**Vastbase G100 V2.2**

**(Build 12)**

**功能说明手册**



北京海量数据技术股份有限公司

【版权声明】

© 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. 开发者指南 4](#_Toc3079)

[1.1. SQL语法 4](#_Toc8319)

[1.1.1. ALTER TABLE PARTITION 4](#_Toc25612)

[1.1.2. CREATE TABLE SUBPARTITION 13](#_Toc12215)

[1.1.3. CREATE DATABASE 35](#_Toc32305)

[1.2. 系统表 45](#_Toc24840)

[1.2.1. PG\_COLLATION 45](#_Toc26018)

[1.2.2. PG\_DATABASE 46](#_Toc18594)

[1.2.3. PG\_EVENT\_TRIGGER 48](#_Toc9381)

[1.2.4. PG\_SUBSCRIPTION\_REL 49](#_Toc15174)

[1.3. 函数 49](#_Toc2650)

[1.3.1. IP白名单查询与更新函数 49](#_Toc27494)

[1.4. AI特性 52](#_Toc2010)

[1.4.1. AI4DB: 数据库自治运维 52](#_Toc13414)

[1.5. 外部数据封装器 61](#_Toc9981)

[1.5.1. Postgres\_FDW 61](#_Toc11445)

[2. 管理员指南 69](#_Toc28805)

[2.1. 数据库使用 69](#_Toc7639)

[2.1.1. 创建和管理分区表 69](#_Toc1165)

[2.1.2. 事件触发器 85](#_Toc14744)

[2.2. 安全性 93](#_Toc32609)

[2.2.1. 安全审计 93](#_Toc24062)

[2.2.2. 数据库启动及安装包完整性校验 100](#_Toc22060)

[2.2.3. 无过度损失的自动恢复 103](#_Toc1469)

[2.3. 备份与恢复 105](#_Toc29227)

[2.3.1. 逻辑备份与恢复 105](#_Toc12588)

[2.4. 日志参考 121](#_Toc17978)

[2.4.1. 审计日志 121](#_Toc13330)

[3. 兼容性手册 126](#_Toc14833)

[3.1. Oracle兼容性 126](#_Toc30413)

[3.1.1. 函数 126](#_Toc24254)

[3.1.2. 内置包 131](#_Toc28051)

[3.1.3. SQL语法 158](#_Toc18061)

[3.1.4. PL/SQL 199](#_Toc24290)

[3.1.5. 组件 201](#_Toc20743)

[3.2. MySQL兼容性 205](#_Toc6751)

[3.2.1. 数据类型 205](#_Toc11117)

[3.2.2. 类型转换 208](#_Toc2993)

[3.2.3. 操作符 210](#_Toc24723)

[3.2.4. SQL语法 213](#_Toc12562)

[3.2.5. 函数 258](#_Toc405)

[3.2.6. PL/SQL 278](#_Toc31674)

[3.2.7. 特性参数 281](#_Toc29999)

[3.3. PostgreSQL兼容性 282](#_Toc4175)

[3.4. SQL Server兼容性 282](#_Toc10982)

[3.4.1. 数据类型 282](#_Toc20462)

[3.4.2. 函数 305](#_Toc6240)

[3.4.3. 操作符 333](#_Toc5788)

[3.4.4. SQL语法 334](#_Toc23200)

[4. 工具参考 336](#_Toc16924)

[4.1. vb\_initdb 336](#_Toc25090)

[4.2. vb\_dump 341](#_Toc27763)

# 开发者指南

## SQL语法

### ALTER TABLE PARTITION

**功能描述**

修改表分区，包括增删分区、切割分区、合成分区、交换分区以及修改分区属性等。

**语法格式**

修改表分区主语法如下：

ALTER TABLE [ IF EXISTS ] { table\_name [\*] | ONLY table\_name | ONLY ( table\_name )}

action [, ... ];

其中action统指如下分区维护子语法。当存在多个分区维护子句时，保证了分区的连续性，无论这些子句的排序如何，Vastbase总会先执行DROP PARTITION再执行ADD PARTITION操作，最后顺序执行其它分区维护操作。

move\_clause |

exchange\_clause |

row\_clause |

merge\_clause |

modify\_clause |

split\_clause |

add\_clause |

drop\_clause |

truncate\_clause

* move\_clause子语法用于移动分区到新的表空间。

MOVE PARTITION { partion\_name | FOR ( partition\_value [, ...] ) } TABLESPACE tablespacename

* exchange\_clause子语法用于把普通表的数据迁移到指定的分区。

EXCHANGE PARTITION|SUBPARTITION { ( partition\_name ) | FOR ( partition\_value [, ...] ) }

WITH TABLE {[ ONLY ] ordinary\_table\_name | ordinary\_table\_name \* | ONLY ( ordinary\_table\_name )}

[ { WITH | WITHOUT } VALIDATION ] [ VERBOSE ]

进行交换的普通表和分区必须满足如下条件：

* 普通表和分区的列数目相同，对应列的信息严格一致，包括：列名、列的数据类型、列约束、列的Collation信息、列的存储参数、列的压缩信息等。
* 普通表和分区的表压缩信息严格一致。
* 普通表和分区的索引个数相同，且对应索引的信息严格一致。
* 普通表和分区的表约束个数相同，且对应表约束的信息严格一致。
* 普通表不可以是临时表，分区表只能是范围分区表，列表分区表，哈希分区表。
* 普通表和分区表上不可以有动态数据脱敏，行访问控制约束。
* 列表分区表，哈希分区表不能是列存储。
* List/Hash/Range类型分区表支持exchange\_clause。

【须知】

* 完成交换后，普通表和分区的数据被置换，同时普通表和分区的表空间信息被置换。此时，普通表和分区的统计信息变得不可靠，需要对普通表和分区重新执行analyze。
* 由于非分区键不能建立本地唯一索引，只能建立全局唯一索引，所以如果普通表含 有唯一索引时，会导致不能交换数据。
* 如果在普通表/分区表上进行了drop column操作，被删除的列依然物理存在，所以需要保证普通表和分区的被删除列也严格对齐才能交换成功。
* 支持增删列之后的表进行分区交换。
* row\_clause子语法用于设置分区表的行迁移开关。

{ ENABLE | DISABLE } ROW MOVEMENT

* merge\_clause子语法用于把多个分区合并成一个分区。

MERGE PARTITIONS { partition\_name } [, ...] INTO PARTITION partition\_name [ TABLESPACE tablespacename ] [ UPDATE GLOBAL INDEX ]

* modify\_clause子语法用于设置分区索引是否可用。

MODIFY PARTITION partition\_name { UNUSABLE LOCAL INDEXES | REBUILD UNUSABLE LOCAL INDEXES }

* split\_clause子语法用于把一个分区切割成多个分区。

SPLIT PARTITION { partition\_name | FOR ( partition\_value [, ...] ) } { split\_point\_clause | no\_split\_point\_clause } [ UPDATE GLOBAL INDEX ]

* 指定切割点split\_point\_clause的语法为。

AT ( partition\_value ) INTO ( PARTITION partition\_name [ TABLESPACE tablespacename ] , PARTITION partition\_name [ TABLESPACE tablespacename ] )

【须知】

* 列存分区表不支持切割分区。
* 切割点的大小要位于正在被切割的分区的分区键范围内，指定切割点的方式只能把一个分区切割成两个新分区。
* 不指定切割点no\_split\_point\_clause的语法为。

INTO { ( partition\_less\_than\_item [, ...] ) | ( partition\_start\_end\_item [, ...] ) }

【须知】

* 不指定切割点的方式，partition\_less\_than\_item指定的第一个新分区的分区键要大于正在被切割的分区的前一个分区（如果存在的话）的分区键，partition\_less\_than\_item指定的最后一个分区的分区键要等于正在被切割的分区的分区键大小。
* 不指定切割点的方式，partition\_start\_end\_item指定的第一个新分区的起始点（如果存在的话）必须等于正在被切割的分区的前一个分区（如果存在的话）的分区键，partition\_start\_end\_item指定的最后一个分区的终止点（如果存在的话）必须等于正在被切割的分区的分区键。
* partition\_less\_than\_item支持的分区键个数最多为4，而partition\_start\_end\_item仅支持1个分区键，其支持的数据类型参见PARTITION BY RANGE(parti...] (参见：开发者指南->SQL语法参考->SQL语法->CREATE-TABLE-PARTITION章节)。
* 在同一语句中partition\_less\_than\_item和partition\_start\_end\_item两者不可同时使用；不同split语句之间没有限制。
* 分区项partition\_less\_than\_item的语法为：

PARTITION partition\_name VALUES LESS THAN ( { partition\_value | MAXVALUE } [, ...] ) [ TABLESPACE tablespacename ]

* 分区项partition\_start\_end\_item的语法如下，其约束参见START END语法描述(参见：开发者指南->SQL语法参考->SQL语法->CREATE-TABLE-PARTITION章节)。

PARTITION partition\_name {{START(partition\_value) END (partition\_value) EVERY (interval\_value)} |{START(partition\_value) END ({partition\_value | MAXVALUE})} |{START(partition\_value)} |{END({partition\_value |MAXVALUE})}} [TABLESPACE tablespace\_name]

* add\_clause子语法用于为指定的分区表添加一个或多个分区。

ADD PARTITION ( partition\_col1\_name = partition\_col1\_value [, partition\_col2\_name = partition\_col2\_value ] [, ...] )

[ LOCATION 'location1' ]

[ PARTITION (partition\_colA\_name = partition\_colA\_value [, partition\_colB\_name = partition\_colB\_value ] [, ...] ) ]

[ LOCATION 'location2' ]

ADD {partition\_less\_than\_item | partition\_start\_end\_item| partition\_list\_item }

* 分区项partition\_list\_item的语法如下。

PARTITION partition\_name VALUES (list\_values\_clause) [ TABLESPACE tablespacename ]

【须知】

* partition\_list\_item仅支持的1个分区键，其支持的数据类型参见PARTITION BY LIST(partit...)(参见：开发者指南->SQL语法参考->SQL语法->CREATE-TABLE-PARTITION章节)。
* 间隔/哈希分区表不支持添加分区。
* drop\_clause子语法用于删除分区表中的指定分区。

DROP PARTITION { partition\_name | FOR ( partition\_value [, ...] ) } [ UPDATE GLOBAL INDEX ]

【须知】

哈希分区表不支持删除分区。

* truncate\_clause子语法用于清空分区表中的指定分区。

TRUNCATE PARTITION { partition\_name | FOR ( partition\_value [, ...] ) } [ UPDATE GLOBAL INDEX ]

* 修改表分区名称的语法。

ALTER TABLE [ IF EXISTS ] { table\_name [\*] | ONLY table\_name | ONLY ( table\_name )}

RENAME PARTITION { partion\_name | FOR ( partition\_value [, ...] ) } TO partition\_new\_name;

**参数说明**

* **table\_name**
* 分区表名。
* 取值范围：已存在的分区表名。
* **partition\_name**
* 分区名。
* 取值范围：已存在的分区名。
* **tablespacename**
* 指定分区要移动到哪个表空间。
* 取值范围：已存在的表空间名。
* **partition\_value**
* 分区键值。
* 通过PARTITION FOR ( partition\_value [, ...] )子句指定的这一组值，可以唯一确定一个分区。
* 取值范围：需要进行重命名的分区的分区键的取值范围。
* **UNUSABLE LOCAL INDEXES**
* 设置该分区上的所有索引不可用。
* **REBUILD UNUSABLE LOCAL INDEXES**
* 重建该分区上的所有索引。
* **ENABLE/DISABLE ROW MOVEMET**
* 行迁移开关。
* 如果进行UPDATE操作时，更新了元组在分区键上的值，造成了该元组所在分区发生变化，就会根据该开关给出报错信息，或者进行元组在分区间的转移。
* 取值范围：
* ENABLE：打开行迁移开关。
* DISABLE：关闭行迁移开关。
* 默认是打开状态。
* **ordinary\_table\_name**
* 进行迁移的普通表的名称。
* 取值范围：已存在的普通表名。
* **{ WITH | WITHOUT } VALIDATION**
* 在进行数据迁移时，是否检查普通表中的数据满足指定分区的分区键范围。
* 取值范围：
* WITH：对于普通表中的数据要检查是否满足分区的分区键范围，如果有数据不满足，则报错。
* WITHOUT：对于普通表中的数据不检查是否满足分区的分区键范围。
* 默认是WITH状态。
* 由于检查比较耗时，特别是当数据量很大的情况下更甚。所以在保证当前普通表中的数据满足分区的分区键范围时，可以加上WITHOUT来指明不进行检查。
* **VERBOSE**
* 在VALIDATION是WITH状态时，如果检查出普通表有不满足要交换分区的分区键范围的数据，那么把这些数据插入到正确的分区，如果路由不到任何分区，再报错。
* 【须知】
* 只有在VALIDATION是WITH状态时，才可以指定VERBOSE。
* **partition\_new\_name**
* 分区的新名称。
* 取值范围：字符串，要符合标识符的命名规范。

**注意事项**

* 添加分区的表空间不能是PG\_GLOBAL。
* 添加分区的名称不能与该分区表已有分区的名称相同。
* 添加分区的分区键值要和分区表的分区键的类型一致。
* 若添加RANGE分区，添加分区键值要大于分区表中最后一个范围分区的上边界。
* 若添加LIST分区，添加分区键值不能与现有分区键值重复。
* 不支持添加HASH分区。
* 如果目标分区表中已有分区数达到了最大值1048575，则不能继续添加分区。
* 当分区表只有一个分区时，不能删除该分区。
* 选择分区使用PARTITION FOR()，括号里指定值个数应该与定义分区时使用的列个数相同，并且一一对应。
* Value分区表不支持相应的Alter Partition操作。
* 列存分区表不支持切割分区。
* 间隔分区表不支持添加分区。
* 哈希分区表不支持切割分区，不支持合成分区，不支持添加和删除分区。
* 列表分区表不支持切割分区，不支持合成分区。
* 只有分区表的所有者或者被授予了分区表ALTER权限的用户有权限执行ALTER TABLE PARTITION命令，系统管理员默认拥有此权限。

**示例**

【说明】

此处示例为exchange\_clause子语法，其余请参考CREATE TABLE PARTITION的示例。(参见：开发者指南->SQL语法参考->SQL语法->CREATE-TABLE-PARTITION章节)。

**示例1：**分区交换。

1、创建分区表。

create table t\_range\_list

( id number,

partition\_key int,

subpartition\_key int,

col2 varchar2(10) )

partition by range(partition\_key)

subpartition by list(subpartition\_key)(

partition p1 values less than (100)

(subpartition sub\_1\_1 values (10),

subpartition sub\_1\_2 values (20)),

partition p2 values less than(200)

(subpartition sub\_2\_1 values (10),

subpartition sub\_2\_2 values (20)),

partition p3 values less than (300)

(subpartition sub\_3\_1 values (10),

subpartition sub\_3\_2 values (20)));

2、插入测试数据。

INSERT INTO t\_range\_list VALUES(1,50,10,'sub\_1\_1');

INSERT INTO t\_range\_list VALUES(2,150,20,'sub\_2\_2');

INSERT INTO t\_range\_list VALUES(3,250,10,'sub\_3\_1');

3、创建用于交换的普通表。

create table t\_exchange

( id number,

partition\_key int,

subpartition\_key int,

col2 varchar2(10) );

4、为普通表插入数据。

INSERT INTO t\_exchange VALUES(1,40,10,'sub\_1\_1');

5、将分区sub\_1\_1的数据与普通表的数据进行交换。

ALTER TABLE t\_range\_list exchange subpartition sub\_1\_1 with table t\_exchange;

6、查看交换后的普通表数据。

select \* from t\_exchange;

返回结果为：

id | partition\_key | subpartition\_key | col2

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

 1 |            50 |               10 | sub\_1\_1

(1 row)

7、查看交换后分区表的数据。

select \* from t\_range\_list;

返回结果为：

id | partition\_key | subpartition\_key | col2

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

 1 |            40 |               10 | sub\_1\_1

 2 |           150 |               20 | sub\_2\_2

 3 |           250 |               10 | sub\_3\_1

(3 rows)

**示例2：**进行增删列操作之后的分区交换。

1、创建range分区表。

CREATE TABLE e (

id INT NOT NULL,

fname VARCHAR(30),

lname VARCHAR(30) )

PARTITION BY RANGE (id)

( PARTITION p0 VALUES LESS THAN (50),

PARTITION p1 VALUES LESS THAN (100),

PARTITION p2 VALUES LESS THAN (150),

PARTITION p3 VALUES LESS THAN (MAXVALUE) );

2、向分区表中插入测试数据。

INSERT INTO e VALUES (1669, 'Jim', 'Smith'), (337, 'Mary', 'Jones'), (16, 'Frank', 'White'), (2005, 'Linda', 'Black');

3、创建普通表，比步骤1中创建的分区表少一列lname。

CREATE TABLE e2 (

id INT NOT NULL,

fname VARCHAR(30));

4、为普通表新增一列lname，但数据类型和分区表中的lname列不同。

alter table e2 add column lname text;

5、交换分区。

alter table e exchange partition p3 with table e2;

返回结果如下，交换失败（同名列lname在普通表和分区表中数据类型不同）。

ERROR: column type or size mismatch in ALTER TABLE EXCHANGE PARTITION

6、删除普通表中的列lname。

alter table e2 drop lname;

7、为普通表再次新增列lname，数据类型与分区表中的列lname相同。

alter table e2 add column lname VARCHAR(30);

8、执行分区交换，交换成功。

alter table e exchange partition p3 with table e2;

9、查看此时的分区表e和普通表e2的数据。

select \* from e;

select \* from e2;

返回结果为：

id | fname | lname

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

16 | Frank | White

(1 row)

​

id  | fname | lname

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

1669 | Jim   | Smith

 337 | Mary  | Jones

2005 | Linda | Black

(3 rows)

**相关链接**

CREATE TABLE PARTITION (参见：开发者指南->SQL语法参考->SQL语法->CREATE TABLE PARTITION章节)，DROP TABLE(参见：开发者指南->SQL语法参考->SQL语法->DROP TABLE章节)

### CREATE TABLE SUBPARTITION

**功能描述**

创建二级分区表。分区表是把逻辑上的一张表根据某种方案分成几张物理块进行存储，这张逻辑上的表称之为分区表，物理块称之为分区。分区表是一张逻辑表，不存储数据，数据实际是存储在分区上的。对于二级分区表，顶层节点表和一级分区都是逻辑表，不存储数据，只有二级分区（叶子节点）存储数据。

二级分区功能可参见管理员指南->数据库使用->创建和管理分区表：[支持两级分区](../%E7%AE%A1%E7%90%86%E5%91%98%E6%8C%87%E5%8D%97/%E6%94%AF%E6%8C%81%E4%B8%A4%E7%BA%A7%E5%88%86%E5%8C%BA.md)。二级分区表的分区方案是由两个一级分区的分区方案组合而来的，一级分区的分区方案详见《Vastbase G100 V2.2 开发者指南 》->SQL语法参考->SQL语法->[CREATE TABLE PARTITION](CREATE-TABLE-PARTITION.md)。

常见的二级分区表组合方案有Range-Range分区、Range-List分区、Range-Hash分区、List-Range分区、List-List分区、List-Hash分区、Hash-Range分区、Hash-List分区、Hash-Hash分区等。目前二级分区仅支持行存表。

**语法格式**

CREATE TABLE [ IF NOT EXISTS ] subpartition\_table\_name

(

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

| table\_constraint

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

)

[ WITH ( {storage\_parameter = value} [, ... ] ) ]

[ COMPRESS | NOCOMPRESS ]

[ TABLESPACE tablespace\_name ]

​

PARTITION BY {

       {RANGE (partition\_key) [ INTERVAL ('interval\_expr') [ STORE IN (tablespace\_name [, ... ] ) ] ] ( partition\_less\_than\_item [, ... ] )} |

       {RANGE (partition\_key) [ INTERVAL ('interval\_expr') [ STORE IN (tablespace\_name [, ... ] ) ] ] ( partition\_start\_end\_item [, ... ] )} |

       {LIST | HASH (partition\_key) (PARTITION partition\_name [VALUES (list\_values\_clause)] opt\_table\_space )}

   }

(

 PARTITION partition\_name1 [ VALUES LESS THAN (val1) | VALUES (val1[, …]) ] [ TABLESPACE tablespace ]

 (

      { SUBPARTITION subpartition\_name1 [ VALUES LESS THAN (val1\_1) | VALUES (val1\_1[, …])]  [ TABLESPACE tablespace ] } [, ...]

 )[, ...]

)

[ { ENABLE | DISABLE } ROW MOVEMENT ];

列约束column\_constraint：

[ CONSTRAINT constraint\_name ]

   { NOT NULL |

     NULL |

     CHECK ( expression ) |

     DEFAULT default\_e xpr |

     GENERATED ALWAYS AS ( generation\_expr ) STORED |

     UNIQUE index\_parameters |

     PRIMARY KEY index\_parameters |

     REFERENCES reftable [ ( refcolumn ) ] [ MATCH FULL | MATCH PARTIAL | MATCH SIMPLE ]

           [ ON DELETE action ] [ ON UPDATE action ] }

[ DEFERRABLE | NOT DEFERRABLE | INITIALLY DEFERRED | INITIALLY IMMEDIATE ]

表约束table\_constraint：

[ CONSTRAINT constraint\_name ]

   { CHECK ( expression ) |

     UNIQUE ( column\_name [, ... ] ) index\_parameters |

     PRIMARY KEY ( column\_name [, ... ] ) index\_parameters |

     FOREIGN KEY ( column\_name [, ... ] ) REFERENCES reftable [ ( refcolumn [, ... ] ) ]

     [ MATCH FULL | MATCH PARTIAL | MATCH SIMPLE ] [ ON DELETE action ] [ ON UPDATE action ] }

[ DEFERRABLE | NOT DEFERRABLE | INITIALLY DEFERRED | INITIALLY IMMEDIATE ]

like选项like\_option：

{ INCLUDING | EXCLUDING } { DEFAULTS | GENERATED | CONSTRAINTS | INDEXES | STORAGE | COMMENTS | RELOPTIONS| ALL }

索引存储参数index\_parameters：

[ WITH ( {storage\_parameter = value} [, ... ] ) ]

[ USING INDEX TABLESPACE tablespace\_name ]

**参数说明**

* **IF NOT EXISTS**

如果已经存在相同名称的表，不会抛出一个错误，而会发出一个通知，告知表关系已存在。

* **subpartition\_table\_name**

二级分区表的名称。

取值范围：字符串，要符合标识符的命名规范。

* **column\_name**

新表中要创建的字段名。

取值范围：字符串，要符合标识符的命名规范。

* **data\_type**

字段的数据类型。

* **COLLATE collation**

COLLATE子句指定列的排序规则（该列必须是可排列的数据类型）。如果没有指定，则使用默认的排序规则。排序规则可以使用select \* from pg\_collation;命令从pg\_collation系统表中查询，默认的排序规则为查询结果中以default开始的行。

* **CONSTRAINT constraint\_name**

列约束或表约束的名称。可选的约束子句用于声明约束，新行或者更新的行必须满足这些约束才能成功插入或更新。 定义约束有两种方法：

* 列约束：作为一个列定义的一部分，仅影响该列。
* 表约束：不和某个列绑在一起，可以作用于多个列。
* **LIKE source\_table [ like\_option ... ]**

二级分区表暂不支持该功能。

* **WITH ( storage\_parameter [= value] [, ... ] )**

这个子句为表或索引指定一个可选的存储参数。参数的详细描述如下所示：

* **FILLFACTOR**

一个表的填充因子（fillfactor）是一个介于10和100之间的百分数。100（完全填充）是默认值。如果指定了较小的填充因子，INSERT操作仅按照填充因子指定的百分率填充表页。每个页上的剩余空间将用于在该页上更新行，这就使得UPDATE有机会在同一页上放置同一条记录的新版本，这比把新版本放置在其他页上更有效。对于一个从不更新的表将填充因子设为100是最佳选择，但是对于频繁更新的表，选择较小的填充因子则更加合适。该参数对于列存表没有意义。

取值范围：10~100

* **ORIENTATION**

决定了表的数据的存储方式。

【说明】

orientation不支持修改。

取值范围：

* COLUMN：表的数据将以列式存储。
* ROW（缺省值）：表的数据将以行式存储。
* **COMPRESSION**

列存表的有效值为LOW/MIDDLE/HIGH/YES/NO，压缩级别依次升高，默认值为LOW。

行存表不支持压缩。

* **MAX\_BATCHROW**

指定了在数据加载过程中一个存储单元可以容纳记录的最大数目。该参数只对列存表有效。

取值范围：10000~60000，默认60000。

* **PARTIAL\_CLUSTER\_ROWS**

指定了在数据加载过程中进行将局部聚簇存储的记录数目。该参数只对列存表有效。

取值范围：大于等于MAX\_BATCHROW，建议取值为MAX\_BATCHROW的整数倍数。

* **DELTAROW\_THRESHOLD**

预留参数。该参数只对列存表有效。

取值范围：0～9999

* **COMPRESS / NOCOMPRESS**

创建一个新表时，需要在创建表语句中指定关键字COMPRESS，这样，当对该表进行批量插入时就会触发压缩特性。该特性会在页范围内扫描所有元组数据，生成字典、压缩元组数据并进行存储。指定关键字NOCOMPRESS则不对表进行压缩。行存表不支持压缩。

缺省值为NOCOMPRESS，即不对元组数据进行压缩。

* **TABLESPACE tablespace\_name**

指定新表将要在tablespace\_name表空间内创建。如果没有声明，将使用默认表空间。

* **PARTITION BY {RANGE | LIST | HASH} (partition\_key)**

对于partition\_key，分区策略的分区键仅支持1列。

分区键支持的数据类型和一级分区表约束保持一致。

* **PARTITION BY RANGE(partition\_key)**

创建范围分区。partition\_key为分区键的名称。

对于从句是VALUES LESS THAN的语法格式：

【须知】

对于从句是VALUE LESS THAN的语法格式，范围分区策略的分区键最多支持4列。

该情形下，分区键支持的数据类型为：SMALLINT、INTEGER、BIGINT、DECIMAL、NUMERIC、REAL、DOUBLE PRECISION、CHARACTER VARYING(n)、VARCHAR(n)、CHARACTER(n)、CHAR(n)、CHARACTER、CHAR、TEXT、NVARCHAR、NVARCHAR2、NAME、TIMESTAMP[(p)] [WITHOUT TIME ZONE]、TIMESTAMP[(p)] [WITH TIME ZONE]、DATE。

对于从句是START END的语法格式：

【须知】

对于从句是START END的语法格式，范围分区策略的分区键仅支持1列。

该情形下，分区键支持的数据类型为：SMALLINT、INTEGER、BIGINT、DECIMAL、NUMERIC、REAL、DOUBLE PRECISION、TIMESTAMP[(p)] [WITHOUT TIME ZONE]、TIMESTAMP[(p)] [WITH TIME ZONE]、DATE。

对于指定了INTERVAL子句的语法格式：

【须知】

对于指定了INTERVAL子句的语法格式，范围分区策略的分区键仅支持1列。

该情形下，分区键支持的数据类型为：TIMESTAMP[(p)] [WITHOUT TIME ZONE]、TIMESTAMP[(p)] [WITH TIME ZONE]、DATE。

* **PARTITION partition\_name VALUES LESS THAN ( { partition\_value | MAXVALUE } )**

指定各分区的信息。partition\_name为范围分区的名称。partition\_value为范围分区的上边界，取值依赖于partition\_key的类型。MAXVALUE表示分区的上边界，它通常用于设置最后一个范围分区的上边界。

【须知】

每个分区都需要指定一个上边界。

分区上边界的类型应当和分区键的类型一致。

分区列表是按照分区上边界升序排列的，值较小的分区位于值较大的分区之前。

* **PARTITION partition\_name {START (partition\_value) END (partition\_value) EVERY (interval\_value)} | {START (partition\_value) END (partition\_value|MAXVALUE)} | {START(partition\_value)} | {END (partition\_value | MAXVALUE)}**

指定各分区的信息，各参数意义如下：

* partition\_name：范围分区的名称或名称前缀，除以下情形外（假定其中的partition\_name是p1），均为分区的名称。
* 若该定义是START+END+EVERY从句，则语义上定义的分区的名称依次为p1\_1, p1\_2, …。例如对于定义“PARTITION p1 START(1) END(4) EVERY(1)”，则生成的分区是：[1, 2), [2, 3) 和 [3, 4)，名称依次为p1\_1, p1\_2和p1\_3，即此处的p1是名称前缀。
* 若该定义是第一个分区定义，且该定义有START值，则范围（MINVALUE, START）将自动作为第一个实际分区，其名称为p1\_0，然后该定义语义描述的分区名称依次为p1\_1, p1\_2, …。例如对于完整定义“PARTITION p1 START(1), PARTITION p2 START(2)”，则生成的分区是：(MINVALUE, 1), [1, 2) 和 [2, MAXVALUE)，其名称依次为p1\_0, p1\_1和p2，即此处p1是名称前缀，p2是分区名称。这里MINVALUE表示最小值。
* partition\_value：范围分区的端点值（起始或终点），取值依赖于partition\_key的类型，不可是MAXVALUE。
* interval\_value：对[START，END) 表示的范围进行切分，interval\_value是指定切分后每个分区的宽度，不可是MAXVALUE；如果（END-START）值不能整除以EVERY值，则仅最后一个分区的宽度小于EVERY值。
* MAXVALUE：表示最大值，它通常用于设置最后一个范围分区的上边界。

【须知】

1、在创建分区表若第一个分区定义含START值，则范围（MINVALUE，START）将自动作为实际的第一个分区。

2、START END语法需要遵循以下限制：

* 每个partition\_start\_end\_item中的START值（如果有的话，下同）必须小于其END值。
* 相邻的两个partition\_start\_end\_item，第一个的END值必须等于第二个的START值；
* 每个partition\_start\_end\_item中的EVERY值必须是正向递增的，且必须小于（END-START）值；
* 每个分区包含起始值，不包含终点值，即形如：[起始值，终点值)，起始值是MINVALUE时则不包含；
* 一个partition\_start\_end\_item创建的每个分区所属的TABLESPACE一样；
* partition\_name作为分区名称前缀时，其长度不要超过57字节，超过时自动截断；
* 在创建、修改分区表时请注意分区表的分区总数不可超过最大限制（1048575）；

3、 在创建分区表时START END与LESS THAN语法不可混合使用。

4、 即使创建分区表时使用START END语法，备份（vb\_dump）出的SQL语句也是VALUES LESS THAN语法格式。

* **INTERVAL ('interval\_expr') [ STORE IN (tablespace\_name [, … ] ) ]**

间隔分区定义信息。

* interval\_expr：自动创建分区的间隔，例如：1 day、1 month。
* STORE IN (tablespace\_name [, … ] )：指定存放自动创建分区的表空间列表，如果有指定，则自动创建的分区从表空间列表中循环选择使用，否则使用分区表默认的表空间。​

【须知】

列存表不支持间隔分区。

* **PARTITION BY LIST(partition\_key)**

创建列表分区。partition\_key为分区键的名称。

* 对于partition\_key，列表分区策略的分区键仅支持1列。
* 对于从句是VALUES (list\_values\_clause)的语法格式，list\_values\_clause中包含了对应分区存在的键值，每个分区的键值数量不超过127个。

分区键支持的数据类型为：INT1、INT2、INT4、INT8、NUMERIC、VARCHAR(n)、CHAR、BPCHAR、NVARCHAR、NVARCHAR2、TIMESTAMP[(p)] [WITHOUT TIME ZONE]、TIMESTAMP[(p)] [WITH TIME ZONE]、DATE。分区个数不能超过 1048575 个。

* **PARTITION BY HASH(partition\_key)**

创建哈希分区。partition\_key为分区键的名称。

对于partition\_key，哈希分区策略的分区键仅支持1列。

分区键支持的数据类型为：INT1、INT2、INT4、INT8、NUMERIC、VARCHAR(n)、CHAR、BPCHAR、TEXT、NVARCHAR、NVARCHAR2、TIMESTAMP[(p)] [WITHOUT TIME ZONE]、TIMESTAMP[(p)] [WITH TIME ZONE]、DATE。分区个数不能超过1048575 个。

* **{ ENABLE | DISABLE } ROW MOVEMENT**

行迁移开关。

如果进行UPDATE操作时，更新了元组在分区键上的值，造成了该元组所在分区发生变化，就会根据该开关给出报错信息，或者进行元组在分区间的转移。

取值范围：

* ENABLE（缺省值）：行迁移开关打开。
* DISABLE：行迁移开关关闭。
* **NOT NULL**

字段值不允许为NULL。ENABLE用于语法兼容，可省略。

* **NULL**

字段值允许NULL ，这是缺省。

这个子句只是为和非标准SQL数据库兼容。不建议使用。

* **CHECK (condition) [ NO INHERIT ]**

CHECK约束声明一个布尔表达式，每次要插入的新行或者要更新的行的新值必须使表达式结果为真或未知才能成功，否则会抛出一个异常并且不会修改数据库。

声明为字段约束的检查约束应该只引用该字段的数值，而在表约束里出现的表达式可以引用多个字段。

用NO INHERIT标记的约束将不会传递到子表中去。

ENABLE用于语法兼容，可省略。

* **DEFAULT default\_expr**

DEFAULT子句给字段指定缺省值。该数值可以是任何不含变量的表达式(不允许使用子查询和对本表中的其他字段的交叉引用)。缺省表达式的数据类型必须和字段类型匹配。

缺省表达式将被用于任何未声明该字段数值的插入操作。如果没有指定缺省值则缺省值为NULL 。

* **GENERATED ALWAYS AS ( generation\_expr ) STORED**

该子句将字段创建为生成列，生成列的值在写入（插入或更新）数据时由generation\_expr计算得到，STORED表示像普通列一样存储生成列的值。

【须知】

* 生成表达式不能以任何方式引用当前行以外的其他数据。生成表达式不能引用其他生成列，不能引用系统列。生成表达式不能返回结果集，不能使用子查询，不能使用聚集函数，不能使用窗口函数。生成表达式调用的函数只能是不可变（IMMUTABLE）函数。
* 不能为生成列指定默认值。
* 生成列不能作为分区键的一部分。
* 生成列不能和ON UPDATE约束字句的CASCADE、SET NULL、SET DEFAULT动作同时指定。生成列不能和ON DELETE约束字句的SET NULL、SET DEFAULT动作同时指定。
* 修改和删除生成列的方法和普通列相同。删除生成列依赖的普通列，生成列被自动删除。不能改变生成列所依赖的列的类型。
* 生成列不能被直接写入。在INSERT或UPDATE命令中, 不能为生成列指定值, 但是可以指定关键字DEFAULT。
* 生成列的权限控制和普通列一样。
* 不能为生成列指定默认值。
* 生成列不能作为分区键的一部分。
* 生成列不能和ON UPDATE约束字句的CASCADE、SET NULL、SET DEFAULT动作同时指定。生成列不能和ON DELETE约束字句的SET NULL、SET DEFAULT动作同时指定。
* 修改和删除生成列的方法和普通列相同。删除生成列依赖的普通列，生成列被自动删除。不能改变生成列所依赖的列的类型。
* 生成列不能被直接写入。在INSERT或UPDATE命令中, 不能为生成列指定值, 但是可以指定关键字DEFAULT。
* 生成列的权限控制和普通列一样。
* 列存表、内存表MOT不支持生成列。外表中仅postgres\_fdw支持生成列。
* **UNIQUE index\_parameters**

**UNIQUE ( column\_name [, ... ] ) index\_parameters**

UNIQUE约束表示表里的一个字段或多个字段的组合必须在全表范围内唯一。

对于唯一约束，NULL被认为是互不相等的。

* **PRIMARY KEY index\_parameters**

**PRIMARY KEY ( column\_name [, ... ] ) index\_parameters**

主键约束声明表中的一个或者多个字段只能包含唯一的非NULL值。

一个表只能声明一个主键。

* **DEFERRABLE | NOT DEFERRABLE**

这两个关键字设置该约束是否可推迟。一个不可推迟的约束将在每条命令之后马上检查。可推迟约束可以推迟到事务结尾使用SET CONSTRAINTS命令检查。缺省是NOT DEFERRABLE。目前，UNIQUE约束、主键约束、外键约束可以接受这个子句。所有其他约束类型都是不可推迟的。

* **INITIALLY IMMEDIATE | INITIALLY DEFERRED**

如果约束是可推迟的，则这个子句声明检查约束的缺省时间。

* 如果约束是INITIALLY IMMEDIATE（缺省），则在每条语句执行之后就立即检查它；
* 如果约束是INITIALLY DEFERRED ，则只有在事务结尾才检查它。

约束检查的时间可以用SET CONSTRAINTS命令修改。

* **USING INDEX TABLESPACE tablespace\_name**

为UNIQUE或PRIMARY KEY约束相关的索引声明一个表空间。如果没有提供这个子句，这个索引将在default\_tablespace中创建，如果default\_tablespace为空，将使用数据库的缺省表空间。

**注意事项**

* 二级分区表有两个分区键，每个分区键只能支持1列。
* 唯一约束和主键约束的约束键包含所有分区键将为约束创建LOCAL索引，否则创建GLOBAL索引。
* 二级分区表的二级分区（叶子节点）个数不能超过1048575个，一级分区无限制，但一级分区下面至少有一个二级分区。
* 二级分区表只支持行存，不支持列存、段页式、hashbucket。
* 不支持Upsert、Merge into。
* 指定分区查询时，如select \* from tablename partition/subpartition (partition\_name);，关键字partition和subpartition注意不要写错。如果写错，查询不会报错，这时查询会变为对表起别名进行查询。
* 不支持对二级分区 subpartition for (values)查询。如select \* from tablename subpartition for (values);。
* 对于二级分区表PARTITION FOR (values)语法，values只能是常量。
* 对于分区表PARTITION/SUBPARTITION FOR (values)语法，values在需要数据类型转换时，建议使用强制类型转换，以防隐式类型转换结果与预期不符。
* 目前Hash分区是按倒序排列的，即通过哈希和取余计算后得到的分区下标与创建顺序相反，同样EXPLAIN计划显示的Selected Partitions的序号排序也与创建顺序相反。
* 不支持密态数据库、账本数据库和行级访问控制。
* 指定分区语句目前不能走全局索引扫描。

**示例**

**示例1：**创建list\_list组合类型的二级分区表并进行truncate（清空）和drop（删除）分区的操作。

1、创建表list\_list（包含一级分区和二级分区）。

CREATE TABLE list\_list

   (

      month\_code VARCHAR2 ( 30 ) NOT NULL ,

      dept\_code VARCHAR2 ( 30 ) NOT NULL ,

      user\_no   VARCHAR2 ( 30 ) NOT NULL ,

      sales\_amt  int

   )

   PARTITION BY LIST (month\_code) SUBPARTITION BY LIST (dept\_code)

   (

     PARTITION p\_201901 VALUES ( '201902' )

     (

      SUBPARTITION p\_201901\_a VALUES ( '1' ),

      SUBPARTITION p\_201901\_b VALUES ( default )

     ),

     PARTITION p\_201902 VALUES ( '201903' )

     (

      SUBPARTITION p\_201902\_a VALUES ( '1' ),

      SUBPARTITION p\_201902\_b VALUES ( '2' )

     )

   );

2、插入测试数据。

insert into list\_list values('201902', '1', '1', 1);

insert into list\_list values('201902', '2', '1', 1);

insert into list\_list values('201902', '1', '1', 1);

insert into list\_list values('201903', '2', '1', 1);

insert into list\_list values('201903', '1', '1', 1);

insert into list\_list values('201903', '2', '1', 1);

3、查看表。

select \* from list\_list;

返回结果为：

month\_code | dept\_code | user\_no | sales\_amt

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

201902     | 1         | 1       |         1

201902     | 1         | 1       |         1

201902     | 2         | 1       |         1

201903     | 1         | 1       |         1

201903     | 2         | 1       |         1

201903     | 2         | 1       |         1

(6 rows)

4、查看其中的一个分区p\_201901。

select \* from list\_list partition (p\_201901);

返回结果为：

month\_code | dept\_code | user\_no | sales\_amt

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

201902     | 1         | 1       |         1

201902     | 1         | 1       |         1

201902     | 2         | 1       |         1

(3 rows)

5、清空其中的一个分区p\_201901。

alter table list\_list truncate partition p\_201901;

6、再次查看这个分区。

select \* from list\_list partition (p\_201901);

返回结果为空，表示该分区已经被清空。

month\_code | dept\_code | user\_no | sales\_amt

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

(0 rows)

7、删除一级分区p\_201901。

alter table list\_list drop partition p\_201901;

8、查看表list\_list。

select \* from list\_list;

返回结果如下，分区p\_201901已被删除。

month\_code | dept\_code | user\_no | sales\_amt

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

201903     | 1         | 1       |         1

201903     | 2         | 1       |         1

201903     | 2         | 1       |         1

(3 rows)

**示例2：**创建一个list\_range分区组合的分区表，并给分区表新增一级分区和二级分区。

1、创建表list\_range。（一级分区为list分区，二级分区为range分区。）

create table list\_range

(     id number not null,

      partition\_key   int,

      subpartition\_key   int,

      col2   varchar2(10)

)

partition by list(partition\_key)

subpartition by range(subpartition\_key)

(

   partition t\_partition\_01 values (100)

   (subpartition sub\_1\_1 values less than (10),

    subpartition sub\_1\_2 values less than (20)

    ),

   partition t\_partition\_02 values (200)

   (subpartition sub\_2\_1 values less than (10),

    subpartition sub\_2\_2 values less than (20)

    )

);

2、新增分区。

alter table list\_range add partition t\_partition\_03 values (300)

( subpartition sub\_3\_1 values less than (10),

subpartition sub\_3\_2 values less than (20)

);

3、在系统表[pg\_partition](PG_PARTITION.md)中查询分区信息。

\x  --启用列式方式显示结果

select \* from pg\_partition where relname='t\_partition\_03';

返回结果如下，查看到其中parentid为19029。

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

relname            | list\_range

parttype           | r

parentid           | 19029

rangenum           | 0

intervalnum        | 0

partstrategy       | l

relfilenode        | 0

reltablespace      | 0

relpages           | 0

reltuples          | 0

relallvisible      | 0

reltoastrelid      | 0

reltoastidxid      | 0

indextblid         | 0

indisusable        | t

reldeltarelid      | 0

reldeltaidx        | 0

relcudescrelid     | 0

relcudescidx       | 0

relfrozenxid       | 0

intspnum           |

partkey            | 2

intervaltablespace |

interval           |

boundaries         |

transit            |

reloptions         | {orientation=row,compression=no,fillfactor=80,wait\_clean\_gpi=n}

relfrozenxid64     | 0

relminmxid         | 0

4、根据上一步查询到的parentid，查询对应分区信息。

select \* from pg\_partition where parentid='19029';

返回结果如下，一级分区和二级分区添加成功。

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

relname            | t\_partition\_03

parttype           | p

parentid           | 19029

rangenum           | 0

intervalnum        | 0

partstrategy       | l

relfilenode        | 0

reltablespace      | 0

relpages           | 0

reltuples          | 0

relallvisible      | 0

reltoastrelid      | 0

reltoastidxid      | 0

indextblid         | 0

indisusable        | t

reldeltarelid      | 0

reldeltaidx        | 0

relcudescrelid     | 0

relcudescidx       | 0

relfrozenxid       | 18570

intspnum           |

partkey            | 3

intervaltablespace |

interval           |

boundaries         | {300}

transit            |

reloptions         | {orientation=row,compression=no,fillfactor=80}

relfrozenxid64     | 18570

relminmxid         | 2

-[ RECORD 2 ]------+----------------------------------------------------------------

relname            | list\_range

parttype           | r

parentid           | 19029

rangenum           | 0

intervalnum        | 0

partstrategy       | l

relfilenode        | 0

reltablespace      | 0

relpages           | 0

reltuples          | 0

relallvisible      | 0

reltoastrelid      | 0

reltoastidxid      | 0

indextblid         | 0

indisusable        | t

reldeltarelid      | 0

reldeltaidx        | 0

relcudescrelid     | 0

relcudescidx       | 0

relfrozenxid       | 0

intspnum           |

partkey            | 2

intervaltablespace |

interval           |

boundaries         |

transit            |

reloptions         | {orientation=row,compression=no,fillfactor=80,wait\_clean\_gpi=n}

relfrozenxid64     | 0

relminmxid         | 0

-[ RECORD 3 ]------+----------------------------------------------------------------

relname            | t\_partition\_01

parttype           | p

parentid           | 19029

rangenum           | 0

intervalnum        | 0

partstrategy       | l

relfilenode        | 0

reltablespace      | 0

relpages           | 0

reltuples          | 0

relallvisible      | 0

reltoastrelid      | 0

reltoastidxid      | 0

indextblid         | 0

indisusable        | t

reldeltarelid      | 0

reldeltaidx        | 0

relcudescrelid     | 0

relcudescidx       | 0

relfrozenxid       | 18566

intspnum           |

partkey            | 3

intervaltablespace |

interval           |

boundaries         | {100}

transit            |

reloptions         | {orientation=row,compression=no,fillfactor=80}

relfrozenxid64     | 18566

relminmxid         | 2

-[ RECORD 4 ]------+----------------------------------------------------------------

relname            | t\_partition\_02

parttype           | p

parentid           | 19029

rangenum           | 0

intervalnum        | 0

partstrategy       | l

relfilenode        | 0

reltablespace      | 0

relpages           | 0

reltuples          | 0

relallvisible      | 0

reltoastrelid      | 0

reltoastidxid      | 0

indextblid         | 0

indisusable        | t

reldeltarelid      | 0

reldeltaidx        | 0

relcudescrelid     | 0

relcudescidx       | 0

relfrozenxid       | 18566

intspnum           |

partkey            | 3

intervaltablespace |

interval           |

boundaries         | {200}

transit            |

reloptions         | {orientation=row,compression=no,fillfactor=80}

relfrozenxid64     | 18566

relminmxid         | 2

**示例3：**对二级分区表进行split（分区切割）操作。

1、创建分区表。

CREATE TABLE list\_list

(

  month\_code VARCHAR2 ( 30 ) NOT NULL ,

  dept\_code VARCHAR2 ( 30 ) NOT NULL ,

  user\_no   VARCHAR2 ( 30 ) NOT NULL ,

  sales\_amt  int

)

PARTITION BY LIST (month\_code) SUBPARTITION BY LIST (dept\_code)

(

 PARTITION p\_201901 VALUES ( '201902' )

 (

  SUBPARTITION p\_201901\_a VALUES ( '1' ),

  SUBPARTITION p\_201901\_b VALUES ( default )

 ),

 PARTITION p\_201902 VALUES ( '201903' )

 (

  SUBPARTITION p\_201902\_a VALUES ( '1' ),

  SUBPARTITION p\_201902\_b VALUES ( default )

 )

);

2、插入测试数据。

insert into list\_list values('201902', '1', '1', 1);

insert into list\_list values('201902', '2', '1', 1);

insert into list\_list values('201902', '1', '1', 1);

insert into list\_list values('201903', '2', '1', 1);

insert into list\_list values('201903', '1', '1', 1);

insert into list\_list values('201903', '2', '1', 1);

3、查看表list\_list。

select \* from list\_list;

返回结果如下：

month\_code | dept\_code | user\_no | sales\_amt

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

201903     | 2         | 1       |         1

201903     | 2         | 1       |         1

201903     | 1         | 1       |         1

201902     | 2         | 1       |         1

201902     | 1         | 1       |         1

201902     | 1         | 1       |         1

(6 rows)

4、查看二级分区p\_201901\_a。

select \* from list\_list subpartition (p\_201901\_a);

返回结果为：

month\_code | dept\_code | user\_no | sales\_amt

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

201902     | 1         | 1       |         1

201902     | 1         | 1       |         1

(2 rows)

5、查看二级分区p\_201901\_b。

select \* from list\_list subpartition (p\_201901\_b);

返回结果为：

month\_code | dept\_code | user\_no | sales\_amt

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

201902     | 2         | 1       |         1

(1 row)

6、将分区p\_201901\_b分割为p\_201901\_b和p\_201901\_c。

alter table list\_list split subpartition p\_201901\_b values (2) into

(

subpartition p\_201901\_b,

subpartition p\_201901\_c

);

7、查看分区p\_201901\_a、p\_201901\_b、p\_201901\_c。

select \* from list\_list subpartition (p\_201901\_a);

select \* from list\_list subpartition (p\_201901\_b);

select \* from list\_list subpartition (p\_201901\_c);

返回结果为：

month\_code | dept\_code | user\_no | sales\_amt

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

201902     | 1         | 1       |         1

201902     | 1         | 1       |         1

(2 rows)

​

month\_code | dept\_code | user\_no | sales\_amt

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

201902     | 2         | 1       |         1

(1 row)

​

month\_code | dept\_code | user\_no | sales\_amt

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

(0 rows)

### CREATE DATABASE

**功能描述**

CREATE DATABASE语法用于创建一个新的数据库。缺省情况下新数据库将通过复制标准系统数据库template0来创建，Vastbase G100 V2.2 Build12版本开始支持创建数据库时自定义指定模板。

**注意事项**

* 只有拥有CREATEDB权限的用户才可以创建新数据库，系统管理员默认拥有此权限。
* 不能在事务块中执行创建数据库语句。
* 在创建数据库过程中，出现类似“Permission denied”的错误提示，可能是由于文件系统上数据目录的权限不足。出现类似“No space left on device”的错误提示，可能是由于磁盘满引起的。
* 事务中不支持创建database。
* 当新建数据库Encoding、LC-Collate 或LC\_Ctype与模板数据库（SQL\_ASCII）不匹配（为'GBK' /'UTF8'/'LATIN1'）时，必须指定template [=] template0。

**语法格式**

CREATE DATABASE database\_name

[ [ WITH ] { [ OWNER [=] user\_name ] |

[ TEMPLATE [=] template ] |

[ ENCODING [=] encoding ] |

[ LC\_COLLATE [=] lc\_collate ] |

[ LC\_CTYPE [=] lc\_ctype ] |

[ DBCOMPATIBILITY [=] compatibilty\_type ] |

[ TABLESPACE [=] tablespace\_name ] |

[ CONNECTION LIMIT [=] connlimit ]}[...] ];

**参数说明**

* **database\_name**

数据库名称。

取值范围：字符串，要符合标识符的命名规范。

【须知】

Vastbase G100 V2.2 Build11版本开始支持在创建数据库的时候使用中文的数据库名。

* **OWNER [ = ] user\_name**

数据库所有者。缺省时，新数据库的所有者是当前用户。

取值范围：已存在的用户名。

* **TEMPLATE [ = ] template**

模板名。即从哪个模板创建新数据库。Vastbase采用从模板数据库复制的方式来创建新的数据库。初始时，Vastbase包含两个模板数据库template0、template1，以及一个默认的用户数据库postgres。

取值范围：template0，用户自定义数据库。

* **ENCODING [ = ] encoding**

指定数据库使用的字符编码，可以是字符串（如'SQL\_ASCII'）、整数编号。

不指定时，默认使用模版数据库的编码。模板数据库template0和template1的编码默认与操作系统环境相关。template1不允许修改字符编码，因此若要变更编码，请使用template0创建数据库。

常用取值：GBK、UTF8、Latin1。

**表 1** Vastbase字符集

| **名称** | **描述** | **语言** | **是否服务器端？** | **ICU?** | **字节/字符** | **别名** |
| --- | --- | --- | --- | --- | --- | --- |
| BIG5 | Big Five | 繁体中文 | 否 | 否 | 1-2 | WIN950，Windows950 |
| EUC\_CN | 扩展UNIX编码-中国 | 简体中文 | 是 | 是 | 1-3 | - |
| EUC\_JP | 扩展UNIX编码-日本 | 日文 | 是 | 是 | 1-3 | - |
| EUC\_JIS\_2004 | 扩展UNIX编码-日本, JIS X 0213 | 日文 | 是 | 否 | 1-3 | - |
| EUC\_KR | 扩展UNIX编码-韩国 | 韩文 | 是 | 是 | 1-3 | - |
| EUC\_TW | 扩展UNIX编码-中国台湾 | 繁体中文 | 是 | 是 | 1-3 | - |
| GB18030 | 国家标准 | 中文 | 是 | 否 | 1-4 | - |
| GBK | 扩展国家标准 | 简体中文 | 是 | 否 | 1-2 | WIN936，Windows936 |
| ISO\_8859\_5 | ISO 8859-5, ECMA 113 | 拉丁语/西里尔语 | 是 | 是 | 1 | - |
| ISO\_8859\_6 | ISO 8859-6, ECMA 114 | 拉丁语/阿拉伯语 | 是 | 是 | 1 | - |
| ISO\_8859\_7 | ISO 8859-7, ECMA 118 | 拉丁语/希腊语 | 是 | 是 | 1 | - |
| ISO\_8859\_8 | ISO 8859-8, ECMA 121 | 拉丁语/希伯来语 | 是 | 是 | 1 | - |
| JOHAB | JOHAB | 韩语 | 否 | 否 | 1-3 | - |
| KOI8R | KOI8-R | 西里尔语（俄语） | 是 | 是 | 1 | KOI8 |
| KOI8U | KOI8-U | 西里尔语（乌克兰语） | 是 | 是 | 1 | - |
| LATIN1 | ISO 8859-1, ECMA 94 | 西欧 | 是 | 是 | 1 | ISO88591 |
| LATIN2 | ISO 8859-2, ECMA 94 | 中欧 | 是 | 是 | 1 | ISO88592 |
| LATIN3 | ISO 8859-3, ECMA 94 | 南欧 | 是 | 是 | 1 | ISO88593 |
| LATIN4 | ISO 8859-4, ECMA 94 | 北欧 | 是 | 是 | 1 | ISO88594 |
| LATIN5 | ISO 8859-9, ECMA 128 | 土耳其语 | 是 | 是 | 1 | ISO88599 |
| LATIN6 | ISO 8859-10, ECMA 144 | 日耳曼语 | 是 | 是 | 1 | ISO885910 |
| LATIN7 | ISO 8859-13 | 波罗的海 | 是 | 是 | 1 | ISO885913 |
| LATIN8 | ISO 8859-14 | 凯尔特语 | 是 | 是 | 1 | ISO885914 |
| LATIN9 | ISO 8859-15 | 带欧罗巴和口音的LATIN1 | 是 | 是 | 1 | ISO885915 |
| LATIN10 | ISO 8859-16, ASRO SR 14111 | 罗马尼亚语 | 是 | 否 | 1 | ISO885916 |
| MULE\_INTERNAL | Mule内部编码 | 多语种编辑器 | 是 | 否 | 1-4 | - |
| SJIS | Shift JIS | 日语 | 否 | 否 | 1-2 | Mskanji，ShiftJIS，WIN932，Windows932 |
| SHIFT\_JIS\_2004 | Shift JIS, JIS X 0213 | 日语 | 否 | 否 | 1-2 | - |
| SQL\_ASCII | 未指定（见文本） | 任意 | 是 | 否 | 1 | - |
| UHC | 统一韩语编码 | 韩语 | 否 | 否 | 1-2 | WIN949，Windows949 |
| UTF8 | Unicode, 8-bit | 所有 | 是 | 是 | 1-4 | Unicode |
| WIN866 | Windows CP866 | 西里尔语 | 是 | 是 | 1 | ALT |
| WIN874 | Windows CP874 | 泰语 | 是 | 否 | 1 | - |
| WIN1250 | Windows CP1250 | 中欧 | 是 | 是 | 1 | - |
| WIN1251 | Windows CP1251 | 西里尔语 | 是 | 是 | 1 | WIN |
| WIN1252 | Windows CP1252 | 西欧 | 是 | 是 | 1 | - |
| WIN1253 | Windows CP1253 | 希腊语 | 是 | 是 | 1 | - |
| WIN1254 | Windows CP1254 | 土耳其语 | 是 | 是 | 1 | - |
| WIN1255 | Windows CP1255 | 希伯来语 | 是 | 是 | 1 | - |
| WIN1256 | Windows CP1256 | 阿拉伯语 | 是 | 是 | 1 | - |
| WIN1257 | Windows CP1257 | 波罗的海 | 是 | 是 | 1 | - |
| WIN1258 | Windows CP1258 | 越南语 | 是 | 是 | 1 | ABC, TCVN，TCVN5712，VSCII |

【注意】

并非所有的客户端API都支持上面列出的字符集。

SQL\_ASCII设置与其他设置表现得相当不同。如果服务器字符集是SQL\_ASCII，服务器把字节值0-127根据 ASCII标准解释，而字节值128-255则当作无法解析的字符。如果设置为SQL\_ASCII，就不会有编码转换。因此，这个设置基本不是用来声明所使用的指定编码， 因为这个声明会忽略编码。在大多数情况下，如果你使用了任何非ASCII数据，那么使用 SQL\_ASCII设置都是不明智的，因为Vastbase将无法帮助你转换或者校验非ASCII字符。

【须知】

* 指定新的数据库字符集编码必须与所选择的本地环境中（LC\_COLLATE和LC\_CTYPE）的设置兼容。
* 当指定的字符编码集为GBK时，部分中文生僻字无法直接作为对象名。这是因为GBK第二个字节的编码范围在0x40-0x7E之间时，字节编码与ASCII字符@A-Z[]^`a-z{|}重叠。其中@[]^'{|}是数据库中的操作符，直接作为对象名时，会语法报错。例如“侤”字，GBK16进制编码为0x8240，第二个字节为0x40，与ASCII“@”符号编码相同，因此无法直接作为对象名使用。如果确实要使用，可以在创建和访问对象时，通过增加双引号来规避这个问题。
* 若客户端编码为A，服务器端编码为B，则需要满足数据库中存在编码格式A与B的转换，例如：若服务器端编码为gb18030，由于当前数据库不支持gb18030与gbk的相互转换，所以此时设置客户端编码格式为gbk时，会报错“Conversion between GB18030 and GBK is not supported.”。数据库能够支持的所有的编码格式转换详见系统表pg\_conversion。
* **LC\_COLLATE [ = ] lc\_collate**

指定新数据库使用的字符集。例如，通过lc\_collate = 'zh\_CN.gbk'设定该参数。

该参数的使用会影响到对字符串的排序顺序（如使用ORDER BY执行，以及在文本列上使用索引的顺序）。默认是使用模板数据库的排序顺序。

取值范围：有效的排序类型。

* **LC\_CTYPE [ = ] lc\_ctype**

指定新数据库使用的字符分类。例如，通过lc\_ctype = 'zh\_CN.gbk'设定该参数。该参数的使用会影响到字符的分类，如大写、小写和数字。默认是使用模板数据库的字符分类。

取值范围：操作系统支持的字符分类。

【说明】

* 对于lc\_collate和lc\_ctype参数的取值范围，取决于本地环境支持的字符集。
* 例如：在Linux操作系统上，可通过locale -a命令获取操作系统支持的字符集列表，在应用lc\_collate和lc\_ctype参数时可从中选择用户需要的字符集和字符分类。
* **DBCOMPATIBILITY [ = ] compatibility\_type**

指定兼容的数据库的类型，默认兼容O。

取值范围：A、B、C、PG、MSSQL。分别表示兼容O、MY、TD、POSTGRES和SQL Server。

【说明】

* A兼容性下，数据库将空字符串作为NULL处理，数据类型DATE会被替换为TIMESTAMP(0) WITHOUT TIME ZONE。
* 将字符串转换成整数类型时，如果输入不合法，B兼容性会将输入转换为0，而其它兼容性则会报错。
* PG兼容性下，CHAR和VARCHAR以字符为计数单位，其它兼容性以字节为计数单位。例如，对于UTF-8字符集，CHAR(3)在PG兼容性下能存放3个中文字符，而在其它兼容性下只能存放1个中文字符。
* **TABLESPACE [ = ] tablespace\_name**

指定数据库对应的表空间。

取值范围：已存在表空间名。

* **CONNECTION LIMIT [ = ] connlimit**

数据库可以接受的并发连接数。

【须知】

* 系统管理员不受此参数的限制。
* connlimit数据库主节点单独统计，Vastbase整体的连接数 = connlimit \* 当前正常数据库主节点个数。

取值范围：>=-1的整数。默认值为-1，表示没有限制。

**有关字符编码的一些限制：**

* 若区域设置为C（或POSIX），则允许所有的编码类型，但是对于其他的区域设置，字符编码必须和区域设置相同。
* 若字符编码方式是SQL\_ASCII，并且修改者为管理员用户时，则字符编码可以和区域设置不相同。
* 编码和区域设置必须匹配模板数据库，除了将template0当作模板。 因为其他数据库可能会包含不匹配指定编码的数据，或者可能包含排序顺序受LC\_COLLATE和LC\_CTYPE影响的索引。复制这些数据会导致在新数据库中的索引失效。template0是不包含任何会受到影响的数据或者索引。

**示例**

**示例1：**创建数据库。

1、创建jim和tom用户。

CREATE USER jim PASSWORD 'Aa@12345';

CREATE USER tom PASSWORD 'Aa@12345';

2、创建一个"UTF-8"编码的数据库music。

CREATE DATABASE music ENCODING 'UTF-8' template = template0;

3、创建数据库music2，并指定所有者为jim。

vastbase=# CREATE DATABASE music2 OWNER jim;

4、用模板template0创建数据库music3，并指定所有者为jim。

CREATE DATABASE music3 OWNER jim TEMPLATE template0;

5、设置music数据库的连接数为10。

ALTER DATABASE music CONNECTION LIMIT= 10;

6、将music名称改为music4。

ALTER DATABASE music RENAME TO music4;

7、将数据库music2的所属者改为tom。

vastbase=# ALTER DATABASE music2 OWNER TO tom;

8、设置music3的表空间为PG\_DEFAULT。

ALTER DATABASE music3 SET TABLESPACE PG\_DEFAULT;

9、关闭在数据库music3上缺省的索引扫描。

ALTER DATABASE music3 SET enable\_indexscan TO off;

10、重置enable\_indexscan参数。

ALTER DATABASE music3 RESET enable\_indexscan;

11、删除数据库和用户。

DROP DATABASE music2;

DROP DATABASE music3;

DROP DATABASE music4;

DROP USER jim;

DROP USER tom;

**示例2：**创建中文名称的数据库。

1、创建数据库。

create database 数据库;

2、创建英文名称的数据库。

create database db\_1096053;

3、将英文数据库名称修改为中文。

alter database db\_1096053 rename to 新数据库;

4、查询数据库。

\l

返回结果为：

List of databases

Name | Owner | Encoding | Collate | Ctype | Access privileges

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

postgres | vastbase | UTF8 | en\_US.utf8 | en\_US.utf8 |

template0 | vastbase | UTF8 | en\_US.utf8 | en\_US.utf8 | =c/vastbase +

| | | | | vastbase=CTc/vastbase

template1 | vastbase | UTF8 | en\_US.utf8 | en\_US.utf8 | =c/vastbase +

| | | | | vastbase=CTc/vastbase

数据库 | vastbase | UTF8 | en\_US.utf8 | en\_US.utf8 |

新数据库 | vastbase | UTF8 | en\_US.utf8 | en\_US.utf8 |

(5 rows)

**示例3：**使用自定义模板创建数据库。

1. 创建一个数据库A并切换至数据库A。

create database A；

\A

1. 在数据库A中创建表并插入数据。

create table aa\_table(a int);

insert into aa\_table values(1);

1. 以数据库A为模板创建数据库B并切换至数据B(模板数据库的兼容类型必须和新创建的数据库兼容类型一致)。

create database B template=A；

\c B

1. 查询新创建的数据库B中是否有A中所创建的表。

select \* from aa\_table;

返回结果如下，表示支持使用自定义数据库模板创建数据库。

a

---

1

(1 row)

## 系统表

### PG\_COLLATION

PG\_COLLATION系统表描述可用的排序规则，本质上从一个SQL名称映射到操作系统本地类别。

**表 1** PG\_COLLATION字段

| **名称** | **类型** | **引用** | **描述** |
| --- | --- | --- | --- |
| collname | name | - | 排序规则名（每个名称空间和编码唯一）。 |
| collnamespace | oid | [PG\_NAMESPACE](https://docs.vastdata.com.cn/zh/docs/VastbaseG100Ver2.2.10/doc/%E5%BC%80%E5%8F%91%E8%80%85%E6%8C%87%E5%8D%97/PG_NAMESPACE.html).oid | 包含这个排序规则的名称空间的OID。 |
| collowner | oid | [PG\_AUTHID](https://docs.vastdata.com.cn/zh/docs/VastbaseG100Ver2.2.10/doc/%E5%BC%80%E5%8F%91%E8%80%85%E6%8C%87%E5%8D%97/PG_AUTHID.html).oid | 排序规则的所有者。 |
| collencoding | integer | - | 排序规则可用的编码，兼容PostgreSQL所有的字符编码类型，如果适用于任意编码为-1。 |
| collcollate | name | - | 这个排序规则对象的LC\_COLLATE。 |
| collctype | name | - | 这个排序规则对象的LC\_CTYPE。 |
| collpadattribute | name | - | 列校对规则。 NO PAD：把字符串尾端的空格当作一个字符处理，即字符串等值比较不忽略尾端空格。 PAD SPACE：字符串等值比较忽略尾端空格。 |

### PG\_DATABASE

PG\_DATABASE系统表存储关于可用数据库的信息。

**表 1** PG\_DATABASE字段

| **名称** | **类型** | **描述** |
| --- | --- | --- |
| datname | name | 数据库名称。 |
| datdba | oid | 数据库所有人，通常为其创建者。 |
| encoding | integer | 数据库的字符编码方式。 |
| datcollate | name | 数据库使用的排序顺序。 |
| datctype | name | 数据库使用的字符分类。 |
| datistemplate | Boolean | 是否允许作为模板数据库。 |
| datallowconn | Boolean | 如果为假，则没有用户可以连接到这个数据库。这个字段用于保护template0数据库不被更改。 |
| datconnlimit | integer | 该数据库上允许的最大并发连接数，-1表示无限制。 |
| datlastsysoid | oid | 数据库里最后一个系统OID 。 |
| datfrozenxid | xid32 | 用于跟踪该数据库是否需要为了防止事务ID重叠而进行清理。当前版本该字段已经废弃使用，为保持前向兼容，保留此字段，新增datfrozenxid64用于记录此信息。 |
| dattablespace | oid | 数据库的缺省表空间。 |
| datcompatibility | name | 数据库兼容模式，当前支持四种兼容模式：A、B、C、PG，分别表示兼容Oracle、MySQL 、TD和PostgreSQL。 |
| datacl | aclitem[] | 访问权限。 |
| datfrozenxid64 | xid | 用于跟踪该数据库是否需要为了防止事务ID重叠而进行清理。 |
| datminmxid | xid | 该数据库中中所有在这个之前的多事务ID已经被一个事务ID替换。这用于跟踪该数据库是否需要为了防止事务ID重叠或者允许收缩pg\_clog而进行清理。它是此数据库中所有表的pg\_class.relminmxid中的最小值。 |
| datpadattribute | name | 列校对规则。 NO PAD：把字符串尾端的空格当作一个字符处理，即字符串等值比较不忽略尾端空格。 PAD SPACE：字符串等值比较忽略尾端空格。  【说明】  从Vastbase G100 V2.2 Build12版本开始支持在创建数据库时指定列校对规则。 |

### PG\_EVENT\_TRIGGER

PG\_EVENT\_TRIGGER系统表，用于维护事件触发器的元信息。

|  |  |  |  |
| --- | --- | --- | --- |
| **名称** | **类型** | **引用** | **描述** |
| Event | Name | - | 触发器名称（必须唯一）。 |
| Evtevent | Name | - | 触发器触发事件的标识符。 |
| Evtowner | Oid | Pg\_authid\_oid | 事件触发器的拥有者。 |
| Evtfoid | Oid | Pg\_proc\_oid | 将被调用的函数。 |
| Evtenabled | Char | - | 控制事件触发器触发的会话复制角色模式。   * 0：触发器在“origin”和“local”模式触发。 * D：触发器被禁用。 * R：触发器在“replica”模式触发。 * A：触发器总是触发。 |
| Evttags | Text[] | - | 触发器将触发的命令标签。如果为空，此触发器的触发不会受命令标签的限制。 |

### PG\_SUBSCRIPTION\_REL

系统表PG\_SUBSCRIPTION\_REL包含每个订阅中每个被复制表的状态，是多对多的映射关系。

该系统表仅包含运行CREATE SUBSCRIPTION或ALTER SUBSCRIPTION ... REFRESH PUBLICATION后对订阅已知的表。

srsubstate | "char" | not null

srcsn | bigint | not null

srsublsn | text

| **名称** | **类型** | **描述** |
| --- | --- | --- |
| srsubid | oid | 订阅标识符。 |
| srrelid | oid | 订阅关系标识符。 |
| srsubstate | "char" | 订阅状态。   * i：初始化 * d：基础数据正在复制 * f：基础数据复制完成 * s：已与增量复制进度同步 * r：准备好增量复制 |
| srcsn | bigint | 基础数据复制过程中快照csn。 |
| srsublsn | text | 在s或r状态中，用于同步增量复制进度的远端LSN，否则为空。 |

## 函数

### IP白名单查询与更新函数

**功能描述**

Vastbase G100支持IP白名单查询和更新函数，可以查询和更新pg\_hba.conf文件中的配置条目。

* 调用read\_hba\_config函数，查询pg\_hba.conf文件中配置条目信息；
* 调用modify\_hba\_config函数，增加、删除、修改pg\_hba.conf文件中的配置。

**注意事项**

IP白名单查询函数：

SELECT \* FROM read\_hba\_config();

IP白名单更新函数：

SELECT \* FROM modify\_hba\_config(1,'new\_item'::cstring);

SELECT \* FROM modify\_hba\_config(2,item\_id);

SELECT \* FROM modify\_hba\_config(3,item\_id,'new\_item');

**参数说明**

* **1/2/3**

操作标识符。

取值范围：

* 1，新增操作的标识符，表示使用函数为pg\_hba.conf文件中新增配置。
* 2，删除操作的标识符，表示使用函数删除指定的pg\_hba.conf文件配置。
* 3，修改操作的标识符，表示使用函数修改指定的pg\_hba.conf文件配置。
* **new\_item**

当操作标识符为2时，new\_item代表pg\_hba.conf文件中新增的内容；操作标识符为3时，new\_item代表修改后的配置内容。

new\_item格式需符合pg\_hba.conf中配置要求，即满足如下形式：

TYPE   DATABASE       USER       ADDRESS       METHOD

* **item\_id**

配置条目ID，即配置条目位于pg\_hba.conf文件中的行数。

**注意事项**

只有系统管理员才能执行read\_hba\_config和modify\_hba\_config函数。

**示例**

1、查询pg\_hba.conf文件中配置。

SELECT \* FROM read\_hba\_config();

查询结果为：

id | type  | database | user |     ip      | method

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

89 | local | all      | all  |              | trust

91 | host  | all      | all  | 127.0.0.1/32 | trust

93 | host  | all      | all  | ::1/128      | trust

99 | host  | all      | all  | 0.0.0.0/0    | md5

(4 rows)

2、修改pg\_hba.conf文件中配置。

SELECT \* FROM modify\_hba\_config(3,89,'host all all 172.16.105.58/22 md5');

修改成功，回显为：

modify\_hba\_config

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

t

(1 row)

3、删除pg\_hba.conf文件中配置。

SELECT \* FROM modify\_hba\_config(2,89);

删除成功，回显为：

modify\_hba\_config

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

t

(1 row)

4、新增pg\_hba.conf文件中配置并查询。

SELECT \* FROM modify\_hba\_config(1,'local all all trust'::cstring);

SELECT \* FROM read\_hba\_config();

查询当前配置文件内容：

id | type  | database | user |     ip      | method

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

90 | host  | all      | all  | 127.0.0.1/32 | trust

92 | host  | all      | all  | ::1/128      | trust

98 | host  | all      | all  | 0.0.0.0/0    | md5

99 | local | all      | all  |              | trust

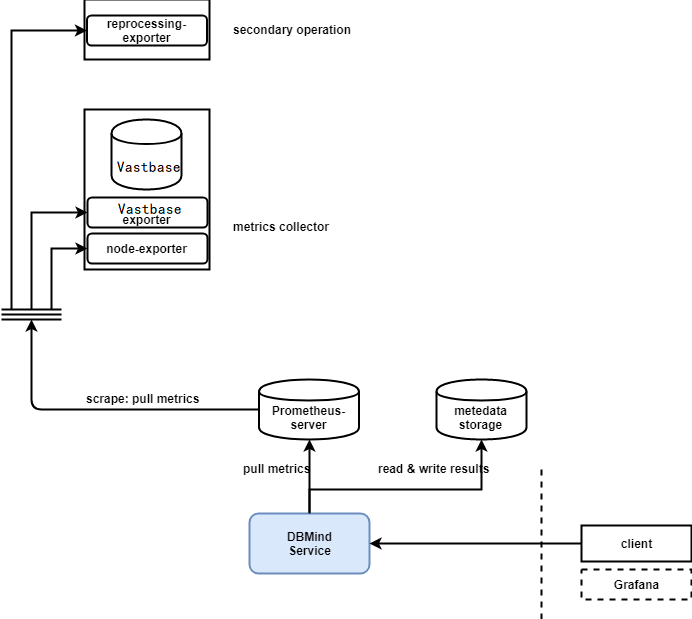
(4 rows)

## AI特性

### AI4DB: 数据库自治运维

AI4DB主要用于对数据库进行自治运维和管理，从而帮助数据库运维人员减少运维工作量。

在实现上，DBMind的AI4DB框架具有监控和服务化的性质，同时也提供即时AI工具包，提供开箱即用的AI运维功能（如索引推荐）。

AI4DB的监控平台以开源的Prometheus为主，DBMind提供监控数据生产者exporter, 可与Prometheus平台完成对接。DBMind的AI4DB服务架构如下图所示：

**组件说明**

* DBMind Service：DBMind后台服务，可用于定期离线计算，包括慢SQL根因分析、时序预测等。
* Prometheus-server：Prometheus 监控指标存储的服务器。
* metadatabase：DBMind在离线计算结束后，将计算结果存储在此处，支持Vastbase、SQLite等数据库。
* client: 用户读取DBMind离线计算结果的客户端，目前仅实现命令行客户端；若采用vastbase等数据库存储计算DBMind计算结果，则用户可以自行配置Grafana等可视化工具对该结果进行可视化。
* opengauss-exporter：用户从vastbase数据库节点上采集监控指标，供DBMind服务进行计算。
* node-exporter：Prometheus官方提供的exporter, 可用于监控该节点的系统指标，如CPU和内存使用情况。
* reprocessing-exporter：用于对Prometheus采集到的指标进行二次加工处理，例如计算CPU使用率等。

**环境配置**

DBMind外置AI功能需要运行在Python 3.7版本及以上，需要的第三方依赖包记录在AI功能根目录（$GAUSSHOME/bin/dbmind）的requirements.txt文件中（包括requirements-x86.txt与requirements-arrch64.txt，用户可根据自己平台类型选择）中，可以通过pip install命令安装依赖，如：

pip install requirements-x86.txt

如果用户没有安装齐全所需的依赖，则当用户执行gs\_dbmind命令时，会再次提醒用户安装第三方依赖。需要注意，该文件提供了DBMind所需的第三方依赖，若用户环境存在第三方包冲突等情况，用户可根据实际情况进行处理。

详细的环境配置过程请参考DBmind环境配置章节。

#### DBMind的AI子功能

用户可以通过gs\_dbmind的component子命令启动对应的AI子功能，下述章节展示不同AI功能的具体内容和使用详情。

##### SQL Rewriter: SQL语句改写

**功能描述**

SQL Rewriter是一个SQL改写工具，根据预先设定的规则，将查询语句转换为更为高效或更为规范的形式，使得查询效率得以提升。

【说明】

* 本功能不适用包含子查询的语句。
* 本功能只支持SELECT语句和DELETE对整个表格删除的语句。
* 本功能包含11个改写规则，对不符合改写规则的语句，不会进行处理。
* 本功能会对原始查询语句和改写后语句进行屏幕输出，不建议对包含涉敏感信息的SQL语句进行改写。
* union转union all规则避免了去重，从而提升了查询性能，所得结果有可能存在冗余。
* 语句中如包含order by+ 指定列名或group by+ 指定列名，无法适用SelfJoin规则。
* SQL改写工具不保证查询语句等价转换，其目的是提升查询语句效率。

**语法格式**

使用SQL Rewriter前，您可以通过以下指令获取帮助：

gs\_dbmind component sql\_rewriter --help

显示如下帮助信息：

usage: [-h] [--db-host DB\_HOST] [--db-user DB\_USER] [--schema SCHEMA]

db\_port database file

SQL Rewriter

positional arguments:

db\_port Port for database

database Name for database

file File containing SQL statements which need to rewrite

optional arguments:

-h, --help show this help message and exit

--db-host DB\_HOST Host for database

--db-user DB\_USER Username for database log-in

--schema SCHEMA Schema name for the current business data

**参数说明**

| **参数** | **参数说明** |
| --- | --- |
| db\_port | 数据库端口号。 |
| database | 数据库名称。 |
| file | 包含多个查询语句的文件路径。 |
| --db-host | （可选）数据库主机号。 |
| --db-user | （可选）数据库用户名。 |
| --schema | （可选，模式为public）模式。 |

**示例**

1. 登录vastbase数据库。

vsql -r -d vastbase

1. 创建测试数据。

CREATE TABLE test1(id int,name varchar);

INSERT INTO test1(id,name) VALUES(1,'zhangsan');

INSERT INTO test1(id,name) VALUES(2,'lisi');

INSERT INTO test1(id,name) VALUES(3,'zhangsan');

INSERT INTO test1(id,name) VALUES(4,'wangwu');

INSERT INTO test1(id,name) VALUES(5,'xiaoming');

INSERT INTO test1(id,name) VALUES(6,'xiaohong');

INSERT INTO test1(id,name) VALUES(7,'LISI');

1. 退出数据库。

\q

1. 创建需要改写的SQL文件queries.sql，写入以下内容。

SELECT id FROM test1 GROUP BY ID HAVING id=1;

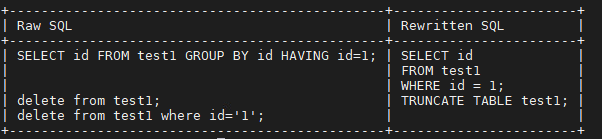
delete from test1;

delete from test1 where id='1';

1. 执行如下命令使用SQL Rewriter工具改写SQL。

gs\_dbmind component sql\_rewriter 5732 vastbase queries.sql --db-host 127.0.0.1 --db-user vastbase\_ai --schema public

返回结果为改写后的查询语句，无法改写的语句，显示为空：



##### Anomaly detection: 异常检测

**功能描述**

Anomaly detection 异常检测模块主要基于统计方法来实现时序数据来发现数据中存在的可能的异常情况。该模块框架解耦，可以实现不同异常检测算法的灵活替换，而且该模块功能可以根据时序数据的不同特征来自动选择算法，支持异常值检测、阈值检测、箱型图检测、梯度检测、增长率检测、波动率检测和状态转换检测。

**语法格式**

gs\_dbmind component anomaly\_detection --help

显示如下帮助信息：

usage: anomaly\_detection.py [-h] --action {overview,plot} -c CONF -m METRIC -s

START\_TIME -e END\_TIME [-H HOST] [-a ANOMALY]

Workload Anomaly detection: Anomaly detection of monitored metric.

optional arguments:

-h, --help show this help message and exit

--action {overview,plot}

choose a functionality to perform

-c CONF, --conf CONF set the directory of configuration files

-m METRIC, --metric METRIC

set the metric name you want to retrieve

-s START\_TIME, --start-time START\_TIME

set the start time of for retrieving in ms

-e END\_TIME, --end-time END\_TIME

set the end time of for retrieving in ms

-H HOST, --host HOST set a host of the metric, ip only or ip and port.

-a ANOMALY, --anomaly ANOMALY

set a anomaly detector of the metric(increase\_rate,

level\_shift, spike, threshold)

Process finished with exit code 0

**参数说明**

| **参数** | **参数说明** | **取值范围** |
| --- | --- | --- |
| -h, --help | 帮助命令。 | - |
| --action | 动作参数。 | * overview：概览 * plot：可视化 |
| -c，--conf | 配置文件目录。 | - |
| -m，--metric-name | 指定显示指标名。 | - |
| -H, --host | 指定数据来源地址信息，通过地址信息进行过滤。 | -ip地址或者ip地址加端口号 |
| -a, --anomaly | 指定异常检测方式，用于过滤。 | - |
| -s, --start-time | 显示开始时间的时间戳，单位毫秒；或日期时间格式为 %Y-%m-%d %H:%M:%S。 | 正整数或日期时间格式 |
| -e, --end-time | 显示结束时间的时间戳，单位毫秒；或日期时间格式为 %Y-%m-%d %H:%M:%S。 | 正整数或日期时间格式 |

【说明】

在输入anomaly detection的参数时，--start-time至少要比--end-time早30秒以上。

**使用示例**

假设指标采集系统运行正常，并且用户已经初始化了配置文件目录confpath，则可以通过下述命令实现本特性的功能。

* 仅启动异常检测功能。

gs\_dbmind service start --conf confpath --only-run anomaly\_detection

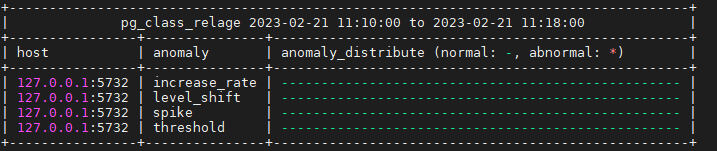
* 对于某一指标，在全部节点上，从timestamps1到timestamps2时间段内的数据进行概览。

gs\_dbmind component anomaly\_detection --conf confpath --action overview --metric metric\_name --start-time timestamps1 --end-time timestamps2

例如：

gs\_dbmind component anomaly\_detection --conf confpath --action overview --metric pg\_class\_relage --start-time '2023-02-21 11:10:00' --end-time '2023-02-21 11:18:00'

返回结果如下：



* 对于某一指标，在特定节点上，从timestamps1到timestamps2时间段内的数据进行概览。

gs\_dbmind component anomaly\_detection --conf confpath --action overview --metric metric\_name --start-time timestamps1 --end-time timestamps2 --host ip\_address --anomaly anomaly\_type

例如：

gs\_dbmind component anomaly\_detection --conf confpath --action overview --metric pg\_class\_relage --start-time '2023-02-21 11:10:00' --end-time '2023-02-21 11:18:00' --host 127.0.0.1:5732

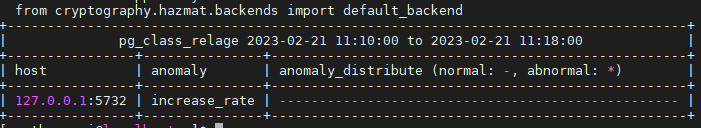
* 对于某一指标，在全部节点上，从timestamps1到timestamps2时间段内的数据，以特定异常检测方式进行概览。

gs\_dbmind component anomaly\_detection --conf confpath --action overview --metric metric\_name --start-time timestamps1 --end-time timestamps2 --anomaly anomaly\_type

例如：

gs\_dbmind component anomaly\_detection --conf confpath --action overview --metric pg\_class\_relage --start-time '2023-02-21 11:10:00' --end-time '2023-02-21 11:18:00' --anomaly increase\_rate

返回结果为：



* 对于某一指标，在特定节点，从timestamps1到timestamps2时间段内的数据，以特定异常检测方式进行概览。

gs\_dbmind component anomaly\_detection --conf confpath --action overview --metric metric\_name --start-time timestamps1 --end-time timestamps2 --host ip\_address --anomaly anomaly\_type

例如：

gs\_dbmind component anomaly\_detection --conf confpath --action overview --metric pg\_class\_relage --start-time '2023-02-21 11:10:00' --end-time '2023-02-21 11:18:00' --host 127.0.0.1:5732 --anomaly increase\_rate

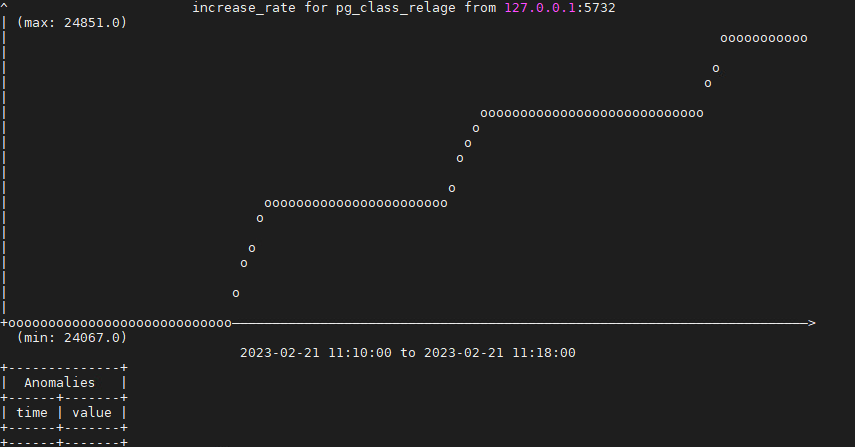
* 对于某一指标，在特定节点，从timestamps1到timestamps2时间段内的数据，以特定异常检测方式进行可视化展示。

gs\_dbmind component anomaly\_detection --conf confpath --action plot --metric metric\_name --start-time timestamps1 --end-time timestamps2 --host ip\_address --anomaly anomaly\_type

例如：

gs\_dbmind component anomaly\_detection --conf confpath --action plot --metric pg\_class\_relage --start-time '2023-02-21 11:10:00' --end-time '2023-02-21 11:18:00' --host 127.0.0.1:5732 --anomaly increase\_rate

返回结果为：



* 停止已启动的服务。

gs\_dbmind service stop --conf confpath

## 外部数据封装器

### Postgres\_FDW

**功能描述**

postgres\_fdw功能提供了外部数据封装器postgres\_fdw插件，它可以被用来访问存储在外部Vastbase服务器中的数据。

用户可以通过指定参数[remote\_kind](" \l "kind)的值来控制远端连接的数据库为Vastbase或是PostgreSQL 11。

**语法格式**

1、加载postgres\_fdw扩展。

CREATE EXTENSION [ IF NOT EXISTS ] extension\_name

 [ WITH ] [ SCHEMA schema\_name ]

      [ VERSION version ]

      [ FROM old\_version ]

2、创建一个新的外部服务器。

CREATE SERVER server\_name

FOREIGN DATA WRAPPER fdw\_name

OPTIONS ( { option\_name ' value ' } [, ...] ) ;

3、创建一个用户到一个外部服务器的新映射。

CREATE USER MAPPING FOR { user\_name | USER | CURRENT\_USER | PUBLIC }

 SERVER server\_name

[ OPTIONS ( option 'value' [ , ... ] ) ]

4、创建外表。

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

column\_name type\_name [ OPTIONS ( option 'value' [, ... ] ) ] [ COLLATE collation ] [ column\_constraint [ ... ] ]

[, ... ]

] )

SERVER server\_name

[ OPTIONS ( option 'value' [, ... ] ) ]

这里column\_constraint 可以是:

[ CONSTRAINT constraint\_name ]

{ NOT NULL |

NULL |

DEFAULT default\_expr }

**参数说明**

* **server\_name**

server的名称。

取值范围：长度必须小于等于63。

* **fdw\_name**

指定外部数据封装器的名称。

取值范围：oracle\_fdw，mysql\_fdw，postgres\_fdw，mot\_fdw。

* **CREATE SERVER中OPTIONS参数包括：**
* host（必选）：要连接的主机名，本功能填写远端服务器的IP地址。
* port（必选）：主机服务器的端口号，本功能填写远端服务器上集群的端口号。
* dbname（必选）：数据库名，本功能填写远端服务器上的数据库名。
* remote\_kind（非必选）：远程连接类型，可选取值为vastbase（默认）或postgres。若用户指定其为postgres可以连接到远端PostgreSQL 11数据库。如果不指定此参数，默认按照远程连接到Vastbase数据库的逻辑进行处理。
* hostaddr(非必选)：与之链接的主机的IP地址，是标准的IPv4地址格式，比如，172.28.40.9。如果机器支持IPv6，那么也可以使用IPv6的地址。如果声明了一个非空的字符串，那么使用TCP/IP通讯机制。
* connect\_timeout(非必选)：链接的最大等待时间，以秒计（用十进制整数字符串书写），0或者不声明表示无穷。不建议把链接超时的值设置得小于2秒。
* options(非必选)：添加命令行选项以在运行时发送到服务器。
* keepalives(非必选)：控制客户端侧的TCP保持激活是否使用。缺省值是1，意思为打开，但是如果不想要保持激活，你可以更改为0，意思为关闭。通过Unix域套接字做的链接忽略这个参数。
* keepalives\_idle(非必选)：在TCP应该发送一个保持激活的信息给服务器之后，控制不活动的秒数。0值表示使用系统缺省。通过Unix域套接字做的链接或者如果禁用了保持激活则忽略这个参数。
* keepalives\_interval(非必选)：在TCP保持激活信息没有被应该传播的服务器承认之后，控制秒数。0值表示使用系统缺省。通过Unix域套接字做的链接或者如果禁用了保持激活则忽略这个参数。
* keepalives\_count(非必选)：添加命令行选项以在运行时发送到服务器。例如，设置为-c comm\_debug\_mode=off设置guc参数comm\_debug\_mode参数的会话的值为off。
* use\_remote\_estimate(非必选)：控制postgres\_fdw是否发出EXPLAIN命令以获取运行消耗估算。默认值为false。
* fdw\_startup\_cost(非必选)：执行一个外表扫描时的启动耗时估算。这个值通常包含建立连接、远端对请求的分析和生成计划的耗时。默认值为100。
* fdw\_typle\_cost(非必选)：在远端服务器上对每一个元组进行扫描时的额外消耗。这个值通常表示数据在server间传输的额外消耗。默认值为0.01。
* **user\_name**

要映射到外部服务器的一个现有用户的名称。 CURRENT\_USER和USER匹配当前用户的名称。 当PUBLIC被指定时，一个所谓的公共映射会被创建，当没有特定用户的映射可用时将会使用它。

* **server\_name**

将为其创建用户映射的现有服务器的名称。

* **CREATE USER MAPPING中OPTIONS参数包括：**
* user(必选)：远端服务器上Vastbase的用户名，属于普通用户。
* password（必选）：远端服务器上Vastbase用户对应的密码
* **table\_name**

外表的表名。取值范围：字符串，要符合标识符的命名规范。

* **column\_name**

外表中的字段名。取值范围：字符串，要符合标识符的命名规范。

* **type\_name**

**字段的数据类型。**

* **SERVER server\_name**

外表的server名称。

* **OPTIONS 参数包括**
* schema\_name（必选）：远端server的schema名称。如果不指定的话，将使用外表自身的schema名称作为远端的schema名称。
* table\_name（必选）：远端server的表名。如果不指定的话，将使用外表自身的表名作为远端的表名。
* column\_name（非必选）：远端server的表的列名。如果不指定的话，将使用外表自身的列名作为远端的的表的列名。

**注意事项**

* 两个postgres\_fdw外表间的SELECT JOIN不支持下推到远端Vastbase执行，会被分成两条SQL语句传递到远端Vastbase执行，然后在本地汇总处理结果。
* 不支持IMPORT FOREIGN SCHEMA语法。
* 不支持对外表进行CREATE TRIGGER操作。
* 若需要Vastbase访问远端PostgreSQL 11的分区表，需要设置GUC参数sql\_beta\_feature的值为partition\_fdw\_on。关于此参数的详细说明及配置方法可以参考《Vastbase G100 V2.2 开发者指南 》->GUC参数说明->其他优化器选项->[sql\_beta\_feature](%E5%85%B6%E4%BB%96%E4%BC%98%E5%8C%96%E5%99%A8%E9%80%89%E9%A1%B9.md/" \l "section29331229203010)。

**示例**

**示例1：**使用postgres\_fdw连接到远程vastbase数据库，并访问外表。

**前置条件：**

在本地服务器和远端服务器创建数据库用户之前，建议关闭远端数据库和本地数据库的强制修改密码功能。（参数默认启用，修改后重启生效。）

alter system set password\_force\_alter=off;

1、在本地服务器创建用户。

create user hzy with sysadmin password '123456Aa';

grant all on database vastbase to hzy;

2、远端服务器修改配置文件pg\_hba.conf的配置参数，新增下列参数。

host all   all       0.0.0.0/0         sha256

3、远端服务器修改配置文件postgresql.conf中的listen\_addresses 参数，值为本地服务器的IP地址。

4、远端服务器中创建用户。

create user hzy with sysadmin password '123456Aa';

grant all on database vastbase to hzy;

5、远端服务器使用hzy用户创建测试表。

create table student( student\_no int4 primary key, student\_name varchar(30), age int2 );

insert into student values(1,'xiaoming',12);

6、在本地服务器中，使用新建的用户登录系统。

\c - hzy

【说明】

以下操作均在本地服务器上执行。

7、加载postgres\_fdw扩展。

create extension postgres\_fdw;

8、创建外部服务器。

create server server\_to\_200 foreign data wrapper postgres\_fdw options (host '172.16.105.54',port '5432',dbname 'vastbase');

9、创建一个用户到外部服务器的映射。

create user MAPPING FOR hzy SERVER server\_to\_200 OPTIONS (user 'hzy',password '123456Aa');

10、创建外表。

create foreign table local\_foreign\_table\_student( student\_no int4 , student\_name varchar(30), age int2) server server\_to\_200 options(schema\_name 'public', table\_name 'student');

11、访问外表。

select \* from local\_foreign\_table\_student;

访问外表成功，返回结果为：

student\_no | student\_name | age

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

         1 | xiaoming     |  12

(1 row)

12、对外表执行DML操作并再次查看外表。

insert into local\_foreign\_table\_student values (2,'xiaohong',11);

update local\_foreign\_table\_student set student\_name = 'xiaoqing' where student\_no = 1;

delete from local\_foreign\_table\_student where age = 11;

select \* from local\_foreign\_table\_student ;

操作成功，返回结果为：

student\_no | student\_name | age

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

         1 | xiaoqing     |  12

(1 row)

**示例2：**使用postgres\_fdw连接到远程PostgreSQL 11数据库，并执行查询表的操作。

**前置条件：**以下步骤在远端服务器执行。

1、具备远端PostgreSQL 11的数据库环境。

2、在远端数据库中创建数据库用户u1。

create user u1 password 'Test1234';

3、创建并切换至测试数据库dtest下。

create database dtest;

\c dtest;

4、以用户u1的身份在远端测试数据库中创建测试表t1，并插入数据。

\c - u1

create table t1(id int,name text);

insert into t1 values(101,'zhangsan');

insert into t1 values(102,'lisi');

**示例步骤：**以下步骤在本地服务器中执行。

1、创建扩展postgres\_fdw。

create extension postgres\_fdw;

2、创建DBLINK，指定参数remote\_kind为'postgres'。

create database link link1 connect to u1 identified by 'Test1234' using postgres\_fdw(dbname 'dtest', host '127.0.0.1', port '5432', remote\_kind 'postgres');

3、查询远端PG数据库中的t1表。

select \* from public.t1@link1;

返回结果如下：查询到t1中的数据，且数据正确。

id  | name

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

101 | zhangsan

102 | lisi

(2 rows)

**示例3：**使用postgres\_fdw连接到远程PostgreSQL 11数据库，并查询分区表的数据。

前置条件：以下步骤在远端服务器执行。

【注意】

远程连接前请确认远端PostgreSQL数据库已经在配置文件pg\_hba.conf中允许其它用户访问此数据库。

1、具备远端PostgreSQL 11的数据库环境。

2、在远端数据库中创建数据库用户u2。

create user u2 password 'Test1234';

3、创建并切换至测试数据库dtest下。

create database dtest;

\c dtest;

4、以用户u2的身份在远端测试数据库中创建分区表。

\c - u2

create table test\_partition(col1 int ,col2 text) partition by list(col1);

create table test\_subpartition1 partition of test\_partition for values in (1);

create table test\_subpartition2 partition of test\_partition for values  in (2);

create table test\_subpartition3 partition of test\_partition for values  in (3);

create table test\_subpartition4 partition of test\_partition for values  in (4);

create table test\_subpartition5 partition of test\_partition for values  in (5);

5、向分区表test\_partition中插入数据。

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

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

insert into test\_partition values (3,'test3');

insert into test\_partition values (4,'test4');

insert into test\_partition values (5,'test5');

**示例步骤：**以下步骤在本地服务器中执行。

1、创建扩展postgres\_fdw。

create extension postgres\_fdw;

2、创建DBLINK，指定参数remote\_kind为'postgres'。

create database link link2 connect to u2 identified by 'Test1234' using postgres\_fdw(dbname 'dtest', host '127.0.0.1', port '5432', remote\_kind 'postgres');

3、开启分区表查询开关。

set sql\_beta\_feature=PARTITION\_FDW\_ON;

4、查询远端PG数据库中的分区表。

select \* from public.test\_partition@link2;

返回结果如下：查询到test\_partition表中的数据，且数据正确。

col1 | col2

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

  1 | test1

  2 | test2

  3 | test3

  4 | test4

  5 | test5

(5 rows)

# 管理员指南

## 数据库使用

### 创建和管理分区表

#### 分区表概述

**功能描述**

Vastbase数据库支持的分区表：Range分区表、List分区表、Hash分区表和system分区表。

* Range分区表：范围分区根据您为每个分区建立的分区键值的范围将数据映射到分区。这种分区方式是最常见的，并且分区键值经常采用日期，例如将销售数据按照月份进行分区。
* List分区表：将数据中包含的键值分别存储在不同的分区中，依次将数据映射到每一个分区，分区中包含的键值由创建分区表时指定，键值最多不超过127个。
* Hash分区表：将数据根据内部哈希算法依次映射到每一个分区中，包含的分区个数由创建分区表时指定。
* system分区表：系统分区可以对没有分区键的表进行分区。创建分区时可以指定分区名称，也可以不指定分区名称，由系统自动分配。

分区表和普通表相比具有以下优点：

* 改善查询性能：对分区对象的查询可以仅搜索自己关心的分区，提高检索效率。
* 增强可用性：如果分区表的某个分区出现故障，表在其他分区的数据仍然可用。
* 方便维护：如果分区表的某个分区出现故障，需要修复数据，只修复该分区即可。
* 均衡I/O：可以把不同的分区映射到不同的磁盘以平衡I/O，改善整个系统性能。

**语法格式**

创建分区表、修改表分区（增删分区、切割分区、合成分区、交换分区、以及修改分区属性等）的语法格式和参数说明请参考《Vastbase G100 V2.2 开发者指南 》->SQL语法参考->SQL语法->[CREATE TABLE PARTITION](CREATE-TABLE-PARTITION.md)、[ALTER TABLE PARTITION](ALTER-TABLE-PARTITION.md)。

**参数说明**

无。

**注意事项**

* 普通表若要转成分区表，需要新建分区表，然后把普通表中的数据导入到新建的分区表中。因此在初始设计表时，请根据业务提前规划是否使用分区表。
* Hash分区只支持1个分区键，分区键值能用表的普通字段来指定，分区范围的值只支持常量表达式、数值或字符串常量，不支持新增和删除分区。
* system分区表仅在数据库兼容模式为Oracle时有效（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。
* system分区表暂不支持二级分区、组合分区。
* 暂不支持对列存表、MOT表进行分区操作。
* 交换分区的普通表，必须与分区表的字段个数、字段类型都完全一致时才可以进行exchange操作，且普通表不能是临时表。（支持增删列之后的表进行分区交换。）

**示例**

**示例1：** 按照以下方式对Range分区表进行操作。

1、创建分区表。

CREATE TABLE customer\_address

(

  ca\_address\_sk       integer                  NOT NULL   ,

  ca\_address\_id       character(16)            NOT NULL   ,

  ca\_street\_number    character(10)                       ,

  ca\_street\_name      character varying(60)               ,

  ca\_street\_type      character(15)                       ,

  ca\_suite\_number     character(10)                       ,

  ca\_city             character varying(60)               ,

  ca\_county           character varying(30)               ,

  ca\_state            character(2)                        ,

  ca\_zip              character(10)                       ,

  ca\_country           character varying(20)               ,

  ca\_gmt\_offset       numeric(5,2)                        ,

  ca\_location\_type    character(20)

)

TABLESPACE example1

PARTITION BY RANGE (ca\_address\_sk)

(

       PARTITION P1 VALUES LESS THAN(5000),

       PARTITION P2 VALUES LESS THAN(10000),

       PARTITION P3 VALUES LESS THAN(15000),

       PARTITION P4 VALUES LESS THAN(20000),

       PARTITION P5 VALUES LESS THAN(25000),

       PARTITION P6 VALUES LESS THAN(30000),

       PARTITION P7 VALUES LESS THAN(40000),

       PARTITION P8 VALUES LESS THAN(MAXVALUE)

TABLESPACE example2

)

ENABLE ROW MOVEMENT;

当结果显示为如下信息，则表示创建成功:

CREATE TABLE

【说明】

创建列存分区表的数量建议不超过1000个。

2、插入数据。

将表customer\_address的数据插入到表web\_returns\_p2中。

例如在数据库中创建了一个表customer\_address的备份表tpcds.web\_returns\_p2，现在需要将表tpcds.customer\_address中的数据插入到表tpcds.web\_returns\_p2中，则可以执行如下命令。

创建备份表web\_returns\_p2并插入数据。

CREATE TABLE web\_returns\_p2

(

  ca\_address\_sk       integer                  NOT NULL   ,

  ca\_address\_id       character(16)            NOT NULL   ,

  ca\_street\_number    character(10)                       ,

  ca\_street\_name      character varying(60)               ,

  ca\_street\_type      character(15)                       ,

  ca\_suite\_number     character(10)                       ,

  ca\_city             character varying(60)               ,

  ca\_county           character varying(30)               ,

  ca\_state            character(2)                        ,

  ca\_zip              character(10)                       ,

  ca\_country           character varying(20)               ,

  ca\_gmt\_offset       numeric(5,2)                        ,

  ca\_location\_type    character(20)

)

TABLESPACE example1

PARTITION BY RANGE (ca\_address\_sk)

(

       PARTITION P1 VALUES LESS THAN(5000),

       PARTITION P2 VALUES LESS THAN(10000),

       PARTITION P3 VALUES LESS THAN(15000),

       PARTITION P4 VALUES LESS THAN(20000),

       PARTITION P5 VALUES LESS THAN(25000),

       PARTITION P6 VALUES LESS THAN(30000),

       PARTITION P7 VALUES LESS THAN(40000),

       PARTITION P8 VALUES LESS THAN(MAXVALUE) TABLESPACE example2

)

ENABLE ROW MOVEMENT;

​

INSERT INTO web\_returns\_p2 SELECT \* FROM customer\_address;

3、重命名分区。

ALTER TABLE web\_returns\_p2 RENAME PARTITION P8 TO P\_9;

ALTER TABLE web\_returns\_p2 RENAME PARTITION FOR (40000) TO P8;

4、查询分区P6。

SELECT \* FROM web\_returns\_p2 PARTITION (P6);

SELECT \* FROM web\_returns\_p2 PARTITION FOR (35888);

5、删除分区表和表空间。

DROP TABLE web\_returns\_p2;

DROP TABLE customer\_address;

**示例2：** 按照以下方式对Hash分区表进行操作。

创建分区表（ Hash分区只支持1个分区键，分区键值能用表的普通字段来指定，分区范围的值只支持常量表达式、数值或字符串常量，不支持新增和删除分区。） 语法形式如下：

CREATE TABLE [IF NOT EXISTS] table\_name

…

PARTITION BY HASH (columns\_name)

hash\_partition\_desc

…

1、创建分区表。

CREATE TABLE t\_hash\_1

(c1 integer,

c2 date,

c3 text)

PARTITION BY HASH (c1)

(

PARTITION t\_hash\_p1,

PARTITION t\_hash\_p2

);

2、插入数据并查询结果。

insert into t\_hash\_1 values(1,'2020-07-29','a');

SELECT \* FROM t\_hash\_1 PARTITION (t\_hash\_p1);

返回结果为：

c1 |         c2          | c3

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

 1 | 2020-07-29 00:00:00 | a

(1 row)

3、更新数据并查询结果。

UPDATE t\_hash\_1 set c1=3 where c1=1;

SELECT \* FROM t\_hash\_1 PARTITION (t\_hash\_p2);

返回结果为：

c1 |         c2          | c3

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

 3 | 2020-07-29 00:00:00 | a

(1 row)

4、删除数据并查询结果。

DELETE t\_hash\_1 where c1=3;

SELECT \* FROM t\_hash\_1;

返回结果为：

c1 | c2 | c3

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

(0 rows)

**示例3：** 按照以下方式对List分区表进行操作。

1、创建分区表。

CREATE TABLE t\_list

(c1 integer,

c2 date,

c3 text)

PARTITION BY LIST (c2)

(

PARTITION p1 VALUES ('2019-10-12'),

PARTITION p2 VALUES ('2019-10-13'),

PARTITION p3 VALUES ('2019-10-14')

);

2、修改分区表行迁移属性。

alter table t\_list enable row movement;

3、新增分区。

ALTER TABLE t\_list ADD PARTITION P4 VALUES ('2019-10-15');

4、删除分区。

ALTER TABLE t\_list DROP PARTITION p4;

5、插入数据并查询结果。

insert into t\_list values(1,'2019-10-13','test');

SELECT \* FROM t\_list PARTITION (p2);

返回结果为：

c1 |         c2          | c3

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

 1 | 2019-10-13 00:00:00 | test

(1 row)

查看其它的分区：

SELECT \* FROM t\_list PARTITION (p1);

返回结果为：

c1 | c2 | c3

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

(0 rows)

6、更新数据并查询结果。

update t\_list set c2='2019-10-12' where c1=1;

SELECT \* FROM t\_list PARTITION (p2);

返回结果为：

c1 | c2 | c3

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

(0 rows)

SELECT \* FROM t\_list PARTITION (p1);

返回结果为：

c1 |         c2          | c3

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

 1 | 2019-10-12 00:00:00 | test

(1 row)

**示例4：** 按照以下方式对system分区表进行操作。

1、创建分区表（指定分区名称），并向其中插入数据。

create table system\_par\_tab(

c1 integer,

c2 date,

c3 text

)

partition by system

(

partition part\_1,

partition part\_2,

partition part\_3

);

2、创建分区表（系统自动分配分区名称），并向其中插入数据。

create table system\_par\_tab2(

id number,

code varchar2(10),

description varchar2(50)

)

partition by system;

3、查询系统自动生成的分区名称。

查询系统表pg\_partition。

\d+ system\_par\_tab2

select relname,parentid,partstrategy from pg\_partition where relname='system\_par\_tab2';

返回结果为：

    relname     | parentid | partstrategy

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

system\_par\_tab2 |    16707 | h

system\_par\_tab2 |    16707 | s

(2 rows)

获取上面的parentid进行查询。

select \* from pg\_partition where parentid='16707';

返回结果为：

    relname     | parttype | parentid | partitionid | rangenum | intervalnum | partstrategy | subpartstrategy | r

elfilenode | reltablespace | relpages | reltuples | relallvisible | reltoastrelid | reltoastidxid | indextblid | i

ndisusable | reldeltarelid | reldeltaidx | relcudescrelid | relcudescidx | relfrozenxid | intspnum | partkey | sub

partkey | intervaltablespace | interval | boundaries | transit |                           reloptions

              | subparttemplate | relfrozenxid64 | relminmxid

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

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

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

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

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

system\_par\_tab2 | r        |    16707 |           0 |        0 |           0 | s            | n               |

        0 |             0 |        0 |         0 |             0 |             0 |             0 |          0 | t

          |             0 |           0 |              0 |            0 | 0            |          |         |

       |                    |          |            |         | {orientation=row,compression=no,fillfactor=80,wai

t\_clean\_gpi=n} |                 |              0 |          0

system\_par\_tab2 | p        |    16707 |           0 |        0 |           0 | h            | n               |

    16711 |             0 |        0 |         0 |             0 |         16712 |             0 |          0 | t

          |             0 |           0 |              0 |            0 | 14918        |          |         |

       |                    |          |            |         | {orientation=row,compression=no,fillfactor=80}

              |                 |          14918 |          2

(2 rows)

其中system\_par\_tab2 就是系统自动生成的分区名。

4、向分区表中插入数据。

insert into system\_par\_tab partition(part\_1) values(1,'2022-01-01','p1');

insert into system\_par\_tab partition(part\_2) values(2,'2022-02-01','p2');

insert into system\_par\_tab partition(part\_3) values(3,'2022-03-01','p3');

【须知】

向一个system分区表中插入数据时必须指定其分区名称。

5、查询分区表。

select \* from system\_par\_tab;

返回结果为：

c1 |         c2          | c3

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

 1 | 2022-01-01 00:00:00 | p1

 2 | 2022-02-01 00:00:00 | p2

 3 | 2022-03-01 00:00:00 | p3

(3rows)

6、更新数据。

update system\_par\_tab partition(part\_2) set c3='p5' where c1 ='2';

7、新增分区。

alter table system\_par\_tab add partition part\_4;

8、删除分区。

alter table system\_par\_tab drop partition part\_4;

9、清空分区表。

truncate table system\_par\_tab2;

10、清空指定分区.

alter table system\_par\_tab truncate partition part\_1;

**示例5：** system分区交换。

1、创建分区表并插入数据。

create table system\_par3

(c1 integer,

c2 date,

c3 text)

partition by system

(

partition p1,

partition p2,

partition p3

);

insert into system\_par3 partition(p1) values(1,'2022-01-01','p1');

insert into system\_par3 partition(p2) values(2,'2022-02-01','p2');

insert into system\_par3 partition(p3) values(3,'2022-03-01','p3');

2、创建普通表并插入数据。

create table system\_part

(c1 integer,

c2 date,

c3 text);

insert into system\_part values(4,'2022-04-01','p2');

insert into system\_part values(5,'2022-05-01','p2');

3、交换分区。

alter table system\_par3 exchange partition p2 with table system\_part;

4、查看交换结果。

select \* from system\_par3;

返回结果为：

c1 |         c2          | c3

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

 1 | 2022-01-01 00:00:00 | p1

 4 | 2022-04-01 00:00:00 | p2

 5 | 2022-05-01 00:00:00 | p2

 3 | 2022-03-01 00:00:00 | p3

(4 rows)

select \* from system\_par3 partition(p2);

返回结果为：

c1 |         c2          | c3

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

 4 | 2022-04-01 00:00:00 | p2

 5 | 2022-05-01 00:00:00 | p2

(2 rows)

select \* from system\_part;

select \* from system\_part;

返回结果为：

c1 |         c2          | c3

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

 2 | 2022-02-01 00:00:00 | p2

(1 row)

【说明】​

交换分区的普通表，必须与分区表的字段个数、字段类型都完全一致时才可以进行exchange操作，且普通表不能是临时表。

#### 支持二级分区

**功能描述**

二级分区功能，即在原有的range分区、list分区、hash分区、interval分区的基础上再次进行分区。

二级分区可以对表中的每个分区再次进行分区。分区类型有range、list、hash三种，一级与二级分区的分区类型可以任意组合。二级分区支持使用CREATE/ALTER/SELECT语法，用于二级分区的创建与增删改查。二级分区的相关信息可在系统表[PG\_PARTITION](../%E5%BC%80%E5%8F%91%E8%80%85%E6%8C%87%E5%8D%97/PG_PARTITION.md)中获取。

**语法格式**

创建、修改和删除二级分区的语法格式请参考：开发者指南->SQL语法参考->SQL语法->[CREATE TABLE](../%E5%BC%80%E5%8F%91%E8%80%85%E6%8C%87%E5%8D%97/CREATE-TABLE-SUBPARTITION.md)  SUBPARTITION、[ALTER TABLE S](../%E5%BC%80%E5%8F%91%E8%80%85%E6%8C%87%E5%8D%97/ALTER-SUBSCRIPTION.md)UBPARTITION。

用户也可以使用如下命令在vsql客户端中查询相关SQL语法的使用帮助信息。

\h create table subpartition

\h alter table subpartition

​

【说明】

二级分区的增删改查：对二级分区表插入数据的语法与普通表的语法没有任何差别。当向二级分区表插入数据时，会根据一级分区和二级分区规则，把数据分布到匹配的二级分区中进行存储。

以下列举了部分与二级分区相关的常用SQL句式。

* 新增一级与二级分区：

ALTER TABLE table\_name ADD partition\_desc [ ( subpartition\_desc [, ... ] ) ]

* 删除指定一级分区包括属于它的所有二级分区：

ALTER TABLE table\_name DROP PARTITION { partition\_name | { FOR （ value [ , … ] ） }

* 为指定一级分区新增二级分区：

ALTER TABLE table\_name MODIFY PARTITION partition\_name ADD subpartition\_desc

* 删除指定二级分区：

ALTER TABLE table\_name DROP SUBPARTITION { subpartition\_name | { FOR （ value [ , … ] ） }

* 删除数据：

DELETE FROM table\_name [ PARTITION { ( partition\_name ) | { FOR ( value [ , ... ] ) } } ] [ WHERE ...]

DELETE FROM table\_name [ SUBPARTITION { ( partition\_name ) | { FOR ( value [ , ... ] ) } } ] [ WHERE ...]

* 更新数据：

UPDATE table\_name [ PARTITION { ( partition\_name ) | { FOR ( value [ , ... ] ) } } ] SET [ WHERE ...]

UPDATE table\_name [ SUBPARTITION { ( partition\_name ) | { FOR ( value [ , ... ] ) } } ] SET [ WHERE ...]

* 在指定二级分区查找数据：

SELECT ... FROM table\_name SUBPARTITION (subpartition\_name)

* 单独查询某个分区或子分区数据：

SELECT ... FROM table\_name PARTITION { ( partition\_name ) | FOR ( value [ , ... ] ) }

SELECT ... FROM table\_name SUBPARTITION { ( subpartition\_name ) | {FOR ( value [, … ] ) }

* 当查询二级分区表的数据时（不能指定分区或子分区），如果查询条件中包含有分区键的条件，则在生成执行计划时会根据分区键条件过滤掉不符合条件的分区或子分区：

EXPLAIN SELECT ... FROM table\_name WHERE ...

**参数说明**

* **table\_name**

表名。

* **partition\_name**

分区表的名称。

* **subpartition\_name**

二级分区表的名称。

* **SUBPARTITION BY [RANGE | LIST | HASH ]**

根据column\_name指定的字段进行二级分区，分区类型可以是RANGE、LIST、 HASH其一。

* **SUBPARTITION TEMPLATE ( subpartition\_desc [, … ])**

常规子分区模板定义语法，适用于Range/List/Hash子分区，当分区定义中没有指定子分区的定义时会根据子分区模板自动生成二级分区。

若没有提供子分区模板，同时partition\_desc也不指定subpartition\_desc，则将创建一个默认子分区。

* **hash\_subpartition\_by\_quantity**

Hash分区持有的子分区模板定义语法（也可用与子分区定义语法），指定创建N个Hash子分区，同时可选指定Hash子分区的tablespace名列表。

**注意事项**

interval分区不能作为二级分区。

**示例**

**示例1：**二级分区的创建，修改和删除。

1、创建一个分区表并插入数据。

create table t\_part\_list\_range

( id number not null,

partition\_key int,

subpartition\_key int,

col2 varchar2(10)

)

partition by list(partition\_key)

subpartition by range(subpartition\_key)

(

partition t\_partition\_01 values (100)

(subpartition sub\_1\_1 values less than (10),

subpartition sub\_1\_2 values less than (20)

),

partition t\_partition\_02 values (200)

(subpartition sub\_2\_1 values less than (10),

subpartition sub\_2\_2 values less than (20)

)

);

insert into t\_part\_list\_range values(1,100,5,'sub\_1\_1');

insert into t\_part\_list\_range values(2,100,15,'sub\_1\_2');

insert into t\_part\_list\_range values(3,200,5,'sub\_2\_1');

insert into t\_part\_list\_range values(4,200,15,'sub\_2\_2');

insert into t\_part\_list\_range values(5,200,16,'sub\_2\_2');

select \* from t\_part\_list\_range subpartition for (100,5);

2、新增一级与二级分区。

alter table t\_part\_list\_range add partition t\_partition\_03 values (300)

( subpartition sub\_3\_1 values less than (10),

subpartition sub\_3\_2 values less than (20)

);

3、删除指定一级分区包括属于它的所有二级分区。

alter table t\_part\_list\_range drop partition t\_partition\_02;

4、为指定一级分区新增二级分区。

alter table t\_part\_list\_range modify partition t\_partition\_01 add subpartition sub\_1\_3 values less than (30);

5、删除指定二级分区。

alter table t\_part\_list\_range drop subpartition sub\_1\_3;

**示例2：**一级分区为interval分区，二级分区为list分区，分区键类型为字符类型；创建并查看分区结果。

1、创建测试表，包含一级分区和二级分区。

CREATE TABLE t\_subpartition\_interval\_list(

partition\_key date,

subpartition\_key varchar(20),

test varchar(20)

)partition by range(partition\_key)

interval('12 month')

subpartition by list(subpartition\_key)

(partition partition\_p1 VALUES LESS THAN ('2019-01-01')

(subpartition sub\_1\_1 values('test1'),

subpartition sub\_1\_2 values('test2'),

subpartition sub\_1\_3 values (default)

),

partition partition\_p2 VALUES LESS THAN ('2021-01-01')

(subpartition sub\_2\_1 values('test1'),

subpartition sub\_2\_2 values('test2'),

subpartition sub\_2\_3 values (default)

)

);

2、插入测试数据。

INSERT INTO t\_subpartition\_interval\_list VALUES('2018-09-02','test','test1');

INSERT INTO t\_subpartition\_interval\_list VALUES('2020-12-12','test2','test2');

INSERT INTO t\_subpartition\_interval\_list VALUES('2022-02-02','interval','test3');

3、查询已有分区对应数据。

select \* from t\_subpartition\_interval\_list subpartition(sub\_1\_3);

select \* from t\_subpartition\_interval\_list subpartition(sub\_2\_2);

返回结果为：

  partition\_key    | subpartition\_key | test

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

2018-09-02 00:00:00 | test             | test1

(1 row)

​

  partition\_key    | subpartition\_key | test

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

2020-12-12 00:00:00 | test2            | test2

(1 row)

4、查询表新增分区名。

select relname,parttype,partstrategy,boundaries

from pg\_partition

where parentid = (select oid from pg\_class where relname = 't\_subpartition\_interval\_list')

union all

select relname,parttype,partstrategy,boundaries from pg\_partition

where parentid in (

select oid from pg\_partition

where parentid = (

select oid from pg\_class where relname ='t\_subpartition\_interval\_list'

)

order by relname)

order by relname;

返回结果为：

          relname            | parttype | partstrategy |       boundaries

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

partition\_p1                 | p        | r            | {2019-01-01}

partition\_p2                 | p        | r            | {2021-01-01}

sub\_1\_1                      | s        | l            | {test1}

sub\_1\_2                      | s        | l            | {test2}

sub\_1\_3                      | s        | l            |

sub\_2\_1                      | s        | l            | {test1}

sub\_2\_2                      | s        | l            | {test2}

sub\_2\_3                      | s        | l            |

sys\_p1                       | p        | i            | {"2023-01-01 00:00:00"}

sys\_p1\_subpartdefault1       | s        | l            |

t\_subpartition\_interval\_list | r        | i            |

(11 rows)

5、查询新增分区对应数据。

select \* from t\_subpartition\_interval\_list subpartition(sys\_p1\_subpartdefault1);

返回结果为：

  partition\_key    | subpartition\_key | test

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

2022-02-02 00:00:00 | interval         | test3

(1 row)

### 事件触发器

**功能描述**

事件触发器是一个数据库范围内可见的对象，用来捕捉某些DDL操作，只要指定的事件发生并且该触发器相关的 WHEN 条件（如果有）被满足，该触发器的函数将被执行。

**特性支持**

* 支持使用C、C++或是 pgsql来编写一个事件触发器，但不能使用 SQL 语言函数编写。
* 支持on schema限定条件。
* 支持Logon和Logoff系统事件。
* 支持startup/shutdown系统事件。
* 支持ddl\_command\_start 事件。在 CREATE、ALTER、DROP、SECURITY LABEL、COMMENT、GRANT或者 REVOKE命令的执行之前触发。在事件触发器引发前不会做受影响对象是否存在的检查。不过，一个例外是，这个事件不会被目标是共享对象数据库、角色以及表空间的 DDL命令触发，也不会被目标是事件触发器的 DDL 命令触发。事件触发器机制不支持这些对象类型。ddl\_command\_start 也会在 SELECT INTO 命令的执行之前触发，因为这等价于CREATE TABLE AS。
* 支持ddl\_command\_end 事件。在同一组命令的执行之后触发。为了得到发生的DDL操作的更多细节，可以从ddl\_command\_end 事件触发器代码中使用集合返回函数pg\_event\_trigger\_ddl\_comands()。注意该触发器是在那些动作已经发生之后（但是在事务提交前）引发，并且因此系统目录会被读作已更改。
* 支持sql\_drop 事件。为任何删除数据库对象的操作在 ddl\_command\_end 事件触发器之前触发。要列出已经被删除的对象，可以从 sql\_drop 事件触发器代码中使用集合返回函数pg\_event\_trigger\_dropped\_objects()。注意该触发器是在对象已经从系统目录删除以后执行因此不能再查看它们。
* 支持table\_rewrite 事件。在表被命令ALTER TABLE和ALTER TYPE的某些动作重写之前触发。虽然其他控制语句，例如 CLUSTER 和 VACUUM也可以用来重写表，但是它们不会触发table\_rewrite 事件。

**创建事件触发器**

**语法格式**

CREATE EVENT TRIGGER name

ON event [ ON DATABASE | SCHEMA ]

[ WHEN filter\_variable IN (filter\_value [, ... ]) [ AND ... ] ]

EXECUTE PROCEDURE function\_name()

CREATE EVENT TRIGGER name

after Logon|startup ON datbase

EXECUTE PROCEDURE function\_name()

CREATE EVENT TRIGGER name

before Logoff | Shutdown ON datbase

EXECUTE PROCEDURE function\_name()

**参数说明**

* **name**

事件触发器名称。在该数据库中这个名称必须唯一。

* **event**

会触发对给定函数调用的事件名称，分别为 ddl\_command\_start、ddl\_command\_end、sql\_drop、 table\_rewrite。

* **filter\_variablet**

用来过滤事件的变量名称。这可以用来限制触发器只为它支持的那一部分情况引发。当前唯一支持的 filter\_variable 是TAG。

* **filter\_value**

与该触发器要为其引发的 filter\_variable 相关联的一个值列表。对于TAG这表示一个命令标签列表（例如 DROP FUNCTION）。

* **fumction name**

一个用户提供的函数，它被声明为没有参数并且返回类型 event trigger。

**修改事件触发器**

**语法格式**

ALTER EVENT TRIGGER name DISABLE

ALTER EVENT TRIGGER name ENABLE [REPLICA ALWAYS ]

ALTER EVENT TRIGGER name OWNER TO {new\_owner | CURRENT\_USER |SESSION\_USER}

ALTER EVENT TRIGGER name RENAME TO new name

**参数说明**

* **name**

要修改的现有触发器的名称。

* **new\_owner**

该事件触发器的新拥有者的用户名。

* **new\_name**

该事件触发器的新名称。

* **DISABLE/ENABLE[REPLICA|ALWAYS]TRIGGER**

该参数用于配置事件事件触发器是否被触发。一个被禁用的触发器对系统来说仍然是可知的，但是当其触发事件发生时却不会执行它。

**删除事件触发器**

**语法格式**

DROP EVENT TRIGGER [IF EXISTS ]name [CASCADE | RESTRICT]

**参数说明**

* **IF EXISTS**

如果该事件触发器不存在则不会抛出一个错误，而是发出一个提示。

* **name**

要移除的事件触发器的名称。

* **CASCADE**

自动删除依赖于该触发器的对象，然后删除所有依赖于那些对象的对象。

* **RESTRICT**

如果有任何对象依赖于该触发器，则拒绝删除它。这是默认值。

**注意事项**

* 不能在一个中止的事务中执行事件触发器（其他函数也一样）。因此，如果一个 DDL 命令出现错误失败，将不会执行任何相关的 ddl\_command\_end 触发器。反过来，如果一个ddl\_command\_start 触发器出现错误失败，将不会引发进一步的事件触发器，并且不会尝试执行该命令本身。类似地，如果一个 ddl\_command\_end 触发器出现错误失败，DDL 命令的效果将被回滚，就像其他包含事务中止的情况中那样。
* 当一个事件上指定了多个事件触发器，那么当触发该事件相关的事件触发器时，会根据事件触发器的名称的字母序升序方式，来一一调用这些触发器。

**示例**

**示例1：**创建ddl\_command\_start事件触发器，指定多条命令触发。

1. 创建事件触发器函数。

CREATE OR REPLACE FUNCTION eth1()

RETURNS event\_trigger

LANGUAGE plpgsql

AS $$ BEGIN

RAISE NOTICE 'test\_event\_trigger: % %', tg\_event, tg\_tag;

END;

$$;

2、创建事件触发器。

CREATE EVENT TRIGGER estart ON ddl\_command\_start WHEN TAG IN ('CREATE TABLE','ALTER TABLE','DROP TABLE') EXECUTE PROCEDURE eth1();

3、执行事件触发器中指定命令CREATE TABLE。

CREATE TABLE t\_eventtr(id int);

结果返回如下：

NOTICE: test\_event\_trigger: ddl\_command\_start CREATE TABLE

CREATE TABLE

4、执行事件触发器中指定命令ALTER TABLE。

ALTER TABLE t\_eventtr RENAME TO t\_eventtr\_new;

结果返回如下：

NOTICE: test\_event\_trigger: ddl\_command\_start ALTER TABLE

ALTER TABLE

5、执行事件触发器中指定命令DROP TABLE。

DROP TABLE t\_eventtr\_new;

结果返回如下：

NOTICE: test\_event\_trigger: ddl\_command\_start DROP TABLE

DROP TABLE

**示例2：**创建ddl\_command\_end事件触发器。

1. 创建事件触发器函数。

CREATE OR REPLACE FUNCTION eth1()

RETURNS event\_trigger

LANGUAGE plpgsql

AS $$ BEGIN

RAISE NOTICE 'test\_event\_trigger: % %', tg\_event, tg\_tag;

END;

$$;

2、创建测试表t\_eventtr。

CREATE TABLE t\_eventtr(id int);

3、创建事件触发器。

CREATE EVENT TRIGGER eend ON ddl\_command\_end EXECUTE PROCEDURE eth1();

4、删除测试表t\_eventtr。

DROP TABLE t\_eventtr;

结果返回如下：

NOTICE: test\_event\_trigger: ddl\_command\_end DROP TABLE

DROP TABLE

**示例3：**创建sql\_drop事件触发器。

1. 创建事件触发器函数。

CREATE OR REPLACE FUNCTION test\_event\_trigger\_drop()

RETURNS event\_trigger LANGUAGE plpgsql AS $$

DECLARE

obj record;

BEGIN

FOR obj IN SELECT \* FROM pg\_event\_trigger\_dropped\_objects()

LOOP

RAISE NOTICE '% dropped object: % %.% %',

tg\_tag,

obj.object\_type,

obj.schema\_name,

obj.object\_name,

obj.object\_identity;

END LOOP;

END;

$$;

1. 创建测试表t\_eventtr。

CREATE TABLE t\_eventtr(id int);

3、创建事件触发器edrop。

CREATE EVENT TRIGGER edrop ON sql\_drop EXECUTE PROCEDURE test\_event\_trigger\_drop();

4、删除表t\_eventtr。

DROP TABLE t\_eventtr;

结果返回如下：

NOTICE: DROP TABLE dropped object: table public.t\_eventtr public.t\_eventtr

NOTICE: DROP TABLE dropped object: type public.t\_eventtr public.t\_eventtr

NOTICE: DROP TABLE dropped object: type public.\_t\_eventtr public.t\_eventtr[]

DROP TABLE

**示例4：**创建table\_rewrite事件触发器。

1、创建事件触发器函数。

CREATE OR REPLACE FUNCTION eth1()

RETURNS event\_trigger

LANGUAGE plpgsql

AS $$ BEGIN

RAISE NOTICE 'test\_event\_trigger: % %', tg\_event, tg\_tag;

END;

$$;

2、创建事件触发器。

CREATE EVENT TRIGGER erewrite ON table\_rewrite EXECUTE PROCEDURE eth1();

**示例5：**修改事件触发器。

1、创建事件触发器函数。

CREATE OR REPLACE FUNCTION eth1() RETURNS event\_trigger AS $$ BEGIN RAISE NOTICE 'test\_event\_trigger: % %',tg\_event,tg\_tag; END $$ LANGUAGE plpgsql;

2、创建事件触发器。

CREATE EVENT TRIGGER logon\_event\_trigger AFTER logon ON database EXECUTE PROCEDURE eth1();

3、执行以下语句修改事件触发器名称。

ALTER EVENT TRIGGER logon\_event\_trigger RENAME TO test\_eventtri\_new;

4、系统表中查询改名后的事件触发器。

SELECT \* FROM pg\_event\_trigger WHERE evtname = 'test\_eventtri\_new';

结果返回如下：

evtname | evtevent | evtowner | evtfoid | evtenabled | isschema | evttags

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

test\_eventtri\_new | logon | 10 | 18586 | O | f |

(1 row)

**示例6：**创建Logon/Logoff系统事件触发器。

1、创建测试表test2。

create table test2(id int,col timestamp);

2、创建触发器函数eth3()和eth4()。

CREATE OR REPLACE FUNCTION eth3()

RETURNS event\_trigger

LANGUAGE plpgsql

AS $$ BEGIN

insert into test2 values(1,systimestamp);

END;

$$;

CREATE OR REPLACE FUNCTION eth4()

RETURNS event\_trigger

LANGUAGE plpgsql

AS $$ BEGIN

insert into test2 values(2,systimestamp);

END;

$$;

3、创建Logon/Logoff事件触发器。

CREATE EVENT TRIGGER logon\_eventtri after Logon ON database EXECUTE PROCEDURE eth3();

CREATE EVENT TRIGGER logoff\_eventtri before Logoff ON database EXECUTE PROCEDURE eth4();

4、退出数据库重新登陆。

\q

退出后在命令行执行登陆命令。

vsql -r

5、查询test2中记录的Logon/Logoff事件。

SELECT \* FROM test2;

结果返回如下：

id | col

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

2 | 2023-02-16 11:39:30

1 | 2023-02-16 11:39:32

(2 rows)

## 安全性

### 安全审计

#### **审计密钥管理和加密运算**

**功能描述**

Vastbase G100支持审计使用硬件密码卡的密钥管理和加密运算，并产生审计数据。密钥管理和加密运算使用光电安辰密码卡实现。

**注意事项**

* 本功能仅在Vastbase G100 2.2 Build 12及以后版本支持使用。
* 本功能需要与光电安辰加密卡配合使用。

**背景信息**

审计的密码卡功能如下：

* 内置函数提供密码卡产生随机数的功能并增加审计。

函数名：cryptocard\_gen\_random()

语法格式：

cryptocard\_gen\_random(len)

审计类型：crypt\_operation

审计详情信息：generate random success/failed,algorithm:NULL

* 内置函数提供密码卡计算数据摘要的功能并增加审计。

函数名：cryptocard\_hash()

语法格式：

cryptocard\_hash(data\_1,algorithm)

审计类型：crypt\_operation

审计详情信息：hash operation success/failed,algorithm:sm3

* 内置函数提供密码卡生成数据签名的功能并增加审计。

函数名：cryptocard\_ecc\_sign()

语法格式：

cryptocard\_ecc\_sign(data\_2,keyidx.pucpwd)

审计类型：crypt\_operation

审计详情信息：data signature success/failed,algorithm:sm2

* 内置函数提供密码卡签名校验的功能并增加审计。

函数名：cryptocard\_ecc\_verify()

语法格式：

cryptocard\_ecc\_verify(data\_2,keyidx,pucpwd,rvalue,svalue)

审计类型：crypt\_operation

审计详情信息：signature verification success/failed,algorithm:sm2

* 内置函数提供密码卡数据加密的功能并增加审计。

函数名：cryptocard\_encrypt()

语法格式：

cryptocard\_encrypt(data\_3,algorithm\_2,key,iv)

审计类型：crypt\_operation

审计详情信息：encryption success/failed,algorithm:sm4

* 内置函数提供密码卡数据解密的功能并增加审计。

函数名：cryptocard\_decrypt()

语法格式：

cryptocard\_decrypt(data\_4,algorithm\_2,key,iv)

审计类型：crypt\_operation

审计详情信息：decryption success/failed,algorithm:sm4

* 新增密钥协商审计，SSL连接成功时，如果加密套件使用密钥协商则增加审计信息。

审计类型：crypt\_operation

审计详情信息：ECDH success,cipher info(密钥信息)

**参数说明**

* **len**

要产生的随机数的长度。

取值范围：1~1024，其中1和1024可取。

* **data\_1**

计算摘要的数据。

* **data\_2**

产生签名的数据。

取值范围：要求数据长度为32字节。

* **data\_3**

数据明文。

* **data\_4**

数据密文。

* **algorithm\_1**

算法名。

取值范围：目前仅支持sm3。

* **algorithm\_2**

加密算法名

取值范围：目前仅支持sm4。

* **keyidx**

密码卡内部签名密钥对的索引。

取值范围：目前仅支持sm2密钥对。

* **pucpwd**

签名密钥的私钥访问控制码。

取值范围：密码卡上创建密钥时，由用户指定的控制码。

* **rvaue**

数据签名中的r值，cryptocard\_ecc\_sign函数会输出此值。

* **svalue**

数据签名中的s值，cryptocard\_ecc\_sign函数会输出此值。

* **key**

密钥

取值范围：长度为16的字符串。

* **iv**

取值范围：长度为16的字符串。

**操作步骤**

**前置条件：**所在服务器已安装配置广电安辰密码卡。

1、将获取到的动态库so文件放至$GAUSSHOME/lib目录下。

2、修改配置配置文件参数。

vi $PGDATA/postgresql.conf

use\_crypto=on

3、启动数据库并登录数据库。

4、开启审计参数。

alter system set audit\_enabled=on

5、调用内置函数。

* 密码卡产生随机数。

select cryptocard\_gen\_random(2);

结果显示为：

cryptocard\_gen\_rasndom

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

4Kc=

(1 rows)

* 密码卡计算数据摘要。

select cryptocard\_hash(1234567,'sm3');

结果显示为：

cryptocard\_hash

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

axSfng41J17cDvSUGfEBA4S0N62Sc4hZjvWUm5Gwx2Y=

(1 rows)

* 密码卡生成数据签名。

【说明】

前提条件为：已通过密码卡创建签名密钥对，并记录密钥索引和签名私钥的私钥访问控制码。

select cryptocard\_ecc\_sign('abcdefghigklmnopabcdefghigklmnop','001','Aa123456');

结果显示为：

cryptocard\_ecc\_sign

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

R\_VALUE:cl6cqoorWCTc5qY2bPvUdhz7g30goKMB7YijDAlLhiU=,S\_VALUE:FBCg/IkG80wXuqzDbGiT+69C2wjpaB2NPCVc1R3FLMo=

* 密码卡签名校验。

select cryptocard\_ecc\_verify('abcdefghigklmnopabcdefghigklmnop','001','Aa123456','签名返回的R值','签名返回的S值');

【说明】

* 当cryptocard\_ecc\_verify函数入参中签名返回的R值，签名返回的S值与密码卡生成数据签名时调用的函数cryptocard\_ecc\_sign的返回的R\_VALUE、S\_VALUE后跟的值一一对应，则校验结果为t；
* 签名返回的R值、 签名返回的S值与cryptocard\_ecc\_sign的返回的R\_VALUE、S\_VALUE后跟的值不对应，则校验结果为f。
* 密码卡数据加密。

select cryptocard\_encrypt(1234567890,'sm4','1234567890123456','abcdefghigklmnop');

结果显示为：

cryptocard\_encrypt

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

4xf+A9ZI8I7WVQ==

(1 rows)

* 密码卡数据解密。

select cryptocard\_decrypt('加密得到的密文','sm4','1234567890123456','abcdefghigklmnop');

【说明】

加密得到的结果是指用加密卡进行数据加密，即使用cryptocard\_encrypt函数加密得到的结果。本例以明文为1234567890为例，加密后得到密文为4xf+A9ZI8I7WVQ==。

结果展示为：

cryptocard\_decrypt

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

1234567890

(1 rows)

6、查看审计日志

\x --列式展示查询结果

select \* from pg\_query\_audit(now()-0.001,now()) where type='crypt\_operation';

结果显示为：

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

time | 2023-02-23 11:22:17+08

type | crypt\_operation

result | ok

userid | 10

username | test

database | postgres

client\_conninfo | gs\_clean@::1@SSL\_ON

object\_name | 空

detail\_info | ECDH success,cipher info(ECDHE-ECDSA-AES128-GCM-SHA256)

node\_name | node1

thread\_id | 281378338527344@730437737484187

local\_port | 2044

remote\_port | 44670

privilege | (ACL\_NO\_RIGHTS)

-[ RECORD 2 ]---+--------------------------------------------------------

time | 2023-02-23 11:22:17+08

type | crypt\_operation

result | ok

userid | 10

username | test

database | vastbase

client\_conninfo | gs\_clean@::1@SSL\_ON

object\_name | 空

detail\_info | ECDH success,cipher info(ECDHE-ECDSA-AES128-GCM-SHA256)

node\_name | node1

thread\_id | 281378372212848@730437737506897

local\_port | 2044

remote\_port | 44672

privilege | (ACL\_NO\_RIGHTS)

-[ RECORD 3 ]---+--------------------------------------------------------

time | 2023-02-23 11:22:17+08

type | crypt\_operation

result | ok

userid | 10

username | test

database | postgres

client\_conninfo | gs\_clean@::1@SSL\_ON

object\_name | 空

detail\_info | ECDH success,cipher info(ECDHE-ECDSA-AES128-GCM-SHA256)

node\_name | node1

thread\_id | 281378338527344@730437737530533

local\_port | 2044

remote\_port | 44674

privilege | (ACL\_NO\_RIGHTS)

### 数据库启动及安装包完整性校验

**功能描述**

Vastbase G100支持对重要文件和重要路径进行安全性管理。

* 数据库启动时，对块和数据库参数文件进行检验，并在过程中进行日志回显。
* 对重要文件可以生成对应的MD5文件、提供MD5检验功能、提供恢复功能。

**语法格式**

./pkg\_integrity\_check.sh [OPTION]

其中[option]具体内容如下：

* 查看帮助信息。

-h|--help

* 基于生成的md5sum文件列表检测指定路径下文件的MD5值，查看重要文件是否被修改。

-c|--check MD5\_SUM\_FILE\_LIST

* 为指定路径生成md5sum文件列表。

-g|--generate SRCDIR

* 基于原始安装包和md5sum文件列表恢复指定路径下的重要文件。

-r|--restore INSTALLER MD5\_SUM\_FILE\_LIST

**参数说明**

* **MD5\_SUM\_FILE\_LIST**

存放生成的MD5文件的文件夹路径。

* **SRCDIR**

待生成MD5文件的对象路径。

* **INSTALLER**

原始安装包所在的路径。

**注意事项**

* 本功能仅在Vastbase G100 2.2 Build 12及以后版本支持使用。
* 原始安装包所在的路径要具体到tar包。

**示例**

1、启动数据库时进行块检测和数据库参数文件检验，并进行日志回显。

vb\_ctl start

部分启动日志回显为：

0 [BACKEND] LOG: the config file /home/vastbase/data/vastbase/postgresql.conf verify start.

2023-02-16 09:31:50.859 [unknown] [unknown] localhost 47423508904576 0[0:0#0] 0   [BACKEND] LOG: the config file /home/vastbase/data/vastbase/postgresql.conf ve rify success.

2、进入数据库安装路径即home/vastbase/local/vastbase目录，对指定路径（本例中即对/home/vastbase/local/vastbase/bin/路径）生成md5sum文件列表。

cd /home/vastbase/local/vastbase

./bin/pkg\_integrity\_check.sh -g bin/

3、查看是否对重要路径下的文件有md5校验功能。

./bin/pkg\_integrity\_check.sh -c bin/md5sumf

回显结果如下（部分展示）：

bin/alarmItem\_bak2.conf: OK

bin/alarmItem\_bak.conf: OK

bin/clusterdb: OK

bin/cluster\_guc.conf: OK

bin/collect\_debug\_info.sh: OK

bin/constant: OK

bin/createdb: OK

bin/createlang: OK

bin/createuser: OK

bin/dbmind/app/diagnosis/cluster/entry.py: OK

bin/dbmind/app/diagnosis/cluster/\_\_init\_\_.py: OK

...

4、对已生成的md5文件进行恢复。

【说明】

此处为了模拟文件丢失情况，修改了bin目录下某篇文件的文件名，用户需注意。

mv alarmItem.conf alarmItem\_bak.conf --用户需注意

./bin/pkg\_integrity\_check.sh -r /soft/vb\_212/vastbase-installer/Vastbase-G100-2.2\_Build12\(11698\)-centos\_7-x86\_64.tar.gz bin/md5sumf

回显结果为：

Restore will take time, please wait a few minutes.

Begin to decompress the package...

Begin to restore...

md5sum: bin/alarmItem.conf: No such file or directory

md5sum: WARNING: 1 listed file could not be read

Begin to recheck ...

line 1

line bin/alarmItem.conf:

md5sum: bin/alarmItem.conf: No such file or directory

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

​

 =>

9053245116a12ce4c8d98aa4a0b429d1 bin/alarmItem.conf

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

Successfully restored.

5、查看bin目录下情况：

ll bin/a\*

此时bin目录下，丢失的alarmItem.conf文件被成功恢复回来：

-rw-r----- 1 vastbase vastbase 7297 Feb 15 02:44 bin/alarmItem\_bak.conf

-rw-r----- 1 vastbase vastbase 7297 Feb 15 02:44 bin/alarmItem.conf

### 无过度损失的自动恢复

**功能描述**

Vastbase G100支持无过度损失的自动恢复，即当数据库由于GUC参数无效导致进程启动失败时，数据库可以通过单用户single模式登录进入维护模式，并通过该模式修改无效的guc参数。

数据库通过单用户single模式登录时，将模式信息记录到全局上下文中，后续对GUC参数校验时上报错误信息，进行正常执行不退出，修改完成后用户可手动退出。

**注意事项**

* 本特性仅支持在Vastbase G100 2.2 Build 12及以后版本使用。
* 数据库正常启动时，不能通过single单用户模式登录。
* 不可通过single单用户模式修改不支持alter system方式修改的参数。具体参数类别信息参考《[VASTDATA]Vastbase G100 V2.2 开发者指南》->GUC参数说明->[重设参数](%E9%87%8D%E8%AE%BE%E5%8F%82%E6%95%B0.md)的表1。

**操作步骤**

1、切换到数据库安装目录下的bin目录下，当安装为标准化安装时，路径如下：

cd /home/vastbase/local/vastbase/bin

2、使用单用户模式登录：

vastbase --single -D ;

3、修改系统参数：

alter system set parameter TO value;

4、退出当前编辑。

5、启动数据库。

vb\_ctl start

**示例**

**前提条件：**guc参数设置错误，数据库无法正常启动。

【说明】

本例以错误示例配置文件中password\_force\_alter = offff;为例。

1、查看数据库状态。

vb\_ctl status

数据库无法正常启动，此时状态为：

[2023-02-20 18:17:49.995][10800][][vb\_ctl]: vb\_ctl status,datadir is /home/vastbase/data/vastbase

no server running

2、切换进入数据库安装目录下使用单用户模式登陆数据库。

cd /home/vastbase/local/vastbase/bin

./vastbase --single -D /home/vastbase/data/vastbase --ignore\_system\_indexes=on --enable\_indexscan=off --enable\_bitmapscan=off vastbase

进入修改参数编辑模式：

[BACKEND] WARNING: parameter "password\_force\_alter" can not set to "offff"

​

PostgreSQL stand-alone backend 9.2.4

backend>

3、在编辑模式下修改password\_force\_alter参数的值。

alter system set password\_force\_alter to off;

4、ctrl+d键退出当前编辑模式。

5、启动数据库。

vb\_ctl start

数据库成功启动，回显的部分结果为：

[2023-02-20 18:22:33.135][10806][][vb\_ctl]:  done

[2023-02-20 18:22:33.135][10806][][vb\_ctl]: server started (/home/vastbase/data/vastbase)

## 备份与恢复

### 逻辑备份与恢复

#### vb\_dump

**背景信息**

vb\_dump是Vastbase用于导出数据库相关信息的工具，用户可以自定义导出一个数据库或其中的对象(模式、表、视图等），回收站对象除外。支持导出的数据库可以是默认数据库postgres，也可以是自定义数据库。

vb\_dump工具由安装Vastbase数据库的操作系统用户执行。

vb\_dump工具在进行数据导出时，其他用户可以访问Vastbase数据库(读或写）。

vb\_dump工具支持导出完整一致的数据。例如，T1时刻启动vb\_dump导出A数据库，那么导出数据结果将会是T1时刻A数据库的数据状态，T1时刻之后对A数据库的修改不会被导出。

vb\_dump工具在进行数据导出时生成的列不会被转储。

vb\_dump支持导出兼容v1版本数据库的文本格式文件。

vb\_dump支持将数据库信息导出至纯文本格式的SQL脚本文件或其他归档文件中。

* 纯文本格式的SQL脚本文件：包含将数据库恢复为其保存时的状态所需的SQL语句。通过gsql运行该SQL脚本文件，可以恢复数据库。即使在其他主机和其他数据库产品上，只要对SQL脚本文件稍作修改，也可以用来重建数据库。
* 归档格式文件：包含将数据库恢复为其保存时的状态所需的数据，可以是tar格式、目录归档格式或自定义归档格式，详见[导出文件格式](#导出文件格式)。该导出结果必须与vb\_restore配合使用来恢复数据库，vb\_restore工具在导入时，系统允许用户选择需要导入的内容，甚至可以在导入之前对等待导入的内容进行排序。

**主要功能**

vb\_dump可以创建四种不同的导出文件格式，通过[-F或者--format=]选项指定，具体如导出文件格式所示。

导出文件格式

| **格式名称** | **-F的参数值** | **说明** | **建议** | **对应导入工具** |
| --- | --- | --- | --- | --- |
| 纯文本格式 | p | 纯文本脚本文件包含SQL语句和命令。命令可以由vsql命令行终端程序执行，用于重新创建数据库对象并加载表数据。 | 小型数据库，一般推荐纯文本格式。 | 使用vsql工具恢复数据库对象前，可根据需要使用文本编辑器编辑纯文本导出文件。 |
| 自定义归档格式 | c | 一种二进制文件。支持从导出文件中恢复所有或所选数据库对象。 | 中型或大型数据库，推荐自定义归档格式。 | 使用vb\_restore可以选择要从自定义归档/目录归档/tar归档导出文件中导入相应的数据库对象。 |
| 目录归档格式 | d | 该格式会创建一个目录，该目录包含两类文件，一类是目录文件，另一类是每个表和blob对象对应的数据文件。 | - |
| tar归档格式 | t | tar归档文件支持从导出文件中恢复所有或所选数据库对象。tar归档格式不支持压缩且对于单独表大小应小于8GB。 | - |

【说明】

可以使用vb\_dump程序将文件压缩为目录归档或自定义归档导出文件，减少导出文件的大小。生成目录归档或自定义归档导出文件时，默认进行中等级别的压缩。vb\_dump程序无法压缩已归档导出文件。

**注意事项**

* 禁止修改-F c/d/t 格式导出的文件和内容，否则可能无法恢复成功。对于-F p 格式导出的文件，如有需要，可根据需要谨慎编辑导出文件。
* 为了保证数据一致性和完整性，vb\_dump会对需要转储的表设置共享锁。如果表在别的事务中设置了共享锁，vb\_dump会等待锁释放后锁定表。如果无法在指定时间内锁定某个表，转储会失败。用户可以通过指定--lock-wait-timeout选项，自定义等待锁超时时间。
* 不支持加密导出存储过程和函数。

**语法格式**

vb\_dump [OPTION]... [DBNAME]

【说明】

可以使用vb\_dump程序将文件压缩为目录归档或自定义归档导出文件，减少导出文件的大小。生成目录归档或自定义归档导出文件时，默认进行中等级别的压缩。vb\_dump程序无法压缩已归档导出文件。

**参数说明**

**DBNANE**

指定要连接的数据库。“DBNAME”前面不需要加短或长选项。

* 方式一：不需要-d，直接指定“DBNAME”。

vb\_dump -p port\_number vastbase -f dump1.sql

* 方式二：

export PGDATABASE=vastbase

vb\_dump -p port\_number -f dump1.sql

环境变量： PGDATABASE

**通用参数**

* -f, --file=FILENAME：将输出发送至指定文件或目录。如果省略该参数，则使用标准输出。如果输出格式为(-F c/-F d/-F t)时，必须指定-f参数。如果-f的参数值含有目录，要求当前用户对该目录具有读写权限，并且不能指定已有目录。
* -F, --format=c|d|t|p：选择输出格式。格式如下：
* p|plain：输出一个文本SQL脚本文件(默认）。
* c|custom：输出一个自定义格式的归档，并且以目录形式输出，作为vb\_restore输入信息。该格式是最灵活的输出格式，因为能手动选择，而且能在恢复过程中将归档项重新排序。该格式默认状态下会被压缩。
* d|directory：该格式会创建一个目录，该目录包含两类文件，一类是目录文件，另一类是每个表和blob对象对应的数据文件。
* t|tar：输出一个tar格式的归档形式，作为vb\_restore输入信息。tar格式与目录格式兼容；tar格式归档形式在提取过程中会生成一个有效的目录格式归档形式。但是，tar格式不支持压缩且对于单独表有8GB的大小限制。此外，表数据项的相应排序在恢复过程中不能更改。
* -v, --verbose：指定verbose模式。该选项将导致vb\_dump向转储文件输出详细的对象注解和启动/停止次数，向标准错误流输出处理信息。
* -V, --version：打印vb\_dump版本，然后退出。
* -Z, --compress=0-9：指定使用的压缩比级别。
* 取值范围：0~9
* 0表示无压缩。
* 1表示压缩比最小，处理速度最快。
* 9表示压缩比最大，处理速度最慢。
* 针对自定义归档格式，该选项指定单个表数据片段的压缩，默认方式是以中等级别进行压缩。tar归档格式和纯文本格式目前不支持压缩。
* --lock-wait-timeout=TIMEOUT：请勿在转储刚开始时一直等待以获取共享表锁。如果无法在指定时间内锁定某个表，就选择失败。可以以任何符合SET statement\_timeout的格式指定超时时间。
* -?, --help：显示vb\_dump命令行参数帮助，然后退出。

**转储参数**

* -a, --data-only：只输出数据，不输出模式(数据定义)。转储表数据、大对象和序列值。
* -b, --blobs：该参数为扩展预留接口，不建议使用。
* -c, --clean：在将创建数据库对象的指令输出到备份文件之前，先将清理(删除）数据库对象的指令输出到备份文件中。(如果目标数据库中没有任何对象，gs\_restore工具可能会输出一些提示性的错误信息）该选项只对文本格式有意义。针对归档格式，可以调用gs\_restore时指定选项。
* -C, --create：备份文件以创建数据库和连接到创建的数据库的命令开始。(如果命令脚本是这种方式执行，可以先指定任意数据库用于执行创建数据库的命令，数据不会恢复到指定的数据库中，而是恢复到创建的数据库中。）该选项只对文本格式有意义。针对归档格式，可以在调用gs\_restore时指定选项。
* -E, --encoding=ENCODING：以指定的字符集编码创建转储。默认情况下，以数据库编码创建转储。(得到相同结果的另一个办法是将环境变量“PGCLIENTENCODING”设置为所需的转储编码。）
* -n, --schema=SCHEMA：只转储与模式名称匹配的模式，此选项包括模式本身和所有它包含的对象。如果该选项没有指定，所有在目标数据库中的非系统模式将会被转储。写入多个-n选项来选择多个模式。此外，根据vsql的d命令所使用的相同规则，模式参数可被理解成一个pattern，所以多个模式也可以通过在该pattern中写入通配符来选择。使用通配符时，注意给pattern打引号，防止shell扩展通配符。

【说明】

当-n已指定时，vb\_dump不会转储已选模式所附着的任何其他数据库对象。因此，无法保证某个指定模式的转储结果能够自行成功地储存到一个空数据库中。

当-n指定时，非模式对象不会被转储。

转储支持多个模式的转储。多次输入-n schemaname转储多个模式。

例如：

vb\_dump -h host\_name -p port\_number postgres -f backup/bkp\_shl2.sql -n sch1 -n sch2

在上面这个例子中，sch1和sch2会被转储。

* -N, --exclude-schema=SCHEMA：不转储任何与模式pattern匹配的模式。pattern将参照针对-n的相同规则来理解。可以通过输入多次-N，不转储与任何pattern匹配的模式。当同时输入-n和-N时，会转储与至少一个-n选项匹配、与-N选项不匹配的模式。如果有-N没有-n，则不转储常规转储中与-N匹配的模式。转储过程支持排除多个模式，输入-N exclude schema name排除多个模式。

例如：

vb\_dump -h host\_name -p port\_number postgres -f backup/bkp\_shl2.sql -N sch1 -N sch2

在上面这个例子中，sch1和sch2在转储过程中会被排除。

* -o, --oids：转储每个表的对象标识符(OIDs），作为表的一部分数据。该选项用于应用以某种方式参照了OID列的情况。如果不是以上这种情况，请勿使用该选项。
* -O, --no-owner：不输出设置对象的归属这样的命令，以匹配原始数据库。默认情况下，vb\_dump会发出ALTER OWNER或SET SESSION AUTHORIZATION语句设置所创建的数据库对象的归属。如果脚本正在运行，该语句不会执行成功，除非是由系统管理员触发(或是拥有脚本中所有对象的同一个用户）。通过指定-O，编写一个任何用户都能存储的脚本，且该脚本会授予该用户拥有所有对象的权限。该选项只对文本格式有意义。针对归档格式，可以在调用vb\_restore时指定选项。
* -s, --schema-only：只转储对象定义(模式），而非数据。
* -S, --sysadmin=NAME：该参数为扩展预留接口，不建议使用。
* -t, --table=TABLE：指定转储的表(或视图、或序列、或外表）对象列表，可以使用多个-t选项来选择多个表，也可以使用通配符指定多个表对象。当使用通配符指定多个表对象时，注意给pattern打引号，防止shell扩展通配符。当使用-t时，-n和-N没有任何效应，这是因为由-t选择的表的转储不受那些选项的影响。

【说明】

-t参数选项个数必须小于等于100。

如果-t参数选项个数大于100，建议使用参数--include-table-file来替换。

当-t已指定时，vb\_dump不会转储已选表所附着的任何其他数据库对象。因此，无法保证某个指定表的转储结果能够自行成功地储存到一个空数据库中。

-t tablename只转储在默认搜索路径中可见的表。-t \*.tablename转储数据库下所有模式下的tablename表。-t schema.table转储特定模式中的表。

-t tablename不会导出表上的触发器信息。

对于表名中包含大写字母的表，在使用-t参数指定导出时需对表名添加"来导出。如对于表"abC"，导出需指定-t "abC"；如对于表schema."abC"，导出需指定-t schema."abC"。

例如：

vb\_dump -h host\_name -p port\_number postgres -f backup/bkp\_shl2.sql -t schema1.table1 -t schema2.table2

在上面这个例子中，schema1.table1和schema2.table2会被转储。

* --include-table-file=FILENAME：指定需要dump的表文件。
* -T, --exclude-table=TABLE：不转储的表(或视图、或序列、或外表）对象列表，可以使用多个-T选项来选择多个表，也可以使用通配符指定多个表对象。当同时输入-t和-T时，会转储在-t列表中，而不在-T列表中的表对象。

例如：

vb\_dump -h host\_name -p port\_number postgres -f backup/bkp\_shl2.sql -T table1 -T table2

在上面这个例子中，table1和table2在转储过程中会被排除。

* --exclude-table-file=FILENAME：指定不需要dump的表文件。

【说明】

同--include-table-file，其内容格式如下：

schema1.table1

schema2.table2

......

* -x, --no-privileges|--no-acl：防止转储访问权限(授权/撤销命令）。
* -q, --target：指定导出兼容其他版本数据库的文本文件，目前支持v1和v5参数。v1参数用于导出v5数据库的数据为兼容v1的文本文件。v5参数用于导出v5数据库的数据为v5格式的文本文件，减少了导入v5时的可能的报错情况。在使用v1参数时，建议和--exclude-guc="enable\_cluster\_resize"，--exclude-function，--exclude-with等选项共用，否则导入到v1时可能报错。
* --exclude-guc：导出的文本文件中，不包括相关guc参数的set命令，目前只支持enable\_cluster\_resize。
* --exclude-function：不导出函数和存储过程。
* --exclude-with：导出的表定义，末尾不添加WITH(orientation=row，compression=on）这样的描述。
* --binary-upgrade：该参数为扩展预留接口，不建议使用。
* --binary-upgrade-usermap="USER1=USER2"：该参数为扩展预留接口，不建议使用。
* --column-inserts|--attribute-inserts：以INSERT命令带列名(INSERT INTO表(列、…）值…）方式导出数据。这会导致恢复缓慢。但是由于该选项会针对每行生成一个独立分开的命令，所以在重新加载某行时出现的错误只会导致那行丢失，而非整个表内容。
* --disable-dollar-quoting：该选项将禁止在函数体前使用美元符号$，并强制使用SQL标准字符串语法对其进行引用。
* --disable-triggers：该参数为扩展预留接口，不建议使用。
* --exclude-table-data=TABLE：指定不转储任何匹配表pattern的表这方面的数据。依照针对-t的相同规则理解该pattern。可多次输入--exclude-table-data来排除匹配任何pattern的表。当用户需要特定表的定义但不需要其中的数据时，这个选项很有帮助。排除数据库中所有表的数据，参见[--schema-only](#schema-only)。
* --inserts：发出INSERT命令(而非COPY命令）转储数据。这会导致恢复缓慢。但是由于该选项会针对每行生成一个独立分开的命令，所以在重新加载某行时出现的错误只会导致那行丢失，而非整个表内容。注意：如果重排列顺序，可能会导致整个恢复失败。列顺序改变时，--column-inserts选项不受影响，虽然会更慢。
* --no-publications：不转储发布。
* --no-security-labels：该参数为扩展预留接口，不建议使用。
* --no-subscriptions：不转储订阅。
* --no-tablespaces：不输出选择表空间的命令。使用该选项，无论默认表空间是哪个，在恢复过程中所有对象都会被创建。该选项只对文本格式有意义。针对归档格式，可以在调用gs\_restore时指定选项。
* --no-unlogged-table-data：该参数为扩展预留接口，不建议使用。
* --non-lock-table：该参数为扩展预留接口，不建议使用。
* --include-alter-table：转储表删除列。该选项会记录列的删除。
* --quote-all-identifiers：强制对所有标识符加引号。为了向后续版本迁移，且其中可能涉及引入额外关键词，在转储相应数据库时该选项会有帮助。
* --section=SECTION：指定已转储的名称区段(pre-data、data和post-data）。
* --serializable-deferrable：转储过程中使用可串行化事务，以确保所使用的快照与之后的数据库状态一致；要实现该操作需要在无异常状况的事务流中等待某个点，因为这样才能保证转储成功，避免引起其他事务出现serialization\_failure要重新再做。但是该选项对于灾难恢复没有益处。对于在原始数据库进行升级的时候，加载一个数据库的拷贝作为报告或其他只读加载共享的转储是有帮助的。没有这个选项，转储会反映一个与任何事务最终提交的序列化执行不一致的状态。如果当vb\_dump启动时，读写事务仍处于非活动状态，即便使用该选项也不会对其产生影响。如果读写事务处于活动状态，转储的开始时间可能会延迟一段不确定的时间。
* --use-set-session-authorization：输出符合SQL标准的SET SESSION AUTHORIZATION命令而不是ALTER OWNER命令来确定对象所有权。这样令转储更加符合标准，但是如果转储文件中的对象的历史有些问题，那么可能不能正确恢复。并且，使用SET SESSION AUTHORIZATION的转储需要数据库系统管理员的权限才能转储成功，而ALTER OWNER需要的权限则低得多。
* --with-encryption=AES128：指定转储数据需用AES128进行加密。
* --with-key=KEY
* AES128密钥规则如下：
* 密钥长度为8~16个字符。
* [至少包含大写字母(A-Z）、小写字母(a-z）、数字(0-9）、非字母数字字符(限定为~!@#$%^&\*()-\_=+|[{}];:,<.>/?）四类字符中的三类字符。](mailto:至少包含大写字母(A-Z）、小写字母(a-z）、数字(0-9）、非字母数字字符(限定为~!@#$%^&*()-_=+|[{}];:,<.>/?）四类字符中的三类字符。)

【说明】

* 使用vb\_dump工具进行加密导出时，仅支持plain格式导出。通过-F plain导出的数据，需要通过gsql工具进行导入，且如果以加密方式导入，在通过gsql导入时，需要指定--with-key参数。
* 不支持加密导出存储过程和函数。
* --with-salt=RANDVALUES：vb\_dumpall使用此参数传递随机值。
* --include-extensions：在转储中包含扩展。
* --include-depend-objs：备份结果包含依赖于指定对象的对象信息。该参数需要同-t/--include-table-file参数关联使用才会生效。
* --exclude-self：备份结果不包含指定对象自身的信息。该参数需要同-t/--include-table-file参数关联使用才会生效。
* --pipeline：使用管道传输密码，禁止在终端使用。
* --dont-overwrite-file：文本、tar以及自定义格式情况下会重写现有文件。这对目录格式不适用。

例如：

设想这样一种情景，即当前目录下backup.sql已存在。如果在输入命令中输入-f backup.sql选项时，当前目录恰好也生成backup.sql，文件就会被重写。

如果备份文件已存在，且输入--dont-overwrite-file选项，则会报告附带‘转储文件已经存在’信息的错误。

vb\_dump -p port\_number postgres -f backup.sql -F plain --dont-overwrite-file

* --table-conditions="[tablename; whereclause ]limitclause"

导出格式为纯文本格式时，仅导出部分符合条件的数据。

* 导出整个指定库/模式时，使用limit子句限制每张表的导出行数。limit n意为导出的每张表仅包含n行copy（insert）数据。
* 单表/多表导出时，使用where子句/limit子句限制表的导出条件或导出行数。参见[示例7](" \l "%E7%A4%BA%E4%BE%8B7)。
* whereclause用于指定导出条件，仅导出符合条件的结果集数据。
* limitclause用于限制导出行数，limit n意为导出（符合条件的）前n行。

【说明】

* 指定tablename时表示单表数据过滤条件，不指定tablename时表示多表过滤条件。
* 支持使用多个--table-conditions分别指定每张表的导出条件，
* 当指定了单表limit时，不支持再指定多表的limit条件。
* 支持的单表条件的筛选关键词包括：where、order、limit、offset、fetch。
* 支持的多表条件的筛选关键词包括：limit、offset、fetch。
* 当指定了-o选项后，对于有OIDS的表，将会忽略--table-conditions中的条件并导出全部表数据。

【说明】

* -s --schema-only和-a --data-only不能同时使用。
* -c --clean和-a --data-only不能同时使用。
* --inserts --column-inserts和-o --oids不能同时使用，因为INSERT命令不能设置OIDS。
* --role和--rolepassword必须一起使用。
* --binary-upgrade-usermap和--binary-upgrade必须一起使用。
* --include-depend-objs --exclude-self需要同-t --include-table-file参数关联使用才会生效
* --exclude-self必须同--include-depend-objs一起使用。

**连接参数**

* -h, --host=HOSTNAME：指定主机名称。如果数值以斜杠开头，则被用作到Unix域套接字的路径。缺省从PGHOST环境变量中获取(如果已设置），否则，尝试一个Unix域套接字连接。该参数只针对Vastbase外，对Vastbase内本机只能用127.0.0.1。
* 例如：主机名
* 环境变量：PGHOST
* -p, --port=PORT：指定主机端口。在开启线程池情况下，建议使用 pooler port，即主机端口+1。
* 环境变量：PGPORT
* -U, --username=NAME：指定所连接主机的用户名。不指定连接主机的用户名时，用户默认系统管理员。
* 环境变量：PGUSER
* -w, --no-password：不出现输入密码提示。如果主机要求密码认证并且密码没有通过其它形式给出，则连接尝试将会失败。 该选项在批量工作和不存在用户输入密码的脚本中很有帮助。
* -W, --password=PASSWORD：指定用户连接的密码。如果主机的认证策略是trust，则不会对系统管理员进行密码验证，即无需输入-W选项；如果没有-W选项，并且不是系统管理员，“Dump Restore工具”会提示用户输入密码。
* --role=ROLENAME：指定创建转储使用的角色名。选择该选项，会使vb\_dump连接数据库后，发起一个SET ROLE角色名命令。当所授权用户(由-U指定）没有vb\_dump要求的权限时，该选项会起到作用，即切换到具备相应权限的角色。某些安装操作规定不允许直接以超系统管理员身份登录，而使用该选项能够在不违反该规定的情况下完成转储。
* --rolepassword=ROLEPASSWORD：指定角色名的密码。

**说明**

如果Vastbase有任何本地数据要添加到template1数据库，请谨慎将vb\_dump的输出恢复到一个真正的空数据库中，否则可能会因为被添加对象的定义被复制，出现错误。要创建一个无本地添加的空数据库，需从template0而非template1复制，例如：

CREATE DATABASE foo WITH TEMPLATE template0;

tar归档形式的文件大小不得超过8GB(tar文件格式的固有限制）。tar文档整体大小和任何其他输出格式没有限制，操作系统可能对此有要求。

由vb\_dump生成的转储文件不包含优化程序用来做执行计划决定的统计数据。因此，最好从某转储文件恢复之后运行ANALYZE以确保最佳效果。转储文件不包含任何ALTER DATABASE…SET命令，这些设置由vb\_dumpall转储，还有数据库用户和其他完成安装设置。

**示例**

【说明】

使用vb\_dump转储数据库为SQL文本文件或其它格式的操作，如下所示。

* 示例中“Bigdata@123”表示数据库用户密码；
* “MPPDB\_backup.sql”表示导出的文件；
* “5432”表示数据库服务器端口；
* “postgres”表示要访问的数据库名。
* 导出操作时，请确保该目录存在并且当前的操作系统用户对其具有读写权限。

**示例1：**执行vb\_dump，导出postgres数据库全量信息，导出的MPPDB\_backup.sql文件格式为纯文本格式。

vb\_dump -U vastbase -W Bigdata@123 -f MPPDB\_backup.sql -p 5432 postgres -F p

vb\_dump[port='5432'][postgres][2022-08-03 11:45:15]: The total objects number is 440.

vb\_dump[port='5432'][postgres][2022-08-03 11:45:15]: [100.00%] 440 objects have been dumped.

vb\_dump[port='5432'][postgres][2022-08-03 11:45:15]: dump database postgres successfully

vb\_dump[port='5432'][postgres][2022-08-03 11:45:15]: total time: 941 ms

使用vsql程序从纯文本导出文件中导入数据。

**示例2：**执行vb\_dump，导出postgres数据库全量信息，导出的MPPDB\_backup.tar文件格式为tar格式。

vb\_dump -U vastbase -W Bigdata@123 -f MPPDB\_backup.tar -p 5432 postgres -F t

vb\_dump[port='5432'][postgres][2022-08-03 11:47:37]: The total objects number is 440.

vb\_dump[port='5432'][postgres][2022-08-03 11:47:37]: [100.00%] 440 objects have been dumped.

vb\_dump[port='5432'][postgres][2022-08-03 11:47:37]: dump database postgres successfully

vb\_dump[port='5432'][postgres][2022-08-03 11:47:37]: total time: 896 ms

**示例3：**执行vb\_dump，导出postgres数据库全量信息，导出的MPPDB\_backup.dmp文件格式为自定义归档格式。

vb\_dump -U vastbase -W Bigdata@123 -f MPPDB\_backup.dmp -p 5432 postgres -F c

vb\_dump[port='5432'][postgres][2022-08-03 11:48:13]: The total objects number is 440.

vb\_dump[port='5432'][postgres][2022-08-03 11:48:13]: [100.00%] 440 objects have been dumped.

vb\_dump[port='5432'][postgres][2022-08-03 11:48:13]: dump database postgres successfully

vb\_dump[port='5432'][postgres][2022-08-03 11:48:13]: total time: 899 ms

**示例4：**执行vb\_dump，导出postgres数据库全量信息，导出的MPPDB\_backup文件格式为目录格式。

vb\_dump -U vastbase -W Bigdata@123 -f MPPDB\_backup -p 5432 postgres -F d

vb\_dump[port='5432'][postgres][2022-08-03 11:48:51]: The total objects number is 440.

vb\_dump[port='5432'][postgres][2022-08-03 11:48:51]: [100.00%] 440 objects have been dumped.

vb\_dump[port='5432'][postgres][2022-08-03 11:48:51]: dump database postgres successfully

vb\_dump[port='5432'][postgres][2022-08-03 11:48:51]: total time: 893 ms

**示例5：**执行vb\_dump，导出postgres数据库信息，但不导出MPPDB\_temp.sql中指定的表信息。导出的MPPDB\_backup.sql文件格式为纯文本格式。

vb\_dump -U vastbase -W Bigdata@123 -p 5432 postgres --exclude-table-file=MPPDB\_temp.sql -f backup/MPPDB\_backup.sql

vb\_dump[port='5432'][postgres][2022-08-03 11:50:50]: The total objects number is 1367.

vb\_dump[port='5432'][postgres][2022-08-03 11:50:50]: [100.00%] 1367 objects have been dumped.

vb\_dump[port='5432'][postgres][2022-08-03 11:50:50]: dump database postgres successfully

vb\_dump[port='5432'][postgres][2022-08-03 11:50:50]: total time: 37017 ms

**示例6：**执行vb\_dump，仅导出依赖于指定表testtable的视图信息。然后创建新的testtable表，再恢复依赖其上的视图。

前置步骤：进入数据库创建测试表testtable。

c postgres

create table testtable(id int);

1、备份仅依赖于testtable的视图。

vb\_dump -s -p 5432 postgres -t PUBLIC.testtable --include-depend-objs --exclude-self -f backup/MPPDB\_backup.sql -F p

vb\_dump[port='5432'][vastbase][2022-08-03 11:58:23]: The total objects number is 432.

vb\_dump[port='5432'][vastbase][2022-08-03 11:58:23]: [100.00%] 432 objects have been dumped.

vb\_dump[port='5432'][vastbase][2022-08-03 11:58:23]: dump database vastbase successfully

vb\_dump[port='5432'][vastbase][2022-08-03 11:58:23]: total time: 1014 ms

2、修改testtable名称。

vsql -p 5432 postgres -r -c "ALTER TABLE PUBLIC.testtable RENAME TO testtable\_bak;"

3、创建新的testtable表。

CREATE TABLE PUBLIC.testtable(a int, b int, c int);

4、还原依赖于testtable的视图。

vsql -p 5432 postgres -r -f MPPDB\_backup.sql

**示例7：**使用--table conditions限定导出条件。

1、创建测试用数据库dbtest\_07。

create database dbtest\_06;

2、创建测试表table7。

create table t\_backup1(id int,col text);

3、插入测试数据。

insert into t\_backup1 values(1,'张三');

insert into t\_backup1 values(2,'李四');

insert into t\_backup1 values(3,'王五');

insert into t\_backup1 values(4,'赵六');

4、使用--table conditions限定导出条件，导出数据。

vb\_dump vastbase -t t\_backup1 --table-condition="t\_backup1; where id > 1 limit 1" -f backup1.sql  #命令1

vb\_dump vastbase -t t\_backup1 --table-condition="t\_backup1; where id > 1" -f backup2.sql  #命令2

vb\_dump vastbase -t t\_backup1 --table-condition="t\_backup1; limit 1" -f backup3.sql  #命令3

【说明】

命令1：导出id大于1的行，仅导出第一条符合要求的记录。

命令2：导出id大于1的行，不限制记录条数。

命令3：仅导出第一条符合要求的记录。

## 日志参考

### 审计日志

#### **审计日志加密存储**

**功能描述**

Vastbase G100支持对审计文件进行加密处理。

**注意事项**

* 本功能仅在Vastbase G100 2.2 Build 12及以后版本支持使用。
* 启动数据库输入audit\_master\_key、audit\_encrypt\_key密钥时区分大小写，应与初始化时设置的密钥一一对应。

**GUC参数说明**

* **audit\_encrypt\_algorithm**

审计日志加密参数，指定审计日志加密算法。

取值范围：加密算法必须为SM4。

* **audit\_master\_key**

主密钥参数，设置主密钥。

取值范围：初始化时设置主密钥长度必须等于16位。

* **audit\_encrypt\_key**

加密密钥参数，设置加密密钥。

取值范围：初始化时设置加密密钥长度必须等于16位。

**操作步骤**

1、初始化配置审计日志加密算法、主密钥和加密密钥，开启审计日志加密存储功能。

vb\_initdb -D ./data --audit\_encrypt\_algorithm=SM4 --audit\_master\_key="0123456789123456" --audit\_encrypt\_key="1111222233334444"

【说明】

./data是数据库数据目录，即环境变量$PGDATA的值。

2、数据库启动时输入主密钥以获取加密密钥，校验加密密钥正确则正常启动，启动过程提示语句如下：

[BACKEND] LOG: the config file /home/vastbase/data/vastbase\_test/postgresql.conf verify success.

please input the master key:

3、vsql连接数据库，开启审计参数设置审计表。

alter system set audit\_enabled=on;

alter system set audit\_dml\_state=1;

【说明】

audit\_enabled参数用于控制审计进程的开启和关闭，开启后审计信息会写入审计文件，详细信息参见《[VASTDATA]Vastbase G100 V2.2 开发者指南》->GUC参数说明->审计->[审计开关](../%E5%BC%80%E5%8F%91%E8%80%85%E6%8C%87%E5%8D%97/%E5%AE%A1%E8%AE%A1%E5%BC%80%E5%85%B3.md)。

audit\_dml\_state参数用于觉得是否对具体表的dml操作进行审计，取值为1时表示开启对dml操作的审计，详细信息参见《[VASTDATA]Vastbase G100 V2.2 开发者指南》->GUC参数说明->审计->[操作审计](../%E5%BC%80%E5%8F%91%E8%80%85%E6%8C%87%E5%8D%97/%E6%93%8D%E4%BD%9C%E5%AE%A1%E8%AE%A1.md)。

4、用户对表做dml操作。

5、使用pg\_query\_audit接口可查询审计日志。

select \* from pg\_query\_audit(now()-1,now());

【说明】

pg\_query\_audit()函数用于查看数据库主节点审计日志，详细信息参见《[VASTDATA]Vastbase G100 V2.2 开发者指南》->SQL语法参考->SQL基本要素->[安全函数](../%E5%BC%80%E5%8F%91%E8%80%85%E6%8C%87%E5%8D%97/%E5%AE%89%E5%85%A8%E5%87%BD%E6%95%B0.md)。

6、查看pg\_audit目录下加密存储的审计日志文件，如0\_adt（审计索引不进行加密存储）。

vi ./data/pg\_audit/0\_adt

【说明】

./data/pg\_audit/为审计文件的存储路径，用户可通过audit\_directory参数设置，详细信息参见《[VASTDATA]Vastbase G100 V2.2 开发者指南》->GUC参数说明->审计->[审计开关](../%E5%BC%80%E5%8F%91%E8%80%85%E6%8C%87%E5%8D%97/%E5%AE%A1%E8%AE%A1%E5%BC%80%E5%85%B3.md)。

#### **审计日志自动转存**

**功能描述**

Vastbase G100支持当审计日志空间满后审计日志自动转存备份的功能。当审计日志空间满且未开启停写策略（audit\_stop\_policy=off）时，自动将最早的审计文件转存，即将其从审计目录转移至转存目录。

**注意事项**

* 本功能仅在Vastbase G100 2.2 Build 12及以后版本支持使用。
* 对于不同的数据库实例，建议不要将审计日志转存目录配置为相同的目录，否则会造成转存审计日志数据混乱。

**背景信息**

* 与审计日志备份相关的参数参见[表1](" \l "table1)：

**表1** 审计日志相关配置参数和内置函数

| **配置项** | **配置项类别** | **含义** |
| --- | --- | --- |
| audit\_dump\_directory | 配置参数 | 用于设置审计日志自动转储目录。  默认值为空，设置路径应为绝对路径，否则报错。 |
| pg\_query\_audit\_dump() | 内置函数 | 用于查询转存日志。 |
| pg\_query\_audit\_info() | 内置函数 | 用于查询存储审计日志空间信息。 |

* 设置审计日志转存目录。

alter system set audit\_dump\_directory='audit\_dump\_dir'

【须知】

audit\_dump\_dir为用户自定义的转存审计日志的目录。

转存目录需要为绝对路径，否则报错。

转存目录在其他数据库实例目录下时，转存目录下无审计日志文件。

* 查询转存日志

select detail\_info,type from pg\_query\_audit\_dump(now()-1,now());

* 查询审计日志空间信息

select \*from pg\_query\_audit\_info();

**操作步骤**

**前提条件：**

1、开启审计参数，修改单个审计日志、审计空间大小。

alter system set audit\_enabled=on;

alter system set audit\_rotation\_size=1024;

alter system set audit\_space\_limit=4096;

2、开启dml操作审计。

alter system set audit\_dml\_state=1;

开启审计日志自动转存：

1、查看停写策略、转存目录。

show audit\_stop\_policy;

show audit\_dump\_directory;

【说明】

* 停写策略audit\_stop\_policy相关内容参见《[VASTDATA]Vastbase G100 V2.2 开发者指南》->GUC参数说明->审计->[审计开关](../%E5%BC%80%E5%8F%91%E8%80%85%E6%8C%87%E5%8D%97/%E5%AE%A1%E8%AE%A1%E5%BC%80%E5%85%B3.md)。
* 设置审计日志自动转储之前应该关闭停写策略。

查看结果为停写策略处于关闭状态，转存目录为空：

audit\_stop\_policy

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

off

(1 row)

​

audit\_dump\_directory

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

​

(1 row)

2、设置转存目录。

（1）创建转存目录。

mkdir /home/test/pg\_audit

（2）进入数据库客户端设置转存目录参数值。

alter system set audit\_dump\_directory='/home/test/pg\_audit';

（3）重启数据库

vb\_ctl restart

（4）连接进入vsql客户端工具后查看转存目录值。

show audit\_dump\_directory;

查询结果为：

audit\_dump\_directory

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

/home/vastbase/pg\_audit

(1 row)

3、查看第一条审计信息。

select \* from pg\_query\_audit\_dump('2022-12-31',now()) limit 1;

4、查看转存目录空间信息。

select \* from pg\_query\_audit\_info();

# 兼容性手册

## Oracle兼容性

### 函数

#### EMPTY\_CLOB

**功能描述**

EMPTY\_CLOB函数用于构造空的LOB供后续使用流插入。调用函数后将返回空的CLOB。

**语法格式**

empty\_clob()

**参数说明**

无。

**注意事项**

无。

**示例**

**示例1：**insert语句中使用empty\_clob函数。

1、创建测试表并插入数据。

create table oracle\_empty\_clob(a int,b clob);

insert into oracle\_empty\_clob values(1,empty\_clob());

2、查看插入结果。

select \* from oracle\_empty\_clob;

返回结果为：

a | b

---+---

1 |

(1 row)

**示例2：**在匿名块中使用empty\_clob函数。

1、设置serveroutput 为on（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）。

set serveroutput on;

2、创建匿名块，打印出判断empty\_clob函数的返回值是否为空的结果。

DECLARE

dummy clob;

BEGIN

dummy := empty\_clob();

IF dummy = empty\_clob() THEN

dbms\_output.put\_line( 'Dummy is empty' );

ELSE

dbms\_output.put\_line( 'Dummy is not empty' );

END IF;

END;

/

返回结果为：

Dummy is empty

ANONYMOUS BLOCK EXECUTE

**示例3：**在函数中使用empty\_clob函数。

1、创建函数。

CREATE OR REPLACE FUNCTION blob\_to\_varchar (blob\_in IN BLOB)

RETURN VARCHAR2

IS

v\_varchar VARCHAR2(4000);

v\_start PLS\_INTEGER := 1;

v\_buffer PLS\_INTEGER := 4000;

BEGIN

if DBMS\_LOB.GETLENGTH(blob\_in) is null then

return empty\_clob();

end if;

DBMS\_OUTPUT.put\_line('TEST:' || CEIL(DBMS\_LOB.GETLENGTH(blob\_in)));

FOR i IN 1..CEIL(DBMS\_LOB.GETLENGTH(blob\_in) / v\_buffer)

LOOP

v\_varchar := UTL\_RAW.CAST\_TO\_VARCHAR2(DBMS\_LOB.SUBSTR(blob\_in, v\_buffer, v\_start));

v\_start := v\_start + v\_buffer;

END LOOP;

RETURN v\_varchar;

end;

/

2、调用函数。

select blob\_to\_varchar(utl\_raw.cast\_to\_raw('abcd')) from dual;

select blob\_to\_varchar(utl\_raw.cast\_to\_raw('测试')) from dual;

select blob\_to\_varchar(utl\_raw.cast\_to\_raw(' ')) from dual;

select blob\_to\_varchar(null) from dual;

返回结果依次为：

TEST:4

blob\_to\_varchar

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

abcd

(1 row)

​

TEST:6

blob\_to\_varchar

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

测试

(1 row)

​

TEST:1

blob\_to\_varchar

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

​

(1 row)

​

blob\_to\_varchar

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

​

(1 row)

#### PG\_GET\_VIEWDEF

**功能描述**

Vastbase G100在Oracle或MySQL兼容模式下，支持保存和查看创建视图时的注释信息。

**语法格式**

pg\_get\_viewdef(iscreatesql\_bool,'view\_name')

pg\_get\_viewdef(iscreatesql\_bool,view\_id)

【说明】

pg\_get\_viewdef()函数用于为视图获取底层的SELECT命令，更多用法可参考《Vastbase G100 V2.2 开发者指南 》->SQL语法参考->SQL基本要素->函数和操作符->[系统信息函数](%E5%BC%80%E5%8F%91%E8%80%85%E6%8C%87%E5%8D%97/%E7%B3%BB%E7%BB%9F%E4%BF%A1%E6%81%AF%E5%87%BD%E6%95%B0.md)。

**参数说明**

* **view\_name**

视图名称。

* **viewid**

视图ID，可以通过如下语句从PG\_CLASS系统表中查询。

select oid from pg\_class where relname='view\_name';

* **iscreatesql\_bool**

是否使用用户创建视图时的sql。

* true：使用用户创建视图时的sql获取视图对象定义。
* false：使用原本的pg\_get\_viewdef函数获取视图对象定义。

**注意事项**

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MySQL的数据库db\_mysql下。

CREATE DATABASE db\_mysql dbcompatibility='B';

\c db\_mysql

2、创建测试表并插入测试数据。

CREATE TABLE products\_1133053(

ProductID int,

Name varchar,

Price int,

Quantity int);

INSERT INTO products\_1133053 values(101, 'Laptop', 15000, 100);

INSERT INTO products\_1133053 values(102, 'Desktop', 20000, 150);

INSERT INTO products\_1133053 values(104, 'Mobile', 3000, 200);

INSERT INTO products\_1133053 values(105, 'Tablet', 4000, 250);

3、创建带注释的物化视图。

CREATE MATERIALIZED VIEW view\_1133053 as

/\* THIS IS FOR TEST

zhongwenceshi 中文测试中文测试中文测试@####

&&\*9r20t80----------======+++====

\*/

SELECT \* FROM products\_1133053;

4、通过视图名称查看视图的定义。

SELECT pg\_get\_viewdef(true, 'view\_1133053');

返回结果如下，可以查看到定义视图时的完整注释。

              pg\_get\_viewdef

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

                                           +

/\* THIS IS FOR TEST                       +

zhongwenceshi 中文测试中文测试中文测试@####+

&&\*9r20t80----------======+++====         +

\*/                                         +

SELECT \* FROM products\_1133053;

(1 row)

5、查询视图的oid。

select oid from pg\_class where relname='view\_1133053';

返回结果如下：

oid

-------

18503

(1 row)

6、用上一步得到的视图ID查看视图的定义。

SELECT pg\_get\_viewdef(true,18503);

返回结果如下，可以查看到定义视图时的完整注释。

              pg\_get\_viewdef

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

                                           +

/\* THIS IS FOR TEST                       +

zhongwenceshi 中文测试中文测试中文测试@####+

&&\*9r20t80----------======+++====         +

\*/                                         +

SELECT \* FROM products\_1133053;

(1 row)

### 内置包

#### DBMS\_LOB

**功能描述**

DBMS\_LOB用于提供一组处理BLOB（二进制数据类型）、CLOB（字符数据类型）和[BFILE数据类型](BFILE%E6%95%B0%E6%8D%AE%E7%B1%BB%E5%9E%8B.md)（存储数据库外的大型对象，指向操作系统物理文件）的 PL/SQL 包。其中包括常量、函数、存储过程。Vastbase G100对该内置包的支持情况如下：

支持以下常量：

| **常量** | **值** |
| --- | --- |
| [LOBMAXSIZE](" \l "LOBMAXSIZE) | 1073741832 |
| [DEFAULT\_CSID](" \l "DEFAULT_CSID) | 0 |
| [DEFAULT\_LANG\_CTX](" \l "DEFAULT_LANG_CTX) | 0 |

支持以下函数：

| **函数** | **描述** |
| --- | --- |
| [GETLENGTH](" \l "LOBGETLENGTH) | 获取BLOB、CLOB和BFILE数据类型的长度。 |
| [FILEOPEN](" \l "LOBFILEOPEN) | 打开一个BFILE文件。 |
| [READ](" \l "LOBREAD) | 从一个BFILE文件中读取指定长度的内容。 |
| [FILECLOSE](" \l "FILECLOSE) | 关闭一个BFILE文件。 |
| [SUBSTR](" \l "SUBSTR) | 该函数用于返回 LOB 中从指定位置开始的部分内容，从指定的偏移量开始。 |
| [INSTR](" \l "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。 |

支持以下存储过程：

| **存储过程** | **描述** |
| --- | --- |
| [OPEN](" \l "OPEN) | 根据传入的参数打开对应的LOB对象。 |
| [COPY](" \l "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](" \l "WRITEAPPEND) | 将指定数据添加到一个大对象的末尾。 |
| [CREATETEMPORARY](" \l "CREATETEMPORARY) | 用来在临时表空间中创建临时BLOB或CLOB。 |
| [FREETEMPORARY](" \l "FREETEMPORARY) | 释放临时表空间中的临时BLOB或CLOB。 |
| [CONVERTTOBBLOB](" \l "CONVERTTOBBLOB) | 将CLOB从指定偏移开始的amount个字符转换到BLOB中，并返回转换后的偏移量。 |

**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。

* **LOBMAXSIZE**

**示例**

1、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。

set serveroutput on;

2、打印常量lobmaxsize的值。

declare

begin

dbms\_output.put\_line(dbms\_lob.lobmaxsize);

end;

/

返回结果为：

1073741823

ANONYMOUS BLOCK EXECUTE

* **DEFAULT\_CSID**

**示例**

1、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。

set serveroutput on;

2、打印常量default\_csid的值。

declare

begin

dbms\_output.put\_line(dbms\_lob.default\_csid);

end;

/

返回结果为：

0

ANONYMOUS BLOCK EXECUTE

* **DEFAULT\_LANG\_CTX**

**示例**

1、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。

set serveroutput on;

2、打印常量default\_lang\_ctx的值。

declare

begin

dbms\_output.put\_line(dbms\_lob.default\_lang\_ctx);

end;

/

返回结果为：

0

ANONYMOUS BLOCK EXECUTE

* **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。

**示例**

1、操作系统环境下创建文件，输入数据并保存。

vi /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。

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'));

4、设置serveroutput 为on（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）。

set serveroutput on;

5、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;

6、调用函数。

select f\_read\_bfile();

返回结果为：

file data:

张三

李四

​

f\_read\_bfile

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

​

(1 row)

* **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。

**示例**

参见dbms\_lob.fileopen的[示例](" \l "bfile%E6%93%8D%E4%BD%9C)。

* **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。

**示例**

参见dbms\_lob.fileopen的[示例](" \l "bfile%E6%93%8D%E4%BD%9C)。

* **SUBSTR**

**语法格式**

DBMS\_LOB.SUBSTR (

  lob\_loc    IN   BLOB,

  amount     IN   INTEGER := 32767,

  offset     IN   INTEGER := 1)

 RETURN RAW;

DBMS\_LOB.SUBSTR (

  lob\_loc    IN   CLOB   CHARACTER SET ANY\_CS,

  amount     IN   INTEGER := 32767,

  offset     IN   INTEGER := 1)

 RETURN VARCHAR2 CHARACTER SET lob\_loc%CHARSET;

DBMS\_LOB.INSTR (

  file\_loc   IN   BFILE,

  amount     IN   INTEGER := 32767,

  offset     IN   INTEGER := 1)

 RETURN RAW;

**参数说明**

* lob\_loc：LOB的locator。
* file\_loc：FILE的locator。
* amount：要读取的BLOB的字节数或CLOB的字符数。默认值为32767。
* offset：LOB从1开始的字节数（BLOB）或字符数（CLOB）偏移量。默认值为1。

**返回值**

* 传入BLOB或BFILE类型时返回RAW类型。
* 传入CLOB类型时返回VARCHAR2类型。
* 当任意传入值为NULL，或amount<1、amount>32767、offset<1、offset>LOBMAXSIZE时，返回NULL。

**示例**

调用SUBSTR函数。

select dbms\_lob.substr('ABCD'::clob);

select dbms\_lob.substr('ABCD');

select dbms\_lob.substr('this is lob function test',1,5) as a1;

select dbms\_lob.substr('this is lob function test',length('this is lob function test'),1) as a2;

返回结果为：

substr

--------

ABCD

(1 row)

​

substr

--------

ABCD

(1 row)

​

a1

----

​

(1 row)

​

          a2

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

this is lob function test

(1 row)

* **INSTR**

**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)

* **OPEN**

**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**

**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**

**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**

**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**

**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

* **CONVERTTOBBLOB**

**语法格式**

DBMS\_LOB.CONVERTTOBBOLB(

dest\_blob  IN OUT NOCOPY  BLOB,

src\_clob   IN     CLOB CHARACTER SET ANY\_CS,

amount     IN      INTEGER,

dest\_offset IN OUT INTEGER,

src\_offset  IN OUT INTEGER,

blob\_csid   IN     NUMBER,

lang\_context IN OUT INTEGER,

warning     OUT   INTEGER);

**参数说明**

* dest\_lob IN：表示一个BLOB类型的大对象。
* dest\_lob OUT：表示转换后的BLOB类型的大对象。
* src\_clob：表示一个CLOB类型的大对象。
* amount：表示在参数src\_clob所指定的大对象中要转换的字符数量。
* dest\_offset IN：BLOB类型的目标大对象中字节的位置。
* dest\_offset OUT：写操作完成后，在BLOB类型大对象中字节的位置。
* src\_offset IN：转换操作中，CLOB类型大对象开始的位置。
* src\_offset OUT：在转换操作完成后，CLOB类型大对象中字符的位置。
* blob\_csid：BLOB类型大对象中的字符集ID。

【说明】

0并不是真实存在的字符集ID，为0时表示使用源CLOB的字符集。暂不支持其它取值。

* lang\_context IN：在转换操作中使用的语言环境。默认值为0。
* lang\_context OUT：转换后的语言环境。
* warning：如果转换成功，则返回0；如果遇到不可转换的字符，则返回1。

**示例**

1、调用CONVERTTOBBLOB存储过程。

declare

b blob := utl\_raw.cast\_to\_raw('ABCDEFGH');

c clob := '测试123';

amount INTEGER := 4;

dest\_offset INTEGER := 1;

src\_offset INTEGER := 2;

csid INTEGER := 0;

lang\_ctx INTEGER := 0;

warning INTEGER;

begin

dbms\_lob.converttoblob(b,c,amount,dest\_offset,src\_offset,csid,lang\_ctx,warning);

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

dbms\_output.put\_line(dest\_offset||' ' ||src\_offset||' '||lang\_ctx||' '||warning);

end;

/

返回结果为：

试123GH

7 6 0 0

ANONYMOUS BLOCK EXECUTE

#### DBMS\_UTILITY

**功能描述**

DBMS\_UTILITY包提供了很多不同的子程序工具。该内置包包含以下子程序：

| **子程序** | **描述** |
| --- | --- |
| COMMA\_TO\_TABLE | 用于将逗号间隔的字符串转化成数组类型。 |
| TABLE\_TO\_COMMA | 用于将数组类型转换成逗号间隔的字符串。 |
| EXEC\_DDL\_STATEMENT | 用于执行DDL语句。 |
| FORMAT\_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 | 用于获得在该范围内字符串的hash值。 |
| GET\_TIME | 用于记录当前时刻的时间（单位是百分之一秒）。 |
| ANALYZE\_SCHEMA | 对指定schema中所有的表、列、索引列进行抽样估计并重建统计信息。 |

**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：
* DBMS\_UTILITY.LNAME\_ARRAY：对应varchar2[4000]。
* DBMS\_UTILITY.NAME\_ARRAY：对应varchar2[30]。
* DBMS\_UTILITY.UNCL\_ARRAY：对应varchar2[227]。
* 执行 DBMS\_UTILITY.ANALYZE\_SCHEMA的用户需要具有指定模式的 USAGE 权限以及该模式下的表的 VACUUM 权限。

**内置包子程序**

**COMMA\_TO\_TABLE**

**语法格式**

DBMS\_UTILITY.COMMA\_TO\_TABLE(

list in VARCHAR2,

tablen OUT BINARY\_INTEGER,

tab OUT varchar2[]);

**参数说明**

* **list**

VARCHAR2类型，要分隔的字符串。

* **tablen**

BINARY\_INTEGER类型，分隔后的数组长度。

* **tab**

VARCHAR2[]类型，分隔后的数组。

**注意事项**

* list必须是非空的逗号间隔符类型。字符串里面双引号中的逗号不计算在内。
* 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。

**示例**

直接调用COMMA\_TO\_TABLE函数将逗号间隔的字符串转化成数组类型。

select dbms\_utility.comma\_to\_table('a,b,c,d'::text);

返回结果为：

comma\_to\_table

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

(4,"{a,b,c,d}")

(1 row)

**TABLE\_TO\_COMMA**

**语法格式**

DBMS\_UTILITY.TABLE\_TO\_COMMA(

tab in varchar2[],

tablen OUT BINARY\_INTEGER,

list OUT VARCHAR2);

**参数说明**

* **tab**

VARCHAR2[]类型，输入的数组。

* **tablen**

BINARY\_INTEGER类型，输入的数组长度。

* **list**

VARCHAR2类型，形成的字符串。

**示例**

直接调用TABLE\_TO\_COMMA函数将数组类型转换成逗号间隔的字符串。

select dbms\_utility.table\_to\_comma(array['a','b','c','d']);

返回结果为：

table\_to\_comma

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

(4,"a,b,c,d")

(1 row)

**GET\_TIME**

**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。

**示例**

1、创建存储过程调用GET\_TIME获取当前时间值，用于计算操作的执行之间。

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(1);

perform pg\_sleep(1);

perform pg\_sleep(1);

perform pg\_sleep(1);

perform pg\_sleep(1);

finish\_time := DBMS\_UTILITY.get\_time();

elapsed\_time := finish\_time - start\_time;

return elapsed\_time;

END

$$ language plpgsql;

1. 调用函数。

select get\_runtime();

返回结果为：

get\_runtime

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

501

(1 row)

**GET\_HASH\_VALUE**

**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。

**示例**

直接调用GET\_HASH\_VALUE函数

select dbms\_utility.get\_hash\_value('afeaw',0,1000);

返回结果为：

get\_hash\_value

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

84

(1 row)

**EXEC\_DDL\_STATEMENT**

**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。

**示例**

1、创建存储过程调用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;

/

2、查询test\_eds的属性。

\dti test\_eds

返回结果为：

List of relations

Schema | Name | Type | Owner |Table| Storage

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

public | test\_eds | table|vastbase| | {orientation=row,compression=no,fillfactor=80}

(1 row)

**FORMAT\_CAL\_STACK**

**语法格式**

DBMS\_UTILITY.FORMAT\_CALL\_STACK

RETURN VARCHAR2;

**示例**

示例请参考EXEC\_DDL\_STATEMENT示例。

**ANALYZE\_SCHEMA**

**语法格式**

DBMS\_UTILITY.ANALYZE\_SCHEMA (

schema IN VARCHAR2,

method IN VARCHAR2,

estimate\_rows IN NUMBER DEFAULT NULL,

estimate\_percent IN NUMBER DEFAULT NULL,

method\_opt IN VARCHAR2 DEFAULT NULL);

**参数说明**

* **schema**

schema名称。

* **method**

操作。支持的取值如下：

* ESTIMATE：估计。
* COMPUTE：计算。
* DELETE：清空。

【说明】

当method为 ESTIMATE时，estimate\_rows或estimate\_percent中至少一个应该有值。

* **estimate\_rows**

估计统计信息所依据的行数。

* **estimate\_percent**

估计统计信息所依据的行百分比。

* **method\_opt**

要分析的对象类型。支持如下取值：

* FOR TABLE
* FOR ALL [INDEXED] COLUMNS [SIZE n]
* FOR ALL INDEXES

**示例**

1. 创建测试表并插入数据。

create table t\_system\_part\_01

(c1 integer,

c2 date,

c3 text)

partition by system

(

partition p1,

partition p2,

partition p3

);

insert into t\_system\_part\_01 partition(p1) values(1,'2022-01-01','p1');

insert into t\_system\_part\_01 partition(p2) values(2,'2022-02-01','p2');

insert into t\_system\_part\_01 partition(p3) values(3,'2022-03-01','p3');

1. 创建全局索引。

create unique index idx\_t\_system\_part\_48 on t\_system\_part\_01(c1) global;

1. 输出查询t\_system\_part\_01的执行计划。

explain select \* from t\_system\_part\_01;

返回结果为：

QUERY PLAN

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

Partition Iterator (cost=0.00..21.04 rows=1104 width=44)

Iterations: 3

-> Partitioned Seq Scan on t\_system\_part\_01 (cost=0.00..21.04 rows=1104 width=44)

Selected Partitions: 1..3

(4 rows)

4、使用ANALYZE\_SCHEMA存储过程对“public”中所有的表进行抽样估计并重建统计信息。

exec DBMS\_UTILITY.ANALYZE\_SCHEMA('public','ESTIMATE',null,90,'for table');

返回结果为：

NOTICE: PL/SQL procedure successfully completed.

analyze\_schema

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

(1 row)

5再次输出查询t\_system\_part\_01的执行计划。

explain select \* from t\_system\_part\_01;

返回结果如下，可见抽样估计之后查询计划中的此次扫描的成本，估计行数及大小都发生了变化。

QUERY PLAN

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

Partition Iterator (cost=0.00..3.03 rows=3 width=15)

Iterations: 3

-> Partitioned Seq Scan on t\_system\_part\_01 (cost=0.00..3.03 rows=3 width=15)

Selected Partitions: 1..3

(4 rows)

#### DBMS\_SESSION

**功能描述**

Vastbase G100在Oracle兼容模式下支持内置包DBMS\_SESSION的部分功能，该内置包提供了查询和设置会话相关状态的方法。

Vastbase G100支持该内置包的如下模块：

| **函数** | **描述** |
| --- | --- |
| modify\_package\_state | 用于重置当前会话中所有package的状态，重新初始化包。 |

**语法格式**

modify\_package\_state

function modify\_package\_state(

flag in TINYINT   --1or2

) return bool;

**参数说明**

**flag in**

用户指定的入参，类型为TINYINT。当入参为1，2时函数返回为true代表调用成功，其他情况下返回false。

**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。
* flag in传入内核中区分不同的逻辑处理，为2时性能更优。

**示例**

1、创建并切换至兼容模式为Oracle的数据库下。

create  database dbtest dbcompatibility 'A';

\c dbtest;

2、设置serveroutput 为on（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）。

set serveroutput =on;

3、创建包含全局变量v\_stack\_depth的pkg1。

create or replace package pkg1 is

v\_stack\_depth number;

function get\_stack\_depth return number;

end;

/

4、在包体初始化中对全局变量赋值，并打印初始化信息。

create or replace package body pkg1 is

function get\_stack\_depth return number

is

begin

v\_stack\_depth := v\_stack\_depth + 1;

return v\_stack\_depth;

end;

begin

v\_stack\_depth := 0;

dbms\_output.put\_line('初始化pkg1');

end;

/

5、连续调用包中函数，函数返回全局变量v\_stack\_depth的值。

select pkg1.get\_stack\_depth from dual;

select pkg1.get\_stack\_depth from dual;

返回结果依次为：

初始化pkg1

get\_stack\_depth

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

              1

(1 row)

​

get\_stack\_depth

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

              2

(1 row)

6、重置会话中所有package的初始化值。

call dbms\_session.modify\_package\_state(2);

返回结果为：

modify\_package\_state

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

t

(1 row)

7、检测是否重新初始化。（连续调用包中函数返回v\_stack\_depth的值。）

select pkg1.get\_stack\_depth from dual;

select pkg1.get\_stack\_depth from dual;

返回结果依次为：

初始化pkg1

get\_stack\_depth

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

              1

(1 row)

​

get\_stack\_depth

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

              2

(1 row)

成功使用内置包函数DBMS\_SESSION.modify\_package\_state重置函数入参为1。

### SQL语法

#### **CREATE 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，PACKAGE由两部分组成：包（PACKAGE）和包体（PACKAGE BODY）组成。

包头是包的说明部分，是对外的操作接口，对应用是可见的；包体是包的代码和实现部分，对应用来说是不可见的。

Vastbase G100在Oracle兼容性模式下，CREATE PACKAGE支持编译控制选项pragma serially\_reusable，可以将package设定为连续可复用包，此选项可以使得对应package分配的内存在调用后被释放，复用package时包内变量都被初始化为其默认值。

当包体存在初始化块时，Vastbase G100支持不同包体的初始化块/包体函数进行相互调用。

**语法格式**

CREATE [OR REPLACE] PACKAGE BODY [schema\_name.]package\_name IS | AS

declarations

implementations;

PRAGMA SERIALLY\_REUSABLE;

[BEGIN EXCEPTION]

END;

**参数说明**

* **schema\_name**

已经存在模式名称。

* **package\_name**

自定义包体名称。

* **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。

* **PRAGMA**

package编译控制选项标记。

* **SERIALLY\_REUSABLE**

编译控制选项，将package设定为连续可复用包。

**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**

* CREATE PACKAGE BODY示例请参考CREATE PACKAGE[示例1](CREATE-PACKAGE.md" \l "%E5%88%9B%E5%BB%BA%E5%8C%85)。
* 带有编译选项pragma serially\_reusable的CREATE PACKAGE BODY示例请参考CREATE PACKAGE[示例10](CREATE-PACKAGE.md" \l "example10)。

**前置条件：**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

**示例1：**package在初始块中互相调用变量。

1、创建包pkg1。

CREATE OR REPLACE PACKAGE pkg1 is

--创建共有函数，可以被调用

function pkg1\_fun(i int) return int;

--公有变量

id1 int:= 1;

end pkg1;

/

2、创建包pkg2。

CREATE OR REPLACE PACKAGE pkg2 is

function pkg2\_fun(i number) return number;

id2 int:= 1;

end pkg2;

/

3、给pkg1包创建包体实现pkg1\_fun函数，调用pkg2包中定义的变量id2。

CREATE OR REPLACE PACKAGE BODY pkg1 is

function pkg1\_fun(i int) return int is

res int;

begin

res := pkg2.id2;  --调用pkg2.id2的值赋予res

return res;

end;

end pkg1;

/

4、给pkg2包创建包体实现pkg2\_fun函数。

CREATE OR REPLACE PACKAGE BODY pkg2 is

function pkg2\_fun(i number) return number is

res int;

begin

res := pkg1.id1; --调用pkg1.id1的值赋予res

return res;

end;

begin

id2 := 20;  --给id2赋值20

end pkg2;

/

5、调用包函数可见pkg1\_fun函数调用了变量id2。

select pkg1.pkg1\_fun(1) from dual;

返回结果为：

pkg1\_fun

--------

     20

(1 row)

示例2：package包函数互相调用包变量，初始化块互相调用包函数。

1、创建包pkg1。

CREATE OR REPLACE PACKAGE pkg1 is

function pkg1\_fun1(i int) return int;

function pkg1\_fun2(i int) return int;

id1 int:= 1;

end pkg1;

/

2、创建包pkg2。

CREATE OR REPLACE PACKAGE pkg2 is

function pkg2\_fun1(i number) return number;

function pkg2\_fun2(i numeric) return numeric;

id2 int:= 2;

end pkg2;

/

3、创建包体pkg1，调用包pkg2中定义的函数pkg2\_fun2。

CREATE OR REPLACE PACKAGE BODY pkg1 is

function pkg1\_fun1(i int) return int is

res int;

begin

res := pkg2.id2;

return res;

end;

function pkg1\_fun2(i int) return int is

res int;

begin

res := i;

return res;

end;

begin

id1 := pkg2.pkg2\_fun2(5);  --调用包pkg2中定义的函数pkg2\_fun2

end pkg1;

/

4、创建包体pkg2，调用包pkg1中定义的函数pkg1\_fun2。

CREATE OR REPLACE PACKAGE BODY pkg2 is

function pkg2\_fun1(i number) return number is

res int;

begin

res := pkg1.id1;

return res;

end;

function pkg2\_fun2(i numeric) return numeric is

res int;

begin

res := i;

return res;

end;

begin

id2 := pkg1.pkg1\_fun2(10);  --调用包pkg1中定义的函数pkg1\_fun2

end pkg2;

/

5、调用包函数。

select pkg1.pkg1\_fun1(100) from dual;

select pkg1.pkg1\_fun2(100) from dual;

select pkg2.pkg2\_fun1(100) from dual;

select pkg2.pkg2\_fun2(100) from dual;

返回结果分别为：

pkg1\_fun1

--------

10

(1 row)

pkg1\_fun2

--------

100

(1 row)

pkg2\_fun1

--------

5

(1 row)

pkg2\_fun2

--------

100

(1 row)

#### **CREATE PACKAGE**

**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。包体相关内容请参考[CREATE PACKAGE BODY](CREATE-PACKAGE-BODY.md)。

Vastbase G100在Oracle兼容模式下，CREATE PACKAGE支持如下功能：

* 支持函数返回嵌套表和关联数组类型。
* 包内支持使用%type定义变量的类型，赋值时包体可自动转换。
* 支持同义词引用对象语法定义。
* 创建package中支持类型调用。
* 创建package时支持使用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。
* 支持在package中定义的record类型作为procedure出参。
* 编译控制选项pragma serially\_reusable，可以将package设定为连续可复用包，此选项可以使得对应package分配的内存在调用后被释放，复用package时包内变量都被初始化为其默认值。

**语法格式**

* 声明包

CREATE [OR REPLACE] PACKAGE [schema\_name.]package\_name IS | AS

  declarations;

END;

* 返回嵌套表和关联数组类型。

create or replace package package\_test as

type table\_test\_type is table of type1 [index by type2];

【说明】

关联数组和嵌套表相关内容请参考《[VASTDATA]Vastbase G100 V2.2 兼容性手册》->Oracle兼容性->PLSQL->[关联数组](PLSQL/%E5%85%B3%E8%81%94%E6%95%B0%E7%BB%84.md)和[嵌套表](PLSQL/%E5%B5%8C%E5%A5%97%E8%A1%A8.md)。

* 包内支持使用%type定义变量的类型，赋值时包体可自动转换。

create or replace package package\_name as

variable\_name table\_name.column\_name%type;

【说明】

%type支持Vastbase支持的类型。

* 支持同义词引用对象语法。

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;

/

* 支持类型调用。

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;

/

* 支持type…is语法，支持record类型作为procedure出参。

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;

/

* 支持record类型作为procedure出参。

【说明】

在package中定义的record类型作为procedure出参的操作步骤为：

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**

--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;

/

* 支持设置编译控制选项pragma serially\_reusable。

CREATE OR REPLACE PACKAGE package\_name IS

PRAGMA SERIALLY\_REUSABLE;

...

END pkg1;

/

**参数说明**

* **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。

* **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。

* **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)。

* **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。

* **type\_name**

类型名。

* **PRAGMA**

package编译控制选项标记。

* **SERIALLY\_REUSABLE**

编译控制选项，将package设置为连续可复用包。

**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。
* %type支持Vastbase支持的类型。
* 编译控制选项pragma serially\_reusable语法不区分大小写。

**示例**

**前置条件：**创建并切换至兼容Oracle的库db\_oracle，检查数据库兼容模式是否为A。

CREATE DATABASE db\_oracle dbcompatibility='A';

\c db\_oracle

show sql\_compatibility;

**示例1：**创建包和包体。<a id='创建包'></a>

1、创建包。

create package sp\_package is

procedure update\_sal(name varchar2,newsal number);

function annual\_income(name varchar2) return number;

end;

/

2、创建包体。

create or replace package body sp\_package is

procedure update\_sal(name varchar2,newsal number)

is

begin

update emp set sal=newsal where ename=name;

end;

function annual\_income(name varchar2)

return number is

annual\_salary number(12,2);

begin

select sal\*12+nvl(comm,0) into annual\_salary from emp

where ename=name;

return annual\_salary;

end;

end;

/

3、删除包体

DROP PACKAGE BODY sp\_package;

**示例2：** 包内存储过程返回int类型嵌套表。

1、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;

/

2、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3：**包内函数返回number(10,2)类型嵌套表，关联数组binary\_integer。

1、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;

/

2、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)

**示例4：**%type语法。

1、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;

/

2、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;

/

3、调用包体。

call pk\_1087209.p1();

返回结果为：

INFO: aaa

INFO:  55

p1

----

​

(1 row)

**示例5：**同义词引用对象。

1、创建普通用户并授权。

CREATE USER user1\_1087651 PASSWORD 'Abc@123456';

grant all on database db\_oracle to user1\_1087651;

CREATE USER user2\_1087651 PASSWORD 'Abc@123456';

grant all on database db\_oracle to user2\_1087651;

2、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);

3、创建包user1\_1087651.pk1\_1087651。

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;

/

4、创建同义词。

create synonym user2\_1087651.syn\_pk2\_1087651 for user1\_1087651.pk1\_1087651;

5、创建包user2\_1087651.pk3\_1087651。

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;

/

6、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;

/

7、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)

**示例6：**package中的类型调用。

1、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);

2、创建包package pk1\_1087293。

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;

/

3、创建包pk2\_1087293。

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;

/

4、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;

/

5、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)

**示例7：** 包规格内声明int类型嵌套表。

1、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;

/

2、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;

/

3、调用函数。

call func\_Pack\_para();

函数返回结果：

func\_pack\_para

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

             5

(1 row)

**示例8：** 包规格内声明int类型嵌套表关联varchar(20)类型数组。

1、设置serveroutput 为on（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）。

set serveroutput on;

2、包中定义变量。

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();

call proc\_Pack\_para();

函数返回结果：

1 一

3 三

2 二

proc\_pack\_para

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

​

(1 row)

**示例9：**package中定义的record类型作为procedure出参。

1、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;

/

2、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;

/

3、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

**示例10：**使用编译控制选项pragma serially\_reusable控制package连续可复用，复用时包内变量都被初始化为其默认值。<a id='example10'></a>

1、创建带控制选项的测试包。

CREATE OR REPLACE PACKAGE z\_pkg IS

n NUMBER := 5;

END z\_pkg;

/

​

CREATE OR REPLACE PACKAGE sr\_pkg IS

PRAGMA SERIALLY\_REUSABLE;

n NUMBER := 5;

END sr\_pkg;

/

​

BEGIN

z\_pkg.n := 10;

sr\_pkg.n := 10;

END;

/

2、开启参数serveroutput（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），然后调用函数，查看复用package时变量是否初始化。

set serveroutput on;

BEGIN

DBMS\_OUTPUT.PUT\_LINE('z\_pkg.n: ' || z\_pkg.n);

DBMS\_OUTPUT.PUT\_LINE('sr\_pkg.n: ' || sr\_pkg.n);

END;

/

结果显示为如下，package复用后变量被初始化为默认值：

z\_pkg.n: 10

sr\_pkg.n: 5

ANONYMOUS BLOCK EXECUTE

#### CREATE TRIGGER

**功能描述**

Vastbase G100在Oracle兼容模式下支持在创建触发器的语句中包含触发器执行的PL/SQL块。即允许DML触发器或边界触发器直接使用一条语句创建触发器，而不需要提前创建触发器函数或过程。

**语法格式**

CREATE [ OR REPLACE ][ CONSTRAINT ] TRIGGER trigger\_name { BEFORE | AFTER | INSTEAD OF } { event [ OR ... ] }

   ON table\_name

   [ FROM referenced\_table\_name ]

   { NOT DEFERRABLE | [ DEFERRABLE ] { INITIALLY IMMEDIATE | INITIALLY DEFERRED } }

   [ FOR [ EACH ] { ROW | STATEMENT } ]

   [ WHEN ( condition ) ]

   [ DECLARE declarations ]

   BEGIN

    statements

   END [ trigger\_name ];

**参数说明**

* **trigger\_name**

触发器名称。

取值范围：符合标识符命名规范的字符串，且最大长度不超过63个字符。

* **OR REPLACE**

支持创建触发器时使用replace，replace仅对同名同表触发器进行替代，同名触发器不同表时不支持替代。

* **statements**

块的语句部分。可以包含创建触发器的语句。

【说明】

更多参数说明请参考：《Vastbase G100 V2.2 开发者指南 》->SQL语法参考->SQL语法->CREATE TRIGGER。

**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的数据库下。

create database db\_1132436 dbcompatibility='A';

\c db\_1132436;

2、创建测试表。

create table employees(id int,salary int);

3、插入测试数据。

insert into employees values(1,6000);

4、创建DML触发器。

create or replace trigger t

before insert or update of salary,id

or delete on employees

begin

case

when inserting then

dbms\_output.put\_line('inserting');

when updating ('salary') then

dbms\_output.put\_line('updating salary');

when updating ('id') then

dbms\_output.put\_line('updating id');

when deleting then

dbms\_output.put\_line('deleting');

end case;

end;

/

5、设置serveroutput 为on（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）。

set serveroutput on;

6、insert触发。

insert into employees values(2,7000);

返回结果为：

inserting

INSERT 0 1

7、update触发。

update employees set salary= salary+1000 where id=1;

返回结果为：

updating salary

UPDATE 1

8、delete触发。

delete employees where id=2;

返回结果为：

deleting

DELETE 1

#### CREATE TYPE

**功能描述**

Vastbase G100支持使用CREATE TYPE创建抽象数据类型，并支持使用create type body实现在create type中声明的方法。

* create type支持table集合和varray集合。
* 对于创建的抽象对象的应用，支持创建对象表，即表的列属性是对象数据类型(抽象数据类型)。对象表分为行对象表和列对象表。
* 行对象表是指直接基于对象类型所建立的表，一个对象表的列代表了对象数据类型的所有属性，即对象类型是子类型时，表的列就是包含了父类型和子类型的所有属性，每一个行记录就是一个对象数据类型的实例。
* 列对象表是指一张表中既有列是标量类型，也有列是对象数据类型。

**语法格式**

* **CREATE TYPE**

定义抽象数据类型

CREATE [OR REPLACE] TYPE typname [FORCE] AS|IS OBJECT ({atrribute datatype [,]}{,element\_spec})[NOT] {[FINAL|INSTANTIABLE]}};

定义子类型

CREATE [OR REPLACE] TYPE typname, [FORCE] UNDER supertype ({atrribute datatype [,] }{,element\_spec})[NOT]{[FINAL|INSTANTIABLET]}};

* **CREATE TYPE BODY**

定义抽象数据类型体

CREATE [OR REPLACE] TYPE BODY typname {is|as} {proc\_decl\_in\_type|func\_decl\_in\_type|constructor\_declaration|map\_order\_func\_declaration} END；

proc\_decl\_in\_type：创建抽象类型时声明的存储过程，声明语法如下：

PROCEDURE procname {[parameter\_list]}{IS|AS} 存储过程体;

func\_decl\_in\_type：创建抽象类型时声明的函数，声明语法如下：

FUNCTION funcname{[parameter\_list]} RETURN datatype {IS|AS} 函数体;

constructor\_declaration：构造函数，声明语法如下：

[FINAL] [INSTANTTABLE]CONSTRUCTOR FUNCTION datatype [([ SELF IN OUT datatype,] parameter datatype [,parameter datatype ]...)]RETURN SELF AS RESULT {IS|AS} 函数体;

map\_order\_func\_declaration：构造函数，声明语法如下：

{MAP|ORDER}MEMBER FUNCTION funcname {[(parameter\_list)]} RETURN datatype {IS|AS} 函数体;

* 创建对象表

CREATE TABLE tablename OF object\_type;

* table集合和varray集合

create or replace type typname is/as VARRAY (size\_limt) | table of datatype;

**参数说明**

* **typname**

抽象类型名称。

* **tablename**

表名称。

* **object\_type**

创建表使用的类型名称，类型已存在。

* **create or replace**

如果没有type则新建一个，如果已经存在，则删除后重新创建。

* **FORCE**

使用FORCE选项可以避免当前的TYPE被其他对象所依赖后而无法重建或删除。不过这种依赖仅限于对象之间，如果依赖对象的是表，则FORCE功能不起作用。

* **{is | as} object**

表示创建对象类型的自定义type 。

* **atrribute**

抽象数据类型中的成员变量名，抽象数据类型中至少包含一个属性，且属性必须唯一，不能与抽象数据类型中其他属性或者方法重名，子类型属性也不能与父类型属性或者方法重名。

* **element\_spec**

定义抽象类型时声明的成员函数，存储过程，构造器函数等子程序。

* **datatype**

处理抽象数据类型的相关方法。

抽象数据类型的方法总共有三类：子程序方法、构造函数、排序函数。方法的声明语法如下：

[{NOT]{[OVERRIDING]|FINAL|INSTANTIABLE]}] {subprogram\_spec |constructor\_spec|map\_order\_function\_spec}

* [{NOT]{[OVERRIDING]|FINAL|INSTANTIABLE]}]

继承子句。表示父类型与子类型关系的子句。

* OVERRIDING：在创建type时，支持对成员函数进行重写（overriding）。即：在子类型type2中创建与父类型type1中成员函数同名同参的成员函数；通过overriding关键字进行覆写，使type2中的该函数可以与type1中的实现不同的功能。

默认值为：NOT OVERRIDING

* FINAL ：指定该类型中的这个方法不能被该类型的任何子类型方法覆盖。

默认值：NOT FINAL。

* INSTANTIABLE：指定抽象类型中该方法是否可以被实例化。

默认值：所有的方法都可以被实例化，暂不支持该选项。

* subprogram\_spec

子程序方法声明：子程序分为member方法和static方法，声明语法如下：

[MEMBER|STATIC] function\_spec|procedure\_spec;

* member方法：访问抽象数类型的实例中的数据类型的方法，该类型的方法有一个隐形的参数SELF，代表的是调用该方法的抽象数据类型的实例，member方法无论是否显示定义第一个参数为SELF，它的一个参数都是SELF，调用方式为：object\_expression.method()，即抽象数据类型的实例.方法()。

【说明】

* 成员方法表示对象实例级别的数据调用。
* 成员方法的 self是内置参数，表示调用当前实例的对象。
* 成员方法的 self可以显示声明，但不是必须的。
* type中定义的对象，在成员方法中可以隐式 self方式的进行访问。
* static 方法：与抽象数据类型相关的方法，static 方法没有隐形参数 SELF，因此在方法实现中不能使用抽象数据类型中访问特定实例的属性，调用方式为typname.method ()，即抽象数据类型.方法()。

【说明】

* 用于访问对象类型，能够在对象类型上执行全局操作，而不需要访问特定对象实例的数据，所以 static方法没有 self参数。
* static方法只能由对象类型调用，不能由对象实例调用。
* function\_spec：函数体。
* procedure\_spec：存储过程体。
* constructor\_spec

构造函数声明。由于抽象数据类型不像标量数据类型一样可以直接赋值，因此需要使用构造函数对其中的属性进行赋值。声明方式如下:

[FINAL][INSTANTIABLE]CONSTRUCTOR FUNCTION datatype[([ SELF IN OUT datatype,] parameter datatype [, parameter datatype ]... )] RETURN SELF AS RESULT;

【注意】

该方法是用户自定义的构造函数，该类方法有一个隐形的参数 SELF(无论是否显示指定，这个参数都存在)，代表的是当前正在调用的抽象数据类型的实例，返回值是该抽象数据类型的实例。用户可以为一个抽象数据类型创建多个构造函数，只要这些构造函数的参数在参数顺序，个数，参数类型上不同即可。默认的构造函数需要传入全部属性的值。

INSTANTIABLE选项暂不支持。

* map\_order\_function\_spec

排序函数是指定抽象数据类型的比较方法和比较规则，声明方法如下：

{MAP|ORDER}MEMBER function\_spec

* MAP函数：按照用户自定义的比较方法和排序规则将多个对象实例排序。

【注意】

* map 成员方法的 self可以显示声明，但不是必须的。
* map成员方法在一个 type中只允许定义一个。
* map成员方法可以将单个实例的对象转化为标量数据类型(如CHAR或REAL)的值，具有预定义顺序的作用，方便进行比较。
* map成员方法只能定义 self参数，方法不能定义为其他任何参数。
* ORDER函数：按照用户自定义的比较方法和规则将两个抽象数据类型对象进行比较大小。

【说明】

* order 成员方法和 map 成员方法不能同时在一个 type 中定义。
* order 成员方法在一个type 中只允许定义一个。
* order 成员方法必须定义一个 object type的输入参数，并且函数返回值类型必须为 integer数据类型。
* order 成员方法是比较两个 type实例的对象大小。

当用户定义一个类型时，这两种函数只能2选1。

* **[NOT]{[FINAL|INSTANTIABLET]}**

表示该类型是否可以继承。

* [NOT] FINAL：表示该类型是否可以创建子类型。

默认值：FINAL，不允许该类型创建子类型。

* [NOT] INSTANTIABLET：表示此对象是否有任何实例，当前版本暂不支持该选项。

**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。
* 在使用OVERRIDING进行成员函数重写时，若希望对参数名也进行检查，需要打开参数[enable\_proc\_param\_name\_override](%E5%8F%82%E6%95%B0%E5%90%8D%E9%87%8D%E8%BD%BD.md)。（参见《Vastbase G100 V2.2 Build12兼容性手册 》->Oracle兼容性->特性参数->[enable\_proc\_param\_name\_override](%E5%8F%82%E6%95%B0%E5%90%8D%E9%87%8D%E8%BD%BD.md)。）
* 静态方法（STATIC）不能重写（OVERRIDING）。

**示例**

前置步骤：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

**示例1：**使用 create or replace type 定义抽象数据类型。

create or replace type addresstype as object

(

province varchar(20),

city varchar(30),

street varchar(40)

);

**示例2：**继承子句。

1、创建父类型，使用not final选项。

create or replace type parenttype as object

(

province varchar(20),

city varchar(30),

street varchar(40)

) not final;

2、创建子类型。

create or replace type childrentype under parenttype

(

home varchar(20),

position varchar(30)

);

**示例3：**使用抽象数据类型创建表并插入数据。

1、创建抽象父类型。

create or replace type parenttype as object

(

province varchar(20),

city varchar(30),

street varchar(40)

)not final;

2、创建抽象子类型。

create or replace type childrentype under parenttype

(

home varchar(20),

position varchar(30)

);

3、创建表。

create table addresstable (id int,address childrentype);

4、插入数据。

insert into addresstable values(1,childrentype('广西省','桂林市','全州县','才湾镇','才湾村'));

5、查询数据。

select \* from addresstable;

返回结果为：

id |               address

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

 1 | (广西省,桂林市,全州县,才湾镇,才湾村)

(1 row)

**示例4：**使用type创建table集合类型。

1、创建抽象类型。

create type typa as object(a int,name varchar2(10),

constructor function typa(m\_a int) return self as result

);

2、创建对象表并插入数据。

create table object\_table of typa;

insert into object\_table values(1,'nanjing');

3、查询对象表中数据。

select \* from object\_table;

返回结果为：

a | name

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

1 | nanjing

(1 row)

**示例5：**创建order成员方法。

create or replace type emp\_type4 as object(

name varchar2(10),

birthdate date,

order member function compare(emp4 emp\_type4) return int

);

返回结果如下，表示创建成功。

CREATE TYPE

**示例6：**使用type创建varray集合。

【说明】

varry是变长数组类型，用于处理pl/sql集合的数据类型。

1、基于type创建varray集合。

create type phone\_typ as object(

country\_code varchar2(2),

area\_code varchar2(3),

ph\_number varchar2(7)

);

2、基于已创建的varray集合再次创建varray集合。

create type phone\_varray\_typ as varray(5) of phone\_typ;

3、创建测试表。

create table dept\_phone\_list(

dept\_no number(5),

phone\_list phone\_varray\_typ

);

4、基于type的varray集合，使用构造函数方式插入数据。

insert into dept\_phone\_list values(

100,phone\_varray\_typ(phone\_typ('01','650','5550123'),phone\_typ('01','650','5550148'),phone\_typ('01','650','5550192'))

);

5、查询数据。

select \* from dept\_phone\_list;

返回结果为：

dept\_no |                         phone\_list

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

    100 | {"(01,650,5550123)","(01,650,5550148)","(01,650,5550192)"}

(1 row)

**示例7：**使用OVERRIDING实现成员函数的重写。

1、设置参数enable\_proc\_param\_name\_override的值为on。（在覆写成员函数时对参数名进行检查。）

set enable\_proc\_param\_name\_override=on;

2、创建一个type对象t\_person。

create type t\_person as object (

id integer,

first\_name varchar(10),

last\_name varchar(10),

member function display\_details(a int) return varchar2

)not final;

​

create type body t\_person as

  member function display\_details(a int) return varchar2 is

begin

   return 'id='|| id || ',name=' || first\_name || ' ' || last\_name ;

   end;

end;

/

3、创建子类型t\_business\_person，对父类型的功能进行重写。

create type t\_business\_person under t\_person (

title varchar2(10),

company varchar2(10),

overriding member function display\_details(a int) return varchar2,

member function display\_details(b int) return varchar2

);

​

create type body t\_business\_person as

overriding member function display\_details(a int) return varchar2 is

begin

   return 'id=' || id || ',name=' || first\_name || ' ' || last\_name || 'title=' || title;

end;

​

member function display\_details(b int) return varchar2 is

begin

   return 'id=' || id || ',name=' || first\_name || ' ' || last\_name || 'title=' || title || 'company=' || company ;

end;

​

end;

/

4、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。

set serveroutput on;

5、分别调用父类型和子类型，将输出不同的display\_details信息。

DECLARE

  v\_person t\_person;

  v\_business\_person t\_business\_person;

BEGIN

  v\_person := t\_person(1,'测试F','测试2');

  v\_business\_person  := t\_business\_person(1,'AA','BB','标题~','公司Co');

​

  dbms\_output.put\_line(v\_person.display\_details(1)); --父类型输出信息

  dbms\_output.put\_line(v\_business\_person .display\_details(a=>1));  --子类型重写后的输出信息

  dbms\_output.put\_line(v\_business\_person  .display\_details(b=>1)); --子类型重写后的输出信息

end;

/

返回结果为：

id=1,name=测试F 测试2

id=1,name=AA BBtitle=标题~

id=1,name=AA BBtitle=标题~company=公司Co

ANONYMOUS BLOCK EXECUTE

#### CREATE VIEW创建只读视图

**功能描述**

Vastbase G100在Oracle兼容模式下支持在创建视图时使用with read only选项标记为只读视图：视图仅支持查询操作，不允许修改原表。

【说明】

禁止通过视图插入、更新、删除原表。

插入、更新、删除原表数据时，视图内容同步更新。

**语法格式**

CREATE [ OR REPLACE ] [DEFINER = user] [ TEMP | TEMPORARY ] [ FORCE | NOFORCE ] VIEW view\_name [ ( column\_name [, ...] ) ]

   [ WITH ( {view\_option\_name [= view\_option\_value]} [, ... ] ) ]

   AS query [ WITH READ ONLY ];

**参数说明**

* **OR REPLACE**

当CREATE VIEW语句中存在OR REPLACE时，表示若以前存在该视图就进行替换，但新查询不能改变原查询的列定义，包括顺序、列名、数据类型、类型精度等，只可在列表末尾添加其他的列。

* **DEFINER = user**

指定user作为视图的属主。

* **TEMP | TEMPORARY**

创建临时视图。

* **FORCE | NOFORCE**

强制创建视图。正常情况下，如果基表不存在，创建视图就会失败。指定force选项可以强制创建视图。

如果仅在存在基表且包含视图的模式的所有者对基表具有特权的情况下才希望创建视图，则指定NOFORCE。NOFORCE是默认值。

* **view\_name**

要创建的视图名称。可以用模式修饰。

取值范围：字符串，符合标识符命名规范。

* **column\_name**

可选的名称列表，用作视图的字段名。如果没有给出，字段名取自查询中的字段名。

取值范围：字符串，符合标识符命名规范。

* **view\_option\_name [= view\_option\_value]**

该子句为视图指定一个可选的参数。

目前view\_option\_name支持的参数仅有security\_barrier，当VIEW试图提供行级安全时，应使用该参数。

取值范围：Boolean类型，TRUE、FALSE

* **query**

为视图提供行和列的SELECT或VALUES语句。

* **with read only**

指定此选项将被标记为只读视图。

**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。
* 物化视图不支持with read only选项。
* 只读视图同步更新对原表数据的修改，不包括原表的表结构变更。
* 只读视图一旦定义，不可修改。（不能使用ALTER VIEW语法修改视图的各种属性。）

**示例**

**前置条件：**创建并切换至兼容模式为Oracle的数据库oracle\_db下。

create database oracle\_db dbcompatibility 'A';

\c oracle\_db;

**示例1：**创建只读视图，无法修改原表。

1、创建测试表并插入数据。

create table test(id int,name text);

insert into test values(1,'aaa'),(2,'bbb'),(3,null),(null,'bbb');

2、创建只读视图。

create view v1 as select \* from test with read only;

3、查看该视图。

select \* from v1;

返回结果为：

id | name

----+------

 1 | aaa

 2 | bbb

 3 |

   | bbb

(4 rows)

4、更新视图。

update v1 set name='ccc' where id=3;

更新失败，无法通过只读视图更新原表。

ERROR: cannot update view "v1"

DETAIL:  Views with read only are not automatically updatable.

**示例2：**创建只读视图，更新原表时，视图内容同步更新。

1、创建测试表并插入数据。

create table class(id int,name text);

insert into class values(1,'aaa'),(2,'bbb'),(3,null),(null,'bbb');

2、创建只读视图。

create view v2 as select \* from class with read only;

3、查看视图内容。

select \* from v2;

返回结果为：

id | name

----+------

 1 | aaa

 2 | bbb

 3 |

   | bbb

(4 rows)

4、为原表添加一个字段。

alter table class add column c1 text;

5、插入一条测试数据。

insert into class values(4,'tom','ttt');

6、再次查看视图。

select \* from v2;

返回结果如下：视图已更新，可以看到新增的一行数据，但不会显示新增的字段。

id | name

----+------

 1 | aaa

 2 | bbb

 3 |

   | bbb

 4 | tom

(5 rows)

### PL/SQL

#### 匿名块中支持占位符作为赋值表达式左值

**功能描述**

Vastbase G100在Oracle兼容模式下支持PL/SQL在匿名块中使用占位符作为赋值表达式左值。

**语法格式**

:占位符 := value;

**参数说明**

**value**

占位符的值或表达式。

**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。
* SQL语句的占位符使用英文冒号“:”开头。

**示例**

**前置条件：**

* 创建并切换至兼容模式为Oracle的数据库下。

create  database dbtest dbcompatibility 'A';

\c dbtest;

* 设置serveroutput 为on（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）。

set serveroutput =on;

**示例1：**占位符作为赋值表达式左值，表达式为系统自带函数。

1、创建函数。

create or replace FUNCTION f1(var in number) return number is

l\_var varchar(10) :='2022-12-28' ;

l\_res number ;

BEGIN

execute immediate 'begin :result := year(:2); end;' using out l\_res,l\_var;

dbms\_output.put\_line(l\_res);

return var;

end;

/

函数创建成功后，调用函数。

call f1(1);

调用函数成功，返回结果为：

f1

----

 1

(1 row)

**示例2：**用占位符作为赋值表达式左值，赋值为null。

1、创建函数。

create or replace FUNCTION f1(var in number) return number is

BEGIN

return null;

end;

/

2、创建匿名块。在匿名块中使用占位符作为赋值表达式左值，赋值为null并输出结果。

declare

l\_var number :=1;

l\_res text;

l\_funname varchar2(20) := 'f1';

begin

execute immediate 'begin :result := ' || l\_funname || '(:2); end;' using out l\_res, l\_var;

dbms\_output.put\_line(l\_res);

end;

/

匿名块创建成功，返回结果为：

ANONYMOUS BLOCK EXECUTE

输出结果为空。

### 组件

#### DBLINK

**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。

在Vastbase G100 V2.2 Build 12版本中增强DBLINK同义词功能，支持DBLINK与SYNONYM结合访问Oracle包中的函数。

【说明】

从Vastbase G100 V2.2 Build10（Patch No.6）开始，当通过DBLINK连接至Vastbase G100或Oracle数据库时，通过analyze tablename@dblinkname语法可获得更多建议，优化建议仅供参考。

**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。

INSERT INTO [${SCHEMA\_NAME}].${TABLE\_NAME}@${DBLINK\_NAME} #插入UPDATE [${SCHEMA\_NAME}].${TABLE\_NAME}@${DBLINK\_NAME} SET... #更新DELETE FROM [${SCHEMA\_NAME}].${TABLE\_NAME}@${DBLINK\_NAME} WHERE... #删除 SELECT ... FROM [${SCHEMA\_NAME}].${TABLE\_NAME}@${DBLINK\_NAME} #查询

**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，并具备DBLINK相关权限。
* 不支持二进制数据类型、二进制大对象类型、xml数据类型。
* 不支持retun为%rowtype行数据类型。
* 由于 Oracle与Vastbase函数的差异性，return值会以最后一个out参数形式存在，直接远程调用函数、存储过程时，数据库会自动补充return参数，不需要用户补充，而使用同义词时，无法确定此时是call调用还是select 调用，无法自动补全return参数，需要用户自行补充，在使用流程章节中将以用例再次详细说明。

**示例**

**前置步骤：**创建postgres\_fdw。

1. 修改postgresql.conf参数，将jdbc\_fdw配置到shared\_preload\_libraries参数中。

shared\_preload\_libraries='jdbc\_fdw'

2、驱动包ojdbc8.jar放置到$GAUSSHOME/lib/postgresql/下。

chmod 777 $GAUSSHOME/lib/postgresql/ojdbc8.jar

3、重启数据库实例。

4、在需要使用jdbc\_fdw的database下执行以下命令。

DROP EXTENSION jdbc\_fdw;

CREATE EXTENSION jdbc\_fdw;

**示例1：**通过DBLINK访问远端数据库。

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)

**示例2：**与同义词结合访问Oracle包中的函数。

1、在Oracle中创建测试数据和包。

drop table tb\_1133599;

create table tb\_1133599(id number(10) ,name varchar(10),info varchar2(10));

insert into tb\_1133599 values(1,'小明','中国');

insert into tb\_1133599 values(2,'vivi','未知');

insert into tb\_1133599 values(3,'zuzu','未知');

drop package ora\_pkg\_1133599;

create or replace package ora\_pkg\_1133599 as

function ora\_pkg\_1133599\_func1(id1 number,c2 out varchar,c3 out varchar2) return number;

procedure ora\_pkg\_1133599\_proc1(id1 number ,col2 out varchar,col3 out varchar2);

end ora\_pkg\_1133599;

/

create or replace package body ora\_pkg\_1133599 as

function ora\_pkg\_1133599\_func1(id1 number,c2 out varchar,c3 out varchar2) return number

as

begin

select name into c2 from tb\_1133599 where id=id1;

select info into c3 from tb\_1133599 where id=id1;

return id1;

end;

create procedure ora\_pkg\_1133599\_proc1(id1 number ,col2 out varchar,col3 out varchar2)

as

begin

select name into col2 from tb\_1133599 where id=id1;

select info into col3 from tb\_1133599 where id=id1;

end;

end ora\_pkg\_1133599;

/

2、在Vastbase中创建DBLINK。

CREATE public DATABASE LINK dblink\_104 CONNECT TO SYSTEM IDENTIFIED BY 'root' USING jdbc\_fdw(

url 'jdbc:oracle:thin:@//172.16.103.104:1521/orcl',

jarfile '/$GAUSSHOME/lib/postgresql/ojdbc8.jar'

);

3、在Vastbase中创建同义词访问Oracle的包。

create SYNONYM syn\_1133599 for ora\_pkg\_1133599@dblink\_104;

4、创建自定义函数。

create or replace function func\_1133599(id int) return varchar as

declare

v\_id number;

v\_name varchar;

v\_info varchar2(10);

begin

if id=1 then

syn\_1133599.ora\_pkg\_1133599\_func1(id,v\_name,v\_info,v\_id);

v\_name=v\_id||v\_name||v\_info;

return v\_name;

elsif id=2 then

syn\_1133599.ora\_pkg\_1133599\_proc1(2,v\_name,v\_info);

v\_name=v\_name||v\_info;

return v\_name;

else

return 'null';

end if;

end;

/

5、调用函数。

call func\_1133599(1);

结果返回如下：

func\_1133599

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

1小明中国

(1 row)

## MySQL兼容性

* + 1. **数据类型**

Vastbase G100兼容MySQL数据类型如下表所示：

| **类型** | **描述** | **存储空间** |
| --- | --- | --- |
| binary | BINARY类型是固定长度二进制字符串，与char相似，可存储字符。不同的是BINARY存储的是二进制值，BINARY无字符集的概念，对其排序和比较都是按照二进制值进行对比。BINARY（N）中的N指的是字节长度。N取值范围为1~255。 | 最大为1GB |
| longtext | 变长字符。 | 最大为1GB，但还需要考虑到列描述头信息的大小， 以及列所在元组的大小限制（也小于1GB），因此longtext类型最大小于1GB。 |
| datetime | datetime[n]用于设置显示毫秒后面的多少位，n表示获取的位数，取值为1。 | 8字节 |
| longblob | longblob是一种存储二进制数据的数据类型，可以用于存储大型二进制对象（BLOB），例如图像，音频和视频等数据。 | 最大为4GB |

**注意事项**

* binary，datetime数据类型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。
* datetime[n]数据类型的实现依赖Vastbase现有timestamp数据类型，因此，在一些场景中，例如\d元命令中，datetime数据类型会被显示成timestamp类型。
* 当datetime[n]类型指定了小数位之后，如果datetime[n]类型的数据小数位都是0，则在Vastbase数据库中，受到timestamp数据类型的局限，其小数位上的0将不做显示。

**示例**

**前置步骤：**创建兼容模式为MySQL的库db\_mysql，并进入。

create database db\_mysql dbcompatibility 'B';

\c db\_mysql

**示例1：**binary类型。

1. 创建测试binary类型的表test\_mysql\_type2并插入测试数据。

CREATE TABLE test\_mysql\_type2(c1 binary);

INSERT INTO test\_mysql\_type2 values('a');

INSERT INTO test\_mysql\_type2 values('张三');

1. 验证binary数据类型。

SELECT \* FROM test\_mysql\_type2;

当结果显示如下信息，则表示验证完成。

c1

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

\x61

\xe5bca0e4b889

(2 rows)

**示例2：**longtext类型。

1. 创建测试longtext类型的表test\_mysql\_type3并插入测试数据。

CREATE TABLE test\_mysql\_type3(c1 longtext);

INSERT INTO test\_mysql\_type3 values('测试abc123');

2、验证longtext数据类型。

SELECT \* FROM test\_mysql\_type3;

当结果显示如下信息，则表示验证完成。

c1

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

测试abc123

(1 row)

**示例3：**datetime类型。

1. 创建测试datetime类型的表test\_mysql\_type4并插入测试数据。

CREATE TABLE test\_mysql\_type4(c1 datetime);

INSERT INTO test\_mysql\_type4 values('2019-03-05 01:53:55.63');

1. 验证datetime数据类型。

SELECT \* FROM test\_mysql\_type4;

当结果显示如下信息，则表示验证完成。

c1

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

2019-03-05 01:53:55.63

(1 row)

**示例4：**longblob类型。

1. 创建测试表并插入数据。

CREATE TABLE t\_longblob

(c1 int primary key auto\_increment,c2 longblob default 'test\_default，测试默认值');

INSERT INTO t\_longblob VALUE(1,'test1');

INSERT INTO t\_longblob(c2) VALUES(default);

1. 查询表中数据。

select \* from t\_longblob;

返回结果为：

c1 | c2

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

1 | 7465737431

2 | 746573745F64656661756C74EFBC8CE6B58BE8AF95E9BB98E8AEA4E580BC

(2 rows)

1. 将longblob类型的二进制数据转换为UTF8编码的形式。

select c1,convert\_from(rawsend(c2::raw),'utf8') from t\_longblob;

【说明】

* convert\_from函数用来以数据库编码的方式转换字符串bytea。详细请参考开发者指南->字符处理函数和操作符章节。
* rawsend是内部处理函数，用于将raw类型数据转换为bytea。

数据表中的存储的longblob类型的二进制数据，经过上述转换之后查询返回结果为：

c1 | convert\_from

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

1 | test1

2 | test\_default，测试默认值

(2 rows)

### 类型转换

#### 支持UUID类型到字符串类型的隐式转换

**功能描述**

Vastbase G100支持[uuid类型](UUID%E7%B1%BB%E5%9E%8B.md)到字符串类型的隐式转换。

**注意事项**

UUID是一个小写十六进制数字的序列，由分字符分成几组，标准的UUID输出格式为：一组8位数字+三组4位数字+一组12位数字，总共32个数字代表128位。在Vastbae G100的不同兼容模式下，UUID的输出格式并不一定包含分字符“-”，详情参考：[uuid类型](UUID%E7%B1%BB%E5%9E%8B.md)。（参见《Vastbase G100 V2.2 开发者指南 》->SQL语法参考->SQL基本要素->数据类型->UUID。

**示例**

【说明】

以下示例均为默认数据库vastbase（A兼容）下的执行结果。

**示例1：**支持uuid到各字符串类型的隐式转换。

1、创建测试表。

create table test(

c11 CHAR(100) ,

c12 CHARACTER(50),

c13 NCHAR(40) ,

c14 VARCHAR(50) ,

c15 CHARACTER VARYING(50) ,

c16 VARCHAR2(40) ,

c17 NVARCHAR(50) ,

c18 NVARCHAR2(50) ,

c19 TEXT );

2、插入测试数据。

insert into test values(uuid(),uuid(),uuid(),uuid(),uuid(),uuid(),uuid(),uuid(),uuid());

3、查看结果：

\x --列式展示查询结果

select \* from test;

返回结果为：

-[ RECORD 1 ]---------------------------------------------------------------------------------------------

c11 | D81D59355A1A7A28B1D58B76563E7841

c12 | 42A84A679C6D93291009213A63FC2F5D

c13 | 13567C0FC4A11571327DBB350E0AB105

c14 | 0B57767E69891E769016246A3D64A508

c15 | 5AD5800221AF6A6471AEE53425818C12

c16 | C26B9D6A65616E5B222EFC1EFC033867

c17 | 6B942921915A4003D3E38735C6B95A09

c18 | 43FA7C2808A7DF520E0BAB654C51F17C

c19 | 0F37DB65758B1607D422DD762C601128

**示例2：**UUID与其他函数一起使用，存在隐式转换。

1、使用substr函数返回指定位置之后的字符串内容。

select substr(uuid(),1);

返回结果为：

            substr

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

733A0A562317040D109A3E2693C12027

(1 row)

2、使用contact函数连接字符串文本。

select concat('1',uuid());

返回结果为：

            concat

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

105E4B8086C9E8131D6249D69E826FA53

(1 row)

### 操作符

#### +运算符精度处理

**功能描述**

Vastbase G100在MySQL兼容模式下使用“+”运算符时，若左值为“0”且右值为任意带数字的字符串（charnum），则0 + charnum将得到与原字符串charnum中相同精度的浮点数或者整数。

**语法格式**

0 + charnum

**参数说明**

**charnum**

带数字的字符串。

**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。

**示例**

**前置条件：**创建并切换至兼容模式为MySQL的数据库db\_mysql下。

CREATE DATABASE db\_mysql dbcompatibility='B';

\c db\_mysql

**示例1：**通过cast(x as y)函数进行类型转换。

【说明】

cast(x as y)为类型转换函数，用于将x转换成y指定的类型。参见《Vastbase G100 V2.2 开发者指南 》->SQL语法参考->SQL基本要素->函数和操作符->[类型转换函数](%E7%B1%BB%E5%9E%8B%E8%BD%AC%E6%8D%A2%E5%87%BD%E6%95%B0.md" \l "cast)。

1、select查看cast(x as y)的结果：

select cast(2/3 as char(6));

返回结果如下：

bpchar

--------

.66666

(1 row)

2、查看0 + cast(x as y)的结果。

select 0+cast(2/3 as char(6));

返回结果如下，0 + cast(x as y)的结果精度与cast(x as y)的返回值精度相同。

?column?

----------

  .66666

(1 row)

**示例2：**在DML语句中使用0+cast(x as y)特性。

1、创建测试表并插入数据。

create table student(stdno int,student\_age char(20),grade float8);

insert into student values(001,'15.00',81.00);

insert into student values(002,'16.00',82.01);

insert into student values(003,'17.00',81.02);

insert into student values(004,'13.00',83.50);

insert into student values(005,'14.00',84.00);

insert into student values(006,'15.00',89.50);

insert into student values(007,'19.000',91.00);

insert into student values(008,'12.0000',93);

insert into student values(008,'11',61.0);

2、查看表student的数据，根据列stdno进行升序排序。

select \* from student order by stdno;

返回结果如下：

stdno | student\_age | grade

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

1 | 15.00 | 81

2 | 16.00 | 82.01

3 | 17.00 | 81.02

4 | 13.00 | 83.5

5 | 14.00 | 84

6 | 15.00 | 89.5

7 | 19.000 | 91

8 | 12.0000 | 93

8 | 11 | 61

(9 rows)

3、执行update操作。

UPDATE student SET student\_age= (0 + CAST(student\_age AS CHAR)) WHERE 1=1;

4、更新成功，查看更新的结果。

select \* from student order by stdno;

返回结果如下：

stdno | student\_age | grade

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

1 | 15 | 81

2 | 16 | 82.01

3 | 17 | 81.02

4 | 13 | 83.5

5 | 14 | 84

6 | 15 | 89.5

7 | 19 | 91

8 | 12 | 93

8 | 11 | 61

(9 rows)

### SQL语法

#### CHANGE COLUMN

**功能描述**

Vastbase G100在MySQL兼容模式下支持使用CHANGE COLUMN语法修改表列的名字和属性。

**语法格式**

ALTER TABLE table\_name change col1 col2 datatype;

**参数说明**

* **table\_name**

表名。

* **col1**

被修改的表列名。

* **col2**

修改之后的表列名，col1和col2可以相同也可以不同。

* **datatype**

修改后的表列的数据类型。

**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。

**示例**

**前置步骤：**创建兼容模式为MySQL的库db\_mysql，并进入。

create database db\_mysql dbcompatibility 'B';

\c db\_mysql

1、创建一张测试表，主键id。

create table t (col1 int);

2、修改表的列名和数据类型。

alter table t change col1 col2 varchar(100);

1. 查询修改后的表结构。

\d t

2、修改表的列名和数据类型。

Table "public.t"

Column | Type | Modifiers | Attalias

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

col2 | varchar(100) | | col1

#### CREATE DATABASE

**功能描述**

Vastbase G100在MySQL兼容模式下支持在CREATE DATABASE时通过PAD\_ATTRIBUTE参数指定字符串等值比较时是否忽略字符串尾端的空格匹配。

**语法格式**

CREATE DATABASE [ IF NOT EXISTS ] database\_name

   [ [ WITH ] {[ OWNER [=] user\_name ]|

          [ TEMPLATE [=] template ]|

          [ ENCODING [=] encoding ]|

          [ LC\_COLLATE [=] lc\_collate ]|

          [ LC\_CTYPE [=] lc\_ctype ]|

          [ DBCOMPATIBILITY [=] compatibility\_type ]|

          [ TABLESPACE [=] tablespace\_name ]|

          [ CONNECTION LIMIT [=] connlimit ]}[...] ]|

          [ PAD\_ATTRIBUTE [=] pad\_attribute\_type ];

**参数说明**

* **PAD\_ATTRIBUTE [=] pad\_attribute\_type**

为新数据库设置默认的列校对规则。可选取值包括：

* N,NO PAD：默认值。把字符串尾端的空格当作一个字符处理，即字符串等值比较不忽略尾端空格。
* S,PAD SPACE：字符串等值比较忽略尾端空格。

【说明】

更多参数说明请参考：《Vastbase G100 V2.2 开发者指南》->SQL语法参考->SQL语法->[CREATE DATABASE](CREATE-DATABASE.md)。

**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。
* 在其它兼容模式的数据库下，可以使用PAD SPACE版本的COLLATION，但不会生效；即字符串等值比较时都不会忽略尾端的空格。（例如：其它兼容模式下，指定COLLATION为C和C-space是等效的，字符串等值比较时都不会忽略尾端的空格。）

**示例**

1、创建兼容模式为MySQL的数据库，同时指定PAD ATTRIBUTE为PAD SPACE。

create database db\_1132613 dbcompatibility 'B' PAD\_ATTRIBUTE 'S';

2、切换至上一步创建的数据库db\_1132613下。

\c db\_1132613

3、查看当前数据库的PAD ATTRIBUTE参数。

show pad\_attribute;

返回结果为：

pad\_attribute

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

PAD SPACE

(1 row)

4、创建具有各种字符串类型的测试表，并为每一列设置不同的列校对规则。（其中列c2、c4、c6的校对规则为\*-space，即忽略空格匹配，c7未指定列校对规则，使用建库时指定的默认校对规则。）

create table tb\_1132417(

c1 text COLLATE "C",

c2 character varying(30) COLLATE "C-space",

c3 varchar(30) COLLATE "zh\_CN",

c4 varchar2(30) COLLATE "zh\_CN-space",

c5 nvarchar2(30) COLLATE "en\_US",

c6 clob COLLATE "en\_US-space",

c7 varchar(30));

5、插入测试数据。（插入的第二条测试数据尾部均带有空格。）

insert into tb\_1132417 values('qqq', 'cqvjj11', 'h2vu2hb3t', 'f4rh535', 'fr2h5fj3c', 'r2hfhcg', 'r2cgcr');

insert into tb\_1132417 values('qqq ', 'cqvjj11 ', 'h2vu2hb3t ', 'f4rh535 ', 'fr2h5fj3c ', 'r2hfhcg ', 'r2cgcr ');

6、查看是否正确识别空格。

select c1='qqq', c2='cqvjj11', c3='h2vu2hb3t', c4='f4rh535', c5='fr2h5fj3c', c6='r2hfhcg', c7='r2cgcr' from tb\_1132417;

返回结果如下：对于插入的第二条尾部带空格的数据，c2、c4、c6、c7可以忽略空格，返回true，其他列不能忽略空格，返回false。

?column? | ?column? | ?column? | ?column? | ?column? | ?column? | ?column?

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

t        | t        | t        | t        | t        | t        | t

f        | t        | f        | t        | f        | t        | t

(2 rows)

#### CREATE TABLE

**功能描述**

Vastbase G100在MySQL兼容模式下，CREATE TABLE语法支持以下用法：

* 支持在建表时指定存储引擎和字符集。
* 支持在建表时指定表级校对规则。

【说明】

其他关于CREATE TABLE的用法请参考参见《Vastbase G100 V2.2 开发者指南 》->SQL语法参考->SQL语法->[CREATE TABLE](../../%E5%BC%80%E5%8F%91%E8%80%85%E6%8C%87%E5%8D%97/CREATE-TABLE.md)。

**语法格式**

CREATE [ [ GLOBAL | LOCAL ] { TEMPORARY | TEMP } | UNLOGGED ] TABLE [ IF NOT EXISTS ] table\_name

 ( { column\_name data\_type [ compress\_mode ] [ COLLATE collation ] [ column\_constraint [ ... ] ]

   | table\_constraint

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

   [, ... ])

   [ AUTO\_INCREMENT [ = ] value ]

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

   [ WITH ( {storage\_parameter = value} [, ... ] ) ]

   [ ON COMMIT { PRESERVE ROWS | DELETE ROWS | DROP } ]

   [ COMPRESS | NOCOMPRESS ]

   [ TABLESPACE tablespace\_name ]

   [ COLLATE [=] collate\_name];

[ ENGINE [=] engine\_name ]

[ [DEFAULT] { CHARSET | CHARACTER SET } [=] charset\_name ];

**参数说明**

* **collate\_name**

表的校对规则。可用的规则可以在系统表[PG\_COLLATION](PG_COLLATION.md)中查询，默认的排序规则为查询结果中以default开始的行。

* **[DEFAULT] { CHARSET | CHARACTER SET } [=] charset\_name**

该参数用于选择表所使用的字符集。

* **ENGINE [=] engine\_name**

该参数用于指定存储引擎。

【说明】

更多参数说明请参考：《Vastbase G100 V2.2 开发者指南 》->SQL语法参考->SQL语法->CREATE TABLE。

**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。
* 支持在建表时指定存储引擎和字符集的功能只做纯语法兼容，不实现功能。

**示例**

**前置步骤：**创建兼容模式为MySQL的库db\_mysql，并进入。

create database db\_mysql dbcompatibility='B';

\c db\_mysql;

示例1：在建表时指定表级校对规则。

1、查看数据库的pad\_attribute参数。

show pad\_attribute;

返回结果为：

pad\_attribute

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

NO PAD

(1 row)

​

【说明】

Vastbase G100从V2.2 Build 12开始支持[初始化数据库时指定pad\_attribute](vb_initdb.md" \l "attribute)以及[创建数据库时指定pad\_attribute](CREATE-DATABASE.md" \l "attribute)的功能，以此来为新数据库设置默认的列校验属性，可选取值包括：

* N,NO PAD：默认值。把字符串尾端的空格当作一个字符处理，即字符串等值比较不忽略尾端空格。
* S,PAD SPACE：字符串等值比较忽略尾端空格。

1. 创建具有各种字符串类型的测试表，并为每一列指定不同的列校对规则，指定表级别COLLATE为"C"。

【说明】

其中列c2、c4、c6的校对规则为\*-space，即忽略空格匹配。c7未指定列校对规则，使用建库时的默认校对规则，本示例建库时未指定，默认为NO PAD。

create table tb\_1132414\_01(

c1 text COLLATE "C",

c2 character varying(30) COLLATE "C-space",

c3 varchar(30) COLLATE "zh\_CN",

c4 varchar2(30) COLLATE "zh\_CN-space",

c5 nvarchar2(30) COLLATE "en\_US",

c6 clob COLLATE "en\_US-space",

c7 varchar(30)) COLLATE "C";

3、插入测试数据。（第一次插入的字符串尾端无空格，第二次插入的字符串尾端存在空格。）

insert into tb\_1132414\_01 values('qqq', 'cqvjj11', 'h2vu2hb3t', 'f4rh535', 'fr2h5fj3c', 'r2hfhcg', 'r2cgcr');

insert into tb\_1132414\_01 values('qqq ', 'cqvjj11 ', 'h2vu2hb3t ', 'f4rh535 ', 'fr2h5fj3c ', 'r2hfhcg ', 'r2cgcr ');

4、查看是否正确识别空格。

select c1='qqq', c2='cqvjj11', c3='h2vu2hb3t', c4='f4rh535', c5='fr2h5fj3c', c6='r2hfhcg', c7='r2cgcr' from tb\_1132414\_01;

返回结果如下：对于插入的第二条尾部带空格的数据，c2、c4、c6可以忽略空格，返回true，其他返回false。

?column? | ?column? | ?column? | ?column? | ?column? | ?column? | ?column?

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

t        | t        | t        | t        | t        | t        | t

f        | t        | f        | t        | f        | t        | f

(2 rows)

**示例2：**在建表时指定存储引擎和字符集。

1、创建一张测试表设置ENGINE和 CHARSET。

create table tab\_1131500 (id int, val1 int, val2 text) ENGINE=InnoDB DEFAULT CHARSET=utf8;

2、给表中插入数据。

insert into tab\_1131500 values(1,1,1);

3、更新表中数据。

update tab\_1131500 set id =2 where id =1 ;

4、查询数据。

select \* from tab\_1131500;

返回结果为：

id | val1 | val2

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

 2 |    1 | 1

(1 row)

#### CREATE TRIGGER

**功能描述**

创建一个触发器。 触发器将与指定的表或视图关联，并在特定条件下执行指定的函数。在MySQL兼容模式下支持以下特性：

* 支持使用DEFINER关键字。
* 创建时支持使用IF NOT EXISTS判断触发器是否已存在。
* 支持[trigger\_order](" \l "trigger_order)中的{FOLLOWS|PRECEDES}来控制触发器的优先触发顺序。
* 支持在trigger\_body部分直接通过在begin...end之间书写代码块。
* 支持使用LEAVE语法。
* 支持用户使用delimiter定义分隔符。在创建触发器SQL末尾可以不使用固定的结尾，整个触发器的语法以delimiter定义的结束符作为结束。

**语法格式**

[delimiter 分隔符]

CREATE [DEFINER = user] [ CONSTRAINT ] TRIGGER [IF NOT EXISTS] name { BEFORE | AFTER | INSTEAD OF } { event [ OR ... ] }

ON table\_name

[ FROM referenced\_table\_name ]

{ NOT DEFERRABLE | [ DEFERRABLE ] { INITIALLY IMMEDIATE | INITIALLY DEFERRED } }

[ FOR [ EACH ] { ROW | STATEMENT } ]

[ WHEN ( condition ) ]

[trigger\_order]

trigger\_body

[ 分隔符

delimiter;]

event部分语法如下：

INSERT

UPDATE [ OF column\_name [, ... ] ]

DELETE

TRUNCATE

trigger\_order部分语法如下：

[ FOLLOWS | PRECEDES ]

**参数说明**

* **delimiter 分隔符**

delimiter为定义结束分隔符关键字，后跟要指定的分隔符。

* **CONSTRAINT**

可选项，指定此参数将创建约束触发器，即触发器作为约束来使用。除了可以使用SET CONSTRAINTS调整触发器触发的时间之外，这与常规触发器相同。 约束触发器必须是AFTER ROW触发器。

* **name**

触发器名称，该名称不能限定模式，因为触发器自动继承其所在表的模式，且同一个表的触发器不能重名。

取值范围：符合标识符命名规范的字符串，且最大长度不超过63个字符。

* **BEFORE**

触发器函数是在触发事件发生前执行。

* **AFTER**

触发器函数是在触发事件发生后执行，约束触发器只能指定为AFTER。

* **INSTEAD OF**

触发器函数直接替代触发事件。

* **event**

启动触发器的事件，取值范围包括：INSERT、UPDATE、DELETE或TRUNCATE，也可以通过OR同时指定多个触发事件。

对于UPDATE事件类型，可以使用下面语法指定列：

UPDATE OF column\_name1 [, column\_name2 ... ]

表示当这些列作为UPDATE语句的目标列时，才会启动触发器，但是INSTEAD OF UPDATE类型不支持指定列信息。

* **table\_name**

需要创建触发器的表名称。

取值范围：数据库中已经存在的表名称。

* **referenced\_table\_name**

约束引用的另一个表的名称。 只能为约束触发器指定，常见于外键约束。

取值范围：数据库中已经存在的表名称。

* **DEFERRABLE | NOT DEFERRABLE**

约束触发器的启动时机，仅作用于约束触发器。这两个关键字设置该约束是否可推迟。

详细介绍请参见《[VASTDATA]Vastbase G100 V2.2 开发者指南》->SQL语法参考->SQL语法->[CREATE TABLE](../../../../../VastbaseG100Ver2.2.10/doc/%E5%BC%80%E5%8F%91%E8%80%85%E6%8C%87%E5%8D%97/CREATE-TABLE.md)。

* **INITIALLY IMMEDIATE | INITIALLY DEFERRED**

如果约束是可推迟的，则这个子句声明检查约束的缺省时间，仅作用于约束触发器。

详细介绍请参见《[VASTDATA]Vastbase G100 V2.2 开发者指南》->SQL语法参考->SQL语法->[CREATE TABLE](../../../../../VastbaseG100Ver2.2.10/doc/%E5%BC%80%E5%8F%91%E8%80%85%E6%8C%87%E5%8D%97/CREATE-TABLE.md)。

* **FOR EACH ROW | FOR EACH STATEMENT**

触发器的触发频率。

FOR EACH ROW是指该触发器是受触发事件影响的每一行触发一次。

FOR EACH STATEMENT是指该触发器是每个SQL语句只触发一次。

未指定时默认值为FOR EACH STATEMENT。约束触发器只能指定为FOR EACH ROW。

* **condition**

决定是否实际执行触发器函数的条件表达式。当指定WHEN时，只有在条件返回true时才会调用该函数。

在FOR EACH ROW触发器中，WHEN条件可以通过分别写入OLD.column\_name或NEW.column\_name来引用旧行或新行值的列。 当然，INSERT触发器不能引用OLD和DELETE触发器不能引用NEW。

INSTEAD OF触发器不支持WHEN条件。

WHEN表达式不能包含子查询。

对于约束触发器，WHEN条件的评估不会延迟，而是在执行更新操作后立即发生。 如果条件返回值不为true，则触发器不会排队等待延迟执行。

* **FOLLOWS**

其他触发器的优先级顺序向后挤压。

* **PRECEDES**

其他触发器的优先级顺序向前挤压。

关于触发器种类：

INSTEAD OF的触发器必须标记为FOR EACH ROW，并且只能在视图上定义。

BEFORE和AFTER触发器作用在视图上时，只能标记为FOR EACH STATEMENT。

TRUNCATE类型触发器仅限FOR EACH STATEMENT。

**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。
* 当前仅支持在普通行存表上创建触发器，不支持在列存表、临时表、unloggd表等类型表上创建触发器。
* 如果为同一事件定义了多个相同类型的触发器，默认依据创建时间的顺序触发，并且支持使用trigger order特征中的{FOLLOWS|PRECEDES}来控制触发器的优先触发顺序。
* 触发器常用于多表间数据关联同步场景，对SQL执行性能影响较大，不建议在大数据量同步及对性能要求高的场景中使用。
* 外键操作不会激活触发器。
* 触发器中不允许return语句，因为触发器不能返回值。要立即退出触发器，使用LEAVE语句。
* 触发器缓存不会检测底层对象的元数据何时发生更改。如果触发器使用表，并且自触发器加载到缓存后该表己更改，则触发器使用过时的元数据进行操作。
* MySQL使用ROW模式复制时，主库的语句不会触发从库的触发器，在Vastbase G100中，复制是通过日志流复制实现，与MySQL采用statement复制模式一致，因此触发器会影响从库。
* delimiter指定分隔符语法仅Vastbase G100 2.2 Build 12及以后版本支持使用。
* 使用delimiter定义分隔符后，使用create trigger语法创建的触发器declare部分定义在begin内部，且每条必须是declare开头的一句话。
* 不使用delimiter定义分隔符，则默认分隔符为/。

**示例**

**前置步骤：**创建并进入兼容MySQL的库my\_test。

CREATE DATABASE my\_test DBCOMPATIBILITY 'B';

\c my\_test

**示例1：** 创建和使用触发器。

1、创建测试数据。

CREATE TABLE t\_follows\_trigger(

id int,rep text,insert\_time timestamp

);

INSERT INTO t\_follows\_trigger values(1,'zhangsan',sysdate);

INSERT INTO t\_follows\_trigger values(2,'lisi',sysdate);

​

CREATE TABLE t\_react(

id int,rep text,insert\_time timestamp

);

2、创建触发器函数。

CREATE OR REPLACE FUNCTION tri\_insert\_func() RETURNS TRIGGER AS

$$

DECLARE

BEGIN

insert into t\_react values(4,'before insert',sysdate);

RETURN NEW;

END

$$ LANGUAGE PLPGSQL;

3、创建触发器trigger\_insert。

CREATE TRIGGER trigger\_insert

BEFORE insert

ON t\_follows\_trigger

FOR EACH ROW

​

EXECUTE PROCEDURE tri\_insert\_func();

4、创建FOLLOWS触发器。

CREATE definer=vbadmin TRIGGER trigger1\_follows

BEFORE insert

ON t\_follows\_trigger

FOR EACH ROW

FOLLOWS trigger\_insert

BEGIN

insert into t\_react values(5,'after follows1',sysdate);

END;

/

CREATE definer=vbadmin TRIGGER trigger2\_follows

BEFORE insert

ON t\_follows\_trigger

FOR EACH ROW

FOLLOWS trigger\_insert

BEGIN

insert into t\_react values(6,'after follows2',sysdate);

END;

/

CREATE definer=vbadmin TRIGGER trigger3\_follows

BEFORE insert

ON t\_follows\_trigger

FOR EACH ROW

FOLLOWS trigger\_insert

BEGIN

insert into t\_react values(7,'after follows3',sysdate);

END;

/

5、执行INSERT操作。

INSERT INTO t\_follows\_trigger values(3,'wangwu',sysdate),(3,'wangwu1',sysdate);

6、验证结果。

select id,rep from (select \* from t\_follows\_trigger union all select \* from t\_react) order by insert\_time ;

结果返回如下：

id |     rep

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

 1 | zhangsan

 2 | lisi

 3 | wangwu

 3 | wangwu1

 4 | before insert

 7 | after follows3

 6 | after follows2

 5 | after follows1

 4 | before insert

 7 | after follows3

 6 | after follows2

 5 | after follows1

(12 rows)

**示例2：** 使用delimiter定义分隔符。

1、创建测试表并插入数据。

create table student1(student1ID int,student1NAME text);

insert into student1 values(1,'张三');

insert into student1 values(2,'李四');

create table Borrowstudent1(student1ID int,student1NAME text);

insert into Borrowstudent1 values(1,'张三');

insert into Borrowstudent1 values(2,'李四');

2、创建触发器，并在创建触发器之间指定分隔符。

delimiter //

create trigger tdustudent2 before update on student1 for each row

begin

declare i,j int default 1;

if new.student1ID!=old.student1ID then

update Borrowstudent1

set Borrowstudent1.student1ID=i+j+1

where Borrowstudent1.student1ID=old.student1ID;

end if;

end;

//

delimiter ;

返回结果为：

CREATE TRIGGER

SET

3、更新数据。

update student1 set student1ID=3 where student1NAME='李四';

select \* from student1;

select \* from Borrowstudent1;

更新student1的数据后，由于触发器成功执行，Borrowstudent1得到同步更新。查询student1和Borrowstudent1后的结果依次为：

student1ID | student1NAME

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

         1 | 张三

         3 | 李四

(2 rows)

​

student1ID | student1NAME

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

         1 | 张三

         3 | 李四

(2 rows)

#### DECLARE CONDITION

**功能描述**

在MySQL兼容模式下支持使用DECLARE ... CONDITION语句声明一个命名的错误条件，可以为错误号或者 SOLSTATE重命名，在异常处理中可以使用声明的 CONDITION。

**语法格式**

DECLARE condition\_name CONDITION FOR condition\_value

condition\_value:

{

mysql\_error\_code

| SQLSTATE [VALUE] sqlstate\_value

}

**参数说明**

* **condition\_name**

条件名称。

* **condition\_value**

指示要与条件名称相关联的特定条件或条件类别，支持以下形式：

* mysql\_error\_code：表示 MySQL 错误代码的整数文字，不支持使用MySQL错误代码0，因为这表示成功而不是错误条件。
* QLSTATE [VALUE]：一个 5 字符的字符串文字，指示 SQLSTATE 值。例如：SQLSTATE '22012'。

**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。
* MySQL中 mysql\_error\_code 与错误信息一一对应，在vastbase 中没有与之对应的关系，只做语法支持。
* 支持在触发器和匿名块中使用。

**示例**

**前置步骤**

1、创建兼容MySQL的库db\_mysql，并进入。

CREATE DATABASE db\_mysql dbcompatibility='B';

\c db\_mysql

2、设置serveroutput为on

set serveroutput = on;

**示例1：**存储过程begin内部声明处理程序，handler方法调用。

1、创建存储过程，begin内部定义condition处理程序。

create or replace procedure testpro() as

declare

a int;

begin

declare t\_condition condition for SQLSTATE '22012' ;

declare exit handler for t\_condition

begin

RAISE NOTICE 'SQLSTATE = %,SQLCODE = %,SQLERROR=%',SQLSTATE,returned\_sqlstate,message\_text;

end;

a = 1/0;

end;

/

2、调用存储过程

call testpro();

结果返回如下：

NOTICE: SQLSTATE = 22012,SQLCODE = 33816706,SQLERROR=division by zero

testpro

---------

(1 row)

**示例2：**存储过程begin内部声明处理程序，exception方法调用。

1、创建测试表。

create table test(id int,info text);

2、创建存储过程。

create or replace procedure testpro() as

declare

a int;

begin

declare t\_condition condition for SQLSTATE '22012' ;

a = 1/0;

EXCEPTION

WHEN t\_condition THEN

begin

insert into test values (1,'success');

RAISE NOTICE 'SQLSTATE = %,SQLCODE = %，SQLERRM = %',SQLSTATE,SQLCODE,SQLERRM;

end;

end;

/

3、调用存储过程。

call testpro();

结果返回如下：

NOTICE: SQLSTATE = 22012,SQLCODE = 33816706，SQLERRM = division by zero

testpro

---------

(1 row)

4、查询数据。

select \* from test;

结果返回如下：

id | info

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

1 | success

(1 row)

#### IGNORE|FORCE INDEX语法

**功能描述**

在MySQL兼容模式下，Vastbase G100支持使用IGNORE|FORCE INDEX语法为优化器提供有关如何在查询处理期间选择索引的信息。

* 使用IGNORE INDEX时，表示不强制使用某个索引来获取查询结果。
* 使用FORCE INDEX时，表示强制使用某个索引来获取查询结果。

**语法格式**

table\_name [[AS]alias] [index\_hint\_list]

index\_hint\_list:

index\_hint[index\_hint]...

index\_hint:

USE {INDEX|KEY}

[FOR {JOIN|ORDER BY|GROUP BY}]([index\_list])|{IGNORE|FORCE}{INDEX|KEY}(index\_list)

index\_list:

index\_name[,index\_name...]

**参数说明**

* **table\_name**

查询用的表名称。

* **[AS]alias**

可选项，表别名。

* **index\_hint**

索引提示。即如何选择索引信息。

* **index\_list**

索引列表。包含当前表的所有索引名（index\_name）。

**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。
* FORCE语法仅Vastbase G100 2.2 Build 11及以后版本支持使用，IGNORE语法仅Vastbase G100 2.2 Build 12及以后版本支持使用。
* 仅支持在SELECT语句中使用该语法。
* USE INDEX和FORCE INDEX不能同时使用。

**示例**

**前置条件：**创建并切换至兼容模式为B的数据库下。

CREATE DATABASE db\_1097149 DBCOMPATIBILITY 'B';

\c db\_mysql

**示例1：**使用force index指定索引。

1、创建测试表，并插入数据。

create table db\_1097149\_tb (col1 int ,col2 int,col3 int,col4 varchar(10));

insert into db\_1097149\_tb values(1,1,1,'a');

insert into db\_1097149\_tb values(1,2,2,'a');

insert into db\_1097149\_tb values(2,2,2,'a');

insert into db\_1097149\_tb values(2,2,3,'b');

insert into db\_1097149\_tb values(2,3,3,'b');

insert into db\_1097149\_tb values(3,3,4,'b');

insert into db\_1097149\_tb values(3,3,4,'a');

insert into db\_1097149\_tb values(3,4,5,'c');

insert into db\_1097149\_tb values(4,4,5,'c');

insert into db\_1097149\_tb values(4,null,1,'c');

2、创建索引。

create index index\_1097149\_1 on db\_1097149\_tb (col1);

create index index\_1097149\_2 on db\_1097149\_tb (col2);

create index index\_1097149\_3 on db\_1097149\_tb (col3);

create index index\_1097149\_4 on db\_1097149\_tb (col4);

3、更新表的统计信息。

analyze db\_1097149\_tb;

4、使用force key指定索引进行查询。

select \* from db\_1097149\_tb force key (index\_1097149\_2) where col2= 3;

select \* from db\_1097149\_tb force key (index\_1097149\_4) where col2= 3 and col4 = 'a';

select \* from db\_1097149\_tb FORCE key (index\_1097149\_1) where col2= 3;

返回结果为：

col1 | col2 | col3 | col4

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

   2 |    3 |    3 | b

   3 |    3 |    4 | b

   3 |    3 |    4 | a

(3 rows)

​

col1 | col2 | col3 | col4

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

   3 |    3 |    4 | a

(1 row)

​

col1 | col2 | col3 | col4

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

   2 |    3 |    3 | b

   3 |    3 |    4 | b

   3 |    3 |    4 | a

(3 rows)

5、查看步骤4中使用的查询执行计划。

explain (costs off)select \* from db\_1097149\_tb force key (index\_1097149\_2) where col2= 3;

explain (costs off)select \* from db\_1097149\_tb force key (index\_1097149\_4) where col2= 3 and col4 = 'a';

explain (costs off) select \* from db\_1097149\_tb force key (index\_1097149\_1) where col2= 3;

返回结果为：

                   QUERY PLAN

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

Index Scan using index\_1097149\_2 on db\_1097149\_tb

  Index Cond: (col2 = 3)

(2 rows)

​

                   QUERY PLAN

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

Index Scan using index\_1097149\_4 on db\_1097149\_tb

  Index Cond: ((col4)::text = 'a'::text)

  Filter: (col2 = 3)

(3 rows)

​

       QUERY PLAN

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

Seq Scan on db\_1097149\_tb

  Filter: (col2 = 3)

(2 rows)

**示例2：**ignore index使用唯一索引。

1、创建测试库并插入数据。

create table ignore\_table (col1 int ,col2 int,col3 int,col4 varchar(10),primary key(col1));

insert into ignore\_table values(1,1,1,'a');

insert into ignore\_table values(2,2,2,'a');

insert into ignore\_table values(3,3,2,'a');

insert into ignore\_table values(4,4,3,'b');

insert into ignore\_table values(5,5,3,'b');

insert into ignore\_table values(6,6,6,'b');

insert into ignore\_table values(7,7,7,'a');

insert into ignore\_table values(8,8,8,'c');

insert into ignore\_table values(9,9,9,'c');

insert into ignore\_table values(10,null,1,'c');

2、创建唯一索引。

create unique index index\_1130449 on ignore\_table (col2);

3、更新表的统计信息。

analyze ignore\_table;

4、使用ignore index进行查询。

select \* from ignore\_table ignore index (index\_1130449) where col2= 3;

select \* from ignore\_table ignore index (index\_1130449) where col2= 3 and col4 = 'a';

select \* from ignore\_table IGNORE INDEX (index\_1130449) where col2= 3;

查询结果依次为：

col1 | col2 | col3 | col4

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

   3 |    3 |    2 | a

(1 row)

​

col1 | col2 | col3 | col4

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

   3 |    3 |    2 | a

(1 row)

​

col1 | col2 | col3 | col4

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

   3 |    3 |    2 | a

(1 row)

5、查看步骤4中使用的查询执行计划。

explain (costs off )select \* from ignore\_table ignore index (index\_1130449) where col2= 3;

explain (costs off )select \* from ignore\_table ignore index (index\_1130449) where col2= 3 and col4 = 'a';

explain (costs off) select \* from ignore\_table IGNORE INDEX (index\_1130449) where col2= 3;

返回结果为：

       QUERY PLAN

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

Seq Scan on ignore\_table

  Filter: (col2 = 3)

(2 rows)

​

                     QUERY PLAN

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

Seq Scan on ignore\_table

  Filter: ((col2 = 3) AND ((col4)::text = 'a'::text))

(2 rows)

​

       QUERY PLAN

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

Seq Scan on ignore\_table

  Filter: (col2 = 3)

**示例3：**使用USE INDEX指定索引。

1、创建测试表，并插入数据。

create table db\_1097154\_tb (col1 int ,col2 int,col3 int,col4 varchar(10));

insert into db\_1097154\_tb values(1,1,1,'a');

insert into db\_1097154\_tb values(1,2,2,'a');

insert into db\_1097154\_tb values(2,2,2,'a');

insert into db\_1097154\_tb values(2,2,3,'b');

insert into db\_1097154\_tb values(2,3,3,'b');

insert into db\_1097154\_tb values(3,3,4,'b');

insert into db\_1097154\_tb values(3,3,4,'a');

insert into db\_1097154\_tb values(3,4,5,'c');

insert into db\_1097154\_tb values(4,4,5,'c');

insert into db\_1097154\_tb values(4,null,1,'c');

2、创建索引。

create index index\_1097154\_1 on db\_1097154\_tb (col1);

create index index\_1097154\_2 on db\_1097154\_tb (col2);

create index index\_1097154\_3 on db\_1097154\_tb (col3);

create index index\_1097154\_4 on db\_1097154\_tb (col4);

3、更新表的统计信息。

analyze db\_1097154\_tb;

4、使用USE INDEX指定索引进行查询。

select col2 from db\_1097154\_tb \* use key (index\_1097154\_2);

select \* from db\_1097154\_tb use key (index\_1097154\_2) where col2= 3;

select \* from db\_1097154\_tb use key (index\_1097154\_4) where col2= 3 and col4 = 'a';

select \* from db\_1097154\_tb USE key (index\_1097154\_2) where col2= 3;

返回结果为：

col2

------

   1

   2

   2

   2

   3

   3

   3

   4

   4

​

(10 rows)

​

col1 | col2 | col3 | col4

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

   2 |    3 |    3 | b

   3 |    3 |    4 | b

   3 |    3 |    4 | a

(3 rows)

​

col1 | col2 | col3 | col4

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

   3 |    3 |    4 | a

(1 row)

​

col1 | col2 | col3 | col4

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

   2 |    3 |    3 | b

   3 |    3 |    4 | b

   3 |    3 |    4 | a

(3 rows)

5、查看步骤4中使用的查询执行计划。

explain (costs off)select col2 from db\_1097154\_tb \* use key (index\_1097154\_2);

explain (costs off)select \* from db\_1097154\_tb use key (index\_1097154\_2) where col2= 3;

explain (costs off)select \* from db\_1097154\_tb use key (index\_1097154\_4) where col2= 3 and col4 = 'a';

explain (costs off) select \* from db\_1097154\_tb USE key (index\_1097154\_2) where col2= 3;

返回结果为：

                     QUERY PLAN

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

Index Only Scan using index\_1097154\_2 on db\_1097154\_tb

(1 row)

​

                   QUERY PLAN

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

Index Scan using index\_1097154\_2 on db\_1097154\_tb

  Index Cond: (col2 = 3)

(2 rows)

​

                   QUERY PLAN

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

Index Scan using index\_1097154\_4 on db\_1097154\_tb

  Index Cond: ((col4)::text = 'a'::text)

  Filter: (col2 = 3)

(3 rows)

​

                   QUERY PLAN

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

Index Scan using index\_1097154\_2 on db\_1097154\_tb

  Index Cond: (col2 = 3)

(2 rows)

#### DELETE

**功能描述**

DELETE语法用于从指定的表里删除满足WHERE子句的行。如果WHERE子句不存在，将删除表中所有行，结果只保留表结构。

Vastbase G100在MySQL兼容模式下，支持多表删除。

其余关于DELETE的用法请参考《[VASTDATA]Vastbase G100 V2.2 开发者指南》->DELETE章节。

**语法格式**

[ WITH [ RECURSIVE ] with\_query [, ...] ]

DELETE [/\*+ plan\_hint \*/] [FROM]

{[ ONLY ] table\_name [ \* ] [ [ [partition\_clause] [ [ AS ] alias ] ] | [ [ [ AS ] alias ] [partitions\_clause] ] ]} [, ...]

[ USING using\_list ]

[ WHERE condition];

**或**

[ WITH [ RECURSIVE ] with\_query [, ...] ]

DELETE [/\*+ plan\_hint \*/]

{[ ONLY ] table\_name [ \* ] [ [ [partition\_clause] [ [ AS ] alias ] ] | [ [ [ AS ] alias ] [partitions\_clause] ] ]} [, ...]

[ FROM using\_list ]

[ WHERE condition ];

**参数说明**

**using\_list**

using子句。using\_list指定关联表的集合时可以同时出现目标表，并且可以定义表的别名并在目标表中使用。其他模式下则目标表不可重复出现在using\_list中。

其他参数请参考《[VASTDATA]Vastbase G100 V2.2 开发者指南》->DELETE章节。

**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。

**示例**

**前置步骤：**创建兼容模式为MySQL的库db\_mysql，并进入。

create database db\_mysql dbcompatibility 'B';

\c db\_mysql

1、创建测试表并插入数据。

create table tb\_1132876(c1 int,c2 int,c3 date,c4 varchar(20),c5 varchar(20));

insert into tb\_1132876 values(1 ,1 ,'1990-12-11','A1','B1');

insert into tb\_1132876 values(2 ,2 ,'1990-12-12','A2 ','B2 ');

create table tb\_1132876\_1 (col1 int,col2 int,col3 date,col4 varchar(20),col5 varchar(20));

insert into tb\_1132876\_1 values(1 ,1 ,'1990-12-11','A1','B1');

insert into tb\_1132876\_1 values(2 ,2 ,'1990-12-12','A2 ','B2 ');

insert into tb\_1132876\_1 values(3 ,3 ,'1990-12-13','A3 ','B3 ');

insert into tb\_1132876\_1 values(4 ,4 ,'1990-12-14','A4 ','B4 ');

2、执行多表删除，使用如下命令删除表tb\_1132876中从c1=2或者c2=2的数据，并且删除表tb\_1132876\_1表中全部数据。

delete from tb\_1132876 t1,tb\_1132876\_1 t2 where t1.c1 in (select col1 from tb\_1132876\_1 where col2 = 2) or c2 = 2 ;

3、查询tb\_1132876表中数据。

select \* from tb\_1132876;

返回结果为：

c1 | c2 | c3 | c4 | c5

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

1 | 1 | 1990-12-11 | A1 | B1

(1 row)

4、查询tb\_1132876\_1表中数据。

select \* from tb\_1132876\_1;

返回结果为：

c1 | c2 | c3 | c4 | c5

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

(0 row)

5、使用如下命令可删除两张表中的全部数据。

delete from tb\_1132876 t1,tb\_1132876\_1 t2;

#### INNER JOIN

**功能描述**

在MySQL兼容模式下支持INNER JOIN连接条件使用WHERE子句。

**语法格式**

SELECT col\_name FROM table1 INNER JOIN table2 WHERE condition

**参数说明**

* **col\_name**

字段名。

* **table1、table2**

内连接的表名。

* **condition**

条件表达式。

**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。

**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、创建测试表并插入数据。

create table test1(id int,name varchar(10),score numeric,date1 date,c1 bytea);

insert into test1 values(1,'aaa',97.1,'1999-12-12','0101');

insert into test1 values(5,'bbb',36.9,'1998-01-12','0110');

insert into test1 values(3,'ccc',89.2,'2200-03-12','0111');

insert into test1 values(7,'uuu',99.9,'1987-01-01','1000');

insert into test1 values(30,'ooo',90.1,'2023-01-30','1001');

insert into test1 values(6,'hhh',60,'2022-12-22','1010');

insert into test1 values(7,'fff',71,'2001-11-23','1011');

insert into test1 values(-1,'yaya',77.7,'2008-09-10','1100');

insert into test1 values(7,'fff',71,'2001-11-23','1011');

create table test2(id int,name varchar(10),score numeric,date1 date,c1 bytea);

insert into test2 values(1,'aaa',99.1,'1998-12-12','0101');

insert into test2 values(2,'hhh',36.9,'1996-01-12','0110');

insert into test2 values(3,'ddd',89.2,'2000-03-12','0111');

insert into test2 values(7,'uuu',60.9,'1997-01-01','1000');

insert into test2 values(9,'ooo',90.1,'2021-01-30','1001');

insert into test2 values(6,'hhh',90,'2022-10-22','1010');

insert into test2 values(11,'eee',71,'2011-11-20','1011');

insert into test2 values(-1,'yaya',76.7,'2008-09-10','1100');

insert into test2 values(7,'uuu',60.9,'1997-01-01','1000');

3、使用where指定连接条件查询数据。

select \* from test1 inner join test2 where test1.id=test2.id and test1.score>60;

结果返回如下：

id | name | score | date1 | c1 | id | name | score | date1 | c1

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

1 | aaa | 97.1 | 1999-12-12 | \x30313031 | 1 | aaa | 99.1 | 1998-12-12 | \x30313031

3 | ccc | 89.2 | 2200-03-12 | \x30313131 | 3 | ddd | 89.2 | 2000-03-12 | \x30313131

7 | fff | 71 | 2001-11-23 | \x31303131 | 7 | uuu | 60.9 | 1997-01-01 | \x31303030

7 | fff | 71 | 2001-11-23 | \x31303131 | 7 | uuu | 60.9 | 1997-01-01 | \x31303030

7 | uuu | 99.9 | 1987-01-01 | \x31303030 | 7 | uuu | 60.9 | 1997-01-01 | \x31303030

-1 | yaya | 77.7 | 2008-09-10 | \x31313030 | -1 | yaya | 76.7 | 2008-09-10 | \x31313030

7 | fff | 71 | 2001-11-23 | \x31303131 | 7 | uuu | 60.9 | 1997-01-01 | \x31303030

7 | fff | 71 | 2001-11-23 | \x31303131 | 7 | uuu | 60.9 | 1997-01-01 | \x31303030

7 | uuu | 99.9 | 1987-01-01 | \x31303030 | 7 | uuu | 60.9 | 1997-01-01 | \x31303030

(9 rows)

4、缺省INNER关键字查询数据。

select \* from test1 join test2 where test1.date1=test2.date1;

结果返回如下：

id | name | score | date1 | c1 | id | name | score | date1 | c1

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

-1 | yaya | 77.7 | 2008-09-10 | \x31313030 | -1 | yaya | 76.7 | 2008-09-10 | \x31313030

(1 row)

5、查询具体列值。

select test1.score from test2 join test1 where test2.id=test1.id and test2.name='yaya';

结果返回如下：

score

-------

77.7

(1 row)

6、使用distinct关键字返回唯一不同的值。

select distinct test1.id from test1 inner join test2 where test1.id=test2.id;

结果返回如下：

id

----

1

3

6

7

-1

(5 rows)

7、使用order by语句。

select \* from test1 inner join test2 where test1.id=test2.id order by 1,2;

结果返回如下：

id | name | score | date1 | c1 | id | name | score | date1 | c1

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

-1 | yaya | 77.7 | 2008-09-10 | \x31313030 | -1 | yaya | 76.7 | 2008-09-10 | \x31313030

1 | aaa | 97.1 | 1999-12-12 | \x30313031 | 1 | aaa | 99.1 | 1998-12-12 | \x30313031

3 | ccc | 89.2 | 2200-03-12 | \x30313131 | 3 | ddd | 89.2 | 2000-03-12 | \x30313131

6 | hhh | 60 | 2022-12-22 | \x31303130 | 6 | hhh | 90 | 2022-10-22 | \x31303130

7 | fff | 71 | 2001-11-23 | \x31303131 | 7 | uuu | 60.9 | 1997-01-01 | \x31303030

7 | fff | 71 | 2001-11-23 | \x31303131 | 7 | uuu | 60.9 | 1997-01-01 | \x31303030

7 | fff | 71 | 2001-11-23 | \x31303131 | 7 | uuu | 60.9 | 1997-01-01 | \x31303030

7 | fff | 71 | 2001-11-23 | \x31303131 | 7 | uuu | 60.9 | 1997-01-01 | \x31303030

7 | uuu | 99.9 | 1987-01-01 | \x31303030 | 7 | uuu | 60.9 | 1997-01-01 | \x31303030

7 | uuu | 99.9 | 1987-01-01 | \x31303030 | 7 | uuu | 60.9 | 1997-01-01 | \x31303030

(10 rows)

#### INSERT IGNORE

**功能描述**

Vastbase G100在MySQL兼容模式下，支持在INSERT语句中使用IGNORE选项。

用户在执行INSERT语句的时候，如果发生主键或唯一索引冲突，SQL语句是会引发ERROR报错，如果使用IGNORE选项，则忽略在执行SQL语句时发生的可忽略的错误，将Error降级为Warning，且继续语句的执行，不会影响其他数据的操作。能使Error降级的场景有：

1. 违反非空约束时：

若执行的SQL语句违反了表的非空约束，使用IGNORE选项可将Error降级为Warning，并根据GUC参数sql\_ignore\_strategy的值采用以下策略的一种继续执行：

* sql\_ignore\_strategy为ignore\_null时，忽略违反非空约束的行的INSERT操作，并继续执行剩余数据操作。
* sql\_ignore\_strategy为overwrite\_null时，将违反约束的null值覆写为目标类型的默认值，并继续执行剩余数据操作。

【说明】

GUC参数sql\_ignore\_strategy相关信息请参考《VASTDATA]Vastbase G100 V2.2 开发者指南》->其它选项。

2、违反唯一约束时：

若执行的SQL语句违反了表的唯一约束，使用IGNORE选项可将Error降级为Warning，忽略违反约束的行的INSERT操作，并继续执行剩余数据操作。

3、分区表无法匹配到合法分区时：

在对分区表进行INSERT操作时，若某行数据无法匹配到表格的合法分区，使用IGNORE选项可将Error降级为Warning，忽略该行操作，并继续执行剩余数据操作。

4、插入值向目标列类型转换失败时：

执行INSERT语句时，若发现新值与目标列类型不匹配，使用IGNORE选项可将Error降级为Warning，并根据新值与目标列的具体类型采取以下策略的一种继续执行：

* 当新值类型与列类型同为数值类型时： 若新值在列类型的范围内，则直接进行插入；若新值在列类型范围外，则以列类型的最大/最小值替代。
* 当新值类型与列类型同为字符串类型时：若新值长度在列类型限定范围内，则以直接进行插入；若新值长度在列类型的限定范围外，则保留列类型长度限制的前n个字符。
* 若遇到新值类型与列类型不可转换时：插入列类型的默认值。

例如，如果没有IGNORE，表中现有主键或唯一索引重复将会导致重复键错误，并且语句将中止。使用IGNORE选项，该行将被丢弃并且不会发生错误，忽略的错误会生成警告。

其它关于INSERT的相关内容请参考《VASTDATA]Vastbase G100 V2.2 开发者指南》->INSERT章节。

**语法格式**

INSERT [/\*+ plan\_hint \*/] [IGNORE] INTO table\_name ...

**参数说明**

* **IGNORE**

使用该参数，如果发生了主键或唯一索引冲突，则会忽略错误并发出一个WARNING。

* **table\_name**

表名。

**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。
* IGNORE支持分区表。

**示例**

**前置步骤：**创建兼容模式为MySQL的库db\_mysql，并进入。

create database db\_mysql dbcompatibility 'B';

\c db\_mysql

1、创建一张测试表，主键id。

CREATE TABLE table\_name(

id int NOT NULL,

name varchar(50) DEFAULT NULL,

age int DEFAULT NULL,

PRIMARY KEY (id)

);

2、插入一条数据。

insert into table\_name values(1,'tom', 20);

3、插入一条数据，id为1，不加ignore（主键冲突）。

insert into table\_name values(1,'Bill',21);

返回结果如下，报主键冲突的错误：

ERROR: duplicate key value violates unique constraint "table\_name\_pkey"

DETAIL: Key (id)=(1) already exists.

4、加上ignore之后插入数据。

insert ignore into table\_name values(1,'Bill',21);

返回结果如下，会有一个WARNING警告：

WARNING: duplicate key value violates unique constraint in table "table\_name"

INSERT 0 0

5、插入多行数据。

insert ignore into table\_name values(1,'Bill',21),(2,'cim',21);

返回结果为：

WARNING: duplicate key value violates unique constraint in table "table\_name"

INSERT 0 1

6、查询表，发现插入的数据(1,'Bill',21)被忽略了，(2,'cim',21)数据成功插入。

select \* from table\_name;

返回结果为：

id | name | age

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

1 | tom | 20

2 | cim | 21

(2 rows)

#### INSERT INTO VALUE

**功能描述**

Vastbase G100在MySQL兼容模式下支持INSERT INTO语法使用value插入数据。

**语法格式**

INSERT INTO table\_name VALUE (expression)[,...];

**参数说明**

* **table\_name**

要插入数据的目标表名。

* **expression**

要插入目标表的有效表达式或值。

**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。

**示例**

**前置步骤：**创建兼容模式为MySQL的库db\_mysql，并进入。

create database db\_mysql dbcompatibility 'B';

\c db\_mysql

1、创建测试表。

create table tb\_1117689(col1 int ,col2 int default 3);

1. 插入数据。

insert into tb\_1117689 value(1,1),(2,2);

1. 查询数据。

select \* from tb\_1117689 order by 1,2;

返回结果为：

col1 | col2

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

1 | 1

2 | 2

(2 rows)

#### UPDATE

**功能描述**

UPDATE语法用于更新表中的数据。Vastbase G100在MySQL兼容模式下，支持多表更新。。

其余关于UPDATE的用法请参考《[VASTDATA]Vastbase G100 V2.2 开发者指南》->UPDATE章节。

**语法格式**

[ WITH [ RECURSIVE ] with\_query [, ...] ]

UPDATE [/\*+ plan\_hint \*/] table\_list

SET {column\_name = { expression | DEFAULT }

|( column\_name [, ...] ) = {( { expression | DEFAULT } [, ...] ) |sub\_query }}[, ...]

[ FROM from\_list] [ WHERE condition ];

where sub\_query can be：

SELECT [ ALL | DISTINCT [ ON ( expression [, ...] ) ] ]

{ \* | {expression [ [ AS ] output\_name ]} [, ...] }

[ FROM from\_item [, ...] ]

[ WHERE condition ]

[ GROUP BY grouping\_element [, ...] ]

[ HAVING condition [, ...] ]

[ ORDER BY {expression [ [ ASC | DESC | USING operator ] | nlssort\_expression\_clause ] [ NULLS { FIRST | LAST } ]} [, ...] ]

[ LIMIT { [offset,] count | ALL } ]

**参数说明**

**table\_list**

一个表的表达式列表，与from\_list类似，但可以同时声明目标表和关联表，仅在多表更新语法中使用。

其他参数请参考《[VASTDATA]Vastbase G100 V2.2 开发者指南》->UPDATE章节。

**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。

**示例**

**前置步骤：**创建兼容模式为MySQL的库db\_mysql，并进入。

create database db\_mysql dbcompatibility 'B';

\c db\_mysql

1、创建测试表并插入数据。

create table tb\_1132824(c1 int,c2 int,c3 date,c4 varchar(20),c5 varchar(20));

insert into tb\_1132824 values(1 ,1 ,'1990-12-11','A1','B1');

insert into tb\_1132824 values(2 ,2 ,'1990-12-12','A2 ','B2 ');

insert into tb\_1132824 values(3 ,3 ,'1990-12-13','A3 ','B3 ');

create table tb\_1132824\_1 (col1 int,col2 int,col3 date,col4 varchar(20),col5 varchar(20));

insert into tb\_1132824\_1 values(1 ,1 ,'1990-12-11','A1','B1');

insert into tb\_1132824\_1 values(2 ,2 ,'1990-12-12','A2 ','B2 ');

insert into tb\_1132824\_1 values(3 ,3 ,'1990-12-13','A3 ','B3 ');

insert into tb\_1132824\_1 values(4 ,4 ,'1990-12-14','A4 ','B4 ');

2、执行多表更新。

update tb\_1132824 t1,tb\_1132824\_1 t2 set t1.c2 = 10 ,t2.col3='2023-12-31';

3、查询表tb\_1132824中数据。

select \* from tb\_1132824;

返回结果如下所示，c2列内容已经被更新为10：

c1 | c2 | c3 | c4 | c5

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

1 | 10 | 1990-12-11 | A1 | B1

2 | 10 | 1990-12-12 | A2 | B2

3 | 10 | 1990-12-13 | A3 | B3

(3 rows)

4、查询表tb\_1132824\_1中数据。

select \* from tb\_1132824\_1;

返回结果如下所示，col3列内容已经被更新为2023-12-31：

col1 | col2 | col3 | col4 | col5

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

1 | 1 | 2023-12-31 | A1 | B1

2 | 2 | 2023-12-31 | A2 | B2

3 | 3 | 2023-12-31 | A3 | B3

4 | 4 | 2023-12-31 | A4 | B4

(4 rows)

5、执行如下语句进行多表更新。

update tb\_1132824,tb\_1132824\_1 set tb\_1132824.c2 = col1+10 ,tb\_1132824\_1.col3=c3 where c1=col1;

6、查询表tb\_1132824中数据。

select \* from tb\_1132824;

返回结果如下所示，c2列的内容已经被更新为col1+10：

c1 | c2 | c3 | c4 | c5

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

1 | 11 | 1990-12-11 | A1 | B1

2 | 12 | 1990-12-12 | A2 | B2

3 | 13 | 1990-12-13 | A3 | B3

(3 rows)

7、查询表tb\_1132824\_1中数据。

select \* from tb\_1132824\_1 order by col1;

返回结果如下所示，当c1=col1时，col3列内容已经被更新为表tb\_1132824中的对应内容：

col1 | col2 | col3 | col4 | col5

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

1 | 1 | 1990-12-11 | A1 | B1

2 | 2 | 1990-12-12 | A2 | B2

3 | 3 | 1990-12-13 | A3 | B3

4 | 4 | 2023-12-31 | A4 | B4

(4 rows)

#### 关联的同名字段不需要指定表名

**功能描述**

Vastbase G100在MySQL兼容模式下，支持多表关联时同名字段不需要指定表名。当存在多表关联的时，对于同名字段，group by和order by 后可以省略掉表名，表名默认跟目标列中出现的表名保持一致。

**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。
* 使用前需将参数"vastbase\_sql\_mode"设置为空。

**示例**

1、创建兼容模式为MySQL的库db\_mysql，并进入。

create database db\_mysql dbcompatibility 'B';

\c db\_mysql

2、设置"vastbase\_sql\_mode"为空。

set vastbase\_sql\_mode='';

3、创建测试表，两张表中存在同名字段"customernumber"。

create table customers(

customernumber integer NOT NULL,

customername varchar(50) NOT NULL,

contactlastname varchar(50) NOT NULL,

contactfirstname varchar(50) NOT NULL,

addressline1 varchar(50) NOT NULL,

addressline2 varchar(50),

city varchar(50) NOT NULL,

state varchar(50),

postalcode varchar(15),

country varchar(50) NOT NULL,

salesrepemployeenumber integer,

creditlimit numeric(10,2))

WITH (orientation=row,compression=no,fillfactor=80);

CREATE TABLE orders(

ordernumber integer(32) NOT NULL,

orderdate date NOT NULL,

requireddate date NOT NULL,

shippeddate date,

status varchar(15) NOT NULL,

comments text,

customernumber integer(32) NOT NULL

);

4、插入测试数据。

insert into customers values('103','Atelier graphique', 'Schmitt','Carine','54,rue Royale',NULL,'Nantes',NULL,'44000','France','1370','21000');

insert into customers values('112','Signal Gift Stores','king','Jean','8489 Strong St.',NULL,'Las Vegas',NULL,'83030','USA','1166','71800');

insert into customers values('114','Australian Collectors','Co.','Ferguson Peter','636 St Kilda Road','Level 3','Melbourne','Victoria','3004','Australia','1611','117300');

insert into orders values('10100','2013-01-06','2013-01-13','2013-01-10','Shipped',NULL,'103');

insert into orders values('10102','2013-01-10','2013-01-18','2013-01-14','shipped',NULL,'112');

insert into orders values('10101','2013-01-09','2013-01-18','2013-01-11','shipped','check on availability.','114');

5、使用group by、having、order by不指定表名查询。

SELECT

customers.customerNumber,

customerName,

orderDate,

LEAD(orderDate,1) OVER (

PARTITION BY customerNumber

ORDER BY orderDate ) nextOrderDate

FROM

orders

INNER JOIN customers USING (customerNumber)

group by customerNumber,orderDate

having customerNumber

order by customerNumber;

结果返回如下：

customerNumber | customerName | orderDate | nextOrderDate

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

103 | Atelier graphique | 2013-01-06 |

112 | Signal Gift Stores | 2013-01-09 |

114 | Australian Collectors | 2013-01-10 |

(3 rows)

#### 无关联条件JOIN

**功能描述**

Vastbase G100在MySQL兼容模式下，支持JOIN不加关联条件，返回结果为笛卡尔积的形式。

**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。

**示例**

**前置步骤：**创建兼容模式为MySQL的库db\_mysql，并进入。

create database db\_mysql dbcompatibility 'B';

\c db\_mysql

1、创建两张测试表并插入数据。

create table student(sid int, name varchar(30), class\_id int);

create table class (cid int, class\_name varchar(30));

insert into student values(1,'a1',100);

insert into student values(2,'a2',101);

insert into class values(100,'class1');

insert into class values(101,'class2');

insert into class values(102,'class3');

2、有关联条件JOIN。

select \* from student inner join class ;

返回结果为：

sid | name | class\_id | cid | class\_name

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

1 | a1 | 100 | 100 | class1

2 | a2 | 101 | 100 | class1

1 | a1 | 100 | 101 | class2

2 | a2 | 101 | 101 | class2

1 | a1 | 100 | 102 | class3

2 | a2 | 101 | 102 | class3

(6 rows)

3、Vastbase在MySQL兼容模式下支持无关联条件的JOIN，也就是说下面语句等价于步骤2，返回结果也与步骤2相同。

select \* from student,class ;

#### 逃逸字符为反斜线时需要双写

**背景信息**

Vastbase G100支持使用SQL LIKE操作符实现模式匹配。通过LIKE语法来判断字符串是否能匹配上LIKE后的模式字符串。如果字符串与提供的模式匹配，则LIKE表达式返回为真，否则返回为假。

当LIKE中要查询特殊字符“%”、“\_”、“\”的时候需要使用反斜杠“\”来进行转义。

在进行模糊匹配时：

* 下划线“\_”代表（匹配）任何单个字符；百分号“%”代表任意串的通配符。
* 如果要匹配文本里的下划线或者百分号，在提供的模式里相应字符必须前导逃逸字符。逃逸字符的作用是禁用元字符的特殊含义，缺省的逃逸字符是反斜线“\”，也可以用ESCAPE子句指定一个不同的逃逸字符。
* 要匹配逃逸字符本身，需要写两个逃逸字符。例如要写一个包含反斜线的模式常量，需要在SQL语句里写两个反斜线。

【说明】

更多模式匹配的相关信息参见《Vastbase G100 V2.2 开发者指南》->SQL语法参考->SQL基本要素->函数和操作符->模式匹配操作符。

**功能描述**

参数standard\_conforming\_strings用于控制普通字符串文本'…'中是否按照SQL标准把反斜线当成了普通文本。设置为off时，在文串常量中写的任何反斜线都需要被双写。因此，写一个匹配单个反斜线的模式实际上要在语句里写四个反斜线（可以通过用ESCAPE选择一个不同的逃逸字符来避免这种情况，这样反斜线就不再是LIKE的特殊字符了。但仍然是字符文本分析器的特殊字符，所以仍需要两个反斜线）。

**注意事项**

* 除了反斜线双写的情况下，逃逸字符只能是单个字符。
* 参数standard\_conforming\_strings的默认值为on。
* 在兼容MySQL数据模式时，用户也可以通过写`ESCAPE''`的方式不选择逃逸字符，这样可以有效地禁用逃逸机制，但是没有办法关闭下划线和百分号在模式中的特殊含义。

**示例**

**示例1：**参数standard\_conforming\_strings为on时，逃逸字符为反斜线不用双写。

1、设置standard\_conforming\_strings参数值为on。

set standard\_conforming\_strings = on;

2、查看参数值是否设置成功。

show standard\_conforming\_strings;

返回结果为如下，设置成功。

standard\_conforming\_strings

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

on

(1 row)

3、模糊查询，逃逸字符使用反斜线（反斜线不双写）。

SELECT '%A\_BBCC' LIKE '\%A\_B%' ESCAPE '\' AS RESULT;

返回结果为真，匹配成功。

result

--------

t

(1 row)

4、模糊查询，逃逸字符使用反斜线（反斜线双写）。

SELECT 'AA\_BBCC' LIKE '\%A\_B%' ESCAPE '\\' AS RESULT;

返回结果如下，逃逸字符不合法。

ERROR: invalid escape string

HINT: Escape string must be empty or one character.

CONTEXT: referenced column: result

**示例2：**参数standard\_conforming\_strings为off时，在文串常量中写的任何反斜线都需要被双写。

【说明】

standard\_conforming\_strings为off时，使用ESCAPE '\'的写法会有编译错误。

1、设置参数standard\_conforming\_strings的值为off。

set standard\_conforming\_strings to off;

2、查看参数值是否设置成功。

show standard\_conforming\_strings;

返回结果如下，设置成功。

standard\_conforming\_strings

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

off

(1 row)

3、设置参数escape\_string\_warning 的值为off 。

set escape\_string\_warning = off ;

【说明】

参数escape\_string\_warning打开时，如果在普通字符串文本'…'中出现了一个反斜线（\）并且standard\_conforming\_strings为off，那么就会发出一个警告。该参数默认值是on。

4、模糊查询，逃逸字符为反斜线。文串常量中写的任何反斜线都需要被双写。

SELECT '\A\_BBCC' LIKE '\A%' ESCAPE '\\' AS RESULT;

返回结果为真，匹配成功。

result

--------

t

(1 row)

#### 支持在UNION [ALL] 语句中使用ORDER BY 子句

**功能描述**

Vastbase G100在MySQL兼容模式下支持使用UNION [ALL]语句将两个或多个查询结果合并，并通过ORDER BY子句排序。

**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。

**示例**

1、创建并切换至兼容模式为MySQL的数据库db\_mysql。

CREATE DATABASE db\_mysql dbcompatibility='B';

\c db\_mysql

2、创建测试表并插入数据。

CREATE TABLE a(id int,msg text);

CREATE TABLE b(id int,msg text);

INSERT INTO a VALUES(1,'a');

INSERT INTO a VALUES(2,'b');

INSERT INTO b VALUES(1,'a');

INSERT INTO b VALUES(2,'b');

3、使用UNION ALL语句返回所有查询结果并使用ORDER BY子句排序。

SELECT \* FROM a UNION ALL SELECT \* FROM b ORDER BY id;

结果返回如下：

id | msg

----+-----

1 | a

1 | a

2 | b

2 | b

(4 rows)

4、使用UNION 语句压缩重复结果返回数据并使用ORDER BY子句排序。

SELECT \* FROM a UNION SELECT \* FROM b ORDER BY id;

结果返回如下：

id | msg

----+-----

1 | a

2 | b

(2 rows)

* + 1. **函数**

#### 时间和日期处理函数

##### CURRENT\_DATE

**功能描述**

Vastbase在MySQL兼容模式下支持使用CURRENT\_DATE函数获取当前日期（不包含时分秒）。

**语法格式**

CURRENT\_DATE()

**参数说明**

无。

**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。

**示例**

**前置步骤：**创建兼容模式为MySQL的库db\_mysql，并进入。

create database db\_mysql dbcompatibility 'B';

\c db\_mysql

**示例1：**直接调用CURRENT\_DATE函数。

select CURRENT\_DATE;

返回结果为：

date

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

2023-02-07

(1 row)

**示例2：**在DML语句中使用CURRENT\_DATE函数。

1. 创建测试表并插入数据。

create table test(id int,col date default current\_date());

insert into test values(1,'2022-01-07');

1. 向test表中插入一条数据。

insert into test values(2,current\_date());

1. 更新id为1的那条数据。

update test set col=current\_date()-30 where id=1;

1. 查询数据（假设当前日期为2023-02-07）。

select \* from test order by id;

返回结果为：

id | col

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

1 | 2023-01-08

2 | 2023-02-07

(2 rows)

##### DATE

**功能描述**

Vastbase G100在MySQL兼容模式下，支持使用DATE函数提取日期或日期时间表达式的中的日期部分。

如果指定的表达式不是一个合法的日期或者日期时间，DATE函数将返回NULL。如果输入参数为NULL，函数也返回 NULL。

**语法格式**

DATE(expr)

**参数说明**

**expr**

输入的日期表达式，该参数为必填项。

**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。

**示例**

**前置步骤：**创建兼容模式为MySQL的库db\_mysql，并进入。

create database db\_mysql dbcompatibility 'B';

\c db\_mysql

**示例1：**直接调用date函数。

select date('2022-01-01 12:30:00');

返回结果为：

date

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

2022-01-01

(1 row)

**示例2：**在DML语句中使用DATE函数。

create table t\_date(id int, col1 date, col2 timestamp with time zone);

insert into t\_date values(1,date '2022-01-01',timestamp '2022-01-01 12:30:00.666 +08');

1. 在INSERT语句中使用DATE函数。

insert into t\_date values(2,date('2022-01-03'),date(timestamp '2022-01-03 12:30:00.666 +08'));

1. 在UPDATE语句中使用DATE函数。

update t\_date set col1=date('2022-02-01 12:00:00') where id=1;

1. 查询数据。

select \* from t\_date;

返回结果为：

id | col1 | col2

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

2 | 2022-01-03 | 2022-01-03 00:00:00+08

1 | 2022-02-01 | 2022-01-01 12:30:00.666+08

(2 rows)

##### DATEDIFF

**功能描述**

Vastbase G100在MySQL兼容模式下支持DATEDIFF函数，用于返回两个日期之间的天数。函数返回date1 - date2的计算结果，date1和date2两个参数必须是有效的日期或日期时间值。

如果输入参数传递的是日期时间值，则DATEDIFF函数仅将日期部分用于计算，并忽略时间部分。

**语法格式**

DATEDIFF(date1,date2)

**参数说明**

* **date1**

输入的日期时间1。

* **date2**

输入的日期时间2。

**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。

**示例**

**前置步骤：**创建兼容模式为MySQL的库db\_mysql，并进入。

create database db\_mysql dbcompatibility 'B';

\c db\_mysql

**示例1：**直接调用DATEDIFF函数。

select datediff(date'2020-2-29',date'2020-3-29');

返回结果为：

datediff

----------

-29

(1 row)

**示例2：**在存储过程中使用DATEDIFF函数。

1. 设置serveroutput 为on（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）。

set serveroutput on;

1. 调用存储过程（当两个输入参数日期值相等时）。

call p\_1131144('2020-1-1','2020-01-01');

返回结果为：

相等

p\_1131144

---------

(1 row)

1. 调用存储过程（当输入的第一个日期值小于第二个日期值，返回第二个日期值）。

call p\_1131144('2021-1-1','2022-1-1');

返回结果为：

2022-1-1

p\_1131144

-----------

(1 row)

1. 调用存储过程（当输入的第一个日期值大于第二个日期值，返回第一个日期值）。

call p\_1131144('2020-1-1','2019-1-1');

返回结果为：

2020-1-1

p\_1131144

-----------

(1 row)

##### UNIX\_TIMESTAMP

**功能描述**

Vastbase G100在MySQL兼容模式下，支持用户通过unix\_timestamp函数获取时间戳。函数的调用有以下两种情况：

* unix\_timestamp()不带date参数调用，返回值为一个unix时间戳，代表的是自'1970-01-01 00:00:00'UTC以来的秒数。
* unix\_timestamp()使用date参数调用，返回值为自'1970-01-01 00:00:00'UTC以来的秒数。

**语法格式**

unix\_timestamp([date])

**注意事项**

**date**

表示时间，为timestamptz字符串。

**示例**

**前置步骤：**创建并切换至兼容模式为MySQL的数据库db\_mysql。

CREATE DATABASE db\_mysql dbcompatibility='B';

\c db\_mysql

**示例1：**在系统表中查询函数信息。

\x    --列式展示查询结果

select \* from pg\_proc where proname='unix\_timestamp';

查询结果为：

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

proname             | unix\_timestamp

pronamespace        | 11

proowner            | 10

prolang             | 12

procost             | 1

prorows             | 0

provariadic         | 0

protransform        | -

proisagg            | f

proiswindow         | f

prosecdef           | f

proleakproof        | f

proisstrict         | t

proretset           | f

provolatile         | v

pronargs            | 0

pronargdefaults     | 0

prorettype          | 23

proargtypes         |

proallargtypes      |

proargmodes         |

proargnames         |

proargdefaults      |

prosrc              | unix\_timestamp

probin              |

proconfig           |

proacl              |

prodefaultargpos    |

fencedmode          | f

proshippable        |

propackage          | f

prokind             | f

proargsrc           |

propackageid        | 0

proisprivate        | f

prosrcoffset        | 0

proargtypesext      |

prodefaultargposext |

allargtypes         |

allargtypesext      |

protypeoid          | 0

protypekind         |

isfinal             | f

-[ RECORD 2 ]-------+---------------------------

proname             | unix\_timestamp

pronamespace        | 11

proowner            | 10

prolang             | 12

procost             | 1

prorows             | 0

provariadic         | 0

protransform        | -

proisagg            | f

proiswindow         | f

prosecdef           | f

proleakproof        | f

proisstrict         | t

proretset           | f

provolatile         | i

pronargs            | 1

pronargdefaults     | 0

prorettype          | 23

proargtypes         | 1184

proallargtypes      |

proargmodes         |

proargnames         |

proargdefaults      |

prosrc              | unix\_timestamp\_timestamptz

probin              |

proconfig           |

proacl              |

prodefaultargpos    |

fencedmode          | f

proshippable        |

propackage          | f

prokind             | f

proargsrc           |

propackageid        | 0

proisprivate        | f

prosrcoffset        | 0

proargtypesext      |

prodefaultargposext |

allargtypes         | 1184

allargtypesext      |

protypeoid          | 0

protypekind         |

isfinal             | f

**示例2：**将unix\_timestamp()值作为字段默认值。

1、创建测试表。

CREATE TABLE t\_unix\_timestamp(c1 int4 default unix\_timestamp('2023-02-03 10:00:00'));

2、向测试表中插入default值，并查询表记录。

insert into t\_unix\_timestamp values(default);

select \* from t\_unix\_timestamp;

查询结果为：

    c1

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

1675418400

(1 row)

#### 字符和数据类型处理函数

##### CONV

**功能描述**

CONV函数用于将数值从一种进制转换为另一种进制，函数返回值为目标进制字符串的形式。

**语法格式**

CONV(N,from\_base,to\_base)

**参数说明**

* **N**

需要转换进制的数值，可以指定为一个整数或一个字符串。

* **from\_base**

当前进制（2到36或-2到-36之间的数字）。

* **to\_base**

目标进制（2到36或-2到-36之间的数字）。

【说明】

* 如果任何参数为null，则返回null。
* 参数N被当做整数，但它的形式可以是一个整数或一个字符串。最小进制是2，最大进制是36。
* 如果from\_base参数是负数，则输入N会被视为有符号数；如果to\_base参数是负数，则输出值被视为有符号数。

**注意事项**

无。

**示例**

**示例1：**直接调用CONV函数将16进制的字符串a转换为2进制的字符串表示。

SELECT CONV('a',16,2);

返回结果为：

conv

--------

'1010'

(1 row)

**示例2：**CONV函数的from\_base参数为负数时，输入的参数N将被视为有符号整数。

SELECT CONV(-2,-10,-2);

结果返回如下：

conv

------

-10

(1 row)

**示例3：**在函数中使用CONV函数。

1、创建测试函数将输入参数a由16进制转换为2进制。

create or replace function func\_test(a int) return text

is testvalue text;

begin

testvalue :=conv(a,16,2);

return testvalue;

end;

/

1. 调用func\_test.

call func\_test(4);

结果返回如下：

func\_test

-----------

100

(1 row)

**示例4：**DML语句中使用CONV函数。

1、创建测试表并插入数据。

create table test(id int,col varchar2(30));

insert into test values(1,'60E');

2、插入和更新数据。

insert into test values(2,conv(1438,10,16));

insert into test values(3,conv(1439,10,16));

update test set col=conv(1566,10,16) where id=1;

3、查询数据。

select id,col,conv(col,16,10) from test order by id;

返回结果为：

id | col | conv

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

1 | 61E | 1566

2 | 59E | 1438

3 | 59F | 1439

(3 rows)

##### FIELD

**功能描述**

FIELD函数用于查找特定字符串在给出的若干个字符串序列中的位置。

函数返回值如下所示：

* 0：代表待查找字符串不在序列中。
* 非0值：代表字符串在序列的位置，以1为起始。

**语法格式**

FIELD(str,str1,str2,str3,...)

**参数说明**

* **str**

待查找字符串。

* **str1,str2,str3,...**

被查找字符串序列，不定长度。

【说明】

* 如果所有传入参数是字符串类型，则这些参数按照字符串规则进行比较。如果所有传入参数是number类型则按 number类型进行比较。否则将按浮点数类型比较。
* 如果 str为 null，则返回 0，因为 null无法和任何值比较。

**注意事项**

函数的输入参数个数不能少于2，否则会返回如下错误提示：

ERROR: Incorrect parameter count in the call to native function 'field'

**示例**

**示例1：**直接调用FIELD函数。

select field(23.17, 3, 23.17899, 23.1777, 23.17);

返回结果为：

field

------

4

(1 row)

**示例2：**在insert语句中使用FILED函数。

1. 创建测试表并插入数据。

create table tb\_1116624(c1 int, c2 varchar(10), c3 text);

insert into tb\_1116624 values(1, 'test1', 'test4');

insert into tb\_1116624 values(2, 'test2', 'test5');

insert into tb\_1116624 values(3, 'test3', 'test3');

insert into tb\_1116624 values(4, 'test4', 'test8');

insert into tb\_1116624 values(5, 'test5', 'test2');

1. 创建需要插入数据的表。

create table test\_insert\_tb\_1116624(c1 int, c2 varchar(10), c3 text);

3、insert语句中使用FIELD函数。

with select\_data\_1116624 as(

select \* from tb\_1116624 order by field(c2, 'test3', 'test1'))

insert into test\_insert\_tb\_1116624 select \* from select\_data\_1116624;

4、查看test\_insert\_tb\_1116624表数据。

select \* from test\_insert\_tb\_1116624;

返回结果为：

c1 | c2 | c3

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

2 | test2 | test5

4 | test4 | test8

5 | test5 | test2

3 | test3 | test3

1 | test1 | test4

(5 rows)

根据结果可知，测试表 test\_insert\_tb\_1116624中插入了经过order by filed排序之后所创建的临时表数据。

##### LOCATE

**功能描述**

LOCATE函数用于查找子字符串在原字符串中的位置。

函数返回值如下所示：

* 0：子字符串不在原字符串中。
* 非0值：子字符串在原字符串的位置，以1为起始。

如果输入值存在null时，则函数返回null。

**语法格式**

LOCATE(substr,str [,pos])

**参数说明**

* **substr**

子字符串。

* **str**

原字符串。

* **pos**

原字符串开始查找的位置，以1为起始，当pos参数给出时，则从pos位置开始。如果pos值小于1，则结果返回0。当pos有小数时，pos取值为整数值加1，例如locate('a','nbabaaa',4.5)，表示从第5个字符开始搜索字符a在'nbabaaa'中的位置。

**注意事项**

无。

**示例**

**示例1：**直接调用LOCATE函数。

指定pos参数。

--正整数

select locate('a','nbanaa',4);

--负数

select locate('a','nba',-4);

--小数

select locate('a','nbabaaa',4.5);

返回结果分别为：

locate

--------

5

(1 row)

locate

--------

0

(1 row)

locate

--------

5

(1 row)

不指定pos参数。

select locate('ABC','KJABCFG');

返回结果为：

locate

--------

3

(1 row)

**示例2：**在函数中使用LOCATE函数。

1、创建函数。

create or replace function fun\_1116772(a in char(16),b in char(16),n out int)return int

as

begin

select locate(a,b) into n;

return n;

end;

/

2、调用函数pro\_1116772。

select fun\_1116772('ABV','HBABV');

返回结果为：

fun\_1116772

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

3

(1 row)

##### SUBSTRING\_INDEX

**功能描述**

Vastbase G100支持使用substring\_index()函数按照指定的分隔符划分字符串，返回以分隔符分割的原字符串中对应索引位的子字符串。

**语法格式**

substring\_index(str,delim,index)

**参数说明**

* **str**

需要操作的原字符串。

* **delim**

分隔符。

* **index**

索引位，匹配 delim 出现的次数。取值可正数可负数：

* index为正数，返回以delim分割之后index对应位置的左边子字符串。
* index为负数，返回以delim分割之后index对应位置的右边子字符串。

**注意事项**

无。

**示例**

**示例1：**使用select语句查看substring\_index()的字符串分割结果。

select substring\_index('中文，测试，test，%','，',3);

select substring\_index('中文，测试，test，%','，',1);

select substring\_index('中文，测试，test，%','，',-1);

返回结果为：

substring\_index

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

中文，测试，test

(1 row)

​

substring\_index

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

中文

(1 row)

​

substring\_index

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

%

(1 row)

**示例2：**对表进行DML操作时调用substring\_index()函数。

1、创建测试表t\_substring\_index。

CREATE TABLE t\_substring\_index(c1 int,c2 varchar(20),c3 json,c4 timestamptz);

2、向分区表中插入测试数据。

INSERT INTO t\_substring\_index VALUES(1,'192.166.2.203','{"一年级":{"一班":{"张三":{"学号":1,"年龄":19,"家乡":"河北"},"李四":{"学号":2,"年龄":20,"家乡":"广东"}},"二班":{"王五":{"学号":1,"年龄":19,"家乡":"湖南"}}},"二年级":{"一班":{"赵六":{"学号":1,"年龄":21,"家乡":"内蒙古"}}}}','2022-02-02 10:10:10+08');

INSERT INTO t\_substring\_index VALUES(2,'192.167.2.203','{"一年级":{"一班":{"张三":{"学号":1,"年龄":19,"家乡":"河北"},"李四":{"学号":2,"年龄":20,"家乡":"广东"}},"二班":{"王五":{"学号":1,"年龄":19,"家乡":"湖南"}}},"二年级":{"一班":{"赵六":{"学号":1,"年龄":21,"家乡":"内蒙古"}}}}','2022-01-02 10:10:10+08');

INSERT INTO t\_substring\_index VALUES(3,'192.168.2.203','{"一年级":{"一班":{"张三":{"学号":1,"年龄":19,"家乡":"河北"},"李四":{"学号":2,"年龄":20,"家乡":"广东"}},"二班":{"王五":{"学号":1,"年龄":19,"家乡":"湖南"}}},"二年级":{"一班":{"赵六":{"学号":1,"年龄":21,"家乡":"内蒙古"}}}}','2021-02-02 10:10:10+08');

3、执行INSERT语句。

insert into t\_substring\_index select \* from t\_substring\_index where substring\_index(substring\_index(c2,'.',2),'.',-1)=166;

4、执行UPDATE语句。

update t\_substring\_index set c4=NULL where substring\_index(c4,'-',1)=2021;

5、执行DELETE语句。

delete from t\_substring\_index where substring\_index(substring\_index(c2,'.',2),'.',-1)=167;

6、以上DML语句成功执行后，查看执行结果。

select \* from t\_substring\_index;

返回结果为：

c1 |     c2       |                                                                                                             c3

                                                                  |           c4

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

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

 1 | 192.166.2.203 | {"一年级":{"一班":{"张三":{"学号":1,"年龄":19,"家乡":"河北"},"李四":{"学号":2,"年龄":20,"家乡":"广东"}},"二班":{"王五":{"学号":1,"年龄":19,"家乡":"湖南"}}

},"二年级":{"一班":{"赵六":{"学号":1,"年龄":21,"家乡":"内蒙古"}}}} | 2022-02-02 10:10:10+08

 1 | 192.166.2.203 | {"一年级":{"一班":{"张三":{"学号":1,"年龄":19,"家乡":"河北"},"李四":{"学号":2,"年龄":20,"家乡":"广东"}},"二班":{"王五":{"学号":1,"年龄":19,"家乡":"湖南"}}

},"二年级":{"一班":{"赵六":{"学号":1,"年龄":21,"家乡":"内蒙古"}}}} | 2022-02-02 10:10:10+08

 3 | 192.168.2.203 | {"一年级":{"一班":{"张三":{"学号":1,"年龄":19,"家乡":"河北"},"李四":{"学号":2,"年龄":20,"家乡":"广东"}},"二班":{"王五":{"学号":1,"年龄":19,"家乡":"湖南"}}

},"二年级":{"一班":{"赵六":{"学号":1,"年龄":21,"家乡":"内蒙古"}}}} |

(3 rows)

#### **条件表达式函数**

##### ISNULL

**功能描述**

Vastbase在MySQL兼容模式下，支持ISNULL函数，用于判读输入的参数是否为null，如果输入参数为null返回true，否则返回false。

**语法格式**

ISNULL(expr)

**参数说明**

**expr**

输入表达式。

**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。

**示例**

**前置步骤：**创建兼容模式为MySQL的库db\_mysql，并进入。

create database db\_mysql dbcompatibility 'B';

\c db\_mysql

**示例1：**直接调用ISNULL函数。

SELECT ISNULL(1+1);

返回结果为：

?column?

----------

f

(1 row)

**示例2：**在存储过程中使用ISNULL函数。

1、创建存储过程，在存储过程中调用ISNULL函数。

create or replace procedure proc1 as

c1 varchar;

c2 varchar;

begin

SELECT ISNULL(NULL) into c1;

SELECT ISNULL('a') into c2;

raise info '%', c1;

raise info '%', c2;

end;

/

1. 调用存储过程。

call proc1();

返回结果为：

INFO: true

INFO: false

proc1

-------

(1 row)

#### **JSON函数**

##### JSON\_VALID

**功能描述**

Vastbase在MySQL兼容模式下，支持JSON\_VALID函数，用于判读输入值是否为合法的JSON数据。如果输入值是合法的JSON数据则返回1，否则返回0。

**语法格式**

JSON\_VALID(val)

**参数说明**

**val**

输入值。

**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。

**示例**

**前置步骤：**创建兼容模式为MySQL的库db\_mysql，并进入。

create database db\_mysql dbcompatibility 'B';

\c db\_mysql

**示例1：**直接调用JSON\_VALID函数。

select json\_valid('"sss"');

返回结果为：

json\_valid

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

1

(1 row)

**示例2：**在DML中使用JSON\_VALID函数。

1. 创建表。

create table tab\_1116530(id int,a int,c text default json\_valid('9'));

1. 插入数据。

insert into tab\_1116530 values(8,json\_valid('1'),'"a"');

insert into tab\_1116530 values('2',json\_valid('1'),'"s"');

insert into tab\_1116530 values('2',9);

1. 查看数据。

select \* from tab\_1116530;

返回结果为：

id | a | c

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

8 | 1 | "a"

2 | 1 | "s"

2 | 9 | 1

(3 rows)

1. 在select语句中使用JSON\_VALID函数。

select id,json\_valid(id) from tab\_1116530;

返回结果为：

id | json\_valid

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

8 | 0

2 | 0

2 | 0

(3 rows)

1. 更新并查看数据。

update tab\_1116530 set id=3 where json\_valid(id)=0;

select \* from tab\_1116530;

返回结果为：

id | a | c

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

3 | 1 | "a"

3 | 1 | "s"

3 | 9 | 1

(3 rows)

1. 删除并查看数据。

delete from tab\_1116530 where json\_valid(id)=0;

select \* from tab\_1116530;

返回结果为：

id | a | c

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

(0 rows)

### PL/SQL

#### 支持@i:=0方式赋值

**功能描述**

Vastbase在MySQL兼容模式下支持SELECT @i:=expr方式赋值查询，输出查询结果并将查询结果存储到"@i"中，若查询结果有多行，则存储最后一行数据。

**语法格式**

@i:=expr

**参数说明**

* **i**

被赋值对象。

* **expr**

赋值语句。

**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。
* 使用时需先设置参数"enable\_set\_variable\_b\_format"为"on"。参数"enable\_set\_variable\_b\_format"设置为"on"时支持使用"@varname"作为变量。
* 仅支持赋值、case语句、操作符计算、调用函数等运算。
* 运算的返回结果仅支持数字类型、字符类型或布尔类型。
* 如参数expr（赋值语句）中包含"@i"，需保证"@i"的类型在计算前后不发生改变。

**示例**

**前置步骤**

1、创建兼容模式为MySQL的库db\_mysql，并进入。

create database db\_mysql dbcompatibility 'B';

\c db\_mysql

2、设置"enable\_set\_variable\_b\_format"为"on"。

set enable\_set\_variable\_b\_format to on;

show enable\_set\_variable\_b\_format;

结果返回如下表示设置成功：

enable\_set\_variable\_b\_format

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

on

(1 row)

**示例1：**调用字符处理函数char\_length，将字符长度赋值给参数"i"。

1、执行语句进行赋值。

select @i:=char\_length('hello');

返回赋值结果如下：

?column?

----------

5

(1 row)

2、查询"@i"的值。

select @i;

结果返回如下，表示赋值成功：

@i

----

5

(1 row)

**示例2：**多变量赋值查询。

1、创建测试表并插入数据。

create table tab\_1130999(a1 int);

insert into tab\_1130999 values(1),(2),(3);

2、执行赋值查询。

select @1130999\_1:=3,@1130999\_2:='aa',@1130999\_3:='你好',@1130999\_4:=a1 from tab\_1130999;

结果返回如下：

?column? | ?column? | ?column? | ?column?

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

3 | aa | 你好 | 1

3 | aa | 你好 | 2

3 | aa | 你好 | 3

(3 rows)

3、查询赋值结果。

select @1130999\_1,@1130999\_2,@1130999\_3,@1130999\_4;

结果返回如下：

@1130999\_1 | @1130999\_2 | @1130999\_3 | @1130999\_4

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

3 | aa | 你好 | 3

(1 row)

**示例3：**使用case语句进行赋值。

1、创建测试表并插入测试数据。

create table tab\_1130965(a1 int,a2 int);

insert into tab\_1130965 values(1,1),(2,3),(3,2),(4,1);

2、使用case语句进行赋值，当"tab\_1130965.a1<3"时，赋值为"tab\_1130965.a1 +3"。

select (@bq1:=case when tab\_1130965.a1<3 then tab\_1130965.a1 +3 else tab\_1130965.a1 end) from tab\_1130965;

结果返回如下：

?column?

----------

4

5

3

4

(4 rows)

3、查询赋值结果。

select @bq1;

结果返回如下：

@bq1

------

4

(1 row)

### 特性参数

#### 其他参数

**sql\_ignore\_strategy**

**参数说明：**在B兼容模式下，该参数可控制ignore\_error的hint在违反非空约束时的处理策略。

该参数属于USERSET类型。

**取值范围：**枚举类型

* ignore\_null：忽略违反非空约束的行的处理。
* overwrite\_null：将违反约束的null值覆写为目标类型的默认值。

**默认值：**ignore\_null

## PostgreSQL兼容性

## SQL Server兼容性

### 数据类型

#### datetime2类型

**功能描述**

Vastbase G100在SQL Server兼容模式下支持datetime2类型。datetime2类型结合了24小时制时间的日期，其格式为yyyy-MM-dd HH:mm:ss.ffffff。

**语法格式**

datetime2[($n)]

**参数说明**

**$n**

用户指定的精度值。

取值范围：0~6，其中0和6可取。

【须知】

当精度大于6时，会提醒用户最大值为6，并强制将精度修改为6。

当精度小于0时，会报错。

**注意事项**

* 本特性仅在数据库兼容模式为SQL Server时能够使用（即创建DB时DBCOMPATIBILITY='MSSQL'），在其他数据库兼容模式下不能使用该特性。
* 当插入字段值的精度超过预设的精度时，只保留值到预设精度；当插入值小于预设精度时，以插入值为准。

**示例**

**前置条件：**创建兼容SQL Server的库db\_sqlserver，并进入。

create database db\_sqlserver dbcompatibility  'MSSQL';

\c db\_sqlserver

**示例1：**创建带datetime2的数据类型的测试表。

1、创建带datetime2字段的测试表。

create table test\_1(id int,col1 datetime2);

2、向测试表中插入带公元、时区的记录。

insert into test\_1 values(1,'294277-1-9 04:00:49.101010');

insert into test\_1 values(2,'4713-12-31 23:59:59.999999 bc');

insert into test\_1 values(3,'713-12-31 23:00:00.111 +08');

insert into test\_1 values(4,'713-12-31 23:00:00.111 bc +08');

insert into test\_1 values(5,'294277-1-9 04:00:49.101010 AD'),(7,null);

【说明】

datetime2类型是在插入数据时，会忽略时区。

AD表示公元后，BC表示公元前。当日期不带BC时默认就是公元后，所以在查询记录时，公元后的日期可以不显示AD，公元前的日期必须带有BC。

3、查询插入记录在表中的情况。

select \* from test\_1 order by col1;

结果展示为：

id |             col1

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

 7 |

 2 | 4713-12-31 23:59:59.999999 BC

 4 | 0713-12-31 23:00:00.111 BC

 3 | 0713-12-31 23:00:00.111

 1 | 294277-01-09 04:00:49.10101

 5 | 294277-01-09 04:00:49.10101

(6 rows)

**示例2：**创建带指定精度范围的datetime2类型的表。

1、创建测试表。

create table test\_2(col1 datetime2(3));

2、向测试表中插入数据。

insert into test\_2 values('0713-12-31 23:00:00.1');

insert into test\_2 values('0713-12-31 23:00:00.111');

insert into test\_2 values('0713-12-31 23:00:00.111111');

3、查询测试表中的数据。

select \* from test\_2;

查询结果显示为不同精度的数据插入测试表后，小数点后都保持为三位：

        col1

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

0713-12-31 23:00:00.1

0713-12-31 23:00:00.111

0713-12-31 23:00:00.111

**示例3：**对datetime2字段指定不同的精度创建表。

1、创建带不同精度的datetime2字段的测试表。

create table test1(col1 datetime2(0));

create table test2(col1 datetime2(6));

create table test(col1 datetime2(7));

create table test(col1 datetime2(-1));

返回结果分别为前两条测试语句成功执行；第三条语句创建表成功，但由于精度超过6提示deduced to 6，即表示创建成功，精度被修改为最高值6；第四条执行失败，因为精度值不能为负数：

CREATE TABLE

CREATE TABLE

WARNING: TIMESTAMP(7) precision reduced to maximum allowed, 6

LINE 1: create table test(col1 datetime2(7));

^

WARNING: TIMESTAMP(7) precision reduced to maximum allowed, 6

CREATE TABLE

ERROR: syntax error at or near "-"

LINE 1: create table test(col1 datetime2(-1));

^

#### mallmoney类型

**功能描述**

Vastbase G100在SQL Server兼容模式下支持smallmoney类型。smallmoney类型是可以代表货币或货币值的数据类型，其支持范围从-92233720368547758.08 到 +92233720368547758.07。

**注意事项**

* 本特性仅在数据库兼容模式为SQL Server时能够使用（即创建DB时DBCOMPATIBILITY='MSSQL'），在其他数据库兼容模式下不能使用该特性。
* smallmoney相关类型转换说明：
* 支持int、bigint、numeric类型隐式转换为smallmoney类型。
* 支持real、float、double precison、varchar、text类型转化为numeric后，再转化为smallmoney类型。
* date、timestamp、blob等类型无法隐式转化为smallmoney类型。
* 插入或更新smallmoney类型的字段值为空、null时，其在表中被记录为$0.00。

**示例**

1、创建兼容SQL Server的库db\_sqlserver，并进入。

create database db\_sqlserver dbcompatibility 'MSSQL';

\c db\_sqlserver

2、创建含有smallmoney类型的表。

create table tb\_smallmoney(

id int,

name varchar,

salary smallmoney);

3、向表中插入数据，插入后查询表记录。

insert into tb\_smallmoney values(1, 'xiaoming', 5000.8334);

insert into tb\_smallmoney values(2, 'xiaolan', 123457123.6771);

insert into tb\_smallmoney values(3, 'xiaohong', 789754.9011);

insert into tb\_smallmoney values(4, 'xiaoming', 983.8954);

select \* from tb\_smallmoney;

查询表记录结果如下：

id |   name   |     salary

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

 1 | xiaoming |       $5,000.83

 2 | xiaolan  | $123,457,123.68

 3 | xiaohong |     $789,754.90

 4 | xiaoming |         $983.90

(4 rows)

4、更新表中一条记录，更新后查询表记录。

update tb\_smallmoney set salary = 7890.5645 where id = 3;

select \* from tb\_smallmoney;

查询表记录结果如下：

id |   name   |     salary

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

 1 | xiaoming |       $5,000.83

 2 | xiaolan  | $123,457,123.68

 4 | xiaoming |         $983.90

 3 | xiaohong |       $7,890.56

(4 rows)

5、删除表中一条记录，删除后查询表记录。

delete from tb\_smallmoney where salary = 983.8954::smallmoney;

select \* from tb\_smallmoney;

查询表记录结果如下：

id |   name   |     salary

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

 1 | xiaoming |       $5,000.83

 2 | xiaolan  | $123,457,123.68

 3 | xiaohong |       $7,890.56

(3 rows)

6、向表中插入一条第三个字段为numeric类型的记录，其可以被隐式转换为smallmoney类型。

insert into tb\_smallmoney values(4,'xiaoli',233.669925::numeric);

select \* from tb\_smallmoney;

查询表结果为如下，第三个字段的类型被隐式转换为了smallmoney：

id | name | salary

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

1 | xiaoming | $5,000.83

2 | xiaolan | $123,457,123.68

3 | xiaohong | $7,890.56

4 | xiaoli | $233.67

(4 rows)

#### sql\_variant类型

**功能描述**

Vastbase G100在SQL Server兼容模式下支持sql\_variant类型，sql\_variant类型可以保存非用户定义类型（除特殊说明的类型）的值，并保留原类型信息，可以用在列、参数、变量和函数返回值中。

**注意事项**

* 本特性仅在数据库兼容模式为SQL Server时能够使用（即创建DB时DBCOMPATIBILITY='MSSQL'），在其他数据库兼容模式下不能使用该特性。
* 基础类型的二进制长度必须 <= 8000字节。
* 本类型支持在表、视图、匿名块、存储过程和自定义函数中使用。
* sql\_variant不支持的类型有：text、ntext、image、varchar(max)、nvarchar(max)、varbinary(max)、xml、timestamp和用户自定义类型。

**示例**

【说明】

sql\_variant类型会自动补齐8000字节，所以实际查询结果中带大量补齐内容。为方面用户查看示例结果，本文档中删去了自动补齐的部分。

**前置条件：**创建兼容SQL Server的库db\_mysql，并进入。

create database db\_sqlserver dbcompatibility  'MSSQL';

\c db\_sqlserver

**示例1：**在系统表中查找sql\_variant类型。

\x    --列式展示查询结果

select \* from pg\_type where typname='sql\_variant';

结果显示为：

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

typname        | sql\_variant

typnamespace   | 11

typowner       | 10

typlen         | -1

typbyval       | f

typtype        | b

typcategory    | U

typispreferred | f

typisdefined   | t

typdelim       | ,

typrelid       | 0

typelem        | 0

typarray       | 0

typinput       | sql\_variantin

typoutput      | sql\_variantout

typreceive     | sql\_variantrecv

typsend        | sql\_variantsend

typmodin       | -

typmodout      | -

typanalyze     | -

typalign       | i

typstorage     | x

typnotnull     | f

typbasetype    | 0

typtypmod      | -1

typndims       | 0

typcollation   | 100

typdefaultbin  |

typdefault     |

typacl         |

**示例2：**字符类型强制转换为sql\_variant类型。

select 'aa'::char::sql\_variant;

select '圈圈圆圆圈圈天天'::char(20)::sql\_variant;

select 'aaaaaaaa'::text::sql\_variant;

结果除了text类型转换失败，其余均成功。其中第二条查询语句中超出基础类型规定长度的内容被截断，合规长度内容被转为sql\_variant类型，其结果展示为：

sql\_variant

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

a

(1 row)

​

sql\_variant

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

圈圈圆圆圈圈

(1 row)

​

ERROR: cannot cast type text to sql\_variant

CONTEXT: referenced column: sql\_variant

**示例3：**sql\_variant对基础类型二进制长度要求<=8000。

1、基础类型二进制长度小于等于8000。

select '我的'::varchar(7999)::sql\_variant;

select '你的'::varchar(8000)::sql\_variant;

结果显示为：

sql\_variant

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

我的

(1 row)

​

sql\_variant

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

你的

(1 row)

2、基础类型二进制长度大于8000。

select repeat('qqqqqqqq',1001)::varchar(8001)::sql\_variant;

结果显示为超出范围，类型转换会失败：

ERROR:  value of basic type must be a binary length <= 8000 byte

CONTEXT: referenced column: repeat

#### ntext类型

**功能描述**

Vastbase G100在SQL Server兼容模式下支持ntext类型。ntext类型是用于存储大量Unicode文本数据的可变长度数据类型。

**注意事项**

* 本特性仅在数据库兼容模式为SQL Server时能够使用（即创建DB时DBCOMPATIBILITY='MSSQL'），在其他数据库兼容模式下不能使用该特性。
* ntext类型为text类型的复用，所以没有类型oid，无法直接通过查询内部表获取信息。

**示例**

**前置条件：**创建兼容SQL Server的库db\_sqlserver，并进入。

create database db\_sqlserver dbcompatibility 'MSSQL';

\c db\_sqlserver

**示例1：** 类型强制转换。

1、类型强制转换为ntext类型。

select 1::ntext;

select 'aa'::char::ntext;

结果显示为：

text

------

1

(1 row)

​

text

------

a

(1 row)

2、将next类型强制转换为其他。

select 'AAAAA'::ntext::varchar;

select 5::ntext::int;

结果显示为：

varchar

---------

AAAAA

(1 row)

​

int4

------

   5

(1 row)

**示例2：**在表中使用ntext类型。

1、创建带ntext类型的表。

create table tab1(a1 ntext PRIMARY KEY);

insert into tab1 values('2'),('ff'),('!'),('li'),('good'),('yes');

2、创建与表1关联的表2。

create table tab2(a1 ntext not null,a2 ntext unique,a3 ntext PRIMARY KEY,a4 ntext default 'good',a5 ntext check(a5 is not null),a6 ntext REFERENCES tab1(a1));

3、使用元命令查看表2信息。

\d+ tab2

表2具体信息如下：

                         Table "public.tab\_1117218"

Column | Type |     Modifiers       | Storage  | Stats target | Description

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

a1     | text | not null             | extended |              |

a2     | text |                      | extended |              |

a3     | text | not null             | extended |              |

a4     | text | default 'good'::text | extended |              |

a5     | text |                      | extended |              |

a6     | text |                      | extended |              |

Indexes:

   "tab\_1117218\_pkey" PRIMARY KEY, btree (a3) TABLESPACE pg\_default

   "tab\_1117218\_a2\_key" UNIQUE CONSTRAINT, btree (a2) TABLESPACE pg\_default

Check constraints:

   "tab\_1117218\_a5\_check" CHECK (a5 IS NOT NULL)

Foreign-key constraints:

   "tab\_1117218\_a6\_fkey" FOREIGN KEY (a6) REFERENCES tt\_1117218(a1)

Has OIDs: no

Options: orientation=row, compression=no, fillfactor=80

4、向表2中插入数据并查看记录。

insert into tab\_1117218 values('my','name','cc','is','li','li');

insert into tab\_1117218 values('aa','bb','how','are','you','!');

select \* from tab2 order by a1;

查询结果为：

a1 | a2  | a3  | a4  | a5  | a6

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

aa | bb   | how | are | you | !

my | name | cc  | is  | li  | li

(2 rows)

#### table类型

**功能描述**

Vastbase G100在SQL Server兼容模式下支持table类型。table类型作为表值参数，以INPUT输入形式传递到存储过程或函数中，不支持作为OUTPUT参数，不能用来定义列的类型。

**注意事项**

* 本特性仅在数据库兼容模式为SQL Server时能够使用（即创建DB时DBCOMPATIBILITY='MSSQL'），在其他数据库兼容模式下不能使用该特性。
* table类型不能用来定义表中列的类型。
* table类型作为表值输入参数传递到存储过程或者函数中，不能执行注入UPDATE、DELETE或INSERT等DML操作；也不能SELECT INTO语句的目标；不支持作为OUTPUT参数。
* 不能使用ALTER TABLE语句来修改表值参数的表结构。
* 在函数或存储过程中使用$n引用表值参数需要添加双引号“”。

**示例**

**前置条件：**创建兼容SQL Server的库db\_sqlserver，并进入。

create database db\_sqlserver dbcompatibility 'MSSQL';

\c db\_sqlserver

**示例1：**将含有单列数值型内容的table类型插入至行存表中。

1、创建单列table类型。

create type tp1 as table(id int1);

2、创建插入表，向其中的表2插入两条数据。

create table tb1(id int);

create table tb2(id int,date1 date);

insert into tb2 values (1,'2022-1-11');

insert into tb2 values (2,'2022-1-12');

3、创建存储过程。

create or replace procedure proc1(v1 tp1)

as

begin

insert into tb1 select \* from v1;

end;

/

4、执行存储过程，将从表2中搜索到的id列数据插入至表1中。

declare

a tp1;

begin

insert into a select id from tb2;

perform proc1(a);

end;

/

5、查询表记录。

select \* from tb1 order by id;

select \* from tb2 order by id;

表1记录为来自表2的字段id内容，表2记录为初始插入内容，查询结果依次如下：

id

----

 1

 2

(2 rows)

​

id |   date1

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

 1 | 2022-01-11

 2 | 2022-01-12

(2 rows)

**示例2：** 将含多列数值型内容的table类型插入至行存表中。

1、创建多列table类型。

create type tp2 as table(

num\_1 numeric,

num\_2 number(5),

num\_3 number(3,3),

num\_4 numeric(4,3),

num\_5 decimal(3,3)

);

2、创建插入表。

create table tb3(

num\_1 numeric,

num\_2 number(5),

num\_3 number(3,3),

num\_4 numeric(4,3),

num\_5 decimal(3,3))WITH (ORIENTATION = COLUMN);

3、创建存储过程，从table类型的数据中筛选出小于num\_1小于6的记录，然后将记录插入至表中。

create or replace procedure proc1(v1 tp2)

as

begin

insert into tb3 select \* from v1 where num\_1<6;

end;

/

4、先向table类型a中插入多条数据，然后执行存储过程。

declare

a tp2;

begin

insert into a values (1,1235,0.123,1.57,0.101);

insert into a values (2.37,11111,0.003,1.999,-0.001);

insert into a values (3.1144,12345,0.993,1.68,0.001);

insert into a values (4.9,129.12,0.3,1,0.91);

insert into a values (5.999,1235.5,0.1234,9.9993,0.1235);

insert into a values (6.1,123,0,-0.99,0);

perform proc1(a);

end;

/

5、查看表记录。

select \* from tb3 order by num\_1;

查询表结果如下，table类型的数据中满足条件的记录被放入表中：

num\_1  | num\_2 | num\_3 | num\_4 | num\_5

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

     1 |  1235 |  .123 |  1.57 |  .101

  2.37 | 11111 |  .003 | 1.999 | -.001

3.1144 | 12345 |  .993 |  1.68 |  .001

   4.9 |   129 |    .3 |     1 |   .91

 5.999 |  1236 |  .123 | 9.999 |  .124

(5 rows)

#### uniqueidentifier类型

**功能描述**

Vastbase G100在SQL Server兼容模式下，支持uniqueidentifier类型。

uniqueidentifier类型是32位的十六进制数字和四个连字符”-“的排列，形如xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx。其中，每个 x 都是 0-9 或 a-f 范围内的十六进制数字。

uniqueidentifier列可用于确保表的多个副本中唯一标识行。

**注意事项**

* 本特性仅在数据库兼容模式为SQL Server时能够使用（即创建DB时DBCOMPATIBILITY='MSSQL'），在其他数据库兼容模式下不能使用该特性。
* 对于长度超过36的字符串转化为uniqueidentifier类型时，Vastbase数据库会报错。
* uniqueidentifier数据类型不支持转换为Vastbase不兼容的SQL Server数据类型。

**示例**

**前置条件：**创建兼容SQL Server的库db\_sqlserver，并进入。

create database db\_sqlserver dbcompatibility 'MSSQL';

\c db\_sqlserver

**示例：**创建带有索引的行存表。

1、创建测试表和创建普通索引和唯一索引。

create table tb1(col1 uniqueidentifier);

create index idx1 on tb1(col1);

2、向测试表中插入数据，并查看记录。

insert into tb1 values('0E984725-C51C-4BF4-9960-E1C80E27ABA0'),('00000000-0000-0000-0000-000000000000'),('aaaaaaaa-aaaa-aaaa-aaaa-aaaaaaaaaaaa');

select \* from tb1 order by col1;

结果展示为：

                col1

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

00000000-0000-0000-0000-000000000000

0e984725-c51c-4bf4-9960-e1c80e27aba0

aaaaaaaa-aaaa-aaaa-aaaa-aaaaaaaaaaaa

(3 rows)

3、修改测试表字段名字，并查看记录。

alter table tb1 rename col1 to col\_new;

select \* from tb1 order by col\_new;

结果展示为：

              col\_new

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

00000000-0000-0000-0000-000000000000

0e984725-c51c-4bf4-9960-e1c80e27aba0

aaaaaaaa-aaaa-aaaa-aaaa-aaaaaaaaaaaa

(3 rows)

4、更新表中数据，并查看记录。

update tb1 set col\_new='12345678-1234-1234-1234-012345678912' where col\_new='0E984725-C51C-4BF4-9960-E1C80E27ABA0';

select \* from tb1 order by col\_new;

结果展示为：

              col\_new

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

00000000-0000-0000-0000-000000000000

12345678-1234-1234-1234-012345678912

aaaaaaaa-aaaa-aaaa-aaaa-aaaaaaaaaaaa

(3 rows)

5、删除表中数据，并查看记录。

delete from tb1 where col\_new='00000000-0000-0000-0000-000000000000';

select \* from tb1 order by col\_new;

结果展示为：

              col\_new

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

12345678-1234-1234-1234-012345678912

aaaaaaaa-aaaa-aaaa-aaaa-aaaaaaaaaaaa

(2 rows)

#### varbinary类型

**功能描述**

Vastbase G100在SQL Server兼容模式下支持varbinary类型。varbinary为可变长度二进制字符串类型，存储的是二进制数据。其中包括三种类型定义：

* varbinary，插入值可变长。
* varbinary(n)，插入值的长度为n，数据长度可以在 1到8000之间，其中包括1和8000。
* varbinary(max)，插入值的长度为max，代表长度为2147483647。

**语法格式**

varbinary

varbinary(n)

varbinary(max)

**参数说明**

* **n**

变长值指定长度。

取值范围：1~8000

单位：byte

* **max**

最大值。

代表长度：2147483647。

单位：byte

**注意事项**

* 本特性仅在数据库兼容模式为SQL Server时能够使用（即创建DB时DBCOMPATIBILITY='MSSQL'），在其他数据库兼容模式下不能使用该特性。
* Vastbase G100进行类型转换时，其他类型转为varbinary类型可能存在结果与SQL Server中类型转换的结果不一致的情况。

**示例**

**前置条件：**创建兼容SQL Server的库db\_sqlserver，并进入。

create database db\_sqlserver dbcompatibility  'MSSQL';

\c db\_sqlserver

**示例1：** 在系统表中查找varbinary类型。

\x    --列式展示查询结果

select \* from pg\_type where typname='varbinary';

结果显示为：

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

typname        | varbinary

typnamespace   | 11

typowner       | 10

typlen         | -1

typbyval       | f

typtype        | b

typcategory    | U

typispreferred | f

typisdefined   | t

typdelim       | ,

typrelid       | 0

typelem        | 0

typarray       | 0

typinput       | varbinaryin

typoutput      | varbinaryout

typreceive     | varbinaryrecv

typsend        | binarysend

typmodin       | varbinarytypmodin

typmodout      | binarytypmodout

typanalyze     | -

typalign       | i

typstorage     | x

typnotnull     | f

typbasetype    | 0

typtypmod      | -1

typndims       | 0

typcollation   | 0

typdefaultbin  |

typdefault     |

typacl         |

**示例2：** 创建带varbinary类型的表，并插入数据。

1、创建带varbinary类型的表，并插入数据，可以成功创建并插入数据。

create table test1(col1 int,col2 varbinary);

insert into test1 values(1,12);

insert into test1 values(3,null);

insert into test1 values(1,'0xFF');

insert into test1 values(1,'0x100');

2、查询表记录。

select \* from test1;

结果显示为：

col1 | col2

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

  1 | 0x0c

  3 |

  1 | 0xff

  1 | 0x01

(4 rows)

**示例3：** 创建带varbinary(n)类型的表。

1、创建带varbinary(n)类型的表。

create table test2(col3 varbinary(0));

create table test3(col3 varbinary(1));

create table test4(col3 varbinary(8000));

create table test5(col3 varbinary(8001));

只有当n值符合1~8000范围内才能创建表成功，否则创建表失败。以上四条建表语句的结果显示为test3、test4表创建成功，test2、test5表创建失败：

ERROR:  varbinary(n): n must be a value from 1 through 8000

CREATE TABLE

CREATE TABLE

ERROR:  varbinary(n): n must be a value from 1 through 8000

2、分别向带varbinary(1)、varbinary(8000)字段的表中插入数据。

insert into test3 values('0x123456');

insert into test4 values('0x123456');

以上两条插入语句结果为插入test3失败，插入test4成功：

ERROR:  value too long for type varbinary(1)

INSERT 0 1

3、查询test4表中插入的记录。

select \* from test4;

结果显示为：

  col3

----------

0x123456

(1 row)

**示例4：** 创建带varbinary(max)类型的表。

1、创建带varbinary(max)类型的表。

create table test6(col4 varbinary(max));

2、向带varbinary(max)类型的表插入数据。

insert into test6 values('\x123456');

select \* from test6;

查询结果为：

  col4

----------

0x123456

(1 row)

### 函数

#### 时间和日期处理函数

##### DATEADD

**功能描述**

Vastbase G100在SQL Server兼容模式下支持DATEADD函数，用于在日期中添加或减去指定的时间间隔。

**语法格式**

DATEADD(datepart,number,date)

**参数说明**

* **datepart**

指定date中要与number整数值相加的部分，支持的参数如下表所示：

| **datepart** | **缩写形式** | **备注** |
| --- | --- | --- |
| year | yy,yyyy | 年 |
| quarter | qq,q | 季度 |
| month | mm,m | 月 |
| dayofyear | dy,y | 一年中的第几天 |
| day | dd,d | 日 |
| week | wk,ww | 一年中的第几周 |
| weekday | dw,w | 星期几 |
| hour | hh | 小时 |
| minute | mi,n | 分钟 |
| second | ss,s | 秒 |
| millisecond | ms | 毫秒 |
| microsecond | mcs | 微秒 |
| nanosecond | ns | 毫微秒 |

* **number**

一个表达式，可解析为 DATEADD 将其与date 的 datepart相加的 int类型整数。

* **date**

要处理的目标时间值。

可解析为下列值之一的表达式:

* time
* timetz
* date
* timestamp
* timestamptz

**注意事项**

* 该功能仅在数据库兼容模式为SQL Server时能够使用（即创建DB时DBCOMPATIBILITY='MSSQL'），在其他数据库兼容模式下不能使用该特性。
* 函数输入有 date ，time，tiemtz，timestamp，timestampz类型，其中有时区信息时其显示结果会有相应的时区信息。

**示例**

**前置步骤：**创建兼容模式为SQL Server的库db\_sqlserver，并进入。

CREATE DATABASE db\_sqlserver DBCOMPATIBILITY='MSSQL'；

\c db\_sqlserver

**示例1：**直接调用DATEADD函数，将指定时间增加1分钟。

select dateadd(mi,1,timestamp'1997-12-31 23:59:59');

结果返回如下：

date\_add

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

1998-01-01 00:00:59

(1 row)

**示例2：**DATEADD函数与now函数套用，返回当前日期加上10年的日期。

select dateadd(year,10,now()) y;

结果返回如下：

y

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

2033-02-03 08:50:43.872425+00

(1 row)

**示例3：**参数为行存表中的列。

1、创建测试表并插入数据。

create table tb\_info(id int,name text,birth date,new int);

insert into tb\_info values(1,'lili','1995-11-11',4);

insert into tb\_info values(2,'kiko','1997-2-15',3);

insert into tb\_info values(3,'lala','2000-1-3',1);

insert into tb\_info values(4,'ming','2002-5-7',-2);

select \* from tb\_info order by id;

返回结果如下：

id | name | birth | new

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

1 | lili | 1995-11-11 | 4

2 | kiko | 1997-02-15 | 3

3 | lala | 2000-01-03 | 1

4 | ming | 2002-05-07 | -2

(4 rows)

2、调用DATEADD函数查询数据。

select dateadd(yy,new,birth),count(dateadd(yy,new,birth)) from tb\_info group by 1 order by 1;

结果返回如下：

date\_add | count

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

1999-11-11 00:00:00 | 1

2000-02-15 00:00:00 | 1

2000-05-07 00:00:00 | 1

2001-01-03 00:00:00 | 1

(4 rows)

##### DATEDIFF

**功能描述**

Vastbase G100在SQL Server兼容模式下支持DATEDIFF函数，用于计算两个日期之间的时间间隔。函数返回指定startdate和enddate之间所跨的指定datepart边界的计数（作为带符号整数值），返回值类型为integer类型。

**语法格式**

datediff(depart,startdate,enddate)

**参数说明**

* **depart**

enddate与startdate之间的间隔差值单位。

取值范围：depart值不能在变量中指定，也不能指定为如'month'这样带引号的字符串。DATEDIFF接受所有有效的depart全名或全名缩写，其中在[表1](" \l "table1)中列出了所有有效的depart值。

**表1** 有效的depart值

|  |  |  |
| --- | --- | --- |
| **日期部分** | **缩写** | **备注** |
| year | yy,yyyy | 年 |
| quater | qq,q | 季度 |
| month | mm,m | 月 |
| dayofyear | dy,y | 一年中的第几天 |
| day | dd,d | 日 |
| week | wk,ww | 一年中的第几周 |
| weekday | dw日期部分返回对应于星期中的某天的数,w  例如：Sunday=1 | 星期几 |
| hour | hh | 小时 |
| minute | mi,n | 分种 |
| second | ss,s | 秒 |
| millisecond | ms | 毫秒 |
| microsecond | mcs | 微秒 |
| nanosecond | ns | 纳秒 |

* **satrtdate**

起始日期。

可以解析为以下之一的表达式：time、timetz、date、timestamp、timestamptz。

* **enddate**

终止日期。

可以解析为以下之一的表达式：time、timetz、date、timestamp、timestamptz。

**注意事项**

* 本特性仅在数据库兼容模式为SQL Server时能够使用（即创建DB时DBCOMPATIBILITY='MSSQL'），在其他数据库兼容模式下不能使用该特性。
* 若返回值超出范围（-2,147,483,648到+2,147,483,647），DATEDIFF函数返回错误。
* 对于指定depart值为millisecond的DATEDIFF函数，startdate和enddate之间的最大差值为24天20小时31分钟23.647秒。
* 对于指定depart值为second的DATEDIFF函数，startdate和enddate的最大差值为68年19天3小时14分7秒。
* Vastbase的时间精度只到微秒级别，纳秒部分的值将四舍五入进入微秒部分。

**示例**

**前提条件：**创建兼容SQL Server的库db\_sqlserver，并进入。

create database db\_sqlserver dbcompatibility 'MSSQL';

\c db\_sqlserver

**示例1：**depart输入为year时直接使用datediff函数。

1. 查询当enddate比startdate小的情况。

select datediff(year,'2023-12-31 23:59:59','2023-01-01 00:00:00');

查询结果为0年：

datediff\_mssql

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

0

(1 row)

1. 查询当enddate比start大的情况。

select datediff(year,'2023-12-31 23:59:59','2024-01-01 00:00:00');

查询结果为enddate与startdate差值，即1年：

datediff\_mssql

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

1

(1 row)

**示例2：**在测试表中使用以ss为depart输入的datediff函数。

1. 创建测试表，并插入数据。

CREATE TABLE t\_datediff\_second(

c1 int,

c2 time,

c3 timetz,

c4 date,

c5 timestamp,

c6 timestamptz

)partition by list(c4)

(partition p1 values('0001-01-01'),

partition p2 values('2023-02-09'),

partition p3 values(default));

INSERT INTO t\_datediff\_second VALUES(1,'2001-01-01 00:00:00+00'::time,'2001-01-01 00:00:00+00'::timetz,'2001-01-01 00:00:00+00','2001-01-01 00:00:00+00','2001-01-01 00:00:00+00');

INSERT INTO t\_datediff\_second VALUES(2,'2030-12-31 23:59:59+08','2030-12-31 23:59:59+08','2030-12-31 23:59:59+08','2030-12-31 23:59:59+08','2030-12-31 23:59:59+08');

INSERT INTO t\_datediff\_second VALUES(3,'2023-02-09 10:04:00+08','2023-02-09 10:04:00+08','2023-02-09 10:04:00+08','2023-02-09 10:04:00+08','2023-02-09 10:04:00+08');

1. 在SELECT的语句中使用DATEDIFF函数（函数以ss作为depart输入）。

select

datediff(ss,A.c2,B.c2),datediff(ss,A.c3,B.c3),datediff(ss,A.c4,B.c4),datediff(ss,A.c5,B.c5),datediff(ss,A.c6,B.c6)

FROM t\_datediff\_second A LEFT JOIN t\_datediff\_second B

ON A.c1=B.c1+1

WHERE B.c1 is not null;

查询结果如下：

datediff\_mssql | datediff\_mssql | datediff\_mssql | datediff\_mssql | datediff\_mssql

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

50159 | 50159 | 249004800 | 249054959 | 249054959

1 | -57599 | -946598400 | -946684799 | -946655999

(2 rows)

##### DATENAME

**功能描述**

Vastbase G100在SQL Server兼容模式下支持DATENAME函数，用于返回表示指定 date 的指定 datepart 的字符串，返回值为varchar类型。

**语法格式**

DATENAME(datepart,date)

**参数说明**

* **datepart**

datepart将返回表示 date 参数的特定部分的整数，支持的datepart参数如下表所示：

| **datepart** | **缩写形式** | **备注** |
| --- | --- | --- |
| year | yy,yyyy | 年 |
| quarter | qq,q | 季度 |
| month | mm,m | 月 |
| dayofyear | dy,y | 一年中的第几天 |
| day | dd,d | 日 |
| week | wk,ww | 一年中的第几周 |
| weekday | dw,w | 星期几 |
| hour | hh | 小时 |
| minute | mi,n | 分钟 |
| second | ss,s | 秒 |
| millisecond | ms | 毫秒 |
| microsecond | mcs | 微秒 |
| nanosecond | ns | 毫微秒 |
| tzoffset | tz | 时区偏移量 |
| iso\_week | isowk,isoww | ISO标准周数 |

* **date**

要返回datepart值的日期、时间或时间戳表达式。日期表达式必须包含datepart类型的值。

可解析为下列值之一的表达式：

* time
* timetz
* date
* timestamp
* timestamptz

**注意事项**

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

**示例**

**前置步骤：**创建兼容模式为SQL Server的库db\_sqlserver，并进入。

CREATE DATABASE db\_sqlserver DBCOMPATIBILITY='MSSQL';

\c db\_sqlserver

**示例1：**直接调用DATENAME函数。

SELECT datename(month,getdate());

返回结果如下：

datename

----------

February

(1 row)

**示例2：**在sql语句中调用DATENAME函数。

1、创建测试表。

create table test(id int,col1 text,col2 varchar,col3 char(8),col4 varchar2(10));

2、在INSERT语句中使用。

insert into test values(1,datename(yyyy,getdate()),

datename(month,'2022-12-11'),datename(day,'1999-09-18'),datename(week,'2099-12-01'));

insert into test(id) values(2);

3、在UPDATE语句中使用。

update test set col3=datename(second,'1999-09-18 20:59:59') where id=2;

4、查询结果。

SELECT \* FROM test;

结果返回如下：

id | col1 | col2 | col3 | col4

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

1 | 2023 | December | 18 | 49

2 | | | 59 |

(2 rows)

##### DATEPART

**功能描述**

Vastbase G100在SQL Server模式下支持DATEPART函数，用于返回表示指定 date 的指定 datepart 的整数，返回值为整数类型。

**语法格式**

DATEPART(datepart,date)

**参数说明**

* **datepart**

datepart将返回表示 date 参数的特定部分的整数，支持的datepart参数如下所示：

| **datepart** | **缩写形式** | **备注** |
| --- | --- | --- |
| year | yy,yyyy | 年 |
| quarter | qq,q | 季度 |
| month | mm,m | 月 |
| dayofyear | dy,y | 一年中的第几天 |
| day | dd,d | 日 |
| week | wk,ww | 一年中的第几周 |
| weekday | dw,w | 星期几 |
| hour | hh | 小时 |
| minute | mi,n | 分钟 |
| second | ss,s | 秒 |
| millisecond | ms | 毫秒 |
| microsecond | mcs | 微秒 |
| nanosecond | ns | 毫微秒 |
| tzoffset | tz | 时区偏移量 |
| iso\_week | isowk,isoww | ISO标准周数 |

* **date**

要返回datepart值的日期、时间或时间戳表达式。日期表达式必须包含datepart类型的值。

可解析为下列值之一的表达式：

* time
* timetz
* date
* timestamp
* Timestamptz

**注意事项**

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

**示例**

**前置步骤：**创建兼容模式为SQL Server的库db\_sqlserver，并进入。

CREATE DATABASE db\_sqlserver DBCOMPATIBILITY='MSSQL'；

\c db\_sqlserver

**示例1：**直接调用DATEPART函数。

SELECT DATEPART(m,timestamp'2007-10-30 12:15:32.1234567');

结果返回如下：

datepart

----------

10

(1 row)

**示例2**：date参数为行存表中的列。

1、创建测试表并插入数据。

create table tb\_info(id int,name text,birth date);

insert into tb\_info values(1,'lili','1999-11-11');

insert into tb\_info values(2,'kiko','2000-2-15');

insert into tb\_info values(3,'lala','2001-1-3');

insert into tb\_info values(4,'ming','1999-5-7');

create table tb\_class(class int,year int);

insert into tb\_class values(4,2001);

insert into tb\_class values(5,2000);

insert into tb\_class values(6,1999);

insert into tb\_class values(7,1998);

2、查询表中数据。

select \* from tb\_info order by id;

结果返回如下：

id | name | birth

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

1 | lili | 1999-11-11

2 | kiko | 2000-02-15

3 | lala | 2001-01-03

4 | ming | 1999-05-07

(4 rows)

select \* from tb\_class order by class;

结果返回如下：

class | year

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

4 | 2001

5 | 2000

6 | 1999

7 | 1998

(4 rows)

3、调用DATEPART函数查询数据。

select datepart(yy,birth),sum(datepart(yy,birth))/datepart(yy,birth) count from tb\_info having datepart(yy,birth) group by 1 order by 1;

结果返回如下：

datepart | count

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

1999 | 2

2000 | 1

2001 | 1

(3 rows)

##### GETDATE

**功能描述**

Vastbase在SQL Server兼容模式下支持GETDATE函数，用于返回当前数据库系统的时间戳。函数的返回类型为timestamp类型，不含数据库时区偏移量，其中结果截取到毫秒。

**语法格式**

getdate()

**注意事项**

* 本特性仅在数据库兼容模式为SQL Server时能够使用（即创建DB时DBCOMPATIBILITY='MSSQL'），在其他数据库兼容模式下不能使用该特性。
* 函数返回结果末尾有0时，末尾的0会被省略，因此与SQL Server的结果有差异。

**示例**

**前置条件：** 创建兼容SQL Server的库db\_sqlserver，并进入。

create database db\_sqlserver dbcompatibility  'MSSQL';

\c db\_sqlserver

**示例1：**查询当前数据库的时间戳。

select getdate();

结果显示为：

getdate

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

2023-01-18 10:39:25.583

**示例2：**在表字段中使用getdate函数。

1、创建测试表，并插入数据。

create table getdate\_test1(id int, mygetdate timestamp default getdate());

insert into getdate\_test1(id) values (1);

2、查询表数据。

select \* from getdate\_test1;

查询结果为：

id |       mygetdate

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

 1 | 2023-02-07 06:19:31.737

(1 row)

**示例3：**在DML语句中使用getdate函数。

1、创建测试表，并向表中插入数据。

create table getdate\_test2(

id int,

mygetdate timestamp

);

insert into getdate\_test2 values(1,getdate());

select \* from getdate\_test2;

查看表记录结果为：

id |       mygetdate

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

 1 | 2023-02-07 07:26:31.803

(1 row)

2、使用getdate函数将表中数据更新为当前系统时间。

update getdate\_test2 set mygetdate=getdate();

select \* from getdate\_test2;

查看表记录结果为：

id |       mygetdate

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

 1 | 2023-02-07 07:28:12.723

(1 row)

3、筛选表中小于当前系统时间的记录。

select \* from getdate\_test2 where mygetdate<getdate();

查看表记录结果为：

id | mygetdate

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

1 | 2023-02-07 07:28:12.723

(1 row)

##### SYSDATETIME

**功能描述**

Vastbase在SQL Server兼容模式下支持使用SYSDATETIME函数返回计算机的日期和时间，返回值为timestamp类型。

**语法格式**

SYSDATETIME()

**注意事项**

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

**示例**

**前置步骤：**创建兼容模式为SQLServer的库db\_sqlserver，并进入。

CREATE DATABASE db\_sqlserver DBCOMPATIBILITY='MSSQL'；

\c db\_sqlserver

**示例1：**直接调用SYSDATETIME函数。

select sysdatetime();

返回结果如下：

sysdatetime

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

2023-02-06 07:44:18.388343

(1 row)

**示例2：**SYSDATETIME函数作为关键字在表中调用。

1、创建测试表。

CREATE TABLE products(

product\_id INT PRIMARY KEY,

product\_name VARCHAR(255) NOT NULL,

sysdatetime date,

sale\_time date,

unit\_price DEC(10,2),

discounted\_price DEC(10,2),

CHECK(unit\_price > 0),

CHECK(discounted\_price > 0),

CHECK(sysdatetime < sysdatetime())

);

2、调用SYSDATETIME函数插入测试数据。

insert into products values(1,'page','2022-10-09',sysdatetime(),3.0,2.5);

insert into products values(2,'pen','2022-11-16',sysdatetime(),10.0,8.4);

3、查询数据进行验证。

select \* from products;

结果返回如下：

product\_id | product\_name | sysdatetime | sale\_time | unit\_price | discounted\_price

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

1 | page | 2022-10-09 | 2023-02-06 | 3 | 2.5

2 | pen | 2022-11-16 | 2023-02-06 | 10 | 8.4

(2 rows)

#### 条件表达式函数

##### ISNULL

**功能描述**

Vastbase在SQL Server兼容模式下支持ISNULL函数，本函数可以判断表达式的值是否为NULL。当表达式值被判断为NULL时，使用指定的值替换NULL值。

**语法格式**

ISNULL(check\_expression,replacement\_value)

**参数说明**

* **check\_expression**

被判断的表达式。

当表达式的值为NULL时，将被指定的值（即replacement\_value值）替换。

* **replacement\_value**

指定的替换值。

当check\_expression表达式的值被判断为NULL时，其值被替换为指定的值（replacement\_value值）。

**注意事项**

* 本特性仅在数据库兼容模式为SQL Server时能够使用（即创建DB时DBCOMPATIBILITY='MSSQL'），在其他数据库兼容模式下不能使用该特性。
* 替换值replacement\_value的类型必须可以隐式转换为表达式check\_expression的类型，当无法隐式转换时，函数报错。
* check\_expression和replacement\_value为必填项，可以填入NULL和空字符串或其他类型值，但不可不填。

**示例**

**前置条件：**创建兼容SQL Server的库db\_sqlserver，并进入。

create database db\_sqlserver dbcompatibility  'MSSQL';

\c db\_sqlserver

**示例1：**select语句中调用ISNULL函数。

1、当判断值不为空时，调用ISNULL函数。

SELECT ISNULL('Hello','W3Schools.com'::varchar);

表达式值不被替换，返回结果为：

isnull

--------

Hello

(1 row)

2、当判断值为空时，调用ISNULL函数。

SELECT ISNULL(null,'W3Schools.com'::varchar);

表达式值被指定的值替换，返回结果为：

isnull

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

W3Schools.com

(1 row)

**示例2：**在查询测试表中数据时调用ISNULL函数。

1、创建测试表并插入数据。

create table tb\_test(id int, name varchar);

insert into tb\_test values(1, NULL), (NULL, 'test2'), (3, 'test3');

select \* from tb\_test;

表中记录的内容为：

id | name

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

 1 |

   | test2

 3 | test3

(3 rows)

2、调用ISNULL函数。

SELECT ISNULL(id, '101'), ISNULL(name, 'testname') from tb\_test order by id;

结果显示如下，其中在查询结果中，id字段为空的记录的id值被替换为101；name字段为空的记录的name值被替换为了testname：

isnull | isnull

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

    1 | testname

    3 | test3

  101 | test2

(3 rows)

**示例3：**当被检查表达式为空字符串时，其值不被替换。

SELECT ISNULL('','W3Schools.com'::varchar);

结果显示如下，表达式不被替换：

isnull

--------

​

(1 row)

#### 统计信息函数

##### COUNT\_BIG

**功能描述**

Vastbase G100支持COUNT\_BIG函数，用于返回表中的非空记录条数，返回值为bigint类型。

**语法格式**

SELECT COUNT\_BIG(columnRef) FROM tab\_name;

**参数说明**

* **columnRef**

列名。

【说明】

可以使用\*或者数据列名作为参数。\*作为参数会返回所有列的数目，列名作为参数则返回该列非空的数目。

* **tab\_name**

查询的表名。

**注意事项**

无。

**示例**

1、创建测试表tb1。

create table tb1(col\_int2 int2 primary key,

col\_int int,

col\_integer integer(5,2),

col\_int4 int4,

col\_int8 int8

);

2、插入测试数据.

insert into tb1 values(1,10,15.34,17,25);

insert into tb1 values(2,20,null,17,49);

insert into tb1 values(3,40,148.66,17,3);

insert into tb1 values(4,10,57.3,null,1196);

insert into tb1 values(5,10,9,null,2);

3、调用count\_big函数查询表中行数。

select count\_big(tb1) from tb1;

结果返回如下：

count\_big

-----------

5

(1 row)

4、调用count\_big函数查询某列行数。

select count\_big(col\_int2) a,count\_big(col\_int) b1,count\_big(distinct col\_int) b2,count\_big(col\_integer) c1,count\_big(all col\_integer) c2,count\_big(col\_int4) d1,count\_big(all col\_int4) d2,count\_big(distinct col\_int4) d3,count\_big(col\_int8) e1,count\_big(all col\_int8) e1,count\_big(distinct col\_int8) e2 from tb1;

结果返回如下：

a | b1 | b2 | c1 | c2 | d1 | d2 | d3 | e1 | e1 | e2

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

5 | 5 | 3 | 4 | 4 | 3 | 3 | 1 | 5 | 5 | 5

(1 row)

#### 字符和数据类型处理函数

##### CHARINDEX

**功能描述**

CHARINDEX函数会在第二个字符表达式中搜索一个字符表达式，返回第一个表达式（如果发现存在）的开始位置。如果没有发现，则返回0，类型为 int。如果 expressionToFind和 expressionToSearch有一个为空串，返回 NULL。

**语法格式**

CHARINDEX(expressionToFind,expressionToSearch [, start\_location])

**参数说明**

* expressionToFind

字符表达式，其中包含要查找的序列。

* expressionToSearch

要搜索的字符表达式。

* start\_location

表示搜索开始位置的integer 或 bigint表达式。如果start\_location 未指定、具有负数值或值为0，搜索将从expressionToSearch 的开头开始。

**注意事项**

无。

**示例**

**示例1：**对字符串使用charindex函数。

select charindex('is','this is a string');

结果返回如下：

charindex

-----------

3

(1 row)

**示例2：**指定start\_location对字符串使用charindex函数。

select charindex('is','this is a string',4);

结果返回如下：

charindex

-----------

6

(1 row)

**示例3：**在表中调用charindex函数，start\_location值为表中的列值。

1、创建测试表并插入测试数据。

CREATE TABLE test\_1130762(student\_id int, city varchar, address varchar, key int);

insert into test\_1130762 values(1, 'beijing', '北京市beijing东城区', 2);

insert into test\_1130762 values(2, 'tianjin', '天津tianjin市天津东城区', 4);

insert into test\_1130762 values(3, 'xian', '西安市xian雁塔区', 4);

2、调用charindex函数查询数据。

select student\_id, CHARINDEX(city, address, key) from test\_1130762 order by student\_id;

结果返回如下：

student\_id | charindex

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

1 | 4

2 | 0

3 | 4

(3 rows)

##### CONVERT

**功能描述**

Vastbase G100在SQL Server兼容模式下支持CONVERT函数，用于将表达式由一种数据类型转换为另一种数据类型，返回值为函数中指定数据类型。

**语法格式**

CONVERT(data\_type[(length)], expression[,style ])

**参数说明**

* **data\_type**

目标数据类型。

* **length**

指定目标数据类型长度的可选整数，适用于允许用户指定长度的数据类型。

默认值：30

* **expression**

有效表达式。

* **style**

指定 CONVERT函数将如何转换表达式的整数表达式。

默认值：0

【说明】

仅支持style值为0。

**注意事项**

* 该功能仅在数据库兼容模式为SQL Server时能够使用（即创建DB时DBCOMPATIBILITY='MSSQL'），在其他数据库兼容模式下不能使用该特性。
* SQL Server时间类型与Vastbase G100不完全一致，该函数在Vastbase G100中使用时间类型timestamp代替datetime类型。

**示例**

**前置步骤：**创建兼容模式为SQL Server的库db\_sqlserver，并进入。

CREATE DATABASE db\_sqlserver DBCOMPATIBILITY='MSSQL';

\c db\_sqlserver

**示例1：**直接调用CONVERT函数，将timestamp类型转换为varchar类型。

select convert(varchar,getdate());

结果返回如下：

getdate

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

Feb 10 2023 02:40PM

(1 row)

**示例2：**在函数和存储过程中调用CONVERT函数。

1、创建函数。

create function fun\_1131634(b1 int,b2 out char(8))return char(8)

as

begin

select (convert(char(8),b1) || 'oo') into b2;

return b2;

end;

/

2、调用新建函数验证结果。

select convert(text,fun\_1131634(3));

结果返回如下：

fun\_1131634

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

3oo

(1 row)

3、创建存储过程。

create procedure pro\_1131634(b1 int,b2 out char(8))

as

begin

select (convert(char(8),b1) || 'oo') into b2;

raise notice '%',b2;

end;

/

4、调用存储过程验证结果。

select convert(text,pro\_1131634(6));

结果返回如下：

NOTICE: 6oo

CONTEXT: referenced column: pro\_1131634

pro\_1131634

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

6oo

(1 row)

##### LEN

**功能描述**

Vastbase在SQL Server兼容模式下支持使用LEN函数返回字符串的字符数，不包含尾随空格。

**语法格式**

LEN(input\_string)

**参数说明**

**input\_string**

描述：一个字符串或字符串表达式，其中包含要计算长度的序列。

取值范围：文字字符串、字符串表达式、字符或二进制数据的列。

**注意事项**

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

**示例**

**前置步骤：**创建兼容模式为SQLServer的库db\_sqlserver，并进入。

CREATE DATABASE db\_sqlserver DBCOMPATIBILITY='MSSQL'；

\c db\_sqlserver

**示例1：**直接调用LEN函数。

select len('123');

结果返回如下：

len

-----

3

(1 row)

**示例2：**函数LEN出现在目标列与其他函数套用。

1、创建测试表并插入数据。

create table tb\_1116312(

ID NUMBER(20),

id1 varchar(10),

id2 char(10),

id3 char(30)

) ;

insert into tb\_1116312 values(generate\_series(1, 1000), generate\_series(1, 1000)||'A ', generate\_series(1, 1000)||' A ',generate\_series(1, 2000)||' AB ');

insert into tb\_1116312 values(null);

insert into tb\_1116312 values(null,'','','');

2、查询列"id3"最大值的长度。

select len(max(id3)) from tb\_1116312 ;

结果返回如下：

len

-----

4

(1 row)

3、查询列"id3"长度的最大值。

select max(len(id3)) from tb\_1116312 ;

结果返回如下：

max

-----

7

(1 row)

##### STR

**功能描述**

Vastbase G100在SQL Server兼容模式下支持STR函数，用于返回数字数据转换来的字符数据，具有指定长度和十进制精度。返回值为varchar类型，转换所得的字符串。

**语法格式**

STR(float\_expression[,length [,decimal]])

**参数说明**

* **float\_expression**

带小数点的近似数字，float数据类型表达式。

* **length**

总长度，包括小数点、符号、数字以及空格。

默认值：10

* **decimal**

小数点后的位数。

默认值：0

**注意事项**

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

**示例**

**前置步骤：**创建兼容模式为SQL Server的库db\_sqlserver，并进入。

CREATE DATABASE db\_sqlserver DBCOMPATIBILITY='MSSQL';

\c db\_sqlserver

**示例1：**直接调用STR函数。

select str(123.45,6,1);

结果返回如下：

str

---------

123.5

(1 row)

**示例2：**对行存表使用DML语句调用STR函数。

1、创建测试表并插入数据。

CREATE table test\_tb\_1132253(id int, income varchar);

insert into test\_tb\_1132253 values(1, '12.3462754');

insert into test\_tb\_1132253 values(2, '222.3462754');

insert into test\_tb\_1132253 values(3, '333.3462754');

2、SELECT语句中调用STR函数处理返回结果。

select id, STR(income, 6, 2) from test\_tb\_1132253;

结果返回如下：

id | str

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

1 | 12.35

2 | 222.35

3 | 333.35

(3 rows)

3、UPDATE语句中调用STR函数处理数据进行更新。

update test\_tb\_1132253 set income = STR(777.889741, 10, 4) where id = 2;

select \* from test\_tb\_1132253 order by id;

结果返回如下：

id | income

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

1 | 12.3462754

2 | 777.8897

3 | 333.3462754

(3 rows)

### 操作符

#### 字符操作符**"+"**

**功能描述**

Vastbase在SQL Server兼容模式下支持对字符串进行加法操作，字符串相加的操作作为字符串的连接返回。

**注意事项**

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

**示例**

**前置步骤：**创建兼容模式为SQL Server的库db\_sqlserver，并进入。

CREATE DATABASE db\_sqlserver DBCOMPATIBILITY='MSSQL';

\c db\_sqlserver

**示例1：**直接使用"+"连接字符串。

select 'aa'+'bb';

结果返回如下：

?column?

----------

aabb

(1 row)

**示例2：**字符连接符和运算操作符同时使用。

1、创建测试表并插入数据。

create table employee\_1131338(

id int,

name varchar,

job\_title varchar,

phone varchar,

address varchar);

insert into employee\_1131338 values(1, '小王', '产品', '17785', 'beijing');

insert into employee\_1131338 values(2, '小张', '研发', '198874', 'shanghai');

insert into employee\_1131338 values(3, '小李', '测试', '4875', 'guangzhou');

insert into employee\_1131338 values(4, '小赵', '测试', '4875', 'guangzhou');

2、查询语句中同时使用字符连接符和运算操作符。

select \* from employee\_1131338 where name = '小' + '王' or id = 1 + 2;

结果返回如下：

id | name | job\_title | phone | address

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

1 | 小王 | 产品 | 17785 | beijing

3 | 小李 | 测试 | 4875 | guangzhou

(2 rows)

【说明】

'小'+'王'中"+"为字符连接符，返回结果为"小王"。

1 + 2中"+"为运算操作符，返回结果为"3"。

### SQL语法

#### OUTPUT

**功能描述**

Vastbase G100在SQL Server兼容模式下支持OUTPUT语法，返回受SQL语句影响的各行中的信息，包括INSERT/UPDATE/DELETE OUTPUT和SELECT OUTPUT INTO。

**注意事项**

本特性仅在数据库兼容模式为SQL Server时能够使用（即创建DB时DBCOMPATIBILITY='MSSQL'），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SQL Server的库db\_sqlserver，并进入。

create database db\_sqlserver dbcompatibility 'MSSQL';

\c db\_sqlserver

2、创建测试表并插入测试数据。

CREATE TABLE t1(id int,name text,city text);

INSERT INTO t1 VALUES(1,'aaa','guangzhou');

INSERT INTO t1 VALUES(2,'bbb','beijing');

INSERT INTO t1 VALUES(3,'ccc','nanjing');

3、使用INSERT OUTPUT语句插入数据。

INSERT INTO t1 VALUES(4,'ddd','xian') output id,name;

结果返回如下：

id | name

----+------

4 | ddd

(1 row)

INSERT 0 1

4、使用UPDATE OUTPUT语句修改数据。

UPDATE t1 SET name='eee' WHERE id=4 OUTPUT id,name;

结果返回如下：

id | name

----+------

4 | eee

(1 row)

UPDATE 1

5、使用DELETE OUTPUT语句删除数据。

DELETE FROM t1 WHERE id=4 OUTPUT id,name;

结果返回如下：

id | name

----+------

4 | eee

(1 row)

DELETE 1

6、使用SELECT OUTPUT INTO创建新表t2。

SELECT id,name OUTPUT into table t2 FROM t1;

结果返回如下：

id | name

----+------

1 | aaa

2 | bbb

3 | ccc

(3 rows)

INSERT 0 3

7、查询t2表中数据

SELECT \* FROM t2;

结果返回如下：

id | name

----+------

1 | aaa

2 | bbb

3 | ccc

(3 rows)

# 工具参考

## vb\_initdb

**功能描述**

vb\_initdb用于初始化数据库，在初始化数据库时，会创建数据库目录、生成系统表、创建默认数据库和模板数据库。

* 系统表

初始化数据库时会生成大量的系统表和视图，其中绝大部分都对任何数据库用户开放查看权限。

【说明】

pg\_user\_status、pg\_auth\_history系统表权限只对初始化数据库用户和sysadmin用户开放。

* 创建数据库
* template1：是一个模板数据库，当以后再创建一个新的数据库时，template1数据库里的所有内容都会拷贝到新数据库中。通过vb\_initdb的参数可以决定template1数据库的设置。
* template0：是Vastbase提供的最初始的备份数据库，当需要时可用template0作为模板生成“干净”的数据库。
* postgres：是一个提供给用户、工具和第三方应用的缺省数据库。
* 在安装时，推荐使用-D参数调用vb\_initdb初始化数据库。如果由于故障恢复等原因，需要重新初始化一个数据库，可以通过执行vb\_initdb来完成。
* 尽管vb\_initdb会尝试创建相应的数据目录，但可能没有权限执行此操作，因为要创建目录的父目录通常被root所拥有。如果要创建数据目录，首先用root用户创建一个空数据目录，然后用chown把该目录的所有权交给数据库用户。
* vb\_initdb决定template1数据库的设置，而该设置将会成为其他数据库的默认设置。
* vb\_initdb初始化数据库的缺省区域和字符集编码。字符集编码、字符编码排序（LC\_COLLATE）和字符集类（LC\_CTYPE，如大写、小写数字等）可以在创建数据库时独立设置。

**语法格式**

vb\_initdb [OPTION]... [DATADIR]

**参数说明**

常用参数如下：

* **-?，--help**

显示vb\_initdb命令的帮助信息，然后退出。

* **-V，--version**

输出vb\_initdb命令的版本信息，然后退出。

* **-A, --auth=METHOD**

指定本地用户连接数据库时的认证方法，即“pg\_hba.conf”配置文件中host和local所在行的认证方法。

取值范围：trust、reject、md5（不安全的算法，为了兼容老版本而存在）、sha256、sm3

默认值：trust（除非用户对本地用户都是信任的，否则不要使用默认值trust）

【须知】

若取值为md5，则需手动修改参数文件 postgresql.conf.sample 中的密码存储类型 password\_encryption\_type 参数的值，修改为0，且放开注释使之生效。vs\_initdb工具需同时配合 -W 的使用。

* **--auth-host=METHOD**

指定本地用户通过TCP/IP连接数据库时的认证方法，即：“pg\_hba.conf”配置文件中host所在行的认证方法（指定此参数则会覆盖-A参数的值）。

取值范围：trust、reject、md5（不安全的算法，为了兼容老版本而存在）、sha256、sm3

默认值：trust

* **--auth-local=METHOD**

指定本地用户通过Unix域套接字连接数据库时的认证方法，即“pg\_hba.conf”配置文件中local所在行的认证方法（指定此参数则会覆盖-A参数的值）。

取值范围：trust、reject、md5（不安全的算法，为了兼容老版本而存在）、sha256、sm3、peer（仅用于local模式）

默认值：trust

* **[-D, --pgdata=]DATADIR**

指定数据目录的位置。

取值范围： DATADIR的取值：用户自定义。不能包括“|”, “;”，“&”，“$”，“<”，“>”，“`”，“\”，“!”这几个字符。

* **--nodename=NODENAME**

初始化的节点名称。

节点的命名需要遵守如下规范：

* 节点名称必须为小写字母（a-z）、下划线（\_）、特殊符号#、数字（0-9）。
* 节点名称必须以小写字母（a-z）或下划线（\_）开头。
* 节点名称不能为空，且最大的长度为64个字符 。
* **-E, --encoding=ENCODING**

为新数据库设置编码格式。

【须知】

* 如果使用此参数，需要加上--locale选项指定支持此编码格式的区域。如果不加--locale选项，则采用系统默认的区域，如果系统默认区域的编码格式和用此参数指定的编码格式不匹配则会导致数据库初始化失败。
* 如果不指定此参数，则使用系统默认区域的编码格式。系统默认区域和编码格式可以使用locale如下命令查看：locale|grep LC\_CTYPE
* 不能包括“|”, “;”，“&”，“$”，“<”，“>”，“`”，“\”，“!”这几个字符。
* **--locale=LOCALE**

为新数据库设置缺省的区域（可以用locale -a查看可用的区域），如果不希望指定特定的区域，则可以用C。

【须知】

* 如果用户设置了数据库的编码格式，则用户选择区域的编码格式必须与用户设置的编码格式一致，否则数据库初始化会失败。
* 不能包括“|”, “;”，“&”，“$”，“<”，“>”，“`”，“\”，“!”这几个字符。
* 示例：用户要将数据库编码格式初始化为GBK，可以采用如下步骤：

1、查看系统支持gbk编码的区域。

ocale -a|grep gbk

zh\_CN.gbk

zh\_SG.gbk

2、初始化数据库时加入--locale=zh\_CN.gbk选项。

* **--dbcompatibility=DBCOMPATIBILITY**

指定兼容的数据库的类型。

取值范围：取值范围：A、B、C、PG。分别表示兼容Oracle、MySQL、TD和POSTGRES。

* **--pad-attribute=PADATTRIBUTE****<a name="attribute"></a>**

为新数据库设置默认的列校对规则，可选取值包括：

* N,NO PAD：默认值。把字符串尾端的空格当作一个字符处理，即字符串等值比较不忽略尾端空格。
* S,PAD SPACE：字符串等值比较忽略尾端空格。

【说明】

Vastbase G100 V2.2 Build12版本开始支持在初始化数据库时指定列校对规则。

* **--no-locale**

和--locale=C等价。

* **--pwfile=FILE**

vb\_initdb从文件中读取数据库超级用户的密码。该文件的第一行被作为密码使用。

【须知】

FILE可以是“相对路径+文件”的形式，也可以是“绝对路径+文件”的形式。相对路径是相对当前路径的。不能包括“|”, “;”，“&”，“$”，“<”，“>”，“`”，“\”，“!”这几个字符。

* **-T, --text-search-config=CFG**

缺省的文本搜索方式。此配置项的值不会做正确性校验，配置成功后，有日志记录提醒当前配置项的取值。

取值范围：english（全文搜索）、simple（普通文本搜索）

默认值：simple

* **-U, --username=NAME**

选择数据库系统管理员的用户名。

取值范围：正常的数据库用户

【须知】

不能包括“|”, “;”，“&”，“$”，“<”，“>”，“`”，“\”，“!”这几个字符。

默认值：运行vb\_initdb的操作系统用户

* **-W, --pwprompt**

vb\_initdb时强制交互式输入数据库管理员的密码。

* **-w, --pwpasswd=PASSWD**

vb\_initdb时通过命令行指定的管理员用户的密码，而不是交互式输入。

【须知】

设置的密码要符合复杂度要求：

最少包含8个字符；

不能和用户名和当前密码（ALTER）相同，或和当前密码反序；

至少包含大写字母（A-Z）、小写字母（a-z）、数字、非字母数字字符（限定为~!@#$%^&\*()-\_=+|[{}];:,<.>/?）四类字符中的三类字符。

* **-X, --xlogdir=XLOGDIR**

声明事务日志存储的目录(所设置的目录，必须满足运行Vastbase的用户有读写权限)。

【须知】

只支持绝对路径。不能包括“|”, “;”，“&”，“$”，“<”，“>”，

“`”，“\”，“!”这几个字符。

* -**S, --security**

安全方式初始化数据库。以-S方式初始化的数据库后，创建的数据库用户权限受到限制，默认不再具有public schema的使用权限。

**使用方法**

执行如下命令初始化数据库。

vb\_initdb -D /home/vastbase/data/vastbase --nodename vastbase -w vbase@123

【说明】

./home/vastbase/data/vastbase为数据库数据目录，可自定义，需存在且在vastbase用户下，需要0700权限。详细请参见《快速入门->非实例化数据库安装->[初始化数据库运行环境](../%E5%BF%AB%E9%80%9F%E5%85%A5%E9%97%A8/%E9%9D%9E%E5%AE%9E%E4%BE%8B%E5%8C%96%E6%95%B0%E6%8D%AE%E5%BA%93%E5%AE%89%E8%A3%85.md" \l "chushihua)》。

## vb\_dump

vb\_dump是Vastbase用于导出数据库相关信息的工具，用户可以自定义导出一个数据库或其中的对象(模式、表、视图等），回收站对象除外。

Vastbase G100 V2.2 Build 12新增vb\_dump工具特性，支持vb\_dump导出SQL文本文件格式时使用子命令--table conditions指定导出条件。

详细用法等内容参见《Vastbase G100 V2.2 管理员指南》->备份与恢复 -> 逻辑备份与恢复 -> vb\_dump。