

北京海量数据技术股份有限公司

**Vastbase G100 V2.2**

**(Build 10)**

**Oracle兼容性手册**

【版权声明】

©2007-2022 北京海量数据技术股份有限公司 版权所有

本文档著作权归 **北京海量数据技术股份有限公司**（简称“海量数据”）所有，未经海量数据事先书面许可，任何主体不得以任何形式复制、 修改、抄袭、传播全部或部分本文档内容。

北京海量数据技术股份有限公司保留所有的权利。

【服务声明】

本文档意在向客户介绍海量数据全部或部分产品、服务的当时的整体概况，部分产品、 服务的内容可能有所调整。您所购买的产品、服务的种类、服务标准等应由您与海量数据之间的商业合同约定，除非双方另有约定，否则，海量数据对本文档内容不做任何明示或模式的承诺或保证。

**目录**

[1. 函数 12](#_Toc31989)

[1.1. ALL 12](#_Toc23963)

[1.2. COSH 13](#_Toc24788)

[1.3. CHARTOROWID 13](#_Toc12699)

[1.4. DECODE 14](#_Toc4640)

[1.5. GROUPING 16](#_Toc4474)

[1.6. GROUPING\_ID 17](#_Toc9311)

[1.7. INSTRB 21](#_Toc7618)

[1.8. JSON\_MERGEPATCH 22](#_Toc11411)

[1.9. MOD 25](#_Toc7078)

[1.10. MONTHS\_BETWEEN 26](#_Toc5769)

[1.11. NLS\_UPPER 27](#_Toc15740)

[1.12. NLS\_LOWER 28](#_Toc29177)

[1.13. NUMTOYMINTERVAL 29](#_Toc18419)

[1.14. ORA\_HASH 30](#_Toc23907)

[1.15. PERCENTILE\_CONT 31](#_Toc16873)

[1.16. RATIO\_TO\_REPORT 32](#_Toc9479)

[1.17. REMAINDER 34](#_Toc3091)

[1.18. ROUND 35](#_Toc7788)

[1.19. ROWIDTOCHAR 36](#_Toc26330)

[1.20. SINH 37](#_Toc19793)

[1.21. SQLERRM 37](#_Toc24596)

[1.22. SYS\_EXTRACT\_UTC 39](#_Toc1284)

[1.23. SYS\_GUID 40](#_Toc12158)

[1.24. TANH 41](#_Toc4451)

[1.25. TABLE 41](#_Toc42)

[1.26. TO\_DSINTERVAL 43](#_Toc3363)

[1.27. TO\_YMINTERVAL 44](#_Toc22078)

[1.28. TO\_SINGLE\_BYTE 46](#_Toc30477)

[1.29. TO\_MULTI\_BYTE 46](#_Toc31663)

[1.30. TO\_TIMESTAMP\_TZ 47](#_Toc26186)

[1.31. TZ\_OFFSET 48](#_Toc29669)

[1.32. VSIZE 49](#_Toc32254)

[1.33. WM\_CONCAT 50](#_Toc13672)

[1.34. 函数调用时无入参 51](#_Toc698)

[1.35. 函数返回结果集字段带参数列表 53](#_Toc8708)

[2. 内置包 54](#_Toc18580)

[2.1. DBMS\_ALERT 54](#_Toc14602)

[2.2. DBMS\_APPLICATION\_INFO 57](#_Toc9957)

[2.3. DBMS\_ASSERT 59](#_Toc7317)

[2.4. DBMS\_CRYPTO 62](#_Toc18163)

[2.5. DBMS\_JOB 78](#_Toc2556)

[2.6. DBMS\_LOB 81](#_Toc3362)

[2.7. DBMS\_LOCK 95](#_Toc9950)

[2.8. DBMS\_OBFUSCATION\_TOOLKIT 100](#_Toc6229)

[2.9. DBMS\_OUTPUT 110](#_Toc20319)

[2.10. DBMS\_PIPE 113](#_Toc29092)

[2.11. DBMS\_PROFILER 123](#_Toc14232)

[2.12. DBMS\_RANDOM 127](#_Toc5793)

[2.13. DBMS\_ROWID 131](#_Toc332)

[2.14. DBMS\_SQL 133](#_Toc6829)

[2.15. DBMS\_STATS 156](#_Toc12369)

[2.16. DBMS\_UTILITY 161](#_Toc16446)

[2.17. DBMS\_XMLDOM 167](#_Toc31580)

[2.18. DBMS\_XMLPARSER 183](#_Toc12543)

[2.19. URL\_TCP 186](#_Toc9231)

[2.20. UTL\_FILE 199](#_Toc29737)

[2.21. UTL\_HTTP 202](#_Toc6254)

[2.22. UTL\_RAW 209](#_Toc31892)

[2.23. UTL\_SMTP 211](#_Toc28982)

[2.24. UTL\_URL 217](#_Toc13381)

[3. 视图 220](#_Toc15268)

[3.1. oracle视图概述 220](#_Toc22926)

[3.2. 静态数据字典视图 220](#_Toc12971)

[3.2.1. ALL\_ALL\_TABLES 220](#_Toc25075)

[3.2.2. ALL\_CONS\_COLUMNS 225](#_Toc27311)

[3.2.3. ALL\_CONSTRAINTS 225](#_Toc31089)

[3.2.4. ALL\_IND\_COLUMNS 227](#_Toc19514)

[3.2.5. ALL\_IND\_PARTITIONS 228](#_Toc10467)

[3.2.6. ALL\_INDEXES 231](#_Toc23885)

[3.2.7. ALL\_OBJECTS 234](#_Toc17552)

[3.2.8. ALL\_PART\_INDEXES 236](#_Toc5685)

[3.2.9. ALL\_PART\_KEY\_COLUMNS 239](#_Toc3114)

[3.2.10. ALL\_PART\_TABLES 239](#_Toc9743)

[3.2.11. ALL\_SEQUENCES 242](#_Toc28613)

[3.2.12. ALL\_SUBPART\_KEY\_COLUMNS 243](#_Toc27742)

[3.2.13. ALL\_TAB\_COLS 243](#_Toc18836)

[3.2.14. ALL\_TAB\_COLUMNS 246](#_Toc22624)

[3.2.15. ALL\_TAB\_PARTITIONS 248](#_Toc6757)

[3.2.16. ALL\_TAB\_SUBPARTITIONS 251](#_Toc9223)

[3.2.17. ALL\_TABLES 254](#_Toc1851)

[3.2.18. ALL\_TRIGGERS 259](#_Toc20419)

[3.2.19. ALL\_TYPES 261](#_Toc2163)

[3.2.20. ALL\_USERS 261](#_Toc11640)

[3.2.21. ALL\_VIEWS 262](#_Toc32600)

[3.2.22. DBA\_ALL\_TABLES 263](#_Toc31836)

[3.2.23. DBA\_CONS\_COLUMNS 263](#_Toc19732)

[3.2.24. DBA\_CONSTRAINTS 264](#_Toc25745)

[3.2.25. DBA\_IND\_COLUMNS 264](#_Toc7770)

[3.2.26. DBA\_IND\_PARTITIONS 264](#_Toc22706)

[3.2.27. DBA\_INDEXES 265](#_Toc31043)

[3.2.28. DBA\_OBJECTS 265](#_Toc21681)

[3.2.29. DBA\_PART\_KEY\_COLUMNS 265](#_Toc18229)

[3.2.30. DBA\_PART\_TABLES 265](#_Toc27827)

[3.2.31. DBA\_ROLE\_PRIVS 268](#_Toc8902)

[3.2.32. DBA\_ROLES 268](#_Toc29654)

[3.2.33. DBA\_SEGMENTS 268](#_Toc12797)

[3.2.34. DBA\_SEQUENCES 270](#_Toc505)

[3.2.35. DBA\_SUBPART\_KEY\_COLUMNS 270](#_Toc22369)

[3.2.36. DBA\_SYNONYMS 270](#_Toc6405)

[3.2.37. DBA\_TAB\_COLUMNS 271](#_Toc6323)

[3.2.38. DBA\_TAB\_PARTITIONS 271](#_Toc27979)

[3.2.39. DBA\_TAB\_SUBPARTITIONS 271](#_Toc20553)

[3.2.40. DBA\_TABLES 271](#_Toc12780)

[3.2.41. DBA\_TRIGGERS 271](#_Toc9823)

[3.2.42. DBA\_TYPES 272](#_Toc4455)

[3.2.43. DBA\_USERS 272](#_Toc31195)

[3.2.44. DBA\_VIEWS 274](#_Toc31686)

[3.2.45. PRODUCT\_COMPONENT\_VERSION 274](#_Toc23646)

[3.2.46. USER\_IND\_PARTITIONS 274](#_Toc32510)

[3.2.47. USER\_PART\_INDEXES 274](#_Toc9102)

[3.2.48. USER\_PART\_TABLES 274](#_Toc17279)

[3.3. 动态性能视图 275](#_Toc25086)

[3.3.1. V$DATABASE 275](#_Toc22017)

[3.3.2. V$DBLINK 278](#_Toc27687)

[3.3.3. V$INSTANCE 278](#_Toc28952)

[3.3.4. V$LICENSE 279](#_Toc29256)

[3.3.5. V$OBJECT\_USAGE 280](#_Toc19483)

[3.3.6. V$OBSOLETE\_PARAMETER 280](#_Toc15916)

[3.3.7. V$OPTION 280](#_Toc25166)

[3.3.8. V$PARAMETER 281](#_Toc23581)

[3.3.9. V$TABLESPACE 281](#_Toc31871)

[3.3.10. V$VERSION 282](#_Toc28826)

[4. SQL语法参考 282](#_Toc22646)

[4.1. 数据类型及操作符 282](#_Toc18565)

[4.1.1. ^=运算符 282](#_Toc25827)

[4.1.2. BFILE数据类型 283](#_Toc24538)

[4.1.3. BINARY\_FLOAT数据类型 286](#_Toc32172)

[4.1.4. DATE类型的减法与乘法 287](#_Toc15171)

[4.1.5. DATE数据类型 289](#_Toc13260)

[4.1.6. TIMESTAMP数据类型 290](#_Toc4850)

[4.2. SQL语法 291](#_Toc17708)

[4.2.1. GROUP BY支持含有表达式或者函数的列 291](#_Toc429)

[4.2.2. ALL语法 293](#_Toc19469)

[4.2.3. CREATE TABLE语句支持自增列 294](#_Toc16633)

[4.2.4. CREATE TYPE语句支持or replace和{is | as} object子句 295](#_Toc11008)

[4.2.5. Insert All&First子句语法 297](#_Toc954)

[4.2.6. INSERT支持别名 302](#_Toc8586)

[4.2.7. PIVOT和UNPIVOT 303](#_Toc19149)

[4.2.8. Q’转义字符语法 308](#_Toc5852)

[4.2.9. ROWID 310](#_Toc11101)

[4.2.10. ROWNUM 311](#_Toc21314)

[4.2.11. SAMPLE抽样采集语法 312](#_Toc32008)

[4.2.12. SELECT... FOR UPDATE 316](#_Toc26182)

[4.3. SELECT\_LIST中可以使用非分组列 318](#_Toc22033)

[4.4. SELECT语句使用MONTH、CONTENT、PERCENT，PASSWORD关键字做列别名 320](#_Toc9869)

[4.5. WITH FUNCTION子句 321](#_Toc11823)

[4.6. 递归查询语法 323](#_Toc25081)

[4.7. 虚拟列语法 330](#_Toc1205)

[4.8. 字符串CHAR/VARCHAR/VARCHAR2指定BYTE和CHAR关键字 331](#_Toc18337)

[4.9. 多个子查询时不需要指定表别名 332](#_Toc3373)

[5. 宏变量 334](#_Toc8844)

[6. PL/SQL 335](#_Toc15771)

[6.1. CREATE FUNCTION中支持PIPELINED和PIPE ROW语法 335](#_Toc29531)

[6.2. EXCEPTION\_INIT预定义异常处理 336](#_Toc13312)

[6.3. PL/pgSQL-TYPE ... IS RECORD 337](#_Toc10319)

[6.4. RAISE\_APPLICATION\_ERROR存储过程 341](#_Toc21724)

[6.5. 创建函数、存储过程、触发器和包时进行编译检查 342](#_Toc28221)

[6.6. 定义游标支持INTO关键字 345](#_Toc21171)

[6.7. 关联数组 347](#_Toc10870)

[6.8. 嵌套表 348](#_Toc3822)

[6.9. 函数与存储过程传入参数与查询字段同名 349](#_Toc1676)

[6.10. 匿名块支持JDBC绑定变量输入输出 352](#_Toc24953)

[6.11. 数组支持RECORD类型 354](#_Toc20856)

[6.12. 自治事务 355](#_Toc20634)

[6.13. 支持oracle的包（Package）和包体（Package body） 360](#_Toc12678)

[6.14. 支持PLSQL-BULK COLLECT INTO 362](#_Toc20666)

[6.15. package中支持函数返回嵌套表和关联数组类型 372](#_Toc22116)

[6.16. package中支持%type语法 375](#_Toc24578)

[6.17. package中支持同义词引用对象语法 376](#_Toc26113)

[6.18. package中支持类型调用 380](#_Toc21353)

[6.19. package中使用游标创建type 384](#_Toc5430)

[6.20. package中支持type...is语法 386](#_Toc22835)

[6.21. 支持record类型作为procedure出参 389](#_Toc15033)

[6.22. 支持type自带构造器使用及new构造器语法 393](#_Toc28836)

[7. 对象管理 395](#_Toc22173)

[7.1. 分区表 395](#_Toc2031)

[7.1.1. 支持全局分区索引 395](#_Toc6901)

[7.1.2. 兼容interval分区表 397](#_Toc15075)

[7.1.3. 支持default分区 399](#_Toc9800)

[7.2. 触发器 401](#_Toc16375)

[7.2.1. 兼容IF Inserting THEN,IF updating THEN,IF deleting THEN 401](#_Toc26369)

[7.2.2. 兼容IF UPDATING('col1') 405](#_Toc16020)

[7.2.3. 允许INSERT使用oid.colname获取值为空 406](#_Toc5883)

[7.3. 序列 408](#_Toc24785)

[7.3.1. SEQUENCE兼容RESTART语法 408](#_Toc30941)

[7.3.2. 兼容NO CACHE、NOCACHE、ORDER、NO ORDER、NOORDER关键字 409](#_Toc17410)

[8. DBLINK库级访问 410](#_Toc3298)

[9. OCI/OCCI接口 412](#_Toc18119)

[10. VPD功能 414](#_Toc24887)

[11. 闪回/闪回查询 417](#_Toc26747)

[12. Xmltype模块 419](#_Toc549)

# 函数

## ALL

**功能描述**

ALL函数用于判断数组中所有元素都是否为真（非零或true）。用逗号分隔的方式输入数组表达式的各个分量。

**语法格式**

op ALL(ARRAY[expr,])

**参数说明**

* op：指定的运算符，诸如‘>’，‘<’，‘=’等等。
* ARRAY：指显示指定数组表达式的关键字，中括号内部可以指定多个表达式。

**示例**

1、创建测试表并插入数据。

create table testall(id int,name char(10));

insert into testall values(1,'amy');

insert into testall values(2,'tom');

insert into testall values(3,'jack');

insert into testall values(97,'jane');

insert into testall values(98,'alice');

insert into testall values(99,'taylor');

2、使用ALL函数。

select \* from testall where id >= all(2,3);

返回结果为：

id | name

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

3 | jack

97 | jane

98 | alice

99 | taylor

(4 rows)

## COSH

**功能描述**

该函数用于计算输入参数的双曲余弦值，并返回结果。

**语法格式**

cosh(arg)

**参数说明**

arg：输入float类型数值。

**示例**

select cosh(1);

返回结果为：

cosh

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

1.54308063481524

(1 row)

## CHARTOROWID

**功能描述**

该函数用于字符串类型转换为将rowid类型。

**语法格式**

chartorowid(char)

**参数说明**

char：输入的字符串值，支持输入CHAR,VARCHAR2,NCHAR,NVARCHAR2的数据类型。

**注意事项**

无。

**示例**

1、创建测试表并插入数据。

DROP TABLE testrowid;

create table testrowid (c1 int,c2 varchar(20));

insert into testrowid values(1,'Lucy');

insert into testrowid values(2,'Sally');

insert into testrowid values(3,'Ben');

insert into testrowid values(4,'Black');

2、查询rowid。

select rowid from testrowid order by rowid;

rowid

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

fksAAA==AAAAAA==AQA=

fksAAA==AAAAAA==AgA=

fksAAA==AAAAAA==AwA=

fksAAA==AAAAAA==BAA=

(4 rows)

3、调用CHARTOROWID函数（其中单引号中的内容为步骤3中的最后一条数据）。

select \* from testrowid where rowid < CHARTOROWID('fksAAA==AAAAAA==BAA=');

返回结果为：

c1 | c2

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

1 | Lucy

2 | Sally

3 | Ben

(3 rows)

## DECODE

**功能描述**

decode函数将输入数值与函数中的对应参数相比较，若与其中一个参数相等则根据输入值返回一个对应的结果。

函数的参数列表是由输入值，若干用于比较的数值及其对应结果值和默认值组成的。假如输入值未能与任何一个比较数值匹配成功，则函数返回默认值。

**语法格式**

DECODE(expr, search1, result1 [, search2, result2 ]... [, default ])

**参数说明**

* expr：可以是数值类型，也可以是字符类型。
* 若expr=search1，则输出result1，expr=search1，输出result1，expr=search2，输出result2……若expr不等于所列出的所有value，则输出为default。

**示例**

调用函数。

* 输入expr=search1，返回result1。

select decode(1,1,'2004-05-07 13:23:44',0,4,5);

当结果显示为如下信息，则表示验证成功

case

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

2004-05-07 13:23:44

(1 row)

* 输入expr=search2返回result2。

select decode(0,1,'2004-05-07 13:23:44',0,4,5);

当结果显示为如下信息，则表示验证成功

case

------

4

(1 row)

## GROUPING

**功能描述**

GROUPING函数用于区分分组后的普通行和聚合行。如果是聚合行，则返回1，反之，则是0。

**语法格式**

GROUPING( col\_expr )

**参数说明**

expr：列名。

**注意事项**

该函数仅适用于select语句并包含GROUP BY的情况。

**示例**

1、创建测试表并插入数据。

create table group\_test (group\_id int, job varchar2(10), name varchar2(10), salary int);

insert into group\_test values (10,'Coding', 'Bruce',1000);

insert into group\_test values (10,'Programmer','Clair',1000);

insert into group\_test values (10,'Architect', 'Gideon',1000);

insert into group\_test values (10,'Director', 'Hill',1000);

insert into group\_test values (20,'Coding', 'Jason',2000);

insert into group\_test values (20,'Programmer','Joey',2000);

insert into group\_test values (20,'Architect', 'Martin',2000);

insert into group\_test values (20,'Director', 'Michael',2000);

insert into group\_test values (30,'Coding', 'Rebecca',3000);

insert into group\_test values (30,'Programmer','Rex',3000);

insert into group\_test values (30,'Architect', 'Richard',3000);

insert into group\_test values (30,'Director', 'Sabrina',3000);

insert into group\_test values (40,'Coding', 'Samuel',4000);

insert into group\_test values (40,'Programmer','Susy',4000);

insert into group\_test values (40,'Architect', 'Tina',4000);

insert into group\_test values (40,'Director', 'Wendy',4000);

2、grouping函数：传入两个expr参数，使用GROUP BY ROLLUP生成分组集.

select group\_id,job,grouping(group\_id),grouping(job),grouping(group\_id,job),sum(salary) from group\_test group by rollup(group\_id, job);

返回结果为：

group\_id | job | grouping | grouping | grouping | sum

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

10 | Architect | 0 | 0 | 0 | 1000

10 | Coding | 0 | 0 | 0 | 1000

10 | Director | 0 | 0 | 0 | 1000

10 | Programmer | 0 | 0 | 0 | 1000

10 | | 0 | 1 | 1 | 4000

20 | Architect | 0 | 0 | 0 | 2000

20 | Coding | 0 | 0 | 0 | 2000

20 | Director | 0 | 0 | 0 | 2000

20 | Programmer | 0 | 0 | 0 | 2000

20 | | 0 | 1 | 1 | 8000

30 | Architect | 0 | 0 | 0 | 3000

30 | Coding | 0 | 0 | 0 | 3000

30 | Director | 0 | 0 | 0 | 3000

30 | Programmer | 0 | 0 | 0 | 3000

30 | | 0 | 1 | 1 | 12000

40 | Architect | 0 | 0 | 0 | 4000

40 | Coding | 0 | 0 | 0 | 4000

40 | Director | 0 | 0 | 0 | 4000

40 | Programmer | 0 | 0 | 0 | 4000

40 | | 0 | 1 | 1 | 16000

| | 1 | 1 | 3 | 40000

(21 rows)

## GROUPING\_ID

**功能描述**

GROUPING\_ID函数相当于将多个GROUPING函数的结果串接成二进制数，返回这个二进制数对应的十进制数。

**语法格式**

GROUPING\_ID(expr [, expr ]...)

**参数说明**

expr：列名。

需要在GROUP BY表达式中，该函数返回由1和0组成的位向量对应的数字，指示哪些GROUP BY表达式不包括在当前分组集中。如果对应的expr包含在生成当前结果行的分组集中（常规分组行），则位为0，如果不包含(超聚合行，GROUPBYROLLUP/CUBE等生成）则为1。当有多个expr时，如2个expr，可能的返回值为：00 ->0，01 ->1，10-> 2，11 -> 3。

**注意事项**

* 该函数仅适用于select语句并包含GROUP BY的情况。
* grouping\_id的字段，必须是group by包含的字段

**示例**

1、创建测试表并插入数据。

create table group\_test (group\_id int, job varchar2(10), name varchar2(10), salary int);

insert into group\_test values (10,'Coding', 'Bruce',1000);

insert into group\_test values (10,'Programmer','Clair',1000);

insert into group\_test values (10,'Architect', 'Gideon',1000);

insert into group\_test values (10,'Director', 'Hill',1000);

insert into group\_test values (20,'Coding', 'Jason',2000);

insert into group\_test values (20,'Programmer','Joey',2000);

insert into group\_test values (20,'Architect', 'Martin',2000);

insert into group\_test values (20,'Director', 'Michael',2000);

insert into group\_test values (30,'Coding', 'Rebecca',3000);

insert into group\_test values (30,'Programmer','Rex',3000);

insert into group\_test values (30,'Architect', 'Richard',3000);

insert into group\_test values (30,'Director', 'Sabrina',3000);

insert into group\_test values (40,'Coding', 'Samuel',4000);

insert into group\_test values (40,'Programmer','Susy',4000);

insert into group\_test values (40,'Architect', 'Tina',4000);

insert into group\_test values (40,'Director', 'Wendy',4000);

2、grouping函数：传入两个expr参数，使用GROUP BY ROLLUP生成分组集.

select group\_id,job,grouping(group\_id),grouping(job),grouping(group\_id,job),sum(salary) from group\_test group by rollup(group\_id, job);

返回结果为：

group\_id | job | grouping | grouping | grouping | sum

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

10 | Architect | 0 | 0 | 0 | 1000

10 | Coding | 0 | 0 | 0 | 1000

10 | Director | 0 | 0 | 0 | 1000

10 | Programmer | 0 | 0 | 0 | 1000

10 | | 0 | 1 | 1 | 4000

20 | Architect | 0 | 0 | 0 | 2000

20 | Coding | 0 | 0 | 0 | 2000

20 | Director | 0 | 0 | 0 | 2000

20 | Programmer | 0 | 0 | 0 | 2000

20 | | 0 | 1 | 1 | 8000

30 | Architect | 0 | 0 | 0 | 3000

30 | Coding | 0 | 0 | 0 | 3000

30 | Director | 0 | 0 | 0 | 3000

30 | Programmer | 0 | 0 | 0 | 3000

30 | | 0 | 1 | 1 | 12000

40 | Architect | 0 | 0 | 0 | 4000

40 | Coding | 0 | 0 | 0 | 4000

40 | Director | 0 | 0 | 0 | 4000

40 | Programmer | 0 | 0 | 0 | 4000

40 | | 0 | 1 | 1 | 16000

| | 1 | 1 | 3 | 40000

(21 rows)

3、调用grouping\_id函数：grouping\_id函数：传入一个expr参数，使用GROUP BY CUBE生成分组集。

select group\_id,job,grouping\_id(group\_id),grouping\_id(job),sum(salary) from group\_test group by cube(group\_id, job);

当返回结果如下时，则表示支持grouping\_id函数

group\_id | job | grouping | grouping | sum

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

10 | Architect | 0 | 0 | 1000

10 | Coding | 0 | 0 | 1000

10 | Director | 0 | 0 | 1000

10 | Programmer | 0 | 0 | 1000

10 | | 0 | 1 | 4000

20 | Architect | 0 | 0 | 2000

20 | Coding | 0 | 0 | 2000

20 | Director | 0 | 0 | 2000

20 | Programmer | 0 | 0 | 2000

20 | | 0 | 1 | 8000

30 | Architect | 0 | 0 | 3000

30 | Coding | 0 | 0 | 3000

30 | Director | 0 | 0 | 3000

30 | Programmer | 0 | 0 | 3000

30 | | 0 | 1 | 12000

40 | Architect | 0 | 0 | 4000

40 | Coding | 0 | 0 | 4000

40 | Director | 0 | 0 | 4000

40 | Programmer | 0 | 0 | 4000

40 | | 0 | 1 | 16000

| | 1 | 1 | 40000

| Architect | 1 | 0 | 10000

| Coding | 1 | 0 | 10000

| Director | 1 | 0 | 10000

| Programmer | 1 | 0 | 10000

(25 rows)

## INSTRB

**功能描述**

INSTRB函数用于搜索一个字符串在指定的字符串中出现的位置，起始搜索位置和出现的次数（第几次）可以通过参数来指定。

当起始搜索位置为负数时，表示从指定字符串的尾部往回开始搜索。返回值为字符串在被搜索的字符串中出现的位置，该位置是按字节数来计算。如果未搜索到字符串，则返回0。

**语法格式**

instrb(C1,C2[,I[,J]]);

**参数说明**

* C1：varchar2类型，被搜索的字符串。
* C2：varchar2类型，指定的字符串。
* I：integer类型，起始搜索位置，默认为1。
* J：integer类型，第J次出现的位置，默认为1。

**注意事项**

无。

**示例**

示例一：默认编码为utf8，调用instrb函数。

select instrb('CORPORATE FLOOR','OR');

返回结果为：

instrb

-------

2

(1 row)

示例二：

select instrb('CORPORATE FLOOR','OR',5,2);

返回结果为：

instrb

-------

14

(1 row)

示例三：

select instrb('CORPORATE FLOOR','OR',-3,2);

返回结果为：

instrb

-------

2

(1 row)

示例四：

select instrb('测试CORPORATE FLOOR','OR');

utf8编码格式下一个中文为三个字节，故返回结果为:

instrb

-------

8

(1 row)

## JSON\_MERGEPATCH

**功能描述**

json\_mergepatch函数可查询和更新json文档的特定部分。

**语法格式**

JSON\_MERGEPATCH

(target\_expr,patch\_expr[returning\_clause] )

**参数说明**

* target\_expr：需要修改的目标表达式（以下简称源）。
* patch\_expr：一个json片段，表达要合并到目标表达式中的补丁（以下简称补丁）。
* json\_returning\_clause：指定操作符的返回类型。默认的返回类型是text。

**注意事项**

* 使用该函数需要具备该函数的访问权限，例如json\_mergepatch函数如果位于pg\_catalog模式下，用户需要具备以下权限：
* pg\_catalog模式的usage权限。
* json\_mergepatch函数的excute权限。
* json的数据类型有：空值null、布尔值false/true、字符串”string”、数组[]和对象{}。数组[]和对象{}称为复杂数据类型，可嵌套使用。
* 补丁必须为数组或者对象，否则会报错。
* 补丁为数组时：
* 源是空值null、布尔值false/true或者字符串”string”时，返回null。
* 源为数组或者对象时，使用补丁替换源。
* 补丁为对象时：
* 源是空值null、布尔值false/true或者字符串”string”时，返回null。
* 源为数组时，去除补丁中值为null的键值对后再替换源。
* 源为对象时，按照如下两个对象合并规则合并。
* 两个对象合并规则：
* 如果补丁不是JSON对象。使用补丁替换源（补丁值为null时，若源中没有对应补丁的键，
* 则该键值对不能出现在结果中）。
* 如果补丁是JSON对象，请执行以下操作:
  + (1)如果源不是对象，则将其视为空对象({})。
  + (2)迭代补丁对象的(p-field:p-value)成员。
* 如果补丁成员的p-value为null，则从源中删除相应的成员。
* 否则，递归：源对应的值和p-value合并的结果，替换相应源字段的值。
* Vastbase处理json数据时会对object的键值对中的键进行排序和去除相同键，因此object中的键值对和输入时的顺序可能会不一样。Vastbase注重合并结果，不保证顺序或者空格的使用上和Oracle完全一致。
* 隐式转化规则新增了text和json/jsonb相互转化外。

**示例**

1、创建测试表并插入数据。

create table json\_mergepatch\_test(Id bigint,Data json );

INSERT INTO json\_mergepatch\_test (id,data) values(1,'{"id":1,"first\_name":"Iron","last\_name":"Man"}');

2、查询数据。

SELECT JSON\_MERGEPATCH(data, '{"last\_name":"banana"}') AS data FROM json\_mergepatch\_test;

返回结果为：

data

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

{"id": 1, "last\_name": "banana", "first\_name": "Iron"}

(1 row)

3、更新表中的数据。

UPDATE json\_mergepatch\_test a SET a.data = JSON\_MERGEPATCH(a.data, '{"last\_name":"banana","new\_element":"surprise"}');

4、查询数据。

SELECT data FROM json\_mergepatch\_test;

返回结果为：

data

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

{"id": 1, "last\_name": "banana", "first\_name": "Iron", "new\_element": "surprise"}

(1 row)

## MOD

**功能描述**

mod(nExp1,nExp2)函数是一个求余函数，即nExp1（被除数）除以nExp2（除数）运算后的余数。

**语法格式**

mod(nExp1,nExp2)

**参数说明**

* nExp2：除数
* nExp1：被除数

**注意事项**

* 两个整数求余时，其值的符号与除数（nExp2）的符号相同，取值为将两个整数看作是正数，再作除法运算。
* 能整除时，其值为0。
* 不能整除时，其值=除数×（整商+1）-被除数
* 例：mod(36,-10)，结果为将36除以10的整商为3，因此结果=10×(3+1)-36=4，取除数（-10）的符号，最终结果-4。
* 两个小数求余，其值=被除数-（整商×除数）之后在第一位小数位进行四舍五入。
* 例：mod(9,1.2)，结果为9除以1.2其整商为7，因此结果9-(7×1.2)=0.6。最终结果0.6。

**示例**

select mod(36,-10);

返回结果为：

mod

-----

6

(1 row)

## MONTHS\_BETWEEN

**功能描述**

该函数用于返回两个日期之间的月份数。

**语法格式**

MONTHS\_BETWEEN(date1, date2)

**参数说明**

date1/date2：指定时间值。取值范围是date或TIMESTAMP WITH TIME ZONE

**注意事项**

* 如果date1比date2晚，则返回结果为正数，反之则为负数，即返回结果为date1 减去 date2 的差值。
* 如果date1和date2都为其所在月份的第几天，或者都为某个月份的最后一天，那么返回值为一个整数。否则，返回的是一个浮点数，整数部分为两个时间差中完整月份的差数，小数部分则为剩余时间差与一个月（31天）的比例值。
* 该功能仅在数据库兼容模式为Oracle时能够使用（即创建DB时DBCOMPATIBILITY='A'），在其他数据库兼容模式下不能使用该特性。

**示例**

select months\_between('2020-05-20'::date,'2020-03-20'::date);

返回结果为：

months\_between

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

2

(1 row)

## NLS\_UPPER

**功能描述**

nls\_upper函数用于将字符转换为大写。

**语法格式**

NLS\_UPPER(char [, 'nlsparam' ])

**参数说明**

* char：需要转换的字符串，可以是char、varchar2、nchar、nvarchar2和clob。
* nlsparam：可选参数，不区分大小写，用于指定排序的方式，常见的有SCHINESE\_RADICAL\_M（部首、笔画）排序、SCHINESE\_STROKE\_M（笔画、部首)排序，SCHINESE\_PINYIN\_M（拼音）排序。

**注意事项**

使用该函数需要具备该函数的访问权限，例如nls\_upper函数如果位于pg\_catalog模式下，用户需要具备以下权限：

* pg\_catalog模式的usage权限
* sys\_extract\_utc函数的excute权限

**示例**

将字符转化为大写。

select nls\_upper('asdasd') from dual;

返回结果为：

nls\_upper

-----------

ASDASD

(1 row)

## NLS\_LOWER

**功能描述**

nls\_lower函数用于将字符转换为小写。

**语法格式**

nls\_lower语法：

NLS\_LOWER(char [, 'nlsparam' ])

**参数说明**

* char：需要转换的字符串，可以是char、varchar2、nchar、nvarchar2和clob。
* nlsparam：可选参数，不区分大小写，用于指定排序的方式，常见的有SCHINESE\_RADICAL\_M（部首、笔画）排序、SCHINESE\_STROKE\_M（笔画、部首)排序，SCHINESE\_PINYIN\_M（拼音）排序。

**注意事项**

使用该函数需要具备该函数的访问权限，例如nls\_lower函数如果位于pg\_catalog模式下，用户需要具备以下权限：

* pg\_catalog模式的usage权限
* sys\_extract\_utc函数的excute权限

**示例**

将字符转化为小写。

select nls\_lower('SADAS') from dual;

返回结果为：

nls\_lower

-----------

sadas

(1 row)

## NUMTOYMINTERVAL

**功能描述**

NUMTOYMINTERVAL函数将数字n转换为年或者月的间隔。

**语法格式**

NUMTOYMINTERVAL(n,'interval\_unit')

**参数说明**

* n：可以是任何NUMBER值或可以隐式转换为NUMBER值的表达式（n为NULL，输出为空）。
* interval\_unit：可以是char、varchar2、nchar或nvarchar2数据类型。interval\_unit的值指定n
* 的单位，并且必须解析为以下字符串值之一（不区分大小写）：
* YEAR：取值范围[-2147483648，2147483647]
* MONTH：取值范围[-178956970.7,178956970.6]

**功能描述**

使用该函数需要具备该函数的访问权限，例如numtoyminterval函数如果位于pg\_catalog模式下，用户需要具备以下权限：

* pg\_catalog模式的usage权限
* numtoyminterval函数的excute权限

**兼容性**

Vastbase G100 V2.2版本与Oracle 11g R2版本相比的差异，Oracle中interval\_unit指定n单位，并且必须解析为以下字符串值之一：DAY，HOUR，MINUTS，SECOND。

**示例**

select sysdate,sysdate+numtoyminterval(3,'year') as res from dual;

返回结果为：

sysdate | res

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

2022-07-14 16:13:19 | 2025-07-14 16:13:19

(1 row)

## ORA\_HASH

**功能描述**

该函数用于为给定表达式计算哈希值。

**语法格式**

ora\_hash（expr[,max\_bucket[,seed\_value]])

**参数说明**

* expr：需要计算hash值的表达式或者列。
* max\_bucket：哈希函数返回的最大桶值，范围0到4294967295，默认为4294967295。
* seed\_value：为同一组数据生成许多不同的结果，范围0到4294967295，默认为0。当seed\_value值不一样时，产生的结果值不一样。

**注意事项**

该功能仅在数据库兼容模式为Oracle时能够使用（即创建DB时DBCOMPATIBILITY='A'），在其他数据库兼容模式下不能使用该特性。

**示例**

1、创建测试表并插入数据。

drop table test\_hash ;

create table test\_hash(c1 int,c2 varchar(20));

insert into test\_hash values(1,'test1');

insert into test\_hash values(2,'test2');

2、调用ORA\_HASH函数。

select ora\_hash(c2,12,11) from test\_hash;

返回结果为：

ora\_hash

----------

5

6

(2 rows)

## PERCENTILE\_CONT

**功能描述**

percentile\_cont是一个假定连续分布模型的逆分布函数，该函数具有一个百分比比值和一个排序规范，其返回值是在排序规范的给定给定百分比值范围内的内插值。

**语法格式**

PERCENTILE\_CONT(percentile)

WITHIN GROUP (ORDER BY expr DESC/ASC)

OVER([PARTITION BY expr\_list])

**参数说明**

* percentile：数值类型的常量，取值范围为0-1之间(包含0和1)。
* WITHIN GROUP：指定了分组元素的排序方式。
* OVER：语句中指定了分组方式。
* DESC/ASC：asc是指定列按升序排列，desc则是指定列按降序排列。

**示例**

SELECT color,percentile\_cont(0.5) WITHIN GROUP (ORDER BY value) over(partition by color) from (VALUES('red',1),('red',3),('blue',2),('blue',4)) v(color,value);

返回结果如下，则表示支持该函数：

color ｜ percrntile\_cont

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

blue | 3

blue | 3

red | 2

red | 2

(4 rows)

## RATIO\_TO\_REPORT

**功能描述**

RATIO\_TO\_REPORT函数是一个分析函数，主要功能根据over窗口函数的作用区间，求出作用区间中的单个值（expr）在整个区间的总值的比重。

注：如果over（）中为空，就是对整张表的一个分组处理，如果over（coln）中写了分组字段名，就按coln字段分组进行运算。

**语法格式**

RATIO\_TO\_REPORT(expr) OVER([query\_partition\_clause])

**参数说明**

* expr：表达式。
* OVER：对应关键字。
* query\_partition\_clause：使用query\_partition\_clause子句定义窗口进行计算。语法格式为partition by col，其中，col为用于分组的列名。如果省略该子句，会计算房前值占所有值的比率。

**注意事项**

无。

**示例**

1、创建测试表并插入数据。

create table day\_empsalary (

depname varchar,

empno bigint,

salary int,

working\_days int

);

insert into day\_empsalary values

('develop', 10, 5200, 12),

('sales', 1, 5000, 10),

('personnel', 5, 3500, 11),

('sales', 4, 4800, 5),

('personnel', 2, 3900, 10),

('develop', 7, 4200, 2),

('develop', 9, 4500, 3),

('sales', 3, 4800, 4),

('develop', 8, 6000, 3),

('develop', 11, 5200, 2);

2、调用分析函数（省略query\_partition\_clause子句）

select depname, empno, salary, ratio\_to\_report(salary) over () from day\_empsalary order by 1,2;

返回结果为:

depname | empno | salary | ratio\_to\_report

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

develop | 7 | 4200 | .089171974522293

develop | 8 | 6000 | .127388535031847

develop | 9 | 4500 | .0955414012738854

develop | 10 | 5200 | .110403397027601

develop | 11 | 5200 | .110403397027601

personnel | 2 | 3900 | .0828025477707006

personnel | 5 | 3500 | .0743099787685775

sales | 1 | 5000 | .106157112526539

sales | 3 | 4800 | .101910828025478

sales | 4 | 4800 | .101910828025478

(10 rows)

3、调用分析函数（不省略query\_partition\_clause子句）

select depname, empno, salary, ratio\_to\_report(salary) over (partition by depname) from day\_empsalary order by 1,2;

返回结果为：

depname | empno | salary | ratio\_to\_report

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

develop | 7 | 4200 | .167330677290837

develop | 8 | 6000 | .239043824701195

develop | 9 | 4500 | .179282868525896

develop | 10 | 5200 | .207171314741036

develop | 11 | 5200 | .207171314741036

personnel | 2 | 3900 | .527027027027027

personnel | 5 | 3500 | .472972972972973

sales | 1 | 5000 | .342465753424658

sales | 3 | 4800 | .328767123287671

sales | 4 | 4800 | .328767123287671

(10 rows)

## REMAINDER

**功能描述**

remainder(n2,n1)函数的主要功能是求余数，该函数返回第一个参数除以第二个参数的余数。

**语法格式**

REMAINDER(n2, n1)

**参数说明**

n1,n2：为数值类型或者可以隐式转换为数值类型的类型。

**注意事项**

* 使用该函数需要具备该函数的访问权限，例如remainder函数如果位于pg\_catalog模式下，用户需要具备以下权限：
* pg\_catalog模式的usage权限
* remainder函数的excute权限
* 如果n1=0或n2=infinity（无穷大），返回结果为：
* 如果参数的类型为number，报错。
* 如果参数是binary\_float或binary\_double，则为NaN。
* 如果n1!=0，则余数为n2-(n1\*N)，其中N是最接近n2/n1的整数。如果n2/n1等于x.5，则N是最接近的偶数。例如：

在remainder(3.5,2)中，n2=3.5,n1=2，按计算规则，n2/n1=3.5/2=1.75,最接近的整数N为2。n2-(n1\*N)=3.5-(2\*2)=-0.5

* 如果n2是一个浮点数，如果余数是0，那么余数的符号就是n2的符号。对于number值，余数0是无符号的。

**示例**

select remainder(3.3,2);

返回结果为：

remainder

-----------

-.7

(1 row)

## ROUND

**功能描述**

该函数可以对日期（date/timestamp)进行处理，返回四舍五入的结果。

**语法格式**

ROUND(date [, fmt ]);

ROUND(timestamp [with[without] time zone] [, fmt ]);

**参数说明**

* date：输入date类型数值。
* timestamp：输入timestamp类型数值。
* fmt ：返回日期或时间四舍五入到格式模型fmt指定的单位。

**示例**

1、使用round（date）函数。

select ROUND('2020-11-18 16:46:52'::date,'yy');

返回结果为：

round

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

2021-01-01 00:00:00

(1 row)

2、使用round（timestamp）函数。

select ROUND('2020-11-18 16:46:52+08'::timestamp with time zone,'hh');

返回结果为：

round

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

2020-11-18 17:00:00+08

(1 row)

## ROWIDTOCHAR

**功能描述**

该函数用于将rowid类型转换为字符串类型。

**语法格式**

rowidtochar(rowid)

**参数说明**

rowid：输入的rowid值。

**注意事项**

无。

**示例**

1、创建测试表并插入数据。

drop table testrowid;

create table testrowid(c1 int,c2 varchar(20));

insert into testrowid values(1,'Lucy');

insert into testrowid values(2,'Sally');

2、调用 rowidtochar函数。

select rowidtochar(rowid) from testrowid;

返回结果如下：

rowidtochar

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

hksAAA==AAAAAA==AQA=

hksAAA==AAAAAA==AgA=

(2 rows)

## SINH

**功能描述**

该函数用于计算输入参数的双曲正弦值，并返回结果。

**语法格式**

sinh(arg)

**参数说明**

arg：输入float类型数值。

**示例**

select sinh(33);

返回结果为：

sinh

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

107321789892958

(1 row)

## SQLERRM

**功能描述**

sqlerrm函数的主要功能是根据入参错误号匹配RAISE\_APPLICATION\_ERROR(error\_code, message)预定义函数定义的错误信息。

**前提条件**

该功能仅在数据库兼容模式为Oracle时能够使用（即创建DB时DBCOMPATIBILITY='A'），在其他数据库兼容模式下不能使用该特性。

**语法格式**

SQLERRM( error\_code number)

**参数说明**

error\_code：错误编号。

**使用场景**

* 当传入的错误编号在-20000到-20999时候，输出RAISE\_APPLICATION\_ERROR(error\_code, message)预定义函数的错误信息。
* 传入的错误编号不在-20000到-20999时候，输出Vastbase自身的错误信息。

**示例**

如果是-20000到-20999，那么raise\_application\_error定义的错误信息，进行输出对应结果。

1、创建兼容Oracle的数据库db\_oracle。

CREATE DATABASE db\_oracle DBCOMPATIBILITY='A';

\c db\_oracle

2、创建存储过程

drop procedure if exists exp\_sqlerrm;

set serveroutput on;

create procedure exp\_sqlerrm(

t1 in date,

t2 in date

)is

begin

if t1<t2 then

raise\_application\_error(-20000,'testErr---Date t1 is earlier than t2.');

end if;

end;

/

3、调用存储过程，打印预定义的错误信息

DECLARE

past\_due EXCEPTION;

PRAGMA EXCEPTION\_INIT(past\_due,-20000);

BEGIN

exp\_sqlerrm('2020-01-01',sysdate);

EXCEPTION

WHEN past\_due THEN

DBMS\_OUTPUT.PUT\_LINE(SQLERRM(-20000));

END;

/

结果返回如下：

testErr---Date t1 is earlier than t2.

## SYS\_EXTRACT\_UTC

**功能描述**

sys\_extract\_utc函数将一个带有时区的时间对象，转变成其等效的国际标准时间（格林威治时间）。即把timestamp with time zone类型，转成timestamp without time zone类型。

**语法格式**

SYS\_EXTRACT\_UTC(datetime\_with\_timezone)

**参数说明**

datetime\_with\_timezone：timestamptz（即timestamp with time zone）类型。

**注意事项**

使用该函数需要具备该函数的访问权限，例如sys\_extract\_utc函数如果位于pg\_catalog模式下，用户需要具备以下权限：

* pg\_catalog模式的usage权限
* sys\_extract\_utc函数的excute权限

**示例**

1、创建测试表。

DROP TABLE t\_utc;

CREATE TABLE t\_utc(c1 timestamptz);

2、向表中插入带有时区的数据。

insert into t\_utc values('2022-02-02 10:00:00 +4:00');

insert into t\_utc values('2022-02-03 12:00:00 -2:00');

insert into t\_utc values('2010-01-03 11:09:00 +0:00');

3、查询数据。

select sys\_extract\_utc(c1) from t\_utc;

返回结果为：

sys\_extract\_utc

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

2022-02-02 06:00:00

2022-02-03 14:00:00

2010-01-03 11:09:00

(3 rows)

## SYS\_GUID

**功能描述**

SYS\_GUID可以生成一个全局唯一的字符串数据值。生成的依据主要是时间和随机数，具有全局唯一性。

**语法格式**

select \* from sys\_guid();

**注意事项**

无。

**示例**

select \* from sys\_guid();

其返回结果如下：

sys\_guid

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

48a5e783-bd94-4038-9547-0d8611aa8a33

(1 row)

## TANH

**功能描述**

该函数用于计算输入参数的双曲正切值，并返回结果。

**语法格式**

tanh(arg)

**参数说明**

arg：输入float类型数值。

**示例**

select tanh(9);

返回结果为：

select tanh(9);

返回结果为：

tanh

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

.999999969540041

(1 row)

## TABLE

**功能描述**

table表函数用于查询函数返回的结果集，并将结果集转换为表的形式（就如同查询普通表一样查询返回的结果集）。

表函数目前可以接受的输入参数有：查询语、游标、嵌套表等集合类型。

**语法格式**

SELECT \* FROM table(arg);

**参数说明**

arg：结果集。

**示例**

**示例1：**

1、创建测试表并插入数据。

create table testtable(id int);

insert into testtable values(123),(1234);

2、使用表函数查询结果。

SELECT \* FROM table(testtable);

返回结果为：

id

------

123

1234

(2 rows)

**示例2：**table()结合数组

create or replace type t\_test as object(

id integer,rq date,mc varchar2(60));

create or replace type t\_test\_table as table of t\_test;

create or replace function f\_test\_array(n in number default null) return t\_test\_table

as

v\_test t\_test\_table;

begin

for i in 1 .. nvl(n,100) loop

v\_test(i) := (i,sysdate,'mc'||i);

end loop;

return v\_test;

end;

/

select \* from table(f\_test\_array(10));

返回结果为：

id | rq | mc

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

1 | 2022-08-19 12:21:45 | mc1

2 | 2022-08-19 12:21:45 | mc2

3 | 2022-08-19 12:21:45 | mc3

4 | 2022-08-19 12:21:45 | mc4

5 | 2022-08-19 12:21:45 | mc5

6 | 2022-08-19 12:21:45 | mc6

7 | 2022-08-19 12:21:45 | mc7

8 | 2022-08-19 12:21:45 | mc8

9 | 2022-08-19 12:21:45 | mc9

10 | 2022-08-19 12:21:45 | mc10

(10 rows)

## TO\_DSINTERVAL

**功能描述**

TO\_DSINTERVAL 函数将时间转换为DAY TO SECOND INTERVAL数据类型，即处理时间时仅包含日期，小时，分钟和秒的部分。

**语法格式**

TO\_DSINTERVAL ('[+ | -] days hours : minutes : seconds ' [ DEFAULT return\_value ON CONVERSION ERROR ]

**参数说明**

* days：天，整形，范围0到999999999。
* hours：小时，整形，范围0到23。
* minutes：分，整形，范围0到59。
* seconds：秒，整形，范围0到59，Vastbase秒后的小数点支持到6位（Oracle秒后的小数点支持到9位）。
* DEFAULT return\_value ON CONVERSION ERROR：可选参数，该子句允许指定此函数在将参数转换为INTERVAL DAY TO SECOND类型时发生错误时返回的值。

**注意事项**

to\_dsinterval函数将传入字符串转换为INTERVAL数据类型。函数接收两种格式字符串。

* SQL格式，兼容SQL标准（ISO/IEC9075）。
* 不允许输入年和月。
* 语法元素之间允许任意个空格。
* ISO格式，兼容ISO 8601:2004标准。
* 不允许输入年和月。
* 语法元素之间不允许有空格。
* 输入的时间如果出现进位，进位之后的结果超出输入限制是允许的。

**示例**

select to\_dsinterval('1 12 : 34 : 56 ') from dual;

返回结果为：

to\_dsinterval

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

1 day 12:34:56

(1 row)

select to\_yminterval('1-1') from dual;

## TO\_YMINTERVAL

**功能描述**

TO\_YMINTERVAL函数将其参数转换为YEAR TO MONTH INTERVAL数据类型，即处理时间时仅包含年和月的部分。

**语法格式**

TO\_YMINTERVAL ( '{ [+|-] years - months } ' [ DEFAULT return\_value ON CONVERSION ERROR ])

**参数说明**

* years：年，整形，Vastbase取值范围0到178956970（Oracle取值范围0到999999999）。
* months：小时，整形，范围0到11。
* u DEFAULT return\_value ON CONVERSION ERROR：可选参数。该子句允许指定此函数在将参数转换为INTERVAL MONTH TO YEAR类型时发生错误时返回的值。

**注意事项**

to\_yminterval函数将传入字符串转换为INTERVAL数据类型。函数接收两种格式字符串。

* SQL格式，兼容SQL标准（ISO/IEC9075）。
* 仅允许出现年和月。
* 语法元素之间允许任意个空格。
* ISO格式，兼容ISO 8601:2004标准。
* 年、月、日、时、分、秒和小数秒，均允许输入，但除年和月以外的内容会被忽略。
* 秒的小数位支持输入9位，超出后报错（虽然秒会被忽略，但会引发报错）。
* Vastbase年数范围为0到178956970。如果输入月数大于等于12时，会出现向年份进位的情况，进位之后应保证年数范围依然在0到178956970，否则报错。
* 语法元素之间不允许有空格。

**示例**

select to\_yminterval('1-1') from dual;

返回结果为：

to\_yminterval

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

1 year 1 mon

(1 row)

## TO\_SINGLE\_BYTE

**功能描述**

TO\_SINGLE\_BYTE函数可以实现字符全角转半角功能，即将输入的字符或字符串从多字节转换成单字节。

**示例**

全角转半角

select to\_single\_byte('１２３＄％　高ＡＳｔｅｓｔ＠【［');

返回结果如下，则表示支持该函数:

to\_single\_byte

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

123$% 高AStest@【[

(1 row)

## TO\_MULTI\_BYTE

**功能描述**

TO\_MULTI\_BYTE函数可以实现字符半角转全角功能，即将输入的字符或字符串从单字节转换成多字节。

**示例**

1、半角转全角

select to\_multi\_byte('123$% 高AStest@【[');

返回结果如下，则表示支持该函数:

to\_multi\_byte

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

１２３＄％　高ＡＳｔｅｓｔ＠【［

(1 row)

2、全角不变

select to\_multi\_byte('１２３＄％　高ＡＳｔｅｓｔ＠【［');

返回结果如下，则表示支持该函数:

to\_multi\_byte

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

１２３＄％　高ＡＳｔｅｓｔ＠【［

(1 row)

## TO\_TIMESTAMP\_TZ

**功能描述**

TO\_TIMESTAMP\_TZ函数用于将字符类型的输入转化成带时区的timestamptz（存储时间戳的数据类型）。

**语法格式**

TO\_TIMESTAMP\_TZ(char [, fmt [, 'nlsparam' ] ])

**参数说明**

* char：接受CHAR、VARCHAR2、NCHAR和NVARCHAR2数据类型的字符串输入。
* fmt：沿用当前数据库时间日期格式。
* nlsparam：仅需支持'NLS\_DATE\_LANGUAGE=''SIMPLIFIED CHINESE''和"NLS\_DATE\_LANG

UAGE=ENGLISH"（即中文和英文输入）。

**注意事项**

* 暂不支持DEFAULT return\_value ON CONVERSION ERROR可选参数。
* 此函数不会将字符串转换为timestamp with local time zone类型。

**示例**

select to\_timestamp\_tz('2003/12/13 10:13:18', 'YYYY/MM/DD HH:MI:SS');

返回结果为：

to\_timestamp\_tz

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

2003-12-13 10:13:18+08

(1 row)

## TZ\_OFFSET

**功能描述**

TZ\_OFFSET函数用于返回与其输入参数相关的时区偏移量。

**语法格式**

select TZ\_OFFSET ( { 'time\_zone\_name'| '{ + | - } hh : mi'| SESSIONTIMEZONE | DBTIMEZONE } )

**参数说明**

* time\_zone\_name：时区名，如'PRC'。
* {+|-} hh:mi：时区相对UTC的偏移，hh范围：0-15；mi范围：0-59。
* SESSIONTIMEZONE：返回当前会话时区。
* DBTIMEZONE：返回数据库时区。

**注意事项**

该功能仅在数据库兼容模式为Oracle时能够使用（即创建DB时DBCOMPATIBILITY='A'），在其他数据库兼容模式下不能使用该特性。

**示例**

* 查询PRC时区的偏移量

select tz\_offset('PRC');

返回结果为：

tz\_offset

-----------

+08:00

(1 row)

* 查询GMT时区的偏移量

select tz\_offset('GMT');

返回结果为：

tz\_offset

-----------

+00:00

(1 row)

* 查询America/Lima所在时区的时区偏移量

select tz\_offset('America/Lima');

返回结果为：

tz\_offset

-----------

-05:00

(1 row)

## VSIZE

**功能描述**

VSIZE函数用于返回表达式的真实存储长度，以字节表示。

**语法格式**

vsize(expr)

**参数说明**

expr：需要返回存储长度的表达式，如果expr为null，函数返回null。

**示例**

1、创建测试表并插入数据。

create table t\_col(id int primary key ,x varchar(20) );

insert into t\_col values (1,'H');

insert into t\_col values (2,'侯');

insert into t\_col values (3,'H侯');

insert into t\_col values (4,'');

select \* from t\_col;

返回结果为：

id | x

----+-----

1 | H

2 | 侯

3 | H侯

4 |

(4 rows)

2、使用VSIZE函数。

select id, x, length(x), vsize(x) from t\_col;

返回结果如下，则表示支持该函数：

id ｜ x | length | vsize

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

1 ｜ H | 1 | 1

2 ｜ 侯 | 1 | 3

3 ｜ H侯 | 2 | 4

4 ｜ | |

## WM\_CONCAT

**功能描述**

WM\_CONCAT函数可以实现行转列功能，即将查询出的某一列的值使用逗号进行隔开拼接，使之成为一条数据，实现字段合并。

**语法格式**

SELECT column, WM\_CONCAT(column1) FROM table GROUP BY column;

**参数说明**

column1：表中的某一列。

**示例**

1、创建测试表并插入数据。

CREATE TABLE shopping (u\_id int, goods text, num int);

INSERT INTO shopping VALUES (1, 'Apple', 2), (1, 'Melon', 4), (1, 'Orange', 3);

INSERT INTO shopping VALUES (2, 'Pear', 5);

INSERT INTO shopping VALUES (3, 'Grape', 1), (3, 'Banana', 1);

2、使用WM\_CONCAT函数1。

SELECT u\_id, WM\_CONCAT(goods) as goods\_sum

FROM shopping

GROUP BY u\_id;

返回结果如下:

u\_id | goods\_sum

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

1 | Apple,Melon,Orange

3 | Grape,Banana

2 | Pear

(3 rows)

3、使用WM\_CONCAT函数2。

SELECT u\_id, WM\_CONCAT(goods || '(' || num || ')' ) as goods\_sum

FROM shopping

GROUP BY u\_id;

返回结果如下：

u\_id | goods\_sum

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

1 | Apple(2),Melon(4),Orange(3)

3 | Grape(1),Banana(1)

2 | Pear(5)

(3 rows)

## 函数调用时无入参

**功能描述**

支持在存储过程，函数和触发器中调用无传入参数的函数时省略括号。

**语法格式**

调用无传入参数的自定义函数，调用语法如下：

方式一：

select function\_name from dual;

方式二：

select \* from tab where col=function\_name;

**注意事项**

如果查询的表字段出现同名函数的时候，where条件不允许输入函数。例如：

create table t2(id int,fun1 varchar2(20));

select t2 where id=fun1;

返回结果为：

ERROR: column "t2" does not exist

LINE 1: select t2 where id=fun1;

^

CONTEXT: referenced column: t2

**示例**

1、创建测试表并插入数据。

create table t(id int, name varchar2(200));

insert into t values (1,'al');

insert into t values (2,'a2');

2、创建存储存储过程。

create or replace function fun1 return number

as

v\_col varchar(100);

begin

select 1 into v\_col from dual;

return v\_col;

end;

/

3、调用方式一。

select fun1 from dual;

返回结果为：

fun1

------

1

(1 row)

4、调用方式二

select \* from t where id = fun1;

返回结果为：

id | name

----+------

1 | al

(1 row)

5、调用方式三

select fun1,id from t where id=2;

返回结果为：

fun1 | id

------+----

1 | 2

(1 row)

## 函数返回结果集字段带参数列表

**功能描述**

该功能实现了用户在执行函数查询时，返回结果集的列名为函数名（参数），即返回结果的字段名称与查询语句中一致。

**语法格式**

无。

**注意事项**

* 使用该功能需要将参数func\_colname\_with\_args设置为on（该参数默认为off）。
* 该功能在数据库兼容模式为Oracle或者MySQL时能够使用（即创建DB时DBCOMPATIBILITY='A'或者'B'），在其他数据库兼容模式下不能使用该特性。

**示例**

1、创建并切换至兼容模式为A的数据库。

CREATE DATABASE db\_oracle DBCOMPATIBILITY 'A';

\c db\_oracle

2、设置func\_colname\_with\_args参数。

set func\_colname\_with\_args=on;

3、执行函数查询。

select to\_char(123) from dual;

返回结果为：

to\_char(123)

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

123

(1 row)

# 内置包

## DBMS\_ALERT

**功能说明**

DBMS\_ALERT用于生成并传递数据库警报信息，是一种在数据库内部和应用程序间通信的一种方式。

DBMS\_ALERT产生的警报是基于事务的，这意味着如果产生警报的事务未被提交，则等待该警报的session会话将不会收到该警报。一个正在等待警报的session将在数据库中被阻塞。该内置包包含以下函数：

|  |  |
| --- | --- |
| **函数** | **描述** |
| [REGISTER](#ALTERREGISTER) | 注册当前会话接受指定的警报。 |
| [REMOVE](#ALTERREMOVE) | 去除对指定警报的注册。 |
| [REMOVEALL](#ALTERREMOVEALL) | 去除对所有警报的注册。 |
| [SIGNAL](#ALTERSIGNAL) | 为指定警报发出信号。 |
| [WAITANY](#ALTERWAITANY) | 等待任何已注册的警报出现。 |
| [WAITONE](#ALTERWAITONE) | 等待指定的警报出现。 |

**注意事项**

无。

**兼容性**

完全兼容。

**示例**

1、去除alert1警报的注册。

select DBMS\_ALERT.REMOVE('alert1');

2、会话1:接受进程(客户端接收)。

select DBMS\_ALERT.REGISTER('alert1');

3、切换至会话2，发送进程（服务器端发出）。

BEGIN;

select DBMS\_ALERT.SIGNAL('alert1', 'hello,this is sending process!');

commit;

4、切换至会话1:接受进程(客户端接收)。

select DBMS\_ALERT.WAITONE('alert1',100);

返回结果为:

"hello,this is sending process!",0

**REGISTER**

**语法格式**

DBMS\_ALERT.REGISTER (

name IN VARCHAR2);

**参数说明**

name：text类型，指定的警报的名字。

**REMOVE**

**语法格式**

DBMS\_ALERT.REMOVE (

name IN VARCHAR2);

**参数说明**

name：text类型，将要被移除的警报的名字（大小写不敏感）。

**REMOVEALL**

**语法格式**

DBMS\_ALERT.REMOVEALL;

**SIGNAL**

**语法格式**

DBMS\_ALERT.SIGNAL (

name IN VARCHAR2,

message IN VARCHAR2);

**参数说明**

* name：text类型，指定的警报的名字。
* message：text类型，本次警报携带的消息字符串，将被传递给执行wait的会话。

**WAITANY**

**语法格式**

DBMS\_ALERT.WAITANY (

name OUT VARCHAR2,

message OUT VARCHAR2,

status OUT INTEGER,

timeout IN NUMBER DEFAULT MAXWAIT);

**参数说明**

* name：text类型，等待的警报的名字。
* message：text类型，收到的警报携带的消息字符串。如果收到的警报在WAITANY之前被SIGNAL产生了多次，那么收到的message消息字符串是最近的一次SIGNAL产生的，之前的SIGNAL产生的message消息将被忽略。。
* status：integer类型，0表示由收到警报返回，1表示由于超时返回。
* timeout：警报等待的最长时间。缺省默认值为86400000（1000天）。

**WAITONE**

**语法格式**

DBMS\_ALERT.WAITONE (

name IN VARCHAR2,

message OUT VARCHAR2,

status OUT INTEGER,

timeout IN NUMBER DEFAULT MAXWAIT);

**参数说明**

* name：text类型，等待的指定的警报的名字。
* message：text类型，收到的警报携带的消息字符串。如果收到的警报在WAITANY之前被SIGNAL产生了多次，那么收到的message消息字符串是最近的一次SIGNAL产生的，之前的SIGNAL产生的message消息将被忽略。。
* status：integer类型，，0表示由收到警报返回，1表示由于超时返回。
* timeout：警报等待的最长时间。缺省默认值为86400000（1000天）。

## DBMS\_APPLICATION\_INFO

**功能描述**

DBMS\_APPLICATION\_INFO通过视图v$SESSION来跟踪脚本运行情况的能力，该内置包允许应用程序在v$session上设置如下三个字段的值：

* client\_info：客户端信息。
* Module：主程序名称，如包的名称。
* Action：程序包中的过程名。

以上三个字段的内容可以在视图V$SESSION中查看。该功能主要用于对各模块的性能跟踪和调试，系统管理员可按模块跟踪性能，还可以使用此信息来跟踪模块的资源使用情况。该内置包包含以下函数：

|  |  |
| --- | --- |
| **函数** | **描述** |
| READ\_CLIENT\_INFO | 读取当前会话通过SET\_CLIENT\_INFO过程的最后一个信息值，即最后一次存储的客户端信息 |
| READ\_MODULE | 读取当前会话的模块和操作字段信息。 |
| SET\_ACTION | 用于设置当前会话下的当前模块下的操作名称。 |
| SET\_CLIENT\_INFO | 用于设置当前会话有关客户端应用程序的其他信息。 |
| SET\_MODUL | 用于设置当前模块的名称。 |

​**注意事项**

该功能仅在数据库兼容模式为Oracle时能够使用（即创建DB时DBCOMPATIBILITY='A'），在其他数据库兼容模式下不能使用该特性。

**兼容性**

Vastbase G100 V2.2版本与Oracle 11g R2版本相比，不支持的函数有：SET\_SESSION\_LONGOPS。

**示例**

1、创建并切换至兼容模式为Oracle的数据库db\_oracle。

CREATE DATABASE db\_oracle dbcompatibility='A';

\c db\_oracle

2、创建函数。

CREATE OR REPLACE FUNCTION test\_dbms\_application\_info RETURNS VOID AS

$$

DECLARE

result1 text:='test';

v\_module text;

v\_action text;

BEGIN

dbms\_application\_info.set\_client\_info('client A');

select \* into result1 from dbms\_application\_info.read\_client\_info();

dbms\_application\_info.set\_module('module A','action A');

dbms\_application\_info.set\_action('action B');

select \* into v\_module,v\_action from dbms\_application\_info.read\_module();

raise notice 'result\_read\_client\_info is %',result1;

raise notice 'result\_read\_module is % and %',v\_module,v\_action;

END;

$$

LANGUAGE plpgsql;

3、调用函数。

select test\_dbms\_application\_info();

返回结果为：

NOTICE: result\_read\_client\_info is client A

CONTEXT: referenced column: test\_dbms\_application\_info

NOTICE: result\_read\_module is module A and action B

CONTEXT: referenced column: test\_dbms\_application\_info

test\_dbms\_application\_info

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

(1 row)

## DBMS\_ASSERT

**功能描述**

该内置包用于验证输入值的属性。该内置包包含以下函数：

|  |  |
| --- | --- |
| **函数** | **描述** |
| [ENQUOTE\_LITERAL](#ENQUOTE_LITERAL) | 输入字符串满足下述要求时，在字符串首位两端补充单引号：  a. 必须保证除首尾两端的单引号之外，其余单引号必须紧邻成对出现，且成对紧邻出现时不能出现在字符串开头。  b. 若原始字符串两端已有单引号，则本接口不再额外在字符串两端补充单引号。 |
| [SCHEMA\_NAME](#SCHEMA_NAME) | 判断输入字符串是否为有效的模式名，若有效则返回该模式名。 |
| [SQL\_OBJECT\_NAME](#SQL_OBJECT_NAME) | 判断输入字符串是否为有效的SQL对象名，若有效则返回该对象名。  输入的对象名可以是如下情况：  a. 若输入的对象名是同义词名，则要求其基本对象必须存在。  b. 该函数可识别的 SQL对象包括表，视图，同义词，索引，序列，包，触发器，函数，存储过程。  c. 输入内容可带从属关系，支持的从属关系输入方式包括：  模式名.表名  模式名.视图名  模式名.索引名  模式名.序列名  模式名.包名  模式名.包名.函数名（SQL语句暂不支持)  模式名.包名.存储过程名（SQL语句暂不支持)  模式名.包名.变量名( SQL语句暂不支持)  模式名.函数名  模式名.存储过程名  模式名.同义词  模式名.表名.字段名  模式名.视图名.字段名  表名.字段名  视图名.字段名 |

​**注意事项**

无。

**兼容性**

Vastbase G100 V2.2版本与Oracle 11g R2版本相比，不支持的函数有：ENQUITE\_NAME、NOOP、QUALIFIED、SIMPLE\_SQL\_NAME。

**ENQUOTE\_LITERAL函数**

**语法格式**

DBMS\_ASSERT.ENQUOTE\_LITERAL (

str VARCHAR2);

**参数说明**

str：要引用的字符串。

**示例**

select DBMS\_ASSERT.ENQUOTE\_LITERAL('abcedf') from dual;

enquote\_literal

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

'abcedf'

(1 row)

**SCHEMA\_NAME函数**

**语法格式**

DBMS\_ASSERT.SCHEMA\_NAME (

str VARCHAR2 CHARACTER SET ANY\_CS)；

**参数说明**

str：有效的模式名称。

**示例**

create schema testname;

select DBMS\_ASSERT.SCHEMA\_NAME('testname');

schema\_name

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

testname

(1 row)

**SQL\_OBJECT\_NAME函数**

**语法格式**

DBMS\_ASSERT.SQL\_OBJECT\_NAME (

str VARCHAR2 CHARACTER SET ANY\_CS);

**参数说明**

str：有效的模式名称。

**示例**

1、创建函数test\_f。

create function test\_f(int) returns int as $$

begin

raise notice 'test\_f%',$1;

return $1;

end $$ language plpgsql;

2、调用SQL\_OBJECT\_NAME函数。

select DBMS\_ASSERT.SQL\_OBJECT\_NAME('test\_f');

返回结果为：

sql\_object\_name

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

test\_f

(1 row)

## DBMS\_CRYPTO

**功能描述**

DBMS\_CRYPTO是一个PL/SQL包，提供用于加密和解密存储的数据的接口，可以对常用的数据类型进行加密和解密，比如raw和大对象clob，blob类型。

* raw类型：存储二进制数据或位串。
* blob类型：存储二进制大对象数据（如数码照片）。
* clob类型：存储大对象单字节的字符数据（如大型文本）。
* 该包包含以下函数：ENCRYPT、DECRYPT、HASH、MAC、RANDOMBYTES、RANDOMINTEGER、RANDOMNUMBER。

该内置包包含以下函数，其中ENCRYPT、DECRYPT函数均存在两个，下面描述时记作ENCRYPT函数1、ENCRYPT函数2、DECRYPT函数1、DECRYPT函数2。

|  |  |
| --- | --- |
| **函数** | **描述** |
| [ENCRYPT函数1](#ENCRYPT1) | 使用用户指定的加密套件来对raw类型数据进行加密，返回raw类型数据。 |
| [ENCRYPT函数2](#ENCRYPT2) | 使用用户指定的加密套件来对blob、clob类型数据进行加密，返回blob类型数据。 |
| [DECRYPT函数1](#DECRYPT1) | 使用用户指定的加密套件来对raw类型数据进行解密。 |
| [DECRYPT函数2](#DECRYPT2) | 使用用户指定的加密套件来对blob类型数据进行解密。 |
| [HASH函数](#HASH) | 使用用户指定的hash算法来返回一个raw类型的哈希值，使用哈希值来验证数据是否已更改。算法标识符是hash\_md5，整数码为2。 |
| [MAC函数](#MAC) | 使用用户指定的MAC算法来返回一个bytea类型数据的MAC哈希值。算法标识符是hmac\_md5，整数码为1。 |
| [RANDOM\_BYTES函数](#RANDOMBYTES) | 生成指定长度（范围为[1, 2000]）的随机字节raw值。 |
| [RANDOM\_INTEGER函数](#RANDOMINTEGER) | 用于返回integer数据类型的完整范围内的随机整数。 |
| [RANDOM\_NUMBER函数](#RANDOMNUMBER) | 用于返回number数据类型范围为[0, 2^128 - 1]内的随机数据。 |

**注意事项**

* 该功能仅在数据库兼容模式为Oracle时能够使用（即创建DB时DBCOMPATIBILITY='A'），在其他数据库兼容模式下不能使用该特性。
* 使用DBMS\_CRYPTO内置包前，需要先加载pgcrypto插件。执行如下命令加载插件：

CREATE EXTENSION pgcrypto;

* ENCRYPT函数和DECRYPT函数支持的加密套件组合码如下表：

| **加密套件标识符** | **整数码** |
| --- | --- |
| encrypt\_des | 1 |
| encrypt\_3des | 3 |
| chain\_ecb | 768 |
| chain\_cbc | 256 |
| pad\_pkcs | 4096 |
| pad\_none | 8192 |
| des\_cbc\_pkcs | 4353 |
| des3\_cbc\_pkcs | 4355 |
| hash\_md5 | 2 |
| hmac\_md5 | 1 |

* 使用加密函数时不能使用标识符名称，必须使用对应的整数码。

例如:

select dbms\_crypto.decrypt('57E647BDA301C9FA'::rawdbms\_crypto.des\_cbc\_pkcs, '123456'::raw);

其中dbms\_crypto.des\_cbc\_pkcs名称需要换成4353

select dbms\_crypto.decrypt('57E647BDA301C9FA'::raw, 4353, '123456'::raw);

* 对于使用两种或以上加密套件标识符的情况，只需要将其整数码数值相加即可。

**兼容性**

Vastbase G100 V2.2版本与Oracle 11g R2版本相比，不支持的函数有：SING、VERIFY、PKDECRYPT、PKENCRYPT。

**ENCRYPT函数1**

**语法格式**

DBMS\_CRYPTO.ENCRYPT(

src IN RAW,

typ IN PLS\_INTEGER,

key IN RAW,

iv IN RAW DEFAULT NULL)

RETURN RAW;

**参数说明**

* src：要破解的RAW数据。
* typ：加密类型。
* key：解密使用的key。
* iv：块密码初始化向量，缺省NULL。

**示例**

* 直接调用。

select dbms\_crypto.encrypt('ABCABC'::raw, 4353, '123456'::raw);

encrypt

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

57E647BDA301C9FA

(1 row)

select dbms\_crypto.decrypt('57E647BDA301C9FA'::raw, 4353,'123456'::raw);

decrypt

---------

ABCABC

(1 row)

* 加密表数据。

1、创建测试表并插入数据。

create table t\_select\_raw(id int, col varchar2(20));

insert into t\_select\_raw values(1,'测试');

insert into t\_select\_raw values(2,'中文');

create table t\_dml\_raw(id int, col raw);

insert into t\_dml\_raw values(1,'1234AABBCCDDEEFF');

2、查询结果加密。

select dbms\_crypto.encrypt(hextoraw(encode(col::bytea, 'hex')), 4353, '123456'::raw) from t\_select\_raw;

encrypt

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

6C5B52D18B146E13

B01C554C5102A8FE

(2 rows)

select dbms\_crypto.encrypt(hextoraw(encode(col::bytea, 'hex')), 4353, hextoraw(encode(col::bytea, 'hex'))) from t\_select\_raw;

encrypt

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

9A5296006CB243A3

3D172C9ECA9B4A43

(2 rows)

select dbms\_crypto.encrypt(hextoraw(encode(col::bytea, 'hex')), 4353, hextoraw(encode(col::bytea, 'hex')), hextoraw(encode(col::bytea, 'hex'))) from t\_select\_raw;

encrypt

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

58AEFBB87999E13C

CE070EDF29322C9F

(2 rows)

3、查询raw数据。

select dbms\_crypto.encrypt(col, 4353, col,col) from t\_dml\_raw;

encrypt

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

8475EECF02BEE7E870B363D667785B6E

(1 row)

4、插入数据加密。

insert into t\_dml\_raw values(2, dbms\_crypto.encrypt(hextoraw(encode('中文'::bytea, 'hex')), 4353, '1'::raw));

insert into t\_dml\_raw values(2, dbms\_crypto.encrypt(hextoraw(encode('abc'::bytea, 'hex')), 4353, hextoraw(encode('2'::bytea, 'hex')), hextoraw(encode('2'::bytea, 'hex'))));

update t\_dml\_raw set col = dbms\_crypto.encrypt(hextoraw(encode('测试'::bytea, 'hex')), 4353, '1'::raw) where id = 1;

select \* from t\_dml\_raw;

5、显示加密后的结果。

select \* from t\_dml\_raw;

id | col

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

2 | 634FE089F003C8AA

2 | 10759AD6661DD7BC

1 | 350968ECE5EBF882

(3 rows)

**ENCRYPT函数2**

**语法格式**

DBMS\_CRYPTO.ENCRYPT(

dst IN OUT NOCOPY BLOB,

src IN BLOB,

typ IN PLS\_INTEGER,

key IN RAW,

iv IN RAW DEFAULT NULL);

DBMS\_CRYPTO.ENCRYPT(

dst IN OUT NOCOPY BLOB,

src IN CLOB CHARACTER SET ANY\_CS,

typ IN PLS\_INTEGER,

key IN RAW,

iv IN RAW DEFAULT NULL);

**参数说明**

* dst：解密数据的输出位置。
* src：要破解的LOB数据。
* typ：加密类型。
* key：解密使用的key。
* iv：块密码初始化向量，缺省NULL。

**示例**

* 对blob类型数据进行加密-直接调用。

select dbms\_crypto.encrypt('abc'::blob,'ABCABC'::blob,4353, '123456'::raw);

encrypt

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

57E647BDA301C9FA

(1 row)

* 对clob类型数据进行加密-直接调用。

select dbms\_crypto.encrypt('abc'::blob,'ABCABC'::clob, 4353, '123456'::raw);

encrypt

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

D552672EE9151C95

(1 row)

**DECRYPT函数1**

**语法格式**

DBMS\_CRYPTO.DECRYPT(

src IN RAW,

typ IN PLS\_INTEGER,

key IN RAW,

iv IN RAW DEFAULT NULL)

RETURN RAW;

**参数说明**

* src：要破解的RAW数据。
* typ：加密类型。
* key：解密使用的key。
* iv：块密码初始化向量，缺省NULL。

**示例**

1、对raw类型数据进行加密,使用相同的方式进行解密-直接调用(获取加密结果)。

select dbms\_crypto.encrypt('ABCABC'::raw, 4353, '123456'::raw);

返回结果为：

encrypt

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

57E647BDA301C9FA

(1 row)

2、将加密结果使用相同的方式进行解密(此处需要使用上一步加密结果数据来替换命令中的加密结果)。

select dbms\_crypto.decrypt('加密后的结果'::raw, 4353, '123456'::raw);

decrypt

---------

ABCABC

(1 row)

**DECRYPT函数2**

**语法格式**

DBMS\_CRYPTO.DECRYPT(

dst IN OUT NOCOPY BLOB,

src IN BLOB,

typ IN PLS\_INTEGER,

key IN RAW,

iv IN RAW DEFAULT NULL);

DBMS\_CRYPT.DECRYPT(

dst IN OUT NOCOPY CLOB CHARACTER SET ANY\_CS,

src IN BLOB,

typ IN PLS\_INTEGER,

key IN RAW,

iv IN RAW DEFAULT NULL);

**参数说明**

* dst：解密数据的输出位置。
* src：要破解的LOB数据。
* typ：加密类型。
* key：解密使用的key。
* iv：块密码初始化向量，缺省NULL。

**示例**

1、对blob类型数据进行解密-直接调用。

select dbms\_crypto.decrypt('abc'::blob,'D552672EE9151C95'::blob, 8449, '123456'::raw, 'ABC'::raw);

decrypt

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

414243414243

(1 row)

2、对blob类型数据进行加密,使用相同的方式进行解密-直接调用(获取加密结果)。

select dbms\_crypto.encrypt('abc'::blob,'ABCABC'::blob,4353, '123456'::raw);

encrypt

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

57E647BDA301C9FA

(1 row)

3、将加密结果使用相同的方式进行解密(此处需要使用上一步加密结果数据来替换命令中的加密结果)。

select dbms\_crypto.decrypt('abc'::blob,'加密后的结果'::blob, 4353, '123456'::raw);

decrypt

---------

ABCABC

(1 row)

**HASH函数**

**语法格式**

DBMS\_CRYPTO.Hash (

src IN RAW,

typ IN PLS\_INTEGER)

RETURN RAW;

DBMS\_CRYPTO.Hash (

src IN BLOB,

typ IN PLS\_INTEGER)

RETURN RAW;

DBMS\_CRYPTO.Hash (

src IN CLOB CHARACTER SET ANY\_CS,

typ IN PLS\_INTEGER)

RETURN RAW;

**参数说明**

* src：要计算的原始数据。
* typ：哈希算法类型。

**示例**

select dbms\_crypto.hash('57E647BDA301C9FA'::blob, 2);

hash

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

0625EE8F90EE27BEA4AD2EC9017E5A2E

(1 row)

select dbms\_crypto.hash('中文测试'::clob, 2);

hash

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

089B4943EA034ACFA445D050C7913E55

(1 row)

**MAC函数**

**语法格式**

DBMS\_CRYPTO.MAC (

src IN RAW,

typ IN PLS\_INTEGER,

key IN RAW)

RETURN RAW;

DBMS\_CRYPTO.MAC (

src IN BLOB,

typ IN PLS\_INTEGER

key IN RAW)

RETURN RAW;

DBMS\_CRYPTO.MAC (

src IN CLOB CHARACTER SET ANY\_CS,

typ IN PLS\_INTEGER

key IN RAW)

RETURN RAW;

**参数说明**

* src：要计算的原始数据。
* typ：MAC算法类型。
* MAC：MAC算法的key值。

**示例**

select dbms\_crypto.mac('中文测试'::clob, 1, '123456ABC'::raw);

mac

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

2CDC70BD0B266731AEE027C185E77D68

(1 row)

**RANDOMBYTES函数**

**语法格式**

DBMS\_CRYPTO.RANDOMBYTES (

number\_bytes IN POSITIVE)

RETURN RAW;

**参数说明**

number\_bytes：要生成的随机字节数。

**示例**

select dbms\_crypto.randombytes(10) from dual;

randombytes

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

073B14CADD7E284F2417

(1 row)

**RANDOMINTEGER函数**

**语法格式**

DBMS\_CRYPTO.RANDOMINTEGER

RETURN BINARY\_INTEGER;

**示例**

select dbms\_crypto.randominteger() from dual;

randominteger

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

-909240370

(1 row)

**RANDOMNUMBER函数**

**语法格式**

DBMS\_CRYPTO.RANDOMNUMBER

RETURN NUMBER;

**示例**

select dbms\_crypto.randomnumber() from dual;

randomnumber

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

11351894157074763785441053451

(1 row)

DBMS\_DEBUG

**功能描述**

DBMS\_DEBUG内置包用于实现服务器端调试器，提供了一种调试服务器端PL/SQL程序单元的方法，调试时分为目标会话（the target session）和调试会话（the debug session），通过两个会话配合完成PL/SQL调试工作。

目标会话用来以调试模式运行代码，调试会话用来监督目标会话。在调试会话中通过使用初始化生成的会话id，结合执行响应的调试api接口与目标会话进行通信对PL/SQL程序单元进行调试。该内置包包含以下函数:

|  |  |
| --- | --- |
| **函数** | **描述** |
| [INITIALIZE](#INITIALIZE) | 初始化目标会话以进行调试。 |
| [DEBUG\_ON](#DEBUG_ON) | 使所有PL/SQL都在调试模式下运行。 |
| [DEBUG\_OFF](#DEBUG_OFF) | 通知目标会话不应再进行调试。 |
| [EXECUTE](#DEBUGEXECUTE) | 在目标会话中执行一段SQL或PL/SQL。 |

**注意事项**

* 只有在INITIALIZE操作之后才能进行DEBUG\_ON操作，然后才能进行PL/SQL操作。
* 在目标会话中如果有DBMS\_DEBUG.DEBUG\_ON已生效，再次执行时会提示已开启调试模式。
* 在目标会话中如果有DBMS\_DEBUG.DEBUG\_OFF已生效，不会进入阻塞状态。

**兼容性**

Vastbase G100 V2.2版本与Oracle 11g R2版本相比，支持的函数有：EXECUTE、INITIALIZE、DEBUG\_ON、DEBUG\_OFF,其他函数暂不支持。

**示例**

1、创建插件。

create extension pldbgapi;

2、调用debug\_on函数。

exec dbms\_debug.debug\_on();

返回结果如下：

NOTICE: Debugger is started successfully, you are SERVER now.

CONTEXT: SQL statement "CALL dbms\_debug.debug\_on\_inner($1,$2)"

PL/pgSQL function dbms\_debug.debug\_on(boolean,boolean) line 3 at PERFORM

debug\_on

----------

(1 row)

fig: **说明：**

该函数没有返回值，执行成功时会有Debugger is started successfully, you are Server now.的提示，没有procedure successfully completed的提示，Oracle的procedure successfully completed为执行过程执行成功时的的统一输出结果，这里与Oracle有差异。

3、调用execute函数。

select dbms\_debug.execute('select \* from pg\_proc',-1,-1,'sds','sdsad');

返回结果为：

WARNING: bind\_results Only 0 or 1 is supported

CONTEXT: referenced column: execute\_inner

SQL statement "select dbms\_debug.execute\_inner($1, $2, $3, $4, $5)"

PL/pgSQL function dbms\_debug.execute(varchar,integer,integer,dbms\_debug\_vc2coll,varchar) line 5 at SQL statement

referenced column: execute

execute

---------

t

(1 row)

4、调用dbms\_debug.debug\_off函数。

exec dbms\_debug.debug\_off();

**INITIALIZE函数**

**语法格式**

DBMS DEBUG.INITIALIZE (

debug\_session\_id IN VARCHAR2 := NULL,

diagnostics IN BINARY\_INTEGER := 0)

RETURN VARCHAR2;

**参数说明**

* debug\_session\_id：会话ID的名称，如果为 NULL，则生成一个唯一的ID。
* diagnostics：指示是否将诊断信息输出，其中0为默认值，表示不诊断输出：1 为打印诊断信息。

**DEBUG\_ON函数**

**语法格式**

DBMS DEBUG.DEBUG\_ON (

no\_client\_side\_plsql\_engine BOOLEAN := TRUE.

immediate BOOLEAN := FALSE);

**参数说明**

* no\_client\_side\_plsql\_engine：默认值为true，除非从客户端PL/SQL引擎进行调试会话，否则应将其保留为默认值，
* immediate：如果值为TRUE，则解释器将立即将其自身切换到调试模式，而不是在调用期间以常规模式继续运行。

**DEBUG\_OFF函数**

**语法格式**

DBMS DEBUG.DEBUG\_OFF;

**EXECUTE函数**

**语法格式**

DBMS DEBUG.EXECUTE (

what IN VARCHAR2,

frame# IN BINARY\_INTEGER,

bind\_results IN BINARY\_INTEGER,

results IN OUT NOCOPY dbms\_debug\_vc2coll,

errm IN OUT NOCOPY VARCHAR2);

**参数说明**

* what：要执行的sql或pl/sgl 源。
* frame#：执行代码的上下文。目前仅支持-1(全局上下文)。
* bind\_results：源是否要绑定到results以便从目标会话返回值：0=否;1=是。
* results：放置结果的集合(如果bind results不为0)，dbms\_debug\_vc2coll为集合类型，也写为sys.dbms\_debug\_vc2coll。
* errm：如果发生错误，显示错误信息，否则为null。

## DBMS\_JOB

**功能描述**

DBMS\_JOB包用于安排和管理作业队列，通过使用作业，可以使数据库定期执行特定的任务。该内置包中包含以下函数：

|  |  |
| --- | --- |
| **函数** | **描述** |
| [SUBMIT](#SUBMIT) | 打开一个cursor，并返回该cursor的id。 |
| [RUN](#RUN) | 运行一次指定的任务。 |
| [REMOVE](#REMOVE) | 移除一个定时任务。 |
| [BROKEN](#BROKEN) | 停止或启动一个定时任务。 |

**注意事项**

无。

**兼容性**

Vastbase G100 V2.2版本与Oracle 11g R2版本相比，不支持的函数有：CHANGE、INSTANCE、INTERVAL、NEXT\_DATE、USER\_EXPORT、WHAT。

**示例**

1、设置 enable\_prevent\_job\_task\_startup参数为off（即job线程不能启动）。

ALTER SYSTEM SET enable\_prevent\_job\_task\_startup TO off;

2、创建测试表并插入数据。

drop table if exists t\_sub;

create table t\_sub(insert\_date timestamp);

3、创建插入数据的定时任务。

select dbms\_job.submit(1,'insert into t\_sub values(sysdate);',sysdate,'''1min''::interval');

select \* from t\_sub;

返回结果为：

insert\_date

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

2022-07-06 09:14:39

(1 row)

4、等待1分钟之后，查询t\_sub。

select \* from t\_sub;

返回结果如下，则表示定时任务创建成功：

insert\_date

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

2022-07-06 09:14:39

2022-07-06 09:15:39

(2 rows)

5、运行正在运行的任务。

select dbms\_job.run(1);

6、移除正在运行的任务：

select dbms\_job.remove(1);

**SUBMIT函数**

**语法格式**

DBMS JOB.SUBMIT (

job OUT BINARY\_INTEGER.

what IN VARCHAR2,

next\_date IN DATE DEFAULT sysdate

interval IN VARCHAR2 DEFAULT 'null'

no\_parse IN BOOLEAN DEFAULT FALSE,

instance IN BINARY INTEGER DEFAULT any\_instance,

force IN BOOLEAN DEFAULT FALSE);

**参数说明**

* job：输出参数，输出正在运行的作业编号。
* what：指定需要运行的存储过程。
* next\_date：下一次job 运行的时间，默认为sysdate。
* interval：时间间隔，默认为 null。
* no\_parse：标记参数。设置为 true 表示数据库会在首次运行该作业时解析与该作业相关联的过程，默认为 false 。
* instance：提交作业后，指定可以运行该作业的实例(支持该参数传入，但不校验并忽略该参数)。
* force：该参数设置为true 时表示任何正整数都可被接受为作业实例，默认为false则指定实例必须是运行着的(支持该参数传入，但不校验并忽略该参数)。

**RUN函数**

**语法格式**

DBMS JOB.RUN (

job IN BINARY\_INTEGER.

force IN BOOLEAN DEFAULT FALSE);

**参数说明**

* job：需要运行的作业编号。
* force：该参数设置为true 时表示任何实例环境与在进程中与运行的作业无关，如果设置为false，则只能在指定实例中运行作业(支持该参数传入，但不校验并忽略该参数)。

**REMOVE函数**

**语法格式**

DBMS JOB.RUN (

job IN BINARY\_INTEGER;

**参数说明**

job：需要删除的作业编号。

**BROKEN函数**

**语法格式**

DBMS JOB.RUN (

job IN BINARY\_INTEGER,

Broken IN BOOLEAN,

next\_date IN DATE DEFAULT sysdate);

**参数说明**

* job：作业编号。
* broken：boolean类型，设置为true表示停止或者禁用作业任务，false则表示不中断（broken）作业任务。
* next\_date：任务作业运行的下一个日期。

## DBMS\_LOB

**功能描述**

DBMS\_LOB用于提供一组对blob（二进制数据类型）、clob（字符数据类型）和bfile数据类型（存储数据库外的大型对象，指向操作系统物理文件）操作的函数或过程。该内置包支持以下函数：

|  |  |
| --- | --- |
| **函数** | **描述** |
| [GETLENGTH](#LOBGETLENGTH) | 获取blob、clob和bfile数据类型的长度。 |
| [FILEOPEN](#LOBFILEOPEN) | 打开一个bfile文件。 |
| [READ](#LOBREAD) | 从一个bfile文件中读取指定长度的内容。 |
| [FILECLOSE](#LOBFILECLOSE) | 关闭一个bfile文件。 |
| OPEN | 根据传入的参数打开对应的LOB对象。 |
| COPY | 根据传入的参数从指定位置开始将源LOB复制到目标LOB。 |
| WRITEAPPEND | 将指定数据添加到一个大对象的末尾。 |
| CREATETEMPORARY | 用来在临时表空间中创建临时BLOB或CLOB。 |
| FREETEMPORARY | 释放临时表空间中的临时BLOB或CLOB。 |
| INSTR | 该函数用来返回LOB中指定模式第n次出现的匹配位置，从指定的偏移量开始。 |

**注意事项**

该功能仅在数据库兼容模式为Oracle时能够使用（即创建DB时DBCOMPATIBILITY='A'），在其他数据库兼容模式下不能使用该特性。

**示例**

1、操作系统环境下创建文件，并输入数据。

vi /home/vastbase/bfile.data

cat /home/vastbase/bfile.data

文件内容为：

张三

李四

2、在数据库中执行如下命令创建目录和表。

drop directory d\_bfile;

drop table t\_bfile;

create directory d\_bfile as '/home/vastbase';

create table testbfile1(id number,bfile\_name bfile);

3、调用bfilename函数构造bfile数据并插入到表中。

insert into testbfile1 values(1,bfilename('d\_bfile', 'bfile.data'));

set serveroutput on;

创建函数：

create or replace function f\_read\_bfile() returns void as $$

declare

buff raw;

amount int := 0 ;

offset int :=0;

lob\_loc bfile;

filesize int;

begin

select bfile\_name into lob\_loc from testbfile1 where id=1;

--打开BFILE文件

dbms\_lob.fileopen(lob\_loc,0);

--获取文件大小

filesize := dbms\_lob.getlength(lob\_loc);

raise notice 'amount:%',amount;

--读取文件全部内容

amount = filesize;

dbms\_lob.read(lob\_loc,amount,0,buff);

dbms\_output.put\_line('file data:');

dbms\_output.put\_line(utl\_raw.cast\_to\_varchar2(buff));

--关闭bfile文件

dbms\_lob.fileclose(lob\_loc);

return;

end;

$$ language plpgsql;

4、调用函数。

select f\_read\_bfile();

返回结果为：

file data:

张三

李四

f\_read\_bfile

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

(1 row)

**GETLENGTH**

**语法格式**

BLOB对象类型：由于存储的内容即为二进制数据，直接返回二进制内容的字节数即可，函数原型为：

DBMS\_LOB.GETLENGTH (

lob\_loc IN BLOB)

RETURN INTEGER;

CLOB对象类型：该对象存储的数据格式为字符数据，返回值为字符数。函数原型为：

DBMS\_LOB.GETLENGTH (

lob\_loc IN CLOB)

RETURN INTEGER;

BFILE对象类型：函数返回bfile数据指向的操作系统物理文件的长度，以字节为单位，bfile文件的长度包含了文件结尾符（eof）。函数原型为：

DBMS\_LOB.GETLENGTH (

file\_loc IN BFILE)

RETURN INT8;

**参数说明**

file\_loc：将要返回其长度的文件位置。

**示例**

**示例1：**

1、创建测试表并插入数据。

create table testclob (data clob);

insert into testclob values('中文测试');

2、查询数据长度。

select dbms\_lob. getlength (data) from testclob;

返回结果为：

getlength

-----------

4

(1 row)

**示例2：**

1、创建测试表并插入数据。

create table testblob (data blob);

insert into testblob values(utl\_raw.cast\_to\_raw('中文测试'));

2、查询数据长度。

select dbms\_lob. getlength (data) from testblob;

返回结果为：

getlength

-----------

12

(1 row)

**FILEOPEN**

**语法格式**

DBMS\_LOB.FILEOPEN (

file\_loc IN OUT BFILE,

open\_mode IN BINARY\_INTEGER );

**参数说明**

* file\_loc：要打开的文件定位。
* open\_mode：文件访问模式。

**READ**

**语法格式**

DBMS\_LOB.READ (

file\_loc IN BFILE,

amount IN OUT INTEGER,

offset IN INTEGER,

buffer OUT RAW);

**参数说明**

* file\_loc：lob要检查的文件。
* amount：读取文件的长度。为了防止打开的文件过大，单次读取的内容超过机器缓冲区导致系统崩溃，规定单次读取的字节数不能大于32767字节，即amount为大于0小于等于32767的正整数。
* offset：用于多次读取文件时设置读取的起始位置，如果offset参数指向bfile文件结尾或超出文件结尾，则将amount参数设置为0，并抛出异常。
* buffer：读取操作的输出缓冲区。

**FILECLOSE**

**语法格式**

DBMS\_LOB.FILECLOSE (

file\_loc IN OUT BFILE);

**参数说明**

file\_loc：要关闭的文件。

**OPEN存储过程**

**语法格式**

DBMS\_LOB.OPEN (

lob\_loc IN OUT NOCOPY BLOB,

open\_mode IN BINARY\_INTEGER);

DBMS\_LOB.OPEN (

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

open\_mode IN BINARY\_INTEGER);

DBMS\_LOB.OPEN (

file\_loc IN OUT NOCOPY BFILE,

open\_mode IN BINARY\_INTEGER := file\_readonly);

**参数说明**

* lob\_loc：LOB类型的对象名。
* open\_mode：打开模式。对于BLOB，CLOB对象，可以选择LOB\_READONLY或者LOB\_READWRITE。对于BFILE类型，只能是FILE\_READONLY。（LOB\_READONLY：只读。LOB\_READWRITE：可读写。FILE\_READONLY：只读。）

**示例**

1、创建含blob、clob类型字段的表。

create table testtable(id int,c\_lob clob,b\_lob blob);

insert into testtable values(1,'clob测试',utl\_raw.cast\_to\_raw('blob'));

2、调用open存储过程。

set serveroutput on;

declare

v\_clob clob;

v\_clob2 clob;

len int;

begin

select c\_lob into v\_clob from testtable where id=1;

dbms\_lob.open(v\_clob,dbms\_lob.lob\_readwrite);

dbms\_output.put\_line('clob1:'||v\_clob);

dbms\_lob.close(v\_clob);

end;

/

返回结果为：

clob1:clob测试

ANONYMOUS BLOCK EXECUTE

**COPY存储过程**

**语法格式**

DBMS\_LOB.COPY (

dest\_lob IN OUT NOCOPY BLOB,

src\_lob IN BLOB,

amount IN INTEGER,

dest\_offset IN INTEGER := 1,

src\_offset IN INTEGER := 1);

DBMS\_LOB.COPY (

dest\_lob IN OUT NOCOPY CLOB CHARACTER SET ANY\_CS,

src\_lob IN CLOB CHARACTER SET dest\_lob%CHARSET,

amount IN INTEGER,

dest\_offset IN INTEGER := 1,

src\_offset IN INTEGER := 1);

**参数说明**

* dest\_lob：目标LOB对象。
* src\_lob：源LOB对象。
* amount：复制的长度。
* dest\_offset：目标LOB中的偏移量。
* src\_offset：源LOB中的偏移量。

**示例**

存储过程调用copy函数，输入参数中的dest\_lob为clob类型。

1、创建含blob类型字段的表并插入数据。

drop table if exists testtable\_01;

create table testtable\_01(id int,c\_lob clob);

insert into testtable\_01 values(1,'blobtest1测试测试');

insert into testtable\_01 values(2,'blobtest2测试测试');

2、调用COPY函数。

set serveroutput on;

DECLARE

vdest\_lob CLOB;

vsrc\_lob CLOB;

amount NUMBER;

dest\_offset NUMBER;

src\_offset NUMBER;

BEGIN

SELECT c\_lob INTO vdest\_lob FROM testtable\_01 WHERE id = 1 ;

SELECT c\_lob INTO vsrc\_lob FROM testtable\_01 WHERE id = 2 ;

amount := DBMS\_LOB.GETLENGTH(vsrc\_lob);

dest\_offset := DBMS\_LOB.GETLENGTH(vdest\_lob) + 1;

src\_offset := 1;

DBMS\_LOB.COPY(vdest\_lob, vsrc\_lob, amount, dest\_offset, src\_offset);

DBMS\_OUTPUT.PUT\_LINE('拷贝结果为: ' || vdest\_lob);

END;

/

返回结果为：

拷贝结果为: blobtest1测试测试blobtest2测试测试

ANONYMOUS BLOCK EXECUTE

**WRITEAPPEND存储过程**

**语法格式**

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);

**参数说明**

* lob\_loc：LOB类型的对象名。
* amount：追加的长度。
* buffer：待追加的内容。

**示例**

调用writeappend函数将数据添加对lob的末尾，lob\_loc类型为clob。

1、创建含测试表。

create table bak\_dbms\_lob(

bak\_id number(4),

bak\_comment clob

);

2、插入测试数据。

insert into bak\_dbms\_lob(bak\_id,bak\_comment) values(1,'a');

insert into bak\_dbms\_lob(bak\_id,bak\_comment) values(2,'ab');

insert into bak\_dbms\_lob(bak\_id,bak\_comment) values(3,'abcdefgccccccc');

insert into bak\_dbms\_lob(bak\_id,bak\_comment) values(4,'a bcdefg');

select \* from bak\_dbms\_lob;

返回结果为：

bak\_id | bak\_comment

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

1 | a

2 | ab

3 | abcdefgccccccc

4 | a bcdefg

(4 rows)

3、调用调用writeappend函数。

set serveroutput on;

declare

lob\_loc clob;

amount integer:= 18;

buffer\_text varchar2(20) := 'added text to clob';

begin

select bak\_comment into lob\_loc from bak\_dbms\_lob where bak\_id = 2;

dbms\_lob.writeappend(lob\_loc,amount,buffer\_text);

dbms\_output.put\_line('添加后的字段为：' || lob\_loc);

end;

/

返回结果为：

添加后的字段为：abadded text to clob

ANONYMOUS BLOCK EXECUTE

**CREATETEMPORARY存储过程**

**语法格式**

DBMS\_LOB.CREATETEMPORARY (

lob\_loc IN OUT NOCOPY BLOB,

cache IN BOOLEAN,

dur IN PLS\_INTEGER := DBMS\_LOB.SESSION);

DBMS\_LOB.CREATETEMPORARY (

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

cache IN BOOLEAN,

dur IN PLS\_INTEGER := 10);

**参数说明**

* lob\_loc：LOB类型的对象名。
* cache：是否将LOB读取到缓冲区。
* dur：指定何时清除临时LOB（10/ SESSION：会话结束时；12/ CALL：调用结束时）。

**示例**

调用createtemporary存储过程，创建临时clob。

set serveroutput on;

DECLARE

v\_clob CLOB;

v\_amount INT;

v\_offset INT;

v\_char VARCHAR2(100);

BEGIN

v\_char := 'Chinese中国人';

v\_offset := 1;

v\_amount := 8;

dbms\_lob.createtemporary(v\_clob,TRUE,12);

FOR i IN 1..2 LOOP

dbms\_lob.writeappend(v\_clob,v\_amount,v\_char);

dbms\_output.put\_line(v\_clob||' 字符长度：'||dbms\_lob.getlength(v\_clob));

END LOOP;

END;

/

返回结果为：

Chinese中 字符长度：8

Chinese中Chinese中 字符长度：16

ANONYMOUS BLOCK EXECUTE

**FREETEMPORARY存储过程**

**语法格式**

DBMS\_LOB.FREETEMPORARY (

lob\_loc IN OUT NOCOPY BLOB);

DBMS\_LOB.FREETEMPORARY (

lob\_loc IN OUT NOCOPY CLOB CHARACTER SET ANY\_CS);

**参数说明**

lob\_loc：LOB类型的对象名。

**示例**

调用freetemporary存储过程，创建临时clob后释放。

set serveroutput on;

DECLARE

v\_clob CLOB;

v\_amount INT;

v\_offset INT;

v\_char VARCHAR2(100);

BEGIN

v\_char := 'Chinese中国人';

v\_offset := 1;

v\_amount := 8;

dbms\_lob.createtemporary(v\_clob,TRUE,12);

dbms\_lob.writeappend(v\_clob,v\_amount,v\_char);

dbms\_output.put\_line(v\_clob||' 字符长度：'||dbms\_lob.getlength(v\_clob));

dbms\_lob.freetemporary(v\_clob);

dbms\_output.put\_line(' 释放后再输出：'||v\_clob);

END;

/

返回结果为：

Chinese中 字符长度：8

释放后再输出：

ANONYMOUS BLOCK EXECUTE

**INSTR函数**

**语法格式**

DBMS\_LOB.INSTR (

lob\_loc IN BLOB,

pattern IN RAW,

offset IN INTEGER := 1,

nth IN INTEGER := 1)

RETURN INTEGER;

DBMS\_LOB.INSTR (

lob\_loc IN CLOB CHARACTER SET ANY\_CS,

pattern IN VARCHAR2 CHARACTER SET lob\_loc%CHARSET,

offset IN INTEGER := 1,

nth IN INTEGER := 1)

RETURN INTEGER;

DBMS\_LOB.INSTR (

file\_loc IN BFILE,

pattern IN RAW,

offset IN INTEGER := 1,

nth IN INTEGER := 1)

RETURN INTEGER;

**参数说明**

* lob\_loc：待查找的LOB对象。
* file\_loc：待查找的BFILE类型LOB对象。
* pattern：以字节为单位或以字符为单位的模式，用于匹配大对象中的内容。如果lob\_loc代表BLOB类型大对象或file\_loc代表BFILE类型大对象，那么模式必须是RAW类型。如果lob\_loc代表CLOB类型大对象，那么模式必须是VARCHAR2类型。
* offset：参数lob\_loc或file\_loc代表的大对象中搜索模式时的开始位置。第一个字节或字符是位置1。默认值为1。
* nth：由给定的偏移量指定起始位置，开始搜索指定模式第n次出现时的位置。默认值为1。

**返回值**

匹配模式的起始偏移量，以字节或字符为单位。如果找不到模式，则返回0。

以下几种情况下返回NULL：

* 任何一个或多个IN参数为NULL或无效。
* offset < 1 或者 offset > LOBMAXSIZE。
* nth < 1 或者 nth > LOBMAXSIZE

**示例**

调用instr函数。

select dbms\_lob.instr('blob test'::clob,'test',1,1);

返回结果为：

instr

-------

6

(1 row)

## DBMS\_LOCK

**功能描述**

DBMS\_LOCK包可以对数据库锁进行管理和控制。

比如某些开发程序，在高并发的情况下，必须控制好并发请求的运行时间和次序，

来保证处理数据的正确性和完整性。对于这种业务需求，可以使用内置包DBMS\_LOCK，把需要做并发控制的参数加上锁，实现并发控制。

dbms\_lock包提供了多种锁模式，各个模式都有自己的冲突列表，各个锁模式的含义及取值如下表：

|  |  |  |
| --- | --- | --- |
| **名字** | **描述** | **值** |
| nl\_mode | NULL | 1 |
| ss\_mode | Sub Shared，表示在一个对象的子部分获取共享锁 | 2 |
| sx\_mode | Sub Exclusive，表示在一个对象的子部分获取独占锁 | 3 |
| s\_mode | Shared，表示对整个对象获取共享锁 | 4 |
| ssx\_mode | Shared Sub Exclusive，表示整个对象已有共享锁，但是还要对某些子部分获取独占锁 | 5 |
| x\_mode | Exclusive，表示对整个对象获取独占锁 | 6 |

当要获取一个对象的锁时，如果这个对象的锁正被其他会话所持有，那么需要根据该对象的锁正被持有的模式以及当前要获取的锁的模式来判断本次是否能够成功获取，这就是不同锁模式的冲突。不同锁模式的冲突列表如下：

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **HELD MODE** | **GET NL** | **GET SS** | **GET SX** | **GET S** | **GET SSX** | **GET X** |
| NL | Success | Success | Success | Success | Success | Success |
| SS | Success | Success | Success | Success | Success | Failed |
| SX | Success | Success | Success | Success | Failed | Failed |
| S | Success | Success | Failed | Success | Failed | Failed |
| SSX | Success | Success | Failed | Failed | Failed | Failed |
| X | Success | Failed | Failed | Failed | Failed | Failed |

该内置包包含以下函数：

|  |  |
| --- | --- |
| **函数** | **描述** |
| [ALLOCATE\_UNIQUE](#ALLOCATE_UNIQUE) | 为指定的锁名分配一个唯一的锁id。 |
| [CONVERT](#CONVERT) | 锁模式的转换。 |
| [REQUEST](#REQUEST) | 请求一个指定模式的锁。 |
| [RELEASE](#RELEASE) | 用来显示释放由REQUEST函数占有的锁。 |
| [SLEEP](#SLEEP) | 指定session睡眠时间。 |

**注意事项**

无。

**兼容性**

Vastbase G100 V2.2版本与Oracle 11g R2版本相比，不支持的函数有:ALLOCATE\_UNIQUE\_AUTONOMOUS。

**示例**

1、 创建并切换至兼容模式为Oracle的数据库db\_oracle。

CREATE DATABASE db\_oracle dbcompatibility='A';

\c db\_oracle

2、创建函数并发控制锁模式nl\_mode与sx\_mode。

create or replace function lock\_test(lock\_mode int ,lock\_expired int ,release\_on\_commit boolean) returns int as $$ --lock\_mode取1到6整数

declare

v\_ret int;

v\_name varchar(100) :='test';

v\_handle varchar(100);

begin

dbms\_output.enable();

dbms\_output.serveroutput(true);

dbms\_output.put\_line('----- session start -----');

select dbms\_lock.allocate\_unique(v\_name,lock\_expired) into v\_handle;

v\_ret :=dbms\_lock.request(v\_handle , lock\_mode , lock\_expired , release\_on\_commit);

if v\_ret <> 0 then

--获取锁失败

dbms\_output.put\_line('---lock request lock failed ---');

return v\_ret;

else

dbms\_output.put\_line('---now request lock success ---');

end if;

--获取锁成功，此处添加请求业务逻辑

dbms\_lock.sleep(lock\_expired+20);

--执行完成，应该要释放掉锁

if v\_ret = 0 then

v\_ret := dbms\_lock.release(v\_handle);

end if;

return v\_ret;

end;

$$ language plpgsql;

3、会话1中执行 CALL lock\_test(1,2,false)，切换至会话2同时执行CALL lock\_test(3,20,false)。

CALL lock\_test(1,2,false);

CALL lock\_test(3,20,false);

会话1和会话2可以同时获得锁，同时出现如下执行结果：

----- session start -----

---now request lock success ---

lock\_test

-----------

0

(1 row)

4、 会话1执行 CALL lock\_test(1,2,true),切换至会话2同时执行CALL lock\_test(3,20,true);

CALL lock\_test(1,2,true);

CALL lock\_test(3,20,true);

会话1和会话2可以同时获得锁，同时出现如下执行结果：

----- session start -----

---now request lock success ---

lock\_test

-----------

0

(1 row)

**ALLOCATE\_UNIQUE**

**语法格式**

dbms\_lock.allocate\_unique(

lockname IN VARCHAR2,

lockhandle OUT VARCHAR2,

expiration\_secs IN INTEGER DEFAULT 864000

) RETURN INTEGER;

**参数说明**

* lockname：产生唯一的LockID。大小不超过128B，大小写敏感。不能以'ORA$'开头，这是为oracle提供的产品保留的。
* lockhandle：返回值，request，convert，release调用。
* expiration\_secs：执行'allocate\_unique'后，Clean Up的时间间隔。

**CONVERT**

**语法格式**

dbms\_lock.convert(

lockhandle IN VARCHAR2,

lockmode IN INTEGER,

timeout IN NUMBER DEFAULT maxwait

) RETURN INTEGER;

**参数说明**

* lockname：生成的唯一ID的锁的名称。
* lockhandle：返回值，request，convert，release调用。
* expiration\_secs：执行'allocate\_unique'后，Clean Up的时间间隔。

**RELEASE**

**语法格式**

dbms\_lock.release(

lockhandle IN VARCHAR2

) RETURN INTEGER;

dbms\_lock.release(

id IN INTEGER

) RETURN INTEGER;

**参数说明**

lockhandle，id：用户分配的锁定标识符。

**REQUEST**

**语法格式**

dbms\_lock.request(

lockhandle IN VARCHAR2,

lockmode IN INTEGER DEFAULT x\_mode,

timeout IN INTEGER DEFAULT maxwait,

release\_on\_commit IN BOOLEAN DEFAULT FALSE

) RETURN INTEGER;

**参数说明**

* lockhandle，id：用户分配的锁定标识符，allocate\_unique取得的handle。
* lockmode：请求的锁的模式。
* timeout：继续尝试授予锁定的秒数，如果在此时间段内无法授予锁，则调用返回值 1 ( timeout)。
* release\_on\_commit：将此参数设置为TRUE以释放提交或回滚时的锁定。否则，锁会一直持有，直到它被显式释放或会话结束。默认为FALSE。

**SLEEP**

**语法格式**

dbms\_lock.sleep(

seconds IN NUMBER

);

**参数说明**

seconds：睡眠时间。

## DBMS\_OBFUSCATION\_TOOLKIT

**功能描述**

DBMS\_OBFUSCATION\_TOOLKIT提供用于加密和解密存储的数据的接口，可以对数据进行DES、Triple DES或者MD5加密。该内置包包含以下函数：

|  |  |
| --- | --- |
| **函数** | **描述** |
| [DESENCRYPT](#TOOKDESENCRYPT) | 用于使用DES算法对输入数据进行加密，并生成加密格式的数据。 |
| [DESDECRYPT](#TOOKDESDECRYPT) | 用于对使用DES算法所生成的加密数据进行解密。 |
| [DESGETKEY](#TOOKDESGETKEY) | 生成一个随机字符串，并对其使用DES算法生成一个加密密钥。 |
| [DES3ENCRYPT](#TOOKDES3ENCRYPT) | 用于使用DES3算法对输入数据进行加密，并生成加密格式的数据。 |
| [DES3DECRYPT](#TOOKDES3DECRYPT) | 用于对使用DES3算法所生成的加密数据进行解密。 |
| [DES3GETKEY](#TOOKDES3GETKEY) | 生成一个随机字符串，并对其使用DES3算法生成一个加密密钥。 |
| [MD5](#TOOKMD5) | 用于使用md5算法生成密码校验码。 |

**注意事项**

* 使用DBMS\_OBFUSCATION\_TOOLKIT内置包前，需要先加载pgcrypto插件。

CREATE EXTENSION pgcrypto;

* 该功能仅在数据库兼容模式为Oracle时能够使用（即创建DB时DBCOMPATIBILITY='A'），在其他数据库兼容模式下不能使用该特性。

**兼容性**

完全兼容。

**DESENCRYPT**

**语法格式**

DBMS\_OBFUSCATION\_TOOLKIT.DESEncrypt(

input IN RAW,

key IN RAW,

encrypted\_data OUT RAW);

DBMS\_OBFUSCATION\_TOOLKIT.DESEncrypt(

input\_string IN VARCHAR2,

key\_string IN VARCHAR2,

encrypted\_string OUT VARCHAR2);

DBMS\_OBFUSCATION\_TOOLKIT.DESEncrypt(

input IN RAW,

key IN RAW)

RETURN RAW;

DBMS\_OBFUSCATION\_TOOLKIT.DESEncrypt(

input\_string IN VARCHAR2,

key\_string IN VARCHAR2)

RETURN VARCHAR2;

**参数说明**

* input：要加密的数据。
* key：加密密钥。
* encrypted\_data：加密数据。
* input\_string：要加密的字符串。
* key\_string：加密密钥字符串。
* encrypted\_string：加密字符串。

**示例**

* 对raw类型进行加密。

select dbms\_obfuscation\_toolkit.desencrypt('ABCDEFEF'::raw,'12345678'::raw);

返回结果为：

desencrypt

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

5837505A4D3578332F44383D

(1 row)

* 对text类型进行加密。

select dbms\_obfuscation\_toolkit.desencrypt('ABCDEFGH'::text,'12345678'::text);

返回结果为：

desencrypt

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

lt5gPq7WJW/hywLARpfLNA==

(1 row)

* 对中文进行加密。

select dbms\_obfuscation\_toolkit.desencrypt('中文中文中文中文'::text,'测试'::text);

返回结果为：

desencrypt

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

uJ0LFYdRFGzJdmhYnaG9XXKrM4wcJMz6ncu4SV204yM=

(1 row)

**DESDECRYPT**

**语法格式**

DBMS\_OBFUSCATION\_TOOLKIT.DESDecrypt(

input IN RAW,

key IN RAW,

decrypted\_data OUT RAW);

DBMS\_OBFUSCATION\_TOOLKIT.DESDecrypt(

input\_string IN VARCHAR2,

key\_string IN VARCHAR2,

decrypted\_string OUT VARCHAR2);

DBMS\_OBFUSCATION\_TOOLKIT.DESDecrypt(

input IN RAW,

key IN RAW)

RETURN RAW;

DBMS\_OBFUSCATION\_TOOLKIT.DESDecrypt(

input\_string IN VARCHAR2,

key\_string IN VARCHAR2)

RETURN VARCHAR2;

**参数说明**

* input：要加密的数据。
* key：加密密钥。
* decrypted\_data：加密数据。
* input\_string：要加密的字符串。
* key\_string：加密密钥字符串。
* decrypted\_string：加密字符串。

**示例**

* 对raw类型进行解密。

select dbms\_obfuscation\_toolkit.desdecrypt(dbms\_obfuscation\_toolkit.desencrypt('ABCDEFEF'::raw,'12345678'::raw),'12345678'::raw);

返回结果为：

desdecrypt

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

ABCDEFEF

(1 row)

* 对text类型进行解密。

select dbms\_obfuscation\_toolkit.desdecrypt(dbms\_obfuscation\_toolkit.desencrypt('ABCDEFGH'::text,'12345678'::text) ,'12345678'::text);

返回结果为：

desdecrypt

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

ABCDEFGH

(1 row)

**DESGETKEY**

**语法格式**

DBMS\_OBFUSCATION\_TOOLKIT.DESGetKey(

seed IN RAW,

key OUT RAW);

DBMS\_OBFUSCATION\_TOOLKIT.DESGetKey(

seed\_string IN VARCHAR2,

key OUT VARCHAR2);

DBMS\_OBFUSCATION\_TOOLKIT.DESGetKey(

seed IN RAW)

RETURN RAW;

DBMS\_OBFUSCATION\_TOOLKIT.DESGetKey(

seed\_string IN VARCHAR2)

RETURN VARCHAR2;

**参数说明**

* seed：长度至少未80个字符的值。
* key：加密密钥。
* key\_string：加密密钥字符串。

**示例**

* 用des算法随机生成加密密钥。

select dbms\_obfuscation\_toolkit.desgetkey('sucesss');

返回结果为：

desgetkey

-----------

yxvntnex

(1 row)

* 用des算法随机生成text类型的加密密钥。

select dbms\_obfuscation\_toolkit.desgetkey('sucesss'::text);

返回结果为：

desgetkey

-----------

zmuewwki

(1 row)

* 用des算法随机生成raw类型的加密密钥。

select dbms\_obfuscation\_toolkit.desgetkey('ABCDEFEF'::raw);

返回结果为：

desgetkey

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

6F746E7073696F71

(1 row)

**DES3ENCRYPT**

**语法格式**

DBMS\_OBFUSCATION\_TOOLKIT.DES3Encrypt(

input IN RAW,

key IN RAW,

encrypted\_data OUT RAW,

which IN PLS\_INTEGER DEFAULT TwoKeyMode

iv IN RAW DEFAULT NULL);

DBMS\_OBFUSCATION\_TOOLKIT.DES3Encrypt(

input\_string IN VARCHAR2,

key\_string IN VARCHAR2,

encrypted\_string OUT VARCHAR2,

which IN PLS\_INTEGER DEFAULT TwoKeyMode

iv\_string IN VARCHAR2 DEFAULT NULL);

DBMS\_OBFUSCATION\_TOOLKIT.DES3Encrypt(

input IN RAW,

key IN RAW,

which IN PLS\_INTEGER DEFAULT TwoKeyMode

iv IN RAW DEFAULT NULL)

RETURN RAW;

DBMS\_OBFUSCATION\_TOOLKIT.DES3Encrypt(

input\_string IN VARCHAR2,

key\_string IN VARCHAR2,

which IN PLS\_INTEGER DEFAULT TwoKeyMode

iv\_string IN VARCHAR2 DEFAULT NULL)

RETURN VARCHAR2;

**参数说明**

* input：要加密的数据。
* key：加密密钥。
* encrypted\_data：加密数据。
* which：如果 = 0，（默认），则TwoKeyMode使用。如果 = 1，则ThreeKeyMode使用。
* iv：初始化向量。
* input\_string：要加密的字符串。
* key\_string：加密密钥字符串。
* encrypted\_string：加密字符串。
* iv\_string：初始化向量。

**示例**

* 对text类型数据加密，密钥长度大于等于16。

select dbms\_obfuscation\_toolkit.des3encrypt('ABCDEFEF'::text,'ABCABCABCABCABCA'::text);

返回结果为：

des3encrypt

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

K9pRtTB3V7mVThiNt+zYmg==

(1 row)

**DES3GETKEY**

**语法格式**

DBMS\_OBFUSCATION\_TOOLKIT.DES3GetKey(

which IN PLS\_INTEGER DEFAULT TwoKeyMode,

seed IN RAW,

key OUT RAW);

DBMS\_OBFUSCATION\_TOOLKIT.DES3GetKey(

which IN PLS\_INTEGER DEFAULT TwoKeyMode,

seed\_string IN VARCHAR2,

key OUT VARCHAR2);

DBMS\_OBFUSCATION\_TOOLKIT.DES3GetKey(

which IN PLS\_INTEGER DEFAULT TwoKeyMode,

seed IN RAW)

RETURN RAW;

DBMS\_OBFUSCATION\_TOOLKIT.DES3GetKey(

which IN PLS\_INTEGER DEFAULT TwoKeyMode,

seed\_string IN VARCHAR2)

RETURN VARCHAR2;

**参数说明**

* which：如果 = 0，（默认），则TwoKeyMode使用。如果 = 1，则ThreeKeyMode使用。
* seed：长度至少为80个字符的值。
* key：加密密钥。
* seed\_string：长度至少为80个字符的字符串。

**示例**

* 对text类型数据加密，加密数据字符长度大于等于80个。

select dbms\_obfuscation\_toolkit.des3getkey(0,'ABCDEFEFABCDEFEFABCDEFEFABCDEFEFABCDEFEFABCDEFEFABCDEFEFABCDEFEFABCDEFEFABCDEFFF'::text);

返回结果为：

des3getkey

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

soQKaLCAvdTZXPxJ

(1 row)

* 对raw类型数据加密，加密数据字符长度小于80个

select dbms\_obfuscation\_toolkit.des3getkey('ABCDEFEF'::raw);

返回结果为：

des3getkey

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

69736F6E6662706D7661716768796374

(1 row)

**MD5**

**语法格式**

DBMS\_OBFUSCATION\_TOOLKIT.MD5(

input IN RAW,

checksum OUT raw\_checksum);

DBMS\_OBFUSCATION\_TOOLKIT.MD5(

input\_string IN VARCHAR2,

checksum\_string OUT varchar2\_checksum);

DBMS\_OBFUSCATION\_TOOLKIT.MD5(

input IN RAW)

RETURN raw\_checksum;

DBMS\_OBFUSCATION\_TOOLKIT.MD5(

input\_string IN VARCHAR2)

RETURN varchar2\_checksum;

**参数说明**

* input：要生成校验码的数据。
* checksum：128位加密信息摘要。
* input\_string：要生成校验码的字符串。
* checksum\_string：128位加密信息摘要。

**示例**

* 用md5算法随机生成raw类型密码校验码。

select dbms\_obfuscation\_toolkit.md5('ABCDEFEF'::raw);

返回结果为：

md5

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

42BE4B48002C72898ADEDCBA9C6370F6

(1 row)

* 用md5算法随机生成text类型密码校验码。

select dbms\_obfuscation\_toolkit.md5('ABCDEFEF'::text);

返回结果为：

md5

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

x18Mr\_x10s"

(1 row)

## DBMS\_OUTPUT

**功能描述**

DBMS\_OUTPUT内置包用于提供在存储过程或包内对外发送信息的功能，该内置包包含以下函数：

|  |  |
| --- | --- |
| **函数** | **描述** |
| enable | 用来使dbms\_output生效。 |
| disable | 用来使dbms\_output失效。 |
| put | 将数据写入到缓存中，等到put\_line/new\_line时一起输出。 |
| put\_line | 将数据写入到缓存中，并将缓存中的数据一起输出。 |
| new\_line | 向put写入的行末尾添加终结符，如果缓存中有数据将其输出。 |
| get\_line | 从缓存中读取一行数据。 |
| get\_lines | 从缓存中读取多行数据。 |

说明：serveroutput 为on 时允许将dbms\_output.put\_line的输出信息输出至vsql的命令界面的屏幕上。使用以下语句设置参数：

SET serveroutput =on

serveroutput默认值是off 可以通过配置.psqlrc文件设置缺省值。

**注意事项**

该功能仅在数据库兼容模式为Oracle时能够使用（即创建DB时DBCOMPATIBILITY='A'），在其他数据库兼容模式下不能使用该特性。

**示例**

1、创建并切换至兼容模式为ORACLE的数据库db\_oracle。

CREATE DATABASE db\_oracle dbcompatibility=’B’;

\c db\_oracle

2、创建测试表并插入数据。

CREATE TABLE emp (

empno NUMBER(4) ,

ename VARCHAR2(10),

job VARCHAR2(9),

mgr NUMBER(4),

hiredate DATE,

sal NUMBER(7,2) CONSTRAINT emp\_sal\_ck CHECK (sal > 0),

comm NUMBER(7,2));

INSERT INTO emp VALUES (7369,'SMITH','CLERK',7902,'17-DEC-80',800,NULL);

INSERT INTO emp VALUES (7499,'ALLEN','SALESMAN',7698,'20-FEB-81',1600,300);

INSERT INTO emp VALUES (7521,'WARD','SALESMAN',7698,'22-FEB-81',1250,500);

INSERT INTO emp VALUES (7566,'JONES','MANAGER',7839,'02-APR-81',2975,NULL);

INSERT INTO emp VALUES (7654,'MARTIN','SALESMAN',7698,'28-SEP-81',1250,1400);

INSERT INTO emp VALUES (7698,'BLAKE','MANAGER',7839,'01-MAY-81',2850,NULL);

INSERT INTO emp VALUES (7782,'CLARK','MANAGER',7839,'09-JUN-81',2450,NULL);

INSERT INTO emp VALUES (7788,'SCOTT','ANALYST',7566,'19-APR-87',3000,NULL);

INSERT INTO emp VALUES (7839,'KING','PRESIDENT',NULL,'17-NOV-81',5000,NULL);

INSERT INTO emp VALUES (7844,'TURNER','SALESMAN',7698,'08-SEP-81',1500,0);

INSERT INTO emp VALUES (7876,'ADAMS','CLERK',7788,'23-MAY-87',1100,NULL);

INSERT INTO emp VALUES (7900,'JAMES','CLERK',7698,'03-DEC-81',950,NULL);

INSERT INTO emp VALUES (7902,'FORD','ANALYST',7566,'03-DEC-81',3000,NULL);

INSERT INTO emp VALUES (7934,'MILLER','CLERK',7782,'23-JAN-82',1300,NULL);

3、创建并调用存储过程。

CREATE OR REPLACE PROCEDURE list\_emp()

IS

v\_empno NUMBER(4);

v\_ename VARCHAR2(10);

cursor emp\_cur is SELECT empno, ename FROM emp ORDER BY empno;

BEGIN

OPEN emp\_cur;

DBMS\_OUTPUT.PUT\_LINE('EMPNO ENAME');

DBMS\_OUTPUT.PUT\_LINE('----- -------');

LOOP

FETCH emp\_cur INTO v\_empno, v\_ename;

EXIT WHEN emp\_cur%NOTFOUND;

DBMS\_OUTPUT.PUT\_LINE(v\_empno || ' ' || v\_ename);

END LOOP;

CLOSE emp\_cur;

END;

/

vastbase=# call list\_emp();

返回结果为：

EMPNO ENAME

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

7369 SMITH

7499 ALLEN

7521 WARD

7566 JONES

7654 MARTIN

7698 BLAKE

7782 CLARK

7788 SCOTT

7839 KING

7844 TURNER

7876 ADAMS

7900 JAMES

7902 FORD

7934 MILLER

list\_emp

----------

(1 row)

## DBMS\_PIPE

**功能描述**

dbms\_pipe包用于在同一个实例中，通过一个管道来实现会话内或会话之间的消息传递。该内置包包含以下函数：

|  |  |
| --- | --- |
| **函数** | **描述** |
| [CREATE\_PIPE](#PIPECREATE_PIPE) | 使用指定的名称显示地创建了一个公有或私有的管道。 |
| [NEXT\_ITEM\_TYPE](#PIPENEXT_ITEM_TYPE) | 返回消息缓存中下一个数据成员的数据类型。 |
| [PACK\_MESSAGE](#PIPEPACK_MESSAGE) | 在本地缓存中构建消息。 |
| [PURGE](#PIPEPURGE) | 清除指定的管道中的消息。 |
| [RECEIVE\_MESSAGE](#PIPERECEIVE_MESSAGE) | 将消息从命名管道复制到本地缓冲区 |
| [REMOVE\_PIPE](#PIPEREMOVE_PIPE) | 清除指定的管道。 |
| [RESET\_BUFFER](#PIPERESET_BUFFER) | 清除本地缓存中的内容。 |
| [SEND\_MESSAGE](#PIPESEND_MESSAGE) | 从会话的本地消息缓冲区中将一条消息发送到指定的管道中。 |
| [UNIQUE\_SESSION\_NAME](#PIPEUNIQUE_SESSION_NAME) | 返回当前会话的一个唯一名称。 |
| [UNPACK\_MESSAGE](#PIPEUNPACK_MESSAGE) | 从本地缓存中获取消息。 |
| [LIST\_PIPES](#PIPELIST_PIPES) | 列出所有管道。 |
| 视图DB\_PIPES | 用于查询数据库中所有的管道。 |

**注意事项**

* 该功能仅在数据库兼容模式为Oracle时能够使用（即创建DB时DBCOMPATIBILITY='A'），在其他数据库兼容模式下不能使用该特性。
* 要执行dbms\_pipe中的函数，用户须授权执行如下命令：

select dbms\_output.serveroutput(true);

**兼容性**

完全兼容。

**示例**

* **CREATE\_PIPE**

创建公有管道

CREATE OR REPLACE FUNCTION test\_create\_pipeone() RETURNS void AS

$$

BEGIN

perform dbms\_pipe.create\_pipe('messageone');

END;

$$

LANGUAGE plpgsql;

select test\_create\_pipeone();

SELECT \* FROM dbms\_pipe.db\_pipes;

创建私有管道

CREATE OR REPLACE FUNCTION test\_create\_pipetwo() RETURNS void AS

$$

BEGIN

perform dbms\_pipe.create\_pipe('messagetwo', 32, true);

END;

$$

LANGUAGE plpgsql;

select test\_create\_pipetwo();

SELECT \* FROM dbms\_pipe.db\_pipes;

* **NEXT\_ITEM\_TYPE**

--会话1

CREATE OR REPLACE FUNCTION test\_next\_item\_type() RETURNS void AS

$$

DECLARE

v\_number NUMBER := 123;

v\_varchar TEXT := 'Character data';

v\_date DATE := date'2020-12-14';

BEGIN

perform dbms\_pipe.pack\_message(v\_number);

Perform dbms\_pipe.pack\_message(v\_varchar);

Perform dbms\_pipe.pack\_message(v\_date);

dbms\_pipe.send\_message('datatypes');

END;

$$

LANGUAGE plpgsql;

select test\_next\_item\_type();

--会话2

CREATE OR REPLACE FUNCTION test\_next\_item\_type1() RETURNS void AS

$$

DECLARE

v\_number NUMBER;

v\_varchar TEXT;

v\_timestamptz TIMESTAMPTZ;

status INTEGER;

BEGIN

dbms\_pipe.receive\_message('datatypes');

-- 返回9

status := dbms\_pipe.next\_item\_type();

dbms\_output.put\_line('next\_item\_type:' || status);

v\_number := dbms\_pipe.unpack\_message\_number();

dbms\_output.put\_line('number item:' || v\_number);

-- 返回11

status := dbms\_pipe.next\_item\_type();

dbms\_output.put\_line('next\_item\_type:' || status);

v\_varchar := dbms\_pipe.unpack\_message\_text();

dbms\_output.put\_line('varchar item:' || v\_varchar);

-- 返回13

status := dbms\_pipe.next\_item\_type();

dbms\_output.put\_line('next\_item\_type:' || status);

v\_timestamptz:= dbms\_pipe.unpack\_message\_timestamp();

dbms\_output.put\_line('timestamptz item:' || v\_timestamptz);

status := dbms\_pipe.next\_item\_type();

dbms\_output.put\_line('next\_item\_type:' || status);

END;

$$

LANGUAGE plpgsql;

select test\_next\_item\_type1();

* **PACK\_MESSAGE**

CREATE OR REPLACE FUNCTION test\_pack\_message1() RETURNS void AS

$$

DECLARE

v\_number NUMBER := 123;

v\_varchar TEXT := 'Character data';

status INTEGER;

BEGIN

perform dbms\_pipe.pack\_message(v\_number);

Perform dbms\_pipe.pack\_message(v\_varchar);

status := dbms\_pipe.send\_message('datatypes');

dbms\_output.put\_line('send\_message status:' || status);

END;

$$

LANGUAGE plpgsql;

select test\_pack\_message1();

* **PURGE**

--会话1

CREATE OR REPLACE FUNCTION test\_create\_pipe2() RETURNS void AS

$$

DECLARE

BEGIN

perform dbms\_pipe.create\_pipe('message');

perform dbms\_pipe.pack\_message('test1');

dbms\_pipe.send\_message('message');

Perform dbms\_pipe.pack\_message('test2');

dbms\_pipe.send\_message('message');

END;

$$

LANGUAGE plpgsql;

--会话2

CREATE OR REPLACE FUNCTION test\_purge() RETURNS void AS

$$

DECLARE

v\_item TEXT;

BEGIN

dbms\_pipe.receive\_message('message', 10);

v\_item := dbms\_pipe.unpack\_message\_text();

dbms\_output.put\_line('Item:' || v\_item);

END;

$$

LANGUAGE plpgsql;

--删除管道中的剩余的message 2

SELECT dbms\_pipe.purge('pipe');

* **RECEIVE\_MESSAGE**

--会话1

select dbms\_pipe.pack\_message('u r testing receive\_message');

select dbms\_pipe.send\_message('test\_receive');

--会话2

select dbms\_pipe.receive\_message('test\_receive');

select dbms\_pipe.unpack\_message\_text();

* **REMOVE\_PIPE**

select dbms\_pipe.create\_pipe('test\_remove');

select \* from dbms\_pipe.db\_pipes;

select dbms\_pipe.remove\_pipe('test\_remove');

* **RESET\_BUFFER**

--会话1

select dbms\_pipe.pack\_message('u r testing reset\_buffer');

select dbms\_pipe.send\_message('reset\_buffer');

--会话2

select dbms\_pipe.receive\_message('reset\_buffer');

select dbms\_pipe.reset\_buffer();

select dbms\_pipe.unpack\_message\_text();

* **SEND\_MESSAGE**

CREATE OR REPLACE FUNCTION test\_send\_message1() RETURNS void AS

$$

DECLARE

BEGIN

perform dbms\_pipe.create\_pipe('message1',1,false);

perform dbms\_pipe.pack\_message('test1');

dbms\_pipe.send\_message('message1');

Perform dbms\_pipe.pack\_message('test2');

dbms\_pipe.send\_message('message1');

END;

$$

LANGUAGE plpgsql;

select test\_send\_message1();

* **UNIQUE\_SESSION\_NAME**

CREATE OR REPLACE FUNCTION test\_unique\_session\_name() RETURNS void AS

$$

DECLARE

v\_session TEXT;

BEGIN

v\_session := dbms\_pipe.unique\_session\_name();

dbms\_output.put\_line('unique\_session\_name:' || v\_session);

END;

$$

LANGUAGE plpgsql;

select test\_unique\_session\_name();

* **UNPACK\_MESSAGE**

--会话1

CREATE OR REPLACE FUNCTION test\_next\_item\_type() RETURNS void AS

$$

DECLARE

v\_number NUMBER := 123;

v\_varchar TEXT := 'Character data';

v\_date DATE := date'2020-12-14';

BEGIN

perform dbms\_pipe.pack\_message(v\_number);

Perform dbms\_pipe.pack\_message(v\_varchar);

Perform dbms\_pipe.pack\_message(v\_date);

dbms\_pipe.send\_message('datatypes');

END;

$$

LANGUAGE plpgsql;

Select test\_next\_item\_type();

--会话2

CREATE OR REPLACE FUNCTION test\_next\_item\_type1() RETURNS void AS

$$

DECLARE

v\_number NUMBER;

v\_varchar TEXT;

v\_timestamptz TIMESTAMPTZ;

status INTEGER;

BEGIN

dbms\_pipe.receive\_message('datatypes');

-- 返回9

status := dbms\_pipe.next\_item\_type();

dbms\_output.put\_line('next\_item\_type:' || status);

v\_number := dbms\_pipe.unpack\_message\_number();

dbms\_output.put\_line('number item:' || v\_number);

-- 返回11

status := dbms\_pipe.next\_item\_type();

dbms\_output.put\_line('next\_item\_type:' || status);

v\_varchar := dbms\_pipe.unpack\_message\_text();

dbms\_output.put\_line('varchar item:' || v\_varchar);

-- 返回13

status := dbms\_pipe.next\_item\_type();

dbms\_output.put\_line('next\_item\_type:' || status);

v\_timestamptz:= dbms\_pipe.unpack\_message\_timestamp();

dbms\_output.put\_line('timestamptz item:' || v\_timestamptz);

status := dbms\_pipe.next\_item\_type();

dbms\_output.put\_line('next\_item\_type:' || status);

END;

$$

LANGUAGE plpgsql;

select test\_next\_item\_type1();

* **LIST\_PIPES**

--查询所有管道信息

SELECT name, items, "limit", private, owner

FROM dbms\_pipe.list\_pipes()

AS (name varchar, items integer, size integer, "limit" integer, private bool, owner varchar);

* **视图DB\_PIPES**

select \* from dbms\_pipe.db\_pipes;

**CREATE\_PIPE**

**语法格式**

DBMS\_PIPE.CREATE\_PIPE (

pipename IN VARCHAR2,

maxpipesize IN INTEGER DEFAULT 8192,

private IN BOOLEAN DEFAULT TRUE)

RETURN INTEGER;

**参数说明**

* pipename： 用户创建的管道名称。
* maxpipesize： 管道允许的最大大小，以字节为单位。
* private ：使用默认值TRUE创建私有管道。调用时可以隐式创建公共管道SEND\_MESSAGE。

**NEXT\_ITEM\_TYPE**

**语法格式**

DBMS\_PIPE.NEXT\_ITEM\_TYPE

RETURN INTEGER;

**PACK\_MESSAGE**

**语法格式**

DBMS\_PIPE.PACK\_MESSAGE (

item IN VARCHAR2|NCHAR|

NUMBER|DATE|RAW|ROWID);

**参数说明**

item：要打包到本地消息缓冲区的项目。

**PURGE**

**语法格式**

DBMS\_PIPE.PURGE (

pipename IN VARCHAR2);

**参数说明**

pipename： 要删除消息的管道名称。

**RECEIVE\_MESSAGE**

**语法格式**

DBMS\_PIPE.RECEIVE\_MESSAGE (

pipename IN VARCHAR2,

timeout IN INTEGER DEFAULT maxwait)

RETURN INTEGER;

**参数说明**

* pipename： 接收消息的管道名称。
* timeout：等待消息的时间，以秒为单位。

**UNIQUE\_SESSION\_NAME**

**语法格式**

DBMS\_PIPE.UNIQUE\_SESSION\_NAME

RETURN VARCHAR2;

**UNPACK\_MESSAGE**

**语法格式**

DBMS\_PIPE.UNPACK\_MESSAGE (

item OUT VARCHAR2|NCHAR|NUMBER|DATE|RAW|ROWID);

**参数说明**

item：从本地消息缓冲区接收下一个解包项目的参数

**LIST\_PIPES**

**语法格式**

DBMS\_PIPE.LIST\_PIPES() ;

## DBMS\_PROFILER

**功能描述**

DBMS\_PROFILER提供了一个接口，用于分析现有的PL/pgSQL应用程序并确定性能瓶颈。该内置包可以收集性能分析数据，用来提高性能或确定PL/pgSQL应用程序的代码覆盖率。

应用程序开发人员可以使用代码覆盖率数据来集中精力进行增量测试。使用该内置包的接口，可以为在会话中执行的所有命名库单元（package body、function、procedure、trigger、anonymous block）生成分析信息。该信息包括执行每行的总次数、执行该行所花费的总时间以及该行的特定执行所花费的最短和最长时间。分析信息存储在数据库表中，可查询数据、构建自定义报告（摘要报告、最热的行、代码覆盖率数据等）以及分析数据。该内置包包含以下函数：

|  |  |
| --- | --- |
| **函数** | **描述** |
| START\_PROFILER | 在用户会话中启动事件探查器数据收集。启动一次数据收集就要START\_PROFILER，中间可以暂停、继续、刷新数据。如果一直不STOP\_PROFILER，收集器就一直在收集 plpgsql执行的性能数据；如果不执行START\_PROFILER就执行STOP\_PROFILER、 暂停、继续、刷新数据也不会出错，但没有实际数据收集。 |
| STOP\_PROFILER | 停止用户会话中的探查器数据收集。此功能将会刷新会话中已收集的数据。 |
| PAUSE\_PROFILER | 暂停用户会话中的探查器数据收集。 |
| RESUME\_PROFILER | 恢复暂停状态中的探查器数据收集。 |
| FLUSH\_DATA | 刷新在用户会话中收集的探查器数据。 |
| ROLLUP\_UNIT | 计算对应runid和unit所花费的时间，并将总花费的时间刷新到PLSQL\_PROFILER\_UNITS表中的total\_time字段中。 |
| ROLLUP\_RUN | 统计指定runid收集器下所有unit所花费的时间，并将花费时间刷新到 PLSQL\_PROFILER\_UNITS表中的total\_time字段中。 |

**注意事项**

* 使用DBMS\_PROFILER内置包前，需要先加载安装目录/share/postgresql目录下的proload.sql文件。加载命令如下（该步骤需要在目标库内部执行）：

\i ./local/vastbase/share/postgresql/proload.sql;

如果该命令执行报错，可以找到preload.sql文件手动执行preload.sql文件中命令创建收集器。收集器相关表说明如下：

|  |  |
| --- | --- |
| **收集器相关表名** | **描述** |
| plsql\_profiler\_runs | 记录运行中的收集器信息。 |
| plsql\_profiler\_units | 有关运行中每个库单元的信息。 |
| plsql\_profiler\_data | 收集器收集的详细数据。 |
| plsql\_profiler\_runnumber序列 | 生成唯一运行编号。 |

* 该功能仅在数据库兼容模式为Oracle时能够使用（即创建DB时DBCOMPATIBILITY='A'），在其他数据库兼容模式下不能使用该特性。
* 可以通过profiler.sql工具产生HTML格式的性能报表，具体使用方法见示例。

**兼容性**

完全兼容。

**示例**

1、创建并切换至兼容模式为Oracle的数据库db\_oracle。

CREATE DATABASE db\_oracle dbcompatibility='A';

\c db\_oracle

2、创建测试表。

create table tab\_test(col int);

3、创建要统计PL/pgSQL性能的存储过程sp\_test。

create or replace procedure sp\_test

as

begin

for i in 1..100

loop

insert into tab\_test values(1);

end loop;

end;

/

4、创建要统计PL/pgSQL性能的存储过程sp\_test2。

create or replace procedure sp\_test2

as

begin

for i in 1..10000

loop

insert into tab\_test values(i);

end loop;

end;

/

5、统计sp\_test的执行性能。

select dbms\_profiler.start\_profiler('sp');

6、调用存储过程sp\_test,sp\_test2

call sp\_test();

select dbms\_profiler.flush\_data();

select dbms\_profiler.pause\_profiler();

call sp\_test2();

select dbms\_profiler.resume\_profiler();

select dbms\_profiler.stop\_profiler();

select dbms\_profiler.rollup\_unit(1,1);

select dbms\_profiler.rollup\_run(1);

7、查询结果。

select \* from plsql\_profiler\_runs;

返回结果为：

runid | related\_run | run\_owner | run\_date | run\_comment | run\_total\_time | run\_system\_info | run\_comment1 | spare1

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

1 | 0 | vastbase | 2022-06-28 01:58:56.578502 | sp | 6668 | | |

(1 row)

select \* from plsql\_profiler\_units;

runid | unit\_number | unit\_type | unit\_owner | unit\_name | unit\_timestamp | total\_time | spare1 | spare2

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

1 | 1 | PROCEDURE | vastbase | sp\_test | 2022-06-28 01:59:02.830512 | 3334 | |

(1 row)

select \* from plsql\_profiler\_data;

runid | unit\_number | line# | total\_occur | total\_time | min\_time | max\_time | spare1 | spare2 | spare3 | spare4

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

1 | 1 | 4 | 1 | 1730 | 1730 | 1730 | | | |

1 | 1 | 6 | 100 | 1604 | 9 | 26 | | | |

(2 rows)

## DBMS\_RANDOM

**功能描述**

dbms\_random是一个PL/SQL包，用于生成随机数和随机字符。该内置包包含以下函数：

|  |  |
| --- | --- |
| **函数** | **描述** |
| [INITIALIZE](#INITIALIZE) | 使用种子值初始化包。 |
| [NORMAL](#NORMAL) | 返回生态分布中随机数。 |
| [RANDOM](#RANDOM) | 生成一个随机数。 |
| [SEEd](#SEED) | 重置种子。 |
| [STRING](#STRING) | 获取随机字符串。 |
| [TERMINATE](#TERMINATE) | 终止包。 |
| [VALUE](#VALUE) | 获取一个大于等于0小于1的随机数，或者得到指定区间的随机数。 |

**注意事项**

无。

**兼容性**

完全兼容。

**INITIALIZE函数**

**语法格式**

DBMS\_RANDOM.INITIALIZE (

val IN BINARY\_INTEGER);

**参数说明**

val：用于生成随机数的种子号。

**示例**

select dbms\_random.initialize(123456);

initialize

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

(1 row)

**NORMAL函数**

**语法格式**

DBMS\_RANDOM.NORMAL

RETURN NUMBER;

**示例**

select dbms\_random.normal();

normal

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

1.560225125239

(1 row)

**RANDOM函数**

**语法格式**

DBMS\_RANDOM.RANDOM

RETURN binary\_integer;

**示例**

select dbms\_random.random();

random

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

1356653482

(1 row)

**SEED函数**

**语法格式**

DBMS\_RANDOM.SEED (

val IN BINARY\_INTEGER);

DBMS\_RANDOM.SEED (

val IN VARCHAR2);

**参数说明**

val：用于生成随机数的种子号或字符串。

**示例**

select dbms\_random.seed('asda');

seed

------

(1 row)

**STRING函数**

**语法格式**

DBMS\_RANDOM.STRING

opt IN CHAR,

len IN NUMBER)

RETURN VARCHAR2;

**参数说明**

* opt：字符串输出格式。
* len：字符串输出长度。

**示例**

select dbms\_random.string('x',9);

string

-----------

ZTZAXREFR

(1 row)

**TERMINATE函数**

**语法格式**

DBMS\_RANDOM.TERMINATE;

**示例**

select dbms\_random.terminate();

terminate

-----------

(1 row)

**VALUE函数**

**语法格式**

DBMS\_RANDOM.VALUE

RETURN NUMBER;

DBMS\_RANDOM.VALUE(

low IN NUMBER,

high IN NUMBER)

RETURN NUMBER;

**参数说明**

* low：生成随机数范围内的最小数，生成的数字可能等于low。
* high：生成随机数的最大值，生成的数字将小于high。

**示例**

vastbase=# select dbms\_random.value();

value

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

.48548461124301

(1 row)

vastbase=# select dbms\_random.value(100,2200);

value

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

1328.65713294595

(1 row)

## DBMS\_ROWID

**功能描述**

该内置包用于我们从PL/SQL和SQL中创建ROWIDs和获取关于ROWIDs的信息。该内置包包含以下函数：

* ROWID\_CREATE函数：rowid\_create函数用于创建一个rowid。

**语法格式**

dbms\_rowid.rowid\_create(

rowid\_type number,

object\_number number,

relative\_fno number,

block\_number number,

row\_number number);

**参数说明**

* rowid\_type：rowid类型（restricted或者extended），目前Vastbase G100的orwid只实现了extended类型，因此该参数目前不生效。
* object\_number：数据库对象oid，Vastbase G100中主要是表的oid。不能小于0，不能为小数。
* relative\_fno：所在数据库文件编号，Vastbase G100的rowid未实现此功能，因此不生效。
* block\_number：数据块编号。不能小于0，不能为小数。
* row\_number：块中偏移量。不能小于0，不能为小数。

**兼容性**

Vastbase G100 V2.2版本与Oracle 11g R2版本相比，不支持的函数有：ROWID\_BLOCK\_NUMBER、 ROWID\_INFO、 ROWID\_OBJECT、ROWID\_RELATIVE\_FNO、ROWID\_ROW\_NUMBER、ROWID\_TO\_ABSOLUTE\_FNO、ROWID\_TO\_EXTENDED、ROWID\_TO\_RESTRICTED、ROWID\_TYPE、ROWID\_VERIFY

**示例**

select dbms\_rowid.rowid\_create(0,1,1,1,1);

返回结果为：

rowid\_create

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

AQAAAA==AQAAAA==AQA=

(1 row)

## DBMS\_SQL

**DBMS\_SQL包中的数据类型**

**功能描述**

在dbms\_sql包中定义record类型和table类型的数据结构。如下数据结构方便在自定义函数或存储过程中进行变量的数据类型定义：

* Record类型有：desc\_rec、desc\_rec2、desc\_rec3、desc\_rec4
* Table类型有：desc\_tab、desc\_tab2、desc\_tab3、desc\_tab4、number\_table、varchar2\_table、date\_table、blob\_table、clob\_table、binary\_double\_table、binary\_float\_table
* 常量类型：dbms\_sql.native、dbms\_sql.v6、dbms\_sql.v7

**说明：**

Record类型记录动态查询中单个列的详细信息，table类型是对应record类型的集合。通常同时搭配着DESCRIBE\_COLUMNS等存储过程获取列信息。

**desc\_rec和desc\_tab类型**

**功能描述**

desc\_rec记录类型保存动态查询中单个列的描述信息。desc\_tab是一个表DESC\_REC Record 类型。

该类型与DESCRIBE\_COLUMNS存储过程搭配使用。

**语法格式**

* desc\_rec

TYPE desc\_rec IS RECORD (

col\_type BINARY\_INTEGER := 0,

col\_max\_len BINARY\_INTEGER := 0,

col\_name VARCHAR2(32) := '',

col\_name\_len BINARY\_INTEGER := 0,

col\_schema\_name VARCHAR2(32) := '',

col\_schema\_name\_len BINARY\_INTEGER := 0,

col\_precision BINARY\_INTEGER := 0,

col\_scale BINARY\_INTEGER := 0,

col\_charsetid BINARY\_INTEGER := 0,

col\_charsetform BINARY\_INTEGER := 0,

col\_null\_ok BOOLEAN := TRUE);

* desc\_tab

TYPE desc\_tab IS TABLE OF desc\_rec INDEX BY BINARY\_INTEGER;

**参数说明**

* col\_type：动态查询中单个列数据类型对应的编号，编号如下：

table th:nth-of-type(1) { width: 20%; }

| **Code** | **Data Type** |
| --- | --- |
| 1 | VARCHAR2(size [BYTE |
| 1 | NVARCHAR2(size) |
| 2 | NUMBER [ (p [, s]) ] |
| 2 | FLOAT [(p)] |
| 8 | LONG |
| 12 | DATE |
| 100 | BINARY\_FLOAT |
| 101 | BINARY\_DOUBLE |
| 180 | TIMESTAMP [(fractional\_seconds\_precision)] |
| 181 | TIMESTAMP [(fractional\_seconds\_precision)] WITH TIME ZONE |
| 231 | TIMESTAMP [(fractional\_seconds\_precision)] WITH LOCAL TIME ZONE |
| 182 | INTERVAL YEAR [(year\_precision)] TO MONTH |
| 183 | INTERVAL DAY [(day\_precision)] TO SECOND [(fractional\_seconds\_precision)] |
| 23 | RAW(size) |
| 24 | LONG RAW |
| 69 | ROWID |
| 208 | UROWID [(size)] |
| 96 | CHAR [(size [BYTE |
| 96 | NCHAR[(size)] |
| 112 | CLOB |
| 112 | NCLOB |
| 113 | BLOB |
| 114 | BFILE |
| 119 | JSON |

* col\_max\_len：动态查询中单个列中数据类型长度
* col\_name：动态查询中单个列名
* col\_name\_len：动态查询中单个列名字符长度
* col\_schema\_name：动态查询中单个列是自定义type类型返回自定义type所属的schema名，否则返回空
* col\_schema\_name\_len：动态查询中单个列的数据类型所属schema名字长度
* col\_precision：动态查询中单个列的数据类型精度，只有数字类型会存在精度，字符或时间等其他类型该值为0
* col\_scale：动态查询中单个列的数据类型允许小数点后几位数
* col\_charsetid：动态查询中单个列数据类型的字符集id，只有字符串类型存在字符集，其余数据类型字符集设置为0。目前只需要输出如下两种字符集类型即可：

| **字符集** | **字符集id** |
| --- | --- |
| GBK | 852 |
| UTF8 | 871 |

* col\_charsetform：动态查询中单个列数据类型的字符集，如果是字符串类型输出1，其余数据类型保持默认值0
* col\_null\_ok：动态查询中单个列的值是否为空

**desc\_rec2和desc\_tab2类型**

**功能描述**

DESC\_REC2是DESC\_TAB2表类型和DESCRIBE\_COLUMNS2存储过程的元素类型。

该类型与DESCRIBE\_COLUMNS2存储过程搭配使用。

**语法格式**

desc\_rec2

和desc\_rec类型一致，只是col\_name变量类型从VARCHAR2(32) => varchar2(32767)

desc\_tab2

TYPE desc\_tab2 IS TABLE OF desc\_rec2 INDEX BY BINARY\_INTEGER;

**desc\_rec3和desc\_ta3类型**

**功能描述**

DESC\_REC3是DESC\_TAB3表类型和DESCRIBE\_COLUMNS3存储过程的元素类型。

该类型与DESCRIBE\_COLUMNS3存储过程搭配使用。

**语法格式**

desc\_rec3

desc\_rec3相比desc\_rec2新增加了两个属性，分别是类型的名字和类型的长度。并且系统的数据类型不会进行填充，只有当是自定义type类型时候才会进行赋值。

desc\_tab3

TYPE desc\_tab3 IS TABLE OF desc\_rec3 INDEX BY BINARY\_INTEGER;

**desc\_rec4和desc\_ta4类型**

**功能描述**

DESC\_REC4是DESC\_TAB4表类型和DESCRIBE\_COLUMNS4存储过程的元素类型。

该类型与DESCRIBE\_COLUMNS4存储过程搭配使用。

**语法格式**

desc\_rec4

desc\_rec4保持和desc\_rec3的定义相同。

desc\_tab4

TYPE desc\_tab4 IS TABLE OF desc\_rec4 INDEX BY BINARY\_INTEGER;

**number\_table、varcha2\_table、date\_table、blob\_table、clob\_table、binary\_double\_table、binary\_float\_table**

**功能描述**

该类型与bind\_array存储过程一起调用，该存储过程根据传入动态SQL进行table变量进行绑定作用。

**语法格式**

DBMS\_SQL.BIND\_ARRAY (

c IN INTEGER,

name IN VARCHAR2,

<table\_variable> IN <datatype>

[,index1 IN INTEGER,

index2 IN INTEGER)] );

**DBMS\_SQL包中的子程序**

**功能描述**

DBMS\_SQL主要功能是提供用于执行动态SQL(DML和DDL)的方法。该内置包包含以下函数：

|  |  |
| --- | --- |
| **函数** | **描述** |
| [OPEN\_CURSOR](#OPEN_CURSOR) | 打开一个cursor，并返回该cursor的id。 |
| [CLOSE\_CURSOR](#CLOSE_CURSOR) | 用于关闭一个cursor。 |
| [COLUMN\_VALUE](#COLUMN_VALUE) | 用于将fetch结果中的指定列保存到返回参数中。 |
| [DEFINE\_COLUMN](#DEFINE_COLUMN) | 用于定义返回的结果中的列结构。 |
| [EXECUTE](#EXECUTE) | 用于执行PARSE函数处理的SQL语句。 |
| [FETCH\_ROWS](#FETCH_ROWS) | 用于载入结果集中的一行结果。 |
| [PARSE](#PARSE) | 解析当前语句的语法。 |
| [RETURN\_RESULT](#RETURN_RESULT) | 用于将已执行语句的结果返回到客户端应用程序。 |
| [BIND\_ARRAY](#BIND_ARRAY) | 该存储过程根据传入动态SQL进行table变量进行绑定作用。 |
| [BIND\_VARIABLE](#BIND_VARIABLE) | 该存储过程根据传入动态SQL进行变量的绑定。 |
| [DESCRIBE\_COLUMNS](#COLUMNS)  [DESCRIBE\_COLUMNS2](#COLUMNS)  [DESCRIBE\_COLUMNS3](#COLUMNS) | 该存储过程输出列的元数据信息。 |
| [DEFINE\_ARRAY](#DEFINE_ARRAY) | 定义要从给定游标中选择的集合，仅用于SELECT语句。 |
| [IS\_OPEN](#IS_OPEN) | 该函数用于判断游标是否打开。 |
| [FETCH\_ROWS](#FETCH_ROWS) | 该函数遍历DBMS\_SQL.EXECUTE执行动态SQL返回结果数据集，DEFINE\_COLUMN定义动态SQL返回的变量名、通过COLUMN\_VALUE存储过程定义返回列的值。 |

**注意事项**

无。

**兼容性**

Vastbase G100 V2.2版本与Oracle 11g R2版本相比，不支持的函数有BIND\_VARIABLE\_PKG、COLUMN\_VALUE\_LONG、DEFINE\_COLUMN\_CHAR、DEFINE\_COLUMN\_LONG、DEFINE\_COLUMN\_RAW、DEFINE\_COLUMN\_ROWID、EXECUTE\_AND\_FETCH、GET\_NEXT\_RESULT、LAST\_ERROR\_POSITION、LAST\_ROW\_ID、LAST\_SQL\_FUNCTION\_CODE、TO\_CURSOR\_NUMBER 、TO\_REFCURSOR、VARIABLE\_VALUE、VARIABLE\_VALUE\_PKG。

**示例**

1、创建测试表并插入数据。

create table dbms\_t1(id number,con varchar(20));

insert into dbms\_t1 values(1.5,'aaa'),(2.0,'bbb'),(3,'ccc');

2、创建存储过程调用函数。

CREATE OR REPLACE function dbms\_demo1() return text AS

cursor\_name INT;

rows\_processed INT;

ret INT;

rec text;

BEGIN

cursor\_name := DBMS\_SQL.open\_cursor();

DBMS\_SQL.PARSE(cursor\_name,'select con from public.dbms\_t1;',0);

ret := DBMS\_SQL.EXECUTE(cursor\_name);

loop

if DBMS\_SQL.FETCH\_ROWS(cursor\_name) > 0 then

select DBMS\_SQL.COLUMN\_VALUE(cursor\_name,1,rec) into rec;

else

exit;

end if;

end loop;

DBMS\_SQL.CLOSE\_CURSOR(cursor\_name);

return rec;

END;

/

3、调用存储过程dbms\_demo1()。

select dbms\_demo1();

当结果显示如下，则表示函数调用成功：

dbms\_demo1

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

ccc

(1 row)

4、创建测试表re\_t1，re\_t2并插入数据。

create table re\_t1(c1 int,c2 varchar(32));

insert into re\_t1 values(1,'111');

insert into re\_t1 values(2,'222');

create table re\_t2(c1 varchar(32),c2 int);

insert into re\_t2 values('111',1);

insert into re\_t2 values('222',2);

5、使用sys\_refcursor类型作为输入。

create or replace procedure re\_proc() as

rc1 sys\_refcursor;

rc2 sys\_refcursor;

begin

open rc1 for select \* from re\_t1;

dbms\_sql.return\_result(rc1);

open rc2 for select \* from re\_t2;

dbms\_sql.return\_result(rc2);

end;

/

6、调用存储过程re\_proc()。

call re\_proc();

当返回结果如下，则表示函数调用成功：

ResultSet #1

c1 | c2

----+-----

1 | 111

2 | 222

(2 rows)

ResultSet #2

c1 | c2

-----+----

111 | 1

222 | 2

(2 rows)

re\_proc

---------

(1 row)

7、使用integer类型作为输入参数。

create or replace procedure re\_proc\_int() as

id number;

id1 number:=dbms\_sql.open\_cursor();

id2 number:=dbms\_sql.open\_cursor();

begin

dbms\_sql.parse(id1,'select \* from public.re\_t1',0);

id:=dbms\_sql.execute(id1);

dbms\_sql.return\_result(id1);

dbms\_sql.parse(id2,'select \* from public.re\_t2',0);

id:=dbms\_sql.execute(id2);

dbms\_sql.return\_result(id2);

end;

/

8、调用存储过程re\_proc\_int()。

call re\_proc\_int();

当结果返回如下，则表示调用成功：

ResultSet #1

c1 | c2

----+-----

1 | 111

2 | 222

(2 rows)

ResultSet #2

c1 | c2

-----+----

111 | 1

222 | 2

(2 rows)

re\_proc\_int

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

(1 row)

**OPEN\_CURSOR**

**语法格式**

DBMS\_SQL.OPEN\_CURSOR (

c IN OUT INTEGER);

**参数说明**

c：需要打开的游标ID号。

**CLOSE\_CURSOR**

**语法格式**

DBMS\_SQL.CLOSE\_CURSOR (

c IN OUT INTEGER);

**参数说明**

c：需要关闭的游标ID数。

**COLUMN\_VALUE**

**语法格式**

DBMS\_SQL.COLUMN\_VALUE (

c IN INTEGER,

position IN INTEGER,

value OUT <datatype>

[,column\_error OUT NUMBER]

[,actual\_length OUT INTEGER]);

**参数说明**

* c：游标ID号。
* position：光标中列的相对位置、语句中的第一列位置为1。
* value：返回指定列的值。
* column\_error：返回指定列值的任何错误代码。
* actual\_length：指定列的值在截断之前的实际长度

**DEFINE\_COLUMN**

**语法格式**

DBMS\_SQL.DEFINE\_COLUMN (

c IN INTEGER,

position IN INTEGER,

column IN <datatype>);

DBMS\_SQL.DEFINE\_COLUMN (

c IN INTEGER,

position IN INTEGER,

column IN VARCHAR2 CHARACTER SET ANY\_CS,

column\_size IN INTEGER);

**参数说明**

* c：游标ID号。
* position：正在定义的行中列的相对位置，语句中第一列的位置为1。
* colum：列的值
* column\_size：列值的最大预期大小（以字节为单位）。

**EXECUTE**

**语法格式**

DBMS\_SQL.EXECUTE (

c IN INTEGER)

RETURN INTEGER;

**参数说明**

c：游标ID号。

**FETCH\_ROWS**

**语法格式**

DBMS\_SQL.FETCH\_ROWS (

c IN INTEGER)

RETURN INTEGER;

**参数说明**

c：ID号。

**PARSE**

**语法格式**

DBMS\_SQL.PARSE (

c IN INTEGER,

statement IN VARCHAR2A,

lb IN INTEGER,

ub IN INTEGER,

lfflg IN BOOLEAN,

language\_flag IN INTEGER[

[,edition IN VARCHAR2 DEFAULT NULL],

apply\_crossedition\_trigger IN VARCHAR2 DEFAULT NULL,

fire\_apply\_trigger IN BOOLEAN DEFAULT TRUE]

[,schema IN VARCHAR2 DEFAULT NULL]

[,container IN VARCHAR2)];

DBMS\_SQL.PARSE (

c IN INTEGER,

statement IN VARCHAR2s,

lb IN INTEGER,

ub IN INTEGER,

lfflg IN BOOLEAN,

language\_flag IN INTEGER[

[,edition IN VARCHAR2 DEFAULT NULL],

apply\_crossedition\_trigger IN VARCHAR2 DEFAULT NULL,

fire\_apply\_trigger IN BOOLEAN DEFAULT TRUE]

[,schema IN VARCHAR2 DEFAULT NULL]

[,container IN VARCHAR2)];

**参数说明**

* c：要解析的语句的游标ID号。
* statement：要解析的SQL语句。
* lb：语句元素的下限。
* ub：语句中元素的上限。
* lfflg：如果设置为TRUN，则在连接的每个元素之后插入换行符。
* language\_flag：指定 SQL 语句的行为。
* edition：指定在以下条件下运行语句的版本：
* 如果是NULL该语句将在当前版本中运行。
* 如果指定了有效容器，则通过NULL表示该语句将在目标容器的默认版本中运行。
* 给定用户和执行语句的版本，用户必须USE对该版本具有特权。
* 以下一般条件适用。字符串的内容作为 SQL 标识符处理；如果版本的实际名称中存在特殊字符或小写字符，则必须用双引号将字符串的其余部分括起来，如果不使用双引号，则内容将大写。
* apply\_crossedition\_trigger：指定要应用于指定 SQL 的正向交叉版本触发器的非限定名称。
* fire\_apply\_trigger：指示apply\_crossedition\_trigger的内容本身是要执行的，还是只能作为用于选择其他触发器的指南。
* schema：指定解析不合格对象名称的架构。如果NULL，则当前架构是有效用户的架构。
* container：要在其中运行的目标容器的名称。

**注意事项**

* dbms\_sql.parse存储过程中，支持常量dbms\_sql.native、dbms\_sql.v7、dbms\_sql.v6参数。

**示例**

parse函数-标志位支持dbms\_sql、dbms\_sql.v6、dbms\_sql.v7三种类型

1、创建测试表。

drop table t\_dbms\_sql cascade;

CREATE TABLE t\_dbms\_sql(c1 int,c2 varchar(200));

2、调用parse函数。

SET SERVEROUTPUT ON;

DECLARE

c integer;

test\_sql varchar2(200);

BEGIN

test\_sql := 'INSERT INTO t\_dbms\_sql VALUES(1,''zhangsan'')';

c := dbms\_sql.open\_cursor;

dbms\_sql.parse(c,test\_sql,dbms\_sql.native);

dbms\_sql.parse(c,test\_sql,dbms\_sql.v6);

dbms\_sql.parse(c,test\_sql,dbms\_sql.v7);

dbms\_sql.parse(c,test\_sql,0);

dbms\_sql.parse(c,test\_sql,1);

dbms\_sql.parse(c,test\_sql,2);

END;

/

**RETURN\_RESULT**

**语法格式**

DBMS\_SQL.RETURN\_RESULT(

rc IN OUT SYS\_REFCURSOR,

to\_client IN BOOLEAN DEFAULT TRUE);

DBMS\_SQL.RETURN\_RESULT(

rc IN OUT INTEGER,

to\_client IN BOOLEAN DEFAULT TRUE);

**参数说明**

* rc：语句游标或引用游标。
* to\_client：将语句结果返回（或不返回）给客户端。如果是FALSE，则将其返回给直接调用者。

**BIND\_ARRAY**

**语法格式**

DBMS\_SQL.BIND\_ARRAY (

c IN INTEGER,

name IN VARCHAR2,

<table\_variable> IN <datatype>);

**参数说明**

* c : 代表执行dbms\_sql.open\_cursor返回的变量
* name：需要绑定的变量名
* <table\_variable>：在dbms\_sql中定义的table类型的数据结构

**示例**

将table类型数据绑定到变量中赋值（number\_table、varchar2\_table、date\_table、blob\_table、clob\_table、binary\_double\_table、binary\_float\_table）

1、创建测试表。

DROP TABLE t\_dbms\_sql CASCADE;

CREATE TABLE t\_dbms\_sql(

c\_number number,

c\_varchar2 varchar2(1000),

c\_date date,

c\_clob clob,

c\_binary\_float binary\_float,

c\_binary\_double binary\_double,

c\_blob blob);

2、调用函数。

SET SERVEROUTPUT ON;

CREATE OR REPLACE FUNCTION func\_dbms\_sql() RETURNS VOID AS

$$

DECLARE

--创建table类型参数

array\_number dbms\_sql.number\_table;

array\_varchar2 dbms\_sql.varchar2\_table;

array\_date dbms\_sql.date\_table;

array\_blob dbms\_sql.varchar2\_table;

array\_clob dbms\_sql.clob\_table;

array\_binary\_double dbms\_sql.binary\_double\_table;

array\_binary\_float dbms\_sql.binary\_float\_table;

c integer;

test\_sql varchar2(200);

e number;

BEGIN

--为number类型嵌套表赋值数不一致

array\_number(1) := 1;

array\_number(2) := 2.2;

array\_number(3) := 33.3;

array\_number(4) := 4;

array\_number(5) := 5.5;

array\_number(6) := 66.6;

--为varchar2类型嵌套表赋值

array\_varchar2(1) := 'test1';

array\_varchar2(2) := 'test2';

array\_varchar2(3) := 'test3';

array\_varchar2(4) := 'test4';

array\_varchar2(5) := 'test5';

array\_varchar2(6) := 'test6';

--为date类型嵌套表赋值

array\_date(1) := '2011-01-01 11:01:01';

array\_date(2) := '2012-02-02 12:02:02';

array\_date(3) := '2013-03-03 13:03:03';

array\_date(4) := '2014-04-04 14:04:04';

array\_date(5) := '2015-05-05 15:05:05';

array\_date(6) := '2016-06-06 16:06:06';

--为blob类型嵌套表赋值

array\_blob(1) := '10101010';

array\_blob(2) := '101010101';

array\_blob(3) := '1010101010';

array\_blob(4) := '10101010101';

array\_blob(5) := '101010101010';

array\_blob(6) := '1010101010101';

--为clob类型嵌套表赋值

array\_clob(1) := 'clob1';

array\_clob(2) := 'clob2';

array\_clob(3) := 'clob3';

array\_clob(4) := 'clob4';

array\_clob(5) := 'clob5';

array\_clob(6) := 'clob6';

--为binary\_float类型嵌套表赋值

array\_binary\_float(1) := '1.111';

array\_binary\_float(2) := '22.2';

array\_binary\_float(3) := '3.333333';

array\_binary\_float(4) := '44.4';

array\_binary\_float(5) := '5.555555';

array\_binary\_float(6) := '66666.66666';

--为binary\_double类型嵌套表赋值

array\_binary\_double(1) := '1.111';

array\_binary\_double(2) := '22.2';

array\_binary\_double(3) := '3.333333';

array\_binary\_double(4) := '44.4';

array\_binary\_double(5) := '5.555555';

array\_binary\_double(6) := '66666.66666';

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

test\_sql := 'INSERT INTO t\_dbms\_sql VALUES(:c\_number,:c\_varchar2,:c\_date,:c\_clob,:c\_binary\_float,:c\_binary\_double,:c\_blob)';

dbms\_sql.parse(c,test\_sql,dbms\_sql.native);

--绑定变量

dbms\_sql.bind\_array(c,':c\_number',array\_number);

dbms\_sql.bind\_array(c,':c\_varchar2',array\_varchar2);

dbms\_sql.bind\_array(c,':c\_date',array\_date);

dbms\_sql.bind\_array(c,':c\_blob',array\_blob);

dbms\_sql.bind\_array(c,':c\_clob',array\_clob);

dbms\_sql.bind\_array(c,':c\_binary\_float',array\_binary\_float);

dbms\_sql.bind\_array(c,':c\_binary\_double',array\_binary\_double);

e := dbms\_sql.execute(c);

DBMS\_SQL.CLOSE\_CURSOR(c);

END;

$$

LANGUAGE plpgsql;

select func\_dbms\_sql();

3、查询表中的数据。

select \* from t\_dbms\_sql;

**BIND\_VARIABLE**

**语法格式**

DBMS\_SQL.BIND\_VARIABLE (

c IN INTEGER,

name IN VARCHAR2,

value IN <datatype>);

**参数说明**

* c : 代表执行dbms\_sql.open\_cursor返回的变量。
* name：需要绑定的变量名。
* <value>：常见的varchar2、number、date等数据类型值。

**示例**

**bind\_variable函数-将常量类型赋值给数值类型字段**

1、创建测试表。

DROP TABLE t\_dbms\_sql CASCADE;

CREATE TABLE t\_dbms\_sql(

c\_int2 int2,

c\_int4 int4,

c\_numeric numeric,

c\_number number,

c\_float4 float4,

c\_float8 float8,

c\_real real);

2、将常量类型赋值给数值类型字段。

SET SERVEROUTPUT ON;

--value参数赋值为常量类型

DECLARE

c number;

c\_number number;

test\_sql varchar2(100);

e number;

BEGIN

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

test\_sql := 'INSERT INTO t\_dbms\_sql VALUES(:c\_int2,:c\_int4,:c\_numeric,:c\_number,:c\_float4,:c\_float8,:c\_real)';

dbms\_sql.parse(c,test\_sql,dbms\_sql.native);

--绑定变量

dbms\_sql.bind\_variable(c,':c\_int2',dbms\_sql.native);

dbms\_sql.bind\_variable(c,':c\_int4',dbms\_sql.v6);

dbms\_sql.bind\_variable(c,':c\_numeric',dbms\_sql.v7);

dbms\_sql.bind\_variable(c,':c\_number',dbms\_sql.native);

dbms\_sql.bind\_variable(c,':c\_float4',dbms\_sql.v6);

dbms\_sql.bind\_variable(c,':c\_float8',dbms\_sql.v7);

dbms\_sql.bind\_variable(c,':c\_real',dbms\_sql.native);

e := dbms\_sql.execute(c);

dbms\_sql.close\_cursor(c);

END;

/

3、查询表中数据。

select \* from t\_dbms\_sql;

返回结果为：

c\_int2 | c\_int4 | c\_numeric | c\_number | c\_float4 | c\_float8 | c\_real

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

1 | 0 | 2 | B096.- | 0 | 2 | 1

(1 row)

**DESCRIBE\_COLUMNS，DESCRIBE\_COLUMNS2，DESCRIBE\_COLUMNS3**

**语法格式**

DBMS\_SQL.DESCRIBE\_COLUMNS (

c IN INTEGER,

col\_cnt OUT INTEGER,

desc\_t OUT DESC\_TAB);

DBMS\_SQL.DESCRIBE\_COLUMNS2 (

c IN INTEGER,

col\_cnt OUT INTEGER,

desc\_t OUT DESC\_TAB2);

DBMS\_SQL.DESCRIBE\_COLUMNS3 (

c IN INTEGER,

col\_cnt OUT INTEGER,

desc\_t OUT DESC\_TAB3);

DBMS\_SQL.DESCRIBE\_COLUMNS3 (

c IN INTEGER,

col\_cnt OUT INTEGER,

desc\_t OUT DESC\_TAB4);

**参数说明**

* c : 代表执行dbms\_sql.open\_cursor返回的变量。
* name：查询语句返回的列数量。
* desc\_t：输出表的元数据信息，参考desc\_tab的table变量。

**DEFINE\_ARRAY**

**语法格式**

DEFINE\_ARRAY (

c IN INTEGER,

position IN INTEGER,

<table\_variable> IN <datatype>

cnt IN INTEGER,

lower\_bnd IN INTEGER);

**参数说明**

* c：已经打开的cursor名称。
* position：指需要返回的字段在查询结果中处于第几列。
* <table\_variable>：接收返回结果需要的变量,与define\_column不同的是此变量是table,而不是普通的字段类型。
* cnt：表示一次可以返回的行数。
* lower\_bnd：指n\_tab的index从哪个数值开始,此数值是递增的.在此例中index是从1开始的，一次得到9行结果集，则有n\_tab(1)到n\_tab(9),如果循环再得到新的结果集,则index继续增长。

**IS\_OPEN**

**语法格式**

DBMS\_SQL.IS\_OPEN (c IN INTEGER) RETURN BOOLEAN;

**参数说明**

c : 代表执行dbms\_sql.open\_cursor返回的变量。

**示例**

**判断游标是否已打开，已打开判断为true，未打开判断为false**

1、定义存储过程。

SET SERVEROUTPUT ON;

CREATE OR REPLACE PROCEDURE proc\_dbms\_sql(i integer) as

DECLARE

c integer;

BEGIN

IF i<3 THEN

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

raise notice '游标状态 is 打开 %',dbms\_sql.is\_open(c) ;

ELSE

raise notice '游标状态 is 关闭 %',dbms\_sql.is\_open(c) ;

END IF;

END;

/

2、调用is\_open函数。

DECLARE

c integer;

BEGIN

--打开游标，判断为true

PERFORM proc\_dbms\_sql(2);

--未打开游标，判断为flase

PERFORM proc\_dbms\_sql(3);

END;

/

返回结果为：

NOTICE: 游标状态 is 打开 t

CONTEXT: referenced column: proc\_dbms\_sql

SQL statement "SELECT proc\_dbms\_sql(2)"

PL/pgSQL function inline\_code\_block line 5 at PERFORM

NOTICE: 游标状态 is 关闭 f

CONTEXT: referenced column: proc\_dbms\_sql

SQL statement "SELECT proc\_dbms\_sql(3)"

PL/pgSQL function inline\_code\_block line 7 at PERFORM

ANONYMOUS BLOCK EXECUTE

## DBMS\_STATS

**功能描述**

DBMS\_STATS包用于查看和修改收集的所有数据库对象的统计信息，以便能估量统计数据（尤其是针对较大的分区表），从而制定出速度更快的SQL执行计划。该内置包中包含以下函数：

* GATHER\_SCHEMA\_STATS函数：收集模式中所有对象的统计信息。

**注意事项**

无。

**兼容性**

Vastbase G100 V2.2版本与Oracle 11g R2版本相比，仅支持GATHER\_SCHEMA\_STATS函数。

**GATHER\_SCHEMA\_STATS函数**

**功能描述**

该函数用于收集指定模式下的所有对象的统计信息。

**语法格式**

* 第一种：

DBMS\_STATS.GATHER\_SCHEMA\_STATS(

ownname VARCHAR2，

estimate\_percent NUMBER DEFAULT 100,

block\_sample BOOLEAN DEFAULT FALSE,

method\_opt VARCHAR2 default 'FOR ALL COLUMNS SIZE AUTO',

degree NUMBER DEFAULT NULL,

granularity VARCHAR2 DEFAULT 'GLOBAL'

cascade BOOLEAN DEFAULT FALSE,

stattab VARCHAR2 DEFAULT NULL,

statid VARCHAR2 DEFAULT NULL,

options VARCHAR2 DEFAULT 'GATHER',

objlist OUT ObjectTab,

statown VARCHAR2 DEFAULT NULL,

no\_invalidate BOOLEAN DEFAULT FALSE,

force BOOLEAN DEFAULT FALSE,

obj\_filter list ( ObjectTab DEFAULT NULL);

* 第二种：

DBMS\_STATS.GATHER\_SCHEMA\_STATS(

ownname VARCHAR2，

estimate\_percent NUMBER DEFAULT 100,

block\_sample BOOLEAN DEFAULT FALSE,

method\_opt VARCHAR2 default 'FOR ALL COLUMNS SIZE AUTO',

degree NUMBER DEFAULT NULL,

granularity VARCHAR2 DEFAULT 'GLOBAL'

cascade BOOLEAN DEFAULT FALSE,

stattab VARCHAR2 DEFAULT NULL,

statid VARCHAR2 DEFAULT NULL,

options VARCHAR2 DEFAULT 'GATHER',

statown VARCHAR2 DEFAULT NULL,

no\_invalidate BOOLEAN DEFAULT FALSE,

force BOOLEAN DEFAULT FALSE,

obj\_filter list ( ObjectTab DEFAULT NULL);

**参数说明**

* ownname：要分析的schema（NULL表示当前schema）。
* estimate\_percent: 确定要采样的行的百分比，有效范围在0.000001~100。
* block\_sample：表示是否使用随机块采样代替随机行采样，默认为false。注意：Vastbase 目前无需支持，可忽略该参数。
* method\_opt：默认值为：FOR ALL COLUMNS SIZE AUTO（仅支持该选项，其
* 他选项不支持）。该选项可接受的语法为：

FOR ALL [INDEXED|HIDDEN] COLUMNS [size clause]

* size clause := SIZE {integer| REPEAT| AUTO |SKEWONLY}
* Integer：直方图数。必须在1~2048范围内。
* REPEAT：仅在已经具有直方名的列上收集直方名。
* AUTO：数据库根据数据分布和列的工作量确定要收集直方图的列。
* SKEWONLY：数据库根据列的数据分布来确定要收集直方图的列。
* degree：并行度，默认degree值为 NULL，也可以使用常量 AUTO DEGREE设
* 置自动并行度（Vastbase 仅支持 NULL，其他选项不支持）。
* granularity：要收集的统计信息的粒度(仅在表已分区时才适用)。取值可以是：'GLOBAL'，'GLOBAL AND PARTITION’，'PARTITION'
* 'GLOBAL'：收集全局统计数据。
* 'GLOBAL AND PARTITION’：收集全局和分区级别的统计数据。
* 'PARTITION'：收集分区级别的统计信息。
* cascade：收集有关索引的统计信息。除了收集表和列统计信息之外，使用此选项
* 等效于在模式中的每个索引上运行 GATHER\_INDEX\_STATS 过程，使用该常量 DBMS\_STATS.AUTO\_CASCADE 让数据库确定是否要收集索引统计信息。这是默认值（支持该参数传入，但不校验并忽略该参数）。
* stattab：用户统计信息表标识符，用于描述将当前统计信息保存在何处。
* statid：标识符。支持该参数传入，但不校验并忽略该参数。
* options：指定需要收集统计信息的对象。有效值如下：Vastbase 仅支持默认值 GATHER（收集 schema 中所有对象的统计信息），其他选项不支持。
* objlist：过旧或空的对象列表（支持该参数传入，但不校验并忽略该参数）。
* statown： 包含的 schema（支持该参数传入，但不校验并忽略该参数）。
* no\_invalidate：是否使相关游标无效，设置为 false 表示使得相关游标无效:设置为 true 则相反（支持该参数传入，但不校验并忽略该参数）。
* force：收集有关对象的统计信息，被锁定的对象也会被统计（支持该参数传入，但不校验并忽略该参数）。
* obj\_filter list：设置对象过滤器列表（支持该参数传入，但不校验并忽略该参数）。

**兼容性**

1、granularity参数只兼容部分选项，暂不支持的选项如下（与oracle的差异）：

* 'ALL'：收集所有(子分区，分区和全局)统计信息。
* 'AUTO'：根据分区类型确定粒度，这是默认值。
* 'DEFAULT'：收集全局和分区级别的统计信息。已过时，为向下兼容保留。
* 'SUBPARTITION'：收集子分区级别的统计信息。

2、GATHER参数目前Vastbase G100 Build 9暂不支持的选项如下（与oracle的差异）：

* GATHER AUTO ：自动收集所有必要的统计信息。
* GATHER STALE：通过查询\*TAB MODIFICATIONS 视图来收集陈旧对象的统计信息。同样，数据库返回发现已过时的对象的列表。
* GATHER EMPTV：收集有关当前没有统计信息的对象的统计信息。同样。数据库返回发现没有统计信息的对象列表。
* LIST AUTO：返回要使用处理的对象的列表 GATHER AUTO。
* LIST STALE：返回通过查看\* TAB MODIFICATIONS 视名确定的陈旧对象的列表。
* LIST EMPTY：返回当前没有统计信息的对象列表。

**示例**

1、创建模式。

create schema zmm;

2、创建分区表。

create table zmm.customer\_address

(

ca\_address\_sk integer not null ,

ca\_address\_id character(16)

)

partition by range (ca\_address\_sk)

(

partition p1 values less than(500),

partition p2 values less than(1000),

partition p3 values less than(1500),

partition p4 values less than(2000),

partition p5 values less than(2500),

partition p6 values less than(3000),

partition p7 values less than(4000),

partition p8 values less than(maxvalue)

)

enable row movement;

3、插入数据。

insert into zmm.customer\_address(ca\_address\_sk) values(generate\_series(1,5000));

4、调用dbms\_stats.gather\_schema\_stats函数收集统计信息。

exec dbms\_stats.gather\_schema\_stats(ownname=>'zmm',estimate\_percent=>100,block\_sample=>false,method\_opt=>'for all columns size auto',degree=>DBMS\_STATS.AUTO\_DEGREE,granularity=>'GLOBAL',cascade=>DBMS\_STATS.AUTO\_CASCADE,stattab=>'',statid=>'',options=>'GATHER AUTO',no\_invalidate=>DBMS\_STATS.AUTO\_INVALIDATE,force=>false,statown=>'');

返回结果如下，则表示函数调用成功：

NOTICE: PL/SQL procedure successfully completed.

gather\_schema\_stats

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

(1 row)

## DBMS\_UTILITY

**功能描述**

DBMS\_UTILITY包提供了很多不同的子程序工具。该内置包包含以下函数：

|  |  |
| --- | --- |
| **函数** | **描述** |
| [COMMA\_TO\_TABLE](#UTLCOMMA_TO_TABLE) | 用于将逗号间隔的字符串转化成数组类型。 |
| [TABLE\_TO\_COMMA](#UTLTABLE_TO_COMMA) | 用于将数组类型转换成逗号间隔的字符串。 |
| [EXEC\_DDL\_STATEMENT](#UTLEXEC_DDL_STATEMENT) | 用于执行DDL语句。 |
| [FORMAT\_CALL\_STACK](#UTLFORMAT_CALL_STACK) | 用于格式化调用堆栈。该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当前函数调用处的所有过程或者函数的调用顺序。该函数可在存储过程、函数或包中调用，并且以可读格式返回当前调用堆栈。 |
| [GET\_HASH\_VALUE](#UTLGET_HASH_VALUE) | 用于获得在该范围内字符串的hash值。 |
| [GET\_TIME](#UTLGET_TIME) | 用于记录当前时刻的时间（单位是百分之一秒）。 |

**注意事项**

LNAME\_ARRAY、UNCL\_ARRAY、NAME\_ARRAY类型用于配合函数comma\_to\_table、table\_to\_comma的出入参。它们对应的类型分别为：

* LANAME\_ARRY：由bigint作为索引的varchar（4000）[]或者varchar2（4000）[]的数组。
* NAME\_ARRY：由bigint作为索引的varchar（30）[]或者varchar2（30）[]的数组。
* UNCL\_ARRY：由bigint作为索引的varchar（227）[]或者varchar2（227）[]的数组。

**兼容性**

Vastbase G100 V2.2版本与Oracle 11g R2版本相比，不支持的函数有：CANONICALIZE、COMPILE\_SCHEMA、CREATE\_ALTER\_TYPE\_ERROR\_TABLE、CURRENT\_INSTANCE Function、DATA\_BLOCK\_ADDRESS\_BLOCK、DATA\_BLOCK\_ADDRESS\_FILE、DB\_VERSION、EXPAND\_SQL\_TEXT、FORMAT\_ERROR\_BACKTRACE、FORMAT\_ERROR\_STACK、GET\_CPU\_TIME、GET\_DEPENDENCY、GET\_ENDIANNESS、GET\_DEPENDENCY、GET\_PARAMETER\_VALUE、GET\_SQL\_HASH、GET\_TZ\_TRANSITIONS、INVALIDATE、IS\_BIT\_SET、IS\_CLUSTER\_DATABASE、MAKE\_DATA\_BLOCK\_ADDRESS、NAME\_RESOLVE、NAME\_TOKENIZE、OLD\_CURRENT\_SCHEMA、OLD\_CURRENT\_USER、PORT\_STRING、SQLID\_TO\_SQLHASH、VALIDATE、WAIT\_ON\_PENDING\_DML。

**示例**

* 创建并切换至兼容模式为Oracle的数据库db\_oracle。

CREATE DATABASE db\_oracle dbcompatibility='A';

\c db\_oracle

* format\_call\_stack函数exec\_ddl\_statement函数

CREATE OR REPLACE FUNCTION checkHexCallStack() returns text

as $$

DECLARE

stack text;

BEGIN

select \* INTO stack from dbms\_utility.format\_call\_stack('o');

return stack;

END;

$$ LANGUAGE plpgsql;

begin

Dbms\_utility.exec\_ddl\_statement('create table test\_eds(col int);');

end;

/

查询test\_eds表，返回结果如下，则表示函数调用成功：

\dti test\_eds

List of relations

Schema | Name | Type | Owner | Table | Storage

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

public | test\_eds | table | vastbase | | {orientation=row,compression=no,fillfactor=80}

(1 row)

* table\_to\_comma函数和comma\_to\_table函数
* 设置serveroutput为true。

select dbms\_output.serveroutput(true);

* 使用存储过程调用函数。

declare

t\_vararray dbms\_utility.lname\_array;

vc\_stringlist varchar2(4000);

n\_idx binary\_integer;

begin

vc\_stringlist := 'lname,ARRAY';

dbms\_utility.comma\_to\_table(vc\_stringlist, n\_idx, t\_vararray);

dbms\_output.put\_line('Total Num : '||to\_char(n\_idx));

for i in 1..n\_idx loop

dbms\_output.put\_line(t\_vararray[i]);

t\_vararray(i) := '['||t\_vararray[i]||']';

dbms\_output.put\_line('<<'||t\_vararray[i]);

end loop;

n\_idx := n\_idx - 1;

dbms\_utility.table\_to\_comma(t\_vararray, n\_idx, vc\_stringlist);

dbms\_output.put\_line('>>>>>');

dbms\_output.put\_line(''||vc\_stringlist);

end;

/

返回结果如下，则表示函数调用成功：

Total Num : 2

lname

<<[lname]

ARRAY

<<[ARRAY]

>>>>>

[lname],[ARRAY]

* get\_time函数

CREATE OR REPLACE FUNCTION get\_runtime() returns number as $$

DECLARE

start\_time NUMBER;

finish\_time NUMBER;

elapsed\_time NUMBER;

BEGIN

start\_time := DBMS\_UTILITY.get\_time();

perform pg\_sleep(2);

finish\_time := DBMS\_UTILITY.get\_time();

elapsed\_time := finish\_time - start\_time;

return elapsed\_time;

END

$$ language plpgsql;

调用get\_runtime函数，返回结果如下，则表示调用成功：

call get\_runtime();

get\_runtime

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

200

(1 row)

* get\_hash\_value函数

select dbms\_utility.get\_hash\_value('1242',0,1000);

返回结果如下，则表示函数调用成功：

get\_hash\_value

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

984

(1 row)

**COMMA\_TO\_TABLE函数**

**语法格式**

DBMS\_UTILITY.COMMA\_TO\_TABLE(

list in VARCHAR|VARCHAR2,

tablen OUT BIGINT,

tab OUT varchar[]);

**参数说明**

* list：逗号间隔的字符串。
* Tablen：数组类型的数量。
* Tab：数组类型的数值。

**注意事项**

* list必须是费控的逗号间隔符类型。字符串里面双引号中的逗号不极端在内。
* 逗号分割列表中的条目不能包含多字节字符，例如连字符（-）。
* tab里的值应为原始列表中剪裁出来的值，不能做任何转换。

**TABLE\_TO\_COMMA函数**

**语法格式**

DBMS\_UTILITY.TABLE\_TO\_COMMA(

tab in varchar[],

tablen OUT BIGINT,

list OUT VARCHAR|VARCHAR2);

**参数说明**

* list：逗号间隔的字符串。
* Tablen：数组类型的数量。
* Tab：数组类型的数值。

**GET\_TIME函数**

**语法格式**

DBMS\_UTILITY.GET\_TIME

return number;

**注意事项**

* 如果需要换算为秒，需要除以100。
* 该函数没有传入参数，返回值为number类型。

**GET\_HASH\_VALUE函数**

**语法格式**

dbms\_utility.get\_hash\_value(

hash\_name varchar,

hash\_base integer,

hash\_size integer);

**参数说明**

* hash\_name：需要被转化为hash值的字符串。
* hash\_base：hash起始位置。
* hash\_size：hash大小，不能为0，若为正值表示从hash起始位置到加上hash
* 大小的范围，为负值表示从integer范围排除从hash起始位置减去hash大小绝对值的范围。

**EXEC\_DDL\_STATEMENT 函数**

**语法格式**

DBMS\_UTILITY.EXEC\_DDL\_STATEMENT (

parse\_string IN VARCHAR2);

**参数说明**

parse\_string：要被执行的DDL语句。

**FORMAT\_CALL\_STACK函数**

**语法格式**

DBMS\_UTILITY.FORMAT\_CALL\_STACK

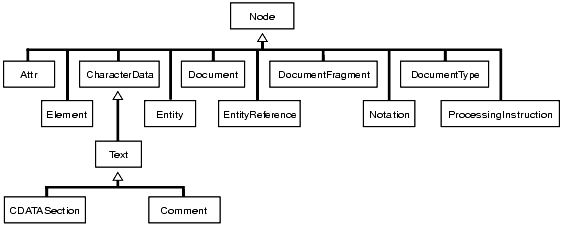
RETURN VARCHAR2;

## DBMS\_XMLDOM

**功能描述**

dbms\_xmldom包用于访问xmltype对象，并实现文档对象模型（DOM）,是用于xml和HTML文档的应用程序编程接口（API）。

DOM把xml文档当成树结构来查看，通过DOM树能访问、修改、删除文档中所有元素以及创建新的元素并插入树中，这些元素、文本以及它们的属性都被认为是节点，这些节点之间是一个继承层次结构。继承层次结构如下图所示：



**节点类型**

* DOMNode类型：即Node节点。代表xml文档树中一个单独的节点，可以是其他任何一种节点。
* DOMAttr类型：即Attr节点。表示Element节点中的属性。
* DOMElement类型：即Element节点，代表xml文档中的一个元素。元素可以包含属性，其他元素或文本。
* DOMText类型：即Text节点。表示元素或属性的文本内容。
* DOMCDATASection类型：即CDATASection节点。表示xml文档中的CDATA区段，CDATA区段时一段不会被解析器解析的文本。
* DOMComment类型：即Comment节点。表示xml文档中注释节点的内容。
* DOMEntity类型：即Entity节点。在xml文档中频繁使用某一条数据时，可以预定义一个这条数据的“别名”，即一个Entity，然后在文档中需要的地方进行调用。
* DOMDocument类型：即Documnet节点。代表整个xml文档，是文档树的根，并提供了对文档数据访问的顶层入口。
* DOMDocumentFragment类型：即DocumentFragment节点。文档中的一部分，表示一个或多个邻接的Document节点和它们的所有子孙节点。DocumentFragment节点不属于文档树。
* DOMNotation类型：即Notation元素。Notation元素描述xml文档中非xml数据的格式。
* DOMDocumentType类型：即DocumentType节点。每个xml文档均有一个DOCTYPE属性，此属性的值可谓空，也可以是一个DocumentType对象。DocumentType对象为xml定义的实体提供接口。
* DOMProcessingInstruction类型：即ProcessingInstruction节点。表示xml文档中的一个处理指令。
* DOMText类型：即DOMText节点。XML文档中的文本。

该内置包包含以下函数：

| **函数** | **描述** |
| --- | --- |
| [dbms\_xmldom.makeNode](#DOMmakeNode) | 将其他节点类型转换成DOMNode类型，以便调用DOMNode的方法。 |
| [dbms\_xmldom.isNull](#DOMisNull) | 检查输入的对象是否为空，为空则返回true，否则返回false。 |
| [dbms\_xmldom.freeNode](#DOMfreeNode) | 释放与DOMNode关联的所有资源。 |
| [dbms\_xmldom.freeNodeList](#DOMfreeNodeList) | 释放与节点列表关联的所有资源。 |
| [dbms\_xmldom.freeDocument](#DOMfreeDocument) | 释放一个DOMDocument对象的所有资源。 |
| [dbms\_xmldom.getFirstChild](#DOMgetFirstChild) | 返回此节点的第一个子节点，如果没有则返回NULL。 |
| [dbms\_xmldom.getLocalName](#DOMgetLocalName) | 返回节点名称的本地部分。 |
| [dbms\_xmldom.getNodeType](#DOMgetNodeType) | 返回节点类型。 |
| [dbms\_xmldom.writeToClob](#DOMwriteToClob) | 使用数据库字符集将xml节点或xml文档写入指定的clob。 |
| [dbms\_xmldom.writeToBuffer](#DOMwriteToBuffer) | 使用数据库字符集将xml节点或xml文档写入指定的缓冲区。 |
| [dbms\_xmldom.getChildNodes](#DOMgetChildNodes) | 返回当前节点的所有子节点。 |
| [dbms\_xmldom.getLength](#DOMgetLength) | 返回当前对象的长度。 |
| [dbms\_xmldom.item](#DOMitem) | 返回节点列表或DOMNamedNodeMap无序列表中index参数对应的项。如果index大于或等于列表中的节点数，则返回空。 |
| [dbms\_xmldom.makeElement](#DOMmakeElement) | 将DOMNode节点转换成DOMElement节点类型。该函数主要用于将DOMElement节点转换成的DOMNode转换回DOMElement，因此除了节点类型为ELEMENT\_NODE的节点，其他节点类型不能调用该函数。 |
| [dbms\_xmldom.getElementsByTagName](#DOMgetElementsByTagName) | 返回包含指定名称的所有DOMElement的DOMNodeList。 |
| [dbms\_xmldom.cloneNode](#DOMcloneNode) | 返回此节点的副本，并用作节点的通用复制构造函数。拷贝的节点没有父节点，父节点为空。 |
| [dbms\_xmldom.getNodeName](#DOMgetNodeName) | 返回节点的节点名称。 |
| [dbms\_xmldom.createDocument](#DOMcreateDocument) | 通过制定的命名空间URI、根元素名和doctype创建一个xml文档。 |
| [dbms\_xmldom.createElement](#DOMcreateElement) | 用于创建一个DOMElement节点。 |
| [dbms\_xmldom.createDocumentFragment](#DOMcreateDocumentFragment) | 用于创建一个DOMDocumentFragment节点。 |
| [dbms\_xmldom.createTextNode](#DOMcreateTextNode) | 用于创建一个DOMText节点。 |
| [dbms\_xmldom.createComment](#DOMcreateComment) | 用于创建一个DOMComment节点。 |
| [dbms\_xmldom.createCDATASection](#DOMcreateCDATASection) | 用于创建一个DOMCDATASection节点。 |
| [dbms\_xmldom.createProcessingInstruction](#DOMcreateProcessingInstruction) | 用于创建一个DOMProcessingInstruction节点。 |
| [dbms\_xmldom.createAttribute](#DOMcreateAttribute) | 用于创建一个DOMAttr节点。 |
| [dbms\_xmldom.appendChild](#DOMappendChild) | 用于将节点newchild添加到该节点的子节点列表的末尾，并返回新添加的节点。如果newchild已经在树中，则先删除。 |
| [dbms\_xmldom.getDocumentElement](#DOMgetDocumentElement) | 用于返回xml文档的根元素节点。 |
| [dbms\_xmldom.setAttribute](#DOMsetAttribute) | 用于通过名称设置DOMElement属性的值。 |
| [dbms\_xmldom.setAttributeNode](#DOMsetAttributeNode) | 用于向DOMElement添加一个新的属性节点。 |
| [dbms\_xmldom.getAttributes](#DOMgetAttributes) | 用于检索包含此节点属性的NamedNodeMap，只有DOMElement包含该属性，其他节点则返回空。 |
| [dbms\_xmldom.getNodeValue](#DOMgetNodeValue) | 根据节点的类型获取该节点的值。 |
| [dbms\_xmldom.getNodeValueAsClob](#DOMgetNodeValueAsClob) | 根据节点的类型获取该节点的值并写入clob中。 |
| [dbms\_xmldom.getChildrenByTagName](#DOMgetChildrenByTagName) | 用于根据指定的名称返回DOMELement的子节点。 |
| [dbms\_xmldom.getOwnerDocument](#DOMgetOwnerDocument) | 用于检索与此节点关联的Document对象。 |
| [dbms\_xmldom.newDOMDocument](#DOMnewDOMDocument) | 返回一个新DOMDocument实例。 |
| [dbms\_xmldom.hasChildNodes](#DOMhasChildNodes) | 数返回该节点是否有子节点。 |
| [dbms\_xmldom.insertBefore](#DOMinsertBefore) | 用于将节点newchild插入到现有的子节点refchild之前。 |
| [dbms\_xmldom.setVersion](#DOMsetVersion) | 用于设置xml文档的版本号。 |

**注意事项**

* 该功能仅在数据库兼容模式为Oracle时能够使用（即创建DB时DBCOMPATIBILITY='A'），在其他数据库兼容模式下不能使用该特性。
* 系统已经安装python3。
* 创建plpython3u插件，需设置环境变量PYTHONHOME，包括以下内容（重启数据库生效）：
* PYTHONHOME：python3安装路径。
* LD\_LIBRARY\_PATH：增加python3安装路径下的lib目录。
* PATH：增加python3安装路径bin目录。

**dbms\_xmldom.makeNode**

**语法格式**

makeNode(t dbms\_xmldom.DOMText)

makeNode(com dbms\_xmldom.DOMComment)

makeNode(cds dbms\_xmldom.DOMCDATASection)

makeNode(dt dbms\_xmldom.DOMDocumentType)

makeNode(n dbms\_xmldom.DOMNotation)

makeNode(ent dbms\_xmldom.DOMEntity)

makeNode(pi dbms\_xmldom.DOMProcessingInstruction)

makeNode(df dbms\_xmldom.DOMDocumentFragment)

makeNode(doc dbms\_xmldom.DOMDocument)

makeNode(elem dbms\_xmldom.DOMElement)

**参数说明**

* t：待转换的DOMText对象
* com：待转换的DOMComment对象
* cds：待转换的DOMCDATASection对象
* dt：待转换的DOMDocumentType对象
* n：待转换的DOMNotation对象
* ent：待转换的DOMEntity对象
* pi：待转换的DOMProcessingInstruction对象
* df：待转换的DOMDocumentFragment对象
* doc：待转换的DOMDocument对象
* elem：待转换的DOMElement

**dbms\_xmldom.isNull**

**语法格式**

isNull(n dbms\_xmldom.DOMNode)

isNull(di dbms\_xmldom.DOMImplementation)

isNull(nl dbms\_xmldom.DOMNodeList)

isNull(nnm dbms\_xmldom.DOMNamedNodeMap)

isNull(cd dbms\_xmldom.DOMCharacterData)

isNull(a dbms\_xmldom.DOMAttr)

isNull(elem dbms\_xmldom.DOMElement)

isNull(t dbms\_xmldom.DOMText)

isNull(com dbms\_xmldom.DOMComment)

isNull(cds dbms\_xmldom.DOMCDATASection)

isNull(dt dbms\_xmldom.DOMDocumentType)

isNull(n dbms\_xmldom.DOMNotation)

isNull(ent dbms\_xmldom.DOMEntity)

isNull(pi dbms\_xmldom.DOMProcessingInstruction)

isNull(df dbms\_xmldom.DOMDocumentFragment)

isNull(doc dbms\_xmldom.DOMDocument)

**dbms\_xmldom.freeNode**

**语法格式**

freeNode(n dbms\_xmldom.DOMnode)

**参数说明**

nl：待释放的节点列表。

**dbms\_xmldom.freeNodeList**

**语法格式**

freeNodeList(nl dbms\_xmldom.DOMNodeList)

**参数说明**

nl：待释放的节点列表

**dbms\_xmldom.freeDocument**

**语法格式**

freeDocument(doc dbms\_xmldom.DOMDocument)

**参数说明**

doc：待释放的DOMDocument节点

**dbms\_xmldom.getFirstChild**

**语法格式**

getFirstChild(n dbms\_xmldom.DOMNode)

**参数说明**

n：DOMNode对象

**dbms\_xmldom.getLocalName**

**语法格式**

getLocalName(a dbms\_xmldom.DOMAttr)

getLocalName(elem dbms\_xmldom.DOMElement)

getLocalName(n dbms\_xmldom.DOMnode, data OUT VARCHAR2)

**参数说明**

* a：DOMAtrr对象
* elem：DOMElement对象
* n：DOMNode对象

**dbms\_xmldom.getNodeType**

**语法格式**

getNodeType(n dbms\_xmldom.DOMNode)

**参数说明**

n：DOMNode对象

**dbms\_xmldom.writeToClob**

**语法格式**

writeToClob(n dbms\_xmldom.DOMNode, cl IN OUT CLOB)

writeToClob(n dbms\_xmldom.DOMNode, cl IN OUT CLOB, pflag IN NUMBER, indent IN NUMBER)

writeToClob(doc dbms\_xmldom.DOMDocument, cl IN OUT CLOB)

writeToClob(doc dbms\_xmldom.DOMDocument, cl IN OUT CLOB, pflag IN NUMBER, indent IN NUMBER)

**参数说明**

* n：DOMNode对象
* cl：指定的clob
* pflag：换行标识。该参数Oracle官方文档无明确说明，经过测试得知取值范围为0-72的整数，所有低三位为4或5的值则表示不换行，其他值则换行。
* indent：缩进长度。该参数Oracle官方文档无明确说明，经过测试得知取值范围为0-12的整数，数值大小表示缩进的空格个数。缩进长度只在pflag为换行时生效。
* doc：xml文档

**dbms\_xmldom.writeToBuffer**

**语法格式**

writeToBuffer(n dbms\_xmldom.DOMNode, buffer IN OUT VARCHAR2)

writeToBuffer(doc dbms\_xmldom.DOMDocument, buffer IN OUT VARCHAR2)

writeToBuffer(df dbms\_xmldom.DOMDocumentFragment, buffer IN OUT VARCHAR2)

**参数说明**

* n：DOMNode节点对象
* doc：xml文档对象
* df：DOMDocumentFragment对象
* buffer：指定的缓冲区

**dbms\_xmldom.getChildNodes**

**语法格式**

getChildNodes(n dbms\_xmldom.DOMNode)

**参数说明**

n：DOMNode节点对象

**dbms\_xmldom.getLength**

**语法格式**

getLength(nl dbms\_xmldom.DOMNodeList)

getLength(nnm dbms\_xmldom.DOMNamedNodeMap)

getLength(cd dbms\_xmldom.DOMCharacterData)

**参数说明**

* nl：DOMNodeList对象
* nnm：DOMNamedNodeMap对象
* cd：DOMCharacterData对象

**dbms\_xmldom.item**

**语法格式**

item(nl dbms\_xmldom.DOMNodeList, idx IN PLS\_INTEGER)

item(nnm dbms\_xmldom.DOMNamedNodeMap, idx IN PLS\_INTEGER)

**参数说明**

* nl：节点列表
* nnm：DOMNamedNodeMap无序列表
* idx：在列表中要检索的索引值

**dbms\_xmldom.makeElement**

**语法格式**

makeElement(n dbms\_xmldom.DOMNode)

**参数说明**

n：DOMNode对象

**dbms\_xmldom.getElementsByTagName**

**语法格式**

getElementsByTagName(elem dbms\_xmldom.DOMElement, name IN VARCHAR2)

getElementsByTagName(elem dbms\_xmldom.DOMElement, name IN VARCHAR2, ns varchar2)

getElementsByTagName(doc dbms\_xmldom.DOMDocument, tagname IN VARCHAR2)

**参数说明**

* elem：DOMElement对象
* name：指定名称
* ns：指定的命名空间
* doc：DOMDocument对象

**dbms\_xmldom.cloneNode**

**语法格式**

cloneNode(n dbms\_xmldom.DOMNode, deep boolean)

**参数说明**

* n：DOMNode对象
* deep：是否拷贝子节点

**dbms\_xmldom.getNodeName**

**语法格式**

getNodeName(n dbms\_xmldom.DOMNode)

**参数说明**

n：DOMNode对象

**dbms\_xmldom.createDocument**

**语法格式**

createDocument(

namespaceuri IN VARCHAR2,

qualifiedname IN VARCHAR2,

doctype IN dbms\_xmldom.DOMType:= NULL)

**语法格式**

* namespaceuri ：命名空间URI
* qualifiedname ：根元素名
* doctype：documentType对象

**dbms\_xmldom.createElement**

**语法格式**

createElement(doc dbms\_xmldom.DOMDocument, tagname IN VARCHAR2)

createElement(doc dbms\_xmldom.DOMDocument, tagname IN VARCHAR2, ns IN VARCHAR2)

**参数说明**

* doc：xml文档
* tagname：DOMElement节点的名称
* ns：命名空间URI

**dbms\_xmldom.createDocumentFragment**

**语法格式**

createDocumentFragment(doc dbms\_xmldom.DOMDocument)

**参数说明**

doc：xml文档

**dbms\_xmldom.createTextNode**

**语法格式**

createTextNode(doc dbms\_xmldom.DOMDocument, data IN VARCHAR2)

**参数说明**

* doc：xml文档
* data：DOMText节点的内容

**dbms\_xmldom.createComment**

**语法格式**

createComment(doc dbms\_xmldom.DOMDocument, data IN VARCHAR2)

**参数说明**

* doc：xml文档
* data：DOMComment节点的内容

**dbms\_xmldom.createCDATASection**

**语法格式**

createCDATASection(doc dbms\_xmldom.DOMDocument, data IN VARCHAR2)

**参数说明**

* doc：xml文档
* data：DOMCDATASection节点的内容

**dbms\_xmldom.createProcessingInstruction**

**语法格式**

createProcessingInstruction(doc dbms\_xmldom.DOMDocument, target IN VARCHAR2, data IN VARCHAR2)

**参数说明**

* doc：xml文档
* target：处理指令的目标
* data：处理指令的内容文本

**dbms\_xmldom.createAttribute**

**语法格式**

createAttribute(doc dbms\_xmldom.DOMDocument, name IN VARCHAR2)

createAttribute(doc dbms\_xmldom.DOMDocument, name IN VARCHAR2, ns IN VARCHAR2)

**参数说明**

* doc：xml文档
* name：属性的名称
* ns：命名空间URI

**dbms\_xmldom.appendChild**

**语法格式**

appendChild(n dbms\_xmldom.DOMNode, newchild IN dbms\_xmldom.DOMNode)

**参数说明**

* n：DOMNode节点
* newchild ：待添加的子节点

**dbms\_xmldom.getDocumentElement**

**语法格式**

getDocumentElement(doc dbms\_xmldom.DOMDocument)

**参数说明**

doc：xml文档

**dbms\_xmldom.setAttribute**

**语法格式**

setAttribute(elem dbms\_xmldom.DOMElement, name IN VARCHAR2, newvalue IN VARCHAR2)

setAttribute(elem dbms\_xmldom.DOMElement, name IN VARCHAR2, newvalue IN VARCHAR2, ns IN VARCHAR2)

**参数说明**

* elem：DOMElement节点
* name：属性名
* newvalue：待设置的值
* ns：命名空间URI

**dbms\_xmldom.setAttributeNode**

**语法格式**

setAttributeNode(elem dbms\_xmldom.DOMElement, newattr IN dbms\_xmldom.DOMAttr)

setAttributeNode(elem dbms\_xmldom.DOMElement, newattr IN dbms\_xmldom.DOMAttr, ns IN VARCHAR2)

**参数说明**

* elem：DOMElement节点
* newattr：新的属性节点
* ns：命名空间URI

**dbms\_xmldom.getAttributes**

**语法格式**

getAttributes(n dbms\_xmldom.DOMNode)

**参数说明**

n：DOMNode节点

**dbms\_xmldom.getNodeValueAsClob**

**语法格式**

getNodeValueAsClob(n dbms\_xmldom.domnode)

**参数说明**

n：DOMNode节点

**dbms\_xmldom.getNodeValue**

**语法格式**

getNodeValue(n dbms\_xmldom.DOMNode)

**参数说明**

n：DOMNode节点

**dbms\_xmldom.getChildrenByTagName**

**语法格式**

getChildrenByTagName(elem dbms\_xmldom.DOMElement, name varchar2)

getChildrenByTagName(elem dbms\_xmldom.DOMElement, name varchar2, ns varchar2)

**参数说明**

* elem：DOMELement节点
* name：指定的名称

**dbms\_xmldom.getOwnerDocument**

**语法格式**

getOwnerDocument(n dbms\_xmldom.DOMNode)

**参数说明**

n：DOMNode节点

**dbms\_xmldom.newDOMDocument**

**语法格式**

newDOMDocument()

newDOMDocument(xmldoc IN xmltype)

newDOMDocument(cl IN clob)

**参数说明**

* xmldoc ：DOMDocument的xmltype的源数据
* cl：DOMDocument的clob的源数据

**dbms\_xmldom.hasChildNodes**

**语法格式**

hasChildNodes(n dbms\_xmldom.DOMNode)

**参数说明**

n：DOMNode节点

**dbms\_xmldom.insertBefore**

**语法格式**

insertBefore(

n dbms\_xmldom.DOMNode,

newchild IN dbms\_xmldom.DOMNode,

refchild IN dbms\_xmldom.DOMNode)

**参数说明**

* n：DOMNode节点
* newchild ：待插入的新节点
* refchild ：现有的子节点

**dbms\_xmldom.setVersion**

**语法格式**

setVersion(doc dbms\_xmldom.DOMDocument, version VARCHAR2)

**参数说明**

* doc：xml文档
* version：待设置的版本号

## DBMS\_XMLPARSER

**功能描述**

dbms\_xmlparser包用于构造一个解析xml文档的解析器，并提供一系列接口对解析树的结构和内容进行操作和访问。dbms\_xmlparser构造的解析树可以被DOM API访问。该内置包包含以下模块：

|  |  |
| --- | --- |
| **函数** | **描述** |
| [NEWPARSER](https://docs.vastdata.com.cn/zh/docs/VastbaseG100Ver2.2.10/doc/%E5%85%BC%E5%AE%B9%E6%80%A7%E6%89%8B%E5%86%8C/Oracle%E5%85%BC%E5%AE%B9%E6%80%A7/%E5%86%85%E7%BD%AE%E5%8C%85/DBMS_XMLPARSER.html" \l "XMLNEWPARSER) | 创建解析器。 |
| [GETDOCUMENT](https://docs.vastdata.com.cn/zh/docs/VastbaseG100Ver2.2.10/doc/%E5%85%BC%E5%AE%B9%E6%80%A7%E6%89%8B%E5%86%8C/Oracle%E5%85%BC%E5%AE%B9%E6%80%A7/%E5%86%85%E7%BD%AE%E5%8C%85/DBMS_XMLPARSER.html" \l "XMLGETDOCUMENT) | 返回解析器构建的DOM树文档的文档节点，只有文档被解析之后才能调用此函数。 |
| [PARSEBUFFER](https://docs.vastdata.com.cn/zh/docs/VastbaseG100Ver2.2.10/doc/%E5%85%BC%E5%AE%B9%E6%80%A7%E6%89%8B%E5%86%8C/Oracle%E5%85%BC%E5%AE%B9%E6%80%A7/%E5%86%85%E7%BD%AE%E5%8C%85/DBMS_XMLPARSER.html" \l "XMLPARSEBUFFER) | 解析存储在指定缓冲区中的xml并存储到解析器中。 |
| [PARSECLOB](https://docs.vastdata.com.cn/zh/docs/VastbaseG100Ver2.2.10/doc/%E5%85%BC%E5%AE%B9%E6%80%A7%E6%89%8B%E5%86%8C/Oracle%E5%85%BC%E5%AE%B9%E6%80%A7/%E5%86%85%E7%BD%AE%E5%8C%85/DBMS_XMLPARSER.html" \l "XMLPARSECLOB) | 解析指clob中的xml并存储到解析器中。 |

**注意事项**

* 该功能仅在数据库兼容模式为Oracle时能够使用（即创建DB时DBCOMPATIBILITY='A'），在其他数据库兼容模式下不能使用该特性。
* 系统已经安装python3。
* 创建plpython3u插件，需设置环境变量PYTHONHOME，包括以下内容（重启数据库生效）：
* PYTHONHOME：python3安装路径。
* LD\_LIBRARY\_PATH：增加python3安装路径下的lib目录。
* PATH：增加python3安装路径bin目录。

**示例**

1、创建plpython3u插件（需设置环境变量PYTHONHOME）。

create extension plpython3u;

2、用解析器把XML解析到指定的CLOB对象之中。

SET SERVEROUTPUT ON;

DECLARE

var clob;

parser DBMS\_XMLPARSER.parser;

BEGIN

var := '<?xml version="1.0"?>

<EMPSET>

<!--comment-->

<![CDATA[cdata]]>

<test>testtext</test>

<EMP num1="1" num2="2">

<EMP\_NO>7782</EMP\_NO>

<ENAME>CLARK</ENAME>

</EMP>

<EMP num="2">

<EMP\_NO>7839</EMP\_NO>

<ENAME>KING</ENAME>

</EMP>

<EMP num="3">

<EMP\_NO>7934</EMP\_NO>

<ENAME>MILLER</ENAME>

</EMP>

</EMPSET>';

parser := DBMS\_XMLPARSER.newParser;

DBMS\_XMLPARSER.parseClob(parser,var);

END;

/

**NEWPARSER**

**语法格式**

FUNCTION newParser

RETURN Parser;

**GETDOCUMENT**

**语法格式**

FUNCTION GETDOCUMENT(

p Parser)

RETURN DOMDocument;

**参数说明**

p：解析器实例。

**PARSEBUFFER**

**语法格式**

PROCEDURE PARSEBUFFER(

p Parser,

doc VARCHAR2);

**参数说明**

* p：解析器实例。
* doc：要解析的XML文档缓冲区。

**PARSECLOB**

**语法格式**

PROCEDURE PARSECLOB(

p Parser,

doc CLOB);

**参数说明**

* p：解析器实例。
* doc：要解析的XML文档缓冲区。

## URL\_TCP

**功能描述**

借助该内置包中的过程和函数，PL/SQL 应用程序可以使用 TCP/IP 与基于 TCP/IP 的外部服务器进行通信。因为许多 Internet 应用程序协议都基于 TCP/IP，所以这个包对使用 Internet 协议和电子邮件的 PL/SQL 应用程序很有用。

**兼容性**

Vastbase G100 V2.2版本与Oracle版本相比，暂不支持的函数或过程有：GET\_LINE\_NCHAR,GET\_TEXT\_NCHAR,SECURE\_CONNECTION。

**UTL\_TCP 类型**

UTL\_TCP包含两个类型：CONNECTION 类型和CRLF类型。

**CONNECTION 类型**

Utl\_tcp包与外界通信时需要包含存储通信内容的对象,需要在包中定义一个connection的类型,该类型的对象可以通过包自带的方法生成, connection类型的定义如下:

TYPE connection IS RECORD (

remote\_host VARCHAR2(255), -- remote host name

remote\_port PLS\_INTEGER, -- remote port number

local\_host VARCHAR2(255), -- local host name

local\_port PLS\_INTEGER, -- local port number

charset VARCHAR2(30), -- character set for on-the-wire communication

newline VARCHAR2(2), -- newline character sequence

tx\_timeout PLS\_INTEGER, -- transfer time-out value (in seconds)

private\_sd PLS\_INTEGER, -- for internal use

);

**参数说明：**

* remote\_host ：建立连接时远程主机的名称，当没有建立连接时为NULL。
* remote\_port：连接的远程主机的端口号，当没有建立连接时为NULL。
* local\_host：用于建立连接的本地主机名，当没有建立连接时为NULL。
* local\_port：用于建立连接的本地主机的端口号，当没有建立连接时为NULL。
* charset：在线字符集。由于数据库中的文本消息可能以不同于网络上预期的字符集（即通信协议指定的字符集，或通信另一端规定的字符集）的字符集进行编码，文本数据库中的消息在网络上发送和接收时被转换为在线字符集或从在线字符集转换。
* newline：换行符序列。
* tx\_timeout：在此连接中放弃读取或写入操作之前，UTL\_TCP 包等待的时间（以秒为单位）。
* private\_sd：仅内部使用。

**CRLF类型**

字符序列回车换行。它是许多通信标准常用的换行符序列。 不要修改它的值。

这个包变量定义了许多 Internet 协议中常用的换行符序列。 这是 write\_line() 的换行符序列的默认值，在打开连接时可以通过参数指定,覆盖默认值。默认的回撤换行的值为CRLF。

CRLF CONSTANT VARCHAR2(2 CHAR);

**内置包子程序**

| **函数** | **描述** |
| --- | --- |
| [OPEN\_CONNECTION](#OPEN_CONNECTION) | 用以创建到其他服务的tcp/ip连接,并返回一个connection对象 |
| [AVAILABLE](#AVAILABLE) | 用于从 TCP/IP 连接读取的字节数。 |
| [READ\_RAW](#READ_RAW) | 此函数在打开的连接上从服务接收二进制数据,返回实际接收到的数据字节数。 |
| [WRITE\_RAW](#WRITE_RAW) | 此函数将二进制消息传输到开放连接上的服务,返回实际传输的数据字节数。 |
| [READ\_TEXT](#READ_TEXT) | 此函数在打开的连接上从服务接收文本数据,返回接受到的文本字节数。 |
| [WRITE\_TEXT](#WRITE_TEXT) | 此功能在开放连接上将文本消息传输到服务，并返回写入数据的字节数。 |
| [READ\_LINE](#READ_LINE) | 此函数在打开的连接上接收来自服务的文本行。 一行以换行、回车 或 回车后跟换行结束，返回实际接收到的数据字符数。 |
| [WRITE\_LINE](#WRITE_LINE) | 此函数将文本行传输到打开连接上的服务。 换行符序列将在传输之前附到消息中。返回实际传输的数据字符数。 |
| [GET\_RAW()](#GET)  [GET\_TEXT()](#GET)  [GET\_LINE()](#GET) | 方便的读取函数形式，返回读取的数据而不是读取的数据量。 |
| [FLUSH](#FLUSH) | 此过程将输出缓冲区中的所有数据（如果使用缓冲区）立即传输到服务器。 |
| [CLOSE\_CONNECTION](#CLOSE_CONNECTION) | 关闭打开的 TCP/IP 连接。 |
| [CLOSE\_ALL\_CONNECTIONS](#CLOSE_ALL_CONNECTIONS) | 关闭所有打开的 TCP/IP 连接。 |

**OPEN\_CONNECTION**

**语法格式**

UTL\_TCP.OPEN\_CONNECTION (remote\_host IN VARCHAR2,

remote\_port IN PLS\_INTEGER,

local\_host IN VARCHAR2 DEFAULT NULL,

local\_port IN PLS\_INTEGER DEFAULT NULL,

in\_buffer\_size IN PLS\_INTEGER DEFAULT NULL,

out\_buffer\_size IN PLS\_INTEGER DEFAULT NULL,

charset IN VARCHAR2 DEFAULT NULL,

newline IN VARCHAR2 DEFAULT CRLF,

tx\_timeout IN PLS\_INTEGER DEFAULT NULL)

RETURN connection;

**参数说明**

| **参数名称** | **参数类型** | **参数描述** |
| --- | --- | --- |
| remote\_host | in | 提供服务的主机名。 |
| remote\_port | in | 要连接服务的监听端口。 |
| local\_host | in | 本地的主机名。 |
| local\_port | in | 本地的端口号。 |
| in\_buffer\_size | in | 输入缓冲区的大小。 使用输入缓冲区可以加快从服务器接收数据的执行性能。 缓冲区的适当大小取决于客户端和服务器之间的数据流以及网络状况。 0 值表示不应使用缓冲区。 NULL 值意味着调用者不关心是否使用了缓冲区。 输入缓冲区的最大大小为 32767 字节。 |
| out\_buffer\_size | in | 输出缓冲区的大小。 使用输出缓冲区可以加快向服务器发送数据的执行性能。 适当的缓冲区大小取决于客户端和服务器之间的数据流以及网络状况。 0 值表示不应使用缓冲区。 NULL 值意味着调用者不关心是否使用了缓冲区。 输出缓冲区的最大大小为 32767 字节。 |
| charset | in | 在线字符集。 由于数据库中的文本消息可能以不同于网络上预期的字符集（即通信协议指定的字符集，或通信另一端规定的字符集）的字符集进行编码，文本 数据库中的消息将在使用 read\_text()、read\_line()、write\_text() 和 write\_line() 在网络上发送和接收时转换为在线字符集或从在线字符集转换。 不需要转换时，将此参数设置为 NULL。 |
| newline | in | 换行符序列。 此换行符序列附加到 write\_line() API 发送的文本行 |
| tx\_timeout | in | 在放弃此连接中的读取或写入操作之前，UTL\_TCP 包应等待的时间（以秒为单位）。 在读取操作中，如果没有数据可立即读取，则此包放弃。 在写操作中，如果输出缓冲区已满并且没有数据要在网络中发送而不被阻塞，则此包将放弃。 零 (0) 表示根本不等待。 NULL 表示永远等待。 |

**注意事项**

UTL\_TCP 包打开的连接可以保持打开状态，并在共享服务器配置中从一个数据库调用传递到另一个数据库调用。但是，必须明确关闭连接。当存储连接的vb记录变量超出vb程序的范围时，连接将保持打开状态。未能关闭不需要的连接可能会导致不必要地占用本地和远程系统资源。

**AVAILABLE**

**语法格式**

UTL\_TCP.AVAILABLE (

c IN OUT NOCOPY connection,

timeout IN PLS\_INTEGER DEFAULT 0)

RETURN INTEGER;

**参数说明**

| **参数名称** | **参数类型** | **参数描述** |
| --- | --- | --- |
| c | in | 用于确定可供读取的数据量的 TCP 连接。 |
| timeout | in | 在放弃并报告没有可用数据之前等待的时间（以秒为单位）。 零 (0) 表示根本不等待。 NULL 表示永远等待。 |

**注意事项**

* 连接必须已经通过调用 open\_connection() 打开。
* 此函数返回的可供读取的字节数可能少于实际可用的字节数。 在某些平台上，此函数可能只返回 1，表示某些数据可用。 如果您担心应用程序的可移植性，请假设此函数在数据可供读取时返回正值，在没有数据可用时返回 0。以下示例说明了以可移植方式使用此函数：

DECLARE

c utl\_tcp.connection

data VARCHAR2(256);

len PLS\_INTEGER;

BEGIN

c := utl\_tcp.open\_connection(...);

LOOP

IF (utl\_tcp.available(c) > 0) THEN

len := utl\_tcp.read\_text(c, data, 256);

ELSE

---do some other things

. . . .

END IF

END LOOP;

END;

**READ\_RAW**

**语法格式**

UTL\_TCP.READ\_RAW (c IN OUT NOCOPY connection,

data IN OUT NOCOPY RAW,

len IN PLS\_INTEGER DEFAULT 1,

peek IN BOOLEAN DEFAULT FALSE)

RETURN INTEGER;

**参数说明**

| **参数名称** | **参数类型** | **参数描述** |
| --- | --- | --- |
| c | in | 从中接收数据的 TCP 连接 |
| data | in | 收到的数据。 |
| len | in | 要接收的数据字节数。 |
| peek | in | 通常，您希望读取数据并将其从输入队列中删除，即消费它。 在某些情况下，您可能只想向前看数据，即偷看它，而不将其从输入队列中删除，以便在下一次调用中仍然可以读取（甚至偷看）。 要将数据保留在输入队列中，请将此标志设置为 TRUE 并在打开连接之前设置输入缓冲区。 您可以查看的数据量（即读取但保留在输入队列中）必须小于输入缓冲区的大小。 |

**注意事项**

* 连接必须已经通过调用 open\_connection() 打开。 在读取指定数量的字符或到达输入结尾之前，此函数不会返回。
* 如果在打开连接时设置了传输超时，则该函数等待每个数据包准备好读取，直到发生超时。 如果发生，该函数停止读取并返回所有读取成功的数据。 如果没有成功读取数据，则会引发 transfer\_timeout 异常。 可以处理异常，稍后可以重试读取操作。

**WRITE\_RAW**

**语法格式**

UTL\_TCP.WRITE\_RAW (c IN OUT NOCOPY connection,

data IN RAW,

len IN PLS\_INTEGER DEFAULT NULL)

RETURN PLS\_INTEGER;

**参数说明**

| **参数名称** | **参数类型** | **参数描述** |
| --- | --- | --- |
| c | in | 要发送数据数据的 TCP 连接 |
| data | in | 发送的数据。 |
| len | in | 要发送的数据字节数。 |

**注意事项**

连接必须已经通过调用 open\_connection() 打开。

**READ\_TEXT**

**语法格式**

UTL\_TCP.READ\_TEXT (c IN OUT NOCOPY connection,

data IN OUT NOCOPY VARCHAR2,

len IN PLS\_INTEGER DEFAULT 1,

peek IN BOOLEAN DEFAULT FALSE) RETURN

INTEGER;

**参数说明**

| **参数名称** | **参数类型** | **参数描述** |
| --- | --- | --- |
| c | in | 从中接收数据的 TCP 连接 |
| data | in | 收到的数据。 |
| len | in | 要接收的数据字节数。 |
| peek | in | 通常，用户希望读取数据并将其从输入队列中移除，即消费它。 在某些情况下，用户可能只想提前查看数据而不将其从输入队列中删除，以便在下一次调用时仍可读取（甚至查看）。 要将数据保留在输入队列中，请将此标志设置为 TRUE，并且必须在打开连接时设置输入缓冲区。 您可以查看的数据量（即读取但保留在输入队列中）必须小于输入缓冲区的大小。 |

**注意事项**

* 连接必须已经通过调用 open\_connection() 打开。
* 在读取指定数量的字符或到达输入结尾之前，此函数不会返回。文本消息将从打开连接时指定的在线字符集转换为数据库字符集，然后再返回给调用者。

**WRITE\_TEXT**

**语法格式**

UTL\_TCP.WRITE\_TEXT (c IN OUT NOCOPY connection,

data IN VARCHAR2,

len IN PLS\_INTEGER DEFAULT NULL)

RETURN INTEGER;

**参数说明**

| **参数名称** | **参数类型** | **参数描述** |
| --- | --- | --- |
| c | in | 需要发送数据的 TCP 连接 |
| data | in | 要发送的数据 |
| len | in | 要发送的数据字节数。 |

**注意事项**

连接必须已经通过调用 open\_connection() 打开。

**READ\_LINE**

**语法格式**

UTL\_TCP.READ\_LINE (c IN OUT NOCOPY connection,

data IN OUT NOCOPY VARCHAR2,

remove\_crlf IN BOOLEAN DEFAULT FALSE,

peek IN BOOLEAN DEFAULT FALSE)

RETURN INTEGER;

**参数说明**

| **参数名称** | **参数类型** | **参数描述** |
| --- | --- | --- |
| c | in | 从中接收数据的 TCP 连接 |
| data | in | 收到的数据。 |
| remove\_crlf | in | 如果为 TRUE，则从接收到的消息中删除尾随 CR/LF 字符。 |
| peek | in | 通常，您希望读取数据并将其从输入队列中删除，即消费它。 在某些情况下，您可能只想向前看数据，即偷看它，而不将其从输入队列中删除，以便在下一次调用中仍然可以读取（甚至偷看）。 要将数据保留在输入队列中，请将此标志设置为 TRUE 并在打开连接之前设置输入缓冲区。 您可以查看的数据量（即读取但保留在输入队列中）必须小于输入缓冲区的大小。 |

**注意事项**

* 连接必须已经通过调用 open\_connection() 打开。
* 在到达行尾或到达输入末尾之前，此函数不会返回。文本消息将从打开连接时指定的在线字符集转换为数据库字符集，然后再返回给调用者。

**WRITE\_LINE**

**语法格式**

UTL\_TCP.WRITE\_LINE (c IN OUT NOCOPY connection,

data IN VARCHAR2 DEFAULT NULL)

RETURN PLS\_INTEGER;

**参数说明**

| **参数名称** | **参数类型** | **参数描述** |
| --- | --- | --- |
| c | in | 需要发送数据的 TCP 连接 |
| data | in | 要发送的数据 |

**注意事项**

连接必须已经通过调用 open\_connection() 打开。

**GET\_RAW(),GET\_TEXT(),GET\_LINE()**

**语法格式**

UTL\_TCP.GET\_RAW (c IN OUT NOCOPY connection,

len IN INTEGER DEFAULT 1,

peek IN BOOLEAN DEFAULT FALSE) RETURN RAW;

UTL\_TCP.GET\_TEXT (c IN OUT NOCOPY connection,

len IN INTEGER DEFAULT 1,

peek IN BOOLEAN DEFAULT FALSE) RETURN VARCHAR2;

UTL\_TCP.GET\_LINE (c IN OUT NOCOPY connection,

remove\_crlf IN BOOLEAN DEFAULT false,

peek IN BOOLEAN DEFAULT FALSE) RETURN

VARCHAR2;

**参数说明**

| **参数名称** | **参数类型** | **参数描述** |
| --- | --- | --- |
| c | in | 从中接收数据的 TCP 连接 |
| len | in | 收到的数据。 |
| peek | in | 通常，您希望读取数据并将其从输入队列中删除，即消费它。 在某些情况下，您可能只想向前看数据，即偷看它，而不将其从输入队列中删除，以便在下一次调用中仍然可以读取（甚至偷看）。 要将数据保留在输入队列中，请将此标志设置为 TRUE 并在打开连接之前设置输入缓冲区。 您可以查看的数据量（即读取但保留在输入队列中）必须小于输入缓冲区的大小。 |
| remove\_crlf | in | 如果为 TRUE，则从接收到的消息中删除尾随 CR/LF 字符。 |

**注意事项**

连接必须已经通过调用 open\_connection() 打开。

**FLUSH**

**语法格式**

UTL\_TCP.FLUSH (c IN OUT NOCOPY connection);

**参数说明**

| **参数名称** | **参数类型** | **参数描述** |
| --- | --- | --- |
| c | in | 要将数据发送到的 TCP 连接。 |

**注意事项**

连接必须已经通过调用 open\_connection() 打开。

**CLOSE\_CONNECTION**

**语法格式**

UTL\_TCP.close\_CLOSE\_CONNECTION (c IN OUT NOCOPY connection);

**参数说明**

| **参数名称** | **参数类型** | **参数描述** |
| --- | --- | --- |
| c | in | 要关闭的 TCP 连接。 |

**注意事项**

* 连接必须已经通过调用 open\_connection() 打开。
* c 的 remote\_host、remote\_port、local\_host、local\_port 和 charset 字段将在连接关闭后重置。
* 必须明确关闭打开的连接。 当存储连接的记录变量超出程序的范围时，打开的连接将保持打开状态。 未能关闭不需要的连接可能会导致不必要地占用本地和远程系统资源。

**CLOSE\_ALL\_CONNECTIONS**

**语法格式**

UTL\_TCP.CLOSE\_ALL\_CONNECTIONS;

**示例**

**请求网页**

下面的代码示例说明了如何使用 TCP/IP 包通过 HTTP 检索网页。 它连接到在端口 80（HTTP 的标准端口）上侦听的 Web 服务器并请求根文档。

DECLARE

c utl\_tcp.connection; -- TCP/IP connection to the Web server

ret\_val pls\_integer;

BEGIN

c := utl\_tcp.open\_connection(remote\_host => 'www.acme.com',

remote\_port => 80,

charset => 'US7ASCII'); -- open connection

ret\_val := utl\_tcp.write\_line(c, 'GET / HTTP/1.0'); -- send HTTP request

ret\_val := utl\_tcp.write\_line(c);

BEGIN

LOOP

dbms\_output.put\_line(utl\_tcp.get\_line(c, TRUE)); -- read result

END LOOP;

EXCEPTION

WHEN utl\_tcp.end\_of\_input THEN

NULL; -- end of input

END;

utl\_tcp.close\_connection(c);

END;

## UTL\_FILE

**功能描述**

UTL\_FILE包是用于提供在存储过程或函数中对系统文件进行读写的功能。该内置包包含以下函数：

|  |  |
| --- | --- |
| **函数** | **描述** |
| fopen | 打开一个文件用于输入或输出。 |
| is\_open | 检查一个文件句柄是否指向一个打开的文件。 |
| get\_line | 从一个打开的文件中读取指定的一行文本。 |
| get\_nextline | 获得下一行。 |
| put | 将一个字符串写入到文件。 |
| put\_line | 在文件中写入一行，在行的末尾写入一个行终结符。 |
| new\_line | 在文件末尾写入一个行终结符。 |
| putf | 格式化输出过程。 |
| fflush | 将文件缓存刷到物理写入。 |
| fclose | 关闭一个文件。 |
| fclose\_all | 关闭所有打开的文件。 |
| fremove | 删除磁盘上的文件。 |
| frename | 重命名一个已经存在的文件。 |
| fremove | 删除磁盘上的文件。 |
| fcopy | 将一个文件中的内容拷贝到另一个文件中。 |
| fgetattr | 获取文件属性。 |
| tmpdir | 获得临时目录路径。 |

**注意事项**

该功能仅在数据库兼容模式为Oracle时能够使用（即创建DB时DBCOMPATIBILITY='A'），在其他数据库兼容模式下不能使用该特性。

**兼容性**

Vastbase G100 V2.2版本与Oracle 11g R2版本相比，不支持的函数有：FGETPOS、FOPEN\_NCHAR、FSEEK、GET\_LINE\_NCHAR、GET\_ROW、PUT\_NCHAR、PUT\_LINE\_NCHAR、PUT\_row。

**示例**

1、创建主备文件，在操作系统中执行如下命令。

echo "111aaa

222bbb

333ccc

444ddd

555eee

666fff

777ggg

888hhh

999iii

101010jjj

111111kkk" > /tmp/test.txt

2、连接数据库，创建并切换至兼容模式为Oracle的数据库db\_oracle。

CREATE DATABASE db\_oracle dbcompatibility='A';

\c db\_oracle

3、在数据库下执行。

insert into utl\_file.utl\_file\_dir values ('/tmp');

4、在数据库下执行文件复制（utl\_file.fcopy是Oracle中的一个“静态参数”，可以设置一个或多个路径。）。

select utl\_file.fcopy('/tmp', 'test.txt', '/tmp', 'test\_cp.txt',3,6);

5、查看复制的文件,在shell下执行。

cat test\_cp.txt

返回结果如下，则表示文件复制成功：

333ccc

444ddd

555eee

666fff

7、连接db\_oracle数据库，fgetattr查看文件属性。

do language plpgsql $$

declare

ex boolean;

flen int;

bsize int;

begin

-- utl\_file.fgetattr('/tmp'::text, 'test.txt'::text, ex, flen, bsize);

select \* from utl\_file.fgetattr('/tmp'::text, 'test.txt'::text) into ex,flen,bsize;

IF ex THEN

raise notice 'File Exists';

ELSE

raise notice 'File Does Not Exist' ;

END IF;

raise notice 'FileLength: %' , flen;

raise notice 'BlockSize: %',bsize;

end;

$$;

返回结果为：

NOTICE: File Exists

NOTICE: FileLength: 0

NOTICE: BlockSize: 4096

ANONYMOUS BLOCK EXECUTE

8、调用存储过程对文件进行读写。

do language plpgsql $$

declare

vInHandle utl\_file.file\_type;

vOutHandle utl\_file.file\_type;

vNewLine VARCHAR2(250);

begin

vInHandle := utl\_file.fopen('/tmp', 'test.txt','R');

LOOP

BEGIN

select \* from utl\_file.get\_line(vInHandle)into vNewLine ;

raise notice '%', vNewLine;

EXCEPTION

WHEN OTHERS THEN

EXIT;

END;

END LOOP;

vOutHandle := utl\_file.fopen('/tmp', 'test\_cp.txt', 'w');

perform utl\_file.put\_line(vOutHandle, 'hello lsxy');

utl\_file.fflush(vOutHandle);

utl\_file.fclose\_all();

end;

$$;

9、在shell执行如下命令，查看文件内容。

cat /tmp/test\_cp.txt

返回结果为：

hello lsxy

## UTL\_HTTP

**功能描述**

使用UTL\_HTTP包可通过SQL和过程语言调用超文本传输协议HTTP，允许通过HTTP访问internet上的数据。该内置包包含以下函数：

|  |  |
| --- | --- |
| **函数** | **描述** |
| [SET\_TRANSFER\_TIMEOUT](#HTTPSET_TRANSFER_TIMEOUT) | 设置UTL\_HTTP从WEB服务器或代理服务器读取HTTP响应的超时值。 |
| [REQUEST](#HTTPREQUEST) | 从指定的URL请求检索数据。可直接用于SQL查询。 |
| [BEGIN\_REQUEST](#HTTPBEGIN_REQUEST) | 开始一个新的HTTP请求。 |
| [SET\_HEADER](#HTTPSET_HEADER) | 用于设置http请求头。 |
| [GET\_RESPONSE](#HTTPGET_RESPONSE) | 用于读取http响应。 |
| [SET\_BODY\_CHATSET](#HTTPSET_BODY_CHATSET) | Content-Type标头中未指定字符集时，设置请求体的字符集。 |
| [WRITE\_TEXT](#HTTPWRITE_TEXT) | 在HTTP请求体中写入一些文本数据。 |
| [READ\_LINE](#HTTPREAD_LINE) | 以文本形式读取http响应正文，直到到达尾行。 |
| [END\_RESPONSE](#HTTPEND_RESPONSE) | 结束HTTP响应，完成HTTP请求和响应。 |
| [END\_REQUEST](#HTTPEND_REQUEST) | 结束HTTP请求。 |
| [GET\_DETAILED\_SQLERRM](#HTTPGET_DETAILED_SQLERRM) | 检索最后引发的异常的详细SQLERRM。 |

**类型说明**

**req类型**

**功能描述**

req用于接受http的request请求。

**语法格式**

TYPE req RECORD(

url VARCHAR2(32767),

method VARCHAR2(64),

http\_version VARCHAR2(64),

);

**参数说明**

* url：http请求的URL，在BEGIN\_REQUEST进行设置。
* method：请求url的方法，在BEGIN\_REQUEST进行设置。
* http\_version：发送请求的http协议版本。

**resq类型**

**功能描述**

使用resq的记录表示http的响应。

**语法格式**

TYPE resq RECORD(

status\_code PLS\_INTEGER,

reason\_phrase VARCHAR2(256),

http\_version VARCHAR2(64),

);

**参数说明**

* url：http请求的URL，在BEGIN\_REQUEST进行设置。
* method：请求url的方法，在BEGIN\_REQUEST进行设置。
* http\_version：发送请求的http协议版本。

**使用流程**

1、定义http.rep和http.resp变量。

2、设置http超时时间。

3、进行url编码，设置header，设置body字符集。

4、获取request请求。

5、获取response。

6、异常处理。

**注意事项**

* 该功能仅在数据库兼容模式为Oracle时能够使用（即创建DB时DBCOMPATIBILITY='A'），在其他数据库兼容模式下不能使用该特性。
* 使用该功能需要创建plpython3u插件。

**兼容性**

Vastbase G100 V2.2版本与Oracle 11g R2版本相比，仅支持本文档介绍的函数，该内置包的其他函数暂不支持。

**示例**

1、系统已经安装python3环境。

2、创建并切换至兼容模式为Oracle的数据库db\_oracle。

CREATE DATABASE db\_oracle dbcompatibility='A';

\c db\_oracle

3、创建plpython3u插件。

create extension plpython3u;

4、使用UTL\_HTTP包创建函数。

CREATE OR REPLACE FUNCTION FN\_HTTP\_GET (v\_url VARCHAR2)

RETURN VARCHAR2

AS

BEGIN

DECLARE

req UTL\_HTTP.REQ;

resp UTL\_HTTP.RESP;

res record;

value varchar2;

v\_text text;

BEGIN

v\_text := '';

BEGIN

req := UTL\_HTTP.BEGIN\_REQUEST( url => v\_url, method => 'GET' );

UTL\_HTTP.SET\_BODY\_CHARSET(req, 'UTF-8');

UTL\_HTTP.SET\_HEADER(req, 'Content-Type', 'text/html');

resp := UTL\_HTTP.GET\_RESPONSE( req );

LOOP

UTL\_HTTP.READ\_LINE( resp, value, TRUE );

--value:=res.data;

v\_text := v\_text || value;

END LOOP;

UTL\_HTTP.END\_RESPONSE( resp );

EXCEPTION

WHEN "utl\_http.end\_of\_body" THEN

UTL\_HTTP.END\_RESPONSE( resp );

WHEN OTHERS THEN

RAISE EXCEPTION '(%)', SQLERRM;

UTL\_HTTP.END\_RESPONSE(resp);

END;

return v\_text;

END;

END;

5、删除插件。

drop extension plpython3u;

6、调用函数。

select FN\_HTTP\_GET('http://172.16.19.135/reviewboard/r/ ');

7、重新创建拓展。

create extension plpython3u;

8、调用函数。

select FN\_HTTP\_GET('http://172.16.19.135/reviewboard/r/ ');

**SET\_TRANSFER\_TIMEOUT**

**语法格式**

UTL\_HTTP.SET\_TRANSFER\_TIMEOUT(timeout);

**参数说明**

timeout：网络传输超时值（以秒为单位）。

**REQUEST**

**语法格式**

UTL\_HTTP.REQUEST(url);

**参数说明**

url：请求的url地址。

**BEGIN\_REQUEST**

**语法格式**

UTL\_HTTP.BEGIN\_REQUEST(r);

**参数说明**

r：http请求的URL。

**SET\_HEADER**

**语法格式**

UTL\_HTTP.SET\_HEADER(r,name,value);

**参数说明**

* r：http请求。
* name：http的header名称。
* value：http的header值。

**GET\_RESPONSE**

**语法格式**

UTL\_HTTP.GET\_RESPONSE(r);

**参数说明**

r：http响应。

**SET\_BODY\_CHATSET**

**语法格式**

UTL\_HTTP.SET\_BODY\_CHATSET(r,charset);

**参数说明**

* r：http响应。
* charset：正文的字符集。

**WRITE\_TEXT**

**语法格式**

UTL\_HTTP.WRITE\_TEXT(r,date);

**参数说明**

* r：http请求。
* date：需要在HTTP请求body正文中发送的文本数据。。

**READ\_LINE**

**语法格式**

UTL\_HTTP.READ\_LINE(r,date,crlf);

**参数说明**

* r：http响应。
* date：http的body文本。
* crlf：是否删除换行符。

**END\_RESPONSE**

**语法格式**

UTL\_HTTP.END\_RESPONSE(r);

**参数说明**

r：http响应。

**END\_REQUEST**

**语法格式**

UTL\_HTTP.END\_REQUEST(r);

**参数说明**

r：http请求。

**GET\_DETAILED\_SQLERRM**

**语法格式**

UTL\_HTTP.GET\_DETAILED\_SQLERRM

RETURN VARCHAR2;

## UTL\_RAW

**功能描述**

UTL\_RAW包提供了一系列对raw类型数据进行处理的函数。

**函数说明**

|  |  |
| --- | --- |
| **函数** | **描述** |
| cast\_to\_raw | 将输入的varchar2类型的数据转换成raw类型的数据，在此过程中数据并未以任何方式修改，仅仅只是修改了数据类型。 |
| cast\_to\_varchar2 | 将输入的raw类型数据转换回varchar2类型。 |

**注意事项**

cast\_to\_varchar2函数的输入字符须为十六进制表示形式的16个字符，否则会返回识别错误。

**示例**

* **cast\_to\_raw函数**

示例一：

select utl\_raw.cast\_to\_raw('test') from dual;

返回结果为'test'字符串每个字符的ASCII码的十六进制表示形式拼接而成，结果如下：

cast\_to\_raw

-----------

74657374

(1 row)

示例二：

select utl\_raw.cast\_to\_raw('测试') from dual;

返回结果为'测试'在utf8编码下中文转换成十六进制的表现形式，如下所示：

cast\_to\_raw

-----------

E6B58BE8AF95

(1 row)

* **cast\_to\_varchar2函数**

示例一：

select utl\_raw.cast\_to\_varchar2('74657374') from dual;

返回结果为:

cast\_to\_varchar2

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

test

(1 row)

示例二：

select utl\_raw.cast\_to\_varchar2('E6B58BE8AF95') from dual;

返回结果为:

cast\_to\_varchar2

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

测试

(1 row)

注意：cast\_to\_varchar2函数的输入字符须为十六进制表示形式的16个字符，否则会返回识别错误。

select utl\_raw.cast\_to\_varchar2('test') from dual;

ERROR: invalid hexadecimal digit: "t"

LINE 1: select utl\_raw.cast\_to\_varchar2('test') from dual;

∧

CONTEXT: referenced column: cast\_to\_varchar2

## UTL\_SMTP

**功能描述**

该内置包提供了发送邮件的功能。该内置包包含以下函数：

|  |  |
| --- | --- |
| **函数** | **描述** |
| [OPNE\_CONNECTION](#open_connection) | 与SMTP服务器建立连接。 |
| [AUTH](#auth) | 向SMTP服务器发送AUTH命令进行身份验证。 |
| [HELO](#helo) | 使用helo命令与SMTP服务器进行初始化握手。 |
| [MAIL](#mail) | 设置与SMTP服务器发送的邮箱地址。 |
| [RCPT](#rcpt) | 指定接收邮箱地址。 |
| [OPNE\_DATA](#open_data) | 发送DATA命令，后面调用write\_data写入要发送的文本信息。 |
| [WRITE\_DATA](#write_data) | 发送邮件文本内容，重复调用可以将数据附加到内容中。 |
| [CLOSE\_DATA](#close_data) | 结束发送电子邮件。 |
| [QUIT](#quit) | 断开与SMTP服务器的连接。 |

**注意事项**

* Vastbase G100 V2.2是基于plpython3u实现以上函数及存储过程，完成发送邮件的功能。
* 该功能仅在数据库兼容模式为Oracle时能够使用（即创建DB时DBCOMPATIBILITY='A'），在其他数据库兼容模式下不能使用该特性。

**兼容性**

Vastbase G100 V2.2版本与Oracle 11g R2版本相比，不支持的函数有：command、command\_replies、date、ehlo、help、noop、starttls、vrfy、write\_raw\_data。

**使用流程**

1、安装python3。

由于该功能基于python3(python3.8)的环境，且带有libpython3.8.so库文件（编译时指定--enable-shared可生成）。

若需要启用tls安全认证，需要安装openssl及openssl-devel，且python3编译时指定--with-ssl-default-suites=openssl，如安装完在python3命令行执行import ssl成功，则说明安装ssl模块成功。

注：麒麟系统import ssl如出现symbol SSLv3\_method not defined的错误，是openssl的版本不支持SSLv3。可在python3源码修改Mudule/\_ssl.c中的SSLv3\_method()为DTLS\_method()，保存后重新编译python3即可。

2、添加如下环境变量。

* PYTHONHOME：python3安装路径
* LD\_LIBRARY\_PATH：增加python3安装路径下的lib目录
* PATH：增加python3安装路径bin目录

3、创建plython3u扩展。

--切换到目标DB

\c targetDB

--创建plpython3u扩展

vastbase=# create extension plpython3u;

--查看到内置包相关函数信息

\df utl\_smtp.\*

4、调用函数发送邮件。

--创建发送邮件函数：（v\_sender填写企业邮箱发送邮箱地址，v\_passwd填写邮箱密码）

CREATE OR REPLACE FUNCTION send\_mail(recipient varchar, subject varchar, message varchar) returns void as

$$

declare

v\_mailhost varchar := '119.147.3.176';

v\_sender varchar := 'xxxx@vastdata.com.cn';

v\_passwd varchar := 'xxxx';

v\_conn utl\_smtp.connection;

v\_msg varchar;

crlf varchar(2) := CHR(10);

v\_ret varchar;

err\_context text;

begin

v\_conn := utl\_smtp.open\_connection(v\_mailhost, 587, 1000, true);

perform utl\_smtp.auth(v\_conn, v\_sender, v\_passwd, 'CRAM-MD5 PLAIN LOGIN');

perform utl\_smtp.helo(v\_conn, v\_mailhost);

perform utl\_smtp.mail(v\_conn, v\_sender);

perform utl\_smtp.rcpt(v\_conn, recipient);

v\_msg := 'From: ' || v\_sender || crlf ||

'To: ' || recipient || crlf ||

'Subject: ' || subject || crlf ||

crlf || message;

raise notice '%', v\_msg;

perform utl\_smtp.open\_data(v\_conn);

perform utl\_smtp.write\_data(v\_conn, v\_msg);

perform utl\_smtp.close\_data(v\_conn);

perform utl\_smtp.quit(v\_conn);

exception

when others then

GET STACKED DIAGNOSTICS err\_context = PG\_EXCEPTION\_CONTEXT;

RAISE INFO 'Error Name:%',SQLERRM;

RAISE INFO 'Error State:%', SQLSTATE;

RAISE INFO 'Error Context:%', err\_context;

end;

$$ LANGUAGE plpgsql;

--调用send\_mail函数发送邮件：（第一个参数填写接收邮箱地址）

select send\_mail('xxx@qq.com', 'utl\_smtp测试', '您好，这是测试邮件！');

**open\_connection函数**

**语法格式**

UTL\_SMTP.OPEN\_CONNECTION (

host IN VARCHAR2,

port IN PLS\_INTEGER DEFAULT 25,

c OUT connection,

tx\_timeout IN PLS\_INTEGER DEFAULT NULL,

wallet\_path IN VARCHAR2 DEFAULT NULL,

wallet\_password IN VARCHAR2 DEFAULT NULL,

secure\_connection\_before\_smtp IN BOOLEAN DEFAULT FALSE,

secure\_host IN VARCHAR2 DEFAULT NULL)

RETURN connection;

**参数说明**

* host：SMTP服务器地址。
* port：SMTP服务器端口，默认25。
* c：SMTP连接。
* tx\_timeout：UTL\_SMTP包在连接中的读或写操作超时等待的时间（秒），默认为NULL。
* wallet\_path：指定客户端钱包，默认为NULL（接口预留，暂不实现）。
* wallet\_password：打开钱包的密码，默认为NULL（接口预留，暂不实现）。
* secure\_connection\_before\_smtp：是否在smtp通信之间建立SSL/TLS的安全连接，默认为false。
* secure\_host：使用安全连接的证书域名，默认为NULL（接口预留，暂不实现）

**auth函数**

**语法格式**

UTL\_SMTP.AUTH (

c IN OUT NOCOPY connection,

username IN VARCHAR2,

password IN VARCHAR2,

schemes IN VARCHAR2 DEFAULT NON\_CLEARTEXT\_PASSWORD\_SCHEMES);

**参数说明**

* c：open\_connection返回的smtp连接。
* username：发送邮件的用户名
* password：发送邮件用户的密码
* schemas：身份验证方案，默认为CRAM-MD5

**helo函数**

**语法格式**

UTL\_SMTP.HELO (

c IN OUT NOCOPY connection,

domain IN VARCHAR2);

**参数说明**

* c：open\_connection返回的smtp连接。
* domain：发送邮件的域名或者主机地址。

**mail函数**

**语法格式**

UTL\_SMTP.MAIL (

c IN OUT NOCOPY connection,

sender IN VARCHAR2,

parameters IN VARCHAR2 DEFAULT NULL);

**参数说明**

* c：open\_connection返回的smtp连接。
* sender：发送邮件的邮箱地址
* parameters：定义邮箱命令的附加参数，默认为NULL（接口预留，暂不实现）

**rcpt函数**

**语法格式**

UTL\_SMTP.RCPT (

c IN OUT NOCOPY connection,

recipient IN VARCHAR2,

parameters IN VARCHAR2 DEFAULT NULL);

**参数说明**

* c：open\_connection返回的smtp连接。
* recipient：接收电子邮件的地址。
* parameters：定义邮箱命令的附加参数，默认为NULL（接口预留，暂不实现）。

**open\_data函数**

**语法格式**

UTL\_SMTP.OPEN\_DATA (

c IN OUT NOCOPY connection);

**参数说明**

c：open\_connection返回的smtp连接。

**write\_data函数**

**语法格式**

UTL\_SMTP.WRITE\_DATA (

c IN OUT NOCOPY connection,

data IN VARCHAR2 CHARACTER SET ANY\_CS);

**参数说明**

* c：open\_connection返回的smtp连接。
* data：要发送消息文本，包括标题，采用RFC822格式。

**quit函数**

**语法格式**

UTL\_SMTP.QUIT (

c IN OUT NOCOPY connection);

**参数说明**

c：open\_connection返回的smtp连接。

**close\_data函数**

**语法格式**

UTL\_SMTP.CLOSE\_DATA (

c IN OUT NOCOPY connection);

**参数说明**

c：open\_connection返回的smtp连接。

## UTL\_URL

**功能描述**

UTL\_URL内置包用于对URL进行转义。

通常URL中会包含英文字母、数字和标点符号，这些字符称为非保留字符。URL中的任何其他字符，包括多字节字符和二进制八位字节码，都必须进行转义才能被web服务器或web浏览器准确处理。一些标点符号，例如美元符号 ( $)、问号 (？)、冒号 (:)和等号 ( =)) 被保留为 URL 中的分隔符，它们被称为保留字符，需要从字面上来处理这些字符，而不是将他们视为分隔符，因此这些字符必须进行转义。

* 非保留字符有：
* A\_Z, a\_z,0\_9
* 连字符 ( -)、下划线 ( \_)、句号 ( .)、感叹号 ( !)、波浪号 ( ~)、星号 ( \*)、重音 ( ')、左括号 ( ()、右括号 ( ))
* 保留字符有：分号 ( ;) 、斜线 ( /)、问号 ( ?)、冒号 ( :)、at 号 ( @)、和号 ( &)、等号 ( =)、加号 ( +)、美元符号 ( $)、百分号 ( %) 和逗号 ( ,)

该内置包包含以下函数：

|  |  |
| --- | --- |
| **函数** | **描述** |
| [ESCAPE](#URLESCAPE) | 对URL进行转义。 |
| [UNESCAPE](#URLUNESCAPE) | 对已转义的URL进行反转义。 |

**注意事项**

* 该功能仅在数据库兼容模式为Oracle时能够使用（即创建DB时DBCOMPATIBILITY='A'），在其他数据库兼容模式下不能使用该特性。
* 目前支持的字符集有gbk，gb2312，utf-8。
* 功能基于python3及插件plpython3u，需要有python3环境（目前测试通过的python版本是python3.7和python3.8），python3 编译时指定--enable-shared。
* 若需要启用tls安全认证，需要安装openssl及openssl-devel，且python3编译时指定。--with-ssl-default-suites=openssl，如安装完在python3命令行执行import ssl成功，则说明安装ssl模块成功。
* 例如：./configure --prefix=/usr/local/python3.8 --enable-shared --with-ssl-default-suites=openssl

**示例**

1、安装python3。

2、添加环境变量。

* PYTHONHOME：python3安装路径
* LD\_LIBRARY\_PATH：增加python3安装路径下的lib目录
* PATH：增加python3安装路径bin目录

3、创建并切换至兼容模式为Oracle的数据库db\_oracle。

CREATE DATABASE db\_oracle dbcompatibility='A';

\c db\_oracle

4、创建plpython3u扩展。

create extension plpython3u;

5、调用utl\_url.escape函数对URL进行转义。

select utl\_url.escape('https://www.baidu.com/参数/=+/',false,'gb2312');

6、调用utl\_url.unescape函数对URL进行反转义。

select utl\_url.unescape('https://www.baidu.com/%B2%CE%CA%FD/=+/','gb2312');

**ESCAPE函数**

**语法格式**

UTL\_URL.ESCAPE (

url IN VARCHAR2 CHARACTER SET ANY\_CS,

escape\_reserved\_chars IN BOOLEAN DEFAULT FALSE,

url\_charset IN VARCHAR2 DEFAULT utl\_http.body\_charset)

RETURN VARCHAR2;

**参数说明**

* url：原始URL。
* escape\_reserved\_chars：是否转义保留字符，true为转义，默认为false。
* url\_charset：定需转换的字符设置为目标字符集。默认值是UTL\_HTTP包的当前正文字符集，默认ISO-8859-1。

**UNESCAPE函数**

**语法格式**

UTL\_URL.UNESCAPE (

url IN VARCHAR2 CHARACTER SET ANY\_CS,

url\_charset IN VARCHAR2 DEFAULT utl\_http.body\_charset)

RETURN VARCHAR2;

**参数说明**

* url：需要反转义的URL。
* url\_charset：定需转换的字符设置为目标字符集。

# 视图

## oracle视图概述

关于对应视图的描述如下：

* ALL\_视图：显示当前用户可访问的信息。
* DBA\_视图：显示整个数据库中所有的信息。
* USER\_：当前用户所拥有的信息。

## 静态数据字典视图

### ALL\_ALL\_TABLES

ALL\_ALL\_TABLES视图提供有关当前用户可访问的表的信息。

| **列名** | **类型** | **描述** |
| --- | --- | --- |
| owner | name | 表的所有者 |
| table\_name | name | 表名 |
| tablespace\_name | name | 表所在的表空间 |
| cluster\_name | varchar(128) | 表所属的集群名称（如果有） |
| iot\_name | varchar(128) | 溢出或映射表条目所属的索引组织表的名称（如果有） |
| status | varchar(8) | 表示表的状态是可用（VALID）还是不可以用(UNUSABLE) |
| pct\_free | numeric | 块中可用空间的最小百分比；分区表为 NULL |
| pct\_used | numeric | 块中已用空间的最小百分比；分区表为 NULL |
| ini\_trans | numeric | 初始交易数量；分区表为 NULL |
| max\_trans | numeric | 最大交易数量；分区表为 NULL |
| initial\_extent | numeric | 初始范围的大小（以字节为单位）；分区表为 NULL |
| next\_extent | numeric | 辅助范围的大小（以字节为单位）；分区表为 NULL |
| min\_extents | numeric | 段中允许的最小范围数；分区表为 NULL |
| max\_extents | numeric | 段中允许的最大范围数；分区表为 NULL |
| pct\_increase | numeric | 范围大小的百分比增加；分区表为 NULL |
| freelists | numeric | 分配给段的进程空闲列表数；分区表为 NULL |
| freelist\_groups | numeric | 分配给段的空闲列表组数 |
| logging | varchar(7) | 是否记录对表的更改 |
| backed\_up | varchar(1) | 自上次修改后表是否已备份 ( Y) 或未 ( N) |
| num\_rows | numeric | 表中的行数 |
| blocks | doubleprecision | 表中使用的块数 |
| empty\_blocks | numeric | 表中的空（从未使用）块数 |
| avg\_space | numeric | 表中的平均可用空间 |
| chain\_cnt | numeric | 表中从一个数据块链接到另一个数据块或已迁移到新块的行数，需要一个链接来保留旧的 rowid。此列仅在您分析表后更新。 |
| avg\_row\_len | numeric | 平均行长度，包括行开销 |
| avg\_space\_freelist\_blocks | numeric | freelist 上所有块的平均可用空间 |
| num\_freelist\_blocks | numeric | freelist 上的块数 |
| degree | numeric | 每个实例用于扫描表的线程数 |
| instances | numeric | 要扫描表的实例数 |
| cache | varchar(5) | 表是否要缓存在缓冲区缓存中 ( Y) 或不 ( N) |
| table\_lock | varchar(8) | 指示表锁定是启用 ( ENABLED) 还是禁用 ( DISABLED) |
| sample\_size | numeric | 用于分析表格的样本量 |
| last\_analyzed | timestamp without time zone | 用于分析表格的样本量 |
| partitioned | character(3) | 指示表是否已分区 ( YES) 或未分区 ( NO) |
| iot\_type | varchar(12) | 如果表是索引组织表，IOT\_TYPE则为IOT、IOT\_OVERFLOW或IOT\_MAPPING。如果表不是索引组织表，IOT\_TYPE则为 NULL。 |
| object\_id\_type | varchar(16) | USER-DEFINED指示对象 ID ( OID) 是SYSTEM GENERATED |
| table\_type\_owner | varchar(128) | 如果是对象表，则创建表的类型的所有者 |
| table\_type | varchar(128) | 如果是对象表，则表的类型 |
| temporary | text | 是否为临时表，(Y）是，(N)否 |
| secondary | character(1) | 指示表是否是由ODCIIndexCreateOracle Data Cartridge 的方法创建的包含域索引内容的辅助对象 ( Y) 或不是 ( N) |
| nested | varchar(7) | 指示表是否为嵌套表 (YES) 或不是 ( NO) |
| buffer\_pool | varchar(7) | 用于表块的缓冲池 |
| flash\_cache | varchar(7) | 用于表块的数据库智能闪存缓存提示(仅限 Solaris 和 Oracle Linux 功能。) |
| cell\_flash\_cache | varchar(7) | 用于表块的单元闪存缓存提示 |
| row\_movement | varchar(8) | 果是分区表，则指示行移动是启用 ( ENABLED) 还是禁用 ( DISABLED) |
| global\_stats | varchar(3) | 统计信息被收集或增量维护，将是YES，否则它将是NO |
| user\_stats | varchar(3) | 统计信息是否由用户直接输入 (YES) 或不是 ( NO) |
| duration | varchar(15) | 临时表的持续时间 |
| skip\_corrupt | varchar(8) | Oracle 数据库在表和索引扫描期间是否忽略标记为损坏的块 ( ENABLED) 或引发错误 ( DISABLED)。要启用此功能，请运行该DBMS\_REPAIR.skip\_corrupt\_blocks过程。 |
| monitoring | varchar(3) | 表是否具有MONITORING属性集 (YES) 或没有 ( NO) |
| cluster\_owner | varchar(128 | 表所属集群的所有者（如果有） |
| dependencies | varchar(8) | 行级依赖性跟踪是启用 ( ENABLED) 还是禁用 ( DISABLED) |
| compression | varchar(8) | 否启用表压缩 ( ENABLED) 或不 ( DISABLED)；分区表为 NULL |
| compress\_for | varchar(30) |  |
| dropped | varchar | 是否已被删除并在回收站中（YES）或不在（NO）；分区表为 NULL。此视图不返回已删除的表的名称。 |
| segment\_created | varchar(3) | 表段是否已创建 (YES) 或未创建 ( NO) |
| inmemory | varchar(8) | 此表的内存中列存储（IM 列存储）是启用 ( ENABLED) 还是禁用 ( DISABLED) |
| inmemory\_priority | varchar(8) | 将此表填充到内存中列存储（IM 列存储）中的优先级 |
| inmemory\_distribute | varchar(15) | 表示表将如何分布在 Oracle Real Application Clusters (Oracle RAC) 环境中的 IM 列存储中 |
| inmemory\_compression | varchar(17) | 内存存储的压缩级别 |
| inmemory\_duplicate | varchar(13) | Oracle RAC 环境中 In-Memory Column Store（IM 列存储）的重复设置 |
| external | varchar(3) | 表是否为外部表 (YES) 或不是 ( NO) |
| cellmemory | varchar(24) | 存储单元闪存缓存中的列压缩值 |
| inmemory\_service | varchar(12) | 如何在各种实例上填充 IM 列存储 |
| inmemory\_service\_name | varchar(1000) | 应在其上填充 IM 列存储的服务的服务名称 |

### ALL\_CONS\_COLUMNS

ALL\_CONS\_COLUMNS视图提供有关当前用户可访问的表上约束指定列的信息。

| **列名** | **类型** | **描述** |
| --- | --- | --- |
| owner | text | 约束列的所有者 |
| schema\_name | varchar(64) | 约束所属的模式名 |
| constraint\_name | varchar(64) | 约束名称 |
| table\_name | varchar(64) | 定义约束的表名 |
| column\_name | varchar(64) | 定义约束的字段名 |
| position | smallint | 列在表中定义的位置 |
| constraint\_def | text | 约束定义 |

### ALL\_CONSTRAINTS

ALL\_CONSTRAINTS视图提供当前用户可访问表上的约束信息。

| **列名** | **类型** | **描述** |
| --- | --- | --- |
| owner | text | 约束定义的所有者 |
| constraint\_name | varchar(64) | 约束名字 |
| constraint\_type | text | c = 检查约束， f = 外键约束， p = 主键约束， u = 唯一约束， t = 约束触发器， x = 排他约束 |
| table\_name | varchar(64) | 表的名字 |
| search\_condition | text | 检查约束条件 |
| search\_condition\_vc | varchar(4000) | 检查约束的搜索条件文本。此列可能会截断搜索条件。 |
| r\_owner | text | 约束中引用表的所有者 |
| r\_constraint\_name | text | 被引用表的唯一约束定义的名称 |
| status | varchar(8) | 约束的执行状态：ENABLED或者DISABLED |
| delete\_rule | text | 外键删除动作代码： a = 无动作， r = 限制， c = 级联， n = 置空， d = 置为默认值 |
| deferrable | boolean | 该约束是否能被延迟 |
| deferred | boolean | 该约束是否默认被延迟 |
| validated | varchar(13) | 仅为支持兼容性。 |
| generated | varchar(14) | 约束的名称是用户生成的 ( USER NAME ) 还是系统生成的 ( GENERATED NAME ) |
| bad | varchar(3) | 此约束是否以不明确的方式 ( ) 指定世纪 BAD (NULL)。 |
| rely | varchar(4) | 当 VALIDATED = NOT VALIDATED 时,此列指示是否要在查询重写 时考虑约束；当 VALIDATED = VALIDATED ，此列没有意义 |
| last\_change | data | 上次启用或禁用约束的时间 |
| index\_owner | varchar(64) | 索引的所有者 |
| index\_name | varchar(64) | 索引的名字 |
| invalid | varchar(7) | 指示约束是否无效 ( INVALID ) 或无效 (NULL) |
| view\_related | varchar(14) | 指示约束是否依赖于视图 ( DEPEND ON VIEW ) 或不 (NULL) |
| orgin\_con\_id | varcahr(256) | 数据来源的容器的 ID |

### ALL\_IND\_COLUMNS

ALL\_IND\_COLUMNS视图提供当前用户可访问索引中包含的列信息。

| **列名** | **类型** | **描述** |
| --- | --- | --- |
| index\_owner | varchar(64) | 索引列的所有者 |
| index\_name | varchar(64) | 索引名称 |
| table\_owner | varchar(64) | 表的所有者 |
| table\_name | varchar(64) | 表名 |
| column\_name | name | 列名 |
| column\_position | smallint | 列的编号。一般列从1开始向上编号。系统列则拥有（任意）赋值编号 |
| column\_length | smallint | 对于一个固定尺寸的类型，typlen是该类型内部表示的字节数。对于一个变长类型，typlen为负值。-1表示一个“varlena”类型（具有长度字），-2表示一个以空值结尾的C字符串 |
| char\_length | numeric | 列的数据类型如果为char，表示字节数。否则为0 |
| descend | character(1) | 默认DESC。指示列是按降序（DESC）还是按升序（ASC）排序 |
| collated\_column\_id | numeric | 此列为其提供语言排序的列的内部序列号 |

### ALL\_IND\_PARTITIONS

ALL\_IND\_PARTITIONS视图提供当前用户可访问数据库中的索引分区的信息。

| **列名** | **类型** | **描述** |
| --- | --- | --- |
| index\_owner | varchar(128) | 索引分区所属分区表索引的所有者的名称 |
| index\_name | varchar(128) | 索引分区所属分区表索引的名称 |
| composite | varchar(3) | 索引分区的名称 |
| partition\_name | varchar(128) | 分区名称 |
| subpartition\_count | bigint | 如果是复合分区表上的本地索引，则分区中的子分区数 |
| high\_value | text | 索引分区所对应分区的上边界 |
| high\_value\_length | integer | 分区界限值表达式的长度 |
| partition\_position | numeric | 分区在索引中的位置 |
| status | varchar(8) | 指示索引分区是否可用 ( USABLE ) 或不可用 ( UNUSABLE ) |
| tablespace\_name | name | 索引分区的表空间名称 |
| pct\_free | numeric | 块中可用空间的最小百分比 |
| ini\_trans | numeric | 初始交易数量 |
| max\_trans | numeric | 最大交易数 |
| initial\_extent | numeric | 初始范围的大小（以字节为单位） |
| next\_extent | numeric | 扩展区的大小（以字节为单位） |
| min\_extent | numeric | 段中允许的最小范围数 |
| max\_extent | numeric | 段中允许的最大范围数 |
| max\_size | numeric | 段中允许的最大块数 |
| pct\_increase | numeric | 扩展区大小增加的百分比 |
| freelists | numeric | 此段中分配的进程空闲列表数 |
| freelist\_groups | numeric | 段中分配的进程空闲列表组数 |
| logging | varchar(7) | 是否记录对索引的更改 |
| compression | varchar(8) | 用于分区的压缩类型 |
| blevel | numeric | B\*-树级别（从根块到叶块的索引深度）。深度 0 表示根块和叶块相同。 |
| leaf\_blocks | numeric | 索引分区中的叶块数 |
| distinct\_keys | numeric | 索引分区中不同键的数量 |
| avg\_leaf\_blocks\_per\_key | numeric | 索引中每个不同值出现的平均叶块数，四舍五入到最接近的整数。对于强制 UNIQUE 和 PRIMARY KEY 约束的索引，此值始终为 1 。 |
| avg\_data\_blocks\_per\_key | numeric | 表中由索引中的不同值指向的数据块的平均数，舍入到最接近的整数。此统计数据是包含包含索引列的给定值的行的数据块的平均数量。 |
| clustering\_factor | numeric | 基于索引值的表中行的顺序量 |
| num\_rows | numeric | 返回的行数 |
| sample\_size | numeric | 用于分析此分区的样本大小 |
| last\_analyzed | timestamp without time zone | 最近分析此分区的日期 |
| buffer\_pool | varchar(7) | 分区的实际缓冲池 |
| flash\_cache | varchar(7) | 用于分区块的数据库智能闪存缓存提示 |
| cell\_flash\_cache | varchar(7) | 用于分区块的单元闪存缓存提示 |
| user\_stats | varchar(3) | 指示统计信息是否由用户直接输入 ( YES ) 或不是 ( NO ) |
| pct\_direct\_access | numeric | 如果索引组织表上的二级索引，则 VALID 猜测 行的百分比 |
| global\_stats | varchar(3) | GLOBAL\_STATS表示是否已收集统计信息，或者是否已从子分区汇总统计信息 或尚未收集统计信息 |
| domidx\_opstatus | varchar(6) | 索引的操作状态 |
| parameters | varchar(1000) | 对于域索引，参数字符串 |
| interval | varchar(3) | 指示分区是否在区间分区表的区间部分 ( YES ) 或分区是否在范围部分 ( NO ) |
| segment\_created | varchar(3) | 指示索引分区段是否已创建（ YES ）或未创建（ NO ）； N/A 表示这个索引是子分区的，在分区级别不存在段 |
| orphaned\_entries | varchar(3) | 指示全局索引是否由于在 DROP/TRUNCATE PARTITION 或 MODIFY PARTITION INDEXING OFF 操作期间延迟索引维护而包含陈旧条目。 |

### ALL\_INDEXES

ALL\_INDEXES视图提供当前用户可访问表上的索引信息。

| **列名** | **类型** | **描述** |
| --- | --- | --- |
| owner | varchar(64) | 索引的所有者 |
| index\_name | varchar(64) | 索引名字 |
| index\_type | varchar(27) | 索引类型，如btree类型 |
| table\_owner | text | 索引对象的所有者 |
| table\_name | varchar(64) | 索引对象的表名 |
| table\_type | varchar(11) | 表的类型 |
| uniqueness | text | 索引是唯一的（UNIQUE）还是非唯一的（NONUNIQUE） |
| compression | character(1) | 是否启用索引压缩 |
| prefix\_length | numeric | / |
| tablespace\_name | text | 索引所属的表空间名字 |
| ini\_trans | numeric | 初始交易数量 |
| max\_trans | numeric | 最大交易数 |
| initial\_extent | numeric | 初始范围的大小 |
| next\_extent | numeric | 次要扩展区的大小 |
| min\_extents | numeric | 段中允许的最小范围数 |
| max\_extents | numeric | 段中允许的最大范围数 |
| pct\_increase | numeric | 扩展区大小增加的百分比 |
| pct\_threshold | numeric | 每个索引条目允许的块空间的阈值百分比 |
| include\_column | numeric | 要包含在索引组织表主键（非溢出）索引中的最后一列的列 ID。此列映射到视图 的 COLUMN\_ID 列。 \*\_TAB\_COLUMNS |
| freelists | numeric | 分配给该段的进程空闲列表数 |
| freelist\_groups | numeric | 分配给该段的空闲列表组数 |
| pct\_free | numeric | 块中可用空间的最小百分比 |
| logging | text | 索引日志是否记录 |
| blevel | numeric | B\*-树级别（从根块到叶块的索引深度）。深度 0 表示根块和叶块相同。 |
| leaf\_blocks | numeric | 索引中的叶块数 |
| distinct\_keys | numeric | 不同索引值的数量。对于强制 UNIQUE 和 PRIMARY KEY 约束的索引，此值与表中的行数相同 ( \*\_TABLES.NUM\_ROWS ) |
| avg\_leaf\_blocks\_per\_key | numeric | 索引中每个不同值出现的平均叶块数，四舍五入到最接近的整数。对于强制 UNIQUE 和 PRIMARY KEY 约束的索引，此值始终为 1 。 |
| avg\_data\_blocks\_per\_key | numeric | 表中由索引中的不同值指向的数据块的平均数，舍入到最接近的整数。此统计数据是包含包含索引列的给定值的行的数据块的平均数量。 |
| clustering\_factor | numeric | 基于索引值的表中行的顺序量。 |
| status | text | 索引是否有效 |
| num\_rows | numeric | 索引中的行数。 |
| sample\_size | numeric | 用于分析指数的样本大小 |
| last\_analyzed | date | 最近分析该指数的日期 |
| degree | varchar(40) | 每个实例用于扫描索引的线程数 |
| instances | varchar(40) | 要扫描的索引的实例数 |
| partitioned | character(3) | 索引是否已分区 |
| temporary | character(1) | 索引是否在临时表上 |
| generated | varchar(1) | 索引的名称是系统生成的 ( Y ) 还是不是 ( N ) |
| secondary | character(1) | 仅为支持兼容性。该值始终为N |
| buffer\_pool | varchar(7) | 用于索引块的缓冲池 |
| flash\_cache | varchar(7) | 用于索引块的数据库智能闪存缓存提示 |
| cell\_flash\_cache | varchar(7) | 用于索引块的单元闪存缓存提示 |
| user\_stats | varchar(3) | 指示统计信息是否由用户直接输入 (YES) 或不是 ( NO) |
| duration | varchar(15) | 指示临时表的持续时间 |
| pct\_direct\_access | numeric | 对于索引组织表上的二级索引， VALID 猜测 行的百分比 |
| ityp\_owner | varchar(128) | 对于域索引，索引类型的所有者 |
| ityp\_name | varchar(128) | 对于域索引，索引类型的名称 |
| parameters | varchar(1000) | 对于域索引，参数字符串 |
| global\_stats | varchar(3) | GLOBAL\_STATS 如果统计信息被收集或增量维护，将 YES 是，否则它将是 NO |
| domidx\_status | varchar(12) | 域索引的状态： |
| domidx\_opstatus | varchar(6) | 对域索引的操作状态 |
| funcidx\_status | varchar(8) | 基于函数的索引的状态 |
| join\_index | character(3) | 仅为支持兼容性。该值始终为NO |
| iot\_redundant\_pkey\_elim | character(3) | 是否从索引组织表的二级索引中消除冗余主键列 (YES) 或不 ( NO) |
| dropped | character(3) | 仅为支持兼容性。该值始终为NO |
| visibility | varchar(9) | 索引是优化器 VISIBLE 还是 INVISIBLE 优化器 |
| domidx\_management | varchar(14) | 如果这是域索引，则指示域索引是系统管理的 ( SYSTEM\_MANAGED ) 还是用户管理的 ( USER\_MANAGED ) |
| segment\_created | varchar(3) | 指示索引段是否已创建 (YES) 或未 ( NO) |
| orphaned\_entries | varchar(3) | 指示全局索引是否由于在 DROP/TRUNCATE PARTITION 或 MODIFY PARTITION INDEXING OFF 操作期间延迟索引维护而包含陈旧条目 |
| indexing | varchar(7) | 指示全局索引是否与基础表分离。 |

### ALL\_OBJECTS

ALL\_OBJECTS视图提供当前用户可访问数据库中所有对象的信息。

| **列名** | **类型** | **描述** |
| --- | --- | --- |
| owner | name | 对象的所有者 |
| object\_name | name | 对象名称 |
| subobject\_name | varchar(128) | 子对象的名称（例如，分区） |
| object\_id | oid | 对象的字典对象编号 |
| data\_object\_id | numeric | 包含对象的段的字典对象编号。 |
| object\_type | name | 对象的类型（例如INDEX，FUNCTION，PROCEDURE，TABLE，INDEX） |
| created | timestamp with time zone | 创建对象的时间戳 |
| last\_ddl\_time | timestamp with time zone | 由 DDL 语句产生的对象和依赖对象的最后修改的时间戳（包括授权和撤销） |
| timestamp | varchar(19) | 对象规范的时间戳（字符数据） |
| status | varchar(7) | 仅为支持兼容性，对象的状态始终VALID |
| temporary | varchar(1) | 指示对象是否是临时对象（当前会话只能看到其自身放置在该对象中的数据）（Y）或（N） |
| generated | varchar(1) | 指示此对象的名称是否是系统生成的 ( Y ) 或不是 ( N ) |
| secondary | varchar(1) | 指示这是否是由 ODCIIndexCreate Oracle Data Cartridge |
| namespace | oid | 对象的命名空间 |
| edition\_name | varchar(128) | 所在的版本名称 |
| sharing | varchar(13) | 该字段的值可以是：DATA LINK：对象是数据链接的或数据链接到根中的对象METADATA LINK：对象是元数据链接或元数据链接到根中的对象EXTENED DATA LINK：对象是扩展数据链接或扩展数据链接到根中的对象NONE：以上均不适用 |
| editionable | varchar(1) | 该字段的值可以是：Y：标记为 EDITIONABLE 的对象N：标记为NONEEDITIONABLE 的对象NULL：在数据库中不可编辑的对象 |
| oracle\_maintained | varchar(1) | 表示对象是否由 Oracle 提供的脚本（例如 catalog.sql 或 catproc.sql）创建和维护。 |
| application | varchar(1) | 对象是否为应用程序公共对象 ( Y ) 或不是 ( N ) |
| default\_collation | varchar(100) | 对象的默认排序规则 |
| duplicated | varchar(1) | 此对象是否在此分片上重复 ( Y ) 或不 ( N ) |
| sharded | varchar(1) | 此对象是否已分片 ( Y ) 或不 ( N ) |
| created\_appid | numeric | 创建对象的应用程序 ID |
| created\_vsnid | numeric | 创建对象的应用程序版本的 ID |
| modified\_appid | numeric | 上次修改对象的应用程序 ID |
| modified\_vsnid | numeric | 上次修改对象的应用程序 ID |

### ALL\_PART\_INDEXES

ALL\_PART\_TABLES视图提供当前用户可访问数据库中的分区表上的索引信息。

| **列名** | **类型** | **描述** |
| --- | --- | --- |
| owner | varchar(128) | 分区表索引的所有者 |
| index\_name | varchar(128) | 分区表索引的名称 |
| table\_name | varchar(128) | 分区表索引所属的分区表名称 |
| partitioning\_type | varchar(9) | 分区表的分区策略 |
| subpartitioning\_type | varchar(9) | 复合分区分区策略 |
| partition\_count | bigint | 分区表索引的分区个数 |
| def\_subpartition\_count | bigint | 复合分区默认的子分区数量（如果指定的话） |
| partitioning\_key\_count | integer | 分区表的分区键个数 |
| subpartitioning\_key\_count | integer | 对于复合分区索引，子分区键中的列数 |
| locality | varchar(6) | 指示分区索引是本地（LOCAL）还是全局（GLOBAL） |
| alignment | varchar(12) | 指示分区索引是带前缀 ( PREFIXED) 还是不带前缀 ( NON\_PREFIXED) |
| def\_tablespace\_name | name | 分区表索引的表空间名称 |
| def\_pct\_free | numeric | 对于本地索引， PCTFREE 添加表分区时使用的默认值 |
| def\_ini\_trans | numeric | 对于本地索引， INITRANS 添加表分区时使用的默认值 |
| def\_max\_trans | numeric | 对于本地索引， MAXTRANS 添加表分区时使用的默认值 |
| def\_initial\_extent | varchar(40) | 对于本地索引，添加表分区或未指定值时使用的默认值（ INITIAL 在 Oracle 块中）DEFAULT INITIAL |
| def\_next\_extent | varchar(40) | 对于本地索引，添加表分区或未指定值时使用的默认值（ NEXT 在 Oracle 块中）DEFAULTNEXT |
| def\_min\_extents | varchar(40) | 对于本地索引，MINEXTENTS添加表分区时使用的默认值，或者DEFAULT如果没有MINEXTENTS指定值 |
| def\_max\_extents | varchar(40) | 对于本地索引， MAXEXTENTS 添加表分区时使用的默认值，或者DEFAULT如果没有 MAXEXTENTS 指定值 |
| def\_max\_size | varchar(40) | 对于本地索引， MAXSIZE 添加表分区时使用的默认值，或者DEFAULT如果没有 MAXSIZE 指定值 |
| def\_pct\_increase | varchar(40) | 对于本地索引， PCTINCREASE 添加表分区时使用的默认值，或者 DEFAULT 如果没有 PCTINCREASE 指定值 |
| def\_freelists | numeric | 对于本地索引， FREELISTS 添加表分区时使用的默认值 |
| def\_freelist\_groups | numeric | 对于本地索引， FREELIST GROUPS 添加表分区时使用的默认值 |
| def\_logging | varchar(7) | 对于本地索引，LOGGING添加表分区时要使用的默认属性 |
| def\_buffer\_pool | varchar(7) | 对于本地索引，添加表分区时要使用的默认缓冲池：DEFAULT,KEEP,RECYCLE,NULL |
| def\_flash\_cache | varchar(7) | 对于本地索引，添加表分区时要使用的默认数据库智能闪存缓存提示：DEFAULT,KEEP,NULL |
| def\_cell\_flash\_cache | varchar(7) | 对于本地索引，添加表分区时要使用的默认单元闪存缓存提示：DEFAULT,KEEP,NULL |
| def\_parameters | varchar(1000) | 域索引的默认参数字符串 |
| interval | varchar(1000) | 区间值字符串 |
| autolist | varchar(3) | 指示本地索引是否通过自动列表分区 (YES)分区 ( NO) |
| interval\_subpartition | varchar(1000) | 子分区间隔值的字符串 |
| autolist\_subpartition | varchar(3) | 指示本地索引是否通过自动列表分区 (YES)进行子分区 ( NO) |

### ALL\_PART\_KEY\_COLUMNS

ALL\_PART\_KEY\_COLUMNS视图提供当前用户可访问数据库中的分区表的分区键列的信息。

| **列名** | **类型** | **描述** |
| --- | --- | --- |
| owner | text | 分区表的所有者 |
| schema\_name | text | 分区表所属的模式名 |
| name | text | 分区名称 |
| object\_type | character(5) | 仅为支持兼容性。该值始终为table |
| column\_name | text | 字段名 |
| column\_position | integer | 分区键中列的位置 |

### ALL\_PART\_TABLES

ALL\_PART\_TABLES视图提供当前用户可访问数据库中的分区表信息。

| **列名** | **类型** | **描述** |
| --- | --- | --- |
| owner | text | 分区表的所有者 |
| table\_name | text | 分区表名 |
| partitioning\_type | text | 分区类型： range、list、hash |
| subpartitioning\_type | text | 子分区类型：range、list、hash |
| partition\_count | bigint | 分区表中的分区数 |
| def\_subpartition\_count | bigint | 复合分区表中，子分区数 |
| partitioning\_key\_count | integer | 分区键的列数 |
| subpartitioning\_key\_count | integer | 复合分区表中，子分区中的列数 |
| status | varchar(8) | 仅为支持兼容性。该值始终为VALID |
| def\_tablespace\_name | varchar(30) | 仅为支持兼容性。该值始终为NULL |
| def\_pct\_free | numeric | 仅为支持兼容性。该值始终为NULL |
| def\_pct\_used | numeric | 仅为支持兼容性。该值始终为NULL |
| def\_ini\_trans | numeric | 仅为支持兼容性。该值始终为NULL |
| def\_max\_trans | numeric | 仅为支持兼容性。该值始终为NULL |
| def\_initial\_extent | varchar(40) | 仅为支持兼容性。该值始终为NULL |
| def\_next\_extent | varchar(40) | 仅为支持兼容性。该值始终为NULL |
| def\_min\_extents | varchar(40) | 仅为支持兼容性。该值始终为NULL |
| def\_max\_extents | varchar(40) | 仅为支持兼容性。该值始终为NULL |
| def\_max\_size | varchar(40) | 仅为支持兼容性。该值始终为NULL |
| def\_pct\_increase | varchar(40) | 仅为支持兼容性。该值始终为NULL |
| def\_freelists | numeric | 仅为支持兼容性。该值始终为NULL |
| def\_freelist\_groups | numeric | 仅为支持兼容性。该值始终为NULL |
| def\_logging | varchar(7) | 仅为支持兼容性。该值始终为YES |
| def\_compression | varchar(8) | 仅为支持兼容性。该值始终为None |
| def\_compress\_for | varchar(30) | 添加分区时使用哪种操作的默认压缩 |
| def\_buffer\_pool | varchar(7) | 仅为支持兼容性。该值始终为DEFAULT |
| def\_flash\_cache | varchar(7) | 仅为支持兼容性。该值始终为NULL |
| def\_cell\_flash\_cache | varchar(7) | 仅为支持兼容性。该值始终为NULL |
| ref\_ptn\_constraint\_name | varchar(30) | 仅为支持兼容性。该值始终为NULL |
| interval | varchar(1000) | 区间值字符串 |
| autolist | varchar(1000) | 指示表是否通过自动列表分区 ( YES )分区 ( NO ) |
| interval\_subpartition | varchar(1000) | 子分区间隔值的字符串 |
| autolist\_subpartition | varchar(1000) | 指示此子 分区是否使用 ( YES )自动列表分区 (NO ) |
| is\_nested | varchar(1000) | 指示分区表是否为嵌套表 ( YES ) 或不是 ( NO ) |
| def\_segment\_creation | varchar(1000) | 指定是否在表级别指定了段创建的默认值 |
| def\_indexing | varchar(1000) | 指示为表指定的索引属性。 |
| def\_inmemory | varchar(1000) | 表示内存列存储是否默认启用（ENABLE）,禁用（DISABLED）,或者未指定 (NONE) 此表中的分区 |
| def\_inmemory\_priority | varchar(1000) | 内存列存储填充的默认优先级 |
| def\_inmemory\_distribute | varchar(1000) | Oracle RAC)环境中默认情况下如何为表的分区分配 内存列存储 |
| def\_inmemory\_compression | varchar(1000) | 内存列存储的默认压缩级别 |
| def\_inmemory\_duplicate | varchar(1000) | Oracle RAC 环境中内存列存储的默认重复设置 |
| def\_read\_only | varchar(1000) | 新分区的默认设置：YES(只读)，NO（读/写） |
| def\_cellmemory | varchar(1000) | 显示父表中的新分区将继承的属性的默认值， CELLMEMORY 除非该行为被显式覆盖 |
| def\_inmemory\_service | varchar(1000) | 默认情况下如何在各种实例上为表的分区填充内存列存储 |
| def\_inmemory\_service\_name | varchar(1000) | 默认情况下应为表的分区填充内存列存储的服务的服务名称。 |

### ALL\_SEQUENCES

ALL\_SEQUENCES视图提供当前用户可访问数据库用户定义的序列信息。

| **列名** | **类型** | **描述** |
| --- | --- | --- |
| sequence\_owner | name | 序列的所有者 |
| sequence\_name | name | 序列名字 |
| min\_value | int16 | 序列的最小值 |
| max\_value | int16 | 序列的最大值 |
| increment\_by | int16 | 序列递增值 |
| cycle\_flag | character(1) | 序列达到最大值是否循环 |
| cache\_size | numeric | 在内存中预分配的序列号数目 |
| last\_number | numeric | 上次保存到磁盘的序列号的值 |
| scale\_flag | varchar(1) | 指示这是否是可伸缩序列 ( Y) 或不是 ( N) |
| extend\_flag | varchar(1) | 指示此可伸缩序列的生成值是否超出MAX\_VALUE或MIN\_VALUE( Y) 或不 ( N) |
| session\_flag | varchar(1) | 指示序列值是会话私有 ( Y) 还是不是 ( N) |
| keep\_value | varchar(1) | 指示在失败后重放期间是否保留序列值 ( Y) 或不 ( N) |

### ALL\_SUBPART\_KEY\_COLUMNS

ALL\_SUBPART\_KEY\_COLUMNS视图提供当前用户可访问数据库分区表的子分区的关键列的信息。

| **列名** | **类型** | **描述** |
| --- | --- | --- |
| owner | text | 分区表的所有者 |
| schema\_name | text | 分区表所属模式名 |
| name | text | 字段所在的表名 |
| object\_type | character(5) | 仅为支持兼容性。该值始终为TABLE |
| column\_name | text | 分区键的列名 |
| column\_position | integer | 列所在分区键的位置 |

### ALL\_TAB\_COLS

ALL\_TAB\_COLS记录了用户所能访问的所有表字段相关的信息。

|  |  |  |
| --- | --- | --- |
| **列名** | **类型** | **描述** |
| owner | name | 表、视图或者集群的所有者 |
| table\_name | name | 表、视图或集群的名称 |
| column\_name | name | 列名 |
| data\_type | text | 列的数据类型 |
| data\_type\_mod | text | 列的数据类型修饰符 |
| data\_type\_owner | name | 列的数据类型所有者 |
| data\_length | integer | 列长度（以字节为单位） |
| data\_precision | integer | NUMBER数据类型的十进制精度  FLOAT数据类型的二进制精度  所有其他数据类型为NULL |
| data\_scale | integer | 数字中小数点右侧的数字 |
| nullable | text | 指示列是否允许NULL。值是指列上N是否存在NOT NULL约束，或者列是否为a PRIMARY KEY的一部分 |
| column\_id | smallint | 创建列的序号 |
| default\_length | text | 列的默认长度 |
| data\_default | text | 列的默认值 |
| num\_distinct | real | 列中的不同值的数量 |
| low\_value | text | 列中低的值 |
| high\_value | text | 列中高的值 |
| density | double precision | 如果柱状图在COLUMN\_NAME上可用，则此列显示值的选择性，该值跨越柱状图中少于2个端点。它不代表跨越2个或多个端点的值的选择性。 |
| num\_nulls | text | 列中NULL的数量 |
| num\_buckets | text | 列的直方图中的桶的数量 |
| last\_analyzed | text | 分析此列的最近的日期 |
| SAMPLE\_SIZE | text | 用于分析此列的样本量 |
| CHARACTER\_SET\_NAME | text | 字符集的名称（CHAR\_CS，NCHAR\_CS） |
| char\_col\_decl\_length | integer | 字符类型列的声明长度 |
| global\_stats | text | 如果收集统计数据，或者进行增量维护，则GLOBAL\_STATS为YES，否则将为NO |
| user\_stats | text | 指示统计信息是否由用户直接输入（YES）或（NO） |
| avg\_col\_len | integer | 列的平均长度（以字节为单位） |
| CHAR\_LENGTH | integer | 以字符的形式显示列的长度，使用的数据类型：CHAR、VARCHAR2、NCHAR、NVARCHAR2 |
| char\_used | text | 表示列使用字节长度语义（B）或字符长度语义（C），或者数据类型是否不是以下任何一种（NULL）：CHAR、VARCHAR2、NCHAR、NVARCHAR2 |
| v80\_fmt\_image | text | 指示列数据是否为8.0版图像格式（YES）或否（NO） |
| data\_upgraded | text | 指示列数据是否已升级为最新类型版本格式（是）或未升级（否） |
| hidden\_column | text | 指示列是否是隐藏列，是（YES）或者否（NO） |
| virtual\_column | text | 指示列是否是虚拟列，是（YES）或者否（NO） |
| segment\_column\_id | text | 列的序列号 |
| internal\_column\_id | smallint | 列的内部序列号 |
| histogram | text | 表示直方图存在的类型：NONE、FREQUENCY、TOP-FREQUENCY、HEIGHT BALANCED、HYBRID |
| qualified\_col\_name | name | 限定列的名称 |

### ALL\_TAB\_COLUMNS

ALL\_TAB\_COLUMNS视图提供当前用户可访问表和视图中所有列的信息。

| **列名** | **类型** | **描述** |
| --- | --- | --- |
| owner | varchar(64) | 字段所在表或视图的所有者 |
| table\_name | varchar(64) | 表名 |
| column\_name | varchar(64) | 列名 |
| data\_type | varchar(128) | 列的数据类型 |
| data\_type\_mod | varchar(3) | 列的数据类型修饰符 |
| data\_type\_owner | varchar(64) | 列数据类型的所有者 |
| data\_length | integer | 列的长度（以字节为单位） |
| data\_precision | integer | 数据类型的精度 |
| data\_scale | integer | 数字小数点右边精度 |
| nullable | bpchar | 列值是否允许为空 |
| column\_id | integer | 列在表中的顺序编号 |
| default\_length | numeric | 列的默认值的长度 |
| data\_default | text | 列的默认值 |
| num\_distinct | numeric | 列中不同值的数量。保留此列是为了与 Oracle7 向后兼容。此信息现在在 {TAB |
| low\_value | raw | 列中的低值。保留此列是为了与 Oracle7 向后兼容。此信息现在在 {TAB |
| high\_value | raw | 列中的高值。保留此列是为了与 Oracle7 向后兼容。此信息现在在 {TAB |
| density | numeric | / |
| num\_nulls | numeric | 列中的 NULL 数 |
| num\_buckets | numeric | 列的直方图中的桶数 |
| last\_analyzed | timestamp without time zone | 最近分析此列的日期 |
| sample\_size | numeric | 用于分析此列的样本量 |
| character\_set\_name | varchar(44) | 字符集名称 |
| char\_col\_decl\_length | numeric | 字符类型列的声明长度 |
| global\_stats | varchar(3) | GLOBAL\_STATS 如果统计信息被收集或增量维护，将 YES 是，否则它将是 NO |
| user\_stats | varchar(3) | 指示统计信息是否由用户直接输入 ( YES ) 或不是 ( NO ) |
| avg\_col\_len | numeric | 列的平均长度（以字节为单位） |
| char\_length | numeric | 以字符显示列的长度。 |
| char\_used | varchar(1) | 指示该列使用 BYTE 长度语义 ( B ) 或 CHAR 长度语义 ( C )，或者数据类型是否不是以下任何一种 (NULL)：CHAR,VARCHAR2,NCGAR,NVACHAR2 |
| v80\_fmt\_image | varchar(3) | 指示列数据是否为 8.0 版图像格式 ( YES ) 或不是 ( NO ) |
| data\_upgraded | varchar(3) | 指示列数据是否已升级为最新类型版本格式（ YES ）或未升级（ NO ） |
| histogram | varchar(15) | 直方图类型 |
| default\_on\_null | varchar(3) | 指示列是否具有 DEFAULT ON NULL 语义 ( YES ) 或没有 ( NO ) |
| identity\_column | varchar(3) | 指示这是否是标识列 ( YES ) 或不是 ( NO ) |
| evaluation\_edition | varchar(128) | 解析表达式列中引用的编辑对象的版本名称 |
| unusable\_before | varchar(128) | 列可用的最旧版本的名称 |
| unusable\_beginning | varchar(128) | 列永久不可用的最旧版本的名称 |
| collation | varchar(100) | 列的排序规则。仅适用于具有字符数据类型的列。 |

### ALL\_TAB\_PARTITIONS

ALL\_TAB\_PARTITIONS视图提供当前用户可访问数据库中的所有分区的信息。

| **列名** | **类型** | **描述** |
| --- | --- | --- |
| table\_owner | varchar(128) | 分区表的+.所有者 |
| table\_name | varchar(128) | 分区表的表名 |
| composite | text | 指示表是复合分区 ( YES ) 还是不是 ( NO ) |
| partition\_name | varchar(128) | 分区名称 |
| subpartition\_count | bigint | 如果是符合分区，分区中的子分区数量 |
| high\_value | text | 对应分区范围最大值 |
| high\_value\_length | integer | 对应分区范围最大值的字节长度 |
| partition\_position | numeric | 分区在表中的位置 |
| tablespace\_name | text | 分区所在表空间名 |
| pct\_free | numeric | 仅为支持兼容性 |
| pct\_used | numeric | 仅为支持兼容性 |
| ini\_trans | numeric | 仅为支持兼容性 |
| max\_trans | numeric | 仅为支持兼容性 |
| initial\_extent | numeric | 仅为支持兼容性 |
| next\_extent | numeric | 仅为支持兼容性 |
| min\_extent | numeric | 仅为支持兼容性 |
| max\_extent | numeric | 仅为支持兼容性 |
| max\_size | numeric | 仅为支持兼容性 |
| pct\_increase | numeric | 仅为支持兼容性 |
| freelists | numeric | 仅为支持兼容性 |
| freelist\_groups | numeric | 仅为支持兼容性 |
| logging | varchar(7) | 是否开启表的logging |
| compression | varchar(8) | 分区是否开启压缩 |
| compress\_for | varchar(30) | 默认压缩方式 |
| num\_rows | numeric | 分区中的数据行数 |
| blocks | double precision | 分区中数据块数 |
| empty\_blocks | numeric | 仅为支持兼容性 |
| avg\_space | numeric | 仅为支持兼容性 |
| chain\_cnt | numeric | 仅为支持兼容性 |
| avg\_row\_len | numeric | 仅为支持兼容性 |
| sample\_size | numeric | 仅为支持兼容性 |
| last\_analyzed | timestamp without time zone | 仅为支持兼容性 |
| buffer\_pool | varchar(7) | 仅为支持兼容性 |
| flash\_cache | varchar(7) | 用于分区块的数据库智能闪存缓存提示 |
| cell\_flash\_cache | varchar(7) | 用于分区块的单元闪存缓存提示 |
| global\_stats | varchar(3) | 仅为支持兼容性 |
| user\_stats | varchar(3) | 仅为支持兼容性 |
| is\_nested | varchar(7) | 指示这是否是嵌套表分区 ( YES ) 或不是 ( NO ) |
| parent\_table\_partition | varchar(7) | 父表对应的分区 |
| interval | varchar(3) | 指示分区是否在区间分区表的区间部分 ( YES ) 或分区是否在范围部分 ( NO ) |
| segment\_created | varchar(4) | 指示简单分区表的分区的实际段创建属性，或复合分区表的子分区的默认值（如果指定），否则 NONE |
| indexing | varchar(4) | 简单分区表的分区的实际索引属性，或复合分区表的子分区的默认值（如果指定），否则为 NONE |
| read\_only | varchar(4) | 分区的默认设置：YES(只读)，NO(读/写)，NONE（没有指定默认设置） |
| inmemory | varchar(4) | 是否为此分区启动内存列存储，ENABLED(启动)，DISABLED（未启动） |
| inmemory\_priority | varchar(4) | 内存列存储填充的优先级 |
| inmemory\_distribute | varchar(4) | 内存列存储在RAC环境中的分布方式 |
| inmemory\_compression | varchar(4) | 内存列存储的压缩级别 |
| inmemory\_duplicate | varchar(4) | oracle RAC中的内存列存储的重复设置 |
| cellmemory | varchar(24) | 存储单元闪存缓存中的列压缩值 |
| inmemory\_service | varchar(12) | 如何在各种实例上填充内存列存储 |
| inmemory\_service\_name | varchar(100) | 填充的内存列存储的服务名称 |

### ALL\_TAB\_SUBPARTITIONS

ALL\_TAB\_SUBPARTITIONS视图提供当前用户可访问数据库中的所有子分区的信息。

| **列名** | **类型** | **描述** |
| --- | --- | --- |
| table\_owner | text | 子分区表的所有者 |
| table\_name | text | 表名 |
| partition\_name | text | 分区名 |
| subpartition\_name | text | 子分区名字 |
| high\_value | text | 子分区范围最大值 |
| high\_value\_length | integer | 子分区范围最大值的字节长度 |
| partition\_position | numeric | 分区在表中的位置 |
| subpartition\_position | numeric | 子分区字段所在表中的位置 |
| tablespace\_name | text | 子分区所在表空间名 |
| pct\_free | numeric | 仅为支持兼容性，该值始终为0 |
| pct\_used | numeric | 仅为支持兼容性，该值始终为0 |
| ini\_trans | numeric | 仅为支持兼容性，该值始终为0 |
| max\_trans | numeric | 仅为支持兼容性，该值始终为0 |
| initial\_extent | numeric | 仅为支持兼容性，该值始终为null |
| next\_extent | numeric | 仅为支持兼容性，该值始终为null |
| min\_extent | numeric | 仅为支持兼容性，该值始终为0 |
| max\_extent | numeric | 仅为支持兼容性，该值始终为0 |
| max\_size | numeric | 段中允许的最大块数 |
| pct\_increase | numeric | 仅为支持兼容性，该值始终为0 |
| freelists | numeric | 仅为支持兼容性，该值始终为null |
| freelist\_groups | numeric | 仅为支持兼容性，该值始终为null |
| logging | varchar(7) | 仅为支持兼容性，该值始终为YES |
| compression | varchar(8) | 仅为支持兼容性，该值始终为NONE |
| compress\_for | varchar(30) | 操作的默认压缩方式 |
| num\_rows | numeric | 子分区的数据量 |
| blocks | double precision | 子分区的数据块数 |
| empty\_blocks | numeric | 仅为支持兼容性，该值始终为null |
| avg\_space | numeric | 仅为支持兼容性，该值始终为null |
| chain\_cnt | numeric | 仅为支持兼容性，该值始终为null |
| avg\_row\_len | numeric | 仅为支持兼容性，该值始终为null |
| sample\_size | numeric | 仅为支持兼容性，该值始终为null |
| last\_analyzed | timestamp without time zone | 仅为支持兼容性，该值始终为null |
| buffer\_pool | varchar(7) | 仅为支持兼容性，该值始终为null |
| flash\_cache | varchar(7) | 用于子分区块的数据库智能闪存缓存提示 |
| cell\_flash\_cache | varchar(7) | 用于子分区块的单元闪存缓存提示 |
| global\_stats | varchar(3) | 仅为支持兼容性，该值始终为YES |
| user\_stats | varchar(3) | 仅为支持兼容性，该值始终为NO |
| interval | varchar(3) | 指示分区是否在区间分区表的区间部分 ( YES ) 或分区是否在范围部分 ( NO ) |
| segment\_created | varchar(4) | 指示表子分区段是否已创建（ YES ）或未创建（ NO ）； N/A 表示该表没有子分区 |
| indexing | varchar(4) | 指示索引属性。ON(子分区索引开启)，OFF(子分区索引关闭) |
| read\_only | varchar(4) | 指示子分区是只读的还是读/写的：YES(只读)，NO(读/写) |
| inmemory | varchar(4) | 是否为此子分区启动内存列存储，ENABLED(启动)，DISABLED（未启动） |
| inmemory\_priority | varchar(4) | 内存列存储填充的优先级 |
| inmemory\_distribute | varchar(4) | 内存列存储在oracle RAC中的分布方式 |
| inmemory\_compression | varchar(4) | 内存列存储的压缩级别 |
| inmemory\_duplicate | varchar(4) | oracle RAC环境中的内存列存储的重复方式设置 |
| inmemory\_service | varchar(12) | 如何在各种实例上填充内存列存储 |
| inmemory\_service\_name | varchar(100) | 内存列存储的服务名称 |
| cellmemory | varchar(24) | 存储单元闪存缓存中的列压缩值 |

### ALL\_TABLES

ALL\_TABLES视图提供当前用户可访问数据库用户定义表的信息。

| **列名** | **类型** | **描述** |
| --- | --- | --- |
| owner | text | 表的所有者 |
| table\_name | text | 表名 |
| tablespace\_name | text | 表所在的表空间名称；分区表，临时表和索引组织表为NULL |
| cluster\_name | varchar(128) | 表所属的集群名称（如果有） |
| iot\_name | varchar(128) | 溢出或映射表条目所属的索引组织表的名称（如果有）。如果IOT\_TYPE列不为 NULL，则该列包含基表名称。 |
| status | varchar(5) | 如果之前DROP TABLE操作失败，则表示该表是不可用的（UNUSABLE）或者有效（VALID） |
| pct\_free | numeric | 块中可用空间的最小百分比；分区表为 NULL |
| pct\_used | numeric | 块中已用空间的最小百分比；分区表为 NULL |
| ini\_trans | numeric | 初始交易数量；分区表为 NULL |
| max\_trans | numeric | 最大交易数量；分区表为 NULL |
| initial\_extent | numeric | 初始范围的大小（以字节为单位）；分区表为 NULL |
| next\_extent | numeric | 辅助范围的大小（以字节为单位）；分区表为 NULL |
| min\_extents | numeric | 段中允许的最小范围数；分区表为 NULL |
| max\_extents | numeric | 段中允许的最大范围数；分区表为 NULL |
| pct\_increase | numeric | 范围大小的百分比增加；分区表为 NULL |
| freelists | numeric | 分配给段的进程空闲列表数；分区表为 NULL |
| freelist\_groups | numeric | 分配给段的空闲列表组数；分区表为 NULL |
| logging | varchar(7) | 是否记录对表的更改；分区表为 NULL |
| backed\_up | varchar(1) | 自上次修改后表是否已备份 ( Y) 或未 ( N) |
| num\_rows | numeric | 表中的行数 |
| blocks | double precision | 表中使用的数据块数 |
| empty\_blocks | numeric | 表中空（从未使用过）数据块的数量。 |
| avg\_space | numeric | 分配给表的数据块中的平均可用空间量（以字节为单位） |
| chain\_cnt | numeric | 表中从一个数据块链接到另一个数据块或已迁移到新块的行数，需要链接以保留旧 ROWID |
| avg\_row\_len | numeric | 表中行的平均长度（以字节为单位） |
| avg\_space\_freelist\_blocks | numeric | freelist 上所有块的平均可用空间 |
| num\_freelist\_blocks | numeric | freelist 上的块数 |
| degree | numeric | 每个实例用于扫描表的线程数，或DEFAULT |
| instances | numeric | 要扫描表的实例数，或DEFAULT |
| cache | varchar(5) | 是否要缓存在缓冲区缓存中 ( Y ) 或不 ( N ) |
| table\_lock | varchar(8) | 表锁定是启用 ( ENABLED) 还是禁用 ( DISABLED) |
| sample\_size | numeric | 用于分析表格的样本量 |
| last\_analyzed | timestamp without time zone | 最近分析表格的日期 |
| partitioned | character(3) | 指示表是否已分区 ( YES ) 或未分区 ( NO ) |
| iot\_type | varchar(12) | 如果表是索引组织表， IOT\_TYPE 则为 IOT 、 IOT\_OVERFLOW 或 IOT\_MAPPING 。如果表不是索引组织表， IOT\_TYPE 则为 NULL。 |
| temporary | character(1) | 是否为临时表: 是(Y)、否(N) |
| secondary | character(1) | 该表是否是由 ODCIIndexCreate Oracle Data Cartridge ( Y ) 的方法创建的辅助对象 ( N ) |
| nested | varchar(3) | 表是否为嵌套表 ( YES ) 或不是 ( NO ) |
| buffer\_pool | varchar(7) | 表的缓冲池 |
| flash\_cache | varchar(7) | 表块的数据库智能闪存缓存提示（仅限 Solaris 和 Oracle Linux 功能。） |
| cell\_flash\_cache | varchar(7) | 表块的单元闪存缓存提示 |
| row\_movement | varchar(8) | 分区行移动是启用 ( ENABLED ) 还是禁用 ( DISABLED ) |
| global\_stats | varchar(3) | GLOBAL\_STATS 如果统计信息被收集或增量维护，将 YES 是，否则它将是 NO |
| user\_stats | varchar(3) | 统计信息是否由用户直接输入 ( YES ) 或不是 ( NO ) |
| duration | varchar(15) | 指示临时表的持续时间： SYS$SESSION - 在会话期间保留行 SYS$TRANSACTION - 行被删除后 COMMIT 。Null - 永久表 |
| skip\_corrupt | varchar(8) | 指示 Oracle 数据库在表和索引扫描期间是否忽略标记为损坏的块 ( ENABLED ) 或引发错误 ( DISABLED )。要启用此功能，请运行 DBMS\_REPAIR . SKIP\_CORRUPT\_BLOCKS 程序。 |
| monitoring | varchar(3) | 此列已过时 |
| cluster\_owner | varchar(128) | 表所属集群的所有者（如果有） |
| dependencies | varchar(8) | 指示行级依赖性跟踪是启用 ( ENABLED ) 还是禁用 ( DISABLED ) |
| compression | varchar(8) | 指示是否启用表压缩 ( ENABLED ) 或不 ( DISABLED )；分区表为 NULL |
| compress\_for | varchar(30) | 默认压缩方式 |
| dropped | varchar | 指示表是否已被删除并在回收站中（ YES ）或不在（ NO ）；分区表为 NULL。此视图不返回已删除的表的名称。 |
| read\_only | varchar(3) | 表段是否 READ-ONLY 存在。可能的值：YES(表段是READ\_ONLY)；NO(表段不是READ\_ONLY)；N/A(不适用。此值出现在分区表中，其中没有与逻辑表对象相关的段。)； |
| segment\_created | varchar(3) | 指示是否创建表段。可能的值：YES(表段已创建)；NO(表段未创建)；N/A(不适用。此值出现在分区表中，其中没有与逻辑表对象相关的段。)； |
| result\_cache | varchar(7) | 表的结果缓存模式注释 |
| clustering | varchar(3) | 表是否有属性聚类子句（ YES ）或没有（ NO ） |
| activity\_tracking | varchar(23) | 是否在表上启用热图跟踪 |
| dml\_timestamp | varchar(25) | 自动数据优化的修改时间、创建时间或两者 |
| has\_identity | varchar(3) | 表是否有标识列 ( YES ) 或没有 ( NO ) |
| container\_data | varchar(3) | 表是否包含特定于容器的数据 |
| inmemory | varchar(8) | 是否为此子分区启动内存列存储，ENABLED(启动)，DISABLED（未启动） |
| inmemory\_priority | varchar(8) | 内存列存储填充的优先级 |
| inmemory\_distribute | varchar(15) | 内存列存储在oracle RAC中的分布方式 |
| inmemory\_compression | varchar(17) | 内存列存储的压缩级别 |
| inmemory\_duplicate | varchar(13) | oracle RAC中的内存列存储的重复设置 |
| default\_collation | varchar(100) | 表的默认排序规则 |
| duplicated | varchar(1) | 表是否在此分片上重复 ( Y ) 或不 ( N ) |
| sharded | varchar(1) | 表是分片 ( Y ) 还是不分片 ( N ) |
| external | varchar(3) | 表是否为外部表 ( YES ) 或不是 ( NO ) |
| cellmemory | varchar(24) | 存储单元闪存缓存中的列压缩值 |
| containers\_default | varchar(3) | 默认情况下是否启用表 CONTAINERS() ( YES ) 或不启用 ( NO ) |
| container\_map | varchar(3) | 指示是否启用表以与 container\_map 数据库属性 ( YES ) 一起使用 ( NO ) |
| extended\_data\_link | varchar(3) | 指示是否启用表以从根 ( YES ) 获取扩展数据链接 ( NO ) |
| extended\_data\_link\_map | varchar(3) | 仅为支持兼容性 |
| inmemory\_service | varchar(12) | 如何在各种实例上填充内存列存储 |
| inmemory\_service\_name | varchar(1000) | 内存列存储的服务名称 |
| container\_map\_object | varchar(3) | 指示是否将表用作 container\_map 数据库属性的值 ( YES ) 或不 ( NO ) |

### ALL\_TRIGGERS

ALL\_TRIGGERS视图提供当前用户可访问数据库表上的触发器信息。

| **列名** | **类型** | **描述** |
| --- | --- | --- |
| owner | text | 触发器的所有者 |
| trigger\_name | text | 触发器名称 |
| trigger\_type | text | 触发类型:before/after |
| triggering\_event | text | 触发事件:insert/update/delete/truncate |
| table\_owner | text | 表的所有者 |
| base\_object\_type | text | 触发对象：表 |
| table\_name | text | 表名 |
| column\_name | varchar(7) | 嵌套表列的名称（如果是嵌套表触发器），否则为 NULL |
| referencing\_names | text | 仅为支持兼容性 |
| when\_clause | varchar(4000) | 仅为支持兼容性 |
| status | text | 指示触发器是启用 ( ENABLED ) 还是禁用 ( DISABLED )；禁用的触发器不会触发 |
| description | text | 仅为支持兼容性 |
| action\_type | varchar(11) | 触发体的动作类型：（CALL，PL/SQL） |
| trigger\_body | text | 触发器定义 |
| crossedition | varchar(7) | 跨版触发器类型（FORWARD,） |
| before\_statement | varchar(3) | 触发器是否有 BEFORE STATEMENT 部分 ( YES ) 或没有 ( NO ) |
| before\_row | varchar(3) | 触发器是否有 BEFORE EACH ROW 部分 ( YES ) 或没有 ( NO ) |
| after\_row | varchar(3) | 触发器是否有 AFTER EACH ROW 部分 ( YES ) 或没有 ( NO ) |
| after\_statement | varchar(3) | 触发器是否有 AFTER STATEMENT 部分 ( YES ) 或没有 ( NO ) |
| instead\_of\_row | varchar(3) | 触发器是否有 INSTEAD OF 部分 ( YES ) 或没有 ( NO ) |
| fire\_once | varchar(3) | 触发器是否仅对进行更改的用户进程触发 ( YES ) 或者触发器是否也将针对复制应用或 SQL 应用进程触发 ( NO ) |
| apply\_server\_only | varchar(3) | 触发器是否仅针对复制应用或 SQL 应用进程 ( YES ) 或不 ( NO ) 触发。如果设置为 YES ，则 的设置 FIRE\_ONCE 无关紧要。 |

### ALL\_TYPES

ALL\_TYPES视图提供当前用户可访问数据库的对象类型信息。

| **列名** | **类型** | **描述** |
| --- | --- | --- |
| owner | text | 类型的所有者 |
| type\_name | text | 类型的名称 |
| type\_oid | oid | 类型的编号 |
| typecode | text | 类型代码，可能值：object、collection、other |
| attributes | integer | 类型中的属性数（如果有） |
| methods | numeric | 类型中的方法数（如果有） |
| predefined | varchar(3) | 类型是否为预定义类型 ( YES ) 或不是 ( NO ) |
| incomplete | varchar(3) | 类型是否为不完整类型 ( YES ) 或不是 ( NO ) |
| final | varchar(3) | 类型是否为最终类型 ( YES ) 或不是 ( NO ) |
| instantiable | varchar(3) | 类型是否为可实例化类型 ( YES ) 或不是 ( NO ) |
| supertype\_owner | varchar(128) | 超类型的所有者（如果类型不是子类型，则为 NULL） |
| supertype\_name | varchar(128) | 超类型的名称（如果类型不是子类型，则为 NULL） |
| local\_attributes | numeric | 子类型中本地（非继承）属性（如果有）的数量 |
| local\_methods | numeric | 子类型中本地（非继承）方法（如果有）的数量 |
| typeid | raw | 类型 ID 值 |

### ALL\_USERS

ALL\_USERS视图提供当前用户可访问数据库的用户信息。

| **列名** | **类型** | **描述** |
| --- | --- | --- |
| username | name | 用户名 |
| user\_id | oid | 用户名所属的编号 |

### ALL\_VIEWS

ALL\_VIEWS视图提供当前用户可访问数据库的视图信息。

| **列名** | **类型** | **描述** |
| --- | --- | --- |
| owner | name | 视图所有者 |
| view\_name | name | 视图名称 |
| text\_length | integer | 视图文本的长度 |
| text | text | 查看文本。仅当行源自当前容器时，此列才会返回正确的值。该 BEQUEATH 子句不会作为 TEXT 该视图中列的一部分出现。 |
| text\_vc | varchar(4000) | 查看文本。此列可能会截断视图文本。该 BEQUEATH 子句不会作为 TEXT\_VC 该视图中列的一部分出现。 |
| type\_text\_length | numeric | 类型化视图的类型子句的长度 |
| type\_text | varchar(4000) | 类型化视图的类型子句 |
| oid\_text\_length | numeric | WITH OID 类型化视图的子句 长度 |
| oid\_text | varchar(4000) | WITH OID 类型化视图的子句 |
| view\_type\_owner | varchar(128) | 如果视图是类型化视图，则视图类型的所有者 |
| view\_type | varchar(128) | 视图类型（如果视图是类型化视图） |
| superview\_name | varchar(128) | 超级视图的名称 |
| editioning\_view | varchar(1) | 留作将来使用 |
| read\_only | varchar(1) | 指示视图是否为只读 ( Y ) 或非 ( N ) |
| container\_data | varchar(3) | 指示视图是否包含特定于容器的数据 |
| bequeath | varchar(12) | 可能的值： CURRENT\_USER : 当视图是 BEQUEATH CURRENT\_USER 视图 时 DEFINER : 当视图是 BEQUEATH DEFINER 视图 时 |
| origin\_con\_id | varchar(256) | 数据来源的容器的 ID |
| default\_collation | varchar(100) | 视图的默认排序规则 |
| containers\_default | varchar(3) | CONTAINERS() 指示默认情况下是否启用视图( YES ) 或不启用 ( NO ) |
| container\_map | varchar(3) | 指示是否启用视图以与 container\_map 数据库属性 ( YES ) 一起使用 ( NO ) |
| extended\_data\_link | varchar(3) | 指示是否启用视图以从根 ( YES ) 获取扩展数据链接 ( NO ) |
| extended\_data\_link\_map | varchar(3) | 仅为支持兼容性 |

### DBA\_ALL\_TABLES

DBA\_ALL\_TABLES视图提供所有用户的表的信息。该视图中的字段与ALL\_ALL\_TABLES相同。具体请参考[ALL\_ALL\_TABLES](#ALL_ALL_TABLES)。

### DBA\_CONS\_COLUMNS

DBA\_CONS\_COLUMNS视图提供所有用户的表上约束指定列信息的信息。视图中的字段与ALL\_CONS\_COLUMNS相同。具体请参考[ALL\_CONS\_COLUMNS](#ALL_CONS_COLUMNS)。

### DBA\_CONSTRAINTS

DBA\_CONSTRAINTS视图提供所有用户的表上的约束信息。视图中的字段与ALL\_CONSTRAINTS相同。具体请参考[ALL\_CONSTRAINTS](#ALL_CONSTRAINTS)。

### DBA\_IND\_COLUMNS

DBA\_IND\_COLUMNS视图提供所有用户的索引中包含的列信息。

| **列名** | **类型** | **描述** |
| --- | --- | --- |
| index\_owner | varchar(64) | 索引列的所有者 |
| schema\_name | varchar(64) | 索引所属的模式名称 |
| index\_name | varchar(64) | 索引名称 |
| table\_owner | varchar(64) | 表的所有者 |
| table\_name | varchar(64) | 表名 |
| column\_name | name | 列名 |
| column\_position | smallint | 列的编号。一般列从1开始向上编号。系统列则拥有（任意）赋值编号 |
| column\_length | smallint | 对于一个固定尺寸的类型，typlen是该类型内部表示的字节数。对于一个变长类型，typlen为负值。-1表示一个“varlena”类型（具有长度字），-2表示一个以空值结尾的C字符串 |
| char\_length | numeric | 列的数据类型如果为char，表示字节数。否则为0 |
| descend | character(1) | 默认DESC。指示列是按降序（DESC）还是按升序（ASC）排序 |

### DBA\_IND\_PARTITIONS

DBA\_IND\_PARTITIONS视图存储数据库中所有索引分区的信息。该视图中的字段与ALL\_IND\_PARTITIONS相同。具体请参考[ALL\_IND\_PARTITIONS](#ALL_IND_PARTITIONS)。

### DBA\_INDEXES

DBA\_INDEXES视图提供所有用户的表上的索引信息。该视图中的字段与ALL\_INDEXES相同。具体请参考[ALL\_INDEXES](#ALL_INDEXES)。

### DBA\_OBJECTS

DBA\_OBJECTS视图提供所有用户的数据库中所有对象的信息。视图中的字段与ALL\_OBJECTS相同。具体请参考[ALL\_OBJECTS](#ALL_OBJECTS)。

### DBA\_PART\_KEY\_COLUMNS

DBA\_PART\_KEY\_COLUMNS视图提供所有用户的数据库中的分区表的分区键列的信息。视图中的字段与ALL\_PART\_KEY\_COLUMNS相同。具体请参考[ALL\_PART\_KEY\_COLUMNS](#ALL_PART_KEY_COLUMNS)。

### DBA\_PART\_TABLES

DBA\_PART\_TABLES视图提供所有用户的数据库中的分区表信息。

| **列名** | **类型** | **描述** |
| --- | --- | --- |
| owner | varchar(128) | 分区表的所有者 |
| table\_name | varchar(128) | 分区表名 |
| partitioning\_type | varchar(9) | 分区类型： range、list、hash |
| subpartitioning\_type | varchar(9) | 子分区类型：range、list、hash |
| partition\_count | bigint | 分区表中的分区数 |
| def\_subpartition\_count | bigint | 复合分区表中，子分区数 |
| partitioning\_key\_count | integer | 分区键的列数 |
| subpartitioning\_key\_count | integer | 复合分区表中，子分区中的列数 |
| status | varchar(8) | 仅为支持兼容性。该值始终为VALID |
| def\_tablespace\_name | name | 仅为支持兼容性。该值始终为NULL |
| def\_pct\_free | numeric | 仅为支持兼容性。该值始终为NULL |
| def\_pct\_used | numeric | 仅为支持兼容性。该值始终为NULL |
| def\_ini\_trans | numeric | 仅为支持兼容性。该值始终为NULL |
| def\_max\_trans | numeric | 仅为支持兼容性。该值始终为NULL |
| def\_initial\_extent | varchar(40) | 仅为支持兼容性。该值始终为NULL |
| def\_next\_extent | varchar(40) | 仅为支持兼容性。该值始终为NULL |
| def\_min\_extents | varchar(40) | 仅为支持兼容性。该值始终为NULL |
| def\_max\_extents | varchar(40) | 仅为支持兼容性。该值始终为NULL |
| def\_max\_size | varchar(40) | 仅为支持兼容性。该值始终为NULL |
| def\_pct\_increase | varchar(40) | 仅为支持兼容性。该值始终为NULL |
| def\_freelists | numeric | 仅为支持兼容性。该值始终为NULL |
| def\_freelist\_groups | numeric | 仅为支持兼容性。该值始终为NULL |
| def\_logging | varchar(7) | 仅为支持兼容性。该值始终为YES |
| def\_compression | varchar(8) | 仅为支持兼容性。该值始终为None |
| def\_compress\_for | varchar(30) | 添加分区时使用哪种操作的默认压缩 |
| def\_buffer\_pool | varchar(7) | 仅为支持兼容性。该值始终为DEFAULT |
| def\_flash\_cache | varchar(7) | 仅为支持兼容性。该值始终为NULL |
| def\_cell\_flash\_cache | varchar(7) | 仅为支持兼容性。该值始终为NULL |
| ref\_ptn\_constraint\_name | varchar(128) | 仅为支持兼容性。该值始终为NULL |
| interval | varchar(1000) | 区间值字符串 |
| autolist | varchar(3) | 指示表是否通过自动列表分区 ( YES )分区 ( NO ) |
| interval\_subpartition | varchar(1000) | 子分区间隔值的字符串 |
| autolist\_subpartition | varchar(3) | 指示此子 分区是否使用 ( YES )自动列表分区 ( NO) |
| is\_nested | varchar(3) | 指示分区表是否为嵌套表 ( YES ) 或不是 ( NO ) |
| def\_segment\_creation | varchar(4) | 指定是否在表级别指定了段创建的默认值：NO（指定延迟），YES（立即指定），NONE（未指定段创建的默认值） |
| def\_indexing | varchar(3) | 指示为表指定的索引属性 |
| def\_inmemory | varchar(8) | 只是内存列存储是否启用( ENABLED )、禁用 ( DISABLED ) 或未指定 ( NONE ) 用于此表中的分区 |
| def\_inmemory\_priority | varchar(8) | 内存列存储填充的优先级 |
| def\_inmemory\_distribute | varchar(15) | 在 Oracle RAC)环境中默认情况下如何为表的分区分配 内存列存储 |
| def\_inmemory\_compression | varchar(17) | 内存列存储的默认压缩级别 |
| def\_inmemory\_duplicate | varchar(13) | Oracle RAC 环境中 内存列存储的默认重复设置 |
| def\_read\_only | varchar(3) | 新分区的默认设置：（YES）新分区的默认设置是只读的。 （NO）新分区的默认设置是读/写。 |
| def\_cellmemory | varchar(24) | 显示父表中的新分区将继承的属性的默认值， CELLMEMORY 除非该行为被显式覆盖 |
| def\_inmemory\_service | varchar(12) | 默认情况下如何在各种实例上为表的分区填充内存列存储 |
| def\_inmemory\_service\_name | varchar(1000) | 默认情况下应为表的分区填充 内存列存储的服务的服务名称。 |

### DBA\_ROLE\_PRIVS

DBA\_ROLE\_PRIVS包含当前数据库所有角色的权限信息。

| **列名** | **类型** | **描述** |
| --- | --- | --- |
| grantee | text | 接受授权的用户或角色名称 |
| granted\_role | text | 被授权角色名称 |
| admin\_option | text | 权限是否与amin option一起使用 |
| default\_role | text | 是否被定义为默认 |

### DBA\_ROLES

DBA\_ROLES包含当前系统的角色信息。

| **列名** | **类型** | **描述** |
| --- | --- | --- |
| role | text | 角色名称 |
| password\_required | text | 指定角色使用密码认证 |

### DBA\_SEGMENTS

DBA\_SEGMENTS描述数据库中的所有段分配的存储。

| **列名** | **类型** | **描述** |
| --- | --- | --- |
| owner | name | 段的所有者 |
| segment\_name | name | 段的名称 |
| partition\_name | varchar(128) | 段所在分区的名称（设置NULL为非分区对象） |
| segment\_type | varchar(18) | 段的类型 |
| segment\_subtype | varchar(10) | LOB 段的子类型：SECUREFILE、ASSM、MSSM和NULL |
| tablespace\_name | name | 包含段的表空间的名称 |
| header\_file | oid | 包含段头的文件编号 |
| header\_block | oid | 包含段头的块的编号 |
| bytes | bigint | 段的大小，以字节为单位 |
| blocks | double precision | 段的大小（以 Oracle 块为单位） |
| extents | numeric | 分配给段的区数 |
| initial\_extent | numeric | 在创建时为段的初始范围请求的大小（以字节为单位） |
| next\_extent | numeric | 要分配给该段的下一个扩展区的大小（以字节为单位） |
| min\_extents | numeric | 段中允许的最小范围数 |
| max\_extents | numeric | 段中允许的最大范围数 |
| max\_size | numeric | 段中允许的最大块数 |
| retention | varchar(7) | SECUREFILE段 的保留选项 |
| minretention | numeric | SECUREFILE分段 的最短保留期限 |
| pct\_increase | numeric | 增加下一个要分配的范围大小的百分比 |
| freelists | numeric | 分配给该段的进程空闲列表数 |
| freelist\_groups | numeric | 分配给该段的空闲列表组数 |
| relative\_fno | numeric | 段头的相对文件号 |
| buffer\_pool | varchar(7) | 用于段块的缓冲池：DEFAULT、KEEP、RECYCLE |
| flash\_cache | varchar(7) | 用于段块的数据库智能闪存缓存提示（仅限 Solaris 和 Oracle Linux 功能） |
| cell\_flash\_cache | varchar(7) | 用于段块的单元闪存缓存提示：DEFAULT、KEEP、NONE |
| inmemory | varchar(8) | 此段 的内存中列存储是启用 ( ENABLED) 还是禁用 (DISABLED ) |
| inmemory\_priority | varchar(8) | 内存列存储填充的优先级 |
| inmemory\_distribute | varchar(15) | 内存列存储在ORACLE RAC环境中的分布方式 |
| inmemory\_duplicate | varchar(13) | Oracle RAC 环境中内存列存储的重复设置 |
| inmemory\_compression | varchar(17) | 内存列存储的压缩级别 |
| cellmemory | varchar(24) | 存储单元闪存缓存中的列压缩值 |

### DBA\_SEQUENCES

DBA\_SEQUENCES视图提供所有用户的数据库分区表的子分区的关键列的信息。视图中的字段与ALL\_SEQUENCES相同。具体请参考[ALL\_SEQUENCES](#ALL_SEQUENCES)。

### DBA\_SUBPART\_KEY\_COLUMNS

DBA\_SUBPART\_KEY\_COLUMNS视图提供了数据库分区表的子分区键列信息。视图中的字段与ALL\_SUBPART\_KEY\_COLUMNS相同。具体请参考[ALL\_SUBPART\_KEY\_COLUMNS](#ALL_SUBPART_KEY_COLUMNS)。

### DBA\_SYNONYMS

DBA\_SYNONYMS提供了数据库同义词的信息。

| **列名** | **类型** | **描述** |
| --- | --- | --- |
| owner | text | 所有者 |
| schema\_name | text | 名字空间名 |
| synonym\_name | text | 同义词名 |
| table\_owner | text | 引用对象的所有者 |
| table\_schema\_name | text | 引用对象所在的名字空间名 |
| table\_name | text | 引用对象名 |
| db\_link | text | 引用数据库连接的名字 |

### DBA\_TAB\_COLUMNS

DBA\_TAB\_COLUMNS视图提供所有用户的表和视图中所有列的信息。视图中的字段与ALL\_TAB\_COLUMNS相同。具体请参考[ALL\_TAB\_COLUMNS](#ALL_TAB_COLUMNS)。

### DBA\_TAB\_PARTITIONS

DBA\_TAB\_PARTITIONS视图提供所有用户的数据库中的所有分区的信息。视图中的字段与ALL\_TAB\_PARTITIONS相同。具体请参考[ALL\_TAB\_PARTITIONS](#ALL_TAB_PARTITIONS)。

### DBA\_TAB\_SUBPARTITIONS

DBA\_TAB\_SUBPARTITIONS视图提供所有用户的数据库中的所有子分区的信息。视图中的字段与ALL\_TAB\_SUBPARTITIONS相同。具体请参考[ALL\_TAB\_SUBPARTITIONS](#ALL_TAB_SUBPARTITIONS)。

### DBA\_TABLES

DBA\_TABLES视图提供所有用户的数据库用户定义表的信息。视图中的字段与ALL\_TABLES相同。具体请参考[ALL\_TABLES](#ALL_TABLES)。

### DBA\_TRIGGERS

DBA\_TRIGGERS视图提供所有用户的数据库表上的触发器信息。视图中的字段与ALL\_TRIGGERS相同。具体请参考[ALL\_TRIGGERS](#ALL_TRIGGERS)。

### DBA\_TYPES

DBA\_TYPES视图提供所有用户的数据库的对象类型信息。视图中的字段与ALL\_TYPES相同。具体请参考[ALL\_TYPES](#ALL_TYPES)。

### DBA\_USERS

DBA\_USERS视图提供所有用户的数据库的用户信息。

| **列名** | **类型** | **描述** |
| --- | --- | --- |
| username | varchar(64) | 用户名 |
| user\_id | oid | 用户id |
| password | varchar(30) | 密码 |
| account\_status | varchar(32) | 用户状态 |
| lock\_date | timestamp without time zone | 如果账户状态是lock，显示被锁日期 |
| expiry\_date | timestamp without time zone | 过期时间 |
| default\_tablespace | varchar(64) | 数据的默认表空间 |
| temporary\_tablespace | varchar(30) | 临时表的默认表空间的名称或表空间组的名称 |
| local\_temp\_tablespace | varchar(30) | 用户的默认本地临时表空间 |
| created | timestamp without time zone | 仅为支持兼容性。该值始终为NULL |
| profile | varchar(30) | Profile名 |
| initial\_rsrc\_consumer\_group | varchar(30) | 仅为支持兼容性。该值始终为NULL |
| external\_name | varchar(4000) | 仅为支持兼容性。该值始终为NULL |
| password\_versions | varchar(12) | 显示帐户存在的密码哈希（也称为“验证器”）的版本列表。 |
| editions\_enabled | varchar(1) | 指示是否已为相应用户启用版本 ( Y) 或未启用 ( N) |
| authentication\_type | varchar(8) | 表示用户的认证机制 |
| proxy\_only\_connect | varchar(1) | 指示用户是否可以直接连接 (N) 或该帐户是否只能由对此帐户具有代理权限的用户代理 (Y)（即，由已被授予此帐户“连接通过”权限的用户 |
| common | varchar(3) | 指示给定用户是否是普通用户：YES(普通用户)，NO(本地用户) |
| last\_login | timestamp with time zone | 上次用户登录时间 |
| oracle\_maintained | varchar(1) | 表示用户是否由 Oracle 提供的脚本（例如 catalog.sql 或 catproc.sql）创建和维护 |
| inherited | varchar(3) | 指示用户定义是否继承自另一个容器 ( YES) 或不是 ( N) |
| default\_collation | varchar(100) | 用户架构的默认排序规则 |
| implicit | varchar(3) | 指示此用户是否是由隐式应用程序创建的普通用户 ( YES) 或不是 ( NO) |
| all\_shard | varchar(3) | 在分片数据库中，此列中的值指示用户是否是在启用分片 DDL 的情况下创建的 |

### DBA\_VIEWS

DBA\_VIEWS视图提供所有用户的数据库的视图信息。视图中的字段与ALL\_VIEWS相同。具体请参考[ALL\_VIEWS](#ALL_VIEWS)。

### PRODUCT\_COMPONENT\_VERSION

兼容Oracle静态数据字典PRODUCT\_COMPONENT\_VERSION，其中包含组件产品的版本和状态信息。

可以使用如下查询语句进行查询：

select \* from PRODUCT\_COMPONENT\_VERSION;

列名 类型 描述 PRODUCT text 产品名称 VERSION text 版本号 STATUS text 发布状态

### USER\_IND\_PARTITIONS

USER\_IND\_PARTITIONS视图用于存储当前用户下的索引分区信息。该视图中的字段与ALL\_IND\_PARTITIONS相同。具体请参考[ALL\_IND\_PARTITIONS](#ALL_IND_PARTITIONS)。

### USER\_PART\_INDEXES

USER\_PART\_INDEXES视图存储当前用户下分区表的索引信息。该视图中的字段（除OWNER外）与ALL\_PART\_INDEXES相同。具体请参考[ALL\_PART\_INDEXES](#ALL_PART_INDEXES)。

### USER\_PART\_TABLES

USER\_PART\_TABLES视图用于提供当前用户的数据库中的分区表信息。该视图中的字段（除OWNER外）与ALL\_PART\_TABLES相同。具体请参考[ALL\_PART\_TABLES](#ALL_PART_TABLES)。

## 动态性能视图

### V$DATABASE

V$DATABASE包含当前数据库的基本信息。

| **列名** | **类型** | **描述** |
| --- | --- | --- |
| dbid | oid | 数据库唯一标识符 |
| name | name | 数据库名称 |
| created | timestamp with time zone | 数据库的创建时间 |
| resetlogs\_change# | integer | 打开的重置日志中的系统更改号 |
| resetlogs\_time | timestamp with time zone | 打开的重置日志的时间戳 |
| prior\_resetlogs\_change# | integer | 先前的重置日志中的版本号 |
| prior\_resetlogs\_time | text | 先前重置日志的时间 |
| log\_mode | text | 归档日志模式 |
| checkpoint\_change# | integer | 上一个scn检查点 |
| archive\_change# | integer | 数据库强制归档scn |
| controlfile\_type | text | 控制文件类型 |
| controlfile\_created | text | 控制文件的创建日期 |
| controlfile\_sequence# | integer | 控制文件序列号由控制文件事务增加 |
| controlfile\_change# | integer | 备份控制文件中的最后一个scn；如果控制文件不是备份，则为null |
| controlfile\_time | text | 备份控制文件中的最后一个时间戳；如果控制文件不是备份，则为null |
| open\_resetlogs | integer | 表明下一个打开的数据库是允许还是需要resetlogs选项 |
| version\_time | text | 版本时间 |
| open\_mode | text | 数据库打开模式 |
| protection\_mode | text | 当前数据库的保护模式 |
| protection\_level | text | 当前数据库的保护级别 |
| remote\_archive | text | remote\_archive\_enable初始化参数的值 |
| activation# | integer | 分配给数据库实例的编号 |
| switchover# | integer | 分配给数据库切换的编号 |
| database\_role | text | 数据库的当前角色 |
| archivelog\_change# | integer | 归档日志的最高编号 |
| archivelog\_compression | text | 归档日志的压缩状态 |
| switchover\_status | text | 表明允许切换 |
| dataguard\_broker | text | data guard代理信息 |
| guard\_status | text | 数据保护状态 |
| supplemental\_log\_data\_min | text | 确保logminer有足够的信息进行工作 |
| supplemental\_log\_data\_pk | text | 将所有具有主键的表的键值放入重做日志中 |
| supplemental\_log\_data\_ui | text | 对于所有拥有唯一键的表，指示修改时放入重做日志中 |
| force\_logging | text | 指示数据库当前是否为强制日志模式 |
| platform\_id | integer | 数据库的平台标示号 |
| platform\_name | text | 数据库的平台名称 |
| recovery\_target\_incarnation# | integer | 该数据库恢复的所有文件的编号 |
| last\_open\_incarnation# | integer | v$database\_incarnation成功打开的记录号 |
| current\_scn | integer | 当前的scn号 |
| flashback\_on | text | 指示闪回功能是否开启 |
| supplemental\_log\_data\_fk | text | 对于拥有外键的表，修改时是否将键值放入重做日志中 |
| supplemental\_log\_data\_all | text | 对于所有列，修改时是否全部放入重做日志中 |
| db\_unique\_name | name | 数据库的唯一标示名称 |
| standby\_became\_primary\_scn | integer | 备机成为主机的scn号 |
| fs\_failover\_status | text | 快速故障转移的状态 |
| fs\_failover\_current\_target | text | db\_unique\_name备用数据库 |
| fs\_failover\_threshold | integer | 尝试使用备用机前的观察时间（单位为秒） |
| fs\_failover\_observer\_present | text | 指示监控者是否连接到数据库 |
| fs\_failover\_observer\_host | text | 当前监控器的主机名 |
| controlfile\_converted | text | 指示还原期间是否隐式转换控制文件 |
| primary\_db\_unique\_name | text | 对于任何节点数据库，此参数标明主节点名称 |
| supplemental\_log\_data\_pl | text | 在支持复制工作时，是否额外在重做日志中记录内部的调用信息 |
| min\_required\_capture\_change# | text | required\_checkpoint\_scn数据库上所有本地捕获过程的最小值 |

### V$DBLINK

V$DBLINK包含当前会话在使用dblink时所有链接信息，在事务提交或回滚时清除。

| **列名** | **类型** | **描述** |
| --- | --- | --- |
| db\_link | text | 数据库链接的名称 |
| owner\_id | text | 数据库链接uid的所有者 |
| logged\_on | text | 数据库链接当前是否已登录 |
| heterogeneous | text | 数据库链接是否异构 |
| protocol | text | 数据库链接的通信协议 |
| open\_cursors | text | 数据库链接是否有打开的游标 |
| in\_transaction | text | 数据库链接当前是否在事务中 |
| update\_sent | text | 数据库链接上是否有更新 |
| commit\_point\_strength | text | 数据库链接上事务的提交点强度 |

### V$INSTANCE

V$INSTANCE描述当前系统实例信息。

| **列名** | **类型** | **描述** |
| --- | --- | --- |
| instance\_number | integer | 实例编号 |
| instance\_name | name | 实例名称 |
| host\_name | text | 当前实例所在主机名称 |
| version | text | 数据库版本信息 |
| startup\_time | timestamp with time zone | 数据库启动时间 |
| status | text | 数据库状态 |
| parallel | text | 指明数据库是否在集群中 |
| thread# | integer | 数据库打开的重做线程数 |
| archiver | text | 自动归档信息 |
| log\_switch\_wait | text | 记录切换日志的等待事件 |
| logins | text | 指明数据库是否处于受限状态 |
| shutdown\_pending | text | 是否正在关闭 |
| database\_status | text | 当前数据库状态 |
| instance\_role | text | 当前实例角色信息 |
| active\_state | text | 当前实例的活动状态 |
| blocked | text | 当前实例是否阻塞 |
| edition | text | 版本信息 |

### V$LICENSE

V$LICENSE包含当前系统license信息。

| **列名** | **类型** | **描述** |
| --- | --- | --- |
| sessions\_max | integer | 允许的最大会话数 |
| sessions\_warning | integer | 并发用户的警告阈值 |
| sessions\_current | integer | 当前并发的会话数量 |
| sessions\_highwater | integer | 会话高水位线 |
| users\_max | integer | 数据库允许的最大用户数 |
| cpu\_count\_current | text | 系统上当前逻辑cpu数 |
| cpu\_core\_count\_current | text | 系统上当前物理cpu数 |
| cpu\_socket\_count\_current | text | 系统上当前的cpu插槽数 |
| cpu\_count\_highwater | text | 自实例启动以来，系统上逻辑cpu的最大值 |
| cpu\_core\_count\_highwater | text | 自实例启动以来，系统上的最大cpu物理核 |
| cpu\_socket\_count\_highwater | text | 自实例启动以来，系统上的最大cpu插槽数 |

### V$OBJECT\_USAGE

V$OBJECT\_USAGE包含数据库收集的用户索引使用情况的统计信息。

| **列名** | **类型** | **描述** |
| --- | --- | --- |
| index\_name | name | 索引名称 |
| table\_name | name | 索引所在的表名 |
| monitoring | text | 该索引是否被监控 |
| used | text | 该索引是否被使用 |
| start\_monitoring | text | 开始监控的时间 |
| end\_monitoring | text | 停止监控的时间 |

### V$OBSOLETE\_PARAMETER

V$OBSOLETE\_PARAMETER包含当前实例的初始化参数信息。

| **列名** | **类型** | **描述** |
| --- | --- | --- |
| name | text | 参数名称 |
| isspecified | boolean | 指明参数是否在参数文件中指定 |

### V$OPTION

V$OPTION包含当前数据库扩展功能信息。

| **列名** | **类型** | **描述** |
| --- | --- | --- |
| parameter | name | 选项功能的名称 |
| value | boolean | 该功能是否开启 |

### V$PARAMETER

V$PARAMETER包含当前实例参数的详细信息。

| **列名** | **类型** | **描述** |
| --- | --- | --- |
| num | bigint | 参数编号 |
| name | text | 参数名称 |
| type | integer | 参数类型 |
| value | text | 当前会话的参数值 |
| display\_value | text | 参数值的智能显示 |
| isdefault | boolean | 是否为默认值 |
| isses\_modifiable | boolean | 是否可以通过alter session修改 |
| issys\_modifiable | boolean | 是否可以通过alter system修改 |
| isinstance\_modifiable | boolean | 集群中每个实例参数是否可以不同 |
| ismodified | boolean | 该参数是否已修改 |
| isadjusted | boolean | 数据库实例的建议值 |
| isdeprecated | boolean | 该参数是否已弃用 |
| isbasic | boolean | 该参数是否为基本参数 |
| description | text | 参数描述 |
| update\_comment | text | 参数更新的相关注释 |
| hash | text | 参数名的哈希值 |

### V$TABLESPACE

V$TABLESPACE包含当前实例表空间的信息。

| **列名** | **类型** | **描述** |
| --- | --- | --- |
| ts# | bigint | 表空间编号 |
| name | name | 表空间名称 |
| included\_in\_database\_backup | text | 该表空间是否可以包含在备份集 |
| bigfile | text | 是否开启大文件 |
| flashback\_on | text | 是否参与到flashback database操作中 |
| encrypt\_in\_backup | text | 该表空间是否为加密 |

### V$VERSION

V$VERSION包含当前软件的版本信息。

| **列名** | **类型** | **描述** |
| --- | --- | --- |
| banner | text | 当前的组件名称与版本号 |

# SQL语法参考

## 数据类型及操作符

### ^=运算符

**功能描述**

实现oracle^=运算符，即不等于运算符，类似于!=和<>。

**兼容性**

完全兼容。

**示例**

1、创建测试表并插入数据。

create table t\_ysf4(

id int,

col\_varchar varchar(30),

col\_int int,

col\_date date

);

insert into t\_ysf4 values(1,'1',1,'1997-02-20');

insert into t\_ysf4 values(2,'2',2,'2007-02-20');

2、使用^=运算符。

select \* from t\_ysf4 where col\_int ^='1';

返回结果如下：

id | col\_varchar | col\_int | col\_date

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

2 | 2 | 2 | 2007-02-20 00:00:00

(1 row)

4、使用^=运算符进行判断。

select '1' ^= '2';

返回结果为：

?column?

----------

t

(1 row)

### BFILE数据类型

**功能描述**

bfile用于存储服务器文件系统中的物理二进制文件数据对象。bfile文件存储在操作系统文件中，而不是在数据库中，bfile列存储对操作系统文件的引用。操作系统访问的任何存储设备都可以保存bfile数据，包括硬盘驱动器、CD-ROM、PhotoCD和DVD。如果操作系统支持对操作系统文件的流模式访问，则数据库可以访问bfile。

bfile类型的数据是：

* 在应用程序运行时不会更改的二进制数据，例如图形。
* 加载到其他大型对象类型（例如BLOB或CLOB）中的数据，可以在其中操作数据。
* 适用于字节流访问的数据，例如多媒体数据。

在对bfile类型的SQL操作中，需要支持bfilename函数。bfilename函数返回一个bfile文件定位器，该定位器指向操作系统中一个物理文件。

bfilename函数主要用于bfile类型数据的构造，构造的数据源为一个directory对象和一个操作系统物理二进制文件名，函数语法格式为：

BFILENAME('directory', 'filename')

* directory对象：指向一个操作系统的物理目录。该目录不应为数据库数据目录，以及操作系统控制文件、日志文件和其他系统文件所在的目录。必须在bfilename函数调用前创建，并且对名区分大小写。
* filename：文件名，操作系统上一个物理二进制文件，由于bfile为只读数据类型，必须在访问文件数据前创建该文件。

bfile数据类型主要用于对bfilename函数构造的数据进行存储，然后调用dbms\_lob内置包的接口对该数据进行操作。bfile数据类型的功能主要分为三个部分：

* bfile数据的存储和打印。
* bfile数据的使用，主要为对操作系统物理文件的读取过程，即分为以下三部分：
* 打开操作系统物理文件。bfile为只读数据类型，因此打开文件的模式只能为只读。
* 对文件进行读取。可根据参数设置来决定读取文件的字节数，如需读取整个文件，可先调用dbms\_lob包的getlength，函数获取文件的大小。需注意的是，单次读取文件的字节数不能超过32767字节，如文件过大，可通过设置参数对文件进行多次读取。
* 关闭文件。对打开的物理文件进行关闭。

bfile数据的修改：对于bfile数据的持久引用（如存储在表中），bfile数据可以作为表的列值进行修改。

**注意事项**

bfilename函数依赖vastbase自带的directory功能。

**使用流程**

1、创建directory对象，创建需访问的文件并将其置于directory对象所指向的操作系统目录中。

2、创建包含bfile数据类型列的表。

3、调用bfilename函数对数据进行构造并插入到表中进行存储。

**示例**

1、操作系统环境下创建文件，并输入数据。

vi /home/vastbase/bfile.data

cat /home/vastbase/bfile.data

文件内容为：

张三

李四

2、在数据库中执行如下命令创建目录和表。

drop directory d\_bfile;

drop table testbfile1;

create directory d\_bfile as '/home/vastbase';

create table testbfile1(id number,bfile\_name bfile);

3、插入数据。

insert into testbfile1 values (1,bfilename('d\_bfile','bfile.data'));

4、查询数据。

select \* from testbfile1;

返回结果为：

id | bfile\_name

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

1 | bfilename('d\_bfile','bfile.data')

### BINARY\_FLOAT数据类型

**功能描述**

binary\_float类型是Oracle支持的一种浮点数类型，大小为4字节。本功能是支持Vastbase数据库能够使用binary\_float这一类型,并且该数据类型也可以包含在plpgsql中使用。

**示例**

* 表的增删改查支持binary\_float数据类型

1、创建测试表并插入数据。

CREATE TABLE test(id int, var binary\_float);

INSERT INTO test VALUES(1,1.25),(2,8.234),(3,0.1234);

2、查询表的数据。

select \* from test;

返回结果为：

id | var

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

1 | 1.25

2 | 8.234

3 | .1234

(3 rows)

3、更新表中的数据并查询结果。

UPDATE test SET var = 7.321 WHERE id = 2;

SELECT \* FROM test;

返回结果为：

id | var

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

1 | 1.25

3 | .1234

2 | 7.321

(3 rows)

4、更新表中的数据并查询结果。

DELETE FROM test WHERE id = 3;

SELECT \* FROM test;

返回结果为：

id | var

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

1 | 1.25

2 | 7.321

(2 rows)

* PL/PGSQL中支持binary\_float数据类型

1、创建存储过程。

create or replace function test()

return binary\_float as

declare

a binary\_float := 1.23;

b binary\_float :=3.45;

begin

return a + b;

end;

/

2、查询结果。

SELECT test();

返回结果为：

test

------

4.68

(1 row)

### DATE类型的减法与乘法

**功能描述**

* date类型的数据相减，结果是number类型（保留小数），结果不含小数部分则展示为整数。
* date类型的数据相减，再和整数类型数据相乘，结果是number类型（保留小数），结果不含小数部分则展示为整数。
* sysdate参与日期类乘法，结果返回数值类型。

**注意事项**

* 受显示格式和数据类型精度等原因影响，如果同样的数据计算形式，结果在小数位上与Oracle不完全一致，此为正常现象。
* 该功能仅在数据库兼容模式为Oracle时能够使用（即创建DB时DBCOMPATIBILITY='A'），在其他数据库兼容模式下不能使用该特性。

**示例**

**示例1：**减法（结果为number类型，由客户端控制保留几位小数）。

select to\_date('20220110 10:10:10','yyyymmdd hh24:mi:ss')-to\_date('20220101 9:9:9','yyyymmdd hh24:mi:ss') res from dual;

结果显示如下：

RES

----------

9.04237268518519

(1 row)

结果说明：

* 单位：天（day）。
* 小数计算方式：日期相差换算为秒数/(24*60*60)。该例中前后相差为9天零1小时零1分零1秒，则结果为9+(1*60*60+1*60+1)/(24*60\*60)≈9.04237268518519，由客户端控制保留几位小数，如果客户端控制保留7位，则最终结果为9.04237269。
* 刚好为整数日时，显示结果不带小数部分。

**示例2：**日期相减的差，再做乘法。

select (to\_date('20220110 10:10:10','yyyymmdd hh24:mi:ss')-to\_date('20220101 9:9:9','yyyymmdd hh24:mi:ss'))\*86400 as res from dual;

结果显示如下：

RES

----------

781261.000000000416

(1 row)

**示例3：**sysdate参与日期类乘法，结果返回数值类型

sysdate减去一个更早的时间，再乘以5。

select (sysdate-to\_date('2020-8-1 12','yyyy-mm-dd hh24'))\*5 from dual;

结果显示如下：

?column?

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

3754.983333333335

(1 row)

### DATE数据类型

**功能描述**

date数据类型用于存储日期和时间信息，对于每一个date值：

* 展示格式为：YYYY/MM/DD，也可以是YYYY/MM/DD HH24:MI:SS
* 存储格式只有一种：YYYY/MM/DD HH24:MI:SS
* date类型的值在数据库图形界面中可以有不同的显示格式（具体格式由工具设置决定），但其实际存储格式只是其中一种，当只存储年月日时，date实际存储的是:年月日0时0分0秒，在图形界面中，其展示格式为：YYYY/MM/DD。
* date类型对插入带有毫秒的部分直接去除毫秒，不进行四舍五入。
* 支持date数据类型和to\_char字符串的转换。

**示例**

1、创建测试表并插入数据。

create table t1(id int,col date,col1 timestamp);

insert into t1 select 1,current\_timestamp,current\_timestamp from dual;

2、将date数据类型转换为to\_char类型，并查询结果。

select id,to\_char(col,'yyyy-mm-dd hh24:mi:ss'),to\_char(col1,'yyyy-mm-dd hh24:mi:ss.ff6') from t1;

返回结果如下(其中col字段去除毫秒，不会做四舍五入)：

id | to\_char | to\_char

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

1 | 2022-07-04 17:15:16 | 2022-07-04 17:15:16.174639

(1 row)

3、获取时间的年份。

select to\_char(col,'yyyy') as nowyear from t1;

返回结果为当前年份：

nowyear

---------

2022

(1 row)

### TIMESTAMP数据类型

**功能描述**

该数据类型是DATE数据类型的扩展，它可以存储更加精确的时间值。Vastbase G100支持14位(秒)或者17位（毫秒）数字格式数输入。

**示例**

1、创建测试表并插入数据。

create table test\_timestamp(a timestamp,b timestamptz);

insert into test\_timestamp values('2022-1-12 12:23:23','2022-1-12 12:23:23');

insert into test\_timestamp values('20220112122323','20220112122323');

2、查询表中数据。

select \* from test\_timestamp ;

返回结果如下：

a | b

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

2022-01-12 12:23:23 | 2022-01-12 12:23:23+08

2022-01-12 12:23:23 | 2022-01-12 12:23:23+08

(2 rows)

## SQL语法

### GROUP BY支持含有表达式或者函数的列

**功能描述**

在Vastbase G100现有group by功能的基础上增强对Oracle group by子句的兼容。针对含有子查询的分组列，不仅支持普通的列，而且支持含有表达式或者函数的列。

**语法格式**

select [grouping\_element],

[expression]

[WHERE condition]

[from table]

group by [grouping\_element];

**参数说明**

* grouping\_element：函数、表达式。
* expression：SQL语句中的子查询。

**注意事项**

该功能仅在数据库兼容模式为Oracle时能够使用（即创建DB时DBCOMPATIBILITY='A'），在其他数据库兼容模式下不能使用该特性。

**示例**

1、创建并切换至兼容模式为ORACLE的数据库db\_oracle。

CREATE DATABASE db\_oracle dbcompatibility='A';

\c db\_oracle

2、创建测试表并插入数据。

create table test\_a(c1 number,c2 varchar2(20));

insert into test\_a values(1,'a');

insert into test\_a values(2,'a\_xxr');

insert into test\_a values(3,'a\_ytr');

insert into test\_a values(4,'b');

insert into test\_a values(5,'b\_xxr');

insert into test\_a values(6,'b\_ytr');

create table test\_b(c1 number,c2 varchar2(20));

insert into test\_b values(11,'a');

insert into test\_b values(22,'b');

insert into test\_b values(33,'aa');

insert into test\_b values(44,'ba');

insert into test\_b values(55,'e');

insert into test\_b values(66,'f');

3、使用group by字句包含函数。

SELECT substr(a.c2, 1, 1)

, (

SELECT b.c1

FROM test\_b b

WHERE b.c2 = substr(a.c2, 1, 1)

)

FROM test\_a a

GROUP BY substr(a.c2, 1, 1);

返回结果为：

substr | c1

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

a | 11

b | 22

(2 rows)

### ALL语法

**功能描述**

ALL操作符用于将值与子查询返回的值列表的结果集进行比较。

**语法格式**

Operator ALL(v1,v2,v3)

Operator ALL(subquery)

**参数说明**

Operator：标准的比较运算符（=，<>，=，>，> =，<，或<！=）。

**注意事项**

该语法必须与比较运算符一起使用。

**兼容性**

完全兼容。

**示例**

1、创建测试表并插入数据。

create table testall(id int,name char(10));

insert into testall values(1,'amy');

insert into testall values(2,'tom');

insert into testall values(3,'jack');

insert into testall values(97,'jane');

insert into testall values(98,'alice');

insert into testall values(99,'taylor');

create table testall1(id int,name char(10));

insert into testall1 values(1,'taylor');

insert into testall1 values(99,'jane');

2、使用ALL(v1,v2,v3)语法查询。

select \* from testall where id < all(3,98);

当结果返回如下，则表示查询成功：

id | name

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

1 | amy

2 | tom

(2 rows)

3、使用ALL(subquery)语法。

select \* from testall where id = all(select id from testall1 where name='taylor');

当结果返回如下，则表示查询成功：

id | name

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

1 | amy

(1 row)

### CREATE TABLE语句支持自增列

**功能描述**

建表语句中支持GENERATED BY DEFAULT AS IDENTITY语法，用来定义自增列。该语法是是可选自定义的，可以不加,不加的列就没有自增序列的效果。

**语法格式**

CREATE TABLE table\_name (

Column\_name1 INTEGER GENERATED BY DEFAULT [on null] AS IDENTITY,

Column\_name2 STRING);

**参数说明**

* table\_name：表名。
* Column\_name：列名。

**注意事项**

* 该功能仅在数据库兼容模式为Oracle时能够使用（即创建DB时DBCOMPATIBILITY='A'），在其他数据库兼容模式下不能使用该特性。
* 自增列的数据类型包括TINYINT、SMALLINT、INTEGER、BINARY\_INTEGER、BIGINT、NUMERIC、DECIMAL、NUMBER、float、DOUBLE PRECISION 、FLOAT 、DEC、等numeric类型的数据类型。
* 如果手动插入的值和自增序列产生的值重复不影响，可以并存。
* 支持generated by default [on null]as identity 其中on null是可选项。
* 一个表只能有一个列被定义为该自增列。

**示例**

1、创建自增列数据类型为numeric的表。

create table t\_identity\_numeric (a numeric(5,3) GENERATED BY DEFAULT AS IDENTITY,b text);

2、插入不带自增行的数据。

insert into t\_identity\_numeric(b) values (1),(5),(3);

insert into t\_identity\_numeric values(1.234,7);

3、查询数据。

select \* from t\_identity\_numeric;

返回结果为：

a | b

-------+--

1 | 1

2 | 5

3 | 3

1.234 | 7

(4 rows)

### CREATE TYPE语句支持or replace和{is | as} object子句

**功能描述**

Vastbase中create type语法可以兼容Oracle的or replace和{is | as} object可选子句，即数据库兼容支持以下语法：

create or replace type xxx

create type xxx is object

create type xxx as object

create or replace type xxx is object

create or replace type xxx as object

create or replace表示如果没有type xxx 则新建一个，如果已经存在，则删除后重新创建。

{is | as} object表示创建对象类型的自定义type

**语法格式**

CREATE [OR REPLACE] TYPE [ schema. ] type\_name { IS | AS } OBJECT（字段1 类型1，字段2 类型2）;

**注意事项**

该功能仅在数据库兼容模式为Oracle时能够使用（即创建DB时DBCOMPATIBILITY='A'），在其他数据库兼容模式下不能使用该特性。

**兼容性**

Vastbase G100 V2.2版本与Oracle 11g R2版本相比，仅兼容CREATE [OR REPLACE] TYPE [ schema. ] type\_name { IS | AS } OBJECT（字段1 类型1，字段2 类型2)；部分，即CREATE TYPE命令新增OR REPLACE和{ IS | AS } OBJECT可选子句。其它可选子句语法兼容.

**示例**

1、创建并切换至兼容模式为Oracle的数据库db\_oracle。

CREATE DATABASE db\_oracle dbcompatibility='A';

\c db\_oracle

2、创建一个自定义类型t\_raw。

create or replace type t\_raw is object (v\_str raw(4000),s text);

返回结果为：

CREATE TYPE

### Insert All&First子句语法

**功能描述**

Insert All&First语句允许用户在一条Insert语句中往不同的表同时插入数据，也可以实现有条件的插入。

* Insert All表示所有满足条件的插入语句都会被执行。
* Insert First在执行第一个满足条件的插入语句后就会结束。

**语法格式**

Insert [ First | All ]

[ WHEN { Condition } THEN INTO table [ [ AS ] alias ] Values (v1, ...), ...]

{ Subquery }

**参数说明**

WHEN { Condition }：任意一个判断表达式，表达式中可以使用常量或引用查询中的表字段。

**注意事项**

无。

**示例**

1、创建测试表并插入数据。

CREATE TABLE small\_orders

( order\_id NUMERIC NOT NULL,

customer\_id NUMERIC NOT NULL,

order\_total NUMERIC );

CREATE TABLE medium\_orders AS SELECT \* FROM small\_orders;

CREATE TABLE large\_orders AS SELECT \* FROM small\_orders;

CREATE TABLE special\_orders

( order\_id NUMERIC NOT NULL,

customer\_id NUMERIC NOT NULL,

order\_total NUMERIC,

other NUMERIC

);

CREATE TABLE orders

( order\_id NUMERIC NOT NULL,

customer\_id NUMERIC NOT NULL,

order\_total NUMERIC ,

other NUMERIC );

INSERT INTO orders SELECT 1, 1, 10000, 1 ;

INSERT INTO orders SELECT 1, 1, 20000, 1 ;

INSERT INTO orders SELECT 1, 1, 30000, 1 ;

INSERT INTO orders SELECT 1, 1, 210000, 1 ;

INSERT INTO orders SELECT 1, 1, 220000, 1 ;

INSERT INTO orders SELECT 1, 1, 110000, 1 ;

INSERT INTO orders SELECT 1, 1, 120000, 1 ;

INSERT INTO orders SELECT 1, 1, 130000, 1 ;

INSERT INTO orders SELECT 1, 1, 140000, 1 ;

INSERT INTO orders SELECT 1, 1, 140000, 1 ;

INSERT INTO orders SELECT 1, 1, 340000, 1 ;

2、使用INSERT ALL向不同表中插入数据。

INSERT ALL

WHEN ottl < 100000 THEN

INTO small\_orders

VALUES(oid, ottl, cid)

WHEN ottl > 100000 and ottl < 200000 THEN

INTO medium\_orders

VALUES(oid, ottl, cid)

WHEN ottl > 200000 THEN

into large\_orders

VALUES(oid, ottl, cid)

SELECT o.order\_id oid, o.customer\_id cid, o.order\_total ottl,

o.other

FROM orders o;

3、查询INSERT ALL结果。

select \* from small\_orders;

返回结果如下，则表示插入数据成功:

order\_id | customer\_id | order\_total

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

1 | 10000 | 1

1 | 20000 | 1

1 | 30000 | 1

(3 rows)

select \* from medium\_orders;

返回结果如下，则表示插入数据成功:

order\_id | customer\_id | order\_total

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

1 | 110000 | 1

1 | 120000 | 1

1 | 130000 | 1

1 | 140000 | 1

1 | 140000 | 1

(5 rows)

select \* from large\_orders;

返回结果如下，则表示插入数据成功:

order\_id | customer\_id | order\_total

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

1 | 210000 | 1

1 | 220000 | 1

1 | 340000 | 1

(3 rows)

select \* from special\_orders;

返回结果如下，则表示插入数据成功:

order\_id | customer\_id | order\_total | other

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

(0 rows)

4、使用INSERT FIRST向不同表中插入数据。

INSERT First

WHEN ottl > 10000 THEN

INTO small\_orders

VALUES(oid, ottl, cid)

INTO small\_orders

VALUES(oid, ottl, cid)

INTO small\_orders

VALUES(oid, ottl, cid)

WHEN ottl > 100000 THEN

INTO medium\_orders

VALUES(oid, ottl, cid)

WHEN ottl > 200000 THEN

into large\_orders

VALUES(oid, ottl, cid)

SELECT o.order\_id oid, o.customer\_id cid, o.order\_total ottl,

o.other

FROM orders o;

5、查询INSERT First结果。

select \* from small\_orders;

返回结果如下，则表示插入数据成功:

order\_id | customer\_id | order\_total

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

1 | 10000 | 1

1 | 20000 | 1

1 | 30000 | 1

1 | 20000 | 1

1 | 20000 | 1

1 | 20000 | 1

1 | 30000 | 1

1 | 30000 | 1

1 | 30000 | 1

1 | 210000 | 1

1 | 210000 | 1

1 | 210000 | 1

1 | 220000 | 1

1 | 220000 | 1

1 | 220000 | 1

1 | 110000 | 1

1 | 110000 | 1

1 | 110000 | 1

1 | 120000 | 1

1 | 120000 | 1

1 | 120000 | 1

1 | 130000 | 1

1 | 130000 | 1

1 | 130000 | 1

1 | 140000 | 1

1 | 140000 | 1

1 | 140000 | 1

1 | 140000 | 1

1 | 140000 | 1

1 | 140000 | 1

1 | 340000 | 1

1 | 340000 | 1

1 | 340000 | 1

(33 rows)

select \* from medium\_orders;

返回结果如下（与第三步查询结果相同时），则表示语句执行成功:

order\_id | customer\_id | order\_total

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

1 | 110000 | 1

1 | 120000 | 1

1 | 130000 | 1

1 | 140000 | 1

1 | 140000 | 1

(5 rows)

select \* from large\_orders;

返回结果如下（与第三步查询结果相同时），则表示语句执行成功:

order\_id | customer\_id | order\_total

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

1 | 210000 | 1

1 | 220000 | 1

1 | 340000 | 1

(3 rows)

select \* from special\_orders;

返回结果如下（与第三步查询结果相同时），则表示语句执行成功:

order\_id | customer\_id | order\_total | other

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

(0 rows)

### INSERT支持别名

**功能描述**

支持在INSERT语句中对表名使用别名。

**语法格式**

INSERT INTO table\_name t1(t1.col1,t1.col2);

**参数说明**

* table\_name：表名。
* t1：表的别名。
* col1：表的列1。
* col2：表的列2。

**注意事项**

无。

**兼容性**

完全兼容。

**示例**

1、创建测试表并插入数据。

create table table1 (col1 int,col2 int);

create table table2 (col1 int,col2 int);

insert into table2 values(1,2);

insert into table2 values(2,3);

insert into table1 t1(t1.col1,t1.col2) select t2.col1,t2.col2 from table2 t2;

2、查询测试表table1。

select \* from table1;

返回结果如下：

col1 | col2

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

1 | 2

2 | 3

(2 rows)

### PIVOT和UNPIVOT

**功能描述**

pivot为select语句的子语句，该语句是将行旋转为列的交叉表查询，在旋转过程中聚合函数、数据透视（pivot data）是整个操作的关键。数据透视时，会对数据透视列值列表中的每个项应用聚合运算符。数据透视列不能包含任意表达式。

在pivot（行转列）语法中pivot for in查询子句支持关键字做为别名，支持的关键字为month、content、percent、password、begin、select、drop和on。

unpivot为select语句的子语句，该语句允许将列旋转为行的交叉表查询。

**语法格式**

* pivot

语法：

SELECT …

FROM <table-expr>

PIVOT

(

aggregate-function(<column>) AS <alias>

FOR <pivot-column> IN(<value1> [AS <alias1>], ..., <valuen> [AS <aliasn>])

) AS <alias>

WHERE …

子句划分：

SELECT …

FROM …

PIVOT[XML]

(pivot\_clause

pivot\_for\_clause

pivot\_in\_clause)

WHERE …

* unpivot

语法：

SELECT …

FROM <table-expr>

UNPIVOT [INCLUDE|EXCULDE NULLS]

(

unpivot\_column1

FOR <unpivot-column2> IN ({(column\_name,...)|column\_name,}...)

) AS <alias\_name>

WHERE …

子句划分：

SELECT …

FROM …

UNPIVOT[INCLUDE|EXCLUDE NULLS]

(unpivot\_clause

unpivot\_for\_clause

unpivot\_in\_clause)

WHERE …

**参数说明**

* pivot\_clause：定义需要聚合的列（pivot是一个聚合操作），即上述语法(部分聚合函数)：

aggregate-function(<column>) AS <alias>

聚合函数必须指定group by子句，但pivot\_clause不包含显式group by子句。相反，pivot\_clause执行隐式group by。隐式分组基于pivot\_clause和 pivot\_for\_clause中未引用的所有列。

* pivot\_for\_clause：定义要分组和透视的列。即上述语法：

FOR <pivot-column>

* pivot\_in\_clause：定义pivot\_for\_clause中列的过滤器（即限制结果的值范围）。pivot\_in\_clause中每个值的聚合将被转置到单独的列中，并且pivot for in查询子句支持关键字做为别名：即上述语法：

IN(<value1> [AS <alias1>], ..., <valuen> [AS <aliasn>])

pivot\_in\_clause子句中与xml结合使用subquery以及any。

* INCLUDE|EXCLUDE NULLS：用来控制unpivot是否包含null的记录，默认是不包含null的。
* unpivot\_clause：行值的列名（即unpivot\_in\_clause中各列的值），即上述语法：

unpivot\_column1

* unpivot\_for\_clause：将原有多个列合并到单个列的列名（即把unpivot\_in\_clause中列名拼接），即上述语法：

FOR unpivot\_column2

* unpivot\_in\_clause：原始表的列名，即上述语法:

IN ({(column\_name,...)|column\_name,}...)

**注意事项**

pivot\_in\_clause：别名有两种方法

<value1> AS <alias1>

和

<value1> <alias1>

其中关键字别名只支持“ AS 不支持“

普通字符串别名“ AS ”和“ ”都支持。

**示例**

1、创建测试表并插入数据。

create table test\_pivot (timetest varchar2(20), test varchar2(20), score number, class number);

insert into test\_pivot values('张三','month', 78, 1);

insert into test\_pivot values('张三','content',87, 2);

insert into test\_pivot values('张三','content',90, 1);

insert into test\_pivot values('张三','content',60, 2);

insert into test\_pivot values('张三','percent',82, 3);

insert into test\_pivot values('张三','percent',88, 1);

insert into test\_pivot values('张三','percent',89, 2);

insert into test\_pivot values('张三','password',90, 2);

insert into test\_pivot values('张三','天文',90, 1);

insert into test\_pivot values('李四','地理',90, 2);

insert into test\_pivot values('李四','month',65, 2);

insert into test\_pivot values('李四','month',95, 1);

insert into test\_pivot values('李四','month',65, 3);

insert into test\_pivot values('李四','content',77, 1);

insert into test\_pivot values('李四','percent',65, 2);

insert into test\_pivot values('李四','percent',85, 2);

insert into test\_pivot values('李四','percent',95, 2);

insert into test\_pivot values('李四','password',85, 3);

insert into test\_pivot values('张三','Physics',100, 2);

insert into test\_pivot values('张三','Math',70, 1);

insert into test\_pivot values('李四','Physics',60, 2);

insert into test\_pivot values('李四','Chinese',65, 2);

2、将行转为列。

select \* from test\_pivot pivot (avg(score) for test in ('month' as month, 'content' as content, 'percent' as percent,'password' as password, '天文', '地理')) order by class;

返回结果为：

timetest | class | month | content | percent | password | '天文' | '地理'

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

李四 | 1 | 95 | 77 | | | |

张三 | 1 | 78 | 90 | 88 | | 90 |

张三 | 2 | | 73.5 | 89 | 90 | |

李四 | 2 | 65 | | 81.6666666666666667 | | | 90

张三 | 3 | | | 82 | | |

李四 | 3 | 65 | | | 85 | |

(6 rows)

3、创建一个测试表 testunpivot。

create table testunpivot(NAME varchar(10),yu int,shu int,ying int,wu int,zong int);

insert into testunpivot values('李四',65,75,85,95,320),('李五',61,71,81,91,324);

4、查询表中的数据。

select \* from testunpivot;

返回结果为：

name | yu | shu | ying | wu | zong

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

李四 | 65 | 75 | 85 | 95 | 320

李五 | 61 | 71 | 81 | 91 | 324

(2 rows)

5、将列转为行。

select \* from testunpivot unpivot((wen,li) for kemu in((yu,shu),(yu,wu),(ying,shu),(ying,wu)));

返回结果为：

name | zong | kemu | wen | li

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

李四 | 320 | yu\_shu | 65 | 75

李四 | 320 | yu\_wu | 65 | 95

李四 | 320 | ying\_shu | 85 | 75

李四 | 320 | ying\_wu | 85 | 95

李五 | 324 | yu\_shu | 61 | 71

李五 | 324 | yu\_wu | 61 | 91

李五 | 324 | ying\_shu | 81 | 71

李五 | 324 | ying\_wu | 81 | 91

(8 rows)

### Q’转义字符语法

**功能描述**

Vastbase G100实现Oracle中的Q’转义字符，该转义字符开启Oracle中的alternative quoting（引号替换）机制。

**语法格式**

Q|q 'quote\_delimiter c quote\_delimiter'

开启alternative quoting（引号替换）机制后，内部c的所有字符，包括单引号（'）、双引号（"）等均被作为字符处理。

**参数说明**

* Q or q ：表示将使用引号替代机制。这种机制允许为文本字符串使用广泛的分隔符。
* 最外面的''是两个单引号，分别位于左quote\_delimiter和右quote\_delimiter的前面和后面。
* c ：是用户字符集的任何成员。可以在由 c 字符组成的文本中包含引号（"）。还可以包括quote\_delimiter，只要它不是紧跟在单引号之后。
* quote\_delimiter：是除空格、制表符和回车之外的任何单字节或多字节字符。quote\_delimiter可以是单引号。但是，如果quote\_delimiter出现在文本本身中，请确保它后面不紧跟单引号。
* 如果左quote\_delimiter是 [, {, <, 或者 (，则右quote\_delimiter必须是相应的], }, >, 或者 )。在所有其他情况下，左和右quote\_delimiter必须是相同的字符。

**注意事项**

无。

**示例**

示例一：

select q'asdffwwda';

返回结果如下：

?column?

----------

sdffwwd

(1 row)

示例二：

select q'[sfeew[]we]';

返回结果为：

?column?

-----------

sfeew[]we

(1 row)

示例三：

1、创建测试表并插入数据。

create table qq\_test(name char(20));

insert into qq\_test values(q'd er'd 'wwd');

2、查询表中的数据。

select \* from qq\_test;

返回结果为：

name

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

er'd 'ww

(1 row)

### ROWID

**功能描述**

ROWID是一个伪列，该值表明了该行在Oracle数据库中的详细物理地址，通过ROWID，数据库可以快速定位某行具体数据的位置。

**注意事项**

* 该功能仅在数据库兼容模式为Oracle时能够使用（即创建DB时DBCOMPATIBILITY='A'），在其他数据库兼容模式下不能使用该特性。
* 子查询中也可以使用rowid。目前具体需求只有一层子查询。对于多层子查询嵌套的时候，在各层中都可以使用rowid不做强制要求。

**兼容性**

完全兼容。

**示例**

1、创建测试表并插入数据。

create table t01(id int,name varchar2(32));

insert into t01(id,name) values(1,'a'),(2,'b'),(3,'c'),(4,'d');

2、查询ROWID。

select rowid,t.\* from (select a.\* from t01 a) t;

返回结果为：

rowid | id | name

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

h1IAAA==AAAAAA==AQA= | 1 | a

h1IAAA==AAAAAA==AgA= | 2 | b

h1IAAA==AAAAAA==AwA= | 3 | c

h1IAAA==AAAAAA==BAA= | 4 | d

(4 rows)

3、子查询外层支持ROWID。

select rowid as rowid\_2,t.\* from (select rowid,a.\* from t01 a) t;

返回结果为：

rowid\_2 | rowid | id | name

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

h1IAAA==AAAAAA==AQA= | h1IAAA==AAAAAA==AQA= | 1 | a

h1IAAA==AAAAAA==AgA= | h1IAAA==AAAAAA==AgA= | 2 | b

h1IAAA==AAAAAA==AwA= | h1IAAA==AAAAAA==AwA= | 3 | c

h1IAAA==AAAAAA==BAA= | h1IAAA==AAAAAA==BAA= | 4 | d

(4 rows)

### ROWNUM

**功能描述**

ROWNUM伪列会在查询时返回行的编号。

**示例**

1、创建临时表并插入数据

CREATE TEMPORARY TABLE empsalary (

depname varchar,

empno bigint,

salary int,

enroll\_date date

);

INSERT INTO empsalary VALUES

('develop', 10, 5200, '2007-08-01'),

('sales', 1, 5000, '2006-10-01'),

('personnel', 5, 3500, '2007-12-10'),

('sales', 4, 4800, '2007-08-08'),

('personnel', 2, 3900, '2006-12-23'),

('develop', 7, 4200, '2008-01-01'),

('develop', 9, 4500, '2008-01-01'),

('sales', 3, 4800, '2007-08-01'),

('develop', 8, 6000, '2006-10-01'),

('develop', 11, 5200, '2007-08-15');

2、执行查询语句。

select rownum,depname,empno,salary from empsalary where rownum <3;

返回结果为：

rownum | depname | empno | salary

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

1 | develop | 10 | 5200

2 | sales | 1 | 5000

(2 rows)

### SAMPLE抽样采集语法

**功能描述**

sample字句允许从表中的随机数据样本中进行选择，而不是从整个表中选择。

**语法格式**

TABLESAMPLE sampling\_method ( argument [, ...] ) [ REPEATABLE ( seed ) ];

**参数说明**

* sampling\_method：采样方法。
* BERNOULLI：随机抽取表的数据行数据，抽样级别为数据行级别。使用全表扫描的采样方法，按采样参数百分比返回。相比于SYSTEM 具有更好的随机性，但是性能要差很多。
* SYSTEM：块级采样方法，随机抽取表的数据块上的数据，按采样参数百分比返回（被采样到的数据块内的所有记录都将被返回）。因此离散度不如BERNOULLI，但是效率高很多。
* SYSTEM\_ROWS：tsm\_system\_rows模块提供了表采样方法SYSTEM\_ROWS，它可以用SELECT命令的TABLESAMPLE子句中。这种表采样方法接受一个整数参数，它是要读取的最大行数。得到的采样将总是包含正好这么多行，除非该表中没有足够的行，在那种情况下整个表都会被选择出来。和内建的SYSTEM采样方法一样， SYSTEM\_ROWS执行块级别的采样，所以采样不是完全随机的，而是服从于聚簇效果，特别是只要求少量行时。
* 注意：SYSTEM\_ROWS不支持 REPEATABLE子句。
* 要采样的表的百分数，表示为一个0到100之间的百分数。取这个参数可以是任意的实数值表达式（除了BERNOULLI以及SYSTEM采样，其他的采样方法可能接受更多或者不同的参数）.
* REPEATABLE(seed)：采样随机种子，如果种子一样多，南无多次采样得到的结果是一样的。如果忽略REPEATABLE则每次都是使用新的seed值，得到不同的结果。seed取值为任何非空浮点值。

**注意事项**

无。

**示例**

1、创建测试表并插入数据。

create table test\_sample (id int4,message text);

insert into test\_sample(id,message) select n,md5(random()::text) from generate\_series(1,1000000) n;

2、使用SYSTEM采样方法进行选择。

select \* from test\_sample tablesample system(0.1);

返回结果如下，则表示采样成功：

id | message

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

108681 | de5d879fcd627af8a0fae0d4e8c99952

108682 | 348e18ff6e162f774569eb0107882023

108683 | 67520adbc0d5b6e0a98238559740abf0

108684 | 9046cc4117baba71ee616de9e389b065

108685 | 8300ccfb5072ae931be4d2eb82d8048a

108686 | 08189d0630f699d10f7cf62bec2837a8

108687 | 9035049b7ae8d70fbde7169b95ba91ec

108688 | a8d7edb1dd058b2fc2a06d34dc036721

108689 | 0a08c43a8d052f9e4cb6d66192b5aca5

108690 | 3e0b3d680e2c78382210170c35d97da4

108691 | 82b8edbaddd56a3533c166bf6110f11c

108692 | 2174f782ee8087a4a99744581e1c90df

108693 | 83682dfb7cb7ab64af11959226996b9e

108694 | 82cac0957b88e4851d5503dcf2ddedb9

108695 | 98d40852958d68f8dd469259a07953d3

108696 | 2c3df1ff75d2bf40cf2ef130427f0099

108697 | 306165368949c2add290394af980b20c

108698 | 985a277365a31ae767a53af700220ff6

108699 | 450ad7e79ded9887bf77adad59d1e30d

108700 | 224dc5087e8238f885d68d7edb8c0a1c

108701 | 0ce5ebfab7b11070689d31b1808038c1

108702 | 5ac2d7bedfe0eb9026916c13e4b32b53

108703 | 4fbe39926d7177b4ce1a418cdc1b3b1c

108704 | 9305630170cd961cce5d391dcdbc47f2

108705 | d0cfef23d155a8644c4f36fd235b60d2

108706 | 1fa533a6e707a7aae2957ef9ad65414c

108707 | 532bacb8035fbcd5e15ddd05180ad4a0

108708 | ed7a3959905c1a5654dc00fc0456c81f

108709 | e42540d240bb61539eaea8359943eee1

108710 | 63f390eb4c546b8eb616bc36f8d184ab

108711 | 01282b0d264b64711dc75a9e26f2cfa6

108712 | 2e62cd74be4073a5b8c1c5d5dbd613f1

108713 | 9a46e4b8b64c3d25b9a8b6c64e7f6931

108714 | c94a77af8e839dd0237560bb4b9e74ba

108715 | baa94146612c4723c4f8d10b95cef519

108716 | 5142bc434e8795a83fef656392cbb67f

Cancel request sent

3、使用SYSTEM采样方法进行选择。

select ctid,\* from test\_sample tablesample BERNOULLI(0.05);

返回结果如下，则表示采样成功过：

ctid | id | message

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

(15,95) | 1520 | fcf619fb848b2bbf4ef634f6e2917e53

(19,79) | 1884 | 9ad257e0a82ddd468a8b06d207b1066f

(21,7) | 2002 | 9c06503b99a9aa92148c3f0237de5384

(96,11) | 9131 | c27b199a6817f30215200af228e883de

(106,40) | 10110 | 194d225196af3794ffe57aabd67efc4f

(121,16) | 11511 | 7870876d5e85705f791714927e27fb2b

(162,5) | 15395 | 6dd96383b1ae18242073fbdd77b67d97

(165,22) | 15697 | 47a3ef5636891bff037b5d4d0daa86e1

(178,83) | 16993 | 6e3b6184e6435b39ad5c4be20e65c5d8

(182,6) | 17296 | 39f018ccf4645f4a173f3793b0219fe2

(193,62) | 18397 | af6316ee9907946673972245a94e4795

(210,79) | 20029 | 2f239bc81e70858a028efffac0875880

(238,10) | 22620 | f6f6d91b804ad3314d6dace36a0af3a5

(238,52) | 22662 | f305a3d35adf5f74d5e4fe8a5e8179cf

(315,44) | 29969 | 4793f9f7eae1f495231cc2c0a9589fd7

(318,50) | 30260 | 8f7bac87228b2ee35cb85715e24ed04c

(341,13) | 32408 | 2d9db260cf7b18fd69e991ed50be0695

(360,28) | 34228 | 7f405f35572d3d6b58c1ab27bcf30a87

(383,50) | 36435 | 38f3fd086c9a3ef3aa3e50e1b6282fdc

(447,48) | 42513 | 7de6905b59fcc6ee21e379e168d7c30c

(447,71) | 42536 | 69be111ed69babbafb7e1e8519be80b4

(466,38) | 44308 | b96a90360c187affd9ecde654490ba04

(481,50) | 45745 | 98e2ede63ff315e140248f72a2c9e896

(484,29) | 46009 | 16bd12b35fc82f98fd491757e3628377

(484,62) | 46042 | e151c49d5ef75098e429da334d89fbf8

(495,3) | 47028 | e73a88f5c4e20e88d0857b190ca1d6fb

(503,6) | 47791 | f083ec2b5b886e1bf2753e3861dc3aa0

(514,71) | 48901 | c2e1bdbaf0b972d7fe06e2fc8eb561d9

(540,70) | 51370 | 88fb8fb941646e9388af6af41bc9a8af

(563,14) | 53499 | 1ae6a5b32b5e5e2e831f6d0b18f6f507

(576,63) | 54783 | 0e311376ccc596be11716d237f3c677b

(590,54) | 56104 | b193f27a18bda8b53e1fac3c00a760af

(592,39) | 56279 | 3f58a044567207fd50f10c9632315248

(617,13) | 58628 | a677c86b9973003543618f1554250492

(623,27) | 59212 | 37a23fa568fc89175cc51933e541448c

(634,95) | 60325 | 38af9a0188c3bf09445b3b6ecdf86c28

Cancel request sent

### SELECT... FOR UPDATE

**功能描述**

SELECT ... FOR UPDATE语句用于锁定满足WHERE查询条件的行，以防止其他用户在您结束事务之前锁定或更新这些行。

**语法格式**

SELECT ... FOR UPDATE [ |WAIT integer| NOWAIT]

**参数说明**

* FOR UPDATE不跟任何参数：如果当前查询的结果中存在锁，则该查询一直等待（等待期间您无法执行其他操作），直到查询结果中的锁被释放后，立即返回查询结果。
* WAIT integer：如果当前查询的结果中存在锁，且超过integer秒后仍未释放锁，则报错，其中integer为等待超时时间。
* integer取值范围：正整数。
* integer取值0：等同NOWAIT。
* NOWAIT：当前查询的结果中存在锁，立即报错。

**注意事项**

* Vastbase G100数据库中可配置参数“update lockwait timeout”，该参数表示对单个锁的超时时间，默认为 120 秒。当select for update wait integer语句因为加锁需要进行等待时，其中integer值为等待所有行级锁的总计时长，因此，可能在总时长还未达指定值前，该语句因为等待单个行级锁超时而退出并解锁。
* Vastbase支持FOR UPDATE后跟表或者视图名，不支持跟列名。
* 该功能仅在数据库兼容模式为Oracle时能够使用（即创建DB时DBCOMPATIBILITY='A'），在其他数据库兼容模式下不能使用该特性。

**兼容性**

Vastbase G100 V2.2版本与Oracle 11G R2版本对比，不支持的选项为：[SKIP LOCKED]。

**示例**

1、创建并切换至兼容模式为ORACLE的数据库db\_oracle。

CREATE DATABASE db\_oracle dbcompatibility=’A’;

\c db\_oracle

2、创建测试表并插入数据。

create table test8(

id number,

orderid number,

productid number,

price number(10,2),

quantity number

);

insert into test8(id,orderid,productid,price,quantity)

values(1,1,1,6,10),

(2,1,2,4,5),

(3,1,3,10,2),

(4,2,1,3,6),

(5,2,2,4,6);

3、在会话1中，在事务中使用for update加锁，暂不提交。

begin;

select \* from test8 where quantity <6 for update;

id | orderid | productid | price | quantity

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

2 | 1 | 2 | 4.00 | 5

3 | 1 | 3 | 10.00 | 2

(2rows)

4、切换至会话2，使用for update nowait 获取同样数据时，提示获取锁失败。

select \* from test8 where quantity <6 for update nowait;

返回结果如下：

ERROR:could not obtain lock on row in relation “test8”

5、切换至会话2，使用for update wait 2 获取同样数据，超时（2s）提示获取锁失败。

elect \* from test8 where quantity <6 for update wait 2;

返回结果如下：

ERROR:could not obtain lock on row in relation “test8”

6、切换至会话2，使用for update wait 600 获取同样数据，n取足够大，阻塞等待会话1事务提交释放锁。

select \* from test8 where quantity <6 for update wait 600;

7、切换至会话1，在等待返回结果的600秒内，提交事务，释放锁。

commit;

8、切换至会话2，成功返回查询结果。

id | orderid | productid | price | quantity

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

2 | 1 | 2 | 4.00 | 5

3 | 1 | 3 | 10.00 | 2

(2rows)

## SELECT\_LIST中可以使用非分组列

**功能描述**

兼容MySQL、MariaDB中select\_list可以使用非分组列。支持在select语句的目标列中，可以使用group by和order by中未列出的列。

**语法格式**

[ WITH [ RECURSIVE ] with\_query [, ...] ]

SELECT [/\*+ plan\_hint \*/] [ ALL | DISTINCT [ ON ( expression [, ...] ) ] ]

{ \* | {expression [ [ AS ] output\_name ]} [, ...] }

[ FROM from\_item [, ...] ]

[ WHERE condition ]

[ GROUP BY grouping\_element [, ...] ]

[ HAVING condition [, ...] ]

[ WINDOW {window\_name AS ( window\_definition )} [, ...] ]

[ { UNION | INTERSECT | EXCEPT | MINUS } [ ALL | DISTINCT ] select ]

[ ORDER BY {expression [ [ ASC | DESC | USING operator ] | nlssort\_expression\_clause ] [ NULLS { FIRST | LAST } ]} [, ...] ]

[ LIMIT { [offset,] count | ALL } ]

[ OFFSET start [ ROW | ROWS ] ]

[ FETCH { FIRST | NEXT } [ count ] { ROW | ROWS } ONLY ]

[ {FOR { UPDATE | SHARE } [ OF table\_name [, ...] ] [ NOWAIT ]} [...] ];

**注意事项**

* 该功能仅在数据库兼容模式为MySQL时能够使用（即创建DB时DBCOMPATIBILITY='B'），在其他数据库兼容模式下不能使用该特性。
* vastbase\_sql\_mode='ONLY\_FULL\_GROUP\_BY'（默认值）时，select list中不可以使用非分组列。
* vastbase\_sql\_mode=''时，select list中可以使用非分组列。

**示例**

1、创建兼容MySQL的库db\_mysql，并进入。

CREATE DATABASE db\_mysql dbcompatibility='B';

\c db\_mysql

2、创建测试表student，并插入测试数据。

CREATE TABLE student(sid int,name varchar2(20),age int, class varchar2(20));

INSERT INTO student VALUES(1,'陈一',22,'java');

INSERT INTO student VALUES(2,'黄二',23,'python');

INSERT INTO student VALUES(3,'张三',21,'java');

INSERT INTO student VALUES(4,'李四',20,'C#');

INSERT INTO student VALUES(5,'王五',19,'python');

INSERT INTO student VALUES(6,'赵六',21,'java');

INSERT INTO student VALUES(7,'钱七',21,'java');

INSERT INTO student VALUES(8,'孙八',20,'C++');

INSERT INTO student VALUES(9,'杨九',19,'C++');

3、设置vastbase\_sql\_mode并验证。

set vastbase\_sql\_mode='';

SHOW vastbase\_sql\_mode;

当结果显示如下信息，则表示设置完成。

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

(1 row)

4、验证结果。

SELECT \* FROM student GROUP BY class;

当结果显示如下信息，则表示验证完成。

sid | name | age | class

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

8 | 孙八 | 20 | C++

4 | 李四 | 20 | C#

2 | 黄二 | 23 | python

1 | 陈一 | 22 | java

(4 rows)

## SELECT语句使用MONTH、CONTENT、PERCENT，PASSWORD关键字做列别名

**功能描述**

select语句中已支持部分关键字作为目标字段的别名，Vastbase G100 V2.2版本增加关键字month、content、percent和password作为目标字段的别名。

**语法格式**

SELECT [/\*+ plan\_hint \*/] [ ALL | DISTINCT [ ON ( expression [, ...] ) ] ]

{ \* | {expression [output\_name ]} [, ...] }

[ FROM from\_item [, ...] ]

[ WHERE condition ]

...

**参数说明**

output\_name：用month、content、percent和password关键字给输出字段起别名。

**注意事项**

该功能仅在数据库兼容模式为Oracle时能够使用（即创建DB时DBCOMPATIBILITY='A'），在其他数据库兼容模式下不能使用该特性。

**示例**

1、创建兼容Oracle的数据库db\_oracle。

CREATE DATABASE db\_oracle DBCOMPATIBILITY='A';

\c db\_oracle

2、验证功能。

select 1 month,2 content,3 percent,4 password from dual;

结果返回如下：

month | content | percent | password

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

1 | 2 | 3| 4

(1 rows)

## WITH FUNCTION子句

**功能描述**

with function功能实现了在with语句中定义临时函数，在后续的子语句中可以重复使用该函数。

**语法格式**

WITH { FUNCTION func\_name ( [ [ argmode ] [ argname ] argtype [ { DEFAULT | = } default\_expr ] [, ...] ] )

[ RETURNS rettype

| RETURNS TABLE ( column\_name column\_type [, ...] ) ] AS ‘definition’ } [, …]

{ SelectStmt }

**参数说明**

* func\_name：自定义的函数名称。取值范围：字符串，要符合标识符命令规范。
* argmode：函数参数的模式。取值范围：IN，OUT，INOUT或VARIADIC（用于声明数组类型的参数），缺省值是IN。只有OUT模式的参数后面能跟VARIADIC。并且 OUT和INOUT模式的参数不能用在RETURNS TABLE的函数定义中。
* argname：函数参数的名称。取值范围：字符串，要符合标识符命令规范。
* argtype：函数参数的类型。
* rettype：函数返回值的数据类型。如果存在OUT或IN OUT参数，可以省略RETURNS子句。如果出现了，那么它必须和输出参数隐含的结果类型兼容：如果有多个输出参数，必须是 RECORD， 如果只有一个输出参数，则与其相同。
* column\_name：字段名称。
* column\_type：字段类型。
* definition：一个定义函数的字符串长，含义取决于语言。它可以是一个内部函数名称、一个指向某个目标文件的路径、一个SQL查询、一个过程语言文本。

**示例**

使用WITH FUNCTION字句调用函数。

WITH FUNCTION withfunc (x integer) RETURNS INTEGER AS $$

BEGIN

RETURN x+1;

END

$$,

FUNCTION withfunc2(x INTEGER, y INTEGER) RETURNS INTEGER AS $$

BEGIN

RETURN x+y;

END;

$$,

FUNCTION withfunc3(x TEXT) RETURNS TEXT AS $$

BEGIN

RETURN x || '-test';

END;

$$

SELECT withfunc(1),withfunc2(2,3),withfunc3('4');

当结果显示如下，则表示函数调用成功：

withfunc | withfunc2 | withfunc3

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

2 | 5 | 4-test

(1 row)

## 递归查询语法

**功能描述**

Vastbase兼容Oracle递归查询START WITH...CONNECT BY...语法。

**语法格式**

START WITH condition CONNECT BY condition

CONNECT BY condition [ START WITH condition ]

**参数说明**

condition：任何条件语句。condition的格式必须为：

[prior column1 = column2 | column1 = prior column2]

**兼容性**

Vastbase G100 V2.2版本与Oracle 11G R2版本对比，语法中不支持的选项为[NOCYCLE]。

**示例**

1、创建测试表并插入数据。

create table employee(

emp\_id number(18),

lead\_id number(18),

emp\_name varchar2(200),

salary number(10,2),

dept\_no varchar2(8)

);

insert into employee values('1',0,'king','1000000.00','001');

insert into employee values('2',1,'jack','50500.00','002');

insert into employee values('3',1,'arise','60000.00','003');

insert into employee values('4',2,'scott','30000.00','002');

insert into employee values('5',2,'tiger','25000.00','002');

insert into employee values('6',3,'wudde','23000.00','003');

insert into employee values('7',3,'joker','21000.00','003');

2、查询以emp\_id为3开始的节点的所有直属节点。

select emp\_id,lead\_id,emp\_name

from employee

start with lead\_id=3

connect by prior emp\_id = lead\_id;

返回结果为：

emp\_id | lead\_id | emp\_name

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

6 | 3 | wudde

7 | 3 | joker

(2 rows)

#### 支持connect\_by\_isleaf

**功能描述**

递归查询中的一个伪列，用于指示当前行是否为树的叶子节点（即没有子节点的节点），如果是返回1，否则返回0。

**语法格式**

SELECT cloumn,exper.. CONNECT\_BY\_ISLEAF "IsLeaf",[LEVEL],

from table

start with condition

connect by [prior column1 = column2 | column1 = prior column2];

**参数说明**

LEVEL：一个伪列（可选），代表当前这个节点所在的层级，对于根节点，level返回1。

**注意事项**

无。

**示例**

1、创建测试表并插入数据。

CREATE TABLE EMP

(EMPNO NUMBER(4) NOT NULL,

ENAME VARCHAR2(10),

JOB VARCHAR2(9),

MGR NUMBER(4),

HIREDATE DATE,

SAL NUMBER(7, 2),

COMM NUMBER(7, 2),

DEPTNO NUMBER(2));

INSERT INTO EMP VALUES (7369, 'SMITH', 'CLERK', 7902,TO\_DATE('17-DEC-1980', 'DD-MON-YYYY'), 800, NULL, 20);

INSERT INTO EMP VALUES (7499, 'ALLEN', 'SALESMAN', 7698,TO\_DATE('20-FEB-1981', 'DD-MON-YYYY'), 1600, 300, 30);

INSERT INTO EMP VALUES (7521, 'WARD', 'SALESMAN', 7698,TO\_DATE('22-FEB-1981', 'DD-MON-YYYY'), 1250, 500, 30);

INSERT INTO EMP VALUES (7566, 'JONES', 'MANAGER', 7839,TO\_DATE('2-APR-1981', 'DD-MON-YYYY'), 2975, NULL, 20);

INSERT INTO EMP VALUES (7654, 'MARTIN', 'SALESMAN', 7698,TO\_DATE('28-SEP-1981', 'DD-MON-YYYY'), 1250, 1400, 30);

INSERT INTO EMP VALUES (7698, 'BLAKE', 'MANAGER', 7839,TO\_DATE('1-MAY-1981', 'DD-MON-YYYY'), 2850, NULL, 30);

INSERT INTO EMP VALUES (7782, 'CLARK', 'MANAGER', 7839,TO\_DATE('9-JUN-1981', 'DD-MON-YYYY'), 2450, NULL, 10);

INSERT INTO EMP VALUES (7788, 'SCOTT', 'ANALYST', 7566,TO\_DATE('09-DEC-1982', 'DD-MON-YYYY'), 3000, NULL, 20);

INSERT INTO EMP VALUES (7839, 'KING', 'PRESIDENT', NULL,TO\_DATE('17-NOV-1981', 'DD-MON-YYYY'), 5000, NULL, 10);

INSERT INTO EMP VALUES (7844, 'TURNER', 'SALESMAN', 7698,TO\_DATE('8-SEP-1981', 'DD-MON-YYYY'), 1500, 0, 30);

INSERT INTO EMP VALUES (7876, 'ADAMS', 'CLERK', 7788,TO\_DATE('12-JAN-1983', 'DD-MON-YYYY'), 1100, NULL, 20);

INSERT INTO EMP VALUES (7900, 'JAMES', 'CLERK', 7698,TO\_DATE('3-DEC-1981', 'DD-MON-YYYY'), 950, NULL, 30);

INSERT INTO EMP VALUES (7902, 'FORD', 'ANALYST', 7566,TO\_DATE('3-DEC-1981', 'DD-MON-YYYY'), 3000, NULL, 20);

INSERT INTO EMP VALUES (7934, 'MILLER', 'CLERK', 7782,TO\_DATE('23-JAN-1982', 'DD-MON-YYYY'), 1300, NULL, 10);

2、查询当前行是否为叶子节点

SELECT ename, CONNECT\_BY\_ISLEAF "IsLeaf",LEVEL

FROM emp

WHERE LEVEL <= 3 AND deptno= 10

START WITH empno = 7839

CONNECT BY PRIOR deptno = mgr AND LEVEL <= 4

order by ename,"IsLeaf";

返回结果为：

ename | IsLeaf | level

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

KING | 1 | 1

(1 row)

#### 支持order siblings by

**功能描述**

排序子句，可对结果集进行排序，仅用于层次查询connect by中，可以让兄弟节点（即同一父级的兄弟）按照指定的列进行排序（升序或者降序）。

**语法格式**

SELECT cloumn,exper..

FROM table

start with condition

connect by[prior column1 = column2 | column1 = prior column2]

ORDER [ SIBLINGS ] BY {expression [ [ ASC | DESC | USING operator ] | nlssort\_expression\_clause

**参数说明**

NULLS FIRST和NULLS LAST：两者都是用来控制空值在输出结果中出现的顺序。

* NULLS FIRST表示在排序表达式上的结果为空值的行总是出现在输出结果的最前面。
* NULLS LAST表示在排序表达式上的结果为空值的行总是出现在输出结果的最后面。

**注意事项**

无。

**示例**

1、创建测试表并插入数据。

CREATE TABLE family\_tree(id int primary key,appellation varchar(30),parent\_id int,age int);

insert into family\_tree values(1,'爷爷',0,88);

insert into family\_tree values(2,'大伯',1,66);

insert into family\_tree values(5,'堂兄',2,32);

insert into family\_tree values(6,'堂妹',2,24);

insert into family\_tree values(3,'爸爸',1,58);

insert into family\_tree values(7,'小明',3,26);

insert into family\_tree values(4,'小姑',1,52);

insert into family\_tree values(8,'表姐',4,28);

insert into family\_tree values(9,'表弟',4,20);

2、对同已父级的兄弟进行排序。

SELECT \* FROM family\_tree

CONNECT BY PRIOR id= parent\_id

start with appellation ='爷爷'

order siblings by age asc;

返回结果为：

id | appellation | parent\_id | age

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

1 | 爷爷 | 0 | 88

4 | 小姑 | 1 | 52

9 | 表弟 | 4 | 20

8 | 表姐 | 4 | 28

3 | 爸爸 | 1 | 58

7 | 小明 | 3 | 26

2 | 大伯 | 1 | 66

6 | 堂妹 | 2 | 24

5 | 堂兄 | 2 | 32

(9 rows)

#### 支持connect\_by\_root

**功能描述**

一个一元运算符，可用于限定列，仅在递归查询中有效，可以根据当前行返回被视为根节点的行的列值，即就是可以根据当前行得到其根节点的列值。

CONNECT\_BY\_ROOT运算符可在SELECT列表、WHERE子句、GROUP BY子句、HAVING子句、ORDER BY子句和ORDER SIBLINGS BY子句中使用，只要SELECT命令用于分层查询。不能在分层查询的CONNECT BY子句或START WITH子句中使用。可将CONNECT\_BY\_ROOT应用于一个涉及列的表达式，但这样做时，该表达式必须用圆括号括起。

**语法格式**

SELECT CONNECT\_BY\_ROOT cloum,expre..

from table

start with condition

connect by [prior column1 = column2 | column1 = prior column2];

**注意事项**

无。

**示例**

1、创建测试表并插入数据。

DROP TABLE family\_tree;

CREATE TABLE family\_tree(id int primary key,appellation varchar(30),parent\_id int,age int);

insert into family\_tree values(1,'爷爷',0,88);

insert into family\_tree values(2,'大伯',1,66);

insert into family\_tree values(5,'堂兄',2,32);

insert into family\_tree values(6,'堂妹',2,24);

insert into family\_tree values(3,'爸爸',1,58);

insert into family\_tree values(7,'小明',3,26);

insert into family\_tree values(4,'小姑',1,52);

insert into family\_tree values(8,'表姐',4,28);

insert into family\_tree values(9,'表弟',4,20);

2、对同已父级的兄弟进行排序。

SELECT connect\_by\_root parent\_id root, connect\_by\_isleaf isleaf,id,appellation,age

FROM family\_tree

CONNECT BY PRIOR id= parent\_id

start with appellation ='爷爷'

order siblings by age;

返回结果为：

root | isleaf | id | appellation | age

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

0 | 0 | 1 | 爷爷 | 88

0 | 0 | 4 | 小姑 | 52

0 | 1 | 9 | 表弟 | 20

0 | 1 | 8 | 表姐 | 28

0 | 0 | 3 | 爸爸 | 58

0 | 1 | 7 | 小明 | 26

0 | 0 | 2 | 大伯 | 66

0 | 1 | 6 | 堂妹 | 24

0 | 1 | 5 | 堂兄 | 32

(9 rows)

## 虚拟列语法

**功能描述**

创建数据表时（CREATE TABLE）可指定字段为虚拟列，在修改数据表（ALTER TABLE）可添加虚拟列字段。

**语法格式**

GENERATED ALWAYS AS ( generation\_expr ) STORED;

**参数说明**

generation\_expr：表达式

**示例**

1、创建测试表并插入数据。

create table t\_virtual(c1 int,c2 int,v\_c3 int GENERATED ALWAYS AS(c1 + c2) stored);

insert into t\_virtual values(1, 2);

select \* from t\_virtual;

返回结果为：

c1 | c2 | v\_c3

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

1 | 2 | 3

(1 row)

2、更新表的内容。

update t\_virtual set c1=2;

select \* from t\_virtual;

c1 | c2 | v\_c3

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

2 | 2 | 4

(1 row)

3、删除表中的字段。

ALTER TABLE t\_virtual drop c1;

select \* from t\_virtual;

c2

----

2

(1 row)

## 字符串CHAR/VARCHAR/VARCHAR2指定BYTE和CHAR关键字

**功能描述**

常见的字符串类型varchar2、char、varchar数据类型的字段长度默认为字节byte，常见写法如varchar(10)。而在UTF-8字符集中，中文汉字一个字符占用了3字节，如果该字段指定限制字符串长度为10，实际上只能插入3个UTF-8汉字字符。

为了解决这种情况，Vastbase G100V2.2版本支持在创建字段的时候显式定义字段的长度单位为byte或者character（即在定义varchar2、char、varchar数据类型时支持指定byte和character关键字）。

* byte为按字节来定义字段长度，是该数据类型的缺省值。
* character为按字符来定义字段长度，可以简写成char。

**语法格式**

varchar(n byte)或者varchar(n character)

**参数说明**

* n character：表明该字段允许输入n个字符，等于n char。
* n byte：表明该字段允许输入n个字节。

**注意事项**

无。

**示例**

创建测试表。

create table test\_char(

C1 char(10 char), --10 character ，允许输入最多10个字符，包括英文和中文

C2 char(10 byte), --10 byte，允许输入最多10个字节，能允许输入的字符数需要以字符具体所占字节数为准计算

C3 varchar(10 character), --10 character

C4 varchar(10), --10 byte

C5 char, --不指定限制长度，默认允许输入最多1个字节

C6 varchar, --不指定限制长度，默认不限制输入长度

C7 varchar2(10 character), --10 character

C8 varchar2(10) --10 byte

);

返回结果如下，则表示字符串char/varchar/varchar2指定byte和char关键字：

CREATE TABLE

## 多个子查询时不需要指定表别名

**功能描述**

在Oracle兼容模式下，Vastbase G100在使用多个子查询时可以省略表的别名。

**语法格式**

SELECT \* FROM

(SELECT ... FROM table\_name WHERE ...),

(SELECT ... FROM table\_name WHERE ...),

(SELECT ... FROM table\_name WHERE ...),

...

**参数说明**

**table\_name**

子查询中涉及到的表名。每个子句中的表名可以是相同的，也可以是不同的。

**注意事项**

* 该功能仅在数据库兼容模式为Oracle时能够使用（即创建DB时DBCOMPATIBILITY='A'），在其他数据库兼容模式下不能使用该特性。
* Vastbase中不支持多个子查询别名相同。

**示例**

1、创建数据库my\_test，检查兼容性。

CREATE DATABASE dbtest DBCOMPATIBILITY 'A';

\c dbtest;

show sql\_compatibility;

2、创建测试表，并插入数据。

create table t\_test(id int,name varchar);

insert into t\_test values(1,'jack');

insert into t\_test values(2,'grace');

insert into t\_test values(3,'frank');

3、进行多个不带表别名的子查询。

select \* from (select id from t\_test where id=1),

(select name from t\_test where id =2);

返回结果如下，表示查询成功。

id | name

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

1 | grace

(1 row)

# 宏变量

**功能描述**

系统提供的宏变量包括dbtimezone,sessiontimezone和systimestamp。

* dbtimezone：返回数据库时区。
* sessiontimezone：返回会话时区。
* systimestamp：返回本机数据库上当前系统日期和时间，精确到毫秒。

**示例**

* dbtimezone

select dbtimezone;

返回结果为：

dbtimezone

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

PRC

(1 row)

* sessiontimezone

select sessiontimezone;

返回结果为：

sessiontimezone

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

PRC

(1 row)

* systimestamp

select systimestamp;

返回结果为：

sysdate

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

2022-07-01 16:43:47

(1 row)

# PL/SQL

## CREATE FUNCTION中支持PIPELINED和PIPE ROW语法

**功能描述**

CREATE FUNCTION中支持PIPELINED和PIPE ROW语法，让函数可以返回一个集合，在函数中还可以使用PIPE ROW语句，用来返回集合中的单个元素。

**语法格式**

CREATE [ OR REPLACE ] FUNCTION func\_name ( ... )

RETURNS rettype PIPELINED

...

**参数说明**

rettype：函数返回值的数据类型。

**注意事项**

PIPE ROW必须与PIPELINED语法配套使用。

**示例**

1、创建函数。

CREATE OR REPLACE FUNCTION test()

RETURNS int

PIPELINED

AS $$

DECLARE

result RECORD;

row RECORD;

BEGIN

pipe row(1);

pipe row(2);

return;

END

$$

LANGUAGE plpgsql;

2、查询结果。

select \* from table(test());

返回结果如下：

test

------

1

2

(2 rows)

## EXCEPTION\_INIT预定义异常处理

**功能描述**

使用exception\_init可以将自定义异常变量名称预自定义的异常的错误代码关联。

**语法格式**

PRAGMA EXCEPTION\_INIT (exception\_name,error\_code)

**参数说明**

* exception\_name：自定义的异常变量名。
* error\_code：自定义错误编码。

**注意事项**

无。

**示例**

1、创建测试表。

create table t2(id int not null,name varchar2(50));

2、声明预定义的异常处理。

declare

e\_missingnull exception;

PRAGMA EXCEPTION\_INIT(e\_missingnull ,-1400);

begin

insert into t2(id) values(null);

exception

when e\_missingnull then

dbms\_output.put\_line(SQLERRM);

rollback;

end;

/

返回结果为：

ERROR: null value in column "id" violates not-null constraint

DETAIL: Failing row contains (null, null).

CONTEXT: SQL statement "insert into t2(id) values(null)"

PL/pgSQL function inline\_code\_block line 5 at SQL statement

## PL/pgSQL-TYPE ... IS RECORD

**功能描述**

Vastbase G100 V2.2在PL/pgSQL-TYPE... IS RECORD(...)中支持嵌套定义和使用RECORD类型。

RECORD类型（记录类型）将多个基本数据类型变量组合成一个整体，作为一个复合数据类型使用，可以看做是一种用户自定义数据类型。

**语法格式**

TYPE 类型名称 IS RECORD (成员名称 数据类型 [NOT NULL] :=[默认值][表达式],..);

**参数说明**

数据类型为：

* RECORD类型。
* 任意SQL兼容的数据类型（varchar、varchar2、char、int、number……）。
* 行类型table\_name%ROWTYPE。
* 复制类型（Copying Types）variable%TYPE。
* 支持nested record类型，nested record指在record类型中嵌套record类型。

**注意事项**

无。

**示例**

1、预先创建测试表并插入数据。

CREATE TABLE EMP

(

EMPNO NUMBER(4) NOT NULL,

ENAME VARCHAR2(10),

JOB VARCHAR2(9),

MGR NUMBER(4),

HIREDATE DATE,

SAL NUMBER(7, 2),

COMM NUMBER(7, 2),

DEPTNO NUMBER(2)

);

INSERT INTO EMP VALUES (7369, 'SMITH', 'CLERK', 7902,TO\_DATE('17-DEC-1980', 'DD-MON-YYYY'), 800, NULL, 20);

INSERT INTO EMP VALUES (7499, 'ALLEN', 'SALESMAN', 7698,TO\_DATE('20-FEB-1981', 'DD-MON-YYYY'), 1600, 300, 30);

INSERT INTO EMP VALUES (7521, 'WARD', 'SALESMAN', 7698,TO\_DATE('22-FEB-1981', 'DD-MON-YYYY'), 1250, 500, 30);

CREATE TABLE DEPT

( DEPTNO NUMBER(2),

DNAME VARCHAR2(14),

LOC VARCHAR2(13)

);

INSERT INTO DEPT VALUES (10, 'ACCOUNTING', 'NEW YORK');

INSERT INTO DEPT VALUES (20, 'RESEARCH', 'DALLAS');

INSERT INTO DEPT VALUES (30, 'SALES', 'CHICAGO');

INSERT INTO DEPT VALUES (40, 'OPERATIONS', 'BOSTON ');

2、定义和使用嵌套的记录类型1。

do language plpgsql $$ DECLARE

TYPE dept\_type IS RECORD (

deptno dept.deptno%TYPE := 80, -- 定义默认值

dname dept.dname%TYPE ,

loc dept.loc%TYPE

) ;

TYPE emp\_type IS RECORD (

empno emp.empno%TYPE ,

ename emp.ename%TYPE ,

job emp.job%TYPE ,

hiredate emp.hiredate%TYPE ,

sal emp.sal%TYPE ,

comm emp.comm%TYPE ,

dept dept\_type

) ;

v\_emp emp\_type ;

BEGIN

SELECT e.empno,e.ename,e.job,e.hiredate,e.sal,e.comm,d.deptno,d.dname,d.loc INTO

v\_emp.empno,v\_emp.ename,v\_emp.job,v\_emp.hiredate,v\_emp.sal,v\_emp.comm,

v\_emp.dept.deptno,v\_emp.dept.dname,v\_emp.dept.loc

FROM emp e, dept d WHERE e.deptno=d.deptno AND e.empno=7369 ;

RAISE notice '雇员编号：%, 姓名：%, ，职位：%, 雇佣日期：%, 基本工资：%, 佣金：%', v\_emp.empno, v\_emp.ename, v\_emp.job, TO\_CHAR(v\_emp.hiredate,'yyyy-mm-dd'), v\_emp.sal, NVL(v\_emp.comm,0);

RAISE notice '部门编号：%, 名称：%, 位置：%', v\_emp.dept.deptno, v\_emp.dept.dname, v\_emp.dept.loc;

END ;

$$;

返回结果为：

NOTICE: 雇员编号：7369, 姓名：SMITH, ，职位：CLERK, 雇佣日期：1980-12-17, 基本工资：800, 佣金：0

NOTICE: 部门编号：20, 名称：RESEARCH, 位置：DALLAS

ANONYMOUS BLOCK EXECUTE

3、定义和使用嵌套的记录类型2。

do language plpgsql $$

DECLARE

TYPE dept\_type IS RECORD(

deptno dept.deptno%TYPE := 80,

dname dept.dname%TYPE,

loc dept.loc%TYPE

);

TYPE emp\_type IS RECORD(

empno emp.empno%TYPE,

ename emp.ename%TYPE,

job emp.job%TYPE,

hiredate emp.hiredate%TYPE,

sal emp.sal%TYPE,

comm emp.comm%TYPE,

dept dept\_type

);

v\_emp emp\_type;

v\_dept dept\_type;

BEGIN

v\_emp.dept.deptno := 90;

v\_emp.dept.dname := 'ACCOUNTING2';

v\_emp.dept.loc := 'NEW YORK';

v\_dept := v\_emp.dept;

RAISE notice '部门编号：%，名称：%，位置：%',v\_dept.deptno,v\_dept.dname,v\_dept.loc;

END;

$$;

返回结果为：

NOTICE: 部门编号：90，名称：ACCOUNTING2，位置：NEW YORK

ANONYMOUS BLOCK EXECUTE

## RAISE\_APPLICATION\_ERROR存储过程

**功能描述**

该存储过程用于声明自定义异常编号和错误信息。

**使用流程**

1、RAISE\_APPLICATION\_ERROR定义异常编号和错误信息。

2、通过exception\_init关联异常编号。

3、捕获异常变量。

**语法格式**

RAISE\_APPLICATION\_ERROR(error\_code,message[,{TURE | FALSE}]);

**参数说明**

* error\_code：用户自定义异常变量编号，范围是-20000~20999。
* message：错误信息，允许最长2k的字符串。
* [{TURE | FALSE}]：默认值是TRUE，输出自定义错误信息。当设置为FALSE，输出系统错误信息。

**注意事项**

无。

**示例**

1、声明自定义异常编号和错误信息。

create or replace procedure account\_status(

due\_date DATE,

today DATE

)authid definer

is

begin

if due\_date < today then

RAISE\_APPLICATION\_ERROR(-20000,'Account past due.');

end if ;

end;

/

2、调用存储过程。

declare

past\_due exception;

PRAGMA EXCEPTION\_INIT(past\_due,-20000);

begin

account\_status('2021-10-22','2021-11-22');

exception

when past\_due then

dbms\_output.put\_line(SQLERRM);

end;

/

返回结果为：

Account past due.

ANONYMOUS BLOCK EXECUTE

## 创建函数、存储过程、触发器和包时进行编译检查

**功能描述**

在创建PL/pgSQL对象时，进行语义检查。增加GUC参数vbplsql\_check，当该参数设置为on或者true生效，在创建PL/pgSQL对象时进行语义检查，否则，不检查。检查功能如下：

* 创建函数需要返回非void时候，判断程序是否有return返回值，如果没有return返回值则创建失败，涉及到的statement有if、case语句。
* 字符串拼接如果使用的双引号进行字符串拼接，则创建对象失败。
* 如果DMshowu 句中出现表不存在的情况，则创建对象失败，不包含动态执行语句。
* 如果DML语句出现语法错误的情况，则创建对象失败，不包含动态执行语句。

**设置方法**

设置GUC参数vbplsql\_check = true或者on。

set vbplsql\_check = true;

**注意事项**

GUC参数 vbplsql\_check和enable\_global\_plancache 不能同时设置为on，否则会出现报错，设置失败。

**示例**

示例一：创建触发器，判断是否有return返回。

1、创建测试表。

drop table if exists t1\_z;

drop function if exists tri\_func;

create table t1\_z(w1 date);

2、创建函数。

CREATE OR REPLACE FUNCTION tri\_func() RETURNS TRIGGER AS

$$

DECLARE

BEGIN

raise notice 'trigger begin: TG\_OP is %',TG\_OP;

if TG\_OP='INSERT' then

insert into t1\_z values('2022-05-19'::date+1);

end if;

END $$ LANGUAGE 'plpgsql';

返回结果如下，则表示函数创建成功:

CREATE FUNCTION

3、创建触发器。

CREATE TRIGGER be\_tri

BEFORE INSERT OR DELETE OR UPDATE

ON t1\_z

FOR EACH ROW

EXECUTE PROCEDURE tri\_func();

创建触发器失败，提示缺少返回值，返回结果如下：

ERROR: control reached end of function without RETURN

示例二：存储过程-DML语句中出现表不存在的情况，则创建对象失败。

1、删除表。

drop table if exists t1\_z;

drop procedure if exists func;

2、创建存储过程。

create or replace procedure func()

as

declare

begin

insert into t1\_z values(to\_date('2022-05-19 135623.123','YYYY-MM-DD HH24MISS.FF'));

end;

/

存储过程创建失败，提示表不存在，返回结果如下：

ERROR: relation "t1\_z" does not exist on node1

LINE 1: insert into t1\_z values(to\_date('2022-05-19 135623.123','YYY...

^

QUERY: insert into t1\_z values(to\_date('2022-05-19 135623.123','YYYY-MM-DD HH24MISS.FF'))

3、调用存储过程。

select func();

调用存储过程失败，提示不存在，返回结果如下：

ERROR: function func() does not exist

LINE 1: select func();

^

HINT: No function matches the given name and argument types. You might need to add explicit type casts.

CONTEXT: referenced column: func

示例三：存储过程-DML语句出现语法错误，则创建对象失败。

1、创建测试表。

drop table if exists t1\_z;

drop procedure if exists func;

create table t1\_z(col1 date);

2、创建存储过程（其中insert语句语法错误）。

create or replace procedure func()

as

begin

insert into t1\_z ('2022-05-19');

end;

/

存储过程创建失败，提示语法错误,返回结果如下：

ERROR: syntax error at or near "'2022-05-19'"

LINE 3: insert into t1\_z ('2022-05-19');

^

QUERY: DECLARE

begin

insert into t1\_z ('2022-05-19');

end

## 定义游标支持INTO关键字

**功能描述**

支持cursor xxx for/is select ... into ...用法。

**语法格式**

DECLARE

CURSOR name FOR SELECT columns INTO ... FROM table\_name;

**参数说明**

INTO ... ：可以是声明的变量，数量与SELECT ...一致。

**注意事项**

无。

**示例**

1、创建测试表并插入数据。

CREATE TABLE stu\_info

(

id INT NOT NULL DEFAULT 0,

name TEXT

);

INSERT INTO stu\_info (id,name) VALUES(1,'name01');

INSERT INTO stu\_info (id,name) VALUES(2,'name02');

INSERT INTO stu\_info (id,name) VALUES(3,'name03');

2、定义一个游标。

do language plpgsql $$

declare

v\_id int;

v\_name TEXT;

cursor cur1 for

select id,name into v\_id,v\_name from stu\_info where id>1;

begin

open cur1;

loop

fetch cur1 into v\_id,v\_name;

if found then

update stu\_info set name='test' where id = v\_id;

else

exit;

end if;

end loop;

end;

$$;

3、查询表中数据。

select \* from stu\_info;

返回结果如下，则表示游标定义成功：

id | name

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

1 | name01

2 | test

3 | test

(3 rows)

## 关联数组

**功能描述**

关联数组也称索引表，是一组key-value键值对，每个key都是一个唯一索引。Vastbase支持在PL/SQL中定义关联数组类型，以及定义和使用关联数组变量，并支持关联数组元素的赋值与访问。

**语法格式**

TYPE typename IS TABLE OF datatype INDEX BY subtype;

**参数说明**

* typename：类型名称。
* datatype：数组元素支持Vastbase数据类型包括数值、字符、时间以及自定义类型，不支持REF CURSOR和NCLOB类型。
* subtype：数组下标的类型，目前支持varchar2，varchar2(n)，varchar，varchar(n)，pls\_integer（整型数字）。

**注意事项**

赋值与访问关联数组变量，访问的下标类型可以是整型数字（INDEX BY PLS\_INTEGER），varchar以及varchar2类型下标。

**示例**

定义关联数组赋值与访问关联数组变量。

CREATE OR REPLACE FUNCTION Fun() RETURNS int AS $$

DECLARE

TYPE ARRAY\_INTEGER IS TABLE OF int index by varchar(20); --定义类型

ARRINT ARRAY\_INTEGER; --定义变量

BEGIN

ARRINT('TOM') := 99; --通过变量下标给数组赋值

RETURN ARRINT['TOM']; --返回数组元素

END;

$$

LANGUAGE plpgsql;

返回结果为:

CREATE FUNCTION

## 嵌套表

**功能描述**

在数据库中嵌套表是一种列类型，用于存储未指定数量的行。未初始化的嵌套表变量是一个空集合，使用前必须对其进行初始化，方法是将其设置为空或分配一个非NULL的值。Vastbase G100 build 9支持在PL/pgSQL中定义嵌套表类型，以及定义和使用嵌套表变量，并支持嵌套表元素的赋值以及访问。

**语法格式**

---嵌套表的声明

TYPE typename IS TABLE OF datatype;

**参数说明**

* typename：类型名称。
* datatype：数组元素支持Vastbase数据类型包括数值、字符、时间以及自定义类型，不支持REF CURSOR和NCLOB类型。

**注意事项**

* 赋值与访问嵌套表变量，访问的下标类型是整型数字下标。
* 嵌套表不支持初始化赋值，直接对数组赋值而不初始化数组。
* 嵌套表的数组会自动增长，访问越界会返回一个NULL，不会报错。
* 嵌套表在存储过程中定义的数组类型，其作用域仅在该存储过程中。

**示例**

定义嵌套表。

CREATE OR REPLACE FUNCTION Fun() RETURNS int AS $$

DECLARE

TYPE ARRAY\_INTEGER IS TABLE OF int; --定义类型

ARRINT ARRAY\_INTEGER; --定义变量

BEGIN

ARRINT[1] := 1; --通过常量下标给数组赋值

FOR I IN 2..16 LOOP

select I into ARRINT[I] from dual; --通过变量下标给数组赋值

END LOOP;

RETURN ARRINT[2];

END;

$$

LANGUAGE plpgsql;

返回结果为：

CREATE FUNCTION

## 函数与存储过程传入参数与查询字段同名

**功能描述**

允许存储过程和函数传入参数和逻辑代码中查询的字段重名。

**语法格式**

* 函数创建

create or replace function 函数名（参数1 模式 数据类型，......）return 数据类型

* 存储过程创建

create or replace function 过程名（参数1 模式 数据类型，......）

**注意事项**

* 在查询的select部分，如果遇到表字段和传入的参数同名的情况，仍然按照表字段进行查询。
* 在查询的select部分，如果遇到表字段不和传入的参数同名，则查询出传入参数值。
* 该功能仅在数据库兼容模式为Oracle时能够使用（即创建DB时DBCOMPATIBILITY='A'），在其他数据库兼容模式下不能使用该特性。

**示例**

1、创建并切换至兼容Oracle的库db\_oracle。

CREATE DATABASE db\_oracle dbcompatibility='A';

\c db\_oracle

2、清除测试用表。

drop table if exists t1;

drop table if exists t2;

3、开启dbms\_output功能。

set serveroutput on;

4、新建测试用表并插入数据（表t1与函数有重名字段id，表t2与函数没有重名字段）。

create table t1(id number, name varchar2(200));

Insert into t1 values(1,'a1');

create table t2(val number, name varchar2(200));

Insert into t2 values(1,'a1');

5、创建函数。

create or replace function func(id number) return number

as

v\_id1 number;

v\_id2 number;

v\_col1 varchar2(100);

v\_col2 varchar2(100);

begin

select id,name into v\_id1, v\_col1 from t1 where id = 1;

select id,name into v\_id2, v\_col2 from t2 where id = 2;

dbms\_output.put\_line('id: ' || id);

dbms\_output.put\_line('v\_id1: ' || v\_id1);

dbms\_output.put\_line('v\_col1: ' || v\_col1);

dbms\_output.put\_line('v\_id2: ' || v\_id2);

dbms\_output.put\_line('v\_col2: ' || v\_col2);

return 0;

end;

/

6、调用函数输入参数为1。

select func(1) from dual;

返回结果为：

id: 1

v\_id1: 1

v\_col1: a1

v\_id2:

v\_col2:

func

------

0

(1 row)

7、调用函数输入参数为2。

select func(2) from dual;

返回结果为：

id: 2

v\_id1: 1

v\_col1: a1

v\_id2: 2

v\_col2: a1

func

------

0

(1 row)

## 匿名块支持JDBC绑定变量输入输出

**功能描述**

该功能实现了JDBC驱动程序在使用匿名代码块的时候可以绑定变量作为输入输出参数。

**注意事项**

* 该功能仅在数据库兼容模式为Oracle时能够使用（即创建DB时DBCOMPATIBILITY='A'），在其他数据库兼容模式下不能使用该特性。
* 只有使用配套的JDBC才可以实现该功能。
* 输入类型支持数值/字符串/日期等数据库常用数据类型。
* 内核输出类型只支持text，JDBC端支持ROWID类型输出。
* 不支持自治事务匿名块绑定变量输入输出。

**使用流程**

1、创建并切换至兼容模式为Oracle的数据库db\_oracle。

CREATE DATABASE db\_oracle dbcompatibility='A';

\c db\_oracle

2、在数据库中建表。

create table t\_rowid(a varchar,b int,c float8,d timestamp,e varchar,f varchar);

3、JAVA端正常连接数据库后进行如下操作。

（1）匿名块+绑定变量SQL语句。

String sql=

"begin\r\n"+

" insert into t\_rowid values(?,?,?,?,?,?) returning rowid into ?;\r\n"+

"insert into t\_rowid values(?) returning rowid into ?;\r\n"+

"select ? into ?;\r\n""+

"end";

(2)输入输出参数声明并执行。

try(CallableStatement pstmt = connection.prepareCall(sql)){

pstmt.setString(1, "hello");

pstmt.setInt(2, 666);

pstmt.setDouble(3, 66.66)

pstmt.setTimestamp(4, new java.sql.Timestamp(new Date() getTime())); pstmt.setString(5, null);

pstmt.setString(6, "0");

pstmt.registerOutParameter(7,Types.VARCHAR);

pstmt.setString(8, "你好");

pstmt.registerOutParameter(9,Types.VARCHAR);

pstmt.setString(10, "hello world");

pstmt.registerOutParameter(11,Types.VARCHAR);

pstmt.execute();

String value =pstrmt.getString(7);

System.out.println("value:"+value);

String val = pstmt.getString(9);

System.out.println("value:"+val);

String valu = pstmt.getString(11);

System.out.println("value:"+valu)

}

3、在数据库中执行结果查询。

select \* fron t\_rowid;

返回结果为：

a | b | c | d | e | f

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

hello | 666 | 66.66 | 2022-01-06 18:23:31:34| | 0

你好

(2 rows)

4、JAVA端接受的结果。

value:0EQAAA==AAAAAA==BAA=

value:0EQAAA==AAAAAA==BQA=

value:hello world

## 数组支持RECORD类型

**功能描述**

该功能允许在PL/SQL程序中定义RECORD类型数组。

**语法格式**

TYPE array\_type IS VARRAY(size) OF data\_type;

**参数说明**

* array\_type：要定义的数组类型名。
* VARRAY：表示要定义的数组类型。
* size：取值为正整数，表示可以容纳的成员的最大数量。
* data\_type：要创建的数组中成员的类型，现在支持record类型。

**注意事项**

无。

**示例**

在程序块中定义数组类型。

declare

TYPE sum\_type IS RECORD

(

sum\_id varchar2(20),

sum\_sum number

);

sum\_rec sum\_type;

TYPE sum\_type\_array IS VARRAY(2) of sum\_type;

sum\_rec\_array sum\_type\_array;

begin

sum\_rec.sum\_id :='GLG001\_CBB\_DAY';

sum\_rec.sum\_sum := 1;

sum\_rec\_array[1] := sum\_rec;

sum\_rec.sum\_id :='GLG001\_CBC\_SUM';

sum\_rec.sum\_sum := 2;

sum\_rec\_array[2] := sum\_rec;

sum\_rec := sum\_rec\_array[1];

raise notice 'sum\_rec\_array[1]：% %', sum\_rec.sum\_id, sum\_rec.sum\_sum;

sum\_rec := sum\_rec\_array[2];

raise notice 'sum\_rec\_array[2]：% %', sum\_rec.sum\_id, sum\_rec.sum\_sum;

end;

/

返回结果为：

NOTICE: sum\_rec\_array[1]：GLG001\_CBB\_DAY 1

NOTICE: sum\_rec\_array[2]：GLG001\_CBC\_SUM 2

ANONYMOUS BLOCK EXECUTE

## 自治事务

**功能描述**

该功能支持在定义pl/pgsql定义函数/存储过程中，在声明部分增加指令：PRAGMA AUTONOMOUS\_TRANSACTION，用来控制对当前存储过程中的内部事务可以进行独立提交，而不影响其他的事务。

该执行方式与普通方式执行函数/存储过程相同。

此外，自治事务还支持如下功能：

* 支持pl/pgsql块内的sql语句使用函数/存储过程参数和定义的变量。
* 允许在触发器函数内，匿名块内声明自治事务。
* 支持自治事务的嵌套使用。

**语法格式**

PRAGMA AUTONOMOUS\_TRANSACTION;

**注意事项**

声明为自治事务的函数/存储过程对返回值以及参数有如下限制：

* 不支持集合作为返回值。
* 不支持组合类型作为返回类型。
* 不支持out参数。

**示例**

1、创建测试表。

create table at\_tb2(id int, val varchar(64));

2、创建声明为自治事务的存储过程。

create or replace procedure at\_test3(i int)

AS

DECLARE

PRAGMA AUTONOMOUS\_TRANSACTION;

BEGIN

insert into at\_tb2 values(1, 'before s1');

insert into at\_tb2 values(2, 'after s1');

if i > 10 then

rollback;

else

commit;

end if;

end;

/

commit; -- 如果数据库未开启默认自动提交，则需要手动提交

3、调用存储过程。

call at\_test3(6);

4、查询表中的数据。

select \* from at\_tb2;

返回结果为：

id | val

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

1 | before s1

2 | after s1

(2 rows)

5、支持pl/pgsql块内的sql语句使用函数/存储过程参数和定义的变量。

create or replace procedure at\_proc(i int)

as

declare

strval varchar;

pragma autonomous\_transaction;

begin

strval:= 'begin:insert by autonomous transaction procedure<at\_proc>';

insert into at\_tb2 values(i,strval);

commit;

end;

/

commit; -- 如果数据库未开启默认自动提交，则需要手动提交

6、调用存储过程。

call at\_proc(1);

7、查询表中的数据。

select \* from at\_tb2;

返回结果为：

id | val

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

1 | before s1

2 | after s1

1 | begin:insert by autonomous transaction procedure<at\_proc>

(3 rows)

8、支持在触发器函数内声明自治事务-创建源表及触发表。

create table test\_trigger\_src\_tbl(id1 int, id2 int, id3 int);

create table test\_trigger\_des\_tbl(id1 int, id2 int, id3 int);

9、创建触发器函数。

create or replace function tri\_insert\_func() returns trigger as

$$

declare

pragma autonomous\_transaction;

begin

insert into test\_trigger\_des\_tbl values(new.id1, new.id2, new.id3);

return new;

end

$$ language plpgsql;

commit; -- 如果数据库未开启默认自动提交，则需要手动提交

10、创建insert触发器。

create trigger insert\_trigger

before insert on test\_trigger\_src\_tbl

for each row

execute procedure tri\_insert\_func();

11、执行insert触发事件并检查触发结果。

insert into test\_trigger\_src\_tbl values(100,200,300);

select \* from test\_trigger\_src\_tbl;

返回结果为：

id1 | id2 | id3

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

100 | 200 | 300

(1 row)

12、查看触发操作是否生效。

select \* from test\_trigger\_des\_tbl;

返回结果为：

id1 | id2 | id3

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

100 | 200 | 300

(1 row)

13、清除表中的数据。

truncate at\_tb2;

14、支持在匿名块内声明自治事务。

declare

pragma autonomous\_transaction;

strval varchar;

begin strval:='insert by autonomous transaction by anonymous block';

insert into at\_tb2 values(1,strval);

commit;

end;

/

15、查询表中数据。

select \* from at\_tb2;

返回结果为：

id | val

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

1 | insert by autonomous transaction by anonymous block

(1 row)

16、支持自治事务嵌套使用-创建嵌套自治事务的存储过程。

create or replace procedure at\_proc\_inner(i int)

as

declare

pragma autonomous\_transaction;

begin

insert into at\_tb2 values(1,'insert by autonomous transaction procedure<at\_proc\_inner>');

if i>5 then

rollback;

else

commit;

end if;

end;

/

create or replace procedure at\_proc\_outer(i int)

as

declare

pragma autonomous\_transaction;

begin

insert into at\_tb2 values(1,'insert by autonomous transaction procedure<at\_proc\_outer>');

at\_proc\_inner(i);

if i>10 then rollback;

else

commit;

end if;

end;

/

17、调用存储过程并查询结果

truncate at\_tb2;

call at\_proc\_outer(8);

select \* from at\_tb2;

返回结果为：

id | val

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

1 | insert by autonomous transaction procedure<at\_proc\_outer>

(1 row)

## 支持oracle的包（Package）和包体（Package body）

**功能描述**

Package是一个模式对象，用于组织过程、函数和变量的一种方式，由包声明和包体组成。

**语法格式**

* 包声明

CREATE [OR REPLACE] PACKAGE [schema\_name.]<package\_name> IS | AS

declarations;

END;

* 包体

CREATE [OR REPLACE] PACKAGE BODY [schema\_name.]<package\_name> IS | AS

declarations

implementations;

[BEGIN

EXCEPTION]

END;

**参数说明**

* 声明包
* OR REPLACE：该参数说明如果创建的包存在,则重新创建包。
* schema\_name：包含此包的模式名称。
* package\_name：包的名称。
* declarations：可以声明公有变量和公有子程序。
* 包体
* declarations：声明私有变量和私有子程序。
* implementations：定义私有子程序和公有子程序。

**注意事项**

无。

**示例**

* 创建包

CREATE OR REPLACE PACKAGE emp\_bonus AS

PROCEDURE calc\_bonus (d date);

END;

/

* 创建包体

CREATE OR REPLACE PACKAGE BODY emp\_bonus AS

PROCEDURE calc\_bonus (date\_hired DATE) IS

BEGIN

DBMS\_OUTPUT.PUT\_LINE

('Employees hired on ' || date\_hired || ' get bonus.');

END;

END;

/

* 删除包体

DROP PACKAGE BODY emp\_bonus;

## 支持PLSQL-BULK COLLECT INTO

**功能描述**

该子句用于指定结果集到集合的这种聚合。

BULK COLLECT 子句可与 SELECT INTO、FETCH INTO 和 EXECUTE IMMEDIATE 命令一起使用，也可与 DELETE、INSERT 和 UPDATE 命令的 RETURNING INTO 子句一起使用。

**注意事项**

该功能仅在数据库兼容模式为Oracle时能够使用（即创建DB时DBCOMPATIBILITY='A'），在其他数据库兼容模式下不能使用该特性。

**SELECT BULK COLLECT**

**语法格式**

SELECT select\_expressions BULK COLLECT INTO collection [, ...] FROM ...;

注意：

* 如果指定了单个集合，则 collection 可以是单个字段的集合，也可以是某一记录类型的集合。如果指定了多个集合，则每个 collection 必须由单个字段组成。
* select\_expressions 必须在数字、顺序和类型兼容性方面与目标集合中的所有字段相匹配。

**示例**

1、创建并切换至兼容模式为Oracle的数据库db\_oracle。

CREATE DATABASE db\_oracle dbcompatibility='A';

\c db\_oracle

2、创建测试表。

create table EMP

(

EMPNO NUMBER(4) PRIMARY KEY,

ENAME VARCHAR2(10),

JOB VARCHAR2(9),

MGR NUMBER(4),

HIREDATE DATE,

SAL NUMBER(7,2),

COMM NUMBER(7,2),

DEPTNO NUMBER(4)

);

3、插入数据。

INSERT INTO EMP VALUES

(7369,'SMITH','CLERK',7902,to\_date('17-12-1980','dd-mm-yyyy'),800,null,20);

INSERT INTO EMP VALUES

(7499,'ALLEN','SALESMAN',7698,to\_date('20-2-1981','dd-mm-yyyy'),1600,300,30);

INSERT INTO EMP VALUES

(7521,'WARD','SALESMAN',7698,to\_date('22-2-1981','dd-mm-yyyy'),1250,500,30);

INSERT INTO EMP VALUES

(7566,'JONES','MANAGER',7839,to\_date('2-4-1981','dd-mm-yyyy'),2975,NULL,20);

INSERT INTO EMP VALUES

(7654,'MARTIN','SALESMAN',7698,to\_date('28-9-1981','dd-mm-yyyy'),1250,1400,30);

INSERT INTO EMP VALUES

(7698,'BLAKE','MANAGER',7839,to\_date('1-5-1981','dd-mm-yyyy'),2850,NULL,30);

INSERT INTO EMP VALUES

(7782,'CLARK','MANAGER',7839,to\_date('9-6-1981','dd-mm-yyyy'),2450,NULL,10);

INSERT INTO EMP VALUES

(7839,'KING','PRESIDENT',NULL,to\_date('17-11-1981','dd-mm-yyyy'),5000,NULL,10);

INSERT INTO EMP VALUES

(7844,'TURNER','SALESMAN',7698,to\_date('8-9-1981','dd-mm-yyyy'),1500,0,30);

INSERT INTO EMP VALUES

(7900,'JAMES','CLERK',7698,to\_date('3-12-1981','dd-mm-yyyy'),950,NULL,30);

INSERT INTO EMP VALUES

(7902,'FORD','ANALYST',7566,to\_date('3-12-1981','dd-mm-yyyy'),3000,NULL,20);

INSERT INTO EMP VALUES

(7934,'MILLER','CLERK',7782,to\_date('23-1-1982','dd-mm-yyyy'),1300,NULL,10);

4、bulk collect 子句与select into使用，多个单一字段，多个集合

set serveroutput on;

DECLARE

TYPE empno\_tbl IS TABLE OF emp.empno%TYPE INDEX BY BINARY\_INTEGER;

TYPE ename\_tbl IS TABLE OF emp.ename%TYPE INDEX BY BINARY\_INTEGER;

TYPE job\_tbl IS TABLE OF emp.job%TYPE INDEX BY BINARY\_INTEGER;

TYPE hiredate\_tbl IS TABLE OF emp.hiredate%TYPE INDEX BY BINARY\_INTEGER;

TYPE sal\_tbl IS TABLE OF emp.sal%TYPE INDEX BY BINARY\_INTEGER;

TYPE comm\_tbl IS TABLE OF emp.comm%TYPE INDEX BY BINARY\_INTEGER;

TYPE deptno\_tbl IS TABLE OF emp.deptno%TYPE INDEX BY BINARY\_INTEGER;

t\_empno EMPNO\_TBL;

t\_ename ENAME\_TBL;

t\_job JOB\_TBL;

t\_hiredate HIREDATE\_TBL;

t\_sal SAL\_TBL;

t\_comm COMM\_TBL;

t\_deptno DEPTNO\_TBL;

BEGIN

SELECT empno, ename, job, hiredate, sal, comm, deptno BULK COLLECT

INTO t\_empno, t\_ename, t\_job, t\_hiredate, t\_sal, t\_comm, t\_deptno

FROM emp;

DBMS\_OUTPUT.PUT\_LINE('EMPNO ENAME JOB HIREDATE ' ||

'SAL ' || 'COMM DEPTNO');

DBMS\_OUTPUT.PUT\_LINE('----- ------- --------- --------- ' ||

'-------- ' || '-------- ------');

FOR i IN 1..t\_empno.COUNT LOOP

DBMS\_OUTPUT.PUT\_LINE(t\_empno(i) || ' ' ||

RPAD(t\_ename(i),8) || ' ' ||

RPAD(t\_job(i),10) || ' ' ||

TO\_CHAR(t\_hiredate(i),'DD-MON-YY') || ' ' ||

TO\_CHAR(t\_sal(i),'99,999.99') || ' ' ||

TO\_CHAR(NVL(t\_comm(i),0),'99,999.99') || ' ' ||

t\_deptno(i));

END LOOP;

END;

/

返回结果为：

EMPNO ENAME JOB HIREDATE SAL COMM DEPTNO

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

7369 SMITH CLERK 17-DEC-80 800 20

7499 ALLEN SALESMAN 20-FEB-81 1,600 300 30

7521 WARD SALESMAN 22-FEB-81 1,250 500 30

7566 JONES MANAGER 02-APR-81 2,975 20

7654 MARTIN SALESMAN 28-SEP-81 1,250 1,400 30

7698 BLAKE MANAGER 01-MAY-81 2,850 30

7782 CLARK MANAGER 09-JUN-81 2,450 10

7839 KING PRESIDENT 17-NOV-81 5,000 10

7844 TURNER SALESMAN 08-SEP-81 1,500 30

7900 JAMES CLERK 03-DEC-81 950 30

7902 FORD ANALYST 03-DEC-81 3,000 20

7934 MILLER CLERK 23-JAN-82 1,300 10

ANONYMOUS BLOCK EXECUTE

**FETCH BULK COLLECT**

**语法格式**

FETCH name BULK COLLECT INTO collection [, ...] [ LIMIT n ];

注意：

* FETCH BULK COLLECT 不是一次从结果集返回一行，而是将结果集中的所有行一次性返回到指定的集合中，除非由 LIMIT 子句进行了限制。
* 如果指定了单个集合，则 collection 可以是单个字段的集合，也可以是某一记录类型的集合。如果指定了多个集合，则每个 collection 必须由单个字段组成。
* 游标（由 name 标识）的 SELECT 列表中的表达式必须在数字、顺序和类型兼容性方面与目标集合中的所有字段相匹配。
* 如果指定了 LIMIT n，则每个 FETCH 返回到集合中的行数将不会超过 n。

**示例**

前三个步骤与SELECT BULK COLLECT示例相同，同样采用上面章节创建的测试表EMP。

bulk collect 子句与fetch into使用，加limit。

DECLARE

TYPE emp\_tbl IS TABLE OF emp%ROWTYPE INDEX BY BINARY\_INTEGER;

t\_emp EMP\_TBL;

CURSOR emp\_cur IS SELECT \* FROM emp;

BEGIN

OPEN emp\_cur;

FETCH emp\_cur BULK COLLECT INTO t\_emp limit 5;

CLOSE emp\_cur;

DBMS\_OUTPUT.PUT\_LINE('EMPNO ENAME JOB HIREDATE ' ||

'SAL ' || 'COMM DEPTNO');

DBMS\_OUTPUT.PUT\_LINE('----- ------- --------- --------- ' ||

'-------- ' || '-------- ------');

FOR i IN 1..t\_emp.COUNT LOOP

DBMS\_OUTPUT.PUT\_LINE(t\_emp(i).empno || ' ' ||

RPAD(t\_emp(i).ename,8) || ' ' ||

RPAD(t\_emp(i).job,10) || ' ' ||

TO\_CHAR(t\_emp(i).hiredate,'DD-MON-YY') || ' ' ||

TO\_CHAR(t\_emp(i).sal,'99,999.99') || ' ' ||

TO\_CHAR(NVL(t\_emp(i).comm,0),'99,999.99') || ' ' ||

t\_emp(i).deptno);

END LOOP;

END;

/

返回结果为：

EMPNO ENAME JOB HIREDATE SAL COMM DEPTNO

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

7369 SMITH CLERK 17-DEC-80 800 20

7499 ALLEN SALESMAN 20-FEB-81 1,600 300 30

7521 WARD SALESMAN 22-FEB-81 1,250 500 30

7566 JONES MANAGER 02-APR-81 2,975 20

7654 MARTIN SALESMAN 28-SEP-81 1,250 1,400 30

ANONYMOUS BLOCK EXECUTE

**EXECUTE IMMEDIATE BULK COLLECT**

**语法格式**

EXECUTE IMMEDIATE 'sql\_expression;'

BULK COLLECT INTO collection [,...]

[USING {[bind\_type] bind\_argument} [, ...]}];

**参数说明**

* collection 指定集合的名称。
* bind\_type 指定 bind\_argument 的参数模式:
* bind\_type 为 IN 时指定 bind\_argument 包含传递给 sql\_expression 的值。
* bind\_type 为 OUT 时指定 bind\_argument 从 sql\_expression 接收值。
* bind\_type 为 IN OUT 时指定将 bind\_argument 传递给 sql\_expression，然后存储 sql\_expression 返回的值。

注意：

* bind\_argument 指定一个参数，该参数包含一个值，该值或传递给 sql\_expression（bind\_type 为 IN 时指定），或从 sql\_expression 接收值（bind\_type 为 OUT 时指定），或者执行这两项操作（bind\_type 为 INOUT 时指定）。
* 如果指定了单个集合，则 collection 可以是单个字段的集合，也可以是某一记录类型的集合；如果指定了多个集合，则每个 collection 必须由单个字段组成。

**示例**

前三个步骤与SELECT BULK COLLECT示例相同，同样采用上面章节创建的测试表EMP。

bulk collect 子句与execute immediate使用。

DECLARE

TYPE emp\_tbl IS TABLE OF emp%ROWTYPE INDEX BY BINARY\_INTEGER;

t\_emp EMP\_TBL;

BEGIN

EXECUTE IMMEDIATE 'SELECT \* FROM emp where deptno=:1' BULK COLLECT INTO t\_emp USING IN 20;

DBMS\_OUTPUT.PUT\_LINE('EMPNO ENAME JOB HIREDATE ' ||

'SAL ' || 'COMM DEPTNO');

DBMS\_OUTPUT.PUT\_LINE('----- ------- --------- --------- ' ||

'-------- ' || '-------- ------');

FOR i IN 1..t\_emp.COUNT LOOP

DBMS\_OUTPUT.PUT\_LINE(t\_emp(i).empno || ' ' ||

RPAD(t\_emp(i).ename,8) || ' ' ||

RPAD(t\_emp(i).job,10) || ' ' ||

TO\_CHAR(t\_emp(i).hiredate,'DD-MON-YY') || ' ' ||

TO\_CHAR(t\_emp(i).sal,'99,999.99') || ' ' ||

TO\_CHAR(NVL(t\_emp(i).comm,0),'99,999.99') || ' ' ||

t\_emp(i).deptno);

END LOOP;

END;

/

返回结果为：

EMPNO ENAME JOB HIREDATE SAL COMM DEPTNO

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

7369 SMITH CLERK 17-DEC-80 800 20

7566 JONES MANAGER 02-APR-81 2,975 20

7902 FORD ANALYST 03-DEC-81 3,000 20

ANONYMOUS BLOCK EXECUTE

**RETURNING BULK COLLECT**

**语法格式**

{ insert | update | delete }

RETURNING { \* | expr\_1 [, expr\_2 ] ...}

BULK COLLECT INTO collection [, ...];

insert、update 和 delete 分别是 INSERT、UPDATE 和 DELETE 命令。

注意：

* 如果指定了单个集合，则 collection 可以是单个字段的集合，也可以是某一记录类型的集合。如果指定了多个集合，则每个 collection 必须由单个字段组成。
* RETURNING 关键字后面的表达式必须在数字、顺序和类型兼容性方面与目标集合中的所有字段相匹配。

**示例**

前置步骤与SELECT BULK COLLECT示例相同，同样采用上面章节创建的测试表EMP。

1、bulk collect 子句与update returning into使用。

DECLARE

TYPE empno\_tbl IS TABLE OF emp.empno%TYPE INDEX BY BINARY\_INTEGER;

TYPE ename\_tbl IS TABLE OF emp.ename%TYPE INDEX BY BINARY\_INTEGER;

TYPE sal\_tbl IS TABLE OF emp.sal%TYPE INDEX BY BINARY\_INTEGER;

t\_empno EMPNO\_TBL;

t\_ename ENAME\_TBL;

t\_sal SAL\_TBL;

BEGIN

UPDATE emp SET sal = sal \* 1.5 RETURNING empno, ename, sal

BULK COLLECT INTO t\_empno, t\_ename, t\_sal;

DBMS\_OUTPUT.PUT\_LINE('EMPNO ENAME SAL ');

DBMS\_OUTPUT.PUT\_LINE('----- ------- -------- ');

FOR i IN 1..t\_empno.COUNT LOOP

DBMS\_OUTPUT.PUT\_LINE(t\_empno(i) || ' ' || RPAD(t\_ename(i),8) ||

' ' || TO\_CHAR(t\_sal(i),'99,999.99'));

END LOOP;

END;

/

返回结果如下：

EMPNO ENAME SAL

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

7369 SMITH 1,200

7499 ALLEN 2,400

7521 WARD 1,875

7566 JONES 4,462.5

7654 MARTIN 1,875

7698 BLAKE 4,275

7782 CLARK 3,675

7839 KING 7,500

7844 TURNER 2,250

7900 JAMES 1,425

7902 FORD 4,500

7934 MILLER 1,950

ANONYMOUS BLOCK EXECUTE

2、bulk collect 子句与insert into returning into使用。

DECLARE

TYPE empno\_tbl IS TABLE OF emp.empno%TYPE INDEX BY BINARY\_INTEGER;

TYPE ename\_tbl IS TABLE OF emp.ename%TYPE INDEX BY BINARY\_INTEGER;

TYPE sal\_tbl IS TABLE OF emp.sal%TYPE INDEX BY BINARY\_INTEGER;

t\_empno EMPNO\_TBL;

t\_ename ENAME\_TBL;

t\_sal SAL\_TBL;

BEGIN

INSERT INTO EMP VALUES (5623,'SMITH','CLERK',7902,to\_date('17-12-1980','dd-mm-yyyy'),800,null,20) RETURNING empno, ename, sal

BULK COLLECT INTO t\_empno, t\_ename, t\_sal;

DBMS\_OUTPUT.PUT\_LINE('EMPNO ENAME SAL ');

DBMS\_OUTPUT.PUT\_LINE('----- ------- -------- ');

FOR i IN 1..t\_empno.COUNT LOOP

DBMS\_OUTPUT.PUT\_LINE(t\_empno(i) || ' ' || RPAD(t\_ename(i),8) ||

' ' || TO\_CHAR(t\_sal(i),'99,999.99'));

END LOOP;

END;

/

返回结果为：

EMPNO ENAME SAL

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

5623 SMITH 800

## package中支持函数返回嵌套表和关联数组类型

**功能描述**

包中支持函数返回嵌套表和关联数组类型。

**语法格式**

create or replace package package\_test as

type table\_test\_type is table of type1 [index by type2];

**参数说明**

* package\_test：用户定义的包名。
* table\_test\_type：用户定义的返回类型设置名。
* type 函数返回类型：可指定具体数据类型如int、number等。
* index by type2 表示函数返回关联数组，其中type2是数组类型，如binary\_integer等。

**注意事项**

该功能仅在数据库兼容模式为Oracle时能够使用（即创建DB时DBCOMPATIBILITY='A'），在其他数据库兼容模式下不能使用该特性。

**示例**

**示例1：**包内存储过程返回int类型嵌套表。

1、创建测试数据库并检查数据库为Oracle兼容模式。

CREATE DATABASE my\_test DBCOMPATIBILITY 'A';

\c my\_test

show sql\_compatibility;

检查兼容模式结果为：

sql\_compatibility

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

A

(1 row)

2、准备测试数据。

create or replace package package\_test as

type table\_test\_type is table of int;

function func\_Pack\_para(tmp1\_in int,tmp2\_in int) return table\_test\_type;

end package\_test;

3、package内函数调用。

create or replace package body package\_test is

function func\_Pack\_para(tmp1\_in int, tmp2\_in int) return table\_test\_type IS

re\_l table\_test\_type;

i int;

begin

re\_l(1) := tmp1\_in;

re\_l(2) := tmp2\_in;

i := re\_l.FIRST;

WHILE i IS NOT NULL LOOP

raise info '%; index:%',re\_l(i),i;

i := re\_l.NEXT(i);

END LOOP;

return re\_l;

end;

end package\_test;

/

call package\_test.func\_Pack\_para(1, 159.7);

函数返回int类型嵌套表成功：

INFO: 1; index:1

INFO: 160; index:2

func\_pack\_para

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

{1,160}

(1 row)

**示例2：**包内函数返回number(10,2)类型嵌套表，关联数组binary\_integer。

1、创建测试数据库并检查数据库为Oracle兼容模式。

CREATE DATABASE my\_test DBCOMPATIBILITY 'A';

\c my\_test

show sql\_compatibility;

检查兼容模式结果为：

sql\_compatibility

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

A

(1 row)

2、准备测试数据。

create or replace package package\_test as

TYPE table\_test\_type is table of number(10,2) index by binary\_integer;

function func\_Pack\_para(tmp1\_in number,tmp2\_in number) return table\_test\_type;

end package\_test;

/

3、package内函数调用。

create or replace package body package\_test is

function func\_Pack\_para(tmp1\_in number, tmp2\_in number) return table\_test\_type IS

re\_l table\_test\_type;

i int;

begin

re\_l(1) := tmp1\_in;

re\_l(2) := tmp2\_in;

i := re\_l.FIRST;

WHILE i IS NOT NULL LOOP

raise info '%; index:%',re\_l(i),i;

i := re\_l.NEXT(i);

END LOOP;

return re\_l;

end;

end package\_test;

/

call package\_test.func\_Pack\_para(1.123, 12345678.159);

嵌套表关联数组成功：

INFO: 1.12; index:1

INFO: 12345678.16; index:2

func\_pack\_para

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

{1.12,12345678.16}

(1 row)

## package中支持%type语法

**功能描述**

包内支持使用%type定义变量的类型，赋值时包体可自动转换。

**语法格式**

create or replace package package\_name as

variable\_name table\_name.column\_name%type;

**参数说明**

* package\_name：用户定义的包名称。
* variable\_name：用户定义的变量名称。
* table\_name.column\_name ：%type参考的类型对应的列名（当调用其他表或包时，用xx.xx指定，如语法中的table\_name.column\_name)。

**注意事项**

* 该功能仅在数据库兼容模式为Oracle时能够使用（即创建DB时DBCOMPATIBILITY='A'），在其他数据库兼容模式下不能使用该特性。
* %type支持vastbase支持的类型。

**示例**

1、创建数据库。

CREATE DATABASE db\_1087209 DBCOMPATIBILITY 'A';

\c db\_1087209

2、创建包，%type定义变量var2的类型为字符型。

create or replace package pk\_1087209 as

var1 varchar(5);

var2 var1%type;

end;

/

3、创建包体，赋值类型为数值型。

create or replace package body pk\_1087209 as

procedure p1 as

begin

var1:='aaa';

var2:=55;

raise info '%', var1;

raise info '%', var2;

end;

end pk\_1087209;

/

4、调用包体。

call pk\_1087209.p1();

返回结果为：

INFO: aaa

INFO: 55

p1

----

(1 row)

## package中支持同义词引用对象语法

**功能描述**

Vastbase G100在Oracle兼容模式下创建package时支持同义词引用对象语法定义。

**语法格式**

CREATE OR REPLACE PACKAGE package\_name AS

type\_name data\_type;

END;

/

CREATE SYNONYM synonym\_name for package\_name;

CREATE OR REPLACE PACKAGE package\_b AS

variable\_name synonym\_name.subtype\_name%type;

END;

/

**参数说明**

* data\_type：Vastbase所有的数据类型。
* %type：包内支持使用%type定义变量的类型，赋值时包体可自动转换。具体使用方法请参考[%type语法](#type语法)。

**注意事项**

该功能仅在数据库兼容模式为Oracle时能够使用（即创建DB时DBCOMPATIBILITY='A'），在其他数据库兼容模式下不能使用该特性。

**示例**

1、创建数据库，确认兼容性。

CREATE DATABASE db\_1087651 DBCOMPATIBILITY 'A';

\c db\_1087651

show sql\_compatibility;

2、创建普通用户并授权。

CREATE USER user1\_1087651 PASSWORD 'Abc@123456';

grant all on database db\_1087651 to user1\_1087651;

CREATE USER user2\_1087651 PASSWORD 'Abc@123456';

grant all on database db\_1087651 to user2\_1087651;

3、创建类型。

create type user1\_1087651.ty1\_1087651 is(

id int2,

num1 int4,

num2 number(5),

num3 number(3,3),

num4 number(4,1),

num5 numeric,

num6 float4,

num7 float8,

num8 real);

4、创建包1。

create or replace package user1\_1087651.pk1\_1087651 as

var1 ty1\_1087651;

var2 int2;

var3 var1.num1%type;

type tp\_rec4 is record(id number(5),num2 var1.num3%type);

rec4 tp\_rec4;

var5 number(4,1);

type tp\_varray6 is varray(3) of var1.num5%type;

varray6 tp\_varray6;

type tp\_tb7 is table of var1.num7%type index by text;

tb7 tp\_tb7;

var8 float4;

var9 real;

end;

/

5、创建同义词。

create synonym user2\_1087651.syn\_pk2\_1087651 for user1\_1087651.pk1\_1087651;

6、创建包3。

create or replace package user2\_1087651.pk3\_1087651 as

var1 syn\_pk2\_1087651.var1%type;

var2 syn\_pk2\_1087651.var2%type;

var3 syn\_pk2\_1087651.var3%type;

rec4 syn\_pk2\_1087651.rec4%type;

var5 syn\_pk2\_1087651.var5%type;

varray6 syn\_pk2\_1087651.varray6%type;

tb7 syn\_pk2\_1087651.tb7%type;

var8 syn\_pk2\_1087651.var8%type;

var9 syn\_pk2\_1087651.var9%type;

end;

/

7、创建包体。

create or replace package body user2\_1087651.pk3\_1087651 as

procedure p1 as

begin

var1:=(12,16,12345.6,0.123,42.3,61.2,7.889,1.241,5);

var2:=12;

var3:=99;

rec4:=(144,0.9991);

var5:=11.2;

varray6(1):=0.637;

tb7(11):=0.247;

var8:=3.141;

var9:=44;

raise info '%',var1;

raise info '%',var2;

raise info '%',var3;

raise info '%',rec4;

raise info '%',var5;

raise info '%',varray6;

raise info '%',tb7;

raise info '%',var8;

raise info '%',var9;

end;

end pk3\_1087651;

/

8、调用。

call user2\_1087651.pk3\_1087651.p1();

返回结果为：

INFO: (12,16,12346,.123,42.3,61.2,7.889,1.241,5)

INFO: 12

INFO: 99

INFO: (144,.9991)

INFO: 11.2

INFO: {.637}

INFO: [11:11]={.247}

INFO: 3.141

INFO: 44

p1

----

(1 row)

## package中支持类型调用

**功能描述**

Vastbase G100在Oracle兼容模式下创建package中支持类型调用。

**语法格式**

CREATE OR REPLACE PACKAGE package\_name AS

type\_name data\_type;

var1 int,

END;

/

CREATE OR REPLACE PACKAGE package\_b AS

variable\_name package\_name.type\_name%type;

END;

/

**参数说明**

* data\_type：Vastbase所有的数据类型。
* %type：包内支持使用%type定义变量的类型，赋值时包体可自动转换。具体使用方法请参考[%type语法](#type语法)。

**注意事项**

* 该功能仅在数据库兼容模式为Oracle时能够使用（即创建DB时DBCOMPATIBILITY='A'），在其他数据库兼容模式下不能使用该特性。
* 用户需要具备创建package的权限。

**示例**

1、创建数据库，检查兼容性。

CREATE DATABASE db\_1087293 DBCOMPATIBILITY 'A';

\c db\_1087293

show sql\_compatibility;

2、创建类型。

create type ty1\_1087293 is(

id int2,

num1 int4,

num2 number(5),

num3 number(3,3),

num4 number(4,1),

num5 numeric,

num6 float4,

num7 float8,

num8 real);

3、创建包1。

create or replace package pk1\_1087293 as

var1 ty1\_1087293;

var2 int2;

var3 var1.num1%type;

type tp\_rec4 is record(id number(5),num2 var1.num3%type);

rec4 tp\_rec4;

var5 number(4,1);

type tp\_varray6 is varray(3) of var1.num5%type;

varray6 tp\_varray6;

type tp\_tb7 is table of var1.num7%type index by text;

tb7 tp\_tb7;

var8 float4;

var9 real;

end;

/

4、创建包2。

create or replace package pk2\_1087293 as

var1 pk1\_1087293.var1%type;

var2 pk1\_1087293.var2%type;

var3 pk1\_1087293.var3%type;

rec4 pk1\_1087293.rec4%type;

var5 pk1\_1087293.var5%type;

varray6 pk1\_1087293.varray6%type;

tb7 pk1\_1087293.tb7%type;

var8 pk1\_1087293.var8%type;

var9 pk1\_1087293.var9%type;

end;

/

5、创建包体。

create or replace package body pk2\_1087293 as

procedure p1 as

begin

var1:=(12,16,12345.6,0.123,42.3,61.2,7.889,1.241,5);

var2:=12;

var3:=99;

rec4:=(144,0.9991);

var5:=11.2;

varray6(1):=0.637;

tb7(11):=0.247;

var8:=3.141;

var9:=44;

raise info '%',var1;

raise info '%',var2;

raise info '%',var3;

raise info '%',rec4;

raise info '%',var5;

raise info '%',varray6;

raise info '%',tb7;

raise info '%',var8;

raise info '%',var9;

end;

end pk2\_1087293;

/

6、调用。

call pk2\_1087293.p1();

返回结果为：

INFO: (abcde,aAw1,ppp,"oopp,,")

INFO: (12AB,lmn,opqe4--4,q,pp4477,hh)

INFO: qqq

INFO: kkk

INFO: 0OLq

INFO: (ammm!\*…1,1a11,-3.1134)

INFO: {aaa1}

INFO: aaa1

INFO: p

INFO: {mno,NULL,mmn}

INFO: aa

p1

----

(1 row)

## package中使用游标创建type

**功能描述**

该功能用于在包package或者匿名块、存储过程、函数中使用record类型创建一个游标的type类型。

**语法格式**

type type\_name is ref cursor return record\_name ;

**参数说明**

* type\_name：定义的类型的名称。
* record\_name：使用数据类型说明符 RECORD 定义的用户定义类型的名称。

**注意事项**

* 创建的type为弱游标类型。
* 该type类型如果作为out参数返回是游标类型。
* 该功能仅在数据库兼容模式为Oracle时能够使用（即创建DB时DBCOMPATIBILITY='A'），在其他数据库兼容模式下不能使用该特性。

**兼容性**

Vastbase G100 V2.2版本与Oracl相比，暂不支持如下语法：

TYPE type\_name IS REF CURSOR

[ RETURN

{ {db\_table\_or\_view | cursor | cursor\_variable}%ROWTYPE

| record%TYPE

| ref\_cursor\_type

}

] ;

**示例**

**示例1：**使用record类型创建一个游标的type类型。

1、创建测试表t。

create table t(id int, name varchar2(30));

2、声明游标类型。

declare

type RECORD\_ACCOUNT\_LIST is record(id int,name varchar2(30));

type CUR\_ACCOUNT\_LIST is ref cursor return RECORD\_ACCOUNT\_LIST;

cur\_accountlist CUR\_ACCOUNT\_LIST;

begin

open cur\_accountlist for select \* from t;

end;

/

返回结果如下，则表示创建成功：

ANONYMOUS BLOCK EXECUTE

**示例2：**弱游标类型。

1、创建测试表t1并插入数据。

create table t1(id int, name varchar2(30));

insert into t1 values(1,'a1');

2、定义一个弱类型游标。

declare

type RECORD\_ACCOUNT\_LIST1 IS RECORD (id int, name1 varchar2(30));

type CUR\_ACCOUNT\_LIST1 is ref cursor return RECORD\_ACCOUNT\_LIST1;

cur\_accountlist1 CUR\_ACCOUNT\_LIST1;

rc\_list RECORD\_ACCOUNT\_LIST1;

begin

open cur\_accountlist1 for select \* from t1;

loop

fetch cur\_accountlist1 into rc\_list;

exit when cur\_accountlist1%notfound;

dbms\_output.put\_line(rc\_list.id||' '||rc\_list.name1);

end loop;

end;

/

返回结果如下，则表示创建成功：

ANONYMOUS BLOCK EXECUTE

## package中支持type...is语法

**功能描述**

在Oracle兼容模式下，用户创建package时使用type...is来定义数据类型。

定义好type后，在本package或其他package函数中，可以只用这个type类型定义变量；在package的body里可以直接使用这个type来定义变量；在其他包里用packagename.typename的形式使用。

**语法格式**

create or replace package package\_name as

type type\_name is record...;//record类型

type type\_name is table of ...; //嵌套表

type type\_name is table of ...index by ...//关联数组

end;

/

**参数说明**

* package\_name： 包名。
* type\_name： 类型名。

**注意事项**

该功能仅在数据库兼容模式为Oracle时能够使用（即创建DB时DBCOMPATIBILITY='A'），在其他数据库兼容模式下不能使用该特性。

**示例**

**示例1：** 包规格内声明int类型嵌套表。

1、创建测试数据库并检查数据库兼容性。

\c vastbase

CREATE DATABASE my\_test DBCOMPATIBILITY 'A';

\c my\_test

show sql\_compatibility;

兼容模式为：

sql\_compatibility

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

A

(1 row)

2、包规格声明包内公有变量。

create or replace package package\_test as

type array\_integer is table of int;

end;

/

3、其他函数中调用int类型嵌套表。

create or replace function func\_Pack\_para() return int

as

func\_arr\_int package\_test.array\_integer;

i int;

begin

func\_arr\_int[1] := 1;

for i in 2..10 loop

select i into func\_arr\_int[i] from dual;

end loop;

return func\_arr\_int[5];

end;

/

4、调用函数。

call func\_Pack\_para();

函数返回结果：

func\_pack\_para

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

5

(1 row)

**示例2：** 包规格内声明int类型嵌套表关联varchar(20)类型数组。

1、创建测试数据库并检查数据库兼容性。

\c vastbase

CREATE DATABASE my\_test DBCOMPATIBILITY 'A';

\c my\_test

show sql\_compatibility;

兼容模式为：

sql\_compatibility

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

A

(1 row)

2、包中定义变量。

set serveroutput on;

create or replace package package\_test2 AUTHID DEFINER IS

Type array\_list\_type IS TABLE OF int INDEX BY varchar2(20);

end;

/

3、包外存储过程调用变量。

create or replace PROCEDURE proc\_Pack\_para AUTHID DEFINER IS

array\_List package\_test2.array\_list\_type;

i varchar2(20);

begin

array\_List('一') := 1;

array\_List('二') := 2;

array\_List('三') := 3;

i := array\_List.FIRST;

WHILE i IS NOT NULL LOOP

DBMS\_OUTPUT.PUT\_LINE (array\_List(i) || ' ' || i);

i := array\_List.NEXT(i);

END LOOP;

end;

/

4、调用存储过程。

call proc\_Pack\_para();

函数返回结果：

1 一

3 三

2 二

proc\_pack\_para

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

(1 row)

## 支持record类型作为procedure出参

**功能描述**

在Oracle兼容模式下，Vastbase G100支持在package中定义的record类型作为procedure出参。

**语法格式**

--创建一个package

create or replace package pkg\_themis\_xmloperate as

--定义t\_tagValue是以int作为数组标号、数组元素是varchar2(30000)的嵌套表类型

type t\_tagValue is table of varchar2(30000)index by binary\_integer;

--定义类型t\_tagclobvalue是以int作为数组标号、数组元素是clob的嵌套表类型

type t\_tagclobvalue is table of clob index binary\_integer;

--定义一个record类型A，该类型里有两种子类型：xmldomnode,boolean

type t\_approve\_info is record(xmldomnode int,is\_local boolean);

--定义一个存储过程，其中t\_approve\_info是一个在上面package里定义好的record类型，在这里被用于出参类型

procedure p\_get\_PD\_from\_approve\_info(

p\_stackid in number,

p\_stackdepth in number,

P\_in\_node in xmldom.domnode,

p\_out\_data out t\_approve\_info,

p\_case\_flow\_id in number default null,

p\_flow\_id in number default null);

end;

/

**注意事项**

* 该功能仅在数据库兼容模式为Oracle时能够使用（即创建DB时DBCOMPATIBILITY='A'），在其他数据库兼容模式下不能使用该特性。
* 由于测试用例关联其他兼容特性，所以需要确保关联的兼容同时具备才可以正常执行。
* 兼容支持在package中使用type...is...语法，比如is table of、is record。

**操作步骤**

1、自定义record类型和这个存储过程都是在一个package的声明里使用的。

2、定义package。

3、定义record类型。

4、定义存储过程，使用前面定义的record类型作为出参。

**示例**

1、创建数据库，检查兼容性。

CREATE DATABASE my\_test DBCOMPATIBILITY 'A';

\c my\_test

show sql\_compatibility;

2、创建一个包，定义record类型为int。

CREATE OR REPLACE PACKAGE pkg AS

TYPE rec\_type IS RECORD(

col\_int int,

col\_smallint smallint,

col\_integer integer,

col\_bigint bigint,

col\_int2 int2,

col\_int4 int4,

col\_int8 int8

);

PROCEDURE print\_rec\_type(

p\_col\_int in int,

p\_col\_smallint in smallint,

p\_col\_integer in integer,

p\_col\_bigint in bigint,

p\_col\_int2 in int2,

p\_col\_int4 in int4,

p\_col\_int8 in int8,

rec out rec\_type);

END;

/

3、包中存储过程作为返回参数。

CREATE OR REPLACE PACKAGE BODY pkg AS

PROCEDURE print\_rec\_type(

p\_col\_int in int,

p\_col\_smallint in smallint,

p\_col\_integer in integer,

p\_col\_bigint in bigint,

p\_col\_int2 in int2,

p\_col\_int4 in int4,

p\_col\_int8 in int8,

rec out rec\_type) IS

BEGIN

rec.col\_int := p\_col\_int ;

rec.col\_smallint:= p\_col\_smallint;

rec.col\_integer := p\_col\_integer ;

rec.col\_bigint := p\_col\_bigint ;

rec.col\_int2 := p\_col\_int2 ;

rec.col\_int4 := p\_col\_int4 ;

rec.col\_int8 := p\_col\_int8 ;

END;

END pkg;

/

4、PLSQL中调用存储过程。

set serveroutput on;

DECLARE

rec\_p pkg.rec\_type;

begin

pkg.print\_rec\_type(1,11,11,11,34.5,34.5,123,rec\_p);

DBMS\_OUTPUT.PUT\_LINE(rec\_p.col\_int );

DBMS\_OUTPUT.PUT\_LINE(rec\_p.col\_smallint);

DBMS\_OUTPUT.PUT\_LINE(rec\_p.col\_integer );

DBMS\_OUTPUT.PUT\_LINE(rec\_p.col\_bigint );

DBMS\_OUTPUT.PUT\_LINE(rec\_p.col\_int2 );

DBMS\_OUTPUT.PUT\_LINE(rec\_p.col\_int4 );

DBMS\_OUTPUT.PUT\_LINE(rec\_p.col\_int8 );

end;

/

返回结果为：

1

11

11

11

35

35

123

## 支持type自带构造器使用及new构造器语法

**功能描述**

Vastbase支持使用Oracle中的type类型自带的构造器函数创建和使用，以及通过使用new构造器函数来创建新的集合对象。

**语法格式**

collection\_name collection\_type [:=collection\_type(...)];

collection\_name collection\_type [:=new collection\_type(...)];

**参数说明**

* collection\_name：集合变量的名字。
* collection\_type：有两层含义，一层它代表着一个先前以及声明的集合类型的名字，另一层它代表和该类型同名的构造函数。

**注意事项**

该功能仅在数据库兼容模式为Oracle时能够使用（即创建DB时DBCOMPATIBILITY='A'），在其他数据库兼容模式下不能使用该特性。

**示例**

**示例1：** 用type自带构造器和new构造器初始化并赋值变量：

1、创建Oracle兼容模式下的数据库并创建type。

create database db1 dbcompatibility 'A';

\c db1

create type type1 as (i int, j varchar(20));

2、用type自带构造器和new构造器初始化并赋值变量。

declare

n1 type1 := type1();

n2 type1 := type1(1,'n1');

n3 type1 := new type1();

n4 type1 := new type1(1, 'n1');

begin

raise notice '%' ,n1;

raise notice '%' ,n2;

raise notice '%' ,n3;

raise notice '%' ,n4;

end;

/

执行结果：

NOTICE: (,)

NOTICE: (1,n1)

NOTICE: (,)

NOTICE: (1,n1)

ANONYMOUS BLOCK EXECUTE

**示例2：** 在声明类型时直接初始化并赋值变量 。

1、创建Oracle兼容模式下的数据库。

create database db1 dbcompatibility 'A';

\c db1

set serveroutput on;

2、在声明类型时直接用type自带构造器初始化并赋值变量。

DECLARE

TYPE nest\_loc\_type IS TABLE OF VARCHAR2( 13 ) NOT NULL;

nest\_loc\_tab nest\_loc\_type := nest\_loc\_type( 'NEW YORK', 'DALLAS', 'CHICAGO' ); -->在声明时直接初始化并赋值：

BEGIN

FOR i IN 1 .. nest\_loc\_tab.COUNT

LOOP

DBMS\_OUTPUT.put\_line( 'nest\_loc\_tab(' || i || ') value is ' || nest\_loc\_tab( i ) );

END LOOP;

END;

/

结果展示为：

nest\_loc\_tab(1) value is NEW YORK

nest\_loc\_tab(2) value is DALLAS

nest\_loc\_tab(3) value is CHICAGO

ANONYMOUS BLOCK EXECUTE

3、在声明类型时直接用new构造器初始化并赋值变量。

DECLARE

TYPE nest\_loc\_type IS TABLE OF VARCHAR2( 13 ) NOT NULL;

nest\_loc\_tab nest\_loc\_type :=new nest\_loc\_type( 'NEW YORK', 'DALLAS', 'CHICAGO' ); -->在声明时使用new直接初始化并赋值

BEGIN

FOR i IN 1 .. nest\_loc\_tab.COUNT

LOOP

DBMS\_OUTPUT.put\_line( 'nest\_loc\_tab(' || i || ') value is ' || nest\_loc\_tab( i ) );

END LOOP;

END;

/

结果展示为：

nest\_loc\_tab(1) value is NEW YORK

nest\_loc\_tab(2) value is DALLAS

nest\_loc\_tab(3) value is CHICAGO

ANONYMOUS BLOCK EXECUTE

# 对象管理

## 分区表

### 支持全局分区索引

**功能描述**

支持创建全局分区索引。分区索引可以分为本地索引（local index）和全局索引（global index），建立分区索引可以分散数据库的IO压力。就像分区表一样，分区索引提高了可管理性、可用性、性能和可伸缩性。它们既可以独立分区(全局索引)，也可以自动链接到表的分区(本地索引)。

**语法格式**

CREATE [ UNIQUE ] INDEX [ [schema\_name.] index\_name ] ON table\_name [ USING method ]

( {{ column\_name | ( expression ) } [ COLLATE collation ] [ opclass ] [ ASC | DESC ] [ NULLS LAST ] }[, ...] )

[GLOBAL] ；

**参数说明**

* UNIQUE： 指定 UNIQUE为唯一值索引
* schema\_ma：方案名称。
* index\_name：自定义的索引名称。
* table\_name：创建索引所基于的表的名称。
* ASC|DESC：创建索引时指定按照升序或者降序进行排序，ASC：升序，DESC：降序。
* GLOBAL：指定创建的索引为全局索引

**注意事项**

无。

**示例**

1、创建一个测试分区表。

create table t\_part\_1(

id int,

col int

)

partition by range (id)

(

partition p1 values less than (10),

partition p2 values less than (20),

partition p3 values less than (30),

partition p4 values less than (maxvalue)

);

2、创建全局分区索引。

create index idx\_tp\_1 on t\_part\_1(id) global;

当结果返回如下，则表示全局索引创建成功：

CREATE INDEX

### 兼容interval分区表

**功能描述**

interval分区是range分区的扩展。

在没有自动interval分区特性之前，在创建范围Range类型分区表时，我们通常会定义一个maxvalue分区，将没有落在当前关注范围内的数据，都放到该分区中，以避免插入的元组的分区键值不能映射到任何分区的错误。然而当业务场景发生变化时，如果没有及时根据数据增长创建新的分区，就可能会导致分区表的数据发生倾斜，大多数数据都放到了这个未进行细分的maxvlaue分区中，这就违反了我们对表进行分区的初衷，我们希望各个分区的数据量均衡，这样才能加快查询。

自动interval分区可以改善这个问题，对于连续数据类型的Range分区，如果插入的新数据值与当前分区均不匹配，它可以自动创建分区。

**语法格式**

CREATE TABLE table\_name

(

...

)

PARTITION BY RANGE(column1)

INTERVAL (expr)

(

PARTITION partition\_name1 VALUES LESS THAN(literal | MAXVALUE) [TABLESPACE tablespace1],

PARTITION partition\_name2 VALUES LESS THAN(literal | MAXVALUE) [TABLESPACE tablespace2]

);

**参数说明**

* PARTITION BY RANGE(column1)：指定一个range分区的列。
* INTERVAL：指定分区间隔。
* expr：字符串类型，指定时间间隔值。

**注意事项**

无。

**示例**

1、创建interval分区表。

CREATE TABLE interval\_normal\_exchange (logdate date not null)

PARTITION BY RANGE(logdate)

INTERVAL('1 month')

(

PARTITION interval\_normal\_exchange\_p1 VALUES LESS THAN('2020-03-01'),

PARTITION interval\_normal\_exchange\_p2 VALUES LESS THAN('2020-04-01'),

PARTITION interval\_normal\_exchange\_p3 VALUES LESS THAN('2020-05-01')

);

2、执行以下语句查询分区情况。

SELECT relname,parttype,partstrategy,boundaries FROM pg\_partition

WHERE parentid = (SELECT oid FROM pg\_class WHERE relname = 'interval\_normal\_exchange')

ORDER BY relname;

返回的分区情况如下:

relname | parttype | partstrategy | boundaries

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

interval\_normal\_exchange | r | i |

interval\_normal\_exchange\_p1 | p | r | {2020-03-01}

interval\_normal\_exchange\_p2 | p | r | {2020-04-01}

interval\_normal\_exchange\_p3 | p | r | {2020-05-01}

(4 rows)

### 支持default分区

**功能描述**

针对range，list和hash分区，支持定义和使用default分区。创建分区表的时候可以指定创建default分区，对于已经创建的分区表，可以新增和删除default分区。

Range类型的分区表，可以通过指定分区范围值为MAXVALUE来包含所有情况的值。

Hash类型的分区表，由于Hash分区表的特性，必然可以包含所有情况的值。

**语法格式**

* 创建分区表时指定创建default分区，defaultf分区必须定义为最后一个分区，SQL语法如下：

CREATE TABLE [IF NOT EXISTS]table\_name

...

PARTITION BY [RANGE | LIST| HASH]({column\_name|(expression)})

(

PARTITION partition\_name...,

PARTITION partition\_name VALUES(DEFAULT)[TABLESPACE tablespace\_name]

)

* 新增default分区(已存在default分区的LIST分区不能新增分区)SQL语法如下：

ALTER TABLE table\_name ADD PARTITION partition\_name VALUES(DEFAULT)

[TABLESPACE tablespace\_name];

* 删除default分区(不能只剩下一个default分区)SQL语法如下：

ALTER TABLE partition\_name DROP PARTITION partition\_name;

**注意事项**

* 该功能仅在数据库兼容模式为Oracle时能够使用（即创建DB时DBCOMPATIBILITY='A'），在其他数据库兼容模式下不能使用该特性。
* 在创建分区表时，不能只定义一个default分区并且default分区必须定义在最后。
* 已经存在default分区的分区表，不能新增分区。
* 删除分区时，不能只剩下一个default分区。
* default分区不能进行SPLIT和MERGE操作。

**示例**

1、创建并切换至兼容模式为ORACLE的数据库db\_oracle。

CREATE DATABASE db\_oracle dbcompatibility='A';

\c db\_oracle

2、创建一个分区表。

create table t\_partition\_list4(col number,name varchar2(20))

partition by list(col)(

partition t\_list\_p1 values(1,3,5,7,9) ,

partition t\_list\_p2 values(2,4,6,8,10) );

2、添加一个默认分区并插入数据。

alter table t\_partition\_list4 add partition t\_list\_default values(default);

insert into t\_partition\_list4 values(1,'t\_list\_p1');

insert into t\_partition\_list4 values(2,'t\_list\_p2');

insert into t\_partition\_list4 values(31,'t\_list\_default');

3、查询分区表中的数据。

* 查询分区表中的数据t\_list\_p1.

select \* from t\_partition\_list4 partition(t\_list\_p1);

返回结果为：

col | name

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

1 | t\_list\_p1

(1 row)

* 查询分区表中的数据t\_list\_p2。

select \* from t\_partition\_list4 partition(t\_list\_p2);

返回结果为：

col | name

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

2 | t\_list\_p2

(1 row)

* t\_list\_default

select \* from t\_partition\_list4 partition(t\_list\_default);

返回结果为：

col | name

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

31 | t\_list\_default

(1 row)

## 触发器

### 兼容IF Inserting THEN,IF updating THEN,IF deleting THEN

**功能描述**

该功能允许在触发器函数中使用IF Inserting THEN，IF updating THEN和IF deleting THEN命令来判断增加、修改和删除记录的dml操作。

**注意事项**

无。

**示例**

* **IF Inserting THEN**

1、创建测试表。

create table tri\_tbl(id int, name varchar(256));

2、创建函数。

CREATE OR REPLACE FUNCTION tri\_func() RETURNS TRIGGER AS

$$

DECLARE

n\_id int := new.id;

o\_id int := old.id;

BEGIN

raise notice 'trigger begin: TG\_OP is %',TG\_OP;

if inserting then

raise notice 'old id:%,old name:%',o\_id,old.name;

raise notice 'new id:%,new name:%',n\_id,new.name;

end if;

return new;

END $$ LANGUAGE plpgsql;

3、创建触发器。

CREATE TRIGGER be\_tri

BEFORE INSERT OR DELETE OR UPDATE

ON tri\_tbl

FOR EACH ROW

EXECUTE PROCEDURE tri\_func();

4、调用函数。

insert into tri\_tbl values(1,'aaaaa'),(2,'bbb');

返回结果为：

NOTICE: trigger begin: TG\_OP is INSERT

NOTICE: old id:<NULL>,old name:<NULL>

NOTICE: new id:1,new name:aaaaa

NOTICE: trigger begin: TG\_OP is INSERT

NOTICE: old id:<NULL>,old name:<NULL>

NOTICE: new id:2,new name:bbb

INSERT 0 2

* **IF updating THEN**

1、创建测试表并插入数据。

CREATE TABLE tri\_tbl1(id int, name varchar(256));

insert into tri\_tbl1 values(1,null),(1,'bbb');

2、创建函数。

CREATE OR REPLACE FUNCTION tri\_func() RETURNS TRIGGER AS

$$

DECLARE

n\_id int := new.id;

o\_id int := old.id;

BEGIN

raise notice 'trigger begin: TG\_OP is %',TG\_OP;

if updating then

raise notice 'old id:%,old name:%',o\_id,old.name;

raise notice 'new id:%,new name:%',n\_id,new.name;

end if;

return new;

END $$ LANGUAGE plpgsql;

3、创建触发器。

CREATE TRIGGER be\_tri

BEFORE INSERT OR DELETE OR UPDATE

ON tri\_tbl1

FOR EACH ROW

EXECUTE PROCEDURE tri\_func();

4、调用函数。

update tri\_tbl1 set name='aaaaa' where id=1;

返回结果为：

NOTICE: old id:1,old name:<NULL>

NOTICE: new id:1,new name:aaaaa

NOTICE: trigger begin: TG\_OP is UPDATE

NOTICE: old id:1,old name:bbb

NOTICE: new id:1,new name:aaaaa

UPDATE 2

* **IF deleting THEN**

1、创建测试表并插入数据。

CREATE TABLE tri\_tb(id int, name varchar(256));

insert into tri\_tb values(1,null),(1,'aaa');

2、创建函数。

CREATE OR REPLACE FUNCTION tri\_func1() RETURNS TRIGGER AS

$$

DECLARE

n\_id int := new.id;

o\_id int := old.id;

BEGIN

raise notice 'trigger begin: TG\_OP is %',TG\_OP;

if deleting then

raise notice 'old id:%,old name:%',o\_id,old.name;

raise notice 'new id:%,new name:%',n\_id,new.name;

end if;

return new;

END $$ LANGUAGE plpgsql;

3、创建触发器。

CREATE TRIGGER be\_tri

BEFORE INSERT OR DELETE OR UPDATE

ON tri\_tb

FOR EACH ROW

EXECUTE PROCEDURE tri\_func1();

4、调用函数。

delete tri\_tb where id=1;

返回结果为：

NOTICE: trigger begin: TG\_OP is DELETE

NOTICE: old id:1,old name:<NULL>

NOTICE: new id:<NULL>,new name:<NULL>

NOTICE: trigger begin: TG\_OP is DELETE

NOTICE: old id:1,old name:aaa

NOTICE: new id:<NULL>,new name:<NULL>

DELETE 2

### 兼容IF UPDATING('col1')

**功能描述**

允许触发器函数中对单个表字段的修改使用IF UPDATING('col1')进行判断单个表字段的更新。

**注意事项**

该功能仅在数据库兼容模式为Oracle时能够使用（即创建DB时DBCOMPATIBILITY='A'），在其他数据库兼容模式下不能使用该特性。

**示例**

1、检查数据库兼容模式DBCOMPATIBILITY为A（即兼容Oracle）。

show sql\_compatibility;

2、创建测试表并插入数据。

DROP TABLE IF EXISTS tri\_tbl;

CREATE TABLE tri\_tbl(id int, name varchar(256),note\_pad varchar(256));

insert into tri\_tbl values(1,null,'aaa');

insert into tri\_tbl values(1,null,'aaa');

3、创建函数。

CREATE OR REPLACE FUNCTION tri\_func() RETURNS TRIGGER AS

$$

DECLARE

n\_id int := new.id;

o\_id int := old.id;

BEGIN

raise notice 'trigger begin: TG\_OP is %',TG\_OP;

if updating('name') then

raise notice 'old name:%',old.name;

raise notice 'new name:%',new.name;

end if;

if updating('note\_pad') then

raise notice 'old note\_pad:%',old.note\_pad;

raise notice 'new note\_pad:%',new.note\_pad;

end if;

return new;

END $$ LANGUAGE 'plpgsql';

4、创建触发器。

CREATE TRIGGER be\_tri

BEFORE INSERT OR DELETE OR UPDATE

ON tri\_tbl

FOR EACH ROW

EXECUTE PROCEDURE tri\_func();

5、调用函数。

update tri\_tbl set name='bbb',note\_pad='666' where id=1;

返回结果为：

NOTICE: trigger begin: TG\_OP is UPDATE

NOTICE: old name:<null>

NOTICE: new name:bbb

NOTICE: old note\_pad:aaa

NOTICE: new note\_pad:666

UPDATE 1

### 允许INSERT使用oid.colname获取值为空

**功能描述**

兼容Oracle的触发器特性，允许触发器在指定条件下取NULL值。在触发器中，支持使用oid.colname语法获取dml操作之前的旧值，如果之前是insert操作，oid.colname结果为空，如果之前值update和delete操作，oid.colname语法获取的值为操作之前的旧值。

**注意事项**

无。

**示例**

1、创建测试表。

CREATE TABLE tri\_tbl(id int, name varchar(256));

2、创建函数。

CREATE OR REPLACE FUNCTION tri\_func() RETURNS TRIGGER AS

$$

DECLARE

n\_id int := new.id;

o\_id int := old.id;

BEGIN

raise notice 'trigger begin: TG\_OP is %',TG\_OP;

if TG\_OP='INSERT' then

raise notice 'old id:%,old name:%',o\_id,old.name;

raise notice 'new id:%,new name:%',n\_id,new.name;

end if;

return new;

END $$ LANGUAGE plpgsql;

3、创建触发器。

CREATE TRIGGER be\_tri

BEFORE INSERT OR DELETE OR UPDATE

ON tri\_tbl

FOR EACH ROW

EXECUTE PROCEDURE tri\_func();

4、调用函数。

insert into tri\_tbl values(1,'aaaaa');

返回结果为：

NOTICE: trigger begin: TG\_OP is INSERT

NOTICE: old id:<NULL>,old name:<NULL>

NOTICE: new id:1,new name:aaaaa

INSERT 0 1

## 序列

### SEQUENCE兼容RESTART语法

**功能描述**

SEQUENCE支持restart语法，通过该语法可以重新设置序列的起始值。

**语法格式**

ALTER SEQUENCE name RESTART [WITH] number;

**参数说明**

* name：序列名称。
* number：序列的起始值。

**注意事项**

无。

**示例**

1、创建一个序列起始值为32。

CREATE SEQUENCE sequence\_test20 START WITH 32;

2、使用nextval返回序列的一个值。

SELECT nextval('sequence\_test20');

返回结果为：

nextval

---------

32

(1 row)

3、修改序列的起始值。

ALTER SEQUENCE sequence\_test20 RESTART WITH 22;

4、查询修改结果。

SELECT nextval('sequence\_test20');

返回结果为：

nextval

---------

22

(1 row)

### 兼容NO CACHE、NOCACHE、ORDER、NO ORDER、NOORDER关键字

**功能描述**

提供类似Oracle的序列功能、支持创建（create）或修改（alter）时使用：no cache、nocache、order、no order、noorder。

**语法格式**

* CREATE

CREATE [ TEMPORARY | TEMP ] SEQUENCE [ IF NOT EXISTS ] name [ INCREMENT [ BY ] increment ]

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

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

[ NOORDER ] [ [ NO ] ORDER ] [ OWNED BY { table\_name.column\_name | NONE } ]

* ALTER

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 ] [ NOCACHE ] [ NO CACHE ] [ [ NO ] CYCLE ]

**参数说明**

* no cache、nocache：相当于cache 1（cache：设置缓存的序列值个数）。
* Order：保证序列号按请求顺序产生。如果想以序列号作为timestamp(时间戳)类型的话，可以采用该选项。
* noorder、no order：与order相对，不能保证序列号按请求顺序产生。

**注意事项**

无。

**示例**

1、 创建序列是使用nocache关键字。

create sequence SEQ\_ID3

minvalue 1

maxvalue 99999999

start with 1

increment by 1

nocache

noorder;

返回结果如下，则表示序列创建成功：

CREATE SEQUENCE

2、修改序列。

alter sequence SEQ\_ID3 order;

返回结果如下，则表示修改成功：

ALTER SEQUENCE

# DBLINK库级访问

**功能描述**

当用户需要跨越本地数据库，访问远程数据库的数据时，可以通过DBLINK像访问本地数据库一样访问远程数据库表中的数据。在Oracle兼容模式下，Vastbase G100支持使用tablename@dblink\_name语法进行表的增删改查功能。

**语法格式**

* 创建DBLINK。

CREATE [PUBLIC] DATABASE LINK ${DBLINK\_NAME} CONNECT TO ${USERNAME}

IDENTIFIED BY '${PASSWORD}' USING ${FDW\_NAME}

({HOST},{PORT},{DBNAME});

* 删除DBLINK。

DROP [PUBLIC] DATABASE LINK ${DBLINK\_NAME}

* 通过DBLINK进行远端表的增删改查操作。
* 插入

INSERT INTO [${SCHEMA\_NAME}].${TABLE\_NAME}@${DBLINK\_NAME}

* 更新

UPDATE [${SCHEMA\_NAME}].${TABLE\_NAME}@${DBLINK\_NAME} SET... #

* 删除

DELETE FROM [${SCHEMA\_NAME}].${TABLE\_NAME}@${DBLINK\_NAME} WHERE...

* 查询

SELECT ... FROM [${SCHEMA\_NAME}].${TABLE\_NAME}@${DBLINK\_NAME}

**参数说明**

* **${DBLINK\_NAME}**

连接名称，可自定义。

* **${SCHEMA\_NAME}**

目标函数或存储过程所在的模式名（或者package名称）。可不指 定，不指定时默认访问database link中Oracle用户对应的模式。

* **${TABLE\_NAME}**

远程访问的数据库表名。

**注意事项**

* 该功能仅在数据库兼容模式为Oracle时能够使用（即创建DB时DBCOMPATIBILITY='A'），在其他数据库兼容模式下不能使用该特性。
* 创建DBLINK时，必须提前创建postgres\_fdw（参见：开发者指南->外部数据封装器 -> PostgreSQL\_FDW章节），并具备DBLINK相关权限。
* 暂不支持同义词的DBLINK。

**示例**

1、在远端数据库创建用户testlink\_user并授权。

create user testlink\_user with sysadmin password '123456Aa';

grant all on database vastbase to testlink\_user;

2、在远端数据库创建表emp\_fdw，并插入数据。

create table emp\_fdw(empno int primary key,ename varchar(30));

insert into emp\_fdw values(1,'foo');

insert into emp\_fdw values(2,'bar');

3、在本地数据库创建用户user\_1 并授权。

create user user\_1 password 'Aa123456';

grant all on database vastbase to user\_1;

4、关闭远端数据库和本地数据库的强制修改密码功能。

alter system set password\_force\_alter=off;

5、在本地数据库，把当前会话里的会话用户标识和当前用户标识都设置为用户user\_1 。

set session session authorization user\_1 password 'Aa123456';

6、在本地数据库创建DBLINK。

CREATE DATABASE LINK dblink\_84 CONNECT TO usr IDENTIFIED BY '123456Aa' USING postgres\_fdw(host '172.16.105.57',port '10929',dbname 'vastbase');

7、在本地数据库通过dblink\_84查询表emp\_fdw的数据。

select \* from emp\_fdw@dblink\_84;

返回结果如下，表示查询到了远端数据库中的emp\_fdw表的内容。

empno | ename

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

1 | foo

2 | bar

(2 rows)

# OCI/OCCI接口

**功能描述**

支持Oracle Call Interface和Oracle C++ Call Interface的函数，具体见下表：

| **函数** | **描述** |
| --- | --- |
| OCILogon | 创建一个简单的登录会话 |
| OCILogoff | 释放使用OCILogon登录的会话 |
| SQ | 执行模块实现函数 |
| OCIStmtPrepar | 函数实现准备要执行的SQL或PL / SQL语句 |
| OCIBindByPos | 在SQL语句或PL / SQL块中的程序变量和占位符之间创建关联 |
| OCIBindByName | 函数实现在SQL语句或PL / SQL块中的程序变量和占位符之间创建关联 |
| OCIBindArrayOfStruct | 函数实现为静态数组绑定设置跳过参数 |
| OCIDefineByPos | 函数实现将选择列表中的项目与类型和输出数据缓冲区关联 |
| OCIDefineArrayOfStruct | 指定在多行多列结构读取数组中使用的静态数组定义所必需的其他属性 |
| OCIStmtExecute | 函数实现将应用程序请求与服务器关联 |
| OCIStmtFetch | 函数实现从查询中获取行 |
| OCITransStart | 函数实现设置事务开始 |
| OCITransCommit | 函数实现事务提交 |
| OCITransRollback | 函数实现事务回滚 |
| OCIHandleAlloc | 函数返回指向已分配和初始化的句柄的指针 |
| OCIHandleFree | 函数实现该调用显式取消分配句柄 |
| OCIErrorGet | 在提供的缓冲区中返回错误消息和Oracle数据库错误代码 |
| OCIDateToText | 函数实现将日期类型转换为字符串 |
| OCINumberFromText | 函数实现将number类型转换为字符类型 |
| OCINumberFromReal | 函数实现将浮点类型转化为number类型 |
| OCINumberfromInt | 函数实现将int类型转换为number类型 |
| OCIDateSysDate | 函数实现获取客户端的当前系统日期和时间 |
| OCIDateCompare | 函数实现两个日期进行比较 |
| OCINumberCmp | 函数实现将两个number类型进行比较 |
| OCINumberAdd | 函数实现将两个number类型值进行相加 |
| OCINumberSub | 函数实现将两个number类型值进行相减 |
| OCINumberMul | 函数实现将两个number类型值进行相乘 |
| OCINumberDiv | 函数实现将两个number类型值进行相除 |
| OCIParamGet | 函数返回由描述句柄或语句句柄中的位置指定的参数的描述符 |

**注意事项**

该功能仅在数据库兼容模式为Oracle时能够使用（即创建DB时DBCOMPATIBILITY='A'），在其他数据库兼容模式下不能使用该特性。

# VPD功能

**功能描述**

通过添加管理策略功能可针对某个模式schema中的某个对象object（包括table或视图view）添加管理策略。

**语法格式/参数说明**

* 添加VPD策略

dbms\_rls.add\_policy(

Object\_name text, --对象名称

policy\_name text, --策略名称

policy\_function text, --策略函数名称

object\_schema text DEFAULT NULL::text, --对象模式，默认为空

function\_schema text DEFAULT NULL::text, --函数模式，默认为空

statement\_types text DEFAULT 'insert,update,delete,select'::text, --策略作用的语句类型，语句类型使用英文逗号分隔，默认为insert,update,delete,select

update\_check boolean DEFAULT false, --是否启动更新检查，默认为false

enable boolean DEFAULT true, --是否启用策略

sec\_relevant\_cols text DEFAULT NULL:text --敏感列，敏感列使用英文逗号分隔

) RETURN VOID --返回值为空

* 删除VPD策略

dbms\_rls.drop\_policy(

object\_name text, --对象名称

policy\_name text, --策略名称

object\_schema text DEFAULT NULL:text --对象模式，默认为空

) RETURN VOID --返回值为空

* 启动/禁用

dbms\_rls.enable\_policy(

object\_name text, --对象名称

policy\_name text, --策略名称

enable boolean, --是否启用策略

object\_schema text DEFAULT NULL:text --对象模式，默认为空

) RETURN VOID --返回值为空

**注意事项**

无。

**示例**

1、创建并设置模式为vpd。

create schema vpd;

set search\_path to vpd;

2、创建测试表并插入数据。

create table t\_vpd (x number);

insert into t\_vpd values (1);

insert into t\_vpd values (2);

insert into t\_vpd values (10001);

insert into t\_vpd values (10002);

3、创建测试函数。

CREATE OR REPLACE FUNCTION vpd.f\_limited\_query\_t(s\_schema IN text,s\_object IN text)

RETURNS text AS $$

BEGIN

RETURN 'x <= 10000';

END;

$$ LANGUAGE plpgsql;

4、添加策略。

BEGIN

DBMS\_RLS.ADD\_POLICY (

object\_schema => 'vpd',

object\_name => 't\_vpd',

policy\_name => 'policy\_t\_vpd\_select',

function\_schema => 'vpd',

policy\_function => 'f\_limited\_query\_t',

statement\_types => 'select'

);

END;

/

5、启用策略。

BEGIN

DBMS\_RLS.enable\_policy(

object\_schema => 'vpd',

object\_name => 't\_vpd',

policy\_name => 'policy\_t\_vpd\_select',

enable => true

);

END;

/

6、如下所示：查询会自动带上where条件。

explain (COSTS false) select \* from t\_vpd;

QUERY PLAN

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

Seq Scan on t\_vpd

Filter: (x <= 10000::numeric)

(2 rows)

# 闪回/闪回查询

**功能描述**

在使用表的过程中，可能会出现误删，或者误改数据，此时闪回查询可以通过语句找回INSERT，UPDATE，DELETE操作之前的数据，看到修改表数据前或者删除数据前的数据装填，支持指定时间的闪回查询操作。

**注意事项**

* track\_commit\_timestamp设置为on。
* autovacuum设置为off。
* 数据表truncate后无法闪回。
* 闪回时间点的数据没有被autocacuum回收。

**使用步骤**

1、修改postgresql.conf，添加参数max\_flashback\_time，开启闪回功能，并重启数据库。

例如设置max\_flashback\_time=60，即最多允许闪回至60秒之内。

vim postgresql.conf

vb\_ctl reatart

2、开启自动提交。

\set AUTOCOMMIT on

\echo :AUTOCOMMIT

3、创建测试表并查询表中数据。

create table t\_a (col1 int, col2 char(20),col3 char(20));

insert into t\_a values(1,'aa','a1');

insert into t\_a values(2,'bb','b1');

insert into t\_a values(3,'cc','c1');

SELECT \* FROM t\_a;

返回结果为：

col1 | col2 | col3

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

1 | aa | a1

2 | bb | b1

3 | cc | c1

(3 rows)

4、修改表中的数据并查询结果。

select pg\_sleep(17);

insert into t\_a values (4,'dd','d4');

SELECT \* FROM t\_a;

返回结果为：

col1 | col2 | col3

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

1 | aa | a1

2 | bb | b1

3 | cc | c1

4 | dd | d4

(4 rows)

5、进行闪回查询。

select \* from t\_a FLASHBACK(15);

col1 | col2 | col3

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

1 | aa | a1

2 | bb | b1

3 | cc | c1

(3 rows)

6、将表闪回至15秒之前。

select pg\_flashback('t\_a',15);

select \* from t\_a;

col1 | col2 | col3

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

1 | aa | a1

2 | bb | b1

3 | cc | c1

(3 rows)

# Xmltype模块

**术语定义**

Xml一般指可扩展标记语言。可扩展标记语言,标准通用标记语言的子集,简称xml。是一种用于标记电子文件使其具有结构性的标记语言。在电子信息中,标记指计算机所能理解的符号,通过此种标记,计算机之间可以处理包含各种信息的文章。

**注意事项**

* 该功能仅在数据库兼容模式为Oracle时能够使用（即创建DB时DBCOMPATIBILITY='A'），在其他数据库兼容模式下不能使用该特性。
* 使用xml相关功能,在初始化的时候,需要在数据库安装时候使用编译选项

--with-libxml。

**Xmltype类型**

Xmltype 是系统定义的类型，用于存储和管理XML数据，并提供了很多的functions，用来保存、检索和操作xml文档和管理节点。

用户建表时可以指定字段类型为Xmltype或者在函数中定义Xmltype类型的变量，此后用户可以使用Xmltype相关的函数进行xml类型数据对象的初始化或者从xml类型字段内容的提取。

**语法格式**

针对表定义Xmltype字段:

Colname xmltyle;

针对函数或者匿名块：

declare

data Xmltype;

begin

null;

end;

/

**示例**（**xmltype增删改查**）

1、创建并切换至兼容模式为Oracle的数据库db\_oracle。

CREATE DATABASE db\_oracle dbcompatibility='A';

\c db\_oracle

2、创建测试表并插入数据。

drop table if exists test\_xmltype\_1087931;

create table test\_xmltype\_1087931(i int, d xmltype);

insert into test\_xmltype\_1087931 values(1,'abc');

insert into test\_xmltype\_1087931 values(2,'<test>abc</test>');

insert into test\_xmltype\_1087931 values(3,'abc \153\154\155 \052\251\124');

3、查询数据。

select \* from test\_xmltype\_1087931;

返回结果为：

i | d

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

1 | abc

2 | <test>abc</test>

3 | abc \153\154\155 \052\251\124

(3 rows)

4、修改数据。

update test\_xmltype\_1087931 set d='abc11' where i=1;

select \* from test\_xmltype\_1087931;

返回结果为：

i | d

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

2 | <test>abc</test>

3 | abc \153\154\155 \052\251\124

1 | abc11

(3 rows)

5、修改数据。

update test\_xmltype\_1087931 set d=' <foo>abc</foo><bar>123</bar>' where i=3;

select \* from test\_xmltype\_1087931;

返回结果为：

i | d

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

2 | <test>abc</test>

1 | abc11

3 | <foo>abc</foo><bar>123</bar>

(3 rows)

6、删除数据。

delete from test\_xmltype\_1087931 where i=1;

select \* from test\_xmltype\_1087931;

返回结果为：

i | d

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

2 | <test>abc</test>

3 | <foo>abc</foo><bar>123</bar>

(2 rows)

**数据初始化**

Xmltype类型提供相应的函数对表字段或者xml类型变量进行初始化。

**Createxml**

**功能描述**

用于创建和返回 Xmltype 实例的函数,从字符串创建 Xmltype 实例,返回生成的Xmltype对象。

**语法格式**

FUNCTION createXML(

xmlData IN varchar2)

RETURN Xmltype

**使用createxml和xmltype函数初始化xml数据**

1、创建并切换至兼容模式为Oracle的数据库db\_oracle。

CREATE DATABASE db\_oracle dbcompatibility='A';

\c db\_oracle

2、创建测试表。

drop table if exists test\_xmltype\_1087938;

create table test\_xmltype\_1087938(

ID int,

name varchar2(256),

data xmltype

);

3、使用正确格式的xml文档。

Insert INTO test\_xmltype\_1087938

VALUES (1,'test xml doc',xmlType.createXML('<?xml version="1.0" encoding="UTF-8" ?>

<collection xmlns="">

<record>

<leader>-----nam0-22-----^^^450-</leader>

<datafield tag="200" ind1="1" ind2=" ">

<subfield code="a">抗震救灾</subfield>

<subfield code="f">奥运会</subfield>

</datafield>

<datafield tag="209" ind1=" " ind2=" ">

<subfield code="a">经济学</subfield>

<subfield code="b">计算机</subfield>

<subfield code="c">10001</subfield>

<subfield code="d">2005-07-09</subfield>

</datafield>

<datafield tag="610" ind1="0" ind2=" ">

<subfield code="a">计算机</subfield>

<subfield code="a">笔记本</subfield>

</datafield>

</record>

</collection>')) ;

Insert INTO test\_xmltype\_1087938

VALUES (2,'test xml doc',xmltype('<?xml version="1.0" encoding="UTF-8" ?>

<collection xmlns="">

<record>

<leader>-----nam0-22-----^^^450-</leader>

<datafield tag="209" ind1=" " ind2=" ">

<subfield code="a">经济学</subfield>

<subfield code="b">计算机</subfield>

<subfield code="c">10001</subfield>

<subfield code="d">2005-07-09</subfield>

</datafield>

</record>

</collection>')) ;

4、查看表数据。

select \*from test\_xmltype\_1087938;

返回结果为：

id | name | data

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

1 | test xml doc | <collection xmlns=""> +

| | <record> +

| | <leader>-----nam0-22-----^^^450-</leader>+

| | <datafield tag="200" ind1="1" ind2=" "> +

| | <subfield code="a">抗震救灾</subfield> +

| | <subfield code="f">奥运会</subfield> +

| | </datafield> +

| | <datafield tag="209" ind1=" " ind2=" "> +

| | <subfield code="a">经济学</subfield> +

| | <subfield code="b">计算机</subfield> +

| | <subfield code="c">10001</subfield> +

| | <subfield code="d">2005-07-09</subfield> +

| | </datafield> +

| | <datafield tag="610" ind1="0" ind2=" "> +

| | <subfield code="a">计算机</subfield> +

| | <subfield code="a">笔记本</subfield> +

| | </datafield> +

| | </record> +

| | </collection

2 | test xml doc | <collection xmlns=""> +

| | <record> +

| | <leader>-----nam0-22-----^^^450-</leader>+

| | <datafield tag="209" ind1=" " ind2=" "> +

| | <subfield code="a">经济学</subfield> +

| | <subfield code="b">计算机</subfield> +

| | <subfield code="c">10001</subfield> +

| | <subfield code="d">2005-07-09</subfield> +

| | </datafield> +

| | </record> +

| | </collection>

(2 rows)

**Xml数据的查询检索**

**Extract**

**功能描述**

Xmltype提供函数extract便于对xml对象中多个满足条件的标签进行提取, 返回的是满足条件的字标签类型。

**语法格式**

extract(xml\_obj Xmltype ,xml\_path varchar2)

其中xml\_path格式如下：

tagname{[|@attname=”attvalue”]}{/tagname{[|@attname=”attvalue”]}{ /tagname{[|@attname=”attvalue”]}}...}

例如：

* 前文中的xml文本中标签collection下有标签record,record下有标签datafield, datafield下有标签subfield. 依次查找 所有的subfield ,其形式为:

'/collection/record/datafield/subfield'

* 针对每一层标签,可以根据@attname=”attvalue”筛选,只查找满足条件的标签。如只找出datafield标签满足条件 tag="209" 且 subfield 满足条件 code="a" 的所有 subfield,使用如下格式:

'/collection/record/datafield[@tag="209"]/subfield[@code="a"]'

**参数说明**

* xml\_obj：xml对象,如xml类型字段或变量。
* xml\_path：xml内容搜索标签的先后顺序。

**示例**

1、创建并切换至兼容模式为Oracle的数据库db\_oracle。

CREATE DATABASE db\_oracle dbcompatibility='A';

\c db\_oracle

2、创建测试表。

drop table if exists test\_xmltype\_1087939;

create table test\_xmltype\_1087939(

ID int,

name varchar2(257),

data xmltype

);

3、插入数据。

Insert INTO test\_xmltype\_1087939

VALUES (1,'test xml doc',xmlType.createXML('<?xml version="1.0" encoding="UTF-8" ?>

<collection xmlns="">

<record>

<leader>-----nam0-22-----^^^450-</leader>

<datafield tag="200" ind1="1" ind2=" ">

<subfield code="a">抗震救灾</subfield>

<subfield code="f">奥运会</subfield>

</datafield>

<datafield tag="209" ind1=" " ind2=" ">

<subfield code="a">经济学</subfield>

<subfield code="b">计算机</subfield>

<subfield code="c">10001</subfield>

<subfield code="d">2005-07-09</subfield>

</datafield>

<datafield tag="610" ind1="0" ind2=" ">

<subfield code="a">计算机</subfield>

<subfield code="a">笔记本</subfield>

</datafield>

</record>

</collection>')) ;

4、使用exact提取标签。

select extract(x.data,'/collection/record/datafield/subfield') xmlseq from test\_xmltype\_1087939 x;

返回结果为：

xmlseq

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

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

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

{"<subfield code=\"a\">抗震救灾</subfield>","<subfield code=\"f\">奥运会</subfield>","<subfield code=\"a\">经济学</subfield>","<subfield

code=\"b\">计算机</subfield>","<subfield code=\"c\">10001</subfield>","<subfield code=\"d\">2005-07-09</subfield>","<subfield code=\"a\"

>计算机</subfield>","<subfield code=\"a\">笔记本</subfield>"}

(1 row)

select extract(x.data,'/collection/record/datafield') xmlseq from test\_xmltype\_1087939 x;

返回结果为：

xmlseq

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

{"<datafield tag=\"200\" ind1=\"1\" ind2=\" \"> +

<subfield code=\"a\">抗震救灾</subfield> +

<subfield code=\"f\">奥运会</subfield> +

</datafield>","<datafield tag=\"209\" ind1=\" \" ind2=\" \">+

<subfield code=\"a\">经济学</subfield> +

<subfield code=\"b\">计算机</subfield> +

<subfield code=\"c\">10001</subfield> +

<subfield code=\"d\">2005-07-09</subfield> +

</datafield>","<datafield tag=\"610\" ind1=\"0\" ind2=\" \">+

<subfield code=\"a\">计算机</subfield> +

<subfield code=\"a\">笔记本</subfield> +

</datafield>"}

(1 row)

**XmlSequence**

**功能描述**

Etract提取的是多个标签, 而 xmlsequence能够将多个Xmltype类型返回成一个自顶级节点往下的xml数组。

**语法格式**

Xmlsequence(&xmldoc);

**示例**

1、创建并切换至兼容模式为Oracle的数据库db\_oracle。

CREATE DATABASE db\_oracle dbcompatibility='A';

\c db\_oracle

2、创建测试表。

show sql\_compatibility;

drop table if exists test\_xmltype\_1087941;

create table test\_xmltype\_1087941(

ID int,

name varchar2(259),

data xmltype

);

3、插入数据。

Insert INTO test\_xmltype\_1087941

VALUES (1,'test xml doc',xmlType.createXML('<?xml version="1.0" encoding="UTF-8" ?>

<collection xmlns="">

<record>

<leader>-----nam0-22-----^^^450-</leader>

<datafield tag="200" ind1="1" ind2=" ">

<subfield code="a">抗震救灾</subfield>

<subfield code="f">奥运会</subfield>

</datafield>

<datafield tag="209" ind1=" " ind2=" ">

<subfield code="a">经济学</subfield>

<subfield code="b">计算机</subfield>

<subfield code="c">10001</subfield>

<subfield code="d">2005-07-09</subfield>

</datafield>

<datafield tag="610" ind1="0" ind2=" ">

<subfield code="a">计算机</subfield>

<subfield code="a">笔记本</subfield>

</datafield>

</record>

</collection>')) ;

4、使用XmlSequence将提取的数据转为数组并通过unnest函数转为表。

select unnest(XMLSequence(extract(x.data,'/collection/record/datafield/subfield'))) xmlseq from test\_xmltype\_1087941 x;

返回结果为：

xmlseq

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

<subfield code="a">抗震救灾</subfield>

<subfield code="f">奥运会</subfield>

<subfield code="a">经济学</subfield>

<subfield code="b">计算机</subfield>

<subfield code="c">10001</subfield>

<subfield code="d">2005-07-09</subfield>

<subfield code="a">计算机</subfield>

<subfield code="a">笔记本</subfield>

(8 rows)

**ExtractValue**

**功能描述**

Extract提取函数用于提取xml对象满足条件的标签,而Extractvalue用于提取满足条件标签的值。

**语法格式**

extract(xml\_obj ,'&xml\_path')

其中 xml\_path 格式格式如下:

tagname{[|@attname=”attvalue”]}{/tagname{[|@attname=”attvalue”]}{ /tagname{[|@attname=”attvalue”]}}...}

**示例**

1、创建并切换至兼容模式为Oracle的数据库db\_oracle。

CREATE DATABASE db\_oracle dbcompatibility='A';

\c db\_oracle

2、创建测试表。

drop table if exists test\_xmltype\_1087942;

create table test\_xmltype\_1087942(

ID int,

name varchar2(260),

data xmltype

);

3、插入数据。

Insert INTO test\_xmltype\_1087942

VALUES (1,'test xml doc',xmlType.createXML('<?xml version="1.0" encoding="UTF-8" ?>

<collection xmlns="">

<record>

<leader>-----nam0-22-----^^^450-</leader>

<datafield tag="200" ind1="1" ind2=" ">

<subfield code="a">抗震救灾</subfield>

<subfield code="f">奥运会</subfield>

</datafield>

<datafield tag="209" ind1=" " ind2=" ">

<subfield code="a">经济学</subfield>

<subfield code="b">计算机</subfield>

<subfield code="c">10001</subfield>

<subfield code="d">2005-07-09</subfield>

</datafield>

<datafield tag="610" ind1="0" ind3=" ">

<subfield code="a">计算机</subfield>

<subfield code="a">笔记本</subfield>

</datafield>

</record>

</collection>')) ;

4、使用ExactValue函数提取满足标签的返回单个值。

select extractvalue(x.data,'/collection/record/leader') as A from test\_xmltype\_1087942 x;

返回结果为：

a

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

-----nam0-22-----^^^450-

(1 row)

select extractvalue(x.data,'/collection/record/datafield[@tag="209"]/subfield[@code="a"]') as A from test\_xmltype\_1087942 x;

返回结果为：

a

--------

经济学

(1 row)

**ExistsNode**

**功能描述**

成员函数。 检查节点是否存在。 如果 XPath 字符串为 NULL 或文档为空，则返回值 0，否则返回 1。

**语法格式**

FUNCTION existsNode(

Xml\_obj Xmltype,

xpath IN varchar2)

RETURN number deterministic;

其中xpath格式参考extract。

**示例**

1、创建并切换至兼容模式为Oracle的数据库db\_oracle。

CREATE DATABASE db\_oracle dbcompatibility='A';

\c db\_oracle

2、创建测试表。

drop table if exists test\_xmltype\_1087946;

create table test\_xmltype\_1087946(

ID int,

name varchar2(264),

data xmltype

);

3、插入数据。

Insert INTO test\_xmltype\_1087946

VALUES (1,'test xml doc',xmlType.createXML('<?xml version="1.0" encoding="UTF-8" ?>

<collection xmlns="">

<record>

<leader>-----nam0-22-----^^^450-</leader>

<datafield tag="200" ind1="1" ind2=" ">

<subfield code="a">抗震救灾</subfield>

<subfield code="f">奥运会</subfield>

</datafield>

<datafield tag="209" ind1=" " ind2=" ">

<subfield code="a">经济学</subfield>

<subfield code="b">计算机</subfield>

<subfield code="c">10001</subfield>

<subfield code="d">2005-07-09</subfield>

</datafield>

<datafield tag="610" ind1="0" ind3=" ">

<subfield code="a">计算机</subfield>

<subfield code="a">笔记本</subfield>

</datafield>

</record>

</collection>')) ;

4、使用ExistsNode函数检查节点是否存在(对于xml文档内有的数据，返回1；对于没有的数据返回0)。

select existsnode(x.data,'/collection/record/datafield[@tag="209"]/subfield[@code="a"]') as a from test\_xmltype\_1087946 x;

返回结果为1：

a

---

1

(1 row)

select existsnode(x.data,'/collection/record/datafield[@tag="209"]/subfield[@code="f"]') as a from test\_xmltype\_1087946 x;

返回结果为0：

a

---

0

(1 row)

**GetStringVal**

**功能描述**

成员函数。 以字符串形式返回xml示例的值。如果 XML 文档大于 VARCHAR2 的最大大小（4000），则在运行时会引发错误。

**语法格式**

FUNCTION getStringVal(xml\_obj Xmltype

)

RETURN varchar2

**示例**

1、创建并切换至兼容模式为Oracle的数据库db\_oracle。

CREATE DATABASE db\_oracle dbcompatibility='A';

\c db\_oracle

2、创建测试表。

drop table if exists test\_xmltype\_1087944;

create table test\_xmltype\_1087944(

ID int,

name varchar2(262),

data xmltype

);

3、插入数据。

Insert INTO test\_xmltype\_1087944

VALUES (1,'test xml doc',xmlType.createXML('<?xml version="1.0" encoding="UTF-8" ?>

<collection xmlns="">

<record>

<leader>-----nam0-22-----^^^450-</leader>

<datafield tag="200" ind1="1" ind2=" ">

<subfield code="a">抗震救灾</subfield>

<subfield code="f">奥运会</subfield>

</datafield>

<datafield tag="209" ind1=" " ind2=" ">

<subfield code="a">经济学</subfield>

<subfield code="b">计算机</subfield>

<subfield code="c">10001</subfield>

<subfield code="d">2005-07-09</subfield>

</datafield>

<datafield tag="610" ind1="0" ind3=" ">

<subfield code="a">计算机</subfield>

<subfield code="a">笔记本</subfield>

</datafield>

</record>

</collection>')) ;

4、查看数据。

select \* from test\_xmltype\_1087944;

返回结果为：

id | name | data

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

1 | test xml doc | <collection xmlns=""> +

| | <record> +

| | <leader>-----nam0-22-----^^^450-</leader>+

| | <datafield tag="200" ind1="1" ind2=" "> +

| | <subfield code="a">抗震救灾</subfield> +

| | <subfield code="f">奥运会</subfield> +

| | </datafield> +

| | <datafield tag="209" ind1=" " ind2=" "> +

| | <subfield code="a">经济学</subfield> +

| | <subfield code="b">计算机</subfield> +

| | <subfield code="c">10001</subfield> +

| | <subfield code="d">2005-07-09</subfield> +

| | </datafield> +

| | <datafield tag="610" ind1="0" ind3=" "> +

| | <subfield code="a">计算机</subfield> +

| | <subfield code="a">笔记本</subfield> +

| | </datafield> +

| | </record> +

| | </collection>

(1 row)

5、使用GetStringVal函数以字符串形式返回xml示例的值。

select getStringVal(extract(x.data,'/collection/record/datafield/subfield')) a from test\_xmltype\_1087944 x;

返回结果为：

a

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

<subfield code="a">抗震救灾</subfield> +

<subfield code="f">奥运会</subfield> +

<subfield code="a">经济学</subfield> +

<subfield code="b">计算机</subfield> +

<subfield code="c">10001</subfield> +

<subfield code="d">2005-07-09</subfield> +

<subfield code="a">计算机</subfield> +

<subfield code="a">笔记本</subfield>

(1 row)

select getStringVal(extract(x.data,'/collection/record/datafield')) a from test\_xmltype\_1087944 x;

返回结果为：

a

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

<datafield tag="200" ind1="1" ind2=" "> +

<subfield code="a">抗震救灾</subfield> +

<subfield code="f">奥运会</subfield> +

</datafield> +

<datafield tag="209" ind1=" " ind2=" "> +

<subfield code="a">经济学</subfield> +

<subfield code="b">计算机</subfield> +

<subfield code="c">10001</subfield> +

<subfield code="d">2005-07-09</subfield>+

</datafield> +

<datafield tag="610" ind1="0" ind3=" "> +

<subfield code="a">计算机</subfield> +

<subfield code="a">笔记本</subfield> +

</datafield>

(1 row)





电话：010-82838118

地址：北京市海淀区学院路30号科大天工大厦B座6层

官网：www.vastdata.com.cn