\_OUTPUT.PUT\_LINE ('ORA**

**F**

**CE');**

**END;**

**$$ LANGUAGE plpgsql;**

**SELECT dbms\_output\_test();**

**DROP FUNCTION dbms\_output\_test();**

**-- DBMS\_OUTPUT.PUT [1]**

**CREATE FUNCTION dbms\_output\_test() RETURNS VOID AS $$**

**DECLARE**

**buff1 VARCHAR(20) := 'ora';**

**buff2 VARCHAR(20) := 'f';**

**buff3 VARCHAR(20) := 'ce';**

**BEGIN**

**PERFORM DBMS\_OUTPUT.DISABLE();**

**PERFORM DBMS\_OUTPUT.ENABLE();**

**PERFORM DBMS\_OUTPUT.SERVEROUTPUT ('t');**

**PERFORM DBMS\_OUTPUT.PUT ('ORA');**

**PERFORM DBMS\_OUTPUT.PUT ('F');**

**PERFORM DBMS\_OUTPUT.PUT ('CE');**

**PERFORM DBMS\_OUTPUT.PUT\_LINE ('');**

**PERFORM DBMS\_OUTPUT.PUT ('ABC');**

**PERFORM DBMS\_OUTPUT.PUT\_LINE ('');**

**END;**

**$$ LANGUAGE plpgsql;**

**SELECT dbms\_output\_test();**

**DROP FUNCTION dbms\_output\_test();**

**-- DBMS\_OUTPUT.PUT [2]**

**CREATE FUNCTION dbms\_output\_test() RETURNS VOID AS $$**

**BEGIN**

**PERFORM DBMS\_OUTPUT.DISABLE();**

**PERFORM DBMS\_OUTPUT.ENABLE();**

**PERFORM DBMS\_OUTPUT.SERVEROUTPUT ('t');**

**PERFORM DBMS\_OUTPUT.PUT ('ORA**

**F**

**CE');**

**PERFORM DBMS\_OUTPUT.PUT\_LINE ('');**

**END;**

**$$ LANGUAGE plpgsql;**

**SELECT dbms\_output\_test();**

**DROP FUNCTION dbms\_output\_test();**

**-- DBMS\_OUTPUT.GET\_LINE [1]**

**CREATE TABLE dbms\_output\_test (id serial,buff VARCHAR(20), status INTEGER);**

**CREATE FUNCTION dbms\_output\_test() RETURNS VOID AS $$**

**DECLARE**

**buff VARCHAR(20);**

**stts INTEGER;**

**BEGIN**

**PERFORM DBMS\_OUTPUT.DISABLE();**

**PERFORM DBMS\_OUTPUT.ENABLE();**

**PERFORM DBMS\_OUTPUT.SERVEROUTPUT ('f');**

**PERFORM DBMS\_OUTPUT.PUT\_LINE ('ORAFCE TEST 1');**

**PERFORM DBMS\_OUTPUT.PUT\_LINE ('ORAFCE TEST 2');**

**SELECT INTO buff,stts line,status FROM DBMS\_OUTPUT.GET\_LINE();**

**INSERT INTO dbms\_output\_test(buff,status) VALUES (buff, stts);**

**SELECT INTO buff,stts line,status FROM DBMS\_OUTPUT.GET\_LINE();**

**INSERT INTO dbms\_output\_test(buff,status) VALUES (buff, stts);**

**PERFORM DBMS\_OUTPUT.DISABLE();**

**PERFORM DBMS\_OUTPUT.ENABLE();**

**END;**

**$$ LANGUAGE plpgsql;**

**SELECT dbms\_output\_test();**

**SELECT \* FROM dbms\_output\_test order by buff;**

**DROP TABLE dbms\_output\_test;**

**DROP FUNCTION dbms\_output\_test();**

**-- DBMS\_OUTPUT.GET\_LINE [2]**

**CREATE TABLE dbms\_output\_test (id serial,buff VARCHAR(20), status INTEGER);**

**CREATE FUNCTION dbms\_output\_test() RETURNS VOID AS $$**

**DECLARE**

**buff VARCHAR(20);**

**stts INTEGER;**

**BEGIN**

**PERFORM DBMS\_OUTPUT.DISABLE();**

**PERFORM DBMS\_OUTPUT.ENABLE();**

**PERFORM DBMS\_OUTPUT.SERVEROUTPUT ('f');**

**PERFORM DBMS\_OUTPUT.PUT\_LINE ('ORAFCE TEST 1');**

**PERFORM DBMS\_OUTPUT.PUT\_LINE ('ORAFCE TEST 2');**

**SELECT INTO buff,stts line,status FROM DBMS\_OUTPUT.GET\_LINE();**

**INSERT INTO dbms\_output\_test(buff,status) VALUES (buff, stts);**

**PERFORM DBMS\_OUTPUT.PUT\_LINE ('ORAFCE TEST 3');**

**SELECT INTO buff,stts line,status FROM DBMS\_OUTPUT.GET\_LINE();**

**INSERT INTO dbms\_output\_test(buff,status) VALUES (buff, stts);**

**SELECT INTO buff,stts line,status FROM DBMS\_OUTPUT.GET\_LINE();**

**INSERT INTO dbms\_output\_test(buff,status) VALUES (buff, stts);**

**SELECT INTO buff,stts line,status FROM DBMS\_OUTPUT.GET\_LINE();**

**INSERT INTO dbms\_output\_test(buff,status) VALUES (buff, stts);**

**PERFORM DBMS\_OUTPUT.DISABLE();**

**PERFORM DBMS\_OUTPUT.ENABLE();**

**END;**

**$$ LANGUAGE plpgsql;**

**SELECT dbms\_output\_test();**

**DROP TABLE dbms\_output\_test;**

**DROP FUNCTION dbms\_output\_test();**

**-- DBMS\_OUTPUT.GET\_LINE [3]**

**CREATE TABLE dbms\_output\_test (id serial,buff VARCHAR(20), status INTEGER);**

**CREATE FUNCTION dbms\_output\_test() RETURNS VOID AS $$**

**DECLARE**

**buff VARCHAR(20);**

**stts INTEGER;**

**BEGIN**

**PERFORM DBMS\_OUTPUT.DISABLE();**

**PERFORM DBMS\_OUTPUT.ENABLE();**

**PERFORM DBMS\_OUTPUT.SERVEROUTPUT ('f');**

**PERFORM DBMS\_OUTPUT.PUT\_LINE ('ORAFCE TEST 1');**

**PERFORM DBMS\_OUTPUT.PUT\_LINE ('ORAFCE TEST 2');**

**SELECT INTO buff,stts line,status FROM DBMS\_OUTPUT.GET\_LINE();**

**INSERT INTO dbms\_output\_test(buff,status) VALUES (buff, stts);**

**PERFORM DBMS\_OUTPUT.PUT ('ORA');**

**SELECT INTO buff,stts line,status FROM DBMS\_OUTPUT.GET\_LINE();**

**INSERT INTO dbms\_output\_test(buff,status) VALUES (buff, stts);**

**SELECT INTO buff,stts line,status FROM DBMS\_OUTPUT.GET\_LINE();**

**INSERT INTO dbms\_output\_test(buff,status) VALUES (buff, stts);**

**PERFORM DBMS\_OUTPUT.DISABLE();**

**PERFORM DBMS\_OUTPUT.ENABLE();**

**END;**

**$$ LANGUAGE plpgsql;**

**SELECT dbms\_output\_test();**

**DROP TABLE dbms\_output\_test;**

**DROP FUNCTION dbms\_output\_test();**

**-- DBMS\_OUTPUT.GET\_LINE [4]**

**CREATE TABLE dbms\_output\_test (id serial,buff VARCHAR(20), status INTEGER);**

**CREATE FUNCTION dbms\_output\_test() RETURNS VOID AS $$**

**DECLARE**

**buff VARCHAR(20);**

**stts INTEGER;**

**BEGIN**

**PERFORM DBMS\_OUTPUT.DISABLE();**

**PERFORM DBMS\_OUTPUT.ENABLE();**

**PERFORM DBMS\_OUTPUT.SERVEROUTPUT ('f');**

**PERFORM DBMS\_OUTPUT.PUT\_LINE ('ORAFCE TEST 1');**

**PERFORM DBMS\_OUTPUT.PUT\_LINE ('ORAFCE TEST 2');**

**SELECT INTO buff,stts line,status FROM DBMS\_OUTPUT.GET\_LINE();**

**INSERT INTO dbms\_output\_test(buff,status) VALUES (buff, stts);**

**PERFORM DBMS\_OUTPUT.NEW\_LINE();**

**SELECT INTO buff,stts line,status FROM DBMS\_OUTPUT.GET\_LINE();**

**INSERT INTO dbms\_output\_test(buff,status) VALUES (buff, stts);**

**SELECT INTO buff,stts line,status FROM DBMS\_OUTPUT.GET\_LINE();**

**INSERT INTO dbms\_output\_test(buff,status) VALUES (buff, stts);**

**PERFORM DBMS\_OUTPUT.DISABLE();**

**PERFORM DBMS\_OUTPUT.ENABLE();**

**END;**

**$$ LANGUAGE plpgsql;**

**SELECT dbms\_output\_test();**

**DROP TABLE dbms\_output\_test;**

**DROP FUNCTION dbms\_output\_test();**

**-- DBMS\_OUTPUT.GET\_LINE [5]**

**CREATE TABLE dbms\_output\_test (id serial,buff VARCHAR(20), status INTEGER);**

**CREATE FUNCTION dbms\_output\_test() RETURNS VOID AS $$**

**DECLARE**

**buff VARCHAR(20);**

**stts INTEGER;**

**BEGIN**

**PERFORM DBMS\_OUTPUT.DISABLE();**

**PERFORM DBMS\_OUTPUT.ENABLE();**

**PERFORM DBMS\_OUTPUT.SERVEROUTPUT ('f');**

**PERFORM DBMS\_OUTPUT.PUT\_LINE ('ORAFCE TEST 1**

**');**

**PERFORM DBMS\_OUTPUT.PUT\_LINE ('ORAFCE TEST 2');**

**SELECT INTO buff,stts line,status FROM DBMS\_OUTPUT.GET\_LINE();**

**INSERT INTO dbms\_output\_test(buff,status) VALUES (buff, stts);**

**SELECT INTO buff,stts line,status FROM DBMS\_OUTPUT.GET\_LINE();**

**INSERT INTO dbms\_output\_test(buff,status) VALUES (buff, stts);**

**PERFORM DBMS\_OUTPUT.DISABLE();**

**PERFORM DBMS\_OUTPUT.ENABLE();**

**END;**

**$$ LANGUAGE plpgsql;**

**SELECT dbms\_output\_test();**

**DROP TABLE dbms\_output\_test;**

**DROP FUNCTION dbms\_output\_test();**

**-- DBMS\_OUTPUT.GET\_LINE [6]**

**CREATE TABLE dbms\_output\_test (id serial,buff VARCHAR(20), status INTEGER);**

**CREATE FUNCTION dbms\_output\_test() RETURNS VOID AS $$**

**DECLARE**

**buff VARCHAR(20);**

**stts INTEGER;**

**BEGIN**

**PERFORM DBMS\_OUTPUT.DISABLE();**

**PERFORM DBMS\_OUTPUT.ENABLE();**

**PERFORM DBMS\_OUTPUT.SERVEROUTPUT ('f');**

**PERFORM DBMS\_OUTPUT.PUT\_LINE ('ORA**

**F**

**CE');**

**SELECT INTO buff,stts line,status FROM DBMS\_OUTPUT.GET\_LINE();**

**INSERT INTO dbms\_output\_test(buff,status) VALUES (buff, stts);**

**PERFORM DBMS\_OUTPUT.DISABLE();**

**PERFORM DBMS\_OUTPUT.ENABLE();**

**END;**

**$$ LANGUAGE plpgsql;**

**SELECT dbms\_output\_test();**

**DROP TABLE dbms\_output\_test;**

**DROP FUNCTION dbms\_output\_test();**

**-- DBMS\_OUTPUT.GET\_LINES [1]**

**CREATE TABLE dbms\_output\_test (id serial,buff VARCHAR(20), status INTEGER);**

**CREATE FUNCTION dbms\_output\_test() RETURNS VOID AS $$**

**DECLARE**

**buff VARCHAR(20);**

**buff1 VARCHAR(20);**

**buff2 VARCHAR(20);**

**buff3 VARCHAR(20);**

**stts INTEGER := 10;**

**BEGIN**

**PERFORM DBMS\_OUTPUT.DISABLE();**

**PERFORM DBMS\_OUTPUT.ENABLE();**

**PERFORM DBMS\_OUTPUT.SERVEROUTPUT ('f');**

**PERFORM DBMS\_OUTPUT.PUT\_LINE ('ORAFCE TEST 1');**

**PERFORM DBMS\_OUTPUT.PUT\_LINE ('ORAFCE TEST 2');**

**PERFORM DBMS\_OUTPUT.PUT\_LINE ('ORAFCE TEST 3');**

**SELECT INTO buff1,buff2,buff3,stts lines[1],lines[2],lines[3],numlines FROM DBMS\_OUTPUT.GET\_LINES(stts);**

**INSERT INTO dbms\_output\_test(buff,status) VALUES (buff1, stts);**

**INSERT INTO dbms\_output\_test(buff,status) VALUES (buff2, stts);**

**INSERT INTO dbms\_output\_test(buff,status) VALUES (buff3, stts);**

**SELECT INTO buff,stts lines[1],numlines FROM DBMS\_OUTPUT.GET\_LINES(stts);**

**INSERT INTO dbms\_output\_test(buff,status) VALUES (buff, stts);**

**PERFORM DBMS\_OUTPUT.DISABLE();**

**PERFORM DBMS\_OUTPUT.ENABLE();**

**END;**

**$$ LANGUAGE plpgsql;**

**SELECT dbms\_output\_test();**

**DROP TABLE dbms\_output\_test;**

**DROP FUNCTION dbms\_output\_test();**

**-- DBMS\_OUTPUT.GET\_LINES [2]**

**CREATE TABLE dbms\_output\_test (id serial,buff VARCHAR(20), status INTEGER);**

**CREATE FUNCTION dbms\_output\_test() RETURNS VOID AS $$**

**DECLARE**

**buff VARCHAR(20);**

**buff1 VARCHAR(20);**

**buff2 VARCHAR(20);**

**stts INTEGER := 2;**

**BEGIN**

**PERFORM DBMS\_OUTPUT.DISABLE();**

**PERFORM DBMS\_OUTPUT.ENABLE();**

**PERFORM DBMS\_OUTPUT.SERVEROUTPUT ('f');**

**PERFORM DBMS\_OUTPUT.PUT\_LINE ('ORAFCE TEST 1');**

**PERFORM DBMS\_OUTPUT.PUT\_LINE ('ORAFCE TEST 2');**

**PERFORM DBMS\_OUTPUT.PUT\_LINE ('ORAFCE TEST 3');**

**SELECT INTO buff1,buff2,stts lines[1],lines[2],numlines FROM DBMS\_OUTPUT.GET\_LINES(stts);**

**INSERT INTO dbms\_output\_test(buff,status) VALUES (buff1, stts);**

**INSERT INTO dbms\_output\_test(buff,status) VALUES (buff2, stts);**

**SELECT INTO buff,stts lines[1],numlines FROM DBMS\_OUTPUT.GET\_LINES(stts);**

**INSERT INTO dbms\_output\_test(buff,status) VALUES (buff, stts);**

**PERFORM DBMS\_OUTPUT.DISABLE();**

**PERFORM DBMS\_OUTPUT.ENABLE();**

**END;**

**$$ LANGUAGE plpgsql;**

**SELECT dbms\_output\_test();**

**DROP TABLE dbms\_output\_test;**

**DROP FUNCTION dbms\_output\_test();**

**-- DBMS\_OUTPUT.GET\_LINES [3]**

**CREATE TABLE dbms\_output\_test (id serial,buff VARCHAR(20), status INTEGER);**

**CREATE FUNCTION dbms\_output\_test() RETURNS VOID AS $$**

**DECLARE**

**buff VARCHAR(20);**

**stts INTEGER := 1;**

**BEGIN**

**PERFORM DBMS\_OUTPUT.DISABLE();**

**PERFORM DBMS\_OUTPUT.ENABLE();**

**PERFORM DBMS\_OUTPUT.SERVEROUTPUT ('f');**

**PERFORM DBMS\_OUTPUT.PUT\_LINE ('ORAFCE TEST 1');**

**PERFORM DBMS\_OUTPUT.PUT\_LINE ('ORAFCE TEST 2');**

**SELECT INTO buff,stts lines[1],numlines FROM DBMS\_OUTPUT.GET\_LINES(stts);**

**INSERT INTO dbms\_output\_test(buff,status) VALUES (buff, stts);**

**PERFORM DBMS\_OUTPUT.PUT\_LINE ('ORAFCE TEST 3');**

**SELECT INTO buff,stts lines[1],numlines FROM DBMS\_OUTPUT.GET\_LINES(stts);**

**INSERT INTO dbms\_output\_test(buff,status) VALUES (buff, stts);**

**SELECT INTO buff,stts lines[1],numlines FROM DBMS\_OUTPUT.GET\_LINES(stts);**

**INSERT INTO dbms\_output\_test(buff,status) VALUES (buff, stts);**

**PERFORM DBMS\_OUTPUT.DISABLE();**

**PERFORM DBMS\_OUTPUT.ENABLE();**

**END;**

**$$ LANGUAGE plpgsql;**

**SELECT dbms\_output\_test();**

**DROP TABLE dbms\_output\_test;**

**DROP FUNCTION dbms\_output\_test();**

**-- DBMS\_OUTPUT.GET\_LINES [4]**

**CREATE TABLE dbms\_output\_test (id serial,buff VARCHAR(20), status INTEGER);**

**CREATE FUNCTION dbms\_output\_test() RETURNS VOID AS $$**

**DECLARE**

**buff VARCHAR(20);**

**stts INTEGER := 1;**

**BEGIN**

**PERFORM DBMS\_OUTPUT.DISABLE();**

**PERFORM DBMS\_OUTPUT.ENABLE();**

**PERFORM DBMS\_OUTPUT.SERVEROUTPUT ('f');**

**PERFORM DBMS\_OUTPUT.PUT\_LINE ('ORAFCE TEST 1');**

**PERFORM DBMS\_OUTPUT.PUT\_LINE ('ORAFCE TEST 2');**

**SELECT INTO buff,stts lines[1],numlines FROM DBMS\_OUTPUT.GET\_LINES(stts);**

**INSERT INTO dbms\_output\_test(buff,status) VALUES (buff, stts);**

**PERFORM DBMS\_OUTPUT.PUT ('ORA');**

**SELECT INTO buff,stts lines[1],numlines FROM DBMS\_OUTPUT.GET\_LINES(stts);**

**INSERT INTO dbms\_output\_test(buff,status) VALUES (buff, stts);**

**SELECT INTO buff,stts lines[1],numlines FROM DBMS\_OUTPUT.GET\_LINES(stts);**

**INSERT INTO dbms\_output\_test(buff,status) VALUES (buff, stts);**

**PERFORM DBMS\_OUTPUT.DISABLE();**

**PERFORM DBMS\_OUTPUT.ENABLE();**

**END;**

**$$ LANGUAGE plpgsql;**

**SELECT dbms\_output\_test();**

**DROP TABLE dbms\_output\_test;**

**DROP FUNCTION dbms\_output\_test();**

**-- DBMS\_OUTPUT.GET\_LINES [5]**

**CREATE TABLE dbms\_output\_test (id serial,buff VARCHAR(20), status INTEGER);**

**CREATE FUNCTION dbms\_output\_test() RETURNS VOID AS $$**

**DECLARE**

**buff VARCHAR(20);**

**stts INTEGER := 1;**

**BEGIN**

**PERFORM DBMS\_OUTPUT.DISABLE();**

**PERFORM DBMS\_OUTPUT.ENABLE();**

**PERFORM DBMS\_OUTPUT.SERVEROUTPUT ('f');**

**PERFORM DBMS\_OUTPUT.PUT\_LINE ('ORAFCE TEST 1');**

**PERFORM DBMS\_OUTPUT.PUT\_LINE ('ORAFCE TEST 2');**

**SELECT INTO buff,stts lines[1],numlines FROM DBMS\_OUTPUT.GET\_LINES(stts);**

**INSERT INTO dbms\_output\_test(buff,status) VALUES (buff, stts);**

**PERFORM DBMS\_OUTPUT.NEW\_LINE();**

**SELECT INTO buff,stts lines[1],numlines FROM DBMS\_OUTPUT.GET\_LINES(stts);**

**INSERT INTO dbms\_output\_test(buff,status) VALUES (buff, stts);**

**SELECT INTO buff,stts lines[1],numlines FROM DBMS\_OUTPUT.GET\_LINES(stts);**

**INSERT INTO dbms\_output\_test(buff,status) VALUES (buff, stts);**

**PERFORM DBMS\_OUTPUT.DISABLE();**

**PERFORM DBMS\_OUTPUT.ENABLE();**

**END;**

**$$ LANGUAGE plpgsql;**

**SELECT dbms\_output\_test();**

**DROP TABLE dbms\_output\_test;**

**DROP FUNCTION dbms\_output\_test();**

**-- DBMS\_OUTPUT.GET\_LINES [6]**

**CREATE TABLE dbms\_output\_test (id serial,buff VARCHAR(20), status INTEGER);**

**CREATE FUNCTION dbms\_output\_test() RETURNS VOID AS $$**

**DECLARE**

**buff VARCHAR(20);**

**stts INTEGER := 1;**

**BEGIN**

**PERFORM DBMS\_OUTPUT.DISABLE();**

**PERFORM DBMS\_OUTPUT.ENABLE();**

**PERFORM DBMS\_OUTPUT.SERVEROUTPUT ('f');**

**PERFORM DBMS\_OUTPUT.PUT\_LINE ('ORA**

**F**

**CE');**

**SELECT INTO buff,stts lines[1],numlines FROM DBMS\_OUTPUT.GET\_LINES(stts);**

**INSERT INTO dbms\_output\_test(buff,status) VALUES (buff, stts);**

**PERFORM DBMS\_OUTPUT.DISABLE();**

**PERFORM DBMS\_OUTPUT.ENABLE();**

**END;**

**$$ LANGUAGE plpgsql;**

**SELECT dbms\_output\_test();**

**DROP TABLE dbms\_output\_test;**

**DROP FUNCTION dbms\_output\_test();**

**-- DBMS\_OUTPUT.NEW\_LINE [1]**

**CREATE TABLE dbms\_output\_test (id serial,buff VARCHAR(20), status INTEGER);**

**CREATE FUNCTION dbms\_output\_test() RETURNS VOID AS $$**

**DECLARE**

**buff1 VARCHAR(20);**

**buff2 VARCHAR(20);**

**stts INTEGER := 10;**

**BEGIN**

**PERFORM DBMS\_OUTPUT.DISABLE();**

**PERFORM DBMS\_OUTPUT.ENABLE();**

**PERFORM DBMS\_OUTPUT.SERVEROUTPUT ('f');**

**PERFORM DBMS\_OUTPUT.PUT ('ORA');**

**PERFORM DBMS\_OUTPUT.NEW\_LINE();**

**PERFORM DBMS\_OUTPUT.PUT ('FCE');**

**PERFORM DBMS\_OUTPUT.NEW\_LINE();**

**SELECT INTO buff1,buff2,stts lines[1],lines[2],numlines FROM DBMS\_OUTPUT.GET\_LINES(stts);**

**INSERT INTO dbms\_output\_test(buff,status) VALUES (buff1, stts);**

**INSERT INTO dbms\_output\_test(buff,status) VALUES (buff2, stts);**

**PERFORM DBMS\_OUTPUT.DISABLE();**

**PERFORM DBMS\_OUTPUT.ENABLE();**

**END;**

**$$ LANGUAGE plpgsql;**

**SELECT dbms\_output\_test();**

**DROP TABLE dbms\_output\_test;**

**DROP FUNCTION dbms\_output\_test();**

**-- DBMS\_OUTPUT.NEW\_LINE [2]**

**CREATE TABLE dbms\_output\_test (id serial,buff VARCHAR(3000), status INTEGER);**

**CREATE FUNCTION dbms\_output\_test() RETURNS VOID AS $$**

**DECLARE**

**buff1 VARCHAR(3000);**

**stts INTEGER := 10;**

**BEGIN**

**PERFORM DBMS\_OUTPUT.DISABLE();**

**PERFORM DBMS\_OUTPUT.ENABLE();**

**PERFORM DBMS\_OUTPUT.SERVEROUTPUT ('f');**

**PERFORM DBMS\_OUTPUT.ENABLE(2000);**

**FOR j IN 1..1999 LOOP**

**PERFORM DBMS\_OUTPUT.PUT ('A');**

**END LOOP;**

**PERFORM DBMS\_OUTPUT.NEW\_LINE();**

**SELECT INTO buff1,stts lines[1],numlines FROM DBMS\_OUTPUT.GET\_LINES(stts);**

**INSERT INTO dbms\_output\_test(buff,status) VALUES (buff1, stts);**

**PERFORM DBMS\_OUTPUT.DISABLE();**

**PERFORM DBMS\_OUTPUT.ENABLE();**

**END;**

**$$ LANGUAGE plpgsql;**

**SELECT dbms\_output\_test();**

**DROP TABLE dbms\_output\_test;**

**DROP FUNCTION dbms\_output\_test();**

**-- DBMS\_OUTPUT.DISABLE [1]**

**CREATE TABLE dbms\_output\_test (id serial,buff VARCHAR(20), status INTEGER);**

**CREATE FUNCTION dbms\_output\_test() RETURNS VOID AS $$**

**DECLARE**

**buff VARCHAR(20);**

**stts INTEGER;**

**BEGIN**

**PERFORM DBMS\_OUTPUT.DISABLE();**

**PERFORM DBMS\_OUTPUT.ENABLE();**

**PERFORM DBMS\_OUTPUT.SERVEROUTPUT ('f');**

**PERFORM DBMS\_OUTPUT.PUT\_LINE ('ORAFCE TEST 1');**

**PERFORM DBMS\_OUTPUT.DISABLE();**

**PERFORM DBMS\_OUTPUT.ENABLE();**

**SELECT INTO buff,stts line,status FROM DBMS\_OUTPUT.GET\_LINE();**

**INSERT INTO dbms\_output\_test(buff,status) VALUES (buff, stts);**

**PERFORM DBMS\_OUTPUT.DISABLE();**

**PERFORM DBMS\_OUTPUT.ENABLE();**

**PERFORM DBMS\_OUTPUT.DISABLE();**

**PERFORM DBMS\_OUTPUT.PUT\_LINE ('ORAFCE TEST 2');**

**PERFORM DBMS\_OUTPUT.ENABLE();**

**SELECT INTO buff,stts line,status FROM DBMS\_OUTPUT.GET\_LINE();**

**INSERT INTO dbms\_output\_test(buff,status) VALUES (buff, stts);**

**PERFORM DBMS\_OUTPUT.DISABLE();**

**PERFORM DBMS\_OUTPUT.ENABLE();**

**PERFORM DBMS\_OUTPUT.PUT\_LINE ('ORAFCE TEST 3');**

**PERFORM DBMS\_OUTPUT.DISABLE();**

**SELECT INTO buff,stts line,status FROM DBMS\_OUTPUT.GET\_LINE();**

**INSERT INTO dbms\_output\_test(buff,status) VALUES (buff, stts);**

**PERFORM DBMS\_OUTPUT.ENABLE();**

**PERFORM DBMS\_OUTPUT.DISABLE();**

**PERFORM DBMS\_OUTPUT.ENABLE();**

**PERFORM DBMS\_OUTPUT.DISABLE();**

**PERFORM DBMS\_OUTPUT.PUT ('ORAFCE TEST 4');**

**PERFORM DBMS\_OUTPUT.ENABLE();**

**PERFORM DBMS\_OUTPUT.NEW\_LINE();**

**SELECT INTO buff,stts line,status FROM DBMS\_OUTPUT.GET\_LINE();**

**INSERT INTO dbms\_output\_test(buff,status) VALUES (buff, stts);**

**PERFORM DBMS\_OUTPUT.DISABLE();**

**PERFORM DBMS\_OUTPUT.ENABLE();**

**PERFORM DBMS\_OUTPUT.PUT ('ORAFCE TEST 5');**

**PERFORM DBMS\_OUTPUT.DISABLE();**

**PERFORM DBMS\_OUTPUT.NEW\_LINE();**

**PERFORM DBMS\_OUTPUT.ENABLE();**

**SELECT INTO buff,stts line,status FROM DBMS\_OUTPUT.GET\_LINE();**

**INSERT INTO dbms\_output\_test(buff,status) VALUES (buff, stts);**

**PERFORM DBMS\_OUTPUT.DISABLE();**

**PERFORM DBMS\_OUTPUT.ENABLE();**

**END;**

**$$ LANGUAGE plpgsql;**

**SELECT dbms\_output\_test();**

**DROP TABLE dbms\_output\_test;**

**DROP FUNCTION dbms\_output\_test();**

**-- DBMS\_OUTPUT.DISABLE [2]**

**CREATE TABLE dbms\_output\_test (id serial,buff VARCHAR(20), status INTEGER);**

**CREATE FUNCTION dbms\_output\_test() RETURNS VOID AS $$**

**DECLARE**

**buff VARCHAR(20);**

**stts INTEGER := 10;**

**BEGIN**

**PERFORM DBMS\_OUTPUT.DISABLE();**

**PERFORM DBMS\_OUTPUT.ENABLE();**

**PERFORM DBMS\_OUTPUT.SERVEROUTPUT ('f');**

**PERFORM DBMS\_OUTPUT.DISABLE();**

**PERFORM DBMS\_OUTPUT.PUT\_LINE ('ORAFCE TEST 1');**

**SELECT INTO buff,stts lines[1],numlines FROM DBMS\_OUTPUT.GET\_LINES(stts);**

**INSERT INTO dbms\_output\_test(buff,status) VALUES (buff, stts);**

**PERFORM DBMS\_OUTPUT.ENABLE();**

**END;**

**$$ LANGUAGE plpgsql;**

**SELECT dbms\_output\_test();**

**DROP TABLE dbms\_output\_test;**

**DROP FUNCTION dbms\_output\_test();**

**-- DBMS\_OUTPUT.ENABLE [1]**

**CREATE TABLE dbms\_output\_test (id serial,buff VARCHAR(20), status INTEGER);**

**CREATE FUNCTION dbms\_output\_test() RETURNS VOID AS $$**

**DECLARE**

**buff VARCHAR(20);**

**status INTEGER;**

**num INTEGER := 2000;**

**BEGIN**

**PERFORM DBMS\_OUTPUT.DISABLE();**

**PERFORM DBMS\_OUTPUT.ENABLE();**

**PERFORM DBMS\_OUTPUT.SERVEROUTPUT ('t');**

**PERFORM DBMS\_OUTPUT.ENABLE(2000);**

**PERFORM DBMS\_OUTPUT.PUT ('ORAFCE TEST 1');**

**PERFORM DBMS\_OUTPUT.NEW\_LINE();**

**PERFORM DBMS\_OUTPUT.ENABLE();**

**END;**

**$$ LANGUAGE plpgsql;**

**SELECT dbms\_output\_test();**

**DROP TABLE dbms\_output\_test;**

**DROP FUNCTION dbms\_output\_test();**

**-- DBMS\_OUTPUT.ENABLE [2]**

**CREATE TABLE dbms\_output\_test (id serial,buff VARCHAR(20), status INTEGER);**

**CREATE FUNCTION dbms\_output\_test() RETURNS VOID AS $$**

**DECLARE**

**buff VARCHAR(20);**

**stts INTEGER;**

**BEGIN**

**PERFORM DBMS\_OUTPUT.DISABLE();**

**PERFORM DBMS\_OUTPUT.ENABLE();**

**PERFORM DBMS\_OUTPUT.SERVEROUTPUT ('f');**

**PERFORM DBMS\_OUTPUT.DISABLE();**

**PERFORM DBMS\_OUTPUT.ENABLE();**

**PERFORM DBMS\_OUTPUT.PUT\_LINE ('ORAFCE TEST 1');**

**SELECT INTO buff,stts line,status FROM DBMS\_OUTPUT.GET\_LINE();**

**INSERT INTO dbms\_output\_test(buff,status) VALUES (buff, stts);**

**PERFORM DBMS\_OUTPUT.PUT ('ORAFCE TEST 2');**

**PERFORM DBMS\_OUTPUT.NEW\_LINE();**

**SELECT INTO buff,stts line,status FROM DBMS\_OUTPUT.GET\_LINE();**

**INSERT INTO dbms\_output\_test(buff,status) VALUES (buff, stts);**

**PERFORM DBMS\_OUTPUT.ENABLE();**

**END;**

**$$ LANGUAGE plpgsql;**

**SELECT dbms\_output\_test();**

**DROP TABLE dbms\_output\_test;**

**DROP FUNCTION dbms\_output\_test();**

**-- DBMS\_OUTPUT.ENABLE [3]**

**CREATE TABLE dbms\_output\_test (id serial,buff VARCHAR(20), status INTEGER);**

**CREATE FUNCTION dbms\_output\_test() RETURNS VOID AS $$**

**DECLARE**

**buff VARCHAR(20);**

**stts INTEGER := 10;**

**BEGIN**

**PERFORM DBMS\_OUTPUT.DISABLE();**

**PERFORM DBMS\_OUTPUT.ENABLE();**

**PERFORM DBMS\_OUTPUT.SERVEROUTPUT ('f');**

**PERFORM DBMS\_OUTPUT.DISABLE();**

**PERFORM DBMS\_OUTPUT.ENABLE();**

**PERFORM DBMS\_OUTPUT.PUT\_LINE ('ORAFCE TEST 1');**

**SELECT INTO buff,stts lines[1],numlines FROM DBMS\_OUTPUT.GET\_LINES(stts);**

**INSERT INTO dbms\_output\_test(buff,status) VALUES (buff, stts);**

**END;**

**$$ LANGUAGE plpgsql;**

**SELECT dbms\_output\_test();**

**DROP TABLE dbms\_output\_test;**

**DROP FUNCTION dbms\_output\_test();**

**-- DBMS\_OUTPUT.ENABLE [4]**

**CREATE FUNCTION dbms\_output\_test() RETURNS VOID AS $$**

**BEGIN**

**PERFORM DBMS\_OUTPUT.DISABLE();**

**PERFORM DBMS\_OUTPUT.ENABLE();**

**PERFORM DBMS\_OUTPUT.SERVEROUTPUT ('t');**

**PERFORM DBMS\_OUTPUT.DISABLE();**

**PERFORM DBMS\_OUTPUT.ENABLE();**

**FOR j IN 1..2000 LOOP**

**PERFORM DBMS\_OUTPUT.PUT ('A');**

**END LOOP;**

**PERFORM DBMS\_OUTPUT.NEW\_LINE();**

**END;**

**$$ LANGUAGE plpgsql;**

**SELECT dbms\_output\_test();**

**DROP FUNCTION dbms\_output\_test();**

**-- DBMS\_OUTPUT.ENABLE [5]**

**CREATE TABLE dbms\_output\_test (id serial,buff VARCHAR(20), status INTEGER);**

**CREATE FUNCTION dbms\_output\_test() RETURNS VOID AS $$**

**DECLARE**

**buff VARCHAR(20);**

**stts INTEGER;**

**BEGIN**

**PERFORM DBMS\_OUTPUT.DISABLE();**

**PERFORM DBMS\_OUTPUT.ENABLE();**

**PERFORM DBMS\_OUTPUT.SERVEROUTPUT ('f');**

**PERFORM DBMS\_OUTPUT.DISABLE();**

**PERFORM DBMS\_OUTPUT.ENABLE(NULL);**

**PERFORM DBMS\_OUTPUT.PUT\_LINE ('ORAFCE TEST 1');**

**SELECT INTO buff,stts line,status FROM DBMS\_OUTPUT.GET\_LINE();**

**INSERT INTO dbms\_output\_test(buff,status) VALUES (buff, stts);**

**PERFORM DBMS\_OUTPUT.DISABLE();**

**PERFORM DBMS\_OUTPUT.ENABLE();**

**END;**

**$$ LANGUAGE plpgsql;**

**SELECT dbms\_output\_test();**

**DROP TABLE dbms\_output\_test;**

**DROP FUNCTION dbms\_output\_test();**

**-- DBMS\_OUTPUT.ENABLE [6]**

**CREATE TABLE dbms\_output\_test (id serial,buff VARCHAR(20), status INTEGER);**

**CREATE FUNCTION dbms\_output\_test() RETURNS VOID AS $$**

**DECLARE**

**buff VARCHAR(20);**

**stts INTEGER;**

**BEGIN**

**PERFORM DBMS\_OUTPUT.DISABLE();**

**PERFORM DBMS\_OUTPUT.ENABLE();**

**PERFORM DBMS\_OUTPUT.SERVEROUTPUT ('f');**

**PERFORM DBMS\_OUTPUT.PUT\_LINE ('ORAFCE TEST 1');**

**PERFORM DBMS\_OUTPUT.ENABLE();**

**SELECT INTO buff,stts line,status FROM DBMS\_OUTPUT.GET\_LINE();**

**INSERT INTO dbms\_output\_test(buff,status) VALUES (buff, stts);**

**PERFORM DBMS\_OUTPUT.DISABLE();**

**PERFORM DBMS\_OUTPUT.ENABLE();**

**END;**

**$$ LANGUAGE plpgsql;**

**SELECT dbms\_output\_test();**

**DROP TABLE dbms\_output\_test;**

**DROP FUNCTION dbms\_output\_test();**

**-- SERVEROUTPUT [1]**

**CREATE FUNCTION dbms\_output\_test() RETURNS VOID AS $$**

**BEGIN**

**PERFORM DBMS\_OUTPUT.DISABLE();**

**PERFORM DBMS\_OUTPUT.ENABLE();**

**PERFORM DBMS\_OUTPUT.SERVEROUTPUT ('f');**

**PERFORM DBMS\_OUTPUT.PUT\_LINE ('ORAFCE TEST 1');**

**END;**

**$$ LANGUAGE plpgsql;**

**SELECT dbms\_output\_test();**

**DROP FUNCTION dbms\_output\_test();**

**CREATE FUNCTION dbms\_output\_test() RETURNS VOID AS $$**

**BEGIn**

**PERFORM DBMS\_OUTPUT.DISABLE();**

**PERFORM DBMS\_OUTPUT.ENABLE();**

**PERFORM DBMS\_OUTPUT.SERVEROUTPUT ('t');**

**PERFORM DBMS\_OUTPUT.PUT\_LINE ('ORAFCE TEST 2');**

**END;**

**$$ LANGUAGE plpgsql;**

**SELECT dbms\_output\_test();**

**DROP FUNCTION dbms\_output\_test();**

**-- SERVEROUTPUT [2]**

**CREATE FUNCTION dbms\_output\_test() RETURNS VOID AS $$**

**BEGIN**

**PERFORM DBMS\_OUTPUT.DISABLE();**

**PERFORM DBMS\_OUTPUT.ENABLE();**

**PERFORM DBMS\_OUTPUT.SERVEROUTPUT ('f');**

**PERFORM DBMS\_OUTPUT.PUT ('ORAFCE TEST 1');**

**PERFORM DBMS\_OUTPUT.NEW\_LINE();**

**END;**

**$$ LANGUAGE plpgsql;**

**SELECT dbms\_output\_test();**

**DROP FUNCTION dbms\_output\_test();**

**CREATE FUNCTION dbms\_output\_test() RETURNS VOID AS $$**

**BEGIN**

**PERFORM DBMS\_OUTPUT.DISABLE();**

**PERFORM DBMS\_OUTPUT.ENABLE();**

**PERFORM DBMS\_OUTPUT.SERVEROUTPUT ('t');**

**PERFORM DBMS\_OUTPUT.PUT ('ORAFCE TEST 2');**

**PERFORM DBMS\_OUTPUT.NEW\_LINE();**

**END;**

**$$ LANGUAGE plpgsql;**

**SELECT dbms\_output\_test();**

**DROP FUNCTION dbms\_output\_test();**

**-- SERVEROUTPUT [3]**

**CREATE FUNCTION dbms\_output\_test() RETURNS VOID AS $$**

**BEGIN**

**PERFORM DBMS\_OUTPUT.DISABLE();**

**PERFORM DBMS\_OUTPUT.ENABLE();**

**PERFORM DBMS\_OUTPUT.SERVEROUTPUT ('f');**

**PERFORM DBMS\_OUTPUT.DISABLE();**

**END;**

**$$ LANGUAGE plpgsql;**

**SELECT dbms\_output\_test();**

**DROP FUNCTION dbms\_output\_test();**

**CREATE FUNCTION dbms\_output\_test() RETURNS VOID AS $$**

**BEGIN**

**PERFORM DBMS\_OUTPUT.DISABLE();**

**PERFORM DBMS\_OUTPUT.ENABLE();**

**PERFORM DBMS\_OUTPUT.SERVEROUTPUT ('t');**

**PERFORM DBMS\_OUTPUT.PUT\_LINE ('ORAFCE TEST 1');**

**END;**

**$$ LANGUAGE plpgsql;**

**SELECT dbms\_output\_test();**

**DROP FUNCTION dbms\_output\_test();**

## DBMS\_PIPE

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

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

示例：

**create or replace procedure dbmspipe\_crtpipe\_pro(in\_cno int) as**

**declare**

**v\_no int;**

**v\_bigint bigint;**

**v\_bytea bytea;**

**v\_date date;**

**v\_int int;**

**v\_num numeric;**

**v\_text text;**

**v\_tmptz timestamp with time zone;**

**begin**

**select c,c\_bigint,c\_bytea,c\_date,c\_int,c\_num,c\_text,c\_tmptz**

**from dbmspipe\_tbl**

**where c=in\_cno**

**into v\_no,v\_bigint,v\_bytea,v\_date,v\_int,v\_num,v\_text,v\_tmptz;**

**perform dbms\_pipe.create\_pipe('bigint\_pipe1'||v\_no);**

**perform dbms\_pipe.pack\_message(v\_bigint);**

**perform dbms\_pipe.send\_message('bigint\_pipe1'||v\_no);**

**perform dbms\_pipe.create\_pipe('bigint\_pipe2'||v\_no,50);**

**perform dbms\_pipe.pack\_message(v\_bigint);**

**perform dbms\_pipe.send\_message('bigint\_pipe2'||v\_no);**

**perform dbms\_pipe.create\_pipe('bigint\_pipe3'||v\_no,100,true);**

**perform dbms\_pipe.pack\_message(v\_bigint);**

**perform dbms\_pipe.send\_message('bigint\_pipe3'||v\_no);**

**perform dbms\_pipe.create\_pipe('bigint\_pipe4'||v\_no,200,false);**

**perform dbms\_pipe.pack\_message(v\_bigint);**

**perform dbms\_pipe.send\_message('bigint\_pipe4'||v\_no);**

**perform dbms\_pipe.create\_pipe('bytea\_pipe1'||v\_no);**

**perform dbms\_pipe.pack\_message(v\_bytea);**

**perform dbms\_pipe.send\_message('bytea\_pipe1'||v\_no);**

**perform dbms\_pipe.create\_pipe('bytea\_pipe2'||v\_no,50);**

**perform dbms\_pipe.pack\_message(v\_bytea);**

**perform dbms\_pipe.send\_message('bytea\_pipe2'||v\_no);**

**perform dbms\_pipe.create\_pipe('bytea\_pipe3'||v\_no,100,true);**

**perform dbms\_pipe.pack\_message(v\_bytea);**

**perform dbms\_pipe.send\_message('bytea\_pipe3'||v\_no);**

**perform dbms\_pipe.create\_pipe('bytea\_pipe4'||v\_no,200,false);**

**perform dbms\_pipe.pack\_message(v\_bytea);**

**perform dbms\_pipe.send\_message('bytea\_pipe4'||v\_no);**

**perform dbms\_pipe.create\_pipe('date\_pipe1'||v\_no);**

**perform dbms\_pipe.pack\_message(v\_date);**

**perform dbms\_pipe.send\_message('date\_pipe1'||v\_no);**

**perform dbms\_pipe.create\_pipe('date\_pipe2'||v\_no,50);**

**perform dbms\_pipe.pack\_message(v\_date);**

**perform dbms\_pipe.send\_message('date\_pipe2'||v\_no);**

**perform dbms\_pipe.create\_pipe('date\_pipe3'||v\_no,100,true);**

**perform dbms\_pipe.pack\_message(v\_date);**

**perform dbms\_pipe.send\_message('date\_pipe3'||v\_no);**

**perform dbms\_pipe.create\_pipe('date\_pipe4'||v\_no,200,false);**

**perform dbms\_pipe.pack\_message(v\_date);**

**perform dbms\_pipe.send\_message('date\_pipe4'||v\_no);**

**perform dbms\_pipe.create\_pipe('int\_pipe1'||v\_no);**

**perform dbms\_pipe.pack\_message(v\_int);**

**perform dbms\_pipe.send\_message('int\_pipe1'||v\_no);**

**perform dbms\_pipe.create\_pipe('int\_pipe2'||v\_no,50);**

**perform dbms\_pipe.pack\_message(v\_int);**

**perform dbms\_pipe.send\_message('int\_pipe2'||v\_no);**

**perform dbms\_pipe.create\_pipe('int\_pipe3'||v\_no,100,true);**

**perform dbms\_pipe.pack\_message(v\_int);**

**perform dbms\_pipe.send\_message('int\_pipe3'||v\_no);**

**perform dbms\_pipe.create\_pipe('int\_pipe4'||v\_no,200,false);**

**perform dbms\_pipe.pack\_message(v\_int);**

**perform dbms\_pipe.send\_message('int\_pipe4'||v\_no);**

**perform dbms\_pipe.create\_pipe('num\_pipe1'||v\_no);**

**perform dbms\_pipe.pack\_message(v\_num);**

**perform dbms\_pipe.send\_message('num\_pipe1'||v\_no);**

**perform dbms\_pipe.create\_pipe('num\_pipe2'||v\_no,50);**

**perform dbms\_pipe.pack\_message(v\_num);**

**perform dbms\_pipe.send\_message('num\_pipe2'||v\_no);**

**perform dbms\_pipe.create\_pipe('num\_pipe3'||v\_no,100,true);**

**perform dbms\_pipe.pack\_message(v\_num);**

**perform dbms\_pipe.send\_message('num\_pipe3'||v\_no);**

**perform dbms\_pipe.create\_pipe('num\_pipe4'||v\_no,200,false);**

**perform dbms\_pipe.pack\_message(v\_num);**

**perform dbms\_pipe.send\_message('num\_pipe4'||v\_no);**

**perform dbms\_pipe.create\_pipe('text\_pipe1'||v\_no);**

**perform dbms\_pipe.pack\_message(v\_text);**

**perform dbms\_pipe.send\_message('text\_pipe1'||v\_no);**

**perform dbms\_pipe.create\_pipe('text\_pipe2'||v\_no,50);**

**perform dbms\_pipe.pack\_message(v\_text);**

**perform dbms\_pipe.send\_message('text\_pipe2'||v\_no);**

**perform dbms\_pipe.create\_pipe('text\_pipe3'||v\_no,100,true);**

**perform dbms\_pipe.pack\_message(v\_text);**

**perform dbms\_pipe.send\_message('text\_pipe3'||v\_no);**

**perform dbms\_pipe.create\_pipe('text\_pipe4'||v\_no,200,false);**

**perform dbms\_pipe.pack\_message(v\_text);**

**perform dbms\_pipe.send\_message('text\_pipe4'||v\_no);**

**perform dbms\_pipe.create\_pipe('tmptz\_pipe1'||v\_no);**

**perform dbms\_pipe.pack\_message(v\_tmptz);**

**perform dbms\_pipe.send\_message('tmptz\_pipe1'||v\_no);**

**perform dbms\_pipe.create\_pipe('tmptz\_pipe2'||v\_no,50);**

**perform dbms\_pipe.pack\_message(v\_tmptz);**

**perform dbms\_pipe.send\_message('tmptz\_pipe2'||v\_no);**

**perform dbms\_pipe.create\_pipe('tmptz\_pipe3'||v\_no,100,true);**

**perform dbms\_pipe.pack\_message(v\_tmptz);**

**perform dbms\_pipe.send\_message('tmptz\_pipe3'||v\_no);**

**perform dbms\_pipe.create\_pipe('tmptz\_pipe4'||v\_no,200,false);**

**perform dbms\_pipe.pack\_message(v\_tmptz);**

**perform dbms\_pipe.send\_message('tmptz\_pipe4'||v\_no);**

**end;**

**/**

**--创建接收pipe message，打印message子存储过程**

**create or replace procedure rec\_subpro(in\_pipename varchar) as**

**declare**

**v\_num numeric;**

**v\_bytea bytea;**

**v\_date date;**

**v\_str varchar;**

**v\_tmptz timestamp with time zone;**

**begin**

**perform dbms\_output.disable();**

**perform dbms\_output.enable();**

**perform dbms\_output.serveroutput ('t');**

**perform dbms\_pipe.receive\_message(in\_pipename);**

**if in\_pipename like '%int%' or in\_pipename like 'num' then**

**v\_num := dbms\_pipe.unpack\_message\_number();**

**perform dbms\_output.put\_line(in\_pipename || ' message: '||v\_num);**

**elsif in\_pipename like '%bytea%' then**

**v\_bytea := dbms\_pipe.unpack\_message\_bytea();**

**perform dbms\_output.put\_line(in\_pipename || ' message: '||v\_bytea);**

**elsif in\_pipename like '%date%' then**

**v\_date := dbms\_pipe.unpack\_message\_date();**

**perform dbms\_output.put\_line(in\_pipename || ' message: '||v\_date::text);**

**elsif in\_pipename like '%text%' then**

**v\_str := dbms\_pipe.unpack\_message\_text();**

**perform dbms\_output.put\_line(in\_pipename || ' message: '||v\_str);**

**elsif in\_pipename like '%tmptz%' then**

**v\_tmptz := dbms\_pipe.unpack\_message\_timestamp();**

**perform dbms\_output.put\_line(in\_pipename || ' message: '||v\_tmptz::text);**

**end if;**

**perform dbms\_pipe.remove\_pipe(in\_pipename);**

**end;**

**/**

**--unpack\_message\_\*测试：接收pipe中各种数据类型的message（record除外），接收后remove\_pipe，打印出message**

**create or replace procedure dbmspipe\_rec\_pro(in\_cno int) as**

**begin**

**call rec\_subpro('bigint\_pipe1'||in\_cno);**

**call rec\_subpro('bigint\_pipe2'||in\_cno);**

**call rec\_subpro('bigint\_pipe3'||in\_cno);**

**call rec\_subpro('bigint\_pipe4'||in\_cno);**

**call rec\_subpro('bytea\_pipe1'||in\_cno);**

**call rec\_subpro('bytea\_pipe2'||in\_cno);**

**call rec\_subpro('bytea\_pipe3'||in\_cno);**

**call rec\_subpro('bytea\_pipe4'||in\_cno);**

**call rec\_subpro('date\_pipe1'||in\_cno);**

**call rec\_subpro('date\_pipe2'||in\_cno);**

**call rec\_subpro('date\_pipe3'||in\_cno);**

**call rec\_subpro('date\_pipe4'||in\_cno);**

**call rec\_subpro('int\_pipe1'||in\_cno);**

**call rec\_subpro('int\_pipe2'||in\_cno);**

**call rec\_subpro('int\_pipe3'||in\_cno);**

**call rec\_subpro('int\_pipe4'||in\_cno);**

**call rec\_subpro('num\_pipe1'||in\_cno);**

**call rec\_subpro('num\_pipe2'||in\_cno);**

**call rec\_subpro('num\_pipe3'||in\_cno);**

**call rec\_subpro('num\_pipe4'||in\_cno);**

**call rec\_subpro('text\_pipe1'||in\_cno);**

**call rec\_subpro('text\_pipe2'||in\_cno);**

**call rec\_subpro('text\_pipe3'||in\_cno);**

**call rec\_subpro('text\_pipe4'||in\_cno);**

**call rec\_subpro('tmptz\_pipe1'||in\_cno);**

**call rec\_subpro('tmptz\_pipe2'||in\_cno);**

**call rec\_subpro('tmptz\_pipe3'||in\_cno);**

**call rec\_subpro('tmptz\_pipe4'||in\_cno);**

**end;**

**/**

**--sessionA:**

**call dbmspipe\_crtpipe\_pro(1);**

**--sessionB:**

**call dbmspipe\_rec\_pro(1);**

**--sessionA:**

**call dbmspipe\_crtpipe\_pro(2);**

**--sessionB:**

**call dbmspipe\_rec\_pro(2);**

**--sessionA:**

**call dbmspipe\_crtpipe\_pro(3);**

**--sessionB:**

**call dbmspipe\_rec\_pro(3);**

**--sessionA:**

**call dbmspipe\_crtpipe\_pro(4);**

**--sessionB:**

**call dbmspipe\_rec\_pro(4);**

**--sessionA:**

**call dbmspipe\_crtpipe\_pro(5);**

**--sessionB:**

**call dbmspipe\_rec\_pro(5);**

## DBMS\_UTILITY

DBMS\_UTILITY包含一个接口： FORMAT\_CALL\_STACK，这个内置函数返回一个格式化的字符串，它显示了执行调用堆栈，直至此函数的调用点处的所有过程或者函数的调用顺序。

示例：

**create or replace function dbms\_uti\_func() returns text as**

**declare**

**v\_str1 text;**

**v\_str2 text;**

**v\_str3 text;**

**v\_str4 text;**

**begin**

**v\_str1 := dbms\_utility.format\_call\_stack();**

**select regexp\_replace(v\_str1,'[ 0-9a-fA-F]{4}[0-9a-fA-F]{4}',' 0','g') into v\_str1;**

**select regexp\_replace(v\_str1,'[45()]','','g') into v\_str1;**

**v\_str2 := dbms\_utility.format\_call\_stack('o');**

**select regexp\_replace(v\_str2,'[ 0-9a-fA-F]{4}[0-9a-fA-F]{4}',' 0','g') into v\_str2;**

**select regexp\_replace(v\_str2,'[45()]','','g') into v\_str2;**

**v\_str3 := dbms\_utility.format\_call\_stack('p');**

**select regexp\_replace(v\_str3,'[ 0-9a-fA-F]{4}[0-9a-fA-F]{4}',' 0','g') into v\_str3;**

**select regexp\_replace(v\_str3,'[45()]','','g') into v\_str3;**

**v\_str4 := dbms\_utility.format\_call\_stack('s');**

**select regexp\_replace(v\_str4,'[ 0-9a-fA-F]{4}[0-9a-fA-F]{4}',' 0','g') into v\_str4;**

**select regexp\_replace(v\_str4,'[45()]','','g') into v\_str4;**

**return 'v\_str1:**

**'||v\_str1 || '**

**v\_str2:**

**'|| v\_str2|| '**

**v\_str3:**

**'|| v\_str3|| '**

**v\_str4:**

**'|| v\_str4;**

**end;**

**/**

**select dbms\_uti\_func();**

## UTL\_FILE

UTL\_FILE包提供文本文件输入和输出功能。

UTL\_FILE包含以下接口：

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

示例：

**create or replace procedure utlfile\_open\_prowa() as**

**declare**

**v\_count int;**

**v\_fileint integer;**

**begin**

**--open\_node=w**

**v\_fileint := utl\_file.fopen('/data1/tbasev5\_autotest/tbaseTest/TbaseV5/pro\_package/data/utl\_file\_dir','file\_w.txt','w');**

**perform utl\_file.put\_line(v\_fileint,'write file test.');**

**if utl\_file.is\_open(v\_fileint) then**

**perform utl\_file.fclose(v\_fileint);**

**end if;**

**v\_fileint := utl\_file.fopen('/data1/tbasev5\_autotest/tbaseTest/TbaseV5/pro\_package/data/utl\_file\_dir','file\_w.txt','w',50);**

**perform utl\_file.new\_line(v\_fileint);**

**perform utl\_file.put\_line(v\_fileint,'一二三四五六七八九十');**

**if utl\_file.is\_open(v\_fileint) then**

**perform utl\_file.fclose(v\_fileint);**

**end if;**

**v\_fileint := utl\_file.fopen('/data1/tbasev5\_autotest/tbaseTest/TbaseV5/pro\_package/data/utl\_file\_dir','file\_w.txt','w',18,'SQL\_ASCII');**

**perform utl\_file.new\_line(v\_fileint);**

**perform utl\_file.put\_line(v\_fileint,'nice to meet you');**

**if utl\_file.is\_open(v\_fileint) then**

**perform utl\_file.fclose(v\_fileint);**

**end if;**

**v\_fileint := utl\_file.fopen('/data1/tbasev5\_autotest/tbaseTest/TbaseV5/pro\_package/data/utl\_file\_dir','file\_w.txt','w',31,'UTF-8');**

**perform utl\_file.new\_line(v\_fileint);**

**perform utl\_file.put\_line(v\_fileint,'abcdefg hijklmn opq rst uvw xyz');**

**if utl\_file.is\_open(v\_fileint) then**

**perform utl\_file.fclose(v\_fileint);**

**end if;**

**v\_fileint := utl\_file.fopen('/data1/tbasev5\_autotest/tbaseTest/TbaseV5/pro\_package/data/utl\_file\_dir','file\_w.txt','w',88,'GBK');**

**perform utl\_file.new\_line(v\_fileint);**

**perform utl\_file.put\_line(v\_fileint,'hello');**

**if utl\_file.is\_open(v\_fileint) then**

**perform utl\_file.fclose(v\_fileint);**

**end if;**

**--open\_mode=a**

**v\_fileint := utl\_file.fopen('/data1/tbasev5\_autotest/tbaseTest/TbaseV5/pro\_package/data/utl\_file\_dir','file\_w.txt','a');**

**perform utl\_file.put\_line(v\_fileint,'write file test.');**

**if utl\_file.is\_open(v\_fileint) then**

**perform utl\_file.fclose(v\_fileint);**

**end if;**

**v\_fileint := utl\_file.fopen('/data1/tbasev5\_autotest/tbaseTest/TbaseV5/pro\_package/data/utl\_file\_dir','file\_w.txt','a',50);**

**perform utl\_file.new\_line(v\_fileint);**

**perform utl\_file.put\_line(v\_fileint,'一二三四五六七八九十');**

**if utl\_file.is\_open(v\_fileint) then**

**perform utl\_file.fclose(v\_fileint);**

**end if;**

**v\_fileint := utl\_file.fopen('/data1/tbasev5\_autotest/tbaseTest/TbaseV5/pro\_package/data/utl\_file\_dir','file\_w.txt','a',18,'SQL\_ASCII');**

**perform utl\_file.new\_line(v\_fileint);**

**perform utl\_file.put\_line(v\_fileint,'nice to meet you');**

**if utl\_file.is\_open(v\_fileint) then**

**perform utl\_file.fclose(v\_fileint);**

**end if;**

**v\_fileint := utl\_file.fopen('/data1/tbasev5\_autotest/tbaseTest/TbaseV5/pro\_package/data/utl\_file\_dir','file\_w.txt','a',31,'UTF-8');**

**perform utl\_file.new\_line(v\_fileint);**

**perform utl\_file.put\_line(v\_fileint,'abcdefg hijklmn opq rst uvw xyz');**

**if utl\_file.is\_open(v\_fileint) then**

**perform utl\_file.fclose(v\_fileint);**

**end if;**

**v\_fileint := utl\_file.fopen('/data1/tbasev5\_autotest/tbaseTest/TbaseV5/pro\_package/data/utl\_file\_dir','file\_w.txt','a',88,'GBK');**

**perform utl\_file.new\_line(v\_fileint);**

**perform utl\_file.put\_line(v\_fileint,'hello');**

**if utl\_file.is\_open(v\_fileint) then**

**perform utl\_file.fclose(v\_fileint);**

**end if;**

**exception**

**when others then**

**raise notice 'EXP: something wrong.';**

**end;**

**/**

**call utlfile\_open\_prowa();**

# 内置视图

## USER\_TAB\_COLUMNS

USER\_TAB\_COLUMNS 视图提供当前用户拥有的表和视图中的所有列的信息。

| **名称** | **类型** | **说明** |
| --- | --- | --- |
| schema\_name | CHARACTER VARYING | 表或视图所在 schema 的名称。 |
| table\_name | CHARACTER VARYING | 列所在的表或视图的名称。 |
| column\_name | CHARACTER VARYING | 列的名称。 |
| data\_type | CHARACTER VARYING | 列的数据类型。 |
| data\_length | NUMERIC | 文本列的长度。 |
| data\_precision | NUMERIC | NUMBER 列的精度（位数）。 |
| data\_scale | NUMERIC | NUMBER 列的小数位数。 |
| nullable | CHARACTER(1) | 列是否可为空，可能值包括：   * Y：列可为空。 * N：列不可为空。 |
| column\_id | NUMERIC | 表中列的相对位置。 |
| data\_default | CHARACTER VARYING | 分配给列的默认值。 |

## ALL\_COL\_COMMENTS

ALL\_COL\_COMMENTS在表列和当前用户可访问的视图上显示注释。

| **名称** | **类型** | **说明** |
| --- | --- | --- |
| OWNER | VARCHAR2(30) | 所有者的名称。 |
| TABLE\_NAME | VARCHAR2(30) | 对象名。 |
| COLUMN\_NAME | VARCHAR2(30) | 列名。 |
| COMMENT | VARCHAR2(4000) | 列注释。 |

## USER\_COL\_COMMENTS

USER\_COL\_COMMENTS在当前用户拥有的表和视图的列上显示注释。其列（OWNER除外）与ALL\_COL\_COMMENTS中的列相同。

## USER\_CONSTRAINTS

USER\_CONSTRAINTS 视图提供当前用户拥有的表中放置的所有约束的信息。

| **名称** | **类型** | **说明** |
| --- | --- | --- |
| owner | TEXT | 约束所有者的名称。 |
| schema\_name | TEXT | 约束所属 schema 的名称。 |
| constraint\_name | TEXT | 约束的名称。 |
| constraint\_type | TEXT | 约束类型。可能值包括：   * C：检查约束 * F：外键约束 * P：主键约束 * U：唯一键约束 * R：引用完整性约束 * V：视图上的约束 * O：具有只读属性，在视图上 |
| table\_name | TEXT | 约束所属表的名称。 |
| search\_condition | TEXT | 应用于检查约束的搜索条件。 |
| r\_owner | TEXT | 引用约束引用的表的所有者。 |
| r\_constraint\_name | TEXT | 引用表的约束定义的名称。 |
| delete\_rule | TEXT | 引用约束的删除规则。可能值包括：   * C：级联 * R：限制 * N：无操作 |
| deferrable | BOOLEAN | 指定了约束是否可延迟（T 或 F）。 |
| deferred | BOOLEAN | 指定约束是否已延迟（T 或 F）。 |
| index\_owner | TEXT | 索引所有者的用户名。 |
| index\_name | TEXT | 索引的名称。 |
| constraint\_def | TEXT | 约束的定义。 |

## USER\_CONS\_COLUMNS

USER\_CONS\_COLUMNS 视图提供当前用户拥有的表中，约束包含的所有列的信息。

| **名称** | **类型** | **说明** |
| --- | --- | --- |
| owner | TEXT | 约束所有者的用户名。 |
| schema\_name | TEXT | 约束所属 schema 的名称。 |
| constraint\_name | TEXT | 约束的名称。 |
| table\_name | TEXT | 约束所属表的名称。 |
| column\_name | TEXT | 约束中引用的列的名称。 |
| position | SMALLINT | 列在对象定义中的位置。 |
| constraint\_def | TEXT | 约束的定义。 |

# 函数

## 数学

### MOD(character varying,integer)

MOD为取余函数，MOD(m,n)，其中m，n都应是数字，同时m支持字符串类型。

语法

**MOD(character varying,integer)**

示例

**postgres=# select mod('10',3) from dual;**

**mod**

**-----**

**1**

**(1 row)**

### SUM(character varying)

SUM为求和函数，SUM(m)，其中m都应是数字，同时m支持字符串类型。

语法

**SUM(character varying)**

示例

**postgres=# create table t\_sum(id int,info varchar(15));**

**CREATE TABLE**

**postgres=# insert into t\_sum select generate\_series(1,100),generate\_series(100,200);**

**INSERT 0 100**

**postgres=# select sum(id),sum(info) from t\_sum;**

**sum | sum**

**------+-------**

**5050 | 15150**

**(1 row)**

### LISTAGG(numeric,text)

LISTAGG聚合查询函数，可实现列转行，通常第一个参数为字符串类型，同时也支持数字。

语法

**LISTAGG( column\_name , text ) WITHIN GROUP( ORDER BY column\_name [, ... { DESC| ASC } ] )。**

示例

支持第一个参数为数字。

**postgres=# with t as (**

**select 1 grp\_no,'B' c1 from dual union all**

**select 1 grp\_no,'A' c1 from dual union all**

**select 1 grp\_no,'C' c1 from dual union all**

**select 2 grp\_no,'E' c1 from dual union all**

**select 2 grp\_no,'D' c1 from dual )**

**select grp\_no,listagg(grp\_no,'->') as t1**

**from t**

**group by grp\_no;**

**grp\_no | t1**

**--------+---------**

**2 | 2->2**

**1 | 1->1->1**

**(2 rows)**

同时支持第一个参数为字符串。

**postgres=# with t as (**

**select 1 grp\_no,'B' c1 from dual union all**

**select 1 grp\_no,'A' c1 from dual union all**

**select 1 grp\_no,'C' c1 from dual union all**

**select 2 grp\_no,'E' c1 from dual union all**

**select 2 grp\_no,'D' c1 from dual )**

**select grp\_no,listagg(c1,'->') as t1**

**from t**

**group by grp\_no;**

**grp\_no | t1**

**--------+---------**

**2 | E->D**

**1 | B->A->C**

**(2 rows)**

## 日期时间

### ADD\_MONTHS(timestamp,text)

ADD\_MONTHS(x, y)函数用于计算某个日期向前或者向后y个月后的时间，通常第2个参数为数字，同时支持字符串。

语法

**ADD\_MONTHS(timestamp, text)**

示例

**postgres=# select sysdate, add\_months(sysdate, '2'), add\_months(sysdate, '-2') from dual;**

**orcl\_sysdate | add\_months | add\_months**

**---------------------+---------------------+---------------------**

**2021-03-19 22:42:20 | 2021-05-19 22:42:20 | 2021-01-19 22:42:20**

**(1 row)**

**postgres=# select sysdate, add\_months(sysdate, 2), add\_months(sysdate, -2) from dual;**

**orcl\_sysdate | add\_months | add\_months**

**---------------------+---------------------+---------------------**

**2021-03-19 22:42:49 | 2021-05-19 22:42:49 | 2021-01-19 22:42:49**

**(1 row)**

### NUMTODSINTERVAL

NUMTODSINTERVAL时间计算函数2个参数第1个通常为数字，第2个参数为单位，可选有'day','hour','minute','second' （忽略大小写）。

语法

**NUMTODSINTERVAL({ string | numeric } , { 'DAY' | 'HOUR' | 'MINUTE' | 'SECOND' })**

示例

**postgres=# select sysdate,sysdate+numtodsinterval(3,'hour') as res from dual;**

**orcl\_sysdate | res**

**---------------------+---------------------**

**2021-03-19 22:47:33 | 2021-03-20 01:47:33**

**(1 row)**

NUMTOYMINTERVAL

NUMTOYMINTERVAL时间计算函数2个参数第1个通常为数字，第2个参数为单位，可选有'year','month'（忽略大小写）。

语法

**NUMTOYMINTERVAL({ string | numeric } , { 'YEAR' | 'MONTH' })**

示例

**postgres=# select sysdate,sysdate+numtoyminterval(3,'month') as res from dual;**

**orcl\_sysdate | res**

**---------------------+---------------------**

**2021-03-19 22:58:35 | 2021-06-19 22:58:35**

**(1 row)**

### DBTIMEZONE

返回数据库时区。

示例

**postgres=# select DBTIMEZONE from dual;**

**dbtimezone**

**------------**

**08:00:00**

**(1 row)**

### SYSDATE

返回数据库当前时间。

示例

**postgres=# select sysdate from dual;**

**orcl\_sysdate**

**---------------------**

**2021-03-19 23:04:08**

**(1 row)**

### MONTHS\_BETWEEN

返回两个日期之间的月份数。

在ORACLE中计算月份的差值，都是月底或是日期相同，都会按整月计算。

如果计算结果不是整数，ORACLE会把31作为分母来计算。

使用限制

1、Oracle和TBase该函数实现相同，但精度不同，Oracle有效数字为9位，TBase为15位。

语法

**MONTHS\_BETWEEN( timestamp, timestamp)**

示例

**postgres=# select months\_between(to\_date('20210331', 'yyyymmdd'), to\_date('20200131', 'yyyymmdd')) as months from dual;**

**months**

**--------**

**14**

**(1 row)**

**postgres=# select months\_between(to\_date('20200331', 'yyyymmdd'), to\_date('20200130', 'yyyymmdd')) as months from dual;**

**months**

**------------------**

**2.03225806451613**

**(1 row)**

**postgres=# select months\_between(to\_date('20200301', 'yyyymmdd'), to\_date('20200130', 'yyyymmdd')) as months from dual;**

**months**

**------------------**

**1.06451612903226**

**(1 row)**

### EXTRACT

Oracle中EXTRACT()函数从oracle 9i中引入的,主要作用于一个date或者interval类型中截取特定的部分。

TBase V5兼容extract函数用法。

语法

**EXTRACT ({ YEAR | MONTH | DAY | HOUR | MINUTE | SECOND | TIMEZONE\_HOUR | TIMEZONE\_MINUTE | }**

**FROM { date\_value\_expession | interval\_value\_expression } )**

使用限制

1. 暂不支持Oracle用法中的参数 timezone\_region、timezone\_abbr；
2. 暂不支持Oracle用法中的 日期计算，如：

**select extract(day from dt2-dt1) day**

**,extract(hour from dt2-dt1) hour**

**,extract(minute from dt2-dt1) minute**

**,extract(second from dt2-dt1) second**

**from (**

**select to\_timestamp('2011-02-04 15:07:00','yyyy-mm-dd hh24:mi:ss') dt1**

**,to\_timestamp('2011-05-17 19:08:46','yyyy-mm-dd hh24:mi:ss') dt2**

**from dual);**

示例

1. 从一个date类型中截取年月日

**postgres=# select extract(year from sysdate) year,extract(month from sysdate) month,extract(day from sysdate) day from dual;**

**year | month | day**

**------+-------+-----**

**2021 | 3 | 24**

**(1 row)**

**postgres=# select extract(month from date '2021-03-24') month from dual;**

**month**

**-------**

**3**

**(1 row)**

1. 获取interval类型特定部分

**postgres=# select extract(year from interval '21' year) year from dual;**

**year**

**------**

**21**

**(1 row)**

3）获取timezone类型中特定部分

**postgres=# select SYSTIMESTAMP,EXTRACT(TIMEZONE\_HOUR FROM SYSTIMESTAMP) hour,EXTRACT(TIMEZONE\_MINUTE FROM SYSTIMESTAMP) minu from dual;**

**orcl\_systimestamp | hour | minu**

**-------------------------------+------+------**

**2021-03-24 18:29:52.199622+08 | 8 | 0**

**(1 row)**

## 字符串

### NCHR

语法：

**NCHR**(number)

NCHR返回具有与国家字符集中的数字等效的二进制字符。返回的值始终为NVARCHAR2。此函数等效于将CHR函数与USING NCHAR\_CS子句一起使用。

示例：

**postgres=# select NCHR(116) from dual;**

**nchr**

**------**

**t**

**(1 row)**

**postgres=# select NCHR(84);**

**nchr**

**------**

**T**

**(1 row)**

### NLS\_UPPER

NLS\_UPPER将字符串转换成大写，并可指定排序方式为拼音、部首或笔画。

语法：

**NLS\_UPPER( x [, y ] )**

其中：

x：字符型表达式

y：可选，指定排序的方式(nls\_sort=) 。

SCHINESE\_RADICAL\_M（部首、笔画）

SCHINESE\_STROKE\_M（笔画、部首)

SCHINESE\_PINYIN\_M（拼音）

示例

**postgres=# with tmp as**

**(**

**select 1 id,'abc' info**

**union all**

**select 2 id,'Acd' info**

**union all**

**select 3 id,'BDe' info**

**)**

**select id,NLS\_UPPER(info,'nls\_sort= SCHINESE\_STROKE\_M')**

**from tmp**

**order by NLS\_UPPER(info,'nls\_sort= SCHINESE\_STROKE\_M');**

**id | nls\_upper**

**----+-----------**

**1 | ABC**

**2 | ACD**

**3 | BDE**

**(3 rows)**

### NLSSORT

语法：

**NLSSORT(char [, 'nlsparam' ])**

用于对中文按拼音、笔画、部首进行排序。

通过设置NSL\_SORT值来实现:

SCHINESE\_RADICAL\_M 按照部首（第一顺序）、笔划（第二顺序）排序

SCHINESE\_STROKE\_M 按照笔划（第一顺序）、部首（第二顺序）排序

SCHINESE\_PINYIN\_M 按照拼音排序

示例：

**CREATE TABLE test (name VARCHAR2(15));**

**insert into test values('中');**

**insert into test values('国');**

**insert into test values('你');**

**insert into test values('好');**

**select \* from test order by nlssort(name,'NLS\_SORT=SCHINESE\_PINYIN\_M');**

**select \* from test order by nlssort(name,'NLS\_SORT=SCHINESE\_STROKE\_M');**

**select \* from test order by nlssort(name,'NLS\_SORT=SCHINESE\_RADICAL\_M');**

### REGEXP\_REPLACE

语法：

**REGEXP\_REPLACE(source\_char, pattern**

**[, replace\_string**

**[, position**

**[, occurrence**

**[, match\_parameter ]**

**]**

**]**

**]**

**)**

regexp\_replace(源字符串，正则表达式，替换的字符，从第几个字符开始匹配（默认为1），第几个匹配组（默认为全部），取值范围）

示例：

**select regexp\_replace('0123045','0','A') from dual;**

**select regexp\_replace('0123045','0','A',2) from dual;**

### REGEXP\_SUBSTR

语法：

**REGEXP\_SUBSTR(source\_string, pattern**

**[, position**

**[, occurrence**

**[, match\_parameter ]**

**]**

**]**

**)**

srcstr：需要进行正则处理的字符串

pattern：进行匹配的正则表达式

position：起始位置，从第几个字符开始正则表达式匹配（默认为1）

occurrence ：标识第几个匹配组，默认为1

modifier：模式（'i'不区分大小写进行检索；'c'区分大小写进行检索。默认为'c'。）

示例：

**select REGEXP\_SUBSTR('as12b','b',1) from dual;**

### INSTR

语法：

**INSTR (string , substring [, position [, occurrence ] ])**

string：源字符串

substring：目标字符串

position：起始位置

occurrence：匹配序号

在string中搜索substring出现的位置。

示例：

**SELECT INSTR('CORPORATE FLOOR','OR', 3, 2) FROM DUAL;**

### LENGTH、LENGTHB

语法：

**{ LENGTH| LENGTHB}(char)**

LENGTH函数返回char的长度。 LENGTH使用输入字符集定义的字符来计算长度。 LENGTHB使用字节而不是字符。

示例：

**select length('CAND IDE') from dual;**

## NULL相关

NVL函数的功能是实现空值的转换，根据第一个表达式的值是否为空值来返回响应的列名或表达式，主要用于对数据列上的空值进行处理，语法格式如：NVL( string1, replace\_with)

如果第一个参数的值为空值，则返回第二个参数的值，否则返回第一个参数的值。如果两个参数的值都为空值，则返回空值。第一个参数和第二个参数可以是任何类型的数据，但两个参数的数据类型必须相同（或能够由隐式转换为相同的类型）。

语法

NVL( string1, replace\_with)

第一个参数和第二个参数可以是任何类型的数据，但两个参数的数据类型必须相同（或能够由隐式转换为相同的类型）。

### NVL(character varying, integer)

支持NVL函数参数为(character varying, integer)。

示例

**postgres=# select nvl('abc'::varchar, 1);**

**nvl**

**-----**

**abc**

**(1 row)**

**postgres=# select nvl('abc'::varchar2, 1);**

**nvl**

**-----**

**abc**

**(1 row)**

**postgres=# select nvl('abc'::text, 1);**

**nvl**

**-----**

**abc**

**(1 row)**

**postgres=# select nvl(null::text, 1);**

**nvl**

**-----**

**1**

**(1 row)**

### NVL(numeric, integer)

支持NVL函数参数为(numeric, integer)。

示例

**postgres=# select nvl(12::numeric,''::integer);**

**nvl**

**-----**

**12**

**(1 row)**

**postgres=# select nvl(12::numeric,'123'::integer);**

**nvl**

**-----**

**12**

**(1 row)**

**postgres=# select nvl(null::numeric,''::integer);**

**nvl**

**-----**

**(1 row)**

**postgres=# select nvl(null::numeric,'123'::integer);**

**nvl**

**-----**

**123**

**(1 row)**

### NVL(bigint, int)

支持NVL函数参数为(bigint, int)。

示例

**postgres=# select nvl(123::bigint,12::int);**

**nvl**

**-----**

**123**

**(1 row)**

**postgres=# select nvl(''::bigint,12::int);**

**nvl**

**-----**

**12**

**(1 row)**

**postgres=# select nvl(null::bigint,12::int);**

**nvl**

**-----**

**12**

**(1 row)**

### NVL(numeric,'')

支持NVL函数参数为(numeric,'')。

示例

**postgres=# select nvl(123.45, '') from dual;**

**nvl**

**--------**

**123.45**

**(1 row)**

**postgres=# select nvl(12::numeric,'');**

**nvl**

**-----**

**12**

**(1 row)**

**postgres=# select nvl(12::numeric,'123');**

**nvl**

**-----**

**12**

**(1 row)**

**postgres=# select nvl(null::numeric,'');**

**nvl**

**-----**

**(1 row)**

**postgres=# select nvl(null::numeric,'123');**

**nvl**

**-----**

**123**

**(1 row)**

### LNNVL

LNNVL用于某个语句的where子句中的条件，如果条件为true就返回false；如果条件为UNKNOWN或者false就返回true。该函数不能用于复合条件如AND, OR, or BETWEEN中。

语法

**LNNVL( expression )**

示例

**postgres=# create table t\_lnnvl(f1 integer, f2 integer);**

**CREATE TABLE**

**postgres=# insert into t\_lnnvl values(1,1),(1,2),(1,3),(1,4);**

**COPY 4**

**postgres=# insert into t\_lnnvl values(1,null);**

**INSERT 0 1**

**postgres=# select \* from t\_lnnvl where lnnvl(f2>2);**

**f1 | f2**

**----+----**

**1 | 1**

**1 | 2**

**1 |**

**(3 rows)**

## 编码与解码

DECODE函数是ORACLE PL/SQL的功能强大的函数之一，用于实现if–then–else 逻辑判断。

语法

DECODE(value, if1, then1, if2,then2, if3,then3, . . . else )

Value 代表某个表的任何类型的任意列或一个通过计算所得的任何结果。当每个value值被测试，如果value的值为if1，decode 函数的结果是then1；如果value等于if2，decode函数结果是then2；等等。事实上，可以给出多个if/then 配对。如果value结果不等于给出的任何配对时，decode 结果就返回else。

这里的if、then及else 都可以是函数或计算表达式。

在此基础上，补充和完善了以下decode函数用法。

### DECODE(numeric, integer, text, numeric)

支持参数为numeric, integer, text, numeric的decode函数调用。

示例

**postgres=# select decode(1::numeric,2::integer,3::text,10::numeric) from dual;**

**decode**

**--------**

**10**

(1 row)

### DECODE(bigint, integer, integer, integer)

支持参数为bigint, integer, integer, integer的decode函数调用。

示例

**postgres=# select decode(1::bigint,2::integer,3::integer,10::integer) from dual;**

**decode**

**--------**

**10**

**(1 row)**

### DECODE(bigint, integer, integer, integer, integer)

支持参数为bigint, integer, integer, integer, integer的decode函数调用。

示例

**postgres=# select decode(1::bigint,2::integer,3::integer,4::integer, 10::integer) from dual;**

**decode**

**--------**

**(1 row)**

**postgres=# select decode(1::bigint,2::integer,3::integer,1::integer, 10::integer) from dual;**

**decode**

**--------**

**10**

**(1 row)**

### DECODE(bigint, bigint, bigint, double precision)

支持参数为bigint, bigint, bigint, double precision的decode函数调用。

示例

**postgres=# select decode(1::bigint,2::bigint,3::bigint,10::double precision) from dual;**

**decode**

**--------**

**10**

**(1 row)**

### DECODE(bigint, integer, integer, double precision)

支持参数为bigint, integer, integer, double precision的decode函数调用。

示例

**postgres=# select decode(1::bigint,2::integer,3::integer,10::double precision) from dual;**

**decode**

**--------**

**10**

**(1 row)**

### DECODE(numeric, bigint, text, bigint, text)

支持参数为numeric, bigint, text, bigint, text的decode函数调用。

示例

**postgres=# select decode(1::numeric,2::bigint,3::text,4::bigint,10::text) from dual;**

**decode**

**--------**

**(1 row)**

**postgres=# select decode(1::numeric,2::bigint,3::text,1::bigint,10::text) from dual;**

**decode**

**--------**

**10**

**(1 row)**

### DECODE(0,0,0,0/0)

支持传入参数为0,0,0,0/0的用法。

示例

**postgres=# select decode(0,0,0,0/0) from dual;**

**decode**

**--------**

**0**

**(1 row)**

### DUMP

dump()函数可以返回一个varchar值，这个值包含了数据类型代码、字节长度和表达式的内部表示形式。

语法

**DUMP(expr[,return\_fmt[,start\_position][,length]])**

其中：

expr：这个参数是要进行分析的表达式（数字或字符串等，可以是各个类型的值）；

return\_fmt：指返回参数的格式，有5种用法：

1）8：以8进制返回结果的值

2）10：以10进制返回结果的值（默认）

3）16：以16进制返回结果的值

4）17：以单字符的形式返回结果的值

5）1000：以上4种加上1000，表示在返回值中加上当前字符集

1008：带字符集名称的八进制符号

1010：带字符集名称的十进制符号

1016：带字符集名称的十六进制符号

1017：带字符集名称的单个字符

start\_position：开始进行返回的字符位置；

length：需要返回的字符长度。

返回结果格式如：Typ=96 Len=3 [CharacterSet=ZHS16GBK]: 61,62,63

1. typ表示当前的expr值的类型，类型查询sql：select typname from pg\_type where oid=?；

Oracle与TBase中类型对比如下：

**Oracle Type TBase Type**

**CODE TYP CODE TYP**

**----- ----------------------- ----- ----------------**

**1 VARCHAR2 781 varchar2**

**2 NUMBER 1700 number**

**8 LONG 不支持**

**12 DATE 1082 date**

**23 RAW 不支持**

**24 LONG RAW 不支持**

**69 ROWID 4657 rowid**

**96 CHAR 1042 bpchar**

**112 CLOB 11905 clob**

**113 BLOB 11904 blob**

**114 BFILE 不支持**

**180 TIMESTAMP 1114 timestamp**

**181 TIMESTAMP WITH TIMEZONE 1184 timestamptz**

**182 INTERVAL YEAR TO MONTH 1186 interval**

**183 INTERVAL DAY TO SECOND 1186 interval**

**208 UROWID 不支持**

**231 TIMESTAMP WITH LOCAL TIMEZONE 不支持**

2、Len表示占用的字节数；这里字节数和Oracle的可能会有差异；

3、具体的存储值。返回的数值就是TBase在自己内部的存储形式。对于非汉字的普通字符串，可以理解为就是它的ASCII码。

示例

**postgres=# select dump('abc') from dual;**

**dump**

**---------------------------------**

**Typ=25 Len=7: 28,0,0,0,97,98,99**

**(1 row)**

**postgres=# select dump('abc',16) from dual;**

**dump**

**---------------------------------**

**Typ=25 Len=7: 1c,0,0,0,61,62,63**

**(1 row)**

**postgres=# select dump('abc',1016) from dual;**

**dump**

**---------------------------------------------------**

**Typ=25 Len=7 CharacterSet=UTF8: 1c,0,0,0,61,62,63**

**(1 row)**

**postgres=# select dump('abc',17,2,2) from dual;**

**dump**

**-------------------**

**Typ=25 Len=7: ?,?**

**(1 row)**

**postgres=# select dump('a=?5') from dual;**

**dump**

**------------------------------------**

**Typ=25 Len=8: 32,0,0,0,97,61,63,53**

**(1 row)**

**postgres=# select chr(97),chr(61),chr(63),chr(53) from dual;**

**chr | chr | chr | chr**

**-----+-----+-----+-----**

**a | = | ? | 5**

**(1 row)**

**postgres=# select ascii('a'),ascii('='),ascii('?'),ascii('5') from dual;**

**ascii | ascii | ascii | ascii**

**-------+-------+-------+-------**

**97 | 61 | 63 | 53**

**(1 row)**

## XML

### EXTRACTVALUE

EXTRACTVALUE函数将XMLType实例和XPath表达式作为参数，并返回结果节点的标量值。

语法：

**EXTRACTVALUE**

**(XMLType\_instance, XPath\_string**

**[, namespace\_string ]**

**)**

示例：

**create table temp (**

**name varchar2(200),**

**age number,**

**temp\_clob CLOB**

**);**

**insert into temp values('aaa',32,'<AAA>**

**<BBB id = "b1"/>**

**<BBB id = "b2">**

**hello**

**</BBB>**

**<BBB name = "bbb"/>**

**<BBB/>**

**</AAA>');**

**select extractvalue(xmltype(a.temp\_clob),'/AAA/BBB[@id="b2"]') from temp a;**

## 转换

### TO\_CLOB

TBase V5兼容Oracle的TO\_CLOB函数，将字符串、数字转换为clob类型。

语法

**TO\_CLOB( { string | numeric | clob })**

示例

**postgres=# select to\_clob('tbase'),dump(to\_clob('tbase')) from dual;**

**to\_clob | dump**

**---------+---------------------------------------------**

**tbase | Typ=11899 Len=9: 36,0,0,0,116,98,97,115,101**

**(1 row)**

**postgres=# select to\_clob(112233),dump(to\_clob(112233)) from dual;**

**to\_clob | dump**

**---------+----------------------------------------------**

**112233 | Typ=11899 Len=10: 40,0,0,0,49,49,50,50,51,51**

**(1 row)**

**postgres=# select typname from pg\_type where oid=11899;**

**typname**

**---------**

**clob**

**(1 row)**

确认函数输出类型11899对应clob。

### ROWIDTOCHAR

Oracle中ROWIDTOCHAR函数转换rowid为varchar2类型。TBase V5兼容该函数，返回类型为text。

使用限制：

1、参数必须为rowid类型，或符合rowid要求的varchar、varchar2、text类型。

语法

**ROWIDTOCHAR(rowid)**

示例

**postgres=# select rowid,ROWIDTOCHAR(rowid),dump(ROWIDTOCHAR(rowid)) from t\_rowid;**

**rowid | rowidtochar | dump**

**----------------------+----------------------+----------------------------------------------------------------------------------------**

**XPK3fw==AQAAAAAAAAA= | XPK3fw==AQAAAAAAAAA= | Typ=25 Len=24: 96,0,0,0,88,80,75,51,102,119,61,61,65,81,65,65,65,65,65,65,65,65,65,61**

**XPK3fw==AgAAAAAAAAA= | XPK3fw==AgAAAAAAAAA= | Typ=25 Len=24: 96,0,0,0,88,80,75,51,102,119,61,61,65,103,65,65,65,65,65,65,65,65,65,61**

**(2 rows)**

**postgres=# select typname from pg\_type where oid=25;**

**typname**

**---------**

**text**

**(1 row)**

确认函数输出类型25对应text。

入参为字符串也可以。

**postgres=# select ROWIDTOCHAR('XPK3fw==AQAAAAAAAAA=');**

**rowidtochar**

**----------------------**

**XPK3fw==AQAAAAAAAAA=**

**(1 row)**

### CHARTOROWID

Oracle中CHARTOROWID函数转换varchar2类型为rowid值,c1,字符串，长度为18的字符串，字符串必须符合rowid格式，ROWID：数据表中行的全局唯一地址,ROWID详解

<https://blog.csdn.net/chiclewu/article/details/25782619>

TBase V5中 rowid长度为20位。

使用限制

1、参数为字符串，用长度必须为20，字符只能为数字、大小写字母、=组合。

语法

**CHARTOROWID( { varchar | text })**

示例

测试表t\_rowid中查看已有rowid：

**postgres=# select rowid,\* from t\_rowid;**

**rowid | f1 | f2**

**----------------------+----+------**

**XPK3fw==AQAAAAAAAAA= | 1 | ONE**

**XPK3fw==AgAAAAAAAAA= | 2 | TWO1**

**(2 rows)**

确认CHARTOROWID函数调用，及函数输出类型

**postgres=# select CHARTOROWID('XPK3fw==AQAAAAAAAAA='),**

**dump(CHARTOROWID('XPK3fw==AQAAAAAAAAA=')) from dual;;**

**chartorowid | dump**

**----------------------+-------------------------------------------------**

**XPK3fw==AQAAAAAAAAA= | Typ=4657 Len=12: 1,0,0,0,0,0,0,0,92,242,183,127**

**(1 row)**

**postgres=# select typname from pg\_type where oid=4657;**

**typname**

**---------**

**rowid**

**(1 row)**

**postgres=#**

类型4657对应rowid。

### TO\_CHAR(十进制，'xxx') 转换为16进制

TO\_CHAR通常用来将日期、数字转换为字符串，但Oracle中还有一种用途，可以用来将十进制转换为16进制。

使用限制

1、需要开启Oracle兼容性，即enable\_oracle\_compatible为on。

语法

**TO\_CHAR( {numeric | string } , 'xxx' )**

示例

**postgres=# select to\_char('255','xxx') from dual;**

**to\_char**

**---------**

**ff**

**(1 row)**

**postgres=# select to\_char(255,'xxx') from dual;**

**to\_char**

**---------**

**ff**

**(1 row)**

**postgres=# select ltrim(to\_char(to\_char(sysdate,'yyyy'),'xxxx')) FROM dual;**

**ltrim**

**-------**

**7e5**

**(1 row)**

包含字符不符合报错：

**postgres=# select CHARTOROWID('XPK3fw==AQAAAAAAAA-=') from dual;;**

**ERROR: invalid symbol "-" while decoding base64 sequence**

长度不符合报错：

**postgres=# select CHARTOROWID('XPK3fw==AQAAAAAAAAa==') from dual;;**

**ERROR: invalid argument string length**

## 大对象

### EMPTY\_BLOB

TBase V5兼容Oracle的EMPTY\_BLOB函数，用于初始化BLOB字段，然后再使用输出流向字段中写数据。

语法

**EMPTY\_BLOB()**

示例

**postgres=# create table t\_empty\_blob (f1 int,f2 blob);**

**CREATE TABLE**

**postgres=# insert into t\_empty\_blob values (1,empty\_blob());**

**INSERT 0 1**

**postgres=# update t\_empty\_blob set f2='TBase V5' where f1=1;**

**UPDATE 1**

**postgres=# select \* from t\_empty\_blob ;**

**f1 | f2**

**----+----------**

**1 | \x5442617365205635**

**(1 row)**

### EMPTY\_CLOB

TBase V5兼容Oracle的EMPTY\_CLOB函数，用于初始化CLOB字段，然后再使用输出流向字段中写数据。

语法

**EMPTY\_CLOB()**

示例

**postgres=# create table t\_empty\_clob (f1 int,f2 clob);**

**CREATE TABLE**

**postgres=# insert into t\_empty\_clob values (1,empty\_clob());**

**INSERT 0 1**

**postgres=# update t\_empty\_clob set f2='TBase V5' where f1=1;**

**UPDATE 1**

**postgres=# select \* from t\_empty\_clob ;**

**f1 | f2**

**----+----------**

**1 | TBase V5**

**(1 row)**

# 存储过程

## 基本结构

语法：

**CREATE [ OR REPLACE ] PROCEDURE [ schema. ]procedure**

**[ (argument [ { IN | OUT | IN OUT } ]**

**[ NOCOPY ]**

**datatype [ DEFAULT expr ]**

**[, argument [ { IN | OUT | IN OUT } ]**

**[ NOCOPY ]**

**datatype [ DEFAULT expr ]**

**]...**

**)**

**]**

**[ invoker\_rights\_clause ]**

**{ IS | AS }**

**{ pl/sql\_subprogram\_body | call\_spec } ;**

示例：

**create or replace procedure exe\_imed\_pro() as**

**begin**

**perform dbms\_output.put\_line('tbase');**

**end;**

call exe\_imed\_pro();

参数：（TBase还不支持 OUT 和 IN OUT参数）

* IN：指定IN表示在调用过程时必须为参数提供一个值。
* OUT：指定OUT以指示该过程在执行后将该参数的值传递回其调用环境。
* IN OUT：指定IN OUT表示在调用过程时必须为参数提供一个值，并且该过程在执行后将值传递回其调用环境。
* 如果省略IN，OUT和IN OUT，则参数默认为IN。

示例：

**create or replace procedure demo4(a\_int in integer)**

**as**

**begin**

**a\_int:=1;**

**end;**

**/**

## 变量声明

### %TYPE

语法：

**{ collection\_name**

**| cursor\_variable\_name**

**| object name**

**| record\_name**

**| record\_name . field\_name**

**| db\_table\_name . column\_name**

**| variable\_name**

**}**

**%TYPE**

**示例：**

**create table t4 (f1 integer,f2 varchar2(10));**

**insert into t4 values(1,'tbase');**

**commit;**

**DECLARE**

**v\_t4\_f2 t4.f2%type;**

**begin**

**select f2 into v\_t4\_f2 from t4 where f1=1;**

**perform dbms\_output.put\_line('v\_t4\_f2='||v\_t4\_f2);**

**end;**

**/**

### 在记录声明中使用%ROWTYPE

示例：

**set search\_path to test\_oracle;**

**DROP FUNCTION catcitygdp;**

**CREATE FUNCTION catcitygdp(nation\_name text) RETURNS text AS $$**

**DECLARE**

**TYPE mycitytype IS RECORD(gdp test\_oracle.tbl\_city.gdp%type, city test\_oracle.tbl\_city.city%type);**

**trow tbl\_city%ROWTYPE;**

**rlt text;**

**lt mycitytype;**

**BEGIN**

**rlt = ':' || nation\_name;**

**FOR trow IN SELECT \* FROM tbl\_city WHERE nation = nation\_name**

**LOOP**

**lt.gdp = trow.gdp;**

**lt.city = trow.city;**

**rlt = trow.city ||'+' ||lt.gdp::text|| rlt;**

**END LOOP;**

**return rlt;**

**END;**

**$$ LANGUAGE plpgsql;**

**select catcitygdp('China');**

### 记录类型

语法：

**TYPE rec\_type IS RECORD ( fields )**

示例：

**CREATE SCHEMA IF NOT EXISTS test\_oracle;**

**set search\_path to test\_oracle;**

**create table tbl\_city(id int, population int, nation varchar(32), city varchar(32), gdp int, others text);**

**insert into tbl\_city(id, population, nation, city, gdp) values(1,500, 'China','Guangzhou', 23000);**

**insert into tbl\_city(id, population, nation, city, gdp) values(2,1500, 'China', 'Shanghai', 29000);**

**insert into tbl\_city(id, population, nation, city, gdp) values(3,1500, 'China', 'Beijing', 25000);**

**insert into tbl\_city(id, population, nation, city, gdp) values(4,1000, 'China', 'Shenzhen', 24000);**

**insert into tbl\_city(id, population, nation, city, gdp) values(5,1000,'USA','New York', 35000);**

**insert into tbl\_city(id, population, nation, city, gdp) values(6,500, 'USA', 'Bostom', 15000);**

**insert into tbl\_city(id, population, nation, city, gdp) values(7,500, 'Japan','Tokyo', 40000);**

**insert into tbl\_city(id, population, nation, city, gdp) values(8,800, 'China', 'Hongkong', 23500);**

**insert into tbl\_city(id, population, nation, city, gdp) values(9,800, 'China', 'Hangzhou', 15500);**

**insert into tbl\_city(id, population, nation, city, gdp) values(10,100, 'USA', 'Los Angele', 15500);**

**set search\_path to test\_oracle;**

**DROP FUNCTION catcity;**

**CREATE FUNCTION catcity(nation\_name text) RETURNS text AS $$**

**DECLARE**

**TYPE mycitytype IS RECORD(city text);**

**trow tbl\_city%ROWTYPE;**

**rlt text;**

**lt mycitytype;**

**BEGIN**

**rlt = ':' || nation\_name;**

**FOR trow IN SELECT \* FROM tbl\_city WHERE nation = nation\_name**

**LOOP**

**lt.city = trow.city;**

**rlt = lt.city ||'+' || rlt;**

**END LOOP;**

**return rlt;**

**END;**

**$$ LANGUAGE plpgsql;**

**select catcity('China');**

## 控制结构

### GOTO

语法：

**GOTO lable**

示例：

**create table goto\_test(id int,name varchar);**

**create or replace function goto\_func(v\_maxnum integer) returns void as**

**declare**

**maxnum integer;**

**begin**

**maxnum := v\_maxnum;**

**for i in 1..maxnum loop**

**if i=3 then**

**goto label;**

**end if;**

**insert into goto\_test values (i,'hasai');**

**end loop;**

**<<label>>**

**update goto\_test set name = 'user1' where id = 2;**

**end;**

**/**

**select goto\_func(5);**

**select \* from goto\_test;**

## 自治事务

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;**

## 动态SQL

EXECUTE IMMEDIATE 命令用于动态运行 SQL 命令。

示例：

**drop table if exists row\_test;**

**create table row\_test(c0 int,c67 varchar,c41 text);**

**insert into row\_test values(1,'zhangsan','BeiJing');**

**insert into row\_test values(2,'lisi','ShangHai');**

**insert into row\_test values(1,'wangwu','ShenZhen');**

**create or replace procedure exe\_imed\_pro(in\_no int) as**

**declare**

**v\_sql text;**

**v\_nm varchar;**

**v\_ad varchar;**

**begin**

**perform dbms\_output.disable();**

**perform dbms\_output.enable();**

**perform dbms\_output.serveroutput ('t');**

**v\_sql := 'select c67,c41 from row\_test where c0='||in\_no;**

**execute immediate v\_sql into v\_nm,v\_ad;**

**perform dbms\_output.put\_line(v\_nm || '-' ||v\_ad);**

**end;**

**/**

**call exe\_imed\_pro(1);**

# SQL与PLSQL

## q’转义字符

一个Q-quote的表达式，用来简化SQL或PLSQL中字符串的表示。

示例：

select q'qMy "good" Name''q' names from dual;

## WITH FUNCTION

在子查询的WITH子句中定义函数，并在普通SQL语句中使用。

语法：

**WITH**

**FUNCTION** <NAME\_FUNCTION>

BEGIN

...

END;

SELECT <NAME\_FUNCTION>

FROM <TABLE>;

/

示例：

**create table tw(t1 int, t2 int);**

**insert into tw values(3, 4);**

**insert into tw values(1, 2);**

**WITH FUNCTION raise\_test\_wf(int) returns int as $$**

**begin**

**raise notice 'This message has too many parameters %', $1;**

**return $1 + 1;**

**end;**

**$$ language plpgsql;**

**FUNCTION raise\_test\_wf2(int) returns int as $$**

**begin**

**raise notice 'This message has too many parameters %', $1;**

**return $1 + 1000;**

**end;**

**$$ language plpgsql;**

**select raise\_test\_wf2(t2), raise\_test\_wf(t1) from tw where raise\_test\_wf(t2) > 0 order by 1;**

**/**

## BULK COLLECT

BULK COLLECT可以使SELECT语句可通过一次提取来检索多行，从而提高了数据检索的速度。

示例：

**create table t5(f1 integer,f2 varchar(10));**

**insert into t5 values(1,'tbase1');**

**insert into t5 values(2,'tbase2');**

**DECLARE**

**TYPE name\_tbl IS TABLE OF VARCHAR2(50);**

**emp\_name name\_tbl;**

**BEGIN**

**SELECT f2**

**BULK COLLECT INTO emp\_name**

**FROM t5;**

**FOR i IN emp\_name.FIRST .. emp\_name.LAST**

**LOOP**

**perform Dbms\_output.put\_line(i);**

**END LOOP;**

**END;**

**/**

## 集合FIRST与LAST

可以使用FIRST与LAST来遍历集合中的元素。

示例：

**create table t5(f1 integer,f2 varchar(10));**

**insert into t5 values(1,'tbase1');**

**insert into t5 values(2,'tbase2');**

**DECLARE**

**TYPE name\_tbl IS TABLE OF VARCHAR2(50);**

**emp\_name name\_tbl;**

**BEGIN**

**SELECT f2**

**BULK COLLECT INTO emp\_name**

**FROM t5;**

**FOR i IN emp\_name.FIRST .. emp\_name.LAST**

**LOOP**

**perform Dbms\_output.put\_line(i);**

**END LOOP;**

**END;**

**/**

## 匿名块

没有名称的块是匿名块。匿名块仅供一次性使用。

示例：

**create table t5(f1 integer,f2 varchar(10));**

**insert into t5 values(1,'tbase1');**

**insert into t5 values(2,'tbase2');**

**DECLARE**

**TYPE name\_tbl IS TABLE OF VARCHAR2(50);**

**emp\_name name\_tbl;**

**BEGIN**

**SELECT f2**

**BULK COLLECT INTO emp\_name**

**FROM t5;**

**FOR i IN emp\_name.FIRST .. emp\_name.LAST**

**LOOP**

**perform Dbms\_output.put\_line(i);**

**END LOOP;**

**END;**

**/**

## CONNECT BY

CONNECT BY级联查询，常用于对具有树状结构的记录查询某一节点的所有子孙节点或所有祖辈节点，或通过CONNECT BY 结合LEVEL来构造数据。

LEVEL关键字，代表树形结构中的层级编号；第一层是数字1，第二层数字2，依次递增。

CONNECT\_BY\_ROOT方法，能够获取第一层集结点结果集中的任意字段的值；例CONNECT\_BY\_ROOT（字段名）。

语法

**SELECT { \* | COLUMN | expression ,...}**

**FROM table [START WITH condition1]**

**CONNECT BY [PRIOR] id=parentid**

一般用来查找存在父子关系的数据，也就是树形结构的数据；其返还的数据也能够明确的区分出每一层的数据。

start with condition1 是用来限制第一层的数据，或者叫根节点数据；以这部分数据为基础来查找第二层数据，然后以第二层数据查找第三层数据以此类推。

connect by [prior] id=parentid 这部分是用来指明oracle在查找数据时以怎样的一种关系去查找；比如说查找第二层的数据时用第一层数据的id去跟表里面记录的parentid字段进行匹配，如果这个条件成立那么查找出来的数据就是第二层数据，同理查找第三层第四层…等等都是按这样去匹配。

另一种写法

**SELECT { \* | COLUMN | expression ,...}**

**FROM table [START WITH condition1]**

**CONNECT BY id=[PRIOR] parentid**

这种用法就表示从下往上查找数据，可以理解为从叶子节点往上查找父级几点，用第一层数据的parentid去跟表记录里面的id进行匹配，匹配成功那么查找出来的就是第二层数据；上面的那种就是从父级节点往下查找叶子节点。

使用限制

1. connect by用法较多，当前TBase V5版本能支持常用的写法，某些写可能不支持；
2. 暂时不支持CONNECT\_BY\_ROOT方法；
3. 暂不支持基于临时表的connect by
4. select \* 输出的字段里多了隐藏字段 ctid、xc\_node\_id、tableoid、\_level\_1，需要根据需要输出指定字段。
5. Connect by 和rownum结合计算，可能会存在逻辑错误。

示例

**postgres=# create table tab\_test (id int, fid int,remark varchar(16));**

**CREATE TABLE**

**postgres=# insert into tab\_test values(0,-1,'根节点');**

**INSERT 0 1**

**postgres=# insert into tab\_test values(1,0,'第二层节点1');**

**INSERT 0 1**

**postgres=# insert into tab\_test values(2,0,'第二层节点2');**

**INSERT 0 1**

**postgres=# insert into tab\_test values(3,1,'第三层节点1');**

**INSERT 0 1**

**postgres=# insert into tab\_test values(4,1,'第三层节点2');**

**INSERT 0 1**

1）从根节点查找叶子节点

**postgres=# select t.id,t.fid,t.remark, level**

**from tab\_test t**

**start with t.id = 0**

**connect by prior t.id = t.fid;**

**id | fid | remark | level**

**----+-----+-------------+-------**

**0 | -1 | 根节点 | 1**

**1 | 0 | 第二层节点1 | 2**

**2 | 0 | 第二层节点2 | 2**

**3 | 1 | 第三层节点1 | 3**

**4 | 1 | 第三层节点2 | 3**

**(5 rows)**

1. 从叶子节点查询上层节点

**postgres=# select t.id,t.fid,t.remark, level**

**from tab\_test t**

**start with t.id = 4**

**connect by t.id = prior t.fid;**

**id | fid | remark | level**

**----+-----+-------------+-------**

**4 | 1 | 第三层节点2 | 1**

**1 | 0 | 第二层节点1 | 2**

**0 | -1 | 根节点 | 3**

**(3 rows)**

**postgres=# select t.id,t.fid,t.remark, level**

**from tab\_test t**

**start with t.id = 4**

**connect by prior t.fid = t.id;**

**id | fid | remark | level**

**----+-----+-------------+-------**

**4 | 1 | 第三层节点2 | 1**

**1 | 0 | 第二层节点1 | 2**

**0 | -1 | 根节点 | 3**

**(3 rows)**

3）生成数字序列结果集

**postgres=# select rownum from dual connect by rownum<=10;**

**rownum**

**--------**

**1**

**2**

**3**

**4**

**5**

**6**

**7**

**8**

**9**

**10**

**(10 rows)**

**postgres=# select rownum from dual connect by level<=10;**

**rownum**

**--------**

**1**

**2**

**3**

**4**

**5**

**6**

**7**

**8**

**9**

**10**

**(10 rows)**

4）字符串分割，由一行变为多行

**postgres=# select REGEXP\_SUBSTR('01#02#03#04', '[^#]+', 1, rownum::int) as newport**

**from dual connect by rownum <= REGEXP\_COUNT('01#02#03#04', '[^#]+');**

**newport**

**---------**

**01**

**02**

**03**

**04**

**(4 rows)**

5）暂不支持临时表的connect by

**postgres=# select t.\*, level**

**from (select 1 as num from dual**

**union**

**select 2 as num from dual**

**) t**

**connect by level <= 3;**

**ERROR: connect by support one table yet**

## DBLINK

数据库链接（DATABASE LINK），简写DBLINK。它是一个存在于本地库中的指针，它使你可以访问远端库上的数据和对象。为了让应用可以访问分布式数据库系统中，非本地库中的数据和用户对象，就需要使用DBLINK。

数据库链接定义了从一个数据库到另一个数据库的通信路径。当应用程序使用数据库链接访问远程数据库时，Oracle数据库代表本地应用程序请求在远程数据库中建立数据库会话。

有两种主要类型的DBLINK：私有DBLINK和公有DBLINK。

私有DBLINK只有创建DBLINK的用户才可以使用；公有DBLINK可以被所有的数据库用户使用。

此处DBLINK是指从TBase访问Oracle，如需Oracle访问TBase，可通过透明网关实现，TBase访问TBase可用postgres\_fdw扩展创建外部表实现。

语法

创建插件：

**CREATE EXTENSION ORACLE\_FDW;**

创建DBLINK：

**CREATE [ PUBLIC ] DATABASE LINK dblink CONNECT TO user IDENTIFIED BY password USING connect\_string;**

DBLINK使用：

**[ schema.]{ TABLE | VIEW }@dblink**

在表或视图名后加@dblink表示访问的是远端数据库对象。

删除DBLINK：

**DROP [ PUBLIC ] DATABASE LINK dblink;**

相关系统表

系统表PG\_DBLINK中存储了已创建的DBLINK信息。

使用限制

1. 创建DBLINK前需要先创建扩展 ORACLE\_FDW;
2. DBLINK支持增删改操作，但删除、修改需要表有主键；
3. 暂不支持访问远端Oracle中同义词对象；
4. 只有超级用户可创建PUBLIC DBLINK;
5. 普通用户创建DBLINK报错：permission denied for foreign-data wrapper oracle\_fdw; 可通过授权解决：grant usage on foreign data wrapper oracle\_fdw to user;
6. Oracle字符集与TBase可能会有差异，当前TBase数据库字符集为GBK或GB18030时，DBLINK访问时Oracle端都会设置为ZHS16GBK；
7. 由于占⽤了关键字“@”，导致原TBase⽀持的“@”操作符失效，如需使⽤，⽤abs函数代替之

**postgres=# select @ -0.5;**

**ERROR: syntax error at or near "@"**

**LINE 1: select @ -0.5;**

**^**

**postgres=# select abs(-0.5);**

**abs**

**-----**

**0.5**

**(1 row)**

示例

**postgres=# create extension ORACLE\_FDW;**

**CREATE EXTENSION**

**创建DBLINK：**

**postgres=# CREATE DATABASE LINK abc CONNECT TO "JENNY" IDENTIFIED BY 'jenny' USING '(DESCRIPTION =**

**(ADDRESS = (PROTOCOL = TCP)(HOST = 100.98.176.136)(PORT = 1521))**

**(CONNECT\_DATA =**

**(SERVER = DEDICATED)**

**(SERVICE\_NAME = oracle.oracle)**

**)**

**)';**

**CREATE DATABASE LINK**

查询dblink系统表：

**postgres=# select \* from pg\_dblink ;**

**dblinkname | dblinkowner | username | created | host**

**| port | dblinkkind | dblink\_foreign\_server**

**------------+-------------+----------+-------------------------------+------------**

**----+------+------------+-----------------------**

**abc | weily | JENNY | 2020-08-11 18:24:30.425309+08 | 100.98.176.**

**136 | 1521 | 0 | ora\_dblink\_abc**

**(1 row)**

访问远程表：

**postgres=# select \* from "JENNY"."FOO"@abc;**

**ID | STR**

**----+-----**

**1 | a**

**(1 row)**

**postgres=# select \* from PG\_TEST@abc;**

**ID | STR**

**----+-----**

**1 | a**

**2 | b**

**4 | d**

**3 | c**

**(4 rows)**

## 分布键为NULL

开启Oracle兼容参数设置后，分布键支持为NULL。

示例

**postgres=# set enable\_oracle\_compatible to on;**

**SET**

**postgres=# create table tbl\_shard\_null(f1 int,f2 varchar(16)) distribute by shard(f1);**

**CREATE TABLE**

**postgres=# \d+ tbl\_shard\_null**

**Table "public.tbl\_shard\_null"**

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

**--------+-----------------------+-----------+----------+---------+----------+--------------+-------------**

**f1 | integer | | | | plain | |**

**f2 | character varying(16) | | | | extended | |**

**Distribute By: SHARD(f1)**

**Location Nodes: ALL DATANODES**

**postgres=# insert into tbl\_shard\_null select null,'null';**

**INSERT 0 1**

**postgres=# select \* from tbl\_shard\_null ;**

**f1 | f2**

**----+------**

**| null**

**(1 row)**

## 支持不同字段类型UNION ALL合并

PostgreSQL只支持相同字段类型UNION ALL合并，TBase V5则支持不同字段类型的合并。

示例

**postgres=# select 1 as f1,2 as f2,'111' as f3 union all select 11,'22',33;**

**f1 | f2 | f3**

**----+----+-----**

**1 | 2 | 111**

**11 | 22 | 33**

**(2 rows)**

## 支持只读事务时可以获取序列的下一个值

Oracle支持只读事务时可以获取序列的下一个值，TBase V5兼容Oracle该用法。

示例

**postgres=# CREATE SEQUENCE seq\_userid;**

**CREATE SEQUENCE**

**postgres=# select nextval('seq\_userid');**

**nextval**

**---------**

**1**

**(1 row)**

**postgres=# begin transaction read only;**

**BEGIN**

**postgres=# select nextval('seq\_userid');**

**nextval**

**---------**

**2**

**(1 row)**

**postgres=# rollback ;**

**ROLLBACK**

**postgres=# select nextval('seq\_userid');**

**nextval**

**---------**

**3**

**(1 row)**

## 支持非分布键关联删除和更新数据

支持非分布键关联删除和更新数据

示例：

**postgres=# create table t1(f1 int not null,f2 int);**

**CREATE TABLE**

**postgres=# create table t2 (f1 int not null, f2 int);**

**CREATE TABLE**

**postgres=# \d+ t1**

**Table "public.t1"**

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

**--------+---------+-----------+----------+---------+---------+--------------+-------------**

**f1 | integer | | not null | | plain | |**

**f2 | integer | | | | plain | |**

**Distribute By: SHARD(f1)**

**Location Nodes: ALL DATANODES**

**postgres=# \d+ t2**

**Table "public.t2"**

**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**

表t1、t2的f1字段为分布键。

**postgres=# insert into t1 values (1,1),(2,2),(3,3),(4,4),(5,5);**

**COPY 5**

**postgres=# insert into t2 values (4,4),(5,5),(6,6),(7,7),(8,8);**

**COPY 5**

用非分布键关联删除数据。

**postgres=# delete from t1 where f2 in (select f2 from t2);**

**DELETE 2**

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

**f1 | f2**

**----+----**

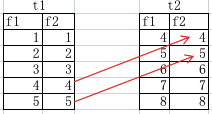
**1 | 1**

**2 | 2**

**3 | 3**

**(3 rows)**

删除符合条件的数据应该是t1.f2 为4、5的两条数据。



**postgres=# insert into t1 values(4,4),(5,5);**

**COPY 2**

用非分布键关联更新数据。

**postgres=# update t1 set f2=100 where exists (select 1 from t2 where t1.f2=t2.f2);**

**UPDATE 2**

同样关联更新的应该是t1.f2 为4、5的两条数据。

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

**f1 | f2**

**----+-----**

**1 | 1**

**2 | 2**

**5 | 100**

**3 | 3**

**4 | 100**

**(5 rows)**

## 支持带复杂子查询删除功能

支持带复杂子查询删除功能。

示例

**postgres=# create table t1( id int not null,a int);**

**CREATE TABLE**

**postgres=# create table t2(a int not null,b int);**

**CREATE TABLE**

**postgres=# insert into t1 values(1,2),(2,3),(3,4);**

**COPY 3**

**postgres=# insert into t2 values(1,2),(2,3),(3,4);**

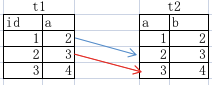
**COPY 3**

子查询中两表关联

**postgres=# delete from t1 where id in(select a.id from t1 a join t2 b on a.a=b.a where a.id>1);**

**DELETE 1**

子查询关联结果符合条件的是t1.id=2这一条。



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

**id | a**

**----+---**

**1 | 2**

**3 | 4**

**(2 rows)**

## PIVOT

PIVOT是Oracle的行转列函数，通过PIVOT函数，可以快速实现行转列的输出，而不需要用DECODE或CASE结合GROUP BY复杂的SQL实现。

TBase V5兼容PIVOT用法。

语法

**SELECT STATEMENT**

**PIVOT (aggreate\_function FOR pivot\_column IN ( list of values) )**

**[ ORDER BY { column\_name | expr } [, ...] ];**

示例

**postgres=# create table stu\_score (stu\_name varchar2(16),course varchar2(16),score number(5,2));**

**CREATE TABLE**

**postgres=# insert into stu\_score values('STU\_A','CHINESE',70);**

**INSERT 0 1**

**postgres=# insert into stu\_score values('STU\_A','ENGLISH',80);**

**INSERT 0 1**

**postgres=# insert into stu\_score values('STU\_A','MATH',81);**

**INSERT 0 1**

**postgres=# insert into stu\_score values('STU\_B','CHINESE',86);**

**INSERT 0 1**

**postgres=# insert into stu\_score values('STU\_B','ENGLISH',77);**

**INSERT 0 1**

**postgres=# insert into stu\_score values('STU\_B','MATH',69);**

**INSERT 0 1**

**postgres=# insert into stu\_score values('STU\_C','CHINESE',80);**

**INSERT 0 1**

**postgres=# insert into stu\_score values('STU\_C','ENGLISH',82);**

**INSERT 0 1**

**postgres=# insert into stu\_score values('STU\_C','MATH',88);**

**INSERT 0 1**

**postgres=# select \* from stu\_score;**

**stu\_name | course | score**

**----------+---------+-------**

**STU\_B | CHINESE | 86.00**

**STU\_B | ENGLISH | 77.00**

**STU\_B | MATH | 69.00**

**STU\_C | CHINESE | 80.00**

**STU\_C | ENGLISH | 82.00**

**STU\_C | MATH | 88.00**

**STU\_A | CHINESE | 70.00**

**STU\_A | ENGLISH | 80.00**

**STU\_A | MATH | 81.00**

**(9 rows)**

**postgres=# select \* from stu\_score**

**pivot ( max(score) for course in ('CHINESE' as CHINESE,'ENGLISH' as ENGLISH,'MATH' as MATH));**

**stu\_name | chinese | english | math**

**----------+---------+---------+-------**

**STU\_B | 86.00 | 77.00 | 69.00**

**STU\_C | 80.00 | 82.00 | 88.00**

**STU\_A | 70.00 | 80.00 | 81.00**

**(3 rows)**

## UNPIVOT

UNPIVOT是Oracle的列转行函数，通过UNPIVOT函数，可以快速实现列转行的输出，而不需要用多个SQL UNION [ALL] 实现。

TBase V5兼容UNPIVOT用法。

语法

**SELECT STATEMENT**

**UNPIVOT( column\_name FOR unpivot\_column IN ( list of values))**

**[ ORDER BY { column\_name | expr } [, ...] ];**

示例

**postgres=# create table stu\_score\_b as select stu\_name,**

**max(case when course='CHINESE' then score end) CHINESE,**

**max(case when course='ENGLISH' then score end) ENGLISH,**

**max(case when course='MATH' then score end) MATH**

**from stu\_score**

**group by stu\_name;**

**INSERT 0 3**

**postgres=# select \* from stu\_score\_b;**

**stu\_name | chinese | english | math**

**----------+---------+---------+-------**

**STU\_B | 86.00 | 77.00 | 69.00**

**STU\_C | 80.00 | 82.00 | 88.00**

**STU\_A | 70.00 | 80.00 | 81.00**

**(3 rows)**

**postgres=# select \***

**from stu\_score\_b**

**unpivot ( score for course in (CHINESE ,ENGLISH ,MATH ))**

**order by stu\_name;**

**stu\_name | score | course**

**----------+-------+---------**

**STU\_A | 81.00 | MATH**

**STU\_A | 80.00 | ENGLISH**

**STU\_A | 70.00 | CHINESE**

**STU\_B | 86.00 | CHINESE**

**STU\_B | 77.00 | ENGLISH**

**STU\_B | 69.00 | MATH**

**STU\_C | 88.00 | MATH**

**STU\_C | 82.00 | ENGLISH**

**STU\_C | 80.00 | CHINESE**

**(9 rows)**

**postgres=# select stu\_name,score,course**

**from stu\_score\_b unpivot ( score for course in (CHINESE ,ENGLISH ))**

**order by stu\_name;**

**stu\_name | score | course**

**----------+-------+---------**

**STU\_A | 70.00 | CHINESE**

**STU\_A | 80.00 | ENGLISH**

**STU\_B | 86.00 | CHINESE**

**STU\_B | 77.00 | ENGLISH**

**STU\_C | 80.00 | CHINESE**

**STU\_C | 82.00 | ENGLISH**

**(6 rows)**

## MERGE INTO

Oracle中MERGE INTO语句是将INSERT和UPDATE语句结合，同时实现INSERT和UPDATE，与表匹配的行进行更新，不匹配的行写入表中。

TBase V5兼容MERGE INTO语法。

语法

**MERGE INTO [schema.]{table|view} [t\_alias]**

**USING { [schema.]{table|view} | subquery } [t\_alias] ON (condition)**

**[ merge\_update\_clause ]**

**[ merge\_insert\_clause ];**

**merge\_update\_clause:**

**WHEN MATCHED THEN**

**UPDATE SET**

**column = {expr | DEFAULT} [, column = {expr | DEFAULT}, ... ]**

**where condition**

**DELETE where condition**

**merge\_insert\_clause:**

**WHEN NOT MATCHED THEN INSERT**

**( column [, column , ... ]**

**VALUES ({expr | DEFAULT} [, ... ]**

**where\_clause**

其中

1、ON (condition) 匹配条件，必须能从被关联的表、视图、子查询得到唯一的记录，否则报错。

示例

**drop table test1;**

**create table test1(id int primary key,name varchar2(10));**

**insert into test1 values(1,'test1');**

**insert into test1 values(2,'test1');**

**insert into test1 values(3,'test1');**

**drop table test2;**

**create table test2(id int primary key,name varchar2(10));**

**insert into test2 values(2,'test2');**

**insert into test2 values(3,'test2');**

**insert into test2 values(4,'test2');**

**insert into test2 values(5,'test2');**

**postgres=# MERGE INTO test1 t**

**USING (**

**select \* from test2**

**) t2 ON (t.id = t2.id)**

**WHEN MATCHED THEN UPDATE SET t.name = t2.name WHERE t.id = t2.id**

**WHEN NOT MATCHED THEN INSERT (id,name) VALUES (t2.id, t2.name) ;**

**MERGE 4**

上面的sql实现的功能为：将test2表的数据与test1表的数据用id关联匹配，匹配到的行用test2.name更新test1.name，将没有匹配到的test2的行写入test1表中。

查询test1表，id=2、3的数据name更新为test2表name值test2，test2表id=4、5的数据写入test1表，结果符合预期。

**postgres=# select \* from test1;**

**id | name**

**----+-------**

**1 | test1**

**2 | test2**

**5 | test2**

**3 | test2**

**4 | test2**

**(5 rows)**

## DELETE table\_name 支持不带FROM

通常标准的SQL语法在DELETE时需要带FROM关键字，如DELETE FROM table WHERE expr，Oracle中可以省略FROM关键字。

TBase V5兼容DELETE时省略FROM关键字用法。

语法

**DELETE [ FROM] table\_name [ WHERE expr ];**

示例

**postgres=# create table tbl\_del (id int,info varchar(16));**

**CREATE TABLE**

**postgres=# insert into tbl\_del values(1,'a'),(2,'b'),(3,'c');**

**COPY 3**

**postgres=# delete tbl\_del where id=3;**

**DELETE 1**

**postgres=# delete from tbl\_del where id=2;**

**DELETE 1**

## UPDATE支持 表名.字段名 = xxx 付值

UPDATE更新表时，支持字段名前加表名，或表别名的写法。

语法

**UPDATE table\_name [ [ as ] table\_alias ]**

**SET [{ table\_name | table\_alias }.]column\_name = { value | expr | column\_name } [, ... ]**

**[ WHERE expr ];**

示例

**postgres=# create table tbl\_upd (id int, info varchar(16));**

**CREATE TABLE**

**postgres=# insert into tbl\_upd values(1,'a'),(2,'b'),(3,'c');**

**COPY 3**

**postgres=# update tbl\_upd set tbl\_upd.info=tbl\_upd.info||'1' where tbl\_upd.id=3;**

**UPDATE 1**

**postgres=# select \* from tbl\_upd where id=3;**

**id | info**

**----+------**

**3 | c1**

**(1 row)**

**postgres=# update tbl\_upd t1 set t1.info=t1.info||'1' where t1.id=2;**

**UPDATE 1**

**postgres=# select \* from tbl\_upd where id=2;**

**id | info**

**----+------**

**2 | b1**

**(1 row)**

**postgres=# update tbl\_upd as t1 set t1.info=t1.info||'1' where t1.id=1;**

**UPDATE 1**

**postgres=# select \* from tbl\_upd where id=1;**

**id | info**

**----+------**

**1 | a1**

**(1 row)**

## HINT

对于一些执行计划不是最优的SQL，可以通过SQL HINT来干预执行计划的生成，例如对指定表的全表扫描，用指定的索引，用指定的表关联算法等等。

TBase V5兼容常用Oracle的HINT用法，同时支持基于pg\_hint\_plan的HINT用法。

语法

**{ DELETE |INSERT | SELECT | UPDATE } /\*+ hint1 [hint2 ...] \*/ statement;**

其中：

1、DELETE、INSERT、SELECT和UPDATE是标识一个语句块开始的关键字，包含提示的注释只能出现在这些关键字的后面，否则提示无效；

2、'+'号表示该注释是一个HINTS，该加号必须立即跟在'/\*'的后面，中间不能有空格；

3、HINT是下面介绍的具体提示之一，如果包含多个提示，则每个提示之间需要用一个或多个空格隔开；

4、如果没有正确的指定HINTS，将忽略该HINTS；

5、大小写忽略。

常用HINT场景：

|  |  |  |
| --- | --- | --- |
| HINT场景 | 兼容Oracle写法 | TBase写法 |
| 强制走mergejoin | use\_merge(table1, table2) | MergeJoin(table1 table2) |
| 强制走nestloop | use\_nl(table1, table2) | NestLoop(table1 table2) |
| 强制走hashjoin | use\_hash(table1, table2) | hashJoin(table1 table2) |
| 强制走索引 | index(table，index) | indexscan(table index) |
| 强制全表扫描 | full(table) | SeqScan(table) |

更多参见pg\_hint\_plan

<http://pghintplan.sourceforge.jp/hint_list.html>

示例

**postgres=# create table hint\_t6(f1 integer,f2 integer) ;**

**CREATE TABLE**

**postgres=# insert into hint\_t6 select t as f1,t as f2 from generate\_series(1,10000) as t;**

**INSERT 0 10000**

**postgres=# create index hint\_t6\_f1\_idx on hint\_t6(f1);**

**CREATE INDEX**

**postgres=# create table hint\_t7(f1 integer,f2 integer) ;**

**CREATE TABLE**

**postgres=# insert into hint\_t7 select t as f1,t as f2 from generate\_series(1,10000) as t;**

**INSERT 0 10000**

**postgres=# create index hint\_t7\_f1\_idx on hint\_t7(f1);**

**CREATE INDEX**

1）强制走MergeJoin

兼容Oracle写法

**postgres=# select /\*+use\_merge(t,t1) \*/ \* from hint\_t6 t,hint\_t7 t1 where t.f1=t1.f1 and t.f1>9999 ;**

**f1 | f2 | f1 | f2**

**-------+-------+-------+-------**

**10000 | 10000 | 10000 | 10000**

**(1 row)**

**postgres=# explain select /\*+use\_merge(t,t1) \*/ \* from hint\_t6 t,hint\_t7 t1 where t.f1=t1.f1 and t.f1>9999 ;**

**QUERY PLAN**

**--------------------------------------------------------------------------------------------------**

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

**Node/s: dn001, dn002**

**-> Merge Join (cost=0.56..163.78 rows=1 width=16)**

**Merge Cond: (t.f1 = t1.f1)**

**-> Index Scan using hint\_t6\_f1\_idx on hint\_t6 t (cost=0.28..2.30 rows=1 width=8)**

**Index Cond: (f1 > 9999)**

**-> Index Scan using hint\_t7\_f1\_idx on hint\_t7 t1 (cost=0.28..148.87 rows=5039 width=8)**

**(7 rows)**

**TBase写法**

**postgres=# select /\*+ MergeJoin(t t1)\*/ \* from hint\_t6 t,hint\_t7 t1 where t.f1=t1.f1 and t.f1>9999 ;**

**f1 | f2 | f1 | f2**

**-------+-------+-------+-------**

**10000 | 10000 | 10000 | 10000**

**(1 row)**

**postgres=# explain select /\*+ MergeJoin(t t1)\*/ \* from hint\_t6 t,hint\_t7 t1 where t.f1=t1.f1 and t.f1>9999 ;**

**QUERY PLAN**

**--------------------------------------------------------------------------------------------------**

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

**Node/s: dn001, dn002**

**-> Merge Join (cost=0.56..163.78 rows=1 width=16)**

**Merge Cond: (t.f1 = t1.f1)**

**-> Index Scan using hint\_t6\_f1\_idx on hint\_t6 t (cost=0.28..2.30 rows=1 width=8)**

**Index Cond: (f1 > 9999)**

**-> Index Scan using hint\_t7\_f1\_idx on hint\_t7 t1 (cost=0.28..148.87 rows=5039 width=8)**

**(7 rows)**

1. 强制走 nestloop

兼容Oracle写法：

**postgres=# explain select /\*+use\_nl(t,t1) \*/ \* from hint\_t6 t,hint\_t7 t1 where t.f1=t1.f1 and t.f1>999 ;**

**QUERY PLAN**

**---------------------------------------------------------------------------------------------**

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

**Node/s: dn001, dn002**

**-> Nested Loop (cost=0.28..1624.87 rows=4548 width=16)**

**-> Seq Scan on hint\_t6 t (cost=0.00..102.99 rows=4548 width=8)**

**Filter: (f1 > 999)**

**-> Index Scan using hint\_t7\_f1\_idx on hint\_t7 t1 (cost=0.28..0.32 rows=1 width=8)**

**Index Cond: (f1 = t.f1)**

**(7 rows)**

**TBase写法：**

**postgres=# explain select /\*+NestLoop(t t1) \*/\* from hint\_t6 t,hint\_t7 t1 where t.f1=t1.f1 and t1.f1>999 ;**

**QUERY PLAN**

**--------------------------------------------------------------------------------------------**

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

**Node/s: dn001, dn002**

**-> Nested Loop (cost=0.28..1624.87 rows=4548 width=16)**

**-> Seq Scan on hint\_t7 t1 (cost=0.00..102.99 rows=4548 width=8)**

**Filter: (f1 > 999)**

**-> Index Scan using hint\_t6\_f1\_idx on hint\_t6 t (cost=0.28..0.32 rows=1 width=8)**

**Index Cond: (f1 = t1.f1)**

**(7 rows)**

1. 强制走 hashjoin

兼容Oracle写法：

**postgres=# explain select /\*+use\_hash(t,t1) \*/ \* from hint\_t6 t,hint\_t7 t1 where t.f1=t1.f1 and t.f1>9999 ;**

**QUERY PLAN**

**--------------------------------------------------------------------------------------------------**

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

**Node/s: dn001, dn002**

**-> Hash Join (cost=2.31..111.61 rows=1 width=16)**

**Hash Cond: (t1.f1 = t.f1)**

**-> Seq Scan on hint\_t7 t1 (cost=0.00..90.39 rows=5039 width=8)**

**-> Hash (cost=2.30..2.30 rows=1 width=8)**

**-> Index Scan using hint\_t6\_f1\_idx on hint\_t6 t (cost=0.28..2.30 rows=1 width=8)**

**Index Cond: (f1 > 9999)**

(8 rows)

TBase写法：

**postgres=# explain select /\*+ hashJoin(t t1) SeqScan(t) SeqScan(t1)\*/ \* from hint\_t6 t,hint\_t7 t1 where t.f1=t1.f1 and t.f1>9999 ;**

**QUERY PLAN**

**-----------------------------------------------------------------------------**

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

**Node/s: dn001, dn002**

**-> Hash Join (cost=103.00..212.30 rows=1 width=16)**

**Hash Cond: (t1.f1 = t.f1)**

**-> Seq Scan on hint\_t7 t1 (cost=0.00..90.39 rows=5039 width=8)**

**-> Hash (cost=102.99..102.99 rows=1 width=8)**

**-> Seq Scan on hint\_t6 t (cost=0.00..102.99 rows=1 width=8)**

**Filter: (f1 > 9999)**

**(8 rows)**

4）强制走索引

**postgres=# create index idx\_hint\_t6\_f1f2 on hint\_t6(f1,f2);**

**CREATE INDEX**

**postgres=# create index idx\_hint\_t7\_f1 on hint\_t7(f1);**

**CREATE INDEX**

**postgres=# explain select \* from hint\_t6 t,hint\_t7 t1 where t.f1=t1.f1 and t.f1>9999 ;**

**QUERY PLAN**

**---------------------------------------------------------------------------------------------**

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

**Node/s: dn001, dn002**

**-> Nested Loop (cost=0.56..6.61 rows=1 width=16)**

**-> Index Scan using idx\_hint\_t6\_f1 on hint\_t6 t (cost=0.28..2.30 rows=1 width=8)**

**Index Cond: (f1 > 9999)**

**-> Index Scan using idx\_hint\_t7\_f1 on hint\_t7 t1 (cost=0.28..4.30 rows=1 width=8)**

**Index Cond: (f1 = t.f1)**

(7 rows)

在没有hint强制走索引时，执行计划用到索引idx\_hint\_t6\_f1、idx\_hint\_t7\_f1。

强制走索引idx\_hint\_t6\_f1f2  
 兼容Oracle写法：

**postgres=# explain select /\*+ index(t idx\_hint\_t6\_f1f2) \*/ \* from hint\_t6 t,hint\_t7 t1 where t.f1=t1.f1 and t.f1>9999 ;**

**QUERY PLAN**

**----------------------------------------------------------------------------------------------**

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

**Node/s: dn001, dn002**

**-> Nested Loop (cost=0.56..7.48 rows=1 width=16)**

**-> Index Scan using idx\_hint\_t6\_f1f2 on hint\_t6 t (cost=0.28..3.17 rows=1 width=8)**

**Index Cond: (f1 > 9999)**

**-> Index Scan using idx\_hint\_t7\_f1 on hint\_t7 t1 (cost=0.28..4.30 rows=1 width=8)**

**Index Cond: (f1 = t.f1)**

**(7 rows)**

TBase写法：

**postgres=# explain select /\*+ indexscan(t idx\_hint\_t6\_f1f2) \*/ \* from hint\_t6 t,hint\_t7 t1 where t.f1=t1.f1 and t.f1>9999 ;**

**QUERY PLAN**

**----------------------------------------------------------------------------------------------**

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

**Node/s: dn001, dn002**

**-> Nested Loop (cost=0.56..7.48 rows=1 width=16)**

**-> Index Scan using idx\_hint\_t6\_f1f2 on hint\_t6 t (cost=0.28..3.17 rows=1 width=8)**

**Index Cond: (f1 > 9999)**

**-> Index Scan using idx\_hint\_t7\_f1 on hint\_t7 t1 (cost=0.28..4.30 rows=1 width=8)**

**Index Cond: (f1 = t.f1)**

**(7 rows)**

**postgres=#**

1. 强制全表扫描

**postgres=# explain select /\*+ index(t idx\_hint\_t6\_f1f2) SeqScan(t1) \*/ \* from hint\_t6 t,hint\_t7 t1 where t.f1=t1.f1 and t.f1>9999 ;**

**QUERY PLAN**

**----------------------------------------------------------------------------------------------------**

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

**Node/s: dn001, dn002**

**-> Hash Join (cost=3.19..112.48 rows=1 width=16)**

**Hash Cond: (t1.f1 = t.f1)**

**-> Seq Scan on hint\_t7 t1 (cost=0.00..90.39 rows=5039 width=8)**

**-> Hash (cost=3.17..3.17 rows=1 width=8)**

**-> Index Scan using idx\_hint\_t6\_f1f2 on hint\_t6 t (cost=0.28..3.17 rows=1 width=8)**

**Index Cond: (f1 > 9999)**

**(8 rows)**

指定SeqScan(t1)后，执行计划中t1表由索引扫描变为全表扫描。

## SQL PROFILE

Oracle中的PROFILE可以用来对用户所能使用的数据库资源进行限制。

TBase V5兼容PROFILE用法。

语法

创建PROFILE：

**CREATE PROFILE profile**

**LIMIT { resource\_parameters**

**| password\_parameters**

**}**

**[ resource\_parameters**

**| password\_parameters**

**]... ;**

**<resource\_parameters>**

**{ { SESSIONS\_PER\_USER**

**| CPU\_PER\_SESSION**

**| CPU\_PER\_CALL**

**| CONNECT\_TIME**

**| IDLE\_TIME**

**| LOGICAL\_READS\_PER\_SESSION**

**| LOGICAL\_READS\_PER\_CALL**

**| COMPOSITE\_LIMIT**

**}**

**{ integer | UNLIMITED | DEFAULT }**

**| PRIVATE\_SGA**

**{ integer [ K | M ] | UNLIMITED | DEFAULT }**

**}**

**< password\_parameters >**

**{ { FAILED\_LOGIN\_ATTEMPTS**

**| PASSWORD\_LIFE\_TIME**

**| PASSWORD\_REUSE\_TIME**

**| PASSWORD\_REUSE\_MAX**

**| PASSWORD\_LOCK\_TIME**

**| PASSWORD\_GRACE\_TIME**

**}**

**{ expr | UNLIMITED | DEFAULT }**

**| PASSWORD\_VERIFY\_FUNCTION**

**{ function | NULL | DEFAULT }**

**}**

profile分配给用户：

**ALTER USER user\_name PROFILE profile\_name;**

删除profile：

**DROP PROFILE profile\_name;**

示例

**cpu\_per\_call limit**

**postgres=# create profile app\_user limit**

**sessions\_per\_user 10**

**cpu\_per\_session 1000**

**cpu\_per\_call 100**

**connect\_time 100**

**logical\_reads\_per\_session 1000**

**logical\_reads\_per\_call 1000**

**composite\_limit 5000000**

**Idle\_Time 1000**

**private\_sga 1000**

**failed\_login\_attempts 5**

**password\_life\_time 1**

**password\_reuse\_time 2**

**password\_reuse\_max 5**

**password\_lock\_time 0.042**

**password\_grace\_time 0.042;**

**CREATE PROFILE**

**postgres=# ALTER RESOURCE COST CPU\_PER\_SESSION 3 CONNECT\_TIME 3 private\_sga 2 LOGICAL\_READS\_PER\_SESSION 2;**

**ALTER RESOURCE COST**

**postgres=# create user cpu\_call password '123';**

**CREATE ROLE**

**postgres=# create profile profile\_cpu\_call limit cpu\_per\_call 10;**

**CREATE PROFILE**

**postgres=# alter user cpu\_call profile profile\_cpu\_call;**

**ALTER ROLE**

**postgres=# \c - cpu\_call**

**You are now connected to database "postgres" as user "cpu\_call".**

**postgres=> create table bb(id int,name varchar(20));**

**ERROR: node:dn001, backend\_pid:13359, nodename:dn002,backend\_pid:20303,message:exceeded call limit on CPU usage**

**postgres=> \c - tbase**

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

**postgres=# drop user cpu\_call;**

**drop profile profile\_cpu\_call;DROP ROLE**

**postgres=# drop profile profile\_cpu\_call;**

**DROP PROFILE**

**cpu\_per\_session limit**

**postgres=# create user cpu\_session password '123';**

**CREATE ROLE**

**postgres=# create profile profile\_cpu\_session limit cpu\_per\_session 10;**

**CREATE PROFILE**

**postgres=# alter user cpu\_session profile profile\_cpu\_session;**

**ALTER ROLE**

**postgres=# \c - cpu\_session**

**You are now connected to database "postgres" as user "cpu\_session".**

**postgres=> create table bb(id int,name varchar(20));**

**\c - tbase**

**drop user cpu\_session;**

**drop profile profile\_cpu\_session;**

**FATAL: node:dn002, backend\_pid:21503, nodename:dn001,backend\_pid:14480,message:exceeded session limit on CPU usage, you are being logged off**

**server closed the connection unexpectedly**

**This probably means the server terminated abnormally**

**before or while processing the request.**

**The connection to the server was lost. Attempting reset: Succeeded.**

**postgres=> \c - tbase**

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

**postgres=# drop user cpu\_session;**

**DROP ROLE**

**postgres=# drop profile profile\_cpu\_session;**

**DROP PROFILE**

**postgres=#**

**logical\_reads\_per\_session limit**

**postgres=# \c - tbase**

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

**postgres=# create user blk\_session password '123';**

**alter user blk\_session profile profile\_blk\_session;**

**CREATE ROLE**

**postgres=# create profile profile\_blk\_session limit logical\_reads\_per\_session 10;**

**\c - blk\_session**

**CREATE PROFILE**

**postgres=# alter user blk\_session profile profile\_blk\_session;**

**create table bb(id int,name varchar(20));**

**ALTER ROLE**

**postgres=# \c - blk\_session**

**You are now connected to database "postgres" as user "blk\_session".**

**postgres=> create table bb(id int,name varchar(20));**

**\c - tbase**

**drop user blk\_session;**

**drop profile profile\_blk\_session;**

**FATAL: node:dn002, backend\_pid:25880, nodename:dn001,backend\_pid:18543,message:exceeded session limit on IO usage, you are being logged off**

**server closed the connection unexpectedly**

**This probably means the server terminated abnormally**

**before or while processing the request.**

**The connection to the server was lost. Attempting reset: Succeeded.**

**postgres=> \c - tbase**

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

**postgres=# drop user blk\_session;**

**DROP ROLE**

**postgres=# drop profile profile\_blk\_session;**

**DROP PROFILE**

**postgres=#**

**logical\_reads\_per\_call limit**

**postgres=# \c - tbase**

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

**postgres=# create user blk\_call password '123';**

**alter user blk\_call profile profile\_blk\_call;**

**CREATE ROLE**

**postgres=# create profile profile\_blk\_call limit logical\_reads\_per\_call 10;**

**CREATE PROFILE**

**postgres=# alter user blk\_call profile profile\_blk\_call;**

**\c - blk\_call**

**create table bb(id int,name varchar(20));**

**ALTER ROLE**

**postgres=# \c - blk\_call**

**You are now connected to database "postgres" as user "blk\_call".**

**postgres=> create table bb(id int,name varchar(20));**

**\c - tbase**

**drop user blk\_call;**

**drop profile profile\_blk\_call;ERROR: node:dn001, backend\_pid:19447, nodename:dn002,backend\_pid:26821,message:exceeded call limit on IO usage**

**postgres=> \c - tbase**

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

**postgres=# drop user blk\_call;**

**DROP ROLE**

**postgres=# drop profile profile\_blk\_call;**

**DROP PROFILE**

**composite\_limit limit**

**postgres=# \c - tbase**

**create user composite password '123';**

**create profile profile\_composite limit composite\_limit 10;**

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

**postgres=# create user composite password '123';**

**alter user composite profile profile\_composite;**

**CREATE ROLE**

**postgres=# create profile profile\_composite limit composite\_limit 10;**

**\c - composite**

**create table bb(id int,name varchar(20));**

**CREATE PROFILE**

**postgres=# alter user composite profile profile\_composite;**

**ALTER ROLE**

**postgres=# \c - composite**

**\c - tbase**

**drop user composite;**

**You are now connected to database "postgres" as user "composite".**

**postgres=> create table bb(id int,name varchar(20));**

**drop profile profile\_composite;**

**FATAL: node:dn002, backend\_pid:27700, nodename:dn001,backend\_pid:20266,message:exceeded COMPOSITE\_LIMIT, you are being logged off**

**server closed the connection unexpectedly**

**This probably means the server terminated abnormally**

**before or while processing the request.**

**The connection to the server was lost. Attempting reset: Succeeded.**

**postgres=> \c - tbase**

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

**postgres=# drop user composite;**

**DROP ROLE**

**postgres=# drop profile profile\_composite;**

**DROP PROFILE**

**private\_sga limit**

**postgres=# \c - tbase**

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

**postgres=# create user sga password '123';**

**create profile profile\_private\_sga limit private\_sga 10;**

**alter user sga profile profile\_private\_sga;**

**\c - sga**

**CREATE ROLE**

**postgres=# create profile profile\_private\_sga limit private\_sga 10;**

**create table bb(id int,name varchar(20));**

**CREATE PROFILE**

**postgres=# alter user sga profile profile\_private\_sga;**

**\c - tbase**

**drop user sga;**

**ALTER ROLE**

**postgres=# \c - sga**

**You are now connected to database "postgres" as user "sga".**

**postgres=> create table bb(id int,name varchar(20));**

**drop profile profile\_private\_sga;FATAL: node:dn001, backend\_pid:21253, nodename:dn002,backend\_pid:28723,message:exceeded private\_sga, you are being logged off**

**server closed the connection unexpectedly**

**This probably means the server terminated abnormally**

**before or while processing the request.**

**The connection to the server was lost. Attempting reset: Succeeded.**

**postgres=> \c - tbase**

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

**postgres=# drop user sga;**

**DROP ROLE**

**postgres=# drop profile profile\_private\_sga;**

**DROP PROFILE**

# 客户端与驱动

## JDBC

TBase兼容PostgreSQL的JDBC，可以按PostgreSQL配置JDBC的方法配置和使用。

### 驱动包获取

标准包可从PostgreSQL官网下载，根据jdk版本选择下载对应驱动包。

官网下载地址：

https://jdbc.postgresql.org/download.html

需要使用TBase V5 Oracle兼容特性，可从TBase产研团队获取定制驱动包。

### 连接配置

jdbc:postgresql://host:port/database/?oracle\_compile=true&...

其中：

host：监控主机ip

port：监听端口

database：连接数据库

?：database与参数分隔符

&：多个参数之间分隔符

oracle\_compile=true，兼容Oracle出参字段是大写,TBase为小写。

支持配置多个host：ip，实现负载均衡：

jdbc:postgresql://host1:port1,host2:port2[,...]/database/?loadBalanceHosts=true&oracle\_compile=true&...

其中：

多个host:ip之间用英文逗号隔开，需要配置参数loadBalanceHosts=true。

更多参数配置参考官网：

<https://jdbc.postgresql.org/documentation/head/connect.html>

### Java代码示例

1.增删改查示例代码如下：

**package com.Postgres;**

**import java.sql.Connection;**

**import java.sql.DriverManager;**

**import java.sql.Statement;**

**import java.sql.PreparedStatement;**

**import java.sql.ResultSet;**

**import java.sql.SQLException;**

**public class TbaseSQLJDBC {**

**private String url = "jdbc:postgresql://127.0.0.1:6666/db\_test2";**

**private String username = "tbaseadmin";**

**private String password = "testtest123";**

**public Connection getConn() {**

**Connection connection = null;**

**try {**

**connection = DriverManager.getConnection(url, username, password);**

**} catch (Exception e) {**

**// TODO Auto-generated catch block**

**e.printStackTrace();**

**}**

**return connection;**

**}**

**public void createTable() {**

**Statement stmt = null;**

**Connection c = getConn();**

**try {**

**Class.forName("org.postgresql.Driver");**

**c = DriverManager.getConnection("jdbc:postgresql://127.0.0.1:6666/db\_test2", "tbaseadmin", "testtest123");**

**c.setAutoCommit(false); // 把自动提交**

**System.out.println("Opened database successfully");**

**stmt = c.createStatement();**

**String sql = "CREATE TABLE tb\_test " +**

**"(ID INT PRIMARY KEY NOT NULL ," +**

**" NAME VARCHAR(20) NOT NULL)";**

**stmt.executeUpdate(sql);**

**System.out.println("Table created successfully");**

**c.commit();**

**stmt.close();**

**c.close();**

**} catch (Exception e) {**

**System.err.println(e.getClass().getName() + ": " + e.getMessage());**

**System.exit(0);**

**}**

**}**

**public int insert(TableTest test) {**

**Connection conn = getConn();**

**int i = 0;**

**String sql = "insert into tb\_test (id,Name) values(?,?)";**

**PreparedStatement pstmt;**

**try {**

**pstmt = (PreparedStatement) conn.prepareStatement(sql);**

**pstmt.setInt(1, test.getId());**

**pstmt.setString(2, test.getName());**

**i = pstmt.executeUpdate();**

**pstmt.close();**

**conn.close();**

**} catch (SQLException e) {**

**e.printStackTrace();**

**}**

**return i;**

**}**

**public int delete(String name) {**

**Connection conn = getConn();**

**int i = 0;**

**String sql = "delete from tb\_test where Name=?";**

**PreparedStatement pstmt;**

**try {**

**pstmt = (PreparedStatement) conn.prepareStatement(sql);**

**pstmt.setString(1, name);**

**i = pstmt.executeUpdate();**

**System.out.println("resutl: " + i);**

**pstmt.close();**

**conn.close();**

**} catch (SQLException e) {**

**e.printStackTrace();**

**}**

**return i;**

**}**

**public int update(TableTest test) {**

**Connection conn = getConn();**

**int i = 0;**

**String sql = "update tb\_test set name=? where id = ?";**

**PreparedStatement pstmt;**

**try {**

**pstmt = (PreparedStatement) conn.prepareStatement(sql);**

**pstmt.setString(1, test.getName());**

**pstmt.setInt(2, 1);**

**i = pstmt.executeUpdate();**

**System.out.println("resutl: " + i);**

**pstmt.close();**

**conn.close();**

**} catch (SQLException e) {**

**e.printStackTrace();**

**}**

**return i;**

**}**

**public Integer getAll() {**

**Connection conn = getConn();**

**String sql = "select \* from tb\_test";**

**PreparedStatement pstmt;**

**try {**

**pstmt = (PreparedStatement)conn.prepareStatement(sql);**

**ResultSet rs = pstmt.executeQuery();**

**int col = rs.getMetaData().getColumnCount();**

**while (rs.next()) {**

**for (int i = 1; i <= col; i++) {**

**System.out.print(rs.getString(i) + "\t");**

**if ((i == 2) && (rs.getString(i).length() < 8)) {**

**System.out.print("\t");**

**}**

**}**

**}**

**} catch (SQLException e) {**

**e.printStackTrace();**

**}**

**return null;**

**}**

**}**

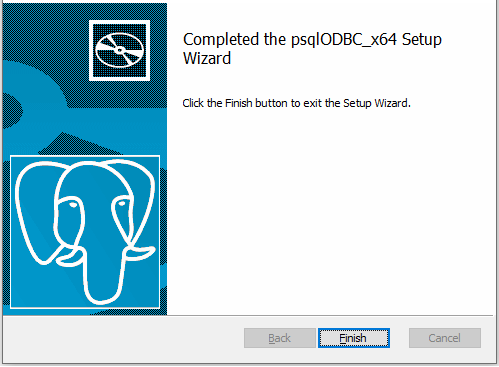
## ODBC

TBase兼容PostgreSQL的ODBC，可以按PostgreSQL配置ODBC的方法，在Windows和linux下配置和使用。

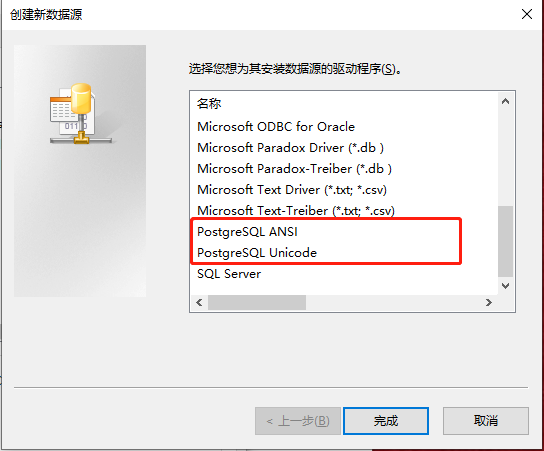
### Windows下安装使用ODBC

TBase可以支持windows10上配置odbc(32位)来访问，odbc下载地址：<https://www.postgresql.org/ftp/odbc/versions/msi/>。

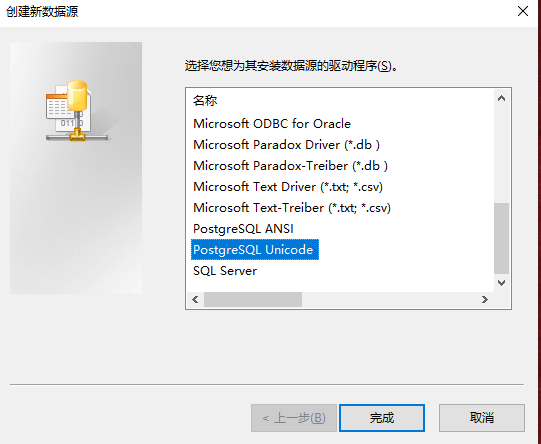
下载32位odbc成后，按照默认安装即可，点击finish完成安装：



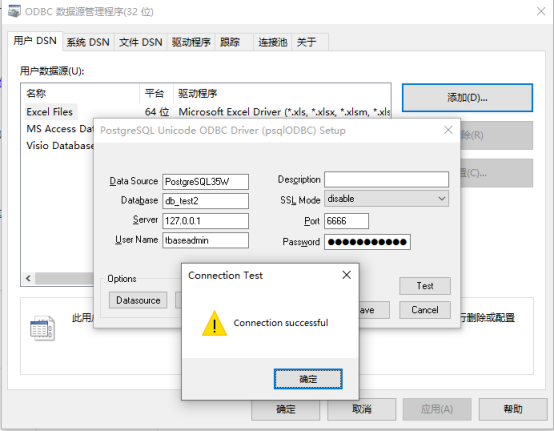
检查是否安装成功，打开控制面板->管理工具->ODBC DATA Source(32-bit)，点击“添加”，显示如下表示安装成功：



选中其中一个驱动类型，点击完成：



填写数据库连接信息，点击Test测试是否能连接成功：



### Linux下安装使用ODBC

#安装odbc驱动

**yum install -y unixODBC**

**yum install -y postgresql-odbc**

#配置odbc数据源

创建 /etc/odbc.ini 文件，修改其中的 Servername、UserName、Password、Port

**vi /etc/odbc.ini**

**[tbase]**

**Description = Test to tbase**

**Driver = PostgreSQL**

**Database = postgres**

**Servername = 172.16.0.65**

**UserName = tbase**

**Password = tbase@TBASE**

**Port = 11345**

**ReadOnly = 0**

**ConnSettings = set client\_encoding to UTF8**

**#odbc接口连接数据**

**isql tbase tbase postgres**

**+---------------------------------------+**

**| Connected! |**

**| |**

**| sql-statement |**

**| help [tablename] |**

**| quit |**

**| |**

**+---------------------------------------+**

**SQL>**

**使用ODBC方式,可以正常操作数据库对象**

**#进行常规的增删改查操作**

**SQL> create table tbl\_odbc\_test(id int,info text);**

**SQLRowCount returns 0**

**SQL> insert into tbl\_odbc\_test(id,info) values(1,'test1'),(2,'test2'),(3,'test3');**

**SQLRowCount returns 3**

**SQL> update tbl\_odbc\_test set info='test33' where id=3;**

**SQLRowCount returns 1**

**SQL> select \* from tbl\_odbc\_test where id=3;**

**+------------+-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------+**

**| id | info |**

**+------------+-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------+**

**| 3 | test33 |**

**+------------+-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------+**

**SQLRowCount returns 1**

**1 rows fetched**

**SQL> delete from tbl\_odbc\_test where id=1;**

**SQLRowCount returns 1**

**SQL> drop table tbl\_odbc\_test;**

**SQLRowCount returns 0**

**SQL>**